ANADOLU ÜNİVERSİTESİ

anadolum eKampüs

anadolum eKampüs ve anadolu mobil

dilediğin yerden, dilediğin zaman, öğrenme fırsatı!

(ekampus.anadolu.edu.tr)

(mobil.anadolu.edu.tr)

ekampus.anadolu.edu.tr

- Takvim
- Duyurular
- Ders Kitabı (PDF)
- Epub
- Html5
- Video
- Canlı Ders
- Sesli Kitap
- Ünite Özeti
- Sesli Özet
- Sorularla Öğrenelim
- Alıştırma
- Deneme Sınavı
- İnfografik
- Etkileşimli İçerik
- Bilgilendirme Panosu
- Çıkmış Sınav Soruları
- Sınav Giriş Bilgisi
- Sınav Sonuçları
- Öğrenci Topluluklari

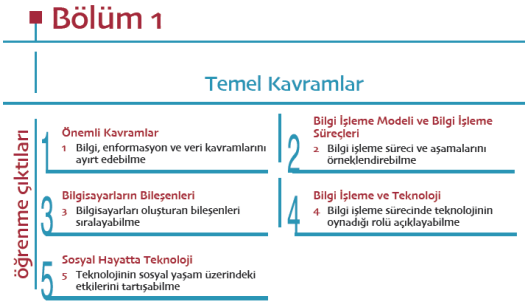
AOSDESTEK AÇIKÖĞRETİM DESTEK SİSTEMİ

aosdestek.anadolu.edu.tr

444 10 26

www.anadolu.edu.tr

/AOFAnadolum /Anadolu_Univ instagram.com/anadoluuniv

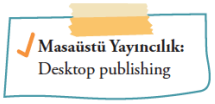


Öğrenme çıktıları

Bölüm içinde hangi bilgi, beceri ve yeterlikleri kazanacağınızı ifade eder.

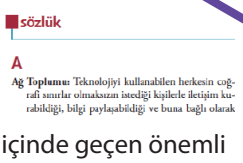
Bölüm Özeti

Bölümün kısa özetini gösterir.



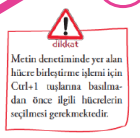
Tanım

Bölüm içinde geçen önemli kavramların tanımları verilir.



Sözlük

Bölüm içinde geçen önemli kavramlardan oluşan sözlük ünite sonunda paylaşılır.

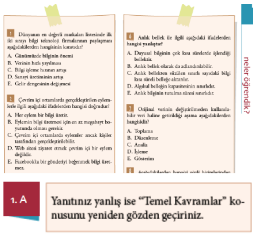


Dikkat

Konuya ilişkin önemli uyarıları gösterir.

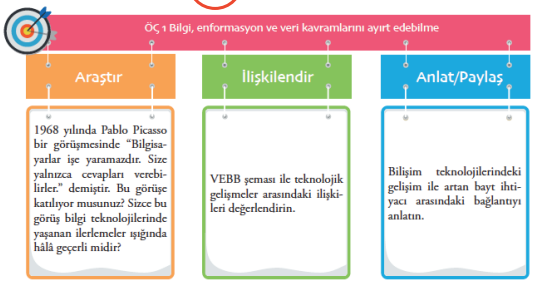
Karekod

Bölüm içinde verilen karekodlar, mobil cihazlarınız aracılığıyla sizi ek kaynaklara, videolara veya web adreslerine ulaştırır.



Neler Öğrendik ve Yanıt Anahtarı

Bölüm içeriğine ilişkin 10 adet çoktan seçmeli soru ve cevapları paylaşılır.



Öğrenme Çıktısı Tablosu

Araştır/İlişkilendir/Anlat-Paylaş

İlgili konuların altında cevaplayacağınız soruları, okuyabileceğiniz ek kaynakları ve konuyla ilgili yapabileceğiniz ekstra etkinlikleri gösterir.

Yaşamla İlişkilendir

Bölümün içeriğine uygun paylaşılan yaşama dair gerçek kesitler veya örnekleri gösterir.

Araştırmalarla İlişkilendir

Bölüm içeriği ile ilişkili araştırmaların ve bilimsel çalışmaları gösterir.

ANADOLU ÜNİVERSİTESİ

İleri Programlama

Editör

Doç.Dr. Cihan KALELİ

Yazarlar

BÖLÜM 1, 2 — Öğr.Gör. Özgür ÖZŞEN

BÖLÜM 3, 4 — Dr.Öğr. Üyesi Ahmet ARSLAN

BÖLÜM 5, 6, 7, 8 — Dr.Öğr. Üyesi Alper Kürşat UYSAL

T.C. ANADOLU ÜNİVERSİTESİ YAYINI NO: 3621
AÇIKÖĞRETİM FAKÜLTESİ YAYINI NO: 2449

Öğretim Tasarımcısı
Doç.Dr. Muhammet Recep Okur

Grafik Tasarım ve Kapak Düzeni
Prof.Dr. Halit Turgay Ünalan

Grafikerler
Gülşah Karabulut
Hilal Özcan
Ayşegül Dibek

Dizgi ve Yayıma Hazırlama
Kader Abpak Arul
Diğdem Koca
Halil Kaya
Kağan Küçük
Gizem Dalmış
Zülfiye Çevir
Saner Coşkun

İLERİ PROGRAMLAMA

E-ISBN
978-975-06-2938-9

ESKİŞEHİR, Aralık 2018

3116-0-0-0-2602-V01

İçindekiler

Önsöz

Sevgili öğrenciler,

Günümüzde yazılım teknolojilerini bünyesinde bulundurmayan bir sektör neredeyse bulunmamaktadır. Tıp dünyasından perakende ürün satış sektörüne kadar birçok farklı alanda hizmetin verilebilmesi bilgisayar programlama kabiliyetleri ile mümkün olabilmektedir. Burada bahsi geçen bilgisayar kavramı da alışılagelmiş bir kişisel bilgisayardan ziyade üzerinde dijital veriyi kaydedip işleyebilen her türlü cihaz anlamına evrilmiş durumdadır. Bunun yanı sıra kablolu ve kablosuz internet altyapısı ile GSM teknolojilerindeki gelişmeler, nesnelerin interneti kavramını doğurmuş ve her türlü nesnenin programlanabilip birbirleri ile haberleşmesinin önünü açmıştır. Bu durum programlama kabiliyetine sahip insan kaynağının önemini gitgide artırmıştır.

İleri programlama kitabı, ülkemizde dünyadaki teknolojik gelişmelere ayak uydurabilecek insan kaynağının eğitimini desteklemek amacıyla hazırlanmıştır. Bu kitap içerisinde dünyaca kabul görmüş ve oldukça yüksek bir kullanım alanı olan Java dili ile programlamanın unsurlarından bahsedilmiştir. Bu kitap ile birlikte, siz değerli öğrencilerimizin Java ile programlama konusunda tecrübe kazanmaları hedeflenmektedir. Bu doğrultuda, kitap içeriğinde sizlere en çok fayda sağlayacak örnekler sunulmaya çalışılmıştır.

İleri programlama kitabının size sadece belirli bir eğitim - öğretim dönemi süresince fayda sağlamasından ziyade tüm hayatınız boyunca bir başvuru kaynağı olması hedeflenmiştir. Bu doğrultuda, kitabın sizlere ulaşmasında emeği geçen herkese, göstermiş oldukları çabadan dolayı teşekkür ederim.

Editör

Doç.Dr. Cihan KALELİ

Bölüm 1

Nesneye Yönelik Programlamaya Giriş

Öğrenme çıktıları

Nesneye Yönelik Programlama Yaklaşımı

1 Nesneye yönelik programlama yaklaşımını tanımlayabilme
2 Sınıf ve nesne kavramlarını açıklayabilme
3 Nesnelerin davranış biçimlerini açıklayabilme

Nesneye Yönelik Çözümleme ve Tasarım

4 Nesneler arası ilişkileri açıklayabilme
5 Nesneye yönelik çözümleme ve tasarım kavramlarını açıklayabilme

Nesneye Yönelik Programlama Temelleri

6 Nesneye yönelik programlama temellerini açıklayabilme
7 Sarmalama, kalıtım, çok biçimlilik kavramlarını açıklayabilme

Anahtar Sözcükler: • Nesneye Yönelik Programlama • Sınıf • Nesne • Metot • Birleşik Modelleme Dili (UML) • Sarmalama (Encapsulation) • Kalıtım (Inheritance) • Çok Biçimlilik (Polymorphism)

GİRİŞ

Yazılımlar, gerçek hayatta karşılaşılan bir takım problemlerin çözülmesi için, üzerinde çalışan cihazın hızlı ve doğru işlem yapabilme gibi birtakım yeteneklerinden faydalanmak üzere yazılmaktadır. Günümüzde sadece bilgisayar ismiyle değil birçok cihaz artık bilgisayarların yerini almış durumdadır. Bu cihazların hızları ve kapasiteleri arttıkça geliştirilen programlar da bu hız ve kapasiteden faydalanacak şekilde gelişmektedir. Bu gelişim ve aynı zamanda değişim, zaman içerisinde yazılım geliştirme yöntemlerinde de değişikliklere sebep olmuş, bununla beraber çeşitli yazılım geliştirme yaklaşımlarının da ortaya çıkmasına neden olmuştur. Bu yaklaşımlar içerisinde en önemlilerden birisi de "*Nesneye yönelik yazılım geliştirme (Object Oriented Software Development)*" yaklaşımıdır.

Nesneye Yönelik Programlama

Kısaca bir programlama yaklaşımıdır. 1960'lı yılların sonuna doğru ortaya çıkan bu yaklaşım, o dönemin yazılım dünyasında beliren bir bunalımın sonucudur. Yazılımların karmaşıklığı ve boyutlarının sürekli artması ve aynı zamanda belli bir nitelik düzeyinin korunması için gerekli bakım maliyeti, zaman, çaba vb. unsurların sürekli artması sonucu nesneye yönelik programlama yaklaşımı çözüm olarak geliştirilmiştir.

Nesneye yönelik programlama yaklaşımı dışında çeşitli yaklaşımlar da bulunmaktadır. Bunlar arasında *Yapısal Programlama (Programming), Bileşen Tabanlı Yazılım Geliştirme (Component Based Software Development), Görünüm Yönelimli Programlama (Aspect Oriented Programming) sıralanabilir. Bunlar kendisinden önceki ve sonraki yaklaşımlara altyapı oluşturmuş ve geniş bir kullanım alanı bulmuşlardır.*

Bu ünitede nesneye yönelik programlama yaklaşımı kavramı ve temellerini oluşturan yapılardan bahsedilecektir.

NESNEYE YÖNELİK PROGRAMLAMA YAKLAŞIMI

Gerçek hayattaki problemlerin bilgisayar ortamında çözülebilmesi için öncelikle problemin uygun bir şekilde bilgisayar ortamına aktarılması gerekmektedir. Bu aşama modelleme aşaması olarak adlandırılır. Modellemede aslında, problemin tamamının zihinde canlandırılıp çözmeye çalışılmasından ziyade, oluşturulan model veya modeller üzerinde sistemin görünüşü, davranışı ya da bazı durumlarda verdiği tepkiler gözlemlenmektedir.

Bilgisayar ortamında oluşturulacak modelin gerçek hayattakine benzerliği ve yakınlığı artarsa, programlama da bir o kadar kolaylaşmaktadır. Bu aşamada nesneye yönelik programlama yaklaşımı, gerçek hayattan alınmış problemi çözmek üzere oluşturulacak modelin, gerçekte var olan nesneler ve bu nesneler arasındaki ilişkilerden faydalanılarak oluşturulmasını ilke edinmiştir. Bu yaklaşım tamamen insanı taklit etmektedir. Bu açıdan, son yılların en dinamik ve verimli programlama yaklaşımı olarak da düşünülebilir. Nesneye yönelik programlama yaklaşımı günümüz yaşantısında insanların uzmanlaştığı gibi, kodlama aşamasında ekip içerisinde de farklı görevler oluşturarak bir anlamda kod fabrikaları oluşturulmuştur. Bu örnekte bahsedilen fabrikalar birer "*sınıf (class)*", ürettiği ürünler ise "*nesne (object)*" olup nesneye yönelik programlama yaklaşımının temel kavramlarını oluşturmaktadır.

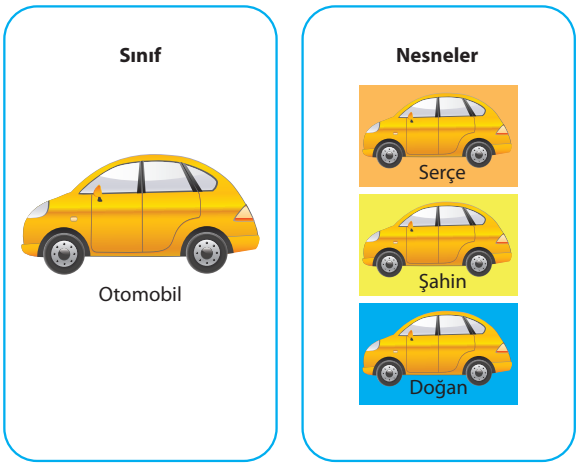
Sınıf (Class), Nesne (Object) Kavramları

Ürünlerin detaylarının belirlendiği, tıpkı sabun kalıbı gibi, sabun kalıbına verilen şekil ve detaylar sonrasında o kalıptan istenilen sayı kadarınca sabun üretilebilmektedir. Buradaki sabun kalıbı sınıfı, o kalıpça üretilen sabunlar ise nesneleri oluşturmaktadır.

Bir otomobil düşünülecek olursa, otomobilin marka ve modeli ne olursa olsun hızlanıp yavaşlayabilen, direksiyonuyla istenilen yöne yönlendirilebilen bir araç hayal edilebilir. Bu özellikler aslında genel davranışlar olup bütün otomobillerde bu davranışları sergilemektedir. Ayrıca, her otomobilin, motoru, lastikleri, kapıları, farları vb. özellikleri de mevcuttur. Her marka ve modelde bu özelliklerin mevcut olduğu bilinmektedir. Şimdi program-

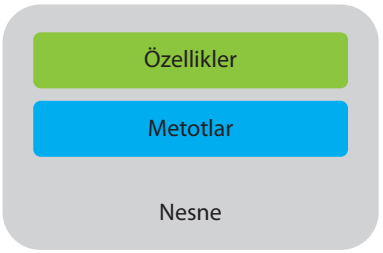
lama dünyasına dönülecek olursa, bu örnekte ki otomobil bir sınıftır. Özetle sınıf, o sınıftan bütün varlıkların ortak özellik ve davranışlarını belirleyen bir yapıdır.

Nesne ise, ait olduğu sınıftan gelen özelliklerin belli olduğu, sınıfı için tanımlı olan davranışları nasıl sergilediğinin bilindiği, somut olarak var olan bir kimliği olan varlık olarak düşünülebilir. Sınıfın yaratılan bir örneğine, örnek nesne (*instance)* ismi verilmektedir. Az önceki örneğimize göre otomobil sınıfına ait olan yaratılan örnek nesne (instance) olarak özel plaka numaralı (örneğin "26 AA 001") bir otomobil de nesnedir. 26 AA 001 plakalı örnek nesne olan otomobil de, markası, modeli, rengi olan, motoru, lastikleri, kapıları, farları bulunan, hızlanabilen ve yavaşlayabilen, direksiyonuyla istenilen yöne yönlenebilen bir varlıktır.



Şekil 1.1 Sınıf ve o sınıftan üretilen nesne kavramı

Her nesnenin belli bir takım özellikleri ve işlevselliğini sağlayan davranışları vardır. Nesneye *yönelik programlama yaklaşımında nesneler, özellikler (properties)* ve *metotlar (methods)* a sahiptir. Nesnelerdeki karakteristikler, özellikleri ve gerçekleştirebileceği eylemler de metot adını almaktadır. Bunu bir örnekle açıklanacak olursa; yukarıdaki örneğimizdeki gibi bir otomobil nesnesinin, hız bilgisi, vites bilgisi, yakıt bilgisi vb. özelliklerini, aracın hızlanması, yavaşlaması, durması gibi eylemlerini de metotlarını temsil etmektedir.



Şekil 1.2 Nesneye yönelik programlama yaklaşımında nesne yapısı

Bir nesnenin yeteneklerine metot ismi verilmektedir. Her bir metot, nesnenin yapabileceği bir *davranışı* simgelemektedir. Önceki sayfalarda bahsedildiği gibi, nesneye yönelik programlama yaklaşımında her bir nesnenin *özellikleri (properties)* ve davranış olarak *metotları (methods)* mevcuttur. Örneğin "otomobil" sınıfına ait bir nesne olarak "murat124" nesnesini düşünülürse, otomobil sınıfı içerisinde özellik olarak tanımlanmış davranışlardan "çalışma", "durma", "hızlanma" eylemleri birer yetenek gibi olup, bu her bir davranış birer metot içerisinde tanımlanmış durumdadır.

Sınıf içerisinde tanımlanmış davranışları simgeleyen metotlar, aynı sınıf içerisindeki özellikleri (properties) değiştirirler. Dolayısıyla objelerde gerçek hayattaki gibi durum ve davranışı olan birer nesne gibi hareketli yapılardır. Metotlar, nesnelere bu hareketliliği sağlayan temel kavramlardır.

Otomobilin hızlanma eylemini gerçekleştirmesi içerisinde örnek olarak hızı 50 km/h, vites bilgisini 3 olarak yakıt bilgisinin ise ilk değerinden bu yana sürekli eksilerek hız özelliğinde değişiklik olacaktır. Benzer şekilde yavaşlama eylemi gerçekleştiği zaman da hız, vites ve yakıt bilgilerinde tekrar değişme olması gerekmektedir. Tıpkı gerçek dünyadaki gibi nesneye yönelik programlama yaklaşımında da benzer bir yaklaşım olmaktadır.

Nesneye yönelik programlama yaklaşımı içerisindeki en önemli yapı nesne olduğu için yapı nesne odaklı olup belirli bir sıraya göre gerçekleşmektedir. Bu sıra içerisinde nesneler tasarlanır, oluşturulur ve yapı içerisinde bu nesneler kullanılır. Nesneler somut veya soyut varlıklar olup birbirleriyle iletişime geçebilmektedir. Örnek olarak "*kalem kutusu*" ismiyle bir sınıf tanımlanırsa, bu sınıfta muhtemelen özellikleri içerisinde sahip olduğu kalemlerin de "*kalem*" sınıfının da olması gerekecektir. Dolayısıyla kalem kutusu sınıfı aynı

zamanda kalem sınıfıyla *ortaklık (association)* kurmak durumundadır. Bu ilişki "*kalem kutusu kalemlere sahiptir*" şeklinde bir ilişki olup *sahiplik (has a)* ilişkisi olarak adlandırılır.

Eğer bir nesne, her zaman bir diğer nesneyi de etkiliyorsa ya da var olmak için diğerine ihtiyaç duyuyorsa o zaman burada da bir *bağımlılık (dependency)* ilişkisi vardır. Bağımlılık ilişkisine örnek olarak otomobil nesnesiyle, tekerlek nesnesini verebiliriz. Tekerleği olmayan bir otomobil düşünülemeyeceği için tekerlek nesnesiyle otomobil nesnesi arasında bağımlılık ilişkisi vardır.

Kitap içerisindeki ilerleyen ünitelerde sınıf ve nesne kavramlarına daha detaylı değinilecektir.

Öğrenme Çıktısı

1 Nesneye yönelik programlama yaklaşımını tanımlayabilme
2 Sınıf ve nesne kavramlarını açıklayabilme
3 Nesnelerin davranış biçimlerini açıklayabilme

Araştır 1

Şekil 1.1 de anlatılan örneğe benzer olarak günlük yaşantınızda kullandığınız başka nesneler için örnekler araştırıp, araştırdığınız örnekler için Şekil 1.2 de gösterilen yapıya göre hangi özellik ve metotların olmasının gerektiğini bulunuz.

İlişkilendir

Yaşantımızda sürekli kullandığımız televizyon nesnesi ile konumuzda anlatılan örnek nesne (instance) kavramını ilişkilendiriniz.

Anlat/Paylaş

Günlük yaşantınızda kullancığınız nesneler için nesneye yönelik programlama yaklaşımı içerisindeki yapıyı arkadaşlarınıza anlatınız.

NESNEYE YÖNELİK ÇÖZÜMLEME VE TASARIM

Çözümleme, sistemin ne yapması gerektiğinin belirlenmesi aşamasıdır. Bu aşamada önemli husus, sistemin tanınması ve istenilen gereksinimlerin iyi anlaşılıp bunların arkasında kalan ve ortaya çıkarılmamış gereksinimlerin belirlenerek çıkarılması ve sistemin kurallarının ve işleyişinin irdelenebilmesidir.

Nesneye yönelik çözümlemede ise, çözümlemenin yapılması aşamasında sınıflardan ve nesnelerden faydalanılmaktadır. Sistemdeki sınıfların özellikleri ve davranışlarının iyi belirlenmesi gerekmektedir.

Tasarım aşaması ise, sistemin *nasıl gerçekleştirileceği* sorusunun yanıtlandığı aşamadır. Bu aşama bir anlamda modelin ortaya çıktığı aşamadır. Model, tasarıma bağlı kalınarak oluşturulur ve çözümleme aşamasında belirlenen gereksinimlerin karşılanıp karşılanmadığı model üzerinde test edilebilmektedir.

Nesneye yönelik tasarım aşamasında modelin oluşturulması için sınıflar ve nesneler kullanılır. Sınıflar arasındaki ilişkiler incelenmekte, gereksinimleri karşılanmak üzere sınıf özellikleri ve davranışları şekillendirilmektedir. Tasarım sonlandırılırken, ortaya nesneye yönelik bir model çıkmaktadır. Bu modelin görsel olarak ifade edilmesi ve daha iyi anlaşılması için bir takım görsel diller kullanılmaktadır. Bu diller arasında en popüler olanı *Birleşik Modelleme Dili (Unified Modelling Language - UML)*'dir. Tasarım aşaması, genellikle sistemin çeşitli açılardan görünüşlerini ifade eden UML diyagramlarıyla sonlanır.

Birleşik Modelleme Dili (Unified Modelling Language – UML)
İş sistemlerinin nasıl modellenebileceğini belirleyen ve açıklayan yöntemlerin bir araya toplanmış halidir. Daha çok yazılım geliştiriciler tarafından kullanılan UML ile nesneye yönelik programlamada temel teşkil eden sınıflar oluşturulabilir ve sınıfların geliştirilmesi aşamasında sağlanan görselliklerle daha kolay anlaşılabilir olmaktadır.

Bu aşama sonrasında ortaya çıkan model, kullanılacak olan uygun programlama diliyle ifade edilecek şekilde kodlanır. Model içerisindeki her aşamanın programlama dilinde bir karşılığı olmak zorundadır. Seçilen herhangi bir nesneye yönelik programlama diliyle o model kodlanabilir. Bir model birçok farklı programlama diliyle ifade edilebilir.

Önemli olan sistemin hangi programlama diliyle geliştirileceği değil, sistem için gerekli olan modelin oluşturulabilmesidir. Sonraki aşamada programlama dili seçimi, ekibin bilgi birikimi, kurumun yatırım ve deneyimleri, sistemin kullanılacağı kişi ve kişilerin özel gereksinimlerine göre yapılmaktadır.

Nesneye Yönelik Tasarım Ölçütleri

Öncelikle tasarım yapısının ayrılabilir, birleştirilebilir, anlaşılabilir ve korunabilir olması gerekmektedir. Bu kavramları kısaca açıklamak gerekirse, ayrılabilirlikten kasıt anlamlı parçalara ayrılabilmesini ifade etmektedir. Burada bahsedilen parça aslında sınıfı temsil etmektedir. Bir problemin alt problemlere bölünebilmesi tasarımın ayrıştırılabilmesiyle mümkün olmaktadır.

Birleştirilebilir olması, sınıfı temsil eden parçanın başka tasarımlarda da tekrar tekrar kullanılabilmesi yani diğer parçalarla birleştirilebilmesi anlamını taşımaktadır. Tasarımın anlaşılabilirliği, bir sınıfın diğer sınıflarla bilgiye gerek duymadan anlaşılabilmesi ve aynı zamanda yapılacak küçük değişikliklerin etkilerinin en az sayıda sınıfa yayılarak uyarlanmasını ifade etmektedir. Son olarak korunabilirlik ise, olası hatalara karşı düzeltilebilmesine yönelik etkilerin geniş bir alana yayılmadan önlenebilmesidir.

Nesneye Yönelik Tasarım İlkeleri

Nesneye yönelik tasarım ilkelerini kısaca şu şekilde sıralamak mümkündür:

- Düşük Bağlılık (Low Coupling)
- Yüksek Uyum (High Cohesion)

Düşük Bağlılık (Low Coupling)

Bir sınıfın diğer sınıflarla bağlılık oranı olarak düşünülebilir. Burada bahsedilen bağlılık aslında kalıtım ilişkisinin bulunmasıyla birlikte diğer sınıfların içyapısından haberdar olması, çeşitli hizmetlerinden yararlanabilmesi ve çalışma prensibinden haberdar olmasını ifade etmektedir. Tabi burada ilişkide bulunulan diğer sınıfların sayısının artması bağlılık oranını arttıran faktörlerdendir. Düşük bağlılığın en önemli faydası bir sınıfta yapılan değişikliğin diğer sınıfların az etkilenerek uyarlanmasının sağlanmasıyla birlikte yeniden kullanılabilirliğin artması olarak söylenebilir.

Yüksek Uyum (High Cohesion)

Burada bahsedilen uyum aslında bir sınıfın sorumluluklarının birbirleriyle uyumlu olma oranı olarak düşünülmektedir. Uyum, modüllerin veya sınıfların birbirine olan benzerliği yani aynı amaca hizmet eden kod bloklarının bulunması anlamını taşımaktadır. Aslında düşük bağlılık ile ters orantıda olan bir kavram olup bağlılığın az olması uyumun artması anlamını taşımaktadır. Bir sınıfın yeniden kullanılabilirliğinin artması, değişikliklerden etkilenmesinin azalması, yüksek uyumun beraberinde getirdiklerinden bazılarıdır. Bu kavramlar bir yazılımın kalitesini yazılım geliştirme yaşam döngüsünü oluşturan kavramların anlatıldığı yazılım mühendisliği gibi derslerde daha detaylı olarak anlatılmaktadır.

Öğrenme Çıktısı

4 Nesneler arası ilişkileri açıklayabilme
5 Nesneye yönelik çözümleme ve tasarım kavramlarını açıklayabilme

Araştır 2

Hayatımızda kullandığımız veya yaşadığımız olayları düşünerek bir düşük bağlılık örneği araştırınız.

İlişkilendir

Bir kullanıcının atm üzerinden yapacağı işlemleri UML diyagramı üzerinde ilişkilendirerek gösteriniz.

Anlat/Paylaş

Hayatımızda karşılaştığımız problemlere karşı nasıl çözüm üretilmesi gerektiğini ve bu çözüm aşamasında problemi analiz etme ve çözüm üretme adımlarının nasıl belirlendiğini çevrenizdekilere anlatınız.

NESNEYE YÖNELİK PROGRAMLAMA TEMELLERİ

1990'lı yıllarda başlayan ve günümüze kadar yoğun olarak kullanılan nesneye yönelik programlama, geliştirilen yazılım üzerinde *bakım (maintenance)*, *genişletilebilirlik (extensibility)* ve *kodun yeniden kullanılabilirliği (reusability)* sağlamaktadır. Kısaca bu yararları açıklayacak olursak;

Yazılımın bakım kolaylığı; nesnelerin ait oldukları sınıflarca sunulduğu şablonlarda temsil edildiği için benzer veya aynı konuyla ilişkili sınıflar bir araya getirilerek sınıf kümeleri oluşturabilmektedir. Bu durumda yazılımın bakımı, ya mevcut sınıflarda değişiklik ya da yeni sınıf eklenmesi anlamına gelmektedir. Bu da yazılımın tamamını hiçbir şekilde etkilememekte, dolayısıyla bakım kolaylığı getirmektedir.

Genişletilebilirlik; mevcut bir sınıfa yeni özellik veya yapılar ekleyerek artan işlevsellik sağlanmasıdır. Bu da nesneye yönelik programlama yaklaşımında kolaylıkla yapılabilmektedir.

Kodun yeniden kullanılabilmesi; üretilen sınıfların her uygulama geliştiricinin kullanımına açık olan bir ortak kütüphanede tutulmasına imkân sağlaması, her kullanıcının bu ortak kütüphanede bulunan sınıfları kullanabilmesi ve dolayısıyla benzer kodları yeniden yazmaktan kurtulması anlamındadır.

Nesneye yönelik programlama yaklaşımının üç temel ilkesi vardır. Bunlar:

- Sarmalama (Encapsulation)
- Kalıtım (Inheritance)
- Çok biçimlilik (Polymorphism)

Sarmalama (Encapsulation)

Yazılan kodun değiştirilebilirliğini sağlamaktadır. Nesneye yönelik programlama içerisinde, bir sınıf içerisindeki nitelikler, programın çalışması sırasında nesnelerin durumlarını oluşturmaktadır. Bir nesnenin durumu ise her zaman anlamlı ve tutarlı olmak zorundadır. Nesne ilk oluşturulduğu anda nesnenin ilk durumunu anlamlı kılabilmek için birtakım işlemler yapılmaktadır. Bu işlemler *"Kurucular (Constructors)"* ismiyle adlandırılan metotlar vasıtasıyla yapılmaktadır. Bu kavrama ilerleyen ünitelerde ayrıntılarıyla değinilecektir.

Nesneye yönelik programlama yaklaşımı içerisinde hemen hemen bütün sınıflar başka bir sınıf tarafından kullanılmak amacıyla üretildiği için bazı özellikler ve metotlar sadece dışarıdan kullanılmak üzere hazırlanmaktadır. Buna karşılık bazı özellikler ve metotlar ise diğerlerine yardımcı olmak, sadece sınıfın iç işlerinde kullanılmak için yazılmaktadırlar. Belli bir sınıfı kullanan diğer sınıfların bunları görmesi veya bilmesi gerekme-

mektedir. Hatta bazı durumlarda bu özellikler ve metotların sadece ufak bir kısmı diğer sınıflar tarafından kullanılmaktadır.

Bir yazılımcının kod geliştirme aşamasında nesneye yönelik programlama yaklaşımı içerisinde belli bir sınıfa baktığında sınıfın özellikleri ve metotlarının hangilerinin kendisine yarayacağını hangilerinin sadece başka metotlarda kullanılacağı ve sınıfa özel olduğunu görmesi zor olabilir. Bazı durumlarda güvenlik ve sağlamlık amacıyla bir sınıfta kullanılan özelliklerin değiştirmesi ve bazı metotlara erişmesi engellenmek istenilebilir.

İşte bütün bu istenilen durumlar için erişim sınırlandırma işlemine sarmalama *(*encapsulation) adı verilmektedir. Sarmalama işleminin yapılabilmesi için *public, private, protected* gibi kelimeler kullanılan yazılım dili içerisinde bulunmaktadır. Bu kelimeler kısaca "*erişim değiştirici (access modifier)*" olarak bilinmektedir. Sınıf içerisindeki özellikler ve metotların başına bu kelimeler getirilerek erişim sınırlandırma işlemi yani sarmalama yapılabilmektedir.

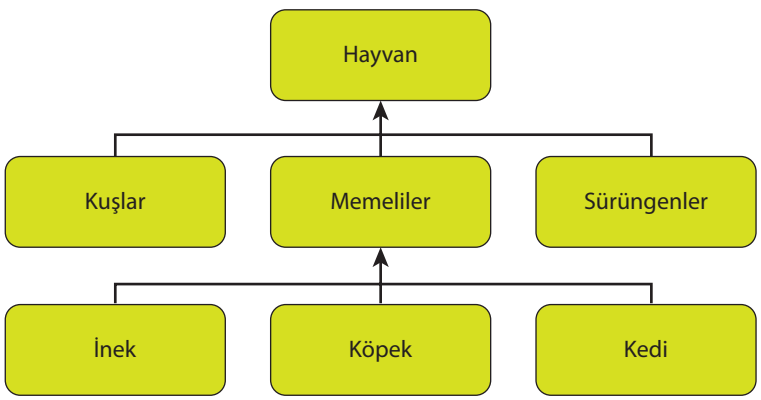
Sarmalama yapısının erişim kısıtlama olduğunu akıllarda tutmak için şu örnek verilebilir. Eski nesil tüplü televizyonlarda birtakım çok kullanılan tuşlar televizyon üzerinde herkesin gözü önünde görülebilecek şekilde yerleştirilmiştir. Ancak bazı özel durumlarda kullanılacak olan, örneğin renk derinliğini, karşıtlığı değiştirme vb. tuşlar ise direk görünürde değil bir kapak içerisinde daha gizli bir ortamda bulunmaktadır. Böyle bir televizyona yeni yürümeyi öğrenmiş bir bebek yaklaştığında, görünürde olan tuşlara rahatça erişebilmekte ancak kapak içindekilere ulaşamayacaktır. Ama bir yetişkin insan televizyondaki karşıtlık değerini değiştirmek için kapak altındaki karşıtlık değiştirme tuşlarına erişebilmektedir. İşte bu yapı bir anlamda nesneye yönelik programlama içerisindeki sarmalama mantığına benzemekte olup erişim kısıtlama yapısına örnek oluşturmaktadır.

Kalıtım (Inheritance)

Kalıtım yapısı, adından da anlaşılacağı üzere birtakım özelliklerin miras alınabilmesi anlamındadır. Yani, bir sınıfın başka bir sınıftaki özelliklere ve metotlarına sahip olmasıdır. Bir anlamda kodun yeniden kullanılabilirliğini sağlayan yapıdır.

Kalıtımla, bazı durumlarda başka birisinin yazdığı sınıfa bazı eklemeler yaparak belli bir işlem için kullanılır hale getirmek için de kullanılabilmektedir. Böylelikle sadece o sınıfta olmayan özellikler ve metotları eklemek suretiyle çok kısa sürede gelişmiş bir sınıf oluşturulabilmektedir. Bazen de ortak birçok özellik içeren iki sınıfta da bulunan özellikleri ve metotları tanımlamaktan kurtulmak için kullanılabilir.

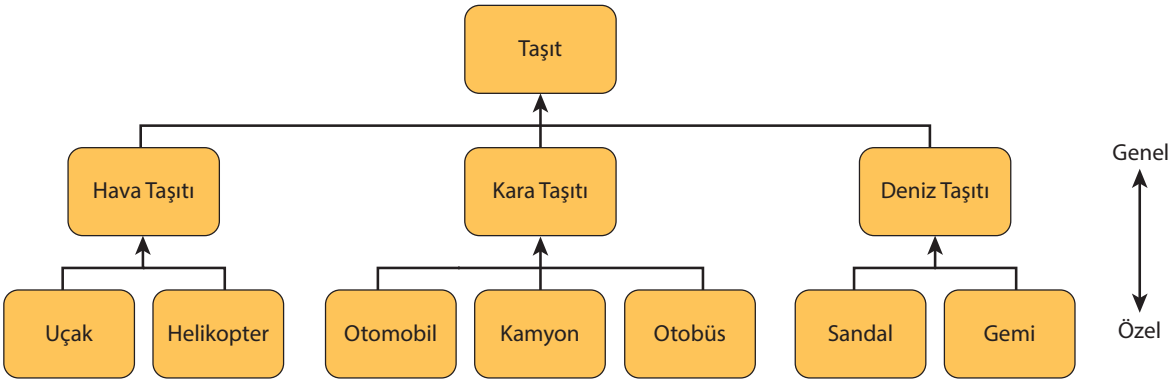
Biraz farklı bir örnekle kalıtım yapısını açıklamak gerekirse, "hayvan" ismiyle bir sınıfımız olduğunu düşünelim, "memeliler", "sürüngenler", "kuşlar" da diğer sınıflarımız olsun. Memeliler, sürüngenler ve kuşlar sınıflarının farklı farklı özellikleri olduğu gibi hepsinin hayvan olmasından dolayı birtakım ortak özellikleri de mevcuttur. Bu sebeple, memeliler, sürüngenler ve kuşlar birer hayvandır diyebiliriz. Yani kısacası memeliler, sürüngenler ve kuşlar, hayvan sınıfından türetilmiş ve her birinin kendine özgü özellikleri vardır diyebiliriz.



Şekil 1.3 UML diyagramıyla Kalıtım örneği

Kalıtım, programlama ortamında da gerçek hayattaki tanımına benzer bir işi gerçekleştirmektedir. Bir sınıfın başka bir sınıftan kalıtım yapması demek, kalıtımı yapan sınıfın diğer sınıftaki nitelik ve metotları kendisine alması demektir. Eğer kalıtımı yapan sınıfa "*alt sınıf*", kendisinden kalıtım yapılan sınıfa "*ata sınıf*" dersek, ata sınıfta tanımlı olan her şeyin alt sınıf için de tanımlı olduğunu söyleyebiliriz.

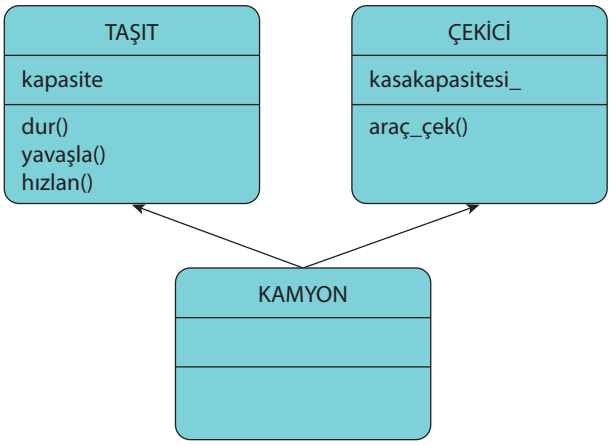
Kalıtım ilişkisi, UML dili içerisinde sembolü ↑ ile ifade edilmektedir ve birden fazla düzeyli olacak şekilde de kurulabilmektedir. Aşağıda örnek bir sınıf ve o sınıfın alt sınıflarından bazılarının gösterildiği bir kalıtım ağacı görülmektedir.



Şekil 1.4 UML diyagramıyla Taşıt sınıfının kalıtım ağacı

Kalıtım hiyerarşisinde yukarıya doğru çıkıldıkça genele, aşağıya doğru inildikçe özele doğru gidilmektedir. UML diyagramlarında kalıtımı belirten ↑ sembolü, kendisinden kalıtan *sınıf yönünde izlendiği zaman, genele gidilmektedir.*

Akıllara "bir sınıf birden fazla sınıftan birtakım özellik ve metotları miras olarak alabilir mi?" şeklinde bir soru da gelebilir. Bu soru *çoklu kalıtım (multiple inheritance)* kavramını ifade etmektedir. Bir örnekle açıklanacak olursa, bir taşıt sınıfı tanımlansın, bu taşıt sınıfının alt sınıfı olarak taşıt sınıfından türetilmiş olan otomobil, gemi, uçak ve çekici sınıfları olsun. Bir kamyon düşünün ki hem taşıt sınıfı özelliklerini taşısın hem de çekici sınıfı özelliklerini üzerinde barındırsın. Bu yapı kalıtım yoluyla miras alacağı özellik ve metotları üst sınıf olarak birden fazla sınıftan alabilmeyi içermektedir.



Şekil 1.5 UML diyagramıyla çoklu kalıtım yapısı

Birden fazla üst sınıftan farklı özellik ve metotları kalıtım yoluyla miras alarak üzerinde barındıracak şekilde sınıf tanımlama yapısı C++ gibi dillerde mümkün olup bu ders kitabında anlatılacak olan Java dilinde doğrudan mümkün değildir. Çoklu kalıtım yapısı tasarım açısından biraz daha karmaşık görünebilmektedir. Java dilinde çoklu kalıtımın olmaması anlamına gelmemektedir. Java'da çoklu kalıtım yapısı dilin kendine has birtakım özellikleriyle, *arayüz (interface)*, *soyut sınıf (abstract class)* gibi yapılarla uygulanabilmektedir. Bu konular da kitabın ilerleyen ünitelerinde ele alınacaktır.

Çok Biçimlilik

Çok biçimlilik, nesnelerin içeride farklı çalışmalarına rağmen, dışarıdan aynı biçimde görünmelerini ifade etmektedir. Bu şekilde, bir grup nesneyi kullanan sınıflar kalıtımla ilgili detayları bilmek zorunda kalmamakta, içerideki değişikliklerden etkilenmeden çalışmaya devam etmektedirler. Aynı sınıftan türetilen sınıflar standart bir şekilde erişilebilme özelliklerine sahip olmaktadır.

Aslında çok biçimlilik, bir nesnenin bir işlemi farklı şekillerde yapabileceğini göstermektedir. Birbirine benzeyen nesneleri ortak özellikleriyle ele alarak ya da nesnelerin aynı işi farklı şekillerde yapabilmesini sağlamaktadır. Bu kavram bir yazıcı örneğiyle açıklanacak olsun ve yazıcının hem renkli çıktı hem de renksiz çıktı alabildiği varsayılsın. Burada yapılan asıl işlev yazıcının çıktı verme işlevi olan "*yazdır*" metodudur. Yazıcı sınıfı barındırdığı yazdır metodu, içerisinde alabilecek değişken parametrelerle farklı tiplerde çıktı üretimini sağlamaktadır. Kullanıcının burada bilmesi gerektiği işlev yazdırma işlevidir. İşlev aynı olup farklı tiplerde farklı çıktı üretiminin sağlanabilmesi yaklaşımı çok biçimlilik örneğidir.

Farklı bir örnek olarak, *ata sınıf* olarak "*asker*" *sınıfı düşünülecek olursa, bu ata sınıftan kalıtım yoluyla türetilen alt sınıflar* olarak "*er*", "*çavuş*" ve "*teğmen*" sınıfları bulunsun. Ata sınıf içerisinde de örnek metot olarak "*selam ver*" metodu işlev olarak asker sınıfı nesnelerine selam verme işlevini yerine getirmek üzere yazılmış olsun. Örnek eylem olarak, rütbeli bir paşanın birliğe geldiği bir olay varsayılsın. Bu durumda kıtadaki bütün askerlerin paşaya selam vereceği eylem için "*asker selam ver*" tipinde bir işlevi sağlamak anlamında, bütün askerlerin selam verme metotlarını çalıştırmak için, her rütbedeki (er, çavuş ve teğmen) sınıflarının "*selam ver*" metotları aynı isimde ve aynı ata sınıftan türetildiği için tek bir metot çağırımıyla gerçekleşebilmektedir. İşte bu işlev de bir anlamda çok biçimlilik olmadan yapılamayacak bir işlevdir.

Öğrenme Çıktısı

6 Nesneye yönelik programlama temellerini açıklayabilme
7 Sarmalama, kalıtım, çok biçimlilik kavramlarını açıklayabilme

Araştır 3

Ata sınıf olarak televizyon sınıfını düşünerek bu sınıfa ait olan alt sınıfların neler olması gerektiğini araştırınız. Televizyon sınıfından türetilebilecek olan alt sınıfları belirleyiniz.

İlişkilendir

Nesneye yönelik programlama kavramlarından kalıtım ve çoklu kalıtım kavramlarıyla bir bebeğin dünyaya gelmesiyle ailesinden ve çeşitli akrabalarından aldığı benzerlik ve davranışları aldığı genetik kalıtım yapısını ilişkilendiriniz.

Anlat/Paylaş

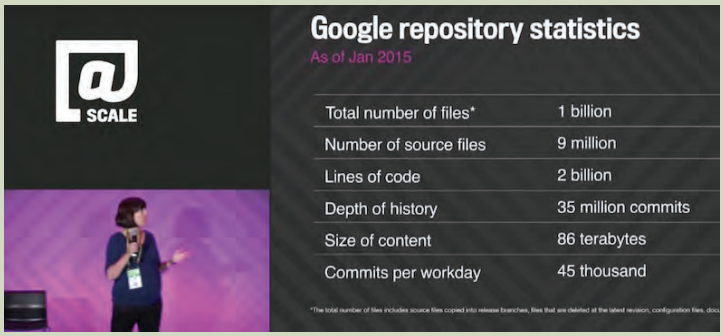
Genetik bilimiyle nesneye yönelik programlama kavramları arasındaki ilişkileri arkadaşlarınıza anlatınız.

Yaşamla İlişkilendir

Tüm Google servisleri 86 terabaytlık 2 milyar satır kod içeriyor!

Geçtiğimiz günlerde Google Mühendislik Müdürü Rachel Potvin, Silikon vadisindeki bir konferansta açıkladığı rakamlarla yeni bir Google resmi çizmemizi sağlıyor. Potvin, 30 dakikalık konuşmasında Google hakkında özellikle yazılım geliştiricilerin ilgisini çekecek bilinen ve bilinmeyen bilgileri paylaşıyor. Google'ın paylaşımlı tek bir devasa bir kod bütünü üzerinde çalıştığını söyleyen Potvin, bunun ilk bakışta çılgınca göründüğünü söylüyor ve böyle bir sistemin neden gerekli olduğunu da açıklıyor.

Google'ın tüm internet servisleri 1 milyar dosya içindeki 2 milyar satırlık bir kod bütünü oluşturuyor. Microsoft'un 2011'de açıkladığı rakamlara göre Windows XP işletim sistemi sadece 45 milyon satır koddan oluşuyordu. Açık kaynaklı işletim sistemi Linux ise 15 milyon satır kod ve 40 bin dosyadan oluşuyor. Facebook ise 2013'de 30 milyon satır koda ulaşmıştı.



85 terabayt (85 bin gigabayt) boyutundaki havuz içinde Google mühendisleri her gün 45 bin kod değişikliği yapıyor ve her hafta 15 milyon satır kod taşıyan 250 bin dosya içinde değişiklikler yapılıyor.

Elbette kod bütünün sadece niceliksel açıdan yorumlamak doğru olmaz ama bu durum Google'ın ne kadar büyük bir gezegen olduğunu görmemizi sağlıyor. Github Sistem Yöneticisi Sam Lambert de bu rakamları 'kesinlikle sarsıcı' olarak tanımlıyor.

Bu noktada Google servisleri altında arama, harita, dokümanlar, takvim ve hatta Youtube gibi bir çok hizmetin yer aldığını hatırlatmakta fayda var. Ve tüm bu servisleri kontrol eden 2 milyar satırlık kod bütünü 25 bin Google mühendisinin elinden geçiyor.

Google kod bütününü de bir çeşit işletim sistemi gibi çalışıyor ve Potvin, ispatlayamayacağını ama Google'ın dünyadaki en büyük tekil kod bütünü üzerinde çalıştığını ifade ediyor.

Kaynak: http://webrazzi.com/2015/09/17/google-servisleri-86-terabayt-2-milyar-satir-kod/

öğrenme çıktıları ve bölüm özeti

1. Nesneye yönelik programlama yaklaşımını tanımlayabilme
2. Sınıf ve nesne kavramlarını açıklayabilme
3. Nesnelerin davranış biçimlerini açıklayabilme

Nesneye Yönelik Programlama Yaklaşımı

1 Uygulamaların gün geçtikçe arttığı, birçok isteğin ve problemin çözüldüğü günümüz dünyasında, yazılım açısından oluşan karmaşıklık ve boyutların artması karşısında yazılımın niteliğinin korunması, gerekli bakım maliyetleri, harcanan zaman vb. unsurların artmasına karşılık nesneye yönelik programlama yaklaşımı doğmuştur.

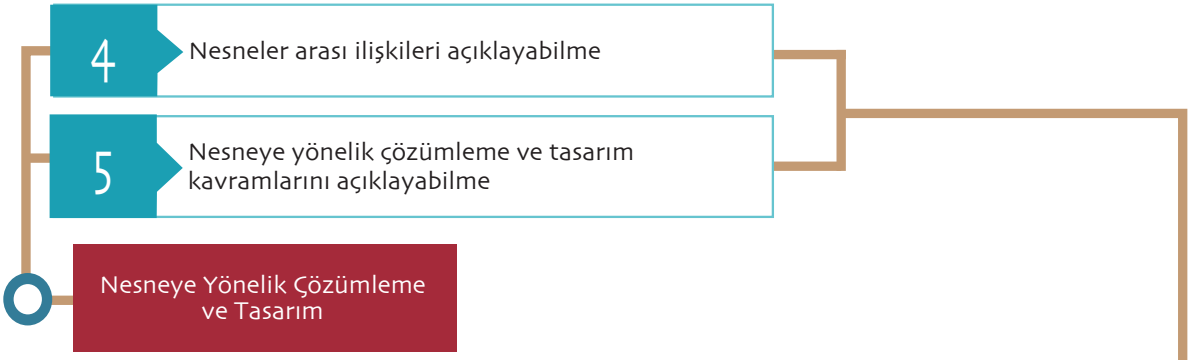
Gerçek hayata olan benzerlikle öne çıkan bu yaklaşım içerisinde, kodlama aşamasında da ekip çalışması ve farklı görevler oluşturma üzerinde durulmuştur. Gerçek hayat modelini oluştururken nesneler ve bu nesneler arasındaki ilişkilerden faydalanılmaktadır. Sınıf ve nesne kavramları üzerine kurulu bu yaklaşım, geliştirilen yazılımın üzerine yeni özelliklerin eklenmesinde, önceki özelliklerin yeniden kullanılabilirliğini sağlayabilmekte ve zaman geçtikçe de bu tür değişimlerle geliştirilebilirlik prensibini taşıyabilmektedir.

2 Gerçek hayattan esinlenilen nesneye yönelik programlama yaklaşımında sınıf kavramı, yazılım içerisinde kullanılacak bütün nesnelerin tasarımının yapıldığı, içerisindeki özelliklerinin belirlendiği ve metotlarla davranışlarının temelinin atıldığı yapılardır. Oluşturulan bu tasarım sonucunda, buna uygun oluşturulan her bir nesne de uygulama içerisinde istenilen özellik ve davranışları sergileyen birer varlık olarak kullanılabilecek durumdadır.

Yazılımın oluşturulma amacı üzerinde çözülecek problemler ve yapılar bu sınıf ve nesne kavramlarıyla ifade edilerek, aralarındaki ilişkiler ve tasarım yapılarıyla problemin çözümünde ve yazılımın geliştirilmesinde büyük kolaylık sunmaktadır.

3 Oluşturulan nesnelerin yazılım dünyasında kullanılması aşamasında kendi yeteneklerinin ve davranış biçimlerinin belirlendiği yapılara metot ismi verilmektedir. Tanımlanan metotlarla o nesnenin özellik değerleri değiştirilebilmekte ve nesnelere gerçek hayattaki gibi durum ve davranış biçimi verebilmektedir.

Kedi sınıfına ait örnek bir nesne olarak tekir isimli Van kedisi düşünülecek olsun. Tekirin gözlerinin rengi, tüylerinin rengi, kilosu, boyu birer özellik olarak tanımlanmakta, koşabilmesi, ağaca tırmanabilmesi, av yakalayabilmesi gibi davranış şekilleri de birer metot olarak tanımlanmaktadır.



4 Nesneye yönelik programlama yaklaşımı içerisindeki en önemli unsur olan nesneler, öncelikle tasarlanır, sonra oluşturulur ve akabinde sistem içerisinde kullanılırlar. Oluşturulan nesneler somut veya soyut yapılar olabilmektedir. Aralarında ilişkiler olabilmektedir. Bir sınıfın bir başka sınıfla ortaklık kurması ve sahiplik kurmasıyla mümkün olmaktadır. Çiçek nesnesiyle çiçeklik nesnesi gibi, çiçeklik, çiçekler için vardır.

Bir nesneyle diğer bir nesne sürekli etkileşim halindeyse yani varlığı diğer bir nesneye bağlıysa, bağımlılık ilişkisi vardır. Bir firmanın bölümü ve bölüm çalışanları düşünülecek olursa, çalışanı olmayan bir bölüm düşünülemez.

5 Çözümleme aşaması, sistemin ne yapması gerektiğinin belirlendiği aşamadır. Sistemin tanınması, gereksinin analizinin yapılması ve bunun sonucunda sistemin kural ve işleyişinin belirlenmesi bu aşamada yapılmaktadır.

Çözümleme ise, sınıf ve nesne yapıları kullanılarak nesneye yönelik gerçekleşmekte, sistemin nasıl gerçekleştirileceğinin belirlendiği, sistem modelinin oluştuğu aşamadır. Bu model oluşturulma kısmında birleşik modelleme dili olan UML dili sıklıkla kullanılmaktadır.

Nesneye yönelik tasarım ölçütleri ve ilkeleriyle birlikte bir yazılımı oluşturan temel unsurlardandır. Düşük bağlılık ve yüksek uyuma bağlı kalarak yapılan tasarımlar UML dili ile belgelendirilerek sağlam temeller üzerinde kodlanmalıdır.

öğrenme çıktıları ve bölüm özeti

6 Nesneye yönelik programlama temellerini açıklayabilme

7 Sarmalama, kalıtım, çok biçimlilik kavramlarını açıklayabilme

Nesneye Yönelik Programlama Temelleri

6 Uygulama geliştirme sürecinde bakım, genişletilebilirlik ve kod yeniden kullanılabilirliği özellikleriyle nesneye yönelik programlama yaklaşımı günümüzün popüler yaklaşımlarındandır. Bu yararlı özellikler ise sarmalama, kalıtım ve çok biçimlilik temel ilkeleri kullanılarak uygulanmaktadır.

7 Nesneye yönelik programlama temel ilkelerinden olan sarmalama ilkesi, erişim kısıtlama olarak da bilinmektedir. Belli bir sınıfın özellik ve metotlarının diğer sınıflarca direk kullanımının kısıtlandığı veya kademeli erişim olanaklarının sağlandığı yapıdır. Erişim değiştiricilerle belirlenen bu kısıtlı erişim olanağı, sistemin güvenliğini ve sağlamlığını arttıran unsurlardandır.

Kodun yeniden kullanılabilirliğinin arttırıldığı, önceden tanımlanmış sınıflar ve o sınıfa ait birtakım özellikleri içeren alt sınıfların oluşturulabildiği, bir ebeveyn ilişkisi içerisinde düşünülebilen yapı ise kalıtım yapısıdır. Birden fazla sınıftan bazı özelliklerin alınarak yeni bir sınıfın oluşturulmasıyla da çoklu kalıtım adı altında da bir yaklaşım daha mevcuttur ancak tasarım görünümü olarak da çoklu kalıtım yöntemi normal kalıtıma nazaran daha karmaşıktır. Java programlama dilinin direk olarak desteklemediği çoklu kalıtım yapısı, daha anlaşılır bir tasarım haliyle java içerisinde, arayüz, soyut sınıf gibi yapılarla mümkün olabilmektedir.

Kalıtım ilkesiyle birlikte düşünülebilen çok biçimlilik ilkesi birbirlerine bağımlı yapılardır. Kalıtım olmadan, çok biçimlilik mümkün olamamaktadır. Nesnelerin içeride farklı çalışmalarına rağmen, dışarıdan aynı biçimde görüşmesi üzerine yatan bu yapıda, nesneyi kullanan sınıflar detayları bilmek zorunda değildir.

1 Aşağıdakilerden hangisi nesneye yönelik programlama yaklaşımının çıkış sebeplerinden biri **değildir?**

A. Yazılımların karmaşıklığı
B. Yazılımların boyutlarındaki artış
C. Yazılımların bakım maliyetleri
D. Yazılım geliştirici sayısındaki artış
E. Yazılım geliştirme sürecinde harcanan zaman artışı

2 Aşağıdakilerden hangisi geçmişten günümüze kullanılmış yazılım geliştirme yaklaşımlarından biri **değildir?**

A. Nesneye Yönelik Programlama
B. Yapısal Programlama
C. Bileşen Tabanlı Yazılım Geliştirme
D. Görünüm Yönelimli Programlama
E. Metotsal Programlama

3 Aşağıdakilerden hangileri nesneye yönelik programlama yaklaşımı içerisindeki temel kavramlarındandır?

A. Nesne - Metot
B. Metot - Sınıf
C. Sınıf - Nesne
D. Sınıf- Özellik
E. Metot – Örnek Nesne

4 Aşağıdakilerden hangisi bir nesnenin yeteneklerinin belirlendiği, nesnenin yapabileceği davranışları oluşturan kısımdır?

A. Sınıf
B. Örnek Nesne
C. Metot
D. Özellik
E. İlişki

5 Aşağıdakilerden hangisi "sabun" nesnesi için örnek nesne olarak düşünülebilir?

A. Sabun kalıbı
B. Sabunluk
C. Lavanta kokusu
D. Su
E. Lavanta kokulu sabun

6 Aşağıdakilerden hangisi "sabun" nesnesi için bir sınıf olarak düşünülebilir?

A. Sabun kalıbı
B. Sabunluk
C. Lavanta kokusu
D. Su
E. Lavanta kokulu sabun

7 Aşağıdakilerden hangisi "sabun" nesnesi için bir özellik olarak düşünülebilir?

A. Sabun kalıbı
B. Sabunluk
C. Lavanta kokusu
D. Su
E. Lavanta kokulu sabun

8 Aşağıdakilerden hangisi nesneye yönelik programlama yaklaşımının temel ilkeleridir?

A. Metot – Özellik - Sınıf
B. Sarmalama – Kalıtım – Çok biçimlilik
C. Sarmalama – Sınıf - Kalıtım
D. Sınıf – Kalıtım – Çok biçimlilik
E. Sınıf – Nesne - Kalıtım

9 Kedi sınıfı için "ata sınıf - alt sınıf" ikilisi olarak aşağıdakilerden hangisi uygundur?

A. Van kedisi - Jaguar
B. Memeli hayvan – Yumuşak
C. Yumuşak - Memeli
D. Memeli hayvan – Van kedisi
E. Jaguar – Van kedisi

10 Otomobil, televizyon, bilgisayar ve telefon sınıflarını düşünecek olursak, aşağıdakilerden hangisi çok biçimlilik işlevi örneğidir?

A. Hızlan
B. Renk
C. Sesi aç
D. Ekran çözünürlüğü değiştir
E. Çalış

neler öğrendik yanıt anahtarı

1. D	Yanıtınız yanlış ise "Giriş" konusunu yeniden gözden geçiriniz.
2. E	Yanıtınız yanlış ise "Giriş" konusunu yeniden gözden geçiriniz.
3. C	Yanıtınız yanlış ise "Nesneye Yönelik Programlama Yaklaşımı" konusunu yeniden gözden geçiriniz.
4. C	Yanıtınız yanlış ise "Sınıf, Nesne Kavramları" konusunu yeniden gözden geçiriniz.
5. E	Yanıtınız yanlış ise "Sınıf, Nesne Kavramları" konusunu yeniden gözden geçiriniz.
6. A	Yanıtınız yanlış ise "Sınıf, Nesne Kavramları" konusunu yeniden gözden geçiriniz.
7. C	Yanıtınız yanlış ise "Sınıf, Nesne Kavramları" konusunu yeniden gözden geçiriniz.
8. B	Yanıtınız yanlış ise "Nesneye Yönelik Programlama Temelleri" konusunu yeniden gözden geçiriniz.
9. D	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.
10. E	Yanıtınız yanlış ise "Çok Biçimlilik" konusunu yeniden gözden geçiriniz.

1 Araştır Yanıt Anahtarı

Araştır 1

Günlük hayatta kullanıcılar mobil cihazları daha çok eğlence ve sosyal medya uygulamalarını kullanma amacıyla kullanmaktadırlar. Günümüz en popüler sosyal medya uygulamaları Twitter ve Facebook uygulamaları mobil uygulamaya örnek gösterilebilir.

Araştır 2

Hayatımızda sürekli alışveriş yapıyoruz. Düşük bağlılık örneği olarak KrediKArtı sınıfı düşünelim. Bu sınıfın örneği (instance) olarak da Ödeme sınıfımız olsun. Ödeme yapma işlemini kredikartı sınıfı içerisinde tanımlamak gerekirse bu işlemi sadece kredi kartıyla değil gerek sms ile ödeme gerekse PayPal gibi kuruluşlarla ödeme gibi bir opsiyonun da kullanılabilirliğini düşünerek ÖdemeAracı isimli bir arayüz kullanılması düşük bağlılığı yani tek bir kredikartı sınıfına olan bağlılığı beraberinde getirmektedir.

Araştır 3

Ata sınıf olarak "televizyon" sınıfı ve bunun alt sınıfları olarak "tüplü tv", "lcd TV", "led TV" ve "led TV" nin alt sınıfları olarak, "3d led TV", "akıllı led TV", "tüplü TV" nin alt sınıfları olarak "rensiz tüplü TV", "renkli tüplü TV", "teletextli TV" düşünülebilir.

Kaynakça

Liang, Y.D. (2015) **Introduction to Java Programming Comprehensive Version 10th Edition,** New Jersey, Pearson

İnternet Kaynakları

docs.oracle.com

en.wikipedia.org

tr.wikipedia.org

gelecegiyazanlar.turkcell.com.tr

webrazzi.com

Bölüm 2

Java Programlama Diline Giriş

Öğrenme çıktıları

Java Platformu

1 Java programlama dilini açıklayabilme
2 Java'nın temel özelliklerini açıklayabilme
3 Java platformunu tanımlayabilme
4 Derleme ve yorumlama kavramlarını açıklayabilme
5 Java sanal makinesini, Java çalışma ortamını, Java geliştirme paketini, Java API ve diğer bileşenleri açıklayabilme ve kullanabilme

Java Tümleşik Geliştirme Ortamı Kullanımı

6 Java Tümleşik geliştirme ortamını açıklayabilme
7 Eclipse ortamını kullanabilme

Java Sözdizimi ve İlk Java Uygulaması Yazımı

8 Java sözdizimini açıklayabilme
9 Temel Java uygulaması yazabilme

Anahtar Sözcükler: • Java Programlama Dili • Java Çalışma Ortamı (JRE) • Java Geliştirme Paketi (JDK) • Java Platformu • Derleme (Compilation) • Yorumlama (Interpretation) • Tümleşik Geliştirme Ortamı (IDE) • Java Sanal Makinesi (JVM) • Eclipse • Java Sözdizimi (Syntax)

GİRİŞ

Java Programlama Dili, Android telefon, tablet, televizyon, araba ve çeşitli giyilebilir cihazlara ek olarak; masaüstü, dizüstü ve sunucular için günümüzde hemen hemen her sektörün ihtiyaçlarını karşılayan masaüstü uygulamalar, web uygulamaları, servisler ve Android uygulamaları geliştirmeye olanak sağlayan programlama ortamında en gözde ve aynı zamanda dünya genelinde yaygın olarak kullanılan nesneye yönelik programlama dilidir.

Tarihi

Sun Microsystems *mühendisi James Gosling* tarafından 1991 yılında ilk olarak elektrikli ev aletlerinin birbiriyle haberleşmesini sağlamak amacıyla bir proje olarak geliştirilmiştir. Orijinal adı bu dilin yaratıcıları James Gosling ve arkadaşları tarafından *OAK* olarak isimlendirilmiştir. Daha sonra bir kafede, bu isimde başka bir programlama dili olduğu fark eden Gosling ve arkadaşları, o anda içtikleri kahve markasından esinlenerek programlama dilinin adını Java olarak değiştirmişlerdir.

Java'nın halka açık ilk sürümü (Java 1.0) 1995 yılında çıkmıştır. Sun Microsystems firması 2010 yılında *Oracle* şirketi bünyesine katılmıştır. Java'nın kontrolü artık Oracle şirketinin elindedir. Bu yıldan itibaren Java'nın gelişmiş sürümleri yayımlanmıştır. Şu anda en güncel sürümü *Java 8*'dir.

2006 yılında Sun Microsystems, Java'nın *GNU (Genel Halk Lisansı)* altında herkes tarafından ulaşılabilir olmasını sağlamıştır. Oracle şirketi de bu projeyi *OpenJDK* adı altında devam ettirmektedir.

Java'nın Temel Özellikleri

Java'nın geliştirilmesindeki amaç yazılan kodun bir kere yazılıp sonra birçok işletim sisteminde çalıştırılabilmesidir. Bu çerçevede sahip olduğu özellikleri şu şekilde sıralamak mümkündür:

- *Nesneye yönelik programlama dili:* İlkel veri tipleri dışında Java içerisindeki bütün elemanlar nesne olarak tanımlanmaktadır.
- *Platform bağımsızlığı:* Yazılan Java programları sanal makine üzerinde çalışırlar, bu sanal makine, yazılan programın direk olarak işletim sistemine erişmemesi ve soyutlanması için kullanılır. Bu özellikle Java programları büyük ölçüde taşınabilir olmaktadır. Bir Java programı değiştirilmeden, desteklenen bütün platformlarda çalışabilir.
- *Sağlamlık:* Güçlü hafıza yönetimi yeteneği sayesinde programcı hatalarını yakalayabilme ve her nesne için güvenli bir çalışma ortamı sağlayabilmektedir.
- *Dağıtık yapıdadır:* Geleneksel programlama dillerinde programın çalıştırılması için gerekli bütün program parçaları bir arada bulunmaktaydı, Java ortamında programın bütün parçaları dağıtık bir şekilde olabilir, İnternet üzerinde bile dağıtılmış olabilmektedir.
- *Çevrilen ve derlenen dil:* Java kaynak kodu, bayt kod formatına dönüştürülür ve bu format hedef platforma bağımlı değildir. Bayt kod, sanal makine tarafından çevrilmektedir. JVM Hotspot-Compiler isminde derleyici mevcuttur ve bu derleyici performans ve kritik bayt kod bilgilerini yerel kod bilgilerine çevirmektedir.
- *Otomatik hafıza yönetimi:* Yeni nesneler oluşturulduğunda hafızanın bölüştürülmesi kontrol edilmektedir. Yazılan programın direk olarak hafızaya erişimi yoktur. Çöp Toplayıcı (Garbage Collector) yapısı sayesinde otomatik olarak aktif işaretçisi olmayan nesneler bellekten silinmekte ve hafıza problemleri oluşmasını engellemektedir.
- *Basittir:* Basitten kasıt Java'yı öğrenmenin kolaylığı değil, programcı açısından birçok şeyin basitleştirilmiş olmasıdır. Örneğin bir C++ dilinde bulunan hafızaya direk erişim (pointer), hafızada yer ayırma (memory allocate), çoklu kalıtım (multiple inheritance) vb. kodlama anlamında daha zor ve riskli yapılar kaldırılmıştır. Programcının tehlikeye düşebileceği durumları ortadan kaldırmak ve riskleri minimize etmek için birtakım yöntemler geliştirilmiştir.
- *Çoklu-İş Parçacıklarını destekler (multithreaded):* Çoklu-İş parçacıklarını (multithreaded) destekliyor olması, programın aynı anda birden fazla işi yerine getirmesine olanak sağlamaktadır. Mesela bir iş parçacığı (thread) veritabanına bağlanırken diğer bir kanal kullanıcı arayüzünü güncelliyor, bir başkası ise yazıcıya bağlanıyor olabilmektedir.

İsim benzerliği açısından akıllara gelebilecek bir başka dil ise *JavaScript*'tir. JavaScript, yaygın olarak web tarayıcılarında kullanılmakta olan dinamik bir programlama dilidir. JavaScript ile yazılan istemci tarafındaki *betikler (script)* sayesinde tarayıcının kullanıcıyla etkileşimde bulunması, tarayıcının kontrol edilmesi, asenkron bir şekilde sunucu ile iletişime geçilmesi ve web sayfası içeriğinin değiştirilmesi gibi işlevler sağlanabilmektedir.

JavaScript ile Java arasında; isimleri, yazılım şekli ve standart kütüphanelerindeki benzerlikler dışında herhangi bir bağlantı yoktur ve iki dilin semantikleri (anlam bilimi) çok farklıdır.

JAVA PLATFORMU

Java sadece bir programlama dili değil aynı zamanda bir yazılım platformudur. Bu platform, işletim sistemi üzerinde çalışan, uygulamalara çeşitli servisler veren, bunu her işletim sistemi için standart bir biçimde yapabilen bir yapıdır. Java'nın temel özellikleri arasında yer alan "*platform bağımsızlığı*" ifadesi buradan gelmektedir. Kendi platformunun yetenekleriyle çok çeşitli cihazda kullanılabilir özelliktedir.

Java programlama dili ise, bu platform üzerinde yazılım geliştirmek için kullanılan dildir. Java ile yazılım geliştirmek için gereken tüm bileşenler ücretsizdir. Ücretsiz olarak birçok Java geliştirme ortamı mevcuttur. Bu ünite altında ilerleyen başlıklar içerisinde bu geliştirme ortamları üzerinde kod yazılımı aşamasından bahsedilecektir.

Akıllara *"Java programları nasıl çalıştırılır?"* sorusu gelebilir. Bu soruya yanıt olarak bir java yazılımı şu aşamalarla geliştirilir;

- Yazılımcı Java kodunu yazar.
- Yazılan kod bir Java derleyicisi ile derlenir. Bu derleme aşaması sonucunda "*bayt kod*" adı verilen bir tür sanal makine kodu ortaya çıkar. Platform bağımsızlığını sağlayan bayt kod'dur. Bir kere bayt kod oluştuktan sonra yazılım sanal makine içeren tüm işletim sistemlerinde çalışabilmektedir.
- Bu bayt kod *Java Sanal Makinesi (Java Virtual Machine)* tarafından işletilir. Bu aşama, her bir bayt kod komutunun teker teker yorumlanmasıyla icra edilebileceği gibi *anında derleme (Just in Time Compilation)* kullanılarak da gerçekleştirilebilmektedir.

Şimdi bu aşamalarda kullanılan kavramlar ve Java platformu içerisindeki yapılara açıklık getirelim.

Derleme (Compilation) ve Yorumlama (Interpretation)

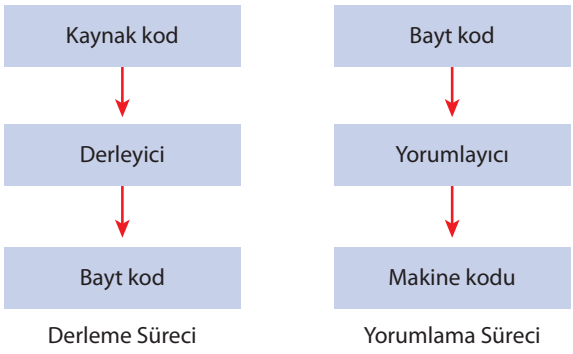
Programlama dili, makine ile insan arasında iletişim kurmak için kullanılan bir dildir. Tabi burada bahsedilen makineden kasıt bilgisayar ve türevi cihazlar olarak düşünülebilir. Bu bağlamda programcı bilgisayara, özellikle bilgisayar üzerindeki işlemciye istediklerini yaptırmak için bir dil kullanmaktadır. Bu dil ne kadar konuşma diline yakınsa, o derece kullanılabilirliği artmaktadır. Diğer taraftan bilgisayarın işlemcisinin dili ise sadece sayılardan oluşan, makine kodları içeren makine dilidir. Dolayısıyla günlük hayattaki dile yakın birtakım istekleri bilgisayara anlatabilmek ve yaptırabilmek için makine diline dönüştürülmesi gerekmektedir.

Bilgisayarın üretildiği ilk yıllarda programcılar kodlarını sayı kodlarını kullanarak yazmaktaydılar. Bu kodlar her işlemci türü için farklı olduğu için, yazılan kodlar sadece belli bir işlemci üzerinde çalışabilmekteydi. Hatta her bir işlemci sürümü için yazılan kod bir önceki sürümde çalışamayabiliyordu. Bu nedenle, programlama dili kavramı ortaya çıkmıştır. Bu yapıda, programcı, bilgisayarın yapması gerekenleri konuşma diline yakın bir dille ifade etmektedir. Bu dilde yazılanlar makine diline çevrilmektedir. Böylece programcılar, hem işlemcilerin kendilerine has olan komut kümelerini öğrenmek zorunda kalmamakta aynı zamanda yazılan programları birden fazla işlemci üzerinde de çalıştırabilmektedirler.

Programlama diliyle yazılan programa "*kaynak kodu (source code)*" ismi verilmektedir. Java diliyle yazılan kaynak kodları uzantı olarak *".java"* uzantısı ile kaydedilmektedir. Makine dilinde ifade edilen koda ise "*ikili kod (binary code)*" denilmektedir. Programlama dilinden makine diline çevirme işlemi iki şekilde yapılmaktadır. Bunlardan birincisi "*anında*", ikincisi ise "*önceden*" çevirme işlemi olarak isimlendirilmektedir.

Anında çevirme işlemi, programlama dilindeki ifadeleri bir yandan okuyup bir yandan makine diline çevirmek demektir. Bu işleme *"yorumlama (interpretation)"* denilmektedir. Yorumlama sürecinde kısım kısım ele alınma yapıldığı için aslında standart bir çalıştırılabilir kod üretilmemektedir. Aynı zamanda yorumlama işlemi aşama aşama yapılmadığı için genellikle ilk hatanın olduğu yerde

programın çalışması kesilmektedir. Önceden çevirme işlemi ise, programlama dilindeki ifadelerin çalıştırılmadan önce makine diline çevrilmesi, sonra da çalıştırılması anlamındadır. Buna da *"derleme (compilation)"* ismi verilmektedir. Derleme işleminin amacı aslında çalışabilir bir kod elde etmektir.



Resim 2.1 Java'da derleme ve yorumlama süreçleri

Derlemenin yorumlamadan en önemli farkı hızlı olmasıdır. Zira makine diline çevirme işlemi sadece bir kere yapılmaktadır. Yorumlama, her çalıştırmada çevirme işlemi yapıldığı için daha yavaş olmaktadır. Derleme işleminin dezavantajı, programdaki her değişiklikte önce derleme işleminin gerekli kılınmasıdır. Sık değişiklik yapılan durumlarda programcı için ciddi sorunlar oluşturabilmektedir. Oysa yorumlama anında yapıldığı için değişiklik de anında yapılmaktadır. Basic, Perl, Phyton, Ruby gibi diller yorumlamalı, C, Pascal, Ada, Agol gibi diller de derlemeli dillerdir.

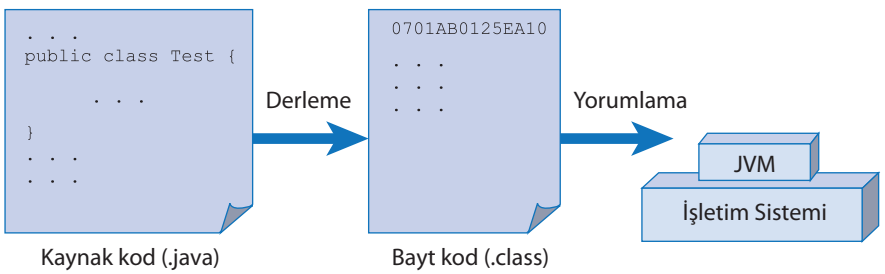
Java Sanal Makinesi (Java Virtual Machine - JVM) ve Bayt Kod (Bytecode)

Java, hem derlemeli hem de yorumlamalı bir dildir. Java programlama diliyle yazılmış kaynak kod, sanal bir işlemcinin anlayabileceği makine koduna çevrilmektedir. Bu kod gerçek bir makine olmadığı için ona makine kodu denmemektedir. Ancak programlama dili gibi konuşma diline yakın da olmadığı için kaynak kodu da değildir. Bu ara dile çevrilmiş koda "*bayt kod (bytecode)*" ismi verilmiştir. Bu ismin verilmesinin bir nedeni de makine kodu gibi baytlardan oluşmasıdır. Java bayt kod dosyaları uzantı olarak *".class"* uzantısı ile kaydedilmektedir.

Yazılan kaynak kod, derleme işlemi yapılarak bayt koda dönüştürülmektedir. Sanal işlemci ise bu bayt kodu makine koduna çevirme işlemini programın çalışması esnasında yapmaktadır. Her işletim sistemi ve işlemciye göre sanal makine, bu bayt kodu ilgili işlemcinin makine diline yorumlamaktadır. Bu sanal makineye "*Java Sanal Makinesi (Java Virtual Machine - JVM)*" ismi verilmektedir ve bu ünite içerisinde kısaca JVM olarak kullanılacaktır.

Java'nın özelliklerinden bahsederken platform bağımsızlığı ifadesi kullanılmıştı. Java'nın "*bir kere yaz, her yerde çalıştır (write once, run anywhere)*" felsefesi ve bu bağlamda platform bağımsızlığını sağlayan bileşen JVM'dir.

Java ilk çıktığı zamanlarda bayt kod işletme hızı çok iyi değildi. Yerine göre sistem üzerindeki yazılımlardan beş hatta on kat yavaş çalışıyordu. Bu nedenle bazı yazılım geliştirme şirketleri "*anında derleme (just in Time – JIT)*" araçları üretmeye başladılar. Burada yapılan aslında bayt kodu, JVM'nin kurulu olduğu gerçek sistemin diline anında derleme yaparak dönüştürmekti. Bu sayede ciddi olarak hız kazanımları yaşandı. 2000 yılından sonra geliştirilen JVM lerde (HotSpot gibi) anında derleme işlevi sanal makine içinde yer almaya başlamış, işlemci hızı ve bellek miktarlarının artışıyla anında derleme yazılımları popülerliğini kaybetmiştir. Günümüzde halen birkaç ürün (Excelsior JET gibi) pazarda bulunsa da genellikle bu yöndeki ihtiyaç azalmıştır.



Resim 2.2 Java uygulaması çalışma aşamaları

JVM, Java bayt kod komutlarını işlediği için, bazı programlama dilleri de kendi derleyicileri yardımıyla kodlarını Java bayt koda dönüştürerek JVM üzerinde çalışmasını sağlayabilmektedirler. Bunlara örnek olarak Scala, Groovy, Jruby gösterilebilir.

Java Çalışma Ortamı (Java Runtime Environment)

Java diliyle yazılmış bir programın çalışması için mutlaka Java yorumlayıcısı olması gerekmektedir. Java uygulamasının çalışması için gerekli minimum gereçleri içeren yapıya "*Java Çalışma Ortamı (Java Runtime Environment – JRE)*" adı verilmektedir.

Java programlarını çalıştırmak için öncelikle çalıştırılacak cihaza eğer yoksa öncelikle JRE kurulması gerekmektedir. Bu işlem cihaz için sadece bir kere yapıldığı için, üzerine JRE kurulmuş bir cihaza herhangi bir program yüklenirken tekrar JRE kurulmasına gerek yoktur. JRE, bazı işletim sistemlerinde kendiliğinden kurulu olmakta, bazılarında ise tercihe bağlı olarak kurulabilmektedir. JRE içerisinde JVM ile birlikte birtakım açık kaynak kodlu Java kütüphaneleri de yer almaktadır.

Java Geliştirme Paketi (Java Development Kit)

Java Geliştirme Paketi, programcıların yazılım geliştirme sürecinde gerekli olan bileşenleri içeren pakete verilen isimdir. Diğer Java bileşenleri gibi *JDK* da ücretsiz olarak sunulmaktadır.

JDK paketini bilgisayarınıza indirmek http://www.oracle.com/technetwork/java/javase/downloads/index.html adresini kullanabilirsiniz.

JDK içerisinde, Java uygulamalarını çalıştırmak için gereken Java Çalışma Zamanı Ortamını da içinde barındırmaktadır. Bu genellikle hususi çalışma zamanı olarak da adlandırılmaktadır, çünkü olağan JRE den ayrıdır ve ilave içeriğe sahiptir. JDK, bir JVM ve konuşlandırma ortamındaki sınıf kütüphanelerinin hepsinden oluşmaktadır. Aynı zamanda geliştiriciler için ekstra kütüphanelerin yanında arayüz tanımlama dili kütüphaneleri gibi ek kütüphaneleri de içermektedir.

Bir Java uygulaması geliştirilmek istendiğinde JRE yetersiz kalmaktadır. JDK denilen yapı içerisinde, Java derleyici, Java yorumlayıcı, uygulama geliştirme araçları, Java API kütüphaneleri, Java geliştiricileri tarafından Java uygulamaları geliştirmek için kullanılan dokümantasyonlar ve ayrıca JRE, JVM bulunmaktadır.

Java API

Java yazılımlarında kullanılan yazılım kütüphanelerine genel olarak verilen isimdir. Java API kullanarak, disk, grafik, ağ, veri tabanı ve güvenlik gibi birçok konuda kullanıcılara erişim imkânı sunulmaktadır.

Java ile İlgili Diğer Bileşen ve Kavramlar

Java Applet: Bayt koda dönüştürülmüş küçük boyutlu Java programlarıdır. İnternet tarayıcıları üzerinde çalışan yapılardır. Applet'ler, web uygulamalarının güvenlik nedeniyle yeteneklerinin az olması yani istemciye (client) erişimlerinin kısıtlı olması gibi nedenlerle tercih edilmektedirler. Örneğin, bir web sayfasından direk yazıcıya çıktı göndermek, usb, com, seri portlara, erişerek çeşitli entegrasyonlar yapmak gibi amaçlarla da kullanılmaktadırlar. Genellikle bazı oyun gibi tarayıcı üzerinde çalışan uygulamalarda Java Applet kullanılmaktadır.

Java Plug-in: JRE ile yüklenen ve İnternet tarayıcısında çalışması gereken Java uygulamasını çalıştıran eklentidir. Java Applet'leri çalıştırmak için Java Plug-in'e ihtiyaç duyulmaktadır. Bu eklentiler ihtiyaç olduğunda tarayıcı üzerinde kendiliğinden çalışmaktadır.

Jar: Java arşiv (Java Archives) isminden gelmektedir. Java'ya özgü paketlenmiş dosya türüdür.

Java Servlet: Java ile geliştirilmiş, sunucu üzerinde çalışan uygulamalardır. Kullanıcıdan alınan verilere göre sonuçlar üreten Java sınıfları olarak düşünülebilir. Bu veriler üzerinde gereken işlemler yapıldıktan sonra önceden tasarlanmış biçimlere uyarlanarak kullanıcıya geri döndürülmektedir.

JSP: Java ile dinamik web sayfaları oluşturmak için hazırlanan uygulamalara Java Server Pages denilmektedir. Sayfanın oluşması bir web istemcisinin istemiyle olmaktadır.

JSF: Model Görünüm Kontrolcüsü (Model View Controller - MVC) mimarisine uygun olarak

hazırlanmış ve Java ile web uygulamaları geliştirmeye olanak tanıyan bir çatıdır. JSP'den farklı olarak kullanıcı arayüzü sağlamaktadır. JSF çatısı JSP ve Servlet uygulamalarını barındırmaktadır.

Model Görünüm Kontrolcüsü (Model View Controller – MVC)
Yazılım mühendisliğinde kullanılan bir mimari desendir. Kullanıcıya yüklü miktarda verinin sunulduğu karmaşık uygulamalarda veri ve gösterimin soyutlanması esasına dayanmaktadır. Böylelikle veriler (model) ve kullanıcı arayüzü (görünüm) birbirini etkilemeden düzenlenebilmektedir.

Java Çöp Toplayıcı (Garbage Collector): Atık veri toplama teknolojisi, Java'dan önce de var olan ama adını Java ile duyurmuş ve yaygın olarak kullanılmaya başlanmış bir tekniktir. C, C++ gibi dillerin en büyük dezavantajlarından birisi olan dinamik bellek yönetimi, kod içerisinde dinamik olarak bellek ayırımı yapabilmeyi ve sonrasında bellek ile iş bittiğinde ayrılan belleğin tekrar iade edilmesi yapısını içermektedir. Bu yer ayırımı ve iade süreci tamamen yazılımcıya bırakılmış olduğu için, unutulma ve yanlış kullanımlar sonucunda *bellek sızıntıları (memory leak)* oluşmakta ve bir süre sonra yazılımın ve işletim sisteminin beklenenden farklı davranmasına neden olabilmektedir. Bu nedenle bugünün tüm C ve C++ yazılımları, işletim sistemleri dâhil olmak üzere az da olsa bellek sızıntısı içermektedir.

Java Çöp Toplayıcısı (Garbage Collector) sayesinde sanal makine akıllı bir biçimde kullanılmayan bellek bölümlerini belirli aralıklarla ya da uyarlamalı yöntemlerle otomatik olarak temizlemekte ve sisteme geri iade etmektedir. Çöp toplama sistemlerinin yapısı oldukça karmaşık olup geçen yıllar içerisinde büyük gelişmeler kaydetmiştir. Tabi çöp toplayıcıların varlığı Java'da bellek sızıntısı olmayacağı anlamına gelmemekte ancak daha ender olarak farklı şekillerde karşımıza çıkmakta ve genellikle tedavi edilmesi daha kolay olmaktadır.

Kısaca Java,

- Bir programlama dili,
- Bir geliştirme ortamı,
- Bir uygulama ortamı,
- Bir kurulum ortamıdır.

Bu anlamda Java, klasik programlama dillerinden farklı olarak çok geniş bir yazılım geliştirme ekosistemi sunmaktadır. Bu ekosisteme giriş olarak öncelikle JDK'nın son sürümünün sisteminize indirilerek kurulması gerekmektedir. Bu kurulumdan sonra Eclipse, Netbeans, IntelliJIDEA gibi geliştirme ortamları kullanılarak uygulama geliştirmeye başlayabilirsiniz.

Öğrenme Çıktısı

1 Java programlama dilini açıklayabilme
2 Java'nın temel özelliklerini açıklayabilme
3 Java platformunu tanımlayabilme
4 Derleme ve yorumlama kavramlarını açıklayabilme
5 Java sanal makinesini, Java çalışma ortamını, Java geliştirme paketini, Java API ve diğer bileşenleri açıklayabilme ve kullanabilme

Araştır 1	İlişkilendir	Anlat/Paylaş
Günümüzde Java dili haricinde popüler olarak kullanılan programlama dillerini nelerdir? İşverenlerin en çok talep ettiği diller hangileridir araştırınız.	Diğer popüler programlama dilleri ile Java'yı ilişkilendiriniz.	Programlama dilleri öğrenmenin faydalarını ve neden programlama dili kullanıldığını çevrenizdekilere anlatınız.

JAVA TÜMLEŞİK GELİŞTİRME ORTAMI KULLANIMI

Java JDK kurulu bir sistemde, Java kaynak kodları yazmak için kullanılan işletim sisteminde bulunan herhangi bir metin editörü bile yeterli olabilmektedir. Bilindiği gibi kaynak kod derleme ve yorumlama aşamalarından geçerek çalışır hale getirilebilmekteydi. Java JDK içerisinde bu derleyici ve yorumlayıcının bulunduğundan bahsedilmişti. Bu başlık altında kaynak kodu yazma aşaması ele alınacaktır.

Kaynak kodu yazarken Java diline has sözdizimi ve kuralları geçerli olacaktır. Dolayısıyla yazılan metin bir anlamda sıradan metin diliyle değil bir programlama diliyle yazılacağı için basit bir metin editöründen farklı bir ortamda yazılmalıdır. Programcıya yardımcı geliştirme ortamları, gerek yazılan dildeki basit imla hatalarının giderilmesinde gerekse dile has yapıların hatırlatıcılarla, sunulan eklentilerle kullanım kolaylığı getirmesi neredeyse bir zorunluluk halini almıştır. Bunlara kısaca "Tümleşik Geliştirme Ortamı (Integrated Development Environment - IDE)" ismi verilmektedir. Bu ortamlar sayesinde programcı hızlı ve rahat bir şekilde yazılım geliştirebilmekte, geliştirme sürecini organize edebilmekte ve bu sürecin daha verimli kullanılması sağlanabilmektedir. Bir tümleşik geliştirme ortamında olması gereken özellikleri şu şekilde sıralamak mümkündür:

- Programlama diline göre sözdizimi renklendirmesi yapabilen kod yazım editörü.
- Kod dosyalarının hiyerarşik olarak görülebilmesi amacıyla hazırlanmış gerçek zamanlı bir çizelge.
- Tümleşik bir derleyici, yorumlayıcı ve hata ayıklayıcı.
- Yazılımın derlenmesi, bağlanması, çalışmaya hazır hale gelmesi ve daha birçok ek işi otomatik olarak yapabilmek amacıyla küçük inşa araçları.

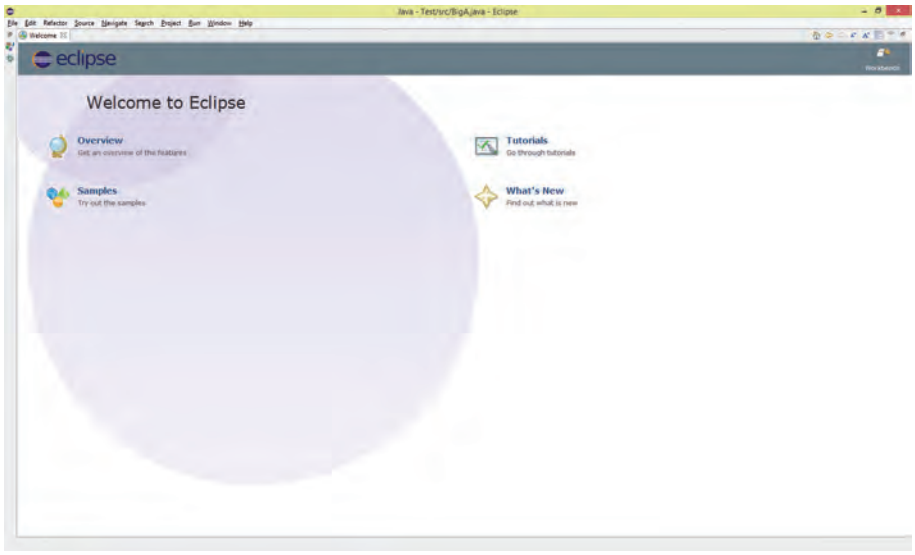
Eclipse ortamını bilgisayarınıza indirmek için https://eclipse.org/downloads/ adresini kullanabilirsiniz.

Bu kitap içerisinde tamamen ücretsiz olan "Eclipse" tümleşik geliştirme ortamı kullanılacaktır.

Resim 2.3 Eclipse ortamının Windows işletim sistemi için uygun sürüm indirme adresi görünümü

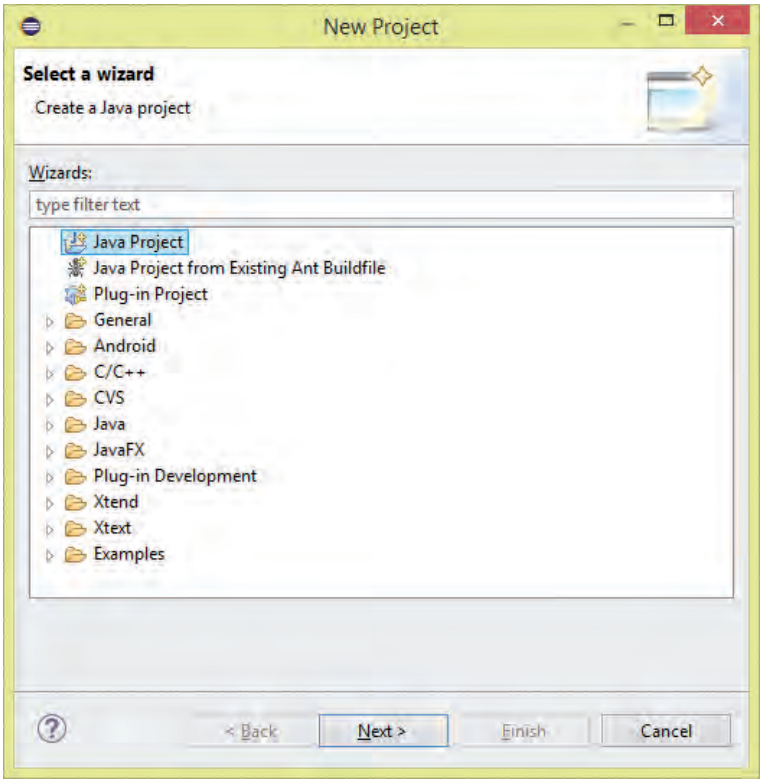
Eclipse'e İlk Bakış

Eclipse sürekli güncellenen bir geliştirme ortamı olduğu için burada gösterilen ekran görüntülerindeki sürüm numaraları farklılık gösterebilmektedir. Aynı zamanda Eclipse için bir kurulum yapılmamaktadır. İndirilen sıkıştırılmış formatlı dosyanın işletim sisteminde istenilen yere açılması ve çalıştırılması yeterli olacaktır. Çalıştırıldığında projenin çalıştırılacağı ortamın seçilmesi istenmektedir. Bu klasör genellikle *"çalışma alanı (workspace)"* olarak adlandırılmaktadır. Bu klasör yolu istenildiğinde değiştirilebilmektedir. İstenilen çalışma alanı seçildikten sonra geliştirme ortamına yeni başlayanlar için yardımcı olan karşılama ekranı gelmektedir.



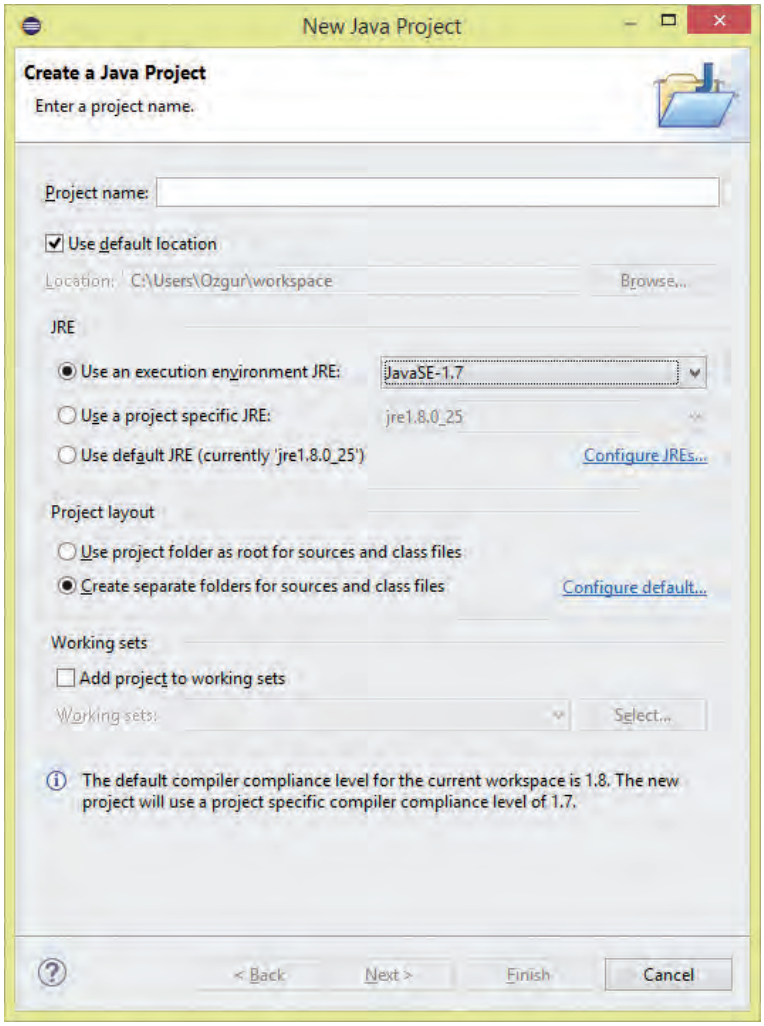
Resim 2.4 Eclipse açılışında karşılama ekranı

Açılış karşılama ekranı üzerinde kısaca gözden geçirme (overview), öğretici dokümanlar (tutorials), örnekler (samples) ve yenilikler (what's new) olmak üzere bir takım ekran sekmeleri bulunmaktadır. Proje ekranına geçmek için *"tezgah (workbench)" tıklandığında boş* Eclipse çalışma ortamı gelecektir. Yeni bir Java projesi oluşturmak için dosya menüsünden sırasıyla *"File->New->Project"* seçilerek proje oluşturma sihirbazı açılacaktır.



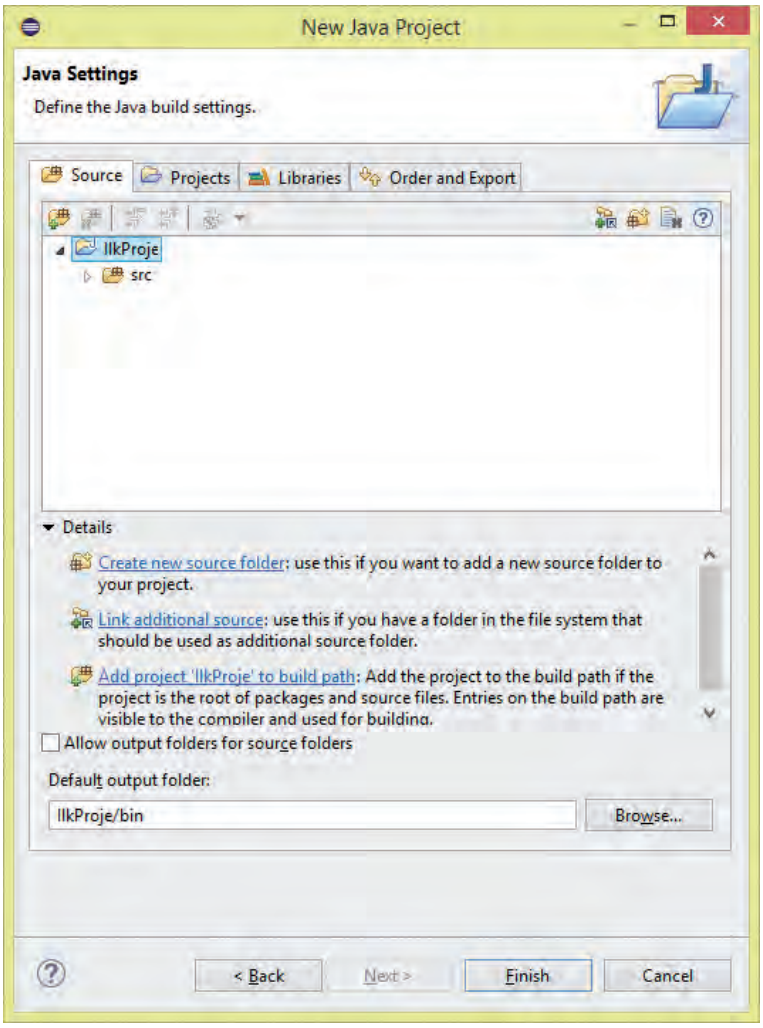
Resim 2.5 Eclipse proje oluşturma sihirbazı ekran görünümü

Bu ekranda Eclipse ortamında oluşturulabilecek tüm proje yapıları görünmektedir. Eclipse ortamı için yüklenebilen eklentilerle bu ekran çok daha farklı proje tiplerine de destek verebilmektedir. Java projesi oluşturabilmek için en üstte olan *"Java Project"* seçilerek ilerleme yapılmalıdır.



Resim 2.6 Java projesi oluşturma ayarları ekran görünümü

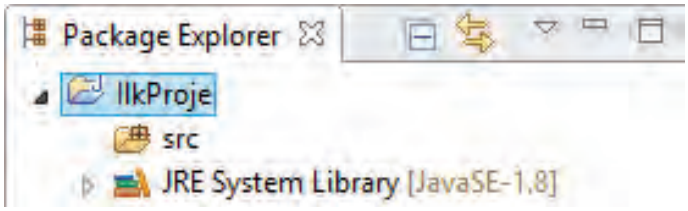
Java projesi oluşturma ekranında en üst kısma proje ismi yazılması gerekmektedir. Proje isminin alt kısmında ise bu projede hangi Java sürümüyle çalışılması, proje içerisindeki klasör ve sınıflar, projenin hangi çalışma alanına kaydedileceği gibi ayarların yapıldığı kısımlar bulunmaktadır.



Resim 2.7 Java projesi klasör ve kütüphane ayarları ekran görünümü

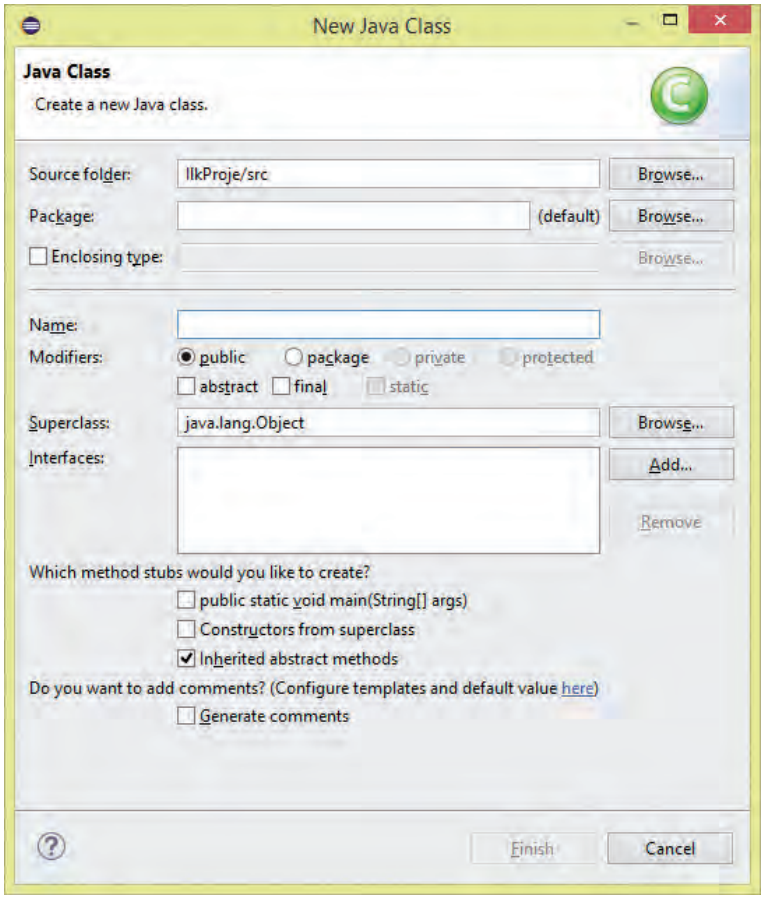
Sonraki aşamada ise Java projesi için eklenmesi istenen kütüphaneler ve yardımcı kaynakların eklenebildiği ekran gelmektedir. Bu ekranda projede kullanılacak temel kütüphaneler eklenmiş durumdadır, bunu görmek için *"kütüphaneler (libraries)"* sekmesi tıklanarak görülebilir. Gerekli ayarlamalar yapıldıktan sonra *"tamamla (finish)"* tuşu tıklanarak proje oluşumu tamamlanmalıdır.

Bu aşamalar sonrasında proje oluşturulmakta ve *"proje gezgini (Project Explorer)"* penceresi içerisinde proje dosyaları görülebilmektedir. Burada görülen *"src"* klasörü, kodlarının yazılacağı sınıfları (Class) barındırmaktadır.



Resim 2.8 Proje gezgini (Project Explorer) penceresi

Burada görülen *"src"* klasörü, kodlarının yazılacağı sınıfları (Class) barındırmaktadır. *"src"* klasörüne sağ tuşla tıklayarak sırasıyla *"New->Class"* seçilerek yeni bir sınıf oluşturulabilmektedir.



Resim 2.9 Java sınıf oluşturma penceresi

Bu ekranda *"paket (package)" kısmı belirli sınıfları bir arada tutan klasörleri göstermektedir. "İsim (Name)"* alanı ise oluşturulacak olan sınıfa isim verilen yerdir. Burada Java'da sınıf isimleri büyük harfle yazılmaktadır. Bu konular ve bu ekranla ilgili diğer konular da ilerleyen ünitelerde anlatılacaktır. Tamamla tuşunun tıklanmasıyla Java sınıfı oluşturulmaktadır.

Öğrenme Çıktısı

6 Java Tümleşik geliştirme ortamını açıklayabilme
7 Eclipse ortamını kullanabilme

Araştır 2

Eclipse haricinde kullanılan diğer tümleşik geliştirme ortamları nelerdir araştırınız?

İlişkilendir

Ben çok tercih edilen dillerin tümleşik geliştirme ortamları ile Eclipse tümleşik geliştirme ortamı arasındaki benzerlikleri belirleyiniz.

Anlat/Paylaş

Tümleşik geliştirme ortamı olmadan programlama dili bilerek uygulama geliştirmenin zorluklarını çevrenizdekilere anlatınız.

JAVA SÖZDİZİMİ VE İLK JAVA UYGULAMASI YAZIMI

Her dilde olduğu gibi programlama dillerinin de kendi içinde belirli kuralları vardır. Bu kurallara söz dizim denilmekte ve bunlara uyarak derleyiciye ve yorumlayıcıya kodu doğru bir biçimde ifade etmek gerekmektedir. Kod yazımında bu kurallara ve neden ihtiyaç duyulduğu şu şekilde sıralamak mümkündür:

- Yazılım üretimindeki %80'lik bir zaman dilimi, kod bakımına (maintanence) harcanmaktadır.
- Çok az yazılım, kullanım zamanı boyunca ilk programcısı tarafından geliştirilmektedir.
- Mühendislerin yeni kodu hızlı ve verimli bir şekilde anlamasının sağlanması önemlidir.
- Kod yazma gelenekleriyle yazılımın okunması ve anlaşılması kolaylaşmaktadır.
- Yazılan kodu piyasaya açmak istendiğinde daha anlaşılır olacağı için daha kolay destek bulunabilmektedir.

Sınıf temelli nesneye yönelik programlama dili olan Java'da *"{ }"* şeklinde süslü parantezler içerisindeki bloklarla kodlar yazılmaktadır. Aynı zamanda *"++"* arttırma ve *"--"* azaltma işleçleri bu dilin belirgin özelliklerindendir. Aşağıdaki Resim 2.10 örnek bir kod parçası içermektedir.

```
1  // Selamlama.java
2  public class Selamlama {
3
4      public static void main(String[] args) {
5          System.out.println("Merhaba arkadaşlar!");
6      }
7
8  }
```

Problems | Javadoc | Declaration | Console | LogCat | Lint Warnings

<terminated> Selamlama [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (28 Eyl 2015 16:42:01)

Merhaba arkadaşlar!

Resim 2.10 Java kod satırı ve çalıştırılması sonrasında konsol ekranındaki görüntüsü

Örnek kod satırlarının ilk satırı *"//"* işareti ile başlayan *açıklama satırı (comment line)* görülmektedir. Açıklama satırları hemen hemen bütün programlama dilinde mevcut olup programcının programlama dili haricinde kendi anlayacağı bir dille yazmış olduğu satır ve satırlardan oluşmaktadır. Yazılan kodun üze-

rinde değişiklik yapılması muhtemel olan kişilere kolaylık sağlaması, bakım yapılabilirliğinin artırılması için doğal dille yazılan kelimeler topluluğundan oluşmaktadır. Bu satırlar derleyici tarafından göz ardı edilmektedir.

Eğer birden fazla satır açıklama satırı olarak yazılacaksa her satırın başına *"//" işleci koyma zorunluluğu yerine aşağıdaki gibi "/*"* ile *"*/" işleçleri arasına alınan satırlar otomatik olarak açıklama satırı olarak belirtilmektedir.*

```
// Bu satır açıklama satırıdır.
/* Bu satır da açıklama satırıdır */
/* Bu
   dört satır
   da açıklama
   satırıdır. */
```

Resim 2.11 Açıklama satırı ve satırları örneği

Resim 2.10 üzerindeki ikinci satırda ise sınıfın adı ve adının önünde o sınıfa erişimin belirleneceği yapılardan biri olan *"public"* erişim belirleyicisi görülmektedir. *"public"* olarak belirlenen erişim belirleyicisi o sınıfın bu sınıf haricinde diğer sınıflar ve paketlerden erişilebilirliğini yani dışarıdan erişilebilir olduğunu ifade etmektedir. Bu yapı nesneye yönelik programlama konusu içerisinde olan sarmalama (encapsulation) yapısı içerisine girmektedir. Sonrasında *"class"* ön ekiyle sınıfın tanımlandığı yapı görülmektedir.

Her Java programının mutlaka en az bir sınıfı ve her sınıfın da bir ismi olmak zorundadır. Aynı zamanda sınıf isimlerinin ilk harfi büyük harf ile başlamalıdır. Bu örneğimizde de sınıf ismimiz *"Selamlama"* olarak belirtilmiştir.

Java'da *özel amaçlı sözcükler (reserved words)* veya *anahtar kelime (keywords)* olarak bilinen yapılar mevcuttur, bunlar derleyici tarafından bilinirler ve başka bir amaç için kullanılmazlar. Bunlara örnek olarak sınıf tanımlamasında kullanılan "class" kelimesi verilebilir. Bu kelime sonrasında yazılan kelimenin derleyici tarafından bir sınıf ismi olarak tanımlanacağını ifade etmektedir. Diğer bir *özel amaçlı* sözcük olarak *"public"*, *"static"* ve *"void"* kelimelerini verebiliriz. *"public"* erişim belirleyicisi dışarıdan erişilebilirliği göstermekte yanındaki "static" ön eki de sınıf tarafından paylaşıldığını ve *"void"* ön ekiyle de *"main"* isimli metodun herhangi bir değer döndürmediği ifade edilmektedir. Burada görülen bu ön ekler ilerleyen ünitelerde çok daha detaylı olarak anlatılacaktır.

Tablo 2.1 Özel karakterler

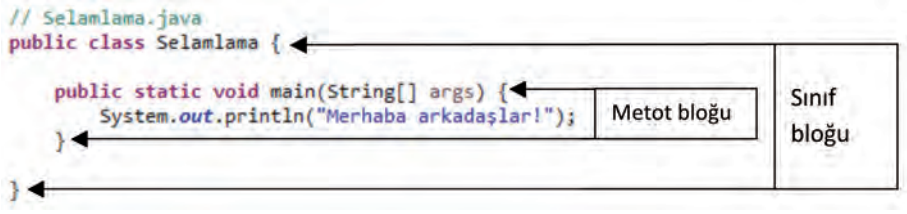
Karakter	İsim	Açıklama
{ }	Süslü parantez açma ve kapatma	Kod bloklarını ifade etmektedir.
( )	Parantez açma ve kapatma	Metotlar için kullanılır.
[ ]	Köşeli parantez açma ve kapatma	Karakter dizileri için kullanılır.
/ /	Çift eğik çizgi	Açıklama satırı için kullanılır.
" "	Tırnak açma ve kapatma	Karakter dizileri olarak kelimeleri gruplamak için kullanılır.
;	Noktalı virgül	Kod satırının sonunu işaret etmek için kullanılır.

Resim 2.10 üzerindeki dört numaralı kod satırında ise metot tanımlaması görülmektedir. Nesneye yönelik programlama konusu içerisinde bir kavram olan metotlar sınıfların içerisindeki davranışsal yetenekleri kazandıran yapılar olup nesneye hareketlilik katan yapıları temsil etmektedir. "main" isimli metodun, Java kod sözdizimine uygun olarak parantez işareti *"( )"* içerisinde aldığı parametreler ve değişken yapısı ve ismi görülmektedir. Java uygulamaları aslında bazı özel koşullara sahip sınıftır. Java'da her birim bir sınıf olmak zorundadır. Bağımsız olarak herhangi bir değişken veya metot yoktur. Her değişken veya metot mutlaka bir sınıfın içerisinde yer alır. Dolayısıyla bir sınıf, uygulama olarak çalıştırılacağı zaman, o sınıfın *"main()"* metodunu bulur ve oradan itibaren çalışır. Her sınıf içerisinde main() metodu olmak zorunda değildir, sadece uygulama sınıfları içerisinde yalnızca tek olarak main() metodu bulunmakta ve Java bu sınıfın bir uygulama programı olduğunu anlamaktadır.

Süslü parantez *"{"* işaretiyle sınıflar ve metotlar gibi kod bloğu içerecek yapılar başlar ve diğer *"}"* işaretiyle de sonlanır. *"Selamlama"* sınıfının da benzer şekilde süslü parantezlerle içerisindeki kod bloğu görülmektedir.

dikkat

Kod bloklarında kullanılan süslü parantezlerde açma parantezi "{" kullanıldığı zaman o kod bloğu için mutlaka bir tane de kapama süslü parantezi "}" kullanılması zorunludur. Aksi takdirde sözdizimi hatası meydana gelir.



Resim 2.12 İç içe kodlanmış metot ve sınıf blokları

Resim 2.10 içerisindeki beş numaralı satırda istenilen yazının yani *"Merhaba arkadaşlar!"* yazısının ekranda yazılması hedeflenmektedir. Java dili içerisinde de *"karakter dizileri (Strings)"* karakterlerin yan yana gelmesiyle oluşan veri tiplerinden olup çift tırnak işareti içerisinde yazılmak zorundadırlar. Java'da her satır sonunda noktalı virgül *";" işareti, satır sonlandırıcı rolü üstlenmekte, o satırın sonuna gelindiğini ve o satır komutlarının bittiğini ifade etmektedir.*

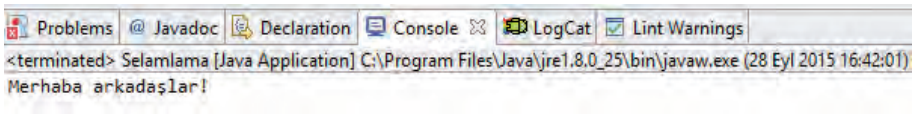
dikkat

Java kaynak kodları "harfe duyarlıdır (case sensitive)". Örneğin "main()" yerine "Main()" yazılırsa sözdizimi hatası meydana gelir.

Tümleşik geliştirme ortamlarının avantajlarından biri olarak derleme, yorumlama, çalıştırma aşamaları için tek tuşla kolaylıkla bu işlevler yapılabilmektedir. Eclipse ortamında menülerin altında hızlı menü tuşları içerisinde play tuşuna benzer bir tuş görülmektedir. Bu tuşa tıklayarak uygulamanın çalışması sağlanabilmektedir.

Resim 2.13 Eclipse ortamındaki uygulama çalıştırma tuşu görünümü

Eclipse ortamı içerisindeki uygulama çıktılarının gösterildiği konsol ekranı, pencerenin alt kısmında yer almakta, çıktılar rahatlıkla görülebilmektedir.



Resim 2.14 Uygulamanın çalıştırılması sonrasında konsol ekranındaki görüntüsü

Öğrenme Çıktısı

8 Java sözdizimini açıklayabilme
9 Temel Java uygulaması yazabilme

Araştır 3

Yukarıdaki örneklere benzer olarak "Ogrenci" adında bir sınıf oluşturunuz, oluşturacağınız sınıf öncesinde açıklama satırı örneğini de ekleyiniz.

İlişkilendir

Java sözdizimi ile başka dillerin sözdizimlerini karşılaştırınız. Benzerlikleri ve farklılıkları belirleyiniz.

Anlat/Paylaş

Bir dilin sözdizimi kuralları ile bir programlama dilinin sözdizimi kurallarının benzerliklerini ve farklılıklarını çevrenizdekilere anlatınız.

Yaşamla İlişkilendir

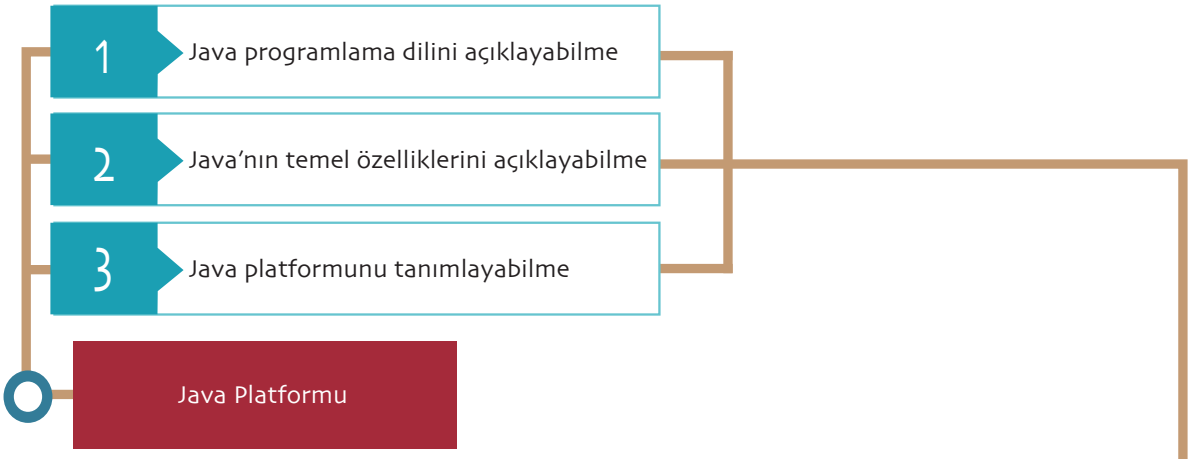
BTOS: 2014 yılında Türkiye teknoloji sektöründe en çok kazanan Java'cılar oldu

Türkiye'de İnsan Kaynakları alanında faaliyet gösteren bilişim danışmanlık şirketi BTOS'un 2014 yılı Bilişim İnsan Kaynakları Raporu'nu yayınlandı. Rapor, Türkiye'de bilişim sektöründe işverenler tarafından 2014 yılında en çok talep edilen giriş ve orta düzey pozisyonlar, teknolojiler/beceriler, çalışanların maaş aralıkları ve sektör tercihlerini sıralıyor.

Talep edilen teknolojilerin projelere göre değişkenlik gösterdiğini not eden raporda, teknolojinin daha çok küçük ve orta ölçekli projelerde kullanıldığı belirtiliyor. Büyük ve kurumsal çözümlerde yaygınlıkla kullanılan Java ise en çok talep gören ikinci teknolojiydi.

Bilişim sektöründe giriş ve orta düzey pozisyonlardaki çalışanların maaş aralıklarını yeteneklerine göre kıyaslayan raporda, bu yılın kazananı Java geliştiricileri. Onu C#. NET ve PHP/MYSQL izliyor. Raporda deneyimsiz adayların, ilk işlerini daha kolay bulabilme konusunda C# (ASP.NET), PHP ve PL/SQL alanında biraz daha şanslı olduğu da belirtiliyor. Raporda paylaşılan tablo İstanbul piyasası baz alınarak oluşturulmuş, maaş aralıkları illere göre değişkenlik gösteriyor.

Kaynak: http://webrazzi.com/2014/12/24/btos-2014-yilinda-turkiye-teknoloji-sektorunde-en-cok-kazanan-javacilar-oldu/



1 Java programlama dili, günümüz dünyasındaki birçok cihaz üzerinde farklı yapılarda uygulama geliştirebilmeye imkân sağlayan nesneye yönelik programlama dilidir. Uygulamaların gün geçtikçe arttığı, birçok isteğin ve problemin çözüldüğü günümüz dünyasında, farklı platformlar karşısında uygulama geliştirme ihtiyacı gündeme gelmiştir. Yazılan uygulamanın platform bağımsızlığı ve aynı zamanda yazılım niteliğinin korunması ile birlikte nesneye yönelik yapıyı da üzerinde barındırması Java'nın popülerliğini arttırmıştır.

2 Farklı platformlar üzerinde çalışabilmesi, nesneye yönelik olması, hata yakalama, güvenli çalışma ortamıyla sağlamlığı, dağıtık olarak program parçaları ile çalışılabilmesi, derlenebilir olması, hafıza yönetiminin otomatik olması, programlama anlamında basit olması, çok kanallı olarak aynı anda birden fazla işi yapabilmesi gibi temel özelliklere sahip olan bir dildir.

3 Java'yı sadece bir programlama dili olarak görmemek gerekmektedir. İşletim sistemi üzerinde çalışan ve çalışan uygulamalara çeşitli servisler veren bir yapı olarak düşünmek gereklidir. Yazılan Java kodları derleme aşamasından geçtikten sonra sanal makine koduna dönüştürülür. Bu sanal makine kodu, herhangi bir sistem üzerinde kurulu olan sanal makineler üzerinde çalıştırılabilecek hale gelmiş durumdadır. Uygulamanın kaynak koddan çalıştırılma aşamasına kadar olan süreç ve bu süreçteki yapılar Java platformunu oluşturmaktadır.

öğrenme çıktıları ve bölüm özeti

4 Derleme ve yorumlama kavramlarını açıklayabilme

5 Java sanal makinesini, Java çalışma ortamını, Java geliştirme paketini, Java API ve diğer bileşenleri açıklayabilme ve kullanabilme

Java Platformu

4 Platform bağımsızlığının en önemli yapılarından biri olan Java Sanal Makinesidir. Kaynak kod sanal makinenin anlayabileceği ve tam anlamıyla makine kodu olmayan ara kod olan bayt koda derlenerek çevrilmektedir. Sanal makine ise bu kodu anlayabilmekte ve satır satır yorumlayarak çalıştırma işlemini yerine getirmektedir. Cihaz üzerindeki donanıma bağlı kalmaksızın bayt kodu o donanımın anlayabileceği makine dilinde çalıştırabilen yapı Java Sanal Makinesidir.

5 Java Sanal Makinesinin de içerisinde bulunduğu aynı zamanda uygulamanın çalışması için gerekli olan minimum gereçleri üzerinde barındıran ortama Java çalışma ortamı ismi verilmektedir.

Yazılım geliştirme ve uygulama yazmak için Java çalışma ortamının da içinde bulunduğu, uygulama geliştirme araçları, Java API, kütüphaneler ve dokümantasyonların da bulunduğu pakete Java geliştirme paketi ismi verilmektedir. Buradaki Java API aslında kullanılan yazılım kütüphanelerinin genel adıdır. Java platformu içerisindeki Java applet, Java plug-in, Jar, Java servlet, JSP, JSF yapıları da uygulama geliştirme çeşitliliğini sağlayan yapılardır.

6 Java Tümleşik geliştirme ortamını açıklayabilme

7 Eclipse ortamını kullanabilme

Java Tümleşik Geliştirme Ortamı Kullanımı

6 Java'nın sadece programlama dilinden öte olup bir platform olmasından dolayı, uygulama geliştirme süreci içerisinde platformun yapılarını kullanabilme ve kod yazma aşamasında programcıya kolaylık sunan çeşitli ortamlar bulunmaktadır. Yazılan kaynak kodun çalıştırılma aşamasına kadar olan süreçte gerek sözdizimi, gerek derlenmesi, yorumlanması ve makine diline çevrilmesi, çalışmaya hazır hale gelip çalışır hali üzerinde çeşitli testlerin yapılabildiği ortamlara tümleşik geliştirme ortamı adı verilmektedir.

7 Eclipse açık kaynak kodlu ve ücretsiz bir tümleşik geliştirme ortamıdır. Geliştirilmesi IBM tarafından başlatılan Eclipse genel olarak java geliştiriliyor olsada Android, PHP, C, C++, Pyhton ve Ruby dillerinin desteğini de elinde bulundurmaktadır. Kurulumu ve kullanımı kolay olan bu ortam birçok uygulama geliştiricinin tercih ettiği geliştirme ortamıdır.

8 Java sözdizimini açıklayabilme

9 Temel Java uygulaması yazabilme

Java Sözdizimi ve İlk Java Uygulaması Yazımı

8 Sonuçta bir programlama dili söz konusu olduğu için o dilin de kendi içerisinde bir takım kuralları mevcut olmalıdır. Kuralsız bir konuşma dili düşünülemeyeceği gibi kuralsız bir programlama dili de düşünülemez.

9 Uygulama geliştirilirken satırlarca kod yazılmaktadır. Bu kodların bakımı, yazılan yapıların hızlı ve verimli bir şekilde anlaşılabilmesi, yeniliklerin eklenmek istediği durumlarda hızlı ve kolayca yapılabilmesi gibi durumlar yazılımın üretim sürecinde yaşanmaktadır. Yazılan kodların dilin sözdizimine uygun bir şekilde yazılarak derleme ve yorumlama süreçlerini sorunsuz yapmak gerekmektedir. Aksi takdirde uygulama çalışmayacaktır.

neler öğrendik?

1 Aşağıdakilerden hangisi Java'nın ilk zamanlardaki ismidir?

A. Sun
B. GNU
C. OAK
D. Oracle
E. JDK

2 Aşağıdakilerden hangisi Java'nın kullandığı hafıza yönetimi yapısıdır?

A. JDK
B. JRE
C. JVM
D. Çöp Toplayıcı
E. Kanal

3 Aşağıdakilerden hangisi Java'nın temel özelliklerinden biri **değildir?**

A. Platform bağımsızlık
B. Dağıtık
C. Sağlamlık
D. İş parçacıklarını destekleme
E. Karmaşıklık

4 Bir programın aynı anda birden fazla işi yerine getirebilmesi aşağıdaki özelliklerden hangisini ifade etmektedir?

A. Platform bağımsızlık
B. Dağıtıklık
C. Sağlamlık
D. Nesneye yöneliklik
E. Çok iş parçacıklılık

5 Aşağıdakilerden hangisi bir java uygulamasının kaynak kod ile makine kodu arasındaki süreci doğru olarak göstermektedir?

A. Kaynak kod ->Derleme -> Bayt kod -> Yorumlama -> Makine kodu
B. Kaynak kod -> Yorumlama -> Bayt kod -> Derleme -> Makine kodu
C. Kaynak kod -> Bayt kod -> Derleme -> Makine kodu
D. Kaynak kod -> Derleme -> Makine kodu
E. Kaynak kod -> Derleme -> Bayt kod -> Makine kodu

6 Aşağıdakilerden hangisi bayt kod hakkında doğru bir ifadedir?

A. Bayt kod derlenerek çalıştırılır.
B. Bayt kod makine kodudur.
C. Derleme aşamasından sonra bayt kod elde edilir.
D. Kaynak kodu yorumlandıktan sonra bayt kod elde edilir.
E. Bayt kod insanın okuyarak anlayabileceği bir koddur.

7 Java çalışma ortamı (JRE) aşağıdakilerden hangisini içermektedir?

A. Java Geliştirme Paketi (JDK)
B. Çoklu iş parçacığı (Multithread)
C. Bayt kod
D. Java Sanal Makinesi (JVM)
E. Eclipse

8 Aşağıdakilerden hangisi dinamik bellek yönetiminin yanlış kullanımları sonucu oluşabilecek bellek sızıntılarına (memory leak) karşı kullandığı yapıdır?

A. Jar
B. JVM
C. Java Çöp toplayıcı (Garbage Collector)
D. Eclipse
E. Java Applet

9 Eclipse ortamında kodların yazılacağı sınıfların bulunduğu klasör aşağıdakilerin hangisidir?

A. src
B. bin
C. res
D. gen
E. JRE System Library

10 Aşağıdakilerden hangisi Java sözdizimine uygun bir açıklama satırı örneği **değildir?**

A. ** Açıklama satırı **
B. /* Açıklama satırı */
C. // Açıklama satırı
D. /* - Açıklama satırı - */
E. /* * Açıklama satırı * */

1. C	Yanıtınız yanlış ise "Java Programlama Dili Nedir?" konusunu yeniden gözden geçiriniz.
2. D	Yanıtınız yanlış ise "Java'nın Temel Özellikleri" konusunu yeniden gözden geçiriniz.
3. E	Yanıtınız yanlış ise "Java'nın Temel Özellikleri" konusunu yeniden gözden geçiriniz.
4. E	Yanıtınız yanlış ise "Java'nın Temel Özellikleri" konusunu yeniden gözden geçiriniz.
5. A	Yanıtınız yanlış ise "Java Platformu" konusunu yeniden gözden geçiriniz.
6. C	Yanıtınız yanlış ise "Java Platformu" konusunu yeniden gözden geçiriniz.
7. D	Yanıtınız yanlış ise "Java Platformu" konusunu yeniden gözden geçiriniz.
8. C	Yanıtınız yanlış ise "Java ile İlgili Bileşen ve Kavramlar" konusunu yeniden gözden geçiriniz.
9. A	Yanıtınız yanlış ise "Java Tümleşik Geliştirme Ortamı Kullanımı" konusunu yeniden gözden geçiriniz.
10. A	Yanıtınız yanlış ise "Java Sözdizimi ve İlk Java Uygulaması Yazımı" konusunu yeniden gözden geçiriniz.

2 Araştır Yanıt Anahtarı

Araştır 1

160 ülkede 420 bin üyesi olan Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE), dünyanın en popüler programlama dillerini sıraladığı IEEE Spectrum listesinin 2017 sonuçlarına göre günümüzde popüler olarak kullanılan programlama dillerini şu şekilde sıralamak mümkündür:

- Python
- C
- C++
- C#
- R
- JavaScript
- PHP
- Go
- Swift
- Arduino
- Ruby
- Assembly
- Scala
- Matlab
- HTML

Programalama dilleri konusunda önemli bir çalışma hazırlayan IEEE, GitHub'daki 300 programlama dilini esas aldığı liste için Google Trendler'den Twitter'a, Hacker News'tan Stack Overflow, Dice ve Reddit'e çok sayıda açık ve kapalı kaynağı taradığını belirtilmektedir. Yine aynı kaynağa dayanarak iş verenlerin en çok talep ettiği diller de şu şekilde sıralanmaktadır:

1. Java
2. C
3. Python
4. C++
5. JavaScript
6. C#
7. PHP
8. HTML
9. Ruby
10. Swift

2 Araştır Yanıt Anahtarı

Araştır 2

Yazılım geliştiricilerin kullandıkları Eclipse haricindeki diğer tümleşik geliştirme ortamlarını şu şekilde sıralamak mümkündür:

- Microsoft Visual Studio
- NetBeans
- PyCharm
- IntelliJ IDEA
- Code::Blocks
- Aptana Studio 3
- Komodo
- RubyMine
- Xcode

Araştır 3

```
// Bir örnek açıklama satırıdır
/* Bu da bir
   başka
   açıklama satırıdır. */

public class Ogrenci {
      public static void main (String[] args) {
            System.out.println("İyi dersler...");
       }
}
```

Kaynakça

Liang, Y.D. (2015) Introduction to Java Programming Comprehensive Version 10th Edition, New Jersey, Pearson

İnternet Kaynakları

docs.oracle.com

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

en.wikipedia.org

tr.wikipedia.org

gelecegiyazanlar.turkcell.com.tr

webrazzi.com

Bölüm 3

Java'da Temel Programlama Deyimleri

Öğrenme Çıktıları

Değişkenler

1 Java programlama dilinde değişken tanımlayabilme
2 Değişken isimlendirme kurallarına uyma
3 Temel veri tiplerini (ilkel tipler, karakter dizgileri, diziler) kullanabilme

Operatörler

4 Java programlama dilindeki operatörleri kullanabilme

Kontrol Akış Deyimleri

5 Kontrol akış deyimlerini kullanarak döngüler ve dallanmalar yaratabilme

Anahtar Sözcükler: • Değişken • İlkel Veri Tipi • Operatör • Dallama • Döngü • Dizi • String • Karar Verme

GİRİŞ

Bu ünitede Java programlama dilinin temelleri üzerinde duracağız. Nesnelerin durumlarının saklandığı alanlar ya da başka bir ifade ile değişkenler kavramını öğreneceğiz. Java programlama diliyle birlikte yerleşik olarak gelen ilkel veri tiplerinin yanı sıra, String ve dizi gibi değişken çeşitleri de bu ünite kapsamında işlenecektir. Türüne göre değer atayabildiğimiz bu değişkenler üzerinde aritmetik, atama, karşılaştırma gibi çeşitli işlemler gerçekleştirmek için var olan operatörleri tanıyacağız. Takip eden bölümlerde ise satır sırasını takip etme mantığı ile çalışan bir programın akışının karar verme, döngü ve dallanma deyimleri ile nasıl şartlı olarak kontrol edilebileceğini göreceğiz. Bu üniteyi tamamladığınız zaman Java programlama dilindeki en temel programlama yapı taşlarını öğrenmiş olacaksınız.

DEĞİŞKENLER

Java nesneye yönelik bir programlama dilidir. Bir Java programında tüm unsurlar birer nesnedir. Genel olarak nesnelerin durumu ve davranışı vardır. Nesnenin davranışını ifade etmek için metotlar kullanılır. Öte yandan bir nesnenin durumu o nesnenin alanlarında saklanır. Örneğin, Kişi nesnesinin alanları ad, soyad, yaş, cinsiyet olabilir.

```
/* Kişi.java */
public class Kişi {
    String ad;
    String soyad;
    char cinsiyet;
    int yaş;
}
```

Bu alanların değerleri değişkenlerde saklanır. Bir kişi nesnesinin ad değişkeninin değeri "İlayda" iken başka bir kişi nesnesinin adının değeri "Melike" olabilir.

Değişkenlerin Adlandırması

Değişkenlere bir isim verirken uyulması geren kurallar vardır ve kurallar kullanılan programlama diline göre değişiklik gösterebilir. Java programlama dilinde değişkene isim verirken aşağıdaki kurallara uymamız gerekir:

- Değişken isimleri küçük büyük harfe duyarlıdır.
- Değişken isimleri harf ile başlamalıdır. Alt çizgi (_) veya dolar işareti ($) ile de başlayabilir ama bunların kullanılması okunurluk açısından tavsiye edilmez.

dikkat

Java programlama dilinde değişken isimleri rakam ile başlayamaz!

- İlk karakterden sonra gelecek olan karakterler harf, sayı, dolar işareti veya alt çizgi olabilir.
- Java programlama dilinde ayrılmış özel kelimelerden (class, abstract, default, super, vb) birisi olamaz. Java programla dilinde değişken ismi olarak kullanılması yasaklı ayrılmış özel kelimelerin tam listesi Tablo 3.1'de sunulmuştur.
- Eğer değişken ismi bir kelimeden oluşuyorsa hepsini küçük harf ile yazmak tavsiye edilir. Örneğin cinsiyet. Eğer birden fazla kelimeden oluşuyor ise, ilk kelimeden sonraki kelimelerin baş harfinin büyük yazılması tavsiye edilir. Örneğin: "satış fiyatı" gibi iki kelimeden oluşan bir değişkenin adının satış fiyatı olarak tanımlanması tavsiye edilir.
- Eğer değişken sabit bir değeri saklayacak ise tamamen büyük harf kullanılması ve kelimelerin alt çizgi ile ayrılması tavsiye edilir. Örneğin java.lang. Math sınıfı içinde π sabit sayısı saklamak için değişken ismi olarak PI seçilmiştir.

```
public static final double PI = 3.14159265358979323846;
```

Diğer bir sabit değer saklayan değişken ismine örnek olarak da java.lang.Long sınıfının azami değerini tutan MAX_VALUE değişkeni verilebilir.

```
public static final long MAX_VALUE = 0x7fffffffffffffffL;
```

Tablo 3.1 Java Dili Anahtar Sözcükler Listesi

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

İlkel Veri Tipleri

Java programlama dilinde bir değişkeni kullanabilmemiz için öncesinde onu tanımlamamız gerekmektedir. Örneğin,

```
int yaş = 25;
```

Bu tanımlama ile programa ismi *yaş* olan bir değişken olduğu, bu değişkenin tam sayı (int) cinsinden değerler alabileceği ve bu değişkenin ilk değerinin 25 olduğu belirtilmiş olur. Bir değişkenin veri tipi, o değişkenin ne türden veri saklayabileceğini belirtir. Bu örneğimizde tam sayı için kullanılan int veri tipini kullandık. Java programla dilinde int veri tipi de dahil olmak üzere toplam sekiz tane ilkel veri tipi vardır. Tablo 3.2'de listelenmiş bu ilkel veri tipleri programlama dili tarafından önceden tanımlanmışlardır. Ayrıca bu veri tiplerinin isimleri ayrılmış özel kelimelerdir ve değişken ismi olarak kullanılamazlar.

Tablo 3.2 Java Programlama Dilinin Sağladığı İlkel Veri Tipleri

Veri Tipi	Varsayılan Değeri	Boyut (bit)	Açıklama
byte	0	8	–128 ile 127 arasındaki tamsayı değerlerini tutmak için kullanılır.
short	0	16	–32768 ile 32767 arasındaki tamsayıları tutmak için kullanılır.
int	0	32	-2^{31} ile $2^{31} - 1$ arasındaki tamsayıları tutmak için kullanılır.
long	0L	64	-2^{63} ile $2^{63} - 1$ arasındaki tamsayıları tutmak için kullanılır.
float	0.0f	32	Gezer noktalı (floating point) sayıları tutmak için kullanılır.
double	0.0d	64	Gezer noktalı (floating point) sayıları tutmak için kullanılır.
char	'\u0000'	16	16-bitlik tek bir Unicode karater değerini tutmak için kullanılır.
boolean	false	1	Sadece doğru (true) yanlış (false) değerlerini alabilir.

Tablo 3.2'de listelenmiş ilkel veri tiplerinin yanı sıra, karakterlerden oluşmuş metin verisi için java.lang.String sınıfını vardır. Çift tırnak içine alınmış bir metin için otomatik olarak yeni bir String nesnesi yaratılır. Örneğin, String ad = "Zeynep" değişkeni teknik olarak String sınıfı ilkel veri tipi değildir; ancak Java programlama dili tarafından özel olarak desteklenen bu sınıfı temel veri tipleri arasında düşünebiliriz.

Bir değişkeni tanımlarken her zaman ona bir değer atamak zorunda değiliz. Eğer açıkça bir değer atanmadan bir değişkeni tanımlarsak, derleyici o değişkene tipine göre bir varsayılan (default) değer atar. Tablo 3.2 her bir ilkel veri tipi için varsayılan değerleri göstermektedir. Bu tablodaki varsayılan değerleri kendiniz de bir kaç satırlık kod yazarak kolaylıkla elde edebilirsiniz. Örneğin aşağıdaki kod String, int, ve char veri tiplerinin derleyici tarafından otomatik atanan varsayılan değerini görmenizi sağlar.

```
public class Kişi {

    String ad;
    String soyad;
    char cinsiyet;
    int yaş;

    public static void main(String[] args) {

        Kişi kişi = new Kişi();

        System.out.println("String varsayılan: " + kisi.ad);
        System.out.println("char varsayılan: " + kisi.cinsiyet);
        System.out.println("int varsayılan: " + kisi.yas);
    }
}
```

Sabitler

İlkel veri tiplerinden birisiyle tanımlanmış bir değişkene değer atarken new anahtar sözcüğünü kullanmayız. Bunun sebebi, ilkel tipler sınıflardan yaratılmış nesneler değildir. Sabitler kodun içinde değişkenlere atadığımız değerlerdir. Örneğin, aşağıda değişkenlere sabit değerler atanmıştır.

```
String ad = "Ahmet";
String soyad = "Arslan";
char cinsiyet = 'E';
int yaş = 35;
```

Tam Sayı Sabitler

Tam sayılar için kullanılan ilkel veri tipleri: byte, short, int ve long'dur. Bunlara sayılardan oluşmuş değerler atanabilir. Burada long tipinde bir değişkene atama yaparken atama yaptığımız sayının long cinsinden olduğunu belirtmek için rakamların sonuna **L** ya da **l** harfini koyulur. long tipi int tipinin yetmediği büyüklükteki tam sayıları tutmak için kullanılır. Örneğin: long l = 15000L;

dikkat

Long sabit değerleri için, küçük harf yerine büyük harf L kullanılması tavsiye edilir. Çünkü küçük harf l ile 1 rakamı birbirine çok benzemektedir ve gözle ayırt etmek zor olabilir.

Tam sayı değerlerini ikilik, onluk veya on altılık sistemde yazabiliriz. Onluk sistem günlük hayatta kullandığımız 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 rakamları ile sayıların yazıldığı sistemdir. On altı'lık sistemde ise sayıları ifade etmek için 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F sembolleri kullanılır. İkilik sistemde ise sadece 0 ve 1 kullanılır.

```
// onluk tabanda 11 sayısı
int onlukSayı = 11;

// onaltılık tabanda 11 sayısı
int onaltılıkSayı = 0xB;

// ikilik tabanda 11 sayısı
int ikilikSayı = 0b01011;

System.out.println(onlukSayı + " " + onaltılıkSayı + " " + ikilikSayı);
```

Yukardaki örnekte, 11 sayısı her üç sistemde de ifade edilmiştir. Bu üç değişkeni ekrana yazdırdığımızda üçününde aynı değeri içerdiğini görürüz. Burada dikkat edilmesi gereken nokta on altılık sistemde bir sayı yazacağımız zaman sayının başına **0x**, ikilik sistemde yazacak isek **0b** koymamız gerektiğidir. Ancak bu şekilde derleyici söz konusu sayı sistemlerini algılayabilir ve ayırt edebilir.

Gezer Noktalı Sabitler

Gezer noktalı sayıları ifade etmek için double ya da float tipleri kullanılır. Double tipinin duyarlılığı daha fazladır. Eğer ondalıklı sayı **F** ya da **f** harfi ile biterse float olarak değerlendirilir. **D** ya da **d** harfi ile bitiyorsa double olarak değerlendirilir. Gezer noktalı sayıları **bilimsel gösterim**-de de yazabiliriz. Bu gösterimde **E** ya da **e** harfi kullanılır. Bu harflerden sonra gelen sayı ise 10'un katını ifade eder. Örneğin, ışığın hızını 3×10^8 bilimsel gösterimde yazmak istersek **3.0e8** olarak yazarız.

Bilimsel Gösterim

Bilim insanlarının çok büyük ya da çok küçük sayıları daha anlaşılır ve okunur olarak ifade etmesidir. Örneğin, 0.0000000056 yerine 5.6×10^{-9} yazılır. Bu sayı Java dilinde float ve double tipleri için 5.6e-9 ya da 5.6E-9 olarak yazılabilir.

```
double d = 0.5d;
float f = 0.5f;
double ışığınHızı = 3.0e8;
```

Okunurluğu artırmak için sayısal değişken değerleri için rakamlı gruplamak ya da ayırmak için alt çizgi _ kullanabiliriz. Örneğin, bir telefon numarasını alan kodundan sonra _ koyarak yazabiliriz. Bu yazım biçimi sadece kozmetik amaçlıdır. Sayının değerinde bir değişim olmaz.

```
long telefonNo = 222_321_3550L;
```

Karakter ve String Sabitler

String ve char sabitleri Unicode (UTF-16) karakterlerinden oluşurlar. Eğer metin editörü ve dosya sistemi izin veriyorsa ASCII dışında kalan karakterleri (ğ, ü, ş, ı, ö, ç, Ğ, Ü, Ş, İ, Ö, Ç, gibi) direk olarak kodun içinde kullanabiliriz. Örneğin: String ad = "Çiğdem";

Bu tür karakterleri sadece ASCII karakterleri kullanarak yazmanın yolu vardır. Her karakterin Unicode değerinin 16'lık tabanda gösteriminin başına \u koyarak yazarsak o karakterin kendini yazmış gibi oluruz.

```
String ad = "\u00c7i\u011fdem";
```

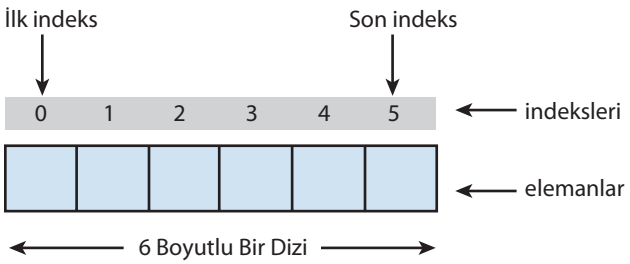
Bu örnekte **Ç** yerine **\u00c7**, **ğ** yerine ise **\u011f** kullanılmıştır. Bu değişkeni ekrana yazdırdığımızda ise çıktı olarak Çiğdem görülecektir.

ASCII
Harflerin bilgisayar ortamında saklanması ve taşınması ile ilgili geliştirilen ilk sistemdir ve Türkçe'de Bilgi Değişimi İçin Amerikan Standart Kodlama Sistemi anlamına gelir. Bu sistemde Amerikan İngilizcesi alfabesinde bulunan harflerin her birine bir sayı atanmış ve harfler bilgisayar ortamında sayı olarak saklanmıştır.

Unicode (Evrensel Kod)
Unicode Consortium organizasyonu tarafından geliştirilen ve her karaktere bir sayı değeri karşılığı atayan bir endüstri standardıdır. Sistemin amacı farklı karakter kodlama sistemlerinin birbiriyle tutarlı çalışmasını ve dünyadaki tüm yazım sistemlerinden metinlerin bilgisayar ortamında tek bir standart altında temsil edilebilmesini sağlamaktır.

Diziler

Diziler, tek bir türde sabit sayıdaki değerleri muhafaza ederler. Bir dizini boyutu ilk yaratıldığında belirlenir, sonradan değişmez.



Şekil 3.1 Altı elemanlı bir dizi

Şekil 3.1'de altı elemanlı bir dizinin hayali gösterimi resimlendirilmiştir. Bir diziyi oluşturan öğelere eleman denir. Her bir elemana ait sayısal bir indeks değeri vardır ve elemanlara bu indekslerle erişilir. Şekil 3.1'de görüleceği üzere, indeks dizini sıfırdan başlar. Bir tipin dizisini tanımlamak için köşeli parantezler [ ] kullanılır. Aynı şekilde indeks değerleri de köşeli parantezler içine yazılır. Aşağıdaki örnekte 6 uzunluğunda int tipinde değerler tutan bir dizi yaratılıp, ilk ve son elemanına atama yapılmıştır.

```
int[] dizi;

dizi = new int[6];

dizi[0] = 10;
dizi[5] = 19;
```

Burada dikkat edilmesi gereken nokta, dizi isimli değişken tanımlandıktan sonra new anahtar sözcüğü ile 6 tamsayı için bellekte yer ayrılmasıdır. Daha sonra köşeli parantezler içerisine indis sayısını yazmak suretiyle dizinin elemanlarına atama ya da erişim yapıldığı gösterilmiştir.

Bir diziyi tanımlamada iki parça vardır: köşeli parantezlerden önce yazılan dizinin tipi ve köşeli parantezlerden sonra yazılan dizinin adı. Dizinin tipi olarak ilkel veri tiplerini (int, byte, long, vb) kullanabileceğimiz gibi sınıflardan yaratılmış nesnelerde kullanabiliriz.

Dizilerle çalışırken, indisin değeri sıfırdan başlar ve toplam eleman sayısının bir eksiğine kadar gider. Bu sınırlar dışında erişim yapmaya çalışırsanız java.lang.ArrayIndexOutOfBoundsException hatası alırsınız. Derleyici dizinin sınırlarının ihlal edilip edilmediğini kontrol etmez. Bu iş programcının sorumluluğundadır.

```
/**
 * Kişileri temsil eden sınıf
 */
public class Kişi {

    String ad;
    String soyad;
    char cinsiyet;
    int yaş = 25;

    public static void main(String[] args) {

        Kişi[] kişiler = new Kişi[6];

        kişiler[0] = new Kişi();
        kişiler[5] = new Kişi();
    }
}
```

Bu örnekte ise, bir önceki örnekte int tipi için yaptığımız şeyi Kişi sınıfından türetilmiş nesneler için tekrarlıyoruz. Buradaki tek fark dizinin elemanlarına atama yaparken new anahtar sözcüğü ile her seferinde yeni bir Kişi nesnesi yaratmamız. İlkel veri tipleri için bunu yapmamıştık.

Bir kaynak diziyi, başka bir hedef diziye kopyalamak işini en hızlı şekilde gerçekleştirmenin yolu java.lang.System#arraycopy metodunu kullanmaktır. Bu metot Java programlama dilinde doğuştan var olduğu (native) için çok hızlı çalışır.

Diziler üzerinde hayatı kolaylaştıran işlemleri gerçekleştiren bir takım metotlar java.util.Arrays sınıfı içinde mevcuttur. Örneğin, dizileri sıralamak ve dizileri String tipine çevirme işlemleri aşağıdaki örnekte gösterilmiştir.

```
// sıralama öncesi bazı tam sayılar
int [] sayılar = {5, -9, 8, 1, 0, -3};
System.out.println(Arrays.toString(sayılar));

// sıralama sonrası tam sayılar
Arrays.sort(sayılar);
System.out.println(Arrays.toString(sayılar));
```

Bu programın çıktısı sıralama öncesi ve sonrası olmak üzere şu şekildedir:

```
[5, -9, 8, 1, 0, -3]
[-9, -3, 0, 1, 5, 8]
```

Görüldüğü üzere, tek satırda dizinin elemanları küçükten büyüğe doğru sıralanmıştır.

Paralel Sıralama

Java 8 sürümü ile gelen bir yenilik olan java.util.Arrays#parallelSort metodu programın çalıştırıldığı bilgisayarın işlemcilerini veya çekirdeklerini kullanarak dizileri paralel bir şekilde sıralar. Çok işlemcili bilgisayarlarda bu işlem java.util.Arrays#sort işleminden daha hızlı olacaktır.

Öğrenme Çıktısı

1 Java programlama dilinde değişken tanımlayabilme
2 Değişken isimlendirme kurallarına uyma
3 Temel veri tiplerini (ilkel tipler, karakter dizgileri, diziler) kullanabilme

Araştır 1

int[] dizi = new int[6]; dizi[6] = 19; Java kodunu çalıştırın ve karşılaşacağınız hata mesajını internetten araştırın. Bu hatanın neden kaynaklandığını düşünün. Bu hata nasıl engellenebilir?

İlişkilendir

100 metrelik bir çit ördüğümüzü düşünelim. Her 10 metrede bir direk çakmamız gerektiği durumda toplamda kaç direk kullanılırız? Çitte 10 tane bölüm olmasına rağmen 11 adet direk çakmamız gerekir. Buna çit problemi adı verilir. Çit problemi ile int[] dizi = new int[6]; dizi[6] = 19; kodunun verdiği hatayı ilişkilendirin.

Anlat/Paylaş

System.out.println ("\u611B"); kodunu çalıştırın ve ekrana basılan karakterin hangi dile ait olduğunu tahmin etmeye çalışın.

OPERATÖRLER

Değişkenler üzerinde çeşitli işlemleri gerçekleştirmeye yarayan özel sembollere operatör denir. Operatörler çeşidine göre bir, iki ya da üç adet işlenen üzerinde çalışırlar ve en sonda bir değer döndürürler. En sık kullanılan operatör değişkenlere değer atamaya yarayan eşittir "=" operatörüdür. Hâlihazırda eşittir atama operatörünü bu ünite içerisinde birçok kere kullandık. Bununla birlikte, Java programlama dilinde toplama, çıkarma, bölme ve çarpma gibi aritmetik işlemler için de operatörler bulunmaktadır.

Aritmetik Operatörler

Java programlama dilindeki 5 temel aritmetik işlem için operatörler Tablo 3.3'te listelenmiştir. Bu operatörler temel matematik derslerinde aşina olduğumuz dört işlemi gerçekleştirmek için kullanılırlar.

Tablo 3.3 Java Programlama Dilinin Aritmetik Operatörleri

Operatör	Açıklama
+	Toplama işlemi
-	Çıkarma işlemi
*	Çarpma işlemi
/	Bölme işlemi
%	Modulo işlemi

```
int kalan = 11 % 3;
System.out.println ("11 mod 3 = " + kalan);
```

Modulo işlemi
Hesaplamada bir sayının diğer bir sayıya bölümünden arta kalan sayıyı verir.

Yukarıdaki program ekrana 11 mod 3 = 2 yazdırır.

Bileşik Atama

Java programlama dilinde basit atama için "=" sembolünü kullandığımızı söylemiştik. Tablo 3.3'te listelenen operatörleri basit atama operatörü ile birleştirerek bileşik atama işlemi gerçekleştirebiliriz. Örneğin, bir x değişkeninin değerini 3 artırmak istediğimizde 'x += 3;' ifadesini kullanabiliriz. Bu ifade 'x = x + 3;' ifadesi ile aynı anlama gelmektedir. Bileşik atamada toplama dışındaki diğer aritmetik operatörleri de kullanabiliriz.

Birli Operatörler

Tek bir işleneni olan operatörlere birli operatör denir. Bu tür operatörler; bir artırma, bir azaltma, negatifleme veya tersleme işlemleri için kullanılırlar. Bu tip operatörler Tablo 3.4'te gösterilmiştir.

Tablo 3.4 Java Programlama Dilinin Birli Operatörleri

Operatör	Açıklama
+	Birli toplama operatörü değerin pozitif olduğunu belirtir.
-	Birli çıkarma operatörü (değeri negatifler)
++	Artırma operatörü, değişkenin değerini 1 artırır.
--	Azaltma operatörü, değişkenin değerini 1 azaltır.
!	Mantıksal terseleme operatörü boolean tipindeki bir değişkenin değerini ters çevirir.

Artırma ve azaltma operatörlerini sonek (x++) ya da önek (++x) olarak kullanmak mümkündür. Eğer bu iki örneği tek bir satır kodda kullanırsak aralarında hiçbir fark olmaz: x'in değeri bir artırılır. Ama daha büyük bir ifadenin parçası olarak kullanırsak aralarındaki fark büyüktür. Değişkenin soluna operatörü yazdığımızda (++x), önce artırma işlemi yapılır daha sonra diğer işlemlere geçilir. Sağına yazdığımızda ise (x++), önce diğer işlemler yapılır sonra artırım işlemi yapılır.

Eşitlik ve İlişki Operatörleri

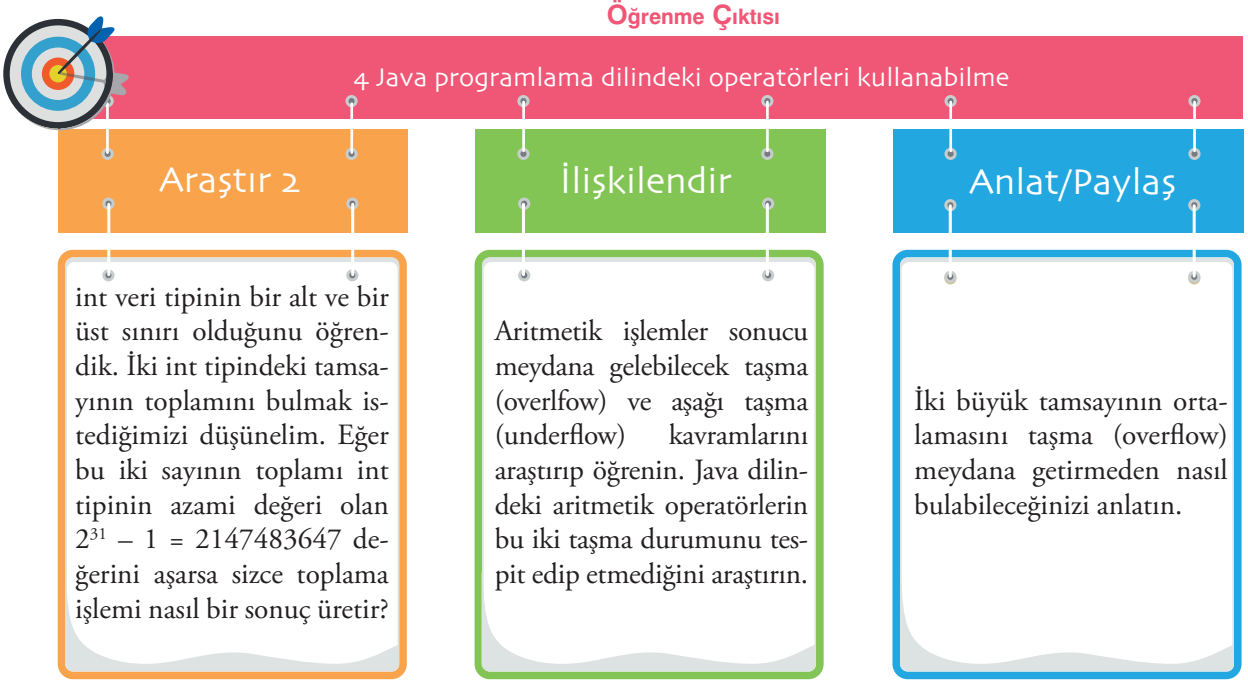
Değişkenlerin değerlerini birbirleriyle karşılaştırmak için kullanılan operatörler Tablo 3.5'te listelenmiştir.

Tablo 3.5 Java Programlama Dilinin Eşitlik ve İlişki Operatörleri

Operatör	Açıklama
==	Eşittir
!=	Eşit değildir
>	Büyüktür
>=	Büyük ya da eşittir
<	Küçüktür
<=	Küçük ya da eşittir

İki ilkel veri tipleri için eşit olma durumu == operatörü ile kontrol edilir. Bu operatör basit atama operatörü = ile karıştırılmamalıdır.

Eşitlik ve ilişki operatörleri boolean tipinde değer döndürürler. Birden fazla boolean döndüren ifadeyi VE ile VEYA operatörleri olan **&&** ile **||** kombine etmek mümkündür. Bu tür operatörleri sıradaki kısımda kontrol akış deyimlerinde yoğun bir şekilde kullanacağız.



KONTROL AKIŞ DEYİMLERİ

Java kodları genelde satır satır yukarıdan aşağıya doğru sırayla çalıştırılır. Öte yandan, kontrol akış deyimleri bu akışı üç şekilde değiştirilebilir:

- karar verme (if-then, if-then-else, switch)
- döngü (for, while, do-while)
- dallanma (break, continue, return)

Bu bölümde Java dilinin desteklediği yukardaki listelenmiş kontrol deyim çeşitleri anlatılmaktadır.

Karar Verme Deyimleri

Günlük hayatta birçok iş bir **koşul**a bağlı olarak karar verme ile gerçekleştirilir. Eğer dönemlik not ortalamamız 3.0'ın üstünde ise onur sertifikası alabiliriz; 15 yaşını tamamlamış olanların kimlik başvurusunu şahsen başvurması gerekmektedir. Bilgisayar programlarında da benzer şekilde sınamaya bağlı olarak karar vermemiz gereken yerler vardır ve farklı koşullara göre farklı kod blokları çalıştırmamız gerekir.

if-then

If-then ifadesi tüm kontrol akış deyimlerinin en temelidir. Belirli bir sınama doğru olarak değerlendirildiğinde programınıza belirli bir kod bölümünü çalıştırmasını söyler. Kontrol akış deyimleri arasında en basit ve en temel olanıdır. Sadece eğer belli bir sınama doğru olarak değerlendirildiğinde programın belirli bir kısmının çalışmasını sağlar. Örneğin, sadece 18 yaşını tamamlamış kişilerin başvurabileceği bir iş ilanı olsun. Yaşı verilen bir kişinin söz konusu işe başvuru yapıp yapamayacağını sınayan program parçacığı şu şekilde olabilir:

```
if (yaş >= 18) {
    işeBaşvur();
}
```

Bu program parçacığında kişinin yaşı yaş isimli bir tam sayı değişken ile ifade edilmektedir. İngilizcede "eğer" anlamına gelen **if** ifadesi ile kişinin yaşının 18'e eşit veya 18'den büyük olup olmadığı kontrol edilir. Eğer kişinin yaşı 18 veya 18'den büyük ise iş başvuru işleminden sorumlu işeBaşvur metodu çalıştırılır. Eğer bu sınama yanlış (false) olarak değerlendirilirse program akışı if-then ifadesinin sonuna atlar. Bir başka ifadeyle işeBaşvur metodu çalıştırılmaz.

Açılış ve kapanış kaşlı ayraçlarını { } kullanmak "then" ifadesinin yalnızca bir satır içermesi koşuluyla isteğe bağlıdır:

```
if (yaş >= 18)

    işeBaşvur();
```

Bu durumda kaşlı ayraçları kullanıp kullanmamak kişisel tercihlerinize bağlıdır.

dikkat

Kaşlı ayraçların { } kullanılmaması kodu daha kırılgan hâle getirebilir. Eğer "then" maddesine ikinci bir satır sonradan eklenirse yeni gerekli kaşlı ayraçları eklemeyi unutmak sık yapılan bir hatadır. Derleyici bu tür hataları yakalayamaz ve yanlış sonuçlar üretir.

if-then-else

If-then-else deyimi, bir "if" sınaması yanlış (false) olarak değerlendirildiğinde, ikincil bir yürütme yolu sağlar. İş başvurusu örneğinde bir if-then-else ifadesi kullanabiliriz. Kişinin yaşının 18'den küçük olması durumunda, işe başvuru yapamayacağını belirten bir hata mesajı gösterecek eylemin çalışmasını sağlayabiliriz.

```
if (yaş >= 18) {
    işeBaşvur();
}
    else {
            System.err.println("işe
başvuramazsınız!");
     }
}
```

Aşağıdaki program, Yabancı Dil Bilgisi Seviye Tespit Sınavı (YDS) puanının harf seviyesi karşılığını verir:

Puan Aralığı	Düzey
90-100	A
80-89	B
70-79	C
60-69	D
50-59	E
0-49	-

```
int YDSPuanı = 82;
char seviye;

if (YDSPuanı >= 90) {
    seviye = 'A';
} else if (YDSPuanı >= 80) {
    seviye = 'B';
} else if (YDSPuanı >= 70) {
    seviye = 'C';
} else if (YDSPuanı >= 60) {
    seviye = 'D';
} else if (YDSPuanı >= 50) {
    seviye = 'F';
} else if
    seviye = '-';
}

System.out.println("Seviye = " + seviye);
```

Bu program 82 YDS puanı için "Seviye = B" şeklinde çıktı verir.

dikkat

YDS Puanı değişkeninin değerinin birden fazla sınama koşulunu sağlayabileceğini fark etmişsinizdir: 82 >= 80 ve 82 >= 90 ancak bir koşul sağlandığında, uygun ifade yürürlüğe girer (seviye = 'B';) ve kalan koşullar değerlendirilmez.

Switch

if-then ve if-then-else ifadelerinden farklı olarak switch ifadesi bir dizi yürütme yollarına sahip olabilir. Bir switch ifadesi, byte, short, char ve int gibi ilkel veri türleriyle çalışır. Ayrıca switch ifadesi numaralandırılmış sıralı tiplerle (Enum), dizgi (String) sınıfında ve bazı ilkel türleri saran Character, Byte, Short ve Integer gibi birkaç özel sınıfla birlikte çalışır.

Aşağıdaki kod örneğinde, değeri haftanın gününü temsil eden int tipinde gün isimli bir değişken tanımlanmıştır. Kod, switch ifadesini kullanarak gün değişkeninin değerinin haftanın hangi gününe denk geldiğini ekrana yazdırır. Bu örnekte, program ekrana "Salı" yazdırır.

```
int gün = 2;
String haftanınGünü;

switch (gün) {
    case 1:
        haftanınGünü = "Pazartesi";
        break;
    case 2:
        haftanınGünü = "Salı";
        break;
    case 3:
        haftanınGünü = "Çarşamba";
        break;
    case 4:
        haftanınGünü = "Perşembe";
        break;
    case 5:
        haftanınGünü = "Cuma";
        break;
    case 6:
        haftanınGünü = "Cumartesi";
        break;
    case 7:
        haftanınGünü = "Pazar";
        break;
    default:
        haftanınGünü = "Geçersiz gün";
        break;
}

System.out.println(haftanınGünü);
```

Bir switch deyiminin gövdesi switch bloğu olarak bilinir. Switch bloğundaki bir ifade ya case ya da default etiketleriyle etiketlenebilir. Örnekte görüldüğü gibi bir switch bloğu içinde case etiketi birden fazla olabilir. Switch deyimi ifadesini değerlendirir ve akabinde ifadenin eşleştiği case etiketinden sonra gelen tüm ifadeleri çalıştırır.

Haftanın gününü gösteren örneği if-then-else kullanarakta yazabilirdik:

```
if (gün == 1)
    System.out.println("Pazartesi");
else if (gün == 2)
    System.out.println("Salı");
else if (gün == 3)
    System.out.println("Çarşamba");
    … // ve devamı
```

If-then-else deyimlerini veya switch deyimini kullanılıp kullanılmayacağına karar vermek okunabilirlik ve test edilen ifadeye dayanır. If-then-else deyimi, değerleri veya koşulların aralıklarına dayalı ifadeleri sınarken switch ifadesi ifadeleri yalnızca tek bir tam sayı, numaralandırılmış değer veya String nesnesine dayalı olarak test eder.

Diğer bir ilgi çekici nokta da break ifadesidir. Her break ifadesi, switch deyimini sonlandırır. Kontrol akışı, switch bloğundaki ilk ifadeyle çalışmaya başlar. Switch bloğunun doğru çalışması için break ifadeleri gereklidir. Eğer break ifadeleri kullanılmazsa eşleşen case etiketinden sonraki tüm ifadeler, sonraki case etiketinin eşleşip eşleşmediğine bakılmaksızın, bir break ifadesi bulunana kadar çalışmaya devam eder. Aşağıdaki program, break ifadeleri kullanılmadığı zaman bir switch bloğundaki ifadelerin nasıl ve hangi sırada çalışacağını gösterir.

dikkat

Switch deyimini kullanırken break ifadelerini koymayı unutmayın!

```
int gün = 5;

switch (gün) {
    case 1:
        System.out.println("Pazartesi");
    case 2:
        System.out.println("Salı");
    case 3:
        System.out.println("Çarşamba");
    case 4:
        System.out.println("Perşembe");
    case 5:
        System.out.println("Cuma");
    case 6:
        System.out.println("Cumartesi");
    case 7:
        System.out.println("Pazar");
}
```

Program, ilk eşleşen case ifadesiyle (case 5:) birlikte sonraki bütün case deyimlerini otomatik olarak çalıştırmaktadır. Programın çıktısı

- Cuma
- Cumartesi
- Pazar

olacaktır. Program beşinci gün olan cumayı ve cumadan sonra gelen diğer günleride yazdırır. Kuşkusuz bu istediğimiz bir durum değildir. İşte bu yüzden break ifadesi kullanılmaktadır.

Aşağıdaki örnekte ise bir kontrol ifadesinde birden fazla case etiketinin kullanımı gösterilmektedir.

```
String haftanınGünü = "Pazar";

switch (haftanınGünü) {
    case "Pazartesi":
    case "Salı":
    case "Çarşamba":
    case "Perşembe":
    case "Cuma":
        System.out.println("iş günü");
        break;
    case "Cumartesi":
    case "Pazar":
        System.out.println("tatil günü");
        break;
    default:
        System.out.println("geçersiz bir gün");
        break;
}
```

Bu örnekte, verilen bir günün iş günümü yoksa tatil günümü olduğu sınanmaktadır. Pazartesi, salı, çarşamba, perşembe, cuma günlerinden herhangi biri olması durumunda ekrana "iş günü" yazdırılmaktadır. Cumartesi veya pazar olması durumunda ise "tatil günü" yazdırılmaktadır.

Burada en sonuncu break ifadesini koymasakta olur çünkü program akışı satır satır işletildiğinden zaten switch bloğunun sonuna gelmişizdir akış break ifadesi olmasada devam eder. Ama bu durumda bile break ifadesinin kullanılması tavsiye edilmektedir çünkü kodun daha sonra değiştirilmesini kolaylaştırır ve kodun hataya olan eğilimini azaltır. Hiçbir case ifadesinin eşleşmediği durumda ise default etiketi ile etiketlenmiş blok çalışır. Örneğin girdi olarak programa "Ocak" verirsek program ekrana "geçersiz bir gün" yazacaktır.

Bu örneğimizde, switch deyiminin ifadesi olarak String nesnesi kullandık. Her bir case etiketindeki ifade ile sınama ifadesi karşılaştırılırken String.equals metodu kullanılır. Bu örnekteki yapılan karşılaştırılmalar aslında küçük/büyük harfe duyarlıdır.

dikkat

Switch deyimi içerisinde String nesnesi kullanabilmek için en düşük Java 7 sürümü gerekir. Bundan önceki JDK sürümlerinde bu özellik desteklenmemektedir.

Döngü Deyimleri

Bazı durumlarda bir kod bloğunu defalarca çalıştırmamız gerekebilir ya da kod bloğunu belirli bir sınama doğru olana kadar çalıştırmak isteyebiliriz. Hatta bazı programlar **sonsuz döngü** içerisinde çalışmaktadır.

while ve do-while

While deyimi, belirli bir koşul geçerli olduğu sürece bir kod bloğunu sürekli çalıştırır. Söz dizimi şu şekilde ifade edilebilir:

```
while (ifade)
{
    // bir şeyler yap
    // başka şeyler daha yap
}
```

While deyimi, bir boolean değeri döndürmesi gereken sınama ifade değerini değerlendirir. Sınama ifadesi doğru (true) olarak değerlendirilirse while deyimi while bloğundaki ifadeleri sırasıyla çalıştırır. While ifadesi, sınama ifadesinin test edilmesine ve içindeki ifadeleri sırayla çalıştırmaya, sınama ifadesinin yanlış (false) olarak değerlendirilene kadar devam eder.

Örneğin, bölme işlemini while ifadesi ile sadece çıkarma işlemi kullanarak yapabiliriz.

```
private static void çıkarmaİleBölme(int bölünen, int bölen)
{

    int bölüm = 0;

    while (bölünen > = bölen) {
        bölünen = bölünen - bölen;
        bölüm++;
    }

    System.out.println("bölüm = " + bölüm);
}
```

Bu örneğimizde, bölünen bölenden büyük veya eşit olduğu sürece bölüneni bölen kadar azaltırız. Her azaltma işleminin sonunda da, bölüm değerini 1 artırırız ve bu iki işlemi bölünen bölenden küçük olana kadar devam ederiz. Döngü sonlandığında, kaç defa çıkarma işlemini yaptığımızı sayan bölüm değişkeni bize tam sayı cinsinden bölme işlemini sonucunu vermiş olur. Bu örneği çıkarmaİleBölme(bölünen=25, bölen=3); değerleri ile çalıştıracak olursak, program çıktı olarak "bölüm = 8" verir.

While deyimi kullanılarak sonsuz döngü şu şekilde gerçeklenebilir:

```
while (true)
{
 // bir şeyler yap
}
```

Java programla dili do-while deyimini de sağlamaktadır ve şu kullanımı şu şekildedir:

```
do {
 // bir şeyler yap
} while (ifade);
```

Bu iki döngü deyimi arasındaki tek fark sınamanın başta veya sonda yapılmasıdır. Bu nedenle do-while deyimi içindeki kod bloğu en az bir kere çalıştırılır çünkü sınama sonda yapılmaktadır.

for

For deyimi, bir değer aralığı üzerinde yürümenin kompakt bir yoludur. Programcılar genellikle bunu "for döngüsü" olarak adlandırırlar çünkü belirli bir koşul sağlanana kadar döngü tekrar eder. For deyiminin genel biçimi şu şekilde ifade edilebilir:

```
for (hazırlık; bitiş; artırım) {
 // bir şeyler yap
}
```

Hazırlık: Döngüyü ilk kullanıma hazırlar, bir defaya mahsus döngünün en başında çalıştırılır.

Bitiş: Sonlandırma ifadesi yanlış (false) olarak değerlendirildiğinde döngü biter.

Artırım: Artırım ifadesi döngünün her bir yinelemesinde çalıştırılır. Genelde bu kısımda bir değişkenin değeri artırılır ya da azaltılır.

Örneğin, aşağıdaki program 1'den 9'a kadar olan rakamları yazdırır. Bu programın çıktısı: 1 2 3 4 5 6 7 8 9 10

```
for (int i = 1; i < 10; i++) {
 System.out.print(i + " ");
}
```

Bu örnekte, hazırlık ifadesi içinde *i* isimli bir değişken tanımlanmıştır. Bu değişkenin kapsamı tanımlanmasından itibaren başlar ve for döngüsü bloğunun sonuna kadar devam eder; bu nedenle de bitiş ve artırım ifadelerinde de kullanılabilir. Bir for deyimini kontrol eden değişken, döngü dışında gerekli değilse değişkeni hazırlık kısmında tanımlamak en iyisidir. For döngüsü kontrol değişkenleri genellik *i*, *j* ve *k* isimleri ile adlandırılır. Bu değişkenleri hazırlık ifadesi içinde tanımlamak değişkenlerin kapsamını sınırlar ve olası hataları azaltır.

For döngüsü için bahsettiğimiz *hazırlık*, *bitiş* ve *artırım* ifadeleri opsiyoneldir. Bu üç ifadeyi kullanmaksak sonsuz bir döngü yaratmış oluruz.

```
// sonsuz döngü
for ( ; ; ) {
 // bir şeyler yap
}
```

for-each

For döngüsünün, diziler ve kolleksiyonlar (listeler, kümeler vs) üzerinde yürümeyi daha kolaylaştıran ve kodu daha okunabilir kılan özel bir hâli vardır. Bu döngüye for-each döngüsü adı verilir. Bu for çeşidinde kontrol değişkeni tanımlandıktan sonra iki nokta (:) işareti kullanılır ve ardından üzerinde yürümek istediğimiz değişkeni yazarız.

```
List<Integer> liste =
IntStream.rangeClosed(1,10).
boxed().collect(Collectors.
toList());

for (int i : liste) {
 System.out.print(i + " ");
}
```

Bu örnekte, 1'den 9'a kadar olan tam sayıları içeren bir liste yaratılmıştır ve bu liste üzerinde for-each döngüsü ile yürünmektedir. Bu programın çıktısı bir önceki örneğin çıktısı ile aynıdır:

```
1 2 3 4 5 6 7 8 9 10
```

Mümkün olduğu durumlarda for döngüsünün bu çeşidini kullanmak tavsiye edilir. Çünkü bu hâli daha kısa ve anlaşılırdır. Ancak, **for-each döngüsü** sadece listenin elemanlarını sırasıyla gezmek içindir. Sadece okuma (read-only) yapmak için kullanılır. Eğer for-each döngüsü içinde liste üzerinde bir değişiklik yapmaya çalışırsanız java.util.ConcurrentModificationException hatası alırsınız.

```
List<Integer> liste =
IntStream.rangeClosed(1,10).
boxed().collect(Collectors.
toList());

for (int i : liste) {
   liste.add(i);
}
```

Örneğin, yukardaki kod parçasında for-each döngüsü içinde listeye ekleme yapılmaktadır. Bu program çalışmaz ve java.util.ConcurrentModificationException hatasını fırlatır.

Dallanma Deyimleri

Program içerisinde sıçrama ya da dallanma yapmak istediğimiz durumlar meydana gelebilir. Örneğin, bir döngü içerisindeyken döngüyü sonlandırmak isteyebiliriz ya da programın etiketlediğimiz bir satırına atlamak isteyebiliriz. Bu tür işlemleri gerçekleştirmek için Java programlama dilinde bize sunulan ifadeler çeşitli ifadeler vardır.

Break

Break ifadesini ilk olarak switch deyimini işlerken görmüştük. Break ifadesi for, while veya do-while döngülerini sonlandırmak için de kullanabiliriz. Bir break ifadesi en içteki switch, for, while veya do-while döngüsünü sonlandırır.

Örneğin, verilen bir dizinin küçükten büyüğe sıralı olup olmadığını test etmek istediğimizi düşünelim. Bu işlem için dizinin bütün elemanlarını for döngüsüyle tek tek gezip, bir elamanın bir sonraki elamandan küçük olması durumuyla ilk karşılaşmamızda dizinin sıralı olmadığını tespit olmuş oluruz. Bu durumda dizinin geri kalan diğer elemanlarını kontrol etmemize gerek yoktur. Sıralı olma kuralını ilk ihlal eden durumla karşılaşır karşılaşmaz break ifadesini kullanarak döngüyü sonlandırabiliriz. Aşağıda anlatılan durumun kodlanmış hâli verilmektedir.

```
int a[] = new int[]{1, 2, 1, 4, 5};

boolean sıralımı = true;

for (int i = 0; i < a.length - 1; i ++) {
    if (a[i] > a[i + 1]) {
        sıralımı = false;
        break;
    }
}

if (sıralımı)
    System.out.println("dizi sıralıdır");
else
    System.out.println("dizi sıralı değildir");
```

Continue

Continue ifadesi for, while veya do-while döngülerinin o anki yinelemesini atlamayı sağlar. En içteki döngünün gövdesinin en sonuna giderek en iç döngünün sınama ifadesinin yeniden değerlendirilmesini edilmesini sağlar.

Örneğin satır satır okumamız gereken bir dosya olsun ve bu dosyanın içinde diyez işareti (#) ile başlayan satırlar yorum satırları olsun. Biz bu yorum satırlarını atlamak istersek continue kullanarak aşağıdaki gibi bir program yazarız.

```
for (String satır : Files.readAllLines(Paths.get("birDosya.txt"))) {

    if (satır.startsWith("#"))
        continue;

    // geri kalan satırları ekrana bastır
    // veya başka bir işlem yap
    System.out.println(satır);

}
```

Bu örnekte, Files.readAllLines metodu bir dosyadaki bütün satırları okuyup bir liste döndürmektedir. Bu liste üzerinde for-each döngüsü ile dönerken eğer satır diyez işareti ile başlıyorsa continue deyimi çağrılır. Döngü bir sonraki satırla devam eder. Aksi hâlde ilgili satır ekrana yazdırılır. Böylece continue kullanarak yorum için yazılmış satırlar atlanıp diğerleri üzerinde işlem yapılmış olur.

dikkat

Files.readAllLines statik metodu Java 8 sürümünde eklenmiş bir özelliktir. Tek satırda dosya içindeki bütün satırları okuyarak dosya işlemlerini oldukça kolaylaştırmıştır ve basitleştirmiştir.

Return

Dallanma deyimlerinin en sonuncusu return ifadesidir. Return ifadesi o anki metoddan çıkar böylece kontrol akışı metodun çağrıldığı yere geri döner. Return ifadesinin iki çeşidi vardır: İlki değer döndüren ve diğeri ise değer döndürmeyendir. Aşağıda bir sayının asal sayı olup olmadığını bulan metot verilmiştir. Bu metodun geri dönderdiği değer true ya da false değerlerinden birini alabilen boolean tipindedir.

Asal Sayılar

Sadece kendisine ve 1 sayısına bölünebilen 1'den büyük pozitif tam sayılardır. Çift sayı olan tek asal sayı 2'dir. Diğer tüm asal sayılar tek sayılardır.

```
boolean sayıAsalmı(int sayı) {

    if (sayı == 2) return true;

    // sayı çift mi diye kontrol et
    // çift sayı ise asal değildir
    if (sayı % 2 == 0) return false;

    // eğer sayı çift değilse
    // sadece tek böleni (3, 5, 7, 9 ...)
    // var mı diye kontrol et
    for (int i = 3; i < sayı; i += 2) {
        if (sayı % i == 0)
            return false;
    }
    // buraya kadar hiç bir return ifadesi
    // ile çıkış olmadıysa sayı asaldır
    return true;
}
```

Bu örnekte, 4 adet return ifadesi kullanılmıştır. 2'nin özel durumu kontrol edilmekte ve sayı 2 ise true değeri return edilmektedir. Daha sonra programın devamında sayının çift olup olmadığı mod operatörü (%) ile kontrol edilip, sayı çift ise false return edilmektedir. Daha sonra 3'ten başlayan ve test edilen sayının değerine kadar giden tek sayıları üzerinde dönen bir for döngüsü çalışmaktadır. Eğer bu döngü içinde sayıyı tam bölen bir tek sayı bulursa anında false return edilmektedir. Eğer döngü sonunda, hiç bir return ifadesi çalışmadıysa metodun en son satırında true değeri geri döndürülür ve metot biter.

Yaşamla İlişkilendir

Türk Lirası simgesi Unicode karakter tablosuna 6.2 sürümü ile eklendi

Unicode Standardın 6.2 sürümü çıktı. Bu yeni sürümde yalnızca U+20BA (8378) karakter numarasıyla yeni Türk Lirası simgesi eklendi. Bununla birlikte, diğer pek çok karakterin özellikleri ve davranışları da düzeltildi. Duygusal emojilerin ve görüntüsel sembollerin bir takım davranışlarında belirgin iyileştirmeler sağlandı. Bazı karakterlerin sınıflandırılması hem iyileştirildi hemde daha iyi dokümante edildi. Unicode Karşılaştırma Algoritması, dokümantasyonunun büyük çapta revizyonu ile 6.2 sürümü için büyük ölçüde geliştirildi. Ayrıca, karşılaştırma ağırlık tablolarında, bazı görüntüsel sembollerin ağırlık değişimi de dahil olmak üzere önemli değişiklikler yapıldı.

Yeni kodlanmış Türk Lirası sembolünün yoğun şekilde kullanılması bekleniyor. Yeni Türk Lirası karakterine olan acil ihtiyacı karşılamak için, Unicode konsorsiyumu 6.2 sürümünün çıkışını hızlandırdı.

TL simgesi belirlenirken, Türk lirasının ve Türkiye ekonomisinin iki belirgin özelliği olan "güven" ve "istikrar içinde yükselen değer" kavramları ön plana çıkartılmıştır. Simgenin çıpaya benzemesi Türk lirasının kıymet saklama aracı olarak "güvenli bir liman hâline geldiğini" vurgulamaktadır. Paralel çizgilerin yukarı eğimli olması ise, Türk lirasının ve Türkiye ekonomisinin "istikrar içinde yükselen değerini" simgelemektedir.

Kaynak: http://blog.unicode.org/2012/09/announcing-unicode-standard-version-62.html

http://www.tcmb.gov.tr/wps/wcm/connect/tcmb+tr/tcmb+tr/bottom+menu/diger+faaliyetler/tl+simgesi

Öğrenme Çıktısı

5 Kontrol akış deyimlerini kullanarak döngüler ve dallanmalar yaratabilme

Araştır 3

Verilen *n* boyutlu bir tamsayı dizisinin elemanlarının toplamını bulan programı döngü ifadelerinden hiçbirini kullanmadan yazmak mümkün müdür? İpucu: özyineleme (recursion) hakkında araştırma yapıp problem üzerinde düşünün.

İlişkilendir

Özyinelemeli (recursive) metotları döngülerle ilişkilendirin.

Anlat/Paylaş

Bir kod parçacığının sonsuz döngü içinde çalışması gerektiği duruma bir örnek bulup anlatın.

1. Java programlama dilinde değişken tanımlayabilme
2. Değişken isimlendirme kurallarına uyma
3. Temel veri tiplerini (ilkel tipler, karakter dizgileri, diziler) kullanabilme

Değişkenler

Java programlama dilinde sekiz ilkel veri türü vardır: byte, short, int, long, float, double, boolean, ve char. Bunlardan byte, short, int ve long tam sayıları; float ve double ondalıklı sayıları saklamak içindir. String sınıfı, karakter dizelerini temsil eder. Derleyici, yukarıdaki türdeki değişkenler için makul bir varsayılan değer atar. Ancak yerel değişkenler için varsayılan değer atanmaz. Sabitler, kaynak kod içinde sabit bir değerin gösterimleridir. Diziler, tek bir türde ama çok sayıda değerleri muhafaza ederler. Bir dizi ancak uzunluğu kadar eleman tutabilir ve bu uzunluk dizinin ilk yaratıldığında belirlenir, sonradan değişmez. Değişkenlerinizi adlandırırken takip edilmesi gereken sözleşmeler veya uyulması zorunlu kurallar vardır.

4. Java programlama dilindeki operatörleri kullanabilme

Operatörler

Java programlama dilinde atama yapma, aritmetik işlemler, karşılaştırma yapma ve eşitlik kontrolü gibi işlemleri gerçekleştirmek için operatörler tanımlanmıştır. Bu operatörlerin kendi içinde bir önceliği sırası vardır. Örneğin, çarpma ve bölmenin toplama ve çıkarmaya işleminden daha önce gerçekleştirilir. İşlemlerin önceliğini ayarlamak için temel matematikte olduğu gibi parantezler kullanabilirsiniz.

5. Kontrol akış deyimlerini kullanarak döngüler ve dallanmalar yaratabilme

Kontrol Akış Deyimleri

If-then ifadesi tüm kontrol akış deyimlerinin en temelidir. Belirli bir sınama doğru olarak değerlendirildiğinde, programımızın belirli bir kod bölümünü çalıştırmasını sağlar. If-then-else deyimi, bir if sınaması false olarak değerlendirildiğinde, ikincil bir yürütme yolu sağlar. If-then ve if-then-else'den farklı olarak switch ifadesi, istediğimiz sayıda yürütme yollarına izin verir. While ve do-while ifadeleri belirli bir koşul geçerliyken sürekli bir kod bloğunu çalıştırır. Do-while ve while arasındaki tek fark, do-while deyimi durma kriterini en üst yerine en sonda değerlendirildiğidir. Bu nedenle, do-while döngüleri en az bir kere çalıştırılır. For ifadesi, bir dizi değer üzerinde gezinmenin kolay bir yolunu sağlar. Biri koleksiyonların ve dizilerin tüm elemanları üzerinde döngü oluşturmak için tasarlanmış olmak üzere iki çeşidi vardır.

neler öğrendik?

1 Aşağıdakilerden hangisi Java programlama dilinde bir değişkenin ismi **olamaz?**

A. 7den70e
B. LINE_SEPARATOR
C. _LINE_SEPARATOR
D. lineSeperator
E. $lineSeperator

2 Aşağıdakilerin hangisinde tam sayı değerlerini tutmak için kullanılan ilkel veri tiplerinin alabileceği değer aralıkları sırasıyla birlikte ve doğru olarak sıralanmıştır?

A. long > byte > int > short
B. long > short > int > byte
C. long > short > byte > int
D. long > int > short > byte
E. long > int > byte > short

3 Bir elektronun kütlesi yaklaşık olarak 9.1093836 x 10^{-31} kilogramdır. Bu değerin **float** veri tipinde ifadesi aşağıdakilerden hangisidir?

A. 9.1093836e-31
B. 9.1093836e31
C. 9.1093836e-31f
D. 9.1093836e31f
E. 9.1093836e-31d

```
int [] sayılar = {5, -9, 8, 1, 0};

int sonuç = (sayılar [1] + sayılar [2]
( * (sayılar [3] - sayılar[4]);

System.out.println(sonuç);
```

4 Yukarıdaki Java program parçacığı çalıştırıldıktan sonra meydana gelen ekran çıktısı aşağıdakilerden hangisidir?

A. -28
B. -1
C. 0
D. 1
E. 28

5 Java programlama dilinde, x isimli değişkende tutulan bir tam sayının tek sayı olup olmadığını doğru bir şekilde kontrol eden ifade aşağıdakilerden hangisidir?

A. x / 2 != 0
B. x % 2 = 0
C. x % 2 = 1
D. x % 2 == 0
E. x % 2 == 1

6 x>=5 sınama ifadesinin eşleniği aşağıdakilerden hangisidir?

A. (x = 5) || (x > 5)
B. (x == 5) || (x > 5)
C. (x = 5) | (x > 5)
D. (x == 5) && (x > 5)
E. (x = 5) & (x > 5)

```
int [] sayılar = {5, -9, 8, 1, 0, -3};

int toplam = 0;

for (int i : sayılar) {
    toplam += i;
}

System.out.println(toplam);
```

7 Yukarıdaki Java program parçacığı çalıştırıldıktan sonra meydana gelen ekran çıktısı aşağıdakilerden hangisidir?

A. -21
B. -15
C. 2
D. 15
E. 21

8 Birden fazla çalıştırma/yürütme yollarına izin veren kontrol akış deyimi aşağıdakilerden hangisidir?

A. switch
B. do-while
C. for
D. return
E. continue

```
int [] sayılar = {5, -9, 8, 1, 0, -3};

int toplam = 0;

for (int i : sayılar) {
    if (i < 0) continue;
    toplam += i;
}

System.out.println(toplam);
```

9 Yukarıdaki Java program parçacığı çalıştırıldıktan sonra meydana gelen ekran çıktısı aşağıdakilerden hangisidir?

A. -15
B. -12
C. 2
D. 14
E. 15

10 Çalışmakta olan metotdan çıkan kontrol akış deyimi aşağıdakilerden hangisidir?

A. case
B. break
C. continue
D. else
E. return

neler öğrendik yanıt anahtarı

1. A	Yanıtınız yanlış ise "Değişkenlerin Adlandırılması" konusunu yeniden gözden geçiriniz.
2. D	Yanıtınız yanlış ise "İlkel Veri Tipleri" konusunu yeniden gözden geçiriniz.
3. C	Yanıtınız yanlış ise "Gezer Noktalı Sabitler" konusunu yeniden gözden geçiriniz.
4. B	Yanıtınız yanlış ise "Diziler" konusunu yeniden gözden geçiriniz.
5. E	Yanıtınız yanlış ise "Operatörler" konusunu yeniden gözden geçiriniz.
6. B	Yanıtınız yanlış ise "Eşitlik ve İlişki Operatörleri" konusunu yeniden gözden geçiriniz.
7. C	Yanıtınız yanlış ise "or Döngüsü" konusunu yeniden gözden geçiriniz.
8. A	Yanıtınız yanlış ise "Karar Verme Deyimleri" konusunu yeniden gözden geçiriniz.
9. D	Yanıtınız yanlış ise "Dallanma Deyimleri" konusunu yeniden gözden geçiriniz.
10. E	Yanıtınız yanlış ise "Dallanma Deyimleri" konusunu yeniden gözden geçiriniz.

3 Araştır Yanıt Anahtarı

Araştır 1

int[] dizi = new int[6]; dizi[6] = 19; kodunu çalıştırdığımızda ArrayIndexOutOfBoundsException hatası alırız. Çünkü burada 6 uzunluğunda bir dizi yaratıp 6. elemana değer atamaya çalışıyoruz. Ancak indis değerleri 0 dan başladığı için, bu dizide en fazla 5. eleman için bellekte yer açılmıştır. Dizinin olmayan bir elemanına erişim yapmaya çalışırsak bu hata ile karşılaşırız. Bu hata sık yapılan bir hata olduğu için alanyazına İngilizce dilinde **fencepost error** olarak geçmiştir. Bu tabir Türkçemize **çit problemi** olarak tercüme edilebilir.

Araştır 2

Java'nın ilkel tamsayı türleri, değerlerini sonlu sayıdaki bitlerde depolarlar (int için 32 bit gibi). Toplama veya çıkarma gibi bir aritmetik işlem bu bitlere sığamayan bir sonuç ürettiğinde, işlemin taşması (azami pozitif değeri aşması) veya aşağı taşması (asgari negatif değeri aşması) durumu ortaya çıkar. Eğer taşma meydana gelirse, değişkenin değeri asgari değere geri döner ve ortadan devam eder. Eğer aşağı taşma meydana gelirse, değişkenin değeri azami değerine döner ve oradan devem eder. Sonuç olarak yeterince büyük iki tamsayıyı toplarsak sonuçta negatif bir değer elde ederiz.

Java'nın tamsayı aritmetik operatörleri taşmaları veya aşağı taşmaları algılamaz. Bir uygulama bu koşulları saptamak üzere tasarlanmadıkça sorunlar ortaya çıkar. Taşma olup olmadığını kontrol etmek meşakkatli bir iştir.

Taşma olduğunda hata fırlatan subtractExact, multiplyExact, addExact v.b. yardımcı metotlar Java 8 sürümünde eklenmiştir. Bu metotlar java.lang.Math sınıfı altında toplanmış olup argümanları üzerinde aritmetik işlemler yaparlar. Örneğin toplama işlemi için Math.addExact(int x, int y) metotu eklenmiştir ve bu metot iki tamsayıyı toplar. Ancak x+y işleminden farklı olarak söz konusu metot eğer taşma meydana gelirse ArithmeticException fırlatır. Böylece uygulama içinde taşma olup olmadığını sınamak mümkün olur.

3 Araştır Yanıt Anahtarı

Araştır 3

Bir dizinin elemanlarının toplamını döngü kullanmadan hesaplamak mümkündür. Bu işlem aşağıdaki örnekte olduğu gibi kendi içinde kendini çağıran bir metot ile yapılabilir. Bu tür metodlara özyinelemeli (**resursive**) metotlar denir. Döngülerle çözülebilen problemler çoğu kez özyinelemeli metotlarla da çözülebilir. Görüldüğü üzere aşağıdaki program hiçbir döngü deyimi içermemektedir ve çıktı olarak doğru cevap olan 18 toplamını bulur.

```
private static int dizininToplamı(int[] dizi, int n) {

 if (n == 0)
 return dizi[n];
 else
 return dizi[n] + dizininToplamı(dizi, n - 1);
}

public static void main(String[] args) {

 int[] sayılar = {5, -9, 8, -1, 18, 0, -3};

 System.out.println(dizininToplamı(sayılar,
sayılar.length - 1));
}
```

Kaynakça

Gallardo, R. and Hommel, S. and Kannan, S. and Gordon, J. and Zakhour, S. (2016)
The Java Tutorial: A Short Course on the Basics (6th Edition). Addison-Wesley Professional.

İnternet Kaynakları

Oracle Java Documentation, https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html
Son Erişim: Ekim 2017.

Bölüm 4

Java'da Sınıf Tanımlama

Öğrenme Çıktıları

Sınıflar

1 Java'da sınıf tanımlayabilme
2 Üye değişken ve metot tanımlayabilme

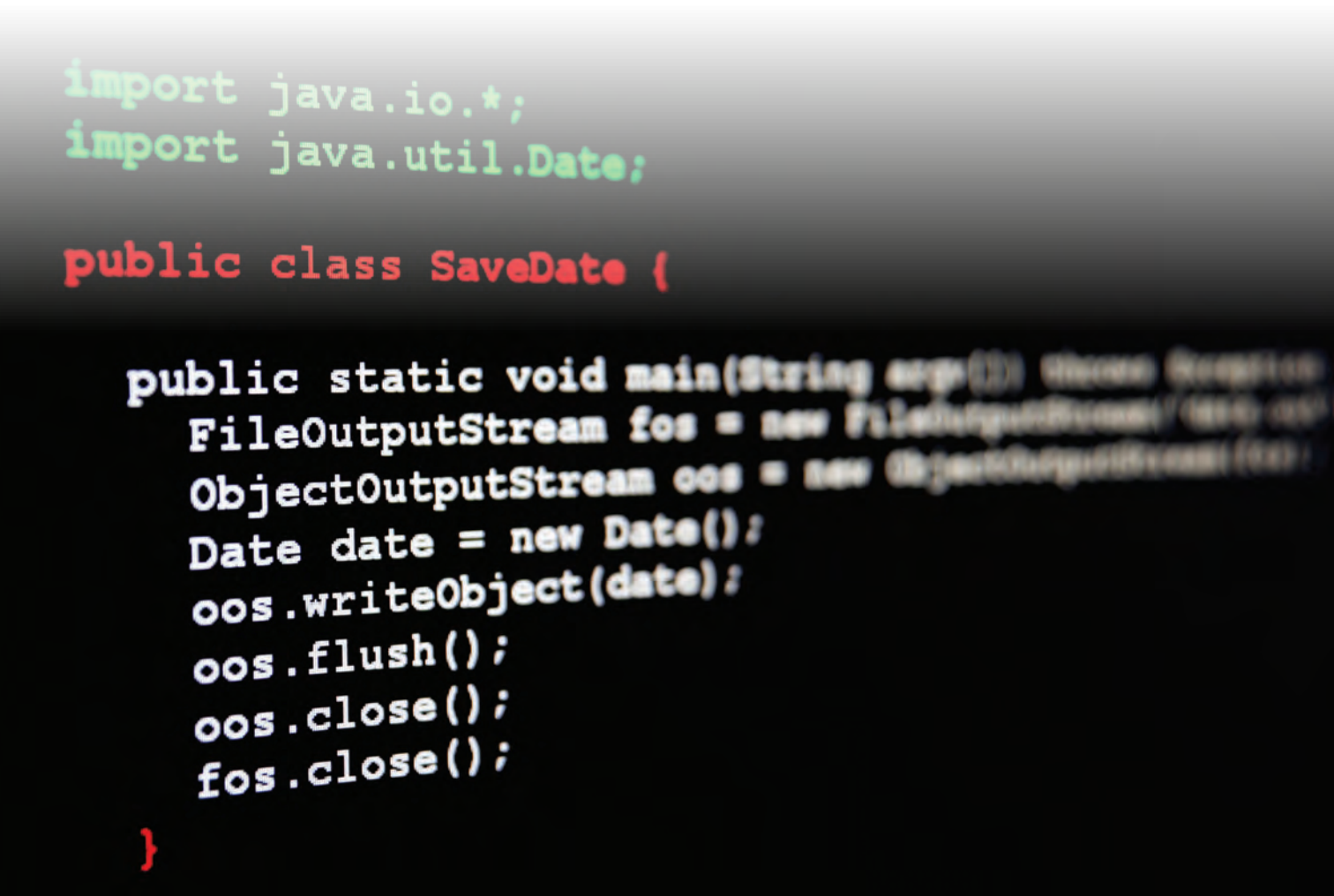
Nesneler

3 Nesneleri yaratma ve kullanma

İç-İçe Sınıflar

4 Java'da sınıf içinde sınıf tanımlayabilme
5 Enum tipinde değişken tanımlayabilme

Anahtar Sözcükler: • Sınıf • Nesne • Üye Değişken • Yaratıcı Metot • Anonim Sınıf • Yerel Sınıf • Enum Tipi • Çöp Toplayıcı

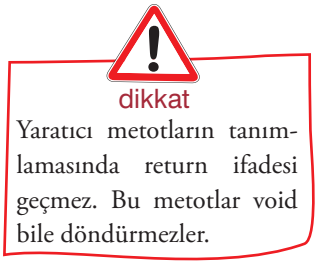


GİRİŞ

Bir önceki ünitede Java programlama temellerini öğrendiğimize göre artık kendi sınıflarımızı tanımlayabiliriz. Bu ünitede üye değişkenler, üye metotlar ve yaratıcı metotlar (constructors) tanımlayarak kendi sınıflarımızı nasıl oluşturabileceğimizi ve bu sınıflardan nesneler yaratıp bu nesneleri kullanmasını öğreneceğiz. Görevini tamamlayan nesnelerin otomatik olarak çöp toplayıcısı tarafından nasıl temizlendiğini kavrayacağız. Tüm nesneler arasında ortak paylaşılan sınıf üyeleri tanıdıktan sonra, sınıfların iç içe tanımlanmasını, başka bir ifadeyle, bir sınıfı başka bir sınıf içinde tanımlamayı göreceğiz. En son olarak kategorik değişkenleri tutmaya yarayan enumation tipi üzerinde duracağız.

SINIFLAR

Java programlama dilinde her unsurun nesnelerden oluştuğunu belirtmiştik. Bu nesnelerin ayrıntılı planlarına ise sınıf denir. Örneğin, bir akıllı telefonun ayrıntılı tasarımını sınıfa benzetirsek, bu tasarımdan üretilen ve satışa sunulan çeşitli renklerde ve özelliklerdeki akıllı telefonlarda bu sınıfa ait nesnelerdir. Bu nedenle tasarısı aynı olmasına rağmen bir akıllı telefon mavi renkte iken diğer biri pembe renkte olabilir.



Java programlama dilinde bir sınıf basit anlamda iki unsurdan oluşur. İlk unsur sınıfın durumunu tarif eden alanlardır (üye değişkenler). İkincisi ise değişkenler üzerinde işlemler yapan metotlardır. Bunlara ek olarak, sınıfı yaratırken kullandığımız, özel bir metot olan yaratıcı metot vardır. Bu metodun ismi sınıfın ismi ile birebir aynıdır ve değer döndürmez.

Java'da sınıf tanımlaması şu şekilde yapılır:

```
public class Sınıfİsmi {

  // alanlar (fields)
  // yaratıcı metot (contructor)
  // metotlar (methods)
}
```

Yukarıdaki örnekte, kaşlı ayraçlar arasında bulunan bölüm sınıf gövdesidir. Sınıf gövdesi bu sınıftan yaratılmış bütün nesnelerin hayat döngüsünü sağlayan tüm kodları içerir. Alanlar (üye değişkenler) sınıfın durumunu belirtir. Yaratıcı metot nesneleri ilk nesneyi yaratırken çalışır. Metotlar ise sınıfın ve o sınıftan yaratılmış nesnelerin davranışını modeller.

Bir sınıf başka bir sınıfı genişletebilir (extends) ve/veya başka bir arayüzü uygulayabilir (implements). Alt-sınıf üst-sınıf ilişkisini anlatan kalıtım (inheritance) konusu Ünite 5'te, arayüzler (interfaces) konusu ise Ünite 6'da ayrıntılı olarak anlatılacaktır. Şimdilik bu kavramların ayrıntısını göz önünde bulundurmadan, sadece sınıf tanımlama söz dizimi açısından incelemeye devam edelim.

```
public final class Long extends Number implements Comparable<Long>
{
}
```

Yukardaki örnekte, java.lang.Long sınıfının nasıl tanımlandığı gösterilmiştir. Burada Long sınıfı Number sınıfının bir alt sınıfıdır. Ayrıca Long sınıfı Comparable arayüzünü sağlamaktadır. Aynı zamanda, bir sınıfın birden fazla arayüzü sağlayabileceğini hatırlatmak gerekir. Örneğin, bir önceki ünitede kullandığımız java.lang.String sınıfının tanımlamasına bakarsak tam 3 tane arayüzü sağladığını görebiliriz. Bu durumda, aşağıdaki örnekte görüleceği üzere, implements anahtar sözcüğünden sonra arayüz isimleri virgül ile ayrılarak yazılır.

```
public final class String
    implements java.io.Serializable, Comparable<String>,
CharSequence {
}
```

Bu iki örneğin ışığında, Java'da sınıf tanımlarken kullanılan bileşenleri sırasıyla aşağıdaki gibi listeleyebiliriz:

- Erişim niteleyiciler (**public**, **private** veya **protected**)
- Erişim dışındaki diğer niteleyiciler (**final**, **static**, veya **abstract**)
- **class** anahtar sözcüğünden sonra sınıfın adı
- Eğer varsa **extends** anahtar sözcüğünden sonra yazılmak üzere, üst sınıf
- Eğer varsa **implements** anahtar sözcüğünden sonra yazılmak üzere, sağladığı arayüz ya da arayüzlerin isimleri
- Son olarak kaşlı ayraçlar { } içine alınmış sınıf gövdesi

Tablo 4.1'de listelen niteleyiciler hem sınıf, hem üye değişken hem de üye metot tanımlanırken kullanılan anahtar sözcüklerdir. Bu sınıf, metot ya da değişkenlere erişim kısıtlaması getirmek için kullanılır. Tablo 4.1 aşağıdan yukarıya olmak üzere, en serbestten en kısıtlayıcı olan niteleyiciye göre sıralanmıştır. Bu tablodaki erişim niteleyiciler dışında niteleyicilerde vardır ama bunlara erişim dışında kalan niteleyiciler denir.

Tablo 4.1 Erişim Niteleyiciler ve Erişim Seviyeleri

Niteleyici	Sınıf	Paket	Alt Sınıf	Dış Dünya
public	✓	✓	✓	✓
protected	✓	✓	✓	X
---	✓	✓	X	X
private	✓	X	X	X

dikkat

Java programlama dilinde bir sınıf birden fazla arayüzü implement edebilir ama sadece bir tek sınıfı extend edebilir. Birden fazla sınıfı extend etmek çoklu-kalıtım Java programlama dilinde desteklenmemektedir!

java.lang.Object sınıf hiyerarşisinin en tepesinde olan sınıftır. Dizilerde dahil olmak üzere, Java programlama dilindeki tüm sınıflar bu sınıfın alt sınıfıdır. Bu kural sınıfların tanımında açıkça "… extends java.lang.Object" denilmesine gerek olmadan geçerlidir.

Üye Değişkenler

Genel olarak değişkenleri üçe ayırabiliriz:

1. Üye değişkenler: Sınıfların içindeki alanlar
2. Yerel değişkenler: Bir metot içindeki veya kaşlı ayraçlar içinde yazılmış kod bloğunun içindeki değişkenler
3. Parametreler: metot tanımlarken kullandığımız metot isminden sonra gelen ayraçlar içinde tanımlanan değişkenler

Dikdörtgen sınıfının aşağıdaki üye değişkenleri nasıl tanımladığına bakalım:

```
public class Dikdörtgen {

    private final int en;
    private final int boy;

    public Dikdörtgen() {
        this.en = 1;
        this.boy = 1;
    }

    public Dikdörtgen(int en, int boy) {
        this.en = en;
        this.boy = boy;
    }
}
```

Üye değişken tanımlarken kullanılan bileşenleri sırasıyla listeleyecek olursak:

- Niteleyiciler (bu örneğimizde private ve final anahtar sözcükleri)
- Alanın tipi (bu örneğimizde int, ama String, dizi ya da başka bir sınıfta olabilirdi)
- Alanın ismi (bu örneğimizde en ve boy)

> Üye değişken tanımlamalarında kullanılan volatile ve transient gibi daha başka niteleyicilerde vardır ama bu niteleyiciler bu kitabın kapsamı dışındadır.

Metot Tanımlama

Şimdi dikdörtgen sınıfımızda, dikdörtgenin çevresini ve alanını hesaplayan iki tane metot tanımlayalım.

```
public class Dikdörtgen {

    private final int en;
    private final int boy;

    public Dikdörtgen() {
        this.en = 1;
        this.boy = 1;
    }

    public Dikdörtgen(int en, int boy) {
        this.en = en;
        this.boy = boy;
    }

    public int hesaplaAlan() {
        return en * boy;
    }

    public int hesaplaÇevre() {
        return 2 * (en + boy);
    }
}
```

Metot tanımlarken kullanılan bileşenleri sırasıyla listeleyecek olursak:

- Niteleyiciler (bu örneğimizde **public** anahtar sözcüğü)
- Döndürdüğü değerin tipi (bu örneğimizde **int** ama **void**, dizi ya da başka bir nesne de olabilirdi.)
- Metot ismi (bu örneğimizde hesaplaAlan ve hesaplaÇevre). Genel olarak metot isimleri küçük harf ile yazılan bir fiil ile başlar. Eğer ikinci bir kelimesi varsa bu ikinci kelime büyük harf ile başlayan sıfat veya isim olur.
- Parametre listesi, parantezler içine yazılır.
- İstisna listesi, **throws** anahtar sözcüğünden sonra yazılır.

Metot İmzası
Bir metodun ismi ve parametre listesinin birleşimine o metodun imzası denir.

dikkat
İsmi aynı olan metotlar tanımlayabiliriz ama bunların imzası farklı olmalıdır. Derleyici imzası aynı olan iki metodu aynı anda tanımlamaya izin vermez.

Aşağıda java.nio.file.Files sınıfına ait iki metot tanımı verilmiştir. Görüldüğü üzere, iki metodun da ismi aynıdır: readAllLines. Fakat dikkat edileceği üzere metotların parametre listeleri farklıdır.

```
public static List<String> readAllLines(Path path, Charset cs)
throws IOException { ... }

public static List<String> readAllLines(Path path) throws
IOException { ... }
```

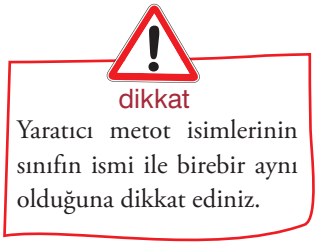
Bu tür aynı isimli metotların kullanılmasına metotların yüklenmesi (overloading methods) adı verilir.

Yaratıcı Metotlar

Yaratıcı metotlar, standart metotlar gibi tanımlanırlar. Fakat bu tür metotlar değer döndürmezler. Aşağıdaki dikdörtgen örneğinde iki tane yaratıcı metot gösterilmiştir. Bu metotlardan birisi argümansızken diğerinin parametre listesi vardır. Argümansız olan metotta, nesneyi yaratırken hiçbir bilgi metoda iletilmemektedir. İkinci durumda ise nesneyi yaratırken dikdörtgenin en ve boy değerleri yaratıcı metota bilgi olarak aktarılmaktadır.

```
public class Dikdörtgen {

    private final int en;
    private final int boy;

    // argümanı olmayan yaratıcı metot
    public Dikdörtgen() {
        this.en = 1;
        this.boy = 1;
    }

    // argüman listesi olan yaratıcı metot
    public Dikdörtgen(int en, int boy) {
        this.en = en;
        this.boy = boy;
    }
 }
```

Her iki yaratıcı metodu kullanan bir fonksiyon aşağıdaki gibi yazılabilir. Görüldüğü gibi, yaratıcı metotları çağırırken **new** anahtar sözcüğünü kullanıyoruz.




```
public static void main(String[] args) {

   Dikdörtgen dikdörtgen1 = new Dikdörtgen();
   Dikdörtgen dikdörtgen2 = new Dikdörtgen(8, 9);

 }
```

Şimdi ise dikdörtgen sınıfımıza, iki dikdörtgeni karşılaştıran bir metot tanımlayalım. Bu karşılaştırma, büyüklük olarak dikdörtgenlerin alanlarını kullansın.

```
public int karşılaştırAlanKullanarak(Dikdörtgen diğer) {
    return this.hesaplaAlan() - diğer.hesaplaAlan();
 }
```

Bu örnek ile hem this anahtar sözcüğünü tanıtılmakta hem de argümanı olan bir metot tanımlanmaktadır. Burada karşılaştırılan iki dikdörtgen nesnesi vardır. Birisi this anahtar sözcüğü ile ifade edilendir. Diğeri ise metodun argümanı olan diğer isimli değişkendir. Bir başka deyişle kendi dikdörtgenimizi dışarıdan girdi olarak gelen diğer bir dikdörtgen ile karşılaştırıyoruz. Karşılaştırma neticesinde hangi dikdörtgenin alanın büyük olduğuna göre bir tam sayı değer geri döndürüyoruz. Eğer kendi dikdörtgenimizin alanı büyük ise pozitif bir tam sayı, diğer dikdörtgenin alanı büyük ise negatif bir tam sayı döndürmekteyiz. Alanların eşit olması durumunda ise karşılaştırAlanKullanarak isimli metodumuz sıfır geri döndürmektedir.

Bir liste ya da dizi içinde tutulan çok sayıda dikdörtgen nesnelerini alanlarının büyüklerine göre sıralamak istersek bu tek satırlık metodu sıralamayı gerçekleştirmek için kullanabiliriz.

Bu sıralama işini yapmak için öncelikle birkaç tane çeşitli en ve boy değerlerine sahip dikdörtgenler yaratalım. Ve öyle bir metot yazalım ki keyfi sayıda verilen dikdörtgenleri sıralayıp ekrana yazdırsın.

```
public class Dikdörtgen {

    private final int en;
    private final int boy;

    // argümanı olmayan yaratıcı metot
    public Dikdörtgen() {
        this.en = 1;
        this.boy = 1;
    }

    // argüman listesi olan yaratıcı metot
    public Dikdörtgen(int en, int boy) {
        this.en = en;
        this.boy = boy;
    }

    public int hesaplaAlan() {
        return en * boy;
    }

    public int hesaplaÇevre() {
        return 2 * (en + boy);
    }
```

```
    public int karşılaştırAlanKullanarak(Dikdörtgen diğer) {
        return this.hesaplaAlan() - diğer.hesaplaAlan();
    }

    @Override
    public String toString() {
        return "en=" + en + " boy=" + boy + " alan=" + hesaplaAlan();
    }

    // keyfi sayıdaki dikdörtgenleri sıralar
    private static void sıralaDikdörtgenleri(Dikdörtgen... dikdörtgenler) {
        Arrays.sort(dikdörtgenler, Dikdörtgen::karşılaştırAlanKullanarak);
        System.out.println(Arrays.toString(dikdörtgenler));
    }

    public static void main(String[] args) {

      // bir kaç dikdörtgen yaratalım ve
      // bunları sıralayıcı metotumuza parametre olarak geçelim
        sıralaDikdörtgenleri(
                new Dikdörtgen(4, 5),
                new Dikdörtgen(5, 4),
                new Dikdörtgen(8, 2),
                new Dikdörtgen(3, 5)
        );
    }
}
```

Eğer bu programı çalıştırırsanız, çıktısının `[en=3 boy=5 alan=15, en=8 boy=2 alan=16, en=4 boy=5 alan=20, en=5 boy=4 alan=20]` olduğunu göreceksiniz. Görüldüğü gibi dikdörtgenler alanlarına göre küçükten büyüğe doğru sıralanmıştır.

Peki bu nasıl oldu? Ayrıntılarına bakalım. Bu örneğimizde dikkat ettiyseniz üzerinde **@Override** olan yeni bir metot ekledik. Bu metot anlaşılacağı üzere adı toString olan özel bir metotdur. Java'da her sınıfın java.lang.Object sınıfının bir alt sınıfı olduğunu belirtmiştik. İşte bu toString metodu da Object sınıfının bir metodudur ve biz burada onu dikdörtgenler için özelleştirip en, boy ve alan bilgisini String olarak geri döndürüyoruz.

Daha sonra ise sıralaDikdörtgenleri(Dikdörtgen... dikdörtgenler) diye metot tanımlıyoruz. Burada üç nokta yan yana gösterimi aslında daha önce gördüğümüz dizilere benzemektedir. Bu değişken biçimi ile metotlara keyfi sayıda, önceden belli olmayan sayıda, eleman atayabiliyoruz. Bu husus, main metodu içinde, sirala Dikdörtgenleri metoduna new ile yaratılmış ve virgülle ayrılmış dikdörtgenlerin parametre olarak geçilmesiyle gösterilmiştir.

Eğer üst sınıfa ait bir metodu kendimize göre yeniden tanımlıyor isek o metodun üstüne **@Override** işaretini koymak tavsiye edilir. Böyle bu metodun üst sınıfına ait bir metot olduğu ayırt edilmiş olur.

dikkat

Üç nokta yan yana ifadesini sadece metotların parametreleri (argümanları) için kullanılabilir. Dolayısıyla üye değişkenler ve yerel değişkenler için bu gösterimi kullanmak mümkün değildir.

Son olarak sıralama işlemi için tek satırlık kod olan Arrays.sort(dikdörtgenler, Dikdörtgen::karşılaştırAlanKullanarak); kullanılmaktadır. Bu tek satırla programa "Benim elimdeki dikdörtgenleri karşılaştırAlanKullanarak isimi metodu kullanarak sırala" komutu verilmektedir.

Ekrana çıktı olarak yazdırmakta kullandığımız yardımcı metot ise Arrays.toString metodudur. Bu yardımcı metot dizinin elemanlarının tek tek toString metodunu çağırarak her eleman için ayrı bir String elde eder ve daha sonra bunları virgülle ayırıp, başına ve sonuna köşeli ayraçları ekleyerek tek bir String geri döndürür. Bu işlem sonunda ekranda [en=3 boy=5 alan=15, en=8 boy=2 alan=16, en=4 boy=5 alan=20, en=5 boy=4 alan=20] ifadesi görülmektedir.

Burada dikkat etmeniz gereken nokta, karmaşık bir işi nasıl kısa ve anlaşılır kod parçacıklarıyla çözüyor olmamızdır. Bunların bir kısmını biz yazdık, diğer bir kısmı içinse Java programlama dili içinde hazır gelen ifadeleri kullandık. Bu işlem aslında yeniden kullanılmak üzere geliştirilmiş ve hayatı kolaylaştıran komponentleri bir araya getirerek, karmaşık bir işi yapan program yazabilmemize bir örnektir. Ayrıca, bu bahsi geçen komponentler birbirleriyle etkileşim içindedir. Bu örnekte, insan diline yakın olan ve okuyunca bir fikir sahibi olabildiğimiz bir kod geliştirmiş olduk. Hem de bunu kısa ve öz şekilde, uzun uzun satırlarca kod yazmadan başarabildik. İşte nesne yönelimli bir programlama dili olan Java'nın gücü bu özelliğinden kaynaklanmaktadır.

Dikdörtgen::karşılaştırAlanKullanarak ifadesi bir **lambda** ifadesidir ve Java 8'in bir özelliğidir. Parametresi metot olan metotlar yazmak için kullanılır. Bir anlamda değişken olarak bir metot kullanmayı mümkün kılan bu yeni teknoloji bu ünitenin konuların dahilinde değildir.

Artık günümüzde sadece yazan kişinin anlayabildiği karmaşık kodlar değil başka programcılarında okuyunca kavrayabildiği ya da en azından kodun ne yaptığı hakkında fikir sahibi olabildiği anlaşılır kodlar makbuldür.

Öğrenme Çıktısı

1 Java'da sınıf tanımlayabilme
2 Üye değişken ve metot tanımlayabilme

Araştır 1

java.lang.Object sınıfına ait equals ve hashCode metotlarını araştırın. Hangi durumlarda bu metotları kendi tanımladığımız sınıflar için özelleştirip yeniden tanımlamak isteyebiliriz?

İlişkilendir

Matematikte karmaşık sayı, bir gerçel bir de sanal kısımdan oluşan bir nesnedir. Karmaşık sayıları temsil eden bir sınıf yazın. Bu sınıfın equals ve hashCode metotlarını nasıl gerçeklerdiniz?

Anlat/Paylaş

Yazdığınız karmaşık sayı sınıfına; toplama, çıkarma, bölme ve çarpma işlemlerini yapan metotları nasıl ekleyebileceğinizi anlatın.

NESNELER

Tipik bir Java programında birçok nesne yaratılır ve bu nesneler birbirleriyle etkileşim içinde çalışarak birtakım işleri yerine getirirler. Bir nesne kendisine tanımlanan işi tamamladığında ise bu nesnenin tükettiği ve kullandığı kaynak boşa çıkarılır. Böylece boşa çıkan bu kaynaklar başka nesnelerin kullanımına açılmış olur.

Şimdi bir önceki Dikdörtgen sınıfımızdan iki adet nesne yaratıp bu nesneleri kullanalım.

```
public static void main(String[] args) {

    Dikdörtgen dikdörtgen1 = new Dikdörtgen();
    Dikdörtgen dikdörtgen2 = new Dikdörtgen(8, 9);

    System.out.println("Birinci dikdörtgenin alanı = " +
dikdörtgen1.hesaplaAlan());

    System.out.println("İkinci dikdörtgenin çevresi = " +
 dikdörtgen2.hesaplaÇevre());
 }
```

Bu örnekte aslında Dikdörtgen tipinde iki tane değişken tanımlıyoruz. Bu değişkenlere dikdörtgen1 ve dikdörtgen2 isimleri verilmiştir. Sonrasında bu iki değişkeni kullanarak sınıf tanımlamasında geçen metotları çağırıyoruz. Birinci dikdörtgenin alanını, ikincisinin ise çevresini ekrana yazdırıyoruz. Bunu yaparken de değişken isminden sonra nokta işareti (.) koyup çağırmak istediğimiz metodun ismini yazıyoruz. Bu programın çıktısı aşağıdaki şekilde olacaktır:

Birinci dikdörtgenin alanı = 1
İkinci dikdörtgenin çevresi = 34

Bir nesneyi yaratırken her zaman onu bir değişkene atamak zorunda değiliz. Aşağıdaki örnekte olduğu gibi direk olarak kullanabiliriz.

```
System.out.println("İsimsiz dikdörtgenin çevresi = " + new
Dikdörtgen(3, 6).hesaplaÇevre());
```

Programın ekran çıktısı: İsimsiz dikdörtgenin çevresi = 18

Varsayılan Yaratıcı Metot
Her sınıfın en az bir tane yaratıcı metodu vardır. Eğer sınıfın içinde açık olarak hiç yaratıcı metot tanımlanmamış ise, hiç argümanı olmayan bir yaratıcı metot Java derleyicisi tarafından otomatik olarak sağlanır. Buna varsayılan yaratıcı metot denir. Varsayılan yaratıcı metot, üst sınıfın argümansız yaratıcı metodunu çağırır. Bu yüzden üst sınıfın argümansız bir yaratıcı metodu yoksa derleyici bu durumda hata verir. Ancak, açık olarak belirtilmemiş üst sınıf olmaması durumunda, örtük olarak üst sınıf `java.lang.Object` sınıfıdır ve bu sınıfın argümansız yaratıcı metodu vardır.

Çöp Toplayıcısı Java programlama dilinde yarattığınız nesnelerin kayıtlarını tutup daha sonra nesnelerle işimiz bittiğinde onları yok etmekle uğraşmanıza gerek yoktur. Bu işlem çöp toplayıcısı tarafından otomatik olarak yapılır. Eğer bir nesne artık kullanılmıyorsa otomatik olarak silinir. Bu işleme çöp toplama denir. Çöp toplama işlemi periyodik olarak uygun zamanlarda otomatik yapılır ve bellek temizlenir. Java'da birkaç çeşit çöp toplayıcısı vardır ve programcı bunlardan hangisi kullanmak istediğini seçebilir.

this Anahtar Sözcüğü

Üye metotlar ve yaratıcı metotlar için, şu anki nesneye erişim **this** anahtar sözcüğü ile yapılır. En sık olarak üye değişken isimleri ile yaratıcı metodun parametre listesindeki değişkenleri ayırt etmek için kullanılır. Daha önceki Dikdörtgen örneğimizi yeniden inceleyelim.

```
public class Dikdörtgen {

    private final int en;
    private final int boy;

    // argümanı olmayan yaratıcı metot
    public Dikdörtgen() {
        this.en = 1;
        this.boy = 1;
    }

    // argüman listesi olan yaratıcı metot
    public Dikdörtgen(int en, int boy) {
        this.en = en;
        this.boy = boy;
    }
}
```

Burada **this**.en üye değişkeni**, int en** ise yaratıcı metodun bir parametresidir. Böylece aynı isimlere sahip olmalarına rağmen this anahtar sözcüğü ile üye değişkene diğer değişkeni atayabiliyoruz. Bir başka kullanım ise **this** anahtar sözcüğünün bir yaratıcı metodu çağırmak için kullanılabilmesidir. Örneğin, yukardaki kodda ilk yapıcı metodu **this** kullanarak yeniden yazabiliriz.

```
public class Dikdörtgen {

    private final int en;
    private final int boy;


    // argümanı olmayan yaratıcı metot
    public Dikdörtgen() {
        this(1, 1);
    }

    // argüman listesi olan yaratıcı metot
    public Dikdörtgen(int en, int boy) {
        this.en = en;
        this.boy = boy;
    }
}
```

Böylece argümanı olmayan yaratıcı metot, diğer yaratıcı metodu çağırmış olur. Eğer böyle bir kullanım yapılacak olursa, **this**(1,1) ifadesi ilk satırda olmak zorundadır.

super anahtar sözcüğü this anahtar sözcüğü ile aynı mantıkta çalışır fakat bu sefer üst sınıfın üyelerine erişim için kullanılır.

Sınıf Üyeler

Bir sınıftan yaratılmış tüm nesneler için ortak olmasını istediğimiz değişken ya da metotları tanımlarken önüne static anahtar sözcüğünü koyarız. Böylece bunlar sınıf üyeleri olmuş olur.

Şu ana kadar hep static olmayan üye değişkenleri ve üye metotları kullandığımızı hatırlayalım. Bunlara ise örnek (instance) üye denir. Her yeni yaratılan nesne için örnek değişkenler farklı olabilmektedir. Eni boyu farklı olan dikdörtgen nesneleri yarattık. Bazı durumlarda bütün nesneler için ortak bir değişken tanımlamak isteyebiliriz. Bu tür değişkenlere sabit alanlar veya sınıf değişkenleri denir. Bu tür değişkenlere erişmek için artık **new** ile yeni nesne yaratmamıza gerek yoktur. Bunlara sınıfın ismi ile erişilebilir. Bu tür değişkenler genellikle sabit değerleri saklamak için kullanılır. Örneğin java.lang.Math sınıfı içinde **e** sayısını tutan değişken sınıf değişkenidir.

```
public final class Math {

    /**
     * The {@code double} value that is closer than any other to
     * <i>e</i>, the base of the natural logarithms.
     */
    public static final double E = 2.7182818284590452354;
 }
```

Bu değişkene erişmek için new `Math()` dememize gerek yoktur. Direk olarak `Math.E` ifadesi ile erişebiliriz. Örneğin, aşağıdaki kodu çalıştırırsak ekrana 2.718281828459045 yazdırılır.

```
public static void main(String[] args) {
    System.out.println(Math.E);
}
```

Nasıl static anahtar sözcüğü ile değişken tanımlayabiliyorsak, aynı şekilde metot tanımlayabiliriz. java.lang.Math sınıfı içindeki mutlak değer fonksiyonu buna bir örnektir. Değişkenlerde olduğu gibi, bu metotlara erişmek için new anahtar sözcüğüne gerek yoktur. Aşağıdaki örnek kod, -5 değerinin mutlak değerini hesaplar ve ekrana 5 yazdırır.

```
public static void main(String[] args) {
    System.out.println(Math.abs(-5));
 }
```

Bu tür metotlar genelde sadece parametre listesindeki değişkenler üzerinde anlamlı işlemler yapan yardımcı metotlardır. java.lang.Math sınıfı içinde minimumu bulan, maksimumu bulan, ondalıklı sayıları yuvarlayan, gibi bir çok static metot örneği bulunabilir.

Eğer static olarak tanımlanmış değişkenlere ilk değer atamak istersek ve bu tek satırdan daha uzun sürecek bir işlemse bunu static k od bloğu içinde yaparız. Örneğin, Anadolu Üniversitesinin harf notlarının katsayı değerlerini içeren static bir dizimiz olsun. Bu dizinin ilk değerlerini aşağıdaki örnekteki atayabiliriz.

```
public class HarfNotları {

    static double[] katsayılar;

    static {

       katsayılar = new double[10];
        katsayılar [0] = 4.0; // AA
        katsayılar [1] = 3.7; // AB
        katsayılar [2] = 3.3; // BA
        katsayılar [3] = 3.0; // BB
        katsayılar [4] = 2.7; // BC
        katsayılar [5] = 2.3; // CB
        katsayılar [6] = 2.0; // CC

        katsayılar [7] = 1.7; // CD
        katsayılar [8] = 1.3; // DC
        katsayılar [9] = 1.0; // DD
        katsayılar [10] = 0.0; // FF
    }
 }
```

Aynı işlemi static olmayan değişkenler içinde yapabiliriz. Tek yapmamız gereken static anahtar sözcüğünü kaldırmak olacaktır.

```
public class HarfNotları {

    double[] katsayılar;

    {

       katsayılar = new double[10];
        katsayılar [0] = 4.0; // AA
        katsayılar [1] = 3.7; // AB
        katsayılar [2] = 3.3; // BA
        katsayılar [3] = 3.0; // BB
        katsayılar [4] = 2.7; // BC
        katsayılar [5] = 2.3; // CB
        katsayılar [6] = 2.0; // CC

        katsayılar [7] = 1.7; // CD
        katsayılar [8] = 1.3; // DC
        katsayılar [9] = 1.0; // DD
        katsayılar [10] = 0.0; // FF
    }
 }
```

Yukarıdaki örnekten de görüleceği üzere, genelde ilk değer atama işlemlerini yaratıcı metot içinde yapılır. Ama kaşlı ayraçlar içine alınmış kod bloğu içinde de yapılabileceğini göstermek için bu örnek kullanılmıştır. Böyle bir kod bloğu, Java derleyicisi tarafından bütün yaratıcı metotların içine kopyalanır.

Sınıf tanımlaması içinde bu şekilde birden fazla bloklarımız olabilir. Derleyici bu blokları yazılma sırasına göre çalıştırır. Örneğin, yukarıdaki bloğu iki bloğa bölebiliriz.

```
public class HarfNotları {

    double[] katsayılar;

    {

        katsayılar = new double[10];
        katsayılar [0] = 4.0; // AA
        katsayılar [1] = 3.7; // AB
        katsayılar [2] = 3.3; // BA
        katsayılar [3] = 3.0; // BB
        katsayılar [4] = 2.7; // BC
        katsayılar [5] = 2.3; // CB
        katsayılar [6] = 2.0; // CC
    }
    {
        katsayılar [7] = 1.7; // CD
        katsayılar [8] = 1.3; // DC
        katsayılar [9] = 1.0; // DD
        katsayılar [10] = 0.0; // FF
    }
}
```

Öğrenme Çıktısı

3 Nesneleri yaratma ve kullanma

Araştır 2	İlişkilendir	Anlat/Paylaş
Geri döndürdüğü değer this olan (return this;) üye metot kullanımına dair bir senaryo düşünün.	Builder pattern kavramını araştırıp, geri döndürdüğü değer this olan (return this;) üye metotlarla ilişkilendirin.	Geri döndürdüğü değer this olan (return this;) üye metot kullanımın ne gibi kolaylıklar sağlayabileceğini anlatın.

İÇ-İÇE SINIFLAR

Java programlama dilinde bir sınıf tanımı içerisinde başka bir sınıfı tanımlamak mümkündür. Bu şekilde tanımlanan sınıfa iç sınıf denir.

```
/**
 * İç-içe Sınıflara Örnek
 */
public class DışSınıf {

    class İçSınıf {
    }
}
```

dikkat

İç sınıf static olarak da tanımlanabilir.

İç sınıf dış sınıfın bir üyesidir. Bir iç sınıf, dış sınıfın diğer bütün üyelerine erişebilir. Bu tür iç sınıf tanımlamasını birbirleriyle alakalı sınıfları bir araya toplamak için kullanabiliriz. Böylece geliştirilen kodun okunabilirliği ve bakımı kolaylaşır.

Yerel Sınıflar

Kaşlı ayraçlar içine alınmış kod bloğunda da sınıf tanımlaması yapılabilir. Yerel sınıflar, iç sınıflara benzerler. İç sınıflarda olduğu gibi, yerel sınıflarda tanımlandıkları dış sınıfın bütün üyelerine erişebilirler. Aşağıdaki örnekte iki tane yerel sınıf tanımlaması verilmiştir.

```
public class Sınıf {
   private int a;
   {
       class YerelSınıf2 {
           public YerelSınıf2() {
               System.out.println(a);
           }
       }
       YerelSınıf2 yerelSınıf2 = new YerelSınıf2();
   }
   public void üyeMetot(int parametre1) {
       class YerelSınıf {
           public YerelSınıf() {
               System.out.println(a);
           }
       }
       YerelSınıf yerelSınıf = new YerelSınıf();
   }
}
```

Anonim Sınıflar

Ancak bir arayüzü sağlayan ya da bir üst sınıfı `extend` ifade ise genişleten bir alt sınıf anonim olarak tanımlanabilir. Bu durumda sınıfın tanımlaması ve yaratılması aynı anda gerçekleşir.

Daha önceki örneğimizde dikdörtgenleri alanlarına göre sıralamıştık. Şimdi çevrelerini kullanarak aynı işlemi yapalım. Kendi tanımladığımız nesneleri küçükten büyüğe sıralayabilmemiz için, önce bu sıralama işlemini nasıl yapacağımızı belirtmemiz gerekir. Aksi hâlde hangi dikdörtgen büyük hangisi küçük olduğunu program akışı belirleyemez. Bu iş için kullanılan arayüz java.util.Comparator arayüzüdür. Aşağıda, çevre uzunluklarının değerlerine göre dikdörtgenleri karşılaştıran sınıf tanımlaması verilmiştir.

```
/**
 * Dikdörtgen nesnelerini çevresine göre karşılaştıran sınıf
 */
public class ÇevreKarşılaştır implements Comparator<Dikdörtgen> {
   @Override
   public int compare(Dikdörtgen o1, Dikdörtgen o2) {
       return o1.hesaplaÇevre() - o2.hesaplaÇevre();
   }
}
```

Aynı sınıfı, bir isim vermeden, aynı anda hem tanımlayıp hem de yaratma işlemi ile anonim sınıf elde edilmiş olur. Aşağıda buna bir örnek verilmiştir.

```
Comparator<Dikdörtgen> karşılaştırıcı = new
Comparator<Dikdörtgen>() {
    @Override
    public int compare(Dikdörtgen o1, Dikdörtgen o2) {
        return o1.hesaplaÇevre() - o2.hesaplaÇevre();
    }
};
```

Şimdi anonim sınıf kullanarak birkaç dikdörtgen nesnesi yaratıp bunları önce bir listeye ekleyelim daha sonra da listemizi sıralayalım.

```
public static void main(String[] args) {

    List<Dikdörtgen> dikdörtgenler = Arrays.asList(
            new Dikdörtgen(4, 5),
            new Dikdörtgen(5, 4),
            new Dikdörtgen(8, 2),
            new Dikdörtgen(3, 5),
            new Dikdörtgen()
    );


    dikdörtgenler.sort(new Comparator<Dikdörtgen>() {
        @Override
        public int compare(Dikdörtgen o1, Dikdörtgen o2) {
            return o1.hesaplaÇevre() - o2.hesaplaÇevre();
        }
    });
 }
```

Görüldüğü gibi sort üye metotuna parametre olarak anonim sınıfımızı vermiş oluyoruz. Bu işlem dikdörtgenler.sort(new ÇevreKarşılaştır()); ifadesi kullanılarakta yapabilir fakat ayrı bir Java dosyasında ayrı bir sınıf tanımlamasından farklı olarak anonim sınıf kullanılınca sıralama işinin hangi kritere göre yapıldığı ilgili sınıfın içinde görülebilir olmaktadır.

Bu programın çıktısı: [en=1 boy=1 çevre=4, en=3 boy=5 çevre=16, en=4 boy=5 çevre=18, en=5 boy=4 çevre=18, en=8 boy=2 çevre=20] olacaktır.

dikkat

Anonim sınıf tanımlaması sadece bir arayüzü implement eden ya da bir sınıfı extend eden sınıflar için mümkündür.

Diğer taraftan, bu çevreye göre sıralama yapmanın daha kısa bir yolu mevcuttur. Bu yolda, hiç sınıf tanımlamadan direk olarak fonksiyonumuzu yazmaya imkân veren lambda ifadesi kullanmaktır.

```
public static void main(String[] args) {

    List<Dikdörtgen> dikdörtgenler = Arrays.asList(
            new Dikdörtgen(4, 5),
            new Dikdörtgen(5, 4),
            new Dikdörtgen(8, 2),
            new Dikdörtgen(3, 5),
            new Dikdörtgen()
    );

    dikdörtgenler.sort(((o1, o2) -> o1.hesaplaÇevre() -
 o2.hesaplaÇevre()));

    System.out.println(dikdörtgenler);
}
```

Enum Tipi

Kategorik değişkenleri saklamak için kullanılırlar. Örneğin, bir insana ait kan grubu bilgisini düşünelim. Bir insanın kan grubu A, B, AB veya 0 olabilir. Bu tür bir veriyi modellemek için **Enum** tanımlaması yapmak uygundur.

Enum
Trafik ışığı renkleri, haftanın günleri gibi önceden belirli sabit değerlerden oluşan verileri temsil etmek için Enum tipi kullanılır.

```
public enum KanGrubu {
    A, B, AB, O
}
```

Kişileri temsil eden sınıfımıza kan grubunuda bir alan olarak ekleyelim.

```
public class Kişi {
   protected final String ad;
   protected final String soyad;

   private final KanGrubu kanGrubu;

   public Kişi(String ad, String soyad, KanGrubu kanGrubu) {
       this.ad = ad;
       this.soyad = soyad;
       this.kanGrubu = kanGrubu;
   }

   public static void main(String[] args) {
       Kişi kişi = new Kişi("ahmet", "arslan", KanGrubu.O);
   }
}
```

Görüldüğü gibi KanGrubu tipi diğer tipler olan int, String gibi kullanıma sahiptir. Sabit değerlere erişmek için ise KanGrubu.*AB* şeklinde mümkündür.

Öğrenme Çıktısı

4 Java'da sınıf içinde sınıf tanımlayabilme
5 Enum tipinde değişken tanımlayabilme

Araştır 3

Enum veri tipi yerine sabit değişkenlerde kullanabilir. Enum tipi kullanmanın ne gibi avantajları olduğunu araştırın.

İlişkilendir

Sırasız niteliksel (nominal) ve sıralı niteliksel (ordinal) istatistiksel veri türlerini araştırıp, Java'nın enum tipi ile ilişkilendirin.

Anlat/Paylaş

Java programlama dili içerisinde hazır gelen enum tiplerinden 5 tanesini bulup anlatın.

Yaşamla İlişkilendir

G1 (Garbage First) Çöp Toplayıcısı

Yürütme zamanında ortaya çıkan bellek ihtiyacını Heap olarak adlandırılan bir bellek alanından new işlecini kullanarak karşılıyoruz. new ile aldığımız alanı işimiz bitince geri vermemiz gerekmez. Bu alanın yönetiminden Çöp Toplayıcı (Garbage Collector) sorumludur. Çöp toplayıcı, temel olarak Heap'de canlı nesneleri saptar ve ölü nesneleri yeniden kullanılabilir bellek alanlarına dönüştürür. Çöp toplayıcı geliştiricinin üzerinden önemli bir yükü alır. Ancak bunun için yürütme zamanında bir bedel ödenir. Java HotSpot Sanal Makinesi içinde çok sayıda çöp toplayıcı yer alır. Bunlardan hangilerinin seçileceği ve en iyilenmesi önemli bir konudur. JDK 7u4 ile birlikte G1 (Garbage First) olarak adlandırılan yeni bir çöp toplayıcısı geldi. Uzun dönemde, G1'in Java platformunun tek çöp toplayıcısı olması hedeflenmektedir. Java Sanal Makinesi'nin en zayıf halkası çöp toplayıcıdır. Çöp toplama sırasında tüm uygulamanın iş parçalarının durdurulduğu, "tüm dünyanın durduğu" (stop-the-world) bir evre vardır. Bu evre boyunca uygulama hiçbir iş yapmaz. Ne yazık ki çöp toplayıcı algoritmaları Heap boyutuna göre ölçeklenebilir değildir. Heap boyutu arttıkça bu süre de hızla artmaktadır. Örneğin 64GB heap boyutu olan bir Java Sanal Makinesi'nde bu süre bir dakikayı aşabilir. G1 bu süre için bir sınır tanımlamamıza olanak sağlıyor olsa da başka yan etkiler gösteriyor. Java 8'de çöp toplama ile ilgili dikkate değer tek gelişme Kalıcı Alan (Permanent Generation) olarak adlandırılan alanın genel Heap alanına eklemlenmesidir. Özellikle Web uygulamalarında sık yapılan güncellemelerden kaynaklanan bu alanının dolmasına nedeni ile aldığımız taşma hatası ile şimdi daha seyrek karşılaşmayı umuyoruz.

Kaynak: http://www.bilprog.ege.edu.tr/~akyazi/index.php/2017/01/08/java-8-ile-gelen-yenilikler/

1 Java'da sınıf tanımlayabilme

2 Üye değişken ve metot tanımlayabilme

Sınıflar

Java'da sınıf tanımlaması sınıfın ismini ve kaşlı ayraçlar içine alınmış sınıf gövdesinden oluşur. Sınıf isminin öncesinde çeşitli niteleyiciler gelebilir. Sınıf gövdesi; alanlardan, metotlardan ve yaratıcı metotlardan meydana gelir. Alanlar sınıfın durumunu, metotlar ise sınıfın davranışını simgeler. Öte yandan, yaratıcı metotlar yeni nesne yaratma işini üstlenirler ve new anahtar sözcüğü ile çağırılırlar. Normal metotların aksine, yaratıcı metotlar hiçbir değer ya da tip döndürmezler.

3 Nesneleri yaratma ve kullanma

Nesneler

Tanımladığımız bir sınıftan birçok nesne üretebiliriz. Bir bakıma sınıflar ayrıntılı tasarı planlarıdır. Bir plandan birçok nesne oluşturabiliriz. Hâliyle bu nesnelerin alanlarının özellikleri farklı farklı olacaktır. Bu nesnelerin metotlarını kullanarak birtakım işler gerçekleştiririz. Günün sonunda işimizin bittiği nesneler, Java çöp toplama mekanizması tarafından otomatik olarak temizlenirler. Böylece kullanılmayan miadı dolmuş nesnelerin tükettiği kaynaklar yeniden sisteme kazandırılmış olur. Programcı nesneleri yok etme işi ile uğraşmaz ve böylece yükü hafiflemiş olur. Eğer bir sınıfa ait değişken ya da metot tanımlarken önüne static anahtar sözcüğünü koyarsak bunlara sınıf üyeleri (class members) adı verilir. Eğer static anahtar sözcüğünü koymazsak, bunlara örnek üyeleri (instance members) denir. Sınıf üyeleri, bir sınıf tanımlamasından üretilmiş bütün nesneler için ortaktır. Bu tür üyelere erişmek için new anahtar sözcüğü ile nesne yaratmak zorunlu değildir. Sınıfın adı ile erişilebilir.

4 Java'da sınıf içinde sınıf tanımlayabilme

5 Enum tipinde değişken tanımlayabilme

İç-İçe Sınıflar

Kodumuzu daha derli toplu hâle getirmek için, bazı sınıfları bazı sınıfların içinde tanımlama ihtiyacı duyabiliriz. İşte böyle durumlarda, her sınıfını kendine ait *.java uzantılı dosya içinde tanımlamak yerine, iç içe geçmiş sınıflar tanımlayabiliriz. Yerel sınıflar ve anonim sınıf tanımlamalarında, iç içe sınıfların bir alt kırılımı olarak değerlendirilebilir. Enum tanımlaması, sınıf tanımlamasına bezer. Tek fark class anahtar sözcüğü yerine enum anahtar sözcüğünün kullanılmasıdır. Enum tipi, kategorik değişkenleri ifade etmek için kullanılırlar. Kategorik değişkenlerden kasıt ise sayılabilir sayıda ve değişmeyen değerlerdir. Örneğin, haftanın günleri ya da yılın ayları gibi.

neler öğrendik?

1

```
 class UnixPath extends AbstractPath {

private final UnixFileSystem fs;

UnixPath(UnixFileSystem var1, byte[] var2) {
     this.fs = var1;
     this.path = var2;
}
private boolean isEmpty() {
    return this.path.length == 0;
}

private UnixPath emptyPath() {
    return new UnixPath(this.getFileSystem(), new byte[0]);
}

public UnixFileSystem getFileSystem() {
    return this.fs;
}
 }
```

Yukarıdaki sınıf tanımlamasında yaratıcı metot aşağıdakilerden hangisidir?

A. AbstractPath
B. UnixPath
C. isEmpty
D. emptyPath
E. getFileSystem

2

```
class StringCharBuffer extends CharBuffer
{
    CharSequence str;

    StringCharBuffer(CharSequence s, int start, int end) {
        super(-1, start, end, s.length());
        int n = s.length();
        if ((start < 0) || (start > n) || (end < start) || (end > n))
            throw new IndexOutOfBoundsException();
        str = s;
    }
}
```

Yukarıdaki kod parçasında, üye değişken aşağıdakilerden hangisidir?

A. str
B. s
C. start
D. end
E. n

3
```
class StringCharBuffer extends CharBuffer
{
    CharSequence str;

    StringCharBuffer(CharSequence s, int start, int end) {
        super(-1, start, end, s.length());
        int n = s.length();
        if ((start < 0) || (start > n) || (end < start) || (end > n))
            throw new IndexOutOfBoundsException();
        str = s;
    }
}
```

Yukarıdaki kod parçasında, yerel değişken aşağıdakilerden hangisidir?

A. str
B. s
C. start
D. end
E. n

4
```
public static File createTempFile(String prefix, String suffix,
                                  File directory)
    throws IOException {
    ...
}
```

Yukarıdaki metot tanımlamasında, istisna aşağıdakilerden hangisidir?

A. File
B. String
C. IOException
D. prefix
E. suffix

5
```
public static List<String> readAllLines(Path path, Charset cs)
throws IOException {
    ...
}
```

Yukarıda tanımlaması verilen metodun imzası aşağıdakilerden hangisidir?

A. List<String> readAllLines(Path path, Charset cs) throws IOException
B. List<String> readAllLines(Path path, Charset cs)
C. List<String> readAllLines
D. readAllLines(Path path, Charset cs)
E. static List<String> readAllLines

6
```
class Sınıf {

   void üyeMetot(int x, int y) {
   ...

    }
}
```

Yukarıda tanımlaması verilen sınıfa, aşağıdaki metotlardan hangisi eklendiğinde **derleyici hata** verir?

A. void üyeMetot(double x, double y) { … }
B. void üyeMetot(int x, int y, int z) { … }
C. void üyeMetot(double x) { … }
D. static int üyeMetot(int a, int b) { … }
E. void üyeMetot(int... x) { ... }

7
```
public static void main(String[] args) {

   Sınıf.üyeMetot(1, 2, 3);
}
```

Yukarıda örnek kullanımı verilen üye metodun tanımlaması aşağıdakilerden hangisidir?

A. static int üyeMetot(int... x) { … }
B. int üyeMetot(int... x) { … }
C. static int üyeMetot(int x, int y) { … }
D. void üyeMetot(int x, int y) { … }
E. static void üyeMetot(int x, int y) { … }

8 Aşağıdakilerden hangisi *anonim* olarak **tanımlanamaz?**

A. public class FileChannelImpl extends FileChannel { … }
B. public class InputStream implements Closeable { … }
C. public class Math { … }
D. public class ThreadLocalRandom extends Random { … }
E. public class Charset implements Serializable { … }

9
```
class Sınıf {

    short a;

    boolean b;

    static int c;

    double d;

    static void üyeMetot() {

    }
}
```

Yukarıda tanımlanan sınıf için, üyeMetot() aşağıdaki değişkenlerden hangisine erişebilir?

A. a
B. b
C. c
D. d
E. a, b, c, d

10
```
class Sınıf {

    short a;

    boolean b;

    static int c;

    double d;

    private void üyeMetot() {

    }
}
```

Yukarıda tanımlanan sınıf için, üyeMetot() aşağıdaki değişkenlerden hangisine erişebilir?

A. a
B. b
C. c
D. d
E. a, b, c, d

neler öğrendik yanıt anahtarı

1. B	Yanıtınız yanlış ise "Yaratıcı Metotlar" konusunu yeniden gözden geçiriniz.
2. A	Yanıtınız yanlış ise "Üye Değişkenler" konusunu yeniden gözden geçiriniz.
3. E	Yanıtınız yanlış ise "Üye Değişkenler" konusunu yeniden gözden geçiriniz.
4. C	Yanıtınız yanlış ise "Metot Tanımlama" konusunu yeniden gözden geçiriniz.
5. D	Yanıtınız yanlış ise "Metot Tanımlama" konusunu yeniden gözden geçiriniz.
6. D	Yanıtınız yanlış ise "Metot Tanımlama" konusunu yeniden gözden geçiriniz.
7. A	Yanıtınız yanlış ise "Metot Tanımlama" konusunu yeniden gözden geçiriniz.
8. C	Yanıtınız yanlış ise "Anonim Sınıflar" konusunu yeniden gözden geçiriniz.
9. C	Yanıtınız yanlış ise "Sınıf Üyeler" konusunu yeniden gözden geçiriniz.
10. E	Yanıtınız yanlış ise "Sınıf Üyeler" konusunu yeniden gözden geçiriniz.

4 Araştır Yanıt Anahtarı

java.lang.Object sınıfına ait iki metot vardır ki bunları kendi tanımladığımız sınıflarda anlamlı bir biçimde yeniden tanımlamamız şiddetle tavsiye olunur.

java.lang.Object#equals(Object obj) Bu metodu nesnelerin eşit olup olmadığının kontrolü için kullanılır. İlkel veri tipleri için eşitlik durumunu == operatörü ile kontrol ediyorduk. Ama nesneler için bir metota ihtiyacımız vardır. Misal kendi tanımladığımız bir sınıftan türetilmiş iki nesnenin hangi durumda eşit olacağını ancak biz bilebiliriz. Örneğin, adı ve soyadı alanlarından oluşan sınıfın equals(Object obj) metodu şu şekilde gerçeklenebilirdi.

Araştır 1

```
public class Kişi {

    private final String ad;
    private final String soyad;

    @Override
    public boolean equals(Object o) {

        if (this == o) return true;
        if (o == null || getClass() !=  o.getClass())
return false;

        Kişi kişi = (Kişi) o;

        if (!ad.equals(kişi.ad)) return false;
        return soyad.equals(kişi.soyad);

    }
}
```

4 Araştır Yanıt Anahtarı

Araştır 1

Burada ilk iki satırda yapılan kontroller standart kontrollerdir ve hemen hemen bütün equals metotlarında ortaktırlar. Metotun argüman listesindeki değişkenimiz Object tipinde olduğu için ilk önce onu Kişi nesnesine çevirirız. Daha sonra eşitlik kontrolüne dahil etmek istediğimiz alanların tek tek eşitlik durumunu, yine alanların equals metotlarını kullanarak kontrol ederiz. Eğer tek bir alan bile farklı ise false geri döndürürüz. Eğer bütün alanlar eşitse true geri döndürürüz.

java.lang.Object#hashCode() Bu metodu nesnelerin T.C. kimlik numarası gibi düşünebilir. Nasıl aynı T.C. kimlik numarasına sahip iki farklı insan olamıyorsa, biz de bu metodu farklı nesneler için olabildiğince farklı tam sayılar döndürecek şekilde gerçeklemeliyiz. Genel olarak bir sınıf için hashCode üretmek istersek o sınıfın alanlarının hashCode değerlerini bir asal sayı ile çarpıp toplayarak gideriz. Böylece farklı özelliklere sahip nesneler için farklı farklı sayılar elde etmiş oluruz. Örneğin; adı ve soyadı alanlarından oluşan sınıfın hashCode metodu şu şekilde gerçeklenebilirdi. Burada asal sayı olarak 31 kullanıldığına dikkat ediniz. Bu sistematik ile farklı adı ve soyadı olan iki farklı nesne için aynı tamsayı değerinin hashCode olarak üretilmesinin önüne geçilmiş olur.

```
public class Kişi {

    private final String ad;
    private final String soyad;

    @Override
    public int hashCode() {
       int result = ad.hashCode();
       result = 31 * result + soyad.hashCode();
        return result;
    }
```

Araştır 2

Java programlama dilinde String'leri uç ucuna ekleyip birleştirmek için java.lang.StringBuilder sınıfı kullanılır. Yeni bir java.lang.StringBuilder nesnesi yarattıktan sonra bu nesnenin append metotunu kullanarak istediğimiz sayıda String'i uç uca ekleyebiliriz.

```
public static void main(String[] args) {

    StringBuilder builder = new   StringBuilder();

    builder.append("Ali")
           .append(' ')
           .append("ata")
           .append(' ')
           .append("bak")
           .append(".");

    System.out.println(builder.toString());
 }
```

4 Araştır Yanıt Anahtarı

Araştır 2

Örneğin yukardaki programın çıktısı "Ali ata bak." olacaktır. İşte bu şekilde kullanımı mümkün kılmak için, metotları bir tren gibi yazabilmek için, return this; ile biten metotlar yazmamız gerekir. Örnekte kullandığımız metodun kaynak koduna bakarsak bunu kendimizde görebiliriz:

```
public StringBuilder append(String str) {
    super.append(str);
    return this;
}
```

Araştır 3

Sabit değişkenler kullanmak yerine enum tipini kullanmak kodun güvenilirliğini artırır. Hataları derleme zamanında tespit etmemizi sağlar. Kan grubu modellediğimiz tipimizi kullanarak bu durumu bir örnek ile açıklayalım.

```
public enum KanGrubu {
    A, B, AB, O
}
```

Hangi kan grubunun hangi kan grubuna kan verebildiğini sınayan bir metot düşenelim.

```
static boolean kanUyumu(KanGrubu alıcı, KanGrubu
verici) {

    if (KanGrubu.O.equals(verici)) return true;

    if (KanGrubu.AB.equals(alıcı)) return true;

    ...
}
```

Bu metot uygulama içerisinde şu şekilde kullanılacaktır:

```
public static void main(String[] args) {

    if (kanUyumu(KanGrubu.AB, KanGrubu.O))
        System.out.println("AB kan grubu 0 kan
grubundan kan alabilir");
    else
        System.out.println("AB kan grubu 0 kan
grubundan kan alamaz!");
}
```

4 Araştır Yanıt Anahtarı

Araştır 3

Şimdi ise aynı kodu `String` sabitler kullanarak yapalım ve neyin yanlış gidebileceğini görelim.

```
public class Test {

    static final String AB = "AB";
    static final String A = "A";
    static final String B = "B";
    static final String O = "O";

    static boolean kanUyumu(String alıcı, String
verici) {

        if (O.equals(verici)) return true;
        if (AB.equals(alıcı)) return true;
        return false;
    }

    public static void main(String[] args) {
        if (kanUyumu(AB, O))
            System.out.println("AB kan grubu O kan
grubundan kan alabilir");
        else
            System.out.println("AB kan grubu O kan
grubundan kan alamaz!");
    }
}
```

Şu aşamda bir sorun yok gibi görünüyor. Ancak kanUyumu metotunun argümanları String tipinde olduğu için, bu metota herhangi bir String değerini parameter olarak geçebiliriz. Örneğin:

```
public static void main(String[] args) {

    if (kanUyumu("C", "D"))
        System.out.println("C kan grubu D kan
grubundan kan alabilir");
    else
        System.out.println("C kan grubu D kan
grubundan kan alamaz!");

}
```

Bu örnekte, gerçekte C ve D diye kan grupları olmamasına rağmen kod derleme hatası vermez. Ancak enum tipi kullandığımız durumda, var olmayan bir kan grubunu istesekte yazamayız. Kod derleme hatası verir. Böylece programcıdan kaynaklanabilecek hataların erken safhada önüne geçilmiş olur. Sabit kullanmak yerine enum tipini kullanmanın avantajı budur.

Kaynakça

Gallardo, R. and Hommel, S. and Kannan, S. and Gordon, J. and Zakhour, S. (2016) *The Java Tutorial: A Short Course on the Basics* (6th Edition). Addison-Wesley Professional.

İnternet Kaynakları

Oracle Java Documentation, https://docs.oracle.com/javase/tutorial/java/javaOO/index.html
Son Erişim: Ekim 2017.

Bölüm 5

Java'da Sarmalama, Kalıtım ve Çok Biçimlilik

Öğrenme çıktıları

Sarmalama

1 Java ile sarmalama kavramını tanımlayabilme

Kalıtım

2 Java ile kalıtım kavramını açıklayabilme

Çok Biçimlilik

3 Java ile çok biçimlilik kavramını tanımlayabilme

Anahtar Sözcükler: • Sarmalama • Kalıtım • Çok Biçimlilik • Aşırı Yükleme • Ezme

GİRİŞ

Bu ünitedeki işleyeceğimiz konu başlıklarına geçmeden önce nesneye yönelik programlama ile ilgili temel kavramları bir hatırlayalım. Önceki ünitelerde edindiğimiz bilgilerin de ışığında nesneler ve sınıflar arasındaki bağlantıyı şu şekilde tarif edebiliriz. Nesne, bir sınıfın gerçek hayattaki bir örneğidir. Örneğin insan isminde bir sınıf tanımlamış olduğumuzu varsayalım. Kendimiz ve etrafımızda her bir birey bu sınıfın örnekleridir. Dolayısıyla kendimiz ve etrafımızdaki bireylerin insan sınıfı türünde bir nesne olduğumuzu söyleyebiliriz.

Sınıf ve nesne kavramlarının üzerine bir takım bilgiler inşa edeceğimiz için önceki ünitelerdeki bahsedilen temel kavramları anımsadık. Hatırlanacağı üzere nesneye yönelik programlamanın sarmalama, kalıtım, çok biçimlilik ilkelerinden önceki ünitelerimizde bahsedilmişti. Bu ünitede bu kavramlardan üzerinde daha ayrıntılı duracağız. Ünitenin devamında ilk olarak sarmalama kavramını ayrıntılı örnekler ile öğreneceğiz. Sarmalama, her bilgiye her kaynağın ulaşamaması amacıyla geliştirilmiş bir terminolojidir. Nesneye yönelik programlamadan bahsettiğimiz için burada kaynakların sınıflar ve bu sınıflar türündeki nesneler olduğunu söyleyebiliriz. Bu konudan bahsederken bu sınırlamanın belirli anahtar kelimeler kullanılarak gerçekleştirilebileceğini öğreneceğiz. Ünitenin devamında bahsedeceğimiz bir diğer kavram da kalıtım kavramıdır. Kalıtım, tıpkı biyolojik olarak canlıların sınıflandırılması gibi sınıfların da hiyerarşik bir şekilde ifade edilmesidir. Örneğin bir kedinin hayvanlar sınıfına, bir menekşenin bitkiler sınıfına dâhil olduğunu söyleyebiliriz. Nesneye yönelik programlamada da bu tür ilişkiler kurmanın gerekli olduğunu bu bölümün devamında açıklayacağız. Bu ünitede son olarak çok biçimlilik kavramından bahsedeceğiz. Çok biçimlilik, kalıtım kavramı ile doğrudan ilgilidir ve temel olarak bir nesnenin davranış şekillerinin duruma göre değişebilmesidir. Kalıtım kavramı gereği, kedi ve köpeği hayvan isimli bir sınıfın altında hiyerarşik olarak ifade edebiliriz. Genel olarak hayvanlar için ses çıkar komutunun olduğunu ve bunların hayvan sınıfı içerisinde tanımlandığını düşünelim. Kedi ve köpek için bu komut verildiğinde çıkan ses gözlemlenebilecek fakat iki tür için de bu sesler farklı olacaktır. Bu durumu çok biçimlilik kavramına temel bir örnek olarak verebiliriz.

SARMALAMA

Sarmalama, bir nesnenin özelliklerinin ve metotlarının erişimini sınırlandırmak amacını taşıyan bir kavramdır. Örneğin bir sürücü aracın kontak anahtarını çevirdiğinde çalıştığını veya çalışmadığını gözlemleyebilir. Ancak bu işlemin gerçekleştirilebilmesi için arka planda araç donanımlarının yaptığı eylemleri gözlemleyemez. Sonuç itibariyle bu eylemlerin ne şekilde gerçekleştirildiği sürücü için değil daha çok bir araç bakım teknisyeni için önemlidir. Bu sebeple bu eylemlerin gerçekleşmesi ile ilgili ayrıntılar araç sürücüsünden gizlenebilir. Nesneye yönelik programlama dillerinde özelliklerin ve metotların erişim yetkilendirmelerinin yapılması için erişim belirleyicileri kullanılır. Bu erişim belirleyicilerinin sağladığı yetkilendirmelere göre bir sınıfın özellik ve metotlarına aynı sınıf içerisinden, aynı **paket** içerisindeki sınıflardan, diğer sınıflardan ve ilgili sınıftan kalıtım yoluyla türeyecek sınıflardan erişilebilmesi mümkün olmaktadır. Java programlama dilinde özelliklerin ve metotların tanımlarında yer alan dört erişim belirleyicisi vardır. Bunlar **private, public, protected** anahtar kelimeleri ve bu üç anahtar kelimesinin de olmadığı **varsayılan (default)** olarak adlandırılan durumdur. Sarmalama kavramını anlatırken daha çok private ve public erişim belirleyicilerini kullanacağız. Katı bir şekilde bilgi gizleme durumu söz konusu olduğunda private erişim belirleyicisinin kullanılması uygun olacaktır. Bu anahtar kelimelerin kullanımını göstermek için aşağıdaki Java kodunu inceleyeceğiz. AnahtarKelimeKullanimi isimli Java sınıfı içerisinde ozellik1'in private, ozellik2'nin public, ozellik3'ün protected ve ozellik4'ün varsayılan erişim belirleyicileri ile tanımlandığını görmekteyiz. Bu dört özelliğin hepsinin türleri **int** olmakla birlikte erişim yetkilendirmeleri farklılıklar içermektedir.

Paket

Paket kavramı, Java programlarında kullanılan ve sınıfların hiyerarşik olarak kataloglanmasına imkân sağlayan bir kavramdır. Bu kavram sayesinde ilgili sınıflar aynı dizin içerisinde yer almış olurlar.

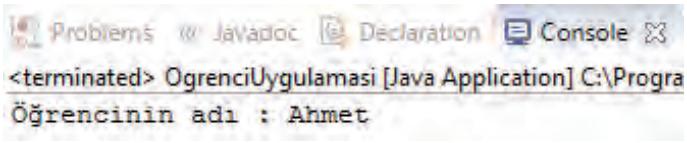
```
/* AnahtarKelimeKullanimi.java */

public class AnahtarKelimeKullanimi {
   private int ozellik1;
   public int ozellik2;
   protected int ozellik3;
   int ozellik4;
}
```

Bu kavrama verebileceğimiz bir diğer örnekte aşağıda Java kodları verilmiş olan sınıflardır. Ogrenci isimli Java sınıfında ogrenciIsim özelliğinin private anahtar kelimesi ile tanımlandığını görmekteyiz. Bu durumda OgrenciUygulamasi isimli sınıf içerisinde tanımlanmış olan ogrenci1 nesnesinin bu özelliğe erişimi mümkün olmayacaktır. Ancak Ogrenci isimli sınıfta tanımladığımız isimAta ve isimGetir metotları vasıtasıyla bu özelliğinin değerine ulaşılabilmesi ve değiştirilmesi mümkün olacaktır. Örnekte de gördüğümüz üzere bahsedilen bu iki metot public anahtar kelimesi ile tanımlanmıştır.

- **Private**
  Bu erişim belirleyicisi ile tanımlanan özelliklere ve metotlara sadece ilgili sınıf tarafından erişilebilir.
- **Public**
  Bu erişim belirleyicisi ile tanımlanan özelliklere ve metotlara bütün sınıflar erişilebilir.
- **Protected**
  Bu erişim belirleyicisi ile tanımlanan özelliklere ve metotlara ilgili sınıf, aynı paket içerisindeki diğer sınıflar ve bu sınıftan kalıtım yoluyla türeyecek sınıflar tarafından erişilebilir.
- **Varsayılan (Default)**
  Bu erişim belirleyicisi ile tanımlanan özelliklere ve metotlara sadece ilgili sınıf ve aynı paket içerisindeki diğer sınıflar tarafından erişilebilir.

```
/* Ogrenci.java */

public class Ogrenci {
  private String ogrenciIsim;

  public void isimAta(String ad){
      ogrenciIsim = ad;
}

  public String isimGetir( ) {
      return ogrenciIsim;
  }
}

/* OgrenciUygulamasi.java */

public class OgrenciUygulamasi {
   public static void main(String[] args) {
      Ogrenci ogrenci1 = new Ogrenci();
      ogrenci1.isimAta("Ahmet");
      System.out.println("Öğrencinin adı : "+ogrenci1.isimGetir());
   }
}
```

OgrenciUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.1'deki ekran görüntüsü ortaya çıkmaktadır.



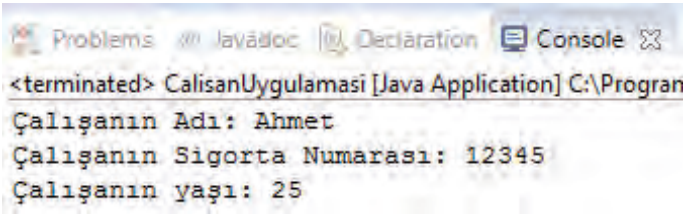
Resim 5.1 OgrenciUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, sarmalama ile ilgili bir örnek daha bulunmaktadır. Bu örnekte Calisan isimli bir sınıf yer almaktadır. Bu sınıfın calisanSigortaNo, calisanIsim ve calisanYas isimli özellikleri bulunmaktadır. Bu özelliklerin tamamı private anahtar kelimesi ile tanımlanmıştır. Ayrıca Calisan isimli sınıfın public anahtar kelimesi ile tanımlanmış sigortaNoGetir, sigortaNoAta, isimGetir, isimAta, yasGetir ve yasAta isimli metotları bulunmaktadır.

```
/* Calisan.java */

public class Calisan {
    private int calisanSigortaNo;
    private String calisanIsim;
    private int calisanYas;

    //Getter and Setter methods
    public int sigortaNoGetir(){
        return calisanSigortaNo;
    }

    public void sigortaNoAta(int yeniSigortaNo){
        calisanSigortaNo = yeniSigortaNo;
    }

    public String isimGetir(){
        return calisanIsim;
    }

    public void isimAta(String yeniIsim){
        calisanIsim = yeniIsim;
    }

    public int yasGetir(){
        return calisanYas;
    }

    public void yasAta(int yeniYas){
        calisanYas = yeniYas;
    }
}
```

```
/* CalisanUygulamasi.java */

public class CalisanUygulamasi {
    public static void main(String args[]){
        Calisan kisi = new Calisan();
        kisi.isimAta("Ahmet");
        kisi.yasAta(25);
        kisi.sigortaNoAta(12345);
        System.out.println("Çalışanın Adı: " + kisi.isimGetir());
       System.out.println("Çalışanın Sigorta Numarası: " +
kisi.sigortaNoGetir());
        System.out.println("Çalışanın yaşı: " + kisi.yasGetir());
    }
}
```

CalisanUygulamasi isimli Java sınıfı içerisinde kisi isimli bir nesne oluşturulmuş ve daha sonra bu nesneye ait özelliklere doğrudan değer atamak yerine isimAta, yasAta ve sigortaNoAta gibi metotlar vasıtasıyla değer atanmıştır. Aynı şekilde bu değerler ekrana yazdırılırken isimGetir, sigortaNoGetir ve yasGetir metotlarının kullanıldığını görmekteyiz. CalisanUygulamasi sınıfının içerisinde calisanYas özelliğinin kisi.calisanYas şeklinde kullanılması mümkün olmamaktadır. Örnek kodlarımızı bilgisayar ortamında çalıştırırken böyle bir değişiklik yapmak istersek derleyici hatası ile karşı karşıya kalırız. Çünkü calisanYas özelliği Calisan sınıfı içerinde private anahtar kelimesi kullanılarak tanımlanmıştır. CalisanUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.2'deki ekran görüntüsü ortaya çıkmaktadır.

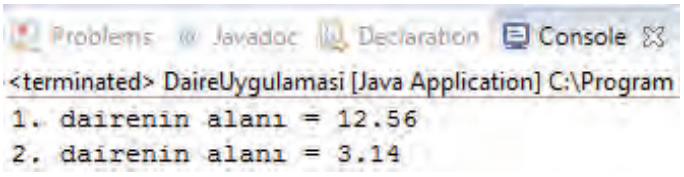


Resim 5.2 CalisanUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

Bu konuda bir diğer örnek olarak aşağıdaki Daire isimli sınıfı verebiliriz. Daire sınıfı aşağıdaki tanımlandığı şekliyle yarıçap, renk ve pi sayısının değerini gösteren özelliklere sahiptir. DaireUygulamasi isimli sınıf içerisinden Daire sınıfı türündeki daire1 ve daire2 nesnelerinin çağrıldığını ve hesaplanan alanların ekrana yazdırıldığını görmekteyiz. Bu örnekte DaireUygulamasi sınıfı bu alan hesabının hangi matematiksel yöntemlerle ne şekilde yapıldığını görmemekte fakat sonucuna erişebilmektedir.

```
/* Daire.java */

public class Daire {
        private double yaricap;
        private String renk;
        private double piSayisi = 3.14;
        public Daire() {
            yaricap = 1.0;
            renk = "mavi";
        }
        public Daire(double r, String c) {
            yaricap = r;
            renk = c;
        }
        public double alanHesapla() {
            return yaricap * yaricap * piSayisi;
        }
}

/* DaireUygulamasi.java */

public class DaireUygulamasi {
    public static void main(String[] args) {
        Daire daire1 = new Daire(2.0, "mor");
        System.out.println("1. dairenin alanı = "+ daire1.
alanHesapla());

        Daire daire2 = new Daire();
        System.out.println("2. dairenin alanı = "+ daire2.
alanHesapla());

    }
}
```

DaireUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.3'teki ekran görüntüsü ortaya çıkmaktadır.



Resim 5.3 DaireUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

Bu alt başlık altında sarmalama kavramının tanımı yapılmış ve Java programlama ortamında bu kavrama ait çeşitli örnekler verilmiştir. Örneklerin tamamında görüldüğü üzere sarmalama kavramı yardımıyla nesnelerin özellikleri ve metotlarının içerikleri bu nesneleri kullananlardan gerek olduğu oranda gizli tutulabilmektedir.

Öğrenme Çıktısı

1 Java ile sarmalama kavramını tanımlayabilme

Araştır 1

C# programlama dilinde sarmalama kavramı için kullanılan erişim belirleyicilerini araştırınız ve bunları listeleyiniz. C# programlama dilinde kullanılan erişim belirleyicilerini sayı ve isimlendirme açısından Java programlama dilinde kullanılanlar ile kıyaslayınız.

İlişkilendir

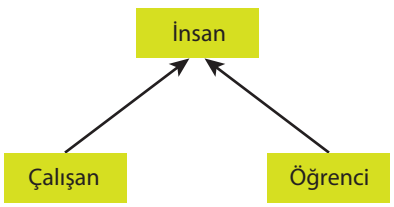
Sarmalama kavramı ile erişim belirleyicilerini ilişkilendirin.

Anlat/Paylaş

Sarmalama kavramının var olmadığı durumda ortaya çıkabilecek problemleri düşünün.

KALITIM

Kalıtım, nesneye yönelik programlamanın temel kavramlarından birisidir. Bu kavram sayesinde sınıfları hiyerarşik bir şekilde ifade edebiliriz. Bu sayede yazılan program kodlarının yeniden kullanılabilirliği de artar. Bir sınıftan kalıtım yoluyla yeni bir sınıf türetildiğinde bu yeni sınıfa alt sınıf denilir. Mevcut sınıf ise üst sınıf olarak adlandırılır. Şekil 5.1'de kalıtım kavramına günlük hayatımızdaki sınıflardan temel bir örnek verilmiştir. Şekilde, Çalışan ve Öğrenci isimli sınıfların İnsan isimli sınıftan kalıtım yoluyla türemiş olduğu görülmektedir. Çalışan ve Öğrenci sınıfları alt sınıf olarak adlandırılırken İnsan sınıfı üst sınıf olarak tanımlanmaktadır.

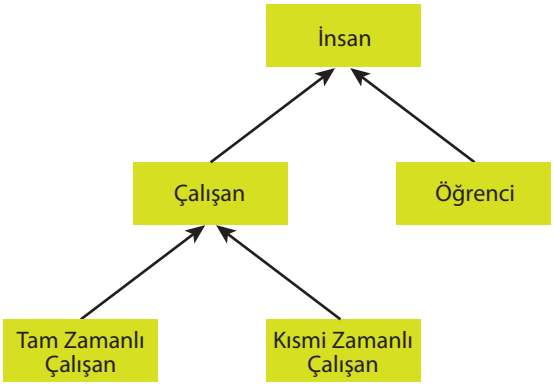


Şekil 5.1 Kalıtım kavramına temel bir örnek

Şekil 5.1'deki örneğin Java program kodu ile ifade edilişi aşağıda verilmiştir. Java programlama ortamında kalıtım extends anahtar kelimesi ile sağlanır.

```
/* Insan.java */

public class Insan {
}
/* Calisan.java */

public class Calisan extends Insan {
}
/* Ogrenci.java */

public class Ogrenci extends Insan {
}
```

Yukarıda ki örnekte Ogrenci ve Calisan isimli sınıfların birer üst sınıfı olduğunu görmekteyiz. Büyük çapta programlar yazılırken bu yapının tek bir seviyede sonlanmayabilir ve bir hiyararşi şeklinde büyüyebilir. Şekil 5.2'de bir önceki şekildeki örneğin biraz daha detaylandırılmış halini görmekteyiz. Bu örnekte Calisan isimli sınıftan kalıtım yoluyla tam zamanlı çalışan ve kısmı zamanlı çalışan olmak üzere 2 sınıf daha türetildiğini görmekteyiz.



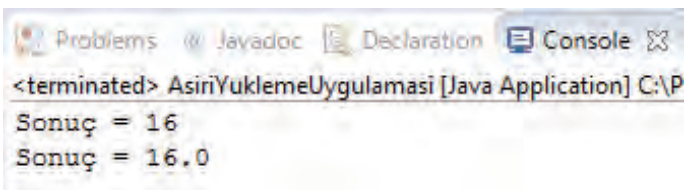
Şekil 5.2 Kalıtım kavramına hiyerarşik bir örnek

Aşırı Yükleme ve Ezme

Kalıtım kavramı içerisinde aşırı yükleme (overloading) ve ezme (overriding) kavramları sıklıkla kullanıldığı için bu kavramlara şimdi kısaca değineceğiz. Aşırı yükleme kavramı, sınıflar içerisinde aynı isimde olmakla birlikte farklı parametreler alan metotların bulunmasıdır. Örneğin aşağıda *AsiriYuklemeUygulamasi* adında bir sınıf yer almaktadır. Bu sınıfın içerisinde *kareHesapla* isminde olmalarına karşın farklı tiplerde parametre alan 2 metot yer almaktadır. *M* nesnesine ait *kareHesapla* metodunun *main* metodunun içerisinde 2 kez çağrıldığını görmekteyiz. Metodun ilk çağrılışında int tipinde tamsayı parametresi alan aynı isimdeki metot çalıştırılmaktadır. Fakat metodun ikinci çağrılışında double tipinde ondalıklı sayı parametresi alan aynı isimdeki diğer metot çalıştırılmaktadır. Bu durum, aşırı yükleme kavramı için temel bir örnektir.

```
/* AsiriYuklemeUygulamasi.java */

public class AsiriYuklemeUygulamasi {
      void kareHesapla(int x){

            x = x*x;
            System.out.println("Sonuç = " + x);
      }
      void kareHesapla(double x){

            x = x*x;
            System.out.println("Sonuç = " + x);
      }
      public static void main(String[] args) {
            AsiriYuklemeUygulamasi m = new AsiriYuklemeUygulamasi();
            m.kareHesapla(4);
            m.kareHesapla(4.0);
      }
}
```

AsiriYuklemeUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.4'teki ekran görüntüsü ortaya çıkmaktadır.

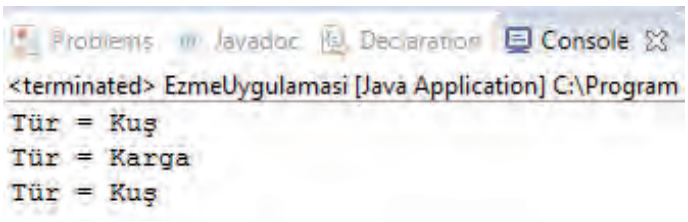


Resim 5. 4 AsiriYuklemeUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

Ezme kavramını ise şu şekilde tarif edebiliriz. Kalıtım kavramı gereği bir sınıftan başka bir sınıf türetildiğinde üst sınıftaki metot türetilen alt sınıfta da otomatik olarak bulunmaktadır. Ancak türetilen yeni sınıf içerisinde bu metotta değişiklik yapmak istersek o metodu ezmemiz gerekir. Ezme kavramına Java programlama dilinde bir örnek verelim. Örneğin Kus sınıfı ve bu sınıflardan türemiş Karga ve Serce sınıfları bulunsun. Aşağıdaki EzmeUygulamasi isimli sınıf içerisinde bu üç sınıf türünde birer nesne yaratılmakta ve turGoster isimli metotları çağırmaktadır. Bu metot, ilk olarak Kus isimli sınıfta tanımlanmış olup bu sınıftan türeyen Karga ve Serce sınıflarına kalıtım yoluyla aktarılmıştır.

```
/* Kus.java */

public class Kus {
    public void turGoster() {
        System.out.println("Tür = Kuş");
    }
}
/* Karga.java */

public class Karga extends Kus{
    public void turGoster() {
        System.out.println("Tür = Karga");
    }
}
/* EzmeUygulamasi.java */

public class EzmeUygulamasi {
     public static void main(String[] args) {
           Kus kus = new Kus();
           Karga karga = new Karga();
           Serce serce = new Serce();
           kus.turGoster();
           karga.turGoster();
           serce.turGoster();
     }
}
```

/* Kus.java */



Resim 5.5 EzmeUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

Türetilen yeni sınıflar içerisinde turGoster metodu tanımlanmamış olsaydı EzmeUygulamasi isimli sınıf çalıştırıldığında "Tür = Kuş" şeklinde 3 adet çıktı üretmesi beklenirdi. Ancak Karga isimli sınıfın içerisinde turGoster metodunun yeniden tanımlandığını ve dolayısıyla ezildiğini görmekteyiz. Bu sebeple Karga sınıfı türündeki nesneye ait turGoster metodu çağrıldığında "Tür = Karga" çıktısı vermesi gerekecektir. Bu sebeple, EzmeUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.5'teki ekran görüntüsü ortaya çıkmaktadır.

Kalıtım Kavramı ile İlgili Örnek Uygulamalar

Aşağıda, kalıtım ile ilgili bir diğer örnek bulunmaktadır. Bu örnekte Sekil isimli bir sınıf yer almaktadır. Bu sınıfın isim, cevre ve alan isimli özellikleri bulunmaktadır. Ayrıca Sekil isimli sınıfın public anahtar kelimesi ile tanımlanmış goruntule isimli metodu bulunmaktadır.

```
/* Sekil.java */

public class Sekil {
        String isim;
        double cevre, alan;
        public Sekil() {
            isim = "belirsiz";
            cevre = 0;
            alan = 0;
        }
        public void goruntule() {
            System.out.println("İsim: " + isim);
            System.out.println("Cevre: " + cevre);
            System.out.println("Alan: " + alan);
            System.out.println();
        }
}
```

Aşağıda, Sekil sınıfından türemiş olan Daire ve Dikdortgen isimlerinde sınıfları görmekteyiz. Fakat bu sınıfların kodlarının içerisinde cevre, alan ozelliklerini ve goruntule metodunu görmemekteyiz. Ancak bu sınıflar kalıtım yoluyla Sekil sınıfından türetildiği için bahsedilen bu özelliklere ve metotlara otomatik olarak sahip durumdadırlar.

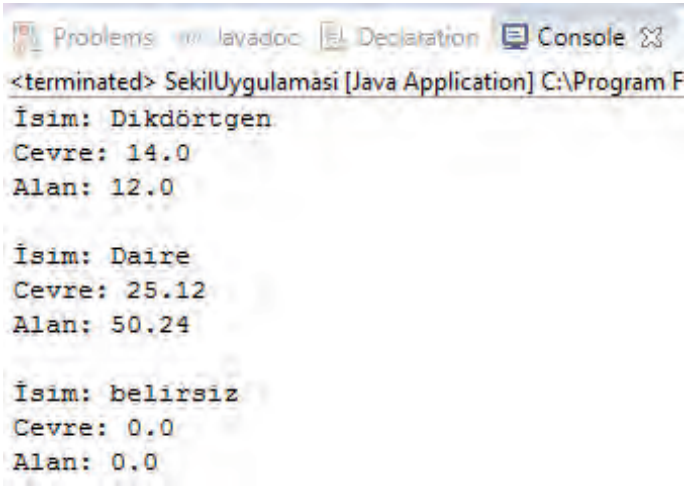
```
/* Daire.java */

public class Daire extends Sekil{
        double yaricap;
        double piSayisi = 3.14;
        public Daire() {
            isim = "Daire";
            yaricap = 0;
        }
        public Daire(double yaricapDegeri) {
            isim = "Daire";
            yaricap = yaricapDegeri;
        }
        public void alanHesapla() {
            alan = piSayisi * yaricap * yaricap;
        }
        public void cevreHesapla() {
            cevre = 2 * piSayisi * yaricap;
        }
}
```

```
/* Dikdortgen.java */

public class Dikdortgen extends Sekil{
        double uzunluk, genislik;
        public Dikdortgen() {
            isim = "Dikdörtgen";
            uzunluk = 0;
            genislik = 0;
        }
        public Dikdortgen(double uzunlukDegeri, double
genislikDegeri) {
             isim = "Dikdörtgen";
             uzunluk = uzunlukDegeri;
             genislik = genislikDegeri;
        }
        public void alanHesapla() {
            alan = uzunluk * genislik;
        }
        public void cevreHesapla() {
            cevre = 2 *(uzunluk + genislik);
        }
}
```

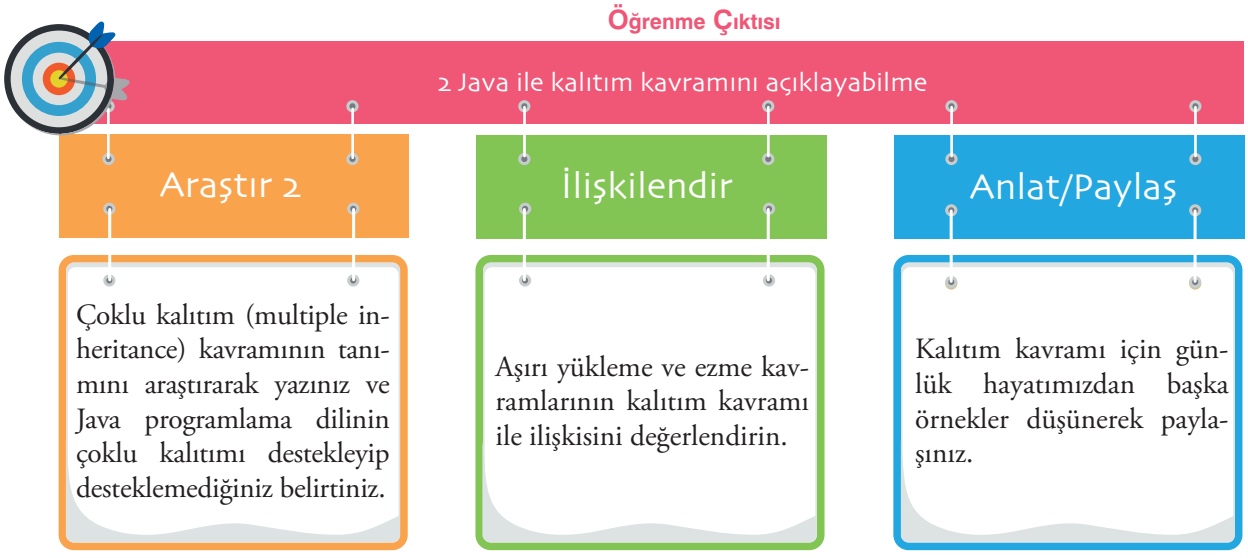
Şimdi bu sınıfların türünde nesnelerin kullanıldığı SekilUygulamasi sınıfına bir göz atalım. Bu sınıf içerisinde sekil, dortgen ve daire isimli nesneler olduğunu görmekteyiz. İlk olarak, dortgen ve daire isimli nesnelerin ilgili metotları çalıştırılarak cevre ve alan değerleri hesaplanmaktadır. Daha sonra, Sekil sınıfında tanımlanmış olan goruntule metodu vasıtasıyla hesaplanan bu değerler ekrana yazdırılmaktadır.

```
/* SekilUygulamasi.java */

public class SekilUygulamasi {
        public static void main(String args[]) {
            Sekil sekil = new Sekil();
            Dikdortgen dortgen = new Dikdortgen(3.0, 4.0);
            Daire daire = new Daire(4.0);

            dortgen.cevreHesapla();
            dortgen.alanHesapla();
            daire.cevreHesapla();
            daire.alanHesapla();
            dortgen.goruntule();
            daire.goruntule();
            sekil.goruntule();
        }
}
```

SekilUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.6'daki ekran görüntüsü ortaya çıkmaktadır. Dortgen ve daire nesneleri için hesaplama sonucu bulunan değerlerin, Sekil nesnesi için ise varsayılan değer olarak atanan sıfır değerinin çıktı olarak görüldüğünü gözlemlemekteyiz.



Resim 5.6 Sekil Uygulamasi sınıfının çalıştırılınca verdiği ekran çıktısı



Yaşamla İlişkilendir

Sağlıklı Genleri Yedekleyen Bitki

Botanik dünyası şu günlerde heyecan verici genetik bir bulguyla çalkalanıyor. Amerikalı bilim adamları, bir tür bitkide hastalıklara neden olan genetik mutasyonların düzeltilmesini sağlayan 'yedekleme sistemi' olabileceğini ortaya çıkardı. Bir önceki kuşaktan hatalı gen miras alan bitkinin dört kuşak öncesine kadar geri giderek sağlıklı geni kopyalayıp kullandığı sanılıyor. Amerikan Purdue Üniversitesi'nden Dr. Robert E. Pruitt, Dr. Susan J. Lolle ve meslektaşlarının gerçekleştirdiği araştırma sonuçları Nature Dergisi'nin internet sitesinde yayınlandı. Araştırma sonuçlarının doğrulanması halinde 19'uncu yüzyılda Gregor Mendel tarafından belirlenen kalıtım yasaları açısından bu bitki türünün bir istisna teşkil edeceği belirtiliyor. Dr. Pruitt ve Dr. Lolle, üç yıl boyunca bir hardal otu türü olan 'Arabidopsis thaliana' bitkisinin dış yüzeyini araştırdılar. Mutasyona uğramış bir gen yüzünden çiçeğin taç yaprakları toplanıyordu. Genin her iki kopyası da mutasyona uğramış olduğundan, normal özelliklere sahip filizlere sahip olması imkansızdı. Ancak bitkinin yeni filizlerinin yüzde 10'unun normal olduğu görüldü. Araştırma sonucunda hatalı genin onarıldığı anlaşıldı. DNA'yı ikinci kez inceleyen araştırmacılar, DNA'da söz konusu genin sağlıklı kopyasının şifreli kopyasını bulamadılar. Bu kez DNA'nın kimyasal kuzeni olan RNA'ya baktılar. RNA'nın tıpkı bilgisayarlarda olduğu gibi DNA'daki genleri yedeklediği ve en az dört kuşak sakladığı yolunda bulgulara ulaştılar. Bu bitki için sözkonusu olan bu 'gen yedekleme sistemi'nin insanlar için de geçerli olabileceği belirtiliyor.

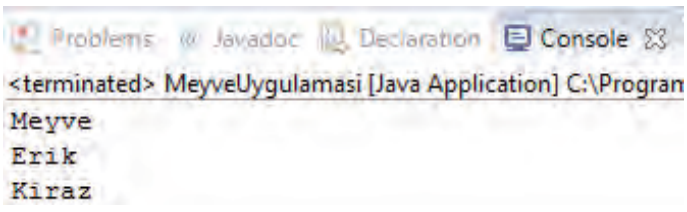
Kaynak: Bu habere http://www.hurriyet.com.tr/saglikli-genleri-yedekleyen-bitki-38710794 internet adresinden ulaşabilirsiniz.

ÇOK BİÇİMLİLİK

Çok biçimlilik, temel olarak bir nesnenin davranış şekillerinin duruma göre değişebilmesidir. Aynı temel sınıftan türetilmiş olan sınıflarda paylaşılan aynı isme sahip metodların bu sınıflarda farklı şekillerde uyarlanabilmesi mümkündür. Bu durum için günlük hayatımızdaki sınıflardan şu şekilde bir örnek verebiliriz. Canlı isminde bir sınıf ve bu sınıftan türemiş olan balık ve kuş isimli başka sınıflar olduğunu düşünelim. Canlı sınıfı içerisinde hareket et adında bir komut bulunsun. Hareket et komutu, balık ve kuş sınıfları için farklı şekillerde gerçekleştirilmelidir. Balık sınıf bu eylemi yüzerek gerçekleştirirken kuş sınıfı yürüyerek veya uçarak gerçekleştirecektir. Bu eylemin nasıl gerçekleştirileceği ise çalışma anında eylemi yapan canlının balık mı kuş mu olacağına göre anlık olarak belirlenmelidir. Bu durumda balık ve kuş sınıflarında aynı isimdeki bu metot farklı şekillerde tekrar tanımlanmalı ve bu yolla canlı sınıfındaki metot ezilmelidir. Aşağıda, bu konuya ait bir örnek bulunmaktadır. Örnekte, Meyve, Erik, Kiraz isimli sınıflar ve MeyveUygulamasi adında bu sınıf türlerinde nesneler içeren bir başka sınıf yer almaktadır. Erik ve Kiraz isimli sınıflar kalıtım yoluyla Meyve isimli sınıftan türemektedir. Meyve sınıfında yer alan isimGoster metodu aşağıdaki program kodlarında görüldüğü üzere bu sınıflarda farklı şekilde tanımlanmıştır.

```
/* Meyve.java */

public class Meyve {
    public void isimGoster() {
        System.out.println("Meyve");
    }
}

/* Erik.java */

public class Erik extends Meyve{
    public void isimGoster() {
        System.out.println("Erik");
    }
}

/* Kiraz.java */

public class Kiraz extends Meyve{
    public void isimGoster() {
        System.out.println("Kiraz");
    }
}

/* MeyveUygulamasi.java */

public class MeyveUygulamasi {
     public static void main(String[] args) {
          Meyve m = new Meyve();
          Meyve e = new Erik();
          Meyve k = new Kiraz();
          m.isimGoster();
          e.isimGoster();
          k.isimGoster();
     }
}
```

MeyveUygulamasi sınıfında üç nesne referansının da Meyve türünde tanımlandığını görmekteyiz. MeyveUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.7'deki ekran görüntüsü ortaya çıkmaktadır. Burada da görüldüğü gibi aynı şekilde Meyve türünde referans değişkenlerine sahip olsalar da üç nesnenin ürettiği çıktılar tamamen farklıdır. Çok biçimlilik kavramı sayesinde programın çalışması sırasında isimGoster metodlarının üç nesne için de farklı çıktılar ürettiğini görmekteyiz.

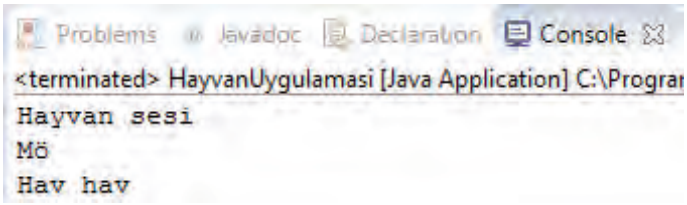


Resim 5.7 MeyveUygulamasi sınıfının çalıştırılınca verdiği ekran çıktısı

Çok biçimlilik kavramına bir başka örnek verelim. Örneğimizde Hayvan isimli sınıf ve bu sınıftan türemiş olan Inek ve Kopek isimlerinde sınıflar bulunmaktadır. Bu sınıfların tamamında sesCikar metodu yer almaktadır. HayvanUygulamasi isimli sınıfta ise bu üç sınıf türünde nesne yaratılmakta ve bu nesnelerin sesCikar metotları çağrılmaktadır. Bu nesnelerin tamamı, program kodlarında da görüleceği gibi Hayvan sınıfı türünde referans değişkenlerine sahiptirler.

```
/* Hayvan.java */

public class Hayvan {
      public void sesCikar(){
            System.out.println("Hayvan sesi");
      }
}

/* Inek.java */

public class Inek extends Hayvan{
      public void sesCikar(){
            System.out.println("Mö");
      }
}

/* Kopek.java */

public class Kopek extends Hayvan{
      public void sesCikar(){
            System.out.println("Hav hav");
      }
      public void sesCikar(double a){
            System.out.println("Hav hav");
      }
}

/* HayvanUygulamasi.java */

public class HayvanUygulamasi {
      public static void main(String[] args) {
            Hayvan h = new Hayvan();
            Hayvan i = new Inek();
            Hayvan k = new Kopek();
            h.sesCikar();
            i.sesCikar();
            k.sesCikar();
      }
}
```

HayvanUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 5.8'deki ekran görüntüsü ortaya çıkmaktadır. Bu metotların referans değişkeni tiplerinin tamamı aynı olmakla birlikte program çalıştığında ürettikleri çıktılar farklıdır. Çok biçimlilik kavramı gereği programın çalışma zamanında sesCikar isimli metotların ait oldukları sınıflar bazında farklılaştığını görmekteyiz.



Resim 5.8 HayvanUygulamasi sınıfının çalıştırılınca verdiği ekran çıktısı

öğrenme çıktıları ve bölüm özeti

1 Java ile sarmalama kavramını tanımlayabilme

Sarmalama

Kitabın bu bölümünde nesneye yönelik programlamanın temel prensibleri olan sarmalama, kalıtım ve çok biçimlilik kavramları ayrıntılı bir şekilde ele alınmıştır. Sarmalama, belirli bir bilgiye farklı kaynakların ihtiyacı olduğu seviyede erişiminin sağlanması ve erişimin gerekli olmadığı bölümlerin gizlenmesidir. Nesneye yönelik programlamadan bahsettiğimiz için bilgi kavramı özellik ve metotları, kaynaklar ise sınıfları temsil etmektedir. Dolayısıyla belirli özellik ve metotlara farklı sınıfların erişim yetkileri sarmalama yoluyla kısıtlanabilmektedir. Bu konuya günlük hayatımızdan temel bir örnek olarak bir sürücünün arabayı çalıştırması eylemi verilmiştir. Bu örnekte bir sürücünün kontak anahtarını çevirdiğinde aracın çalıştığını veya çalışmadığını görebildiği ancak bu işlemin gerçekleştirilebilmesi için arka planda araç donanımlarının yaptığı eylemleri gözlemleyemediğinden bahsedilmiştir.

2 Java ile kalıtım kavramını açıklayabilme

Kalıtım

Ünitenin bir diğer alt başlığı olan kalıtım kavramı ise tıpkı biyolojik olarak canlıların sınıflandırılması gibi sınıfların da hiyerarşik bir şekilde ifade edilmesidir. Bu konuya temel bir örnek olarak bir kedinin hayvanlar sınıfına, bir menekşenin bitkiler sınıfına dâhil olması durumunu verebiliriz. Kalıtım yoluyla bir sınıftan başka sınıflar türetildiğinde bu sınıfın özellik ve metodları türetilen yeni sınıfa aktarılır. Bu durumda türetilen yeni sınıfa alt sınıf, mevcut sınıfa ise üst sınıf denilir. Kalıtım kavramı sayesinde yazılan program kodlarının yeniden kullanılabilirliği de artmaktadır. Kalıtım başlığı altında aşırı yükleme ve ezme kavramları hakkında da bilgi verilmiştir. Aşırı yükleme, sınıflar içerisinde aynı isimde olmakla birlikte farklı parametreler alan metotların bulunmasıdır. Ezme ise mevcut bir sınıftan kalıtım yoluyla başka bir sınıf türetildiği durumda mevcut sınıfta bulunan bazı metotların bu yeni sınıfta içerikleri farklı olarak yeniden tanımlanmasıdır.

3 Java ile çok biçimlilik kavramını tanımlayabilme

Çok Biçimlilik

Çok biçimlilik, temel olarak bir nesnenin davranış şekillerinin duruma göre değişebilmesidir. Canlı isimli sınıftan türetilen balık ve kuş sınıflarının hareket et komutunu farklı şekilde gerçekleştirmesi bu kavrama temel bir örnek olarak verilmiştir. Balık sınıf bu eylemi yüzerek gerçekleştirirken kuş sınıfı yürüyerek veya uçarak gerçekleştirecektir. Bu eylemin nasıl gerçekleştirileceği ise çalışma anında eylemi yapan canlının balık mı kuş mu olacağına göre anlık olarak belirlenecektir. Çok biçimlilik kavramının uygulanması için üst sınıfta tanımlanmış olan bir metodun alt sınıfta ezilmesi ve içeriği farklı şekilde yeniden tanımlanması gerekmektedir.

neler öğrendik?

1 Aşağıda sırasıyla günlük hayatımızdan sınıf örnekleri ve bu sınıflardan kalıtım yoluyla türeyen başka sınıflar yer almaktadır. Buna göre aşağıdakilerden hangisi doğru bir eşleşmedir?

A. Elma - Armut
B. Elma - Kiraz
C. Sebze - Meyve
D. Meyve - Kiraz
E. Yol - Araba

2 Bilgisayar kullanıcılarının bir yazıcıya kâğıt koyup çıktı almaları fakat arka planda yazıcının nasıl çalıştığını bilmemeleri aşağıdaki kavramlardan hangisine bir örnektir?

A. Sarmalama'ya
B. Kalıtım'a
C. Çok biçimlilik'e
D. Ezme'ye
E. Aşırı yükleme'ye

3 Java programlama ortamında kalıtım kavramı aşağıdaki anahtar kelimelerin hangisiyle sağlanır?

A. public
B. private
C. protected
D. extends
E. void

4 Hesaplama yapmak amaçlı oluşturulan bir sınıfta aynı isimde olmalarına karşın farklı tipte parametreler alarak işlemler yapan metotların bulunması aşağıdaki kavramlardan hangisine bir örnektir?

A. Aşırı yükleme'ye
B. Ezme'ye
C. Kalıtım'a
D. Sarmalama'ya
E. Çok biçimlilik'e

5 Sekil isimli sınıftan daire isimli bir başka sınıf türetilmesi ve bu sınıf içerisinde Sekil isimli sınıfta önceden tanımlanmış bir metodun farklı bir şekilde yeniden tanımlanması aşağıdaki kavramlardan hangisine bir örnektir?

A. Sarmalama'ya
B. Sadelik'e
C. Aşırı yükleme'ye
D. Ezme'ye
E. Soyutlama'ya

6 Sekil isimli bir sınıftan kalıtım yoluyla türetilmiş daire ve kare sınıfları bulunmaktadır. Sekil isimli sınıfta ilgili şeklin ekranda görüntülenmesini sağlayan ciz isimli bir metot tanımlanmıştır. Buna göre, Daire ve kare sınıflarına ait nesnelerde ciz metodu çağrıldığında ekranda farklı görünümler ortaya çıkması aşağıdakilerden hangisi ile açıklanır?

A. Çok biçimlilik
B. Sarmalama
C. Soyutlama
D. Sadelik
E. Basitlik

7 Java'da herhangi bir türde değişken tanımlanırken katı bir şekilde bilgi gizleme amacı söz konusu ise aşağıdaki anahtar kelimelerden hangisi kullanılır?

A. public
B. private
C. int
D. double
E. void

8 Aşağıdakilerden hangisi nesneye yönelik programlamanın temel prensiplerinden biri **değildir?**

A. Kalıtım
B. Sarmalama
C. Çok biçimlilik
D. Soyutlama
E. Sadelik

9 Tasıt isimli sınıftan kalıtım yoluyla araba, uçak, tren ve otobüs sınıfları türetilmesi durumunda bu sınıflardan hangisi bir üst sınıf olur?

A. Araba
B. Uçak
C. Tren
D. Otobüs
E. Taşıt

10 Aşağıdakilerden hangisi Sekil isimli bir sınıftan türemiş başka bir sınıftır?

A. Kare
B. Ot
C. Apartman
D. Yol
E. Kiraz

neler öğrendik yanıt anahtarı

1. D	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.	6. A	Yanıtınız yanlış ise "Çok Biçimlilik" konusunu yeniden gözden geçiriniz.
2. A	Yanıtınız yanlış ise "Sarmalama" konusunu yeniden gözden geçiriniz.	7. B	Yanıtınız yanlış ise "Sarmalama" konusunu yeniden gözden geçiriniz.
3. D	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.	8. E	Yanıtınız yanlış ise "Giriş" konusunu yeniden gözden geçiriniz.
4. A	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.	9. E	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.
5. D	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.	10. A	Yanıtınız yanlış ise "Kalıtım" konusunu yeniden gözden geçiriniz.

5 Araştır Yanıt Anahtarı

Araştır 1

C# programlama dilince sarmalama kavramı için aşağıda listelenen 5 adet erişim belirleyicisi kullanılmaktadır.

- Public
- Private
- Protected
- Internal
- Protected Internal

Buna karşın Java programlama dilinde 4 adet erişim belirleyicisi bulunmaktadır. C# programlama dilinde kullanılan 3 adet erişim belirleyicisinin isim ve içerik olarak Java programlama dili ile aynı olduğunu görmekteyiz.

Araştır 2

Çoklu kalıtım (multiple inheritance), bir sınıfın iki veya daha fazla üst sınıftan kalıtım yoluyla türetilmesidir. Java programlama dili çoklu kalıtımı doğrudan desteklemez ancak dolaylı yollarla bu kavram belirli oranda karşılanabilmektedir.

Araştır 3

Kalıtım kavramı, çok biçimlilik kavramının ortaya çıkmasına sebep olmaktadır. Kalıtım kavramı ile türetilen yeni sınıflardaki bazı farklılıklar çok biçimlilik kavramı sayesinde ifade edilebilmektedir. Ünite içerisinde verilen örnekten yola çıkacak olursak balık ve kuş isimli iki sınıfın canlı isimli bir üst sınıftan türetildiğini görmekteyiz. Ancak bu iki sınıf için hareket etme şekilleri farklı olduğundan çok biçimlilik kavramının kullanılması gerekmektedir. Bu sayede balık sınıfı bu eylemi yüzerek gerçekleştirirken kuş sınıfı yürüyerek veya uçarak gerçekleştirebilmektedir.

Kaynakça

Bryant, J. (2012). *Java 7 for Absolute Beginners*, Apress yayınları.

Sharan, K. (2014). *Beginning Java 8 Fundamentals: Language Syntax, Arrays, Data Types, Objects, and Regular Expressions*, Apress yayınları.

Ramnath, S., Dathan, B. (2011). *Object-Oriented Analysis and Design*, Springer-Verlag yayınları.

Bölüm 6

Java'da Soyut Sınıflar ve Arayüzler

Öğrenme çıktıları

1 Soyut Sınıflar

1 Soyut sınıf kavramını tanımlayabilme

2 Arayüzler

2 Arayüz kavramını açıklayabilme

Anahtar Sözcükler: • Sınıf • Soyut Sınıf • Nesne • Arayüz

GİRİŞ

Bu ünitede, soyut sınıf ve arayüz kavramları üzerinde duracağız. Ünitenin içeriğinde de görüleceği gibi bu iki başlığın aynı ünitede işlenmesinin sebebi kavramların birbirine benzer işlevleri olmasıdır. Bu kavramlar tanıtılırken birbirleri ile benzerlik ve farklılıkları da ortaya konulmuş olacaktır. Bu üniteye kadar sınıf ve nesne ilişkisini şu şekilde tarif ettik. Nesne, basit anlamda bir sınıfın gerçek hayattaki bir örneğidir. İnsan isimli bir sınıf tanımlandığında etrafımızdaki bireylerin onun bir örneği olduğunu söyleyebilmekteyiz. Ancak, nesneye yönelik programlamada her bir sınıfın bir örneği olmayabilir ve dolayısıyla o sınıf türünde nesneler oluşturamayabiliriz. Ana hatlarıyla soyut sınıf kavramı bu duruma işaret etmektedir. Herhangi bir sınıf, soyut olarak tanımlandığında o sınıf türünde nesne oluşturulabilmesi söz konusu olmayacaktır. Soyut sınıflar daha çok ilgili alt sınıflara ait detayların tanımlanması amacıyla kullanılırlar. Dolayısıyla soyut sınıflar, içeriğinde birtakım özelliklerin ve metotların tanımlanabildiği fakat kendi türünde nesnelerin yaratılamadığı sınıflardır. Ünitenin devamında bahsedeceğimiz bir diğer kavram da arayüz kavramıdır. Arayüz, bir sınıfta olması gereken metotları tanımlamaya yarayan bir yapıdır; fakat kesinlikle bir sınıf değildir. Amacı daha çok bir işlemin gerçekleşmesi için hangi metotların var olması gerektiğini belirlemektir. Arayüzlerin bu metotların içeriklerini tanımlamak gibi bir görevi yoktur. Bir arayüz, daha sonra herhangi bir sınıf tarafından uygulandığında bu metotların içerikleri ilgili sınıf tarafından tanımlanacaktır. Örneğin büyüklükleri karşılaştırma amacıyla bir arayüz tanımlandığında bunun farklı sınıflar üzerindeki uygulamaları da farklı olacaktır. Bu arayüzü uygulayan bir sınıf tarihlerin birbiriyle karşılaştırılmasını yaparken bir başka sınıf miktarların birbiriyle karşılaştırmasını yapabilir. Arayüz içerisinde sadece bu metodun ismi, hangi parametreleri alacağı ve ilgili metodun hangi tipte değer döndüreceği gibi bilgiler yer alacaktır. Java standart kütüphanesi içerisinde de programcıların uygulayabilecekleri bir takım hazır arayüzler bulunmaktadır.

SOYUT SINIFLAR

Soyut sınıf, giriş bölümünde de bahsedildiği gibi temel olarak kendisi türünde bir nesne oluşturulamayacak olan bir sınıftır. Nesneye yönelik programlamada sınıflar arasındaki kalıtım ilişkileri bir hiyerarşi şeklinde ifade edilebilir. Önceki ünitelerde bir sınıftan kalıtım yoluyla yeni bir sınıf türetildiğinde bu yeni sınıfın alt sınıf ve mevcut sınıfın üst sınıf olarak adlandırıldığından bahsetmiştik. Bu hiyerarşinin en üzerinde yer alan üst sınıflar kimi zaman sadece alt sınıflara ait işlemlerin tanımlanması açısından kullanılabilir. Böyle bir durumda bu üst sınıfın türünden nesne oluşturulması düşünülmemekte ve bunu ifade etmek için de bu sınıfın soyut olarak tanımlanması mümkün olmaktadır. Soyut sınıflarda bazı metotların içeriği tanımlamış fakat bazılarının içeriğinin tanımlanması ilgili alt sınıflara bırakılmıştır. Soyut sınıf, Java programlama dilinde **abstract** anahtar kelimesi ile tanımlanmaktadır. Aşağıda, soyut sınıf kullanımına temel bir örnek olarak *Sekil* isimli Java sınıfı verilmiştir. *Sekil* sınıfı tanımlanırken **abstract** anahtar kelimesinin kullanıldığını görmekteyiz. Bu sınıf, *alanHesapla* ve *alanGoster* isimli iki adet metoda sahiptir. Örneği dikkatle incelediğimiz zaman *alanHesapla* metodunun başında da **abstract** anahtar kelimesinin bulunduğunu gözleyebiliriz. Buna karşın *alanGoster* metodunun içerisinde böyle bir anahtar kelime bulunmamaktadır. Bir metodun önünde **abstract** anahtar kelimesinin bulunması o metodun içeriğinin belirlenmesinin kendisinden kalıtım yoluyla türeyecek olan yeni sınıflara bırakıldığı anlamına gelmektedir. Böyle metotlar, **soyut metot** olarak adlandırılmaktadır. Sınıf içerisinde yer alan bir diğer metot olan *alanGoster* ise içerik olarak tanımlanmış durumda olduğundan böyle bir anahtar kelimeye ihtiyaç bulunmamaktadır. Bu örnekte, *Sekil* sınıfından türeyecek olan *Daire* ve *Dikdörtgen* gibi sınıfların alan hesaplama yöntemlerinin farklı olacağı düşünülmüş ve bu sebeple ilgili metodun içeriğinin belirlenmesi bu sınıftan türetilecek diğer sınıflara bırakılmıştır.

Soyut metot: Bu metotlar, tanımları itibariyle sadece soyut sınıflar içerisinde bulunurlar fakat soyut sınıflarda mutlaka soyut metot tanımlanma zorunluluğu yoktur.

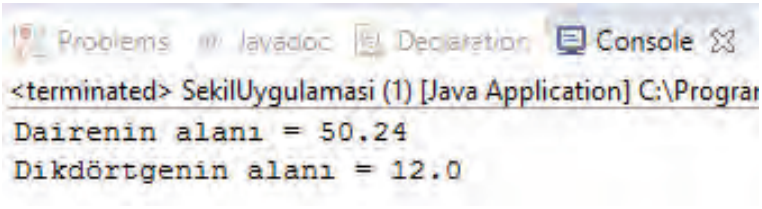
```
/* Sekil.java */

public abstract class Sekil {
  double alan;
  public abstract void alanHesapla();
  public double alanGoster() {
     return alan;
     }
}
```

Aşağıda, Sekil soyut sınıfından kalıtım yoluyla türetilen Daire ve Dikdörtgen sınıflarına ait Java kodları bulunmaktadır. Bu sınıflar içerisinde Sekil sınıfında yer alan soyut metotların içeriklerinin tanımlanması gerekmektedir. İlgili Java program kodları içerisinde bu metodun her iki sınıf için de farklı içeriklerle tanımlandığını görmekteyiz. Bir diğer metot olan alanGoster metodu ise kalıtım yoluyla Daire ve Dikdörtgen sınıflarına aktarılmaktadır. Dolayısıyla alanGoster metodunun herhangi bir yeni tanımlama yapılmadan bu sınıf türündeki nesneler tarafından çağrılabilmesi mümkün olacaktır.

```
/* Daire.java */

public class Daire extends Sekil{
  double yaricap;
  double piSayisi = 3.14;
  public void alanHesapla() {
     alan = piSayisi * yaricap * yaricap;
  }
  public Daire(double yaricapDegeri) {
     yaricap = yaricapDegeri;
     }
}
/* Dikdortgen.java */

public class Dikdortgen extends Sekil{
  double uzunluk, genislik;
  public void alanHesapla() {
     alan = uzunluk * genislik;
  }
  public Dikdortgen(double uzunlukDegeri, double genislikDegeri)
{
     uzunluk = uzunlukDegeri;
     genislik = genislikDegeri;
  }
}
```

Bu sınıfların tamamının kullanımını bir uygulama içerisinde görebilmek için ise aşağıda Java kodlarını içeren SekilUygulamasi sınıfı kullanılmıştır. SekilUygulamasi sınıfı içerisinde Daire ve Dikdörtgen türündeki nesneler oluşturulmuş ve bu nesnelere ait metotlar çağrılmıştır. Sonuç olarak iki nesne için de alanHesapla ve alanGoster metotlarının çalıştırıldığını görmekteyiz. Bu metotlardan birisi soyut sınıf olan Sekil sınıfı içerisinde, diğeri de bu sınıftan kalıtım yoluyla türetilmiş olan Daire ve Dikdörtgen sınıfları içerisinde tanımlanmıştır. SekilUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 6.1'deki ekran görüntüsü ortaya çıkmaktadır.

```
/* SekilUygulamasi.java */

public class SekilUygulamasi {
  public static void main(String[] args) {
        Daire daire = new Daire(4.0);
        Dikdortgen dortgen = new Dikdortgen(3.0, 4.0);
        daire.alanHesapla();
        dortgen.alanHesapla();
        System.out.println("Dairenin alanı =
"+daire.alanGoster());
        System.out.println("Dikdörtgenin alanı =
"+dortgen.alanGoster());
  }
}
```



Resim 6.1 SekilUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

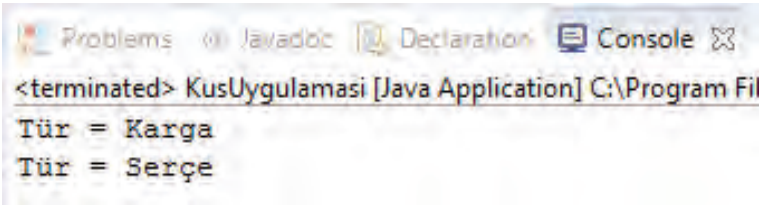
Aşağıda, soyut sınıf kavramı ile ilgili bir örnek daha bulunmaktadır. Bu örnekte soyut sınıf olarak tanımlanmış Kus isimli bir sınıf ve bu sınıftan kalıtım yoluyla türetilmiş olan Karga ve Serce sınıfları yer almaktadır.

```
/* Kus.java */

public abstract class Kus {
    public abstract void turGoster();
}
/* Karga.java */

public class Karga extends Kus {
  public void turGoster() {
        System.out.println("Tür = Karga");
  }
}
/* Serce.java */

public class Serce extends Kus {
  public void turGoster() {
        System.out.println("Tür = Serçe");
  }
}
```

Bu sınıfların tamamının kullanımını bir uygulama içerisinde görebilmek için ise aşağıda KusUygulamasi isimli bir sınıf bulunmaktadır. KusUygulamasi sınıfı içerisinde Karga ve Serce türündeki nesneler oluşturulmuş ve bu nesnelere ait turGoster metodu çağrılmıştır. Bu metot, Kus sınıfı içerisinde bir soyut metot olarak tanımlanmıştır. Bu sebepten dolayı, Karga ve Serce sınıfları içerisinde bu soyut metodun içeriğinin tanımlandığını görmekteyiz. Ayrıca, KusUygulamasi sınıfı içerisindeki son iki satırın yorum satırı şeklinde olduğunu görmekteyiz. Bu satırlar soyut sınıf olan Kus sınıfı türünde bir nesne oluşturulamayacağını göstermek amaçlıdır. Eğer bu iki satır, yorum satırı olmaktan çıkarılıp aktif hâle getirilirse derleyici hatası ile karşılaşılacak ve program çalıştırılamayacaktır.

```
/* KusUygulamasi.java */

public class KusUygulamasi {
    public static void main(String[] args) {
          Karga karga = new Karga();
          Serce serce = new Serce();
          karga.turGoster();
          serce.turGoster();
          // Kus kus = new Kus();
          // kus.turGoster();
    }
}
```

KusUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 6.2'deki ekran görüntüsü ortaya çıkmaktadır.



Resim 6.2 KusUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

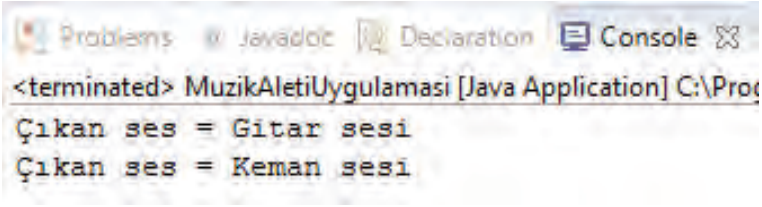
Aşağıda, soyut sınıf kavramı ile ilgili son bir örnek daha bulunmaktadır. Bu örnekte soyut sınıf olarak tanımlanmış MuzikAleti isimli bir sınıf ve bu sınıftan kalıtım yoluyla türetilmiş olan Gitar ve Keman sınıfları yer almaktadır.

```
/* MuzikAleti.java */

public abstract class MuzikAleti {
    public abstract void sesCikar();
}
/* Gitar.java */

public class Gitar extends MuzikAleti {
     public void sesCikar() {
         System.out.println("Çıkan ses = Gitar sesi");          }
}
/* Keman.java */

public class Keman extends MuzikAleti {
     public void sesCikar() {
         System.out.println("Çıkan ses = Keman sesi");
     }
}
```

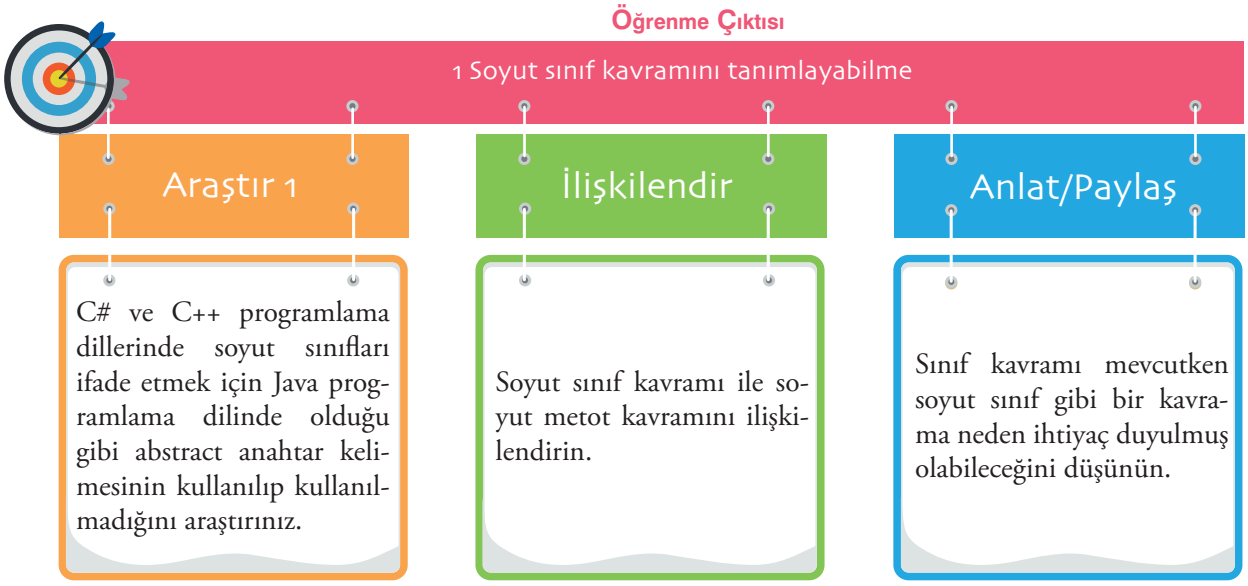
Bu sınıfların tamamının kullanımını bir uygulama içerisinde görebilmek için ise aşağıdaki MuzikAletiUygulamasi isimli sınıfı görmekteyiz. MuzikAletiUygulamasi sınıfı içerisinde Gitar ve Keman türündeki nesneler oluşturulmuş ve bu nesnelere ait sesCikar metodu çağrılmıştır. Bu metot, MuzikAleti sınıfı içerisinde bir soyut metot olarak tanımlanmıştır. Bu sebepten dolayı, Gitar ve Keman sınıfları içerisinde bu soyut metodun içeriğinin tanımlandığını görmekteyiz.

```
/* MuzikAletiUygulamasi.java */

public class MuzikAletiUygulamasi {
     public static void main(String[] args) {
            Gitar gitar = new Gitar();
            Keman keman = new Keman();
            gitar.sesCikar();
            keman.sesCikar();
     }
}
```

MuzikAletiUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 6.3'teki ekran görüntüsü ortaya çıkmaktadır.



Resim 6.3 MuzikAletiUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı



ARAYÜZLER

Arayüz, bir sınıfta olması gereken metotların tanımlandığı bir yapıdır fakat kesinlikle bir sınıf değildir. Arayüzlerin amacı daha çok herhangi bir işlemin gerçekleşmesi için hangi metotların var olması gerektiğini belirlemektir. Bu metotların içerikleri ise arayüzleri uygulayan sınıflar tarafından oluşturulur. Arayüzler, Java programlama dilinde **interface** anahtar kelimesi ise tanımlanır. Örneğin aşağıda Bilgilendir isminde bir arayüz yer almaktadır. Bu arayüz içerisinde turGoster isminde bir metot tanımlanmıştır. Metodun içeriği belirli olmamakla birlikte herhangi bir değer döndürmeyeceği ve parametre almadığı görülmektedir.

```
/* Bilgilendir.java */

public interface Bilgilendir {
     public void turGoster();
}
```

Aşağıda, Bilgilendir ismindeki arayüzü uygulayan iki adet sınıf ve bu sınıfları kullanan main metoduna sahip KusUygulamasi isimli bir sınıf yer almaktadır. İlgili sınıfları, soyut sınıf kavramına örnekler verirken de kullanmaktaydık. Bu örnek, aynı zamanda soyut sınıf kullanıldığı durumdaki işlevlerin arayüz kavramı ile nasıl gerçekleştirilebileceğinin anlaşılması açısından önemlidir.

```
/* Serce.java */

public class Serce implements Bilgilendir {
     public void turGoster() {
           System.out.println("Tür = Serçe");
     }
}
/* Karga.java */

public class Karga implements Bilgilendir {
     public void turGoster() {
        System.out.println("Tür = Karga");          }
}
/* KusUygulamasi.java */

public class KusUygulamasi {
     public static void main(String[] args) {
           Karga karga = new Karga();
           Serce serce = new Serce();
           karga.turGoster();
           serce.turGoster();
     }
}
```

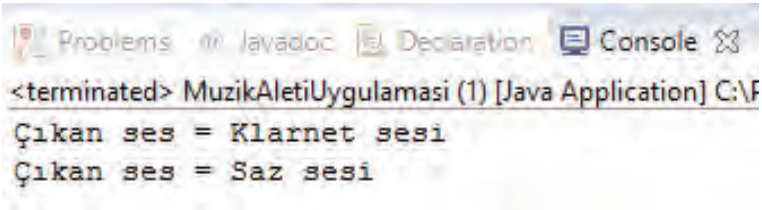
KusUygulamasi sınıfına ait program kodları çalıştırıldığında bir önceki başlıkta yer verdiğimiz Resim 6.2'deki ekran görüntüsünün aynısı ortaya çıkmaktadır. Bu sebeple, programın ekran çıktısı burada tekrar gösterilmemiştir. Aşağıda, bu konu ile ilgili bir örnek daha bulunmaktadır. Bu örnekte MuzikBilgisi isminde bir arayüz yer almaktadır. Bu arayüz içerisinde sesCikar isminde bir metot tanımlanmıştır. Metodun içeriği belirli olmamakla birlikte herhangi bir değer döndürmeyeceği ve parametre almadığı görülmektedir.

```
/* MuzikBilgisi.java */

public interface MuzikBilgisi {
     public void sesCikar();
}
```

Aşağıda, *MuzikBilgisi* ismindeki arayüzü uygulayan *Klarnet* ve *Saz* isimli iki adet sınıf bulunmaktadır. Ek olarak, bu sınıfların kullanımını bir uygulama içerisinde göstermek için *MuzikAletiUygulamasi* isimli bir sınıf mevcuttur.

```
/* Klarnet.java */

public class Klarnet implements MuzikBilgisi {
     public void sesCikar() {
         System.out.println("Çıkan ses = Klarnet sesi");
     }
}
/* Saz.java */

public class Saz implements MuzikBilgisi {
     public void sesCikar() {
         System.out.println("Çıkan ses = Saz sesi");
     }
}
/* MuzikAletiUygulamasi.java */

public class MuzikAletiUygulamasi {
     public static void main(String[] args) {
           Klarnet klarnet = new Klarnet();
           Saz saz = new Saz();
           klarnet.sesCikar();
           saz.sesCikar();
     }
}
```

MuzikAletiUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 6.4'teki ekran görüntüsü ortaya çıkmaktadır. Ekran görüntüsünden de anlaşılacağı gibi her iki sınıf türündeki nesnelerin farklı çıktılar ürettiği görülmektedir. Bunun sebebi *sesCikar* metodunun iki farklı sınıf içerisindeki uygulamalarının farklı olmasıdır.



Resim 6.4 MuzikAletiUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

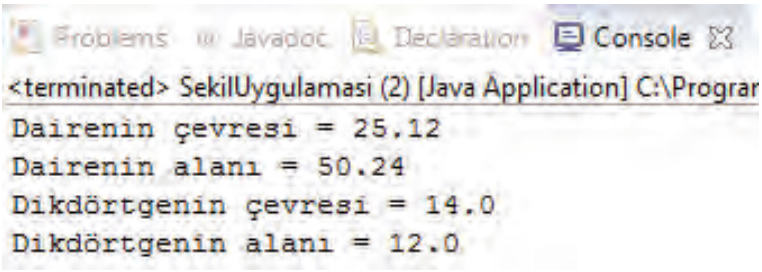
Aşağıda, arayüz kavramına bir örnek daha bulunmaktadır. Bu örnekte *SekilHakkindaBilgi* isminde bir arayüz ve bu arayüzü uygulayan sınıflar yer almaktadır. *SekilHakkindaBilgi* arayüzü, çevre ve alan değerlerinin gösterilmesi amacı taşıyan iki adet metoda sahiptir. Bu arayüzü uygulayan herhangi bir sınıfın çevre ve alan değerlerini hesaplayarak görüntülemesi gerekmektedir. Çevre ve alan değerlerinin hesaplanma şekli her bir geometrik şekil için farklı olacağından bu metotların içeriklerinin detayları arayüzü uygulayacak sınıflara bırakılmıştır. Zaten arayüz kavramı ile böyle bir içerik tanımlamasının mümkün olmadığını daha önce belirtmiştik.

```
/* SekilHakkindaBilgi.java */

public interface SekilHakkindaBilgi {
     public double cevreGoster();
     public double alanGoster();
}
```

Aşağıda kodları verilen *Daire* ve *Dikdortgen* isimli sınıflar *SekilHakkindaBilgi* arayüzünü uygulayan sınıflardır. Bu sınıfların içerisinde, ilgili metotların içeriklerinin tanımlandığını görmekteyiz. Daire ve dikdörtgenin çevre ve alan hesapları farklı şekillerde yapıldığı için bu metotların içerikleri farklıdır. Bu sınıfların tamamının kullanımını bir uygulama içerisinde görebilmek için ise aşağıdaki Java kodlarını içeren *SekilUygulamasi* isimli sınıf kullanılmaktadır. *SekilUygulamasi* sınıfı içerisinde *Daire* ve *Dikdortgen* türündeki nesneler oluşturulmuştur. Daha sonra bu nesnelere ait *cevreGoster* ve *alanGoster* metotları çağrılmıştır.

```
/* Daire.java */

public class Daire implements SekilHakkindaBilgi {
     double yaricap, sonuc;
     double piSayisi = 3.14;
     public double cevreGoster() {
           sonuc = 2 * piSayisi * yaricap;
           return sonuc;
     }
     public double alanGoster() {
           sonuc = piSayisi * yaricap * yaricap;
           return sonuc;
     }
     public Daire(double yaricapDegeri) {
           yaricap = yaricapDegeri;
     }
}
/* Dikdortgen.java */

public class Dikdortgen implements SekilHakkindaBilgi{
     double uzunluk, genislik, sonuc;
     public double cevreGoster() {
           sonuc = 2 * (uzunluk + genislik);
           return sonuc;
     }
     public double alanGoster() {
           sonuc = uzunluk * genislik;
           return sonuc;
     }
     public Dikdortgen(double uzunlukDegeri, double genislikDegeri) {
           uzunluk = uzunlukDegeri;
           genislik = genislikDegeri;
     }
}
/* SekilUygulamasi.java */

public class SekilUygulamasi {
     public static void main(String[] args) {
        Daire daire = new Daire(4.0);
        Dikdortgen dortgen = new Dikdortgen(3.0, 4.0);
        System.out.println("Dairenin çevresi = "+daire.cevreGoster());
        System.out.println("Dairenin alanı = "+daire.alanGoster());
        System.out.println("Dikdörtgenin çevresi = "+dortgen.cevreGoster());
        System.out.println("Dikdörtgenin alanı = "+dortgen.alanGoster());
     }
}
```

SekilUygulamasi sınıfına ait program kodları çalıştırıldığında Resim 6.5'teki ekran görüntüsü ortaya çıkmaktadır.



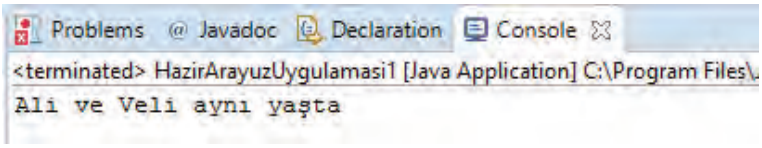
Resim 6.5 SekilUygulamasi sınıfı çalıştırılınca verdiği ekran çıktısı

Arayüzler, programcılar tarafından gerekli durumlarda oluşturulabilir. Bununla birlikte, Java standart kütüphanesi içerisinde de kullanılabilir hazır arayüzler mevcuttur. Aşağıda kodları verilen *Insan* sınıfı *Comparable* adlı Java kütüphanesi arayüzünü uygulayan bir sınıftır. Bu sınıfın içerisinde, *compareTo* isimli metodun içeriğinin tanımlandığını görmekteyiz. Bu metot, *Comparable* arayüzü içerisinde tanımlı olan ve **int** türünde değer döndürmesi beklenen bir metottur. Karşılaştırılan iki nesnenin birbirinden büyük veya küçük olması durumunda bu metodun 1 veya -1 döndürmesi beklenir. Eşitlik durumunda ise 0 değerini döndürmesi beklenmektedir. *Insan* sınıfının kullanımını bir uygulama içerisinde görebilmek için ise aşağıdaki Java kodlarını içeren *HazirArayuzUygulamasi1* isimli sınıf kullanılmaktadır. *HazirArayuzUygulamasi1* sınıfı içerisinde *Insan* türünde nesneler oluşturulmuştur. Daha sonra, bu nesnelere ait *compareTo* metodu kullanılarak insanların birbiriyle yaş bazında kıyaslamaları yapılmıştır.

```
/* Insan.java */

public class Insan implements Comparable {
    private String isim;
    private int yas;
    public Insan(String isim, int yas) {
        this.isim = isim;
        this.yas = yas;
    }
    public int yasGetir() {
        return this.yas;
    }
    public String isimGetir() {
        return this.isim;
    }
    public int compareTo(Object deger) {
     int sonuc = 0;
     Insan insan2 = (Insan) deger;
        if(this.yas == insan2.yas)
           sonuc = 0;
        else if(this.yas > insan2.yas)
           sonuc = 1;
        else if(this.yas < insan2.yas)
           sonuc = -1;

           return sonuc;
    }
}
```

```
/* HazirArayuzUygulamasi1.java */

public class HazirArayuzUygulamasi1 {
     public static void main(String[] args) {
            Insan insan1 = new Insan("Ali", 30);
            Insan insan2 = new Insan("Veli", 30);

            if (insan1.compareTo(insan2)==0)
                  System.out.println(insan1.isimGetir() + " ve " +
insan2.isimGetir() + " aynı yaşta");
            else if (insan1.compareTo(insan2)>0)
                  System.out.println(insan1.isimGetir() + ", " +
insan2.isimGetir() + "'den büyük");
            else if (insan1.compareTo(insan2)<0)
                  System.out.println(insan1.isimGetir() + ", " +
insan2.isimGetir() + "'den küçük");
     }
}
```

HazirArayuzUygulamasi1 sınıfına ait program kodları çalıştırıldığında Resim 6.6'daki ekran görüntüsü ortaya çıkmaktadır.



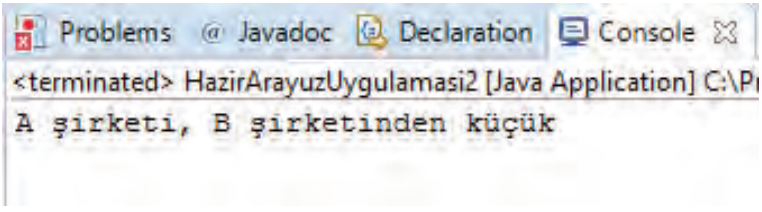
Resim 6.6 HazirArayuzUygulamasi1 sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, Java kütüphaneleri içerisindeki hazır arayüzlerin kullanımına bir örnek daha bulunmaktadır. Program kodları verilen *Sirket* sınıfı, bir önceki örnekteki gibi *Comparable* adlı Java kütüphanesi arayüzünü uygulayan bir sınıftır. Bu sınıfın içerisinde de, *compareTo* isimli metodun içeriğinin tanımlandığını görmekteyiz. *Sirket* sınıfının kullanımını bir uygulama içerisinde görebilmek için ise aşağıdaki Java kodlarını içeren *HazirArayuzUygulamasi2* isimli sınıf kullanılmaktadır. *HazirArayuzUygulamasi2* sınıfı içerisinde *Sirket* türündeki nesneler oluşturulmuştur. Daha sonra, bu nesnelere ait *compareTo* metodu kullanılarak şirketlerin büyüklüklerinin birbiriyle kıyaslamaları yapılmıştır. Kıyaslama ölçütü olarak şirketlerin çalışan sayıları kullanılmaktadır.

```
/* Sirket.java */

public class Sirket implements Comparable {
    private int calisanSayisi;
    private String isim;
    public Sirket (String isim, int calisanSayisi) {
        this.isim = isim;
        this.calisanSayisi = calisanSayisi;
    }
    public int calisanSayisiGetir() {
        return this.calisanSayisi;
    }
    public String isimGetir() {
        return this.isim;
    }
     public int compareTo(Object deger) {
     int sonuc = 0;
     Sirket sirket2 = (Sirket) deger;
        if(this.calisanSayisiGetir() ==
sirket2.calisanSayisiGetir())
            sonuc = 0;
        else if(this.calisanSayisiGetir() >
sirket2.calisanSayisiGetir())
            sonuc = 1;
        else if(this.calisanSayisiGetir() <
sirket2.calisanSayisiGetir())
            sonuc = -1;

            return sonuc;
    }
}
```

```
/* HazirArayuzUygulamasi2.java */

public class HazirArayuzUygulamasi2 {
     public static void main(String[] args) {
            Sirket sirket3 = new Sirket("A şirketi", 30);
            Sirket sirket4 = new Sirket("B şirketi", 40);
            if (sirket3.compareTo(sirket4) == 0)
                  System.out.println(sirket3.isimGetir() + " ve " +
sirket4.isimGetir()+ " aynı büyüklükte");
            else if (sirket3.compareTo(sirket4)>0)
                  System.out.println(sirket3.isimGetir() + ", " +
sirket4.isimGetir()+ "nden büyük");
            else if (sirket3.compareTo(sirket4)<0)
                  System.out.println(sirket3.isimGetir() + ", " +
sirket4.isimGetir()+ "nden küçük");
      }
 }
```

HazirArayuzUygulamasi2 sınıfına ait program kodları çalıştırıldığında Resim 6.7'deki ekran görüntüsü ortaya çıkmaktadır.

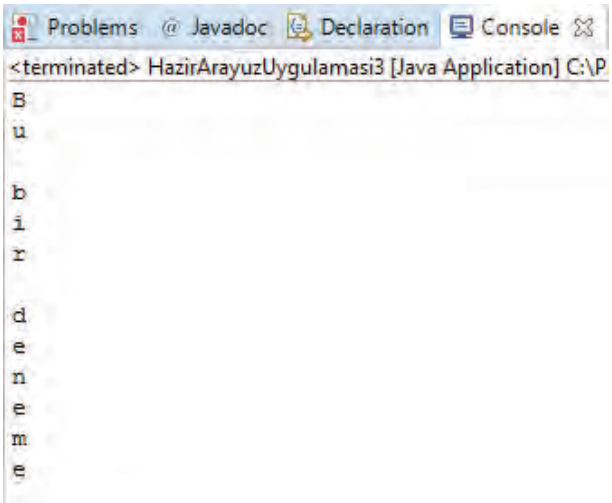


Resim 6.7 HazirArayuzUygulamasi2 sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, Java kütüphaneleri içerisindeki hazır arayüzlerin kullanımına farklı bir örnek daha bulunmaktadır. Kodları verilen *OzellesmisMetin* sınıfı *Iterable* ve *Iterator* adlı Java kütüphanesi arayüzlerini uygulayan bir sınıftır. Bu sınıfın içerisinde, bu iki arayüze ait *hasNext*, *next* ve *iterator* isimli metotların içeriklerinin tanımlandığı görülmektedir. *hasNext* metodu, metnin içerisindeki karakterler üzerinde ilerlenirken metnin sonuna gelinip gelinmediğini algılamak için kullanılmaktadır. Metnin sonuna gelinmediği sürece **true** değeri döndürecektir. Sonuna gelindiğinde ise **false** değerini döndürecektir. *next* metodu, metnin içerisindeki bir sonraki karaktere erişmek için kullanılmaktadır. *iterator* metodunun, bir *Iterator* arayüzü döndürmesi beklenmektedir. Bu uygulamanın amacı, bir döngü kullanılması vasıtasıyla içeriğindeki karakterlerin sırasıyla ayrı ayrı görüntülenebildiği bir metin sınıfı oluşturmaktır. Arayüzler, bu işlevi sağlayan bir metin sınıfının oluşturulması için kullanılmıştır. *OzellesmisMetin* sınıfının kullanımını bir uygulama içerisinde görebilmek için ise aşağıdaki Java kodlarını içeren *HazirArayuzUygulamasi3* isimli sınıf kullanılmaktadır. *HazirArayuzUygulamasi3* sınıfı içerisinde *OzellesmisMetin* türündeki nesneler oluşturulmaktadır.

```
/* OzellesmisMetin.java */

import java.util.Iterator;
public class OzellesmisMetin implements Iterable, Iterator {
  private String metin;
  private int sayac = 0;
  public OzellesmisMetin (String s) {
       metin = s;
  }
  public boolean hasNext() {
       if (sayac < metin.length()){
             return true;
             }
       else {
             return false;
       }
  }
  public Character next() {
    sayac++;
    return metin.charAt(sayac - 1);
  }
  public Iterator iterator() {
    return this;
  }
}
/* HazirArayuzUygulamasi3.java */

public class HazirArayuzUygulamasi3 {
     public static void main(String[] args) {
           OzellesmisMetin x = new OzellesmisMetin ("Bu bir deneme");
         for (Object ch : x){
           ch = (char) ch;
           System.out.println(ch);
         }
     }
}
```

HazirArayuzUygulamasi3 sınıfına ait program kodları çalıştırıldığında Resim 6.8'deki ekran görüntüsü ortaya çıkmaktadır.



Resim 6.8 HazirArayuzUygulamasi3 sınıfı çalıştırılınca verdiği ekran çıktısı

Yaşamla İlişkilendir

Gel sen de geliştir!

Şirketlerin elindeki verileri yazılımcılara açarak yeni ürünlerin çıkmasını sağladıkları API (Uygulama Programlama Arayüzleri) yeni ekonominin en önemli trendleri arasında. Dünyada toplamda 17 bin civarında API bulunurken, Türk şirketlerin de ilgisi her geçen artıyor.

Bir internet sitesine girdiniz. Üye olmak yerine karşınıza çıkan "Facebook ile bağlan" tuşuna bastınız. Her ne kadar basit bir tuşa basmış olsanız da işin arka planında önemli bir konu var. Bu konu API. Yani Uygulama Programlama Arayüzleri (Application Programing Interfaces). Basit bir şekilde anlatacak olursak, şirketler, elinde tuttuğu ve istedikleri verileri yazılımcılarla paylaşıyorlar. Paylaşılan bu veriler üzerine yazılımcılar, şirketlerin de işine yarayacağı bir servis veya ürün geliştiriyor. Kısacası şirketler, kendi hizmetlerini ve servislerini kullanmak isteyen şirketlerle daha kolay entegre oluyor ve "gel sen de geliştir" diyorlar.

Peki bir şirketin API'larını diğer yazılımcılara açması ne anlama geliyor? En önemli sebeplerin başında şirket içindeki yazılımcıların işi kolaylaştırması geliyor. Şirketler, üçüncü parti uygulamalarla entegrasyon için kendi ekiplerini organize etmiyorlar. Bu işi dışarıdan çözebiliyorlar. API'ların diğer bir faydası ise şirketlerin daha fonksiyonel projelere imza atması geliyor. API'ların şirketlere diğer faydaları ise şunlar; kullanıcılara ve müşterilere kolaylık sağlanabiliyor, entegre projeler geliştirerek daha başarılı imza atılabiliyor.

Şirketlerin API izni vermesi yazılım geliştiriciler ve girişimciler tarafından farklı ürünleri çıkmasını sağlıyor. Birçok girişim büyük şirketlerin API'lerinden ilham kaynağı alarak, yep yeni bir ekosistem oluşturmayı başardı. Hatta bu öyle bir boyuta geldi ki, şu anda dünyada toplamda 17 binden fazla API bulunuyor. Türkiye'deki şirketler de hızla API dünyasına adım atmaya başladı. Sektör yetkililerinden aldığımız bilgilere göre şu anda yaklaşık 30 Türk şirketi API'larını yazılımcılara açmış durumda. Bu şirketlerin arasında bankacılık ve finans başta olmak üzere birçok farklı sektör bulunuyor.

Kaynak: Bu habere http://www.hurriyet.com.tr/gel-sen-de-gelistir-40529682 internet adresinden ulaşabilirsiniz.

Öğrenme Çıktısı

2 Arayüz kavramını açıklayabilme

Araştır 2	İlişkilendir	Anlat/Paylaş
Arayüzler ve soyut sınıflar arasındaki temel farkları araştırarak hangi durumda arayüzün tercih edilmesi gerektiğini belirtiniz.	Arayüz kavramı ile sınıf kavramını ilişkilendirin.	Arayüz kavramı mevcutken soyut sınıf kavramına neden ihtiyaç duyulmuş olabileceğini düşünün.

öğrenme çıktıları ve bölüm özeti

1 Soyut sınıf kavramını tanımlayabilme

Soyut Sınıflar

Kitabın bu bölümünde nesneye yönelik programlama ile ilgili soyut sınıf ve arayüz kavramları ele alınmıştır. Bu iki kavram birbirinden farklı olmakla birlikte benzer işlevler gerçekleştirmek amacıyla kullanılmaktadır. Soyut sınıf kavramı, daha önce öğrendiğimiz sınıf kavramı ile farklılıklar içeren bir kavramdır. Sınıf, kendisi türünde nesneler yaratılmasına imkân veren bir yapıdır. Bunu, basit anlamda etrafımızdaki bireylerin tamamının insan sınıfının birer örnekleri olduğu şeklinde açıklayabilmekteyiz. Sınıflar içerisinde, metotlar ve özelliklerin tanımlanması vasıtasıyla nesnelerin gerçekleştirebilecekleri eylemleri tarif edebiliriz. Sınıflar, bir anlamda ilgili nesnelerin kabiliyetlerini gösteren birer taslak niteliğindedir. Soyut sınıflarda ise nesnelerin gerçekleştirebilecekleri eylemlerin bir kısmı tanımlanmış durumdadır. Bir kısım eylemlerin ise sadece isimleri tanımlanmış durumda olup içerik tanımlamaları yapılmamıştır. Sınıf ve soyut sınıf arasındaki en temel fark soyut sınıflar türünde nesne oluşturulamamasıdır. Bunun sebebi ise bu sınıfların içeriklerinin tamamlanmamış durumda olmasıdır. Soyut sınıf, Java programlama dilinde abstract anahtar kelimesi ile tanımlanmaktadır.

2 Arayüz kavramını açıklayabilme

Arayüzler

Ünitede son olarak arayüz kavramı ele alınmıştır. Arayüzler, sınıflarda olması gereken metotların tanımlandığı yapılardır. Arayüzler, Java programlama dilinde interface anahtar kelimesi ile tanımlanır. Arayüzler içerisinde, alınan parametrelerin ve döndürülen değer tiplerinin belirtildiği bir takım metotlar yer almaktadır. Bu metotların içeriklerinin tanımları arayüzü uygulayan sınıflar tarafından yapılmaktadır. Java standart kütüphanesi içerisinde tanımlanmış olan ve programcılar tarafından uygulanabilen hazır arayüzler de mevcuttur. Bu tipteki arayüzlerin kullanımının öğrenilebilmesi için ünite içerisinde Comparable, Iterable ve Iterator arayüzleri üzerinden örnekler verilmiştir. Arayüzler, işlev olarak soyut sınıflar ile benzerlik taşıdıklarından birbiriyle karıştırılabilen kavramlardır. Soyut sınıflar ve arayüzler arasındaki en temel fark arayüzlerin kesinlikle sınıf olmamalarıdır. Arayüzlerin amacı daha çok bir işlemin gerçekleşmesi için gerekli olan metotların belirlenmesidir. Arayüzlerin bu metotların içeriklerini tanımlamak gibi bir görevi yoktur. Arayüzlerden farklı olarak, soyut sınıflar içerisindeki bazı metotların içerik tanımlamaları yapılmış olabilmektedir.

1 Aşağıdaki anahtar kelimelerin hangisi Java programlama dilinde sınıf tanımında kullanıldığında ilgili sınıf bir soyut sınıf olur?

A. public
B. private
C. protected
D. interface
E. abstract

2 Soyut sınıflar ile ilgili olarak aşağıdaki ifadelerden hangisi **yanlıştır?**

A. Soyut sınıflar içerisinde metotlar tanımlanabilir.
B. Soyut sınıflar içerisinde özellikler tanımlanabilir.
C. Soyut sınıflar, kendi türünde nesneler oluşturulabilen sınıflardır.
D. Soyut sınıflardaki metotların tamamı soyut metot olarak tanımlanmayabilir.
E. Soyut sınıflar, başka sınıflar tarafından kalıtım yoluyla miras alınabilirler.

3 Java programlama dilinde arayüzleri tanımlayan anahtar kelime aşağıdakilerden hangisidir?

A. abstract
B. interface
C. protected
D. public
E. private

4 Arayüzler ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. Arayüzler, kesinlikle birer sınıf değildirler.
B. Arayüzler, abstract anahtar kelimesi ile tanımlanırlar.
C. Arayüzler içerisindeki metotların tanımlarında abstract kelimesi bulunur.
D. Arayüzler, kalıtım yoluyla birbirlerinden türeyebilirler.
E. Arayüzler içerisinde en fazla bir metot bulunabilir.

5

```
public class X implements Y {
public void yaz( ) {
            System.out.println("deneme");
      }
}
```

Yukarıdaki Java program kodu ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. yaz, bir soyut metottur.
B. X, bir soyut sınıftır.
C. Y, bir soyut sınıftır.
D. Y, bir arayüzdür.
E. X, bir arayüzdür.

6

```
public abstract class Arac {
      int vitesSayisi;
      int anlikHiz;
      String renk;
      abstract boolean frenYap();
      abstract int vitesSayisiGoster();
      abstract void renkGoster();
      public void hizArtir(){
            anlikHiz = anlikHiz + 1;
      }
}
```

Yukarıdaki Java program kodu ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. *Arac* sınıfı, bir soyut sınıftır.
B. *Arac* sınıfı içerisindeki metotlarının tamamı soyut metot olarak tanımlanmıştır.
C. *frenYap* metodu, bir soyut metot değildir.
D. *renkGoster* metodu, bir soyut metot değildir.
E. *hizArtir* metodu, bir soyut metottur.

7

```
public abstract class A {
}

public class B extends A {
}

public class C extends B {
}
```

Yukarıdaki Java program kodu ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. *A*, bir soyut sınıftır.
B. *B*, bir soyut sınıftır.
C. *C*, bir soyut sınıftır.
D. *A* ve *B*, soyut sınıftırlar.
E. *B* ve *C*, soyut sınıftırlar.

8 Soyut metotlar ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. Soyut sınıfların içerisinde yer alan metotların tamamı soyut metot olmalıdır.
B. Soyut metotlar, abstract anahtar kelimesi kullanılarak tanımlanırlar.
C. Soyut metotların içeriklerinin tamamını içinde bulundukları soyut sınıf tanımlar.
D. Soyut metotlar, soyut olmayan sınıflarda da bulunabilirler.
E. Soyut metotlar, soyut sınıflarda kesinlikle tanımlanamazlar.

9

```
public abstract class D {
}

public class E extends D implements F{
}
```

Yukarıdaki Java program kodu ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. *E*, bir soyut sınıftır.
B. *E*, bir arayüzdür.
C. *F*, bir soyut sınıftır.
D. *F*, bir arayüzdür.
E. *D*, bir arayüzdür.

10

```
public interface Bilgi {
      public void turGoster();
}

public class G implements Bilgi{
}
```

Yukarıda, hatalı bir Java program kodu bulunmaktadır. Bu hatanın giderilmesi için aşağıdakilerden hangisi yapılmalıdır?

A. *G* sınıfı içerisinde *turGoster* metodu tanımlanmalıdır.
B. *G* sınıfının soyut sınıf olarak tanımlanmalıdır.
C. *Bilgi* isimli arayüz soyut sınıf olarak tanımlanmalıdır.
D. *Bilgi* isimli arayüz içerisinde *AA* isimli bir metot tanımlanması durumda
E. *G* sınıfı içerisinde *AA* isimli bir metot tanımlanmalıdır.

1. E	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
2. C	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
3. B	Yanıtınız yanlış ise "Arayüzler" konusunu yeniden gözden geçiriniz.
4. A	Yanıtınız yanlış ise "Arayüzler" konusunu yeniden gözden geçiriniz.
5. D	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
6. A	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
7. A	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
8. B	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
9. D	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.
10. A	Yanıtınız yanlış ise "Soyut Sınıflar" konusunu yeniden gözden geçiriniz.

6 Araştır Yanıt Anahtarı

Araştır 1

C# programlama dilinde soyut sınıflar tıpkı Java programlama dilinde olduğu gibi abstract anahtar kelimesi kullanılarak tanımlanmaktadır. Ancak C++ programlama dilinde bu anahtar kelime kullanılmamaktadır.

Araştır 2

Soyut sınıflar içerisinde bulunan bazı metotların içeriklerinin önceden tanımlanmış olma ihtimali mevcuttur. Ancak arayüzler içerisinde bulunan metotların içerikleri önceden tanımlanmış olamazlar. Metotların sadece şekilsel olarak ortak olmasını istiyor fakat uygulama olarak farklı olmasını istiyorsak soyut sınıf yerine arayüz kullanımının tercih edilmesi gerekir.

Kaynakça

Bryant, J. (2012). *Java 7 for Absolute Beginners*, Apress yayınları.

Sharan, K. (2014). *Beginning Java 8 Fundamentals: Language Syntax, Arrays, Data Types, Objects, and Regular Expressions*, Apress yayınları.

Ramnath, S., Dathan, B. (2011). *Object-Oriented Analysis and Design*, Springer-Verlag yayınları.

Bölüm 7

Java'da Kural Dışı Durum İşleme

öğrenme çıktıları

1 Kural Dışı Durumlar

1 Kural dışı durum kavramını tanımlayabilme

2 Kural Dışı Durum İşleme

2 Kural dışı durumların işlenmesini açıklayabilme

Anahtar Sözcükler: • Kural Dışı Durumlar • Try Catch Bloğu • Exception Sınıfı • Finally Bloğu • Throws İfadesi

GİRİŞ

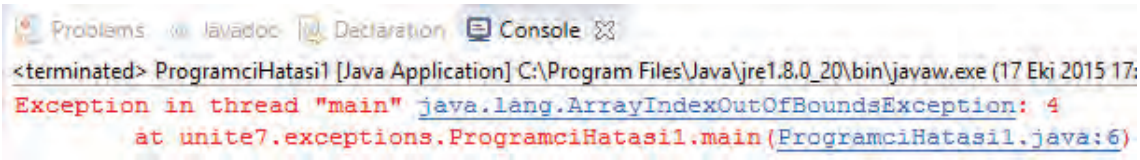
Bu ünitede, kural dışı durum işleme kavramı üzerinde duracağız. İlgili programlama dilinin yazım kurallarına uygun olarak yazdığımız ve çalıştırmak istediğimiz program kodları için genel olarak iki türlü problem ile karşılaşabiliriz. Bunlardan ilki programın derleyici hatası vermesidir. Diğeri ise programın çalışması sırasında hata vererek sonlanmasıdır. Her iki durumun sebebi de kural dışı durumların varlığıdır. Eğer derleyici hatası söz konusu ise kural dışı durumların mevcut olduğu ve bununla ilgili bir çözüm üretmemiz gerektiği şeklinde bir mesaj alırız. Diğer durumda ise programımız çalışma esnasında aniden sonlanacak ve kural dışı durumların oluştuğuna dair bir sistem mesajı ortaya çıkacaktır. Kural dışı durumlara şöyle bir örnek verilebilir: İlgili programın bir dosyadan veri okuması gerektiğini varsayalım. Bu dosya, bir şekilde zarar görmüş veya silinmiş olabilir. Bu durumda, bilgisayar programı dosyanın okunduğu satıra geldiğinde kural dışı durum ile karşılaşılmış olur. Bilgisayar programları yazılırken bu tip durumlar için de ayrıca önlemler alınması gerekir. Aksi durumda, programınız sebebi bilinmeyen bir hata ile sonlanmış olacaktır. Bu örnekte önlem alınması kısmını kural dışı durum işleme olarak adlandırmaktayız. Ünitenin devamında, ilk olarak kural dışı durum kavramı Java programlama dili çerçevesinde incelenecektir. Daha sonra, kural dışı durumların işleniş şekilleri çeşitli örneklerle ele alınacaktır.

KURAL DIŞI DURUMLAR

Kural dışı durumlar, programların çalışmaları esnasındaki normal akışlarını bozan şeyleri temsil etmektedir. Bu durumlarla karşılaşıldığında programlar beklenmedik bir anda sonlanabilirler. Bir kullanıcının hatalı veri girişi yapması, içeriğinden veri okunmak istenilen bir dosyanın bulunamaması, ağ üzerinden başka bir bilgisayar ile kurulacak bağlantıda beklenmedik sorunlar olması gibi hâller bu durumlara örnek olarak verilebilirler. Aşağıda, ProgramciHatasi1 isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf içerisinde, 3 elemanlı bir dizi tanımlanmış ve bu dizinin beşinci elemanı ekrana yazdırılmak istenmiştir. Dizinin beşinci elemanının var olmaması bir kural dışı durumun ortaya çıkmasına sebep olur.

```
/* ProgramciHatasi1.java */

public class ProgramciHatasi1 {
      public static void main(String[] args) {
       int sayi[]={1,2,3};
       System.out.println(sayi[4]);
      }
}
```

ProgramciHatasi1 sınıfına ait program kodları çalıştırıldığında Resim 7.1'deki ekran görüntüsü ortaya çıkmaktadır. Kural dışı durumun oluşması sonrasında, ekran görüntüsünde de görüldüğü üzere düz yazı şeklinde ve altı çizili olarak gösterilen bir takım hata mesajları ile karşılaşılmaktadır. Altı çizili alanlar, kural dışı durumların tipleri ile ilgili bilgi vermektedir. Bunlara ait bilgileri bu bölümün devamında detaylı olarak inceleyeceğiz.

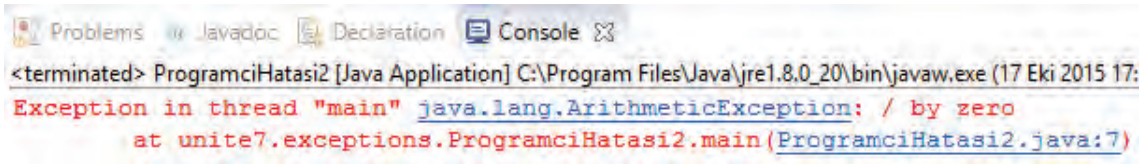


Resim 7.1 ProgramciHatasi1 sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, kural dışı durumlara bir başka örnek olarak ProgramciHatasi2 isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf içerisinde, iki değişken tanımlanarak bu değişkenlere sırasıyla 5 ve 0 sayıları atanmıştır. Daha sonra ise bu iki sayının bölümünün hesaplanması ve ekrana yazdırılması amaçlanmıştır. Ancak, temel matematik bilgilerimizi anımsarsak herhangi bir sayının 0'a bölünmesinin mümkün olmadığını görürüz. Bu olay, bir kural dışı durumun ortaya çıkmasına sebep olur.

```
/* ProgramciHatasi2.java */

public class ProgramciHatasi2 {
      public static void main(String[] args)
{
       int sayi1 = 5;
       int sayi2 = 0;
       int sonuc = sayi1 / sayi2;
       System.out.println(sonuc);
      }
}
```

ProgramciHatasi2 sınıfına ait program kodları çalıştırıldığında Resim 7.2'deki ekran görüntüsü ortaya çıkmaktadır.

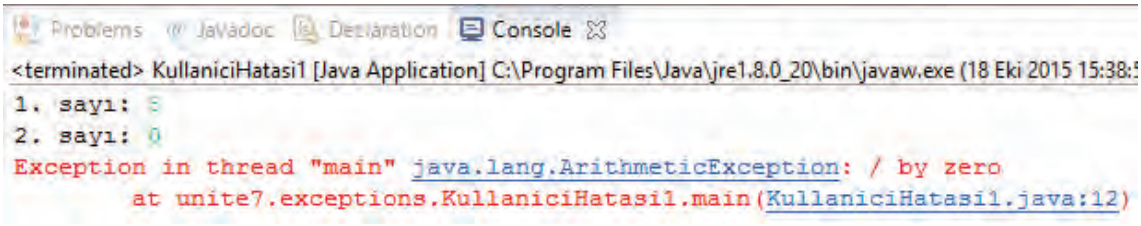


Resim 7.2 ProgramciHatasi2 sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, kural dışı durumlara bir başka örnek olarak *KullaniciHatasi1* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıfa ait kodlar çalıştırıldığında kullanıcıdan sırasıyla 2 adet değer girmesi istenilecek ve girilen bu değerlerin birbirine bölünmesi ile elde edilecek sonuç ekrana yazdırılacaktır.

```
/* KullaniciHatasi1.java */

import java.util.Scanner;
public class KullaniciHatasi1 {
      public static void main(String[] args) {
       int sayi1, sayi2;
       Scanner klavye = new Scanner(System.in);
       System.out.print("1. sayı:");
       sayi1 = klavye.nextInt();
       System.out.print("2. sayı:");
       sayi2 = klavye.nextInt();
       int sonuc = sayi1 / sayi2;
       System.out.println(sonuc);
      }
}
```

Program çalıştırıldıktan sonra kullanıcının sırasıyla 5 ve 0 değerlerini girdiği duruma ait ekran çıktıları Resim 7.3'teki gösterilmiştir. Burada da az önceki durumun aynısı ortaya çıkmakta ve kural dışı durumun gözlenmesi ile program sonlanmaktadır. Herhangi bir sayının 0'a bölünmesi mümkün olmadığı için bir kural dışı durum ortaya çıkmaktadır. Kullanıcı, bu durumun ortaya çıkmayacağı değerler girmemiş olsaydı programın çalışmasında herhangi bir sorun yaşanmayacaktı.



Resim 7.3 KullaniciHatasi1 sınıfı çalıştırılınca verdiği ekran çıktısı

Java programlama dilinde, kural dışı durumlar da tıpkı başka kavramlar gibi sınıflar ve onların türündeki nesneler ile ifade edilirler. Bir kural dışı durum oluştuğunda Java programlama dili bu durum ile ilgili kural dışı durumun tipi, hangi satırdaki kod çalıştırılırken oluştuğu gibi bilgileri içeren bir nesne oluşturacaktır. Tablo 7.1'de kural dışı durumlar ile ilgili Java sınıflarından örnekler verilmiştir. Bu tabloda ilgili Java sınıfının adı ve hangi tür kural dışı durumlar ilgili ilgilendiği bilgileri yer almaktadır. Java programlama dilinde, kural dışı durumları temsil eden sınıflar Throwable isimli sınıftan kalıtım yoluyla türemiş olan Error ile Exception sınıfları ve bu sınıfların alt sınıflarıdır. Bir başka deyişle, Throwable sınıfı diğer kural dışı durum sınıfları için bir üst sınıftır. Java'da kural dışı durumlar, temel olarak **checked** ve **unchecked** kural dışı durumlar olarak ikiye ayrılırlar. Bu ünitede, Exception sınıfından kalıtım yoluyla türemiş kural dışı durumlar ve bunların işlenmesi üzerinde duracağız. Ünitenin tamamında ele alınacak olan kural dışı durumların içerisinde checked ve unchecked kategorilerine giren kural dışı durumlar mevcuttur.

Checked

Java derleyicisi bu kategorideki kural dışı durumların işlenmesini zorunlu kılar. Bunlara karşı bir eylem gerçekleştirilmedikçe derleyici hatası ile karşılaşırız.

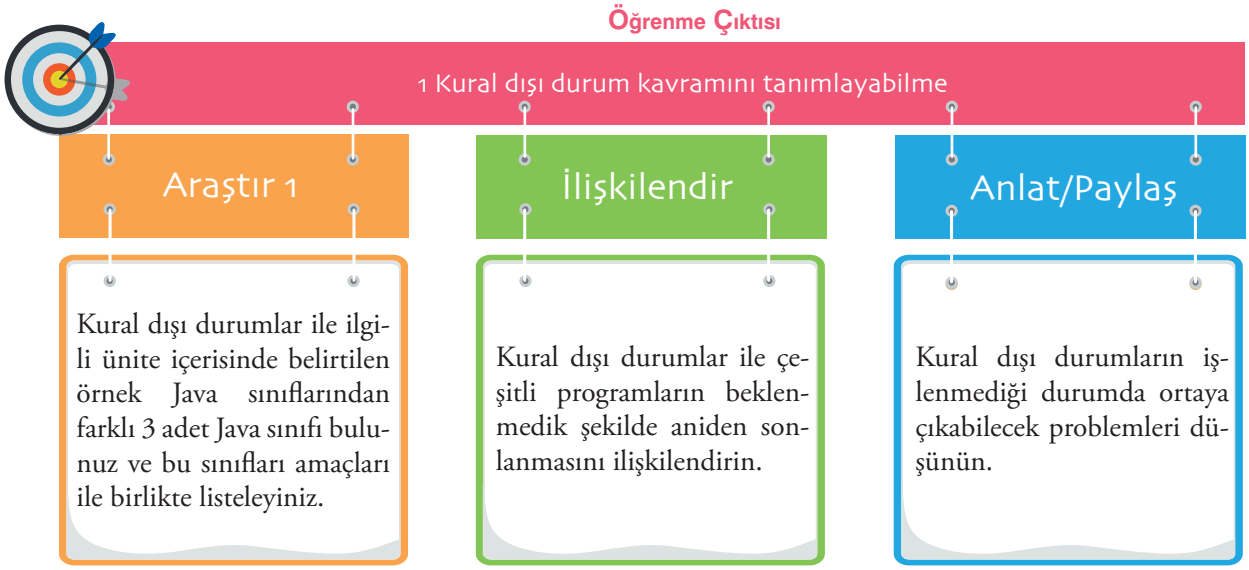
Unchecked

Java derleyicisi bu kategorideki kural dışı durumların işlenmesini zorunlu kılmaz. Derleyici hatalarına sebep olmadıkları için bu kural dışı durumların işlenip işlenmemesi programcıların tercihlerine bırakılmıştır.

Tablo 7.1 Kural dışı durumlarla ilgili Java sınıflarından örnekler

Kural Dışı Durum Sınıfı	İlgilendiği Kural Dışı Durumlar
ArithmeticException	Sıfıra bölünme ve benzeri aritmetik hatalar yapıldığı
ArrayIndexOutOfBoundsException	Dizinin olmayan elemanlarına erişilmek istenildiği
ClassCastException	Hatalı sınıf dönüşümü yapıldığı
IlegalArgumentException	Metotların hatalı parametreler ile çağrıldığı
IOException	Temel girdi çıktı işlemlerinde problem olduğu
NumberFormatException	Bir metnin sayıya hatalı bir şekilde dönüştürüldüğü
UnsupportedOperationException	Desteklenmeyen bir operasyonla karşılaşıldığı

Resim 7.1, 7.2 ve 7.3'e tekrar göz atalım ve karşılaştığımız kural dışı durumları bu bilgilerin ışığında tekrar inceleyelim. Resim 7.1'de ArrayIndexOutOfBoundsException ifadesini görmekteyiz. Resim 7.2 ve 7.3'te ise ArithmeticException ifadesini görmekteyiz. Bunlar, Tablo 7.1'de örneklenen kural dışı durum sınıfları arasında yer almaktadırlar.



KURAL DIŞI DURUM İŞLEME

Kural dışı durumlar oluştuğunda bilgisayar programlarının aniden bir sistem mesajı ile sonlandığını söylemiştik. Bu olayın önüne geçmek için kural dışı durumların işlenmesi söz konusudur. Bu şekilde programın uygun bir şekilde sonlandırılması veya mümkünse çalışmaya devam ettirilmesini sağlayabiliriz. Bir dosyaya veri yazılması esnasında kural dışı durum ile karşılaşıldığını varsayalım. Burada uygulama kural dışı durum dolayısıyla kapanırken dosyada birtakım bozulmalara dahi sebep olabilir. Bir diğer problem ise bilgisayar programını kullanan kişinin anlayamayacağı sistem mesajları ile programın sonlandırılmış olmasıdır. Kural dışı durumların işlenmesi, bu gibi birçok olayın önüne geçilmesini sağlayabilir. Java programlama dilinde, kural dışı durumlar için önlem alınması kimi durumlarda derleyici tarafından zorunlu kılınır. Böyle durumlarda, kural dışı durumlara bir çözüm üretilmesi istenildiğine dair derleyici mesajı alınır ve bununla ilgili bir önlem almadan programın çalıştırılması mümkün olmaz. Ancak, Java derleyicisi birçok kural dışı durum tipinin ele alınmasını zorunlu kılmaz. Bu da ele alınmayan kural dışı durumlar yüzünden programların aniden sonlanmasına sebep olabilir. Bu durumda, iki adet çözüm üretmek mümkündür. İlk çözüm, şimdi bahsedilecek olan **try-catch** blokları vasıtasıyla kural dışı durumların işlenmesidir. İkinci çözüm ise bu konunun devamında bahsedilecek olan **throws** ifadesinin kullanılması ile derleyici ikazlarının göz ardı edilmesidir. Bu bölümün devamında anlatılacak olan bir diğer kavram ise **try-catch** blokları ile birlikte kullanılabilen ve kural dışı durum oluşmasından bağımsız olarak her halukarda çalışmasını istediğimiz kodların yazılabileceği **finally** bloklarının kullanımıdır.

Try-Catch Bloğu Kullanımı

Kural dışı durumların işlenmesi amacıyla Java programlama dilinde **try-catch** blokları kullanılır. Aşağıda, **try-catch** bloğunun yapısı gösterilmiştir. Bununla birlikte, yorum satırı şeklinde ilgili alanların amacı belirtilmektedir. **Try ve catch** blokları arasına, kural dışı durum oluşabileceğini düşündüğümüz satırları yazarız. **Catch** bloğu içerisinde ise bu durum oluştuğunda yapılacaklara dair bilgiler yer almaktadır. **Catch** bloğunun Exception sınıfı türünde bir parametre aldığını görmekteyiz. Bu parametre, doğrudan

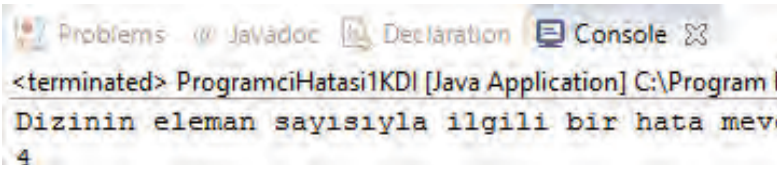
Exception sınıfının kendisi veya bu sınıftan kalıtım yoluyla türeyen diğer sınıflar olabilir.

```
try {
      // Kural dışı durum oluşabilecek satırlar
}
catch (Exception e) {
      // Kural dışı durum oluştuğunda yapılacaklar
}
```

Aşağıda, ProgramciHatasi1KDI isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf, önceki başlık altındaki ProgramciHatasi1 isimli sınıfın **try-catch** bloğu eklenmiş hâlidir. Bu örnekte kural dışı durum işlenmekte ve program kullanıcının anlayabileceği uygun bir hata ile sonlandırılmaktadır. **Catch** bloğu içerisinde Exception sınıfı yerine ondan kalıtım yoluyla türemiş olan ArrayIndexOutOfBoundsException sınıfı kullanılmıştır. Bunun sebebi, sadece o sınıfın ilgilendiği kural dışı durumların işlenmesinin istenilmesidir.

```
/* ProgramciHatasi1KDI.java */

public class ProgramciHatasi1KDI {
      public static void main(String[] args) {
            int sayi[]={1,2,3};
            try {
              System.out.println(sayi[4]);
            }
            catch(ArrayIndexOutOfBoundsException e) {
             System.out.println("Dizinin eleman sayısıyla ilgili bir hata mevcut");
             String mesaj = e.getMessage();
             System.out.println(mesaj);
            }
      }
}
```

ProgramciHatasi1KDI sınıfına ait program kodları çalıştırıldığında Resim 7.4'teki ekran görüntüsü ortaya çıkmaktadır. Buradan da görülebileceği gibi öncelikle kendi tanımladığımız hatayı ekrana yazdırdık. Daha sonra, sistemin hata bilgisini de ayrıca ekrana yazdırmış olduk. Böylelikle, program kullanıcılar tarafından anlaşılabilecek bir hata ile sonlanmış oldu.

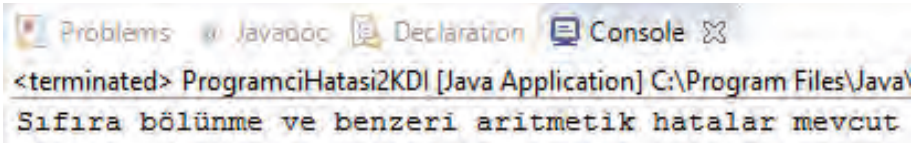


Resim 7.4 ProgramciHatasi1KDI sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, ProgramciHatasi2KDI.java isimli sınıfa ait Java kodları bulunmaktadır. Bu örnekte, kural dışı durum işlenmekte ve program kullanıcının anlayabileceği uygun bir hata ile sonlandırılmaktadır. **Catch** bloğu içerisinde Exception sınıfı yerine ondan kalıtım yoluyla türemiş olan ArithmeticException sınıfı kullanılmıştır.

```
/* ProgramciHatasi2KDI.java */

public class ProgramciHatasi2KDI {
      public static void main(String[] args) {
       int sayi1 = 5;
       int sayi2 = 0;
       int sonuc = 0;
       try {
            sonuc = sayi1 / sayi2;
       }
       catch(ArithmeticException e) {
            System.out.println("Sıfıra bölünme ve benzeri
aritmetik hatalar mevcut");
            System.exit(0);
       }
       System.out.println("Sonuç = "+sonuc);
      }
}
```

ProgramciHatasi2KDI sınıfına ait program kodları çalıştırıldığında Resim 7.5'teki ekran görüntüsü ortaya çıkmaktadır. Buradan da görülebileceği gibi öncelikle kendi tanımladığımız hatayı ekrana yazdırdık. Hatayı belirttikten sonra "System.exit(0)" ile belirtilen satır vasıtasıyla programı kendimiz sonlandırmış olduk. Bu satırı kaldırırsak programın çalışmaya devam edeceğini ve sonuç olarak ekrana 0'ın yazdırılacağını görürüz. Bunun sebebi, sonuc adlı değişkene başlangıçta 0 değerini atamış olmamızdır.

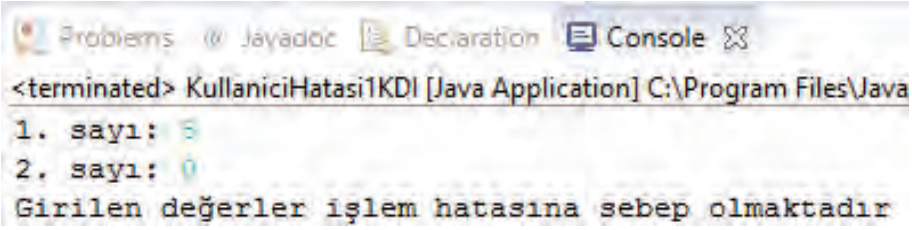


Resim 7.5 ProgramciHatasi2KDI sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, KullaniciHatasi1KDI isimli sınıfa ait Java kodları bulunmaktadır. Bu örnekte yapılan aritmetik işlemlerin sonucu oluşan bir kural dışı durum işlenmektedir. Kullanıcı, program çalıştırıldıktan sonra iki adet tamsayı girecektir. Daha sonra bu sayıların birbirine bölümü ekrana yazdırılacaktır. Kullanıcı, ikinci sayıyı 0 olarak girdiği takdirde bir kural dışı durum ile karşılaşılacaktır. Aksi takdirde, program herhangi bir sorun yaratmadan sonucu ekrana yazacaktır. Kullanıcının sırasıyla 5 ve 0 değerlerini girdiğini varsayalım. Kural dışı durumlar işlenmediği zaman bir önceki başlıkta yer alan Resim 7.3'teki ekran görüntüsü ortaya çıkmaktaydı. Burada ise **try-catch** bloğu kullanılarak kural dışı durumlar işlenmektedir. **Catch** bloğu içerisinde parametre olarak doğrudan Exception sınıfı kullanılmıştır. **Try** catch bloğunun aynı anda birden fazla türde hatayı işlemesini istediğimiz durumlarda doğrudan Exception sınıfını parametre olarak kullanabiliriz.

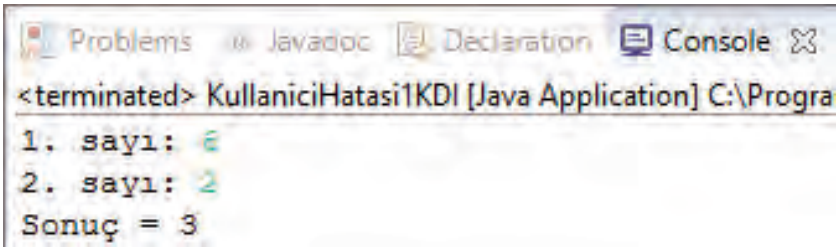
```
/* KullaniciHatasi1KDI.java */

import java.util.Scanner;
public class KullaniciHatasi1KDI {
      public static void main(String[] args) {
       int sayi1, sayi2, sonuc = 0;
       Scanner klavye = new Scanner(System.in);
       System.out.print("1. sayı: ");
       sayi1 = klavye.nextInt();
       System.out.print("2. sayı: ");
       sayi2 = klavye.nextInt();
       try {
             sonuc = sayi1 / sayi2;
             System.out.println("Sonuç = " + sonuc);
       }
       catch(Exception e) {
       System.out.println("Girilen değerler işlem hatasına
sebep olmaktadır");
       }
      }
}
```

KullaniciHatasi1KDI sınıfına ait program kodları çalıştırılıp 5 ve 0 değerleri girildiğinde Resim 7.6'daki ekran görüntüsü ortaya çıkmaktadır.



Resim 7.6 KullaniciHatasi1KDI sınıfının 5 ve 0 değerleri ile verdiği ekran çıktısı

Resim 7.7'de ise KullaniciHatasi1KDI sınıfına ait program kodları çalıştırılıp 6 ve 2 değerleri girildiği durumdaki ekran görüntüsü verilmektedir. Burada kural dışı durum oluşmamakta ve program normal akışını sürdürmektedir.



Resim 7.7 KullaniciHatasi1KDI sınıfının 6 ve 2 değerleri ile verdiği ekran çıktısı

Finally Bloğu Kullanımı

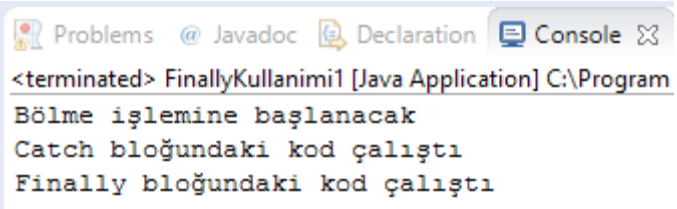
Kural dışı durum oluşup oluşmadığını düşünmeksizin çalışmasını istediğimiz bir takım program kodları varsa bunlar için **finally** bloğu kullanılmalıdır. **Finally** bloğu, üzerinde işlem yapılan açık bir dosyanın kapatılması veya kullanılan bir takım değişkenlerin bellekten temizlenmesi gibi farklı amaçlar ile kullanılabilir. **Finally** bloğu, aşağıda gösterildiği gibi **try-catch** bloğunun hemen sonrasında kullanılır.

```
try {
                // Kural dışı durum oluşabilecek satırlar
        }
        catch (Exception e) {
                // Kural dışı durum oluştuğunda yapılacaklar
        }
        finally {
                // Her durumda mutlaka çalıştırılacaklar
        }
```

Aşağıda, FinallyKullanimi1 isimli sınıfa ait Java kodları bulunmaktadır. Bu örnekte, **try** bloğu içerisindeki kodun çalışması sırasında kural dışı durum oluşmakta ve program **catch** bloğu içerisine girmektedir. **Finally** bloğu kullanıldığı için son olarak o bölümdeki kod mutlaka çalışacaktır.

```
/* FinallyKullanimi1.java */

public class FinallyKullanimi1 {
public static void main(String[] args) {
int a = 10, b = 0, sonuc;
try {
      System.out.println("Bölme işlemine başlanacak");
      sonuc = a / b;
      System.out.println("Bölme işlemi yapıldı");
}
catch (ArithmeticException e) {
      System.out.println("Catch bloğundaki kod çalıştı");
}
finally {
      System.out.println("Finally bloğundaki kod çalıştı");
}
}
```

FinallyKullanimi1 sınıfına ait program kodları çalıştırıldığında Resim 7.8'deki ekran görüntüsü ortaya çıkmaktadır. Ekran çıktısına dikkat edersek öncelikli olarak **try** bloğuna girilmiş ve buradaki ilk mesaj ekrana yazdırılmıştır. Kural dışı durum oluştuğu için **try** bloğu içerisindeki "bölme işlemi yapıldı" mesajının ekrana yazdırılamadığını görmekteyiz. Programın daha sonra **catch** bloğu içerisine girdiğini ve buradaki ilgili mesajı ekrana yazdırdığını görmekteyiz. Son olarak ise **finally** bloğunun içerisindeki kod çalışmakta ve bununla ilgili mesaj ekrana yazdırılmaktadır.

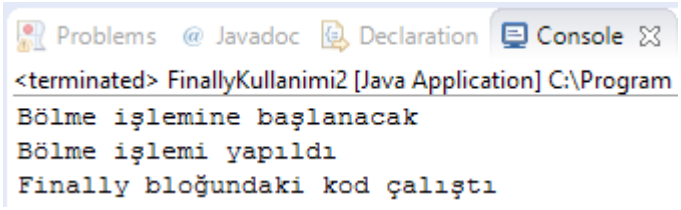


Resim 7.8 FinallyKullanimi1 sınıfının verdiği ekran çıktısı

Aşağıda, FinallyKullanimi2 isimli sınıfa ait Java kodları bulunmaktadır. Bu örnek, genel olarak FinallyKullanimi1 isimli Java sınıfının aynısı olmakla birlikte bir değişkenin değerinin farklı olduğunu görmekteyiz. Bu farklılığın program çıktısı üzerindeki etkisini inceleyeceğiz.

```
/* FinallyKullanimi2.java */

public class FinallyKullanimi2 {
public static void main(String[] args) {
int a = 10, b = 2, sonuc;
try {
      System.out.println("Bölme işlemine başlanacak");
      sonuc = a / b;
      System.out.println("Bölme işlemi yapıldı");
}
catch (ArithmeticException e) {
      System.out.println("Catch bloğundaki kod çalıştı");
}
finally {
      System.out.println("Finally bloğundaki kod çalıştı");
}
}
}
```

FinallyKullanimi2 sınıfına ait program kodları çalıştırıldığında Resim 7.9'daki ekran görüntüsü ortaya çıkmaktadır. Ekran çıktısına dikkat edersek öncelikli olarak **try** bloğuna girildiğini ve buradaki mesajların ekrana yazdırıldığını görürüz. Ancak bölme işlemi sırasında herhangi bir kural dışı durum oluşmadığı için **catch** bloğuna girilmediğini görmekteyiz. Son olarak ise **finally** bloğu içerisindeki mesaj ekrana yazdırılmaktadır. Sonuç olarak, program kodlarının çalışması esnasında gerek FinallyKullanimi1 gerekse FinallyKullanimi2 sınıfları içerisinde **finally** bloğunun içerisine girilmiş olduğunu görmekteyiz. Bu durum, kural dışı durum oluşup oluşmamasından bağımsız olarak **try** bloğunun çalıştığı her durumda **finally** bloğunun mutlaka çalıştığını göstermektedir.



Resim 7.9 FinallyKullanimi2 sınıfının verdiği ekran çıktısı

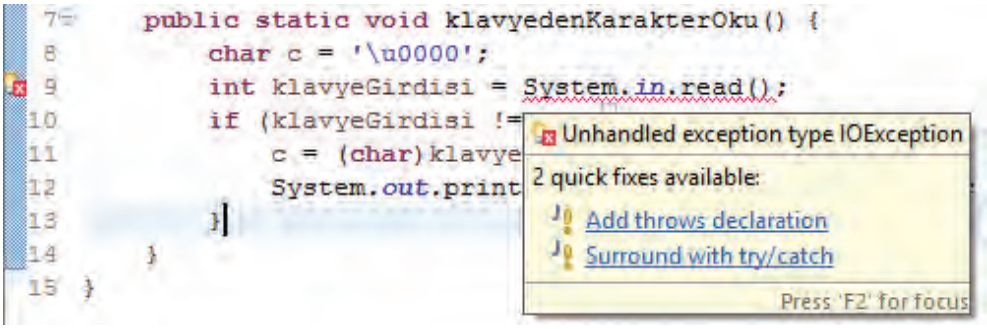
Throws Anahtar Kelimesi Kullanımı

Derleyici, kimi zaman yazılan kodun kural dışı durum üretme potansiyeli olduğunu görür ve programı yazan kişiye bir uyarı gönderir. Bu durumda, uyarıyı giderecek bir aktivite yapmadan program çalıştırılamayacaktır. Bu durumda bir önceki bölümde bahsedilen kural dışı durum işleme yöntemini uygulamak mümkündür. Ancak, bu ikazı gözetmek istersek Java programlama dilinde yer alan **throws** anahtar kelimesine ilgili metodun tanımında yer vermemiz gerekecektir. Aşağıda, ThrowsKullanimi1 isimli sınıfa ait Java kodları bulunmaktadır. ThrowsKullanimi1 sınıfı, klavyeden bir karakter okuyacak ve bu karakteri ekrana yazacaktır. Ancak, mevcut haliyle bu programın çalışabilir olmadığını göreceğiz.

Throws
Bu anahtar kelime, derleyicinin ele alınmasını zorunlu kıldığı checked kural dışı durum kategorisindeki durumlar için kullanılmaktadır.

```
/* ThrowsKullanimi1.java */

public class ThrowsKullanimi1 {
	public static void main(String[] args) {
		klavyedenKarakterOku();
	}
	public static void klavyedenKarakterOku() {
		char c = '\u0000';
		int klavyeGirdisi = System.in.read();
		if (klavyeGirdisi != -1) {
			c = (char)klavyeGirdisi;
			System.out.println("Girilen karakter = "+ c);
		}
	}
}
```

Resim 7.10'da, *ThrowsKullanimi1* sınıfının yol açtığı derleyici hatası gözlenmektedir. Java derleyicisi, yazılan kodun kural dışı durum oluşturma potansiyeli olduğunu anlamakta ve bu durumun bir çözüme kavuşması için ikaz vermektedir. Ekran çıktısında bu durumun çözümü için iki adet öneri üretildiği gözlenmektedir. Bu yöntemlerden birisi bu ünitenin genelinde bahsettiğimiz **try-catch** bloklarının kullanılmasıdır. İkinci çözüm önerisi ise throws anahtar kelimesinin metotların tanımında kullanılmasıdır. **Throws** anahtar kelimesi, ilgili metotların tanımında kullanıldığında bu metotların kural dışı duruma sebep olabileceğini bildiğimizi Java derleyicine belirtmiş oluruz. Bu durumda, derleyici hatası ortadan kalkacaktır. Ancak, **throws** anahtar kelimesi kullanarak **try-catch** bloğu kullanımında olduğu gibi doğrudan bir çözüm üretmemekteyiz. Sadece bu hatayı bildiğimizi ve ihmal etmek istediğimizi derleyiciye bildirmiş oluruz.

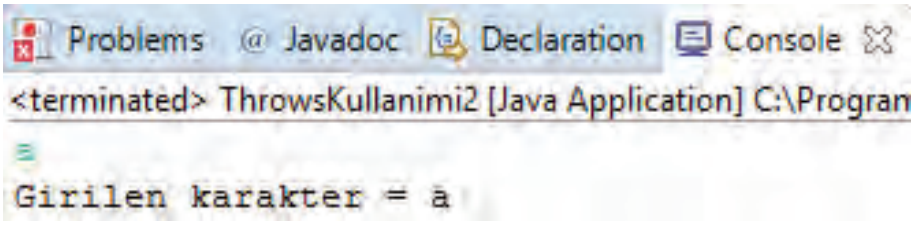


Resim 7.10 ThrowsKullanimi1 isimli sınıfta gözlenen derleyici hatası

Aşağıda, bu problemin throws anahtar kelimesi kullanılarak çözüldüğü ThrowsKullanimi2 isimli Java sınıfı yer almaktadır. Bu örnekte, klavyedenKarakterOku ve main isimli metotlara "**throws** IOException" ifadesinin eklenmesi ile çözüme ulaşılmaktadır. IOException ifadesi bu örnek için uygun olmakla birlikte daha çok kural dışı durumu kapsayan Exception ifadesi da burada kullanılabilir.

```
/* ThrowsKullanimi2.java */

import java.io.IOException;
public class ThrowsKullanimi2 {
      public static void main(String[] args) throws IOException {
            klavyedenKarakterOku();
      }
      public static void klavyedenKarakterOku() throws IOException {
            char c = '\u0000';
            int klavyeGirdisi = System.in.read();
            if (klavyeGirdisi != -1) {
                  c = (char)klavyeGirdisi;
                  System.out.println("Girilen karakter = "+ c);
            }
      }
}
```

ThrowsKullanimi2 sınıfına ait program kodları çalıştırıldığında Resim 7.11'deki ekran görüntüsü ortaya çıkmaktadır. Çıktıda uygulamanın çalıştığı ve klavyeden 'a' karakteri girildiği gözlenmektedir. Program, hatasız bir şekilde çalışmakta ve klavyeden okuduğu karakteri ekrana yazmaktadır.



Resim 7.11 ThrowsKullanimi2 sınıfının verdiği ekran çıktısı

Yaşamla İlişkilendir

THY'den saldırgan yolculara önlem

THY Tanıtım ve Halkla İlişkiler Başkanı Faik Akın yaptığı konuşmada, THY'nin yılda yaklaşık 12 milyon kişi taşıdığını belirterek, bunların içinde zaman zaman uçuş güvenliğini etkileyecek tarzda davranışta bulunan sayıları az da olsa yolcu bulunduğunu ifade etti.

"Havada öfke"nin dünyanın her yerinde yaşanan bir sorun olduğunu kaydeden Akın, özellikle uçakta yolculuk ederken yaşanan bu türden olayların ciddi güvenlik sorunları doğurduğunu söyledi.

Uluslararası Sivil Havacılık Örgütü'nün (ICAO) ve Taşımacılık İşçileri Federasyonu'nun (ITF) Sivil Havacılık Bölümü'nün uçuş güvenliğini tehlikeye düşürecek olaylara ilişkin bazı adımlar attığını belirten Akın, THY'nin de bu çerçevede kural dışı hareket eden yolculara karşı bazı yaptırımların uygulanacağı bir yönerge hazırladığını bildirdi.

Bu tür olaylarda, sorumluluğun uçak kapısının kapanmasına kadar olan bölümde Yer İşletmeleri Başkanlığı'na bağlı personelde, uçağın kapı kapatmasından sonra ise pilotta bulunduğunu anlatan Akın, alınan önlemlerle ilgili şu bilgileri verdi:

Uçuş güvenliği önce yerde başlar. Yer personelinin uçuş öncesinde davranışlarının uçuşun güvenliğini olumsuz yönde etkileyeceğine inandığı bir yolcuyu teşhis etmesi son derece kolay. Check-in aşaması, dinlenme ve boarding bölgelerinde diğer yolcuların huzurunu ve konforunu bozacak şekilde hareket etmesi durumunda söz konusu yolcu veya yolcuların taşınması reddedilecektir.

Sorumlu kaptan pilot, yolculara emir ve talimat vermeye, gerektiğinde emniyeti ve düzeni bozan kişileri uçaktan indirmeye yetkilidir.

Bu tür olaylarda tanık olarak imzaları bulunan üçüncü kişilerin karakol, savcılık veya mahkemelerdeki şahitlik esnasında ulaşım, konaklama, iaşe gibi masrafları THY tarafından karşılanacak.

Uçuş esnasında yolcunun kabul edilemez davranışlarının devamı halinde, hakkında kanuni soruşturma açılacağı aşağıdaki ifade kullanılarak kendisine açıklanacaktır. (Ulusal ve Uluslararası Sivil Havacılık kurallarına aykırı hareket etmektesiniz. Lütfen davranışlarınızı düzeltiniz. Vereceğimiz ortaklık talimatlarına uymazsanız varış meydanında emniyet görevlilerine rapor edileceksiniz ve gerekirse kaptan en yakın, uygun meydana inecek, sizi bırakacaktır. Bu durumda, bundan sonraki biletlerinizle ilgili ödemeler ve uçağın o meydana iniş kalkışıyla ilgili masraflar tarafınıza fatura edilecektir.)

Kural dışı hareket eden yolcunun bu davranışının sürmesi halinde sorumlu kaptan pilotun uygulayacağı yaptırımlardan bir tanesi de yolcunun tesirsiz hale getirilmesidir. Kabin ekibinin gücü bu kişiyi tesirsiz hale getirmeye yetmemesi durumunda yolculardan yardım istenir. Bu kişi için gerektiğinde kelepçe kullanılacak.

Kaynak: Bu habere http://arsiv.ntv.com.tr/news/92855.asp internet adresinden ulaşabilirsiniz.

Öğrenme Çıktısı

2 Kural dışı durumların işlenmesini açıklayabilme

Araştır 2

Throws anahtar kelimesi kullanımı ile try-catch bloğu kullanımı arasındaki farkı araştırarak açıklayınız.

İlişkilendir

try-catch bloğu kullanımı ile finally bloğu kullanımını ilişkilendir.

Anlat/Paylaş

finally bloğu kavramının var olmadığı durumda ortaya çıkabilecek problemleri düşünün.

1 Kural dışı durum kavramını tanımlayabilme

Kural Dışı Durumlar

Kitabın bu bölümünde kural dışı durumlar ve bu durumların işlenmesi kavramları ele alınmıştır. Kural dışı durumlar, programların çalışmaları esnasındaki normal akışlarını bozan herhangi bir problemi temsil etmektedir. Kural dışı durum oluşturma potansiyeli olan kodlar kimi zaman program çalışmadan derleyici tarafından da farkedilebilir ve programı yazan kişinin bununla ilgili bir çözüm üretmesi beklenebilir. Böyle durumlarda derleyici gerekli işlemlerin yapılmaması halinde programın çalışmasına izin vermeyecektir. Bu tip bir problemin çözümü için kural dışı durumların işlenmesi gerekmektedir. Bir kullanıcının hatalı veri girişi yapması, içeriğinden veri okunmak istenilen bir dosyanın bulunamaması, ağ üzerinden başka bir bilgisayar ile kurulacak bağlantıda beklenmedik sorunlar olması gibi haller bu durumlara örnek olarak verilebilirler. Java programlama dilinde kural dışı durumlar da tıpkı başka kavramlar gibi sınıflar ve onların türündeki nesneler ile ifade edilirler. Bir kural dışı durum oluştuğunda, Java programlama dili ilgili kural dışı durumun tipi, hangi satırdaki kod çalıştırılırken oluştuğu gibi bilgileri içeren bir nesne oluşturacaktır.

2 Kural dışı durumların işlenmesini açıklayabilme

Kural Dışı Durum İşleme

Java programlama dilinde kural dışı durumları temsil eden sınıflar Throwable isimli sınıftan kalıtım yoluyla türemiş olan Error ve Exception sınıfları ile bunların alt sınıflarıdır. Bir başka deyişle, Throwable sınıfı diğer kural dışı durum sınıfları için bir üst sınıftır. Kural dışı durumlarla ilgili sınıflara örnek olarak ArithmeticException ve IOException sınıflarını verebiliriz. ArithmeticException sınıfı, sıfıra bölünme ve benzeri aritmetik hatalar yapıldığı durumlarla ilgilenmektedir. IOException sınıfı ise temel girdi çıktı işlemlerinde problem olduğu durumlarla ilgilenmektedir. Java programlama dilinde kural dışı durumların işlenmesi için try-catch blokları kullanılır. Kural dışı durum oluşturma potansiyeli olan program kodları try bloğu içerisine yazılırken, sonrasında yapılacak işlemler catch bloğu içerisine yazılacaktır. Try bloğu içerisindeki kodlar kural dışı durum oluşana kadar adım adım çalışır ve kural dışı durumun oluştuğu noktadan sonraki kısımların atlanmasıyla catch bloğunun içerisindeki kodlara geçilir. Kural dışı durum oluşup oluşmadığını düşünmeksizin çalışması istenilen bir takım program kodları varsa finally bloğu kullanılmalıdır. Finally bloğu, try-catch bloğunun sonuna bir eklenti olarak yazılır. Finally bloğunun, üzerinde işlem yapılan açık bir dosyanın kapatılması veya kullanılan bir takım değişkenlerin bellekten temizlenmesi gibi farklı amaçları olabilir. Ünitede son olarak throws anahtar kelimesi kullanımı anlatılmıştır. Throws anahtar kelimesi, kural dışı durum oluşturma potansiyeli olan kodlar sebebiyle derleyici hatası ile karşılaşıldığı durumda kullanılır. Throws anahtar kelimesi, derleyici hatasını ortadan kaldırmaya yaramaktadır. Bu durumda, kural dışı durumların işlenmesi yerine ihmal edilmesi söz olacaktır. Java programlama dilinde, throws anahtar kelimesinin metotların tanımlarına eklenmesi derleyicinin bu kural dışı durumları göz ardı etmesine ve programın çalışmasına izin vermesine sebep olmaktadır.

neler öğrendik?

1 Temel girdi-çıktı işlemlerinde yaşanan problemler sonucu oluşan kural dışı durumların ifade edilmesi için kullanılan Java sınıfı aşağıdakilerden hangisidir?

A ArithmeticException
B. ClassCastException
C. UnsupportedOperationException
D. IllegalArgumentException
E. IOException

2 Metotların hatalı parametreler ile çağrılması sonucu oluşan kural dışı durumların ifade edilmesi için kullanılan Java sınıfı aşağıdakilerden hangisidir?

A. IllegalArgumentException
B. IOException
C. NumberFormatException
D. ArrayIndexOutOfBoundsException
E. ClassCastException

3 Java programlama dilinde kural dışı durumları temsil eden sınıfların tamamının kalıtım yoluyla türediği sınıf aşağıdakilerden hangisidir?

A. Math
B. ArrayList
C. Throwable
D. Graphics
E. IOException

4 Java programlama dilinde kural dışı durumların işlenmesi için kullanılan ifade bloğu aşağıdakilerden hangisidir?

A. try find
B. try catch
C. try remove
D. find catch
E. find remove

5 Java programlama dilinde kural dışı durumlar işlenirken mutlaka çalışması istenen kodlar aşağıdaki ifade bloklarından hangisi içinde yer almalıdır?

A. try B. catch
C. finally D. throws
E last

6
```
public class Uygulama1 {
    public static void
main(String[] args) {
        try {
            System.out.
print("1");
            System.out.
print("2");
        }
        catch(Exception e) {
            System.out.
print("3");
            System.out.
print("4");
        }
    }
}
```
Yukarıdaki Java program kodu çalıştırıldığında vereceği çıktı aşağıdakilerden hangisidir?

A. 1 B. 12
C. 13 D. 123
E. 1234

7
```
public class Uygulama2 {
    public static void
main(String[] args) {
        try {
            System.out.
print("1");
            int a = 5 / 0;
            System.out.
print("2");
        }
        catch(Exception e) {
            System.out.
print("3");
            System.out.
print("4");
        }
    }
}
```
Yukarıdaki Java program kodu çalıştırıldığında vereceği çıktı aşağıdakilerden hangisidir?

A. 12 B. 13
C. 123 D. 134 E. 1234

8
```
public class Uygulama3 {
      public static void
main(String[] args) {
            try {
              System.out.
print("1");
            }
            catch(Exception e) {
              System.out.
print("2");
            }
            finally{
              System.out.
print("3");
            }
      }
}
```
Yukarıdaki Java program kodu çalıştırıldığında vereceği çıktı aşağıdakilerden hangisidir?

A. 1 B. 2
C. 12 D. 13
E 123

9
```
public class Uygulama4 {
      public static void main
(String[] args) {
      try {
            int a = 5 / 0;
            System.out.print("1");
      }
      catch(Exception e) {
            System.out.print("2");
      }
      finally{
            System.out.print("3");
      }
   }
}
```
Yukarıdaki Java program kodu çalıştırıldığında vereceği çıktı aşağıdakilerden hangisidir?

A. 1 B. 12
C. 13 D. 23
E. 123

10 Java programlama dilinde yazılan bir metod, kural dışı durum oluşma ihtimaline karşı derleyici hatası verdiğinde bu hatayı ortadan kaldırmak için metodun tanımına aşağıdaki anahtar kelimelerden hangisi eklenmelidir?

A. throws B. try
C. catch D. finally
E. except

neler öğrendik yanıt anahtarı

1. E	Yanıtınız yanlış ise "Kural Dışı Durumlar" konusunu yeniden gözden geçiriniz.	6. B	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.
2. A	Yanıtınız yanlış ise "Kural Dışı Durumlar" konusunu yeniden gözden geçiriniz.	7. D	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.
3. C	Yanıtınız yanlış ise "Kural Dışı Durumlar" konusunu yeniden gözden geçiriniz.	8. D	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.
4. B	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.	9. D	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.
5. C	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.	10. A	Yanıtınız yanlış ise "Kural Dışı Durum İşleme" konusunu yeniden gözden geçiriniz.

7 Araştır Yanıt Anahtarı

Araştır 1

Kural dışı durumlar ile ilgili ünitede belirtilmemiş olan 3 adet Java sınıfı örneği aşağıda listelenmektedir.

Kural Dışı Durum Sınıfı	İlgilendiği Kural Dışı Durumlar
FileSystemNotFoundException	Dosya sisteminin bulunamadığı
NoSuchElementException	Elemanları üzerinde tek tek ilerlenilen bir listede daha fazla eleman bulunmadığı
SecurityException	Bir güvenlik ihlali ile karşılaşılan

Araştır 2

Throws anahtar kelimesinin kullanımı sadece derleyici hatasının ortadan kaldırılmasını sağlamaktadır. Throws anahtar kelimesi ile kural dışı durumlarda ne yapılacağına dair bir tanım yapılamaz. try-catch bloğu kullanıldığı durumda ise programın akışı sırasında belirli bir hata ile karşılaşılması durumunda ne yapılacağının belirlenmesi söz konusudur. İki yöntem de programın çalışmasını engelleyen durumu ilk anda ortadan kaldırmakla birlikte program çalışırken kural dışı durum ile karşılaşıldığında aynı etkiyi sağlamaz.

Kaynakça

Sharan, K. (2014). *Beginning Java 8 Fundamentals: Language Syntax, Arrays, Data Types, Objects, and Regular Expressions*, Apress yayınları.

Bölüm 8

Java'da Kullanıcı Arayüzü Tabanlı Programlama

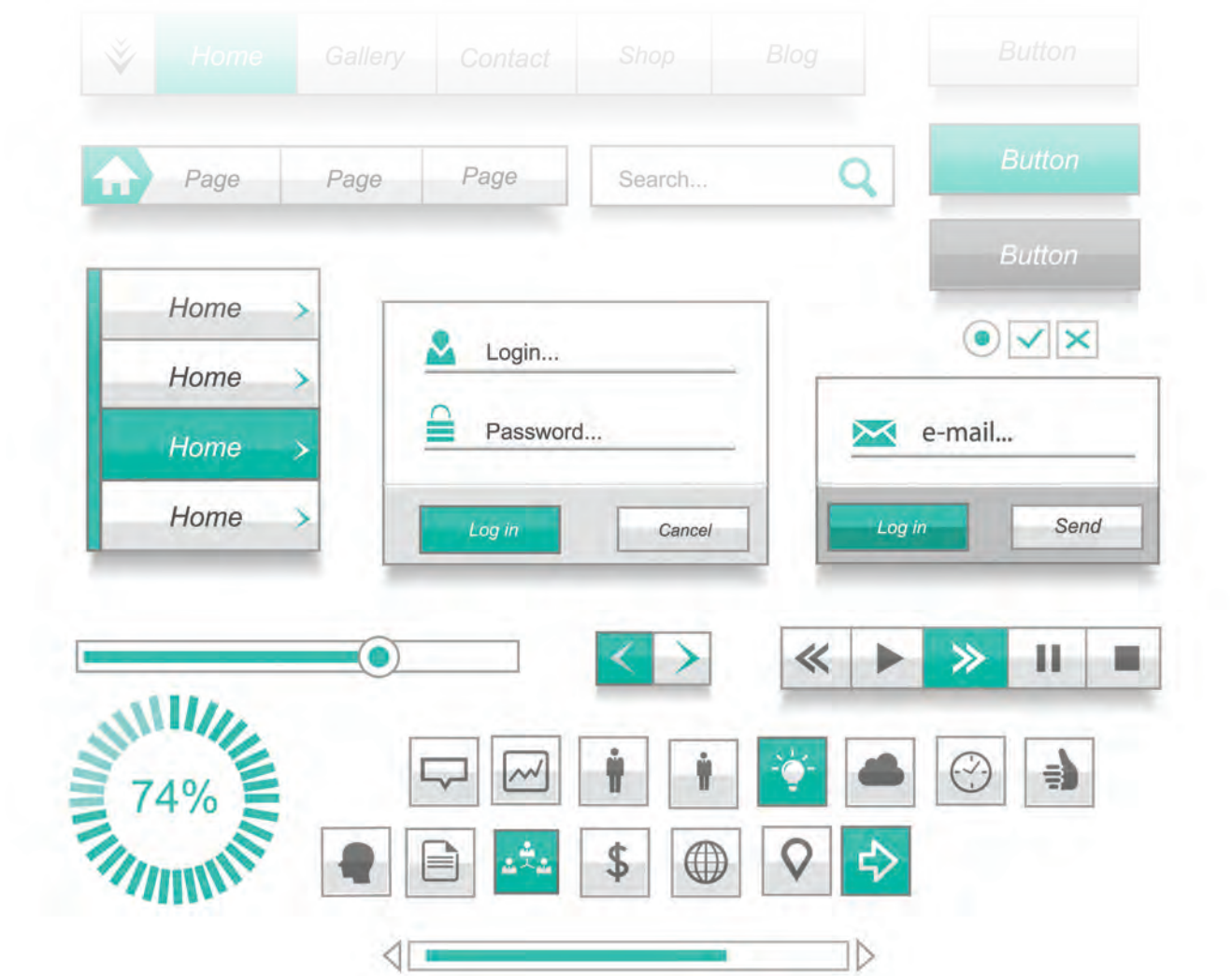
Öğrenme çıktıları

1 JavaFX Temelleri
1 JavaFX kütüphanesini tanımlayabilme

2 Kullanıcı Arayüzü Tasarımı
2 Kullanıcı arayüzü tabanlı programlamayı açıklayabilme

3 Olay Güdümlü Programlama
3 Olay güdümlü programlama kavramını açıklayabilme

Anahtar Sözcükler: • Kullanıcı Arayüzü • Javafx • Bileşen • Olay Güdümlü Programlama



GİRİŞ

Bu ünitede, kullanıcı arayüzü tabanlı programlama kavramı üzerinde duracağız. Bilgisayar programlarının çalışmaları sırasında genellikle kullanıcılardan bir takım bilgiler almaları ve bu bilgilere bağlı çıktılar üretmeleri gerekir. Aynı durum, günlük hayatımızda kullandığımız elektronik aletler için de genel olarak geçerlidir. Örneğin bir hesap makinesi kullanırken üzerinde tuşlar vasıtasıyla girdiğimiz değerlere karşı ekranda bir sonuç gözleriz. Hesap makinesi, üzerinde tanımlı olan işlemleri gerçekleştirerek bir çıktı üretir. Bilgisayar programlarında, bu girdi-çıktı işlemlerini iki farklı şekilde yapabilmek mümkündür. Bunlardan ilki, günümüz şartlarında çokta kullanışlı olmayan konsol veya siyah ekran olarak adlandırdığımız ortamdan veri alış-verişinin yapılmasıdır. İkincisi ise programlar için bir kullanıcı arayüzü oluşturulması ve girdi-çıktı işlemlerinin bu görsel ortamdan sağlanmasıdır. Günümüz programlarının birçoğu grafik kullanıcı arayüzü dediğimiz bu ortamları kullanmaktadır. Bu konuda verebileceğiz en temel örnek Windows işletim sistemi içerisinde kullanılan ofis yazılımlarıdır. Bu tip yazılımlarda kullanıcıların yapacağı işlemler için çeşitli arayüzler mevcuttur. Java programlama ortamında, kullanıcı arayüzü oluşturmak için çeşitli kütüphaneler mevcuttur. Ünitenin içeriğinde güncel kütüphanelerden birisi olan *JavaFX* üzerinde duracağız. Bu kapsamda, grafik kullanıcı arayüzlerinde kullanılan bileşenler de tanıtılmış olacaktır. Her ne kadar birçoğumuz bu bileşenlerle günlük bilgisayar kullanımlarımız esnasında karşılaşıyorsak da bu ünitede bunları kendi programlarımız içerisinde ne şekilde kullanabileceğimizi görmüş olacağız. Ünitenin devamında, ilk olarak *JavaFX* isimli kütüphanenin temel özellikleri incelenecektir. Daha sonra ise çeşitli kullanıcı arayüzü bileşenleri tanıtılacak ve bunların bir düzen içerisinde nasıl birlikte kullanılacağı ele alınacaktır. Son olarak ise bu arayüzlerde kullanıcı etkileşiminin nasıl sağlanacağından bahsedeceğiz. Bununla ilişkili olarak da olay güdümlü programlama kavramı üzerinde duracağız.

JAVAFX TEMELLERİ

Java programlama dilinde, kullanıcı arayüzü geliştirmek için sırasıyla *AWT*, *Swing* ve **JavaFX** isimli kütüphaneler ortaya çıkmıştır. *AWT* kütüphanesi, zaman içinde yerini bir takım kendi eksiklerinin revize edilmesi sonucu oluşturulan *Swing* kütüphanesine terk etmiştir. Bu ünitede anlatılacak *JavaFX* ise programcıların daha hızlı ve etkili şekilde kullanıcı arayüzü oluşturmaları adına sonradan inşa edilen bir Java kütüphanesidir. *JavaFX* kullanımının iki farklı şekilde yapılabilmesi mümkündür. Bunlardan ilki, arayüz görünümü oluşturmak için tamamen Java program kodlarının kullanıldığı durumdur. İkincisi ise *FXML* dediğimiz başka bir dosya yapısının bu sürece dâhil edilmesiyle kullanıcı arayüzünün ayrı bir dosya içerisinde oluşturulmasıdır. Bu ünitede, *JavaFX* uygulamaları yazmak için tamamen Java program kodlarının kullanıldığı yöntem üzerinde duracağız. Kullanıcı arayüzü oluşturmada kullanılan kütüphaneler bir takım arayüz bileşenleri içermelidirler. Aslında bunlar temel seviyede bilgisayar kullanıcılarının da çoğunlukla çeşitli yazılımları kullanırken karşılaştığı ancak hakkında bilgi sahibi olmadığı kavramlardır. Örneğin bir düğme bileşenini ofis yazılımları kullanan birçok kişi görmesine ve kullanmasına rağmen teknik olarak bu konuda bilgi sahibi olmayabilir. Tablo 8.1'de *JavaFX*'te kullanıcı arayüzü oluştururken kullanılan bileşenlere örnekler bulunmaktadır. Bu tabloda bileşenlerin kullanımı için gerekli olan Java sınıfları ve bu bileşenlerin isimleri yer almaktadır.

JavaFX

Bu kütüphane, Java geliştirme paketi (JDK) içerisine 2014 yılında yayınlanan 1.8 sürümünden itibaren dâhil edilmiştir.

Tablo 8.1 Kullanıcı arayüzü bileşenlerinden örnekler.

Java sınıfı	İsim
Button	Düğme
CheckBox	İşaret kutusu
ComboBox	Açılır kutu
Label	Etiket
ListView	Liste
PasswordField	Şifre alanı
RadioButton	Radyo düğmesi
ScrollBar	Kaydırma çubuğu
TextField	Metin alanı

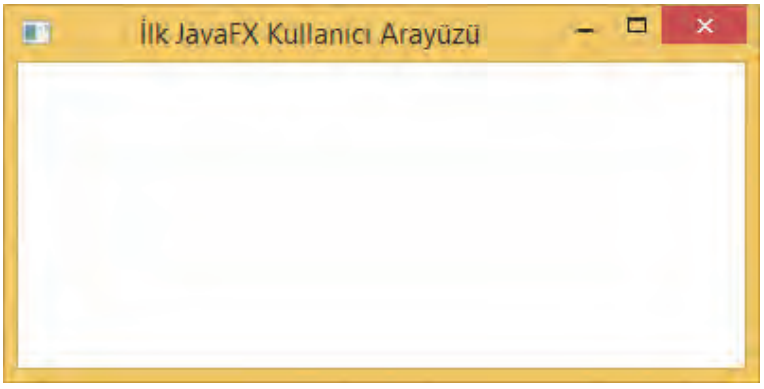
Şimdi de *JavaFX* uygulamalarının genel akışından bahsedeceğiz. Öncelikli olarak *JavaFX* uygulaması olacak sınıfın *Application* sınıfından kalıtım yoluyla türemiş ve *start* metodunu eziyor

olması gerekir. Bahsedilen *start* metodu *Stage* sınıfı türünde bir parametre almaktadır. *Stage* sınıfı, ekranda bir form oluşmasına olanak sağlayacaktır. Birden fazla form oluşturulması söz konusu ise *Stage* sınıfından istenilen sayıda kullanılması gerekecektir. Ancak bu ünitenin içeriğindeki tüm örneklerde tek bir *Stage* sınıfı kullanılmaktadır. Ayrıca, arayüz oluşturmak için *Stage* sınıfı ile ilişkilendirilmiş *Scene* türünde sınıfımız bulunacaktır. *Scene* sınıfının içerilerinde etiket, metin alanı vb. grafik kullanıcı arayüzü elemanları yer alabilmektedir. Aşağıda, *ArayuzUygulama1* isimli sınıfa ait boş bir kullanıcı arayüzü oluşturmaya yarayan Java program kodları bulunmaktadır. Bu program, aynı zamanda bahsedilen sınıfların da temel kullanımlarına örnek teşkil etmektedir.

Scene
Bu sınıf türündeki nesneler aynı anda en fazla bir adet *Stage* türündeki nesne ile ilişkilendirilebilir.

```
/* ArayuzUygulama1.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ArayuzUygulama1 extends Application {
  public static void main(String[] args) {
     Application.launch(args);
  }
  public void start(Stage stage) {
     VBox bilesenYerlesim = new VBox();
     Scene tasiyici = new Scene(bilesenYerlesim, 400, 200);
     stage.setScene(tasiyici);
     stage.setTitle("İlk JavaFX Kullanıcı Arayüzü");
     stage.show();
  }
}
```

Bu program kodunu incelediğimizde *Stage* sınıfı türünde oluşturulan stage isimli nesneye ait üç metodun sırasıyla çağrıldığını görmekteyiz. Burada *setScene* metodu *tasiyici* ismini verdiğimiz *Scene* türündeki nesneyi *stage* nesnesi ile ilişkilendirmektedir. Bunun yanısıra, *setTitle* metodu *stage* nesnesinin başlığını belirlemektedir. Son olarak kullanılan *show* metodu ise *stage* nesnesinin yani form penceresinin görünür hâle gelmesini sağlamaktadır. *Main* metodu ise önceki ünitelerimizdeki şu ana kadar yazdığımız Java programlarında olduğu gibi uygulamanın çalıştırılmasını sağlamaktadır. *Vbox* sınıfının işlevi ise form üzerinde bulunacak olan bileşenlerin ne şekilde yerleştirileceğinin belirlenmesi ise ilgilidir. *Scene* sınıfı türünde *tasiyici* isimli nesne oluşturulurken kullanılan 400 ve 200 parametreleri formun genişliğinin 400 piksel ve yüksekliğinin 200 piksel olmasına sebep olacaktır. *Arayuzuygulama1* sınıfına ait program kodları çalıştırıldığında Resim 8.1'deki ekran görüntüsü ortaya çıkmaktadır. Resim 8.1'de de görüldüğü gibi program çalıştığında sadece başlığı ve ekrandaki boyutları belirli olan boş bir form ortaya çıkmaktadır.



Resim 8.1 ArayuzUygulama1 sınıfı çalıştırılınca verdiği ekran çıktısı.

Az önce de bahsedildiği gibi *Arayuzuygulama1* isimli sınıfta eklenebilecek bileşenlerin ekranda yerleşimini belirlemek için *VBox* sınıfı kullanılmıştır. Ancak *JavaFX* kütüphanesi içerisinde farklı şekilde yerleşimlere olanak sağlayan çeşitli sınıflar bulunmaktadır. Tablo 8.2'de bu amaca hizmet eden Java sınıflarından örnekler verilmektedir.

Tablo 8.2 Bileşenlerin yerleşimi ile ilgili sınıf örnekleri.

Java sınıfı	İşlevi
HBox	Bileşenlere tek bir satırda yatay olarak yerleşim sağlar
VBox	Bileşenlere tek bir sütunda dikey olarak yerleşim sağlar
BorderPane	Bileşenlere üst, alt, sağ, sol, merkez şeklinde yerleşim sağlar
GridPane	Bileşenlere ızgara şeklinde bir yerleşim sağlar

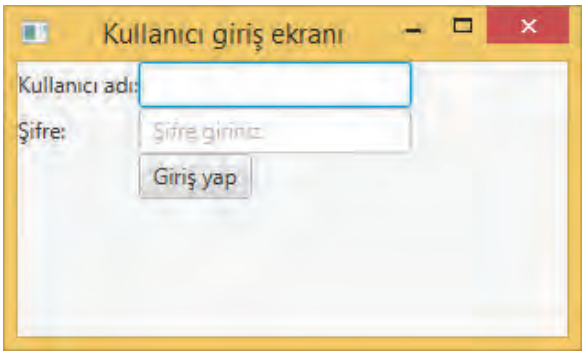
KULLANICI ARAYÜZÜ TASARIMI

Bir önceki başlıkta *JavaFX* ile kullanıcı arayüzü oluşturulması için kullanılan Java sınıflarında bahsettik. Bu bölümde ise farklı bileşenler kullanılarak oluşturulan çeşitli örnekleri inceleyeceğiz. Aşağıda *ArayuzUygulama2* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf içerisinde *etiket*, *metin alani*, *sifre alani* ve *dugme* bileşenleri kullanılmaktadır. *Stage* sınıfı kullanılarak ekranda bir form oluşturulmaktadır. Bileşenlerin yerleşimi için bu örnekte *GridPane* sınıfı kullanılmaktadır. *GridPane* içerisindeki yerleşimin planlanması için *setConstraints* isimli metot mevcuttur. Bu metot sırasıyla bileşen adı, sütun ve satır indeksleri parametre olarak almaktadır. Java programlama dilinde indeks deyimleri 0'dan başlamakta olduğundan "1,2" ile belirtilen indeks 2. sütun ve 3. satırı ifade etmektedir. Sırasıyla bileşenler oluşturulmuş, *GridPane* türündeki nesnenin içerisine eklenmiş ve *GridPane* türündeki nesne de *Scene* türündeki nesneyle ilişkilendirilmiştir. Bileşenlerin *GridPane* türündeki nesnenin içerisine eklenmesi için *getChildren* metodunun sonrasında çağrılan *addAll* metodu kullanılmaktadır. Son olarak ise *Scene* türündeki nesnenin *Stage* türündeki nesne ile ilişkilendirildiğini görmekteyiz.

getChildren
Bu metot, temel anlamda o ana kadar eklenmiş olan bileşenlerin listesini döndürmektedir.

```
/* ArayuzUygulama2.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class ArayuzUygulama2 extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    public void start(Stage stage) throws Exception {
    stage.setTitle("Kullanıcı giriş ekranı");
        GridPane bilesenYerlesim = new GridPane();
        Label etiket1 = new Label("Kullanıcı adı:");
        GridPane.setConstraints(etiket1, 0, 0);
        TextField isim = new TextField( );
        GridPane.setConstraints(isim, 1, 0);
        Label etiket2 = new Label("Şifre:");
        GridPane.setConstraints(etiket2, 0, 1);
        PasswordField sifre = new PasswordField();
        sifre.setPromptText("Şifre giriniz");
        GridPane.setConstraints(sifre, 1, 1);
        Button dugme = new Button("Giriş yap");
        GridPane.setConstraints(dugme, 1, 2);
      bilesenYerlesim.getChildren().addAll(etiket1, isim, etiket2,
sifre, dugme);
        Scene tasiyici = new Scene(bilesenYerlesim, 300, 150);
        stage.setScene(tasiyici);
        stage.show();
    }
}
```

ArayuzUygulama2 sınıfına ait program kodları çalıştırıldığında Resim 8.2'deki ekran görüntüsü ortaya çıkmaktadır. Ekran görüntüsünde de görüldüğü gibi bileşenler sütun ve satır numaralarına göre yerleştirilmiştir. Şifre alanının üzerinde "Şifre giriniz" şeklinde geçici bir metin görülmekte olup bu alana tıklanıldığında bu görüntü kaybolacaktır. Bu geçici metin *setPromptText* metodu ile oluşturulmuştur.

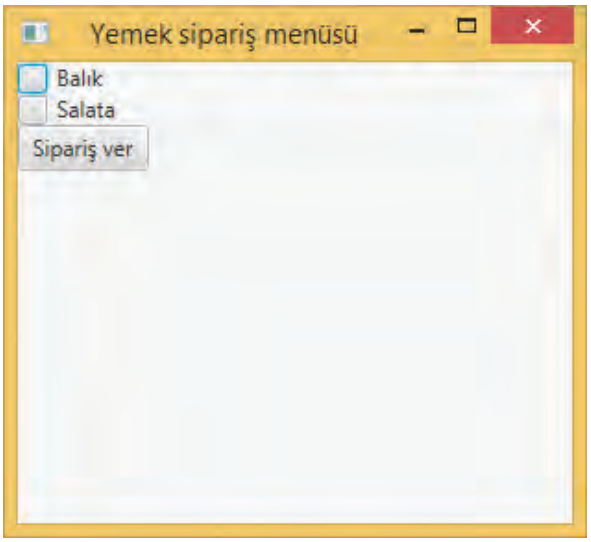


Resim 8.2 ArayuzUygulama2 sınıfı çalıştırılınca verdiği ekran çıktısı.

Aşağıda, *ArayuzUygulama3* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf içerisinde işaret kutusu ve düğme bileşenleri kullanılmaktadır. *Stage* sınıfı kullanılarak ekranda bir form oluşturulmaktadır. Bileşenlerin yerleşimi için bu örnekte *VBox* sınıfı kullanılmaktadır. *VBox* sınıfı kullanıldığı için bileşenlerin tek bir sütunda dikey olarak yerleştiği görülecektir. Sırasıyla bileşenler oluşturulmuş, *VBox* türündeki nesnenin içerisine eklenmiş ve *VBox* türündeki nesne de *Scene* türündeki nesneyle ilişkilendirilmiştir. Son olarak ise *Scene* türündeki nesnenin *Stage* türündeki nesne ile ilişkilendirildiğini görmekteyiz.

```
/* ArayuzUygulama3.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ArayuzUygulama3 extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Yemek sipariş menüsü");
    CheckBox secim1 = new CheckBox("Balık");
    CheckBox secim2 = new CheckBox("Salata");
    Button dugmeGonder = new Button("Sipariş ver");
    VBox bilesenYerlesim = new VBox( );
    bilesenYerlesim.getChildren().addAll(secim1, secim2, dugmeGonder);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
}
```

ArayuzUygulama3 sınıfına ait program kodları çalıştırıldığında Resim 8.3'teki ekran görüntüsü ortaya çıkmaktadır.

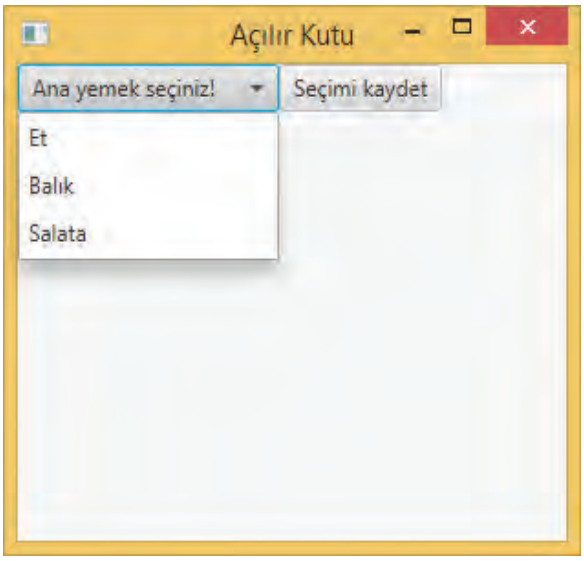


Resim 8.3 ArayuzUygulama3 sınıfı çalıştırılınca verdiği ekran çıktısı.

Aşağıda, *ArayuzUygulama4* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf içerisinde açılır kutu ve düğme bileşenleri kullanılmaktadır. Açılır kutu bileşeninin içerisindeki elemanların *addAll* metoduyla eklendiğiniz görebiliriz. *Stage* sınıfı kullanılarak ekranda bir form oluşturulmaktadır. Bileşenlerin yerleşimi için bu örnekte *HBox* sınıfı kullanılmaktadır. *HBox* sınıfı kullanıldığı için bileşenlerin tek bir satırda yatay olarak yerleştiği görülmektedir. Sırasıyla bileşenler oluşturulmuş, *HBox* türündeki nesnenin içerisine eklenmiş ve *HBox* türündeki nesne de *Scene* türündeki nesneyle ilişkilendirilmiştir. Son olarak ise *Scene* türündeki nesnenin *Stage* türündeki nesne ile ilişkilendirildiğini görmekteyiz.

```
/* ArayuzUygulama4.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class ArayuzUygulama4 extends Application {
    ComboBox<String> acilirKutu;
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Açılır Kutu");
    Button dugme = new Button("Seçimi kaydet");
    acilirKutu = new ComboBox<>();
    acilirKutu.getItems().addAll(
        "Et",
        "Balık",
        "Salata"
    );
    acilirKutu.setPromptText("Ana yemek seçiniz!");
    HBox bilesenYerlesim = new HBox( );
    bilesenYerlesim.getChildren().addAll(acilirKutu, dugme);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
}
```

ArayuzUygulama4 sınıfına ait program kodları çalıştırıldığında Resim 8.4'teki ekran görüntüsü ortaya çıkmaktadır.



Resim 8.4 ArayuzUygulama4 sınıfı çalıştırılınca verdiği ekran çıktısı

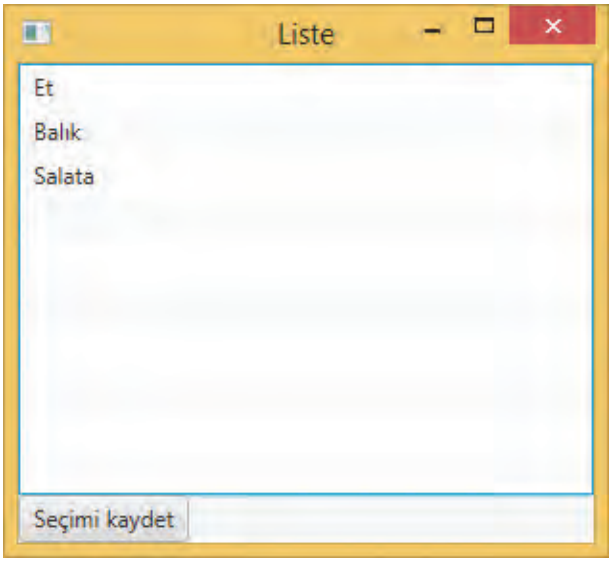
Aşağıda *ArayuzUygulama5* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf içerisinde liste ve düğme bileşenleri kullanılmaktadır. Liste bileşeninin içerisindeki elemanların *addAll* metoduyla eklendiğiniz görebiliriz. Liste bileşenine ait *setSelectionMode* metodu ise aynı anda listenin tek elemanının mı yoksa birden fazla elemanının mı seçimine izin verileceğini belirlemektedir. Bu örnekte birden fazla elemanın seçilmesine izin verilmektedir. *Stage* sınıfı kullanılarak ekranda bir form oluşturulmaktadır. Bileşenlerin yerleşimi için bu örnekte *VBox* sınıfı kullanılmaktadır. *VBox* sınıfı kullanıldığı için bileşenlerin tek bir sütunda dikey olarak yerleştiği görülmektedir. Sırasıyla bileşenler oluşturulmuş, *VBox* türündeki nesnenin içerisine eklenmiş ve *VBox* türündeki nesne de *Scene* türündeki nesneyle ilişkilendirilmiştir. Son olarak ise *Scene* türündeki nesnenin *Stage* türündeki nesne ile ilişkilendirildiğini görmekteyiz.

setSelectionMode

Bu metodun aldığı parametre SelectionMode.SINGLE olduğunda listenin tek elemanının, SelectionMode.MULTIPLE olduğunda ise listenin birden fazla elemanın seçilmesine izin verilmektedir.

```
/* ArayuzUygulama5.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.SelectionMode;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.scene.control.ListView;

public class ArayuzUygulama5 extends Application {
  ListView<String> liste;
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Liste");
    Button dugme = new Button("Seçimi kaydet");
    liste = new ListView<>();
    liste.getItems().addAll("Et", "Balık", "Salata");
liste.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
    VBox bilesenYerlesim = new VBox( );
    bilesenYerlesim.getChildren().addAll(liste, dugme);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
}
```

ArayuzUygulama5 sınıfına ait program kodları çalıştırıldığında Resim 8.5'teki ekran görüntüsü ortaya çıkmaktadır.



Resim 8.5 ArayuzUygulama5 sınıfı çalıştırılınca verdiği ekran çıktısı

Öğrenme Çıktısı

2 Kullanıcı arayüzü tabanlı programlamayı açıklayabilme

Araştır 2

Günlük hayatta kullandığımız programlarda kimi zaten kullanıcı arayüzlerinde yapılan seçim sonrasında "Evet", "Hayır, "Tamam" gibi onaylanması gereken ekranlarla karşı karşıya kalmaktayız. JavaFX kütüphanesi içerisinde bu amaçla kullanılan bileşenler hakkında bir araştırma yaparak bilgi toplayınız.

İlişkilendir

Kullanıcı arayüzü tabanlı programlama ile JavaFX kütüphanesini ilişkilendirin.

Anlat/Paylaş

Kullanıcı arayüzü tabanlı programlama kavramının ortaya çıkmasından önce kullanıcı etkileşimi amacıyla kullanılan yöntemleri düşünün.

Yaşamla İlişkilendir

Gmail'de Yeni E-posta Yazma Sistemi Varsayılan Hale Geldi

Gmail'in Google tarafından hazırlanan yeni mesaj yazma arayüzü artık tüm kullanıcılar için varsayılan hale gelecek. Ekim 2012'den beri Gmail kullanıcıları dilerlerse Gmail'in yeni e-posta yazma arayüzünü tercih edebiliyorlardı ve beğenmezlerse eski arayüze geçme seçenekleri de mevcuttu. Ancak Google tarafından yapılan resmi açıklamaya göre çok kısa bir süre sonra bu yeni tasarım varsayılan hale gelecek ve eski tipte e-posta yazma ekranı artık tarihe karışacak. Pek çok kullanıcının yeni yöntemden rahatsızlık duyduğu Google tarafından da itiraf edilmişti ve sistem ilk devreye sokulduğunda çoğu kullanıcı soğuk yaklaşmıştı. Ancak insanların meşguliyetlerinden dolayı yeni bir arayüz öğrenmek istemediklerine inanan Google, gelen kullanıcı tepkilerine göre gerekli düzenlemekleri ve ayarları gerçekleştirmiş durumda. Bildiğiniz gibi yeni sistem aynı anda hem gelen e-postalarınızı görmenize hem de yeni bir posta yazmanıza imkan tanıyor. Böylece e-postalarınıza yeniden dönmek istediğinizde mesajınızı yarım bırakmadan tüm işlerinizi halledebiliyorsunuz. Kullanıcılara sırasıyla dağıtılmaya başlanan yeni sürüm bir kaç gün içerisinde sizlere de ulaşacaktır.

Kaynak: Bu habere http://www.milliyet.com.tr/gmail-de-yeni-e-posta-yazma-sistemi-varsayilan-hale-geldi-internet-haber-1689226/ internet adresinden ulaşabilirsiniz.

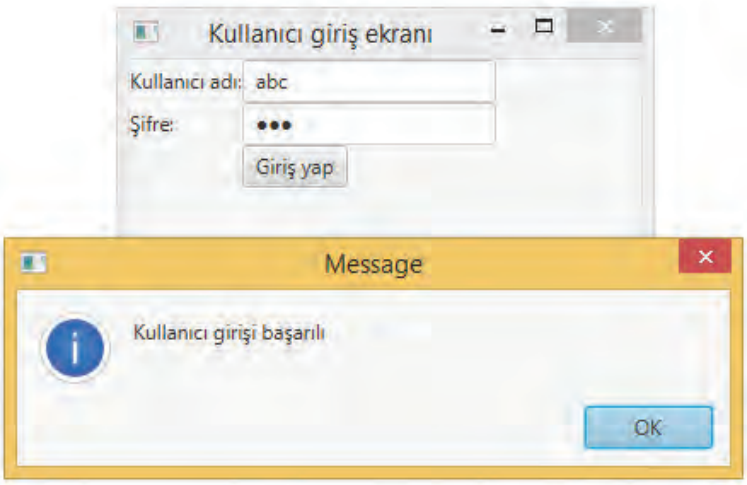
OLAY GÜDÜMLÜ PROGRAMLAMA

Olay kelimesini herhangi bir aktivitenin gerçekleşmesi olarak tanımlayabiliriz. Bu durumda, bu aktivitenin gerçekleşmesine sebep olan varlığı da olayın kaynağı olarak adlandırabiliriz. Örneğin bir insanın vazoya çarpıp onu yere düşürmesini ele alalım. Burada vazonun yere düşmesi gerçekleşmiş olan bir olayı ifade eder. Vazoyu yere düşüren insan ise bu olayın kaynağıdır. Nesneye yönelik programlama mantığı ile yazılmış bilgisayar programları da benzer şekilde çalışmaktadır. Fareyi bir düğmenin üzerine getirip tıkladığımızı varsayalım. Burada düğmenin üzerine tıklanması bir olayı ifade etmektedir. Fare ise o olayın kaynağıdır. Bir diğer taraftan da bu olay gerçekleştikten sonra ne yapılması gerektiği tanımlanmış olmalıdır. Bu tanımlamayı da olay yönetimi olarak isimlendireceğiz. Az önceki örnekten yola çıkarsak fare tarafından düğme tıklandığı zaman ekranda bir mesaj görüntülenmesini isteyebiliriz. Bu durumda olay yönetimi ile ekranda bir mesaj görüntülenmesi sağlanacaktır. Bahsettiklerimizin tamamını ise olay güdümlü programlama olarak adlandıracağız. Bir önceki bölümde Java ile grafik kullanıcı arayüzü oluşturmak konusunda çeşitli örnekler verilmişti. Ancak, o örneklerde olay yönetimi kavramı işlenmemiştir. Örneğin düğmelere tıklandığı zaman ne yapılacağına dair bir tanım bulunmamaktadır. Bu bölümde olay güdümlü programlama ile daha önce oluşturduğumuz arayüzlere işlevsellik sağlayacağız. Bu sebeple, örneklerin yeni kısımlarından bahsedecek olup önceki bölümde kullanıcı arayüzü oluşturmak ile ilgili verdiğimiz bilgilerin tamamını tekrar etmeyeceğiz. Aşağıda *ArayuzUygulama2OGP* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf, bir önceki bölümdeki *ArayuzUygulama2* sınıfının içeriğine olay güdümlü programlama kavramı dâhil edilmiş versiyonudur. *ArayuzUygulama2OGP* sınıfı içerisinde düğme bileşene ait *setOnAction* metodunun eklendiğini görmekteyiz. Düğmenin tıklanması olayı sonrasında bu metodun içeriği çalıştırılacaktır. Dolayısıyla bu metodun içeriğindeki kod olay yönetimini gerçekleştirmek amacını taşımaktadır. Bu metodun içeriğini incelediğimiz zaman kullanıcı adının "abc" ve şifrenin "123" girildiği durumda ekrana "Kullanıcı girişi başarılı" ifadesinin yazılacağını görmekteyiz. Aksi durumda ise "Hatalı veri girişi!" şeklinde bir mesaj görüntülenecektir. Bu mesajların ayrı bir pencerede görüntülenmesi için *Alert* isimli sınıf kullanılmaktadır.

```
/* ArayuzUygulama2OGP.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class ArayuzUygulama2OGP extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Kullanıcı giriş ekranı");
    GridPane bilesenYerlesim = new GridPane();
    Label etiket1 = new Label("Kullanıcı adı:");
    GridPane.setConstraints(etiket1, 0, 0);
    TextField isim = new TextField( );
    GridPane.setConstraints(isim, 1, 0);
    Label etiket2 = new Label("Şifre:");
    GridPane.setConstraints(etiket2, 0, 1);
    PasswordField sifre = new PasswordField();
    sifre.setPromptText("Şifre giriniz");
    GridPane.setConstraints(sifre, 1, 1);
    Button dugme = new Button("Giriş yap");
    GridPane.setConstraints(dugme, 1, 2);
    dugme.setOnAction(e -> {
         String kulBil = isim.getText();
         String sifreBil = sifre.getText();
        if (kulBil.equalsIgnoreCase("abc") &&
sifreBil.equalsIgnoreCase("123")) {
                Alert mesajPenceresi = new
Alert(AlertType.INFORMATION);
                mesajPenceresi.setHeaderText(null);
                mesajPenceresi.setContentText("Kullanıcı girişi başarılı");
                mesajPenceresi.showAndWait();
          } else {
                Alert mesajPenceresi = new Alert(AlertType.ERROR);
                mesajPenceresi.setHeaderText(null);
                mesajPenceresi.setContentText("Hatalı veri girişi!");
                mesajPenceresi.showAndWait();
          }
          });
    bilesenYerlesim.getChildren().addAll(etiket1, isim, etiket2,
sifre, dugme);
    Scene tasiyici = new Scene(bilesenYerlesim, 300, 150);
    stage.setScene(tasiyici);
    stage.show();
  }
}
```

ArayuzUygulama2OGP sınıfına ait program kodları çalıştırıldığında Resim 8.6'daki ekran görüntüsü ortaya çıkmaktadır. Ekran çıktısında, kullanıcı adı ve şifre istenilen şekilde girildiği için "Kullanıcı girişi başarılı" ifadesinin yazıldığı görülmektedir.

Aşağıda, *ArayuzUygulama3OGP* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf, bir önceki bölümdeki *ArayuzUygulama3* sınıfının içeriğine olay güdümlü programlama kavramı dâhil edilmiş versiyonudur. *ArayuzUygulama3OGP* sınıfı içerisinde *dugme* bileşene ait *setOnAction* metodunun eklendiğini görmekteyiz. Bu metodun içerisinde ise *islemYap* isimli metoda yönlendirme mevcuttur. Dolayısıyla düğme tıklandığı zaman *islemYap* isimli metot çalışacaktır. Bu metodun içeriği incelendiği zaman iki adet işaret kutusundan hangilerine tıklanmışsa ekrana onları içeren bir mesaj çıkarılacağı görülmektedir. Hiçbir seçim yapılmamış ise "Sipariş vermediniz" şeklinde bir mesajın görüntülenmesi beklenmektedir. Bu mesajların ayrı bir pencerede görüntülenmesi için burada da *Alert* isimli sınıf kullanılmaktadır.



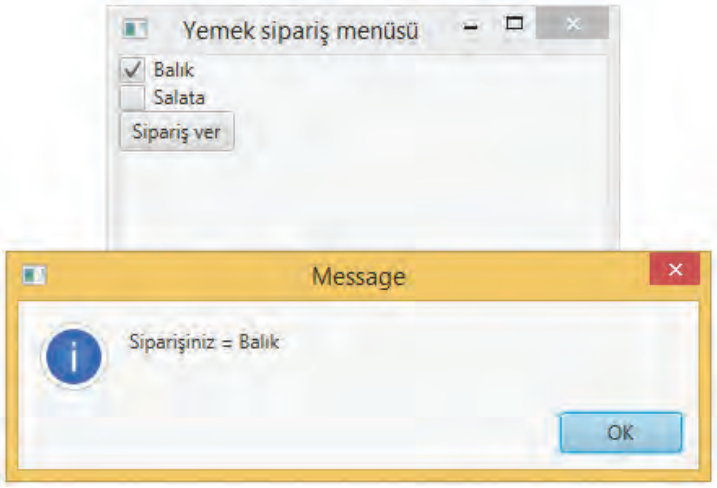
Resim 8.6 ArayuzUygulama2OGP sınıfı çalıştırılınca verdiği ekran çıktısı.

```
/* ArayuzUygulama3OGP.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ArayuzUygulama3OGP extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Yemek sipariş menüsü");
    CheckBox secim1 = new CheckBox("Balık");
    CheckBox secim2 = new CheckBox("Salata");
    Button dugmeGonder = new Button("Sipariş ver");
    dugmeGonder.setOnAction(e -> islemYap(secim1, secim2));
    VBox bilesenYerlesim = new VBox( );
   bilesenYerlesim.getChildren().addAll(secim1, secim2, dugmeGonder);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
  private void islemYap(CheckBox c1, CheckBox c2){
    String mesaj = "";
    if (c1.isSelected() && c2.isSelected()){
          mesaj = "Siparişiniz = Balık + Salata";
    } else if (c1.isSelected()){
          mesaj = "Siparişiniz = Balık";
    }
    else if (c2.isSelected()){
          mesaj = "Siparişiniz = Salata";
    }
    else {
          mesaj = "Sipariş vermediniz";
    }
          Alert mesajPenceresi = new Alert(AlertType.INFORMATION);
          mesajPenceresi.setHeaderText(null);
          mesajPenceresi.setContentText(mesaj);
          mesajPenceresi.showAndWait();
  }
}
```

ArayuzUygulama3OGP sınıfına ait program kodları çalıştırıldığında Resim 8.7'deki ekran görüntüsü ortaya çıkmaktadır. Ekran çıktısında balık seçimi yapıldığı için "Siparişiniz = Balık" ifadesinin yazıldığı görülmektedir. Programı farklı seçimlerle çalıştırdığımız takdirde bahsedilen diğer çıktıları da almamız mümkün olacaktır.

Aşağıda, *ArayuzUygulama4OGP* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf, bir önceki bölümdeki *ArayuzUygulama4* sınıfının içeriğine olay

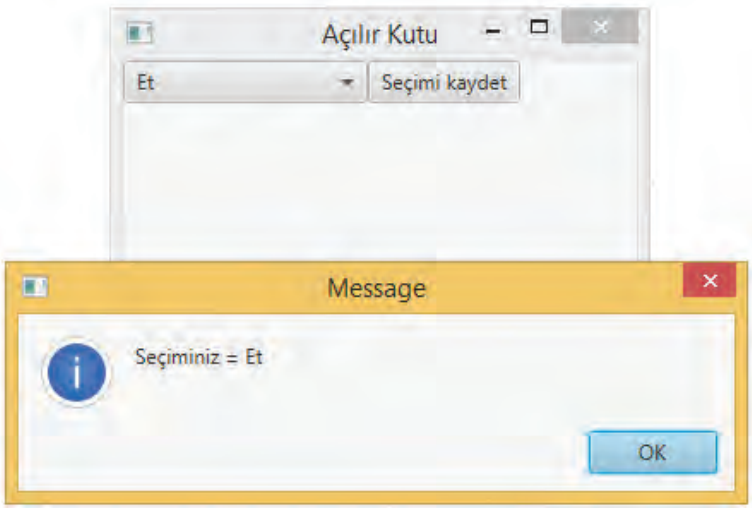
güdümlü programlama kavramı dâhil edilmiş versiyonudur. *ArayuzUygulama-4OGP* sınıfı içerisinde *dugme* bileşene ait *setOnAction* metodunun eklendiğini görmekteyiz. Düğmenin tıklanması olayı sonrasında bu metodun içeriği çalıştırılacaktır. Bu metodun içerisinde ise *islemYap* isimli metoda bir yönlendirme mevcuttur. Bu metodun içeriği incelendiği zaman açılır kutunun hangi elemanı seçilmiş ise ekrana onu içeren bir mesaj çıkarılacağı görülmektedir. Bu mesajların ayrı bir pencere ile görüntülenmesi için burada da *Alert* isimli sınıf kullanılmaktadır.



Resim 8.7 ArayuzUygulama3OGP sınıfı çalıştırılınca verdiği ekran çıktısı.

```
/* ArayuzUygulama4OGP.java */

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class ArayuzUygulama4OGP extends Application {
    ComboBox<String> acilirKutu;
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Açılır Kutu");
    Button dugme = new Button("Seçimi kaydet");
    dugme.setOnAction(e -> islemYap());
    acilirKutu = new ComboBox<>();
    acilirKutu.getItems().addAll(
        "Et",
        "Balık",
        "Salata"
    );
    acilirKutu.setPromptText("Ana yemek seçiniz!");
    HBox bilesenYerlesim = new HBox( );
    bilesenYerlesim.getChildren().addAll(acilirKutu, dugme);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
  private void islemYap(){
    String mesaj = "Seçiminiz = " + acilirKutu.getValue();
          Alert mesajPenceresi = new Alert(AlertType.INFORMATION);
          mesajPenceresi.setHeaderText(null);
          mesajPenceresi.setContentText(mesaj);
          mesajPenceresi.showAndWait();
  }
}
```

ArayuzUygulama4OGP sınıfına ait program kodları çalıştırıldığında Resim 8.8'deki ekran görüntüsü ortaya çıkmaktadır. Ekran çıktısında et seçimi yapıldığı için " Seçiminiz = Et" ifadesinin yazıldığı görülmektedir. Programı farklı seçimlerle çalıştırdığımız takdirde bahsedilen diğer çıktıları da almamız mümkün olacaktır.

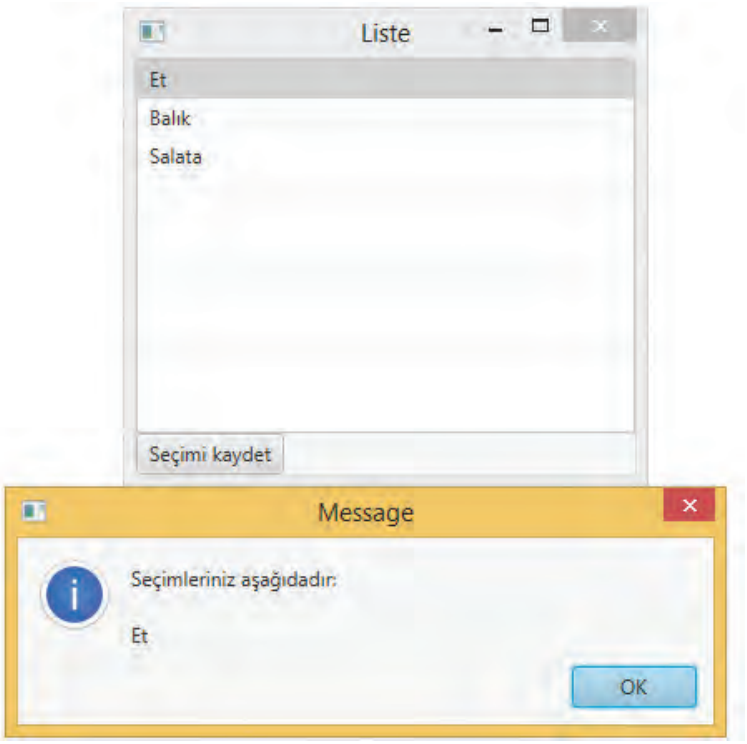


Resim 8.8 ArayuzUygulama4OGP sınıfı çalıştırılınca verdiği ekran çıktısı

Aşağıda, *ArayuzUygulama5OGP* isimli sınıfa ait Java kodları bulunmaktadır. Bu sınıf, bir önceki bölümdeki *ArayuzUygulama5* sınıfının içeriğine olay güdümlü programlama kavramı dâhil edilmiş versiyonudur. *ArayuzUygulama5OGP* sınıfı içerisinde *dugme* bileşene ait *setOnAction* metodunun eklendiğini görmekteyiz. Düğmenin tıklanması olayı sonrasında bu metodun içeriği çalıştırılacaktır. Bu metodun içerisinde ise *islemYap* isimli metoda bir yönlendirme mevcuttur. Bu metodun içeriği incelendiği zaman listenin hangi elemanları seçilmiş ise ekrana onları içeren bir mesaj çıkarılacağı görülmektedir. Bu mesajların ayrı bir pencere ile görüntülenmesi için burada da Alert isimli sınıf kullanılmaktadır.

```
/* ArayuzUygulama5OGP.java */

import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.scene.control.ListView;

public class ArayuzUygulama5OGP extends Application {
  ListView<String> liste;
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    stage.setTitle("Liste");
    Button dugme = new Button("Seçimi kaydet");
    dugme.setOnAction(e -> islemYap());
    liste = new ListView<>();
    liste.getItems().addAll("Et", "Balık", "Salata");
liste.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
    VBox bilesenYerlesim = new VBox( );
    bilesenYerlesim.getChildren().addAll(liste, dugme);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
  private void islemYap(){
  String mesaj = "Seçimleriniz aşağıdadır: \n \n";
  ObservableList<String> secimler =
liste.getSelectionModel().getSelectedItems();
  for(String s: secimler) {
    mesaj += s + "\n";
  }
  Alert mesajPenceresi = new Alert(AlertType.INFORMATION);
    mesajPenceresi.setHeaderText(null);
    mesajPenceresi.setContentText(mesaj);
    mesajPenceresi.showAndWait();
  }
}
```

ArayuzUygulama5OGP sınıfına ait program kodları çalıştırıldığında Resim 8.9'daki ekran görüntüsü ortaya çıkmaktadır. Ekran çıktısında et seçimi yapıldığı için bununla ilgili ifadenin yazıldığı görülmektedir. Programı farklı seçimlerle çalıştırdığımız takdirde farklı çıktılar da almamız mümkündür.



Resim 8.9 ArayuzUygulama5OGP sınıfı çalıştırılınca verdiği ekran çıktısı.

Öğrenme Çıktısı

3 Olay güdümlü programlama kavramını açıklayabilme

Araştır 3

Olay güdümlü programlama için tıpkı bilgisayarda fareye tıklanması gibi değişik örnekleri araştırınız ve bir tanesini açıklayınız.

İlişkilendir

Olay güdümlü programlama ile kullanıcı arayüzü tabanlı programlamayı ilişkilendirin.

Anlat/Paylaş

Olay güdümlü programlama kavramının açıklanması için verilen örnek olan bir insanın vazoya çarpıp onu yere düşürmesi aktivitesine benzer değişik günlük hayat örnekleri düşünün.

1 JavaFX kütüphanesini tanımlayabilme

JavaFX Temelleri

Kitabın bu bölümünde kullanıcı arayüzü tabanlı programlama kavramı ele alınmıştır. Bilgisayar programlarının çalışmaları sırasında genellikle kullanıcılarla etkileşim hâlinde olmaları söz konusudur. Kullanıcılardan bir takım bilgiler alıp bunlara bağlı çıktılar üretirler. Bu iletişim için kullanıcı arayüzlerinin kullanılması gereklidir. Java programlama dilinde kullanıcı arayüzü tasarımı için zaman içerisinde çeşitli kütüphaneler ortaya konmuştur. Bu ünitede, programcıların daha hızlı ve etkili şekilde kullanıcı arayüzü oluşturmaları adına inşa edilen *JavaFX* isimli kütüphane anlatılmıştır.

2 Kullanıcı arayüzü tabanlı programlamayı açıklayabilme

Kullanıcı Arayüzü Tasarımı

JavaFX kütüphanesi düğme, işaret kutusu, açılır kutu, etiket, liste, şifre alanı, radyo düğmesi, kaydırma çubuğu ve metin alanı gibi çeşitli arayüz bileşenlerine sahiptir. Aslında çoğu temel bilgisayar kullanıcısı bu bileşenlerden bazıları ile günlük bilgisayar kullanımlarında karşı karşıya kalmaktadır fakat bu bileşenlerin isimleri ve kullanım amaçları konusunda çok fazla bilgiye sahip değildir. Örneğin bir düğme bileşenini ofis yazılımları kullanan birçok kişi görmesine ve kullanmasına rağmen teknik olarak bu konuda bilgi sahibi olmayabilir. *JavaFX* uygulamaları yazmak için öncelikli olarak *Application* sınıfından kalıtım yoluyla türeyen bir sınıf oluşturulması gereklidir. Diğer bir taraftan da bu sınıfın içerisindeki *start* isimli metodun ezilmesi ve dolayısıyla içeriğinin yeniden tanımlanması gereklidir. Bahsedilen *start* metodu Stage sınıfı türünde bir parametre almaktadır. Stage sınıfı, ekranda bir form oluşmasına olanak sağlamaktadır. Birden fazla form oluşturulması söz konusu ise Stage sınıfından istenilen sayıda kullanılması gerekecektir. Bunun yanısıra, bileşenlerin ekranda farklı şekilde yerleşimlerinin sağlanması için *JavaFX* kütüphanesi içerisinde *HBox*, *VBox*, *BorderPane*, *GridPane* gibi çeşitli sınıflar bulunmaktadır. Bu sınıflar içerisinden *Hbox* bileşenlere tek bir satırda yatay olarak yerleşim sağlamaktadır. Diğerleri de farklı yerleşim olanaklarını sunarak ekran görünümünü çeşitlendirmektedirler. Arayüz oluşturmak tek başına kullanıcı etkileşimi yaratmak için yeterli değildir.

öğrenme çıktıları ve bölüm özeti

3 Olay güdümlü programlama kavramını açıklayabilme

Olay Güdümlü Programlama

Ünitenin devamında bu amaçla kullanılan olay güdümlü programlama kavramından bahsettik. Olay kelimesini herhangi bir aktivitenin gerçekleşmesi, bu aktivitenin gerçekleşmesine sebep olan varlığı da olayın kaynağın olarak adlandırabiliriz. Örneğin bir insanın vazoya çarpıp onu yere düşürdüğü durumda vazonun yere düşmesi bir olayı ifade ederken vazoyu yere düşüren insan olayın kaynağını ifade etmektedir. Nesneye yönelik programlama mantığı ile yazılmış bilgisayar programları da benzer şekilde çalışmaktadır. Örneğin fareyi bir düğmenin üzerine getirip tıkladığımızda düğmenin üzerine tıklanması bir olayı ifade ederken fare o olayın kaynağıdır. Bir diğer taraftan da bu olay gerçekleştikten sonra ne yapılması gerektiği tanımlanmış olmalıdır. Bu tanımlamayı da olay yönetimi olarak isimlendirmekteyiz. Fare tarafından düğme tıklandığı zaman ekranda bir mesaj görüntülenmesi örneğinde mesaj görüntülenmesi olay yönetimi kavramını ifade etmektedir. Kullanıcı etkileşiminin sağlanması ile ilgili kavramların tamamını ise olay güdümlü programlama olarak adlandırmaktayız.

1 Aşağıdakilerden hangisi Java programlama dilinde kullanıcı arayüzü oluşturmada kullanılan kütüphanelerden biridir?

A. Gx
B. Hx
C. Swing
D. Bwt
E. Wing

2 Aşağıdakilerden hangisi *JavaFX* kütüphanesi içerisinde arayüz bileşeni oluşturmak amacıyla kullanılan sınıflardan biri **değildir?**

A. Label
B. Button
C. TextField
D. ComboBox
E. Exception

3 Aşağıdakilerden hangisi *JavaFX* kütüphanesi içerisinde şifre alanı isimli arayüz bileşeni oluşturmak amacıyla kullanılan sınıflardan biridir?

A. PasswordField
B. RadioButton
C. TextField
D. ListView
E. Label

4 *JavaFX* ile kullanıcı arayüzü oluşturmak için aşağıdaki sınıflardan hangisinden kalıtım yoluyla türeyen yeni bir sınıf oluşturulmalıdır?

A. Stage
B. Scene
C. Application
D. Exception
E. Label

5 *JavaFX* kullanıcı arayüzü formunun başlığını belirlemek için kullanılan metot aşağıdakilerden hangisidir?

A. setWidth
B. setHeight
C. setTitle
D. getChildren
E. addAll

6 *JavaFX* bileşenlerinin tek bir sütunda dikey olarak yerleşmelerine olanak sağlayan sınıf aşağıdakilerden hangisidir?

A. VBox
B. HBox
C. BorderPane
D. TextField
E. Button

7 Fare ile kullanıcı arayüzünden bir liste elemanı seçilmesi sonrasında ekranda bir mesaj görüntülenmesi durumunda olay ve olayın kaynağı aşağıdakilerden hangisinde sırasıyla ve doğru olarak verilmiştir?

A. Kullanıcı arayüzünden bir liste elemanı seçilmesi, fare
B. Fare, kullanıcı arayüzünden bir liste elemanı seçilmesi
C. Ekranda mesaj görüntülenmesi, fare
D. Fare, ekranda mesaj görüntülenmesi
E. Ekranda mesaj görüntülenmesi, kullanıcı arayüzünden bir liste elemanı seçilmesi

neler öğrendik?

8

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class KendimiziSinayalim1 extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
     stage.setTitle("Soru");
    TextField bilesen1 = new TextField( );
    CheckBox bilesen2 = new CheckBox("Seç");
    Button bilesen3 = new Button("Gönder");
    VBox bilesenYerlesim = new VBox( );
   bilesenYerlesim.getChildren().addAll(bilesen1, bilesen2, bilesen3);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
}
```

Yukarıdaki Java program kodu çalıştırıldığında vereceği çıktı ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. Metin alanı, işaret kutusu ve düğme içeren bir arayüz oluşur
B. Metin alanı, işaret kutusu ve açılır kutu içeren bir arayüz oluşur
C. Metin alanı ve etiket içeren bir arayüz oluşur
D. Radyo düğmesi ve liste içeren bir arayüz oluşur
E. Radyo düğmesi ve kaydırma çubuğu içeren bir arayüz oluşur

9

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class KendimiziSinayalim2 extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
     stage.setTitle("Soru");
    TextField bilesen1 = new TextField( );
    ComboBox<String> bilesen2 = new ComboBox<>();
    bilesen2.getItems().addAll(
        "eleman1",
        "eleman2"
    );
    Button bilesen3 = new Button("Gönder");
    VBox bilesenYerlesim = new VBox( );
   bilesenYerlesim.getChildren().addAll(bilesen1, bilesen2, bilesen3);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
}
```

Yukarıdaki Java program kodu çalıştırıldığında vereceği çıktı ile ilgili aşağıdaki ifadelerden hangisi doğrudur?

A. Metin alanı, işaret kutusu ve düğme içeren bir arayüz oluşur.
B. Metin alanı, liste ve düğme içeren bir arayüz oluşur.
C. Metin alanı, açılır kutu ve düğme içeren bir arayüz oluşur.
D. Şifre alanı ve liste içeren bir arayüz oluşur.
E. Şifre alanı, kaydırma çubuğu ve etiket içeren bir arayüz oluşur.

10

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class KendimiziSinayalim3 extends Application {
  public static void main(String[] args) {
    launch(args);
  }
  public void start(Stage stage) throws Exception {
    Button dugmeGonder = new Button("Sipariş ver");
    dugmeGonder.setOnAction(e -> metotB( ));
    VBox bilesenYerlesim = new VBox( );
    bilesenYerlesim.getChildren().addAll(dugmeGonder);
    Scene scene = new Scene(bilesenYerlesim, 300, 250);
    stage.setScene(scene);
    stage.show();
  }
  private void metotA( ){
    System.out.println("metotA");
  }
  private void metotB( ){
    System.out.println("metotB");
  }
  private void metotC( ){
    System.out.println("metotC");
  }
  private void metotD( ){
    System.out.println("metotD");
  }
  private void metotE( ){
    System.out.println("metotE");
  }
}
```

Yukarıdaki Java program kodu çalıştırıldığında ve ekrandaki düğmeye fare ile tıklandığında aşağıdaki metotlardan hangisi çalışır?

A. metotA
B. metotB
C. metotC
D. metotD
E. metotE

1. C	Yanıtınız yanlış ise "JavaFX Temelleri" konusunu yeniden gözden geçiriniz.
2. E	Yanıtınız yanlış ise "JavaFX Temelleri" konusunu yeniden gözden geçiriniz.
3. A	Yanıtınız yanlış ise "JavaFX Temelleri" konusunu yeniden gözden geçiriniz.
4. C	Yanıtınız yanlış ise "JavaFX Temelleri" konusunu yeniden gözden geçiriniz.
5. C	Yanıtınız yanlış ise "JavaFX Temelleri" konusunu yeniden gözden geçiriniz.
6. A	Yanıtınız yanlış ise "JavaFX Temelleri" konusunu yeniden gözden geçiriniz.
7. C	Yanıtınız yanlış ise "JavaFX Temelleri ve Kullanıcı arayüzü Tasarımı" konusunu yeniden gözden geçiriniz.
8. A	Yanıtınız yanlış ise "Olay Güdümlü Programlama" konusunu yeniden gözden geçiriniz.
9. A	Yanıtınız yanlış ise "JavaFX Temelleri ve Kullanıcı Arayüzü Tasarımı" konusunu yeniden gözden geçiriniz.
10. B	Yanıtınız yanlış ise "Kullanıcı Arayüzü Tasarımı ve Olay Güdümlü Programlama" konusunu yeniden gözden geçiriniz.

8 Araştır Yanıt Anahtarı

Araştır 1

JavaFX kütüphanesi içerisinde yer alıp bu bölümde bahsedilmeyen 3 adet kullanıcı arayüzü bileşeni örneği aşağıda listelenmektedir.

Java sınıfı	İsim
MenuBar	**Menü çubuğu**
ProgressBar	**İlerleme çubuğu**
ToolBar	**Araç çubuğu**

Araştır 2

Java'da diyalog kutusu oluşturmak amacıyla **Alert** isimli sınıfı kullanabiliriz. Bu sınıftan uygun şekilde nesneler oluşturarak "Evet", "Hayır, "Tamam" gibi onaylanması gereken ekranların kullanıcının karşısına çıkmasını sağlayabiliriz.

Araştır 3

Farenin ekranda hareket etmesi de bir olay örneği olarak verilebilir. Fare ekranda hareket ettikçe üzerinden geçtiği noktaların takip edilmesi ve bu noktaların koordinatlarının olay güdümlü programlama kullanılarak ekrana yazdırılabilmesi mümkündür.

Kaynakça

Sharan, K. (2015). *Learn JavaFX 8 Building User Experience and Interfaces with Java 8*, Apress yayınları.