



Bölüm 3

Sınıflara ve Nesnelere, Nesne Metodlarına Giriş

Java™ How to Program, 10/e

3.1 Introduction

3.2 Instance Variables, *set* Methods and *get* Methods

- 3.2.1 **Account** Class with an Instance Variable, a *set* Method and a *get* Method
- 3.2.2 **AccountTest** Class That Creates and Uses an Object of Class **Account**
- 3.2.3 Compiling and Executing an App with Multiple Classes
- 3.2.4 **Account** UML Class Diagram with an Instance Variable and *set* and *get* Methods
- 3.2.5 Additional Notes on Class **AccountTest**
- 3.2.6 Software Engineering with **private** Instance Variables and **public** *set* and *get* Methods

3.3 Primitive Types vs. Reference Types

3.4 **Account** Class: Initializing Objects with Constructors

- 3.4.1 Declaring an **Account** Constructor for Custom Object Initialization
- 3.4.2 Class **AccountTest**: Initializing **Account** Objects When They're Created

3.5 **Account** Class with a Balance; Floating-Point Numbers

- 3.5.1 **Account** Class with a **balance** Instance Variable of Type **double**
- 3.5.2 **AccountTest** Class to Use Class **Account**

3.6 (Optional) GUI and Graphics Case Study: Using Dialog Boxes

3.7 Wrap-Up



3.2 Instance Variables (Örnek Değişkenleri), set Methods and get Methods (set ve get Metodları)

- ▶ Yarattığınız her sınıf, ileride değişken tanımlamak ve nesneler yaratmak için kullanabileceğiniz yeni bir tip oluşturmaktadır.
- ▶ İhtiyaç duydukça yeni sınıflar tanımlayabilirsiniz. Java'nın extensible (genişleyebilir) bir dil olarak bahsedilmesinin nedeni budur.

3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

```
1 // Fig. 3.1: Account.java
2 // Account class that contains a name instance variable
3 // and methods to set and get its value.
4
5 public class Account
6 {
7     private String name; // instance variable
8
9     // method to set the name in the object
10    public void setName(String name)
11    {
12        this.name = name; // store the name
13    }
14
15    // method to retrieve the name from the object
16    public String getName()
17    {
18        return name; // return value of name to caller
19    }
20 } // end class Account
```

Fig. 3.1 | Account class that contains a name instance variable and methods to set and get its value.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Sınıf Tanımlama Class Declaration

- ▶ Her sınıf tanımlaması access modifier ile başlamaktadır. Sınıf ismi ile aynı isme sahip .java uzantılı dosyada depolanmalıdır.
- ▶ Her sınıf tanımlaması `class` anahtar kelimesi ile gerçekleştirilir.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Identifer (Belirleyiciler) ve Camel Case İsimlendirme

- ▶ Sınıf, metod ve değişken isimleri, belirleyicilerdir.
- ▶ Genel olarak hepsinde camel case isimlendirme kullanılır.
- ▶ Sınıf isimleri **büyük** harf ile başlar metod ve değişken isimleri ise küçük harf ile başlamaktadır.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Instance Variable name

- ▶ Bir nesnenin hayatı boyunca taşıyabildiği özellikleri vardır.
- ▶ Bunlar instance variable'lar olarak bilinir.
- ▶ Instance Variable'lar metodlar çağrılmadan önce, çağrıldığında ve metod işlemesi sonlandığında varlıklarını kaybetmez.
- ▶ Bir sınıf genellikle instance variable'ları üzerinde değişiklik yapan bir veya daha fazla metoda sahiptir.
- ▶ Instance variable'lar sınıf tanımlamasının içerisinde ancak method tanımlamalarının dışında tanımlanır.
- ▶ Her bir nesne (instance) kendi instance variable'ının kopyasına sahiptir.

Öneri

Örnek değişkenleri (Instance Variable) sınıfın üst kısmında listelemeyi tercih ediniz, böylelikle metotlarda kullanılabilecek olan örnek değişkenleri önceden görmüş olursunuz, örnek değişkenlerin sınıf içerisinde farklı yerlere yayılması program okunurluğunu azaltmaktadır.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Erişim Belirleyicileri (Access Modifiers) public ve private

- ▶ Çoğu *instance variable* tanımı `private` anahtar kelimesi ile başlamaktadır.
- ▶ Bu anahtar kelime örnek değişkenlerin sadece o sınıf içerisinde erişilmesine izin vermektedir.
- ▶



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Account sınıfının setName metodu

- ▶ Parametreler virgül ile ayrılarak metotların parametre listesini oluşturur.
- ▶ Her parametrenin ilgili veri tipinin belirtilmesi gerekmektedir.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Parametreler yerel (local) ,

- ▶ Tek bir metodun içerisinde tanımlanan değişkenler yerel değişkenlerdir ve sadece o metod içerisinde erişilebilirler.
- ▶ Metod sona erdiğinde, bu metodun yerel değişkenlerine erişilemez.
- ▶ Bir metodun parametreleri de o metodun yerel değişkenleri olarak kabul edilir.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

setName Metod Body'si

Her metodun body'si ({ }) parantezleri içerisinde bulunmaktadır.

Bir metodun body'si bir veya daha fazla bildirim (statement) içermektedir .



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

Account sınıfının getName metodu

- ▶ Bu metodun geri dönüş tipi ilgili instance variable'ın tipi ile aynıdır.
- ▶ `Void` anahtar kelimesi metodun içerisinde işlemlerin gerçekleştirileceğini ancak herhangi bir bilginin geriye döndürülmeyeceğini belirtmektedir.
- ▶ Metodun yanındaki içi boş parantezler metodun herhangi bir parametre almadığını vurgulamaktadır.
- ▶ Void olmayan metodlar içerisinde bir değer döndürme işlemi gerçekleştirilmelidir.



3.2.1 Account Sınıfı (Örnek Değişkeni ve get ve set metodu)

- ▶ `return` bildirimi, metodun çağrıldığı noktaya değer döndürmektedir.
- ▶ Sınıflar `private set` ve `get` metodları ile `private instance variable`'lara dışarıdan erişmeye izin verir.
- ▶ Bu metodların isimleri `set` veya `get` ile başlamak zorunda değildir ama bu isimlendirme yazılımcılar arasında bir alışkanlık haline gelmiştir.

3.2.2 Account sınıfının bir nesnesini oluşturan ve bu nesneyi kullanan AccountTest Sınıfı

AccountTest Sınıfı (Sürücü Sınıfı)

- ▶ Bir sınıf başka bir sınıfın bir nesnesini yaratıp bu nesnenin metodlarını çağırıyorsa bu sınıf kullandığı sınıfın sürücü sınıfıdır.



```
1 // Fig. 3.2: AccountTest.java
2 // Creating and manipulating an Account object.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     public static void main(String[] args)
8     {
9         // create a Scanner object to obtain input from the command window
10        Scanner input = new Scanner(System.in);
11
12        // create an Account object and assign it to myAccount
13        Account myAccount = new Account();
14
15        // display initial value of name (null)
16        System.out.printf("Initial name is: %s%n%n", myAccount.getName());
17
18        // prompt for and read name
19        System.out.println("Please enter the name:");
20        String theName = input.nextLine(); // read a line of text
21        myAccount.setName(theName); // put theName in myAccount
22        System.out.println(); // outputs a blank line
23    }
```

Fig. 3.2 | Creating and manipulating an Account object. (Part 1 of 2.)



```
24      // display the name stored in object myAccount
25      System.out.printf("Name in object myAccount is:%n%s%n",
26          myAccount.getName());
27  }
28 } // end class AccountTest
```

Initial name is: null

Please enter the name:

Jane Green

Name in object myAccount is:

Jane Green

Fig. 3.2 | Creating and manipulating an Account object. (Part 2 of 2.)

3.2.2 Account sınıfının bir nesnesini oluşturan ve bu nesneyi kullanan AccountTest Sınıfı

Kullanıcıdan girdi almak için Scanner Nesnesi

- ▶ Scanner sınıfının `nextLine` metodu kullanıcı tarafından bir sonraki satıra kadar olan karakterleri alır ve okur, bu karakterleri `String` olarak döndürür.
- ▶ Scanner sınıfının `next` metodu ise tek bir kelime okur yani ilk boşluk (white-space) karakterini görene kadar karakterleri `String` dönüştürür.
- ▶ reads characters until any white-space character is encountered, then returns the characters as a `String`.

3.2.2 Account sınıfının bir nesnesini oluşturan ve bu nesneyi kullanan AccountTest Sınıfı



Bir nesne yaratmak new anahtar kelimesi ve kurucu (Constructors) metodlar

Bir sınıfın nesnesinin yaratılması **new** anahtar kelimesi ile başlamaktadır.

- ▶ Bir kurucu metod (constructor) normal bir metoda benzemektedir ve new anahtar kelimesi ile birlikte üstü kapalı bir şekilde çağrılmaktadır.

3.2.2 Account sınıfının bir nesnesini oluşturan ve bu nesneyi kullanan AccountTest Sınıfı

null—Stringler için varsayılan ilk değer

- ▶ **Yerel değişkenler** otomatik bir şekilde **ilklendirilmezler** (ilk değerlerini almazlar)
- ▶ Java tarafından her instance variable'ına bir varsayılan ilk değer atanmaktadır.
- ▶ String tipinde bir instance variable'ın varsayılan ilk değeri `null` 'dır.

3.2.3 Birden fazla sınıfı olan bir uygulamayı derlemek ve çalıştırmak

- ▶ Javac komutu aynı anda birden fazla sınıf derleyebilir.
- ▶ Komuttan sonra, kaynak kodu dosya adlarını, her boşluk bırakarak listeleyin.
- ▶ Uygulamayı içeren dizinde yalnızca bir uygulamanın dosyaları varsa, tüm sınıflarını `javac *.java` komutuyla derleyebilirsiniz.
- ▶ `*.java` içindeki yıldız (*), geçerli dizinde dosya adı uzantısı `".java"` ile biten tüm dosyaların derlenmesi gerektiğini gösterir.

3.2.4 Account UML Class Diagramı

En Üst Kısım

- ▶ UML'de, her sınıf bir sınıf diyagramında üç bölmeli dikdörtgen olarak modellenmiştir. Üstteki kısım, sınıfın adını yatay olarak koyu renkte ortalanmış halde içerir.

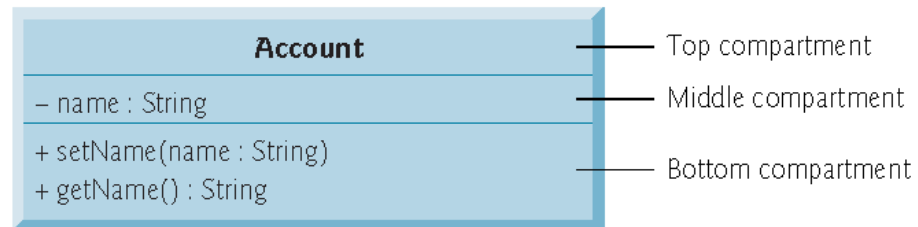


Fig. 3.3 | UML class diagram for class **Account** of Fig. 3.1.



3.2.4 Account UML Class Diagramı

Orta Kısım

- ▶ Orta bölme, Java'daki örnek değişkenlerine (instance variable) karşılık gelen sınıfın niteliklerini (attribute) içerir.

3.2.4 Account UML Class Diagramı

En Alt Kısım

- ▶ Alt bölme, Java'daki metot ve kuruculara karşılık gelen sınıfın metodlarını içerir.
- ▶ UML, örnek değişkenleri bir öznitelik adı, ardından iki nokta üst üste ve türünü temsil eder.
- ▶ Private niteliklerin önüne UML'de eksi işareti (-) gelir.
- ▶ UML modelleri metot adını ve ardından bir parantez dizisini listeleyerek işlemleri gerçekleştirir.
- ▶ İşlem adının önünde bir artı işareti (+), işlemin UML'de herkese açık olduğunu gösterir (diğer bir deyişle, Java'daki public bir metod).

3.2.4 Account UML Class Diagramı

Dönüş Tipleri

- ▶ UML, işlem adını takip eden parantezlerin ardından iki nokta üst üste ve dönüş türünü yerleştirerek bir işlemin dönüş türünü belirtir.
- ▶ UML sınıf diyagramları, değerleri döndürmeyen işlemler için dönüş türleri belirtmez.
- ▶ Örnek değişkenleri *private* olarak tanımlamak, veri gizleme veya bilgi gizleme olarak bilinir.



3.2.4 Account UML Class Diagram with an Instance Variable and *set* and *get* Methods (Cont.)

Parametreler

- ▶ UML, bir metodun parametresini, parametre adını, ardından bir iki nokta üst üste ve işlem adı sonra parantezler arasındaki parametre türünü listeleyerek modeller

3.2.5 AccountTest Sınıfı Hakkında Ek Bilgiler

static Method main

- ▶ Main dışındaki metotların çalıştırılması için açıkça çağırılması gerekir.
- ▶ JVM'nin uygulamanın yürütülmeye başlaması için main metodunu bulmasını ve çağırmasını sağlayan nokta, main'in, metodun sınıfın bir nesnesi olmadan önce çağrılabilen **static** bir metot olduğunu belirten **static** anahtar kelimesine sahip olmasıdır.

3.2.5 AccountTest Sınıfı Hakkında Ek Bilgiler

import Tanımları ile İlgili Notlar

- ▶ Java programlarında kullanacağınız sınıfların çoğunun açıkça import edilmesi gerekir.
- ▶ Aynı dizinde derlenen sınıflar arasında özel bir ilişki vardır.
- ▶ Varsayılan olarak, bu tür sınıfların, default paket olarak bilinen aynı pakette olduğu kabul edilir.
- ▶ Aynı pakette bulunan sınıflar, bu paketdeki diğer sınıfların kaynak kodu dosyalarını üstü kapalı olarak import eder.

3.2.5 AccountTest Sınıfı Hakkında Ek Bilgiler

- ▶ Bir paketteki bir sınıfta aynı paketten başka bir sınıf kullanıldığında bir import tanımlaması gerekli değildir.
- ▶ Paket ismi ve sınıf adını içeren tam bir sınıf adıyla bir sınıfa eriştiğinizde, bir import tanımlaması gerekli değildir.



3.2.6 `private` Örnek Değişkenler

- ▶ Örnek değişkenleri `private` olarak tanımlamaya `data hiding` ya da `information hiding` denilmektedir.



Öneri

Her bir örnek değişkeni ve metod bildirimini bir erişim belirleyicisi (access modifier) ile başlatın.

Genellikle, örnek değişkenler `private` ve yöntemler `public` olarak bildirilmelidir.

İlerleyen bölümlerde, neden özel bir metod tanımlamak isteyebileceğinizi tartışacağız.

3.3 Primitive Tipler ve Reference Tipleri



- ▶ Java'daki tipler iki kategoriye ayrılır: ilkel tipler ve referans tipleri.
- ▶ İlkel tipler **boolean, byte, char, short, int, long, float** ve **double**'dir.
- ▶ Diğer tüm tipler referans tipleridir, bu nedenle nesnelerin tiplerini belirten sınıflar referans tipleridir.
- ▶ İlkel türdeki değişken, aynı anda bildirilen türünün tam olarak bir değerini depolayabilir. İlkel türdeki örnek değişkenleri varsayılan olarak başlatılır. Değişkenler **byte, char, short, int, long, float ve double** değişkenleri 0'la başlatılır.

3.3 Primitive Tipler ve Reference Tipleri



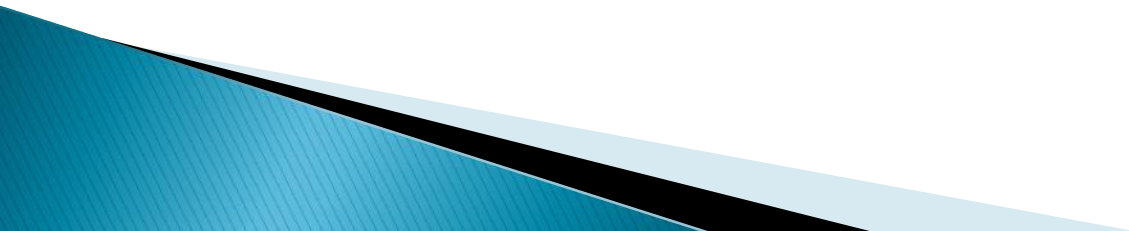
- ▶ **Boolean** türündeki değişkenler *false* olarak ilk değerini alır.
- ▶ Referans tipi değişkenler (referanslar denir), nesnenin bilgisayarın belleğindeki yerini tutar.
- ▶ Bu değişkenler program içerisindeki nesneye erişebilir. Erişilen nesne birçok örnek değişkeni ve metodu içerebilir.
- ▶ Referans tipinde örnek değişkenleri varsayılan olarak **null** değerine başlatılır. Nesnenin metotlarını çağırmak için bir nesneye bir referans gereklidir.
- ▶ İlkel bir değişken, bir nesneye işaret etmemektedir ve bu nedenle bir metodu çağırmak için kullanılamaz.

3.4 Account Sınıfı: Nesneleri Constructorlar (Kurucular) ile ilklendirme



- ▶ Tanımladığınız her sınıf, isteğe bağlı olarak, nesne oluşturulduğunda bir sınıfın bir nesnesini başlatmak için kullanılabilecek parametreleri olan bir kurucu sağlayabilir.
- ▶ Java, oluşturulan her nesne için kurucu (constructor) çağrısı gerektirir.

3.4.1 Özel bir Nesne yaratmak için Account Constructor'ı Bildirmek





```
1 // Fig. 3.5: Account.java
2 // Account class with a constructor that initializes the name.
3
4 public class Account
5 {
6     private String name; // instance variable
7
8     // constructor initializes name with parameter name
9     public Account(String name) // constructor name is class name
10    {
11        this.name = name;
12    }
13
14    // method to set the name
15    public void setName(String name)
16    {
17        this.name = name;
18    }
19
20    // method to retrieve the name
21    public String getName()
22    {
23        return name;
24    }
25 } // end class Account
```

Fig. 3.5 | Account class with a constructor that initializes the name.



3.4.2 Class AccountTest: Initializing Account Objects When They're Created

```
1 // Fig. 3.6: AccountTest.java
2 // using the Account constructor to initialize the name instance
3 // variable at the time each Account object is created.
4
5 public class AccountTest
6 {
7     public static void main(String[] args)
8     {
9         // create two Account objects
10        Account account1 = new Account("Jane Green");
11        Account account2 = new Account("John Blue");
12
13        // display initial value of name for each Account
14        System.out.printf("account1 name is: %s\n", account1.getName());
15        System.out.printf("account2 name is: %s\n", account2.getName());
16    }
17 } // end class AccountTest
```

```
account1 name is: Jane Green
account2 name is: John Blue
```

Fig. 3.6 | Using the Account constructor to initialize the name instance variable at the time each Account object is created.



3.4.2 Class AccountTest: Initializing Account Objects When They're Created (Cont.)

Constructorlar Değerler Döndüremez

Constructorlar parametreleri belirtebilir, ancak dönüş tiplerini belirtmez.

Varsayılan Constructor

- ▶ Bir sınıf herhangi bir kurucu tanımlamazsa, derleyici parametre içermeyen bir varsayılan bir kurucu sağlar ve sınıfın örnek değişkenleri varsayılan değerleri ile başlatılır.
- ▶ ***Bir Constructor Tanımlanan Bir Sınıfta Varsayılan Constructor Metodu Yoktur.***
- ▶ Bir sınıf için bir constructor bildirirseniz, derleyici o sınıf için varsayılan bir constructor oluşturmaz.

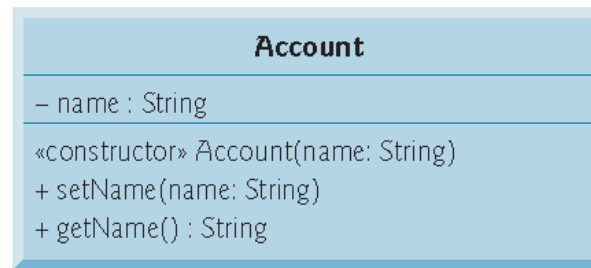


Fig. 3.7 | UML class diagram for **Account** class of Fig. 3.5.



3.5 Account Class with a Balance; Floating-Point Numbers and Type `double`

- ▶ floating-point sayı ondalıklı sayıları belirler.
- ▶ Java ondalık sayılar için iki farklı primitive tip sağlar `float` ve `double`.
- ▶ `float` tipi 7 haneye kadar duyarlılık sağlar.
- ▶ `double` ise `float` değişkenlerin yaklaşık olarak iki katı kadar duyarlılık sağlar (15 haneye kadar duyarlılık)
Floating-point literaller varsayılan olarak `double`'dir.

3.5.1 Account Class double balance Instance Variable



```
1 // Fig. 3.8: Account.java
2 // Account class with a double instance variable balance and a constructor
3 // and deposit method that perform validation.
4
5 public class Account
6 {
7     private String name; // instance variable
8     private double balance; // instance variable
9
10    // Account constructor that receives two parameters
11    public Account(String name, double balance)
12    {
13        this.name = name; // assign name to instance variable name
14
15        // validate that the balance is greater than 0.0; if it's not,
16        // instance variable balance keeps its default initial value of 0.0
17        if (balance > 0.0) // if the balance is valid
18            this.balance = balance; // assign it to instance variable balance
19    }
20
```

Fig. 3.8 | Account class with a double instance variable balance and a constructor and deposit method that perform validation. (Part 1 of 3.)

```
21 // method that deposits (adds) only a valid amount to the balance
22 public void deposit(double depositAmount)
23 {
24     if (depositAmount > 0.0) // if the depositAmount is valid
25         balance = balance + depositAmount; // add it to the balance
26 }
27
28 // method returns the account balance
29 public double getBalance()
30 {
31     return balance;
32 }
33
34 // method that sets the name
35 public void setName(String name)
36 {
37     this.name = name;
38 }
39
```

Fig. 3.8 | Account class with a `double` instance variable `balance` and a constructor and `deposit` method that perform validation. (Part 2 of 3.)



```
40    // method that returns the name
41    public String getName()
42    {
43        return name; // give value of name back to caller
44    } // end method getName
45 } // end class Account
```

Fig. 3.8 | Account class with a double instance variable balance and a constructor and deposit method that perform validation. (Part 3 of 3.)



3.5.2 AccountTest Class

```
1 // Fig. 3.9: AccountTest.java
2 // Inputting and outputting floating-point numbers with Account objects.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     public static void main(String[] args)
8     {
9         Account account1 = new Account("Jane Green", 50.00);
10        Account account2 = new Account("John Blue", -7.53);
11
12        // display initial balance of each object
13        System.out.printf("%s balance: $%.2f%n",
14            account1.getName(), account1.getBalance());
15        System.out.printf("%s balance: $%.2f%n%n",
16            account2.getName(), account2.getBalance());
17    }
18 }
```

Fig. 3.9 | Inputting and outputting floating-point numbers with Account objects.
(Part 1 of 4.)



```
18 // create a Scanner to obtain input from the command window
19 Scanner input = new Scanner(System.in);
20
21 System.out.print("Enter deposit amount for account1: "); // prompt
22 double depositAmount = input.nextDouble(); // obtain user input
23 System.out.printf("%nadding %.2f to account1 balance%n%n",
24     depositAmount);
25 account1.deposit(depositAmount); // add to account1's balance
26
27 // display balances
28 System.out.printf("%s balance: $%.2f%n",
29     account1.getName(), account1.getBalance());
30 System.out.printf("%s balance: $%.2f%n%n",
31     account2.getName(), account2.getBalance());
32
33 System.out.print("Enter deposit amount for account2: "); // prompt
34 depositAmount = input.nextDouble(); // obtain user input
35 System.out.printf("%nadding %.2f to account2 balance%n%n",
36     depositAmount);
37 account2.deposit(depositAmount); // add to account2 balance
38
```

Fig. 3.9 | Inputting and outputting floating-point numbers with Account objects.
(Part 2 of 4.)



```
39      // display balances
40      System.out.printf("%s balance: $%.2f%n",
41          account1.getName(), account1.getBalance());
42      System.out.printf("%s balance: $%.2f%n%n",
43          account2.getName(), account2.getBalance());
44  } // end main
45 } // end class AccountTest
```

Jane Green balance: \$50.00

John Blue balance: \$0.00

Enter deposit amount for account1: 25.53

adding 25.53 to account1 balance

Jane Green balance: \$75.53

John Blue balance: \$0.00

Fig. 3.9 | Inputting and outputting floating-point numbers with `Account` objects.
(Part 3 of 4.)



```
Enter deposit amount for account2: 123.45
```

```
adding 123.45 to account2 balance
```

```
Jane Green balance: $75.53
```

```
John Blue balance: $123.45
```

Fig. 3.9 | Inputting and outputting floating-point numbers with `Account` objects.
(Part 4 of 4.)

3.5.2 AccountTest Class to Use Class Account (Cont.)

- ▶ Scanner sınıfının nextDouble metodu a double bir değer döndürmektedir.

3.5.2 AccountTest Class to Use Class Account (Cont.)



Floating-Point Sayıları Formatlama

- ▶ float or double tipi %f ile yazdırılır.
- ▶ %.2f



3.5.2 AccountTest Class

- ▶ `double` varsayılan değeri `0.0`, and `int` varsayılan değeri `0` 'dır.

3.5.2 AccountTest Class to Use Class Account (Cont.)

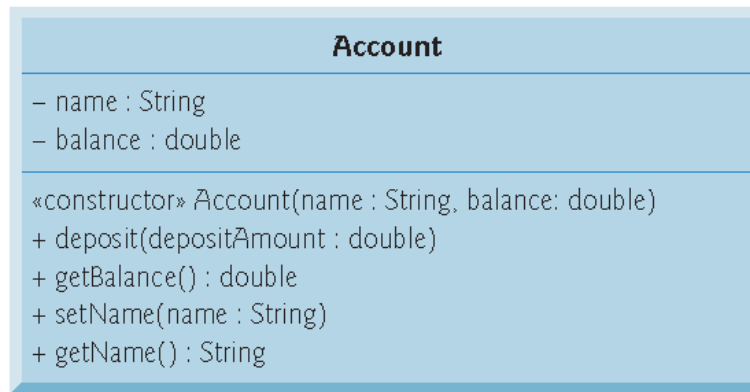


Fig. 3.10 | UML class diagram for **Account** class of Fig. 3.8.