CPE207 Object Oriented Programming

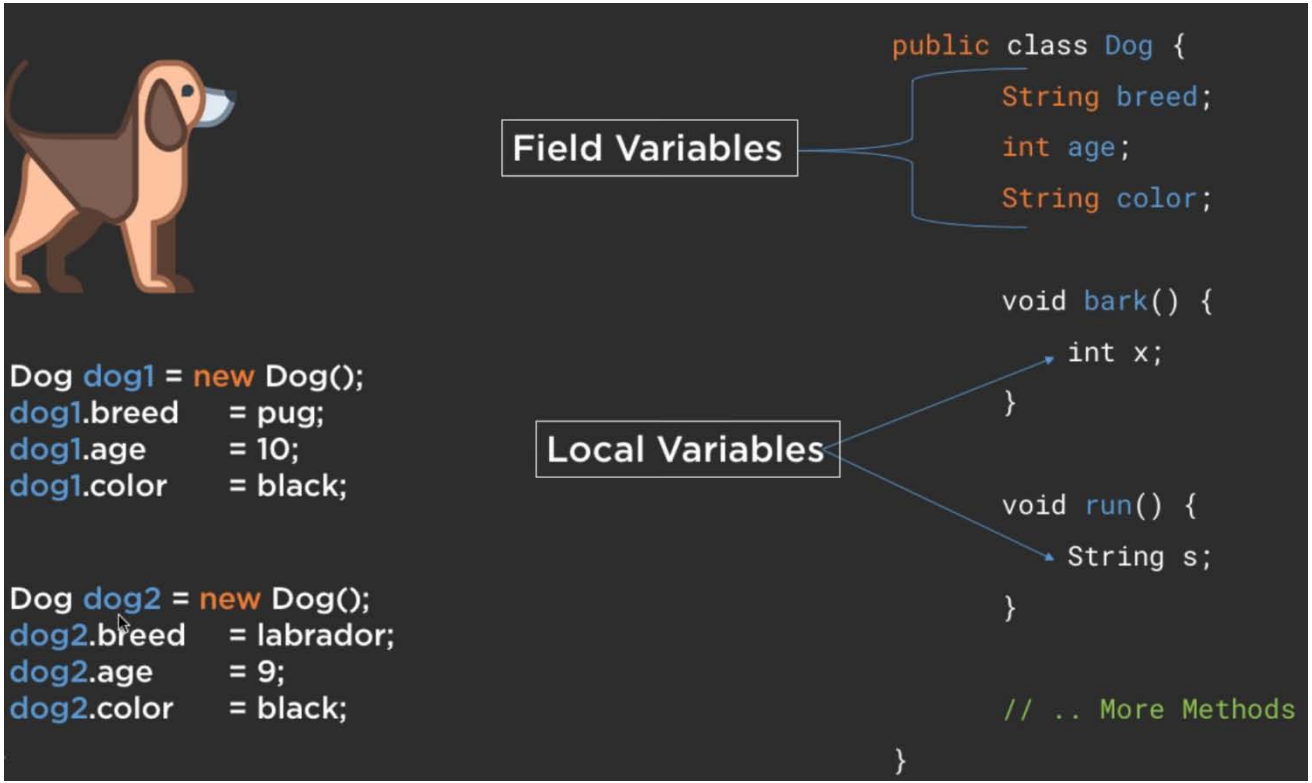
Week 3

Access Modifiers, Class Constructors, UML, Data Hiding(Encapsulation)

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

These Slides mainly adopted from Assist. Prof. Dr. Ozacar Kasim lecture notes

Class declaration



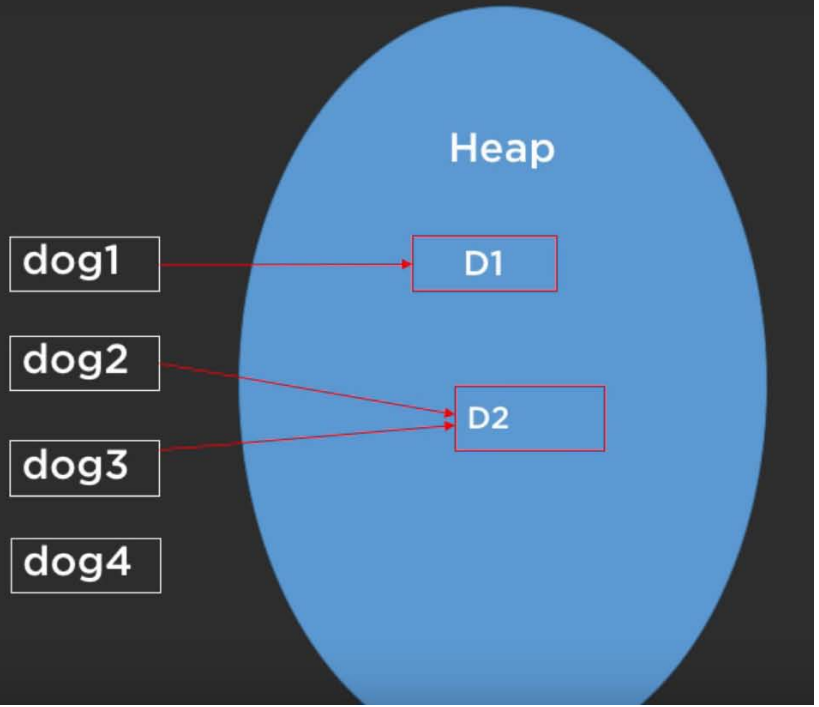
Reference variable, Object and Heap memory

Reference Variable, Objects and Heap Memory

```
Dog dog1 = new Dog();   // D1
Dog dog2 = new Dog();  // D2
Dog dog3 = dog2;
Dog dog4;


dog2.age = 10;
// Both dog2 and dog3 have same age 10
dog3.age = 12;
// Both dog2 and dog3 have same age 12
```

Reason: Both dog2 and dog3 are pointing towards same object D2



Primitive Types vs. Reference Types

- Types in Java are divided into two categories: primitive types and reference types.
- The primitive types are boolean, byte, char, short, int, long, float and double.
- All other types are reference types.
- A primitive-type:
  - variable can store exactly one value of its declared type at a time.
  - instance variables are initialized by default.
  - Variables of types byte, char, short, int, long, float and double are initialized to 0.
  - Variables of type boolean are initialized to false.
- Reference-type:
  - variables (called references) store the location of an object in the computer's memory.
  - The object that's referenced may contain many instance variables and methods.
  - Reference-type instance variables are initialized by default to the value null.
  - A reference to an object is required to call an object's methods.

Access Modifiers?

- Java provides some access modifiers to set access levels for classes, variables, methods, and constructors.
- For classes, you can use either public or default:

Modifier	Description
public	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier.

Access Modifiers?

- For attributes, methods and constructors, you can use the one of the following:

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier.
protected	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter.

Default Access Modifier – No Keyword

- A variable or method declared without any access control modifier is available to any other class in the **same package**.

```
String version = "1.5.1";
boolean processOrder() {
 return true;
}
```

Public Access Modifier – Public

- A class, method, constructor, interface, etc. declared public can be accessed from anywhere.
- Therefore, attributes, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

```
public static void main(String[] arguments)
{
    // ...
}
```

The **main()** method has to be public. Otherwise, it could not be called by a Java interpreter to run the class.

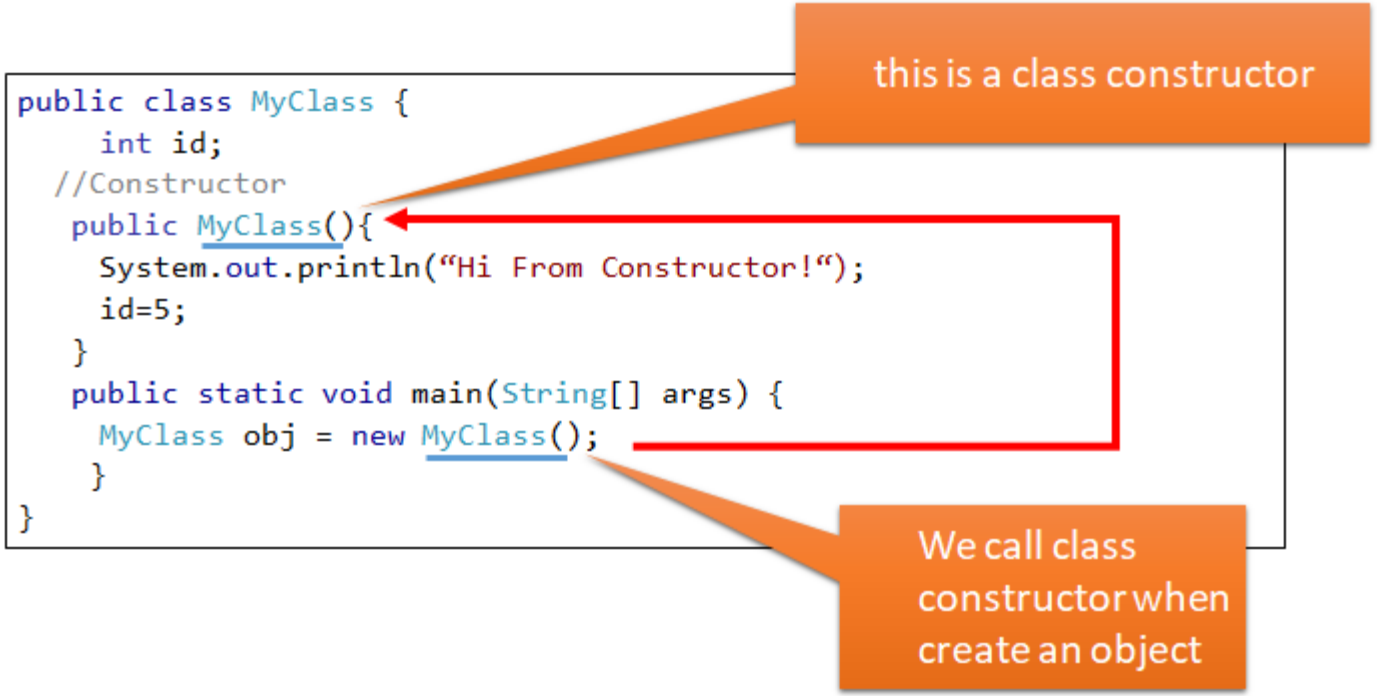
Private Access Modifier – Private

- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level. Class cannot be private.
- Using the private modifier is the main way that an object **encapsulates** itself and hides data from the outside world.

```
class Animal {
   private String name;
}
public class MainClass{
      public static void main(String[] arguments) {
          Animal animal = new Animal();
           animal.name;
      }
}
```

This is will cause a compilation error. Because name is not accessible due to its protection level.

Class Constructors



Class Constructors

- A constructor is a method which is used to initialize an object
- Constructor method of a class has the same name as that of the class, they are called when an object of a class is created.
- **When Attributes of an object are not available while creating objects, the default constructor is called.**
- It is optional to write constructor method(s) in a class but due to their utility they are used.

```
public class MyClass
{

  //constructor
public MyClass()
    {
        ...
    }
   ...
}
```

constructor doesn't have a return type !!!

Class Constructors

Default Constructor

- If a class does not define constructors, the compiler provides a default constructor with no parameters, and the class's instance variables are initialized to their default values.

There's No Default Constructor in a Class That Declares a Constructor

- If you declare a constructor for a class, the compiler will *not* create a *default constructor* for that class.

Constructor Overloading

Constructor with no parameters

Overloaded constructor

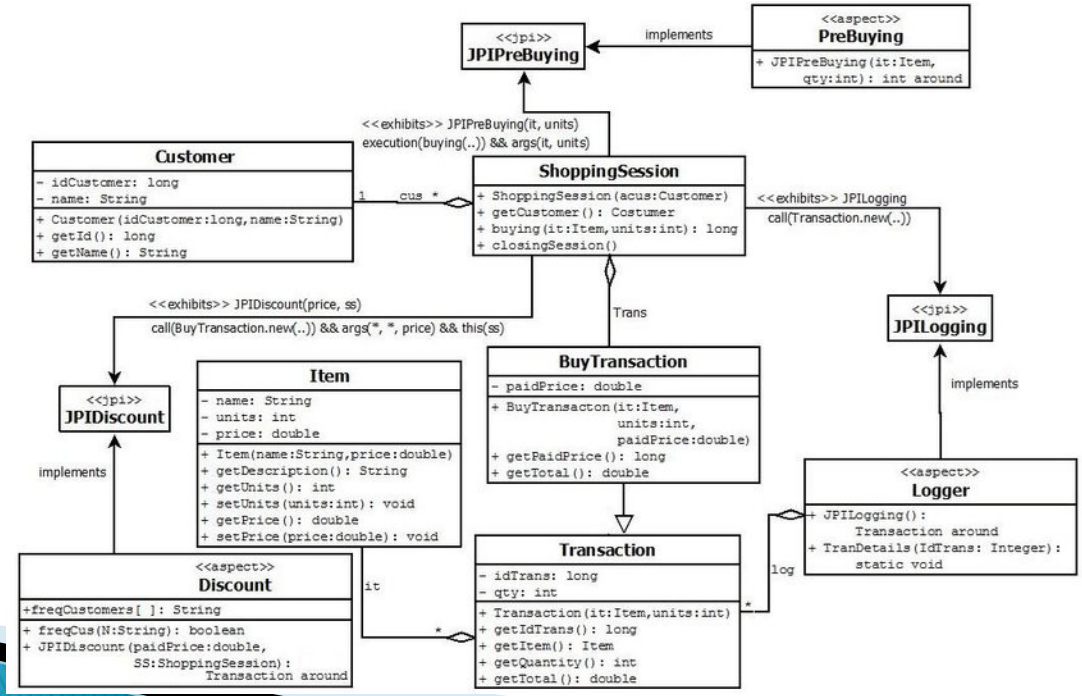
```
class Student{
      private int id;
      private String name;
      private int age;

    Student(){
         id = 100;
         name = "New Student";
         age = 18;
    }
    Student(int id, String name, int age){
         this.id = id;
         this.name = name;
         this.age = age;
    }
}
```

Now, Let's create a student object…

UML: Unified Modelling Language

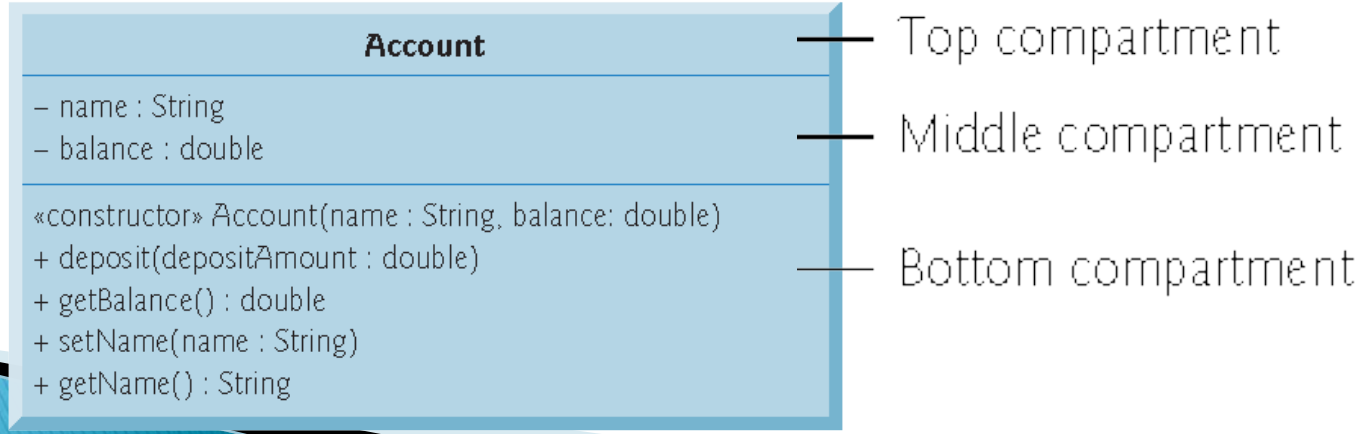
- The **Unified Modeling Language** (**UML**) is a general-purpose, developmental, **modeling language** in the software engineering field.
- It provides a standard way to visualize the design of a system.



UML Class Diagram

Adding the Constructor to Class Account's UML Class Diagram

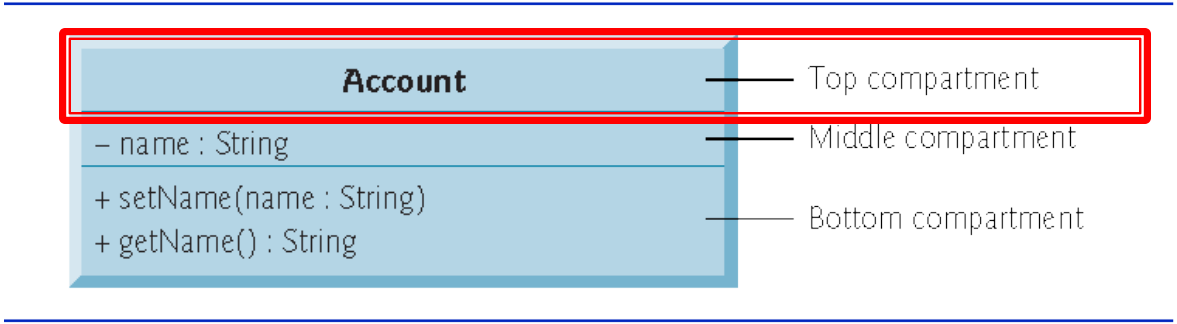
- The UML models constructors in the third compartment of a class diagram.
- To distinguish a constructor from a class's operations, the UML places the word "constructor" between guillemets (« and ») before the constructor's name.



Account UML Class Diagram with an Instance Variable and *set* and *get* Methods

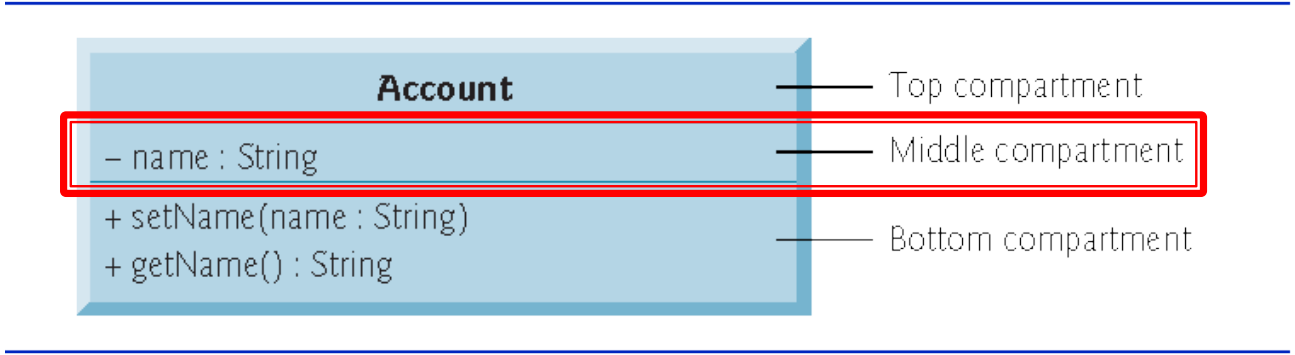
Top Compartment

- In the UML, each class is modeled in a class diagram as a rectangle with three compartments. The top one contains the class's name centered horizontally in boldface.



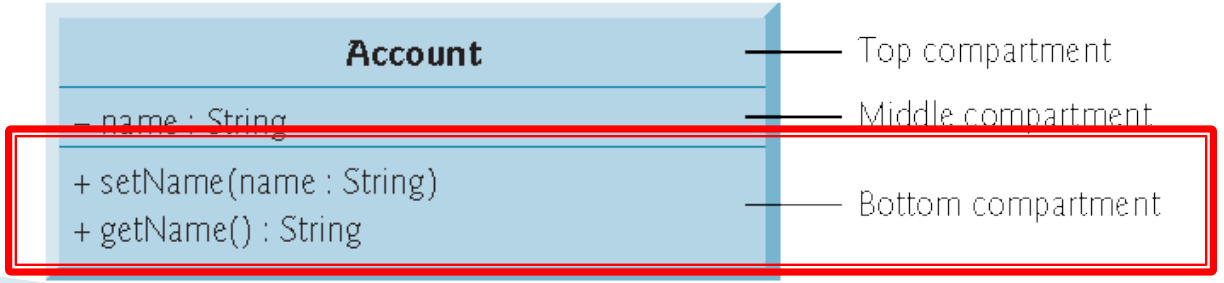
Middle Compartment

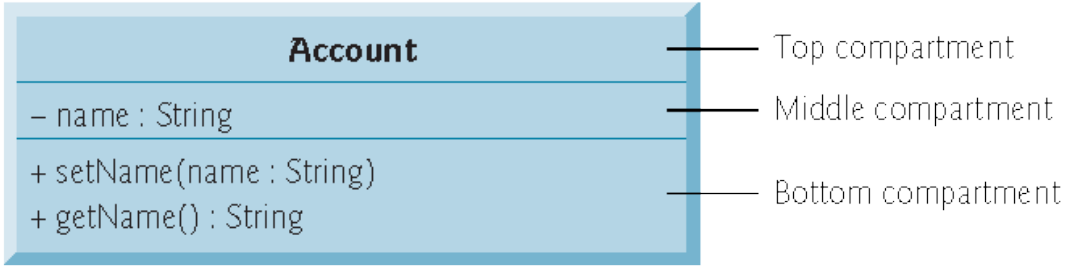
- It contains the class's attributes, which correspond to instance variables in Java. Here minus (–) means that the attribute is private



Bottom Compartment

- It contains the class's operations, which correspond to methods and constructors in Java.
- The UML represents instance variables as an attribute name, followed by a colon and the type.
- Private attributes are preceded by a minus sign (-) in the UML.
- The UML models operations by listing the operation name followed by a set of parentheses.
- A plus sign (+) in front of the operation name indicates that the operation is a public one in the UML (i.e., a public method in Java).





Return Types

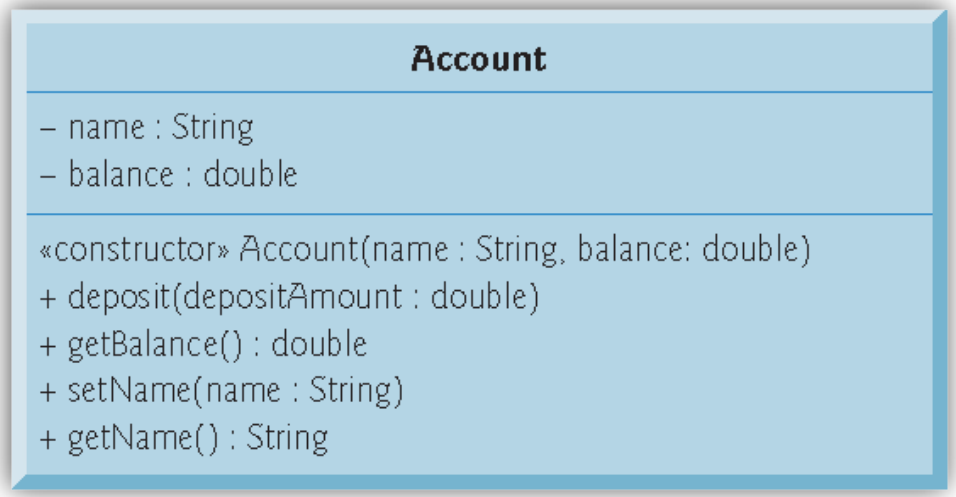
- The UML indicates an operation's return type by placing a colon and the return type after the parentheses following the operation name.
- UML class diagrams do not specify return types for operations that do not return values.
- Declaring instance variables private is known as data hiding or encapsulation.

Parameters

- The UML models a parameter of an operation by listing the parameter name, followed by a colon(:) and the parameter type between the parentheses after the operation name

Thanks ☺

An Example : Lets create an Account Class using UML



Account Class

```
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
```

```
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
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
```

AccountTest Class to Use Class Account

```
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
```

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
```

```
Enter deposit amount for account2: 123.45

adding 123.45 to account2 balance

Jane Green balance: $75.53
John Blue balance: $123.45
```