Java Programming

Classes and Objects in Java

Lesson 2 Objectives

- Declare classes and use them to create objects.
- Implement class’s attributes and behaviors.
- Differentiate between local and instance variables.
- Use constructors to initialize data.
- Call object's methods to perform their tasks.

Objects

- Object is a general term that stands for many things.
  - For example: A *student*, a *desk*, a *circle*, a *point*, a *mailbox*, can all be viewed as objects.
  - For any of the objects above you may distinguish some **properties** and **behaviors**.
  - A Mailbox object has a property called **password**.
  - A student object has a property called **lastname**.
- These properties are known as **data fields**.

Classes

- A **collection of related objects** is called **class**.
- Java uses a programming unit called **class** to define a group of related objects.
- A class in Java is a *collection of* ***instance variables****, which describe the* ***properties*** *of class objects, and* ***methods****, which describe the behaviors*.

Classes

- Each class you create becomes **a new type** that can be used to declare variables and create objects.
- You can declare new classes as needed; this is one reason Java is known as an extensible language.
- Classes promote **reusability**.

Account Class with an Instance Variable, a set Method and a get Method

```
//  Account.java
// Account class that contains a name instance variable
// and methods to set and get its value.
public class Account
{
   private String name; // instance variable

   // method to set the name in the object
   public void setName(String name)
   {
      this.name = name; // store the name
   }

   // method to retrieve the name from the object
   public String getName()
   {
      return name; // return value of name to caller
   }
} // end class Account
```

Class Declaration

- Each class declaration that begins with the access modifier public must be stored in a file that has the **same name as the class and ends with the .java filename extension**.
- Every class declaration contains keyword class followed immediately by the class's name.

Identifiers and Camel Case Naming

- Class, method and variable names are identifiers.
- By convention all use **camel case names**.
- Class names begin with an uppercase letter
- Method and variable names begin with a lowercase letter.

Instance Variable name

- An object has **attributes** that are implemented as **instance variables** and carried with it throughout its lifetime.
- Instance variables exist before methods are called on an object, while the methods are executing and after the methods complete execution.
- A class normally contains one or more methods that manipulate the instance variables that belong to particular objects of the class.
- Instance variables are declared inside a class declaration but outside the bodies of the class's method declarations.
- Each object (instance) of the class has its own copy of each of the class's instance variables.

Access Modifiers public and private

- Most **instance-variable** declarations are preceded with the keyword private, which is an access modifier.
- Variables or methods declared with access modifier private are **accessible only to methods of the class** in which they're declared.

setName Method of Class Account

- **Parameters** are declared in a **comma-separated parameter list**, which is located inside the parentheses that follow the method name in the method declaration.
- Multiple parameters are separated by commas.
- Each **parameter must specify a type** followed by a variable name.

Parameters Are Local Variables

- Variables declared in the body of a particular method are **local variables** and can be used only in that method.
- When a method terminates, the values of its local variables are lost.
- A method's parameters are local variables of the method.

setName Method Body

- Every method's body is delimited by left and right braces ({ and }).
- Each method's body contains one or more statements that perform the method's task(s).

getName Method of Class Account

- The method's **return type** specifies the type of data returned to a method's caller.
- Keyword **void** indicates that a method will perform a task but will not return any information.
- **Empty parentheses** following a method name indicate that the method does not require any parameters to perform its task.
- When a method that specifies a return type other than void is called and completes its task, the method must return a result to its calling method.

getName Method of Class Account

- The return statement passes a value from a called method back to its caller.
- Classes often provide **public methods** to allow the class's clients to set or get private instance variables.
- The names of these methods need not begin with set or get, but this **naming convention is recommended**.

Driver Class AccountTest

- A class that creates an object of another class, then calls the object's methods, is a **driver class**.

Driver Class AccountTest

```
import java.util.Scanner;

public class AccountTest
{
   public static void main(String[] args)
   {
      // create a Scanner object to obtain input from the command window
      Scanner input = new Scanner(System.in);

      // create an Account object and assign it to myAccount
      Account myAccount = new Account();

      // display initial value of name (null)
      System.out.printf("Initial name is: %s%n%n", myAccount.getName());
```

Driver Class AccountTest (Cont)

```
      // prompt for and read name
      System.out.println("Please enter the name:");
      String theName = input.nextLine(); // read a line of text
      myAccount.setName(theName); // put theName in myAccount
      System.out.println(); // outputs a blank line

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

Instantiating an Object—Keyword new and Constructors

- A class instance creation expression begins with keyword new and creates a new object.
- A **constructor** is similar to a method but is **called implicitly by the new operator** to initialize an object's instance variables at the time the object is created.

Calling Class Account's getName Method

- To call a method of an object, follow the object name with a dot separator, the method name and a set of parentheses containing the method's arguments:

```
myAccount.setName(theName); // put theName in myAccount
```

null—the Default Initial Value for String Variables

- Local variables are not automatically initialized.
- Every **instance variable has a default initial value -** a value provided by Java when you do not specify the instance variable's initial value.
- The default value for an instance variable of type String is null.

Calling Class Account's setName Method

- A method call supplies values—known as **arguments - f**or each of the method's parameters.
- Each argument's value is assigned to the corresponding parameter in the method header.
- The number of arguments in a method call must match the number of parameters in the method declaration's parameter list.
- The argument types in the method call must be consistent with the types of the corresponding parameters in the method's declaration.

Compiling and Executing an App with Multiple Classes

- The **javac** command can compile multiple classes at once.
- Simply list the source-code filenames after the command with each filename separated by a space from the next.
- If the directory containing the app includes only one app's files, you can compile all of its classes with the command **javac *.java**.
- The asterisk (*) in *.java indicates that all files in the current directory ending with the filename extension ".java" should be compiled.

private Instance Variables and public set and get Methods

- Declaring **instance variables private** is known as **data hiding** or **information hiding**.

Account Class: Initializing Objects with Constructors

- Each class you declare can **optionally provide a constructor with parameters** that can be used to initialize an object of a class when the object is created.
- Java requires a constructor call for every object that's created.

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

- Constructors Cannot Return Values
- Constructors can specify parameters but not return types.
- 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.

Code example (Fig. 3.6: AccountTest.java)

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

- A floating-point number is a number with a decimal point.
- Java provides two primitive types for storing floating-point numbers in memory - float and double.
- Variables of type float represent single-precision floating-point numbers and have **seven significant digits**.
- Variables of type double represent double-precision floating-point numbers.
- These require twice as much memory as float variables and provide **15 significant digits** - approximately double the precision of float variables.
- Floating-point literals are of type **double** **by default**.

AccountTest Class to Use Class Account

- The default value for an instance variable of type double is 0.0, and the default value for an instance variable of type int is 0.

Formatting Floating-Point Numbers for Display

- The format specifier %f is used to output values of type float or double.
- The format specifier %.2f specifies that two digits of precision should be output to the right of the decimal point in the floating-point number.

Code example (Fig. 3.8)

Using Dialog Boxes for IO

```
// Fig. 3.13: NameDialog.java, Obtaining user input from a dialog.
import javax.swing.JOptionPane;

public class NameDialog
{
   public static void main(String[] args)
   {
      // prompt user to enter name
      String name = JOptionPane.showInputDialog("What is your name?");
      // create the message
      String message =
         String.format("Welcome, %s, to Java Programming!", name);
      // display the message to welcome the user by name
      JOptionPane.showMessageDialog(null, message);
   }
} // end class NameDialog
```

Code Example

- Modify the AccountTest example by replacing Scanner class with JOptionPane class

References

- Textbook
- Java documentation