Unit 8 | Objects and Classes

8.2 Defining Classes for Objects

An object represents an entity in the real world that can be distinctly identified.

An object has a unique identity, state, and behavior.

```
- The state of an object (also known as its properties or attributes) is
represented by data fields with their current values.
- The behavior of an object (also known as its actions) is defined by methods. To
invoke a method on an object is to ask the object to perform an action.
```

Objects of the same type are defined using a common class. A class is a template, blueprint, or contract that defines what an object's data fields and methods will be. An object is an instance of a class. You can create many instances of a class. Creating an instance is referred to as instantiation. The terms object and instance are often interchangeable.

Java class uses variables to define data fields and methods to define actions. Additionally, a class provides methods of a special type, known as constructors, which are invoked to create a new object. A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.

8.3 Defining Classes and Creating Objects

You can put the two classes into one file, but only one class in the file can be a public class. Furthermore, the public class must have the same name as the file name.

8.4 Constructing Objects Using Constructors

Constructors are a special kind of method. They have three peculiarities:

- A constructor must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.

It is a common mistake to put the void keyword in front of a constructor. For example,

```
public Circle() {
}
```

In this case, Circle() is a method, not a constructor.

A class normally provides a constructor without arguments (e.g., Circle()). Such a constructor is referred to as a no-arg or no-argument constructor. A class may be defined without constructors. In this case, a no-arg constructor with an empty body is implicitly defined in the class. This constructor, called a default constructor, is provided automatically only if no constructors are explicitly defined in the class.

8.5 Accessing Objects via Reference Variables

Newly created objects are allocated in the memory. They can be accessed via reference variables.

8.5.1 Reference Variables and Reference Types

Objects are accessed via object reference variables, which contain references to the objects. Such variables are declared using the following syntax:

```
ClassName objectRefVar;
```

A class is essentially a programmer-defined type. A class is a reference type, which means that a variable of the class type can reference an instance of the class.

> **Note**
>
> An object reference variable that appears to hold an object actually contains a reference to that object. Strictly speaking, an object reference variable and an object are different, but most of the time the distinction can be ignored. So it is fine, for simplicity, to say that myCircle is a Circle object rather than use the longer-winded description that myCircle is a variable that contains a reference to a Circle object.

> **Note**
>
> Arrays are treated as objects in Java. Arrays are created using the new operator. An array variables actually a variable that contains a reference to an array.

8.5.2 Accessing an Object's Data and Methods

After an object is created, its data can be accessed and its methods invoked using the dot operator (.), also known as the object member access operator:

- objectRefVar.dataField references a data field in the object.
- objectRefVar.method(arguments) invokes a method on the object.

The data field radius is referred to as an instance variable, because it is dependent on a specific instance. For the same reason, the method getArea is referred to as an instance method, because you can invoke it only on a specific instance. The object on which an instance method is invoked is called a calling object.

> **Caution**
>
> Recall that you use Math.methodName(arguments) (e.g., Math.pow(3, 2.5)) to invoke a method in the Math class. Can you invoke getArea() using Circle.getArea()? The answer is no. All the methods in the Math class are static methods, which are defined using the static keyword. However, getArea() is an instance method, and thus nonstatic. It must be invoked from an object using objectRefVar.methodName(arguments) (e.g., myCircle.getArea()).

> **Note**
>
> Usually you create an object and assign it to a variable. Later you can use the variable to reference the object. Occasionally an object does not need to be referenced later. In this case, you can create an object without explicitly assigning it to a variable, as shown below:

```
new Circle();`

// or

System.out.println("Area is " + new Circle(5).getArea());
```

> The former statement creates a Circle object. The latter creates a Circle object and invokes its getArea method to return its area. An object created in this way is known as an anonymous object.

8.5.3 Reference Data Fields and the null Value

If a data field of a reference type does not reference any object, the data field holds a special Java value, null. null is a literal just like true and false. While true and false are Boolean literals, null is a literal for a reference type.

The default value of a data field is null for a reference type, 0 for a numeric type, false for a boolean type, and '\u0000' for a char type. However, Java assigns no default value to a local variable inside a method.

> **Caution**
>
> NullPointerException is a common runtime error. It occurs when you invoke a method on a reference variable with null value. Make sure you assign an object reference to the variable before invoking the method through the reference variable.

8.5.4 Differences Between Variables of Primitive Types and Reference Types

Every variable represents a memory location that holds a value. When you declare a variable, you are telling the compiler what type of value the variable can hold. For a variable of a primitive type, the value is of the primitive type. For a variable of a reference type, the value is a reference to where an object is located.

When you assign one variable to another, the other variable is set to the same value. For a variable of a primitive type, the real value of one variable is assigned to the other variable. For a variable of a reference type, the reference of one variable is assigned to the other variable.

> **Note**
>
> After the assignment statement c1 = c2 *(c1 and c2 reference type of values)*, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer useful and therefore is now known as garbage. Garbage occupies memory space. The Java runtime system detects garbage and automatically reclaims the space it occupies. This process is called garbage collection.

> **Tip** If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The JVM will automatically collect the space if the object is not referenced by any

> reference variable.

8.6 Using Classes from the Java Library

8.6.2 The Random Class

> **Note**
>
> The ability to generate the same sequence of random values is useful in software testing and many other applications. In software testing, you can test your program using a fixed sequence of numbers before using different sequences of random numbers.

8.7 Static Variables, Constants and Methods

An instance variable is tied to a specific instance of the class; it is not shared among objects of the same class.

If you want all the instances of a class to share data, use static variables, also known as class variables. Static variables store values for the variables in a common memory location. Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected. Java supports static methods as well as static variables. Static methods can be called without creating an instance of the class.

> **Tip**
>
> Use ClassName.methodName(arguments) to invoke a static method and ClassName.staticVariable to access a static variable. This improves readability, because the user can easily recognize the static method and data in the class.

> **Design Guide**
>
> How do you decide whether a variable or method should be an instance one or a static one? A variable or method that is dependent on a specific instance of the class should be an instance variable or method. A variable or method that is not dependent on a specific instance of the class should be a static variable or method. For example, every circle has its own radius. Radius is dependent on a specific circle. Therefore, radius is an instance variable of the Circle class. Since the getArea method is dependent on a specific circle, it is an instance method. None of the methods in the Math class, such as random, pow, sin, and cos, is dependent on a specific instance. Therefore, these methods are static methods. The main method is static and can be invoked directly from a class.

> **Caution**
>
> It is a common design error to define an instance method that should have been defined static. For example, the method factorial(int n) should be defined static, as shown below, because it is independent of any specific instance.

8.8 Visibility Modifiers

You can use the public visibility modifier for classes, methods, and data fields to denote that they can be accessed from any other classes. If no visibility modifier is used, then by default the classes, methods, and data fields are accessible by any class in the same package. This is known as package-private or package-access.

> **Note**

> Packages can be used to organize classes. To do so, you need to add the following line as the first noncomment and nonblank statement in the program:
>
> ```
> package packageName;
> ```
>
> If a class is defined without the package statement, it is said to be placed in the default package. Java recommends that you place classes into packages rather using a default package.

In addition to the public and default visibility modifiers, Java provides the private and protected modifiers for class members. This section introduces the private modifier. The protected modifier will be introduced in §11.13, "The protected Data and Methods." The private modifier makes methods and data fields accessible only from within its own class.

> **Caution**
>
> The private modifier applies only to the members of a class. The public modifier can apply to a class or members of a class. Using modifiers public and private on local variables would cause a compile error.

> **Note** In most cases, the constructor should be public. However, if you want to prohibit the user from creating an instance of a class, use a private constructor. For example, there is no reason to create an instance from the Math class, because all of its data fields and methods are static. To prevent the user from creating objects from the Math class, the constructor in java.lang.Math is defined as follows:
>
> ```
> private Math() {
> }
> ```

8.9 Data Field Encapsulation

To prevent direct modifications of data fields, you should declare the data fields private, using the private modifier. This is known as data field encapsulation. A private data field cannot be accessed by an object from outside the class that defines the private field. But often a client needs to retrieve and modify a data field. To make a private data field accessible, provide a get method to return its value. To enable a private data field to be updated, provide a set method to set a new value.

> **Note**
>
> Colloquially, a get method is referred to as a getter (or accessor), and a set method is referred to as a setter (or mutator).

> **Design Guide** To prevent data from being tampered with and to make the class easy to maintain, declare data fields private.

8.10 Passing Objects to Methods

You can pass objects to methods. Like passing an array, passing an object is actually passing the reference of the object.

8.11 Array of Objects

An array of objects is actually an array of reference variables.

> **Note**
>
> When an array of objects is created using the new operator, each element in the array is a reference variable with a default value of null.