Chapter 9 Objects and Classes

Classes

Classes are constructs that define objects of the same type. A Java class uses variables to define data fields and methods to define behaviors. Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

```java
class Circle {
    /** The radius of this circle */
    double radius = 1.0;

    /** Construct a circle object */
    Circle() {
    }

    /** Construct a circle object */
    Circle(double newRadius) {
        radius = newRadius;
    }

    /** Return the area of this circle */
    double getArea() {
        return radius * radius * 3.14159;
    }
}
```
Example: Defining Classes and Creating Objects

Objective: Demonstrate creating objects, accessing data, and using methods.

Constructors

```java
Circle() {
}
Circle(double newRadius) {
    radius = newRadius;
}
```

Constructors are a special kind of methods that are invoked to construct objects.
Constructors, cont.

A constructor with no parameters is referred to as a no-arg constructor.

- Constructors 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.

Default 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.

Declaring Object Reference Variables

To reference an object, assign the object to a reference variable.

To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
```

Example:
```
Circle myCircle;
```
Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Create a circle

myCircle
no value

Circle
radius: 5.0

Assign object reference to myCircle

myCircle
reference value

Circle
radius: 5.0
Trace Code, cont.

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Declare yourCircle

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Create a new Circle object

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

Assign object reference to yourCircle

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();
yourCircle.radius = 100;

myCircle reference value

yourCircle reference value

radius: 5.0

radius: 1.0
Trace Code, cont.

Circle myCircle = new Circle(5.0);
Circle yourCircle = new Circle();

Reference Data Fields

The data fields can be of reference types. For example, the following Student class contains a data field name of the String type.

```java
public class Student {
    String name; // name has default value null
    int age; // age has default value 0
    boolean isScienceMajor; // isScienceMajor has default value false
    char gender; // c has default value 'u0000'
}
```

The null Value

If a data field of a reference type does not reference any object, the data field holds a special literal value, null.
Garbage Collection

As shown in the previous figure, after the assignment statement `c1 = c2`, `c1` points to the same object referenced by `c2`. The object previously referenced by `c1` is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

Instance Variables, and Methods

Instance variables belong to a specific instance.

Instance methods are invoked by an instance of the class.

Instance variables and methods are specified by omitting the `static` keyword.

Static Variables, Constants, and Methods

Static variables are shared by all the instances of the class.

Static methods are not tied to a specific object.

Static constants are final variables shared by all the instances of the class.
Static Variables, Constants, and Methods, cont.

To declare static variables, constants, and methods, use the `static` modifier.

Example of Using Instance and Class Variables and Method

Objective: Demonstrate the roles of instance and class variables and their uses. This example adds a class variable `numberOfObjects` to track the number of Circle objects created.

Visibility Modifiers and Accessor/Mutator Methods

By default, the class, variable, or method can be accessed by any class in the same package.

- **public**
  The class, data, or method is visible to any class in any package.

- **private**
  The data or methods can be accessed only by the declaring class.
  The get and set methods are used to read and modify private properties.
Passing Objects to Methods

- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

The this Keyword

- The `this` keyword is the name of a reference that refers to an object itself. One common use of the `this` keyword is reference a class’s *hidden data fields*.
- Another common use of the `this` keyword to enable a constructor to invoke another constructor of the same class.

Reference instance variables

```java
public class F {
  private int i = 5;
  private static double k = 0;

  void setI(int i) {
    this.i = i;
  }

  static void setK(double k) {
    F.k = k;
  }
}

Suppose that f1 and f2 are two objects of F.

f1 = new F(); f2 = new F();

Invoking f1.setI(10) to execute
  this.i = 10, where this refers f1

Invoking f2.setI(45) to execute
  this.i = 45, where this refers f2
```
Calling Overloaded Constructor

```java
class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public Circle() {
        this(1.0);
    }

    public double getArea() {
        return this.radius * this.radius * Math.PI;
    }
}
```

Every instance variable belongs to an instance represented by this, which is normally omitted.