Chapter 02 Classes and Objects

Having learned the material in earlier chapters, you are able to solve many programming problems using selections, loops, arrays, maps, etc. However, these features are not sufficient for developing large-scale software systems. This chapter presents the introduction of object-oriented programming (OOP), which will enable you to develop large-scale software systems effectively.

1. Introduction

Object-oriented Programing (OOP) is the most efficient and most commonly used programming methodology. The demand for software developers is increasing. Understanding the basics of OOP is a must for developers.

OOP is neither a tool nor a programming language—it is just a concept. Some programming languages are designed to follow this concept. Java is one of them. There are other object-oriented languages, such as Python, C#, and so on.

In OOP, we try to think about our software components as small objects, and create relationships between them to solve a problem.

2. Defining Classes and Creating Objects

Object-oriented programming (OOP) involves programming using objects. An **object** represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, and even a loan can all be viewed as objects.

Objects of the same type are defined using a common class. A **class** is like a template or blueprint that defines what an object's data fields and methods will be. In most circumstances, a class itself cannot actually do anything; it is just used to create objects.

For example, let's say we have a `Person` class. Here, when we say Person, we do not mean any particular person, but we are referring to a person being in general.

Let's take a closer look at the properties and behaviors of a `Person` class. There are hundreds of properties that you can list for a person, but for the sake of simplicity, we can say that the following are the properties of a human being:

- Height
- Weight
- Age

We can do the same for behavioral properties. There are hundreds of particular behaviors that a person can perform, but here we will only consider the following:

- Walk
- Talk
- Eat

Let's say we have John who is a person. So, John is an object of the `Person` class.

2.1 Creating a Class

A **class** is a user defined data-type which has data members and function members.

- **Data members** are variables inside a class. They are known as **fields**, **data fields**, or **attributes**. They define the properties of the objects of a class.
- **Function members** are the functions used to perform certain actions. They are known as **methods**. A method will be executed when it is called. They define the behaviors of the objects of a class.

In Java, a class is defined using keyword `class`. To create a class in Java, you have to follow a particular syntax:

```
class ClassName {
    // fields
    // methods
};
```

For example, create a `Cylinder` class:

```
class Cylinder {
    // state or field
    double radius;
    double height;

    // behavior or method
    double calculateArea(){
        return radius * radius * 3.14;
    }
    double calculateVolume(){
        return radius * radius * 3.14 * height;
    }
}
```

Here, we defined a class named `Cylinder`. The variables `radius` and `height` declared inside the class are known as fields. And, the functions `calculateArea()` and `calculateVolume` are known as methods of the class.

2.2. Creating Objects

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.

When a class is defined, no memory is allocated. But when it is instantiated (i.e., an object is created) memory is allocated. We can also create objects of a class within the class itself, or in other classes.

In Java, to create objects, you have to follow a particular syntax:

```
ClassName object = new ClassName();
```

For example, we can create objects of `Cylinder` class (defined in the previous example) as follows:

```
Cylinder cylinder1 = new Cylinder();
```

Here, an object named `cylinder1` of the `Cylinder` class is created.

We have used the `new` keyword along with the constructor of the class to create an object. Constructors are similar to methods and have the same name as the class. For example, `Cylinder`() is the constructor of the `Cylinder` class.

2.3 Accessing Members

An object's members can be accessed through the dot (.) operator via the object's name. Syntax:

```
object.member;
```

For example, let's assign `2.5` to the `radius` variable of `cylinder1`:

```
cylinder1.radius = 2.5;
```

For example, let's call the `calculateArea` method inside the `Cylinder` class for object `cylinder1`:

```
cylinder1.calculateArea();
```

Example 01: A program to illustrate the working of objects and classes.

```
//Program.java
class Cylinder {
    // states, fields or attributes
    double radius;
    double height;

    // behaviors or methods
    double calculateArea(){
        return radius * radius * 3.14;
    }
    double calculateVolume(){
        return radius * radius * 3.14 * height;
    }
}
```

```
class Program {
    public static void main(String[] args) {
        // Create object
        Cylinder cylinder1 = new Cylinder();

        // Access fields and methods
        cylinder1.radius = 2.5;
        cylinder1.height = 3;

        double area = cylinder1.calculateArea();
        double volume = cylinder1.calculateVolume();

        // Display the result
        System.out.println("area = " + area);
        System.out.println("volume = " + volume);
    }
}
```

Output:

```
area = 19.625
volume = 58.875
```

3. Constructors

A constructor is a special type of method that:

- Has the same name as that of the class.
- Does not have a return type; not even `void`.
- Is called automatically when an object is created.

A constructor is primarily used to initialize objects. They are also used to run a default code when an object is created.

Let's look at an example of creating a constructor:

```
class Wall {
    // Create a constructor
    Wall(){
        // code
    }
}
```

Here, `Wall()` is a constructor of the class `Wall`. It has the same name as that of the class and does not have a return type.

We cannot define a static constructor in Java, If we are trying to define a constructor with the static keyword a compile-time error will occur. In general, static means class level. A constructor is called when an object of a class is created, so no use of the static constructor.

In Java, constructors can be divided into 3 types:

- No-Arg Constructor
- Parameterized Constructor
- Default Constructor

3.1 No-Arg Constructors

Similar to methods, a Java constructor may or may not have any parameters. If a constructor does not accept any parameters, it is known as a no-argument constructor.

Example 02: Create a No-argument constructor.

```
//Program.java
class Wall {
    double length;

    // Constructor with no parameter
    Wall() {
        length = 5.5; // initialize variable

        System.out.println("Creating a wall.");
        System.out.println("Length = " + length);
    }
}
class Program {
    public static void main(String[] args) {
        Wall wall1 = new Wall();
    }
}
```

Output:

```
Creating a wall.
Length = 5.5
```

In the above example, when the `wall1` object is created, the `Wall()` constructor is called. This sets the `length` variable of the object to `5.5`.

3.2 Parameterized Constructors

In Java a constructor with parameters is known as a **parameterized constructor**. This is the preferred method to initialize data members.

Example 03: A program that demonstrates the use of parameterized constructor.

```
//Program.java
class Wall {
    double length;
    double height;

    // Parameterized constructor to initialize variables
    Wall(double l, double h) {
        length = l;
        height = h;
    }
    double calculateArea() {
        return length * height;
    }
}
class Program {
    public static void main(String[] args) {
        // Create objects and initialize their data members
        Wall wall1 = new Wall(10.5, 8.6);
        Wall wall2 = new Wall(8.5, 6.3);

        System.out.println("Area of Wall 1: " + wall1.calculateArea());
        System.out.println("Area of Wall 2: " + wall2.calculateArea());
    }
}
```

Output:

```
Area of Wall 1: 90.3
Area of Wall 2: 53.55
```

Here, we have created a parameterized constructor `Wall()` that has two parameters: `double l` and `double h`. The values contained in these parameters are used to initialize the member variables `length` and `height`.

When we create an object of the `Wall` class, we pass the values for the member variables to the parameters.

3.3 Default Constructors

A class may be defined without constructors. If we have not defined a constructor in a class, the Java compiler will automatically create a no-arg constructor, which is called **default constructor**, with an empty code and no parameters. However, if you create at least one constructor in a class, no default constructor will be created.

4. The `this` Keyword

The `this` keyword refers to the current object. The most common use of the `this` keyword is to eliminate the confusion between class attributes and parameters with the same name

Example 04: Using the `this` keyword.

```
//Program.java
class Wall {
    double length;
    double height;

    Wall(double length, double height) {
        this.length = length;
        this.height = height;
    }
    double calculateArea() {
        return length * height;
    }
}
class Program {
    public static void main(String[] args) {
        Wall wall1 = new Wall(10.5, 8.6);

        System.out.println("Area = " + wall1.calculateArea());
    }
}
```

Output:

```
Area = 90.3
```

4. Constructors Overloading

Like regular methods, constructors can be overloaded (i.e., multiple constructors with the same name but different signatures), making it easy to construct objects with different sets of data values.

Example 05: Create multiple constructors.

```
//Program.java
class Wall {
    double length;
    double height;

    // No-arg constructor
    Wall() {
        length = 1;
        height = 2;
    }
    //Parameterized constructor with one parameter
    Wall(double length) {
        length = 3;
        this.length = length;
    }
    // Parameterized constructor with two parameters
    Wall(double length, double height) {
        this.length = length;
        this.height = height;
    }
    double calculateArea() {
        return length * height;
    }
}
class Program {
    public static void main(String[] args) {
        Wall wall1 = new Wall(10.5, 8.6);

        System.out.println("Area = " + wall1.calculateArea());
    }
}
```

Output:

```
Area = 90.3
```

5. Anonymous Objects

Usually, you create a named object and later access its members through its name. Occasionally, you may create an object and use it only once. In this case, you do not have to name it. Such objects are called **anonymous objects**.

The syntax to create an anonymous object using the no-arg constructor is

```
new ClassName()
```

The syntax to create an anonymous object using the constructor with arguments is

```
new ClassName(arguments)
```

Example 06: Using anonymous objects.

```
//Program.java
class Wall {
    double length;
    double height;

    Wall(double length, double height) {
        this.length = length;
        this.height = height;
    }
    double calculateArea() {
        return length * height;
    }
}
class Program {
    public static void main(String[] args) {

        System.out.println("Area = " + new Wall(10.5, 8.6).calculateArea());
    }
}
```

Output:

```
Area = 90.3
```

5. Array of Objects

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

Syntax to declare an array of objects:

```
ClassName[ ] objectArrayReference; // preferred
```

Alternatively, we can also declare an Array of Objects as :

```
ClassName objectArrayReference[ ];
```

For example, if you have a class Student then we can create an array of Student objects as given below:

```
Student[ ] studentObjects;   // preferred
```

Or

```
Student studentObjects[];
```

Syntax to declare and create an array of objects:

```
ClassName[ ] obj = new ClassName[length];
```

For example, if you have a class `Circle`, and we want to declare and create an array of 10 `Circle` objects, then it will be written as:

```
Circle[] circleArray = new Circle[10];
```

And once an array of objects is created like this, then the individual elements of the array of objects needs to be created using the `new` keyword.

To initialize `circleArray`, you can use a for loop as follows:

```
for (int i = 0; i < circleArray.length; i++) {
      circleArray[i] = new Circle();
}
```

Example 07: A example that demonstrates how to use an array of objects.

```
//Program.java
class Circle {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }
    double getArea() {
        return Math.pow(radius, 2) * 3.14;
    }
}
class Program {
    public static void main(String[] args) {
        Circle[] circleArray = new Circle[3];

        // Create object elements
        for (int i = 0; i < circleArray.length; i++) {
            circleArray[i] = new Circle(Math.random());
        }

        // Display the area of each circle
        for (int i = 0; i < circleArray.length; i++) {
            double area = circleArray[i].getArea();

            System.out.printf("Area of Circle %d: %.2f\n", i, area);
        }
    }
}
```

Output:

```
Area of Circle 0: 0.48
Area of Circle 1: 0.28
Area of Circle 2: 2.22
```

Exercises

1. Design a class named `Rectangle` that contains:
   - Two `float` fields named `width` and `height`.
   - Constructors that create a rectangle with the specified width and height. The default values are `1` and `2` for the width and height, respectively.
   - A method named `getArea`() that returns the area of this rectangle.
   - A method named `getPerimeter`() that returns the perimeter.

   Write a test program that creates two Rectangle objects—one with width `4` and height `40` and the other with width `3.5` and height `35.7`. Display the width, height, area, and perimeter of each rectangle.

2. Define a class called `Calculator` that contains:
   - Three fields named `num1`, `operator` and `num2`.
   - A constructor that creates a calculator with the specified `num1`, `operator` and `num2`.
   - Methods: `add`(), `subtract`(), `multiply`(), `divide`(), `power`() and `modulo`().

   Write a test program that asks the user to enter the value for the two number and the operator, then display the result. Here is a sample run:

```
Enter number1, operator and number2 separated by a space: 5.5 + 12
5.5 + 12 = 17.5
```

3. Write a simple ATM program. First, define a class named `Account` that contains `account_no`, `name`, and `balance` and `password`. Then, create and initialize five account objects and store them in an array or map. When the program starts, ask the user to enter the `account_no` and `password` to login. If login is not succeeded, ask the user to try again. If login succeeded, display the following menus:
   a. Balance
   b. Withdraw
   c. Deposit
   d. Transfer
   e. Exit the program

4. Define two classes called `Score` and `Student`.
   - The `Score` class contains:
     - Fields: `math_score`, `phy_score` and `chem_score`
     - A constructor that creates a score object with the specified score for each subject.
     - A method named `getTotal`() that returns the total score of the three subjects.

   - The `Student` class contains:
     - Fields: `id`, `name`, and `score`.
     - A constructor that creates a student with the specified `id`, `name` and `score`.
     - A method named `readStudent`() that will ask the user to enter `id`, `name` and `score` of a student.

   For the test program, create and initialize three student objects and store them in a list or dictionary. Then, display a menu that will allow the user to select any of the following features:
   a. Add a new student
   b. Delete student by id
   c. Search student by id
   d. Display all students
   e. Exit the program

   Note:
   - For searching, display the student if found, otherwise, displays `"Search Not Found"`.
   - When display student(s), the student data should be arranged in a tabular format, for example:

ID	Name	Math	Physics	Chemistry	Total
1	Lucy	50	30	70	150
2	John	60	40	80	180
3	Alex	70	50	90	210

Reference

[1] Y. Daniel Liang. ‘Introduction to Java Programming’, 11e – 2019

[2] https://www.programiz.com/java-programming