Object Oriented Programming – Java 1

Lecture 4

Classes and Objects

Dr. Obuhuma James

Description

After introducing Java programming in the last two topics, this topic introduces classes and object as the very first main concept of object-oriented programming. The focus is on how to create user-defined classes, use the same classes to define objects, and finally use the objects to perform specific tasks like to access and execute methods. Two types of methods, namely, accessor and mutator methods introduced in the previous topic will be expounded further with examples. The concept of classes and objects will be helpful for the rest of the course.

Learning Outcomes

By the end of this topic, you will be able to:

- Understand the relationship between classes and objects.
- Create classes and their objects in relation to the Unified Modeling Language (UML).
- Create mutator and accessor methods and execute them using objects of their classes.

Overview of Classes and Objects

A class is a group of objects with related properties. Objects in this case represents real-world things. This means that an object "is a" concrete example of a class [1]. For instance, a first-year "is a" Student. Looking at it in terms of instantiation, then, a first-year is an instantiation of the Student class. In Object-oriented Analysis and Design, a class is represented by a class name, attributes/properties and operations that can be performed on it. This is as per the Unified Modeling Language (UML) that defines a set of system models required during system development. Thus, objects are then created from the class. As a result, classes end up becoming data types, in this case, Abstract Data Types (ADTs).

System Modeling Using Class Diagrams

Determination of classes to be created for a given system begins with system modeling. System modeling is the process of developing abstract models of a system. Each model aims at presenting a different view or perspective of the system. System models are developed based on the requirements generated through the requirements engineering process. In most cases, system modeling is anchored over the Unified Modeling Language (UML) which is an ISO standard that outlines a set of models required to offer graphical representation of systems.

System modeling helps in communicating system requirements in a graphical manner. This aids analysts to understand the functionality of the system as well as facilitate communication with customers.

There are many different types of system models, each of which serves a different purpose or communicates a specific message. These models include context diagrams, data flow diagrams, use case diagrams, class diagrams, entity relationship diagrams, activity diagrams, communication/collaboration diagrams, sequence diagrams, statechart diagrams, deployment diagrams, among others.

Class diagrams represent a static perspective of a given application. Apart from just visualizing, describing and documenting a system, class diagrams also aid in construction of executable codes of any system. They describe the attributes and operations on classes identified for the system. In addition, they also describe the constraints imposed on the system. According to Kendall and Kendall [4], class diagrams are used during the development of object-oriented system models by showing system classes and their associations, where:

- A class is represented by a compartmented rectangle where the top compartment contains attributes of the class while the lower compartment contains the properties of the class.
- Links or relationships between classes may be associations, compositions, or generalization, each of which has a different connection symbol used.

This topic focuses on association kind of relationship between classes. Later in the course, the topic on inheritance will demonstrate generalization. Thus, Figure 1 shows an example of a class diagrams founded on association relationships among three classes, namely, Student, Course, and Section. Each class is represented by a compartmented rectangle outlining the class name, class attributes, and operations to be performed on the class.

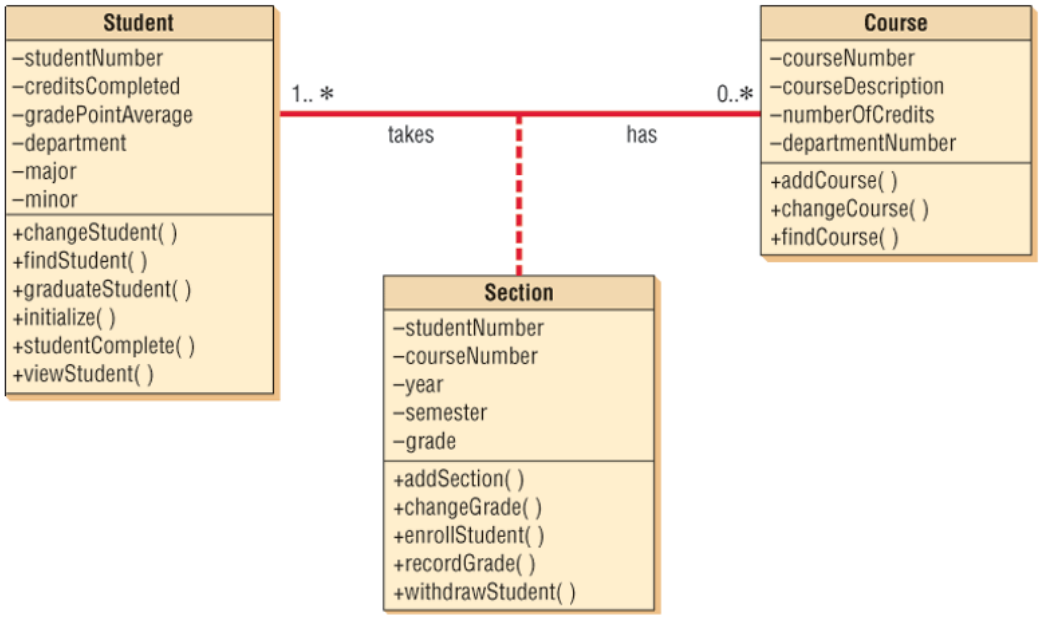

Figure 1. Example of an Association Class Diagram [4]

It is worth noting that each class attribute and operation is preceded by a plus (+), minus (-), or hash (#) sign representing public, private, or protected status respectively as shown in Figure 1. This aids programmers in determination of access specifiers to be associated with each of the elements.

In summary, class diagrams aid in analysis and design of the static view of an application, communicate responsibilities of a system, act as a basis for component and deployment diagrams, and also aid in forward and reverse engineering.

Creating a Program with Multiple Classes

Consider a simple example of a system that computes the area of a circle. The system could be represented by the class diagram shown in Figure 2.

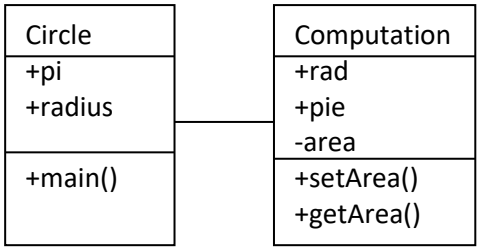

Figure 2. Class Diagram Example

Based on the class diagram, the following interpretation can be made:

- The class diagram is made up of two classes, namely, circle and computation.
- The two classes are related through an association kind of relationship.
- The circle class comprises of the pi and radius attributes and the main() method, all of which are public.
- The computation class is made up of public rad and pie attributes, private area attribute, and two public methods, namely setArea() and getArea().

The Circle and Computation classes could be created normally based on concepts covered in previous topics. However, using such an approach will fail to appropriately implement the association relationship between the two classes. Thus, for successful implementation, the following points have to be put into consideration first:

1. The setArea() and getArea() methods of the Computation class must be defined as setter/mutator and getter/accessor methods respectively.
2. Within the main() method of the Circle class, an object of the Computation class must be defined to facilitate communication between the two classes and subsequent execution of the setArea() and getArea() methods from the main() method.
3. The two classes have to be created within the same package since they are both a part of one program.

Figure 3 shows the code for the Computation class which is the most appropriate class definition to begin with since its object will be created in the Circle class at a later stage, in the next subsection.

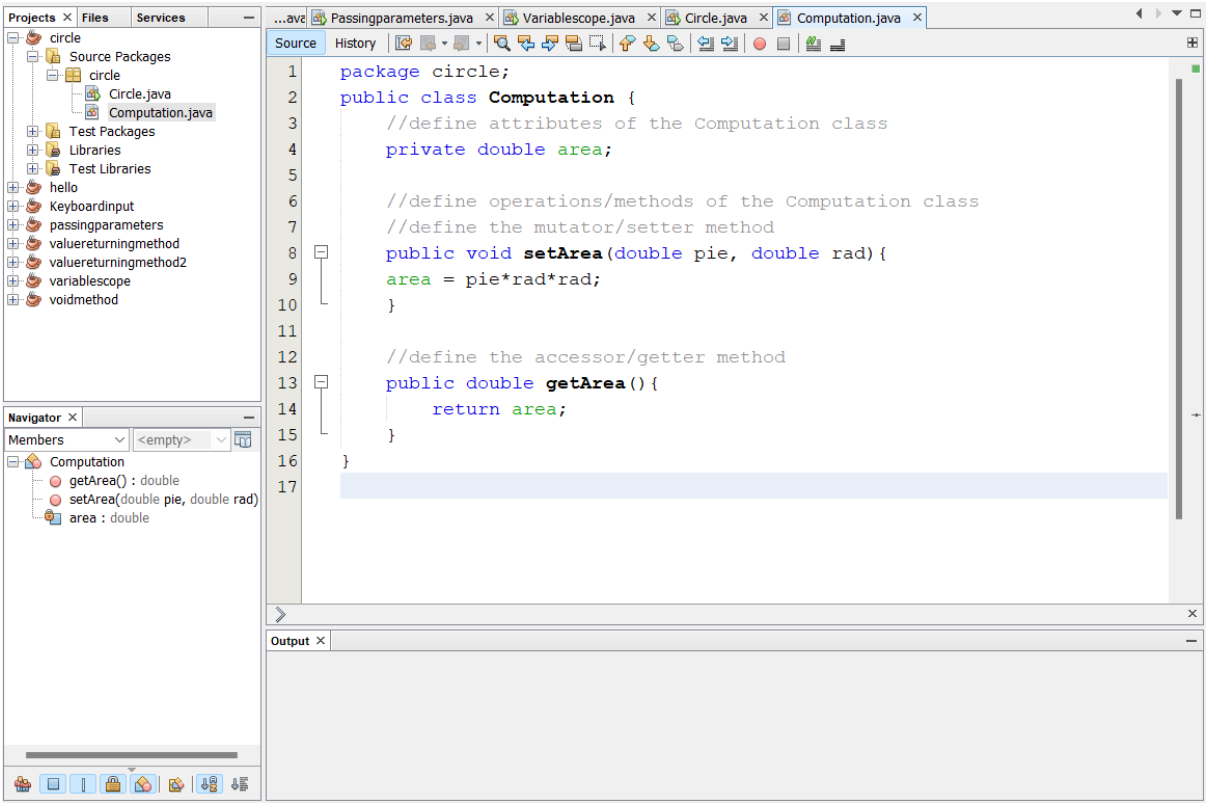

Figure 3. Computation Class with Setter and Getter Methods

Note that the setArea() is a setter/mutator method while getArea() is a getter/accessor method, both of which define the operations performed by the Computation class. As a rule of convention, setter/mutator methods are void methods used to perform computations and set or change values for specific variables. In this case, the setArea() method computes the area of a circle then sets the result to the area variable. On the other hand, getter/accessor methods are value-returning methods that return results set to specific variables by setter/mutator methods. In this case, the getArea() method returns the result set in the area variable by the setArea() method. This means that the setter/mutator method has to be executed first before the getter/accessor method is used to retrieve and return its results.

Since the Computation class has no main() method, the Circle class has to be used to execute both its setter and getter methods. This can only be achieved through definition of an object of the Computation class within the main() method of the Circle class. This will be discussed in the next subsection.

Objects

An object creates an instance of a given class within another class. Thus, to execute methods of a class in another class, an object of the class containing the methods has to be defined first. This aids in the creation of an instance of that class.

Syntax

classname objectname = new classname();

Example

To create an object of the Computation class for use in the Circle class, the following statement should be used.

Computation compute = new Computation();

In this example, Computation is the class, while compute is a user-defined variable representing the object name. The concept of creating objects of a class embraces the aspect of Abstract Data Types (ADTs), such that the object being created appears as a variable of the class type. For instance, in our example, compute is a variable of type Computation. Thus, the Computation class is an Abstract Data Type (ADT).

The object of a given class can now be used to execute methods defined in the class.

Syntax

objectname.methodname(argument-list);

Example

To execute the setArea() method of the Computation class, the following statement should be used. Assume that the two parameters of the methods are represented by values 3.142 and 2.0 respectively.

compute.setArea(3.142, 2.0);

Putting all this together, the program code in Figure 4 could hence be used to implement the Circle class that references the Computation class. Take note of the comments in the code that explain roles of the different code segments.

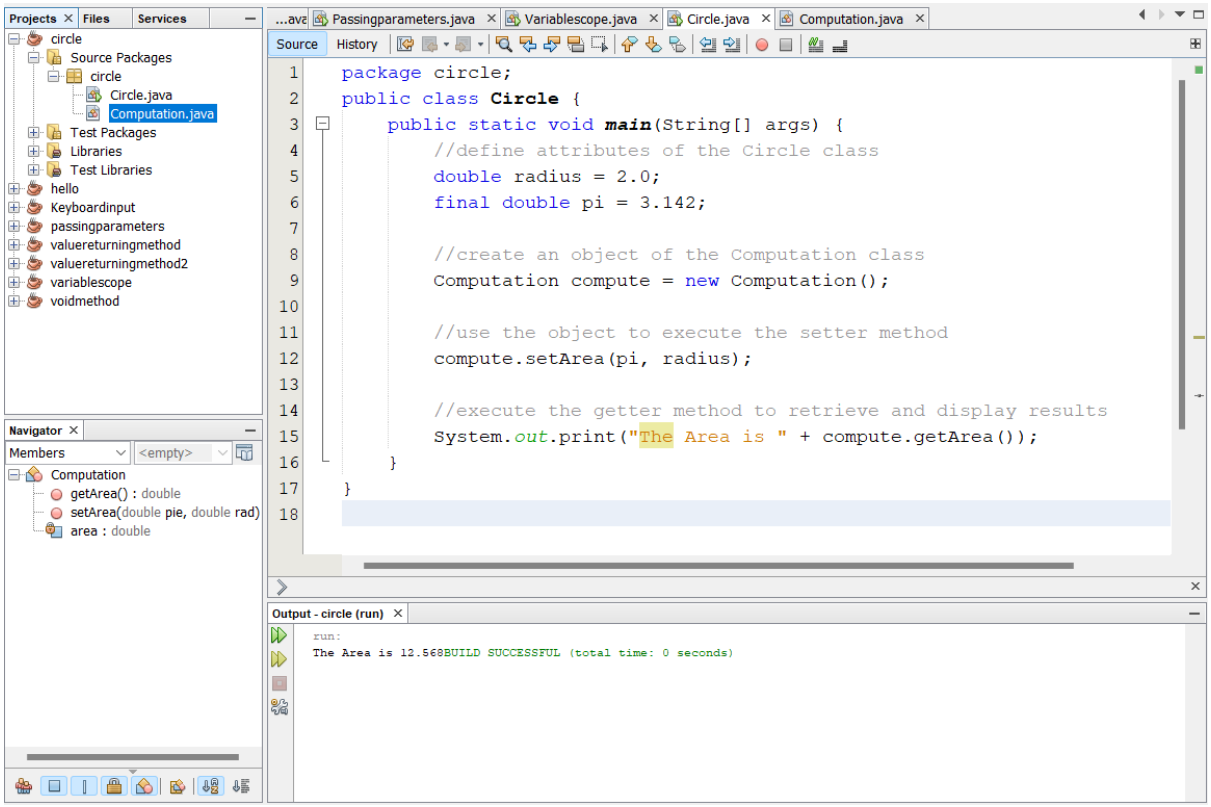

Figure 4. Circle Class with the Computation Object

Based on the example, the Circle class begins with a definition of its attributes, namely, radius and pi. Since computation of the area is performed in a separate class, an object of the Computation class is created, in this case, compute. The object is then used to execute the setArea() method by passing the pi and radius to it as parameters. Finally, the object is also used to execute the getArea() method which returns the area to the Circle class. The output window shows the area as 12.568 after execution of the setter and getter methods of the Computation class.

The Object-Oriented Aspect of Classes and Objects

The concepts of classes and objects is the foundation for the object-oriented programming paradigm. All programming languages under the paradigm must embrace the use of classes and objects as described in this topic. Other object-oriented features like inheritance, encapsulation, polymorphism, among others, all of which are founded on the existence of classes and objects will be covered later in the course.

Summary

The topic has established a foundation for the concept of object-orientated programming style. Class and objects form the first feature of object-oriented programming to be discussed in this course, where a class is a group of related objects. Thus, an object is an instance of a class. The topic has also explained how classes and their objects become data types, in this case Abstract Data Types. Finally, the process of defining and executing mutator and accessor methods has been discussed and demonstrated. Other features of the object-oriented paradigm will be covered later in the course, all of which are founded over classes and objects.

Check Points

1. Using an example of your own choice, discuss how UML class diagrams relate to object-oriented programming.
2. Using an example, demonstrate how the structure of a class in Java embraces UML class diagrams.
3. Differentiate between mutator and accessor methods as used in object-oriented programming.
4. Describe the process of executing mutator and accessor methods using an object of their class.
5. Describe how classes become Abstract Data Types.

Core Textbooks

1. Joyce Farrell, Java Programming, 7th Edition. Course Technology, Cengage Learning, 2014, ISBN-13 978-1-285-08195-3.
2. Malik, Davender S. JavaTM Programming: From Problem Analysis to Program Design, International Edition, 5th Edition, Cengage Learning.

Other Resources

3. Daniel Liang, Y. "Introduction to Java Programming, Comprehensive." (2011).
4. Malik, Davender S. JavaTM Programming: From Problem Analysis to Program Design, International Edition, 4th Edition, Cengage Learning, 2011.
5. Shelly, Gary B., et al. Java programming: comprehensive concepts and techniques. Cengage Learning, 2012.

References

[1] Farrell, J., Java Programming, 7th Edition. Course Technology, Cengage Learning, 2014, ISBN-13 978-1-285-08195-3.

[2] Malik, D. S., JavaTM Programming: From Problem Analysis to Program Design, International Edition, 5th Edition, Cengage Learning.

[3] Sebester, R. W., Concepts of Programming Languages, 12th Edition, Pearson, 2018, ISBN 0-321-49362-1.

[4] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.