Object Oriented Programming – Java 1

Lecture 9

Encapsulation and Abstraction

Dr. Obuhuma James

Description

Data and information hiding is crucial in programming, especially for security purposes. This topic hence explores encapsulation and abstraction as other features of object-oriented programming, in addition to objects, classes and inheritance already covered. Both encapsulation and abstraction aid in data hiding of some form. The two look somewhat similar, however, they are different in concept and implementation.

Learning Outcomes

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

- Describe the concept of encapsulation as a feature of the object-oriented programming style.
- Describe the concept of abstraction as a feature of the object-oriented programming style.
- Differentiate between encapsulation and abstraction by demonstrating their implementation.

Overview of Encapsulation and Abstraction

Data and information hiding is a critical aspect in programming, particularly in object-oriented programming where security and controls on access and use of data is required. This is in addition to other object-oriented programming features that include classes and inheritance covered in earlier topics and polymorphism to be covered in the next topic. Encapsulation and abstraction offer two ways of implementing data and information hiding in object-oriented programming. The two approaches are different in concept and implementation despite the fact that they always seem to be similar. For instance, while encapsulation hides the internal working from the outside world, abstraction hides the complexity of the code. This will be discussed in subsequent subsections of this topic with examples of how they are implemented.

Encapsulation

Encapsulation is the process of enclosing data and methods within objects [1]. Using classes, data and operations can be combined into a single unit through a process called encapsulation [2]. In return, an object becomes a self-contained entity, operations can (directly) access the

data, but the internal state of an object cannot be manipulated directly [2]. Thus, declaring variables as public makes them to be assigned values directly using the assignment operator. In some instances, especially when data hiding is required, encapsulation necessitates variables to be declared as private. In which case, a client application should be able to access such variables only through public interfaces [1]. Such interfaces are a class's public methods. This hence refers back to the concept of setters/mutators and getters/accessors already covered in an earlier topic in this course. Setter methods set values to private variables while getter methods return values set to the variables. Both methods act as interfaces to data hidden through private variables. They can allow further controls to be enforced on data values to be set to private variables and what is to be retrieved from the variables [1].

Hidden implementation methods embraced through setters and getters are often perceived to be existing in a black box [1] kind of setup. This means that they can be used without having to bother to understand how they work. For instance, when you create a product() method to compute the product of two integer values, at the point of execution, only two integer arguments are supplied to the method at the point of calling it to execute. Whatever happens behind the scenes to compute the product is of no concern to the user. The user is only interested in receiving the product in return. Thus, according to Farrell [1], an important principle of object-oriented programming is implementation of information hiding, which describes the encapsulation of method details.

Implementing Encapsulation

Consider the Computation class example shown in Figure 1. The class has two methods, one acting as a setter while the other one is a getter. These are the setArea() and getArea() methods respectively.

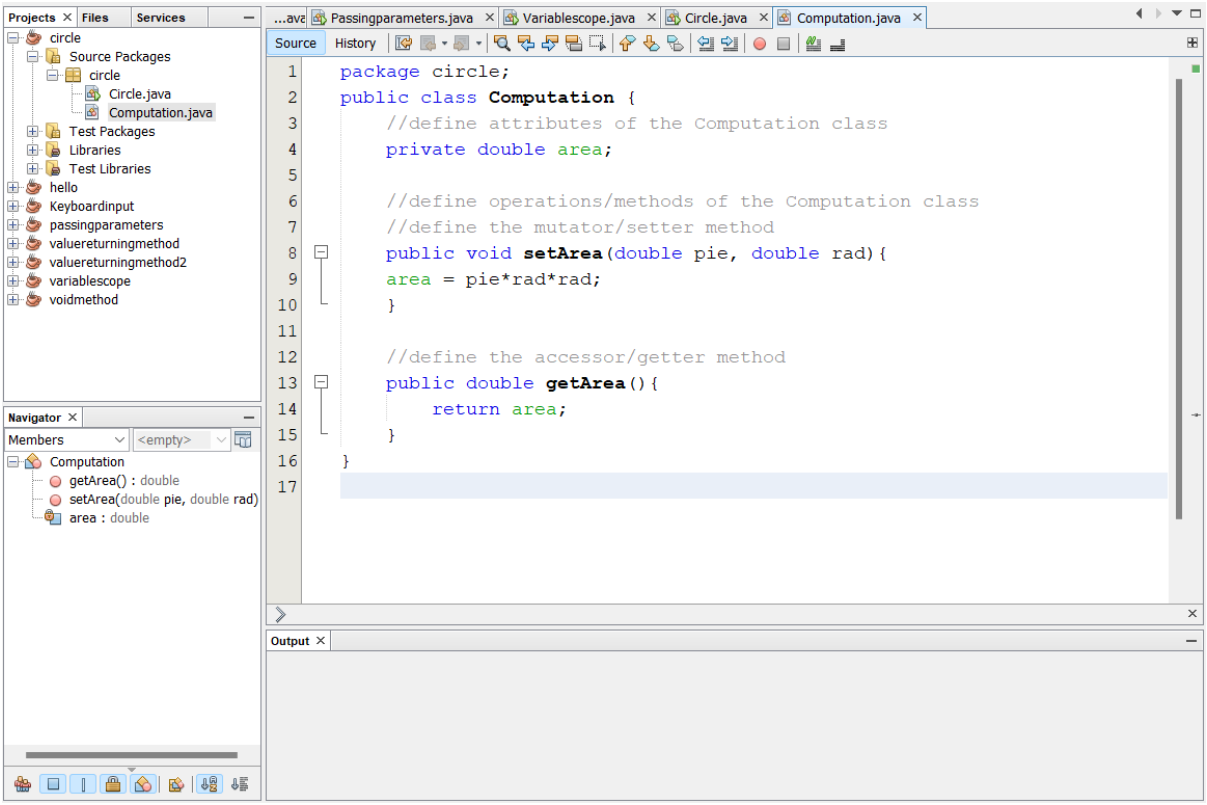

Figure 1. Computation Class with Setter and Getter Methods

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. A private double variable called area has been defined to hold the area of the circle that will be set by the setArea() and returned by the getArea() methods.

The concept of encapsulation is demonstrated in the program in two ways:

1. The area variable is defined as a private variable, as shown in line 4, to limit on its accessibility. This in essence is a form of information hiding since client applications can only access the variable through public interfaces.
2. The setArea() and getArea() methods represent setter and getter methods respectively. The setter method sets a value to the private area variable while the getter method returns the set area value. Both methods act as interfaces to data hidden in the private area variable. The two methods act as black boxes, such that whatever happens in them is hidden from the user, in this case, the rest of the application. The call to execute the setArea() method only needs to provide the pie

and radius as arguments while the call to the getArea() only expects to receive the returned area value as shown in line 12 and 15 of Figure 2.

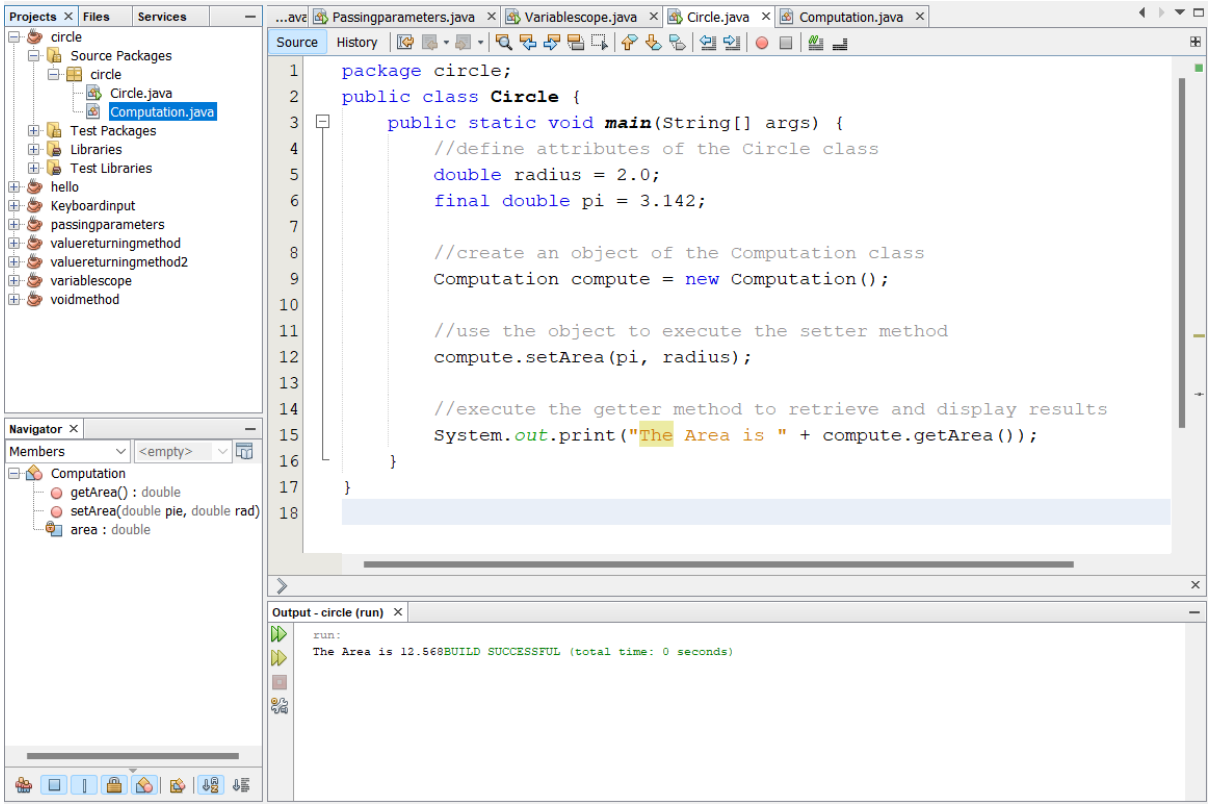

Figure 2. Circle Class with the Computation Object

In summary, a fully encapsulated class can be created by making all the data members of the class private. After which, setter and getter methods can then be defined as black boxes for setting and returning data in the variables. Thus, a well-designed class uses private instance variables, getter methods, and (if needed) setter methods to implement encapsulation [2]. By so doing, encapsulation accrues the following benefits in programming:

1. Class variables can be made read-only or write-only by having only the getter or setter method respectively at one given point in time.
2. A class can have total control over what is stored in its variables. For instance, specific logic can be provided in setter methods to dictate what can happen or not happen on a given variable regarding data to be set in it.
3. It is a way to achieve increased security through data hiding since other classes will not be able to access the data through the private data members.
4. Encapsulated classes are easy for testing, thus, a better approach for unit testing.

5. The use of setters and getters enhances flexibility in programming since the programmer can modify sections of the program code without affecting other sections.

Abstraction

Abstraction is an object-oriented programming feature that allows a method name to be used to encapsulate a series of statements [1]. Abstraction is achieved through the definition of abstract classes that contain abstract methods, where an abstract method is declared with the keyword “abstract”, it has no body to facilitate implementation, and must be implemented in a subclass [1]. Abstraction can hence be visualized as a process of exposing only (all) the necessary details while hiding other details. This means that objects are reduced to their essentials such that only the necessary characteristics are exposed to users.

Implementing Abstraction

The first step is to define an abstract class using the abstract keyword.

Syntax

```
abstract class classname{
        //class body
}
```

The following are some of the characteristics of abstract classes:

1. Objects of an abstract class cannot be directly defined for use. Only objects of its subclass can be defined and used.
2. An abstract class can contain abstract methods (methods without bodies) as well as non-abstract methods (methods with bodies). Note that non-abstract classes, like those covered upto this far cannot contain abstract methods.
3. There is always a default constructor in an abstract class, it can also have a parameterized constructor.

The second step is to define abstract methods, which are methods with no implementation i.e., without method bodies. Declaration of abstract methods includes the abstract keyword and must end with a semicolon after the parenthesis.

Syntax

accessspecifier abstract returntype methodname();

Figure 3 shows an example of an abstract class that specifies the basic structure required for an animal subclass.

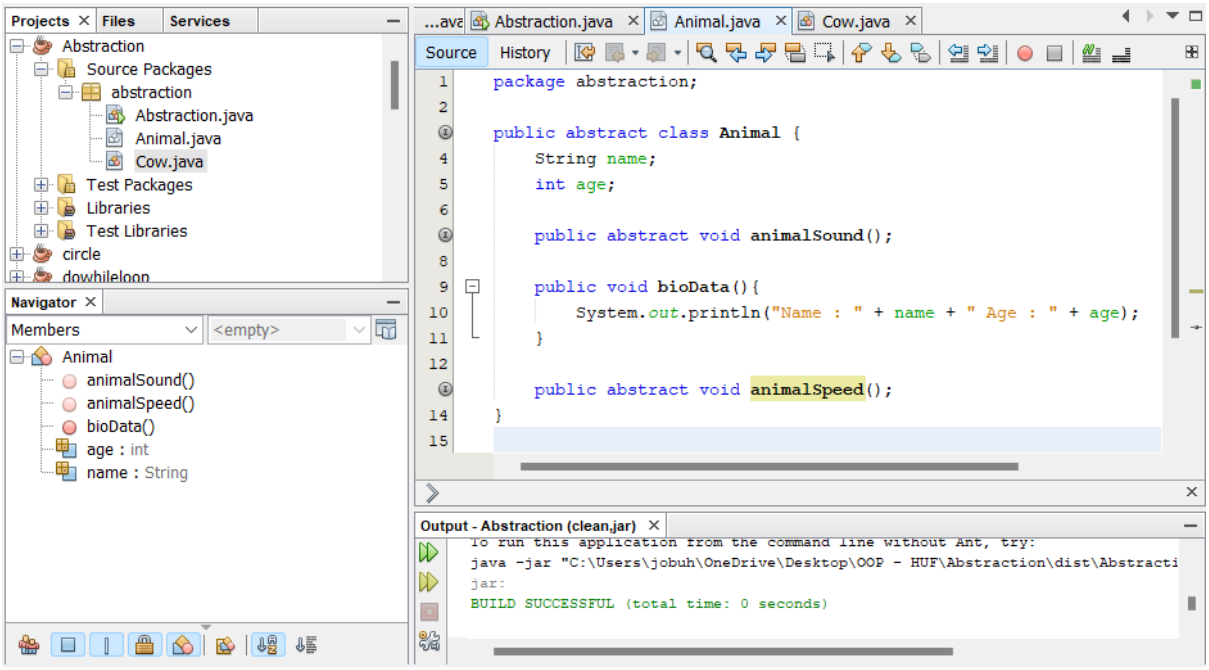

Figure 3. Example of an Abstract Class with Methods

The abstract class enforces and organizes exactly what each subclass of the Animal class should have. It basically outlines the structure (or idea) without caring about its implementation, which is already a form of information hiding. In this case, based on the abstract Animal class, it means that every animal has a name and age. Furthermore, it has a unique sound and speed of movement as specified by the animalSound() and animalSpeed() abstract methods. Part of the abstract class is the bioData() method that is non-abstract. The method basically collates each animal's bio data by implementing it within the abstract class.

Implementation of abstract methods has to be done in a non-abstract class that inherits the abstract class. Non-abstract methods defined in the subclass should hence have same names as those defined in the abstract class. Figure 4 shows an example Cow class that inherits from the Animal class. The two methods animalSound() and animalSpeed() in this case are used to implement their matching methods in the abstract class.

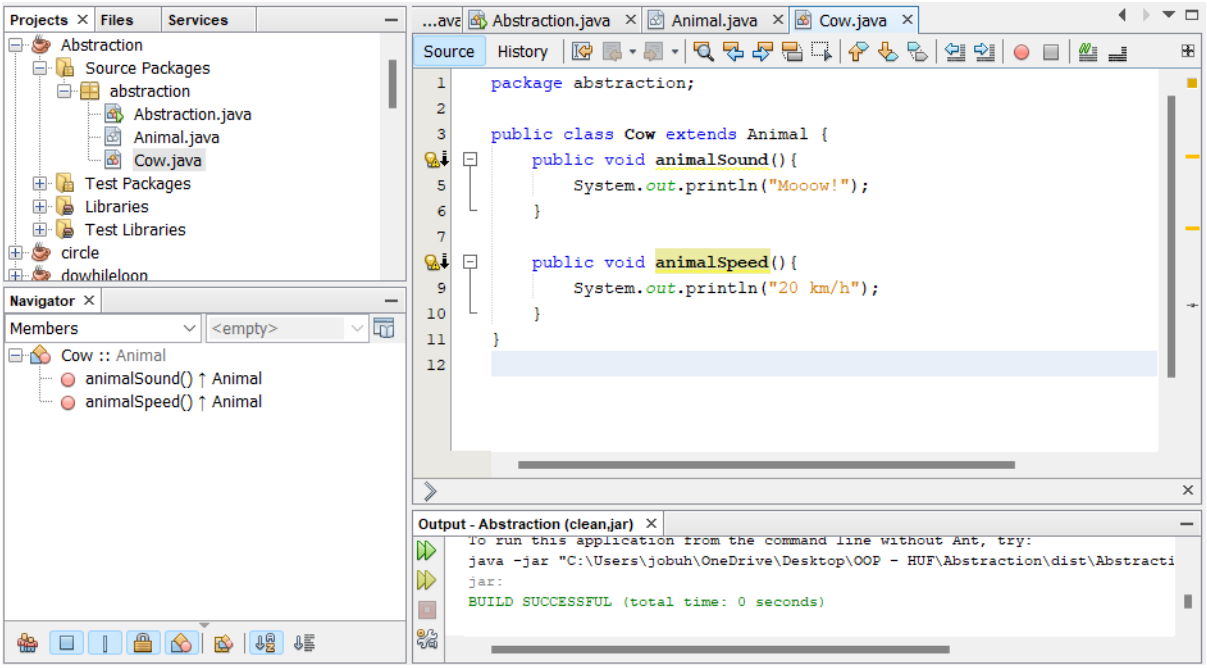

Figure 4. Example of How to Implement Abstract Methods

To package this together, Figure 5 outlines an Abstraction class used to instantiate the abstract class. Notice the fact that line 5 creates an object of the Cow class instead of the Animal class. This is because abstract classes cannot be instantiated directly, thus, objects of abstract classes cannot be created.

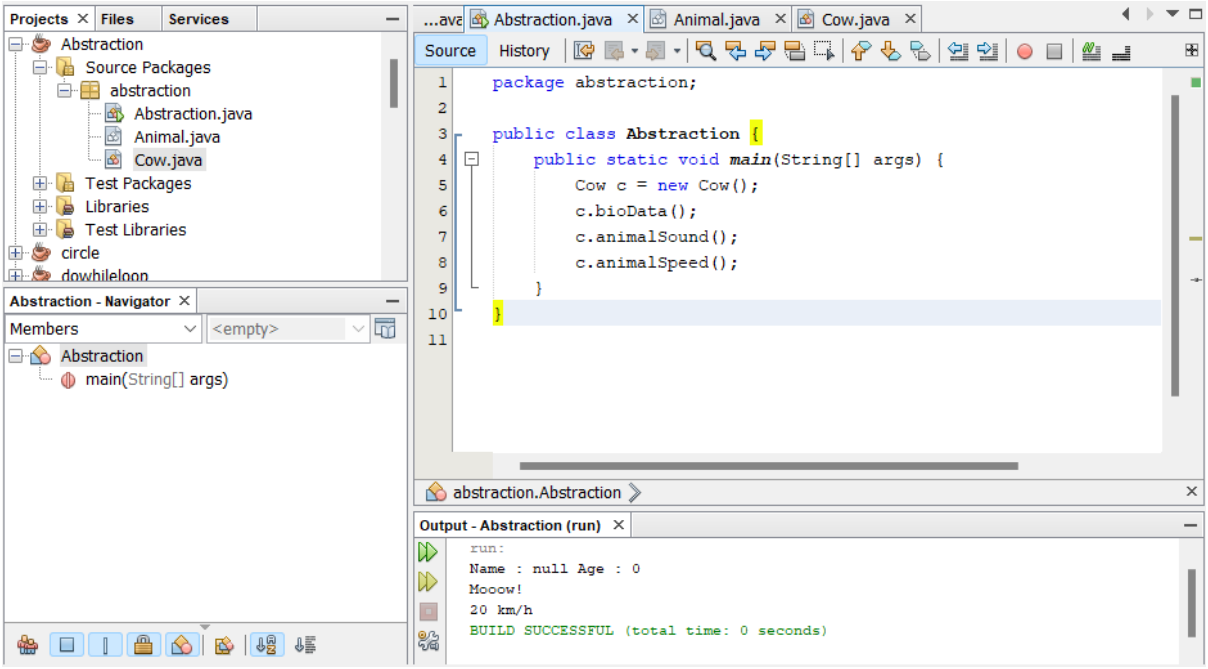

Figure 5. Example of How to Create Objects of an Abstract Class

Hence, by creating objects of subclasses of abstract classes allows you to execute abstract classes indirectly. In this case an object of the Cow class allows executions of the Animal class. Lines 6 to 8 then calls the respective methods to execute, including the non-abstract method specified in the abstract class. The output window shows the results of executing all the three methods.

In summary, abstraction can be at two levels, namely, data abstraction and control abstraction. Data abstraction is the process of hiding certain details and showing only essential information to the user. This can be achieved using either abstract classes or interfaces. Control abstraction on the other hand is the process of determining statements that are similar and that repeat many times and exposing them as a single unit of work. This is commonly the approach used to create functions geared at performing specific tasks. By so doing, abstraction results to the following benefits in programming:

1. Reduction in the complexity of viewing things.
2. Increase in software reuse and avoidance of code duplication.
3. Increase application security and confidentiality since only necessary details are exposed to the user.
4. Ease maintenance burden where classes can be quickly understood and debugged with little fear of harming other modules.

Differences Between Data Encapsulation and Data Abstraction

The following are some notable differences between data encapsulation and data abstraction as applied in object-oriented programming:

1. Encapsulation is normally perceived to be one step beyond abstraction.
2. Abstraction hides unnecessary detail but shows the essential information. Encapsulation on the other hand hides the code and data into a single entity or unit aimed at shielding the data from the outside world.
3. Encapsulation binds the data members and methods together while data abstraction shows the external details of an entity to the user as it hides the details of its implementation.

4. Abstraction provides access to a specific part of data while encapsulation hides the data.

Summary

The topic has covered the concept of data hiding as applied to object-oriented programming by exploring abstraction and encapsulation. The two mechanisms used for data hiding have been discussed and demonstrated. While encapsulation hides the internal working from the outside world, abstraction hides the complexity of the code. Encapsulation is implemented using private variables, setter and getter methods. On the other hand, abstraction focuses on the use of abstract classes and abstract methods. The two aspects of programming are amongst the main features of object-oriented languages.

Check Points

1. Discuss the concept of information hiding as applied to object-oriented programming.
2. Differentiate between encapsulation and abstraction as applied in data and information hiding.
3. Using an example, discuss encapsulation as a technique for data and information hiding in object-oriented programming.
4. Using an example, discuss abstraction as a technique for data and information hiding in object-oriented programming.
5. State and explain the main benefits of encapsulation in object-oriented programming.
6. State and explain the main benefits of abstraction in object-oriented programming.

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.

[5] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.