Object Oriented Programming – Java 1

Lecture 8

Inheritance

Dr. Obuhuma James

Description

This topic explores inheritance as the second feature of object-oriented programming. Inheritance is a concept applied based on the nature of relationship between two or more classes where one class takes up characteristics of another class. The use of UML class diagrams to represent inheritance and how to transform it to a program is discussed and demonstrated in the topic. The concept of inheritance will be useful in other topics, including the topic on abstraction.

Learning Outcomes

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

- Describe the concept of inheritance as a feature of object-oriented programming.
- Create a class diagram that represents inheritance.
- Interpret and transform a class diagram to a program implementation of inheritance.

Overview of Inheritance

Creation and usage of classes in object-oriented programming goes beyond objects, methods, arrays, and other aspects covered in previous topics. Inheritance is one of the salient features of the object-oriented programming paradigm. Through inheritance, a class borrows (inherits) all or part of the characteristics of another class. These characteristics come as additional characteristics to those already possessed by the inheriting class. According to Farrell [1], inheritance can be viewed as a mechanism that enables one class, the subclass, to inherit both the behaviour and attributes of another class, the super class.

Inheritance thus means that there must be two or more participating classes where one class acts as a parent class while the other acts as a child class. The child class possesses its own unique features in addition to those inherited from its parent. The parent class is sometimes referred to as the base class or super class while the child class is sometimes referred to as the derived class or subclass. There are two types of inheritance, namely, single inheritance and multiple inheritance [2]. In single inheritance, the subclass is derived from one super class while in multiple inheritance, the subclass is derived from more than one superclass [2]. This course will only demonstrate single inheritance which is the only type of inheritance supported in Java.

UML Class Diagram Representation for Inheritance

In topic four on classes and objects, the application of the Unified Modeling Language (UML) in system modeling is discussed. The focus was more on class diagrams that are later transformed into classes in object-oriented programming. As earlier mentioned, a class diagram is composed of classes with relationships among them. There are different relationships that can be established between two classes. This includes association, composition, aggregation, generalization, among others. In topic four, the focus was on association kind of relationship, where the concept of creation and usage of objects formed the basis of the discussion.

This topic shifts gears to generalization as a form of relationship between classes. Generalisation symbolizes inheritance. In object-oriented programming languages, like Java, generalization is implemented using the class inheritance mechanisms built into the language [5]. According to [5], during system modeling, examination of classes forming the system is important as a way of determining existence and scopes for generalization. In such cases, it means that common information will be maintained in one place only, for instance in one class. This design practice means that if any changes are proposed, then focus is only made to specific classes affected by the change.

In UML class models, generalization is denoted by an association with an arrowhead pointing to the more general class [5]. For instance, in Figure 1, the Hospital Doctor and General Practitioner classes inherit attributes and operations from the Doctor class.

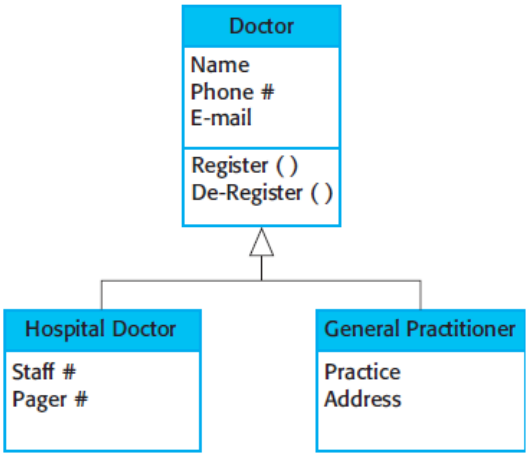

Figure 1. Class Diagram Showing Inheritance [5]

In essence, the attributes and operations associated with the higher-level class, the Doctor class, are also associated with the lower-level classes, Hospital Doctor and Practitioner Doctor classes. Thus, the lower-level classes, subclasses, inherits the attributes and operations from their superclass. Each lower-level class then adds more specific attributes and operations that it possesses.

Implementing Inheritance

Consider a simple example of two classes that implement inheritance as shown in the class diagram in Figure 2.

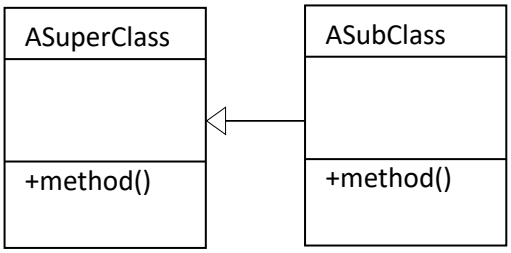

Figure 2. Class Diagram Example

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

- There are two classes, ASuperClass and ASubClass.
- Based on the generalization kind of relationship, the ASubClass inherits from the ASuperClass.

To create a program that implements the model, the ASuperClass, which represents the base class is created normally. Thus, the following code should be used.

```
package inheritanceexample;
public class ASuperClass{
        public ASuperClass(){
                System.out.print("Executing the super class constructor");
        }
}
```

Since the ASubClass represents the subclass, its class has to be defined in such a manner that it implements inheritance. The keyword "extends" is used in this case. Thus, the following syntax is used in defining the class.

```
accessspecifier class SubClass extends SuperClass{
        //class body
}
```

Thus, to create a program for the ASubClass, the following program code should be used.

```
package inheritanceexample;
public class ASubClass extends ASuperClass{
        public ASubClass(){
                System.out.print("Executing the subclass constructor");
        }
}
```

To demonstrate the full concept of inheritance, another class with a main() method has to be defined that will be used to create an object of the subclass. The program in Figure 3 show all the three classes together with the class that has the main() method. Comments have been added to explain the role of each code segment.

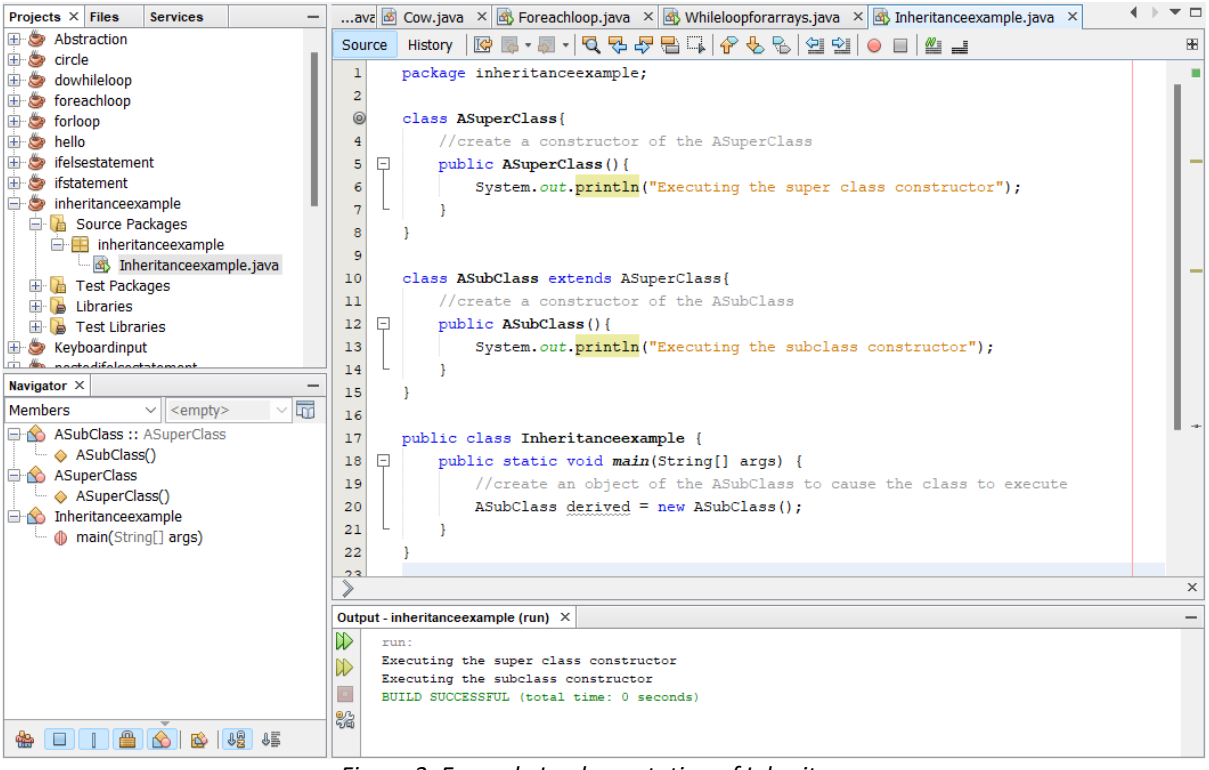

Figure 3. Example Implementation of Inheritance

Notice the fact that both statements from the ASuperClass and ASubClass are part of the output, as shown in the output window, yet in the main() method, the object of the ASubClass

was the only one created. This means that the ASubClass inherits properties of the ASuperClass thus the output of the statement “Executing the super class constructor” in addition to the statement “Executing the subclass constructor” contained in the ASubClass.

Using the same approach, any form of inheritance can be modelled using a class diagram then further implemented using an object-oriented programming language.

Benefits of Inheritance

Creation of derived classes through inheritance accrues a number of benefits including [1]:

1. Saving on time.
2. Reduction of errors.
3. Reduction of the amount of fresh learning required to use a new class.

Summary

The topic has covered inheritance as one of the concepts of object-oriented programming. The differences between base and derived classes have been clearly outlined. A demonstration of how to implement inheritance guided by UML class diagrams has been done. Finally, benefits of creating derived classes as a result of inheritance have also been outlined.

Check Points

1. Discuss the concept of inheritance as applied to object-oriented programming.
2. Differentiate between a base class and derived class as applied to inheritance.
3. Consider a car class and vehicle class, draw a UML class diagram for the two classes.
4. In reference to question 3, implement the model using the Java programming language.
5. State and explain the main benefits of inheritance 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.