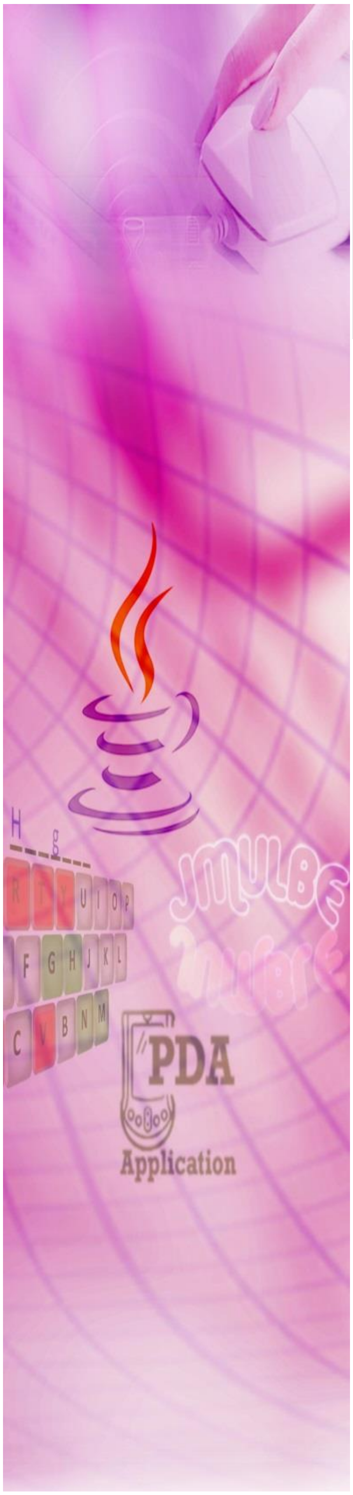

Session 7

Instructor Inputs

Session Overview

This session covers Chapter 5 of the book, Introduction to Java - SG. In this session, the students will learn how the inheritance and polymorphism features are implemented in Java.

Implementing Inheritance

Handling Tips

Start the session by asking the students about the inheritance feature of the object-oriented programming and if they have ever applied it in a programming language. Now, discuss the Classic Jumble Word game scenario in detail with the students to explain the need of using inheritance for the game. Then, tell them that in Java, the classes can be related to each other and a class can acquire the features of another class. Now, tell them that to accomplish the preceding requirement, they can use the inheritance feature in Java.

Then, explain the types of inheritance supported by Java. Explain each inheritance type and ensure that the students are able to differentiate amongst the various types of inheritance. Tell them that in Java, inheritance from more than one class is not supported. Further, after ensuring that the students have gained clarity on the concept of inheritance, tell them that in Java, they can implement inheritance by extending a non abstract class and an abstract class and by implementing an interface. Explain the single level inheritance example given in the SG. Further, explain the abstract classes and relate them to a blueprint, and then explain their implementation with the help of the code snippet given in the SG. Emphasize on the usage of the `this` and `super` keywords.

Ask the students if they have any questions regarding the inheritance of classes and ensure that they have clearly understood the concept of inheritance. Then, introduce the concept of interface in Java. Tell them that an interface is a service contract that all the classes implementing the interface need to follow. Ensure that the students are able to understand the difference between an abstract class and an interface. Further, you can tell the students when to use an interface or an abstract class with the help of the information given in the Additional Inputs topic. Before concluding the topic, explain the example for implementing multiple interfaces given in the Additional Examples topic.

Additional Inputs

The following points must be considered when implementing abstract classes or interfaces:

- With an abstract class, you can provide a default implementation for some of the methods. However, with an interface, you cannot provide any implementation.
- An abstract class should be used if you want to declare non public members. In an interface, all methods must be public.
- If you think you may need to add methods in the future, then an abstract class is a better choice. If you add new methods to an interface, then all the classes that are already implementing that interface will need to be changed to implement the new methods.
- Interfaces are also good when you want to have something similar to multiple inheritance, since you can implement multiple interfaces.

Additional Examples

Inheriting an Interface

The following code demonstrates how to implement multiple interfaces in a class:

```
interface Greeting
{
void print();
}
interface Calculate
{
void calculation();
}

class Addition implements Greeting, Calculate
{
int a,b=5;
public void print() {
System.out.println("Welcome User !!");
}
public void calculation()
{
System.out.println("The addition result is " +(a+b));
}

public static void main(String args[])
{
Addition obj = new Addition();
obj.print();
obj.calculation();

}
}
```

The output of the preceding code is:

```
Welcome User !!
The addition result is 5
```

Implementing Polymorphism

Handling Tips

Ask the students if they can recall the concept of polymorphism in case of object-oriented programming. Elicit their responses, and then tell them that polymorphism is the ability of an object to take many forms. Next, introduce the two categories of polymorphism to the students.

Tell the students that the static polymorphism is implemented through method overloading or constructor overloading. In case of static polymorphism, methods can exist with the same name but with a different argument lists. Explain the concept with the help of the code snippet given in the SG. Before concluding the topic, explain the example for the static polymorphism given in the Additional Examples topic.

Then, tell the students that the dynamic polymorphism refers to the change in the form depending upon the circumstances at runtime. Dynamic polymorphism is implemented through method overriding. Next, explain the scenario of the Classic Jumble Word game and how to implement polymorphism for the same. Before concluding the topic, explain the example for the dynamic polymorphism given in the Additional Examples topic.

Additional Examples

Static Polymorphism

The following code demonstrates the concept of static polymorphism:

```
public class PolymorphismDemo {

    void product(int x, int y) {
    System.out.println("The product of two numbers is "+(x*y));

    }
    void product(int x, int y, int z) {
    System.out.println("The product of three numbers is "+(x*y*z));
    }

    public static void main(String[] args) {
        // TODO code application logic here
PolymorphismDemo obj1 = new PolymorphismDemo ();
obj1.product(10, 20);
obj1.product(10,20,30);

    }
}
```

The output of the preceding code is:

```
The product of two numbers is 200
The product of three numbers is 6000
```

Dynamic Polymorphism

The following code demonstrates the concept of dynamic polymorphism:

```
class Person {

    String name = "Ricky";
    int age = 50;

    public void Display()
    {
        System.out.println("The name of the person is: " + name);
        System.out.println("The age of the person is: " + age);
    }
}
```

```
class Student extends Person
{
    String name = "Eric";
    int age = 20;
    public void Display() {
        System.out.println("The name of the student is: " + name);
        System.out.println("The age of the student is: " + age);
        super.Display();
    }
}
class Test {
    public static void main(String args[]) {
        Person obj = new Person();
        Student obj1 = new Student();
        obj1.Display();
    }
}
```

The output of the preceding code is:

```
The name of the student is: Eric
The age of the student is: 20
The name of the person is: Ricky
The age of the person is: 50
```

Activity 5.1: Implementing Inheritance and Polymorphism

Handling Tips

Discuss the problem statement with the students.

The solution files, **Employee.java**, **MainMenu.java**, **Person.java**, and **Student.java**, for this activity are located at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 05\Activity 5.1\Solution**

FAQs

- *Why does Java not support multiple inheritance?*

  Ans: Java doesn't support multiple inheritance in order to reduce ambiguity. In case of multiple inheritances, when a class inherits methods from the two superclasses with the same method signature, then it becomes confusing to guess the appropriate method that will be called.

- *Can a constructor be inherited in Java?*

  Ans: No, a constructor cannot be inherited in Java but the constructor of a superclass can be invoked from the subclass.

- *Can a final method be overridden?*

  Ans: No, a final method cannot be overridden.