Lab Manual

Programming with Python and Java (CS 29008)

School of Electronics Engineering, KIIT DU

Chapter 6

Lab 7: Polymorphism in Java

Objective

To understand and apply the concepts of polymorphism in a Java program.

Outcome

After completing this lab, students would be able to develop Java programs using compile-time and run-time polymorphisms.

6.1 What is a polymorphism?

Polymorphism is a property in object-oriented programming that allows a single task to be performed in multiple ways. It helps identify and differentiate between code entities that are similar, thus enhancing efficiency. There are two types of polymorphism in Java:

- compile-time or static polymorphism
- run-time or dynamic polymorphism

In a Java program, it is possible to define multiple constructors or methods with the same identifier. At compile time, when multiple methods or constructors with similar identifiers are encountered, they are resolved on the basis of the number and types of parameters passed to them. This is known as **compile-time** or **static** polymorphism which is achieved by overloading constructors or methods.

Method overriding is a process in which a child class has the same methods as declared in the parent class. This means that if the child class contains

the same methods as the parent class, then the function in the child class is overridden. In Java, method overriding is dynamically resolved at run-time, making it a **run-time** or **dynamic** polymorphism.

6.2 Constructor overloading

Constructor overloading is a feature in object-oriented programming that allows a class to have multiple constructors with different parameter lists, which can differ in the number, order, or types of parameters. It provides flexibility and supports different initialization scenarios for objects of the class. The choice of which constructor to invoke is determined during object creation based on the arguments passed.

Listing 6.1: An example of constructor overloading

```
class Student {
private int age;
private String name;
private String gender;
public Student() { // constructor 1 -- default
  age = 0; name = gender = "";
}
public Student(int a) { // constructor 2
          age = a; name = gender = "";
}
public Student(int a, String n) { // constructor 3
          age = a; name = n; gender = "";
}
// constructor 4
public Student(int a, String n, String g) {
          age = a; name = n; gender = g;
}
void displayData() {
        System.out.println("Name: " + name);
        System.out.println("Gender: " + gender);
        System.out.println("Age: " + age);
    System.out.println("-----------------");
}}
class Main {
        public static void main(String args[]) {
        Student s1 = new Student();
```

```
        Student s2 = new Student(21);
        Student s3 = new Student(21, "XYZ");
        Student s4 = new Student(21, "XYZ", "Male");
        s1.displayData();
        s2.displayData();
        s3.displayData();
        s4.displayData();
}}
```

6.3 Method overloading

Method overloading is a useful feature in object-oriented programming that allows a class to define multiple methods with the same name, but with differing parameter lists within the same scope. This means that we can have several methods with identical names in a class, as long as their parameter types, order, or number differ. This feature can be helpful in situations where we want to perform the same operation on different data types or with different input parameters.

Listing 6.2: An example of method overloading

```
class Student {
private int age;
private String name;
private String gender;
public Student() { // constructor 1 -- default
  age = 0; name = gender = "";
}
void showData() {
        System.out.println("Name: " + name);
        System.out.println("Gender: " + gender);
        System.out.println("Age: " + age);
}
void showData(String name) {
        System.out.println("Name: " + name);
}
void showData(String name, String gender) {
        System.out.println("Name: " + name);
        System.out.println("Gender: " + gender);
}
void showData(String name, String gender, int age) {
```

```
        System.out.println("Name: " + name);
        System.out.println("Gender: " + gender);
        System.out.println("Age: " + age);
}}
class Main {
        public static void main(String args[]) {
                Student s = new Student();
                s.showData();
                s.showData("ABC");
                s.showData("ABC", "MALE");
                s.showData("ABC", "MALE", 21);
}}
```

6.4 Method overriding

In the case of inheritance, if a method is present in both the parent class and the child class, the method in the child class will override the same method in the parent class. This is known as method overriding. When this happens, the method will perform one operation in the parent class and a different operation in the child class. For instance,

Listing 6.3: An example of method overriding

```
class Person {
String name;
void info(){
System.out.println("Name is..." + name);
}  }
class Student extends Person {
int roll;
void info(){
System.out.println("Roll is..." + roll);
}  }
class Main {
public static void main(String args[]) {
Student stud = new Student();
stud.info();
}}
```

Note that, Both the parent class and the child class must have the same method name, the same return type and the same parameter list.

6.5 Lab 7 Exercises

1. Create a class **TimeFormat** that contains three member variables, **hour**, **minutes**, **seconds** of type **int**. Define four constructors with no argument, one argument, two arguments, and three arguments. Also, define a method inside the class **TimeFormat** to display the time in the format **HH:MM:SS** = . [Hint: Follow Listing 6.1 to write the Java program].

2. Define a Java class **BoolPattern** with three Boolean member variables: **A**, **B**, and **C**. Using method overloading, generate binary patterns for one, two, and three Boolean variables. For instance, the output should be:

Binary patterns for one variable
{0, 1}
Binary patterns for two variables
{00, 01, 10, 11}
Binary patterns for three variables
{000, 001, 010, 011, 100, 101, 110, 111}

3. Create a Java program that defines a parent class called **Student** with two member variables: **name** and **roll**. Initialize these variables using a no-argument constructor and include a member method **info()** to print your name and roll number.

Create three child classes: **FirstSem**, **SecondSem**, and **ThirdSem**, each extending the **Student** class. Each child class should have a member variable **SGPA** which is initialized in a no-argument constructor.

Include a member method **info()** in all three child classes that prints the name, roll number and SGPA for all three semesters using inheritance and method overriding.