Topics to cover:

1. Polymorphism and Inheritance
2. Further programmings

1. Polymorphism and Inheritance

1.1 Polymorphism

What is Polymorphism in Java?

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

Types of polymorphism:

Compile-Time Polymorphism
Run-Time Polymorphism

Compile-Time Polymorphism:

It is also known as static polymorphism. This type of polymorphism is achieved by method overloading.

```
class PolyExample{
    int Multiply(int a, int b)
    {
        return a * b;
    }

    double Multiply(double a, double b)
    {
        return a * b;
    }
}

class Polymorphism{
    public static void main(String[] args)
    {
        PolyExample pe = new PolyExample();
        System.out.println(pe.Multiply(2, 4));
        System.out.println(pe.Multiply(5.5, 6.3));
    }
}
```

Output:

```
8
34.65
```

Run-Time Polymorphism:

It is also known as Dynamic Method Dispatch. It is a process in which a method call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. Method overriding occurs when a derived class has a definition for one of the member methods of the base class. That base function is said to be overridden. Before we show an example program on run-time polymorphism, we need to discuss inheritance.

1.2 Inheritance

It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Syntax :

```
class derived-class extends base-class
{
  //methods and fields
}
```

We group the "inheritance concept" into two categories:

I. **subclass (child)** - the class that inherits from another class
II. **superclass (parent)** - the class being inherited from

To inherit from a class, use the extends keyword.

Example:

```
import java.io.*;

// Base or Super Class
class Employee {
   int salary = 60000;
}

// Inherited or Sub Class
class Engineer extends Employee {
   int benefits = 10000;
}

// Driver Class
class Inheritance{
   public static void main(String args[])
   {
      Engineer E1 = new Engineer();
      System.out.println("Salary : " + E1.salary
                        + "\nBenefits : " + E1.benefits);
    }
}
```

Output:

```
Salary : 60000
Benefits : 10000
```

In the above example, the Engineer class which is the subclass inherits the attributes and methods from the Employee class which is the superclass. The “Employee” class has salary as its only variable. The “Engineer” subclass is derived from the “Employee” class and thus is able to access its “salary” variable. In addition, it also adds a new variable called “benefits”.

We can implement run-time polymorphism using inheritance. Following example illustrates this.

Example:

```
class Shape {

    // Method of parent class
    void showArea()
    {
        System.out.println("This will show area of a shape");
    }
}


class Circle extends Shape {
    void showArea() { System.out.println("This will show area of a circle"); }
}


class Rectangle extends Shape {
    void showArea()
    {
        System.out.println("This will show area of a rectangle");
    }
}


class AreaShape {
    public static void main(String[] args)
    {
        Shape a;

        // Now we will be calling print methods
        // inside main() method

        a = new Circle();
        a.showArea();

        a = new Rectangle();
        a.showArea();
    }
}
```

Output:

```
This will show area of a circle
This will show area of a rectangle
```

```
class ScienceClass{
        String studentName="";
        void show ( ){
                Sys.,/././.;
        }
}
```

In the above example, the Shape class is the parent class that has the showArea() method with no arguments. This method is re-written in the Circle and Rectangle classes with the same signature i.e., the method is overridden in the subclasses. Now, the specific message is printed depending upon the call at runtime. For example, the superclass object “a” is assigned to the Circle class initially and thus the corresponding message in the showArea() method from the Circle class is printed. Afterwards, “a” is assigned to the Rectangle class and thus a different message is printed, based on the showArea() method from the Rectangle class.

Task: Write a program using polymorphism to convert a number from either binary or to decimal.

```
class NumberType {

   // Method of parent class
   void convert(int num)
   {
      System.out.println("This will convert the number "+num);
   }
}


class Decimal extends NumberType {
   void convert(int num) {
   int decimal = 0;
   int n = 0;
   while(true){
      if(num == 0){
        break;
      } else {
          int temp = num%10;
          decimal += temp*Math.pow(2, n);
          num = num/10;
          n++;
       }
    }
    System.out.println("The decimal representation is: "+decimal); }
}


class Binary extends NumberType {
    void convert(int num)
    {
     int binary[] = new int[40];
     int index = 0;
     System.out.print("The binary representation is: ");
     while(num > 0){
       binary[index++] = num%2;
       num = num/2;
     }
     for(int i = index-1;i >= 0;i--){
       System.out.print(binary[i]);
     }
System.out.println();//new line
    }
}


class NumberConversion{
    public static void main(String[] args)
    {
        NumberType a;

        a = new Decimal();
        a.convert(11011);

        a = new Binary();
        a.convert(3);
    }
}
```

Output:

```
The decimal representation is: 27
The binary representation is: 11
```