

Polymorphism:

- Polymorphism is one of the features of Object oriented programming.
- Polymorphism is an ability of an object to take many forms.
- It allows objects of different classes to be treated as objects of a common superclass.
- Polymorphism occurs when there is inheritance. I.e. classes that are related to each other by inheritance.
- As inheritance allows one to inherit attributes and methods from other classes, polymorphism allows one to perform a single work in different ways.
- It provides code reusability as it allows to represent different data types through a single method/property.
- Types of Polymorphism:
 - Compile time polymorphism
 - Known as static polymorphism
 - Achieved through method overloading
 - Runtime Polymorphism
 - Known as dynamic polymorphism.
 - A method call to the overridden method is resolved at runtime.
 - It is achieved through method overriding.

Method Overloading(Compile time polymorphism):

- Here, multiple methods are created with the same name in the class, but they accept different types or numbers of parameters.

- The appropriate method to be called is determined by the number and types of arguments used during the method call at compile time.
- The method call is resolved before the program runs. I.e. during compile time.

Program 1:

```
class Calculation {
    public int add(int a, int b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }

    public String add(String a, String b) {
        return a + b;
    }
}

public class MainOverload {
    public static void main(String[] args) {
        Calculation c1 = new Calculation();

        int sum1 = c1.add(5, 10);
        double sum2 = c1.add(2.5, 3.7);
        String s1 = c1.add("Hello", " BIT");

        System.out.println("Sum of integers: " + sum1);
        System.out.println("Sum of doubles: " + sum2);
        System.out.println("Concatenated strings: " + s1);
    }
}
```

Output:

```
Sum of integers: 15
Sum of doubles: 6.2
Concatenated strings: Hello BIT
```

Method Overriding(Runtime polymorphism):

- When a method of subclass has same name and type as the method of super class , then the method in superclass is overridden.
- When the overridden method is called, it refers to the sub class method and the super class method is hidden. i.e. the method from sub class overrides the method of superclass.

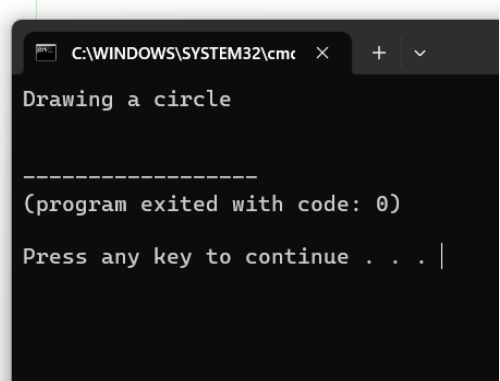
•

Program 2:

```
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

public static void main(String[] args)
{
    Shape c=new Circle();
    c.draw();
}
}
```



```
C:\WINDOWS\SYSTEM32\cmd  X  +  v
Drawing a circle
-----
(program exited with code: 0)
Press any key to continue . . . |
```

Program 3:

```
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {

    public void draw() {
        System.out.println("Drawing a circle");
    }
}

class Rectangle extends Shape {

    public void draw() {
        super.draw();
        System.out.println("Drawing a rectangle");
    }
}

public class MainOverloading {
    public static void main(String[] args) {
        Shape s1 = new Circle();
        Shape s2 = new Rectangle();

        s1.draw();
        s2.draw();
    }
}
```

Output:

```
Drawing a circle  
Drawing a shape  
Drawing a rectangle
```

Program 4:

```
class Student {  
    String name;  
    int age;  
    public Student(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    public void printStudentDetails() {  
        System.out.println("Name: " + name);  
        System.out.println("Age: " + age);  
    }  
}  
class BITStudent extends Student {  
    int score;  
  
    public BITStudent(String name, int age, int score) {  
        super(name, age);  
        this.score = score;  
    }  
    public void printStudentDetails() {  
        super.printStudentDetails();  
        System.out.println("Score: " + score);  
        System.out.println("Student type: BIT Student");  
    }  
}
```

```
class BBAStudent extends Student {
    private int marks;

    public BBAStudent(String name, int age, int marks) {
        super(name, age);
        this.marks = marks;
    }
    public void printStudentDetails() {
        super.printStudentDetails();
        System.out.println("Marks: " + marks);
        System.out.println("Student type: BBA Student");
    }
}

public class MainOverride {
    public static void main(String[] args) {
        Student student1 = new BITStudent("Ram", 20, 95);
        Student student2 = new BBAStudent("Shyam", 19, 80);
        System.out.println("Details of BIT Student:");
        student1.printStudentDetails();
        System.out.println("\nDetails of BBA Student:");
        student2.printStudentDetails();
    }
}
```

Output:

Details of BIT Student:

Name: Ram

Age: 20

Score: 95

Student type: BIT Student

Details of BBA Student:

Name: Shyam

Age: 19

Marks: 80

Student type: BBA Student