Module 3: Polymorphism

Prof. Pramod Bide

Concept of Polymorphism

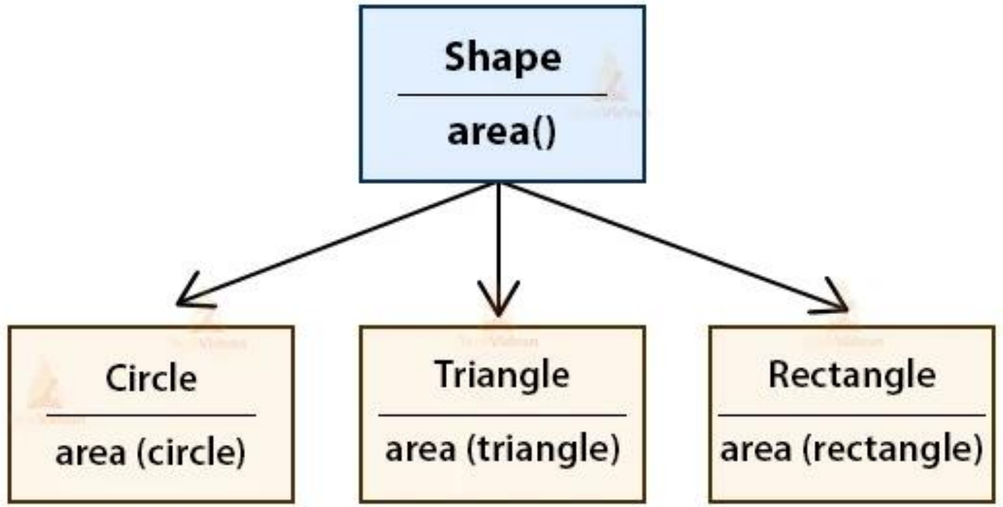
- The word polymorphism means having many forms.
- In simple words, we can define Java Polymorphism as the ability of a message to be displayed in more than one form.

Real-life Illustration of Polymorphism in Java: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So the same person possesses different behaviors in different situations. This is called polymorphism.

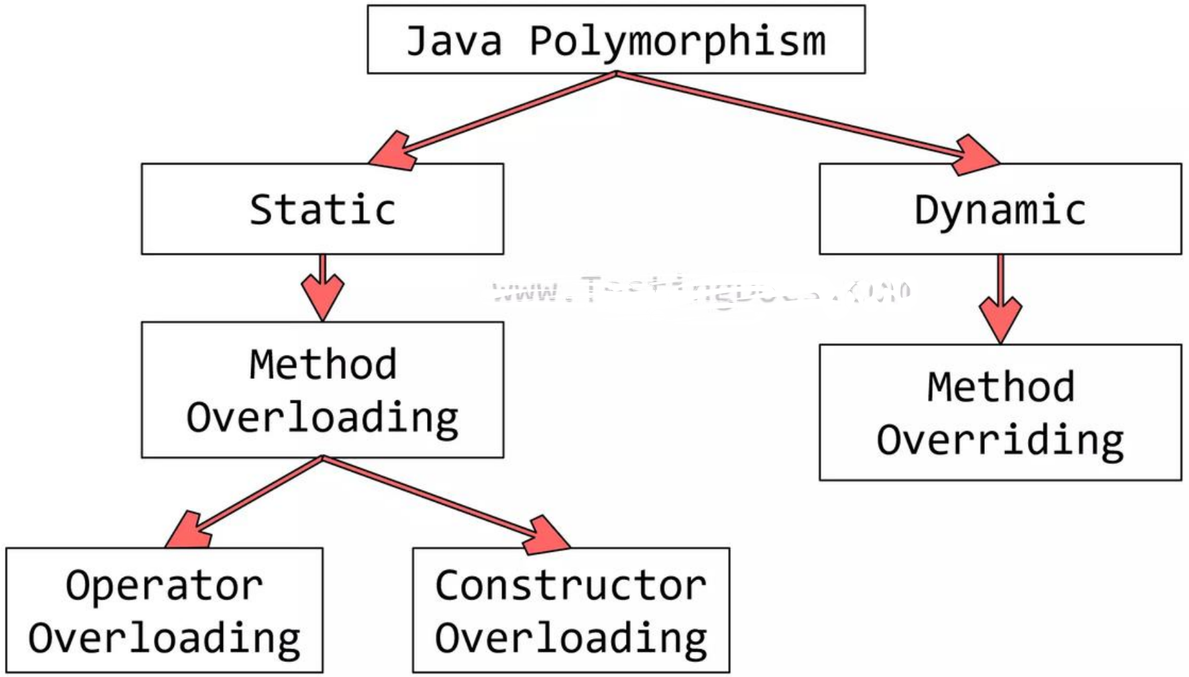
Concept of Polymorphism

- Polymorphism is considered one of the important features of Object-Oriented Programming.
- 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.

Example of Polymorphism



Types of Polymorphism



In Java Polymorphism is mainly divided into two types:

- Static Polymorphism (Compile-time Polymorphism)
- Dynamic Polymorphism (Runtime Polymorphism)

Static Polymorphism

- Static polymorphism happens when the compiler chooses the right method or function to call at compile-time based on the quantity, nature, and order of inputs.
- It is also known as compile-time polymorphism.
- This type of polymorphism is achieved by method overloading or constructor overloading.

Static Polymorphism-Method overloading

- When there are multiple functions with the same name but different parameters then these functions are said to be overloaded.
- Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

Static Polymorphism-Method overloading

Calculator.java

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
    public double add(double a, double b) {
        return a + b;
    }
    public int add(int a, int b, int c) {
        return a + b + c;
    }
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        System.out.println(calculator.add(5, 10));
        System.out.println(calculator.add(2.5, 3.7));
        System.out.println(calculator.add(1, 2, 3));
    }
}
```

input

```
15
6.2
6


...Program finished with exit code 0
```

- It allows the Calculator class to have multiple methods with the same name (add) but different parameter lists (int, double, int int int). This enables the compiler to determine which method to call based on the arguments passed during compile time.
- In this code, the add method is overloaded three times with different parameter types, demonstrating static polymorphism. Depending on the arguments passed, the appropriate add method (int, double, or int int int) will be invoked at compile time, providing flexibility and convenience in using the Calculator class.

Static Polymorphism-Constructor overloading

- Constructor overloading is based on the concept of polymorphism, where different constructors with the same name but different parameter lists can be called based on the arguments passed during object creation.
- This enables the class to accommodate different scenarios and simplify object instantiation.

Static Polymorphism-Constructor overloading

```java
class Rectangle {
    public int width;
    public int height;

    // Constructor that takes width and height parameters
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    // Constructor that takes a width parameter and sets the height to the same value
    public Rectangle(int width) {
        this(width, width);
    }
}


class PrepBytes{
    public static void main(String args[]){
        Rectangle rec1 = new Rectangle(5);
        System.out.println("Height of Rectangle is " + rec1.height);
        System.out.println("Width of Rectangle is " + rec1.width);
    }
}
```

input

```
Height of Rectangle is 5
Width of Rectangle is 5


...Program finished with exit code 0
Press ENTER to exit console.
```

- In this example, we have created a Rectangle class with two constructors.
- The first constructor takes width and height parameters and initializes the corresponding fields to the values of the parameters.
- The second constructor takes a width parameter and calls the first constructor using the "this" keyword, passing in the width parameter for both the width and height parameters.
- This allows the second constructor to reuse the logic of the first constructor while providing a simpler interface for creating squares.

Dynamic Polymorphism

- Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time.
- It is also known as runtime polymorphism or dynamic method dispatch. We can achieve dynamic polymorphism by using the method overriding.

Dynamic Polymorphism

Properties of Dynamic Polymorphism:

- It decides which method is to execute at runtime.
- It can be achieved through dynamic binding.
- It happens between different classes.
- It is required where a subclass object is assigned to a super-class object for dynamic polymorphism.
- Inheritance involved in dynamic polymorphism.

Dynamic Polymorphism-Method Overriding

- It provides a specific implementation to a method that is already present in the parent class.
- It is used to achieve run-time polymorphism. Remember that, it is not possible to override the static method.
- Hence, we cannot override the main() method also because it is a static method.

Dynamic Polymorphism-Method Overriding

Rules for Method Overriding

- The name of the method must be the same as the name of the parent class method.
- The number of parameters and the types of parameters must be the same as in the parent class.
- There must exist an IS-A relationship (inheritance).

We call an overridden method through a reference of the parent class. The type of object decides which method is to be executed and it is decided by the JVM at runtime.

Dynamic Polymorphism-Method Overriding

Demo.java

```java
//parent class
class Sample
{
//method of the parent class
public void display()
{
System.out.println("Overridden Method");
}
}
//derived or child class
public class Demo extends Sample
{
//method of child class
public void display()
{
System.out.println("Overriding Method");
}
public static void main(String args[])
{
//assigning a child class object to parent class reference
Sample obj = new Demo();
//invoking display() method
obj.display();
} 
}
```

input

```
Overriding Method


...Program finished with exit code 0
Press ENTER to exit console.
```

- In the following example, we have created two classes named Sample and Demo. The Sample class is a parent class and the Demo class is a child or derived class. The child class is overriding the dispaly() method of the parent class.
- We have assigned the child class object to the parent class reference. So, in order to determine which method would be called, the type of the object would be decided by the JVM at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

Data Conversion

- Data conversion in polymorphism refers to the automatic conversion of data types by the compiler or runtime environment to match the parameters of overloaded methods during method invocation.
- This allows for flexibility and convenience when calling methods with different parameter types.

What is Automatic Type Promotion?

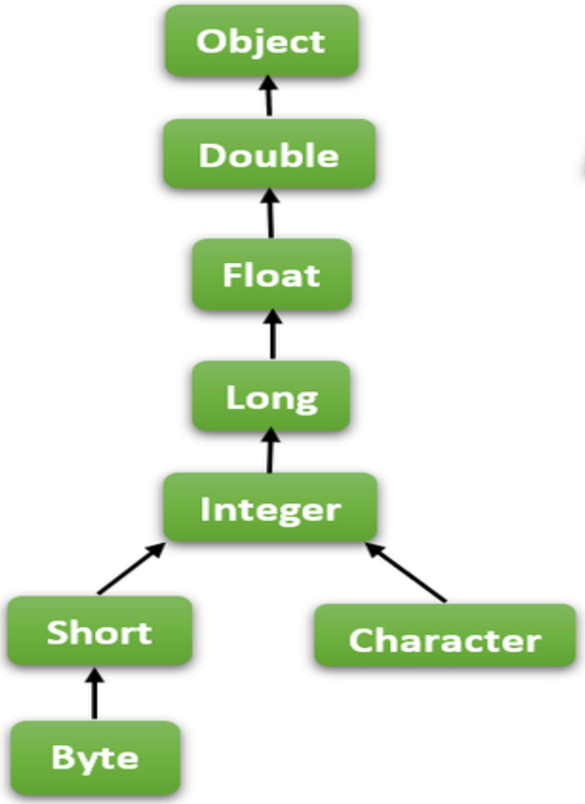
The name Type Promotion specifies that a small size datatype can be promoted to a large size datatype. i.e., an Integer data type can be promoted to long, float, double, etc. This Automatic Type Promotion is done when any method which accepts a higher size data type argument is called with the smaller data type.

This is important to remember is Automatic Type Promotion is only possible from small size datatype to higher size datatype but not from higher size to smaller size. i.e., integer to character is not possible.

```
class ATP {

    // A method that accept double as parameter
    public static void method(double d)
    {
        System.out.println(
            "Automatic Type Promoted to Double-" + d);
    }

    public static void main(String[] args)
    {
        // method call with int as parameter
        method(2);
    }
}
```



Small Size Datatype to High Size Datatype conversion is possible

Pros & Cons of Polymorphism

Advantages of Polymorphism in Java

1. Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
2. Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained.
3. Supports dynamic binding, enabling the correct method to be called at runtime, based on the actual class of the object.
4. Enables objects to be treated as a single type, making it easier to write generic code that can handle objects of different types.

Disadvantages of Polymorphism in Java

1. Can make it more difficult to understand the behavior of an object, especially if the code is complex.
2. This may lead to performance issues, as polymorphic behavior may require additional computations at runtime.