HNDIT3012 - OBJECT ORIENTED PROGRAMMING

LECTURE 07 – POLYMORPHISM

WHAT IS POLYMORPHISM?

- The derivation of the word Polymorphism is from two different Greek words- poly and morphs.
- “Poly” means numerous, and “Morphs” means forms.
- So, polymorphism means innumerable forms.
- Polymorphism, therefore, is one of the most significant features of Object-Oriented Programming.

REAL-LIFE EXAMPLES OF POLYMORPHISM

- A woman can be a mother, a daughter, a sister, and a friend, all at the same time, i.e. she performs other behaviors in different situations.

- The human body has different organs. Every organ has a different function to perform; the heart is responsible for blood flow, the lungs for breathing, the brain for cognitive activity, and the kidneys for excretion. So we have a standard method function that performs differently depending upon the organ of the body.

POLYMORPHISM IN JAVA EXAMPLE

- A superclass named “Shapes” has a method called “area()”. Subclasses of “Shapes” can be “Triangle”, “circle”, “Rectangle”, etc.
- Each subclass has its way of calculating area.
- Using Inheritance and Polymorphism means, the subclasses can use the “area()” method to find the area’s formula for that shape.

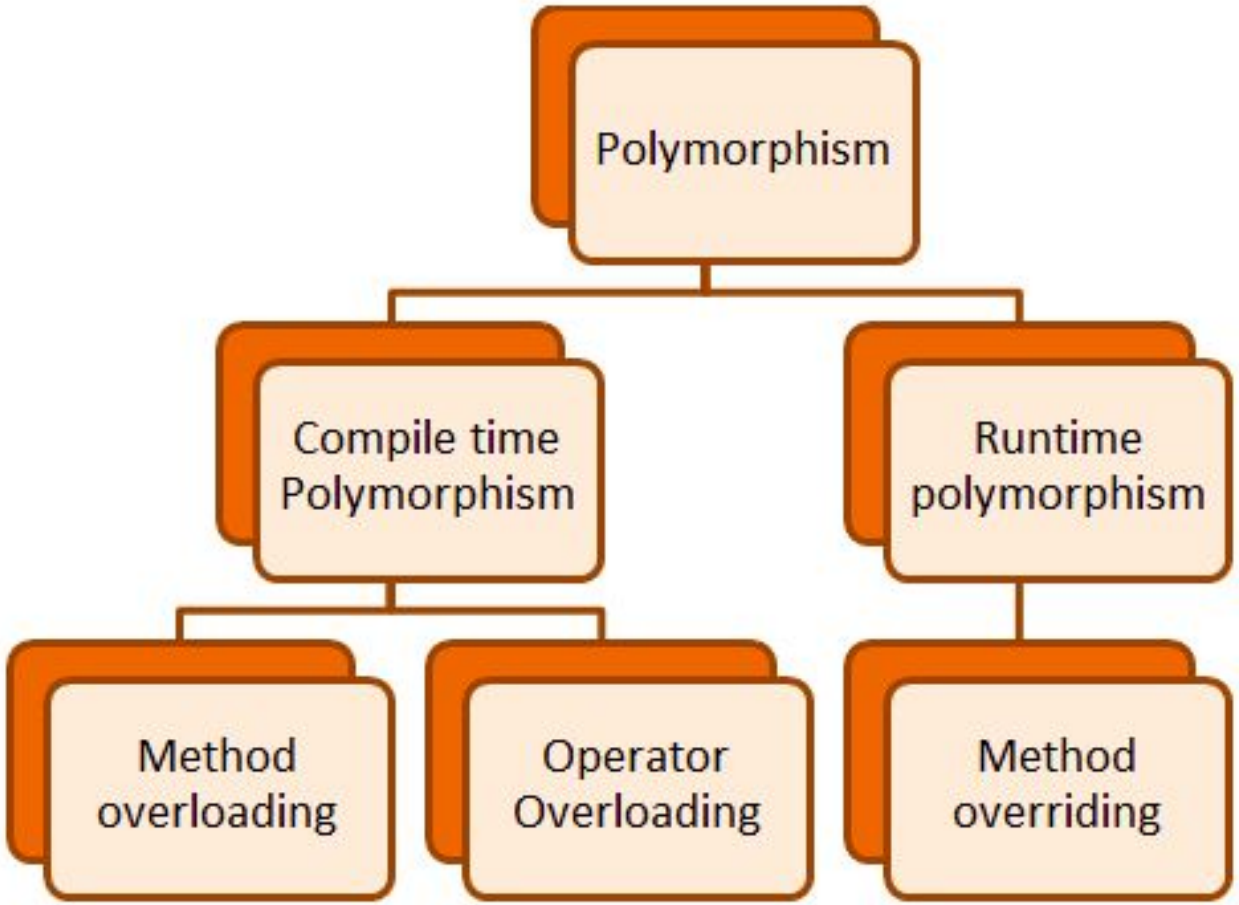
EXAMPLE

```
class Shapes {
    public void area() {
        System.out.println("The formula for area of "); }
        }
        class Triangle extends Shapes {
            public void area() {
                System.out.println("Triangle is ½ * base * height ");
            }
        }
        class Circle extends Shapes {
            public void area() {
                System.out.println("Circle is 3.14 * radius * radius ");
            }
        }
```

EXAMPLE

```
class Main {
    public static void main(String[] args) {
        Shapes myShape = new Shapes(); // Create a Shapes object Shapes
        myTriangle = new Triangle(); // Create a Triangle object Shapes
        myCircle = new Circle(); // Create a Circle object
        myShape.area();
        myTriangle.area();
        myShape.area();
        myCircle.area();
    }
}
```

TYPES OF POLYMORPHISM



TYPES OF POLYMORPHISM

- Compile time /static polymorphism
  - The compiler knows which method has been called.
  - It can be achieved through method **overloading**.

- Runtime /Dynamic polymorphism
  - The compiler doesn't know which method has been called at compile-time.
  - JVM decides which method is called at runtime.
  - It can be achieved through method **overriding**.

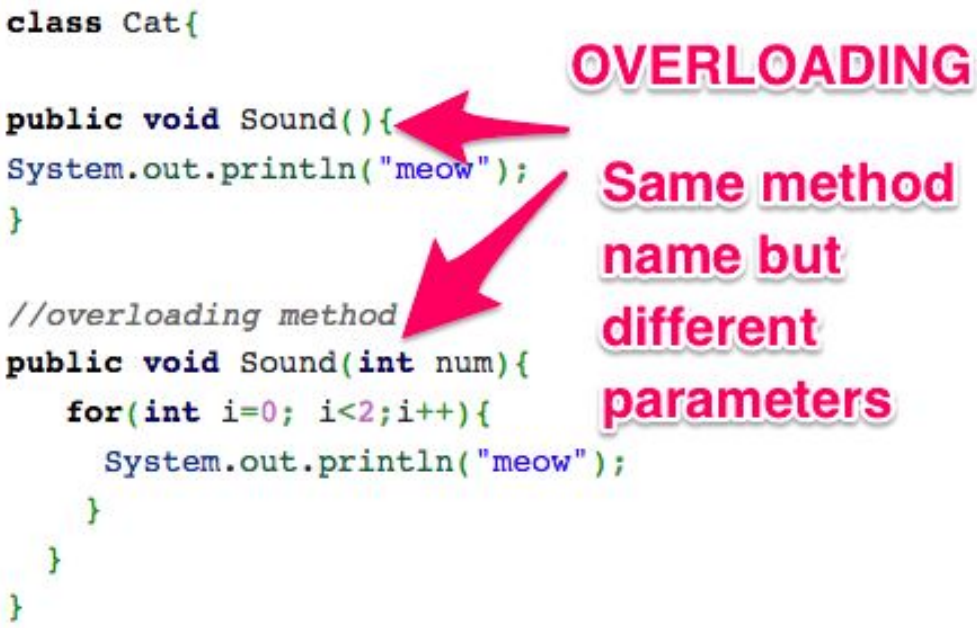
TYPES OF POLYMORPHISM

- Polymorphism in Java via two different methods:
  - Method Overloading
  - Method Overriding

METHOD OVERLOADING IN JAVA

- **Method overloading** is the process that can create **multiple methods** of the **same name** in the **same class**, and all the methods work in **different ways**.

- Method Overloading is when a class has **multiple methods** with the **same name**, but the **number, types**, and **order of parameters** and the **return type** of the methods are **different.**



EXAMPLE

With Method Overloading

```
int add(int x, int y)
{
    return(x+y);
}
int add(int x, int y,int z)
{
    return(x+y+z);
}
int add(int w, int x,int y, int z)
{
    return(w+x+y+z);
}
```

JAVA EXAMPLE

```
class Shapes {
        public void area() {
            System.out.println("Find area ");
        }
        public void area(int r) {
            System.out.println("Circle area = "+3.14*r*r);
        }
        public void area(double b, double h) {
            System.out.println("Triangle area="+0.5*b*h);
        }
        public void area(int l, int b) {
            System.out.println("Rectangle area="+l*b);
        }
            }
```

JAVA EXAMPLE

```
class Main {
    public static void main(String[] args) {
        Shapes myShape = new Shapes(); // Create a Shapes object
        myShape.area();
        myShape.area(5);
        myShape.area(6.0,1.2);
        myShape.area(6,2);
    }
}
```

QUESTION

- Write down two methods to add two integers and three integer numbers in java using polymorphism.

```
package staticPolymorphism;
    public class Addition {
        void sum(int a, int b) {
            int c = a+b;
            System.out.println(" Addition of two numbers :" +c);
        }
        void sum(int x, int y, int z) {
            int c = x+y+z;
            System.out.println(" Addition of three numbers :" +c);
        }
    public static void main(String[] args) {
        Addition obj = new Addition();
        obj.sum ( 30,90);
        obj.sum(45, 80, 22);
    }
    }
```

METHOD OVERRIDING IN JAVA

- Method overriding is the process when the subclass or a **child class** has the **same method, parameters, and return type** as declared in the **parent class**.

- The method to be called is determined based on the object which is being referred to by the reference variable.

- This is also known as **Upcasting**.

JAVA EXAMPLE

```
class Vehicle{
	//defining a method
	void run(){
		System.out.println("Vehicle is moving");
	}
}
//Creating a child class
class Car2 extends Vehicle{
	//defining the same method as in the parent class
	void run(){
		System.out.println("car is running safely");
	}
	public static void main(String args[]){
		Car2 obj = new Car2();//creating object
		obj.run();//calling method
	}
}
```

EXAMPLE 02

```
//parent class Animal
class Animal{
    // creating place method
       void place(){
            System.out.println("Animals live on earth.");
       }
}
// Dog class extends Animal class
class Dog extends Animal{
    // overriding place method
       void place(){
            System.out.println("Dog lives in kennel.");
       }
}
// Horse class extends Animal class
class Horse extends Animal{
    // overriding place method
       void place(){
            System.out.println("Horse lives in stable.");
       }
}
```

EXAMPLE 02

```
// Rabbit class extends Animal class
class Rabbit extends Animal{
    // overriding place method
      void place(){
            System.out.println("Rabbit lives in burrow.");
      }
}
class Polymorphism{
    public static void main(String[] args) {
       // creating object of Animal class
      Animal A = new Animal();
      A.place();
      A = new Dog();
      A.place();
      A = new Horse();
      A.place();
      A = new Rabbit();
      A.place();
    }
}
```

STATIC VS. DYNAMIC POLYMORPHISM

Static Polymorphism	Dynamic Polymorphism
It is achieved through method overloading.	It is achieved through method overriding.
It is called compile-time polymorphism.	It is called runtime polymorphism.
Known as static binding or early binding.	Known as dynamic binding or late binding.
It has a high execution speed.	It has a low execution speed.

CHARACTERISTICS OF POLYMORPHISM

- polymorphism has other characteristics as follows.
  - Coercion
  - Internal Operator Overloading
  - Polymorphic Variables or Parameters
  - Subtype polymorphism

COERCION

- The implicit conversion of one data type into another without changing its context is known as coercion.
- Java has inbuilt functionality called coercion to avoid such errors, where a smaller data type is automatically typecasted into a more significant data type according to need.
- Example
  - consider a variable whose data type is an integer and another variable whose data type is double. If we add these two numbers, we will get a type error.
  - In this case, the integer value will be typecasted to double value, and then addition takes place. Hence type error is avoided.

EXAMPLE - COERCION

```
class Polymorphism{
    public static void main(String[] args) {
        int num = 165;
        String str = "Hello";
            // concatenating str and num
        String ans =  str+num;
        System.out.println(ans);
    }
}
```

Output

Hello165

INTERNAL OPERATOR OVERLOADING

- in Java, where an operator is used in more than one way.
- Example, the '+' symbol is used to add two numbers or used to concatenate two strings.

```
class Polymorphism{
    public static void main(String[] args) {
        // adding numbers
        int num1 = 741, num2 = 852;
        String str1 = "Hello", str2 = "World!";
            // concatenating two strings
        int sum = num1+num2;
        String final_str = str1 + str2;

        System.out.println("Sum = "+sum);
        System.out.println("Final String = "+final_str);
    }
}
```

Output:
Sum = 1593
Final String = HelloWorld!

POLYMORPHIC VARIABLES OR PARAMETERS

- Variables having different values under different circumstances is called polymorphic variable.
- It can be said that every object or instance variable in Java is a polymorphic variable because every object or instance variable has an IS-A relationship with its own classes and sub-classes.
- Variables holding different values at the execution time are known as polymorphic variables.

EXAMPLE

```
class Country{
    // creating info method
        void info(){
                System.out.println("I am from Sri Lanka.");
        }
}
class Gampaha extends Country{
    // overriding info method
        void info(){
                System.out.println("I am from Gampaha, District of Sri
Lanka.");
        }
}
class Polymorphism{
    public static void main(String[] args) {
                        Country ob;
                ob = new Country();
                ob.info();
                ob = new Gampaha();
                ob.info();
    }
}
```

Output

I am from Sri Lanka.

I am from Gamapaha, District of Sri Lanka.

EXAMPLE 02

```
class Parent{
	String display(int num){
		String temp = " "+num;
		return temp;
	}
}
class Child extends Parent{
	String data = "Hello World ";
	String display(int num){
		int data = num;
		this.data = this.data+data;
		return this.data;
	}
}
class Polymorphism{
    public static void main(String[] args) {
			Parent obj;
		obj = new Child();
		System.out.println(obj.display(404));
    }
}
```

Output
Hello World
404

SUBTYPE POLYMORPHISM

- The ability to use the subclass instead of the superclass is called subtype polymorphism.
- In other words, subtype polymorphism is about **upcasting** and **late binding**.
- Example
  - Let us assume tiger, lion, and elephant are derived classes of parent class animal.
  - Traditionally we create an object of tiger class and store the information. Similarly, we do this for all the animals.
  - But in subtype polymorphism, we make an array of animal class, and then we upcast objects of every tiger, lion, and elephant to animal class. Store the upcasted objects in an array of animal class.

EXAMPLE

```
// parent class Shape
class Shape{
    // creating area method
      void area(){
            System.out.println("Area of various shapes are
calculated here.");
      }
}
// Square class extends Shape class
class Square extends Shape{
      int a;
    // parametric constructor
      Square(int side){
            this.a = side;
      }
      void area(){
            System.out.println("Side of square is : "+a);
            System.out.println("Area of square is : "+(a*a)+"\n");
      }
}
```

SUB CLASSES

```
// Circle class extends Shape class
class Circle extends Shape{
	int r;
    // parametric constructor
	Circle(int a){
		this.r = a;
	}
	void area(){
		System.out.println("Radius of circle is : "+r);
		System.out.println("Area of circle is : "+(3.14*r*r)+"\n");
	}
}
class Rectangle extends Shape{
	int l, b;
    // parametric constructor
	Rectangle(int w, int h){
		this.l = w;
		this.b = h;
	}
	void area(){
		System.out.println("Sides of rectangle are : "+l+", "+b);
		System.out.println("Area of rectangle is : "+(2*(l + b))+"\n");
	}
}
```

MAIN CLASS

```
class Polymorphism{
    public static void main(String[] args) {
        // array of Shape class
            Shape [] arr = {
                new Square(10), new Circle(15), new Rectangle(10, 15)
            };
            for(int i=0; i<arr.length; i++){
                arr[i].area();
            }
    }
}
```