Day 4

Java Main programming

Drawback of array derived type

* Size of the array is fixed

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.

**What is Polymorphism in Java?**

**Polymorphism in Java** is the task that performs a single action in different ways.

So, languages that do not support polymorphism are not ‘Object-Oriented Languages’, but, ‘Object-Based Languages’. Ada, for instance, is one such language. Since Java supports polymorphism, it is an [Object-Oriented Language](https://www.mygreatlearning.com/academy/learn-for-free/courses/oops-in-java/?gl_blog_26164).

Polymorphism occurs when there is inheritance, i.e. there are many classes that are related to each other.

[Inheritance](https://www.mygreatlearning.com/academy/learn-for-free/courses/inheritance-in-java/?gl_blog_26164) is a powerful feature in Java. Inheritance lets one class acquire the properties and attributes of another class. Polymorphism in Java allows us to use these inherited properties to perform different tasks. Thus, allowing us to achieve the same action in many different ways.

**Real-Life Examples of Polymorphism**

An individual can have different relationships with different people. A woman can be a mother, a daughter, a sister, 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, lungs for breathing, brain for cognitive activity, and 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 “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 code:

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 ");

  }

}

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****

You can perform Polymorphism in Java via two different methods:

1. Method Overloading
2. Method Overriding

### **What is Method Overloading in Java?**

[**Method overloading**](https://www.mygreatlearning.com/blog/method-overloading-in-java/)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 occurs when there is more than one method of the same name in the class.

Example for method overloading

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);

  }

}

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);

  }

}

### **What is Method Overriding in Java?**

Method overriding is the process when the subclass or a child class has the same method as declared in the parent class.

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 } }

### **What is Compile-Time Polymorphism in Java?**

**Compile-Time Polymorphism**in Java is also known as **Static Polymorphism.** Furthermore, the call to the method is resolved at compile-time. Compile-Time polymorphism is achieved through **Method Overloading**. This type of polymorphism can also be achieved through **Operator Overloading**. However, Java does not support Operator Overloading.

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. Java allows the user freedom to use the same name for various functions as long as it can distinguish between them by the type and number of parameters.

Example:

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 a, int b, int e)

{

int c = a+b+e;

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);

}

}

### **What is Runtime Polymorphism in Java?**

**Runtime polymorphism** in Java is also popularly known as **Dynamic Binding or Dynamic Method Dispatch.** In this process, the call to an overridden method is resolved dynamically at runtime rather than at compile-time. You can achieve Runtime polymorphism via **Method Overriding**.

Method Overriding is done when a child or a subclass has a method with the same name, parameters, and return type as the parent or the superclass; then that function overrides the function in the superclass. In simpler terms, if the subclass provides its definition to a method already present in the superclass; then that function in the base class is said to be overridden.

Also, it should be noted that runtime polymorphism can only be achieved through functions and not data members.

Overriding is done by using a reference variable of the superclass. 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**.

Upcasting takes place when the Parent class’s reference variable refers to the object of the child class. For example:

In this example, we are creating one superclass Animal and three subclasses, Herbivores, Carnivores, and Omnivores. Subclasses extend the superclass and override its eat() method. We will call the eat() method by the reference variable of Parent class, i.e. Animal class. As it refers to the base class object and the base class method overrides the superclass method; the base class method is invoked at runtime. As Java Virtual Machine or the JVM and not the compiler determines method invocation, it is, therefore, runtime polymorphism.

class Animal{

  void eat(){

System.out.println("Animals Eat");

}

}

class herbivores extends Animal{

  void eat(){

System.out.println("Herbivores Eat Plants");

}

  }

class omnivores extends Animal{

  void eat(){

System.out.println("Omnivores Eat Plants and meat");

}

  }

class carnivores extends Animal{

  void eat(){

System.out.println("Carnivores Eat meat");

}

  }

class main{

  public static void main(String args[]){

    Animal A = new Animal();

    Animal h = new herbivores(); //upcasting

    Animal o = new omnivores(); //upcasting

    Animal c = new carnivores(); //upcasting

    A.eat();

    h.eat();

    o.eat();

    c.eat();

  }

}