Polymorphism

Table Of Content

Introduction To Polymorphism

- Polymorphism is also called as **parent reference to child objects**
- POLY means **many** MORPHS means **forms**. An object with many forms is called as POLYMORPHISM.
- It is also known as **creating a reference to parent class and through that we access the child objects**
- The word polymorphism means **having many forms**. In simple words, we can define polymorphism as the **ability of a message to be displayed in more than one form.**
- POLYMORPHISM is used to achieve the **'code reduction'** and **flexibility** of the program.

Steps To Achieve Polymorphism

1. **INHERITANCE** must be present.
2. classes must be **RELATED**
3. **METHOD OVERRIDING** should be present

4. use the concept of **'PARENT REF TO CHILD OBJECT'**
5. **POLYMORPHISM**

▼ **Polymorphism Example**

```
package Object_Oriented_Concepts.Polymorphism;

class vehicle
{
    void brand()//form1
    {
        System.out.println("My favourite cycle, bike, and car brand");
    }
}

class cycle extends vehicle
{
    void brand()//form2
    {
        System.out.println("My favourite cycle brand is 'POLYGON & HERO' ");
    }
}

class bike extends vehicle
{
    void brand()//form3
    {
        System.out.println("My favourite bike brand is 'HERO, HONDA, & ATHER' ");
    }
}

class car extends vehicle
{
    void brand()//form4
    {
        System.out.println("My favourite car brand is 'TATA, TESLA, & HYUNDAI' ");
    }
}

public class _1_polymorphismExample
{
    public static void main(String[] args)
    {
        //---------reference
        vehicle collections  = new vehicle();//creating reference to parent class

        //parent reference to child objects(new cycle, new car, new bike)
        collections = new cycle();
        collections.brand();
        System.out.println();
        collections = new bike();
        collections.brand();
        System.out.println();
        collections = new car();
        collections.brand();
```

```
    }
}
//output
My favourite cycle brand is 'POLYGON & HERO'

My favourite bike brand is 'HERO, HONDA, & ATHER'

My favourite car brand is 'TATA, TESLA, & HYUNDAI'
```

NOTE

▼ **Polymorphism Example**

```
package _1Java_Codes_From_Basics._16polymorphismInJava;

class vehicle
{
    void brand()
    {
        System.out.println("Best Collections"); //form 1
    }
}

class cycle extends vehicle
{
    void brand()
    {
        System.out.println("My favourite cycle is 'Polygon' and 'Hero'");//form 2
    }
}

class bike extends vehicle
{
    void brand()
    {
        System.out.println("My favourite bike is 'Bajaj Pulzar' and 'Yamaha MT-1
5'");//form 4
    }
}

class car extends vehicle
{
    void brand()
    {
        System.out.println("My favourite car is 'Tata' and 'Hyundai'");//form 3
    }
}

public class _1polymorphismEx1
{
    public static void main(String[] args)
    {//---------reference
        vehicle collections  = new vehicle();//creating reference to parent class
```

```
        //parent reference to child objects(new cycle, new car, new bike)
        collections = new cycle();
        collections.brand();
        collections = new bike();
        collections.brand();
        collections = new car();
        collections.brand();
    }
}
//output
My favourite cycle is 'Polygon' and 'Hero'
My favourite bike is 'Bajaj Pulzar' and 'Yamaha MT-15'
My favourite car is 'Tata' and 'Hyundai'
```

- In the above program we have achieved polymorphism by using the concept of 'parent ref to child object' but by using this technique we **cannot access the specialized methods.**
- To access the specialized method we need to use a technique called '**DOWNCASTING**'

UPCASTING

- Upcasting refers **to the creation of child object and assigning the parent reference** to it.
  advantage of upcasting is achieving the polymorphism.
- ***UPCASTING SYNTAX:***

  parent_class_name reference_Variable_of_ParentClass = new child_class_name();

  reference_Variable_of_ParentClass.child_class_overriding_method_name

DOWNCASTING

- it means **temporary converting the parent ref to child object to access the specialized method**
- ***DOWNCASTING SYNTAX:*** *((child_ClassName)(reference_Variable_of ParentClass)).specializedMethod()*

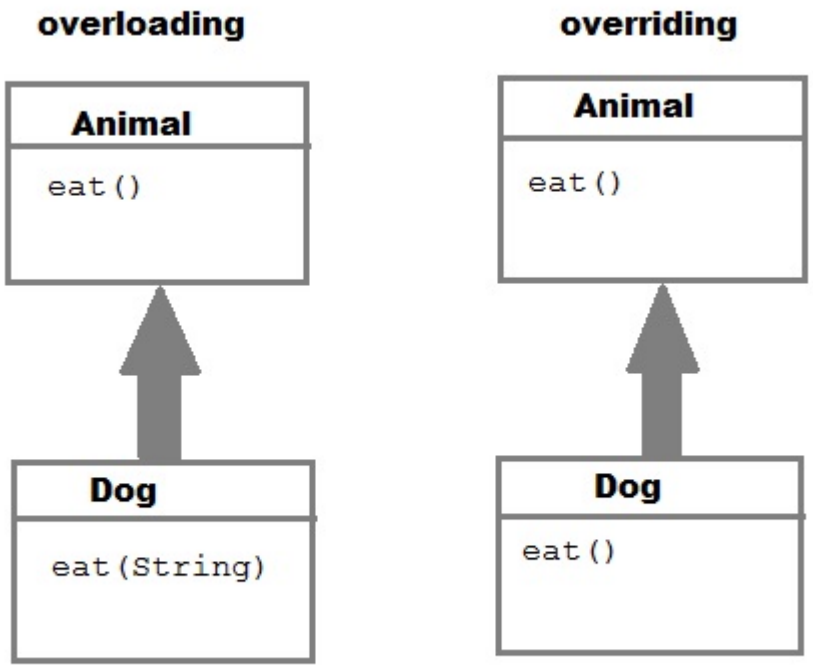
```
brand cars = new brand();

((Tata)(cars)).originBrand();

//Tata: child class
//cars: reference variable of parent class
//originBrand(): specialized method
```

▼ POLYMORPHISM Upcasting & Downcasting Example

```
package Object_Oriented_Concepts.Polymorphism;

/**
 * @Note
 * Downcasting - technique to access specialized method
 * Downcasting syntax -  ((child_class_name)(parent_reference_variable)).specializ
ed_method_name();
 * Downcasting example - ((tata)(vb)).proud();
 *
 * Upcasting - technique to access overridden method
 * Upcasting syntax  -  parent_class_name reference_variable = new child_class();
 *--------------------| reference_variable.overriding_method();
 * Upcasting example - vb = new tesla();
 *--------------------|vb.brand();
 */

class vehicle_Brands
{
    void brand()
    {
        System.out.println("Different brand belongs to different countries");
    }
}

class tata extends vehicle_Brands
{
    void brand()
    {
        System.out.println("TATA brand belong to 'INDIA' ");
    }

    void proud()//specialized method
    {
        System.out.println("'TATA IS PROUD INDIAN BRAND'");
    }
}

class mahindra extends vehicle_Brands
{
    void brand()
    {
        System.out.println("MAHINDRA brand belong to 'INDIA' ");
    }

    void proud()//specialized method
    {
        System.out.println("'MAHINDRA IS PROUD INDIAN BRAND'");
    }
}


class tesla extends vehicle_Brands
{
    void brand()
```

```
    {
        System.out.println("TESLA brand belong to 'USA' ");
    }
}

class hyundai extends  vehicle_Brands
{
    void brand()
    {
        System.out.println("HYUNDAI brand belong to 'SOUTH KOREA' ");
    }
}


public class _2_upcastingAndDowncasting
{
    public static void main(String[] args)
    {
        vehicle_Brands vb = new vehicle_Brands();

        vb = new tata();
        vb.brand();
        //DOWNCASTING SYNTAX: ((child_ClassName)(referenceVariable_of ParentClas
s)).specializedMethod()
        ((tata)(vb)).proud(); //DOWNCASTING

        vb = new mahindra();
        vb.brand();
        ((mahindra)(vb)).proud();


        vb = new tesla();
        vb.brand();//UPCASTING
        vb = new hyundai();
        vb.brand();
    }
}
//output
TATA brand belong to 'INDIA'
'TATA IS PROUD INDIAN BRAND'
MAHINDRA brand belong to 'INDIA'
'MAHINDRA IS PROUD INDIAN BRAND'
TESLA brand belong to 'USA'
HYUNDAI brand belong to 'SOUTH KOREA'
```

Method Overloading And Method Overriding In Polymorphism



Method Overloading

- --> METHOD OVERLOADING --> VIRTUAL POLYMORPHISM --> STATIC POLYMORPHISM --> EARLY BINDING.

▼ **Virtual Polymorphism Using Method Overloading**

```
package Object_Oriented_Concepts.Polymorphism;

// VIRTUAL POLYMORPHISM USING THE METHOD OVERLOADING.
class plane
{
    String name;
    void takeOff()
    {
        System.out.println("plane is taking off");
    }
    void fly()
    {
        System.out.println("plane is flying");
    }
    void land()
    {
        System.out.println("plane is landing");
    }
}
class cargoPlane extends plane
{
```

```
    {
        super.name = "Cargo Plane";
    }
    void fly()
    {
        System.out.println(name+" plane is flying");
    }
    void carryCargo()
    {
        System.out.println(name+" is used to carry goods");
    }
}
class passengerPlane extends plane
{
    {
        super.name = "Passenger Plane";
    }
    void fly()
    {
        System.out.println(name+" plane is flying");
    }
    void carryPassenger()
    {
        System.out.println(name+" is used to carry passengers");
    }
}
class fighterJet extends plane
{
    {
        super.name = "Fighter Plane";
    }
    void fly()
    {
        System.out.println(name+" is flying");
    }
    void carryArms()
    {
        System.out.println(name+" is used to carry arms");
    }
}
class airport
{
    //method overloading here
    void allow(cargoPlane p)
    {
        p.takeOff();
        p.fly();
        p.carryCargo();
        p.land();
    }

    void allow(passengerPlane p)
    {
        p.takeOff();
        p.fly();
        p.carryPassenger();
        p.land();
    }
```

```
    void allow(fighterJet p)
    {
        p.takeOff();
        p.fly();
        p.carryArms();
        p.land();
    }
}
public class _3_methodOverloading
{
    public static void main(String[] args)
    {
        cargoPlane cp = new cargoPlane();
        passengerPlane pp = new passengerPlane();
        fighterJet fj = new fighterJet();
        airport a = new airport();

        System.out.println("--------------------------");
        a.allow(cp);
        System.out.println("--------------------------");
        a.allow(pp);
        System.out.println("--------------------------");
        a.allow(fj);
    }
}
//output
--------------------------
plane is taking off
Cargo Plane plane is flying
Cargo Plane is used to carry goods
plane is landing
--------------------------
plane is taking off
Passenger Plane plane is flying
Passenger Plane is used to carry passengers
plane is landing
--------------------------
plane is taking off
Fighter Plane is flying
Fighter Plane is used to carry arms
plane is landing

Process finished with exit code 0
```

Method Overriding

- --> METHOD OVERRIDING --> RUNTIME POLYMORPHISM --> DYNAMIC POLYMORPHISM --> LATE BINDING.

▼ Run Time Polymorphism Using Method Overriding

```
package _1Java_Codes_From_Basics._16polymorphismInJava;
// RUNTIME POLYMORPHISM USING THE METHOD OVERRIDING.
class Plane
```

```
{
    String name;
    void takeOff()
    {
        System.out.println("plane is taking off");
    }
    void fly()
    {
        System.out.println("plane is flying"); //overriding here
    }
    void land()
    {
        System.out.println("plane is landing");
    }
}
class CargoPlane extends Plane
{
    {
        super.name = "Cargo Plane";
    }
    void fly()
    {
        System.out.println(name+" plane is flying");//overriding here
    }
    void carryCargo()
    {
        System.out.println(name+" used to carry goods");
    }
}
class PassengerPlane extends Plane
{
    {
        super.name = "Passenger Plane";
    }
    void fly()
    {
        System.out.println(name+" plane is flying");//overriding here
    }
    void carryPassenger()
    {
        System.out.println(name+" used to carry passengers");
    }
}
class FighterJet extends Plane
{
    {
        super.name = "Fighter Plane";
    }
    void fly()
    {
        System.out.println(name+" is flying");//overriding here
    }
    void CarryArms()
    {
        System.out.println(name+" used to carry arms");
    }
}
class Airport
```

```
{
    //method overriding here
    //NOTE: We create a reference to parent class using that reference we call met
hod of parent class
    void Allow(Plane p)
    {
        p.takeOff();
        p.fly();
        p.land();
    }
}

public class _4methodOverloadingRunPolymorphismEx4
{
    public static void main(String[] args)
    {
        CargoPlane cargo = new CargoPlane();
        PassengerPlane passenger = new PassengerPlane();
        FighterJet fighter = new FighterJet();
        Airport air = new Airport();

        System.out.println("--------------------------");
        air.Allow(cargo);
        System.out.println("--------------------------");
        air.Allow(passenger);
        System.out.println("--------------------------");
        air.Allow(fighter);
    }
}
//output
--------------------------
plane is taking off
Cargo Plane plane is flying
plane is landing
--------------------------
plane is taking off
Passenger Plane plane is flying
plane is landing
--------------------------
plane is taking off
Fighter Plane is flying
plane is landing
```

Important Rules In Method Overriding With Respect To Polymorphism

- **RULE 1:** In method overriding the method signature should be same i.e **The method name and argument type should be same in both parent and child class.**

  ▼ **RULE 1**

```
public void test(int a)
{
```

```
  ---------
    -----------
    -----
}

--> test(int a)       // method signature
```

- **RULE 2: Non-static to static method overriding and static to non-static method overriding is not permitted in java.**
- **RULE 3: Static members cannot be overriding.**

Final Keyword

1. **Final On Class**
    - If a class is made as final then that class **will not participate in inheritance**.

    ▼ Example

```
final class A
{
  void  fun1()
  {
    System.out.println("INSIDE PARENT CLASS");

  }
}
class B extends A  //Parent class A shouldn't be inherited
{
  void  fun1()
  {
    System.out.println("INSIDE child CLASS");

  }
  public static void main(String[] args)
  {
    A a = new A();
    a.fun1();
    B b = new B();
    b.fun1();
  }
}
```

2. **Final On Method**
    - if a method is made as final then that method **can be inherited to child class but cannot be overridden**

    ▼ Example

```
class A
{
  final void  fun1()
  {
    System.out.println("INSIDE PARENT CLASS");

  }
}
class B extends A
{
  void  fun1()
  {
    System.out.println("INSIDE child CLASS");//overridden is not possible
  }
  public static void main(String[] args)
  {
    //A a = new A();
    //a.fun1();
    B b = new B();
    b.fun1();
    //b.fun1();
  }
}
```

3. **Final On Keyword**

- If the variable is made as final then the value of the variable **cannot be altered in other words it will become static in nature.**

▼ **Example**

```
class A
{
  final int speed = 300;
  int   fun1()
  {
    int speed = 250;
    //System.out.println(speed);
    return speed;
  }
}
class B
{
  public static void main(String[] args)
  {
    A a = new A();
    System.out.println(a.fun1());
  }
}
//output
250
```