Polymorphism

- Code reusability
- flexibility
- Reduction in complexity

Let's see how to achieve these advantages


```
class Plane //parent class
{
    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 //child class
{
    void takeOff() // Overridden method
    {
        System.out.println("CargoPlane is taking off from long size runway");
    }
    void fly()  // Overridden method
    {
        System.out.println("CargoPlane is flying at low heights");
    }
    void land() // Overridden method
    {
        System.out.println("CargoPlane is landing at long sized runways");
    }
}
```

```java
class PassengerPlane extends Plane // child class
{
    void takeOff() // Overridden method
    {
        System.out.println("PassengerPlane is taking off from medium sized runway");
    }
    void fly() // Overridden method
    {
        System.out.println("PassengerPlane is flying at medium heights");
    }
    void land() // Overridden method
    {
        System.out.println("PassengerPlane is landing at medium sized runways");
    }
}
class FighterPlane extends Plane // child class
{
    void takeOff() // Overridden method
    {
        System.out.println("FighterPlane is taking off from short sized runway");
    }
    void fly() // Overridden method
    {
        System.out.println("FighterPlane is flying at great heights");
    }
    void land() // Overridden method
    {
        System.out.println("FighterPlane is landing at short short sized runways");
    }
}

class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        Plane ref; //Parent type reference
        ref=cp; //assigning child type reference to parent type
        ref.takeOff();//one reference one behaviour
        ref.fly();
        ref.land();

        ref=pp; //assigning child type reference to parent type
        ref.takeOff();//same reference same behaviour
        ref.fly();
        ref.land();

        ref=fp; //assigning child type reference to parent type
        ref.takeOff();//same reference same behaviour
        ref.fly();
        ref.land();
    }
}
```

Output:

```
CargoPlane is taking off from long size runway
CargoPlane is flying at low heights
CargoPlane is landing at long sized runways
PassengerPlane is taking off from medium sized runway
PassengerPlane is flying at medium heights
PassengerPlane is landing at medium sized runways
FighterPlane is taking off from short sized runway
FighterPlane is flying at great heights
FighterPlane is landing at short short sized runways
```

The above code has achieved polymorphism but it hasn't achieved advantages of polymorphism yet. So next we will see how to achieve code reduction and code flexibility.

```
class Airport
{
    void permit(Plane ref)//Code reduction by
    {                     //passing parent type
        ref.takeOff();    //reference as parameter
        ref.fly();        //and calling the same function
        ref.land();
    }
}
class Demo
{
    public static void main(String[] args)
    {
        CargoPlane cp = new CargoPlane();
        PassengerPlane pp = new PassengerPlane();
        FighterPlane fp = new FighterPlane();

        Airport a = new Airport();
        a.permit(cp); //code flexibility is achieved
        a.permit(pp); //by making use of one function
        a.permit(fp); //call for all the child type ref

    }
}
```

In the above code we have now achieved the advantages of polymorphism.