Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways. For example, lets say we have a class Animal that has a method sound(). Since this is a generic class so we can't give it a implementation like: Roar, Meow, Oink etc. We had to give a generic message.

```
public class Animal{
  public void sound(){
    System.out.println("Animal is making a sound");
  }
}
```

Now lets say we two subclasses of Animal class: Horse and Cat that extends (see Inheritance) Animal class. We can provide the implementation to the same method like this:

```
public class Horse extends Animal{
  @Override
  public void sound(){
    System.out.println("Neigh");
  }
}
public class Cat extends Animal{
  @Override
  public void sound(){
    System.out.println("Meow");
  }
```

}

As you can see that although we had the common action for all subclasses sound() but there were different ways to do the same action. This is a perfect example of polymorphism (feature that allows us to perform a single action in different ways). It would not make any sense to just call the generic sound() method as each Animal has a different sound. Thus we can say that the action this method performs is based on the type of object.

There are two types of polymorphism in java:

1) Static Polymorphism also known as compile time polymorphism
2) Dynamic Polymorphism also known as runtime polymorphism

Compile time Polymorphism (or Static polymorphism)

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism. Method Overloading: This allows us to have more than one method having the same name, if the parameters of methods are different in number, sequence and data types of parameters. We have already discussed Method overloading here: If you didn't read that guide, refer: Method Overloading in Java

Example of static Polymorphism

Method overloading is one of the way java supports static polymorphism. Here we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason this is also known as compile time polymorphism.

```
class SimpleCalculator

{

   int add(int a, int b)

   {

       return a+b;

   }

   int  add(int a, int b, int c)
```

```
    {
        return a+b+c;
    }
}
public class Demo
{
  public static void main(String args[])
  {
        SimpleCalculator obj = new SimpleCalculator();
        System.out.println(obj.add(10, 20));
        System.out.println(obj.add(10, 20, 30));
  }
}
```

Output:

30

60

Runtime Polymorphism (or Dynamic polymorphism)

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism. I have already discussed method overriding in detail in a separate tutorial, refer it: Method Overriding in Java.

Example

In this example we have two classes ABC and XYZ. ABC is a parent class and XYZ is a child class. The child class is overriding the method myMethod() of parent class. In this example we have child class object assigned to the parent class reference so in order to determine which

method would be called, the type of the object would be determined at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

To understand the concept of overriding, you should have the basic knowledge of inheritance in Java.

```
class ABC{
   public void myMethod(){
        System.out.println("Overridden Method");
   }
}
public class XYZ extends ABC{
   public void myMethod(){
        System.out.println("Overriding Method");
   }
   public static void main(String args[]){
        ABC obj = new XYZ();
        obj.myMethod();
   }
}
```

Output:

```
Overriding Method
```