Types of polymorphism in java- Runtime and Compile time polymorphism

BY CHAITANYA SINGH | FILED UNDER: OOPS CONCEPT

In the last tutorial we discussed Polymorphism in Java. In this guide we will see **types of polymorphism**. 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
```

When an overridden method is called through a reference of parent class, then type of the object determines which method is to be executed. Thus, this determination is made at run time.
Since both the classes, child class and parent class have the same

method `animalSound`. Which version of the method(child class or parent class) will be called is determined at runtime by JVM.

Few more overriding examples:

```
ABC obj = new ABC();
obj.myMethod();
// This would call the myMethod() of parent class ABC

XYZ obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class XYZ

ABC obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class XYZ
```

In the third case the method of child class is to be executed because which method is to be executed is determined by the type of object and since the object belongs to the child class, the child class version of myMethod() is called.