Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding

Polymorphism in Java with example

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");
   }
}
```

and

```
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.

What is polymorphism in programming?

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. As we have seen in the above example that we have defined the method sound() and have the multiple implementations of it in the different-2 sub classes.
Which sound() method will be called is determined at runtime so the example we gave above is a **runtime polymorphism example**.

Types of polymorphism and method overloading & overriding are covered in the separate tutorials. You can refer them here:
1. Method Overloading in Java – This is an example of compile time (or static polymorphism)
2. Method Overriding in Java – This is an example of runtime time (or dynamic polymorphism)
3. Types of Polymorphism – Runtime and compile time – This is our next tutorial where we have covered the types of polymorphism in detail. I would recommend you to go though method overloading and overriding before going though this topic.

Lets write down the complete code of it:

Example 1: Polymorphism in Java

Runtime Polymorphism example:
Animal.java

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

Horse.java

```
class Horse extends Animal{
   @Override
   public void sound(){
      System.out.println("Neigh");
   }
   public static void main(String args[]){
            Animal obj = new Horse();
            obj.sound();
   }
}
```

Output:

```
Neigh
```

Cat.java

```
public class Cat extends Animal{
   @Override
   public void sound(){
      System.out.println("Meow");
   }
   public static void main(String args[]){
            Animal obj = new Cat();
            obj.sound();
   }
}
```

Output:

```
Meow
```

Example 2: Compile time Polymorphism

Method Overloading on the other hand is a compile time polymorphism example.

```
class Overload
{
   void demo (int a)
   {
     System.out.println ("a: " + a);
   }
   void demo (int a, int b)
   {
```

```
      System.out.println ("a and b: " + a + "," + b);
   }
   double demo(double a) {
      System.out.println("double a: " + a);
      return a*a;
   }
}
class MethodOverloading
{
   public static void main (String args [])
   {
      Overload Obj = new Overload();
      double result;
      Obj .demo(10);
      Obj .demo(10, 20);
      result = Obj .demo(5.5);
      System.out.println("O/P : " + result);
   }
}
```

Here the method demo() is overloaded 3 times: first method has 1 int parameter, second method has 2 int parameters and third one is having double parameter. Which method is to be called is determined by the arguments we pass while calling methods. This happens at ~~runtime~~ compile time so this type of polymorphism is known as compile time polymorphism.

Output:

```
a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25
```

Types of polymorphism in java- Runtime and Compile time polymorphism

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.

Static and dynamic binding in java

Association of method call to the method body is known as binding. There are two types of binding: **Static Binding** that happens at compile time and **Dynamic Binding** that happens at runtime. Before I explain static and dynamic binding in java, lets see few terms that will help you understand this concept better.

What is reference and object?

```
class Human{
....
}
class Boy extends Human{
  public static void main( String args[]) {
     /*This statement simply creates an object of class
      *Boy and assigns a reference of Boy to it*/
     Boy obj1 = new Boy();

     /* Since Boy extends Human class. The object creation
      * can be done in this way. Parent class reference
      * can have child class reference assigned to it
      */
     Human obj2 = new Boy();
  }
}
```

Static and Dynamic Binding in Java

As mentioned above, association of method definition to the method call is known as binding. There are two types of binding: Static binding and dynamic binding. Lets discuss them.

Static Binding or Early Binding

The binding which can be resolved at compile time by compiler is known as static or early binding. The binding of static, private and final methods is compile-time. **Why?** The reason is that the these method cannot be overridden and the type of the class is determined at the compile time. Lets see an example to understand this:

Static binding example

Here we have two classes Human and Boy. Both the classes have same method walk() but the method is static, which means it cannot be overriden so even though I have used the object of Boy class while creating object obj, the parent class method is called by it. Because the reference is of Human type (parent class). So whenever a binding of static, private and final methods happen, type of the class is determined by the compiler at compile time and the binding happens then and there.

```
class Human{
  public static void walk()
  {
     System.out.println("Human walks");
  }
}
class Boy extends Human{
  public static void walk(){
     System.out.println("Boy walks");
  }
  public static void main( String args[]) {
     /* Reference is of Human type and object is
      * Boy type
      */
     Human obj = new Boy();
     /* Reference is of HUman type and object is
      * of Human type.
      */
     Human obj2 = new Human();
     obj.walk();
     obj2.walk();
  }
}
```

Output:

```
Human walks
Human walks
```

Dynamic Binding or Late Binding

When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late Binding. Method Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method and in this case the **type of the object** determines which method is to be executed. The type of object is determined at the run time so this is known as dynamic binding.

Dynamic binding example

This is the same example that we have seen above. The only difference here is that in this example, overriding is actually happening since these methods are **not** static, private and final. In case of overriding the call to the overriden method is determined at runtime by the type of object thus late binding happens. Lets see an example to understand this:

```
class Human{
  //Overridden Method
  public void walk()
  {
     System.out.println("Human walks");
  }
}
class Demo extends Human{
  //Overriding Method
  public void walk(){
     System.out.println("Boy walks");
  }
  public static void main( String args[]) {
     /* Reference is of Human type and object is
      * Boy type
      */
     Human obj = new Demo();
     /* Reference is of HUman type and object is
      * of Human type.
      */
     Human obj2 = new Human();
     obj.walk();
     obj2.walk();
  }
}
```

Output:

```
Boy walks
Human walks
```

As you can see that the output is different than what we saw in the static binding example, because in this case while creation of object obj the type of the object is determined as a Boy type so method of Boy class is called. Remember the type of the object is determined at the runtime.

Static Binding vs Dynamic Binding

Lets discuss the **difference between static and dynamic binding in Java**.

1. Static binding happens at compile-time while dynamic binding happens at runtime.
2. Binding of private, static and final methods always happen at compile time since these methods cannot be overridden. When the method overriding is actually happening and the reference of parent type is assigned to the object of child class type then such binding is resolved during runtime.
3. The binding of overloaded methods is static and the binding of overridden methods is dynamic.