Polymorphism in Java with example

BY CHAITANYA SINGH | FILED UNDER: OOPS CONCEPT

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
```