POLYMORPHISM In Java

Prepared By Ada Fetic

Additional Java Examples can be found on GitHub
https://github.com/fbada/JavaPolymorphism

Definition: Polymorphism is the ability to take multiple forms. Specifically, in object-oriented programming, polymorphism is where a subclass can define its unique behaviors and share some of the same behaviors as its superclass.

In essence, polymorphism is an object's ability to behave in multiple forms. We use the method of overloading and overriding to achieve the polymorphism. It occurs when a parent class reference refers to a child class object. If the superclass is Abstract or Interface, the class can only be a reference type. The abstract class object must be extended to a regular class, and for the interface, it must be an implemented class.

Type vs. Instance

>> an obj MUST have a superclass type AND a subclass instance

Access

>> objects can only access the methods of their type(not their instance)

>>casting is required to access the methods of their instance

Overridden Methods

>> if a subclass overrides a method, the polymorphic obj will execute the overridden at runtime

>>Extra<<

instanceOf Operator

>> used to determine if an obj is an instance of a particular class

Two types of polymorphism are commonly mentioned.

- Dynamic Polymorphism
- Static Polymorphism

->> Overloading(static, compile time): means the same method name and different parameters occur in the same class. We can overload static, final, and private methods. The return type can be different or identical.

->> Overriding(dynamic, runtime): means the same method name and parameter, occurring in different classes with inheritance relationship(ls-A). We cannot override static, final, and private methods. In a method, the overriding return type must be the same or more convenient. Static methods cannot be overridden.

Sample code to show Polymorphism in action:

```
public class Animal {

    public void makeSound(){
        System.out.println("unknown animal sound");
    }
}
```

Dog class extends from the Animal

```
public class Dog extends Animal {

    @Override
    public void makeSound(){
        System.out.println("woof");
    }

    public void fetch(){
        System.out.println("fetch is fun!");
    }
}
```

class

Cat class extending from Animal Class

```
public class Cat extends Animal {

    @Override
    public void makeSound(){
        System.out.println("meow");
    }

    public void scratch(){
        System.out.println("I am a cat. I scratch things.");
    }

}
```

Now to test it all together and see how the Animal class and the two subsequent classes can be used in Polymorphism.

```
public class Zoo {

    public static void main(String[] args){

        Dog rocky = new Dog();
        rocky.fetch();
        rocky.makeSound();
        feed(rocky);

        Animal sasha = new Dog();
        sasha.makeSound();
        feed(sasha);

        sasha = new Cat();
        sasha.makeSound();
        ((Cat) sasha).scratch();
        feed(sasha);
    }

    public static void feed(Animal animal){

        if(animal instanceof Dog){
            System.out.println("here's your dog food");
        }

        else if(animal instanceof Cat){
            System.out.println("here's your cat food");
        }
    }
}
```

SUMMARY:

What is Polymorphism?

• Polymorphism is a very important concept in OOP because it enables to change of the behavior of the applications in the run time based on the object on which the invocation happens.

It is the ability of an object toAn object can behave in multiple forms. The most common use of polymorphism in Java is when a parent class reference type of variable is used to refer to a child class object.
Example in UI Automation:

```
WebDriver driver = new ChromeDriver();
```

Real-Life Scenario

Suppose you are in the classroom at that time you will behave like a student.
Suppose when you are at home you behave like a son or daughter.
Suppose you are in the market at that time you behave like a customer.

TWO TYPES => **Compile Time** which is static, and **Run-Time** Polymorphism, which is related to child and parent
class.

The best example of compile-time or static polymorphism is the method of

overloading. Whenever an object is bound with its functionality at compile time is known as compile-time or static polymorphism in java. **Compile-time polymorphism is also known as static polymorphism. We can achieve static polymorphism through method overloading in java.**

The best example of run-time or dynamic polymorphism is the method overriding. Whenever an object is bound with its functionality at run-time is known as run-time or dynamic polymorphism in java.
Run-time polymorphism is also known as dynamic polymorphism. We can achieve dynamic polymorphism through method overriding.

Polymorphism is implemented using the concept of Method overloading and method overriding. This can only happen when the classes are under the parent and child relationship using inheritance.

What is the difference between Polymorphism and Inheritance?

- Like in the real world, **INHERITANCE** is used to define the relationship between two classes
  - In Java, we have a Parent class (also known as a superclass) and a child class (also known as a subclass).

o A child class can reuse all the codes written in the Parent class and only write code for behavior different from the Parent.
o Inheritance is meant for code reuse.

- On the other hand, **POLYMORPHISM** is the ability of an object to behave in multiple forms.
  - o It is classified as overloading and overriding.
- By the way, they are related to each other because of their inheritance which makes Polymorphism possible without any relationship between two classes. It is not possible to write polymorphic code.
  - **o Dynamic Polymorphism→Overriding**
  - **o Static Polymorphism→Overloading**