Polymorphism in Java

What Is Polymorphism?

Polymorphism is the concept of one entity providing multiple implementations or behaviours. (Something like an Avatar!)

Peter Parker Is Polymorphic

As a high school student, Peter goes to school and hangs out with friends. As Spider Man, he bashes up baddies. Peter has multiple behaviours based on the context. Peter is polymorphic.

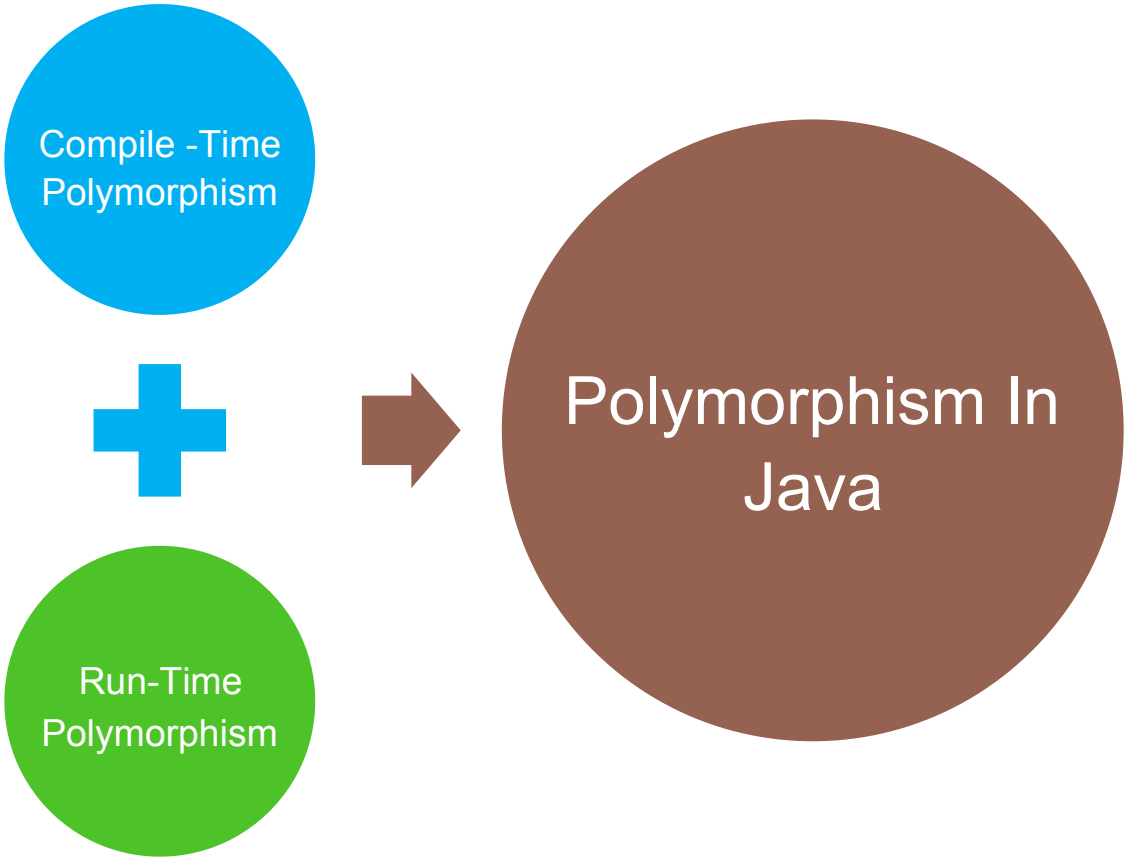
Java And Polymorphism

Polymorphism

Java provides multiple forms of polymorphism. It allows you to define the same method name to do multiple different things. It also allows child classes to redefine/ override parents' behaviours for the same method.

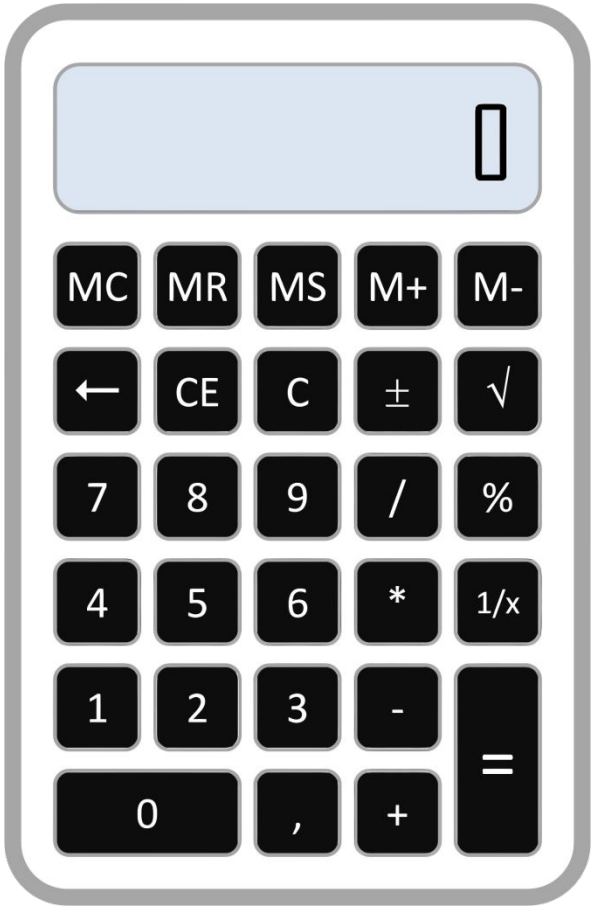
Java Provides 2 Types Of Polymorphism

Polymorphism



Compile-Time Or Static Polymorphism

Compile-time polymorphism refers to behaviour that is resolved when your Java class is compiled. Method overloading is an example of compile-time polymorphism. Method overloading is how you can use method with same name to do different things based on the parameters passed.
The add method in Class Calculator for example can add 2 integers or floats based on the types of parameters.



A calculator can add 2 integers. It can also add 2 floats. The addition method adds differently based on the inputs.

Illustration – Method Overloading

The Class PolymorphicCalculator provides multiple implementation of the method add.

```
Public class PolymorphicCalculator
{
    //Add method to add integers
     int add (int a, int b)
    {
            System.out.println("Adding 2 Integers");
            return a+b;
    }
 //Another Add method to add floats. Same name
//  differentiated by data types of parameters and
//  return types
    float add (float a, float b)
    {
            System.out.println("Adding 2 Floats");
            return a+b;
    }
}
```

Illustration – Method Overloading

Adding a main method to Polymorphic Calculator. The compiler chooses the correct add method to call based on the data types of actual arguments passed.

```
Public static void main(String args[])
{
PolymorphicCalculator calc =
new  PolymorphicCalculator ();
//calling the add method with integer
arguments
System.out.println(calc.add (3,5));
//calling the add method with float arguments
System.out.println(calc.add(3.5,5.6));
}
}
```

The output will be:

```
Adding 2 Integers
8
Adding 2 Floats
9.1
```

Run-Time Or Dynamic Polymorphism

Run-time polymorphism refers to behaviour that is resolved when your Java class is run by the JVM. Method overriding by the sub-class is an example of run-time polymorphism.
Method overriding allows child classes to provide their own implementation of a method also defined in the parent class.
The JVM decides which version of the method (the child's or the parent's) to call based on the object through which the method is invoked.

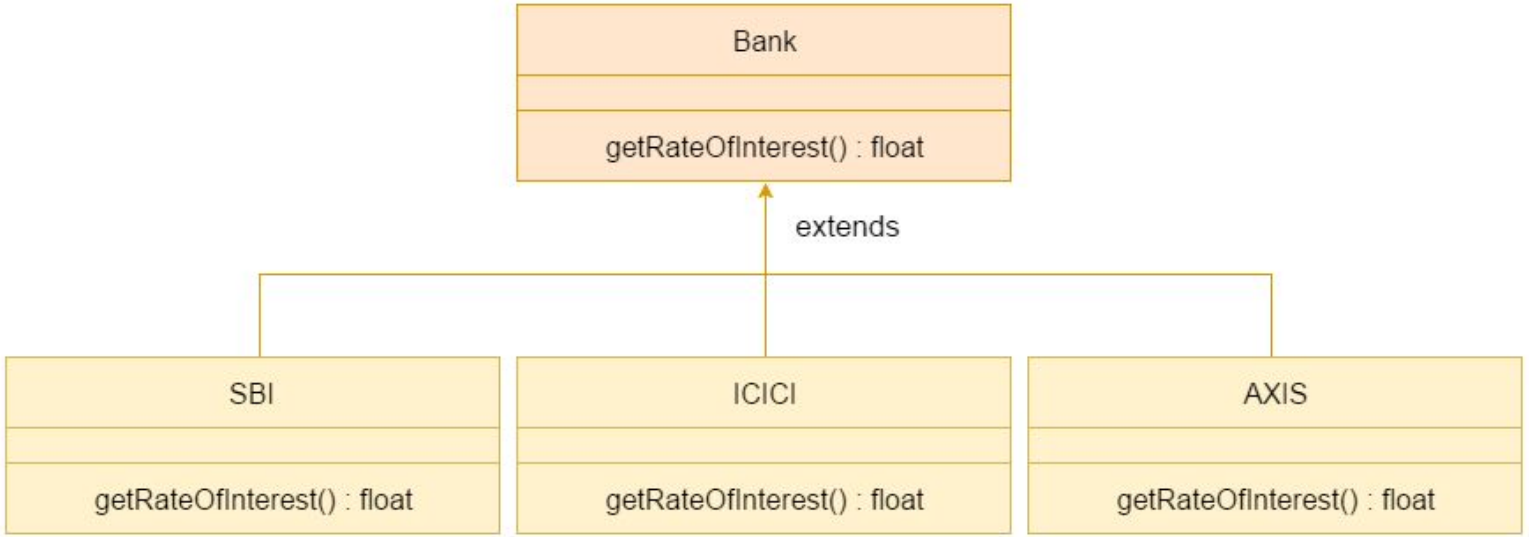
Example

```
class Bike{
    void run(){System.out.println("running");}
  }
  class Splendor extends Bike{
    void run(){System.out.println("running safely with 60km");}

    public static void main(String args[]){
      Bike b=new Bike();
      b.run();
      b=new Splendor();
      b.run();
    }
  }
```

Output:
Running
running safely with 60km

Using upcasting method



```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Advantages Of Polymorphism

- Code Cleanliness
- Ease Of Implementation
- Aligned With Real World
- Overloaded Constructors
- Reusability And Extensibility