Polymorphism

Dr. Krishnendu Guha

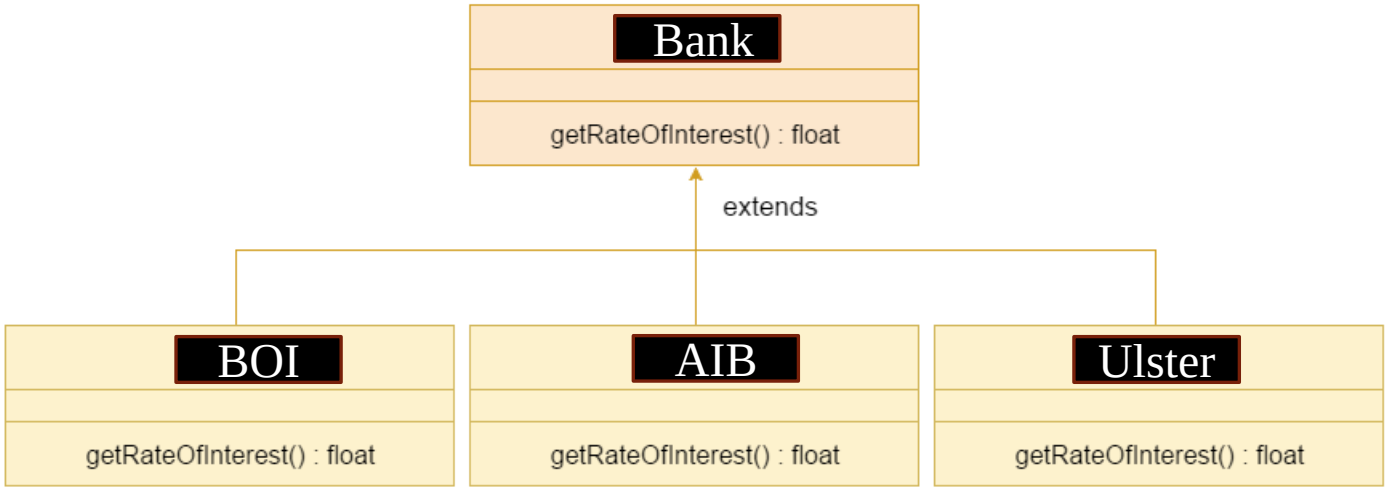
Assistant Professor/ Lecturer

School of Computer Science and Information Technology

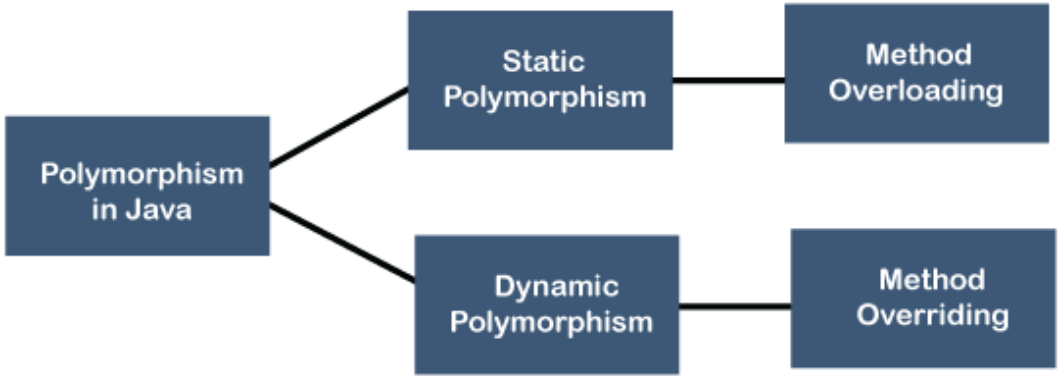
University College Cork

What is Polymorphism?

- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*.
- Polymorphism is derived from **2 Greek words: poly and morphs**. The word **"poly" means many and "morphs" means forms**. So, polymorphism means many forms.
- It occurs when we have ***many classes that are related to each other by inheritance***.
- Polymorphism uses those methods to perform different tasks and is an example of loose coupling.
- Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, BOI, AIB, and Ulster banks are providing 8.4%, 7.3%, and 9.7% rate of interest.



- Polymorphism can be **static or dynamic**
- Static Polymorphism in Java is a type of polymorphism that **collects the information for calling a method at compilation time**
- Dynamic Polymorphism is a type of polymorphism that **collects the information for calling a method at runtime.**
  - In this, multiple methods can be defined with same name and signature in the superclass and subclass



Static Polymorphism

- Static polymorphism relates to **method overloading**.
- In this case, any errors are resolved at **compile time**.
- The name is **static** as the **code is not executed during compilation**.
- Method overloading is in the same class, where **more than one method has the same name but different signatures.**
- Example:

```
void sum (int a , int b);
void sum (int a , int b, int c);
void sum (float a, double b);
```

Dynamic Polymorphism

- Dynamic polymorphism relates to **method overriding.**
- The call to an overridden method are resolved at **run time**.
- Method Overriding is **redefining a super class method in a sub class.**
- Method overriding is when one of the methods in the superclass is redefined in the sub-class.
- In this case, the **signature of the method remains the same.**

Example:

```
class X{
	public int sum(){
		// some code
	}
}
class Y extends X{
	public int sum(){
		//overridden method
		//signature is same
	}
}
```

Method Overloading

- When there are **multiple functions with the same name** but **different parameters** then these functions are said to be **overloaded**.

- Functions can be overloaded by **changes in the number of arguments or/and a change in the type of arguments**.

```
class A1 {
        // Method with 2 integer parameters
        static int Multiply(int a, int b){
                // Returns product of integer numbers
                return a * b;
        }
        // Method 2
        // With same name but with 2 double parameters
        static double Multiply(double a, double b){
                // Returns product of double numbers
                return a * b;
        }
}

// Main class
class Main {
        // Main driver method
        public static void main(String[] args){
                // Calling method by passing
                // input as in arguments
                System.out.println(A1.Multiply(2, 4));
                System.out.println(A1.Multiply(5.5, 6.3));
        }
}
```

```
8
34.65
```

```java
// Java program for Method Overloading
// by Using Different Numbers of Arguments

// Class 1
// Helper class
class Helper {

    // Method 1
    // Multiplication of 2 numbers
    static int Multiply(int a, int b)
    {

        // Return product
        return a * b;
    }

    // Method 2
    // // Multiplication of 3 numbers
    static int Multiply(int a, int b, int c)
    {

        // Return product
        return a * b * c;
    }
}
```

```java
// Class 2
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Calling method by passing
        // input as in arguments
        System.out.println(Helper.Multiply(2, 4));
        System.out.println(Helper.Multiply(2, 7, 3));
    }
}
```

```
8

42
```

Method Overriding

It provides a specific implementation to a method that is already present in the parent class. it is used to achieve run-time polymorphism. Remember that, it is **not possible to override** the **static** method. Hence, we cannot override the main() method also because it is a static method.

```
//Creating a parent class.
class Bank{
        int getRateOfInterest(){return 0;}
}
//Creating child classes.
class BOI extends Bank{
        int getRateOfInterest(){return 8;}
}
class AIB extends Bank{
        int getRateOfInterest(){return 7;}
}
class Ulster extends Bank{
        int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Main{
        public static void main(String args[]){
                AIB a=new AIB();
                BOI b=new BOI();
                Ulster u=new Ulster();
                System.out.println("AIB Rate of Interest: "+a.getRateOfInterest());
                System.out.println("BOI Rate of Interest: "+b.getRateOfInterest());
                System.out.println("Ulster Rate of Interest: "+u.getRateOfInterest());
        }
}
```

```
AIB Rate of Interest: 7
BOI Rate of Interest: 8
Ulster Rate of Interest: 9
```

Super Keyword

- The **super** keyword in Java is a reference variable which is used to **refer immediate parent class object.**
- Whenever you **create the instance of subclass, an instance of parent class is created implicitly** which is referred by super reference variable.
  - Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

super is used to refer immediate parent class instance variable.

```
class University{
        String name="UCC";
}
class Dept extends University{
        String name="CSIT";
        void printName(){
                System.out.println(name);//prints name of Dept
                System.out.println(super.name);//prints name of University
        }
}
class Main{
        public static void main(String args[]){
        Dept d=new Dept();
        d.printName();
        }
}
```

```
CSIT
UCC
```

super can be used to invoke parent class method

```
class University{
        void name(){System.out.println("UCC");}
}
class Dept extends University{
        void name(){System.out.println("CSIT");}
        void student_total(){System.out.println("500");}
        void part(){
                super.name();
                student_total();
        }
}
class Main{
        public static void main(String args[]){
                Dept d=new Dept();
                d.part();
}
}
```

```
UCC
500
```

super is used to invoke parent class constructor.

```
class University{
        University(){
                System.out.println("UCC");
        }
}
class Dept extends University{
        Dept(){
                super();
                System.out.println("CSIT");
        }
}
class Main{
        public static void main(String args[]){
        Dept d=new Dept();
}
}
```

```
UCC
CSIT
```

Program with this and super

```
class Person{
        int id;
        String name;
        Person(int id,String name){
                this.id=id;
                this.name=name;
        }
}
class Emp extends Person{
        float salary;
        Emp(int id,String name,float salary){
                super(id,name);//reusing parent constructor
                this.salary=salary;
        }
        void display(){System.out.println(id+" "+name+" "+salary);}
}
```

```
class Main{
        public static void main(String[] args){
                Emp e1=new Emp(1,"John",45000);
                e1.display();
        }
}
```

```
1 John 45000.0
```


69STATUS.SRKH.IN
Thank You
ANY QUESTIONS?