POLYMORPHISM IN JAVA

Velocity

AUGUST 4, 2022

VELOCITY

Pune

Polymorphism in java

Polymorphism-

One entity that behaves differently in different cases called as polymorphism.

We can achieve polymorphism in two ways

-Method overloading

-Method overriding

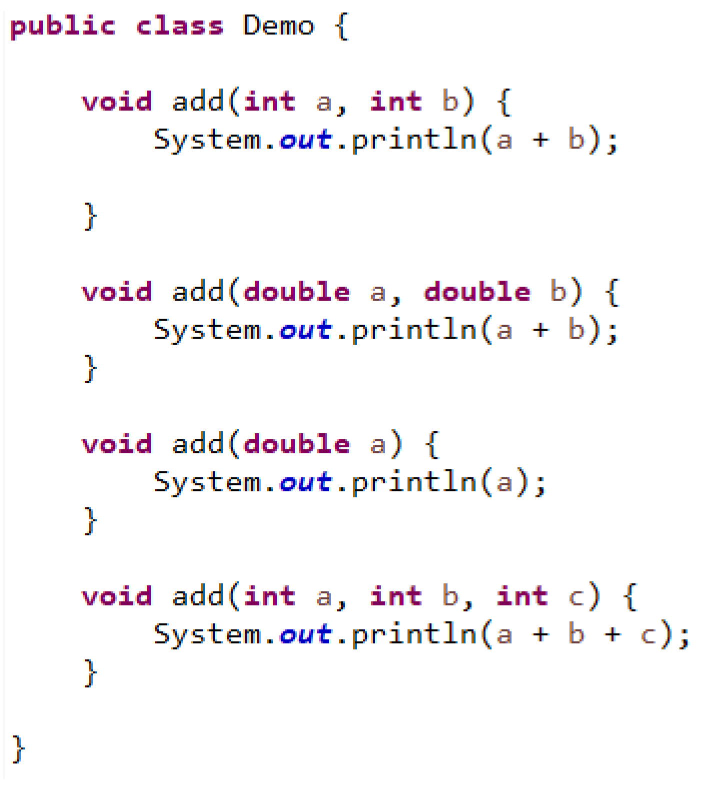
1. Method overloading

It is the same method name with different argument called as Method overloading. There is no need of super and sub class relationship. It is called as early binding, compile time polymorphism or static binding.

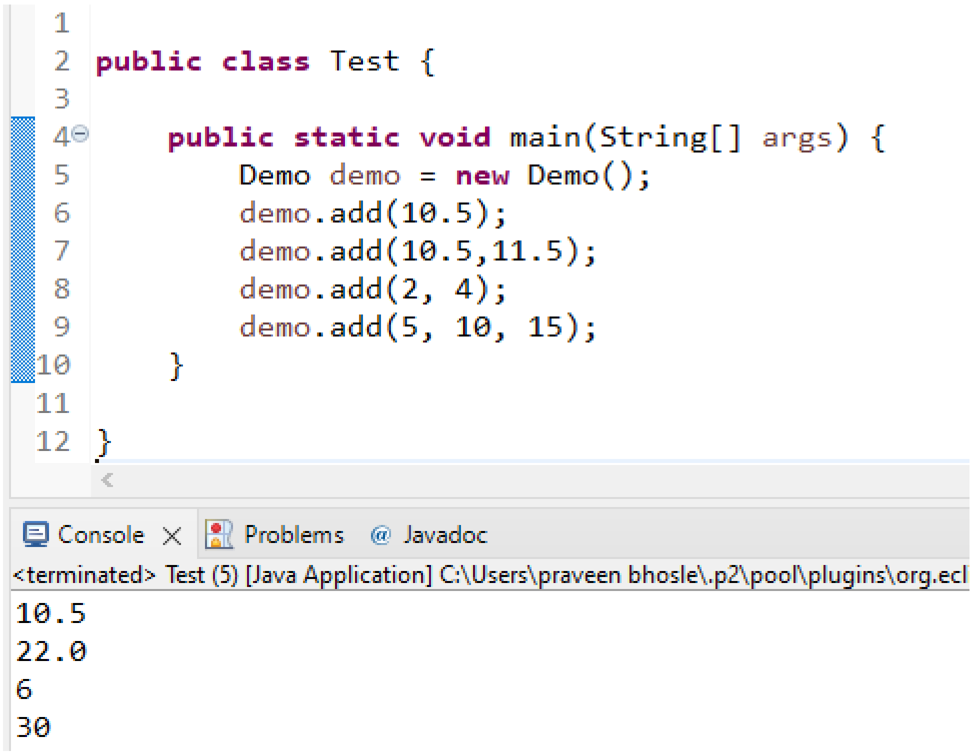
Rules for method overloading

- Method name must be same.
- Parameter or argument must be different.
- Return type is anything
- Access specifier is anything
- Exception thrown is anything

Example:



```
public class Demo {

    void add(int a, int b) {
        System.out.println(a + b);

    }

    void add(double a, double b) {
        System.out.println(a + b);
    }

    void add(double a) {
        System.out.println(a);
    }

    void add(int a, int b, int c) {
        System.out.println(a + b + c);
    }

}
```



```java

public class Test {

    public static void main(String[] args) {
        Demo demo = new Demo();
        demo.add(10.5);
        demo.add(10.5,11.5);
        demo.add(2, 4);
        demo.add(5, 10, 15);
    }

}
```

```
<terminated> Test (5) [Java Application] C:\Users\praveen bhosle\.p2\pool\plugins\org.ecl
10.5
22.0
6
30
```

Why we need method overloading?

Suppose we got the business requirement from the client in last year

Class Employee {

Void addStudent (String firstname, string lastname, string city) { }

End user is calling this method from class as below

//End User 1

addStudent (“ram”,”pawar”,”Pune”);

//End User 2

addStudent (“ram”,”deshmukh”,”Mumbai”);

After that I got the new requirement from the client in current year, to update the pan card details also in it.

What options we have?

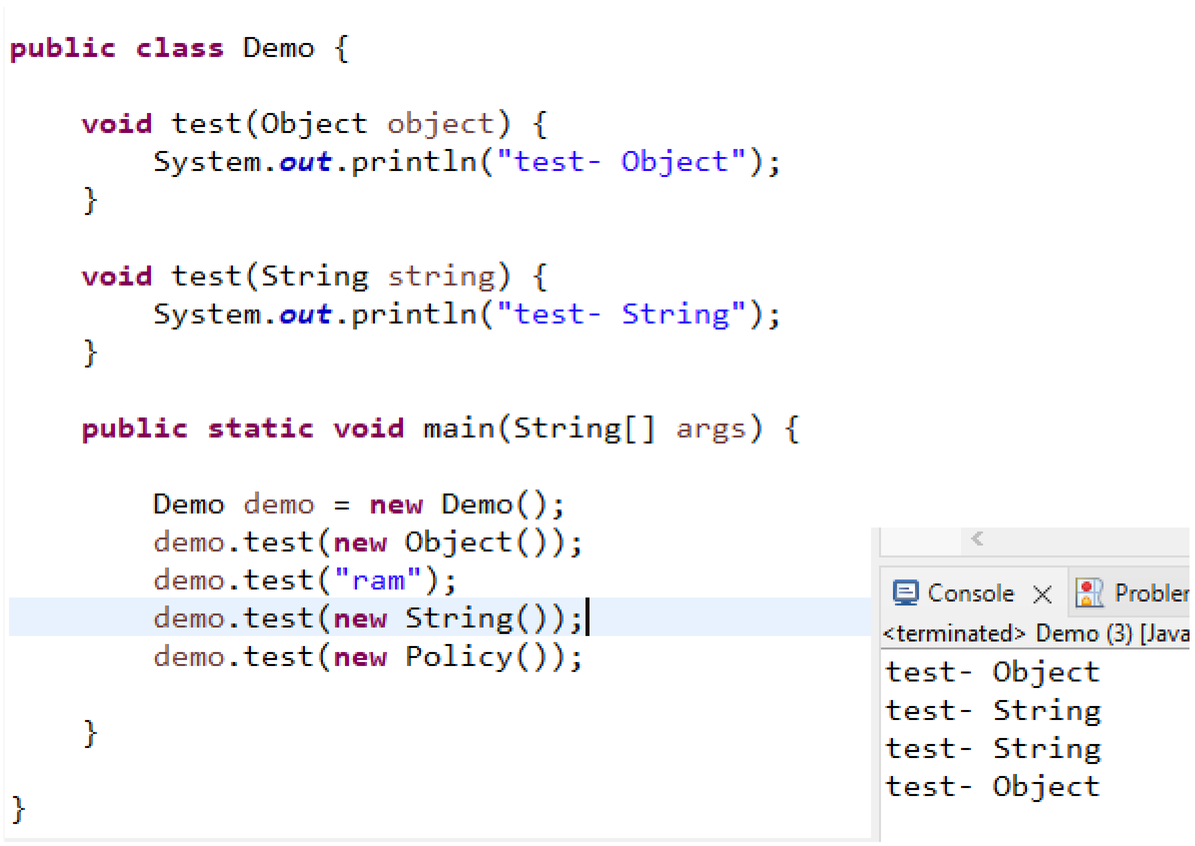
1.Modified into the existing method.

2. Create the new method with new parameter.

First way: Modifying into existing method is not good approach, as it will increase the unit testing of it. If we are making the changes into existing method, then how user will call the method, I mean to say they need to add one more extra field, and in future again, you got requirement to add one more field so every time user need to change at their side, this is not the good thing.

Second way, create the same method in that class and add the new field into it. If client wants pan card details so he can call that method otherwise calls the first method if pan card is not required.

Example -2



```
public class Demo {

    void test(Object object) {
        System.out.println("test- Object");
    }

    void test(String string) {
        System.out.println("test- String");
    }

    public static void main(String[] args) {

        Demo demo = new Demo();
        demo.test(new Object());
        demo.test("ram");
        demo.test(new String());
        demo.test(new Policy());

    }

}
```

Console ✕ Probler

<terminated> Demo (3) [Java

```
test- Object
test- String
test- String
test- Object
```

Q>Why it is called as compile time polymorphism?

A> Because it is decided at compile time which one method should get called that’s why it is called as compile time polymorphism.

Method Overriding

It is the same method name with same argument called as method overriding.

There is need of super and sub relationship. It is called as late binding, run time polymorphism or dynamic binding. etc.

Rules for method overriding-

Method name must be same.

Return type must be same or different.

Access specifier is anything.

Parameters must be same.

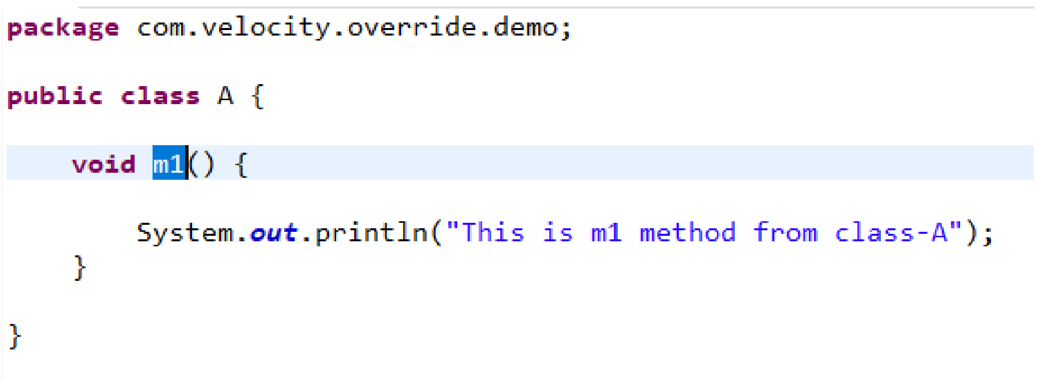
Note- we can extend the method scope in overriding but not reduce the visibility of it.

Why we need method overriding

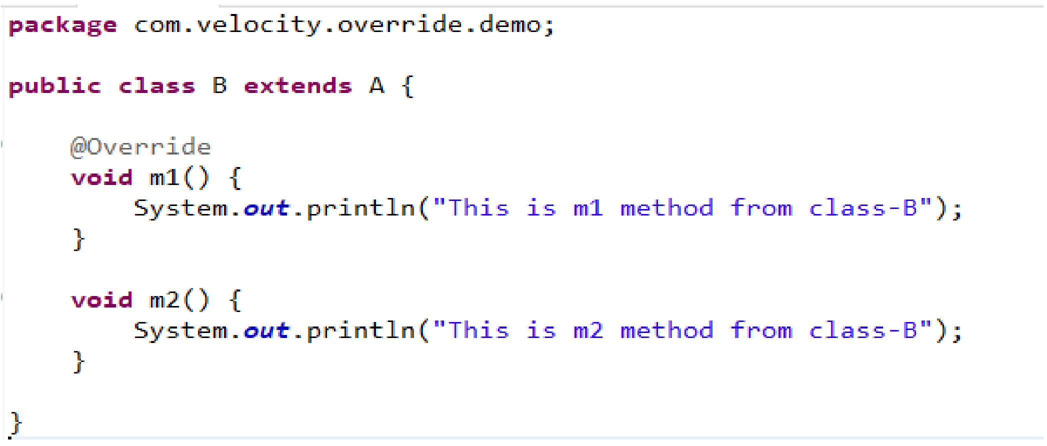
Maintainability

Readability of code.

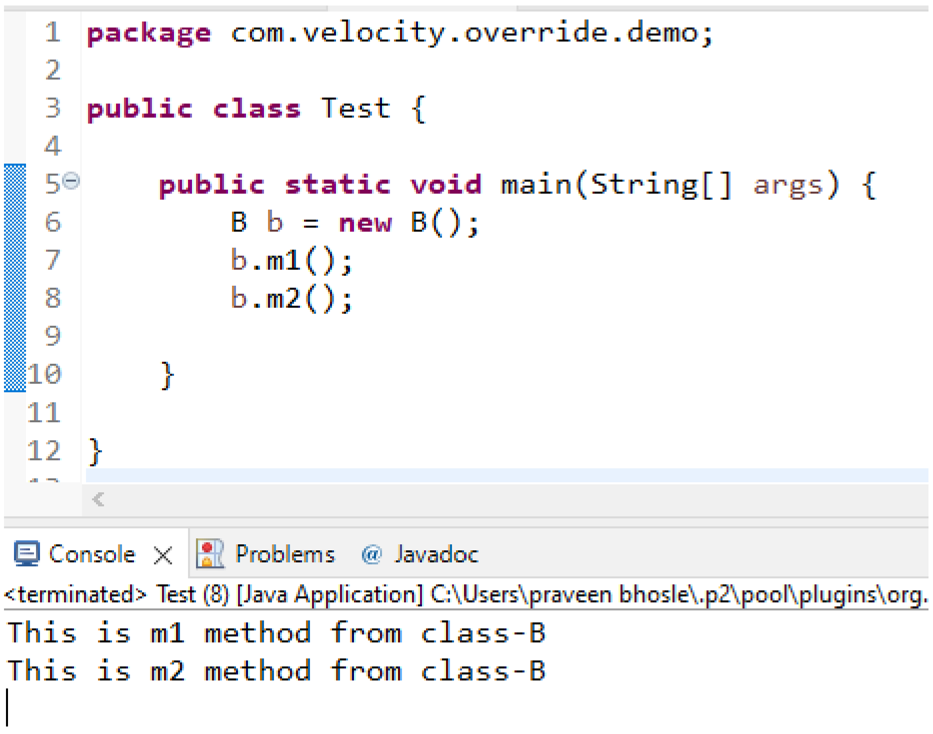
Example



```
package com.velocity.override.demo;

public class A {

    void m1() {

        System.out.println("This is m1 method from class-A");
    }

}
```



```
package com.velocity.override.demo;

public class B extends A {

    @Override
    void m1() {
        System.out.println("This is m1 method from class-B");
    }

    void m2() {
        System.out.println("This is m2 method from class-B");
    }

}
```



```java
package com.velocity.override.demo;

public class Test {

    public static void main(String[] args) {
        B b = new B();
        b.m1();
        b.m2();

    }

}
```

```
<terminated> Test (8) [Java Application] C:\Users\praveen bhosle\.p2\pool\plugins\org.
This is m1 method from class-B
This is m2 method from class-B
```

Program Explanation-

-In the above program, B is implementing the method m1 () with the same signature as super class A i.e. m1 () of class B is overriding m1() of class A.

-If you want to add new features to existing class, then you should not disturb the existing class. You should always write the subclass of that class that is the best practice.

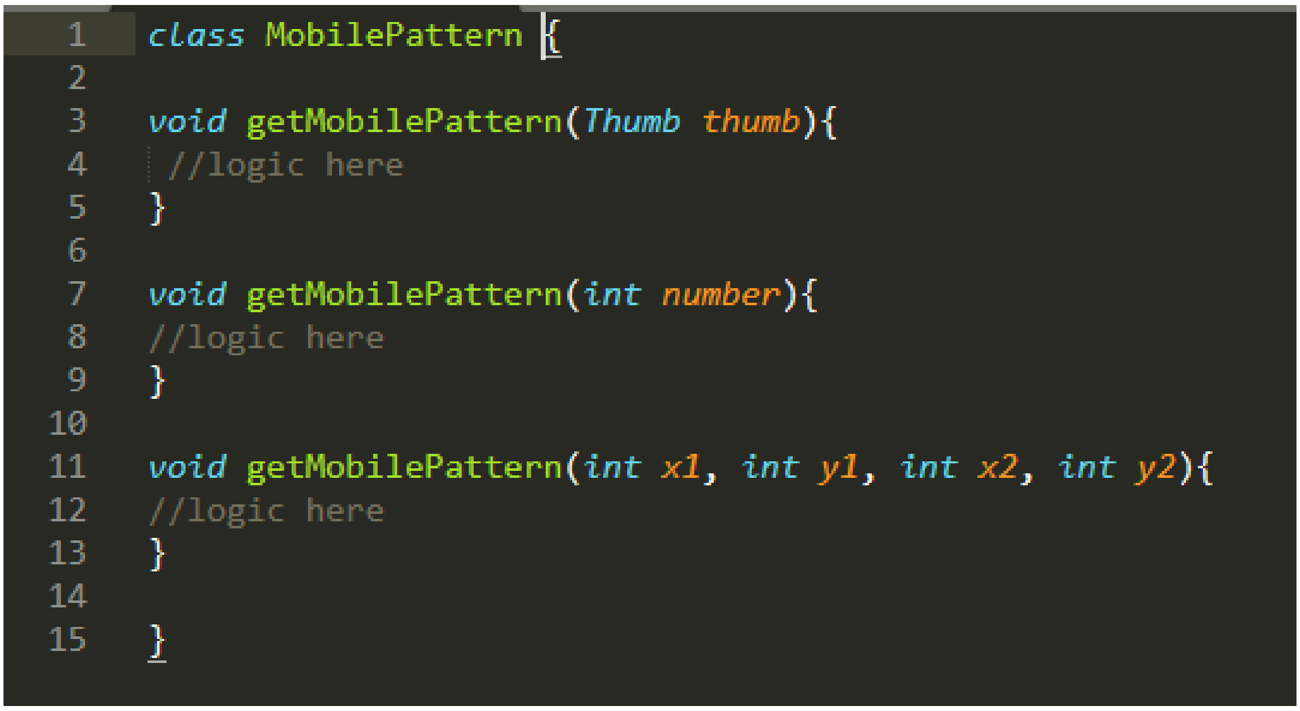
Why we write the sub class

1. To add the new features
2. To inherit the existing functionality.

Subclass method's access modifier must be the same or higher than the superclass method access modifier

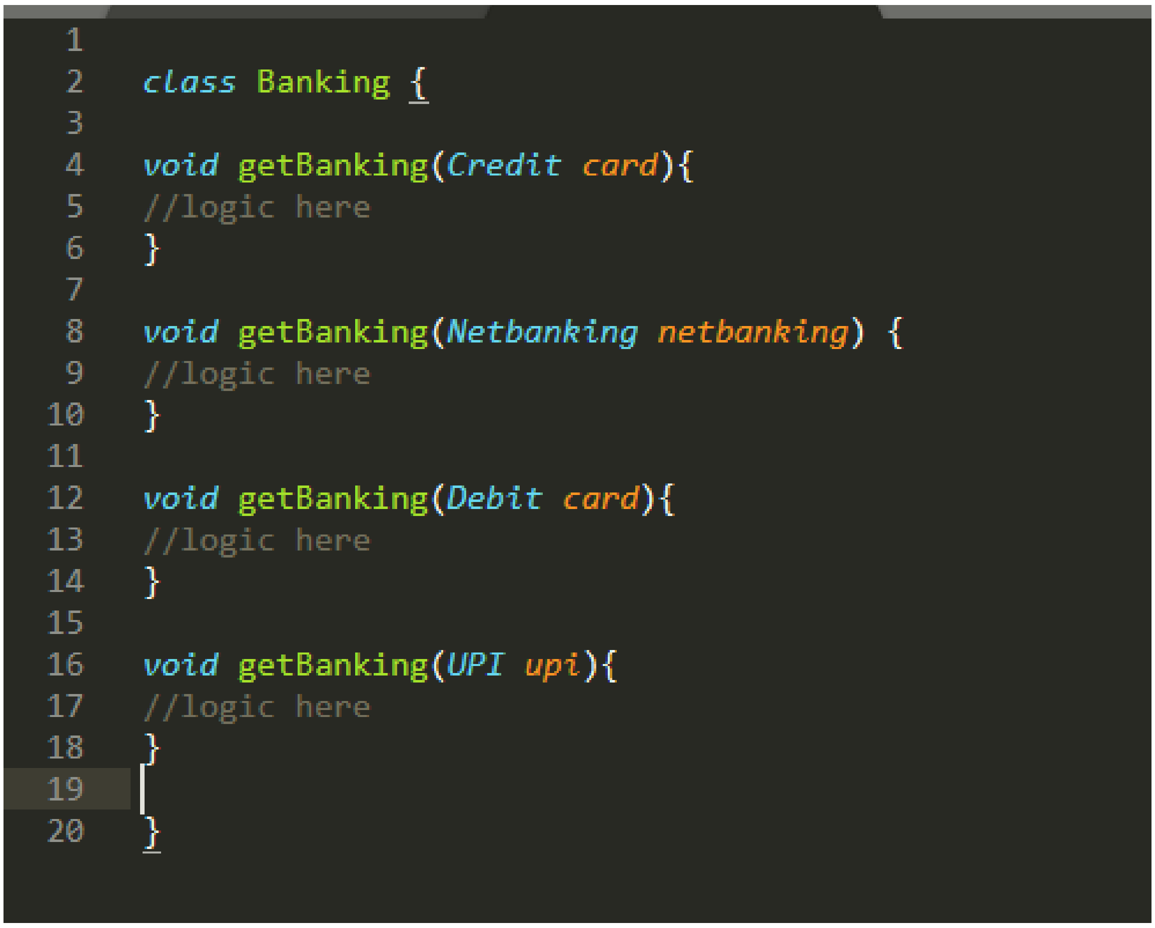
superclass	In subclass, we can have access specifier
public	public
protected	protected, public
default	default, protected, public
private	We cannot override the private

Method Overloading- Live Example-



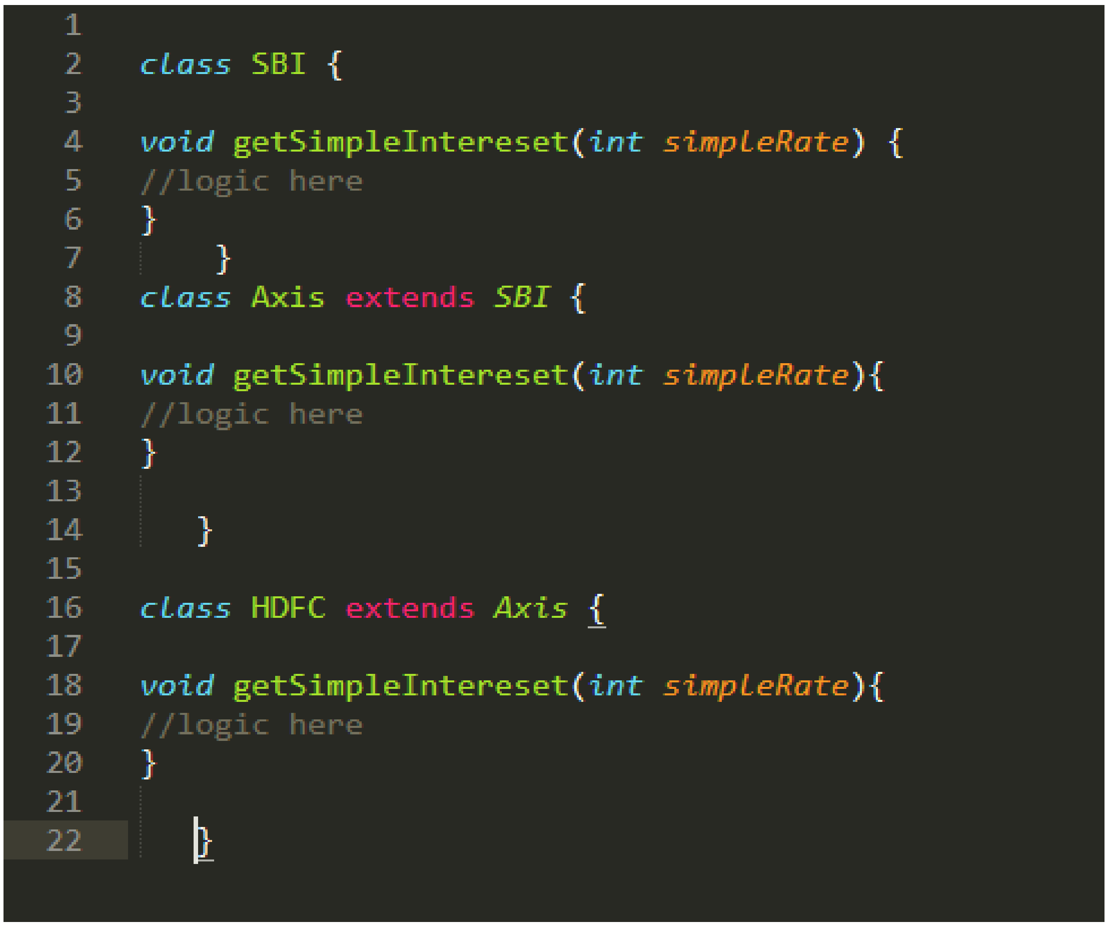
```
class MobilePattern {

void getMobilePattern(Thumb thumb){
 //logic here
}

void getMobilePattern(int number){
//logic here
}

void getMobilePattern(int x1, int y1, int x2, int y2){
//logic here
}

}
```

Example 2



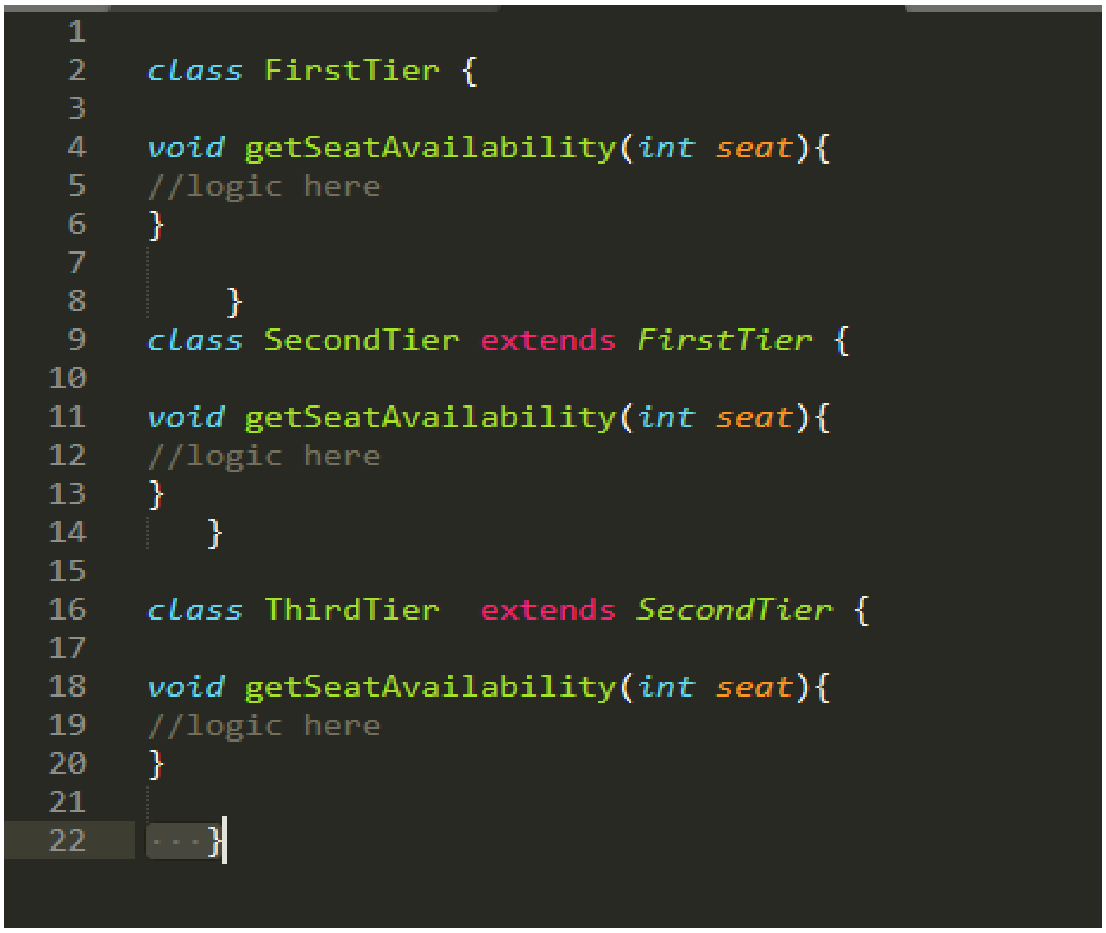
```

class Banking {

void getBanking(Credit card){
//logic here
}

void getBanking(Netbanking netbanking) {
//logic here
}

void getBanking(Debit card){
//logic here
}

void getBanking(UPI upi){
//logic here
}

}
```

Method Overriding- Live Example-1



```

class SBI {

void getSimpleIntereset(int simpleRate) {
//logic here
}
    }
class Axis extends SBI {

void getSimpleIntereset(int simpleRate){
//logic here
}

   }

class HDFC extends Axis {

void getSimpleIntereset(int simpleRate){
//logic here
}

   }
```

Live Example-2



```

class FirstTier {

void getSeatAvailability(int seat){
//logic here
}

    }
class SecondTier extends FirstTier {

void getSeatAvailability(int seat){
//logic here
}
   }

class ThirdTier  extends SecondTier {

void getSeatAvailability(int seat){
//logic here
}

   }
```