COM2104: Advanced Programming

LECTURE 5: ABSTRACT&INTERFACE

Objectives

- Know what is abstract class and how to implement it.
- Know what is interface and how to implement it.
- Know how to refer to UML diagram to create abstract classes and interface.

ABSTRACT CLASS

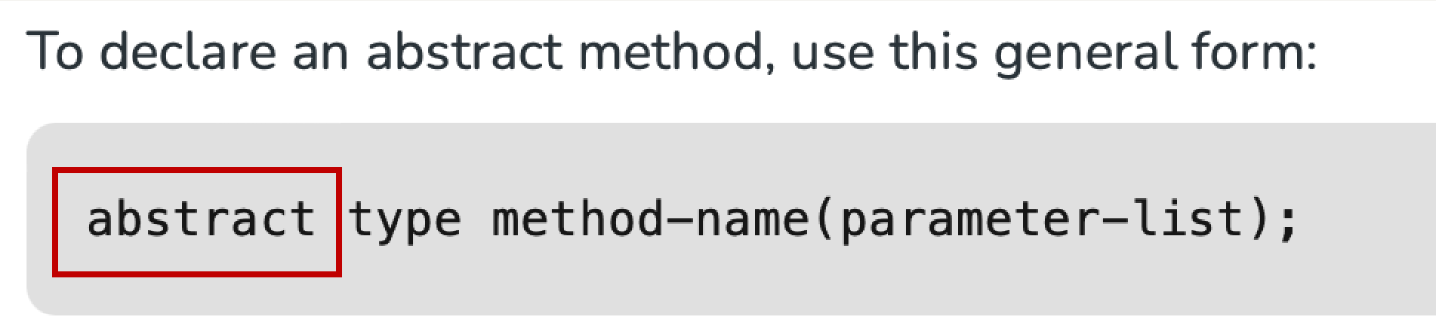
Abstract Class

- In Java, abstract class is declared with the abstract keyword. It may have both abstract and non-abstract methods(methods with bodies).
- An abstract is a Java modifier applicable for classes and methods in Java but *not for Variables*.
- Java abstract class is a class that cannot be instantiated by itself, it needs to be subclassed by another class to use its properties.

Abstract methods

Don't provide the Implementation

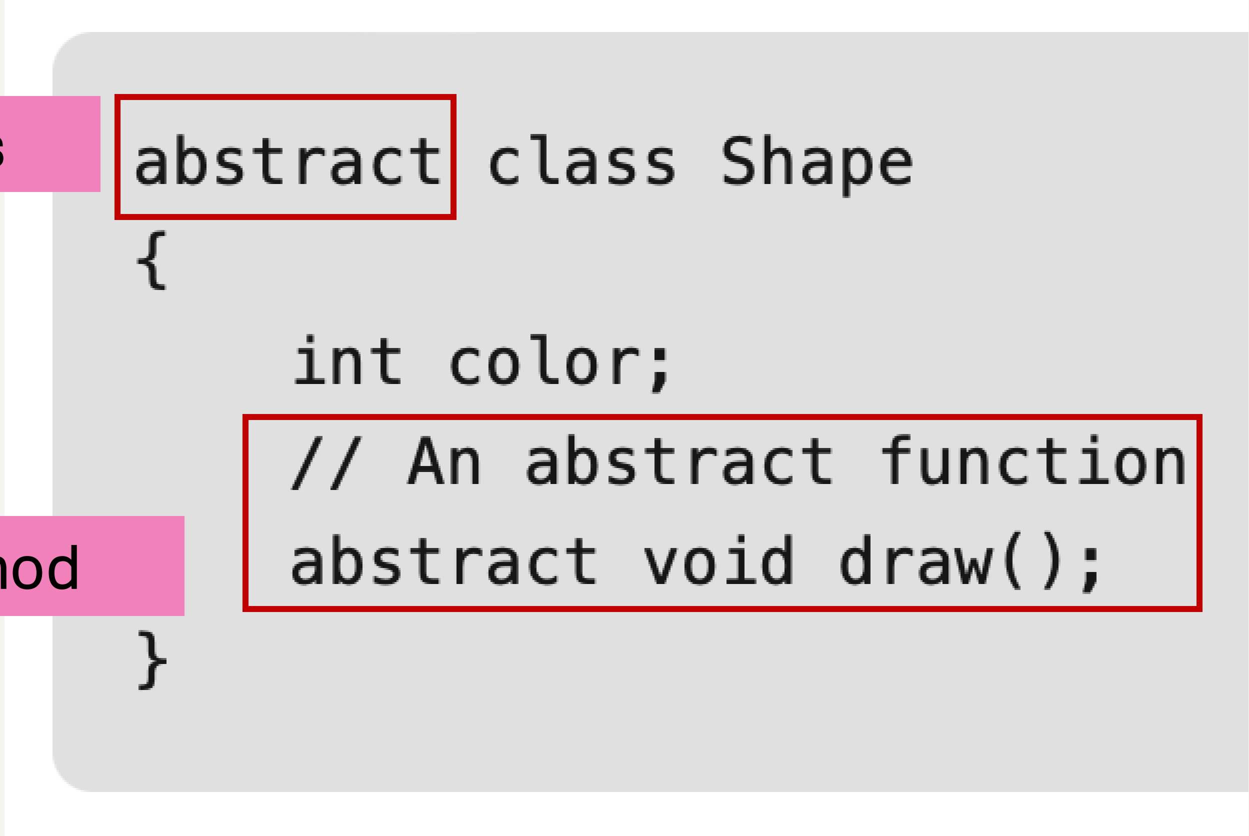
- The abstract Method uses abstract keyword to indicate.
- This method is used for creating blueprints for classes or interfaces. These methods are defined but don't provide the implementation.
- Abstract Methods can only be implemented using subclasses or classes that implement the interfaces.



To declare an abstract method, use this general form:

```
abstract type method-name(parameter-list);
```

One simple example about Abstract Class



```
abstract class Shape
{
     int color;
     // An abstract function
     abstract void draw();
}
```

Abstract class

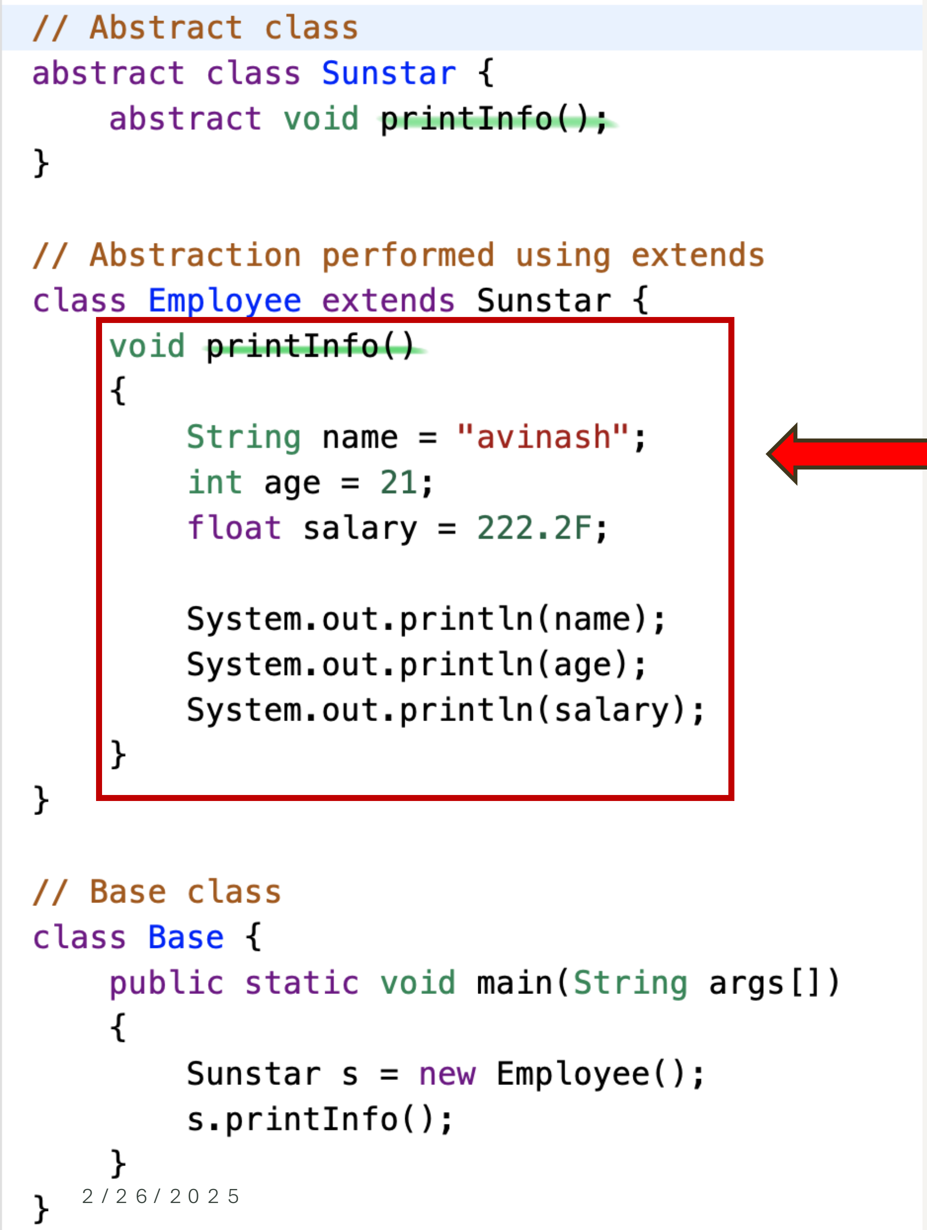
Abstract method

Subclass

Child class of one abstract class

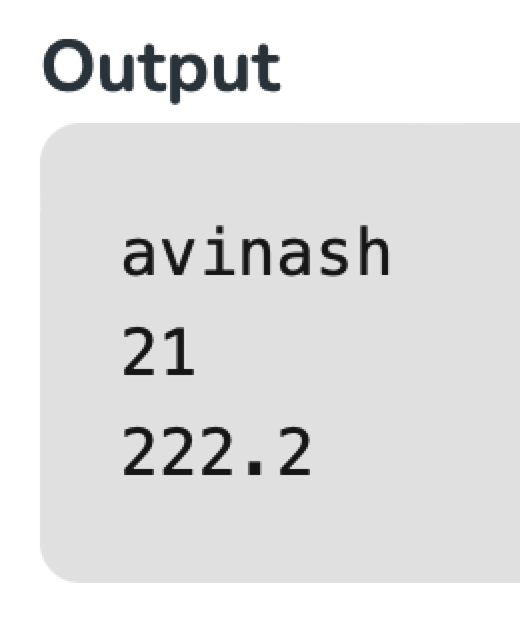
- The child class used ***extends*** keyword for its abstract class.
- All abstract methods in the abstract class should be specifically implemented in the child class.

1. Example of Abstract Class that has Abstract methods



```
// Abstract class
abstract class Sunstar {
    abstract void printInfo();
}

// Abstraction performed using extends
class Employee extends Sunstar {
    void printInfo()
    {
        String name = "avinash";
        int age = 21;
        float salary = 222.2F;

        System.out.println(name);
        System.out.println(age);
        System.out.println(salary);
    }
}

// Base class
class Base {
    public static void main(String args[])
    {
        Sunstar s = new Employee();
        s.printInfo();
    }
}
```

The implementation of abstract method realized in the child class.



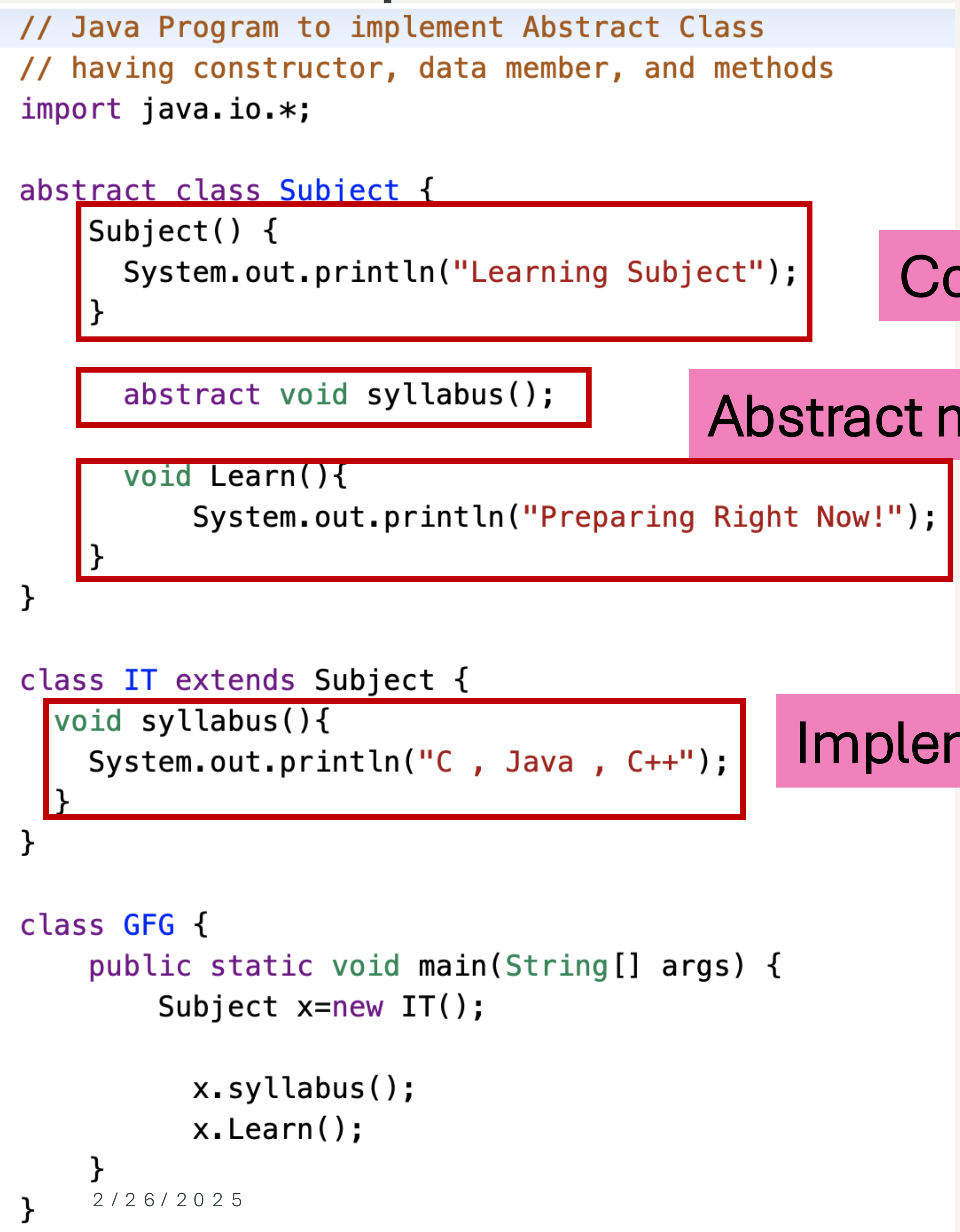
Output

```
avinash
21
222.2
```

2. Elements abstract class can have

- data member
- abstract method
- method body (non-abstract method)
- constructor
- main() method.

One example



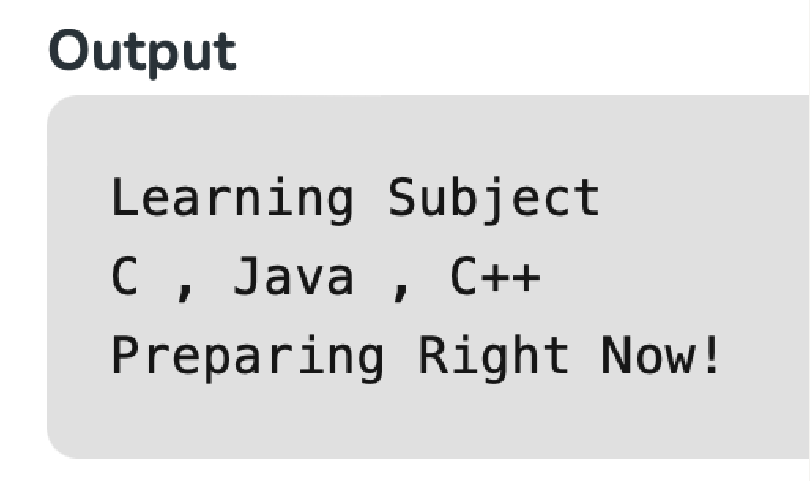
```java
// Java Program to implement Abstract Class
// having constructor, data member, and methods
import java.io.*;

abstract class Subject {
    Subject() {
      System.out.println("Learning Subject");
    }

      abstract void syllabus();

      void Learn(){
          System.out.println("Preparing Right Now!");
    }
}

class IT extends Subject {
  void syllabus(){
    System.out.println("C , Java , C++");
  }
}

class GFG {
    public static void main(String[] args) {
        Subject x=new IT();

          x.syllabus();
          x.Learn();
    }
}
```

Constructor

Abstract method

Non-Abstract method

Implementation of abstract method

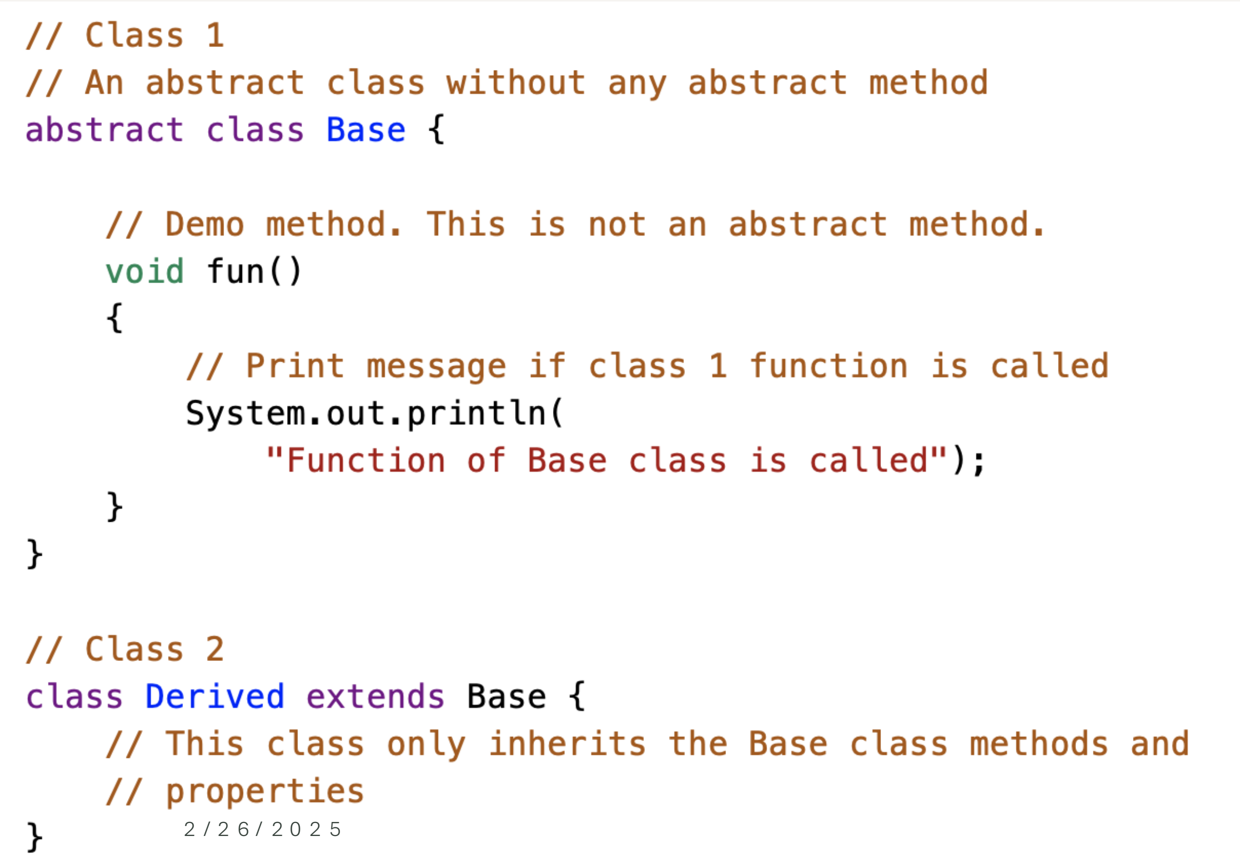


Output

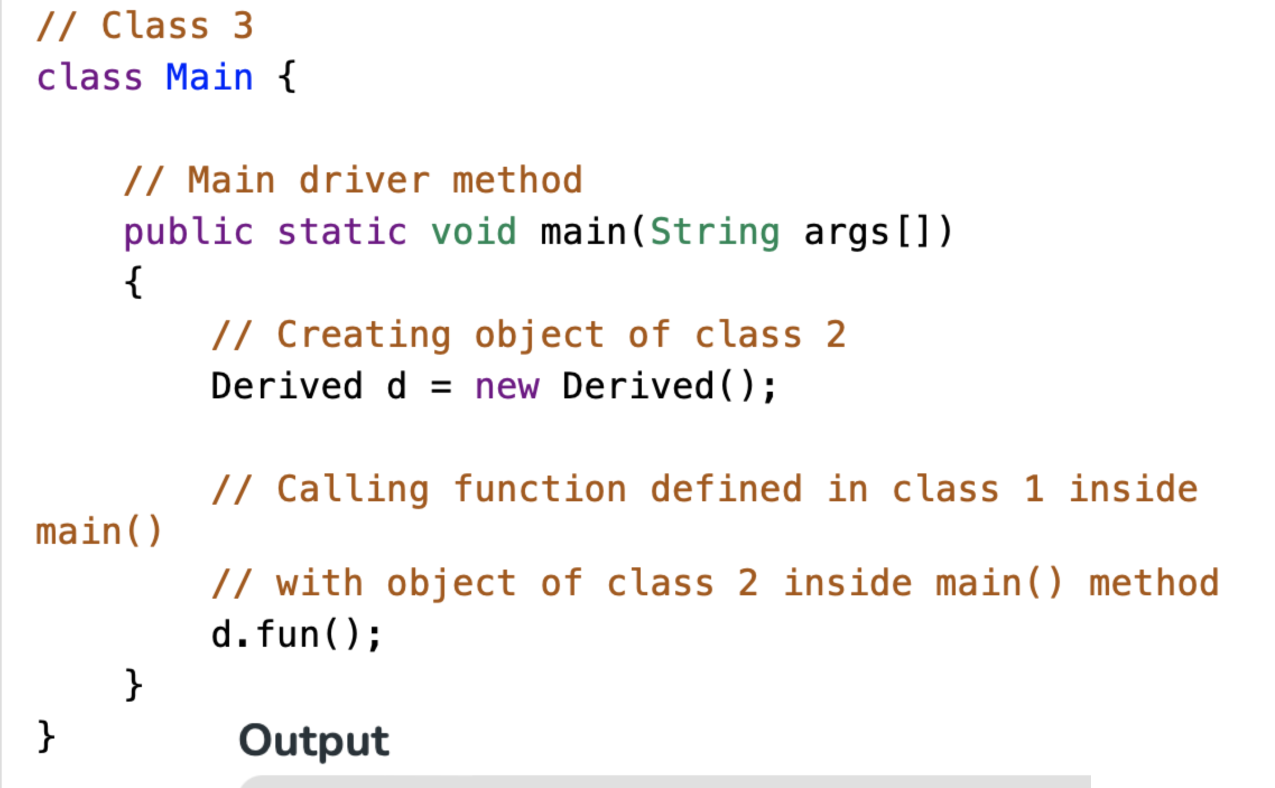
```
Learning Subject
C , Java , C++
Preparing Right Now!
```

3. Abstract class without abstract methods

In Java, we can have **an abstract class without any abstract methods**. This allows us to **create classes that cannot be instantiated but can only be inherited.**

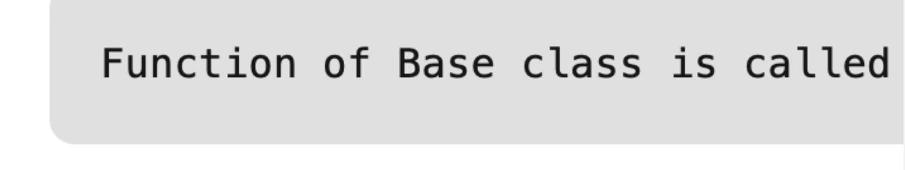


```java
// Class 1
// An abstract class without any abstract method
abstract class Base {

    // Demo method. This is not an abstract method.
    void fun()
    {
        // Print message if class 1 function is called
        System.out.println(
            "Function of Base class is called");
    }
}

// Class 2
class Derived extends Base {
    // This class only inherits the Base class methods and
    // properties
}
```



```java
// Class 3
class Main {

    // Main driver method
    public static void main(String args[])
    {
        // Creating object of class 2
        Derived d = new Derived();

        // Calling function defined in class 1 inside
main()
        // with object of class 2 inside main() method
        d.fun();
    }
}
```

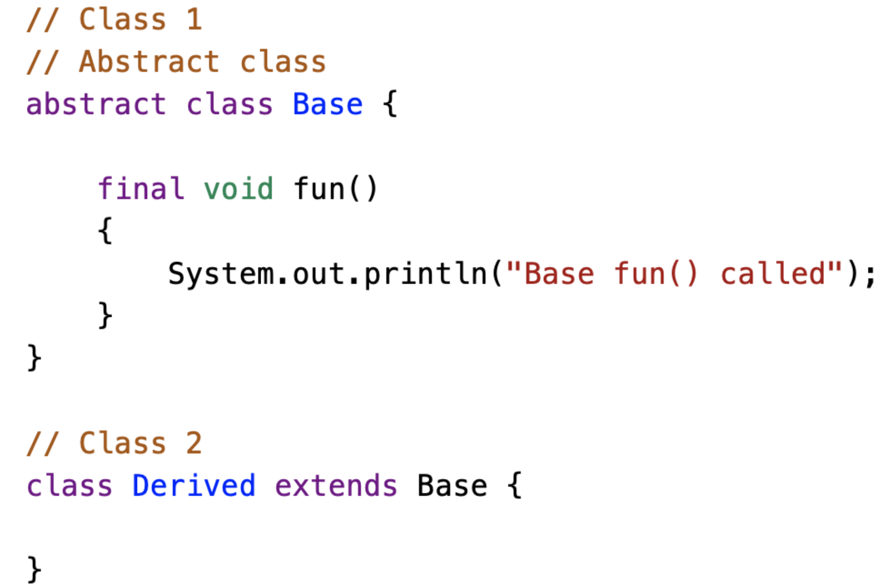
Output



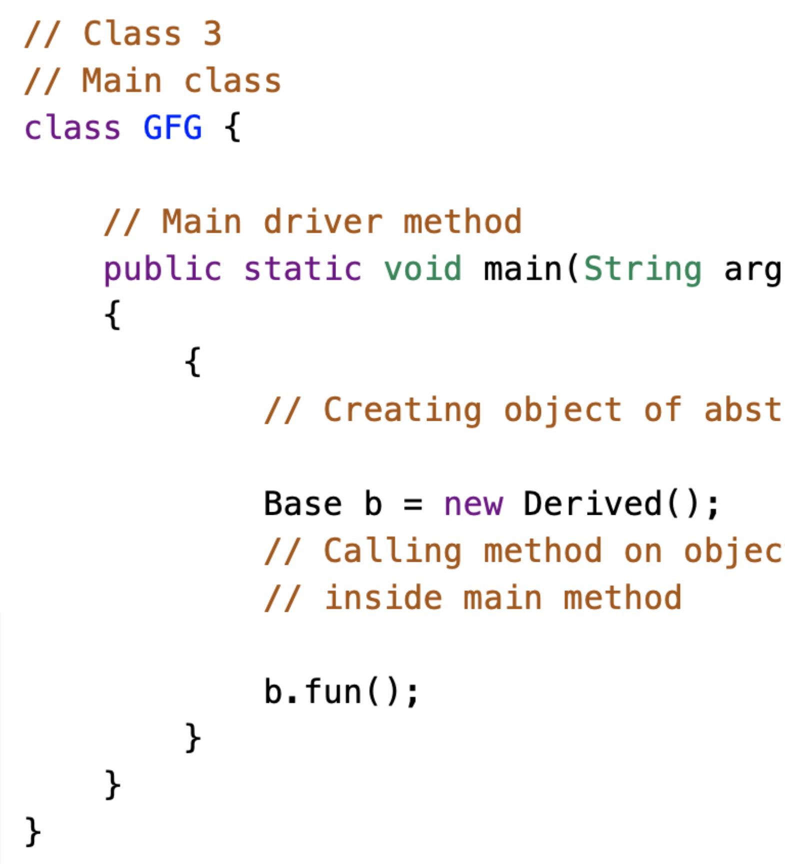
```
Function of Base class is called
```

4. Abstract class with **final** methods

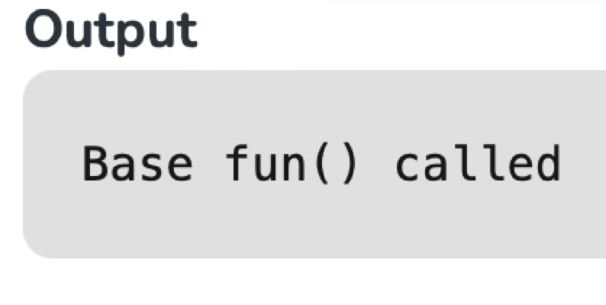
Abstract classes can also have final methods (methods that cannot be overridden).



```
// Class 1
// Abstract class
abstract class Base {

    final void fun()
    {
        System.out.println("Base fun() called");
    }
}

// Class 2
class Derived extends Base {

}
```



```
// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        {
            // Creating object of abstract class

            Base b = new Derived();
            // Calling method on object created above
            // inside main method

            b.fun();
        }
    }
}
```



Output

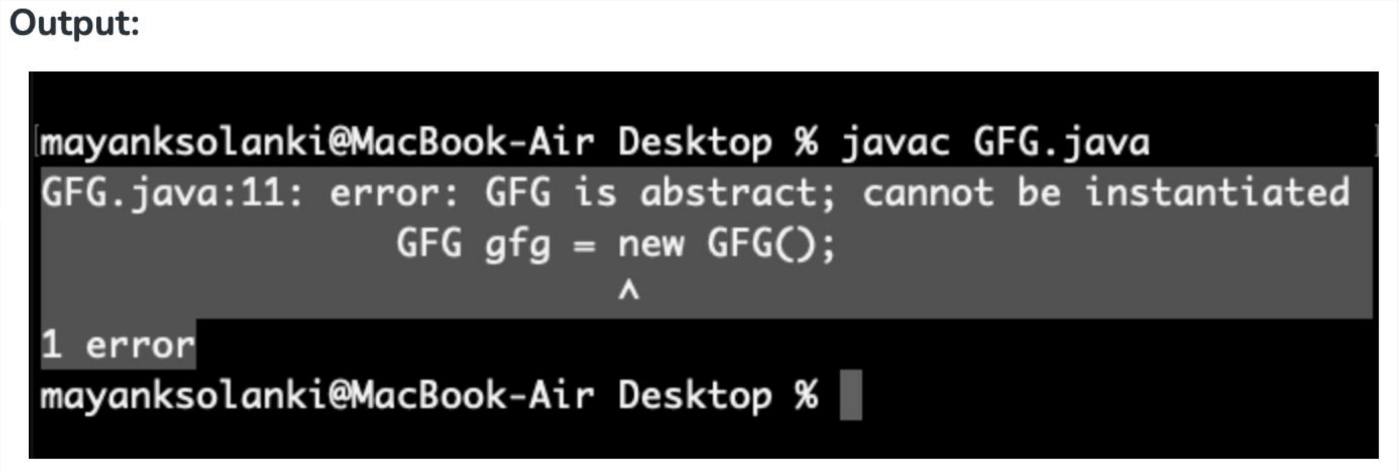
```
Base fun() called
```

5.Abstract class cannot be instantiated.

```java
// Java Program to Illustrate Abstract Class

// Main class
// An abstract class
abstract class GFG {

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

        // Trying to create an object
        GFG gfg = new GFG();
    }
}
```

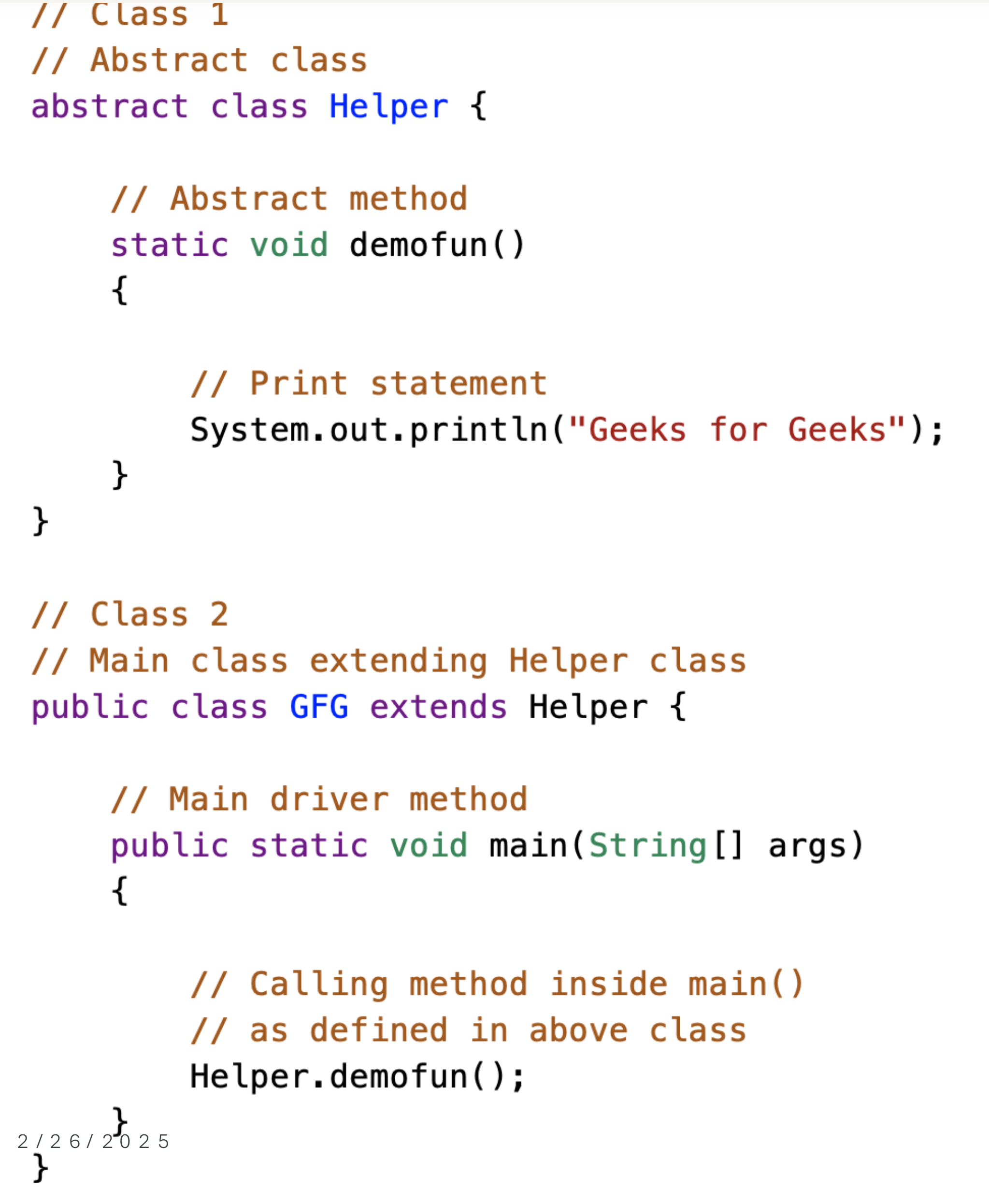
For any abstract java classes we are not allowed to create an object i.e., for an abstract class instantiation is not possible.

Output:



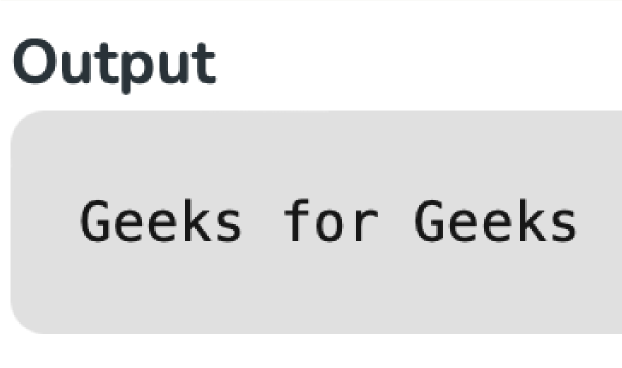
```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
GFG.java:11: error: GFG is abstract; cannot be instantiated
                GFG gfg = new GFG();
                          ^
1 error
mayanksolanki@MacBook-Air Desktop %
```

6. Abstract class with static method



```
// Class 1
// Abstract class
abstract class Helper {

    // Abstract method
    static void demofun()
    {

        // Print statement
        System.out.println("Geeks for Geeks");
    }
}

// Class 2
// Main class extending Helper class
public class GFG extends Helper {

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

        // Calling method inside main()
        // as defined in above class
        Helper.demofun();
    }
}
```

We can define static methods in an abstract class that can be called independently without an object.



Output

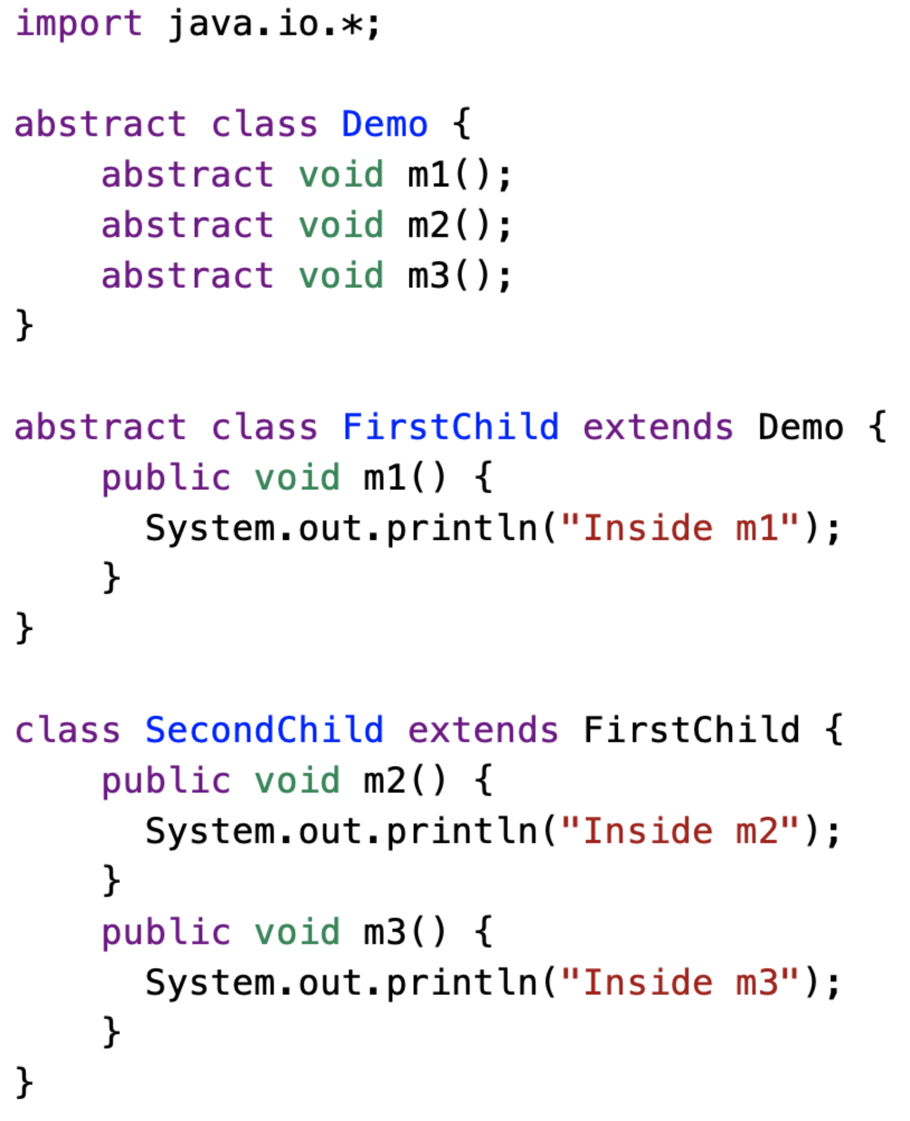
```
Geeks for Geeks
```

7. Over one child class for an abstract class

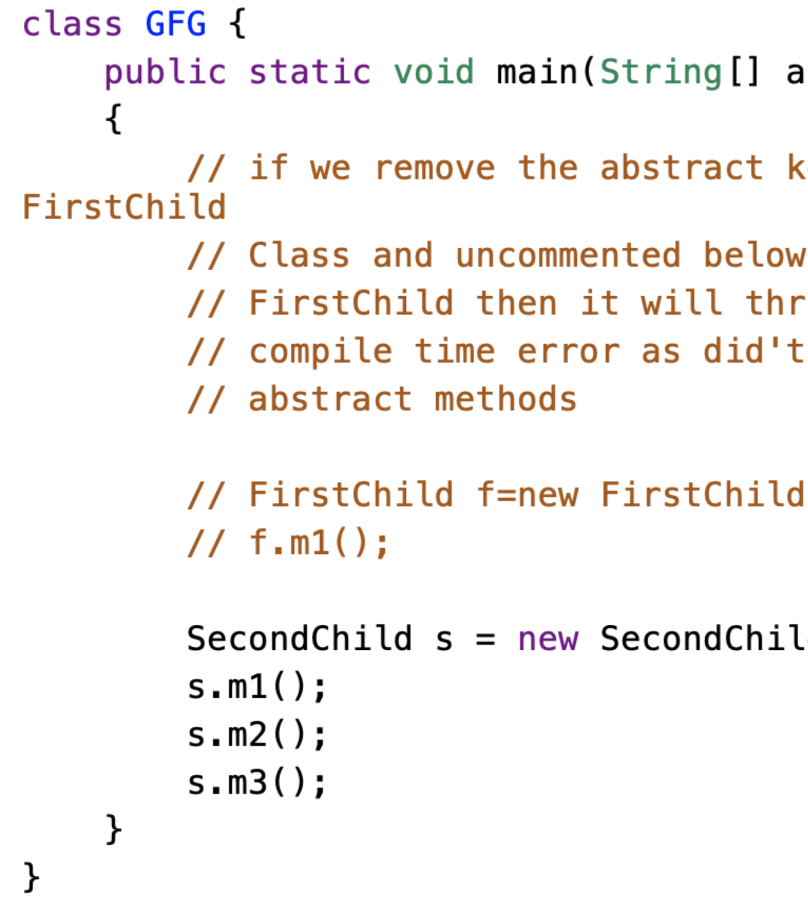
- If a **class contains at least one abstract method** then **compulsory that we should declare the class as abstract** otherwise we will get a compile-time error.

- If the **Child class** is unable to provide implementation to all abstract methods of the **Parent abstract class** then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract methods.

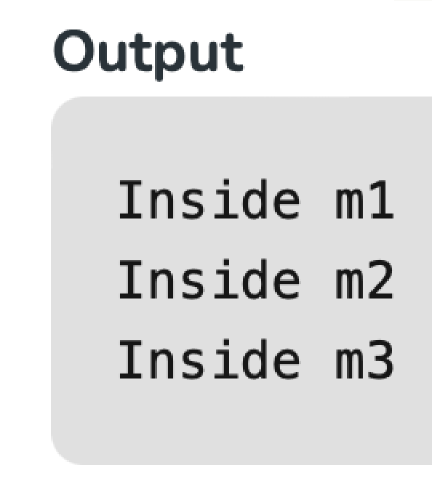
One example



```java
import java.io.*;

abstract class Demo {
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class FirstChild extends Demo {
    public void m1() {
      System.out.println("Inside m1");
    }
}

class SecondChild extends FirstChild {
    public void m2() {
      System.out.println("Inside m2");
    }
    public void m3() {
      System.out.println("Inside m3");
    }
}
```



```java
class GFG {
    public static void main(String[] args)
    {
        // if we remove the abstract keyword from
FirstChild
        // Class and uncommented below obj creation for
        // FirstChild then it will throw
        // compile time error as did't override all the
        // abstract methods

        // FirstChild f=new FirstChild();
        // f.m1();

        SecondChild s = new SecondChild();
        s.m1();
        s.m2();
        s.m3();
    }
}
```



Output

```
Inside m1
Inside m2
Inside m3
```

Some important observations about abstract classes

- 1. An instance of an abstract class cannot be created.
- 2. Constructors are allowed.
- 3. We can have an abstract class without any abstract methods.
- 4. There can be a final method in abstract class but any abstract methods in class (abstract class) cannot be declared as final.
- 5. We can define static methods in an abstract class.

Some important observations about abstract classes

- 6. If a **class** contains at least one abstract method then compulsory should declare a class as abstract
- 7. If the **Child class** is unable to provide implementation to all abstract methods of the **Parent class** then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method.

Summary for abstract class

- An abstract class is a class that cannot be initiated by itself, it needs to be subclassed by another class to use its properties.
- An abstract class can be created using “abstract” keywords.
- We can have an abstract class **without** any abstract methods.

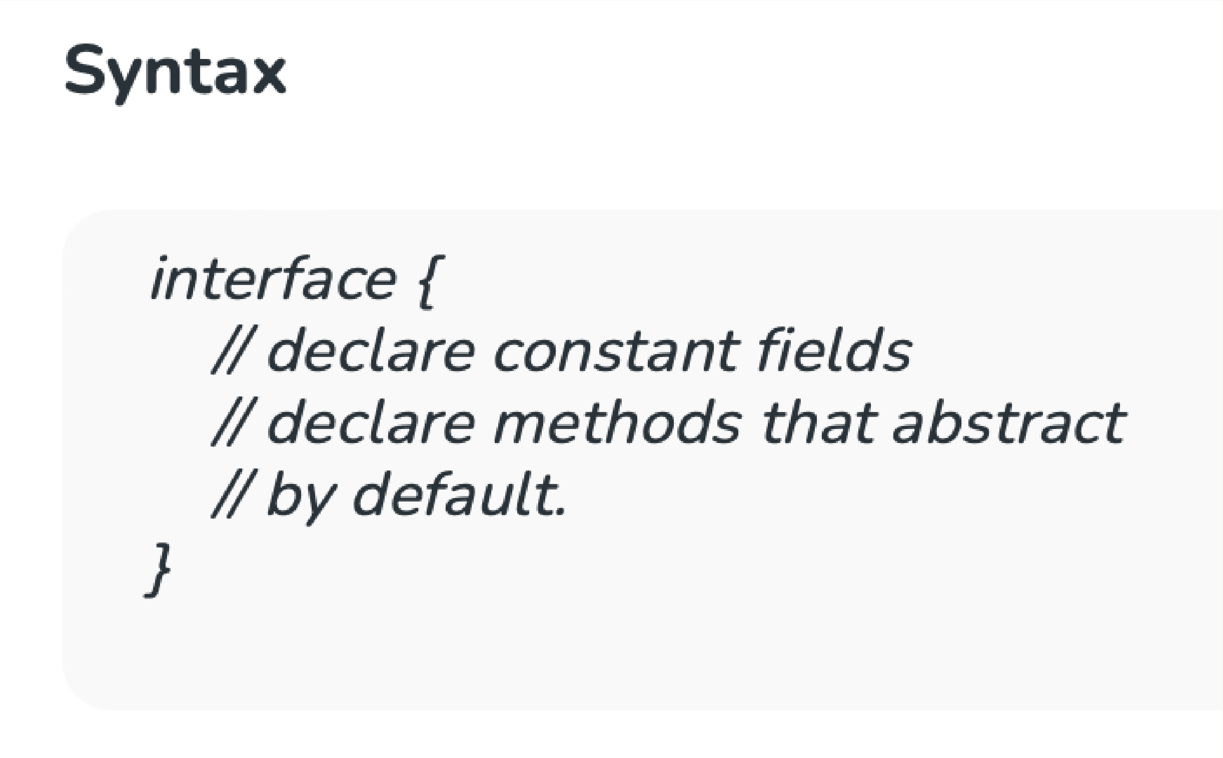
INTERFACE

Java Interface

An interface in Java is a blueprint of a behavior. A Java interface contains static constants and abstract methods.

- The interface in Java is *a* mechanism to achieve **abstraction**.
- By default, variables in an interface are **public, static, and final**.
- Interfaces primarily define methods that other classes must implement.
- Java Interface also represents the IS-A relationship.

Create an interface



Syntax

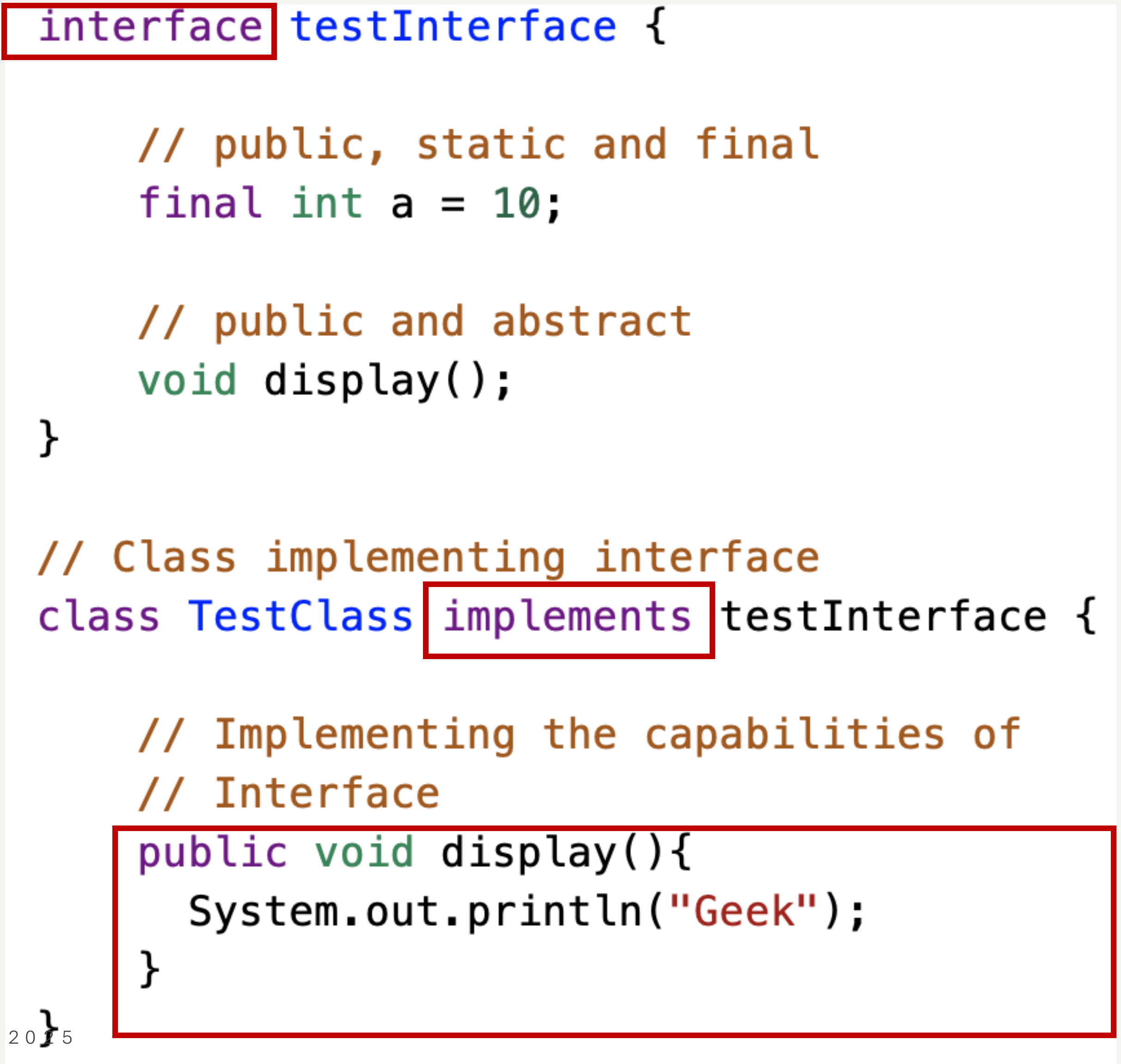
```
interface {
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

- To declare an interface, use the interface keyword. It is used to provide total abstraction.
- All the methods in an interface are declared with an empty body and are public and all fields are public, static, and final by default.

A class implement an interface

- To implement the interface, use the implements keyword.
- A class that implements an interface must implement all the methods declared in the interface.

One example

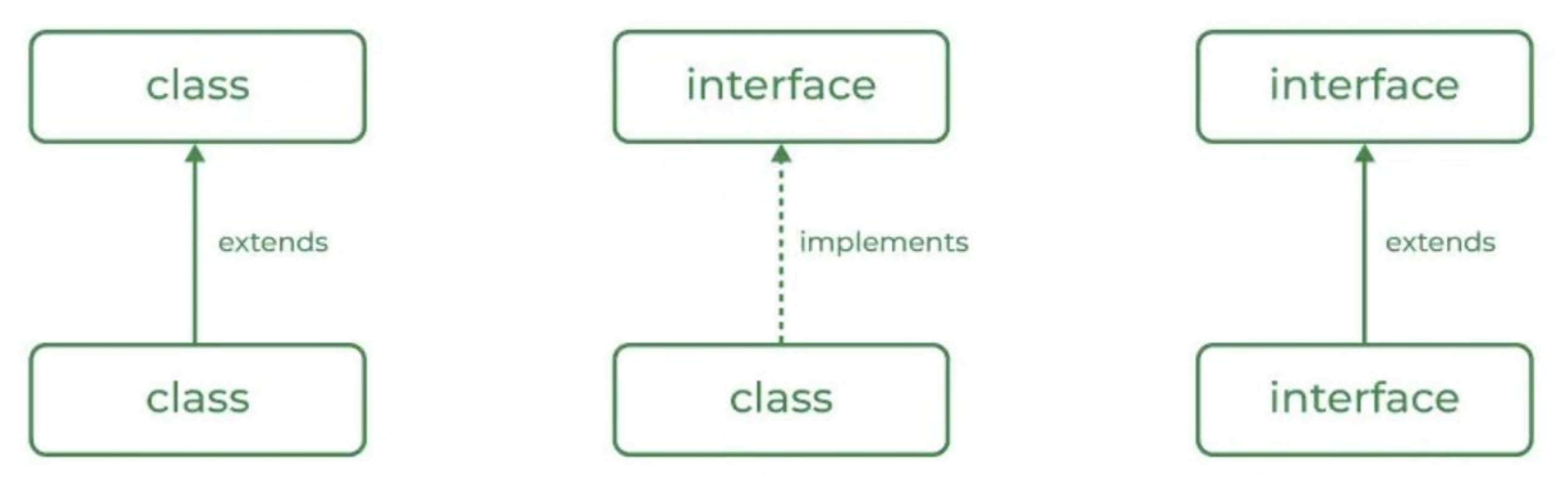


```java
interface testInterface {

    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}

// Class implementing interface
class TestClass implements testInterface {

    // Implementing the capabilities of
    // Interface
    public void display(){
      System.out.println("Geek");
    }
}
```

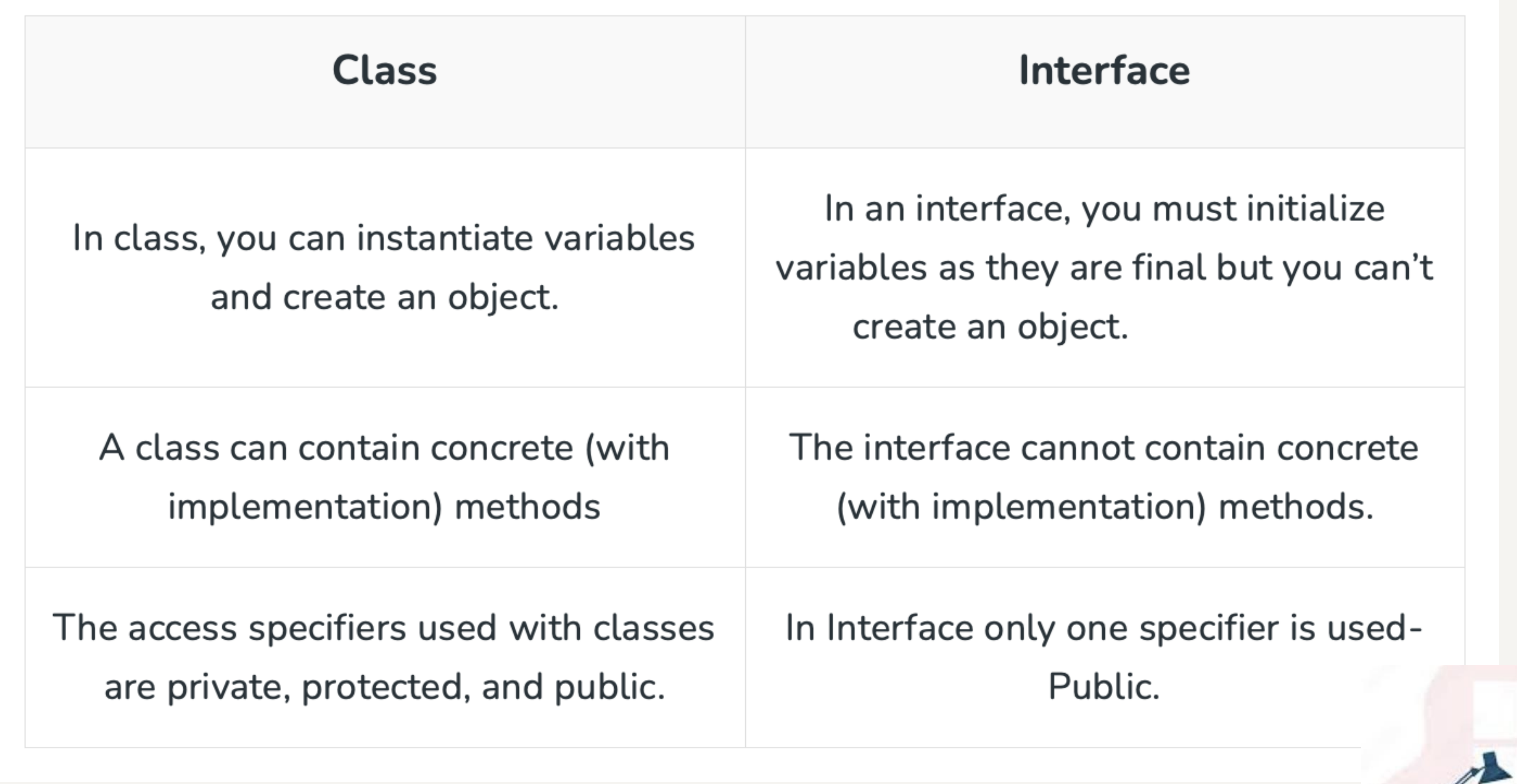
Implementation for the method declared in the interface.

Relationship between Class and Interface

- A class can extend another class, and similarly, an interface can extend another interface.
- However, only a class can implement an interface, and the reverse (an interface implementing a class) is not allowed.



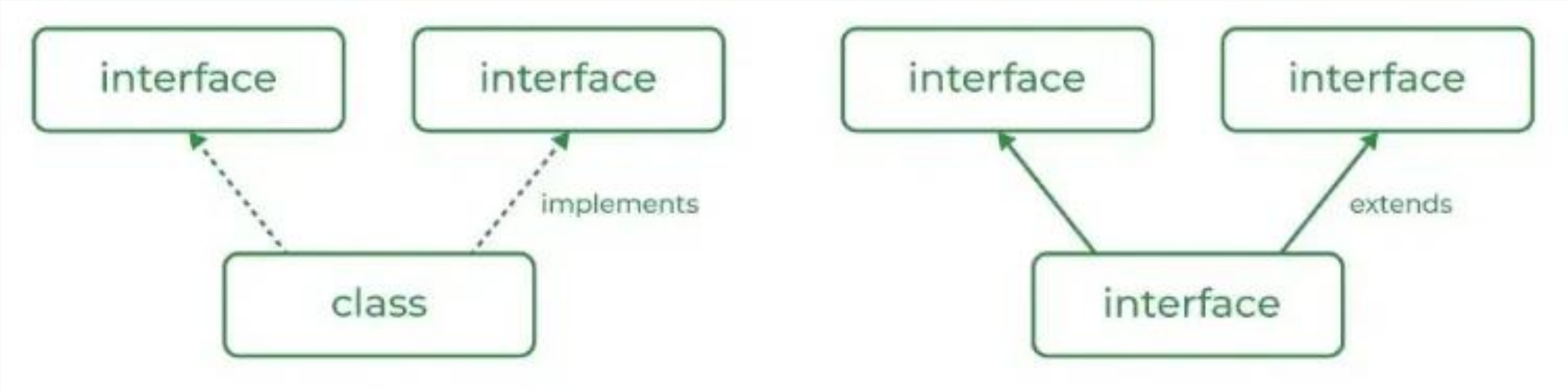
Difference Between Class and Interface



Class	Interface
In class, you can instantiate variables and create an object.	In an interface, you must initialize variables as they are final but you can’t create an object.
A class can contain concrete (with implementation) methods	The interface cannot contain concrete (with implementation) methods.
The access specifiers used with classes are private, protected, and public.	In Interface only one specifier is used- Public.

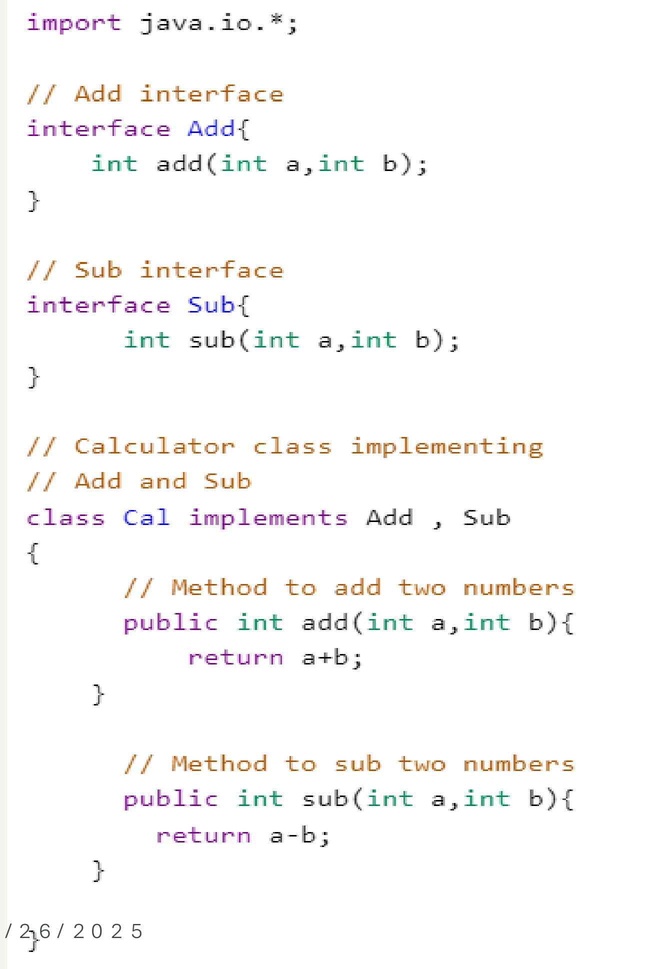
Multiple Inheritance in Java Using Interface

A class could implement multiple interfaces. One interface could inherit from multiple interfaces.

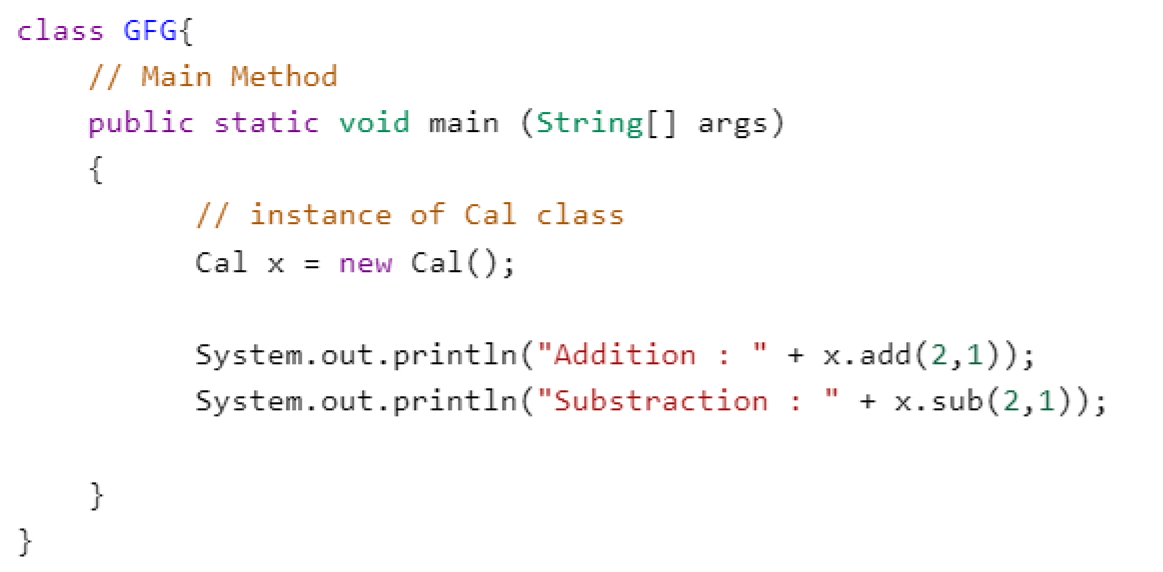


One example

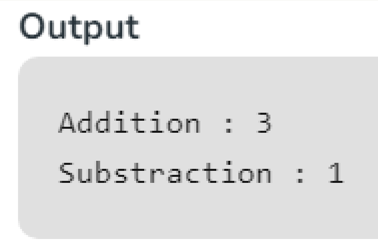
One class implements multiple interfaces



```
import java.io.*;

// Add interface
interface Add{
    int add(int a,int b);
}

// Sub interface
interface Sub{
      int sub(int a,int b);
}

// Calculator class implementing
// Add and Sub
class Cal implements Add , Sub
{
      // Method to add two numbers
      public int add(int a,int b){
          return a+b;
    }

      // Method to sub two numbers
      public int sub(int a,int b){
        return a-b;
    }
}
```



```
class GFG{
    // Main Method
    public static void main (String[] args)
    {
          // instance of Cal class
          Cal x = new Cal();

          System.out.println("Addition : " + x.add(2,1));
          System.out.println("Substraction : " + x.sub(2,1));

    }
}
```



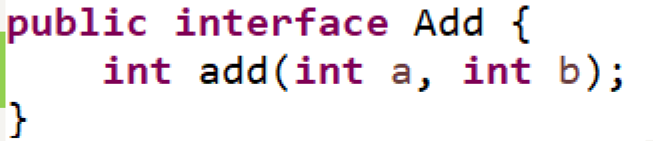
Output

```
Addition : 3
Substraction : 1
```

Another Example

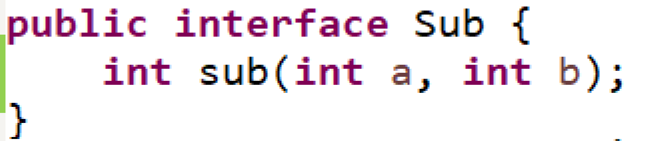
One interface extends multiple interfaces.

1



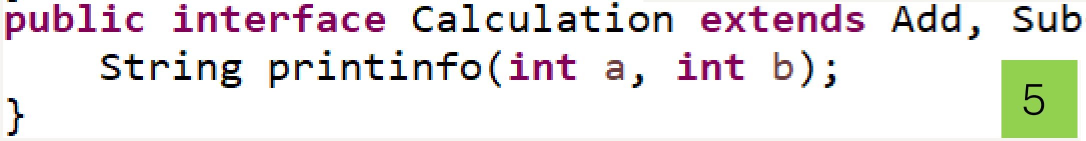
```java
public interface Add {
     int add(int a, int b);
}
```

2



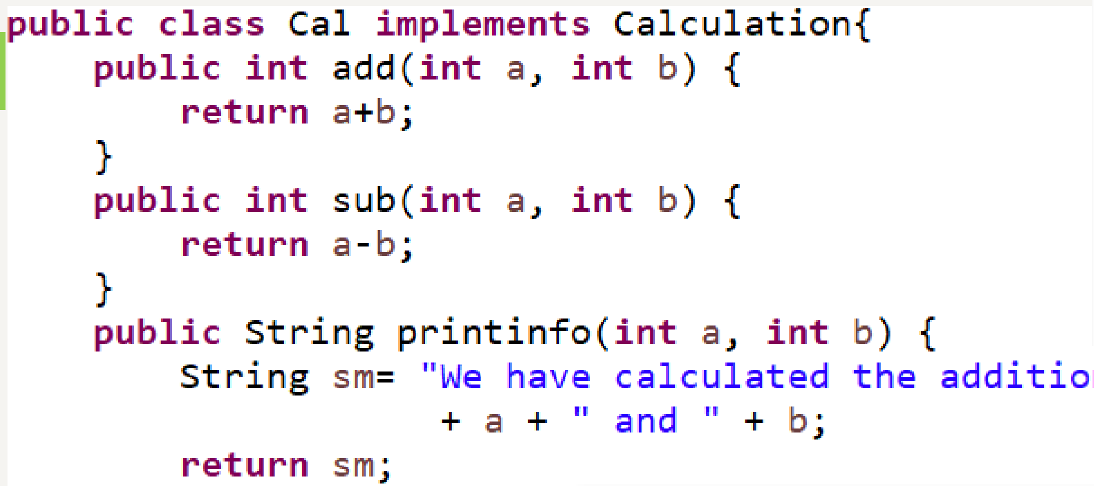
```java
public interface Sub {
     int sub(int a, int b);
}
```

3



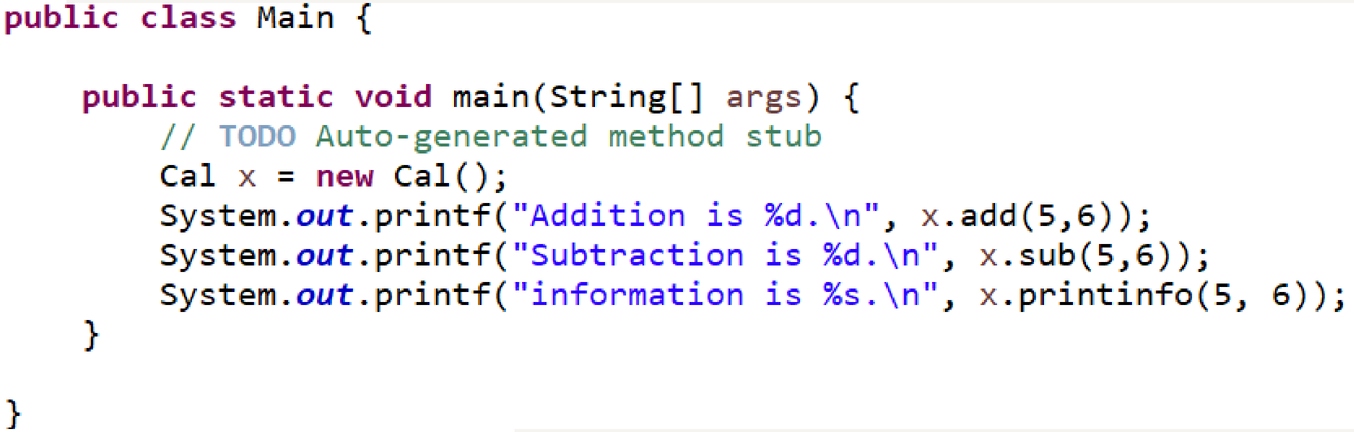
```java
public interface Calculation extends Add, Sub{
     String printinfo(int a, int b);
}
```

4



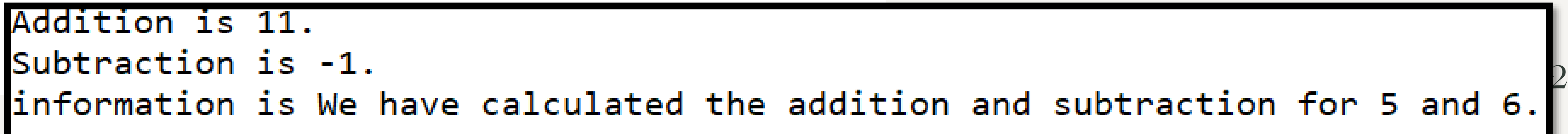
```java
public class Cal implements Calculation{
    public int add(int a, int b) {
        return a+b;
    }
    public int sub(int a, int b) {
        return a-b;
    }
    public String printinfo(int a, int b) {
        String sm= "We have calculated the addition and subtraction for "
                    + a + " and " + b;
        return sm;
    }
 }
```

5



```java
public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Cal x = new Cal();
        System.out.printf("Addition is %d.\n", x.add(5,6));
        System.out.printf("Subtraction is %d.\n", x.sub(5,6));
        System.out.printf("information is %s.\n", x.printinfo(5, 6));
    }

}
```

Output

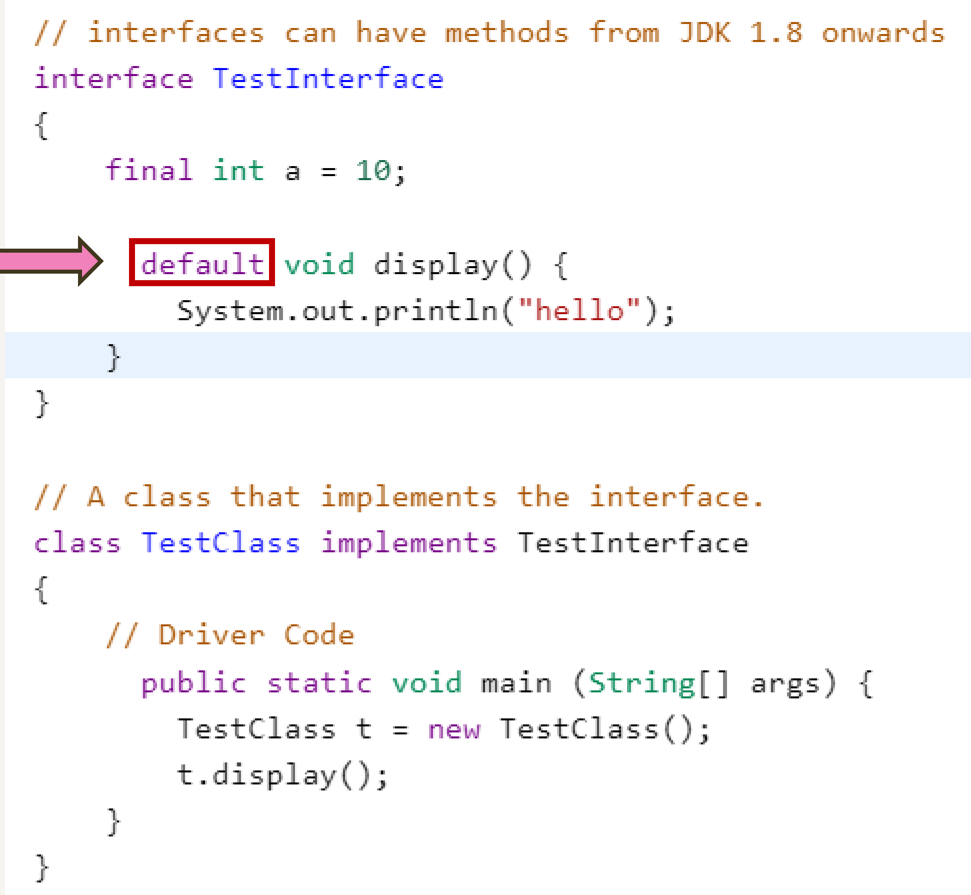


```
Addition is 11.
Subtraction is -1.
information is We have calculated the addition and subtraction for 5 and 6.
```

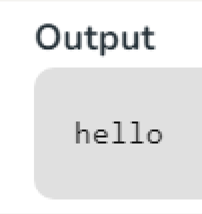
Default Methods for an interface

- Interfaces can define methods with **default** implementations. "default" is a keyword in here.

Using **default** keyword, we could implement the methods in the interface.



```
// interfaces can have methods from JDK 1.8 onwards
interface TestInterface
{
    final int a = 10;

    default void display() {
      System.out.println("hello");
    }
}

// A class that implements the interface.
class TestClass implements TestInterface
{
    // Driver Code
      public static void main (String[] args) {
        TestClass t = new TestClass();
        t.display();
    }
}
```

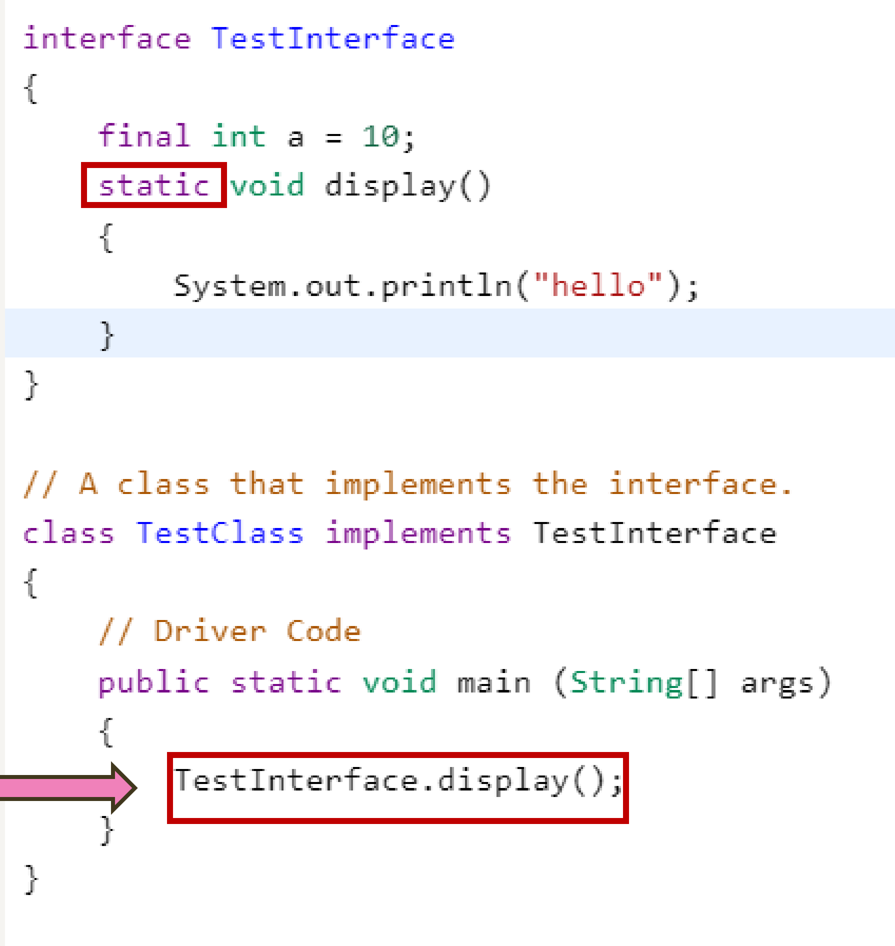


Output

```
hello
```

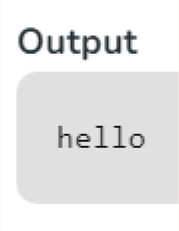
Static Methods for an interface

- Interfaces can include static methods.
- These methods are called directly using the interface name and are not inherited by implementing classes.



```
interface TestInterface
{
    final int a = 10;
    static void display()
    {
        System.out.println("hello");
    }
}

// A class that implements the interface.
class TestClass implements TestInterface
{
    // Driver Code
    public static void main (String[] args)
    {
        TestInterface.display();
    }
}
```

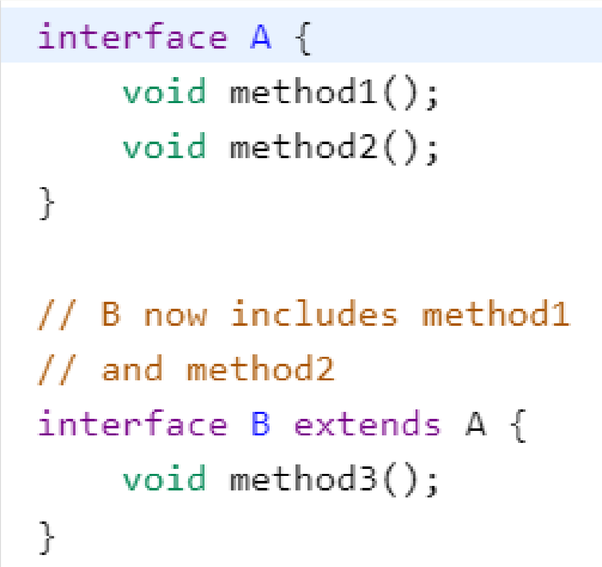
Using static keyword, we could implement the method in the interface and call the method via the interface directly.



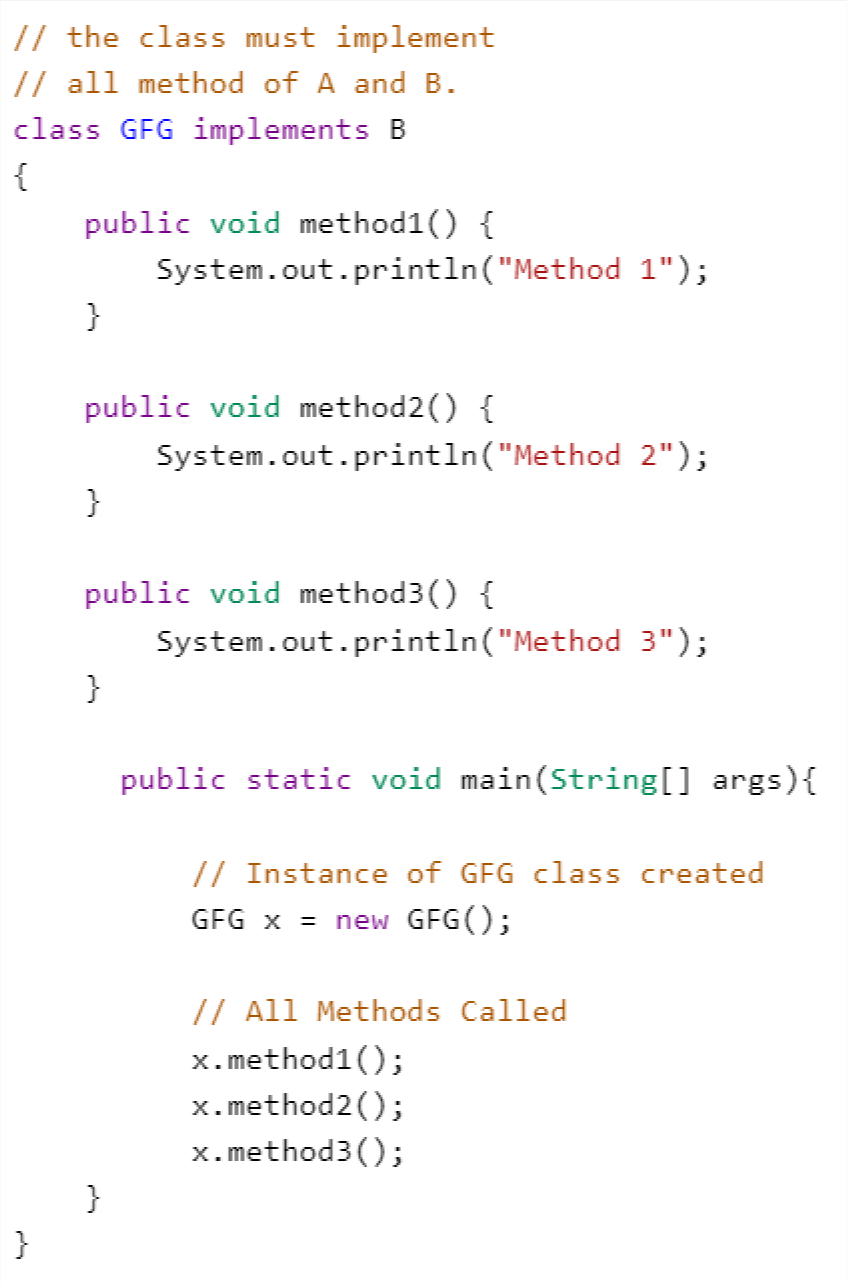
Output

```
hello
```

Extending Interfaces

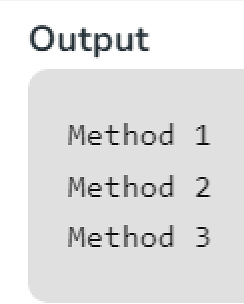


```
interface A {
    void method1();
    void method2();
}

// B now includes method1
// and method2
interface B extends A {
    void method3();
}
```



```
// the class must implement
// all method of A and B.
class GFG implements B
{
    public void method1() {
        System.out.println("Method 1");
    }

    public void method2() {
        System.out.println("Method 2");
    }

    public void method3() {
        System.out.println("Method 3");
    }

      public static void main(String[] args){

          // Instance of GFG class created
          GFG x = new GFG();

          // All Methods Called
          x.method1();
          x.method2();
          x.method3();
    }
}
```

When a class implements an interface that inherits another interface, it must provide an implementation for all methods required by the interface inheritance chain.



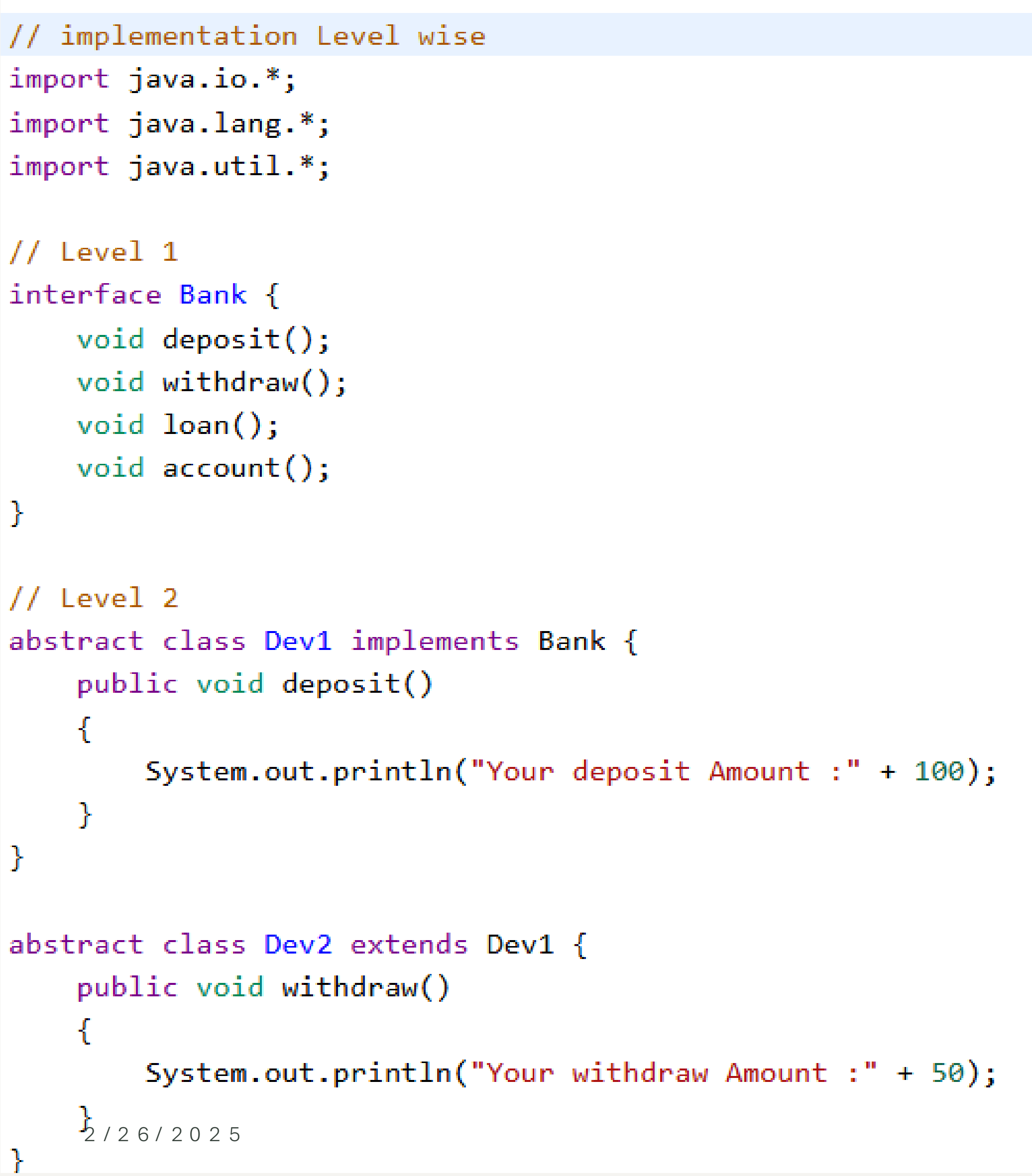
Output

```
Method 1
Method 2
Method 3
```

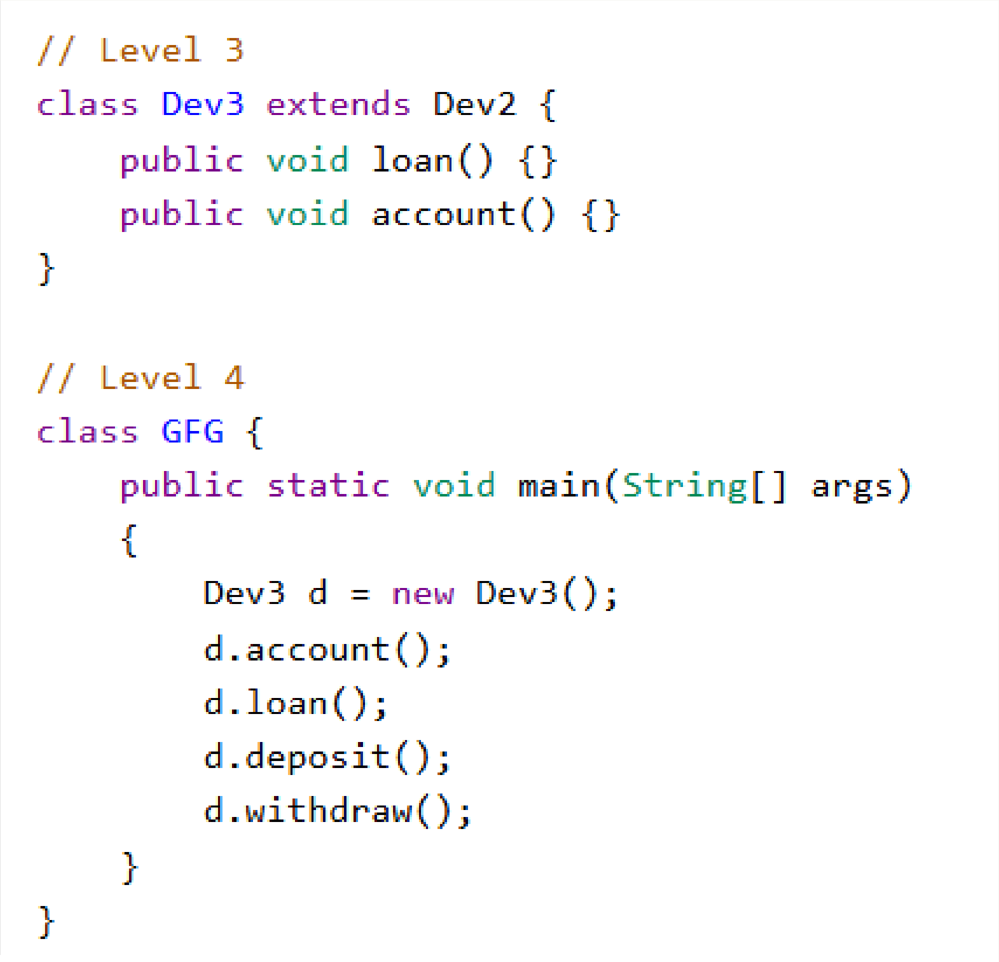
Pay attention for using interfaces

- The interface contains multiple abstract methods(but no usage of **abstract** keyword for a method), so write the implementation in implementation classes.
- If the implementation is unable to provide an implementation of all abstract methods, then declare the implementation class with an abstract modifier, and complete the remaining method implementation in the next created child classes.
  - It is possible to declare multiple child classes but at final we have completed the implementation of all abstract methods.

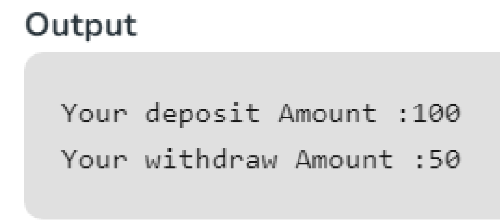
One Example



```java
// implementation Level wise
import java.io.*;
import java.lang.*;
import java.util.*;

// Level 1
interface Bank {
    void deposit();
    void withdraw();
    void loan();
    void account();
}

// Level 2
abstract class Dev1 implements Bank {
    public void deposit()
    {
        System.out.println("Your deposit Amount :" + 100);
    }
}

abstract class Dev2 extends Dev1 {
    public void withdraw()
    {
        System.out.println("Your withdraw Amount :" + 50);
    }
}
```



```java
// Level 3
class Dev3 extends Dev2 {
    public void loan() {}
    public void account() {}
}

// Level 4
class GFG {
    public static void main(String[] args)
    {
        Dev3 d = new Dev3();
        d.account();
        d.loan();
        d.deposit();
        d.withdraw();
    }
}
```

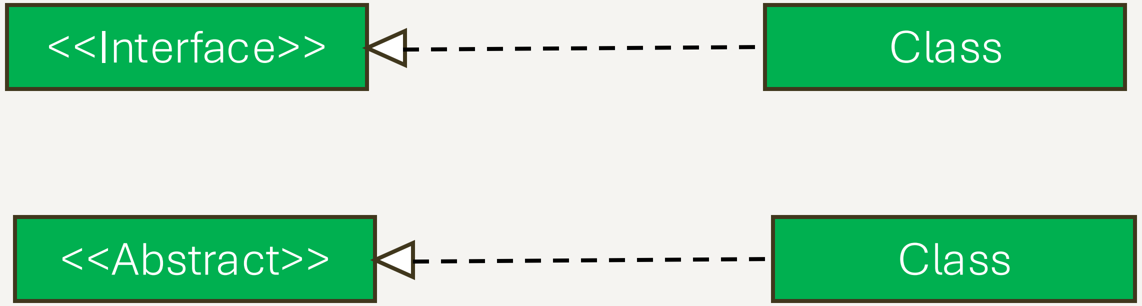


Output

```
Your deposit Amount :100
Your withdraw Amount :50
```

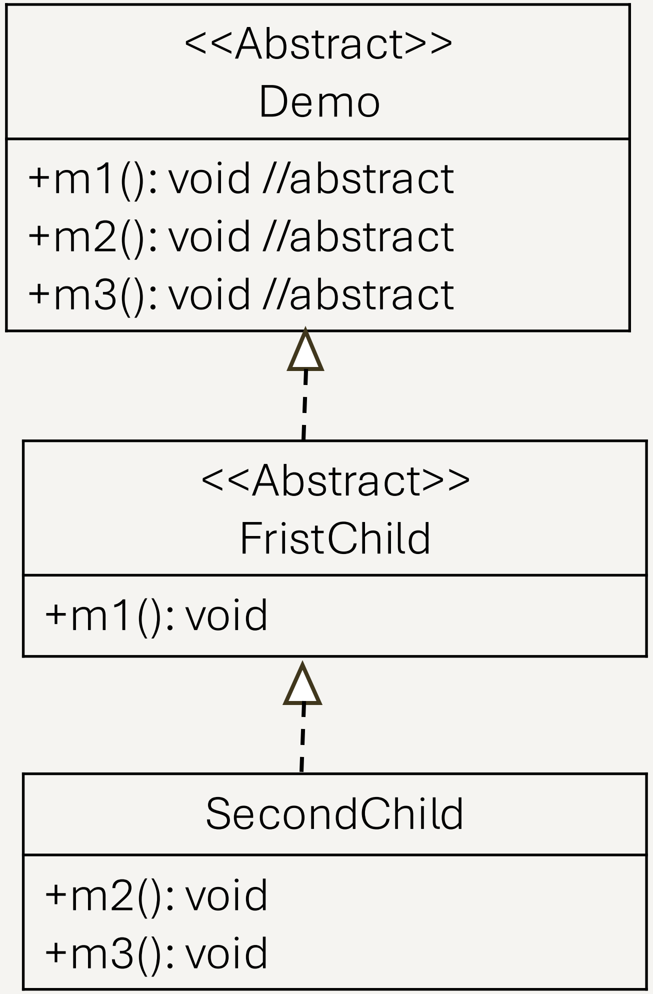
UML FOR ABSTRACT & INTERFACE

Usage of connection



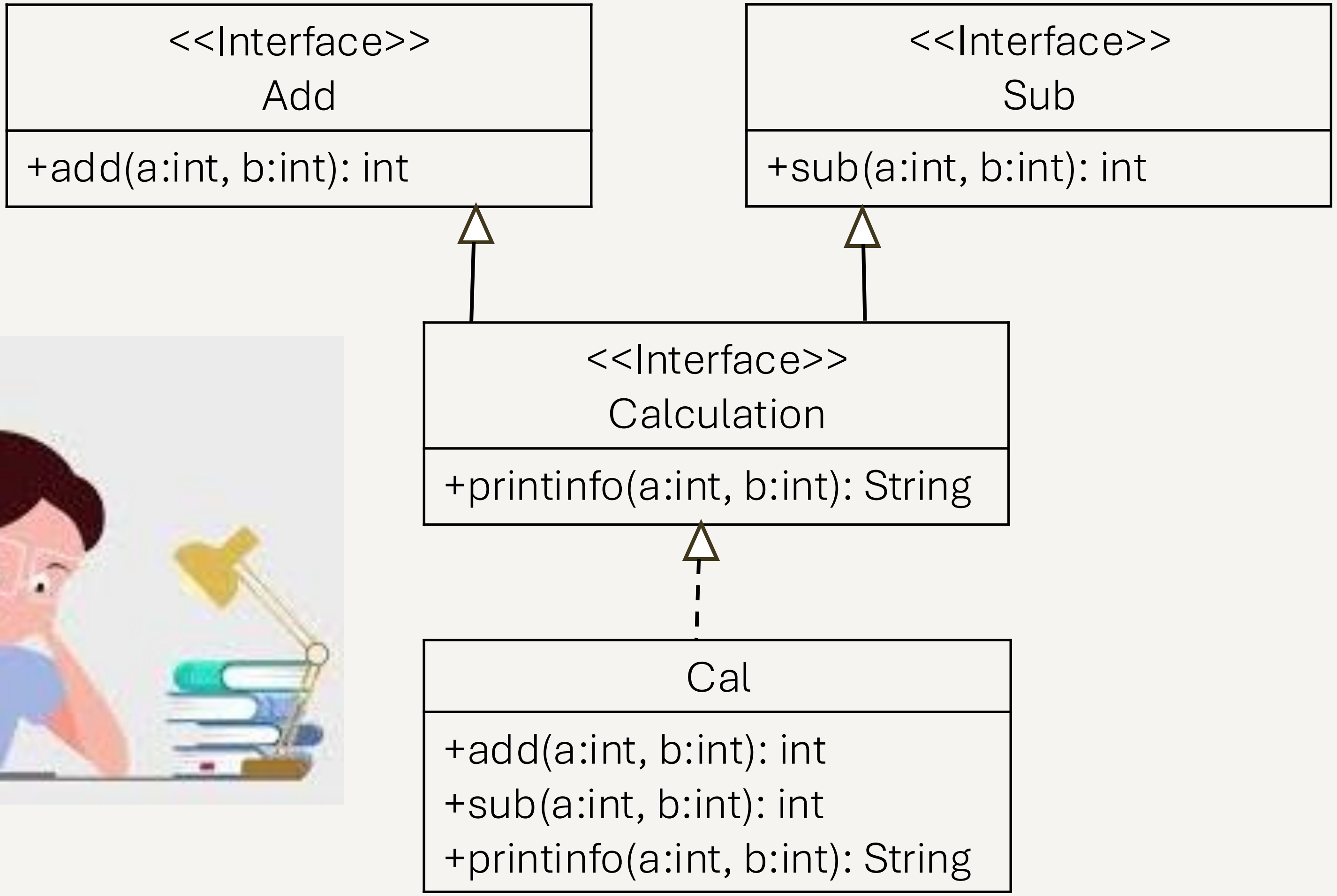
UML for abstract class and its implemented class

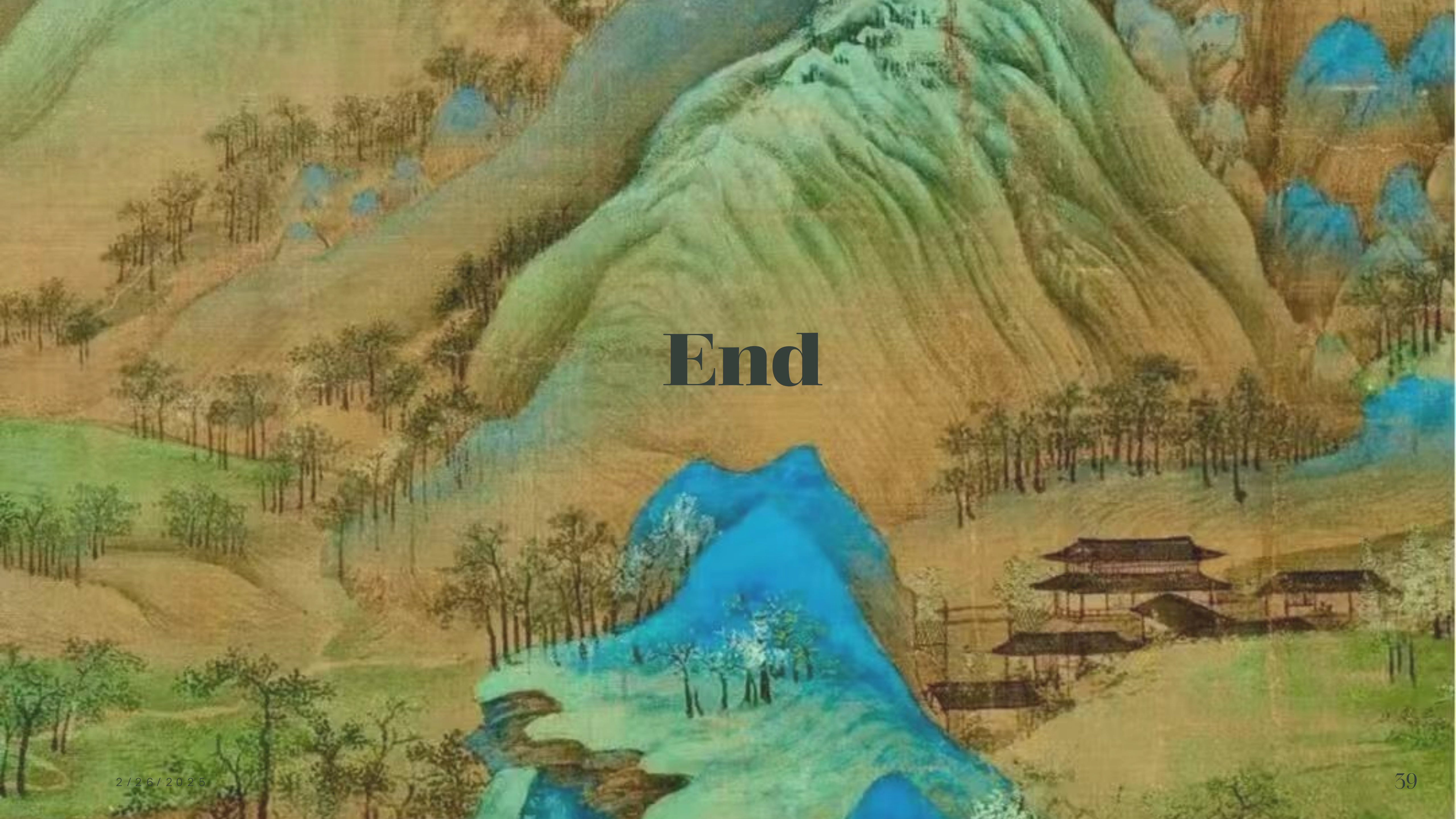
For code in the left panel of P16.



UML for interface class and its child class

For code in P29, 1-4





End