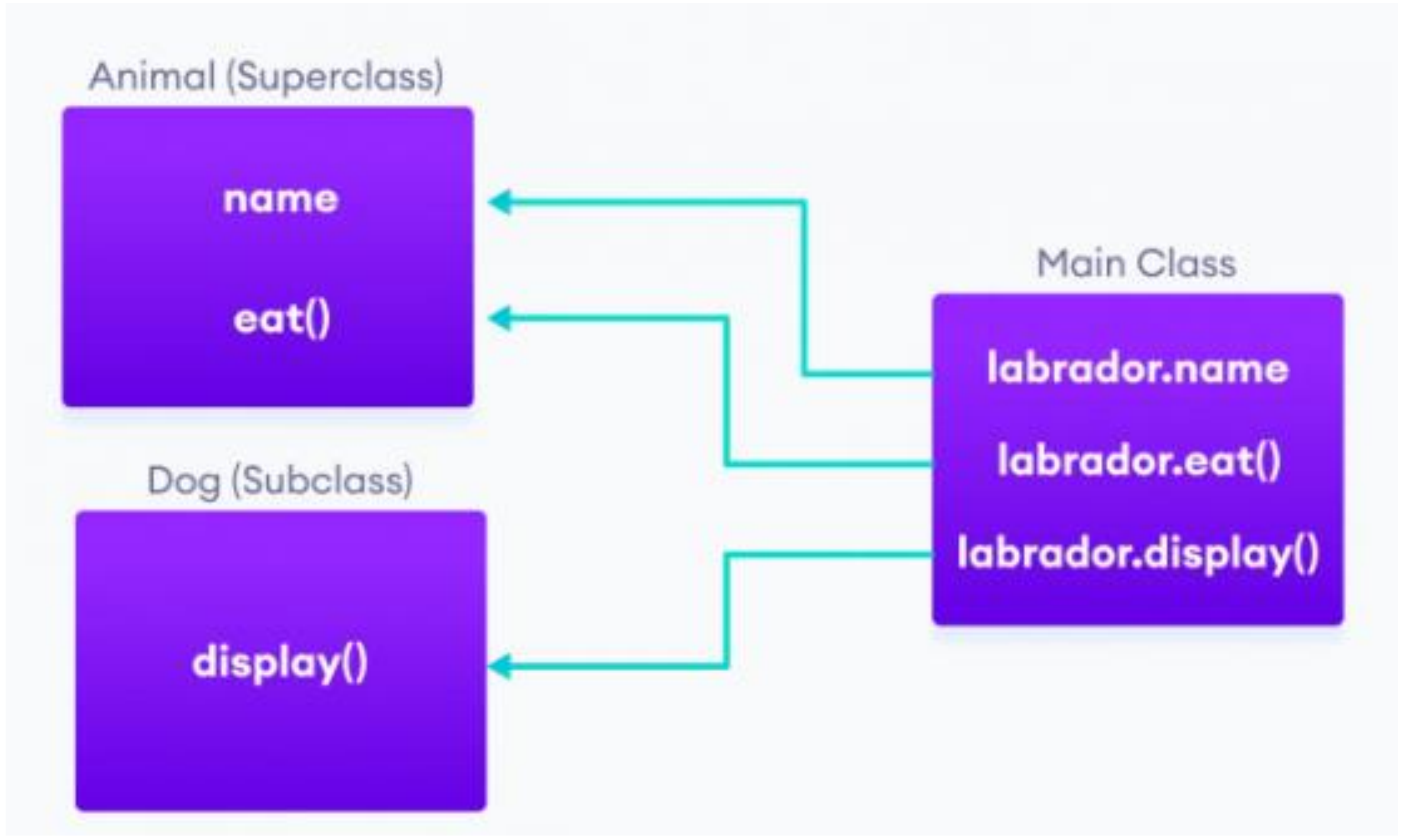
Module 2: Inheritance

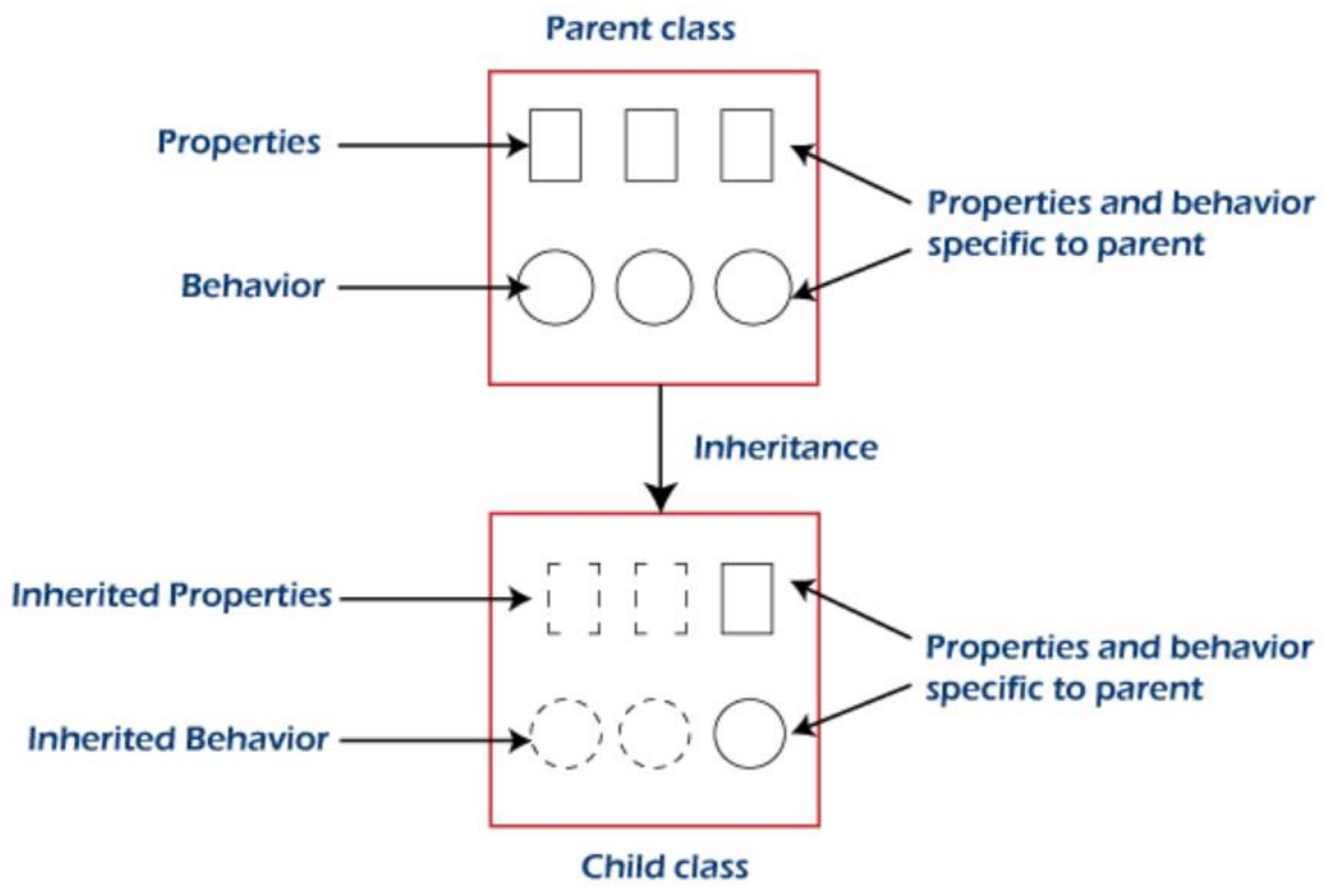
Prof. Pramod Bide

Concept of Inheritance

- Inheritance is one of the key features of OOP that allows us to create a new class from an existing class.
- The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).
- The extends keyword is used to perform inheritance in Java.



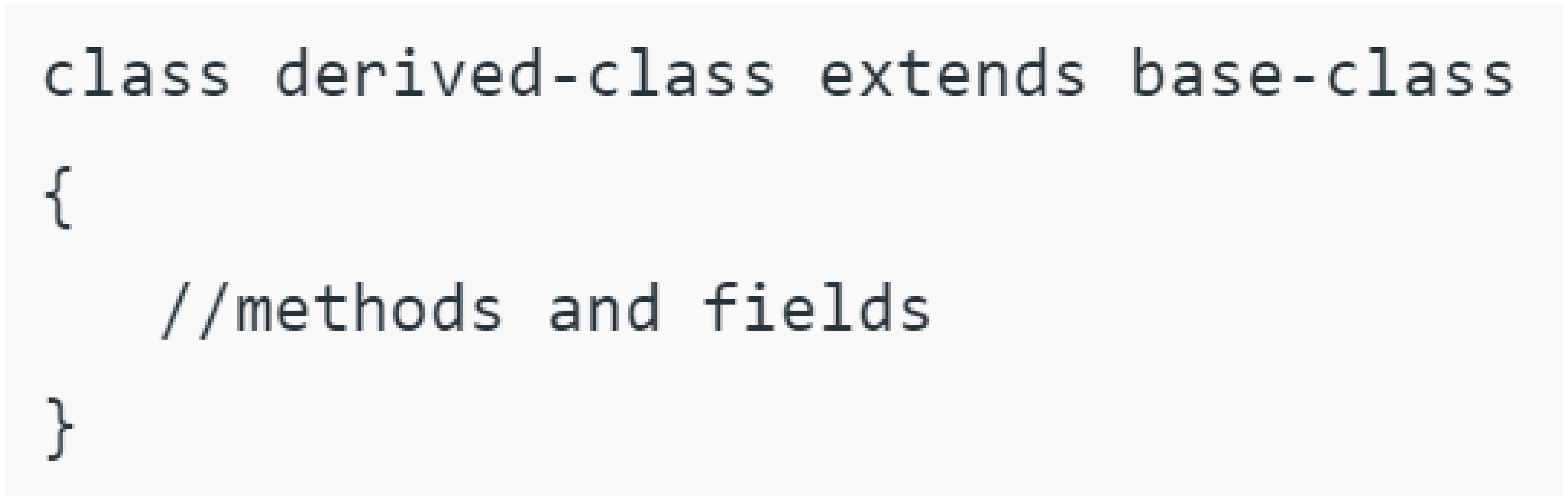
Concept of Inheritance



How to Use Inheritance in Java?

- The extends keyword is used for inheritance in Java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality.

Syntax :



```
class derived-class extends base-class
{
   //methods and fields
}
```

Is-a relationship

In Java, inheritance is an is-a relationship. That is, we use inheritance only if there exists an is-a relationship between two classes. For example,

Car is a Vehicle
Orange is a Fruit
Surgeon is a Doctor
Dog is an Animal

Here, Car can inherit from Vehicle, Orange can inherit from Fruit, and so on.

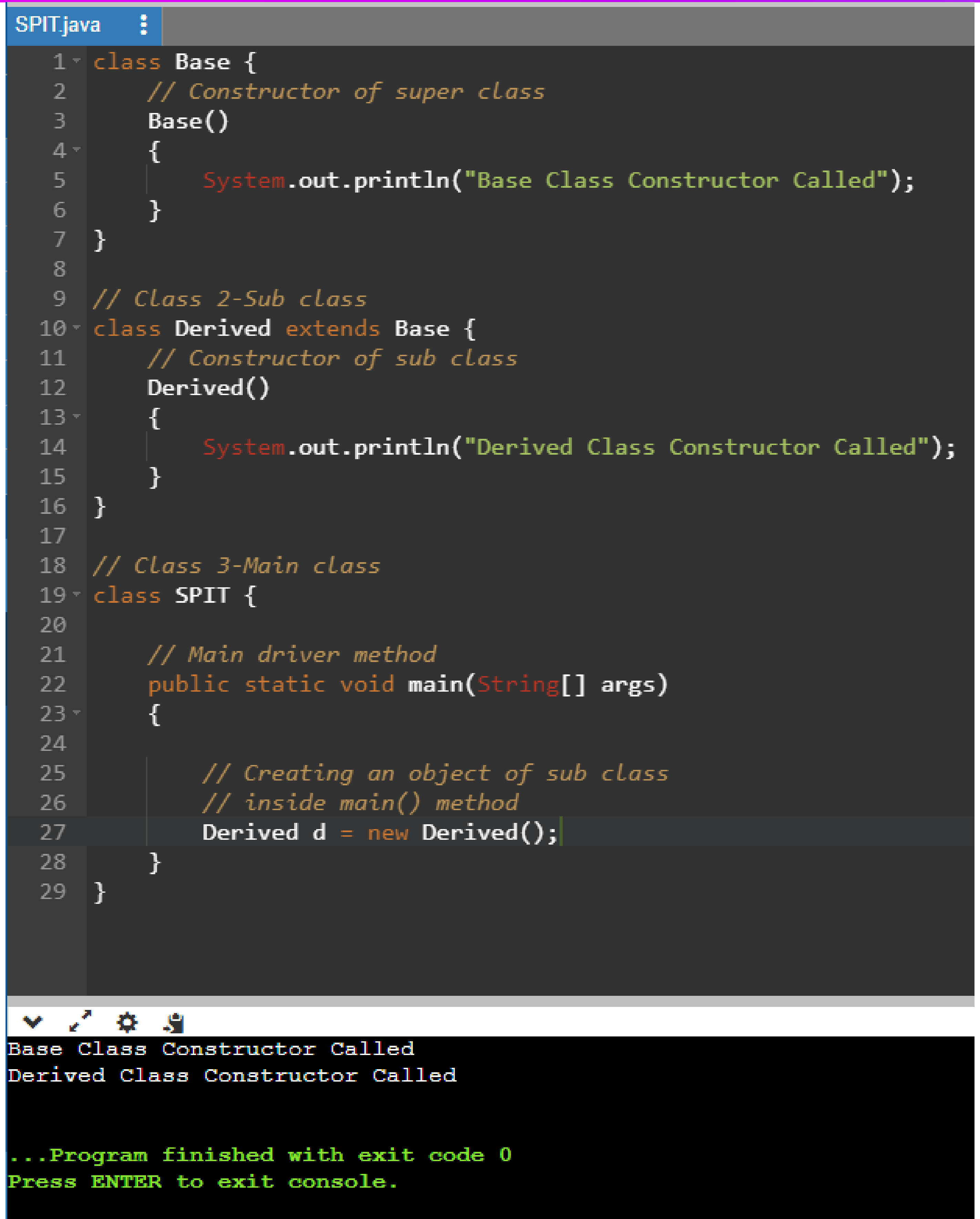
Need for Inheritance

- Code Reusability: The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- Method Overriding: Method Overriding is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.
- Abstraction: The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

Constructor Inheritance

- Constructors in Java are used to initialize the values of the attributes of the object serving the goal to bring Java closer to the real world.
- We already have a default constructor that is called automatically if no constructor is found in the code.
- But if we make any constructor say parameterized constructor in order to initialize some attributes then it must write down the default constructor because it now will be no more automatically called.
- NOTE: In Java, constructor of the base class with no argument gets automatically called in the derived class constructor.

Constructor Inheritance Example



SPIT.java

```java
class Base {
    // Constructor of super class
    Base()
    {
        System.out.println("Base Class Constructor Called");
    }
}

// Class 2-Sub class
class Derived extends Base {
    // Constructor of sub class
    Derived()
    {
        System.out.println("Derived Class Constructor Called");
    }
}

// Class 3-Main class
class SPIT {

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

        // Creating an object of sub class
        // inside main() method
        Derived d = new Derived();
    }
}
```

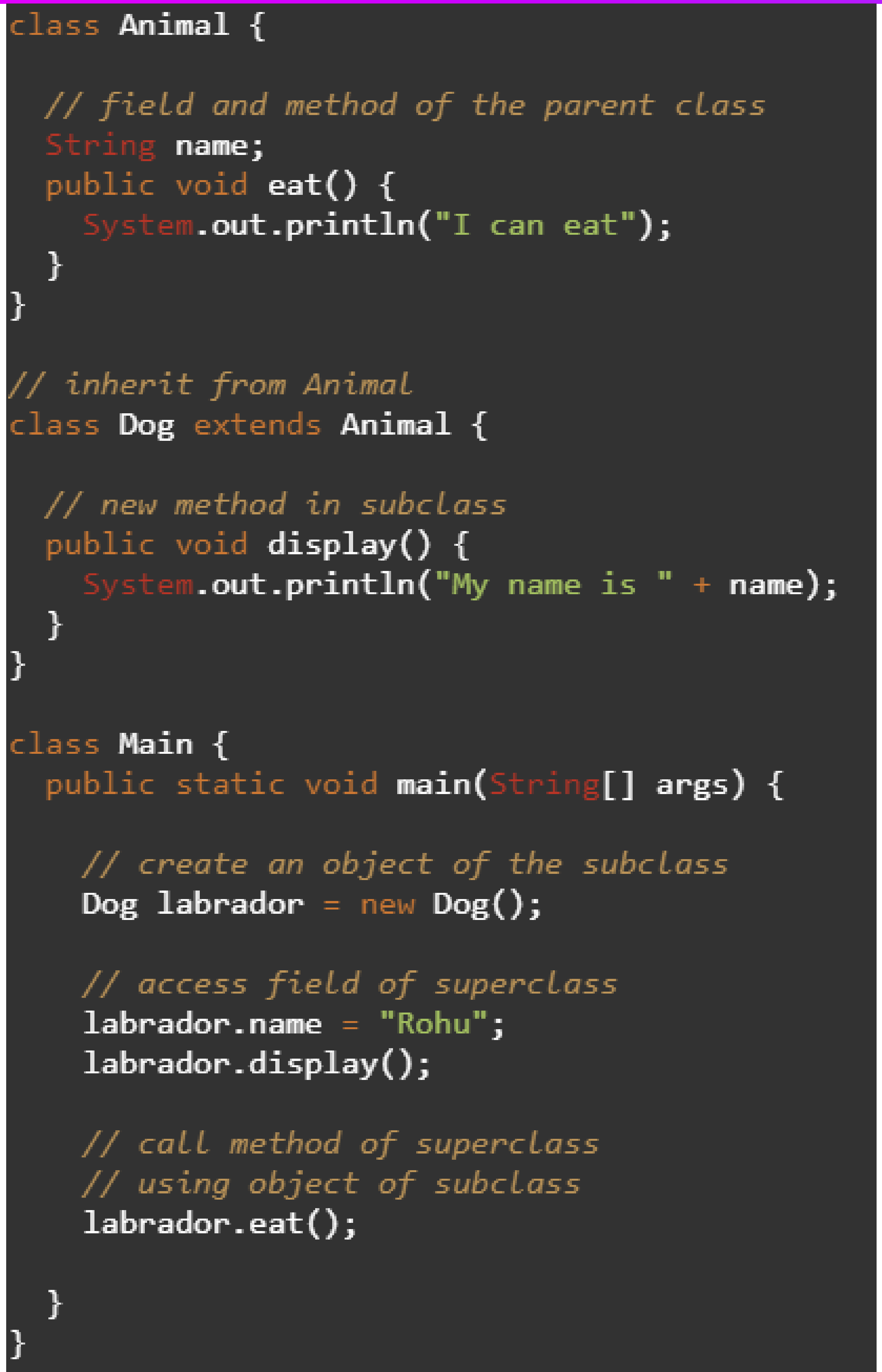
```
Base Class Constructor Called
Derived Class Constructor Called


...Program finished with exit code 0
Press ENTER to exit console.
```

- Here first superclass constructor will be called thereafter derived(sub-class) constructor will be called because the constructor call is from top to bottom.
- If there was any class that our Parent class is extending then the body of that class will be executed thereafter landing up to derived classes.

Terms used in Inheritance

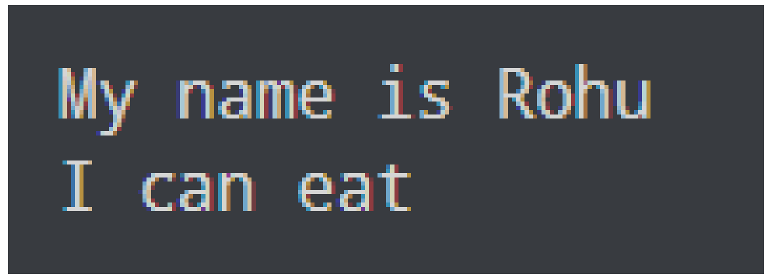
- **Class**: Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- **Super Class/Parent Class**: The class whose features are inherited is known as a superclass(or a base class or a parent class).
- **Sub Class/Child Class**: The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example of Inheritance



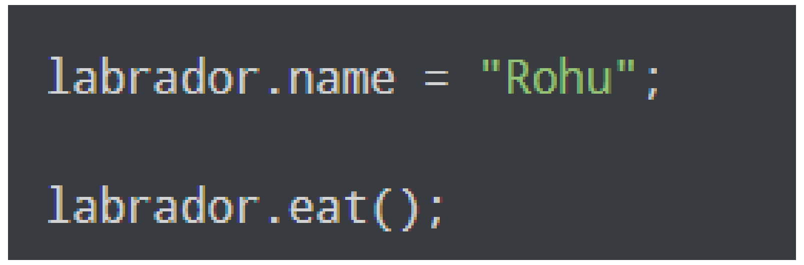
```
class Animal {

  // field and method of the parent class
  String name;
  public void eat() {
    System.out.println("I can eat");
  }
}

// inherit from Animal
class Dog extends Animal {

  // new method in subclass
  public void display() {
    System.out.println("My name is " + name);
  }
}

class Main {
  public static void main(String[] args) {

    // create an object of the subclass
    Dog labrador = new Dog();

    // access field of superclass
    labrador.name = "Rohu";
    labrador.display();

    // call method of superclass
    // using object of subclass
    labrador.eat();

  }
}
```

Output:



```
My name is Rohu
I can eat
```

- In the above example, we have derived a subclass Dog from superclass Animal. Notice the statements,



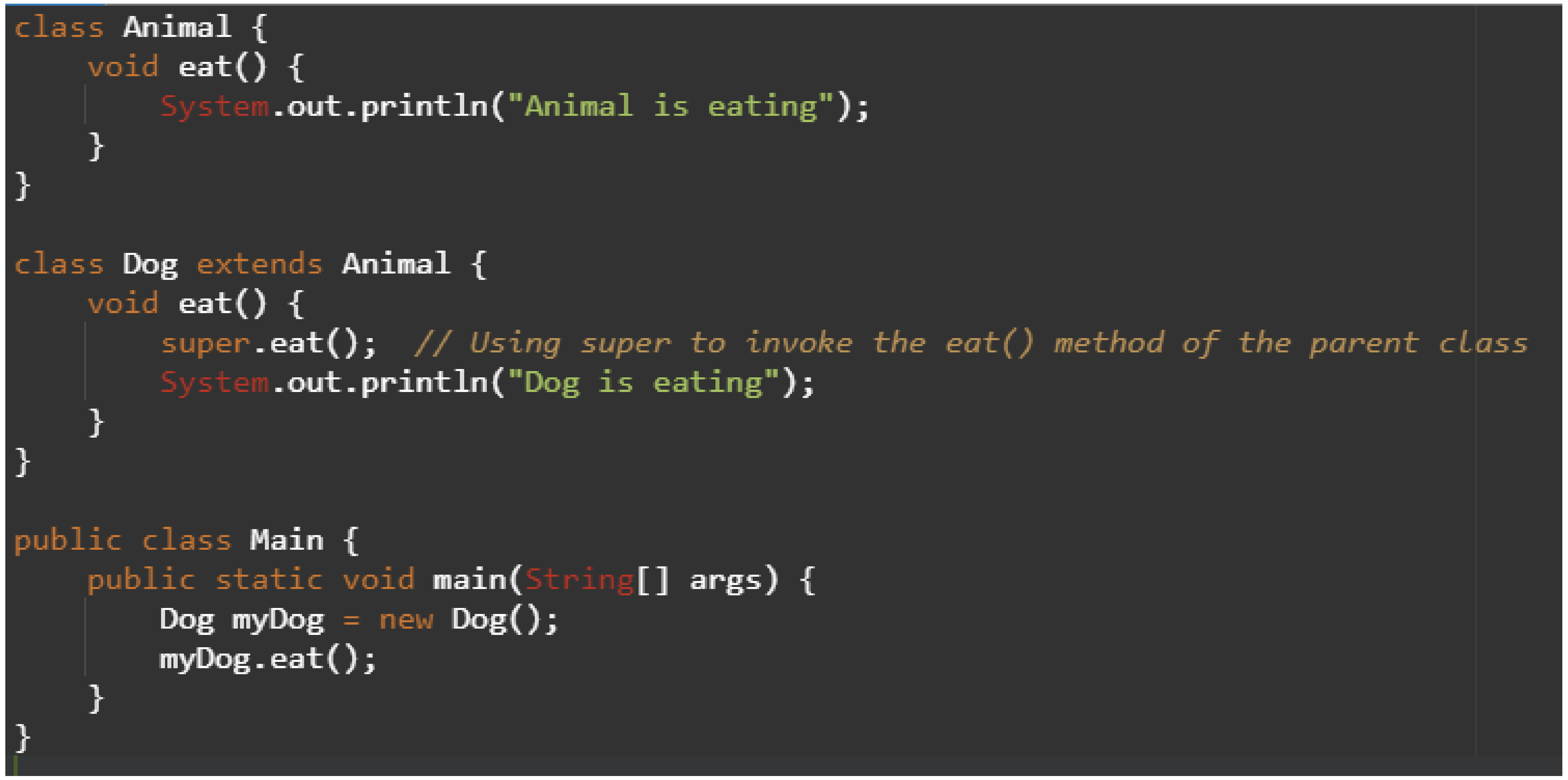
```
labrador.name = "Rohu";

labrador.eat();
```

- Here, labrador is an object of Dog. However, name and eat() are the members of the Animal class.
- Since Dog inherits the field and method from Animal, we are able to access the field and method using the object of the Dog.

this & super keyword

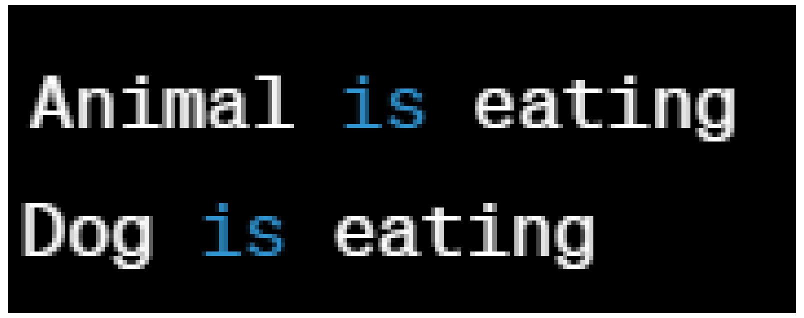
- `super` keyword in Java is used to refer to the immediate parent class object. It is often used to invoke the parent class methods, access parent class fields, or explicitly call the parent class constructor.
- `this` keyword in Java is used to refer to the current instance of the class. It is often used to differentiate between instance variables and parameters with the same name, and to invoke the current class methods.

super keyword example



```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

class Dog extends Animal {
    void eat() {
        super.eat();  // Using super to invoke the eat() method of the parent class
        System.out.println("Dog is eating");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();
    }
}
```

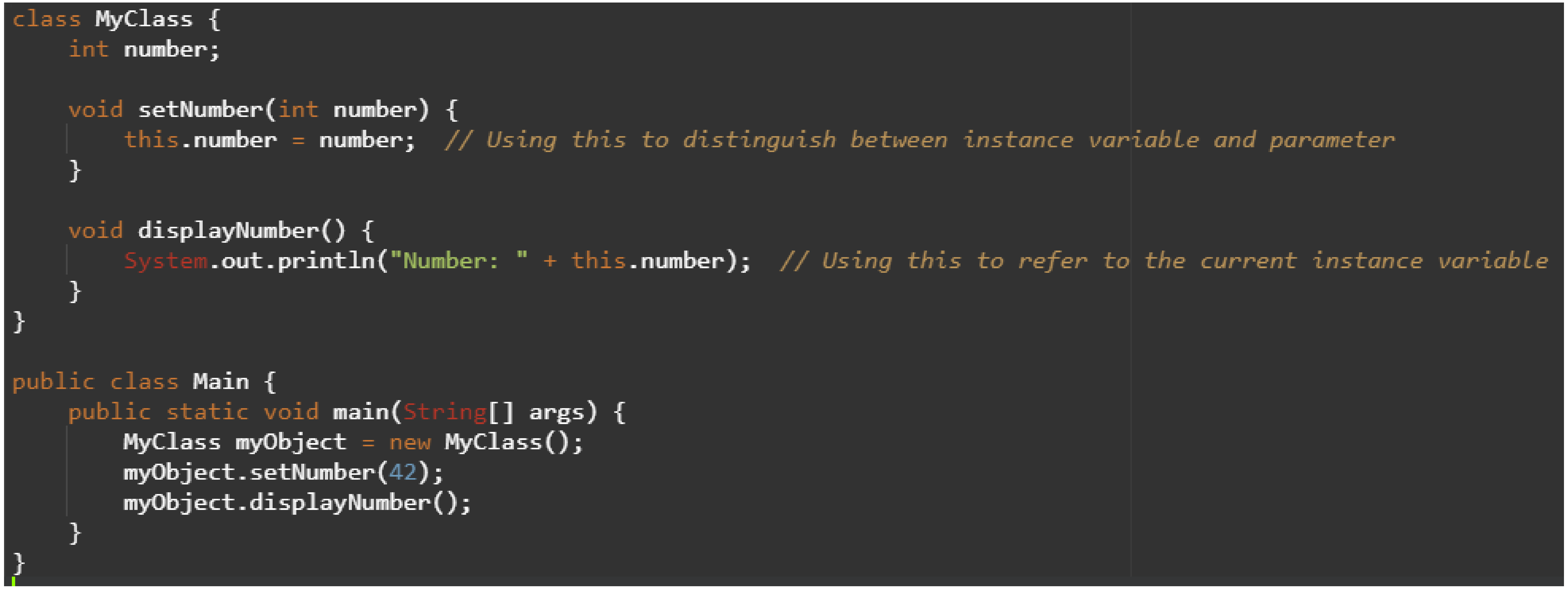
Output:



```
Animal is eating
Dog is eating
```

The code demonstrates inheritance in Java, where the super keyword is used to invoke a method from the immediate parent class while extending its functionality in the child class.

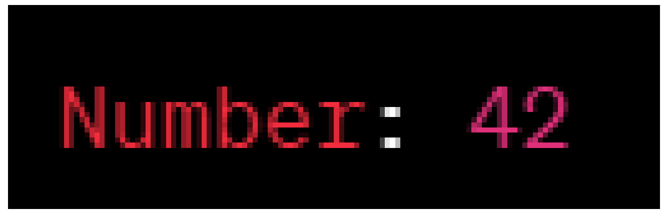
this keyword example



```java
class MyClass {
    int number;

    void setNumber(int number) {
        this.number = number;  // Using this to distinguish between instance variable and parameter
    }

    void displayNumber() {
        System.out.println("Number: " + this.number);  // Using this to refer to the current instance variable
    }
}

public class Main {
    public static void main(String[] args) {
        MyClass myObject = new MyClass();
        myObject.setNumber(42);
        myObject.displayNumber();
    }
}
```

this keyword example

Output:



```
Number: 42
```

This code illustrates the use of the this keyword in Java to distinguish between instance variables and parameters within a class, enhancing code clarity and avoiding naming conflicts.

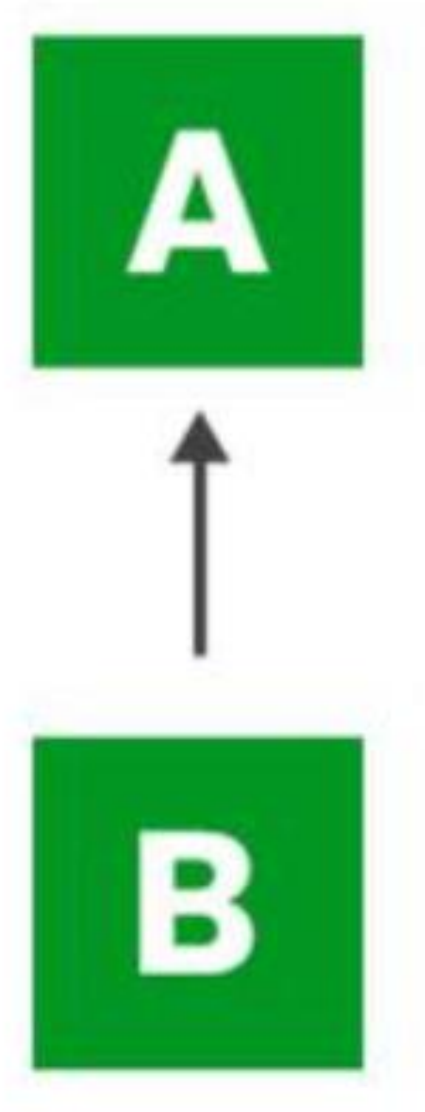
Types of Inheritance

There are 5 different types of inheritance which are supported by Java.

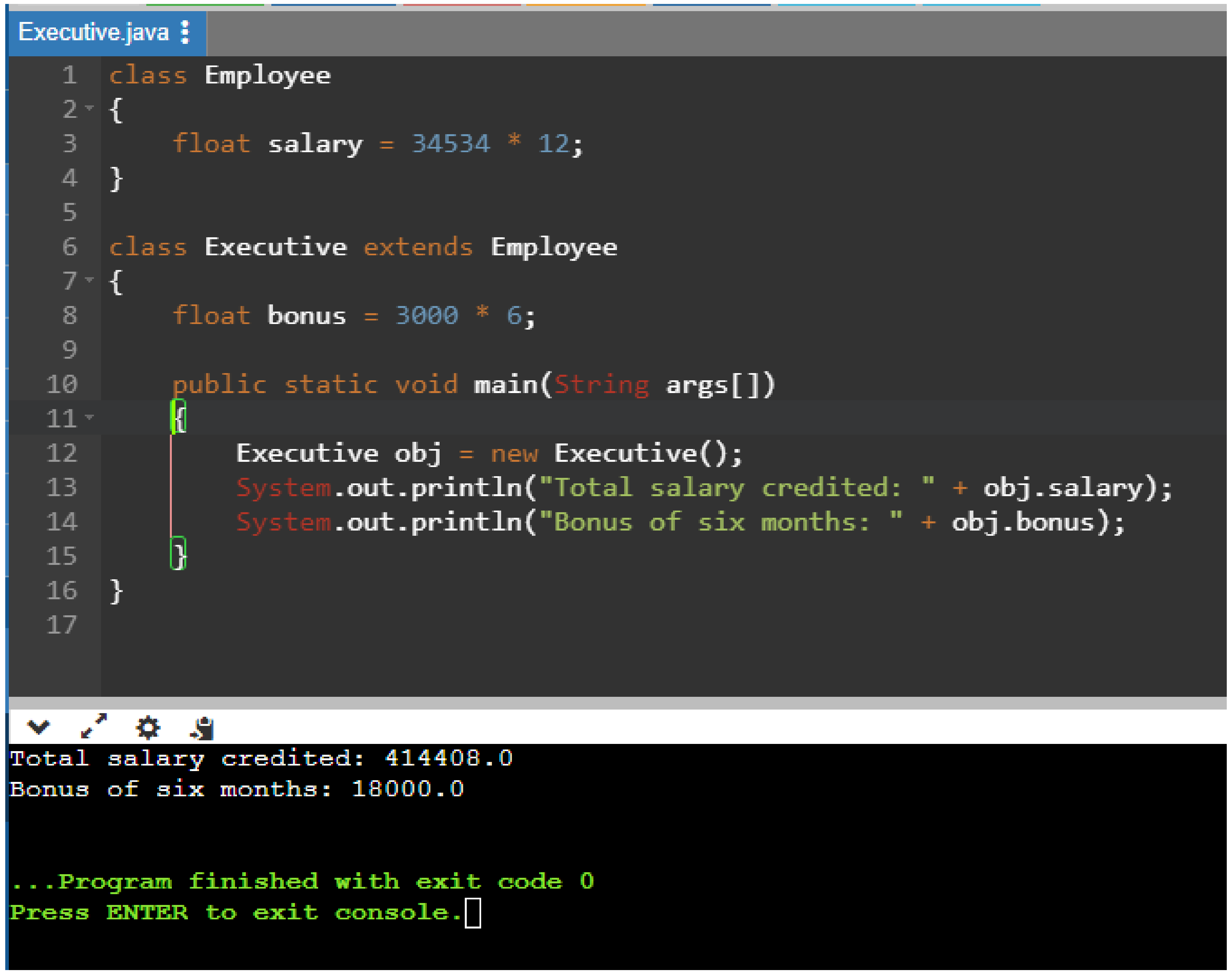
- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance

Single Inheritance

- In single inheritance, subclasses inherit the features of one superclass.
- In the image alongside, class A serves as a base class for the derived class B.



Single Inheritance Example



Executive.java

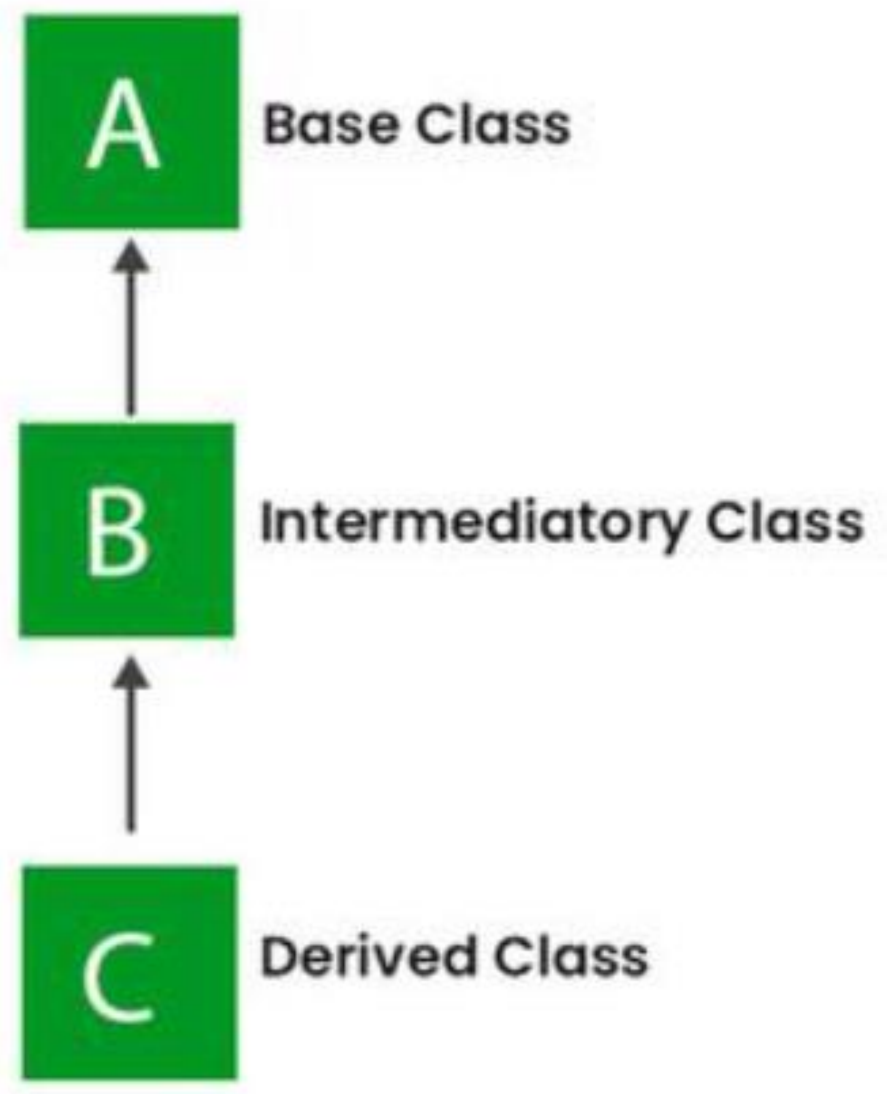
```java
class Employee
{
    float salary = 34534 * 12;
}

class Executive extends Employee
{
    float bonus = 3000 * 6;

    public static void main(String args[])
    {
        Executive obj = new Executive();
        System.out.println("Total salary credited: " + obj.salary);
        System.out.println("Bonus of six months: " + obj.bonus);
    }
}
```

```
Total salary credited: 414408.0
Bonus of six months: 18000.0


...Program finished with exit code 0
Press ENTER to exit console.
```

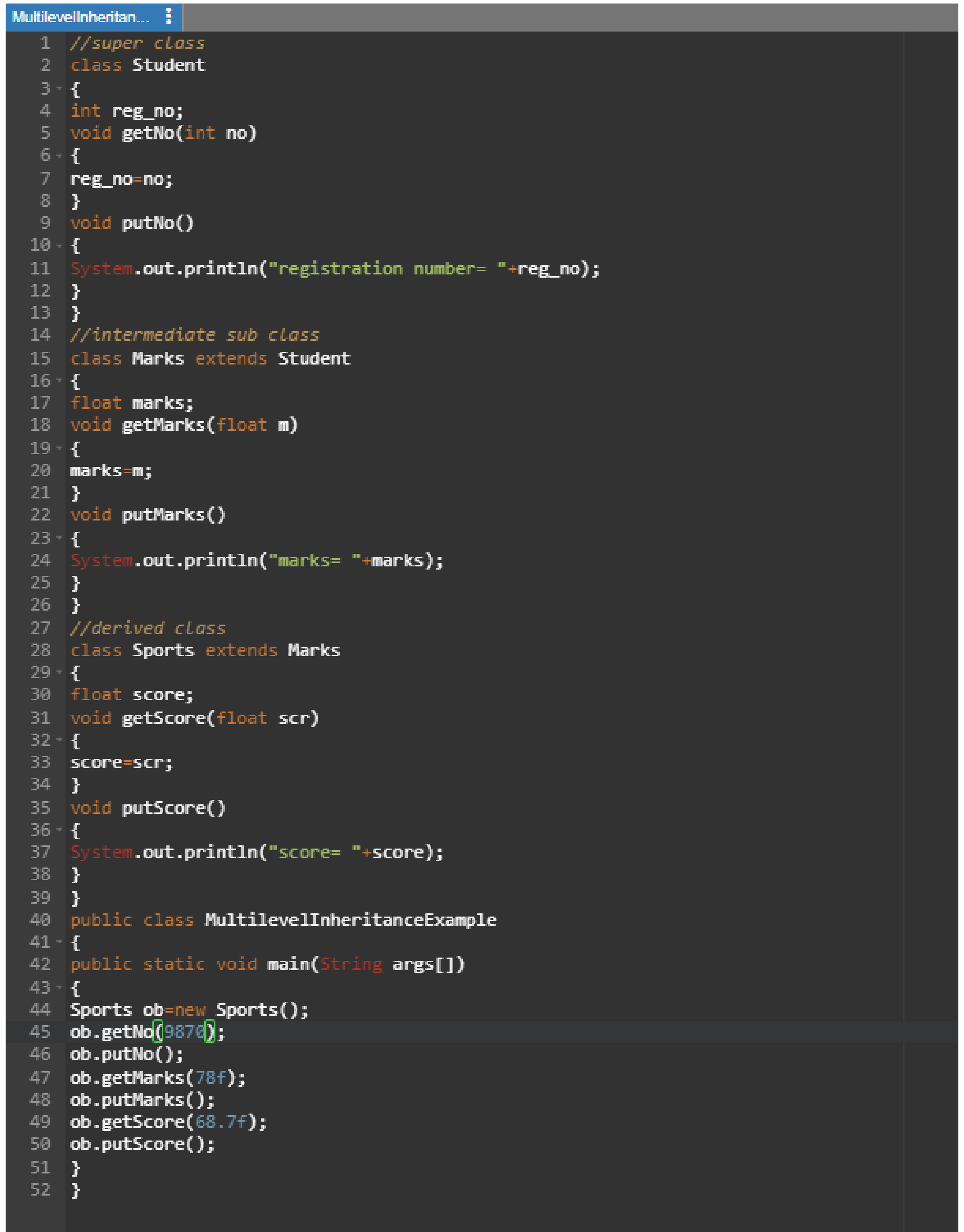
- In the alongside figure, Employee is a parent class and Executive is a child class.
- The Executive class inherits all the properties of the Employee class.

Multilevel Inheritance

- In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes.
- In the alongside image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.
- NOTE: In Java, a class cannot directly access the grandparent’s members.

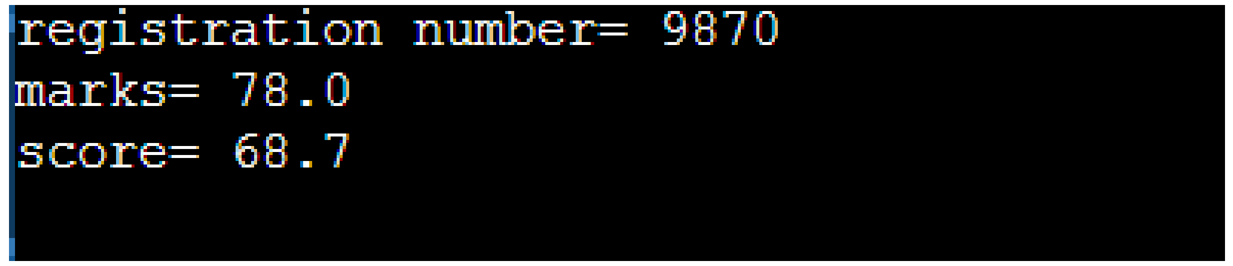


Multilevel Inheritance Example



```java
//super class
class Student
{
int reg_no;
void getNo(int no)
{
reg_no=no;
}
void putNo()
{
System.out.println("registration number= "+reg_no);
}
}
//intermediate sub class
class Marks extends Student
{
float marks;
void getMarks(float m)
{
marks=m;
}
void putMarks()
{
System.out.println("marks= "+marks);
}
}
//derived class
class Sports extends Marks
{
float score;
void getScore(float scr)
{
score=scr;
}
void putScore()
{
System.out.println("score= "+score);
}
}
public class MultilevelInheritanceExample
{
public static void main(String args[])
{
Sports ob=new Sports();
ob.getNo(9870);
ob.putNo();
ob.getMarks(78f);
ob.putMarks();
ob.getScore(68.7f);
ob.putScore();
}
}
```

Output:

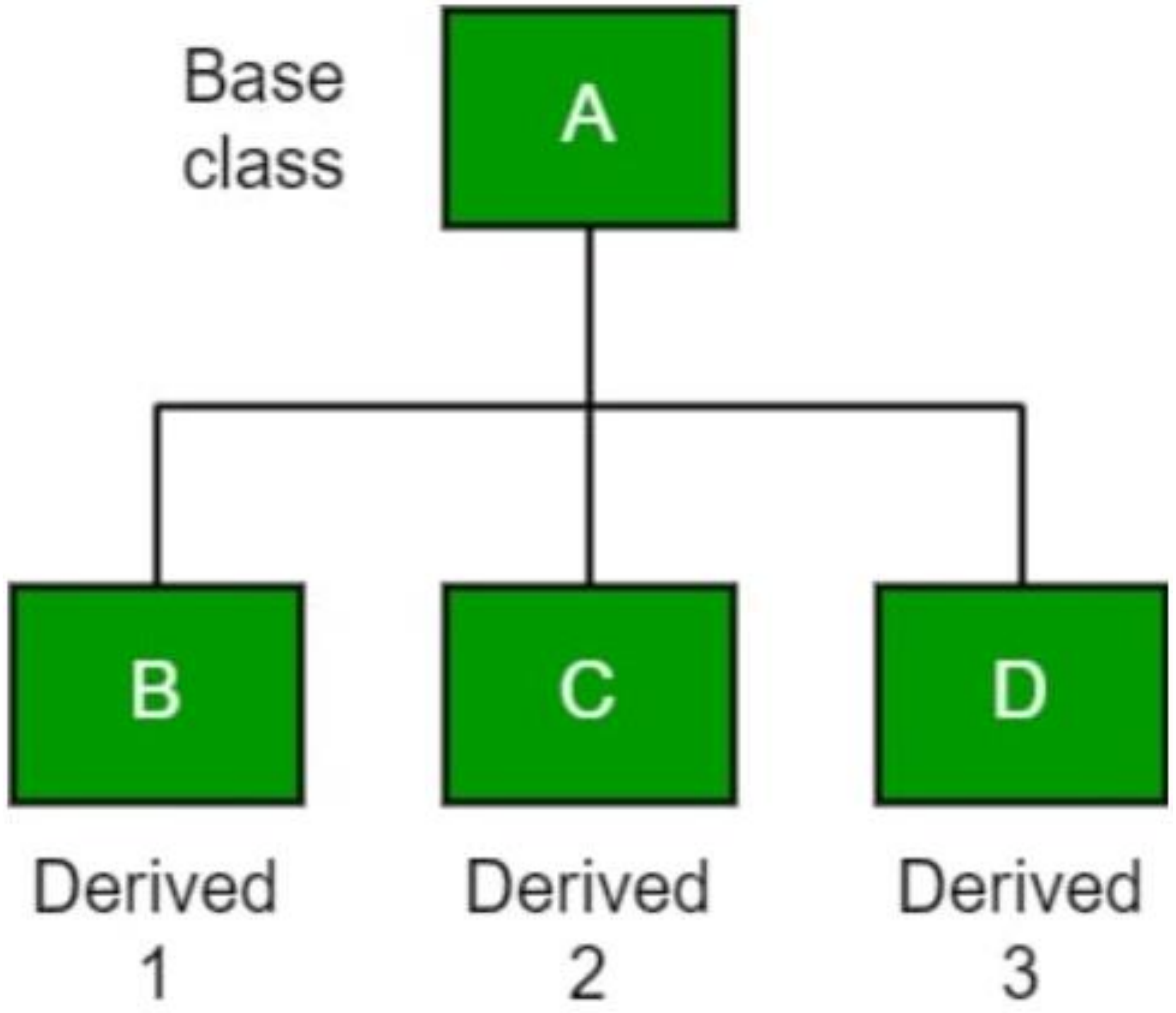


```
registration number= 9870
marks= 78.0
score= 68.7
```

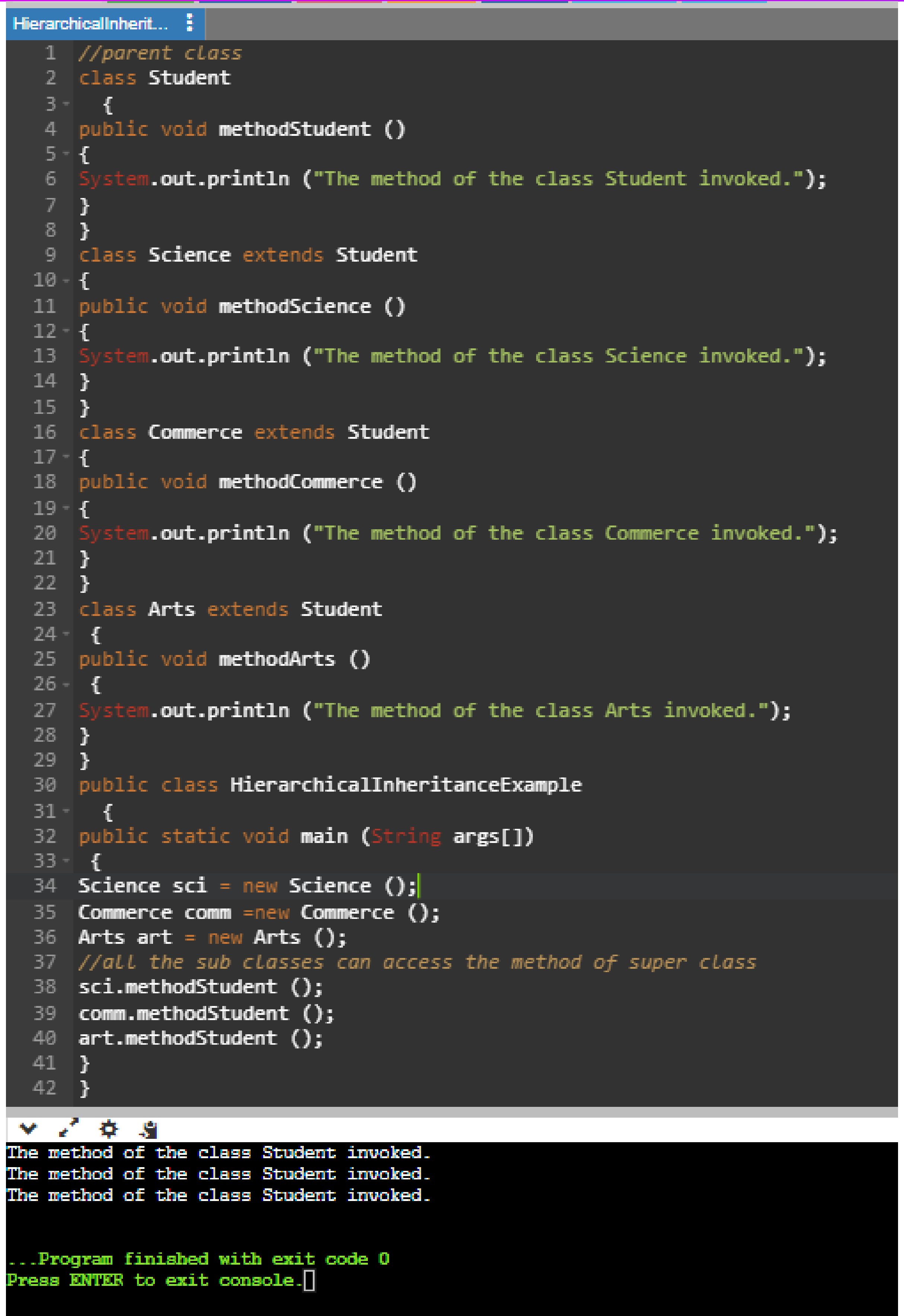
- In the alongside code, the class Marks inherits the members or methods of the class Students.
- The class Sports inherits the members of the class Marks.
- Therefore, the Student class is the parent class of the class Marks and the class Marks is the parent of the class Sports.
- Hence, the class Sports implicitly inherits the properties of the Student along with the class Marks.

Hierarchical Inheritance

- In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass.
- In the alongside image, class A serves as a base class for the derived classes B, C, and D.



Hierarchical Inheritance Example



HierarchicalInherit...

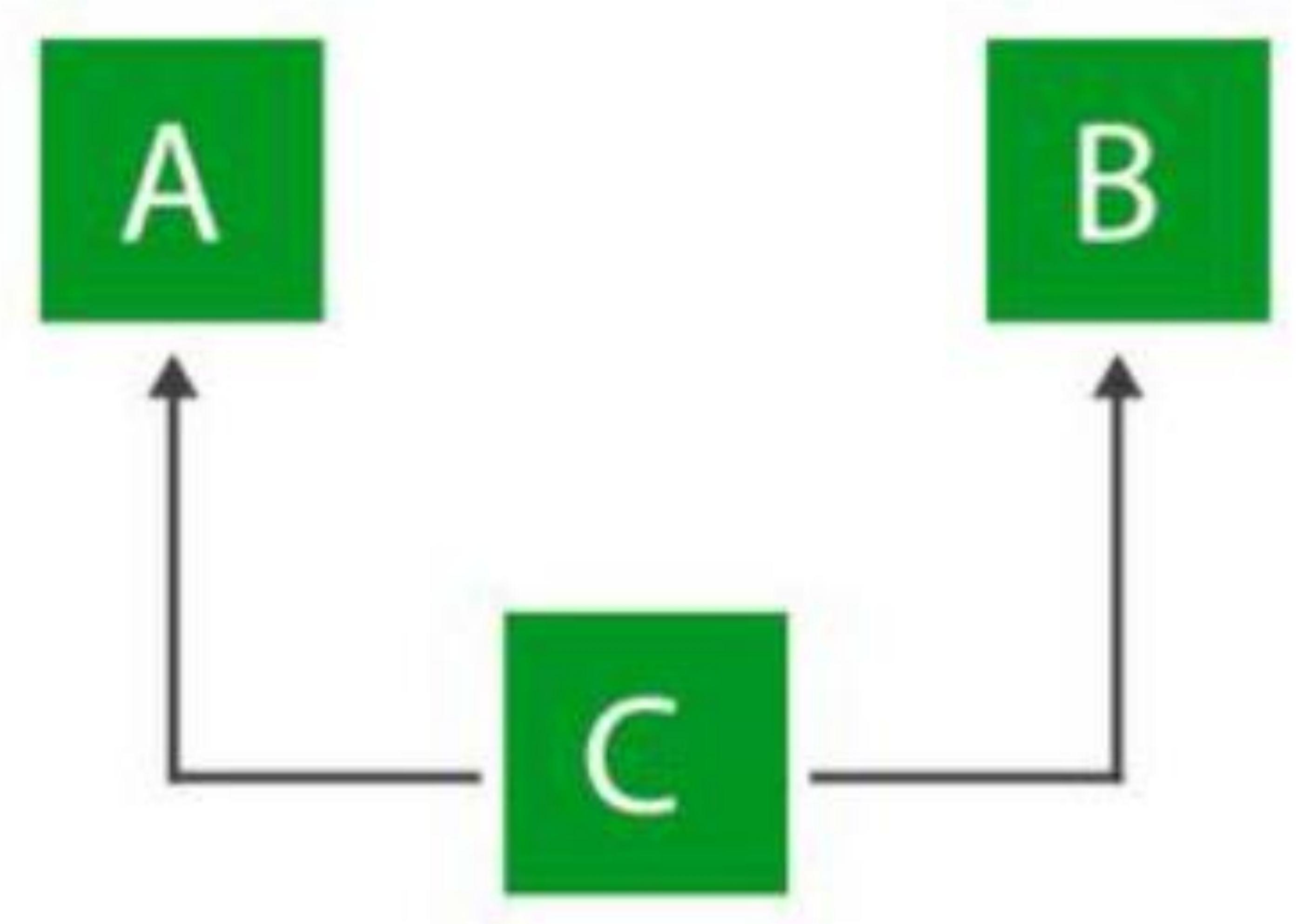
```
//parent class
class Student
  {
public void methodStudent ()
{
System.out.println ("The method of the class Student invoked.");
}
}
class Science extends Student
{
public void methodScience ()
{
System.out.println ("The method of the class Science invoked.");
}
}
class Commerce extends Student
{
public void methodCommerce ()
{
System.out.println ("The method of the class Commerce invoked.");
}
}
class Arts extends Student
 {
public void methodArts ()
 {
System.out.println ("The method of the class Arts invoked.");
}
}
public class HierarchicalInheritanceExample
  {
public static void main (String args[])
 {
Science sci = new Science ();
Commerce comm =new Commerce ();
Arts art = new Arts ();
//all the sub classes can access the method of super class
sci.methodStudent ();
comm.methodStudent ();
art.methodStudent ();
}
}
```

```
The method of the class Student invoked.
The method of the class Student invoked.
The method of the class Student invoked.


...Program finished with exit code 0
Press ENTER to exit console.
```

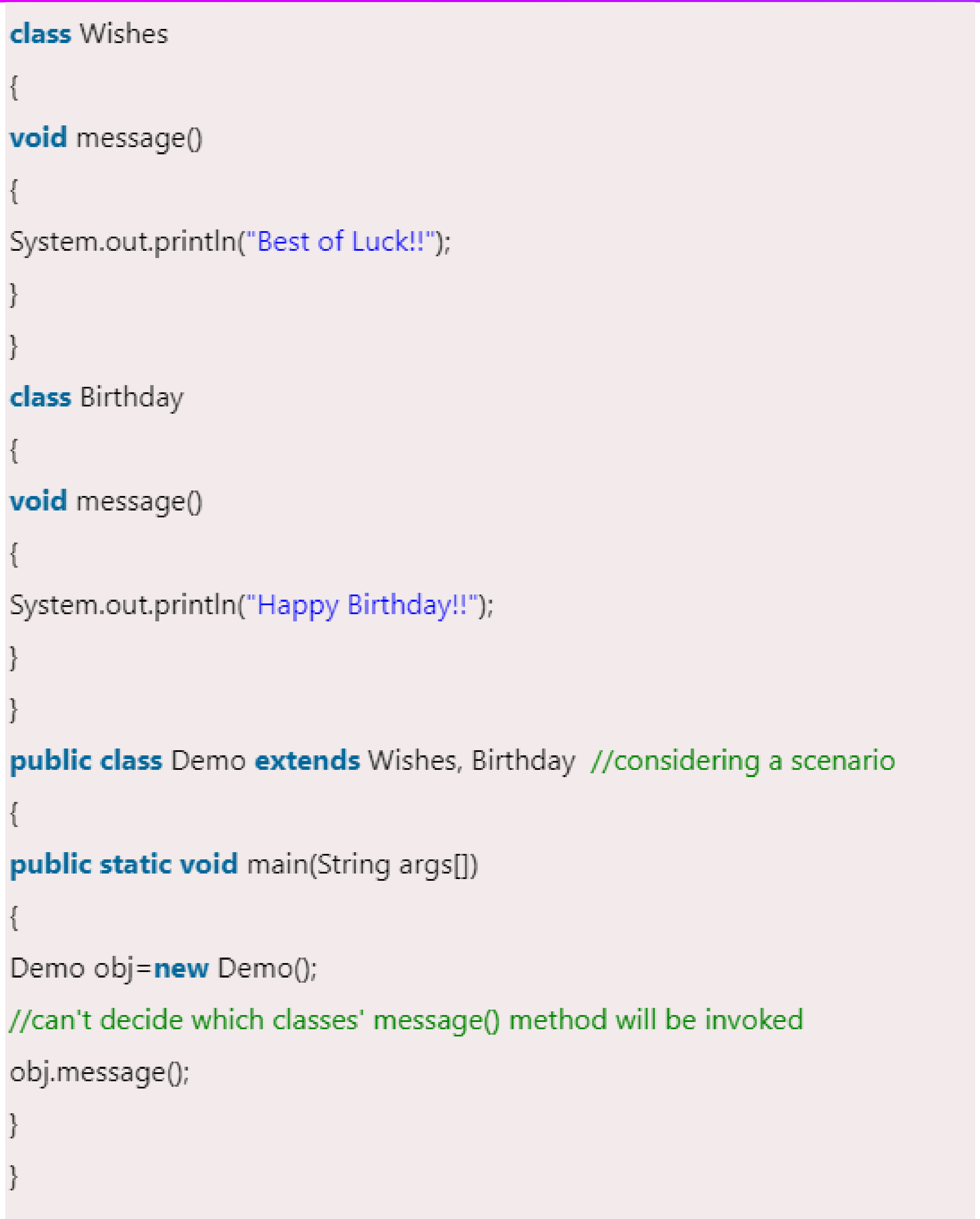
- In the alongsside code, the classes Science, Commerce, and Arts inherit a single parent class named Student.

Multiple Inheritance

- In Multiple Inheritance, one class can have more than one superclass and inherit features from all parent classes.
- Note that Java does not support multiple inheritances with classes.
- In Java, we can achieve multiple inheritances only through Interfaces. In the image below, Class C is derived from interfaces A and B.



Multiple Inheritance Example

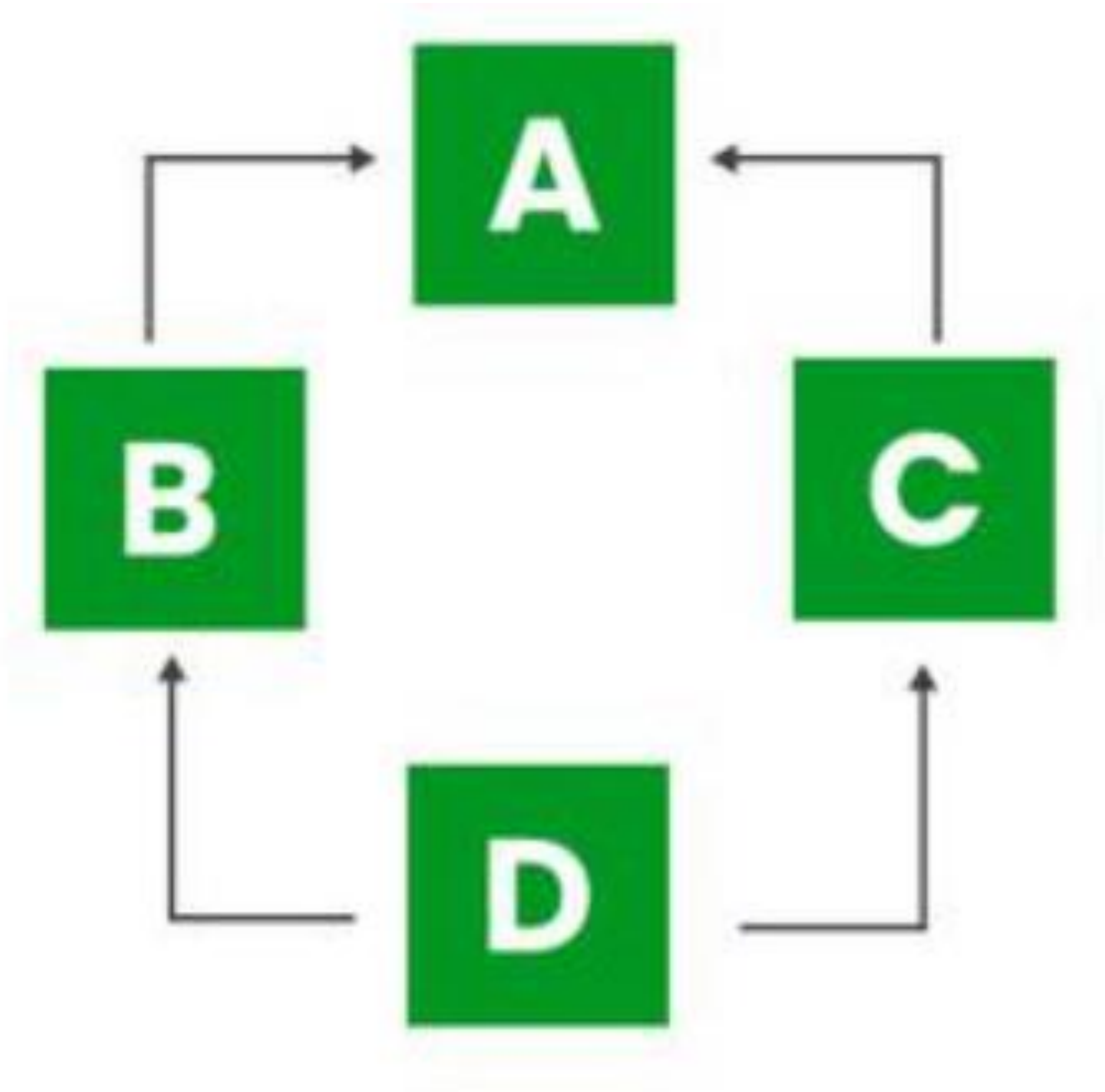


```
class Wishes
{
void message()
{
System.out.println("Best of Luck!!");
}
}
class Birthday
{
void message()
{
System.out.println("Happy Birthday!!");
}
}
public class Demo extends Wishes, Birthday  //considering a scenario
{
public static void main(String args[])
{
Demo obj=new Demo();
//can't decide which classes' message() method will be invoked
obj.message();
}
}
```

- The above code gives error because the compiler cannot decide which message() method is to be invoked.
- Due to this reason, Java does not support multiple inheritances at the class level but can be achieved through an interface.

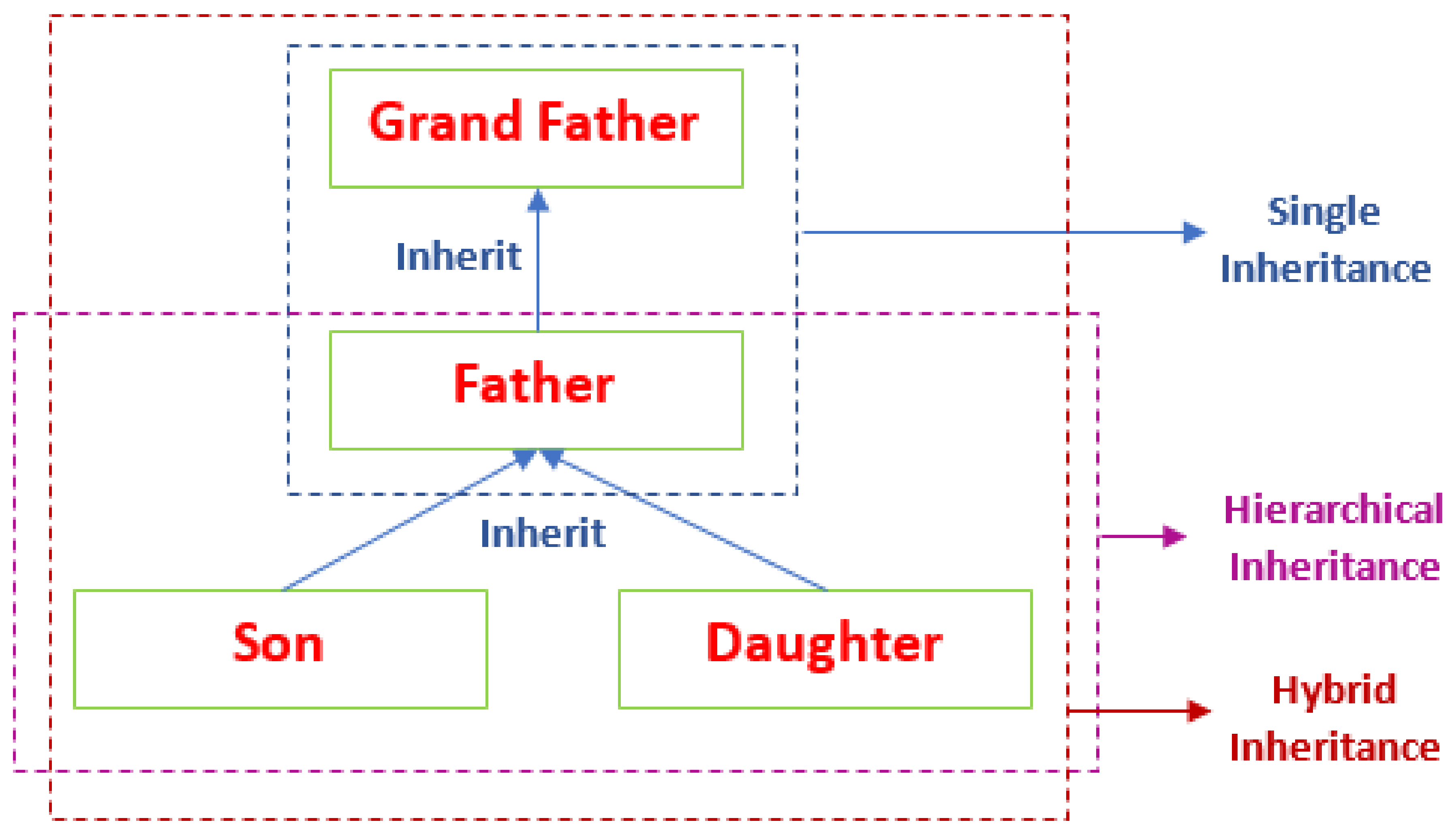
Hybrid Inheritance

- It is a mix of two or more of the above types of inheritance.
- Since Java doesn’t support multiple inheritances with classes, hybrid inheritance involving multiple inheritance is also not possible with classes.
- In Java, we can achieve hybrid inheritance only through Interfaces if we want to involve multiple inheritance to implement Hybrid inheritance.

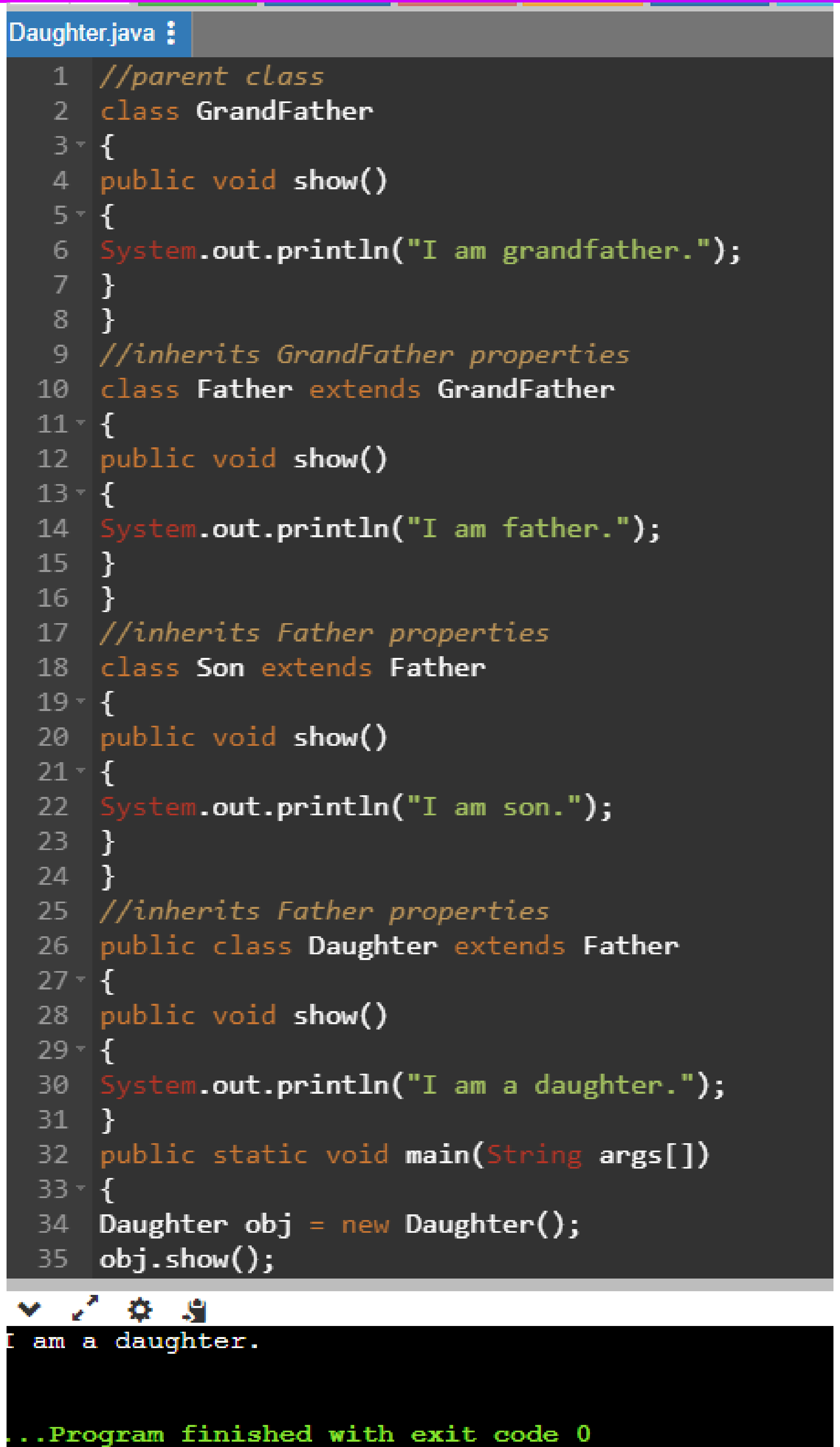


Hybrid Inheritance

- However, it is important to note that Hybrid inheritance does not necessarily require the use of Multiple Inheritance exclusively.
- It can be achieved through a combination of Multilevel Inheritance and Hierarchical Inheritance with classes, Hierarchical and Single Inheritance with classes.



Hybrid Inheritance Example



Daughter.java

```java
//parent class
class GrandFather
{
public void show()
{
System.out.println("I am grandfather.");
}
}
//inherits GrandFather properties
class Father extends GrandFather
{
public void show()
{
System.out.println("I am father.");
}
}
//inherits Father properties
class Son extends Father
{
public void show()
{
System.out.println("I am son.");
}
}
//inherits Father properties
public class Daughter extends Father
{
public void show()
{
System.out.println("I am a daughter.");
}
public static void main(String args[])
{
Daughter obj = new Daughter();
obj.show();
```

```
I am a daughter.

...Program finished with exit code 0
```

- In the alongside code, GrandFather is a super class.
- The Father class inherits the properties of the GrandFather class.
- Since Father and GrandFather represents single inheritance.
- Further, the Father class is inherited by the Son and Daughter class.
- Thus, the Father becomes the parent class for Son and Daughter.
- These classes represent the hierarchical inheritance. Combinedly, it denotes the hybrid inheritance.