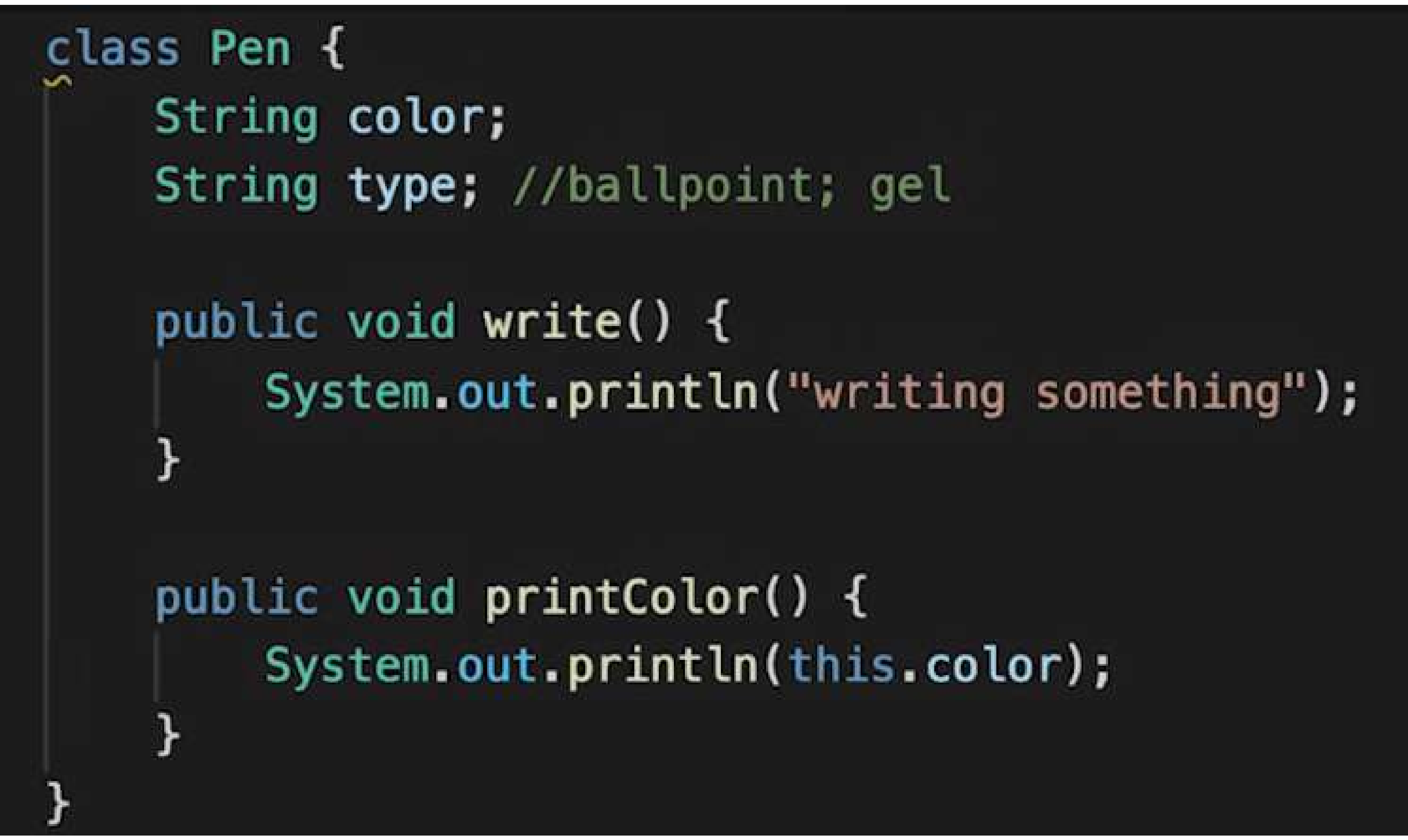
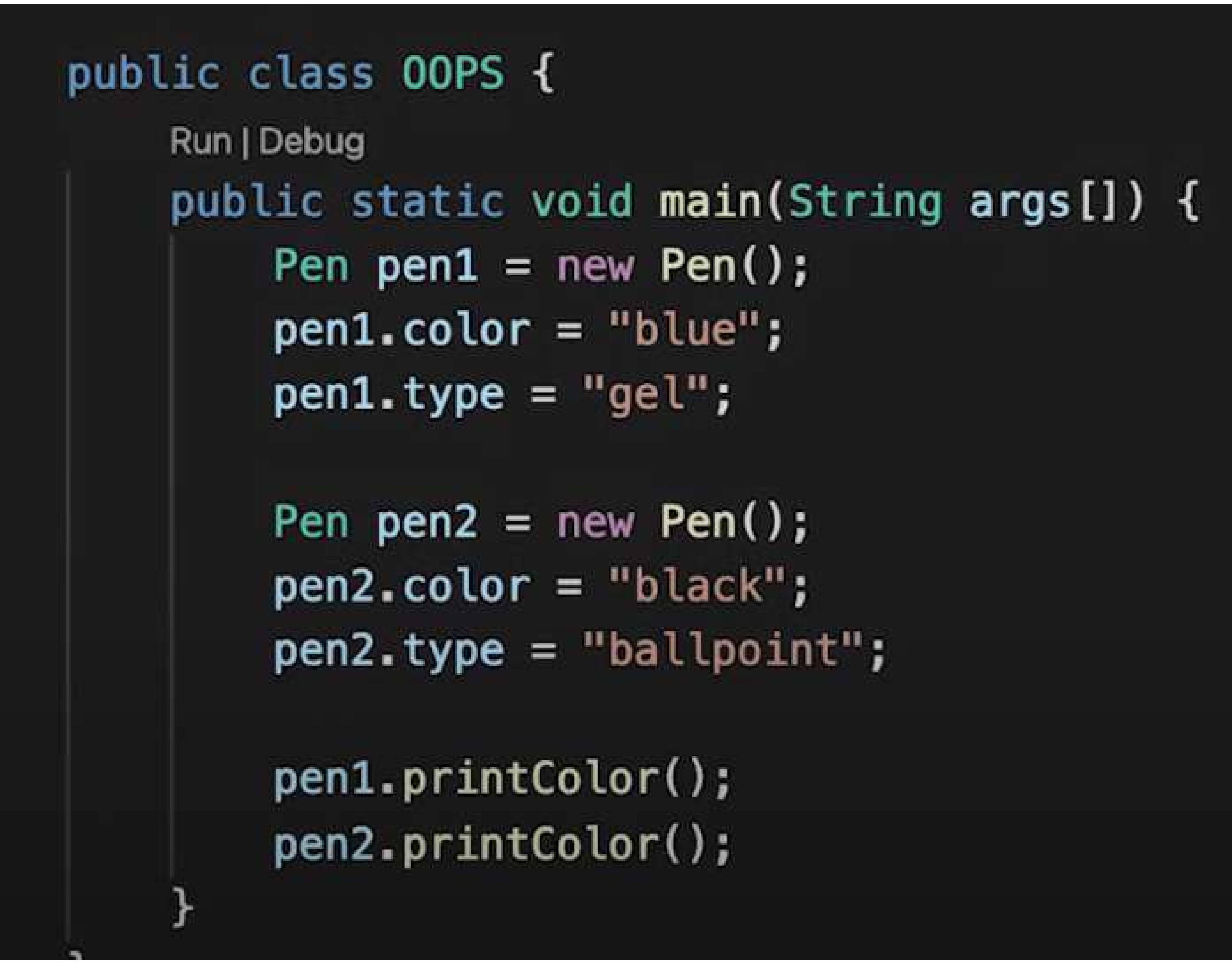
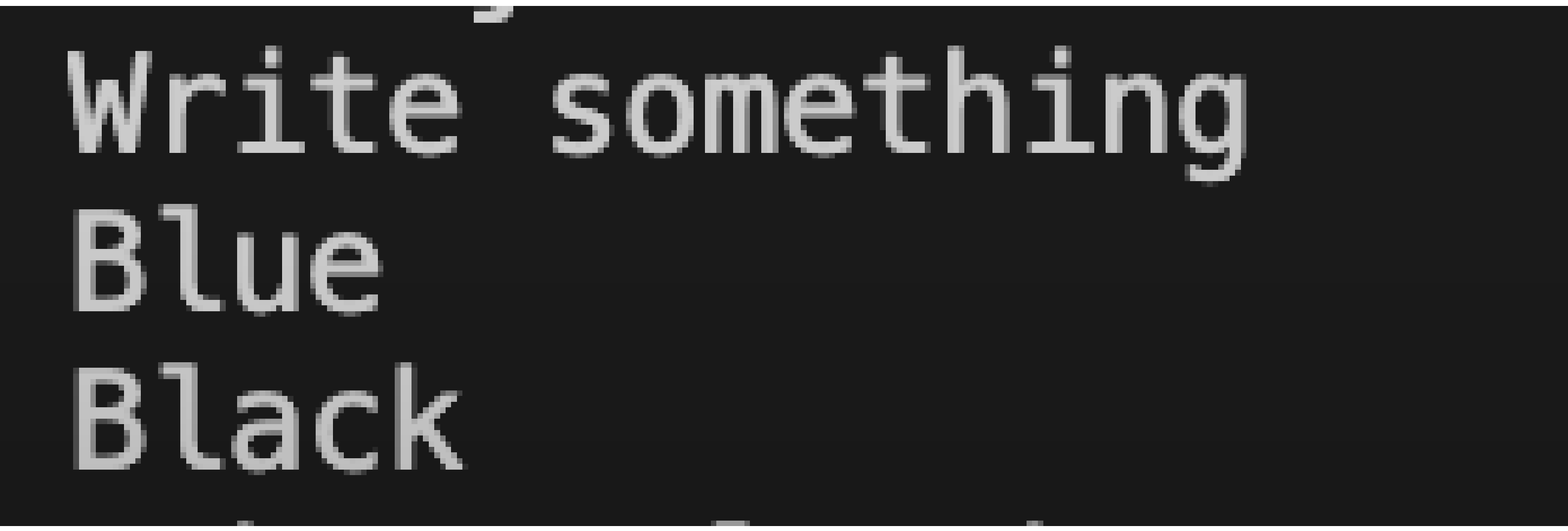


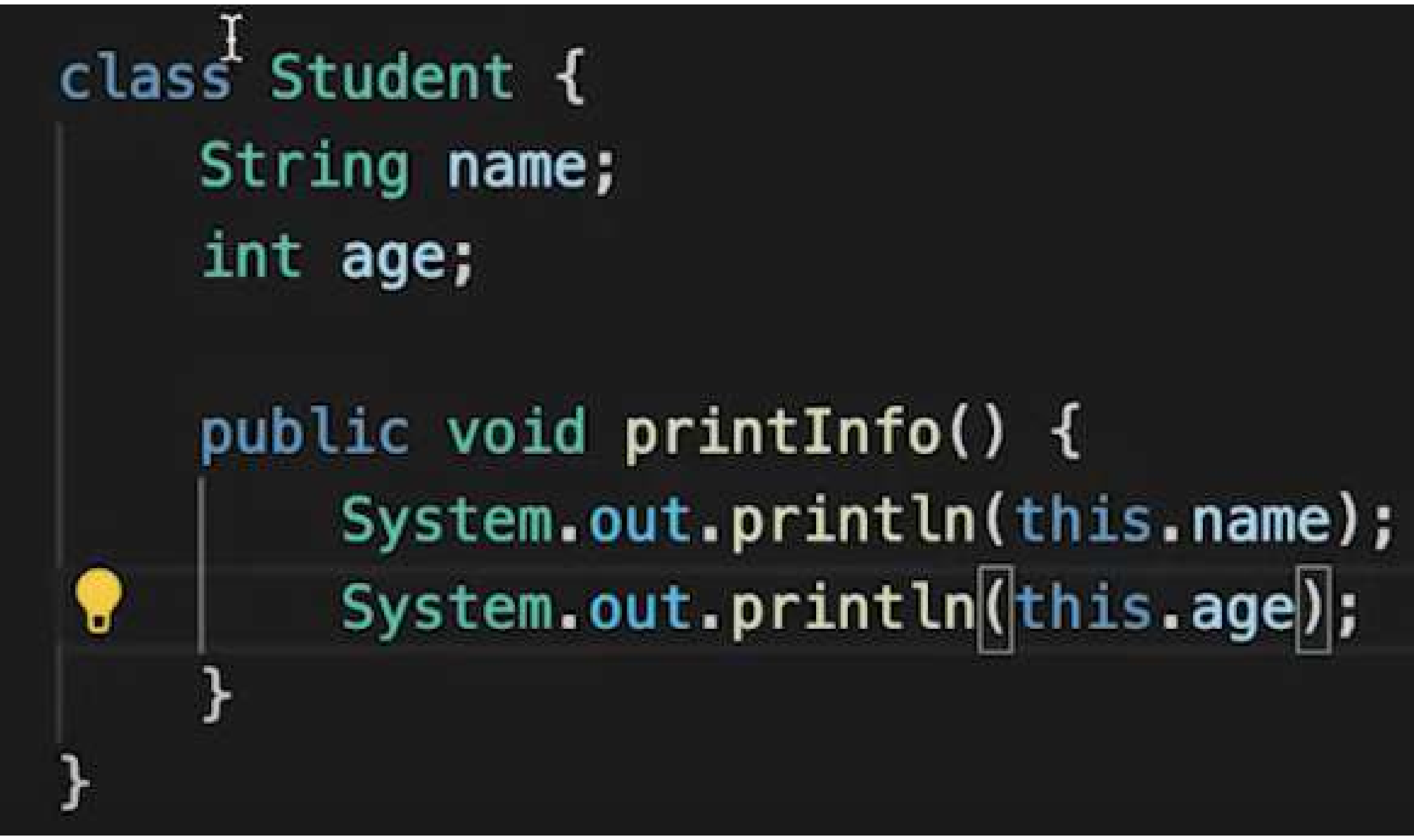
```java
class Pen {
    String color;
    String type; //ballpoint; gel

    public void write() {
        System.out.println("writing something");
    }

    public void printColor() {
        System.out.println(this.color);
    }
}
```



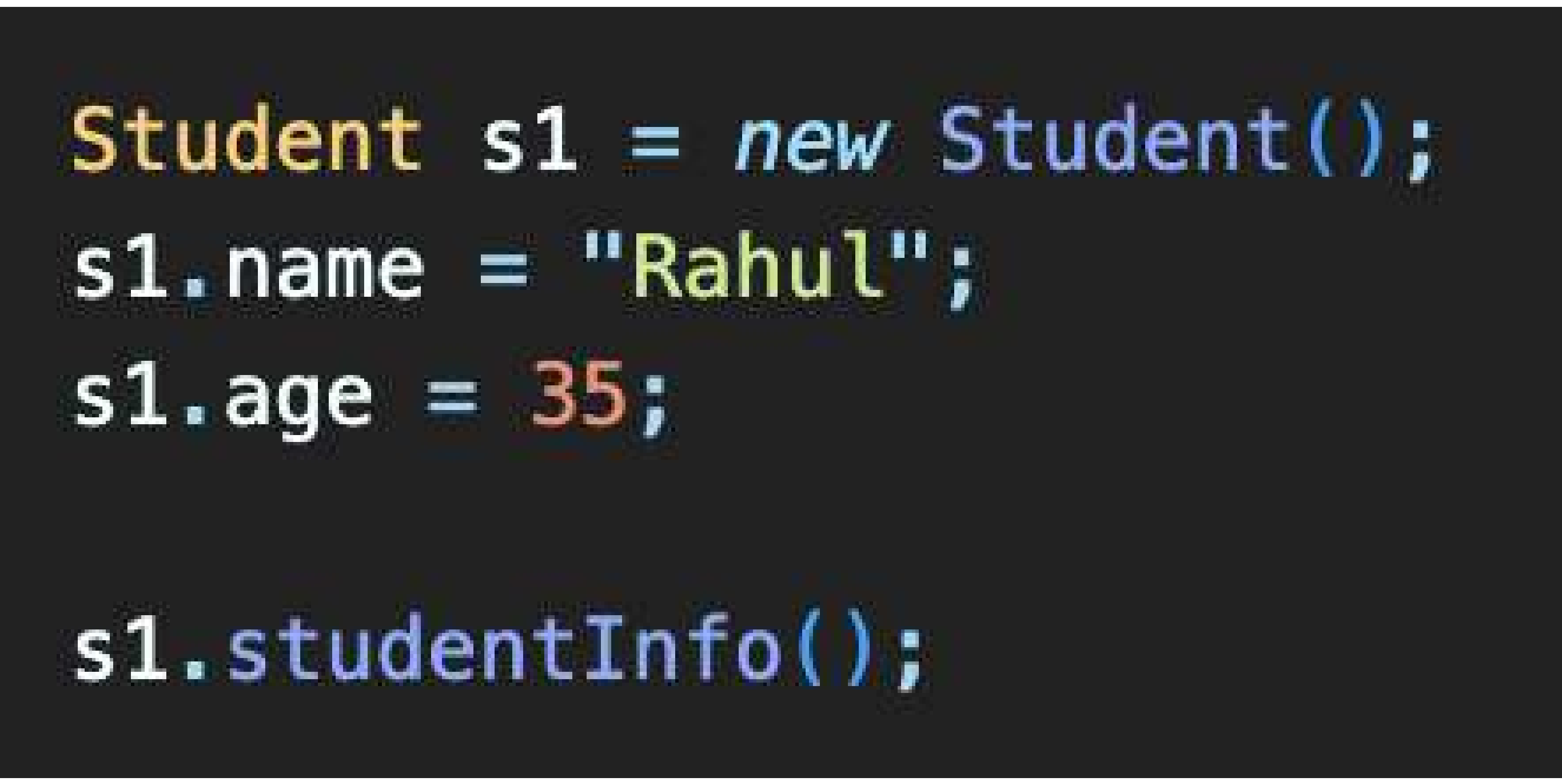
```java
public class OOPS {
    public static void main(String args[]) {
        Pen pen1 = new Pen();
        pen1.color = "blue";
        pen1.type = "gel";

        Pen pen2 = new Pen();
        pen2.color = "black";
        pen2.type = "ballpoint";

        pen1.printColor();
        pen2.printColor();
    }
```



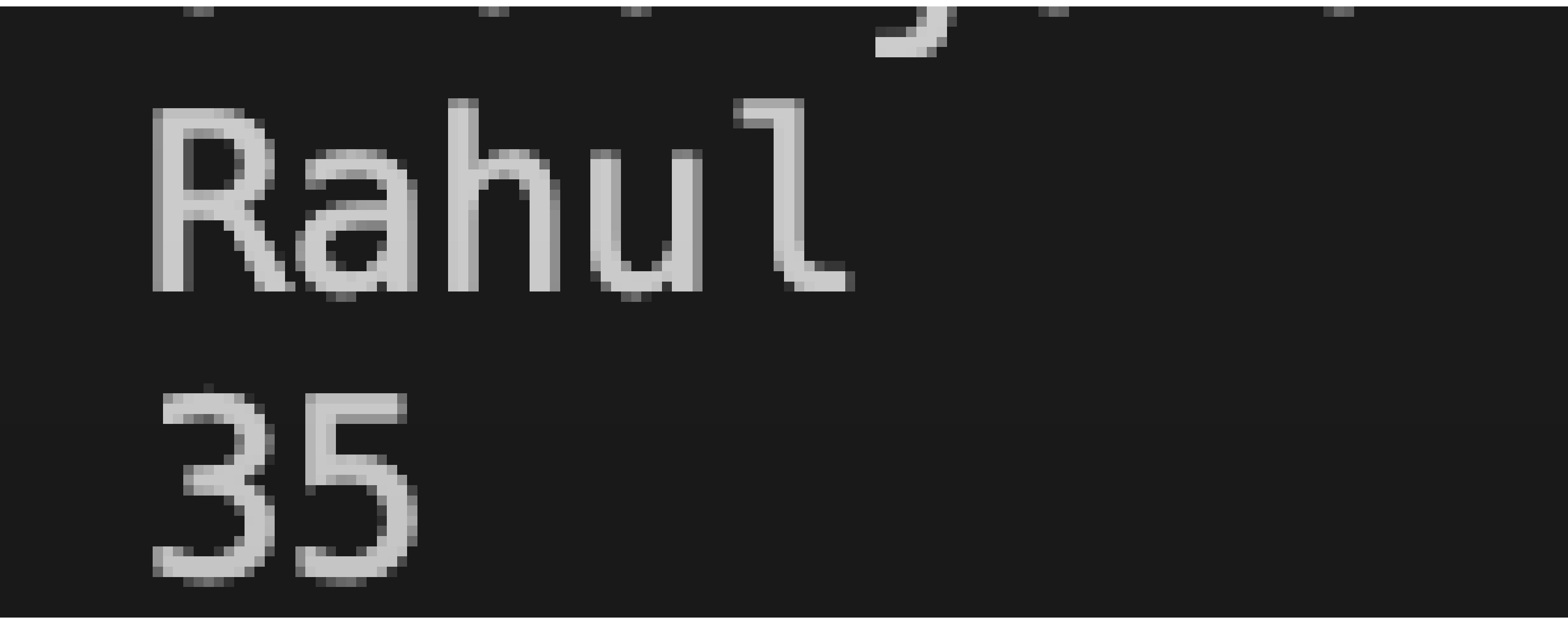
```
Write something
Blue
Black
```



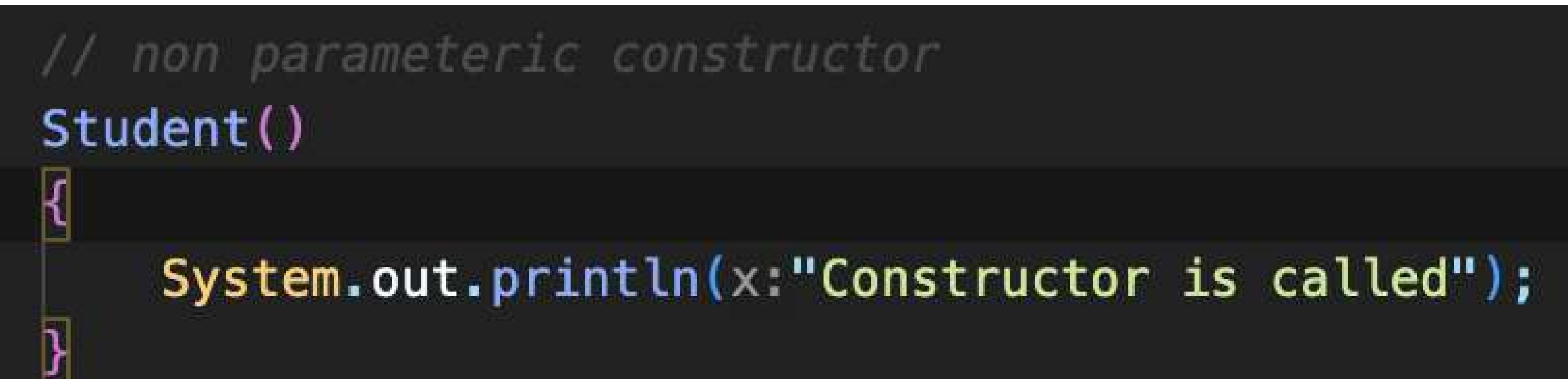
```
class Student {
    String name;
    int age;

    public void printInfo() {
        System.out.println(this.name);
        System.out.println(this.age);
    }
}
```



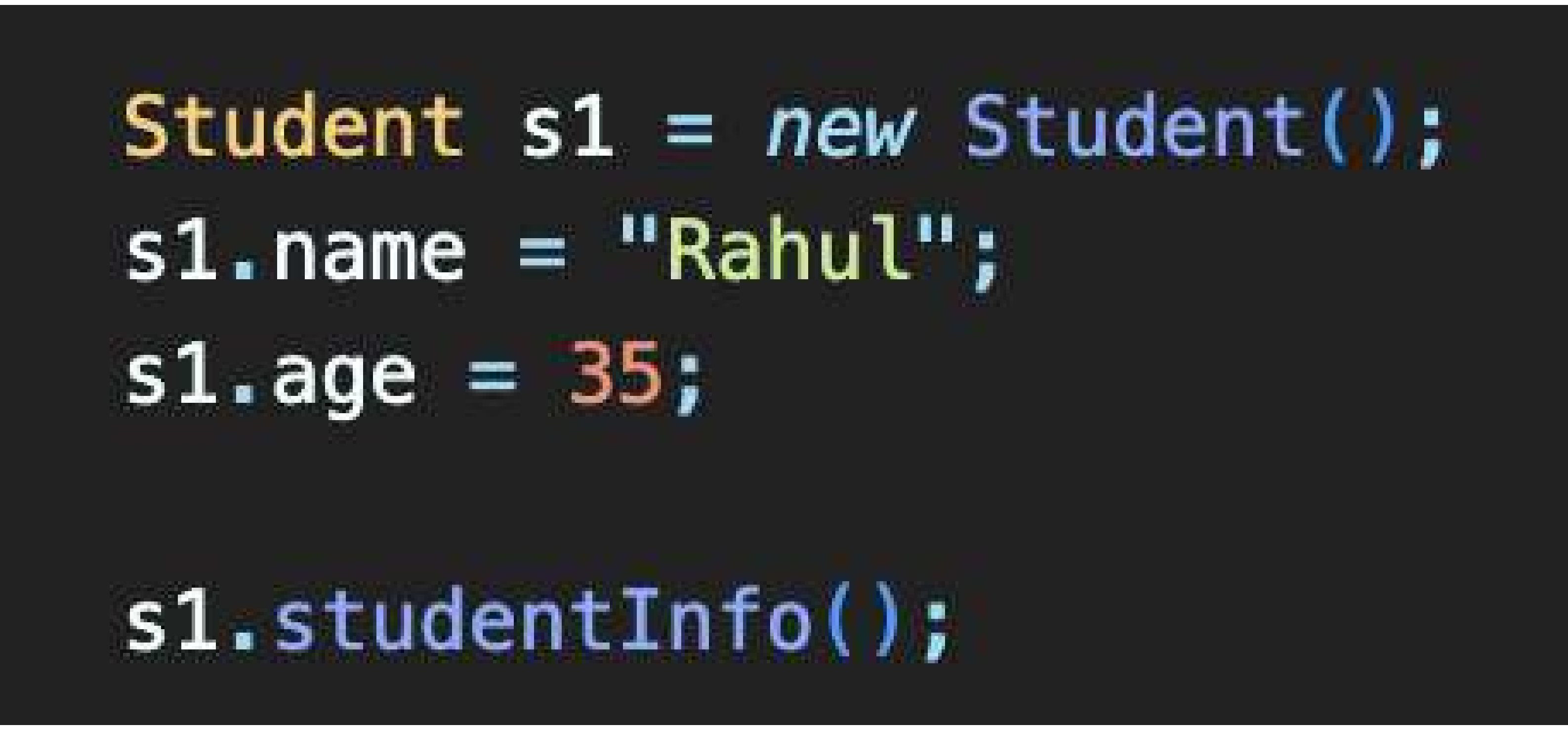
```java
Student s1 = new Student();
s1.name = "Rahul";
s1.age = 35;

s1.studentInfo();
```

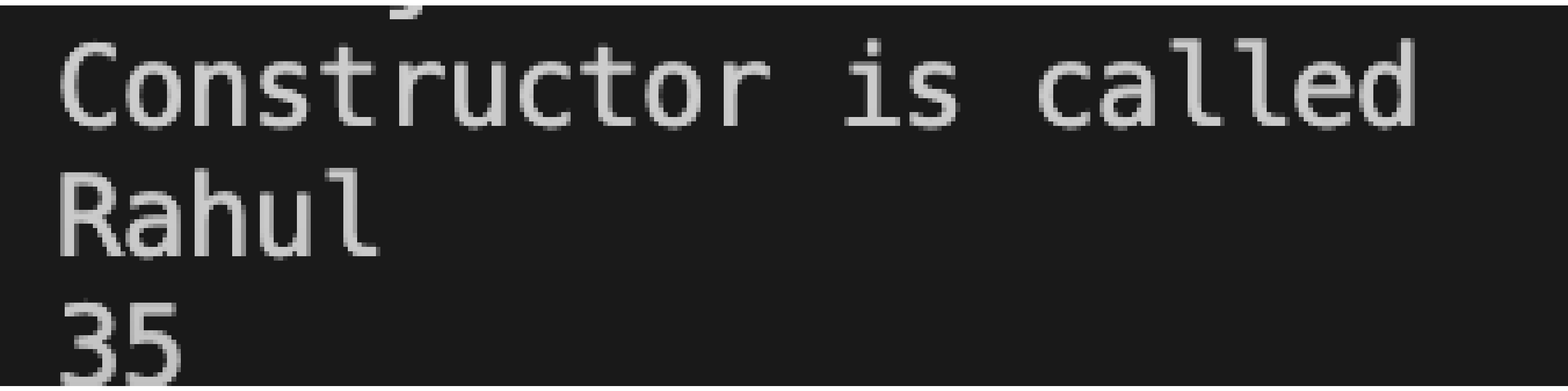


```
Rahul
35
```

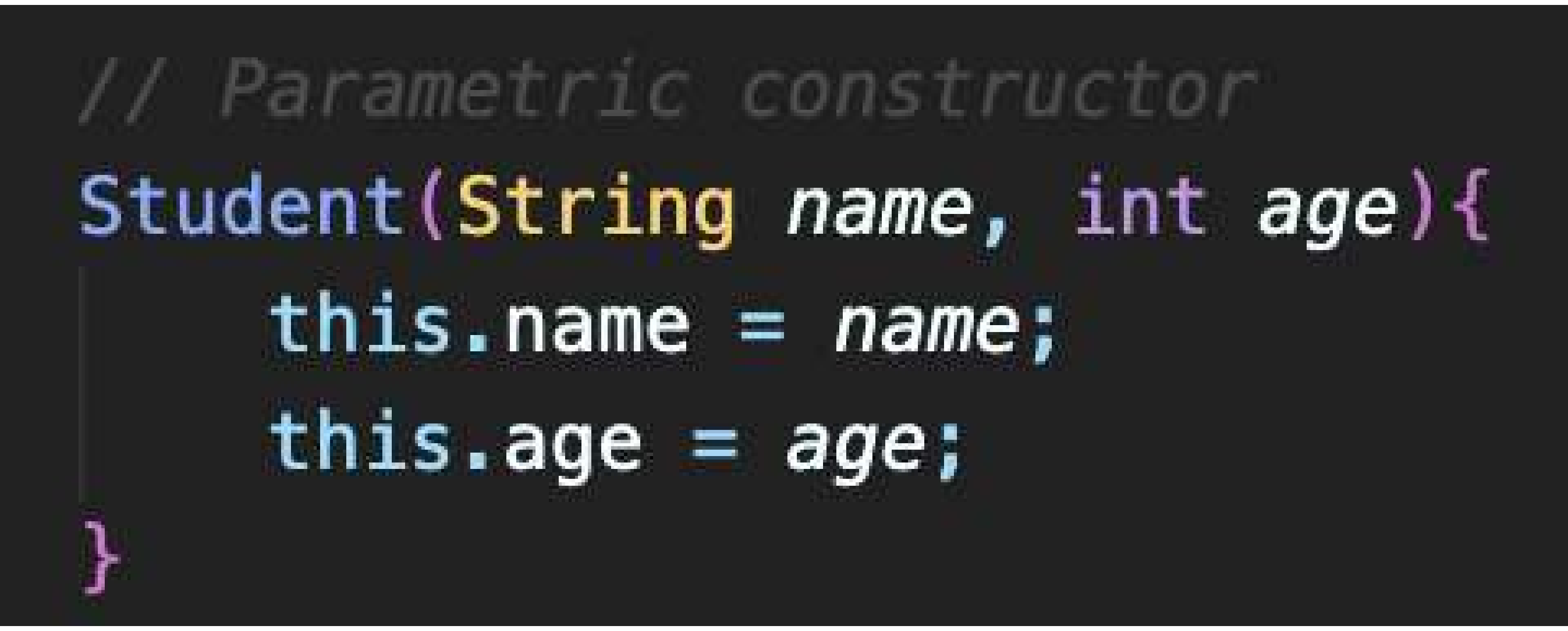


```java
// non parameteric constructor
Student()
{
    System.out.println(x:"Constructor is called");
}
```

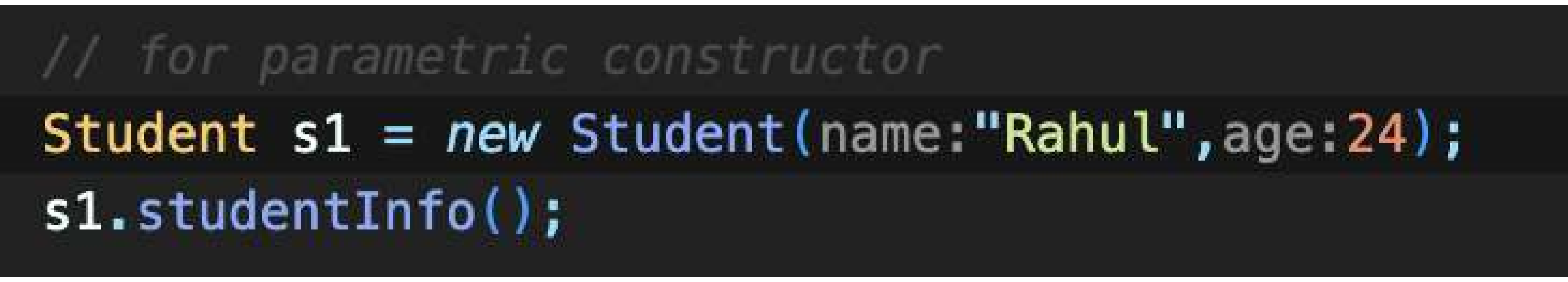

```java
Student s1 = new Student();
s1.name = "Rahul";
s1.age = 35;

s1.studentInfo();
```



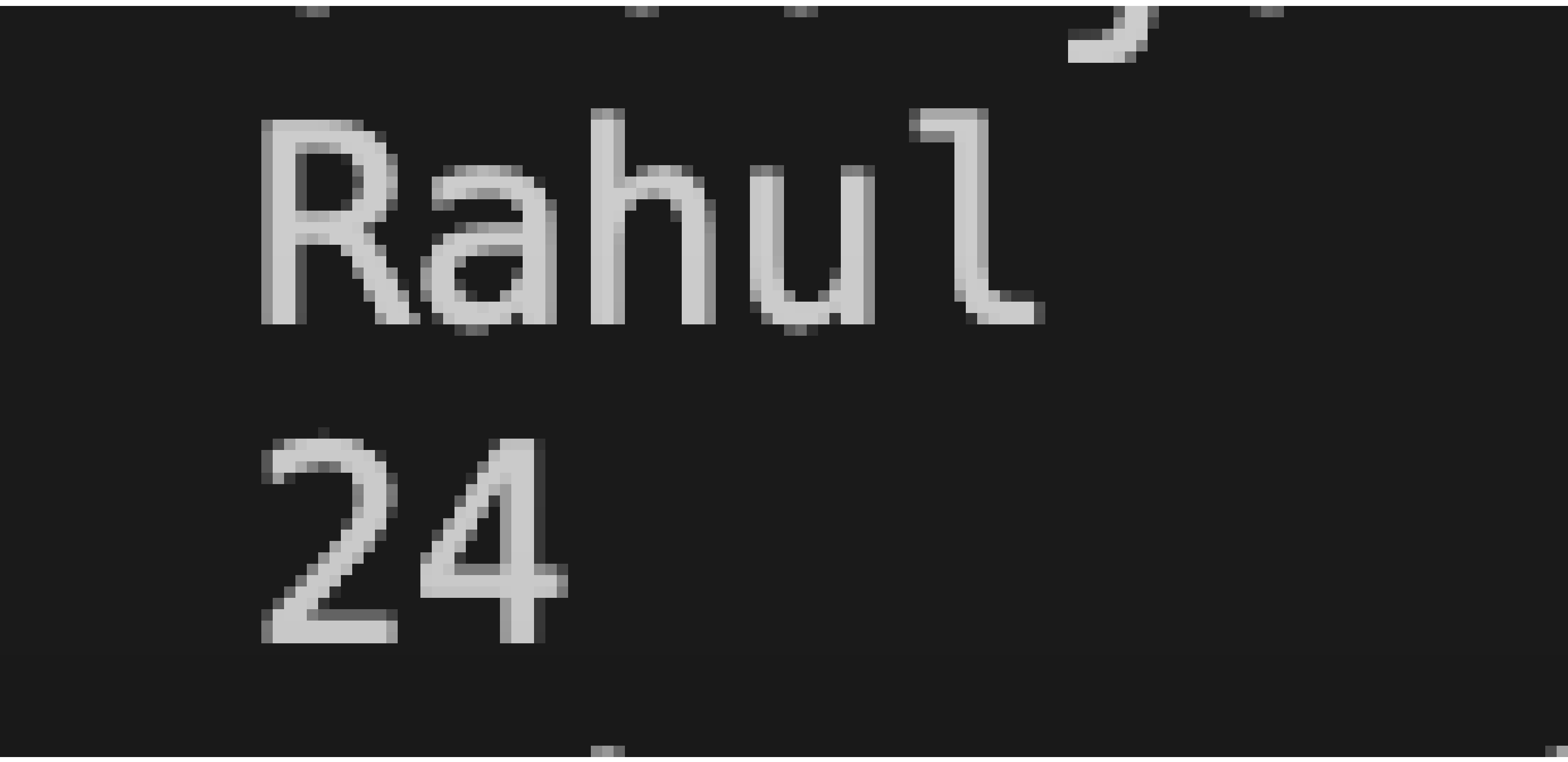
```
Constructor is called
Rahul
35
```



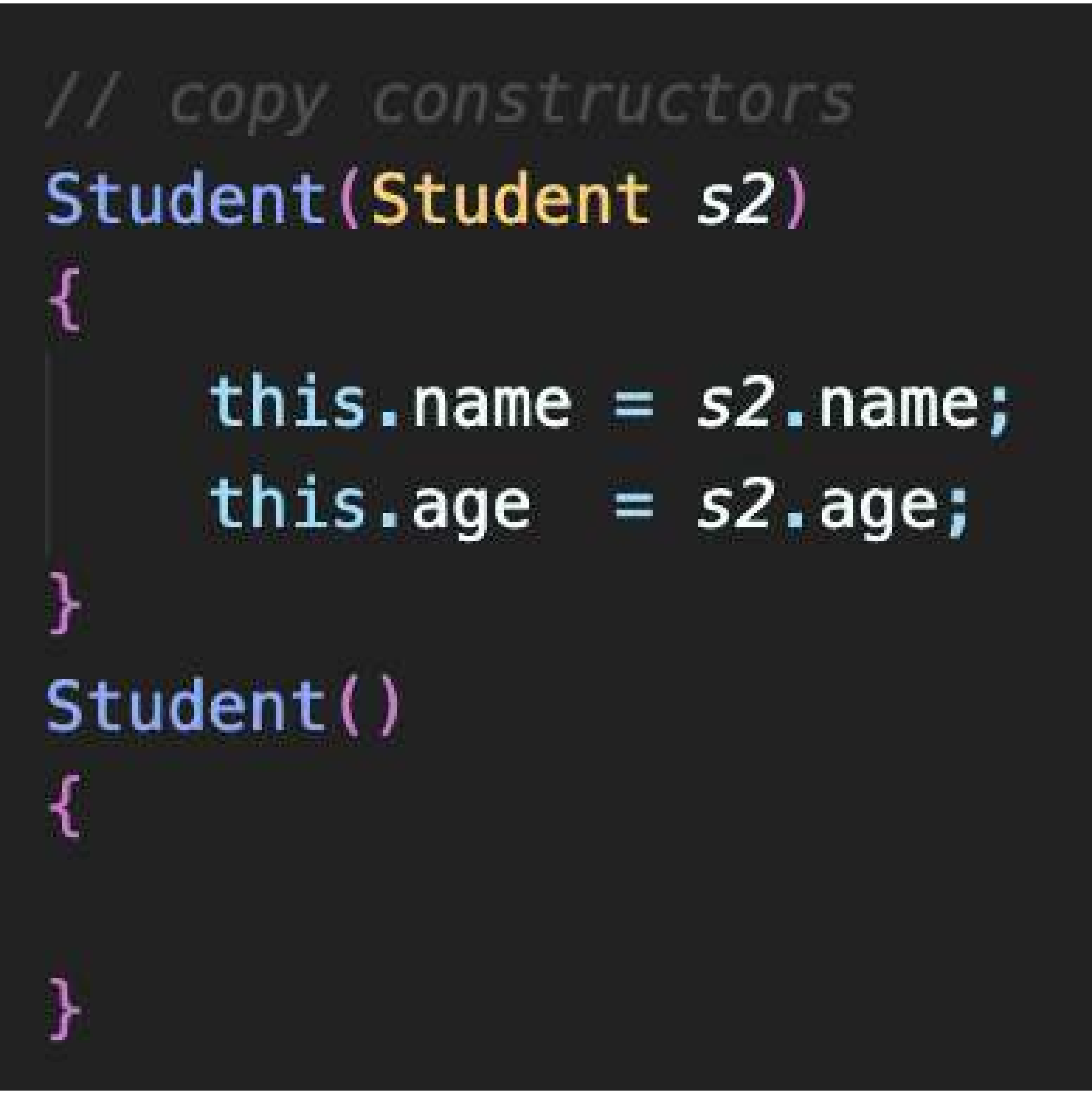
```java
// Parametric constructor
Student(String name, int age){
    this.name = name;
    this.age = age;
}
```



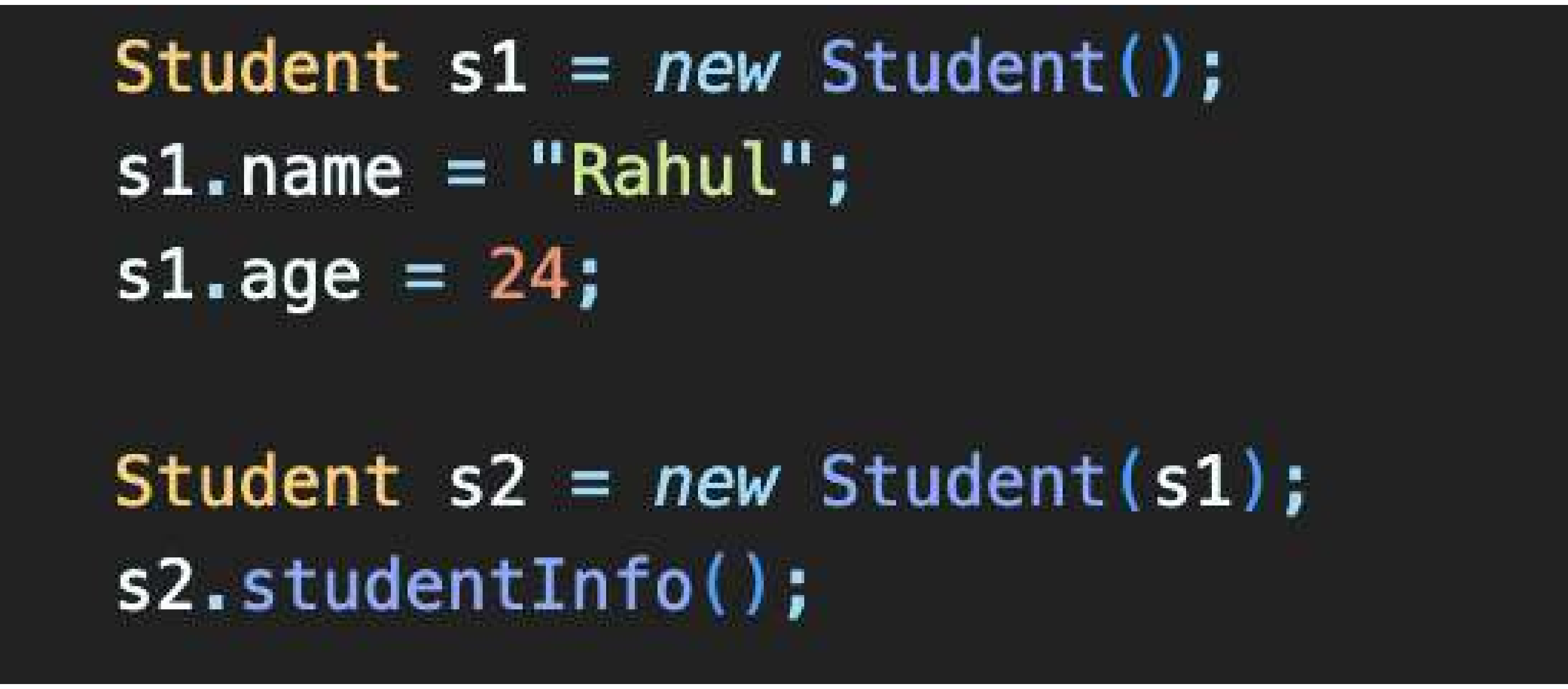
```
// for parametric constructor
Student s1 = new Student(name:"Rahul",age:24);
s1.studentInfo();
```



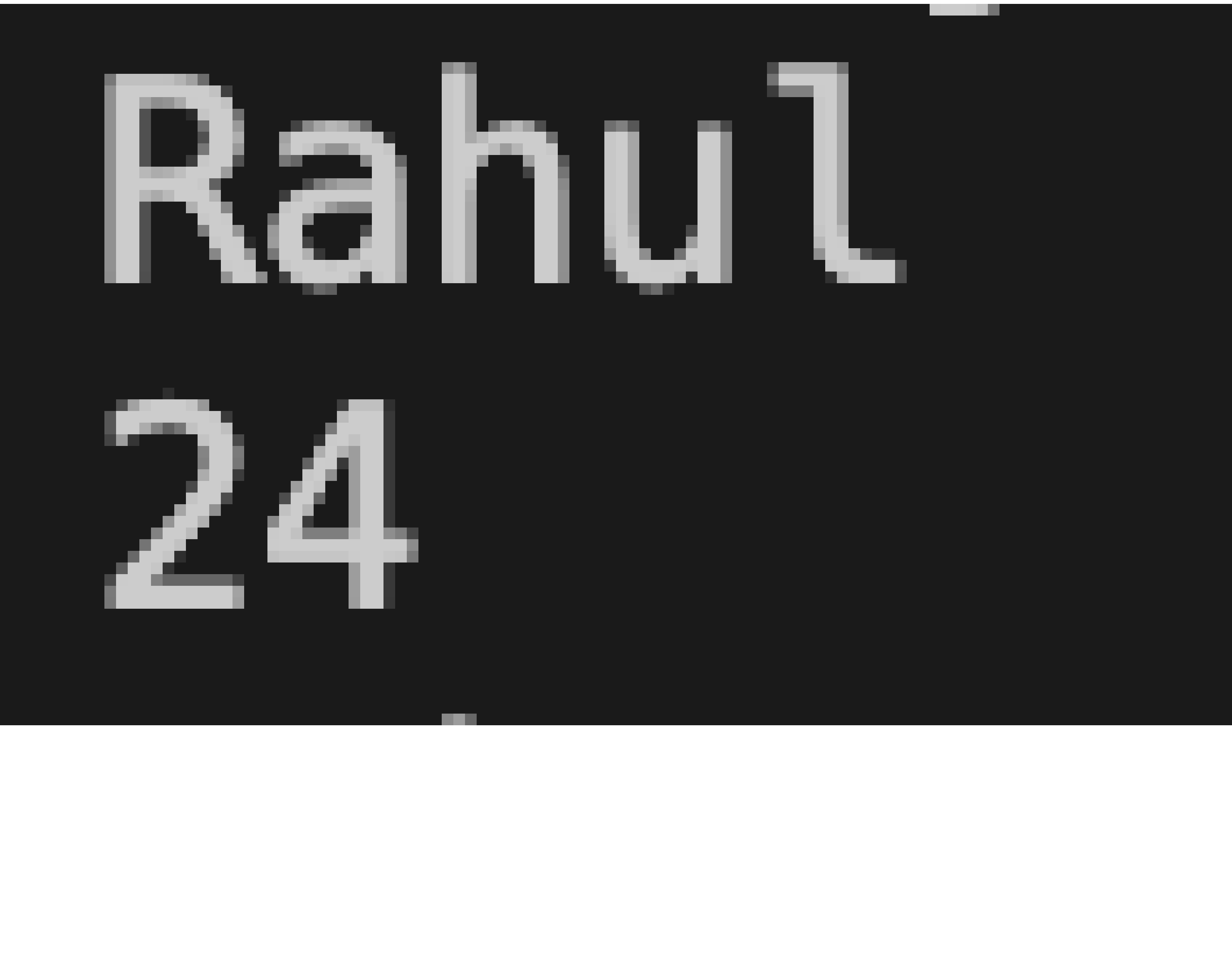
```
Rahul
24
```



```
// copy constructors
Student(Student s2)
{
    this.name = s2.name;
    this.age  = s2.age;
}
Student()
{

}
```



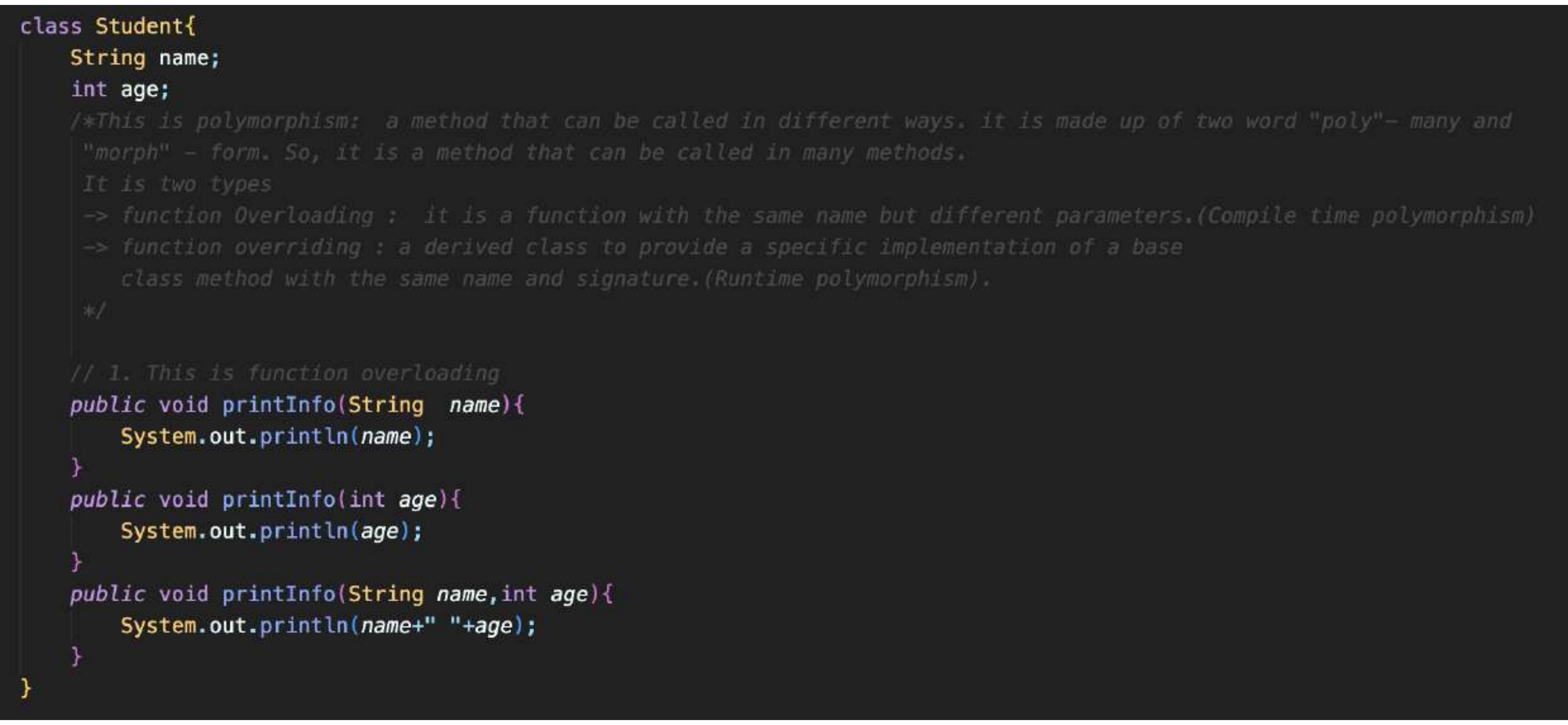
```java
Student s1 = new Student();
s1.name = "Rahul";
s1.age = 24;

Student s2 = new Student(s1);
s2.studentInfo();
```



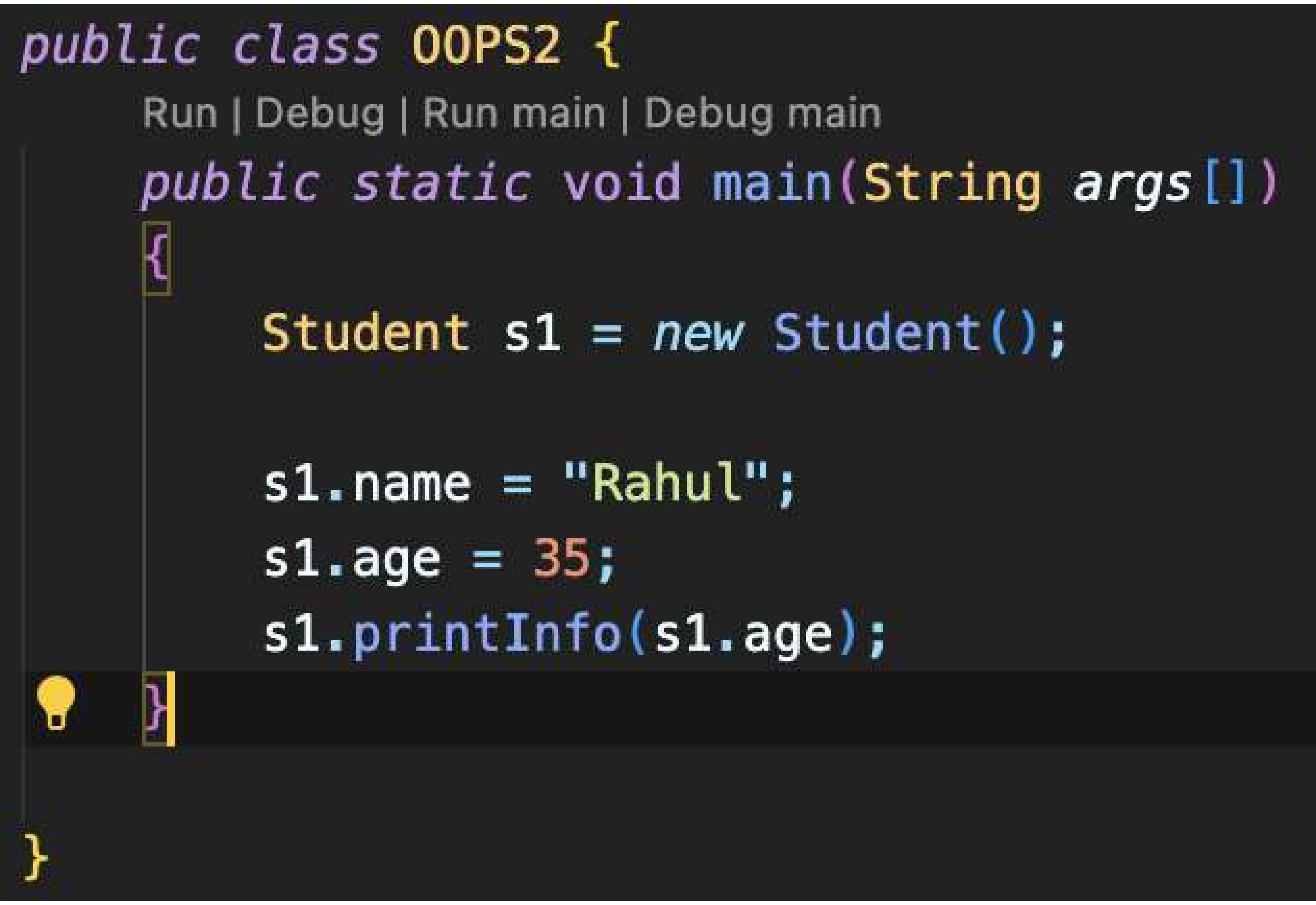
```
Rahul
24
```

Polymorphism

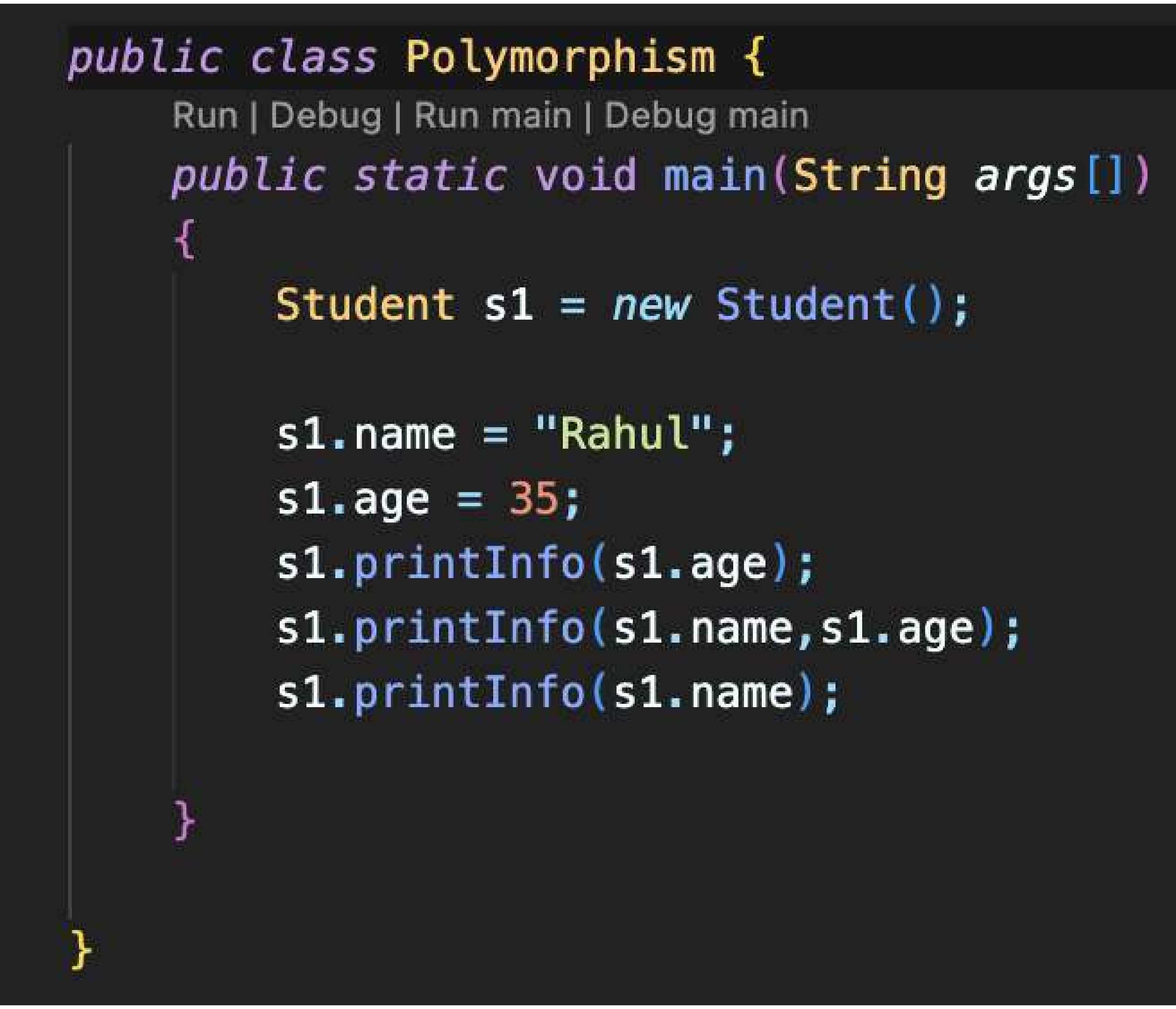
```
    System.out.println(this.name);
    System.out.println(this.age);
}

Student(Student s2) {
    this.name = s2.name;
    this.age = s2.age;
}

Student() {
```



```java
class Student{
    String name;
    int age;
    /*This is polymorphism:  a method that can be called in different ways. it is made up of two word "poly"- many and
     "morph" - form. So, it is a method that can be called in many methods.
     It is two types
     -> function Overloading :  it is a function with the same name but different parameters.(Compile time polymorphism)
     -> function overriding : a derived class to provide a specific implementation of a base
        class method with the same name and signature.(Runtime polymorphism).
     */

    // 1. This is function overloading
    public void printInfo(String  name){
        System.out.println(name);
    }
    public void printInfo(int age){
        System.out.println(age);
    }
    public void printInfo(String name,int age){
        System.out.println(name+" "+age);
    }
}
```



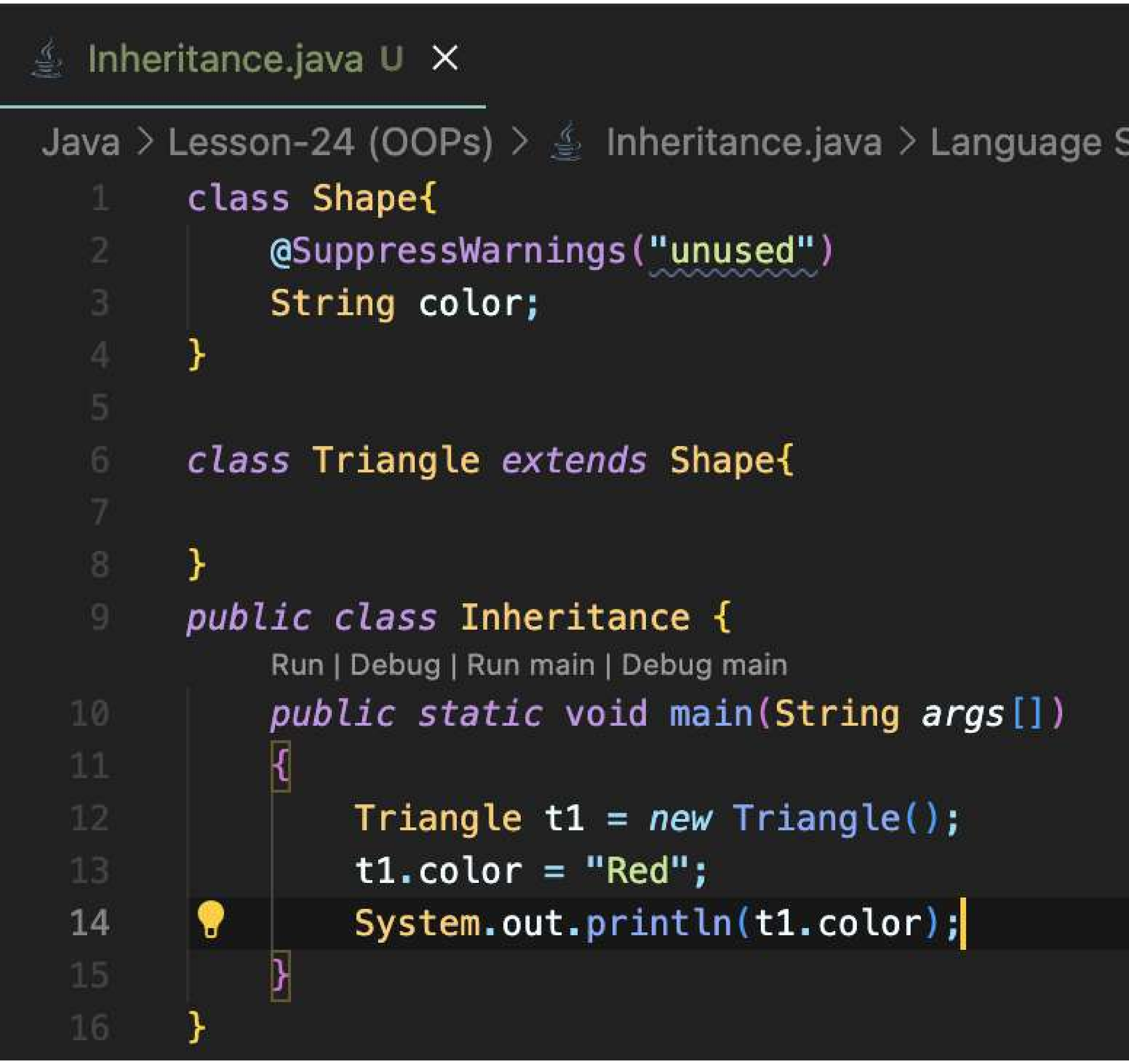
```java
public class OOPS2 {
    public static void main(String args[])
    {
        Student s1 = new Student();

        s1.name = "Rahul";
        s1.age = 35;
        s1.printInfo(s1.age);
    }

}
```



```java
public class Polymorphism {
    public static void main(String args[])
    {
        Student s1 = new Student();

        s1.name = "Rahul";
        s1.age = 35;
        s1.printInfo(s1.age);
        s1.printInfo(s1.name,s1.age);
        s1.printInfo(s1.name);

    }

}
```

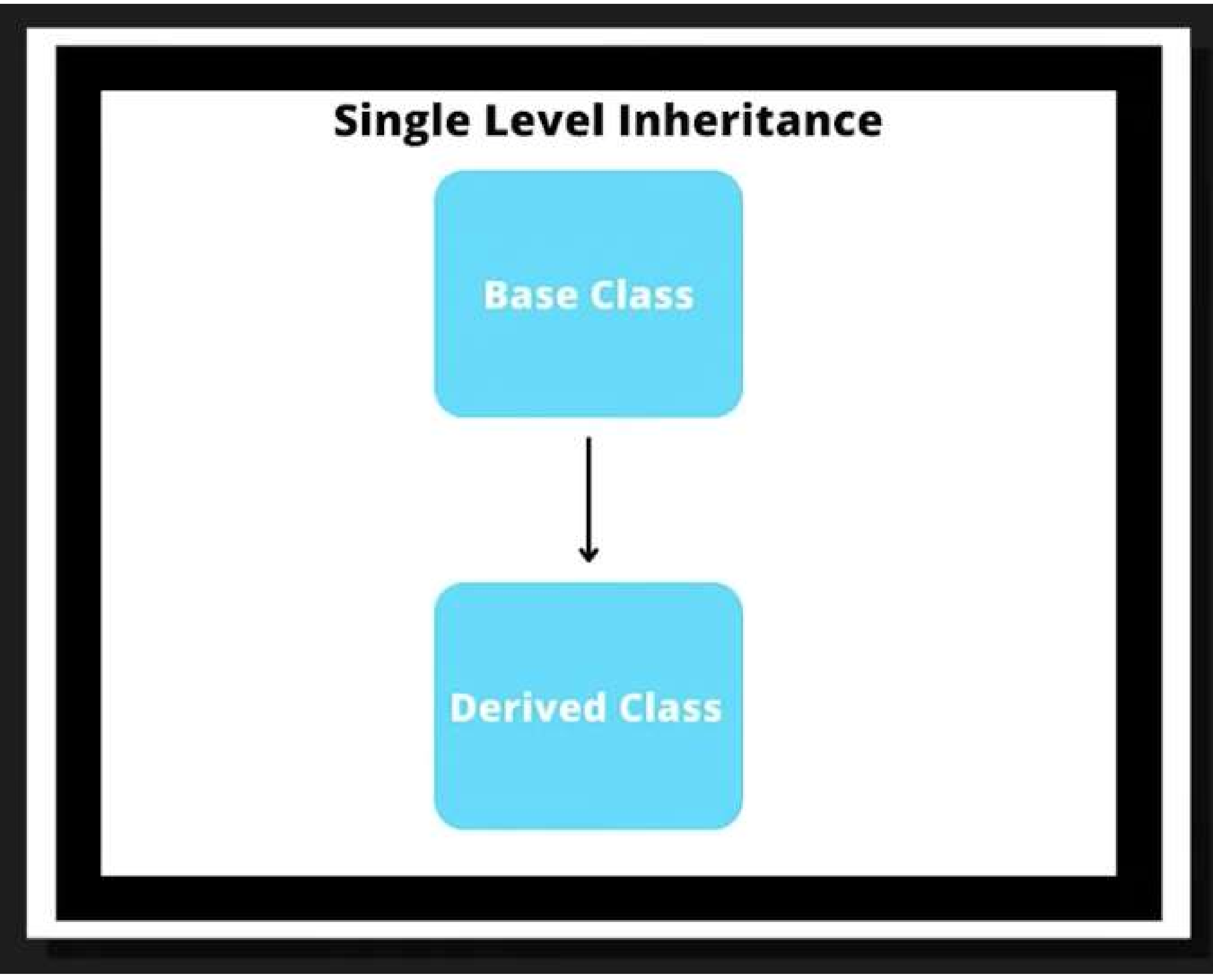
```
35
Rahul  35
Rahul
```

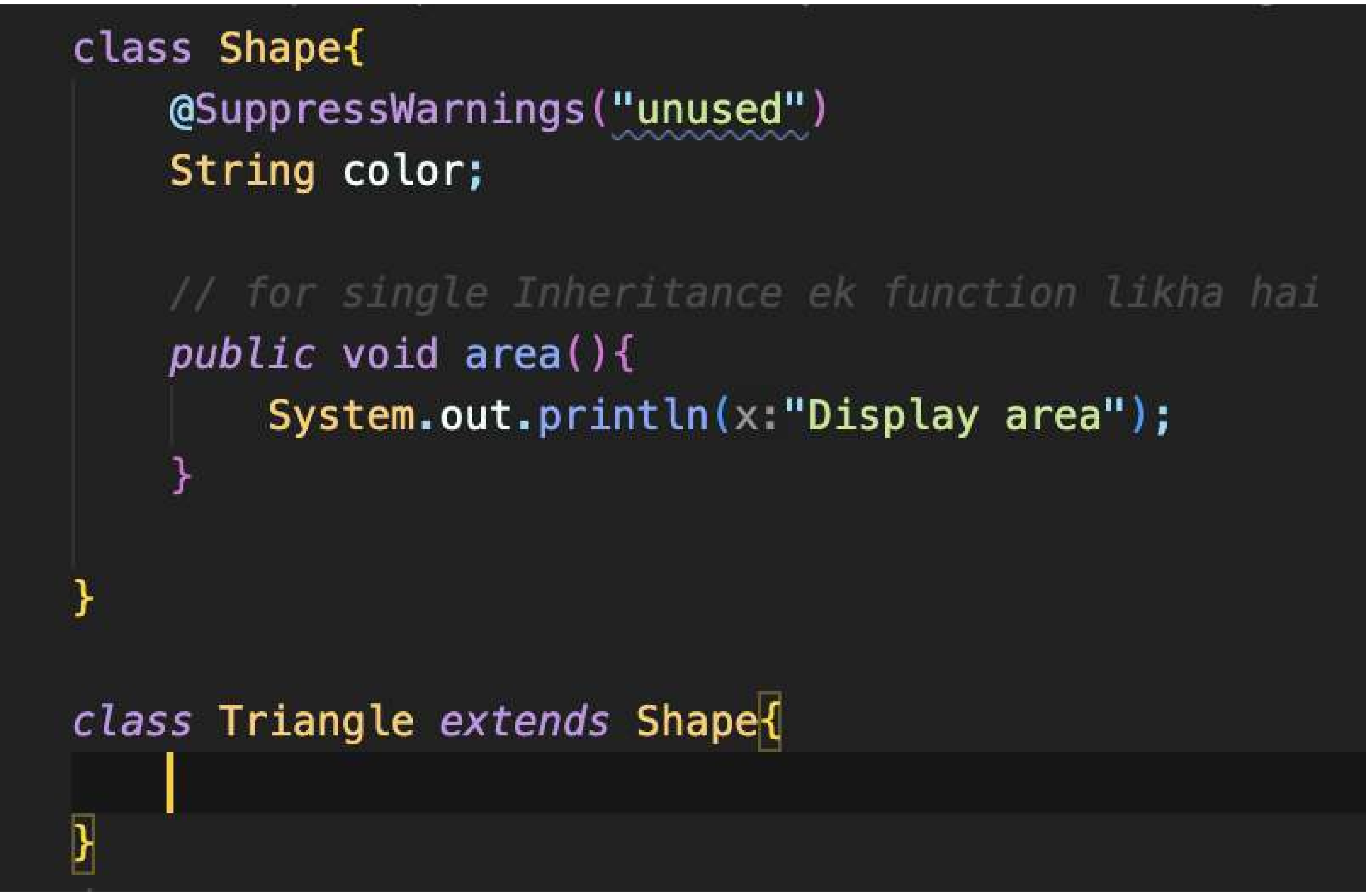
Inheritance
oid main(String args[]) {


Inheritance.java U
Java > Lesson-24 (OOPs) > Inheritance.java > Language S
Run | Debug | Run main | Debug main

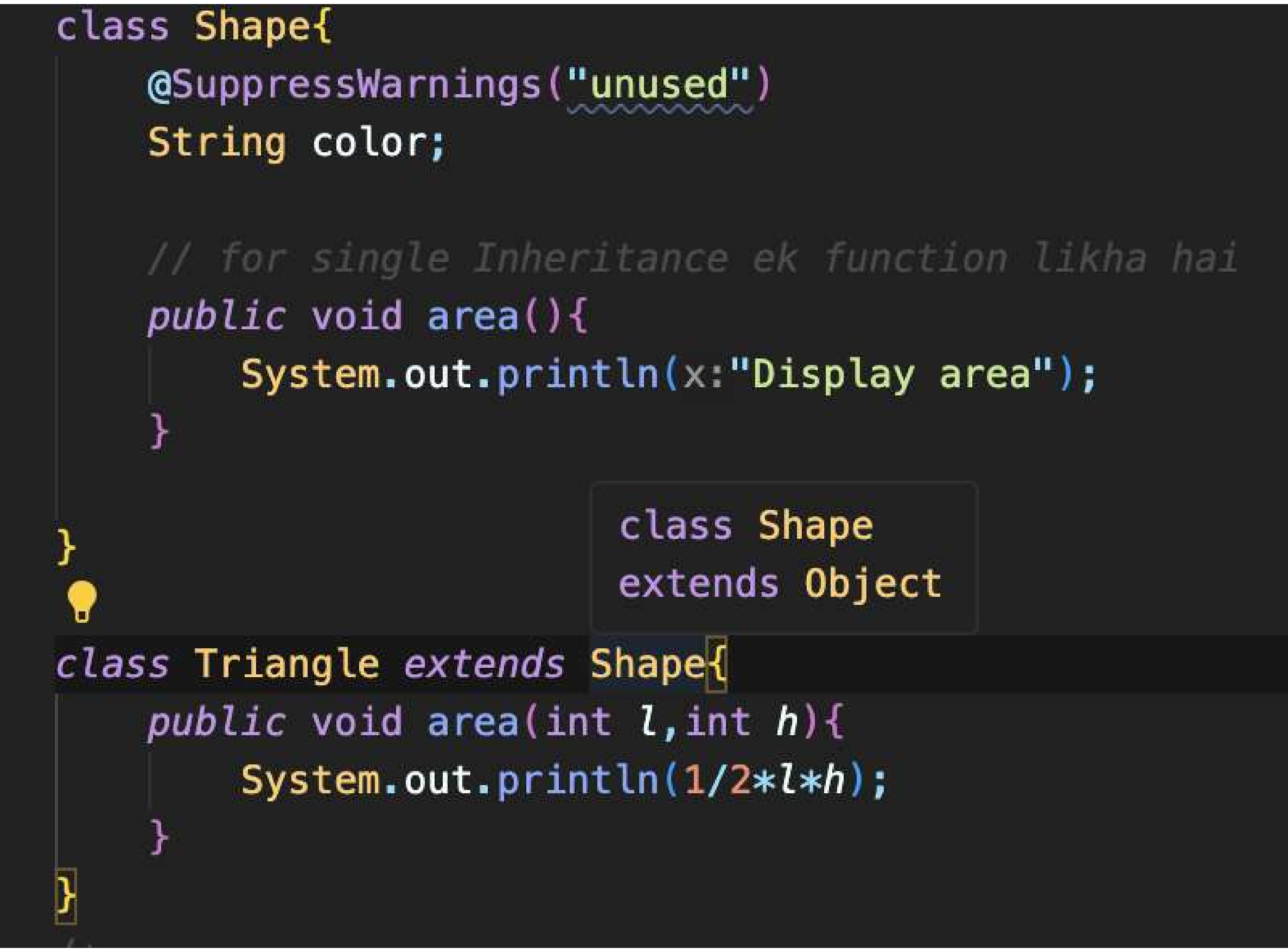
```java
class Shape{
    @SuppressWarnings("unused")
    String color;
}

class Triangle extends Shape{

}
public class Inheritance {
    public static void main(String args[])
    {
        Triangle t1 = new Triangle();
        t1.color = "Red";
        System.out.println(t1.color);
    }
}
```

Red

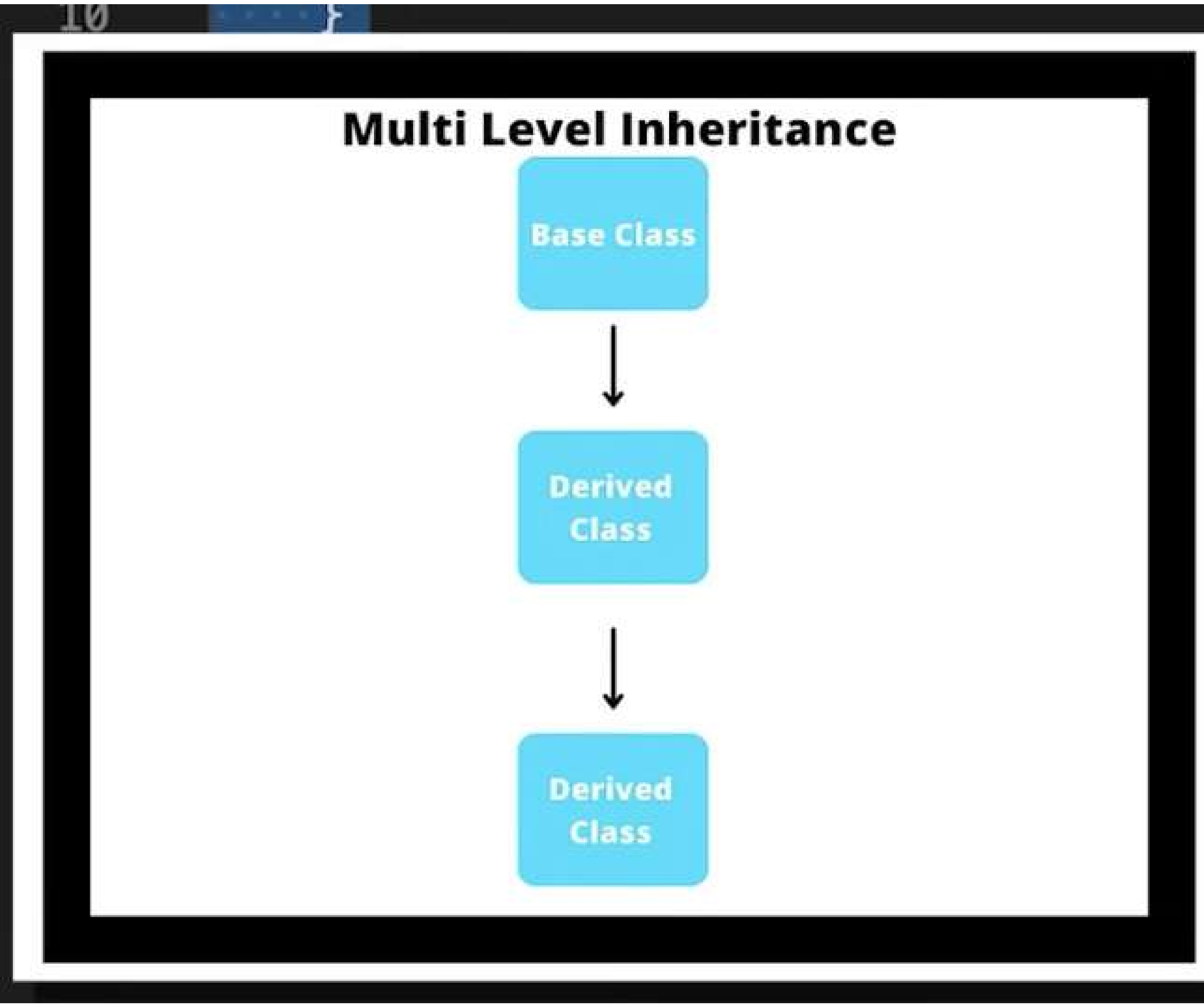

Single Level Inheritance
Base Class
Derived Class

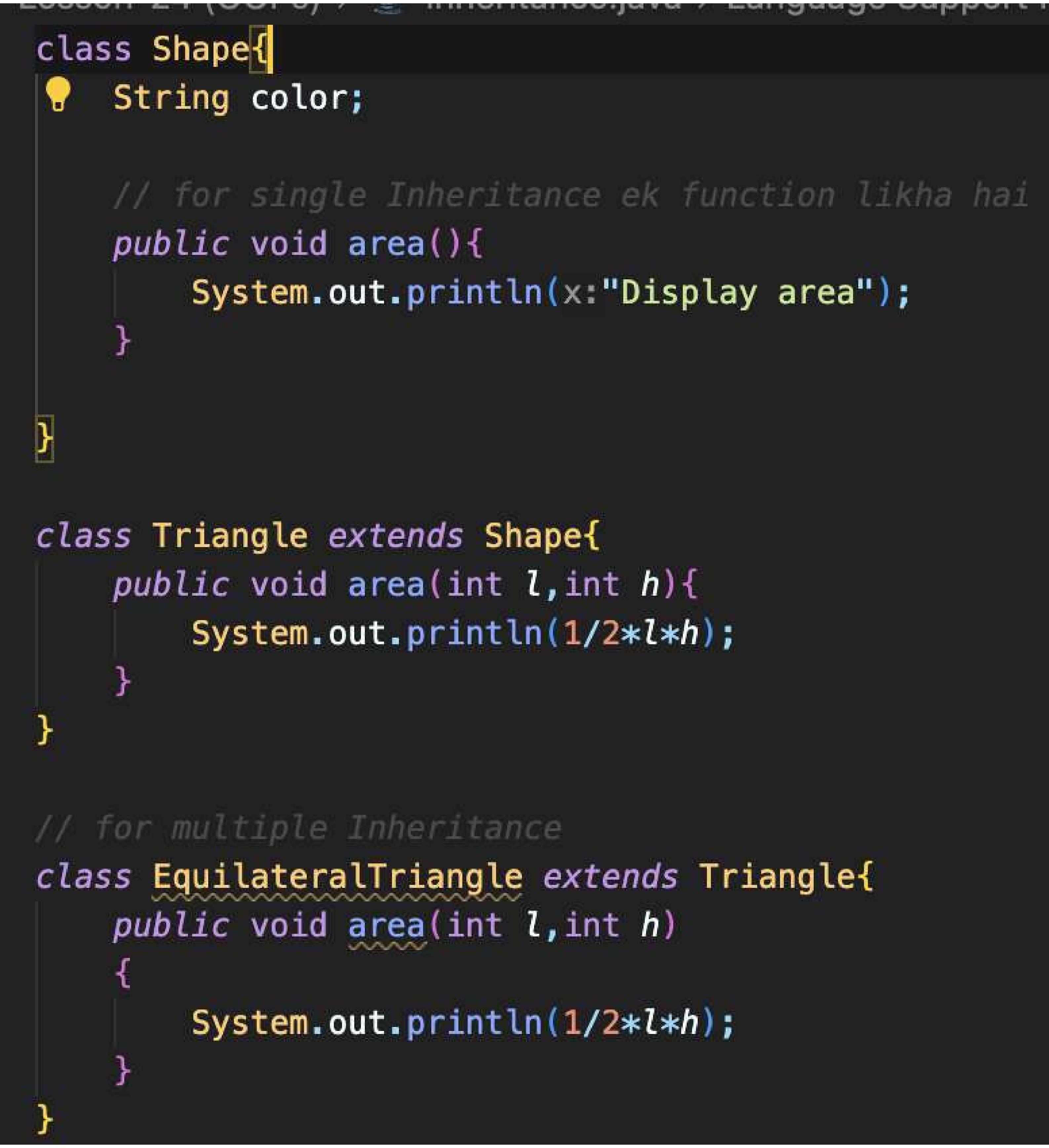


```
class Shape{
    @SuppressWarnings("unused")
    String color;

    // for single Inheritance ek function likha hai
    public void area(){
        System.out.println(x:"Display area");
    }

}

class Triangle extends Shape{
    
}
```



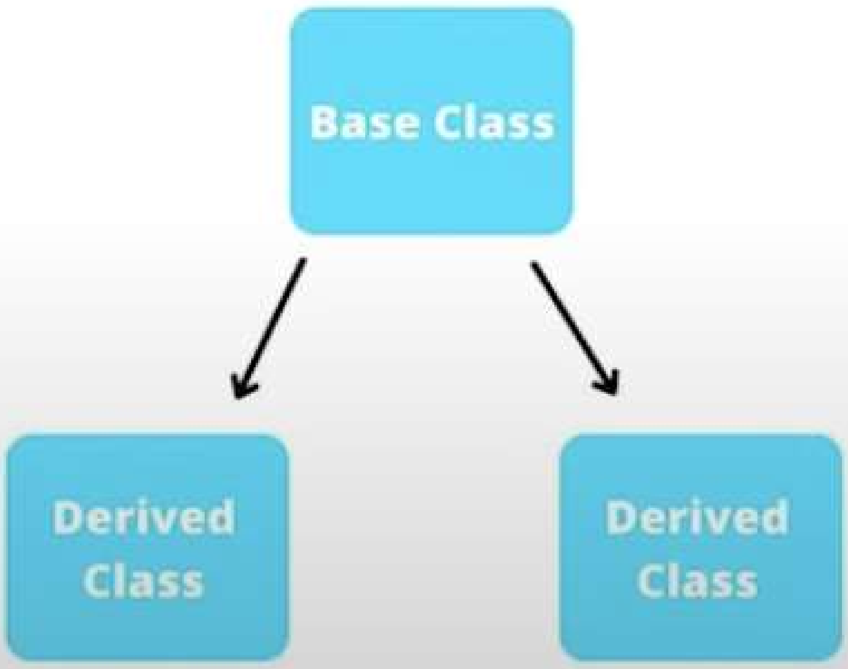
```
class Shape{
    @SuppressWarnings("unused")
    String color;

    // for single Inheritance ek function likha hai
    public void area(){
        System.out.println(x:"Display area");
    }

}
class Triangle extends Shape{
    public void area(int l,int h){
        System.out.println(1/2*l*h);
    }
}
```

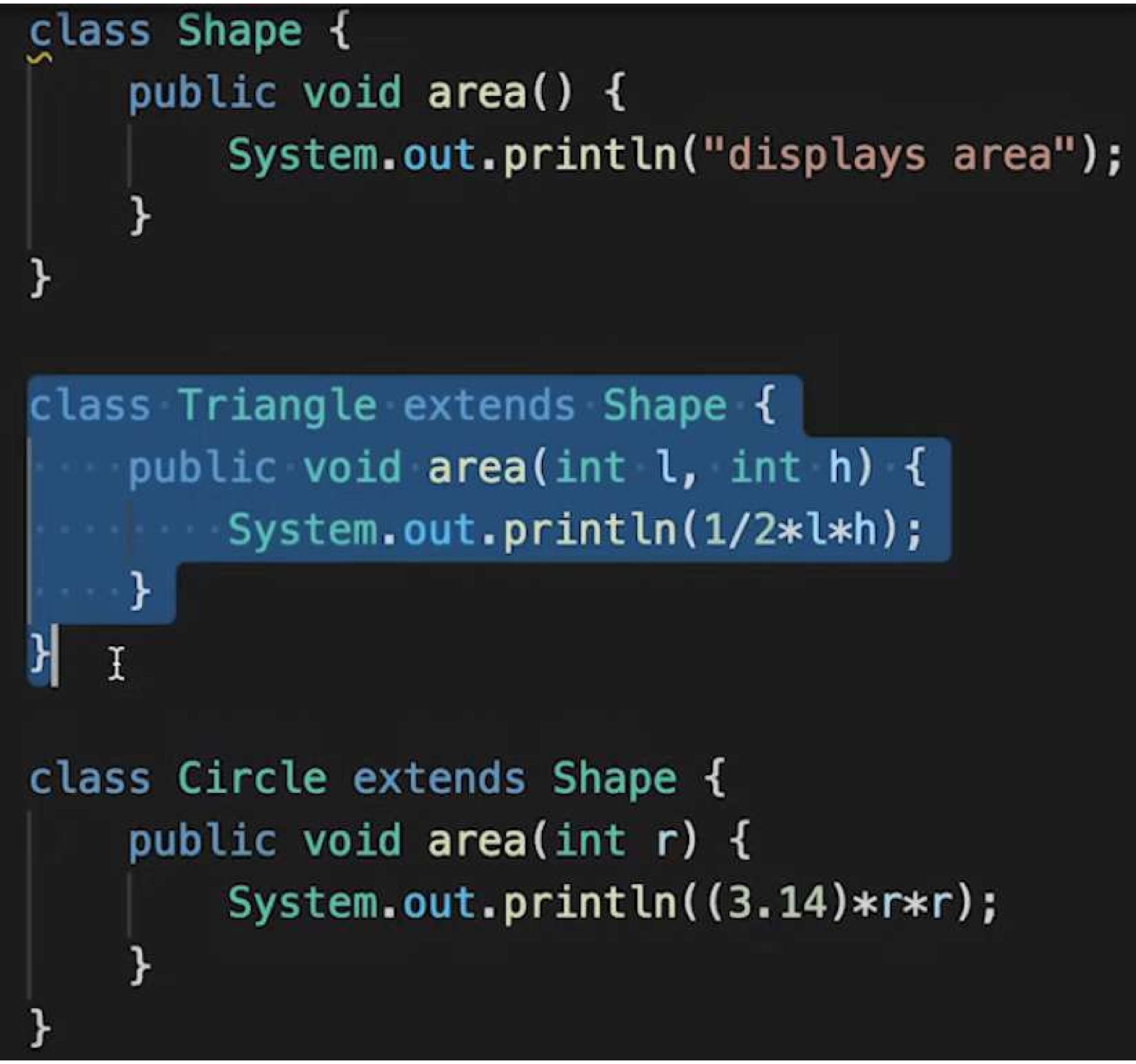

Multi Level Inheritance
Base Class
Derived Class
Derived Class



```java
class Shape{
    String color;

    // for single Inheritance ek function likha hai
    public void area(){
        System.out.println(x:"Display area");
    }

}

class Triangle extends Shape{
    public void area(int l,int h){
        System.out.println(1/2*l*h);
    }
}

// for multiple Inheritance
class EquilateralTriangle extends Triangle{
    public void area(int l,int h)
    {
        System.out.println(1/2*l*h);
    }
}
```

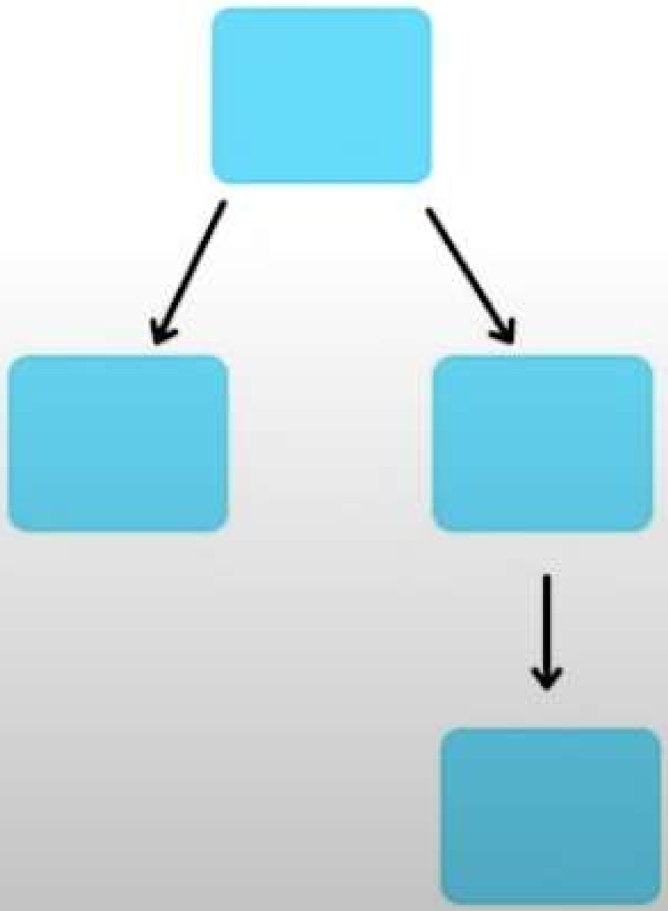
Hierarchial Inheritance

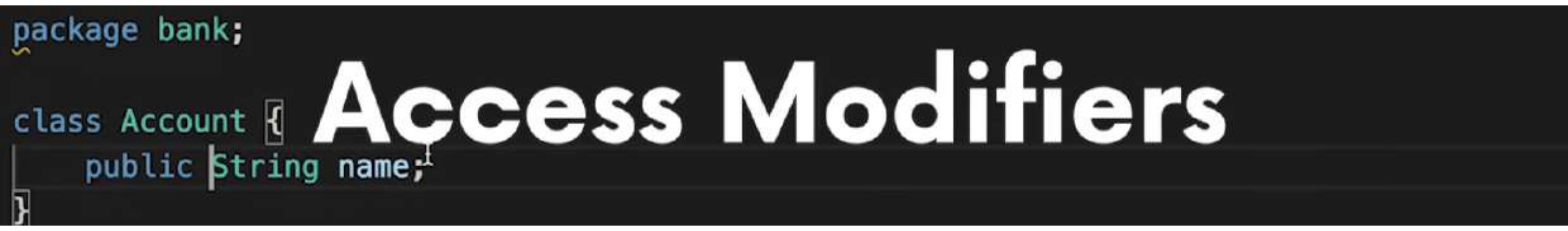




```java
class Shape {
    public void area() {
        System.out.println("displays area");
    }
}

class Triangle extends Shape {
    public void area(int l, int h) {
        System.out.println(1/2*l*h);
    }
}

class Circle extends Shape {
    public void area(int r) {
        System.out.println((3.14)*r*r);
    }
}
```

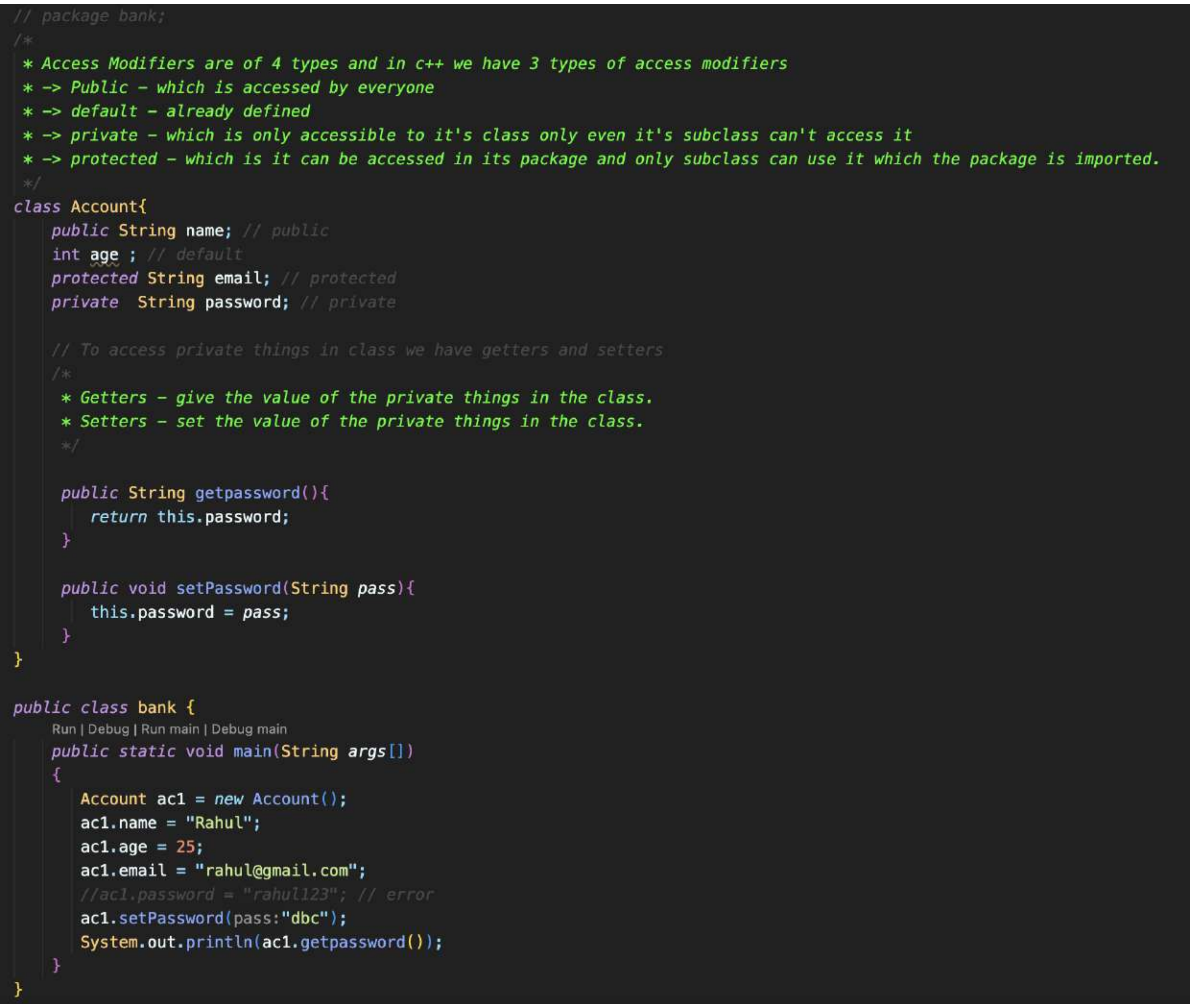
Hybrid Inheritance





Access Modifiers

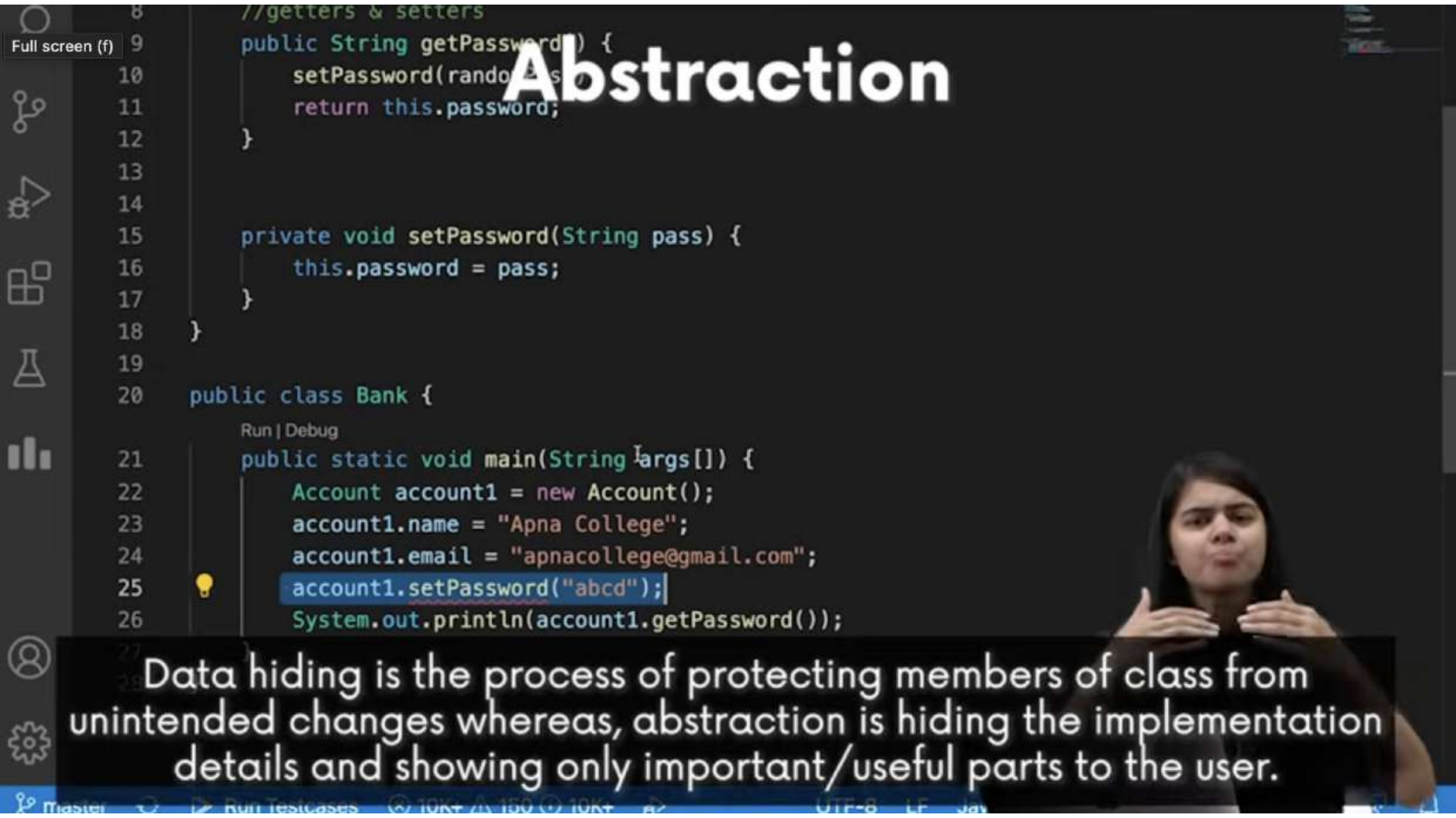
```
package bank;

class Account {
    public String name;
}
```



```
// package bank;
/*
 * Access Modifiers are of 4 types and in c++ we have 3 types of access modifiers
 * -> Public - which is accessed by everyone
 * -> default - already defined
 * -> private - which is only accessible to it's class only even it's subclass can't access it
 * -> protected - which is it can be accessed in its package and only subclass can use it which the package is imported.
 */
class Account{
    public String name; // public
    int age ; // default
    protected String email; // protected
    private  String password; // private

    // To access private things in class we have getters and setters
    /*
     * Getters - give the value of the private things in the class.
     * Setters - set the value of the private things in the class.
     */

     public String getpassword(){
        return this.password;
     }

     public void setPassword(String pass){
        this.password = pass;
     }
}

public class bank {
    Run | Debug | Run main | Debug main
    public static void main(String args[])
    {
        Account ac1 = new Account();
        ac1.name = "Rahul";
        ac1.age = 25;
        ac1.email = "rahul@gmail.com";
        //ac1.password = "rahul123"; // error
        ac1.setPassword(pass:"dbc");
        System.out.println(ac1.getpassword());
    }
}
```

```
g getPassword() {
ord(r
his.password;
```

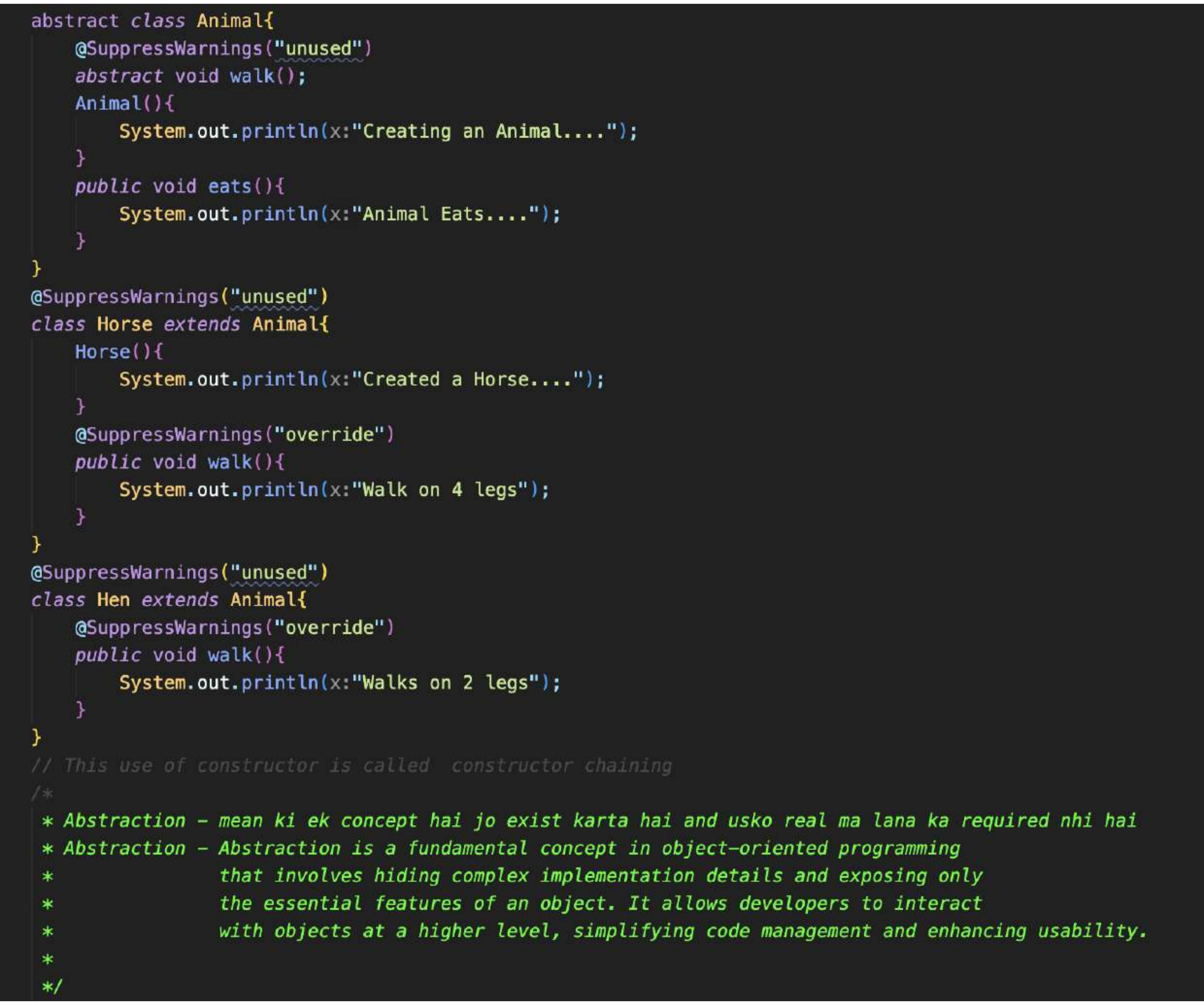
```
 getPassword() {
rd(rand
is.password;
```



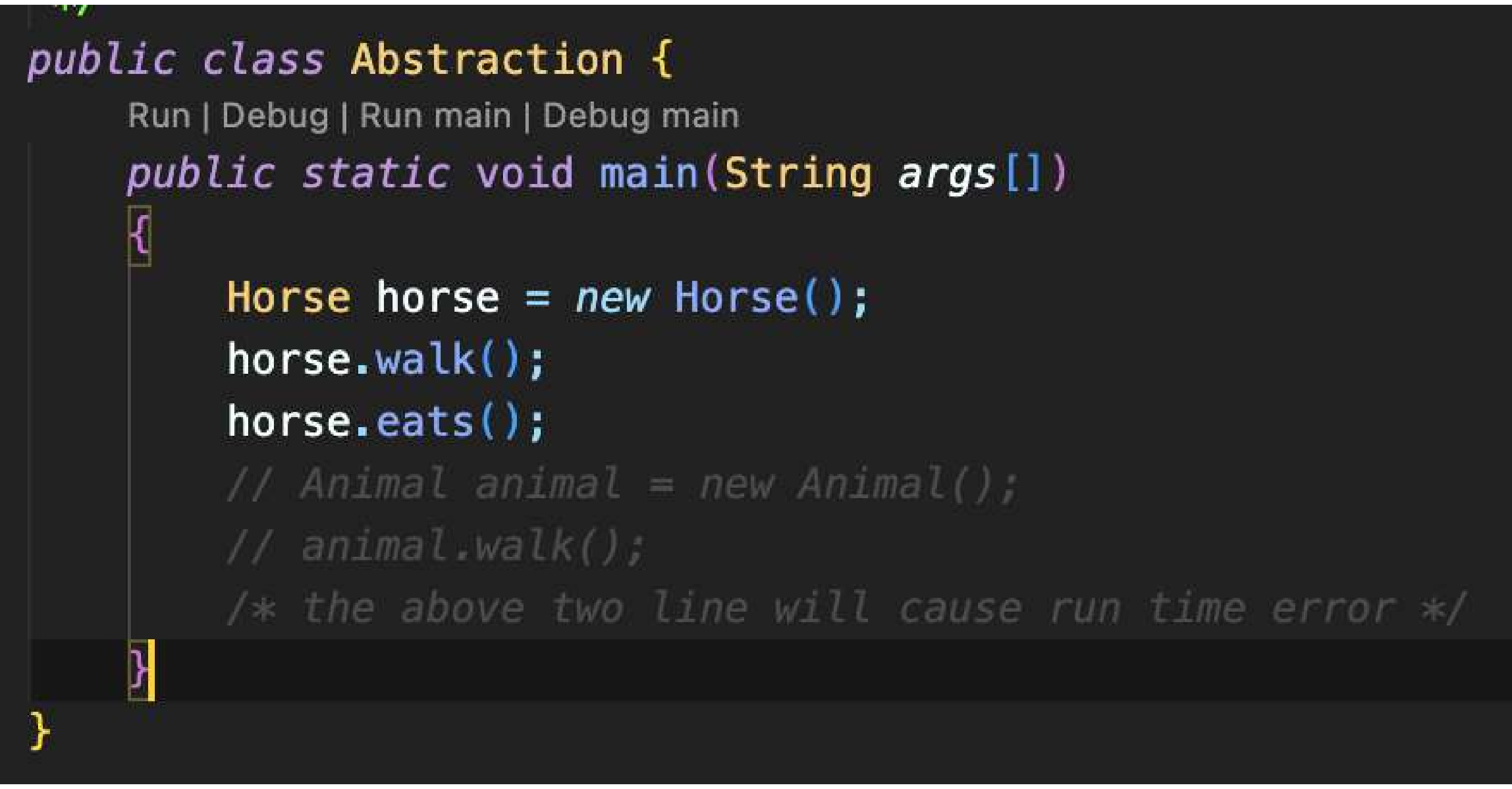
Abstraction

```
    //getters & setters
    public String getPassword() {
        setPassword(rando
        return this.password;
    }


    private void setPassword(String pass) {
        this.password = pass;
    }
}

public class Bank {
    public static void main(String args[]) {
        Account account1 = new Account();
        account1.name = "Apna College";
        account1.email = "apnacollege@gmail.com";
        account1.setPassword("abcd");
        System.out.println(account1.getPassword());
```

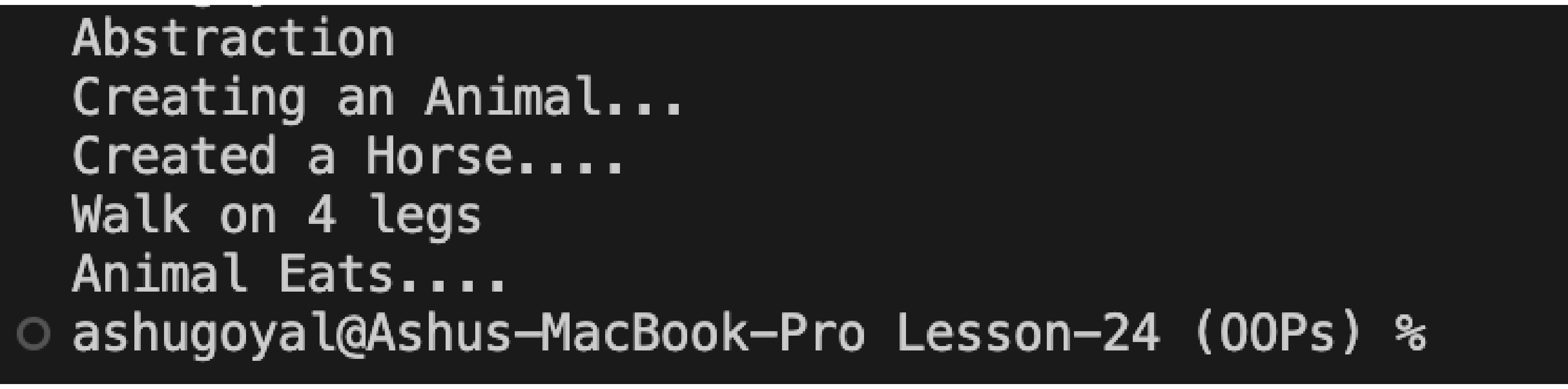
Data hiding is the process of protecting members of class from unintended changes whereas, abstraction is hiding the implementation details and showing only important/useful parts to the user.



```java
abstract class Animal{
    @SuppressWarnings("unused")
    abstract void walk();
    Animal(){
        System.out.println(x:"Creating an Animal....");
    }
    public void eats(){
        System.out.println(x:"Animal Eats....");
    }
}
@SuppressWarnings("unused")
class Horse extends Animal{
    Horse(){
        System.out.println(x:"Created a Horse....");
    }
    @SuppressWarnings("override")
    public void walk(){
        System.out.println(x:"Walk on 4 legs");
    }
}
@SuppressWarnings("unused")
class Hen extends Animal{
    @SuppressWarnings("override")
    public void walk(){
        System.out.println(x:"Walks on 2 legs");
    }
}
// This use of constructor is called  constructor chaining
/*
 * Abstraction - mean ki ek concept hai jo exist karta hai and usko real ma lana ka required nhi hai
 * Abstraction - Abstraction is a fundamental concept in object-oriented programming
 *               that involves hiding complex implementation details and exposing only
 *               the essential features of an object. It allows developers to interact
 *               with objects at a higher level, simplifying code management and enhancing usability.
 *
 */
```

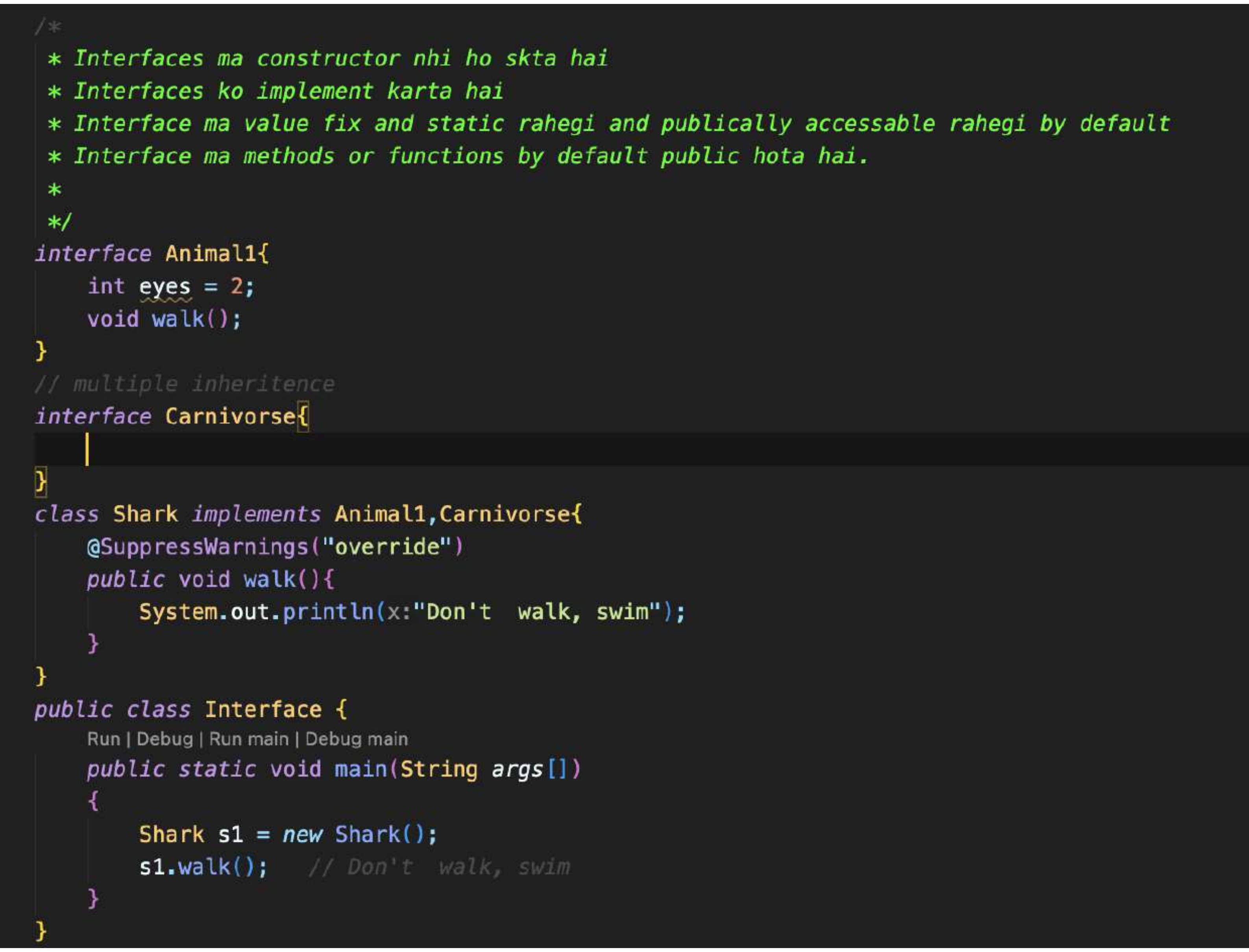


```
public class Abstraction {
    Run | Debug | Run main | Debug main
    public static void main(String args[])
    {
        Horse horse = new Horse();
        horse.walk();
        horse.eats();
        // Animal animal = new Animal();
        // animal.walk();
        /* the above two line will cause run time error */
    }
}
```

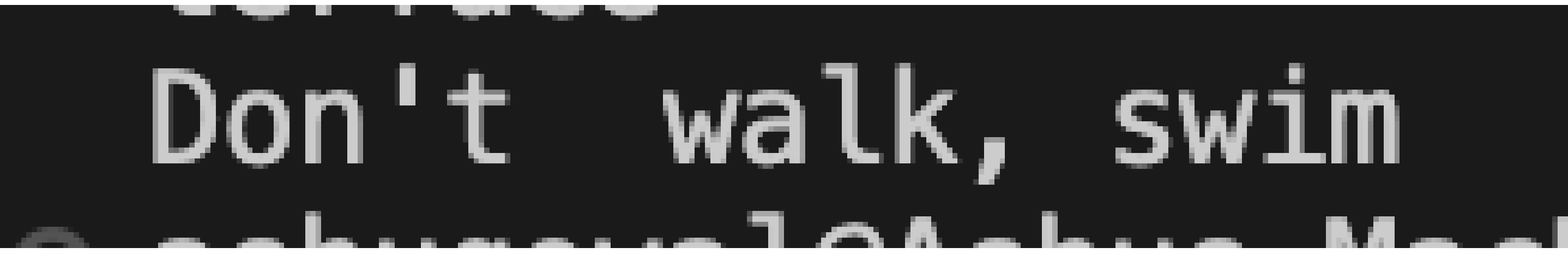


```
Abstraction
Creating an Animal...
Created a Horse....
Walk on 4 legs
Animal Eats....
ashugoyal@Ashus-MacBook-Pro Lesson-24 (OOPs) %
```

```
Animal {
oid walk();

.out.println("You are creating a new Animal");
```

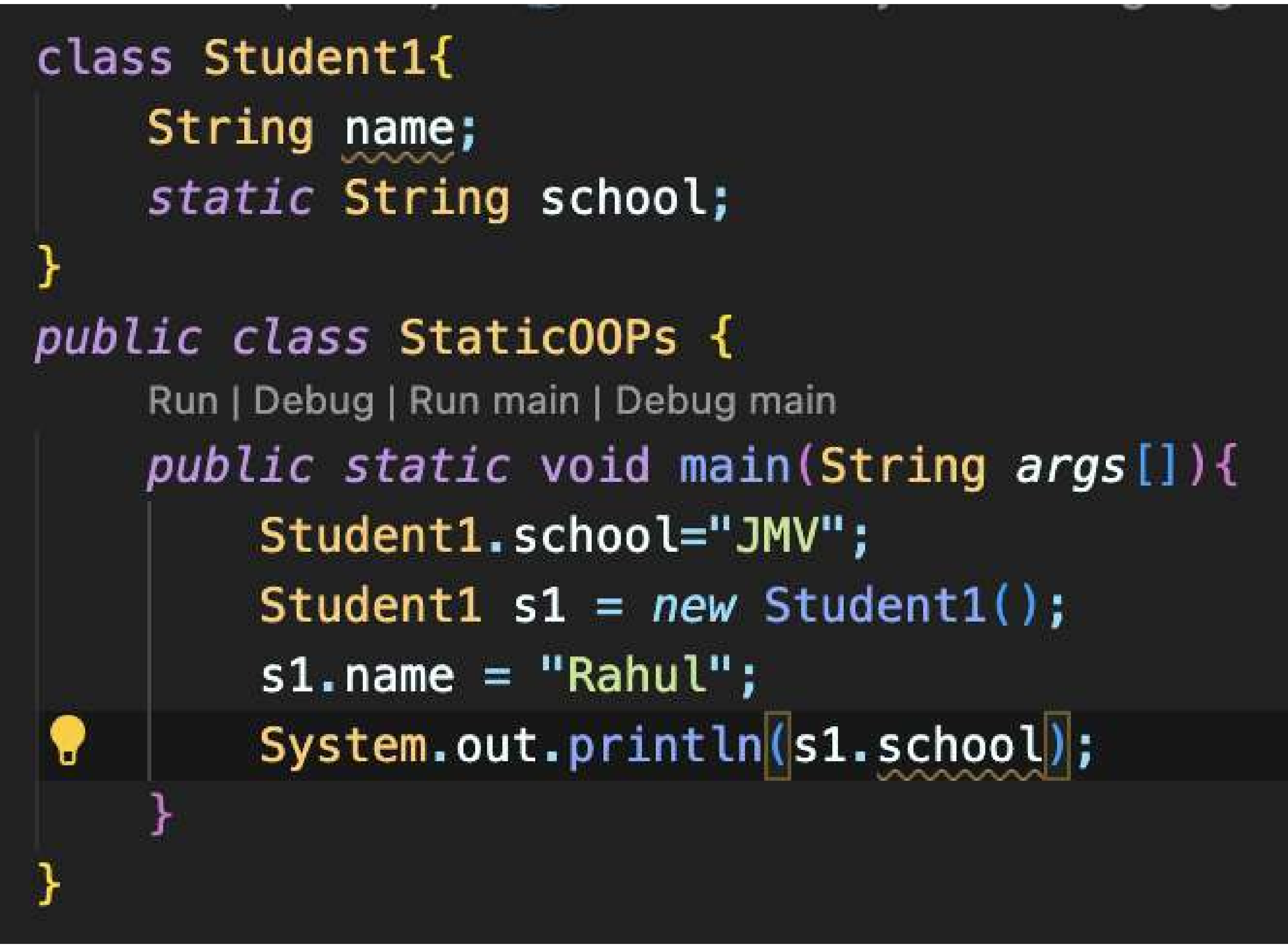


```
/*
 * Interfaces ma constructor nhi ho skta hai
 * Interfaces ko implement karta hai
 * Interface ma value fix and static rahegi and publically accessable rahegi by default
 * Interface ma methods or functions by default public hota hai.
 *
 */
interface Animal1{
    int eyes = 2;
    void walk();
}
// multiple inheritence
interface Carnivorse{
    
}
class Shark implements Animal1,Carnivorse{
    @SuppressWarnings("override")
    public void walk(){
        System.out.println(x:"Don't  walk, swim");
    }
}
public class Interface {
    Run | Debug | Run main | Debug main
    public static void main(String args[])
    {
        Shark s1 = new Shark();
        s1.walk();   // Don't  walk, swim
    }
}
```

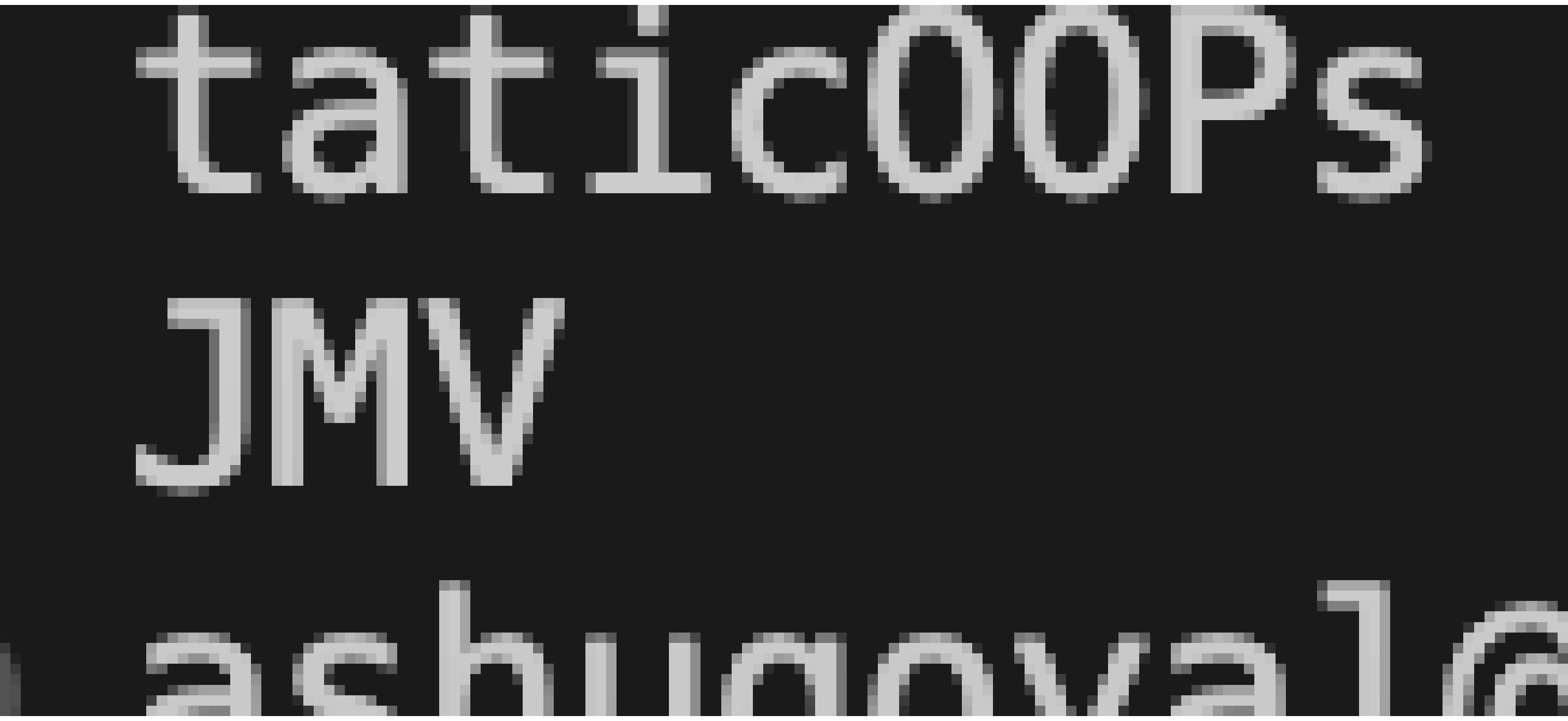


```
 Don't  walk, swim
```

Static Keyword



```
class Student1{
    String name;
    static String school;
}
public class StaticOOPs {
    Run | Debug | Run main | Debug main
    public static void main(String args[]){
        Student1.school="JMV";
        Student1 s1 = new Student1();
        s1.name = "Rahul";
        System.out.println(s1.school);
    }
}
```



```
taticOOPs
JMV
ashugoyal
```