BÜŞRA NUR AVCI

Software QA Automation Test Engineer

https://www.linkedin.com/in/b%C3%BC%C5%9Fra-nur-avci-b2592a157/

WHAT'S THE OBJECT ORIENTED PROGRAMMING SYSTEM ?

OOPs is a form of programming in which every function is objectively abstracted. To give an example; it is the way many objects we see in real life are transferred to the computer environment.

WHAT'S AN OBJECT ?

An object is a software element that contains data and methods (functions) to operate on this data. The object also carries its own functionality. Objects can be used repeatedly. In OOP (Object Oriented Programming), an object is a "small building block" that contains the properties of a class.

WHAT'S A CLASS ?

It is a system that stores variables and methods together. A class is a comprehensive combination of code consisting of objects.

THE 4 BASIC CONCEPTS OF OOP

{ ENCAPSULATION }

{ POLYMORPHISM }

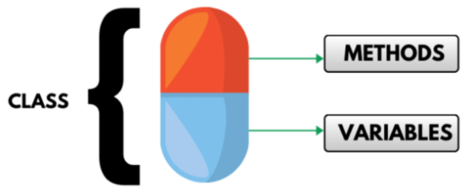
{ INHERITANCE }

{ ABSTRACTION }

1) ENCAPSULATION

Encapsulation is known as *data hiding*. To hide data we make access modifier is " *PRIVATE* ".

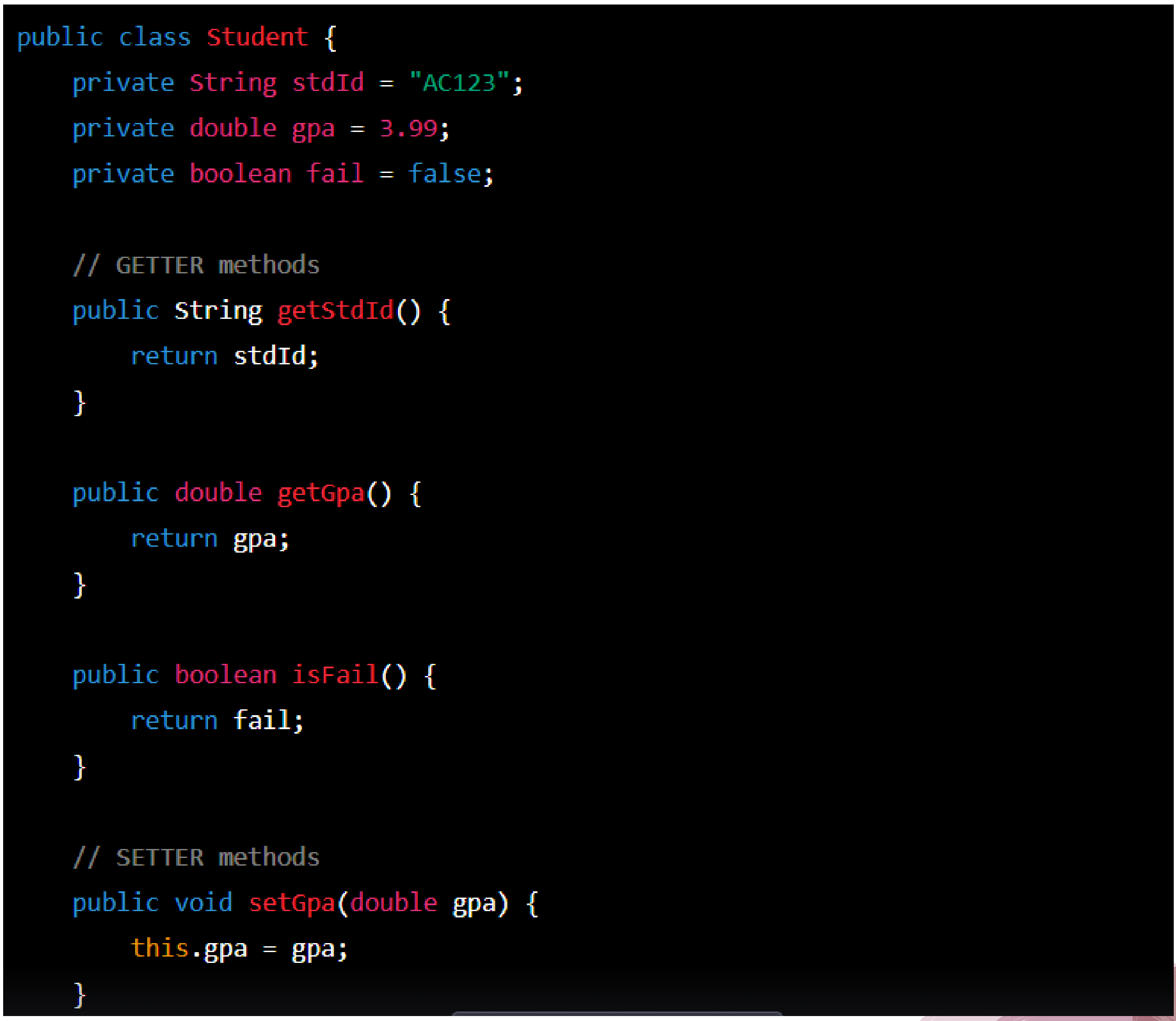
With Encapsulation, variables and methods of a class are hidden from other classes and can only be accessed through the member function of the class in which they are declared. In this way, we encapsulate the data that is important for our project, and cannot be accessed or manipulated by anyone.



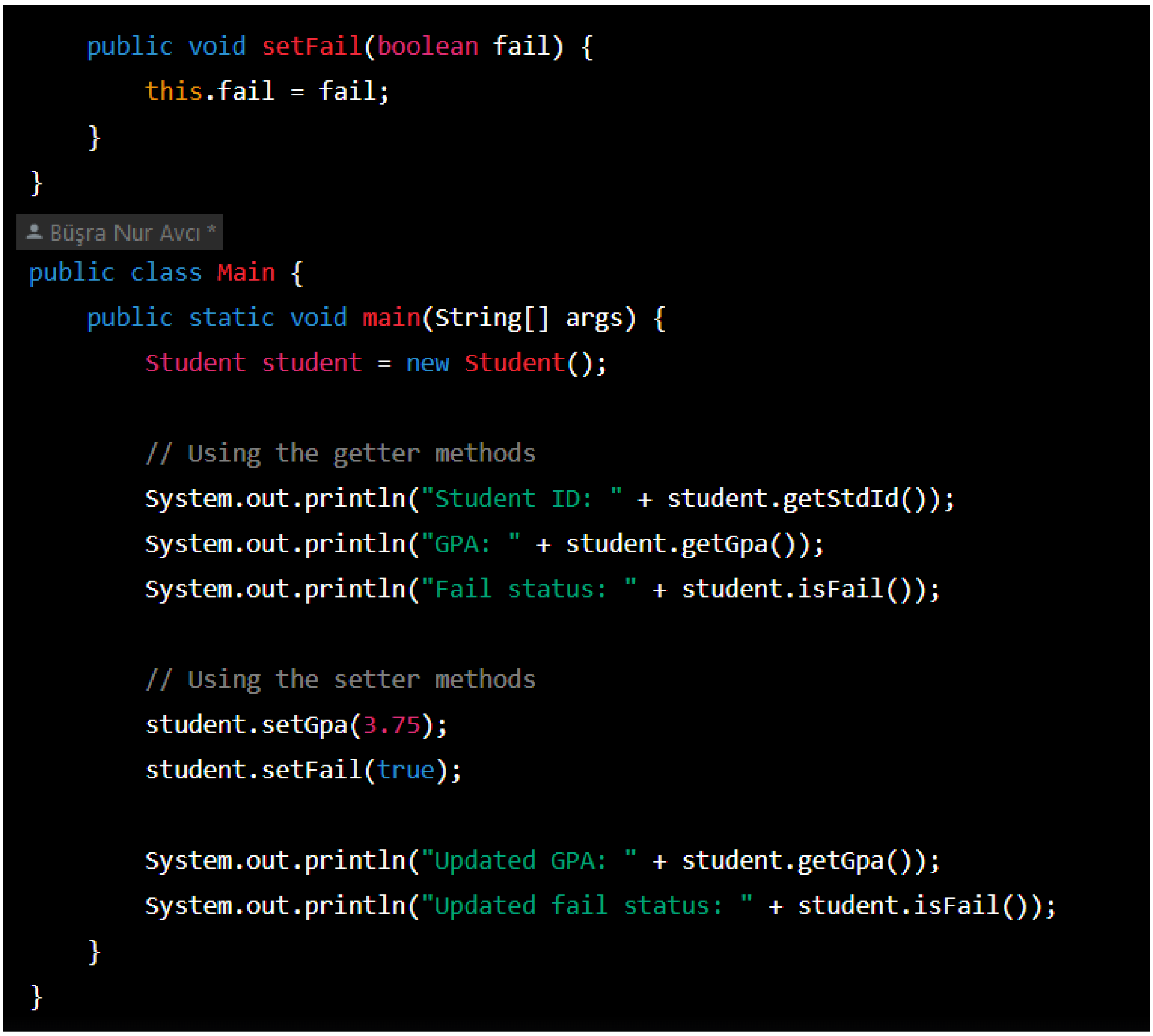
After we hiding data we may need to read or update the data.So we use some methods for those.

These are “Java Beans ”, which is mean → *GETTER && SETTER* methods.

EXAMPLE :



```
public class Student {
    private String stdId = "AC123";
    private double gpa = 3.99;
    private boolean fail = false;

    // GETTER methods
    public String getStdId() {
        return stdId;
    }

    public double getGpa() {
        return gpa;
    }

    public boolean isFail() {
        return fail;
    }

    // SETTER methods
    public void setGpa(double gpa) {
        this.gpa = gpa;
    }
```

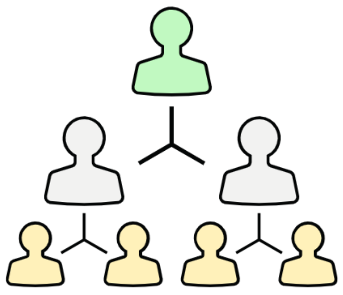


```java
    public void setFail(boolean fail) {
        this.fail = fail;
    }
}
public class Main {
    public static void main(String[] args) {
        Student student = new Student();

        // Using the getter methods
        System.out.println("Student ID: " + student.getStdId());
        System.out.println("GPA: " + student.getGpa());
        System.out.println("Fail status: " + student.isFail());

        // Using the setter methods
        student.setGpa(3.75);
        student.setFail(true);

        System.out.println("Updated GPA: " + student.getGpa());
        System.out.println("Updated fail status: " + student.isFail());
    }
}
```

2)INHERITANCE

Inheritance in OOP = When a class derives from another class. It is about "parent/child" relationship.

The 'child' class will inherit all the public and protected properties and methods from the parent class, but parent class cannot use anything from child class. In addition, it can have its own properties and methods. → An inherited class is defined by using the <u>**"extends"**</u> keyword.



$$\text{PARENT CLASS} \xrightarrow{\text{relationship}} \text{CHILD CLASS}$$

"<u>HAS-A</u>" relationship

$$\text{CHILD CLASS} \xrightarrow{\text{relationship}} \text{PARENT CLASS}$$

"<u>IS-A</u>" relationship

WHY WE USE INHERITANCE ?

- *To prevent "repetition" ,*
- *To make "maintenance" easy,*
- *Easy update and reusability.*

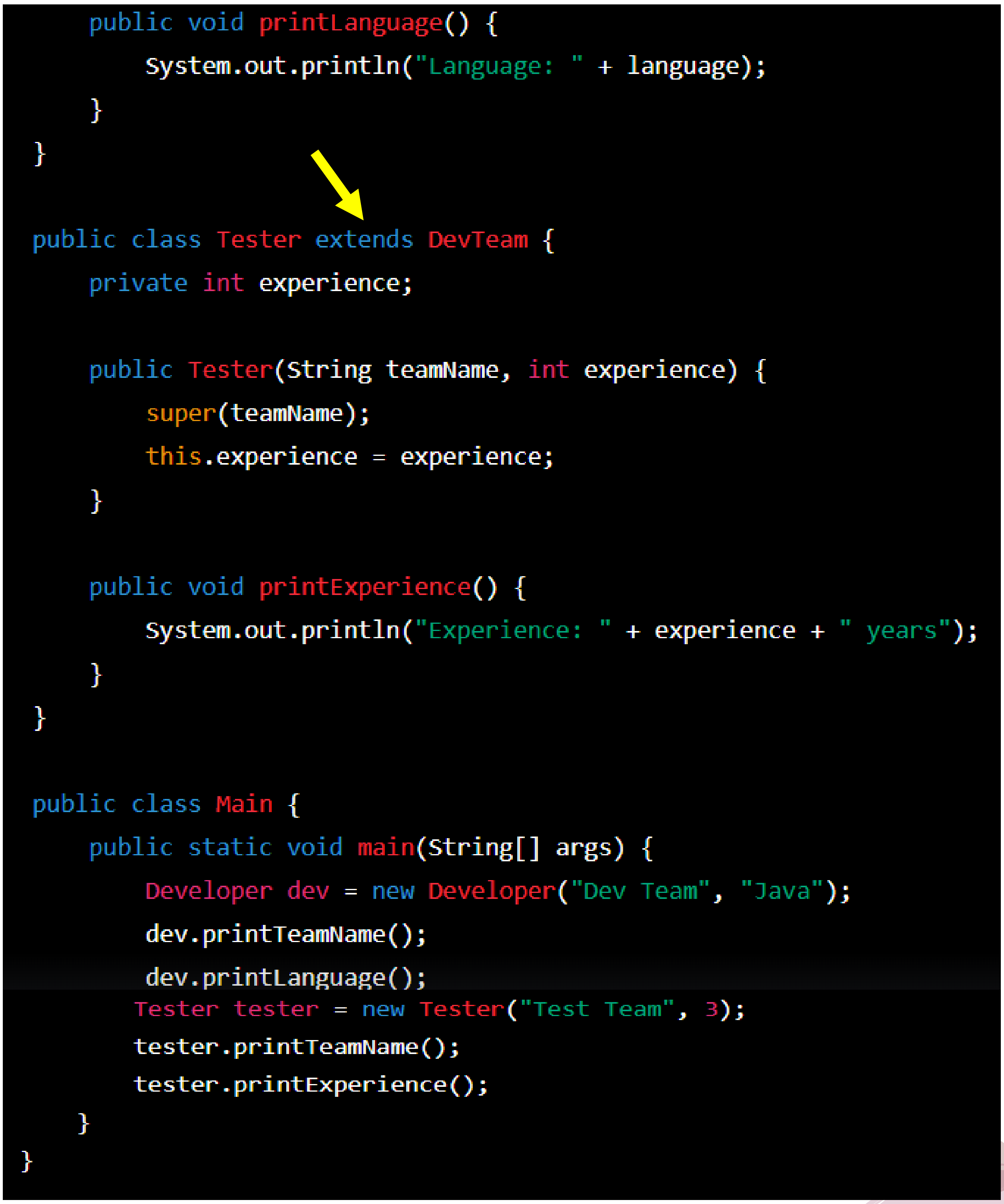
IMPORTANT

Inheritance can only be "MULTILEVEL inheritance",

It cannot be "MULTIPLE inheritance"

Here's how you could use inheritance to create a "DevTeam": parent class and two child classes; "Developer" and "Tester", that inherit from it :

```
public class DevTeam {
    private String teamName;

    public DevTeam(String teamName) {
        this.teamName = teamName;
    }

    public void printTeamName() {
        System.out.println("Team Name: " + teamName);
    }
}

public class Developer extends DevTeam {
    private String language;

    public Developer(String teamName, String language) {
        super(teamName);
        this.language = language;
    }
```



```java
    public void printLanguage() {
        System.out.println("Language: " + language);
    }
}

public class Tester extends DevTeam {
    private int experience;

    public Tester(String teamName, int experience) {
        super(teamName);
        this.experience = experience;
    }

    public void printExperience() {
        System.out.println("Experience: " + experience + " years");
    }
}

public class Main {
    public static void main(String[] args) {
        Developer dev = new Developer("Dev Team", "Java");
        dev.printTeamName();
        dev.printLanguage();
        Tester tester = new Tester("Test Team", 3);
        tester.printTeamName();
        tester.printExperience();
    }
}
```

→In this example, DevTeam is the parent class that has a teamName field and a method printTeamName() that prints out the team name to the console. Developer and Tester are the child classes that inherit from DevTeam.

Developer adds a new language field and a printLanguage() method that prints out the language to the console.

Tester adds a new experience field and a printExperience() method that prints out the experience to the console.

In the main() method, we create instances of Developer and Tester and call their respective methods to print out the team name and additional information specific to their roles.

3) POLYMORPHISM

(Method Override && Method Overloading)

Polymorphism in Java refers to the ability of objects to take on multiple forms or types. Specifically, it refers to the ability of objects of different classes that are related by inheritance to be treated as objects of a common superclass or interface. Polymorphism is a powerful feature of Java that enables code reuse, improves code readability, and makes it easier to work with complex class hierarchies.

There are two types of polymorphism in Java: compile-time (also known as static) polymorphism and runtime (also known as dynamic) polymorphism.

Compile-time polymorphism is achieved through *"method overloading"*, where multiple methods with the same name but different parameters are defined in a class. (just for one same class).The correct method to be called is determined at compile-time based on the type and number of arguments provided.

Runtime polymorphism is achieved through *"method overriding"*, where a subclass(child) provides its own implementation of a method defined in its superclass(parent). The correct method to be called is determined at runtime based on the actual type of the object being referenced.

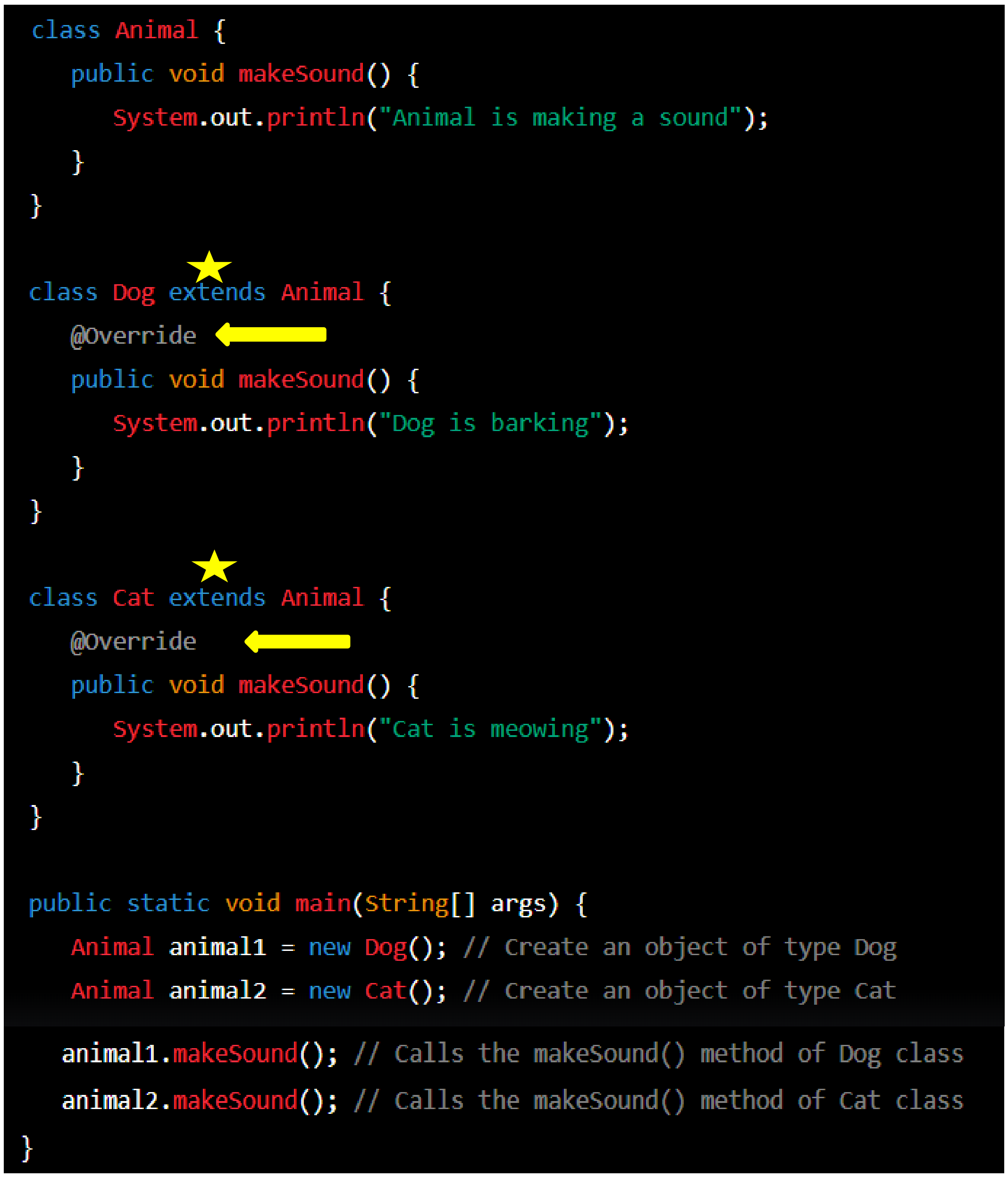
EXAMPLE :

METHOD OVERLOADING :

```java
class Calculator {
   public int add(int x, int y) {
      return x + y;
   }

   public double add(double x, double y) {
      return x + y;
   }
}

public static void main(String[] args) {
   Calculator calculator = new Calculator();

   // Call the add() method with int arguments
   int result1 = calculator.add(3, 5);
   System.out.println("Result1: " + result1);

   // Call the add() method with double arguments
   double result2 = calculator.add(3.5, 2.7);
   System.out.println("Result2: " + result2);
}
```

EXAMPLE :

METHOD OVERRIDING :



```
class Animal {
   public void makeSound() {
      System.out.println("Animal is making a sound");
   }
}

class Dog extends Animal {
   @Override
   public void makeSound() {
      System.out.println("Dog is barking");
   }
}

class Cat extends Animal {
   @Override
   public void makeSound() {
      System.out.println("Cat is meowing");
   }
}

public static void main(String[] args) {
   Animal animal1 = new Dog(); // Create an object of type Dog
   Animal animal2 = new Cat(); // Create an object of type Cat
   animal1.makeSound(); // Calls the makeSound() method of Dog class
   animal2.makeSound(); // Calls the makeSound() method of Cat class
}
```

Rules for using Method Overriding

1) **Parameter list and return type of the overridden method' name must be the same.**

2) **The access modifier must be either the same or wider. That is, if the method in the superclass is "public", the overridden method in the subclass must be at least "public". If the method in the superclass is "protected", the overridden method in the subclass can be either "protected" or "public". If the method in the superclass is "default" or "package-private", the overridden method in the subclass can also be "default" or "public". However, if the method in the superclass is "private", the method in the subclass cannot be overridden !! If the overridden method is "final" or "static", also this method cannot be overridden !!**

In summary, when overriding a method in Java, you must keep the signature of the method and ensure that the access modifier is compatible with the access modifier of the method in the superclass. Also, exceptions and the keywords "final" or "static" should be checking.

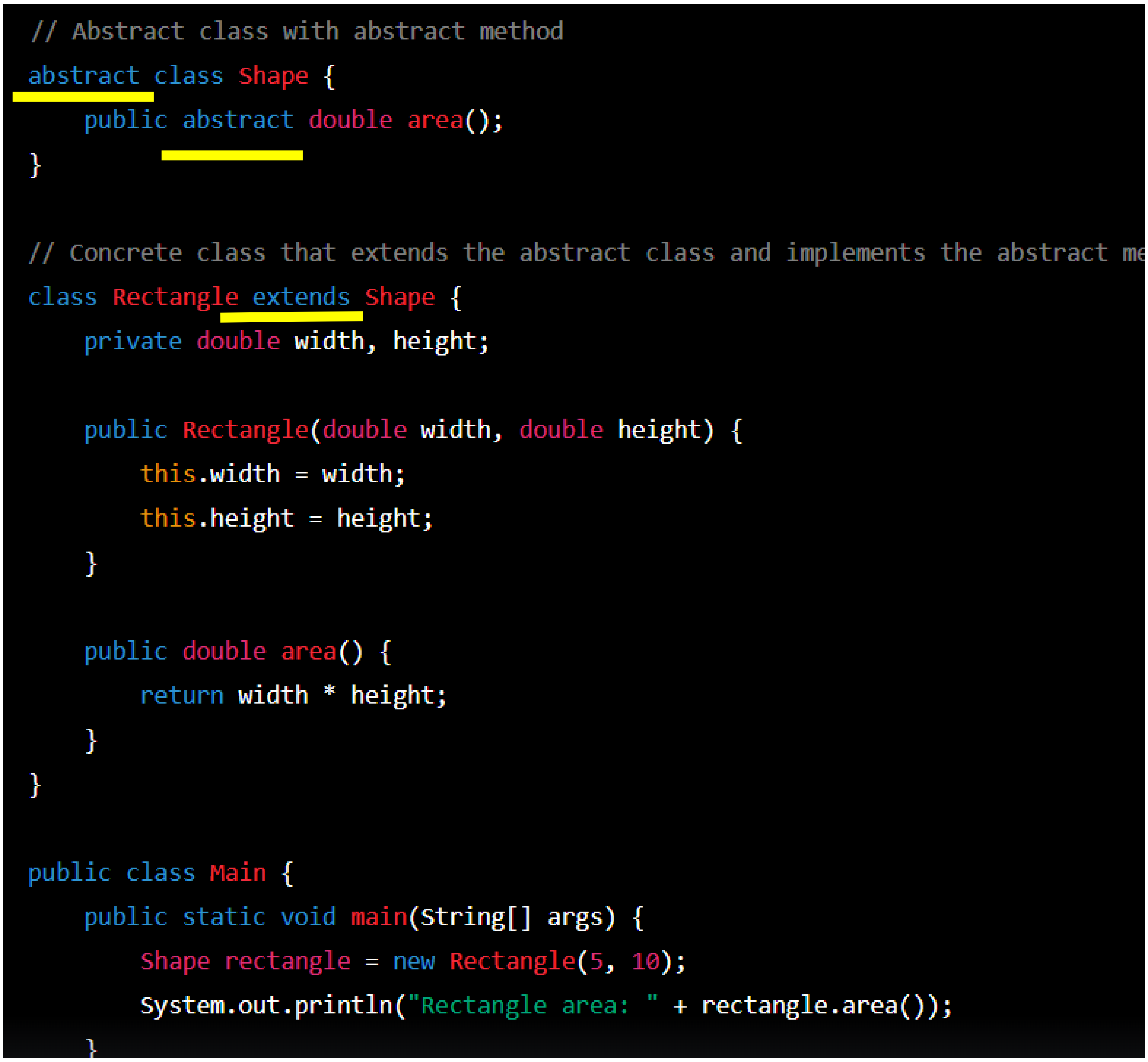
4) ABSTRACTION

Abstract classes are classes that cannot be instantiated directly and may contain abstract methods (that have no implementation) as well as concrete methods (that have an implementation). Abstract classes are typically used to define a common interface for a group of related classes, while allowing each class to implement the interface in its own way.

Abstract methods doesn't have "Method Body" ! That's why abstract methods will be mandatory to 'Override' by all children classes. Why? Because of abstract methods doesn't have method body so it cannot create object. Other important rule is, abstract methods can be created just in Abstract Class.

[illegible] Java, a class is not allowed to inherit from more than one class. However, a class can implement more than one interface. Abstract classes, unlike interfaces, can contain both concrete and abstract methods. Therefore, inheriting from an abstract class can be used as an alternative to multiple inheritance with interfaces.

EXAMPLE :



```java
// Abstract class with abstract method
abstract class Shape {
    public abstract double area();
}

// Concrete class that extends the abstract class and implements the abstract me
class Rectangle extends Shape {
    private double width, height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double area() {
        return width * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape rectangle = new Rectangle(5, 10);
        System.out.println("Rectangle area: " + rectangle.area());
    }
```

SEE YOU
SOON