**WEEK 1 – Java**

1. **Given:**

**public class TaxUtil**

**{**

**double rate = 0.15;**

**public double calculateTax(double amount) {**

**return amount \* rate;**

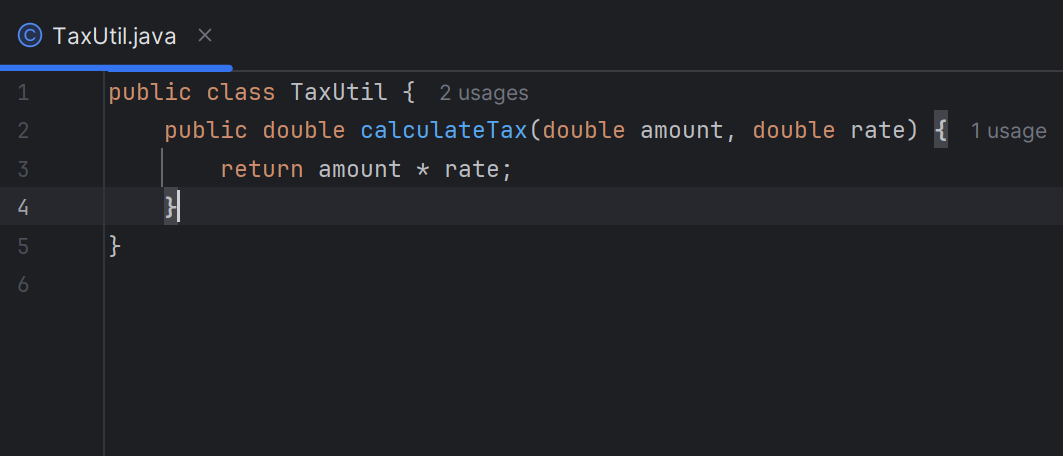
**}**

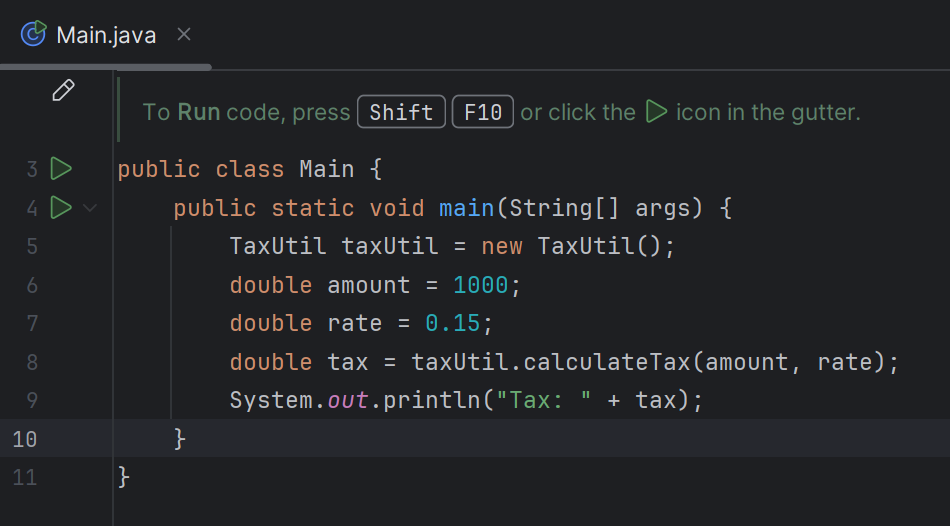
**}**

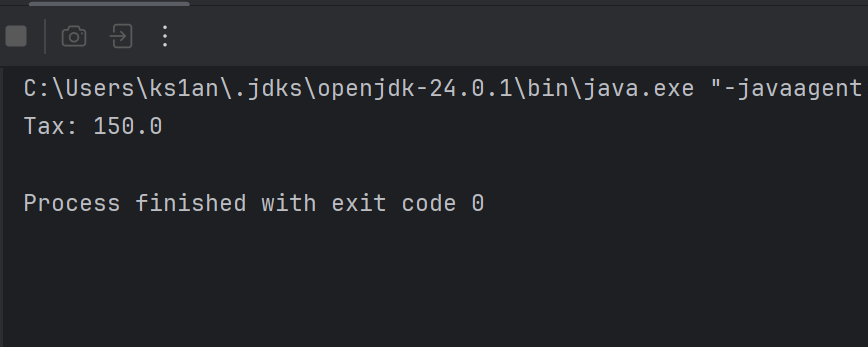
**Would you consider the method calculateTax() a 'pure function'? Why or why not?**

**If you claim the method is NOT a pure function, please suggest a way to make it pure.**

1. The method calculateTax() is not a pure function because a pure function always produces the same output for the same input. The method depends on the instance variable rate. If rate is changed calculateTax(amount) would return different results for the same amount. To make calculateTax() a pure function, its dependency on external state must be removed which can be done by passing rate as a parameter or making it a constant.







1. **What will be the output for following code?**

**class Super**

**{**

**static void show() {**

**System.out.println("super class show method");**

**}**

**static class StaticMethods {**

**void show() {**

**System.out.println("sub class show method");**

**}**

**}**

**public static void main(String[]args) {**

**Super.show();**

**new Super.StaticMethods().show();**

**}**

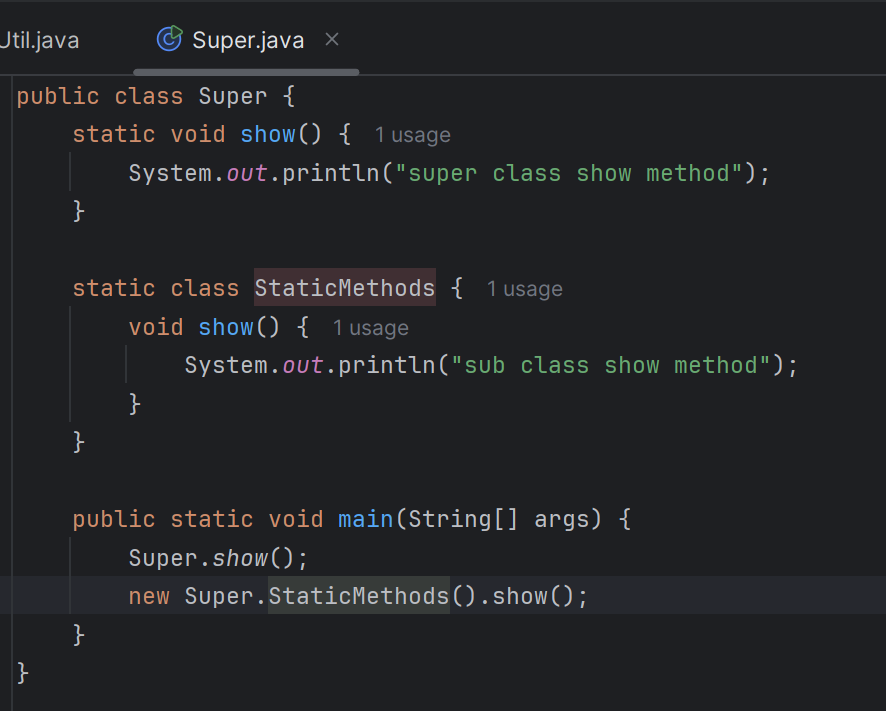
**}**

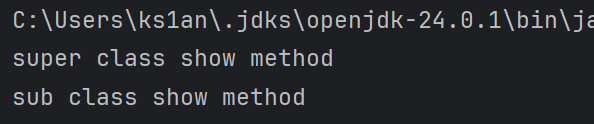
1. Output:

super class show method

sub class show method

Explanation: In the given code, Super.show() is a static method call that prints "super class show method". Then, an object of the static nested class StaticMethods is created, and its instance method show() is called, which prints "sub class show method". Since one is a static method and the other is an instance method in a nested class, both are called correctly, resulting in two lines of output.





1. **What will be the output for the following code?**

**class Super {**

**int num = 20;**

**public void display() {**

**System.out.println("super class method");**

**}**

**}**

**public class ThisUse extends Super {**

**int num;**

**public ThisUse(int num) {**

**this.num = num;**

**}**

**public void display() {**

**System.out.println("display method");**

**}**

**public void Show() {**

**this.display();**

**display();**

**System.out.println(this.num);**

**System.out.println(num);**

**}**

**public static void main(String[] args) {**

**ThisUse o = new ThisUse(10);**

**o.show();**

**}**

**}**

1. Output:

display method

display method

10

10

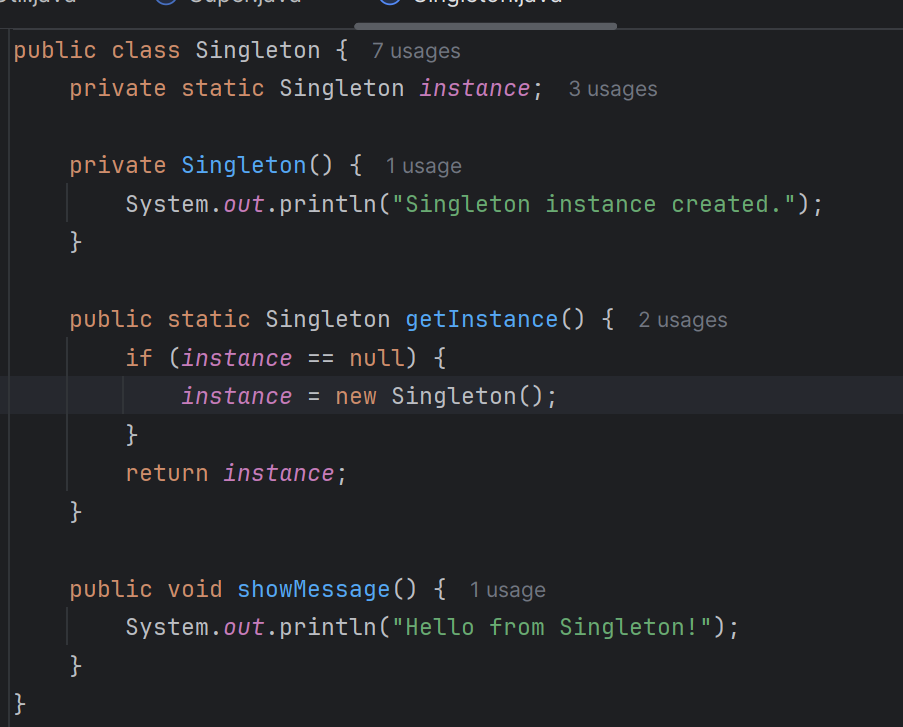
Explanation:

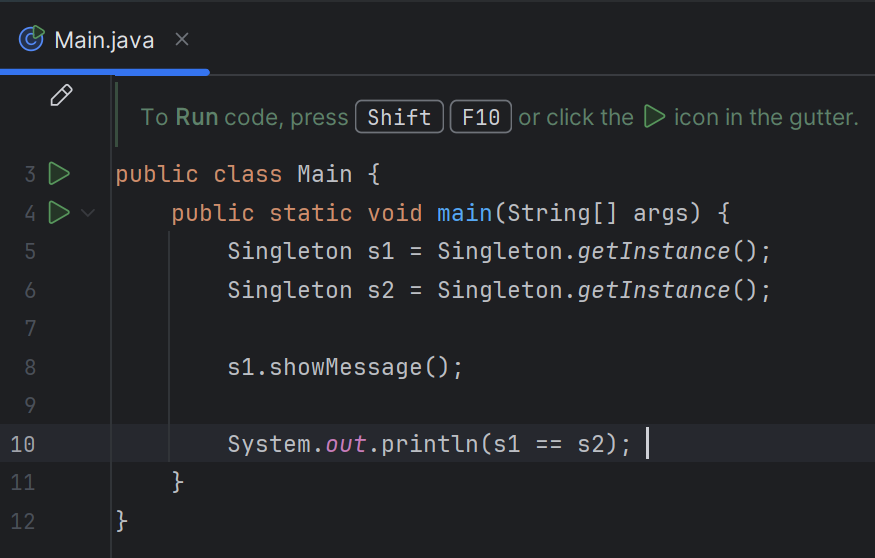
* The class ThisUse extends the Super class and overrides its display() method.
* The Super class has a variable num initialized to 20 and a method display() that prints "super class method".
* The subclass ThisUse declares its own num variable and sets it using the constructor.
* In the main method, an object of ThisUse is created with the value 10 passed to the constructor, which sets this.num in ThisUse to 10.
* When the show() method is called, it first invokes this.display() and then simply display(), both of which call the overridden display() method in ThisUse, printing "display method" twice.
* Finally, this.num and num both refer to the subclass’s variable, which is 10, so 10 is printed twice.
* The superclass’s variable num is shadowed and not used, and the superclass’s display() method is overridden and never called.

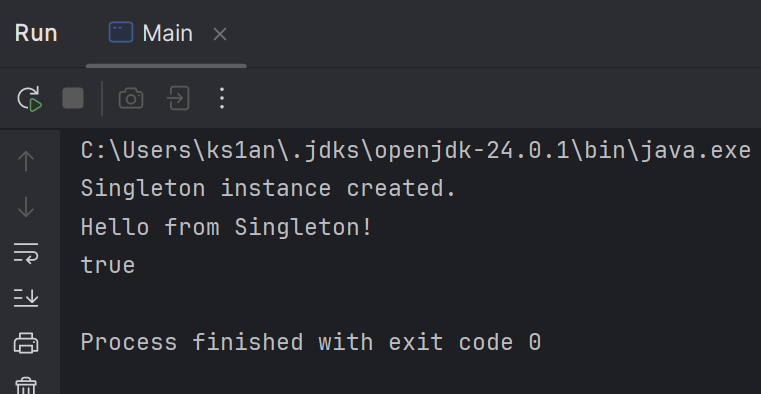
1. **What is the singleton design pattern? Explain with a coding example.**
2. The Singleton Design Pattern ensures that a class has only one instance throughout the program and provides a global access point to that instance.

Features:

* Private constructor to prevent instantiation from outside.
* Static instance of the class.
* Public static method to get the instance.







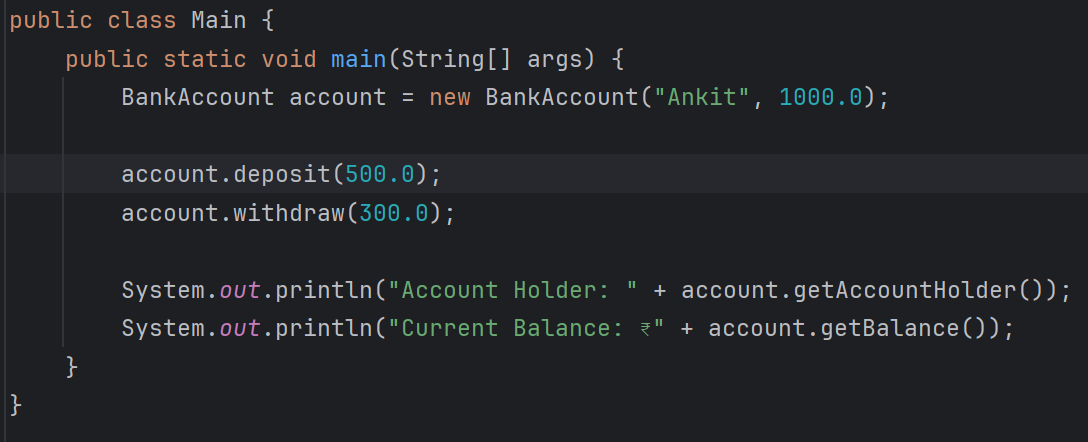
Output shows that both references point to the same instance.

1. **How do we make sure a class is encapsulated? Explain with a coding example.**
2. Encapsulation is one of the core principles of object-oriented programming. It means binding data and methods together and restricting direct access to the class’s internal details.

To make sure a class is properly encapsulated:

* Make all member variables private.
* Provide public getter and setter methods to access and modify those variables.





The balance variable is private and it can’t be changed directly from outside the class. Access to sensitive data is controlled via public methods with validation logic. This ensures data safety and prevents accidental misuse.

1. **Perform CRUD operation using ArrayList collection in an EmployeeCRUD class for the below Employee**

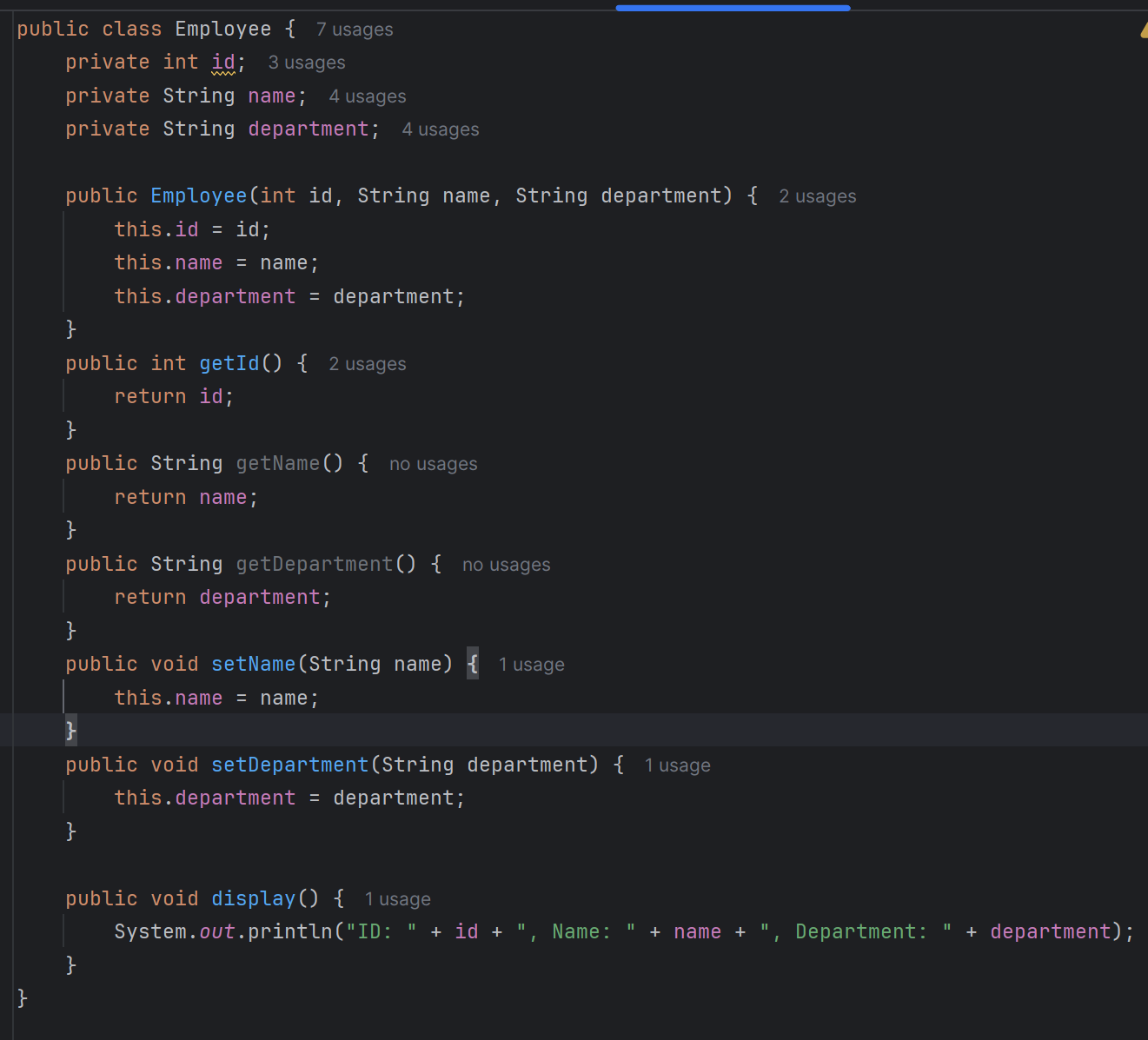
**class Employee{**

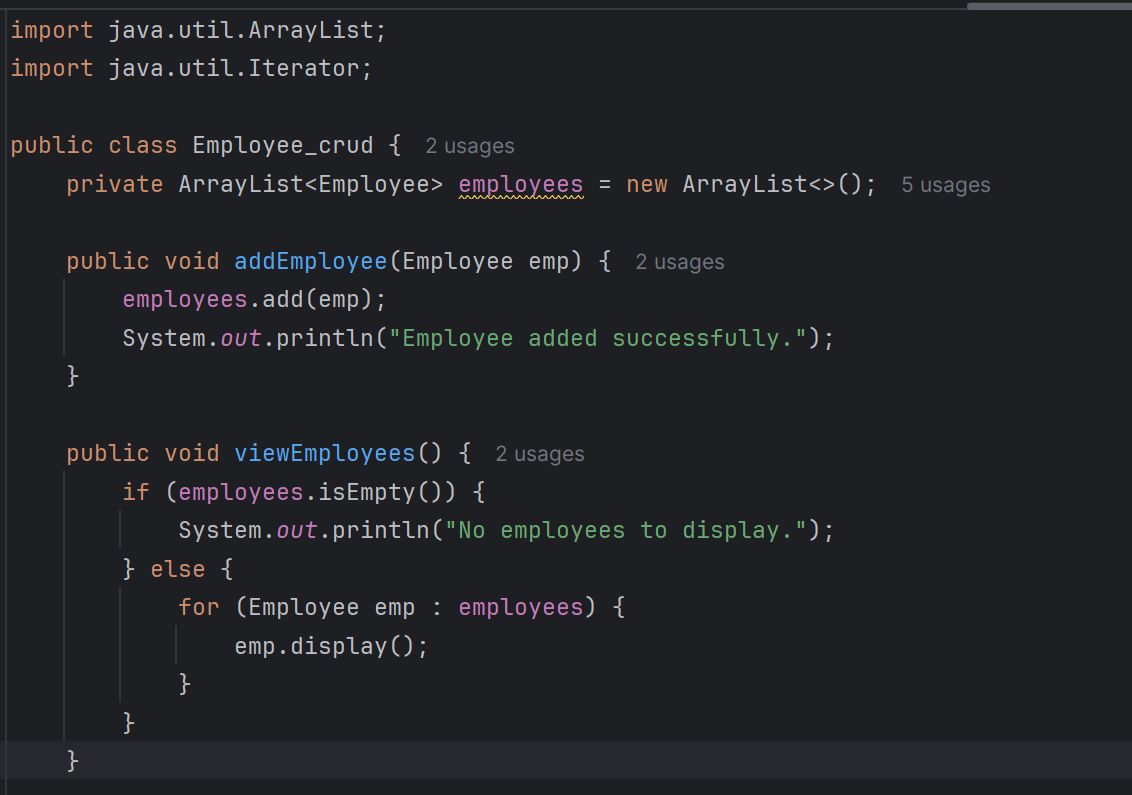
**private int id;**

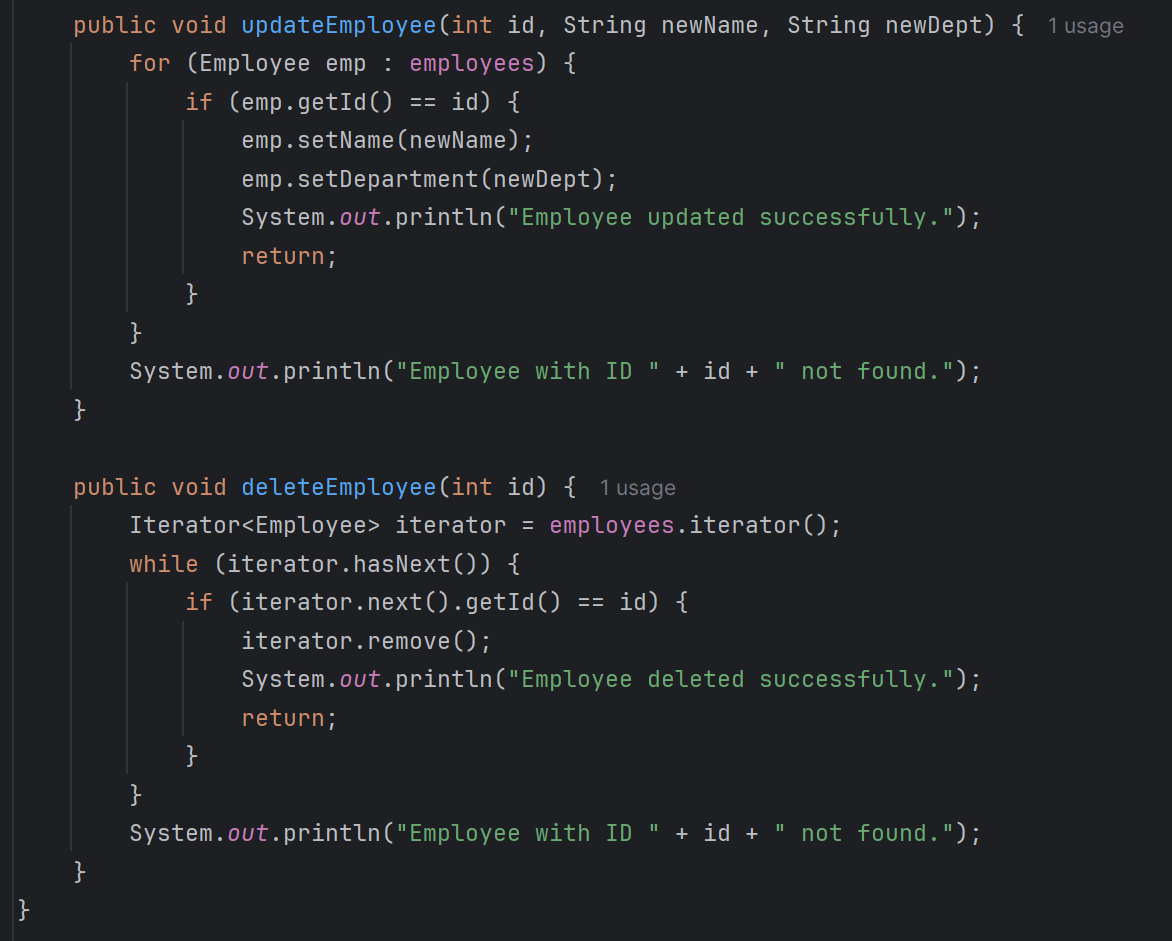
**private String name;**

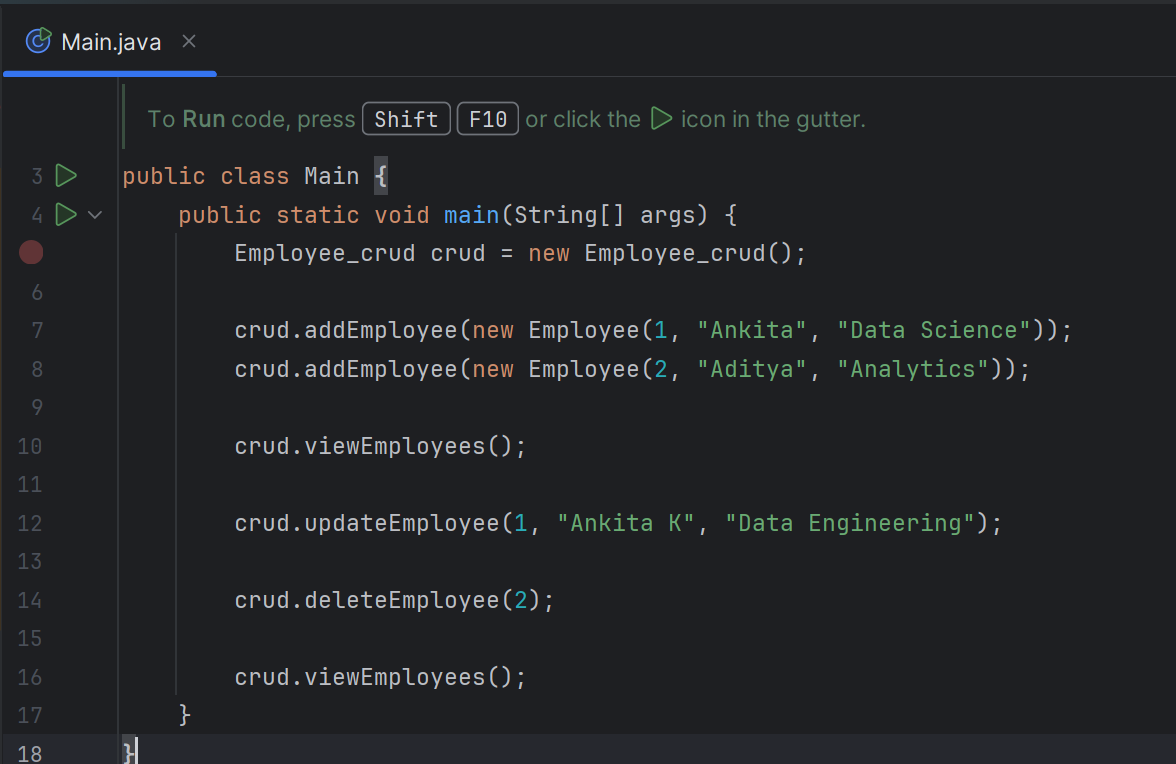
**private String department;**

**}**









Output:

