# Factory Method Design Pattern

By: Romerico David, Aimable Mugwaneza, Kehinde Osunniran

## What is the Factory Method Design Pattern?

- A **creational design pattern**.
- Defines an interface for creating objects but lets subclasses decide which class to instantiate.

## Key Properties of Factory Method

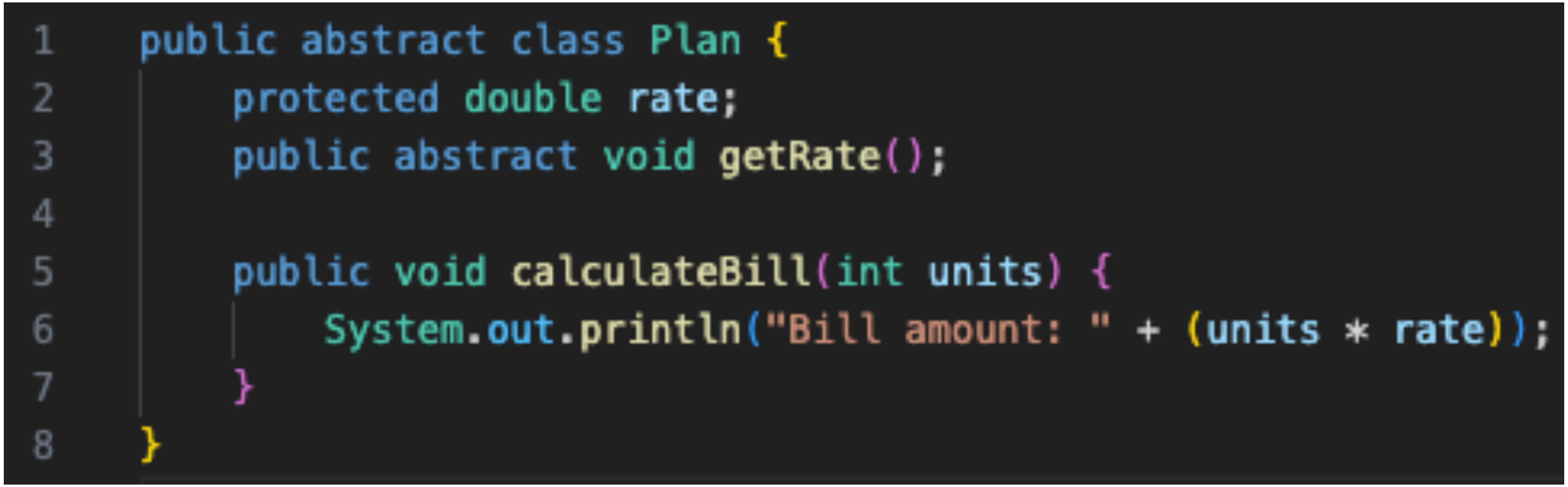
- The client doesn't instantiate concrete classes directly
- The factory method encapsulates the creation logic
- New plan types can be added without modifying existing client code

## Motivation for Factory Method

- Imagine you’re building a logistics application that handles different types of transport, like trucks and ships. You want the application to create these transport objects without needing to know the exact type in advance. The Factory Method pattern helps by allowing you to define a general method in a base class that lets subclasses decide which specific type of transport to create. This way, your code can request a transport object without worrying about whether it’s a truck, ship, or any other type.

## Electricity Billing System Example

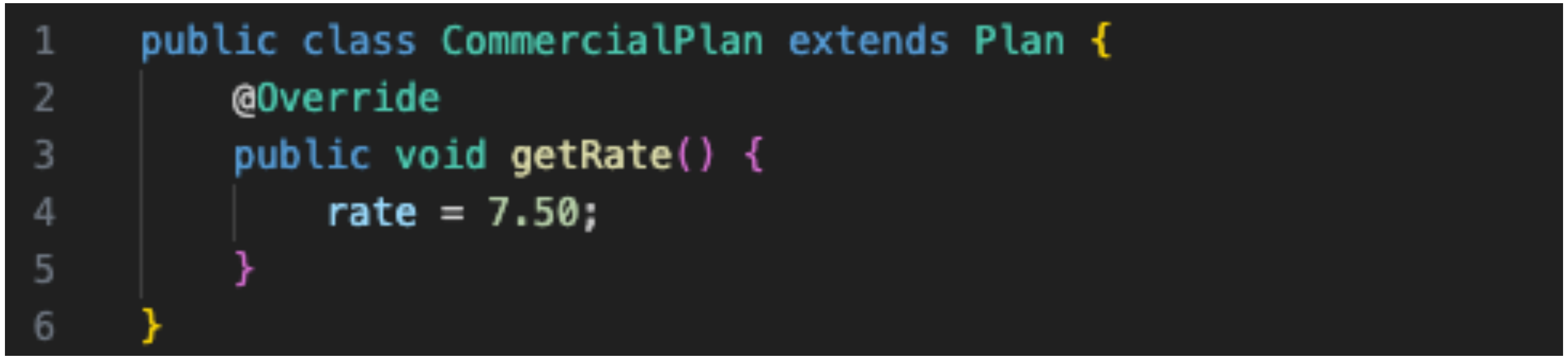
- Calculate the bill for different types of customers.
- Types of Plans:
    - **Domestic Plan**
    - **Commercial Plan**
    - **Institutional Plan**



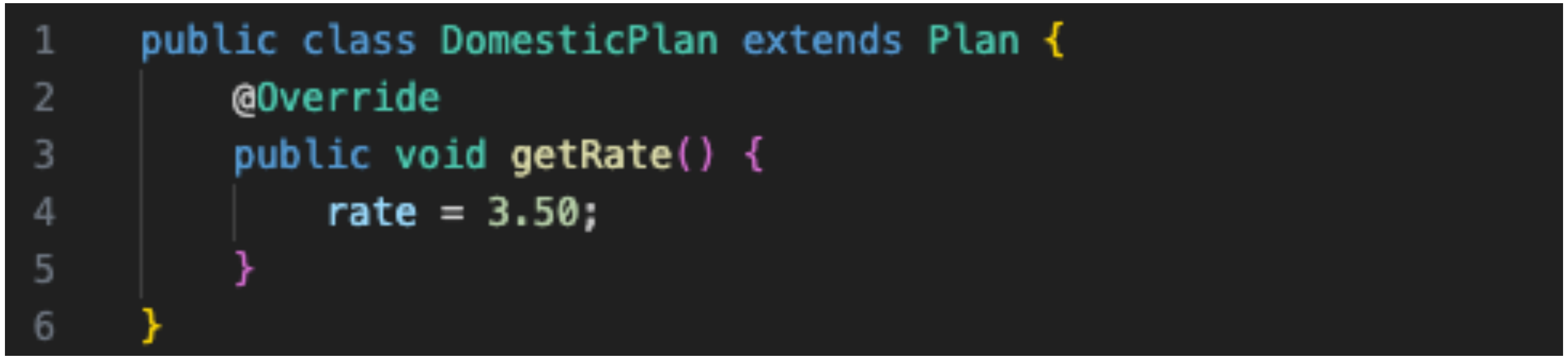
```java
public abstract class Plan {
    protected double rate;
    public abstract void getRate();

    public void calculateBill(int units) {
        System.out.println("Bill amount: " + (units * rate));
    }
}
```

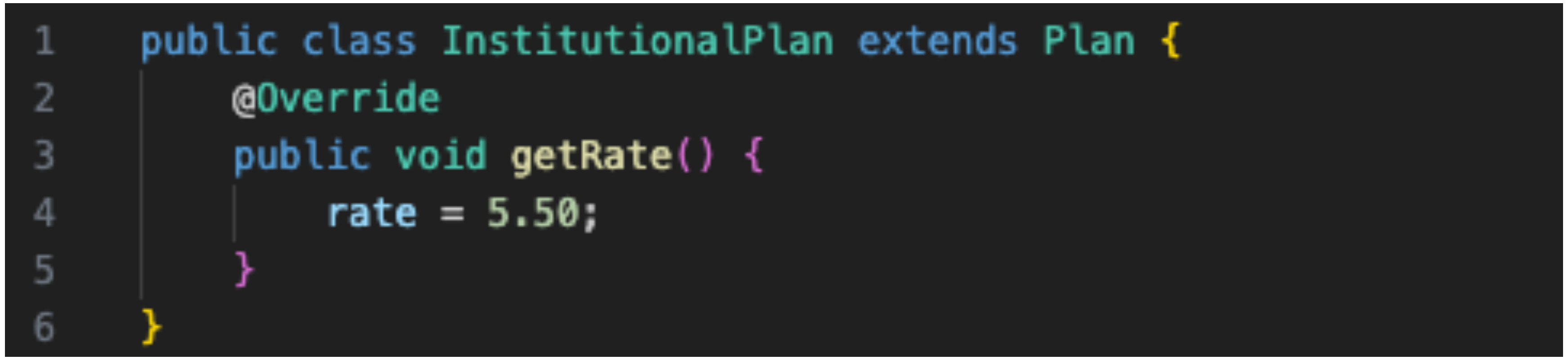
Plan is the interface that defines the operations that all objects the factory method creates must implement



```java
public class CommercialPlan extends Plan {
    @Override
    public void getRate() {
        rate = 7.50;
    }
}
```

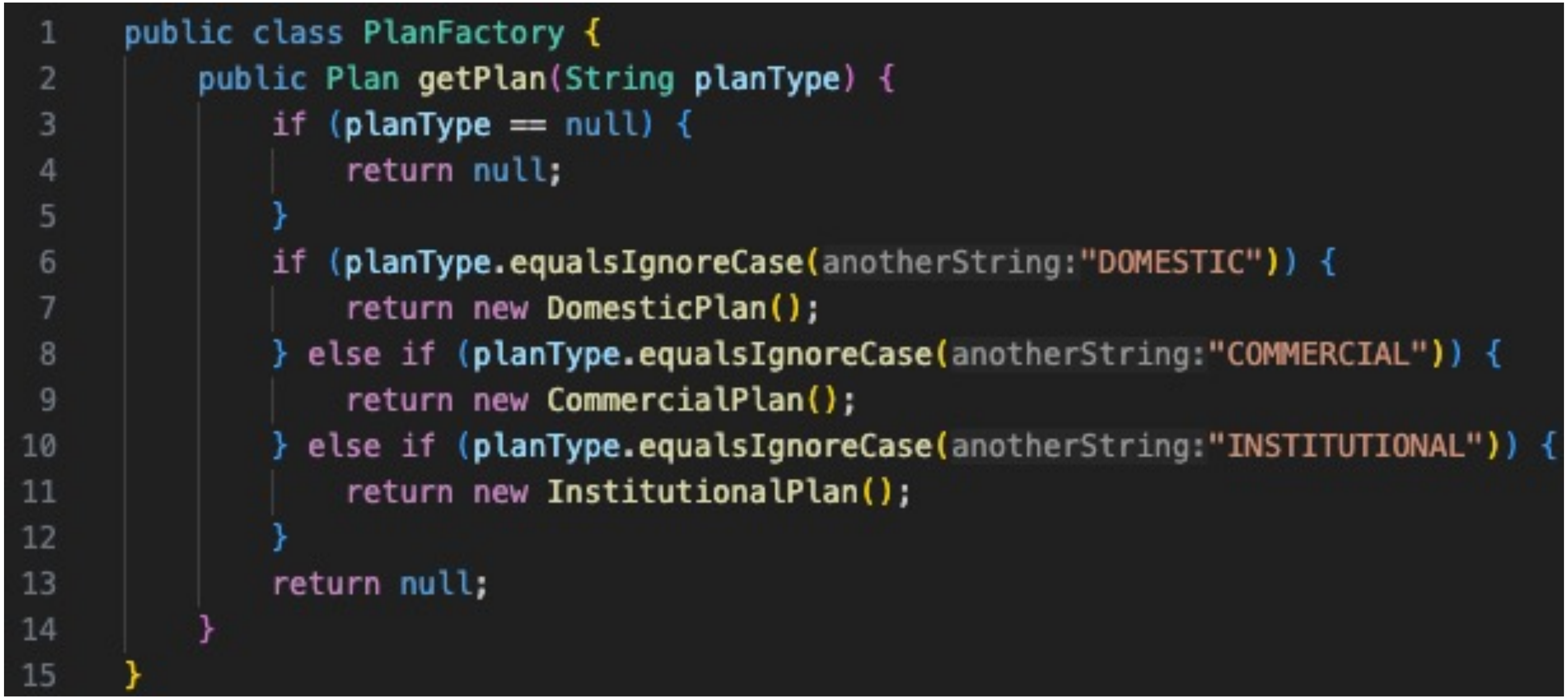


```java
public class DomesticPlan extends Plan {
    @Override
    public void getRate() {
        rate = 3.50;
    }
}
```

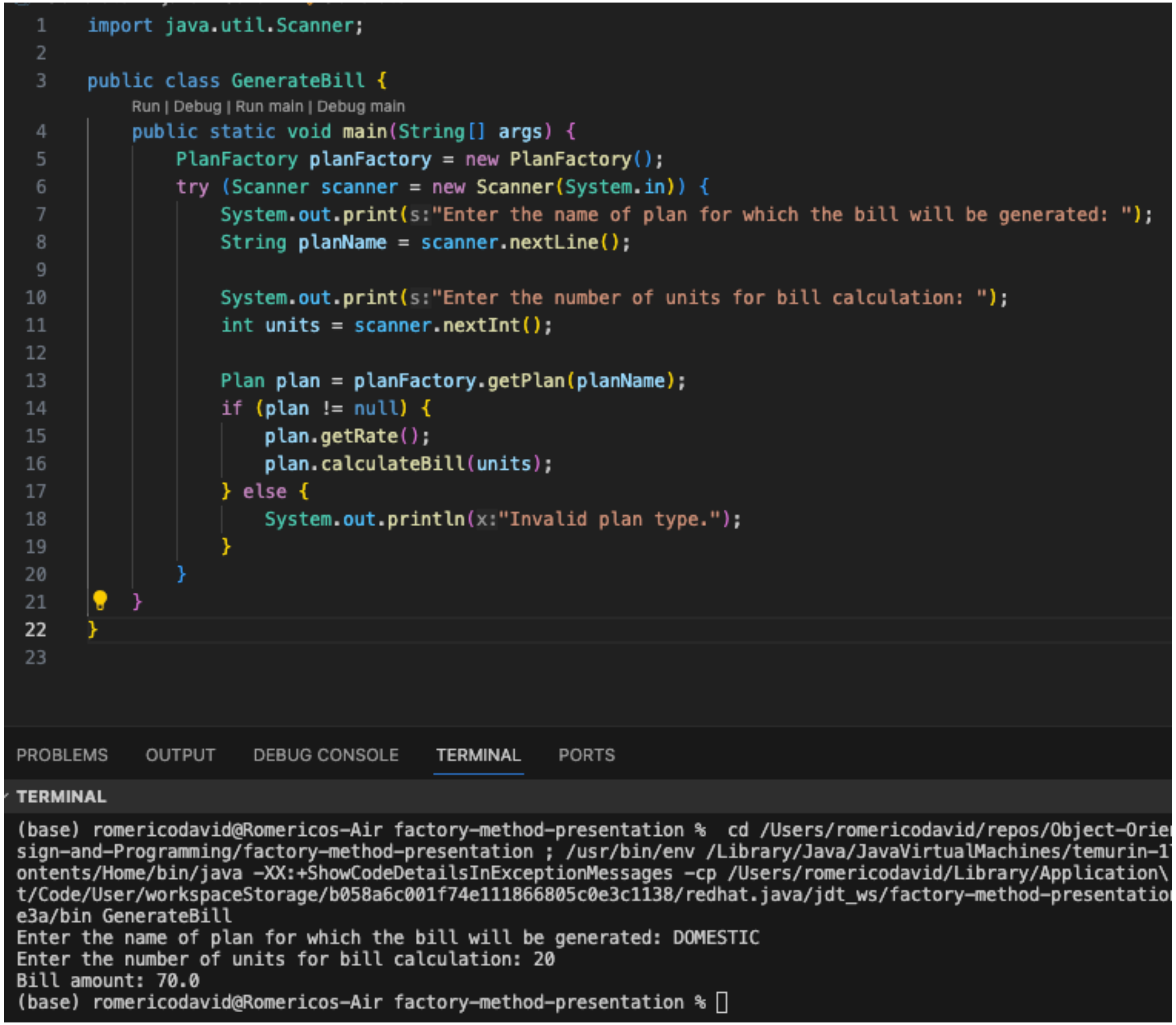


```java
public class InstitutionalPlan extends Plan {
    @Override
    public void getRate() {
        rate = 5.50;
    }
}
```

These are the actual classes that define the behavior for each type of product the factory can create


```java
public class PlanFactory {
    public Plan getPlan(String planType) {
        if (planType == null) {
            return null;
        }
        if (planType.equalsIgnoreCase(anotherString:"DOMESTIC")) {
            return new DomesticPlan();
        } else if (planType.equalsIgnoreCase(anotherString:"COMMERCIAL")) {
            return new CommercialPlan();
        } else if (planType.equalsIgnoreCase(anotherString:"INSTITUTIONAL")) {
            return new InstitutionalPlan();
        }
        return null;
    }
}
```

This is the class that defines and implement the factory method which returns an object of type Plan


```java
import java.util.Scanner;

public class GenerateBill {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        PlanFactory planFactory = new PlanFactory();
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.print(s:"Enter the name of plan for which the bill will be generated: ");
            String planName = scanner.nextLine();

            System.out.print(s:"Enter the number of units for bill calculation: ");
            int units = scanner.nextInt();

            Plan plan = planFactory.getPlan(planName);
            if (plan != null) {
                plan.getRate();
                plan.calculateBill(units);
            } else {
                System.out.println(x:"Invalid plan type.");
            }
        }
    }
}
```

```
(base) romericodavid@Romericos-Air factory-method-presentation %  cd /Users/romericodavid/repos/Object-Orie
sign-and-Programming/factory-method-presentation ; /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-1
ontents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/romericodavid/Library/Application\
t/Code/User/workspaceStorage/b058a6c001f74e111866805c0e3c1138/redhat.java/jdt_ws/factory-method-presentatio
e3a/bin GenerateBill
Enter the name of plan for which the bill will be generated: DOMESTIC
Enter the number of units for bill calculation: 20
Bill amount: 70.0
(base) romericodavid@Romericos-Air factory-method-presentation % 
```

## Advantages vs Disadvantages

### Advantages

- **Loose Coupling**: Clients are decoupled from concrete classes.
- **Open/Closed Principle**: Easy to introduce new plan types.
- **Reusability**: Common creation logic is centralized reducing duplication

### Disadvantages

- **Complexity**: Increases the number of classes.
- **Subclassing**: Requires subclassing to change the product's class.

## References

- https://www.geeksforgeeks.org/factory-method-for-designing-pattern/
- https://en.wikipedia.org/wiki/Factory_method_pattern
- https://sourcemaking.com/design_patterns/factory_method