Design Patterns: Factory and Singleton
Exercises 1 and 2

Imane Fouad, UM6P

Exercise 1: Singleton Database

Objective

Create a Java program that ensures only one instance of a database exists using the Singleton design pattern.

Implementation

```java
class Database{
    static private Database instance;
    private String name;
    public Database(String name){
        this.name = name;

    }
    public static synchronized Database getInstance(String name){
        if (instance == null){
            instance = new Database(name);
        }
        return instance;

    }
    public void getConnection(){
        System.out.println("You are connected to the database "+name);

    }
    Run | Debug
    public static void main(String[] args){
        Database instance = getInstance(name:"E-commerce");
        Database instance2 = getInstance(name:"Githop");
        instance.getConnection();
        instance2.getConnection();
        System.out.println(instance==instance2);
    }
}
```

Figure 1: Singleton Database Implementation

Exercise 2: Factory Pattern

Part 1: Naive Solution

Program Classes Implementation

```
public class Client {
    public static void main1(int number) {
        if(number==1){
            Program1 p = new Program1();
            System.out.println(x:"I am main1");
            p.go();
        }
        else if (number==2){
            Program2 p = new Program2();
            System.out.println(x:"I am main1");
            p.go();
        }
        else if (number==3){
            Program3 p = new Program3();
            System.out.println(x:"I am main1");
            p.go();
        }
        else {
                System.err.println(x:"Error");
            }
    }

    public static void main2(int number) {
        if(number==1){
            Program1 p = new Program1();
            System.out.println(x:"I am main2");
            p.go();
        }
        else if (number==2){
            Program2 p = new Program2();
            System.out.println(x:"I am main2");
            p.go();
        }
        else if (number==3){
            Program3 p = new Program3();
            System.out.println(x:"I am main2");
            p.go();
        }
        else {
                System.err.println(x:"Error");
            }
    }

    public static void main3(int number) {
        if(number==1){
            Program1 p = new Program1();
            System.out.println(x:"I am main3");
            p.go();
```

Figure 2: Client with Conditional Object Creation

```
        }
        else if (number==2){
            Program2 p = new Program2();
            System.out.println(x:"I am main3");
            p.go();
        }
        else if (number==3){
            Program3 p = new Program3();
            System.out.println(x:"I am main3");
            p.go();
        }
        else {
                System.err.println(x:"Error");
            }
    }
}
```

Figure 3: ...

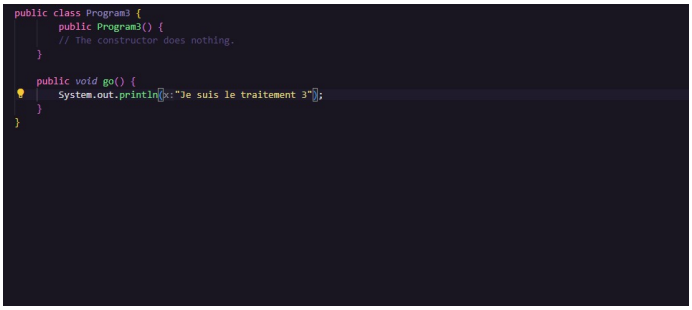

Figure 4: Program3 Class

Client Implementation with Conditional Logic

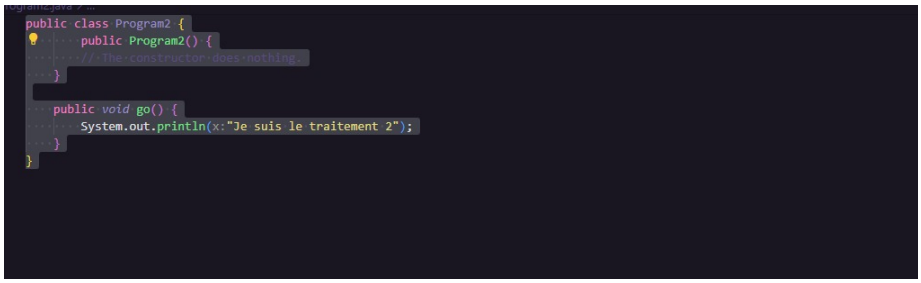

Figure 5: program 2

Part 2: Factory Pattern Solution

Initial Class Diagram Concept

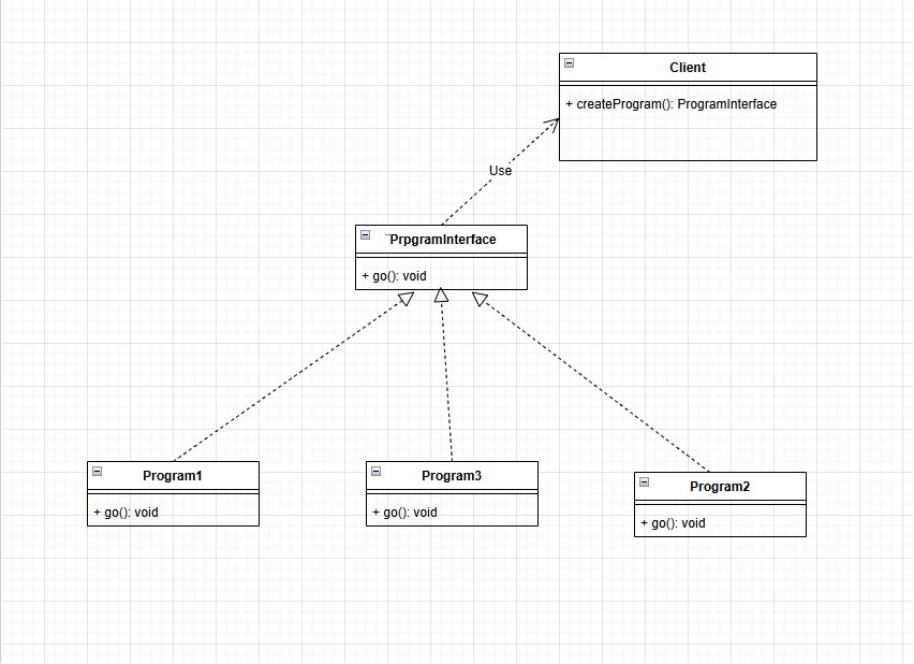

Figure 6: Initial Factory Pattern Concept

Detailed Factory Pattern Design

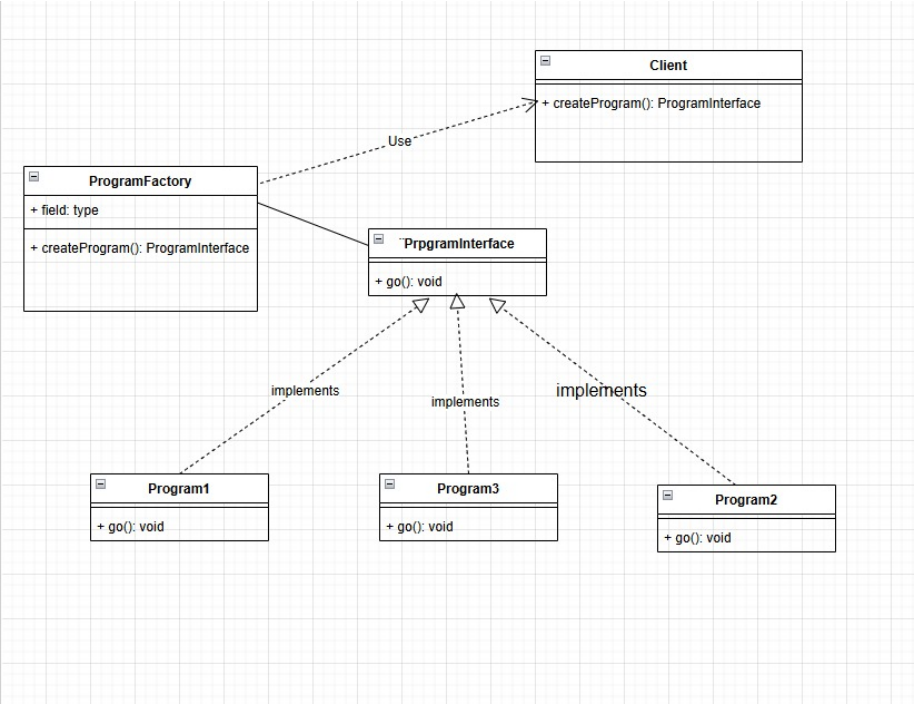

Figure 7: Detailed Factory Pattern Class Diagram