TP: Program Management System

Zineb ELANSARI

October 2025

Contents

1 Exercice 1

```java

public class database {

    private static database instance;
    private String name;

     private database(String name) {
        this.name = name;
    }
    public static 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);
    }
    public static void main(String[] args) {

        database db1 = database.getInstance("Zineb");
        db1.getConnection();

        database db2 = database.getInstance("Asmaa");
        db2.getConnection();


        if (db1 == db2) {
            System.out.println("Both db1 and db2 refer to the same database instance.");
        } else {
            System.out.println("Different instances exist – Singleton pattern failed.");
        }
    }
}
```

Figure 1: Code implementation

Output:

```
Output
You are connected to the databaseZineb
You are connected to the databaseZineb
Both db1 and db2 refer to the same database instance.

=== Code Execution Successful ===
```

Figure 2: Output 1

2 Exercice 2

2.1 Naive Solution

2.1.1 Implementation of Programs

```java
public class Program1 {  2 usages
    public Program1() {  1 usage

    }

    public void go() {  1 usage
        System.out.println("Je suis le traitement 1");
    }
}
```

Figure 3: Implementation of Program1

```java
public class Program2 {  2 usages
    public Program2() {  1 usage

    }

    public void go() {  1 usage
        System.out.println("Je suis le traitement 2");
    }
}
```

Figure 4: Implementation of Program2

```java
public class Program3 {  2 usages
    public Program3() {  1 usage

    }

    public void go() {  1 usage
        System.out.println("Je suis le traitement 3");
    }
}
```

Figure 5: Implementation of Program3

2.1.2 Client Implementation

```java
public class Client {
    public static void main(String[] args) {
        int choice = 2;

        if (choice == 1) {
            Program1 p = new Program1();
            p.go();
        } else if (choice == 2) {
            Program2 p = new Program2();
            p.go();
        } else if (choice == 3) {
            Program3 p = new Program3();
            p.go();
        } else {
            System.out.println("Mauvais choix !");
        }
    }
}
```

Figure 6: The updated implementation of class Client

2.1.3 Output Example

```
C:\Users\pc\.jdks\openjdk-25\bin\java.exe
Je suis le traitement 2

Process finished with exit code 0
```

Figure 7: Example output for n = 2

2.1.4 Problems with This Solution

Issues Identified

- **Code Duplication:** The pattern `new ProgramX()` and `.go()` is repeated
- **Tight Coupling:** Client depends directly on concrete classes
- **Maintenance Difficulty:** Each new program requires changes in multiple places
- **Scalability Issues:** Adding 10 programs means 10 if-else blocks

2.2 Apply Design Patterns

2.2.1 Class Diagram

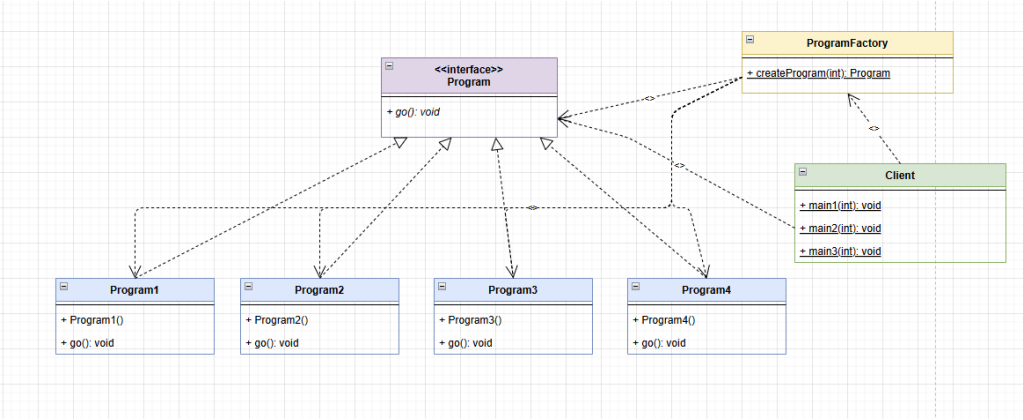

Figure 8: Class Diagram with Factory Pattern

2.2.2 Implementation of Interface

```
public interface Program {  no usages  3 implementations
    void go();  no usages  3 implementations
}
```

Figure 9: Implementation of Interface Program

2.2.3 Implementation of Program Classes

```
public class Program1 implements Program {  2 usages
    public void go() {  1 usage
        System.out.println("Je suis le traitement 1");
    }
}
```

Figure 10: Implementation of Program1

```
public class Program2 implements Program {  2 usages
    public void go() {  1 usage
        System.out.println("Je suis le traitement 2");
    }
}
```

Figure 11: Implementation of Program2

```java
public class Program3 implements Program {  2 usages
    public void go() {  3 usages
        System.out.println("Je suis le traitement 3");
    }
}
```

Figure 12: Implementation of Program3

2.2.4 Factory Implementation

```java
public class ProgramFactory {  1 usage
    public static Program createProgram(int number) {  1 usage
        if (number == 1) {
            return new Program1();
        } else if (number == 2) {
            return new Program2();
        } else if (number == 3) {
            return new Program3();
        } else {
            return null;
        }
    }
}
```

Figure 13: Implementation of ProgramFactory

2.2.5 Client Implementation

```java
public class Client {
    public static void main(String[] args) {
        int choice = 3;

        Program p = ProgramFactory.createProgram(choice);

        if (p != null) {
            p.go();
        } else {
            System.out.println("Mauvais choix !");
        }
    }
}
```

Figure 14: Implementation of Class Client

2.2.6 Output Example

```
C:\Users\pc\.jdks\openjdk-25\bin\java.exe
Je suis le traitement 3

Process finished with exit code 0
```

Figure 15: Example of output with n = 3

2.3 Adding Program4

2.3.1 Can we add a Program4?

Answer: Yes, we can.

```java
public class Program4 implements Program {  no usages
    public void go() {  1 usage
        System.out.println("Je suis le traitement 4");
    }
}
```

Figure 16: Implementation of Program4

```java
public class ProgramFactory {  1 usage
    public static Program createProgram(int number) {  1 usage
        if (number == 1) {
            return new Program1();
        } else if (number == 2) {
            return new Program2();
        } else if (number == 3) {
            return new Program3();
        }else if (number == 4) {
            return new Program4();
            }
         else {
            return null;
        }
    }
}
```

Figure 17: Update of ProgramFactory

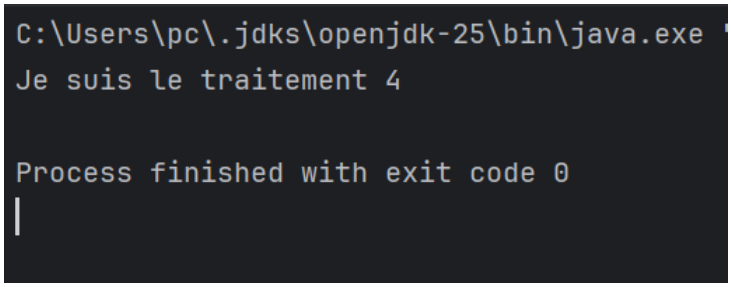

Figure 18: Example output with n = 4

2.3.2 Was it complicated to add Program4?

Advantages of Factory Pattern

- **Easy to Add:** Only 2 files modified (Program4 creation + Factory)
- **No Client Changes:** Client.java remains untouched
- **Single Responsibility:** Factory handles object creation
- **Loose Coupling:** Client depends on interface, not concrete classes
- **Scalable:** Can add Program5, Program6... easily