Design Patterns: Factory & Singleton

Imane Fouad, UM6P

Exercise 1:

Write a Java program that creates only one instance of a database using the Singleton design pattern. The database should have a single attribute called name, which represents the name of the database, and a method named getConnection() that simply prints the message "You are connected to the database *name*. In the main program, test that the database is unique by attempting to create two databases with different names and showing that both references point to the same instance.

Exercise 2:

Consider a project with two classes:

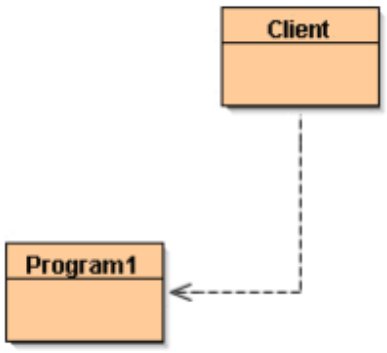
- A *Program1* class, that simply displays a message when the go method is called (in a real situation, it would perform some specific processing).
- A *Client* class, that calls Program1.

```
1  public class Client
2  {
3   public static void main1()
4   {
5   Program1 p = new Program1();
6   System.out.println("I am main1");
7   p.go();
8   }
9
10  public static void main2()
11  {
12  Program1 p = new Program1();
13  System.out.println("I am main2");
14  p.go();
15  }
16
17  public static void main3()
18  {
19  Program1 p = new Program1();
20  System.out.println("I am main3");
21  p.go();
22  }
23 }
```

```
1  public class Program1
2  {
3   public Program1()
4   {
```

```
5   // The constructor does nothing.
6   }
7   public void go()
8   {
9   System.out.println("Je suis le traitement 1");
10  }
11 }
```

The class diagram looks like this:



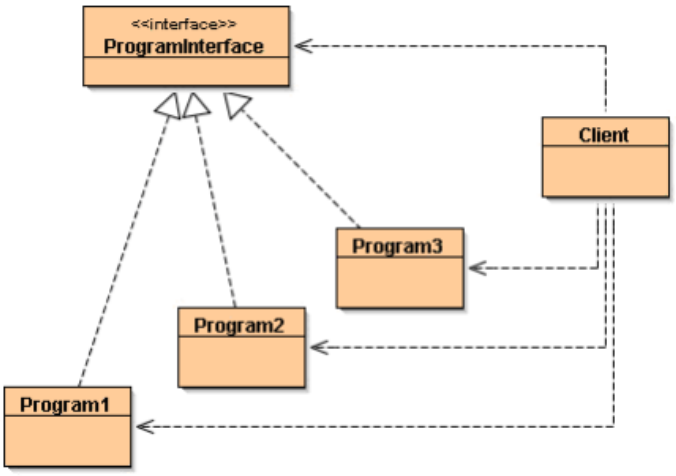
1- Naive Solution:

We want to add two classes, Program2 and Program3, to our project. These classes will display the same message on the screen as Program1, except that they will include their program number (2 or 3).

In the Client, we want to modify the main function so that, depending on the integer parameter passed as an argument (1, 2, or 3), it launches the processing of Program1, Program2, or Program3.

1. Implement the required classes as specified.
2. Update the client class to incorporate the required functionality.
3. What do you notice? What problems are caused by such a solution?

By coding it properly, the class diagram you should obtain:



2- Apply Design Patterns

In the three main functions of Client, did you duplicate the code for creating Program objects? Duplication of codes is a bad practice. Delegate the creation of Program objects to a class named *ProgramFactory*.

- Design the **class diagram** that meets this requirement.
- Provide the code for the different entities that participate in the program.
- Can you add a Program4? Was it complicated to implement? Did you have to modify the Client code?

Exercise 3 : Monster Battle Game

You are asked to build a **Monster Battle Game**. The game should use **Singleton and Factory** design patterns to manage game state and monsters creation.

The game should include:

- One **GameManager** controlling the game.
- Dragon, Goblin, Wizard monsters each with unique attack behavior and health points
- Turn-based battle between monsters with health points and attack behavior.

Game Warning!

1 - Draw the corresponding class diagram (with the required design patterns) .

Part 1: Game Manager

1. Implement a `GameManager` class that ensures **only one instance exists**.

2. Methods:
    - `startGame()` – prints a welcome message.
3. Verify that multiple attempts to create instances return the same object

Part 2: Monsters

The player can choose from three types of monsters: Dragon, Goblin, and Wizard. Each monster has unique health points and attack behavior, and you can assign the damage value for their attacks as you see fit.

All monsters should implement the following methods:

- attack(): prints the attack message, and attack the opponent.
- getHealth(): returns the current health points.
- takeDamage(int damage): reduces the monster's health by the given amount.

Implement the corresponding code for these monsters.

Part 3: Game Client: Battle Interaction

Implement a `GameClient` class with `main()` that:

- Starts the game using `GameManager`.
- Prompts the user to choose two monsters.
- Creates monsters using `MonsterFactory`.
- Alternates attacks between monsters until one monster's health reaches 0.
- Updates the player score using `GameManager`.
- Displays health after each attack and announces the winner.

Game Warning!

Implement the game according to the requirements, ensuring the code is clean, reusable, and properly applies the required design patterns.