	Course Name: Design Patterns/Thinking LAB	**EXPERIMENT NO. 1**	
	Course Code: 20CP210P **Faculty: Dr. Ketan Sabale**	**Branch: CSE**	**Semester: IV**
Submitted by: Rhythm Shah **Roll no: 22BCP071**			

Objective: To familiarize students with standard Creational design patterns.
Experiment: Explain the factory design pattern and write a program using any object-oriented programming language to demonstrate the working of factory design pattern.

Theory:

If we need to create objects but we don't know the types exactly and we need to create new objects again and again by directly changing the code which creates problem for the user, so to overcome all these problems Factory Design Pattern is used. The Factory handles the creation of objects and we can tell the factory to create them for us.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.
The factory design pattern is a creational pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created. It falls under the category of creational design patterns, which deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

It is also used to hide the implementation details of the object. You can easily change or extend the types of objects created by the factory without modifying the client code.

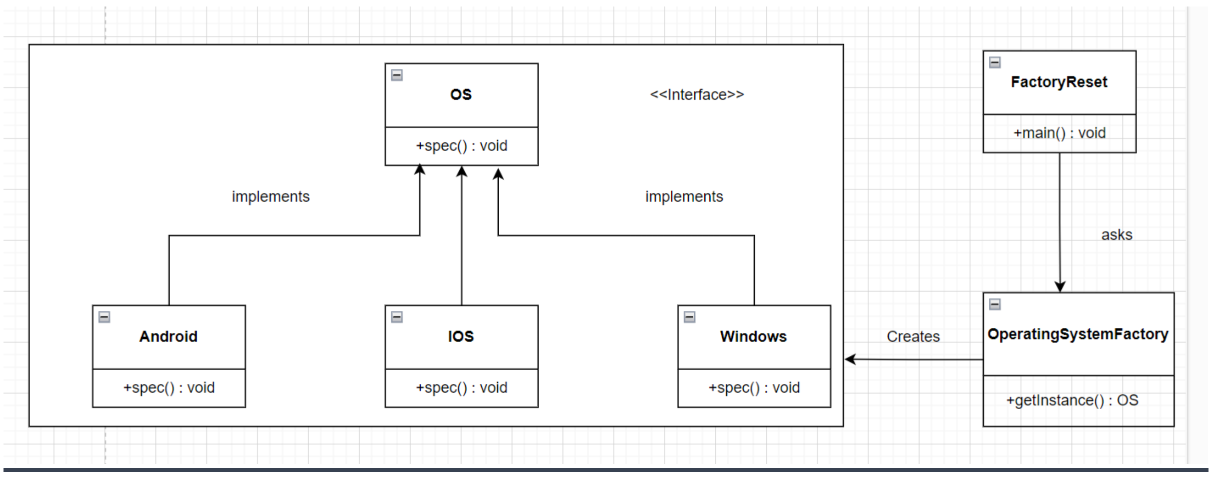
Example(explanation of my implementation**):**

In this code there is an implementation of the Factory Design Pattern, which is a creational pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

We're going to create an OS interface and concrete classes implementing the OS interface. A factory class OperatingSystemFactory is defined as a next step.

Factory Reset, our demo class will use OperatingSystemFactory to get a OS object. It will pass information (Android/IoS/Windows) to OperatingSystemFactory to get the type of object it needs.

UML DIAGRAM:



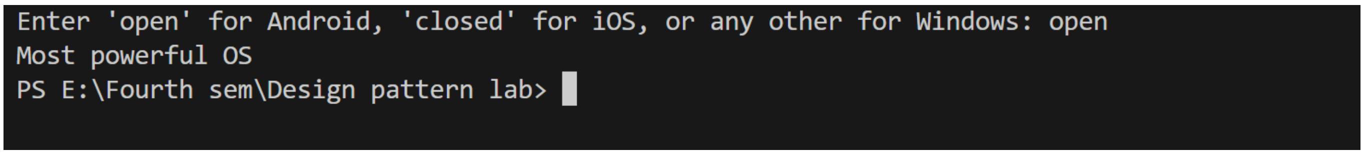
CODE:

```
import java.util.Scanner;

interface OS {

   void spec();

}
```

```
class Android implements OS {
    public void spec() {
        System.out.println("Most powerful OS");
    }
}
class IOS implements OS {
    public void spec() {
        System.out.println("Most secure OS");
    }
}
class Windows implements OS {
    public void spec() {
        System.out.println("I am about to die");
    }
}
class OperatingSystemFactory {
    public OS getInstance(String str) {
        if (str.equals("open"))
            return new Android();
        else if (str.equals("closed"))
            return new IOS();
        else
            return new Windows();
    }
}

public class factoryreset {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter 'open' for Android, 'closed' for iOS, or any other for Windows:
");
```

```
        String userInput = scanner.nextLine();
        OperatingSystemFactory osf = new OperatingSystemFactory();
        OS obj = osf.getInstance(userInput);
        obj.spec();
        scanner.close();
    }
}
```

OUTPUT:



```
Enter 'open' for Android, 'closed' for iOS, or any other for Windows: open
Most powerful OS
PS E:\Fourth sem\Design pattern lab>
```