UNIVERSITY OF INFORMATION TECHNOLOGY AND SCIENCES (UITS)
DEPARTMENT OF INFORMATION TECHNOLOGY

LAB REPORT : 4

IT-324 : DESIGN PATTERN LAB

Factory Design Pattern

Submitted To:

Sifat Nawrin Nova
Lecturer,
Department of IT, UITS
Email:sifat.nawrin@uits.edu.bd

Submitted By:

Name:Nazmul Zaman
Student ID:2014755055
Department of IT, UITS

June 26, 2022 Department of IT, UITS © All rights reserved.

Contents

1 Abstraction

In this lab report we learn about Factory Design Pattern method and how it's work and why we used this method.
Abstract Factory Pattern says that just define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes. That means Abstract Factory lets a class returns a factory of classes.The Factory Method Pattern is also known as Virtual Constructor.

2 Theory

Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.A Factory Pattern or Factory Method Pattern says that just de-fine an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words, subclasses are responsible to create the instance. A Factory Pattern or Factory Method Pattern says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.

3 Objective

Consider we want to implement a Factorybuilder through Factory,Windows,Ios,OsFactory,Android. Let's try to implement this with the help of factory method design pattern.We will also learn advantages and disadvantages of Observer pattern.

4 What is the goal of the factory method pattern?

The factory pattern aims to solve a fundamental problem in instantiation – i.e., the creation of a concrete object of a class – in object-oriented programming. In principle, creating an object directly within the class that needs or should use this object is possible, but very inflexible. It binds the class to this object and makes it impossible to change the instantiation independently of the class. This kind of code is avoided in the factory pattern approach by first defining a separate operation for creating the object – the factory method. As soon as this is called up, it generates the object instead of the class constructor already mentioned..

5 Advantage and disadvantage

Advantage :
1.Factory Method Pattern allows the sub-classes to choose the type of objects to create.

Disadvantage :
1.High number of required classes
2.Extension of the application is very elaborate

6 Usage of Factory Design Pattern

1.When a class doesn't know what sub-classes will be required to create
2.When a class wants that its sub-classes specify the objects to be created.
3.When the parent classes choose the creation of objects to its sub-classes.

7 Working procedure

1:First of all Create an interface
. 2: Then Create concrete classes implementing the same interface. 3: Create a Factory to generate object of concrete class based on given infor-mation.
4: Use the Factory to get object of concrete class by passing an information such as type.
5: Finally Verify the output.

8 Source Code

8.1 FactoryPattern class/Main Class)

```java

package factorypattern;

public class FactoryPattern {


    public static void main(String[] args) {


        OsFactory  osf = new OsFactory();
        Factory obj = osf.getInstance ("Open source");

        obj.spec();


    }

}
```

?

8.2 (Android Class)

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package factorypattern;

public class Android  implements Factory{

    public void spec()
    {
        System.out.println("Mostly use OS");
    }

}
```

?

8.3 (Factory Class)

```java

package factorypattern;

/**
 *
 * @author A
 */
public interface Factory {


    void spec();


}
```

?

8.4 (Windows Class)

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/
   license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class
   .java to edit this template
 */
package factorypattern;

public class Windows implements Factory{

    public void spec()
    {
        System.out.println("Others OS");
    }

}
```

?

8.5 (Ios Class)

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/
   license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class
   .java to edit this template
 */
package factorypattern;

/**
 *
 * @author A
 */
```

```java
public class Ios  implements Factory {

   public void spec()
    {

    }

}
```

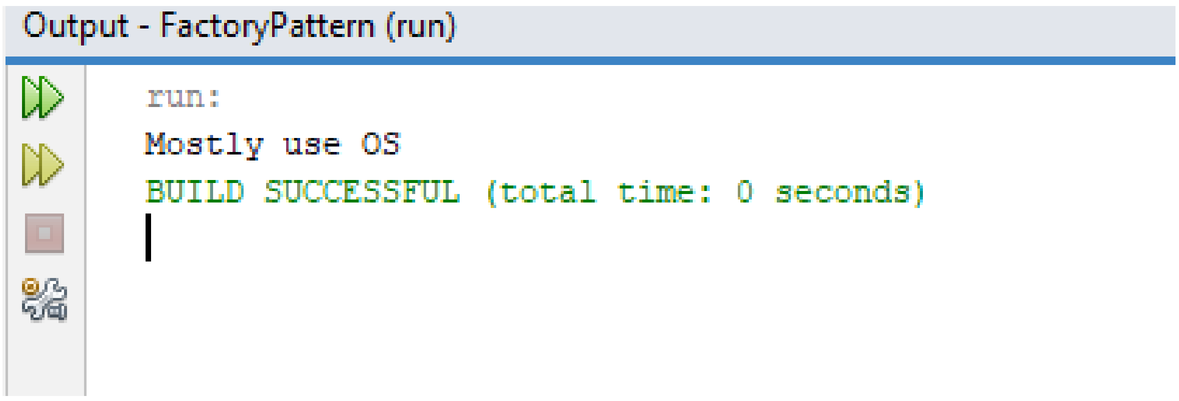
?

8.6 (OsFactory Class)

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/
   license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class
   .java to edit this template
 */
package factorypattern;

public class OsFactory {

    public Factory getInstance  (String str)
    {
        if(str.equals("Open source"))
        {
            return new Android();
        }

        else if(str.equals("Closed Source"))

        {
             return new Ios();
        }
         else
         return new Windows ();

    }

}
```

?

8.7 Output



```
run:
Mostly use OS
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 1: Output

9 Conclusion

In this lab report we learn about factory design pattern in practically. How we used factory pattern design pattern with real life and also getting deeply knowledge about builder pattern method and how does it works.

10 References

1.https://www.tutorialspoint.com/design$_{p}attern/singleton_{p}attern.htm$
2.$https : //www.geeksforgeeks.org/singleton-design-pattern-introduction$
3.$https : //www.javatpoint.com/singleton - design - pattern - in - java$