Factory Design Pattern

Notes By Bhavuk Jain

Factory Design Pattern is useful in the cases where we want to **create objects based upon certain conditions** or business logic. It helps us *prevent the duplication* of writing conditional checks or our business logic to create an object again and again wherever we need that object. In a nutshell, *factory pattern allows us to separate out the conditions for creation of objects.*

Core Concept

Suppose we require an object of type **Score**, in our class, when the condition states that it should be a Cricket Score object, so one way we could do that is to use if clause.

Example:-

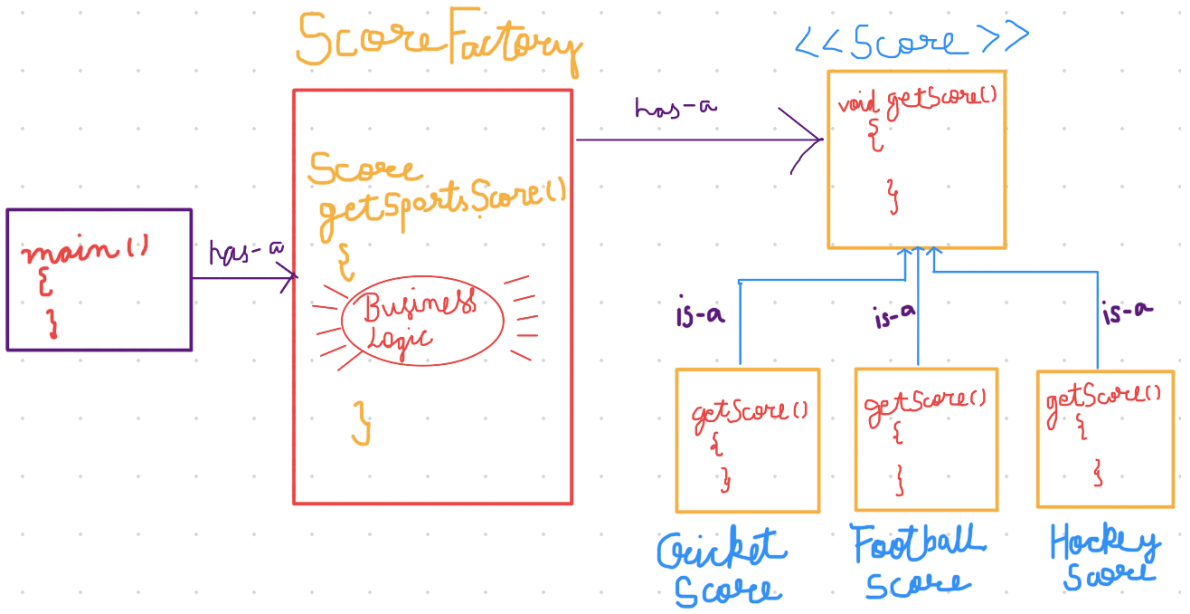
```
if(some condition is true) {
return new CricketScore();
}
```

This seems alright when the business logic is not too complex and there are only a few number of classes. But, if the business logic was more complex than just a single if statement and there are multiple classes which require an object of type **Score**, be it ***Cricket, Football* or *Hockey***, then it would become very *cumbersome to inject the business logic in each class* where we require that object.

So, in order to **separate the business logic** for object generation from the classes in which we require the object, we **delegate this work to another class** (known as the **Factory**) which would take care of ***object creation based on the business logic conditions*** we pass to it.

As the name suggests, ***Factory is something that manufactures/creates something***. In our case, we can have a **ScoreFactory** Class, which would generate objects of type **Score** for us.

You can imagine it to be something like this:-



Code Snippets

(Read through the code comments for better perspective)

1. We have an interface **Score** with an abstract method **getScore()**

```java
package factorypattern;

public interface Score {
    // Abstract Method getScore()
    public void getScore();
}

```

2. We have some **classes** which implement **Score** interface and provide implementation to the **getScore()** method

a) Cricket

```java
package factorypattern;

public class Cricket implements Score {
    // Provide implementation to getScore()
    public void getScore(){
        System.out.println("This is the score of Cricket!");
    }
}

```

b) Football

```java
package factorypattern;

public class Football implements Score {
    // Provide implementation to getScore()
    public void getScore(){
        System.out.println("This is the score of Football!");
    }
}

```

c) Hockey

```java
package factorypattern;

public class Hockey implements Score {
    // Provide implementation to getScore()
    public void getScore(){
        System.out.println("This is the score of Hockey!");
    }
}

```

3. We have a **ScoreFactory** class which has a **getSportsScore()** method. In this method, we will receive some conditions as arguments, based on which it would return an object of type **Score**

```java
package factorypattern;

public class ScoreFactory {
    /* getSportsScore(args) -> It receives a condition ,
       based on which objects are returned.
     */
    public  Score getSportsScore(String sport){
        switch (sport){
            case "Cricket": {
                return new Cricket();
            }
            case "Football": {
                return new Football();
            }
            case "Hockey": {
                return new Hockey();
            }
            default:{
              return null;
            }
        }
    }
}
```

4. Here, we create an **object** of **ScoreFactory** and use its object to get the score of different sports. We call the ***getSportsScore(arg)*** with an argument which will act the condition based on which we will get the desired **object,** be it ***Cricket, Football or Hockey***.

```java
package factorypattern;

public class Main {
    public static void main(String[] args) {
        // Create an object of the Score Factory
        ScoreFactory sf = new ScoreFactory();

        // Get objects of different Sports from Score Factory
        Score cricket = sf.getSportsScore("Cricket");
        Score football = sf.getSportsScore("Football");
        Score hockey = sf.getSportsScore("Hockey");

        // Call getScore() of different sports
        cricket.getScore();
        football.getScore();
        hockey.getScore();
    }
}
```

Output

```
Run    factorypattern.Main

"C:\Program Files\Java\jdk-17\bin\java.exe"
This is the score of Cricket!
This is the score of Football!
This is the score of Hockey!

Process finished with exit code 0
```

Thanks for reading!

Bhavuk Jain