Factory Design Pattern

Design the factory method in which multiple objects are stored called as factory design pattern.

Why?

Sometime our application or frameworks don't know what kind of object has to create at run time. It only knows when it has to create.

Another issue is that class will contain object of another classes, this can be easily achieved by just using the new keyword and the class constructor. The problem with this approach is that it is very hard coded approach to create the object as this creates dependency between two classes.

To overcome this above situation, we should go for factory pattern.

When?

When user wants the specific object at run time.

Example:-

Suppose we have requirement to book AC ticket for First tier, second tier and Third tier.

Step 1 create the interface Booking.

Step 2- create the concreate class First tier will implement the same interface

Step 3- create the concreate class Second tier will implement the same interface

Step 4- create the concreate class Third tier will implement the same interface

Step 5- create class BookingFactory to generate the object of concreate class.

Step 6- Create the factory class to get the object of concreate class by passing the data.

Step 1 create the interface Booking

```
package com.test;

//step-1
public interface Booking {
```

```
    public String getACClass();
}
```

Step 2- create the concreate class First tier will implement the same interface

```
package com.test;

public class FirstTier implements Booking {

    @Override
    public String getACClass() {
        return "first class-AC- Seat availability:10";
    }

}
}
```

Step 3- create the concreate class Second tier will implement the same interface

```
package com.test;

public class SecondTier implements Booking {

    @Override
    public String getACClass() {
        return "Second class- AC- Seat availability:8";
    }


}
```

Step 4- create the concreate class Third tier will implement the same interface

```
package com.test;

public class ThirdTier implements Booking {

    @Override
    public String getACClass() {
        return "Third class-AC Seat availability:2";
```

```
	}

}
```

Step 5- create class BookingFactory to generate the object of concreate class.

```
package com.test;

public class BookingFactory {

	// design the factory method here
	// input should be first,second,third
	public static Booking createBooking(String input) {
		if (input.equalsIgnoreCase("first")) {
			return new FirstTier();
		} else if (input.equalsIgnoreCase("second")) {
			return new SecondTier();
		} else if (input.equalsIgnoreCase("third")) {
			return new ThirdTier();
		}
		throw new IllegalArgumentException("Invalid input
from user..");
	}
}
```

Step 6- Create the factory class to get the object of concreate class by passing the data.

```
package com.test;

import java.util.Scanner;

public class TestMain {

	public static void main(String[] args) {
		try {

			String booking;
```

```
                System.out.println("Enter the AC class
type>>");
                Scanner scanner = new Scanner(System.in);
                booking = scanner.next();
                Booking b =
BookingFactory.createBooking(booking);
                System.out.println(b.getACClass());
                scanner.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

Output-

```
Enter the AC class type>>
first
first class-AC- Seat availability:10
```