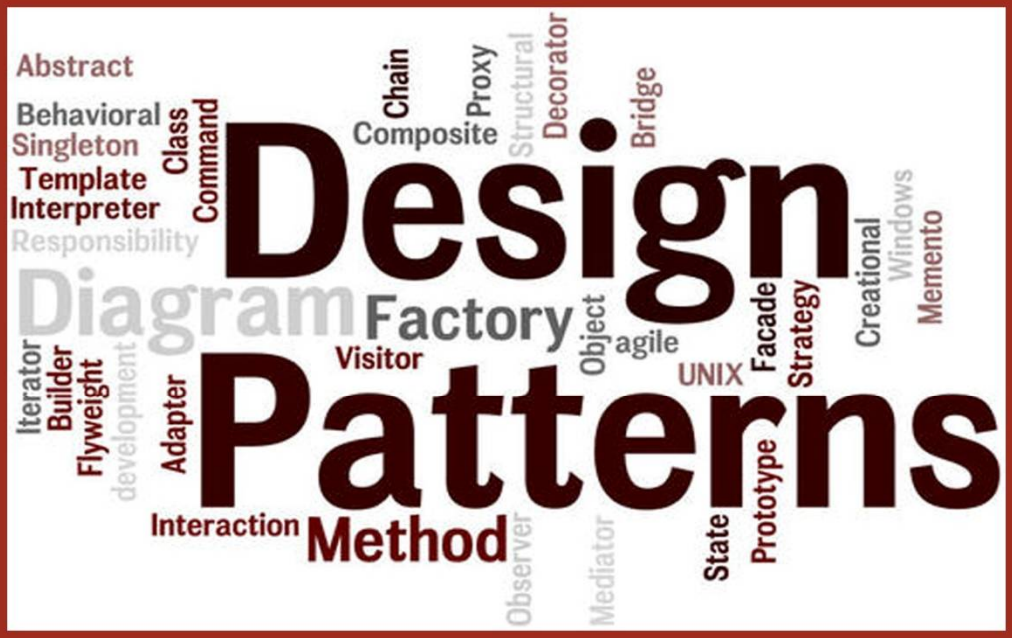
JAVA means DURGA SOFT

Design Patterns

Core Level Design Patterns

1. Factory Method



India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Core Level Design Patterns

Factory Method:

Problem: When a java class is having private constructor, then outside of that class object creation of that class is not possible.

Solution: Use Factory method design pattern.

Def: Factory method is java method, having a capability of constructing and returning its own java class object.

Application Areas:

- When java class is having only private constructor then in order to create object of that java class outside of that class, we use Factory method.
- To make java class as Immutable java class (like java.lang.String), we use factory method as helper class or code.

Rules to use this Factory method:

1. The Factory method should have current class name as return type.
2. Method definition should have logic of creating & returning current class object.
3. Factory method must be a public method.

Note:

- The factory method can be a **static factory method** or an **instance factory method.**
- If current class is an abstract class, then method should return one of its subclass Object.

- The **Static Factory Method** is useful to create an object of a java class outside of that class when that class is having only private constructor.

Ex: Some of the examples for pre defined static factory methods are:

I. Class c= Class.forName();

II. Thread t=Thread.currentThread();

Note Java class can have Factory methods even though that class is having public constructors

- The **Instance Factory Method** is useful to create a new object for java class by using existing object and its data.

  Ex: Example for pre defined instance factory method is:

  String s1="welcome to durga software solutions"; // Existing Object

  String s2=s1.substring(3,7); // New Object i.e. it gives **come**

Now we can see one example on how we can define user defined static factory method and instance factory method

❖ **Sample Code:**

```
// FactoryEx.java
class Test
{
        int x;

        //private zero argument constructor
        private Test()
        {
                System.out.println("Test: 0-arg Con");
        }

        //static factory method
        public static Test staticFactoryMethod()
        {
                Test t=new Test();
                t.x=5;
                return t;
        }
```

```
        //instance factory method
        public Test instanceFactoryMethod()
        {
                Test t=new Test();
                t.x=this.x+5;
                return t;
        }

        public String toString()
        {
                return "x="+x;
        }
}; //class: Test
public class FactoryEx
{
        public static void main(String[] args) throws Exception
        {
                Test t1=Test.staticFactoryMethod();
                System.out.println(t1); // which internally calls toString()
                Test t2=t1.instanceFactoryMethod();
                System.out.println(t2);
        }//main
}//class: FactoryEx
```

Output:

```
D:\Java\Design Patterns\Programs\Factory Method>javac FactoryEx.java

D:\Java\Design Patterns\Programs\Factory Method>java  FactoryEx
Test: 0-arg Con
x=5
Test: 0-arg Con
x=10
```

❖ Now we can see one example on how we can develop our own **Immutable java class** by using Factory method**.**

Definition of Immutable Java Class: Immutable class is a java class which once created, its contents cannot be changed. Immutable objects are the java objects whose state cannot be changed once constructed. E.g. String class

Advantages with Immutable Nature:

- Immutable objects are automatically thread-safe, the overhead caused due to use of synchronization is avoided.
- Once created the state of the immutable object cannot be changed so there is no possibility of them getting into an inconsistent state.
- The references to the immutable objects can be easily shared or cached without having to copy or clone them as there state cannot be changed ever after construction.
- The best use of the immutable objects is as the keys of a map.

Rules to follow to develop an immutable class:

1. All properties of that object are to be marked as private.

2. Do not provide any methods such as setters or any methods that may alter the properties of that object. If you do, make sure they return a new instance of your object.
3. All properties and methods of that object are to be marked as final. Marking the methods as final is extremely important. We don't want anyone carelessly overriding our functions that may compromise our immutable behavior.

Sample Code:

```
// Demo.java
public final class Demo
{
	private String name;
	private  int no;

	public Demo()
	{
		System.out.println("0- argument constructor");
	}
	public Demo(String name,int no)
	{
		this.name=name;
		this.no=no;
		System.out.println("2- arguments constructor");
	}
	public Demo(String s)
	{
		this.name=s;
		System.out.println("1- argument constructor");
	}
	public Demo(int n)
```

```
	{
		this.no=n;
		System.out.println("1- argument constructor");
	}
	//instance factory method
	public Demo modifyName(String s)
	{
		Demo d=new Demo(s);
		d.no=this.no;
		return d;
	}

	//instance factory method
	public Demo modifyNo(int n)
	{
		Demo d=new Demo(n);
		d.name=this.name;
		return d;
	}
	public String toString()
	{
		return "Name="+name+"	No="+no;
	}
};
class ImmutableTest
{
	public static void main(String[] args)
	{
		Demo d1=new Demo("durga", 35);
```

```
        System.out.println("Hashcode of d1 Object: "+d1.hashCode());
        System.out.println(d1.toString());
        System.out.println();

        Demo d2=d1.modifyName("venkat");
        System.out.println("Hashcode of d2 Object: "+d2.hashCode());
        System.out.println(d2); // which internally calss toString() to print those values
        System.out.println();

        Demo d3=d1.modifyNo(25);
        System.out.println("Hashcode of d3 Object: "+d3.hashCode());
        System.out.println(d3);
    }
}
```

Output:

```
D:\Java\Design Patterns\Programs\Immutable Class\Ex2>javac Demo.java

D:\Java\Design Patterns\Programs\Immutable Class\Ex2>java ImmutableTest
2- arguments constructor
Hashcode of d1 Object: 1671711
Name=durga      No=35

1- argument constructor
Hashcode of d2 Object: 11394033
Name=venkat     No=35

1- argument constructor
Hashcode of d3 Object: 4384790
Name=durga      No=25
```