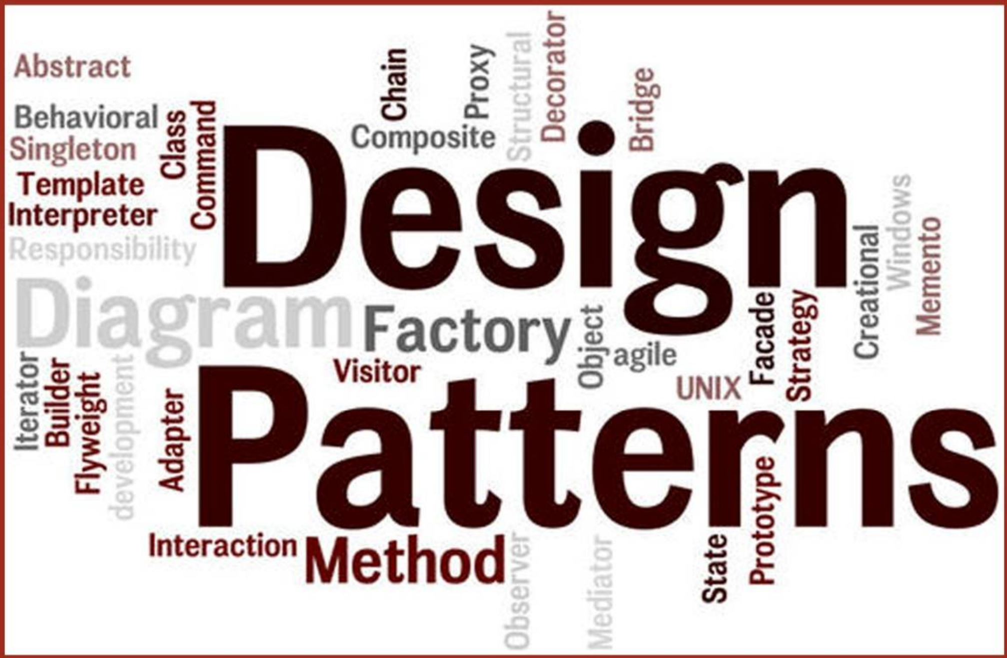
JAVA means DURGA SOFT

Design Patterns

Core Level Design Patterns

5. Abstract Factory Pattern



India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Abstract Factory Pattern

The Abstract Factory pattern is one level of abstraction higher than the Factory Method pattern. You can use this pattern to return one of several related classes of objects, each of which can return several different objects on request. In other words, the Abstract Factory is a factory object that returns one of several groups of classes. You might even decide which class to return from that group by using a Simple Factory.

This pattern is an enhancement of Factory patterns which provides one level of abstraction higher than factory pattern. This means that the abstract factory returns the factory of classes. Like Factory pattern returned one of the several sub-classes, this returns such factory which later will return one of the sub-classes.

It provides a way to encapsulate a group of several related factories. This method is used when to return one of several related classes of objects and each of which have the capability of returning several objects of different types on request. This pattern provides separation from the implementation details of a set of objects from its general usage. This pattern hides the concrete subclass from the client and should be used when the system is independent of how the components are organized.

Def: Provide an interface for creating families of related or dependent objects without

specifying their concrete classes.

Intent:

- ✓ Abstract Factory offers the interface for creating a family of related objects, without explicitly specifying their classes.

Application Areas: We should use the Abstract Factory design pattern when:

- A system should be independent of how its products are created, composed,and represented.
- A system should be configured with one of multiple families of products.
- A family of related product objects is designed to be used together, and you need to enforce this constraint.
- You want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.

Ex:

In JDBC applications, **DriverManager.getConnection()** implements Factory pattern in the process of implementing Abstract Factory. Because this method returns one implementation class object of java.sql.Connection(I) i.e.Connection Object and that connection object can be used to create Statement objcet.

eg: Connection con=Drivermanager.getConnection("jdbc:odbc:durga","uname","pwd");

Statement st=con.createStatement();

Here we are able to work with the JDBC Connection & Statement objcets with out specifying their class names.

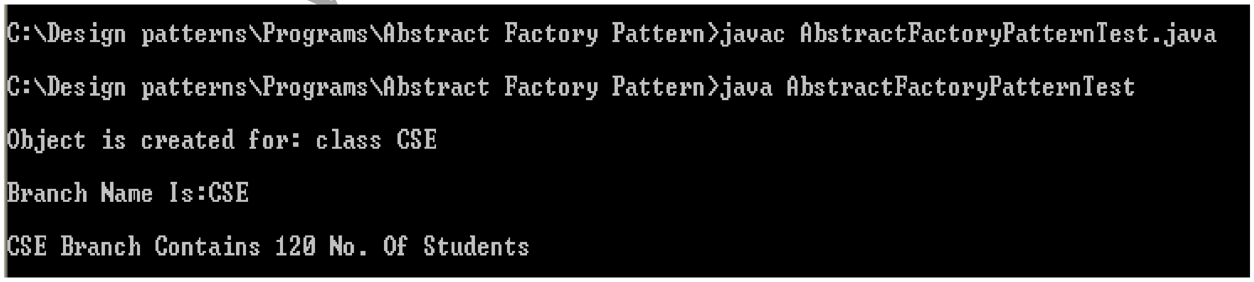
Sample Code:

// AbstractFactoryPatternTest.java

```
abstract class College
{
        public abstract Display getBranchName();
        public abstract Display getCount();
};
class CSE extends College
{
        public Display getBranchName()
        {
                return new Display("CSE");
        }
        public Display getCount()
        {
                return new Display("120");
        }
};
class IT extends College
{
        public Display getBranchName()
        {
                return new Display("IT");
        }
        public Display getCount()
        {
                return new Display("90");
        }
};
class MCA extends College
{
	public Display getBranchName()
	{
		return new Display("MCA");
	}
	public Display getCount()
	{
		return new Display("60");
	}
};
class Display
{
	public String value;
	public Display(String s)
	{
		this.value=s;
	}
	public String getValue()
	{
		return value;
	}
};
public class AbstractFactoryPatternTest
{
	private College cname;
	public static void main(String[] args)
	{
		AbstractFactoryPatternTest af=new AbstractFactoryPatternTest();
		College c=af.getCollege("IT");
		System.out.println();
```

```
                System.out.println("Object is created for: "+c.getClass());
                System.out.println();
                System.out.println("Branch Name Is:"+c.getBranchName().getValue());
                System.out.println();
                System.out.println(c.getBranchName().getValue()+"Branch
                        Contains "+c.getCount().getValue()+" No.Of Students");
        }
        //Factory Pattern Logic
        public College getCollege(String s)
        {
                if(s.equals("CSE"))
                        cname=new CSE();
                else if (s.equals("IT"))
                        cname=new IT();
                else if (s.equals("MCA"))
                        cname=new MCA();
                return cname;
        }
}
```

Output:



```
C:\Design patterns\Programs\Abstract Factory Pattern>javac AbstractFactoryPatternTest.java

C:\Design patterns\Programs\Abstract Factory Pattern>java AbstractFactoryPatternTest

Object is created for: class CSE

Branch Name Is:CSE

CSE Branch Contains 120 No. Of Students
```