Creational Design Patterns

By Raza Sikander
Project Engineer
CDAC Hyderabad

Creational Design Pattern

- Creational design patterns are concerned with the way of creating objects.
- These design patterns are used when a decision must be made at the time of instantiation of a class

Creational Design Pattern

- Singleton Pattern
- Builder Pattern
- Factory Method Pattern
- Abstract Factory Pattern
- Prototype Pattern

Singleton Pattern

“Define a class that has only one instance and provides a global point of access to it”

Singleton Pattern

“Define a class that has only one instance and provides a global point of access to it”

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.

Singleton Pattern

There are two forms of singleton design pattern

- Early Instantiation: creation of instance at load time
- Lazy Instantiation: creation of instance when required

Singleton Pattern

Advantage of Singleton design pattern

- Saves memory because object is not created at each request. Only single instance is reused again and again.

Singleton Pattern

Usage of Singleton design pattern

- Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

Singleton Pattern

Early Instantiation

```
package DesignPatterns;

public class Singleton {
    private static Singleton instance = new Singleton(); //Eager instantiated

    private Singleton()
    {
        System.out.println("Singleton being initialized");
    }

    public static Singleton getInstance()
    {
        return instance;
    }
}
```

Singleton Pattern

Lazy Instantiation

```
package DesignPatterns;

public class Singleton {
    private static Singleton instance = null;

    private Singleton()
    {
        System.out.println("Singleton being initialized");
    }

    public static Singleton getInstance() //Only instantiated when getInstance is invoked
    {
        if(instance == null)
            instance = new Singleton();
        return instance;
    }
}
```

Singleton Pattern

Implementation of Singleton Pattern Example

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Start Page | DesignPattern.java | SingletonPattern.java

Source History

```java
package designpattern;

/**
 *
 * @author shaik
 */

class Singleton
{
        // static variable single_instance of type Singleton
        private static Singleton single_instance = null;

        // variable of type String
        public String s;

        // private constructor restricted to this class itself
        private Singleton()
        {
                s = "Hello I am a string part of Singleton class";
        }

        // static method to create instance of Singleton class
        public static Singleton getInstance()
        {
                if (single_instance == null)
                        single_instance = new Singleton();

                return single_instance;
        }
}
```

designpattern.Singleton

Output | Navigator

34:11 INS

Type here to search

ENG IN 12:17 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Start Page | DesignPattern.java | SingletonPattern.java

```java
public class SingletonPattern{
        public static void main(String args[])
        {
                // instantiating Singleton class with variable x
                Singleton x = Singleton.getInstance();
                Singleton y = Singleton.getInstance();
                Singleton z = Singleton.getInstance();
                //Print hash code of the object
                System.out.println(x.hashCode());
                System.out.println(y.hashCode());
                System.out.println(z.hashCode());


                // changing variable of instance x
                x.s = (x.s).toUpperCase();

                System.out.println("String from x is " + x.s);
                System.out.println("String from y is " + y.s);
                System.out.println("String from z is " + z.s);
                System.out.println("\n");

                // changing variable of instance z
                z.s = (z.s).toLowerCase();

                System.out.println("String from x is " + x.s);
                System.out.println("String from y is " + y.s);
                System.out.println("String from z is " + z.s);
        }
}
```

designpattern.SingletonPattern

Output | Navigator

37:1 INS

Type here to search

ENG IN 12:20 12-11-2019

Builder Pattern

"Construct a complex object from simple objects using step-by-step approach"

Builder Pattern

"Construct a complex object from simple objects using step-by-step approach"

It is mostly used when object can't be created in single step like in the de-serialization of a complex object.

Builder Pattern

The main advantages of Builder Pattern are as follows:

- It provides clear separation between the construction and representation of an object.
- It provides better control over construction process.
- It supports to change the internal representation of objects.

Builder Pattern

Implementation of Builder Pattern Example

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Start Page | DesignPattern.java | SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java

Source History

```java
13  public class BuilderPattern1 {
14
15          //required parameters
16          private String HDD;
17          private String RAM;
18
19          //optional parameters
20          private boolean isGraphicsCardEnabled;
21          private boolean isBluetoothEnabled;
22
23          public String getHDD() {
24                  return HDD;
25          }
26          public String getRAM() {
27                  return RAM;
28          }
29          public boolean isGraphicsCardEnabled() {
30                  return isGraphicsCardEnabled;
31          }
32          public boolean isBluetoothEnabled() {
33                  return isBluetoothEnabled;
34          }
35          private BuilderPattern1(ComputerBuilder builder) {
36                  this.HDD=builder.HDD;
37                  this.RAM=builder.RAM;
38                  this.isGraphicsCardEnabled=builder.isGraphicsCardEnabled;
39                  this.isBluetoothEnabled=builder.isBluetoothEnabled;
40          }
41          @Override
            public String toString() {
```

designpattern.BuilderPattern1 > getHDD

Output | Navigator

23:33 INS

Type here to search

ENG IN 12:37 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Start Page | DesignPattern.java | SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java

Source History

```java
        //Builder Class
        public static class ComputerBuilder{

                // required parameters
                private String HDD;
                private String RAM;

                // optional parameters
                private boolean isGraphicsCardEnabled;
                private boolean isBluetoothEnabled;

                public ComputerBuilder(String hdd, String ram){
                        this.HDD=hdd;
                        this.RAM=ram;
                }

                public ComputerBuilder setGraphicsCardEnabled(boolean isGraphicsCardEnabled) {
                        this.isGraphicsCardEnabled = isGraphicsCardEnabled;
                        return this;
                }

                public ComputerBuilder setBluetoothEnabled(boolean isBluetoothEnabled) {
                        this.isBluetoothEnabled = isBluetoothEnabled;
                        return this;
                }

                public BuilderPattern1 build(){
                        return new BuilderPattern1(this);
                }

```

designpattern.BuilderPattern1 > ComputerBuilder > ComputerBuilder

Output | Navigator

60:18 INS

Type here to search

ENG IN 12:38 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Start Page | DesignPattern.java | SingletonPattern.java | **BuilderPattern1.java** | BuilderPattern2.java

Source History

```java
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package designpattern;

/**
 *
 * @author shaik
 */

public class BuilderPattern2 {

        public static void main(String[] args) {
                //Using builder to get the object in a single line of code and
                //without any inconsistent state or arguments management issues
                BuilderPattern1 comp = new BuilderPattern1.ComputerBuilder(
                                "500 GB", "2 GB").setBluetoothEnabled(true)
                                .setGraphicsCardEnabled(true).build();
                System.out.println(comp.toString());
        }

}
```

designpattern.BuilderPattern2

Output | Navigator

13:1 INS

Type here to search

ENG IN 12:38 12-11-2019

Factory Method Pattern

define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate

Factory Method Pattern

define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate

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

Factory Method Pattern

Usage of Factory Design Pattern

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

Factory Method Pattern

Implementation of Factory Method Pattern Example

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

...ava | SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java | FactoryDP1.java | FactoryDP2.java | FactoryDP3.java | FactoryDP4.java | FactoryDP5.java

Source History

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package CreationalDesignPattern;

/**
 *
 * @author shaik
 */

public abstract class FactoryDP1 { // Computer ABstarct

        public abstract String getRAM();
        public abstract String getHDD();
        public abstract String getCPU();

        @Override
        public String toString(){
                return "RAM= "+this.getRAM()+", HDD="+this.getHDD()+", CPU="+this.getCPU();
        }
}
```

Output

8:4 INS

Type here to search

ENG INTL 13:01 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

...ava SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java | FactoryDP1.java | FactoryDP2.java | FactoryDP3.java | FactoryDP4.java | FactoryDP5.java

Source History

```java
14
15  public class FactoryDP2 extends FactoryDP1 { //Personal Computer
16
        private String ram;
        private String hdd;
        private String cpu;
20
21      public FactoryDP2(String ram, String hdd, String cpu){
22          this.ram=ram;
23          this.hdd=hdd;
24          this.cpu=cpu;
25      }
26      @Override
        public String getRAM() {
28          return this.ram;
29      }
30
31      @Override
        public String getHDD() {
33          return this.hdd;
34      }
35
36      @Override
        public String getCPU() {
38          return this.cpu;
39      }
40
41  }
42
```

CreationalDesignPattern.FactoryDP2

Output

15:65 INS

Type here to search

ENG INTL 13:01 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

...ava | SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java | FactoryDP1.java | FactoryDP2.java | FactoryDP3.java | FactoryDP4.java | FactoryDP5.java

Source History

```
11      */
12
13
14  public class FactoryDP3 extends FactoryDP1 {//Server Computer
15
16          private String ram;
17          private String hdd;
18          private String cpu;
19
20          public FactoryDP3(String ram, String hdd, String cpu){
21                  this.ram=ram;
22                  this.hdd=hdd;
23                  this.cpu=cpu;
24          }
25          @Override
26          public String getRAM() {
27                  return this.ram;
28          }
29
30          @Override
31          public String getHDD() {
32                  return this.hdd;
33          }
34
35          @Override
36          public String getCPU() {
37                  return this.cpu;
38          }
39
40  }
41
```

CreationalDesignPattern.FactoryDP3

Output

14:62 INS

Type here to search

ENG INTL 13:01 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

...ava SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java | FactoryDP1.java | FactoryDP2.java | FactoryDP3.java | FactoryDP4.java | FactoryDP5.java

Source History

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package CreationalDesignPattern;

/**
 *
 * @author shaik
 */



public class FactoryDP4 { //Computer Factory

        public static FactoryDP1 getComputer(String type, String ram, String hdd, String cpu){
                if("PC".equalsIgnoreCase(type)) return new FactoryDP2(ram, hdd, cpu);
                else if("Server".equalsIgnoreCase(type)) return new FactoryDP3(ram, hdd, cpu);

                return null;
        }
}
```

CreationalDesignPattern.FactoryDP4

Output

16:45 INS

Type here to search

ENG INTL 13:01 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

...ava SingletonPattern.java | BuilderPattern1.java | BuilderPattern2.java | FactoryDP1.java | FactoryDP2.java | FactoryDP3.java | FactoryDP4.java | FactoryDP5.java

Source History

```
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package CreationalDesignPattern;

/**
 *
 * @author shaik
 */


public class FactoryDP5 { //Client

        public static void main(String[] args) {
                FactoryDP1 pc = FactoryDP4.getComputer("pc","2 GB","500 GB","2.4 GHz");
                FactoryDP1 server = FactoryDP4.getComputer("server","16 GB","1 TB","2.9 GHz");
                System.out.println("Factory PC Config::"+pc);
                System.out.println("Factory Server Config::"+server);
        }

}
```

CreationalDesignPattern.FactoryDP5

Output

14:35 INS

Type here to search

ENG INTL 13:01 12-11-2019

Abstract Factory Pattern

Abstract Factory pattern is almost similar to Factory Pattern except the fact that its more like factory of factories.

Abstract Factory Pattern

define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes

Abstract Factory Pattern

Advantage of Abstract Factory Pattern

- Abstract Factory Pattern isolates the client code from concrete (implementation) classes.
- It eases the exchanging of object families.
- It promotes consistency among objects.

Abstract Factory Pattern

Implementation of Abstract Factory Method Pattern Example

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | **AbstractFP6.java** | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

Source History

```java
/**
 *
 * @author shaik
 */
//1 Computer.java
//2 PC.java
//3 Server.java
//4 ComputerAbstractFactory.java
//5 PCFactory.java
//6 ServerFactory.java
//7 ComputerFactory.java
//8 TestDesignPatterns.java


public abstract class AbstractFP1 {

    public abstract String getRAM();
    public abstract String getHDD();
    public abstract String getCPU();

    @Override
    public String toString(){
        return "RAM= "+this.getRAM()+", HDD="+this.getHDD()+", CPU="+this.getCPU();
    }
}
```

Output | Navigator

20:1 INS

Type here to search

ENG IN 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | **AbstractFP6.java** | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

```java

public class AbstractFP2 extends AbstractFP1 { //PC.java

    private String ram;
    private String hdd;
    private String cpu;

    public AbstractFP2(String ram, String hdd, String cpu){
        this.ram=ram;
        this.hdd=hdd;
        this.cpu=cpu;
    }
    @Override
    public String getRAM() {
        return this.ram;
    }

    @Override
    public String getHDD() {
        return this.hdd;
    }

    @Override
    public String getCPU() {
        return this.cpu;
    }

}

```

Output | Navigator | 20:1 | INS

Type here to search | ENG IN | 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | **AbstractFP6.java** | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

Source History

```java
public class AbstractFP3 extends AbstractFP1 {

    private String ram;
    private String hdd;
    private String cpu;

    public AbstractFP3(String ram, String hdd, String cpu){
        this.ram=ram;
        this.hdd=hdd;
        this.cpu=cpu;
    }
    @Override
    public String getRAM() {
        return this.ram;
    }

    @Override
    public String getHDD() {
        return this.hdd;
    }

    @Override
    public String getCPU() {
        return this.cpu;
    }

}
```

Output | Navigator

20:1 INS

Type here to search

ENG IN 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | **AbstractFP6.java** | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

Source History

```
8   /**
9    *
10   * @author shaik
11   */
12
13  //1 Computer.java
14  //2 PC.java
15  //3 Server.java
16  //4 ComputerAbstractFactory.java
17  //5 PCFactory.java
18  //6 ServerFactory.java
19  //7 ComputerFactory.java
20  //8 TestDesignPatterns.java
21
    public interface AbstractFP4 {
23
            public AbstractFP1 createComputer();
25
26  }
27
```

Output | Navigator

21:1

Type here to search

ENG IN 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | **AbstractFP6.java** | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

C:\Users\shaik\Documents\NetBeansProjects\DesignPattern\src\CreationalDesignPattern\AbstractFP5.java

```
//7 ComputerFactory.java
//8 TestDesignPatterns.java

public class AbstractFP5 implements AbstractFP4 {

        private String ram;
        private String hdd;
        private String cpu;

        public AbstractFP5(String ram, String hdd, String cpu){
                this.ram=ram;
                this.hdd=hdd;
                this.cpu=cpu;
        }
        @Override
        public AbstractFP1 createComputer() {
                return new AbstractFP2(ram,hdd,cpu);
        }

}
```

Output | Navigator

21:1 INS

Type here to search

ENG IN 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | AbstractFP6.java | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

```java
//3 Server.java
//4 ComputerAbstractFactory.java
//5 PCFactory.java
//6 ServerFactory.java
//7 ComputerFactory.java
//8 TestDesignPatterns.java

public class AbstractFP6 implements AbstractFP4 {

        private String ram;
        private String hdd;
        private String cpu;

        public AbstractFP6(String ram, String hdd, String cpu){
                this.ram=ram;
                this.hdd=hdd;
                this.cpu=cpu;
        }

        @Override
        public AbstractFP1 createComputer() {
                return new AbstractFP3(ram,hdd,cpu);
        }

}
```

Output | Navigator

21:1 INS

Type here to search

ENG IN 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | AbstractFP6.java | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

Source History

```
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package CreationalDesignPattern;

/**
 *
 * @author shaik
 */
//1 Computer.java
//2 PC.java
//3 Server.java
//4 ComputerAbstractFactory.java
//5 PCFactory.java
//6 ServerFactory.java
//7 ComputerFactory.java
//8 TestDesignPatterns.java


public class AbstractFP7 {

        public static AbstractFP1 getComputer(AbstractFP4 factory){
                return factory.createComputer();
        }
}
```

Output | Navigator

20:1

Type here to search

ENG IN 18:14 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

AbstractFP1.java | AbstractFP2.java | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | AbstractFP6.java | FactoryDP2.java | AbstractFP7.java | AbstractFP8.java

```java
//1 Computer.java
//2 PC.java
//3 Server.java
//4 ComputerAbstractFactory.java
//5 PCFactory.java
//6 ServerFactory.java
//7 ComputerFactory.java
//8 TestDesignPatterns.java


public class AbstractFP8 {

        public static void main(String[] args) {
                testAbstractFactory();
        }

        private static void testAbstractFactory() {
                AbstractFP1 pc = AbstractFP7.getComputer(new AbstractFP5("2 GB","500 GB","2.4 GHz"));
                AbstractFP1 server = AbstractFP7.getComputer(new AbstractFP6("16 GB","1 TB","2.9 GHz"));
                System.out.println("AbstractFactory PC Config::"+pc);
                System.out.println("AbstractFactory Server Config::"+server);
        }
}
```

Output | Navigator

22:1 IN

Type here to search

ENG IN 18:14 12-11-2019

Prototype Design Pattern

cloning of an existing object instead of creating new one and can also be customized as per the requirement

Prototype Pattern

The main advantages of prototype pattern are as follows:

- It reduces the need of sub-classing.
- It hides complexities of creating objects.
- The clients can get new objects without knowing which type of object it will be.
- It lets you add or remove objects at runtime.

Prototype Pattern

Usage of Prototype Pattern

- When the classes are instantiated at runtime.
- When the cost of creating an object is expensive or complicated.
- When you want to keep the number of classes in an application minimum.
- When the client application needs to be unaware of object creation and representation.

Prototype Pattern

Implementation of Prototype Pattern Example

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

...ava | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | AbstractFP6.java | FactoryDP2.java | AbstractFP7.java | AbstactFP8.java | EmpProto.java | PrototypePattern.java

Source | History

```
 * @author shaik
 */


import java.util.ArrayList;
import java.util.List;

public class EmpProto implements Cloneable{

        private List<String> empList;

        public EmpProto(){
                empList = new ArrayList<String>();
        }

        public EmpProto(List<String> list){
                this.empList=list;
        }
        public void loadData(){
                //read all employees from database and put into the list
                empList.add("Pankaj");
                empList.add("Raj");
                empList.add("David");
                empList.add("Lisa");
        }

        public List<String> getEmpList() {
                return empList;
        }
```

CreationalDesignPattern.EmpProto

Output | Navigator

39:1 INS

Type here to search

ENG IN 18:32 12-11-2019

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

...ava | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | AbstractFP6.java | FactoryDP2.java | AbstractFP7.java | AbstactFP8.java | EmpProto.java | PrototypePattern.java

Source History

```
30                  empList.add("Pankaj");
31                  empList.add("Raj");
32                  empList.add("David");
33                  empList.add("Lisa");
34          }
35
36          public List<String> getEmpList() {
37                  return empList;
38          }
39
40          @Override
          public Object clone() throws CloneNotSupportedException{
                  List<String> temp = new ArrayList<String>();
43                  for(String s : this.getEmpList()){
44                          temp.add(s);
45                  }
46                  return new EmpProto(temp);
47          }
48
          }
50
```

CreationalDesignPattern.EmpProto

Output | Navigator

49:2/10:213 IN

Type here to search

ENG 18:33

DesignPattern - NetBeans IDE 8.0.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

...ava | AbstractFP3.java | AbstractFP4.java | AbstractFP5.java | AbstractFP6.java | FactoryDP2.java | AbstractFP7.java | AbstactFP8.java | EmpProto.java | PrototypePattern.java

Source History

```java

import java.util.List;


public class PrototypePattern {

        public static void main(String[] args) throws CloneNotSupportedException {
                EmpProto emps = new EmpProto();
                emps.loadData();

                //Use the clone method to get the Employee object
                EmpProto empsNew = (EmpProto) emps.clone();
                EmpProto empsNew1 = (EmpProto) emps.clone();
                List<String> list = empsNew.getEmpList();
                list.add("John");
                List<String> list1 = empsNew1.getEmpList();
                list1.remove("Pankaj");

                System.out.println("emps List: "+emps.getEmpList());
                System.out.println("empsNew List: "+list);
                System.out.println("empsNew1 List: "+list1);
        }

}

```

CreationalDesignPattern.PrototypePattern > main

Output | Navigator

31:32

Type here to search

ENG 18:33

Assignment

1. Create a Builder Pattern for a Mobile which has following parameters
    a. Ram
    b. Os
    c. Internal Storage
    d. Battery Capacity
    e. Company
2. Create a Factory Pattern and Abstract Factory Pattern of Vehicles which to create a car, bus ,bike,truck which has input parameters as No. of passengers, No of Tyres, Engine Capacity etc
3. Create a Prototype pattern for an educational institute which should have some sample lists like students, facutly, incharge, attenders etc

References

- https://www.javatpoint.com/singleton-design-pattern-in-java
- https://sandeepdass003.wordpress.com/2018/02/23/eager-and-lazy-instantiation-in-singleton-design-pattern-implementation/
- https://www.javatpoint.com/builder-design-pattern
- https://www.journaldev.com/1425/builder-design-pattern-in-java
- https://www.javatpoint.com/factory-method-design-pattern
- https://www.journaldev.com/1392/factory-design-pattern-in-java
- https://www.journaldev.com/1418/abstract-factory-design-pattern-in-java
- https://www.javatpoint.com/abstract-factory-pattern
- https://www.javatpoint.com/prototype-design-pattern
- https://www.journaldev.com/1440/prototype-design-pattern-in-java