**Prototype Design Pattern**

**Assignment - 1**

Name: Heet Dobariya Roll No. - 22BCP177 Group - G5

* **Prototype Design Pattern :**

The Prototype Design Pattern is a creational design pattern that deals with object creation mechanisms, trying to create objects in a manner suitable to the situation. The pattern creates new objects by copying an existing object, known as the prototype. This process is especially useful when the cost of creating an object is more expensive or complex than copying an existing one.

The Prototype pattern involves creating new objects by copying an existing object, known as the prototype. The basic idea is to use a prototype instance as a blueprint for creating new objects, allowing the copying of properties, data, and structure from the prototype to the new instances.

* **Types of Clones :**

1. Shallow Clone:

--> A shallow clone creates a new object but does not create copies of the nested objects within the original object.

--> In other words, the cloned object and the original object share references to the same nested objects.

--> Changes made to the nested objects in the cloned instance may affect the original instance and vice versa.

1. Deep Clone:

--> A deep clone creates a new object and also creates copies of all the nested objects within the original object.

--> The cloned object and the original object have their independent copies of nested objects.

--> Changes made to the nested objects in the cloned instance do not affect the original instance and vice versa.

* **Program :** Create a Prototype Design Pattern for Vehicle Example.
* **Code :**

import java.util.\*;

public class Vehicle implements Cloneable

{

private List<String> vehiclelist;

public Vehicle()

{

this.vehiclelist = new ArrayList<String>();

}

public Vehicle(List<String> list)

{

this.vehiclelist = list;

}

public void insertData()

{

vehiclelist.add("Mercedes AMG G63");

vehiclelist.add("Land Rover Defender");

vehiclelist.add("Mercedes G600");

vehiclelist.add("Cadillac Escalade V");

vehiclelist.add("Volkswagen Virtus");

}

public List<String> getVehicleList()

{

return this.vehiclelist;

}

// Shallow Copy

public Object clone1() throws CloneNotSupportedException

{

return super.clone();

}

// Deep Copy

public Object clone2() throws CloneNotSupportedException

{

List<String> tempList = new ArrayList<String>();

for (String s: this.vehiclelist)

{

tempList.add(s);

}

return new Vehicle(tempList);

}

}

import java.util.\*;

public class PrototypePattern

{

public static void main(String[] args) throws CloneNotSupportedException

{

Vehicle a = new Vehicle();

a.insertData();

Vehicle b = (Vehicle) a.clone1();

System.out.println("For Shallow Copy:");

System.out.println("A: "+a.getVehicleList());

System.out.println("B: "+b.getVehicleList());

Vehicle c = (Vehicle) a.clone2();

System.out.println("For Deep Copy:");

System.out.println("A: "+a.getVehicleList());

System.out.println("B: "+c.getVehicleList());

List<String> list = c.getVehicleList();

list.add("Porsche Boxter");

System.out.println("B: "+c.getVehicleList());

c.getVehicleList().remove("Mercedes G600");

System.out.println("B: "+list);

System.out.println("A: "+a.getVehicleList());

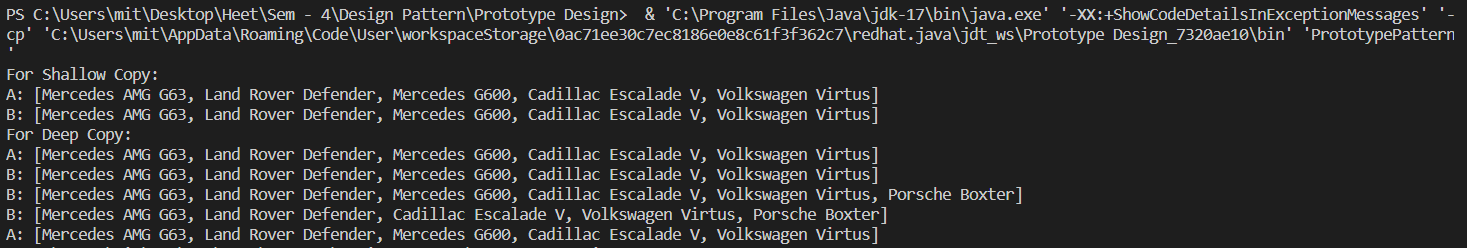
// b.modify();

}

}

// Deep Copy can copy nested objects also but Shallow Copy will not copy the nested objects.

* **Output :**

****

* **Advantages of Prototype Design Pattern :**

1) Efficient creation: Clone existing items to save time and

resources.

2) Flexibility: Enables dynamic runtime changes to object

structures.

1. Reuse promotes code reuse by utilising existing designs.

* **Disadvantages of Prototype Design Pattern :**

1) Lifecycle Management: Managing the lifecycle of cloned

objects, particularly those with nested structures, can be

challenging.

2) Initialization Challenges: It can be challenging to properly

initialise cloned objects, particularly ones with dependencies.

3) Unwanted Modifications: Because prototypes and clones share

structures, they might cause unwanted changes that must be

handled carefully.