**Abstract Design Pattern**

**Assignment - 1**

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

* **Abstract Design :**

The Abstract Factory design pattern is a creational pattern that provides an interface for creating families of related or dependent objects without specifying their concrete classes. It falls under the Gang of Four (GoF) design patterns and aims to abstract the process of object creation, making it possible to create families of related objects without specifying their concrete classes.

Key components of the Abstract Factory pattern include:

1. \*Abstract Factory Interface:\*

- Declares a set of methods that create the abstract product objects. These methods typically represent the various types of objects that can be created within a family.

2. \*Concrete Factories:\*

- Implement the Abstract Factory interface and are responsible for creating concrete product objects. Each concrete factory corresponds to a specific family of related products.

3. \*Abstract Product:\*

- Declares the interface for a type of product object created by the abstract factory. These products form a family of related objects.

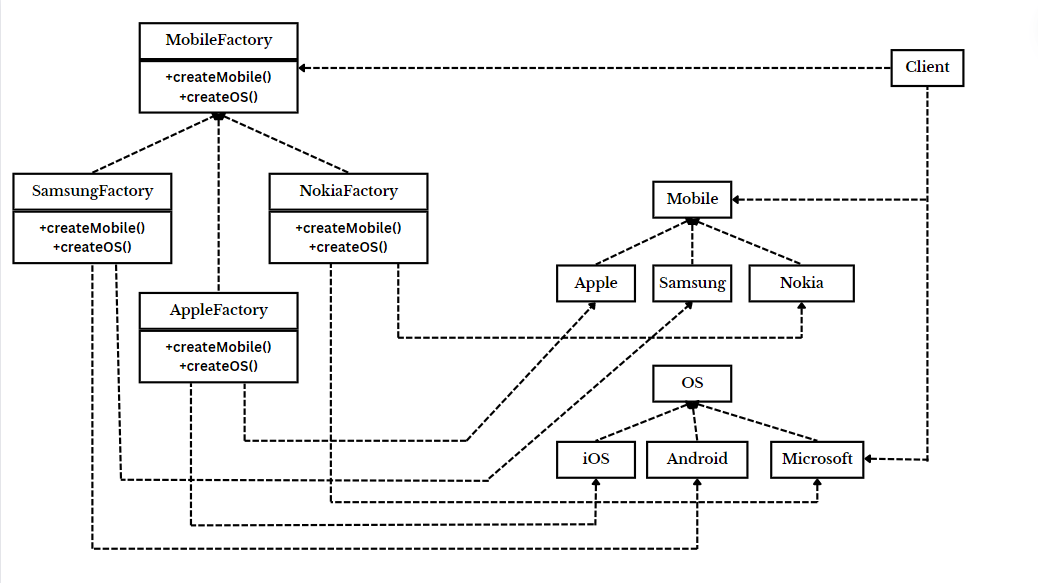
4. \*Concrete Products:\*

- Implement the Abstract Product interface and represent the actual product objects that are created by the concrete factories.

The Abstract Factory pattern allows a client to use an abstract interface to create families of related or dependent objects, making it easy to substitute different families of objects without altering the client code. This promotes flexibility and helps in designing systems that are independent of the way their objects are created, composed, and represented.

One common use case for the Abstract Factory pattern is in graphical user interface (GUI) libraries, where different platforms (such as Windows, macOS, or Linux) may have different implementations for buttons, windows, and other UI elements. The Abstract Factory pattern enables the creation of UI components that are specific to a particular platform, ensuring consistency within each family of related objects.

* **Program :** Create a Builder Design Pattern for Mobile Factory example.
* **UML Diagram :**



* **Code :**

// Abstract Product OS

interface OS

{

void displayOS();

}

// Concrete Products of OS: AndroidOS, WindowsOS, iOS

class AndroidOS implements OS

{

@Override

public void displayOS()

{

System.out.println("Powered by Android");

}

}

class WindowsOS implements OS

{

@Override

public void displayOS()

{

System.out.println("Powered by Windows");

}

}

class iOS implements OS

{

@Override

public void displayOS()

{

System.out.println("Powered by iOS");

}

}

// Abstract Product Mobile

interface Mobile

{

void displayInfo();

}

// Concrete Products of Mobile SamsungMobile, NokiaMobile, AppleMobile

class SamsungMobile implements Mobile

{

@Override

public void displayInfo()

{

System.out.println("This is Samsung Mobile");

}

}

class NokiaMobile implements Mobile

{

@Override

public void displayInfo()

{

System.out.println("This is Nokia Mobile");

}

}

class AppleMobile implements Mobile

{

@Override

public void displayInfo()

{

System.out.println("This is Apple Mobile");

}

}

// Abstract Factory MobileFactory

interface MobileFactory

{

Mobile createMobile();

OS createOS();

}

// Concrete Factories SamsungFactory, NokiaFactory, AppleFactory

class SamsungFactory implements MobileFactory

{

@Override

public Mobile createMobile()

{

return new SamsungMobile();

}

@Override

public OS createOS()

{

return new AndroidOS();

}

}

class NokiaFactory implements MobileFactory

{

@Override

public Mobile createMobile()

{

return new NokiaMobile();

}

@Override

public OS createOS()

{

return new WindowsOS();

}

}

class AppleFactory implements MobileFactory

{

@Override

public Mobile createMobile()

{

return new AppleMobile();

}

@Override

public OS createOS()

{

return new iOS();

}

}

// Client Code

public class AbstractFactoryExample

{

public static void main(String[] args)

{

MobileFactory samsungFactory = new SamsungFactory();

Mobile samsungMobile = samsungFactory.createMobile();

OS samsungOS = samsungFactory.createOS();

samsungMobile.displayInfo();

samsungOS.displayOS();

MobileFactory nokiaFactory = new NokiaFactory();

Mobile nokiaMobile = nokiaFactory.createMobile();

OS nokiaOS = nokiaFactory.createOS();

nokiaMobile.displayInfo();

nokiaOS.displayOS();

MobileFactory appleFactory = new AppleFactory();

Mobile appleMobile = appleFactory.createMobile();

OS appleOS = appleFactory.createOS();

appleMobile.displayInfo();

appleOS.displayOS();

}

}

* **Output :**

