**Facade Design Pattern**

**Assignment - 1**

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

* **Façade Design Pattern :**

The facade design pattern is a structural design pattern used in software development. It involves creating a simplified interface that hides the complexities of a system or a set of classes behind it. This interface provides a unified interface to a larger body of code, making it easier to use and understand. In other words, the facade pattern provides a single, simplified interface to a complex system, making it easier for clients to interact with.

The facade design pattern is commonly used in software development when there is a need to provide a simplified interface to a complex subsystem or set of classes. Some common usage scenarios include:

1. Complex API or library: When working with a complex API or library, developers may implement a facade to provide a simpler and more intuitive interface for common tasks, abstracting away the underlying complexity.

2. Legacy code integration: When integrating legacy code into a new system, a facade can be used to encapsulate the legacy code's functionality and provide a modern, streamlined interface for interacting with it.

3. Cross-platform development: Facades can be useful in cross-platform development to provide a unified interface for platform-specific features, allowing developers to write code that works across different platforms without having to deal with platform-specific intricacies directly.

Advantages of using the facade design pattern include:

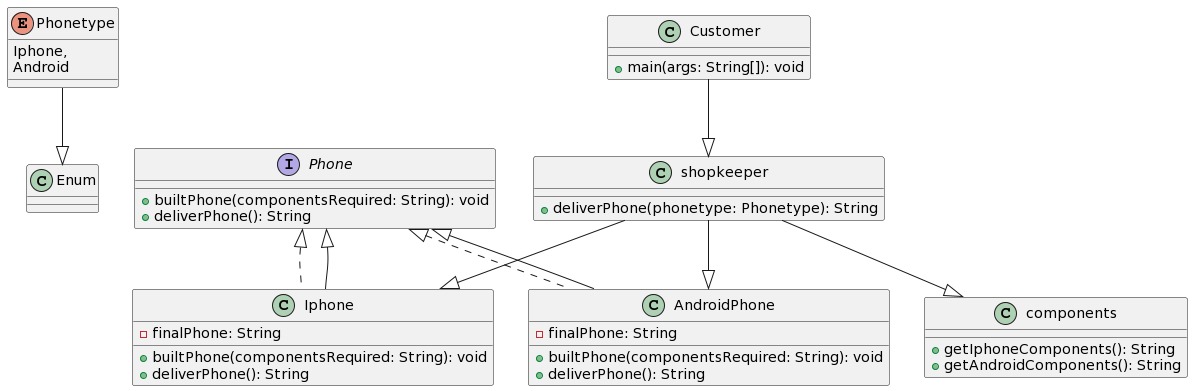
1. Simplified interface: Facades provide a simplified and easy-to-understand interface to a complex system, reducing the learning curve for developers who need to work with the system.

2. Encapsulation of complexity: By encapsulating the complex implementation details behind a facade, developers can hide the complexity of the underlying system and focus on using the provided interface.

3. Improved maintainability: Facades promote code maintainability by decoupling the client code from the implementation details of the subsystem. This makes it easier to make changes to the underlying system without affecting the clients that use it.

4. Enhanced flexibility: Facades can help improve flexibility by allowing developers to change the underlying implementation of a subsystem without affecting the clients that rely on its interface. This makes it easier to evolve the system over time as requirements change.

* **Program :** Implement facade design pattern for phone example.
* **UML Diagram :**



* **Code :**

public enum PhoneType{

iPhone,

Android

}

public interface Phone{

    public void builtPhone(String componentsrequired);

    public String deliverPhone();

}

public class Components {

    public String getiPhoneComponents(){

        return "IOS,3200,5.5";

    }

    public String getAndroidComponents(){

        return "Qualcom,2600,5";

    }

}

public class AndroidPhone implements Phone{

    public String finalPhone;

    @Override

    public void builtPhone(String componentsrequired){

        finalPhone = "Android phone with components: " + componentsrequired;

    }

    @Override

    public String deliverPhone(){

        return finalPhone;

    }

}

public class IPhone implements Phone{

    public String finalPhone;

    @Override

    public void builtPhone(String componentsrequired){

        finalPhone = "Apple phone with components: " + componentsrequired;

    }

    @Override

    public String deliverPhone(){

        return finalPhone;

    }

}

public class Shopkeeper {

    // facade which hides the implementation

    public static String deliverPhone(PhoneType phoneType){

        Components comp = new Components();

        switch(phoneType){

            case iPhone:

                Phone IPh=new IPhone();

                String phComp=comp.getiPhoneComponents();

                IPh.builtPhone(phComp);

                return IPh.deliverPhone();

            case Android:

                Phone Android=new AndroidPhone();

                String PhComp=comp.getAndroidComponents();

                Android.builtPhone(PhComp);

                return Android.deliverPhone();

        }

        return null;

    }

}

public class Customer {

    public static void main(String args[]){

        //Using facade

        System.out.println("--------Facade--------");

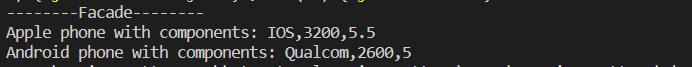
        System.out.println(Shopkeeper.deliverPhone(PhoneType.iPhone));

        System.out.println(Shopkeeper.deliverPhone(PhoneType.Android));

    }

}

* **Output :**

****