Lab Sheet: Implementing the Façade Design Pattern in Java

Objective: Understand and implement the Façade design pattern in Java with a real-world example.

Introduction: The Façade pattern provides a simplified interface to a set of interfaces in a subsystem, making it easier to use. It involves a single class, known as the Façade, which communicates with the subsystem's multiple components on behalf of the client.

Real-world Example: Consider a multimedia system with various components like AudioPlayer, VideoPlayer, and Projector. The Façade class will simplify the interactions for the client by providing a unified interface to control the multimedia system.

Step 1: Identify Subsystem Components

Identify the subsystem components that need to be encapsulated. In our example, these components are AudioPlayer, VideoPlayer, and Projector.

Step 2: Create Subsystem Classes

Create individual classes for each subsystem component. Use class names like **AudioPlayer**, **VideoPlayer**, and **Projector**. Implement methods in each class to perform specific tasks related to the component.

Step 3: Create the Façade Class

Create a Façade class, named **MultimediaFacade**. This class will provide a simple and unified interface for the client to interact with the subsystem components.

Step 4: Define Methods in Façade Class

Define methods in the **MultimediaFacade** class corresponding to the tasks that the client may perform. For example, methods like **playAudio()**, **playVideo()**, and **projectOnScreen()**.

Step 5: Implement Façade Methods

Inside each method of the **MultimediaFacade** class, call the relevant methods from the subsystem components (AudioPlayer, VideoPlayer, and Projector) to perform the required tasks.

Step 6: Client Code

Now, the client can use the **MultimediaFacade** class to interact with the multimedia system without dealing with the complexities of individual subsystem components.

Step 7: Testing

Write a simple client program to test the implementation. Instantiate the **MultimediaFacade** class and use its methods to control the multimedia system. Verify that the subsystem components work together seamlessly through the Façade.

Note to Students:

1. Follow the steps systematically to design and implement the Façade pattern.
2. Use appropriate access modifiers for methods in subsystem classes to encapsulate their functionality.
3. Ensure that the client code interacts only with the Façade class and not directly with the subsystem components.
4. Test your implementation thoroughly to confirm that the Façade simplifies the interaction with the subsystem.

Example Code Structure (Class and Method Names):

1. AudioPlayer class:
    - void turnOn()
    - void play(String audio)
    - void turnOff()
2. VideoPlayer class:
    - void turnOn()
    - void play(String video)
    - void turnOff()
3. Projector class:
    - void turnOn()
    - void projectOnScreen()
    - void turnOff()
4. MultimediaFacade class:
    - void playAudio(String audio)
    - void playVideo(String video)
    - void projectOnScreen()

Implementing the Adapter Design Pattern

Objective: Understand and implement the Adapter design pattern to make incompatible objects work together using the "Fitting square pegs into round holes" example.

Introduction: The Adapter pattern allows objects with incompatible interfaces to work together. In this lab, we'll adapt the SquarePeg class to fit into the RoundHole by using the SquarePegAdapter.

Step 1: Identify Incompatible Classes

Identify the incompatible classes, in this case, **SquarePeg** and **RoundHole**.

Step 2: Create the Adapter Class

Create an adapter class named **SquarePegAdapter**. This class should extend the **RoundPeg** class (the target interface) and contain an instance of the **SquarePeg** class.

Step 3: Implement Adapter Methods

Override the **getRadius** method in the **SquarePegAdapter** class to adapt the square peg's width to a compatible round peg radius.

Step 4: Update Client Code

Update the client code in the **Demo** class to demonstrate the use of the Adapter pattern. Comment out the lines that attempt to fit a square peg directly into a round hole, and replace them with instances of the **SquarePegAdapter**.

Step 5: Test the Adapter

Run the program and observe that square pegs can now fit into round holes using the Adapter.

Note to Students:

1. Pay attention to the existing classes (**RoundHole**, **RoundPeg**, **SquarePeg**, and **SquarePegAdapter**) and understand their roles.
2. Focus on how the Adapter (**SquarePegAdapter**) allows the integration of square pegs into the existing system designed for round pegs.
3. Test your implementation thoroughly to confirm that the Adapter pattern is working as expected.

Example Code Structure (Class and Method Names):

1. **SquarePegAdapter** class:
    - **SquarePegAdapter(SquarePeg peg)**
    - **@Override double getRadius()**
2. Updated **Demo** class:
    - Replace direct attempts to fit square pegs into round holes with instances of **SquarePegAdapter**.