17 Structural: Facade Pattern for a Home-Theater Controller

**Scenario** A consumer-electronics retailer ships a DIY home-theater bundle: DVD player, projector, surround sound, and smart lights. Customers complain that powering everything on/off in the right sequence is tedious.  
 Your job is to expose a **single, easy API** that hides this complexity while keeping individual device classes unchanged for future vendors.

#### **Tasks**

1 **Problem statement** (analysis/facade\_need.md)  
 *List three pain-points a user faces without a facade (sequence, volume preset, lights dimming). Explain how tight coupling would grow if the UI talked to every subsystem directly.*

2 **Subsystem implementation** (src/main/java/theater/)

* DVDPlayer, Projector, SoundSystem, Lights exactly as specified. Leave their APIs untouched.

3 **Facade** (HomeTheaterFacade.java)

* Constructor takes the four devices.
* startMovie(String title) carries out: lights dim→ projector on/input→ sound on/volume 20→ DVD on/play.
* endMovie() reverses the process.

4 **Client demo** (FacadeDemo.java)

* Instantiate devices, wrap in facade, play *Inception*, then shut down. Print console.

5 **Unit tests** (src/test/java/)

* StartSequenceTest – capture System.out; assert dimming occurs **before** projector on.
* ShutdownSequenceTest – ensure devices turn off in order and lights turn on last.

6 **Reflection** (reflection.md)

* How does Facade improve SRP for the UI layer?
* What happens if a new StreamingBox device is added? Outline changes.
* List one risk of hiding too much subsystem detail.

#### **Deliverables**

analysis/facade\_need.md

src/main/java/theater/DVDPlayer.java

src/main/java/theater/Projector.java

src/main/java/theater/SoundSystem.java

src/main/java/theater/Lights.java

src/main/java/theater/HomeTheaterFacade.java

src/main/java/theater/FacadeDemo.java

src/test/java/StartSequenceTest.java

src/test/java/ShutdownSequenceTest.java

reflection.md

README.md

## **Detailed Solution**

### **1 Subsystem classes**

// DVDPlayer.java

package theater;

public class DVDPlayer {

public void turnOn(){ System.out.println("DVD Player is turned on."); }

public void playMovie(String m){ System.out.println("Playing movie: "+m); }

public void turnOff(){ System.out.println("DVD Player is turned off."); }

}

// Projector.java, SoundSystem.java, Lights.java implemented as in brief

### **2 Facade**

// HomeTheaterFacade.java

package theater;

public class HomeTheaterFacade {

private final DVDPlayer dvd;

private final Projector projector;

private final SoundSystem sound;

private final Lights lights;

public HomeTheaterFacade(DVDPlayer d,Projector p,SoundSystem s,Lights l){

this.dvd=d; this.projector=p; this.sound=s; this.lights=l;

}

public void startMovie(String title){

System.out.println("\nSetting up the home theater system...");

lights.dim(10);

projector.turnOn();

projector.setInput("DVD Player");

sound.turnOn(); sound.setVolume(20);

dvd.turnOn(); dvd.playMovie(title);

System.out.println("Enjoy the movie!");

}

public void endMovie(){

System.out.println("\nShutting down the home theater system...");

dvd.turnOff();

projector.turnOff();

sound.turnOff();

lights.turnOn();

System.out.println("Goodbye!");

}

}

### **3 Client**

// FacadeDemo.java

package theater;

public class FacadeDemo {

public static void main(String[] args){

HomeTheaterFacade ht = new HomeTheaterFacade(

new DVDPlayer(), new Projector(), new SoundSystem(), new Lights());

ht.startMovie("Inception");

ht.endMovie();

}

}

*Output* matches the brief.

### **4 Test snippet**

/\* StartSequenceTest \*/

String out = capture(() -> {

HomeTheaterFacade ht = makeFacade();

ht.startMovie("Matrix");

});

assertTrue(out.indexOf("Lights dimmed") < out.indexOf("Projector is turned on"));

*capture helper redirects System.out and returns the string.*

### **Reflection (excerpt)**

• UI interacts with \*one\* façade → SRP satisfied; devices evolve independently.

• Adding StreamingBox: implement class, extend facade methods; no change to UI.

• Risk: if advanced users need per-device control, façade may over-simplify; expose

subsystem handles via getters or provide an “advanced” API.

The **Facade Pattern** here hides a multi-step orchestration behind two simple calls, drastically reducing client complexity while preserving subsystem autonomy.