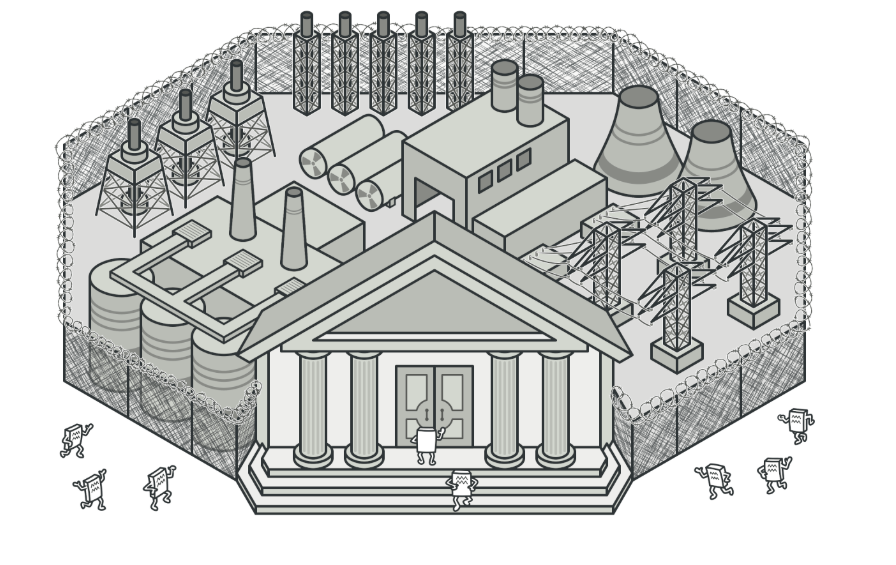
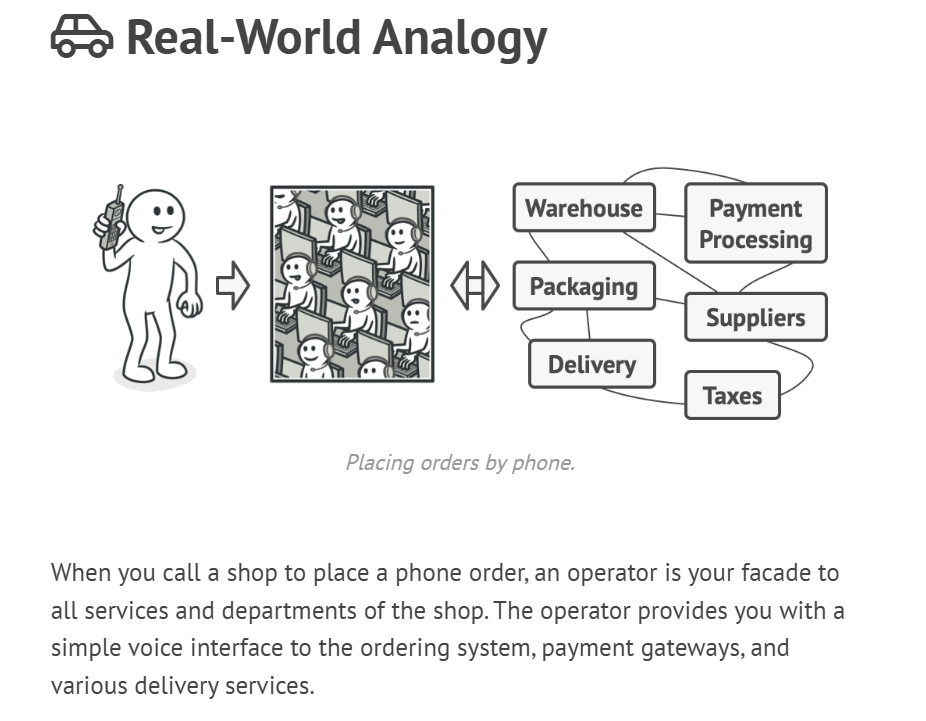
**Facade** is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes.





**🛠 Example: Home Theater System 🎥**

**Subsystems (Complex Components)**

class Projector {

void turnOn() { System.out.println("Projector is ON"); }

void turnOff() { System.out.println("Projector is OFF"); }

}

class SoundSystem {

void setVolume(int level) { System.out.println("Sound volume set to " + level); }

}

class BluRayPlayer {

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

}

**Facade Class (Simplifies the System)**

class HomeTheaterFacade {

private Projector projector;

private SoundSystem sound;

private BluRayPlayer bluRay;

public HomeTheaterFacade() {

this.projector = new Projector();

this.sound = new SoundSystem();

this.bluRay = new BluRayPlayer();

}

public void watchMovie(String movie) {

System.out.println("\n🎬 Starting Movie Night!");

projector.turnOn();

sound.setVolume(10);

bluRay.playMovie(movie);

}

public void endMovie() {

System.out.println("\n📴 Shutting down Home Theater...");

projector.turnOff();

}

}

**Client Code (Using the Facade)**

public class FacadePatternDemo {

public static void main(String[] args) {

HomeTheaterFacade homeTheater = new HomeTheaterFacade();

homeTheater.watchMovie("Inception");

homeTheater.endMovie();

}

}

**🎯 Benefits of Facade Pattern**

✅ **Reduces Complexity** → Hides internal workings.  
✅ **Loose Coupling** → Clients depend only on Facade, not subsystems.  
✅ **Improves Maintainability** → Changing internals **doesn’t affect the client**.

**❌ Without Facade (Bad Practice)**

Imagine the client directly controlling **Projector, Sound, and Blu-ray Player**:

Projector p = new Projector();

SoundSystem s = new SoundSystem();

BluRayPlayer b = new BluRayPlayer();

p.turnOn();

s.setVolume(10);

b.playMovie("Inception");

🔴 **Problem:** More dependencies, more maintenance headaches!