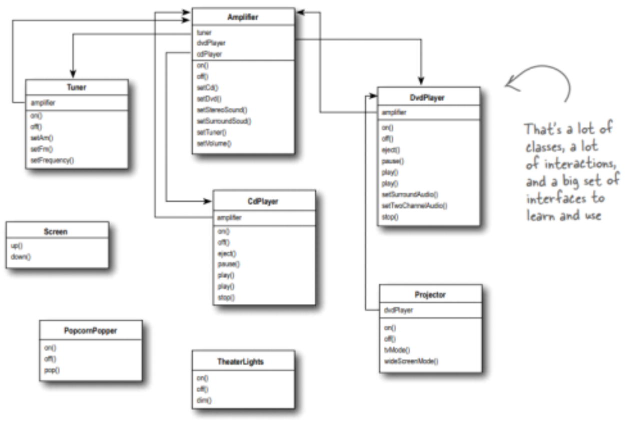
FAÇADE DESIGN PATTERN

Tuesday, September 12, 2023 2:28 PM

when do we need it?

when we need to do as little as possible to do a set of task.

Ex: Turning on a Home theatre



Watching a movie (the hard way)

Pick out a DVD, relax, and get ready for movie magic. Oh, there's just one thing – to watch the movie, you need to perform a few tasks:

1. Turn on the popcorn popper
2. Start the popper popping
3. Dim the lights
4. Put the screen down
5. Turn the projector on
6. Set the projector input to DVD
7. Put the projector on wide-screen mode
8. Turn the sound amplifier on
9. Set the amplifier to DVD input
10. Set the amplifier to surround sound
11. Set the amplifier volume to medium (5)
12. Turn the DVD Player on
13. Start the DVD Player playing

I'm already exhausted and all I've done is turn everything on!

Let's check out those same tasks in terms of the classes and the method calls needed to perform them:

```
popper.on();
popper.pop();

lights.dim(10);

screen.down();

projector.on();
projector.setInput(dvd);
projector.wideScreenMode();

amp.on();
amp.setDvd(dvd);
amp.setSurroundSound();
amp.setVolume(5);

dvd.on();
dvd.play(movie);
```

Six different classes involved!

Turn on the popcorn popper and start popping...

Dim the lights to 10%...

Put the screen down...

Turn on the projector and put it in wide screen mode for the movie...

Turn on the amp, set it to DVD, put it in surround sound mode and set the volume to 5...

Turn on the DVD player... and FINALLY, play the movie!

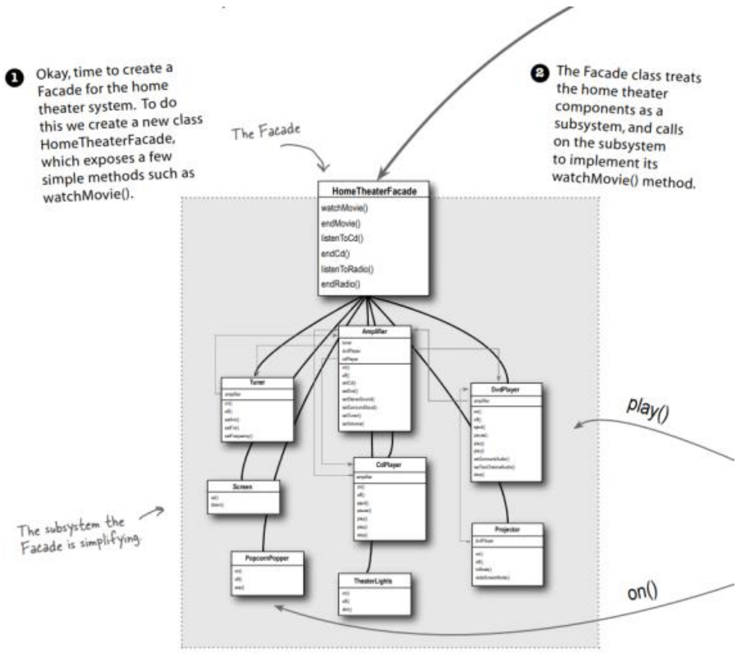
All this Task need to be done so as just to start watching :)

But there's more...

- When the movie is over, how do you turn everything off? Wouldn't you have to do all of this over again, in reverse?
- Wouldn't it be as complex to listen to a CD or the radio?
- If you decide to upgrade your system, you're probably going to have to learn a slightly different procedure.

what about stopping? & other stuff for that you need to write implementation again.

Easier way using facade design pattern.



1 Okay, time to create a Facade for the home theater system. To do this we create a new class HomeTheaterFacade, which exposes a few simple methods such as watchMovie().

2 The Facade class treats the home theater components as a subsystem, and calls on the subsystem to implement its watchMovie() method.

```
public class HomeTheaterFacade {
 Amplifier amp;
 Tuner tuner;
 DvdPlayer dvd;
 CdPlayer cd;
 Projector projector;
 TheaterLights lights;
 Screen screen;
 PopcornPopper popper;
 public HomeTheaterFacade(Amplifier amp,
 Tuner tuner,
 DvdPlayer dvd,
 CdPlayer cd,
 Projector projector,
 Screen screen,
 TheaterLights lights,
 PopcornPopper popper) {
 this.amp = amp;
 this.tuner = tuner;
 this.dvd = dvd;
 this.cd = cd;
 this.projector = projector;
 this.screen = screen;
 this.lights = lights;
 this.popper = popper;
 }
 // other methods here

public void watchMovie(String movie) {
 System.out.println("Get ready to watch a movie...");
 popper.on();
 popper.pop();
 lights.dim(10);
 screen.down();
 projector.on();
 projector.wideScreenMode();
 amp.on();
 amp.setDvd(dvd);
 amp.setSurroundSound();
 amp.setVolume(5);
 dvd.on();
 dvd.play(movie);
}
public void endMovie() {
 System.out.println("Shutting movie theater down...");
 popper.off();
 lights.on();
 screen.up();
 projector.off();
 amp.off();
 dvd.stop();
 dvd.eject();
 dvd.off();
}
}
```

```
public class HomeTheaterTestDrive {
    public static void main(String[] args) {
        // instantiate components here

        HomeTheaterFacade homeTheater =
                new HomeTheaterFacade(amp, tuner, dvd, cd,
                        projector, screen, lights, popper);

        homeTheater.watchMovie("Raiders of the Lost Ark");
        homeTheater.endMovie();
    }
}
```

Here we're creating the components right in the test drive. Normally the client is given a facade, it doesn't have to construct one itself.

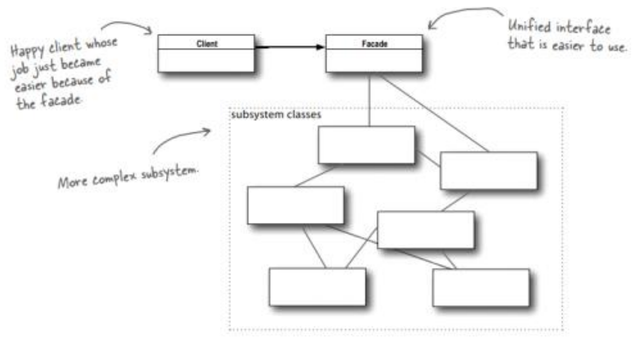
First you instantiate the Facade with all the components in the subsystem.

Use the simplified interface to first start the movie up, and then shut it down.

CONSOLE OUTPUT:

```
%java HomeTheaterTestDrive
Get ready to watch a movie...
Popcorn Popper on
Popcorn Popper popping popcorn!
Theater Ceiling Lights dimming to 10%
Theater Screen going down
Top-O-Line Projector on
Top-O-Line Projector in widescreen mode (16x9 aspect ratio)
Top-O-Line Amplifier on
Top-O-Line Amplifier setting DVD player to Top-O-Line DVD Player
Top-O-Line Amplifier surround sound on (5 speakers, 1 subwoofer)
Top-O-Line Amplifier setting volume to 5
Top-O-Line DVD Player on
Top-O-Line DVD Player playing "Raiders of the Lost Ark"
Shutting movie theater down...
Popcorn Popper off
Theater Ceiling Lights on
Theater Screen going up
Top-O-Line Projector off
Top-O-Line Amplifier off
Top-O-Line DVD Player stopped "Raiders of the Lost Ark"
Top-O-Line DVD Player eject
Top-O-Line DVD Player off
%
```

The Facade Pattern provides a unified interface to a set of interfaces in a subsytem. Facade defines a higher-level interface that makes the subsystem easier to use.

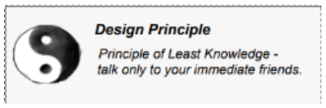

Happy client whose job just became easier because of the facade.
Client
Facade
Unified interface that is easier to use.
subsystem classes
More complex subsystem.

The Principle of Least Knowledge → states call only immediate friends.

allowed in facade design pattern.

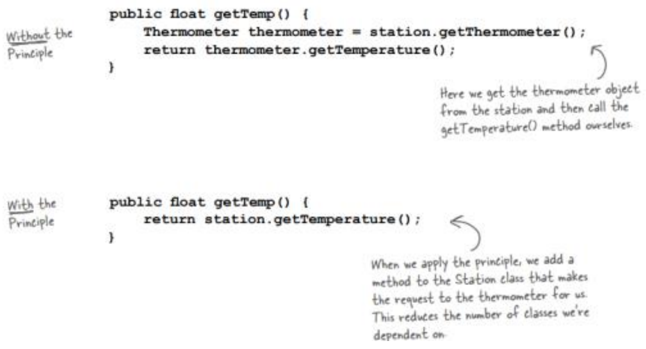
The Principle of Least Knowledge → ([illegible]

→ To be followed in facade design pattern.



Design Principle

Principle of Least Knowledge - talk only to your immediate friends.



Without the Principle

```
public float getTemp() {
    Thermometer thermometer = station.getThermometer();
    return thermometer.getTemperature();
}
```

Here we get the thermometer object from the station and then call the getTemperature() method ourselves.

With the Principle

```
public float getTemp() {
    return station.getTemperature();
}
```

When we apply the principle, we add a method to the Station class that makes the request to the thermometer for us. This reduces the number of classes we're dependent on.

Youtube channel : Shubham Harikesh