FACADE

Chandan R. Rupakheti

Week 5-2

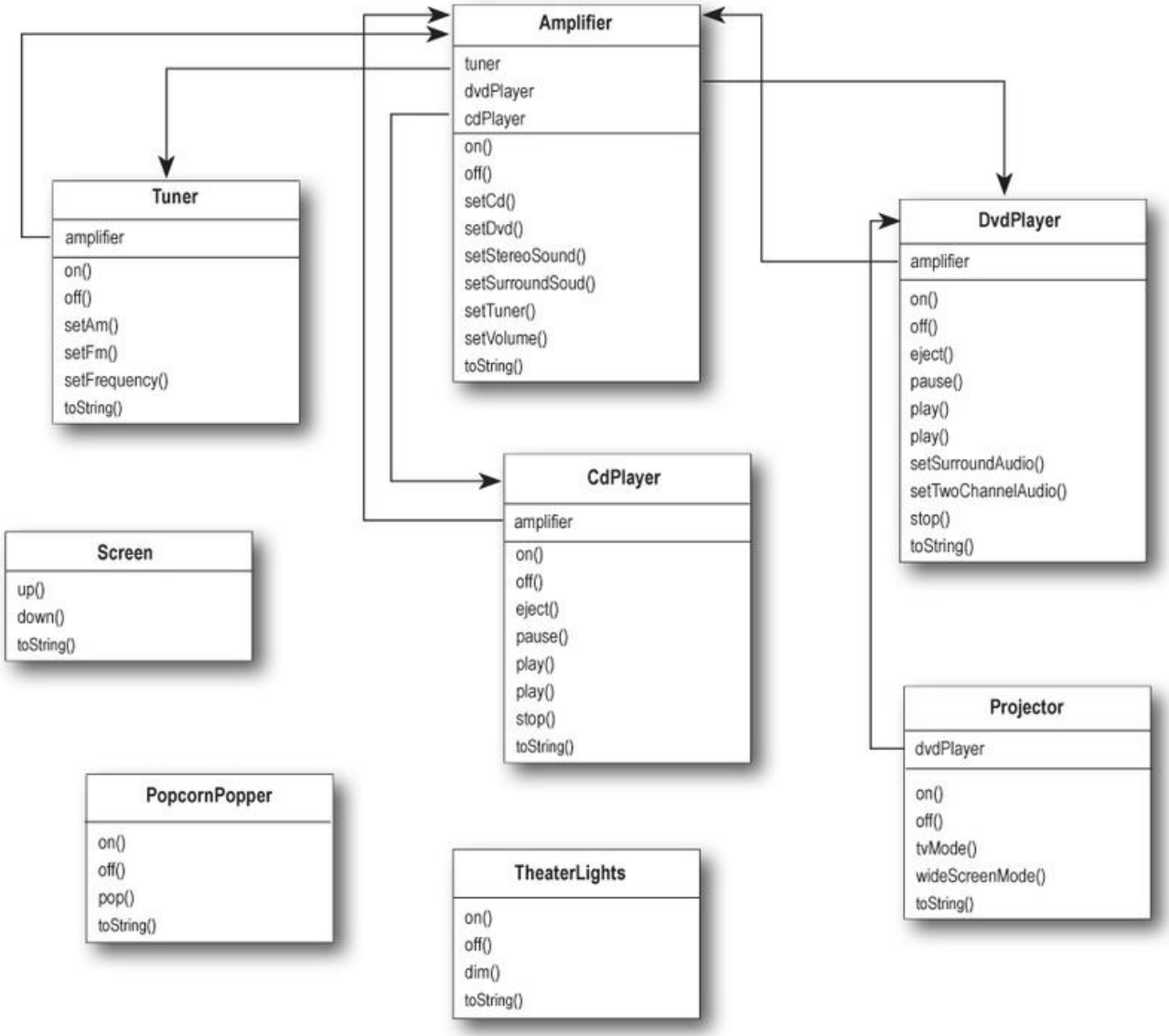
Today

We're going to look at a pattern that alters an interface to hide all the complexity of one or more classes behind a clean, well-lit façade

Understand why we software professionals are anti-social ☺

But first, let's prepare for the client meeting …

Home Sweet Home Theater



That's a lot of classes, a lot of interactions, and a big set of interfaces to learn and use.

You've spent weeks running wire, mounting the projector, making all the connections, and fine tuning. Now it's time to put it all in motion and enjoy a movie...

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**

Same tasks in classes/methods

Six different classes involved!

```
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);
```

Turn on the popcorn popper and start popping...

Dim the lights to 10%...

Put the screen down...

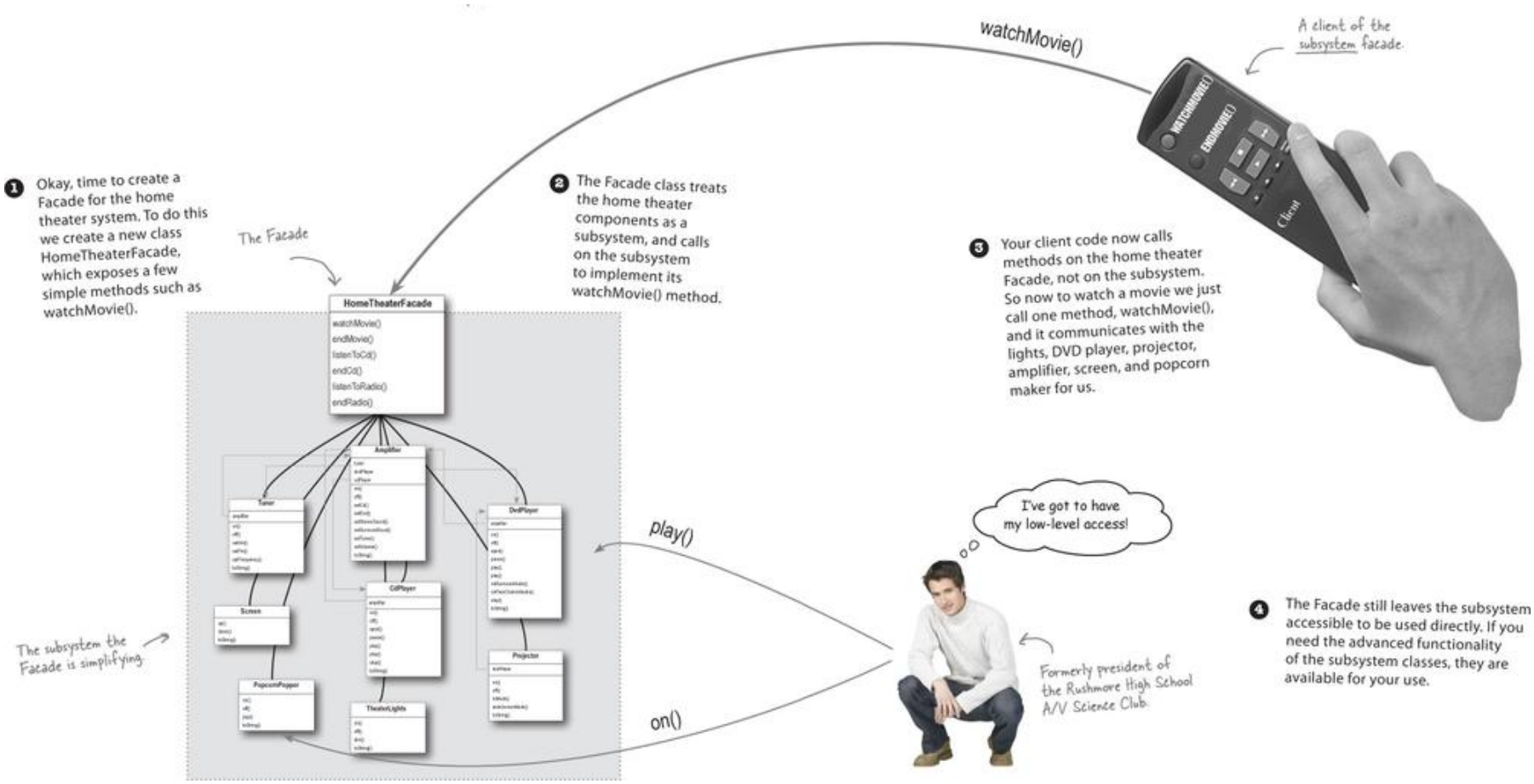
Turn on the projector and put it in wide screen mode for the movie...

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!

When the movie is over, how do you turn everything off?

Lights, Camera, Facade!



Constructing your home theater façade

```
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

}
```

Here's the composition; these are all the components of the subsystem we are going to use.

The facade is passed a reference to each component of the subsystem in its constructor. The facade then assigns each to the corresponding instance variable.

We're just about to fill these in...

Implementing the simplified interface

```
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);
}
```

watchMovie() follows the same sequence we had to do by hand before, but wraps it up in a handy method that does all the work. Notice that for each task we are delegating the responsibility to the corresponding component in the subsystem.

```
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();
}
```

And endMovie() takes care of shutting everything down for us. Again, each task is delegated to the appropriate component in the subsystem.

Time to watch a movie, piece of cake!

```
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();
    }

}
```

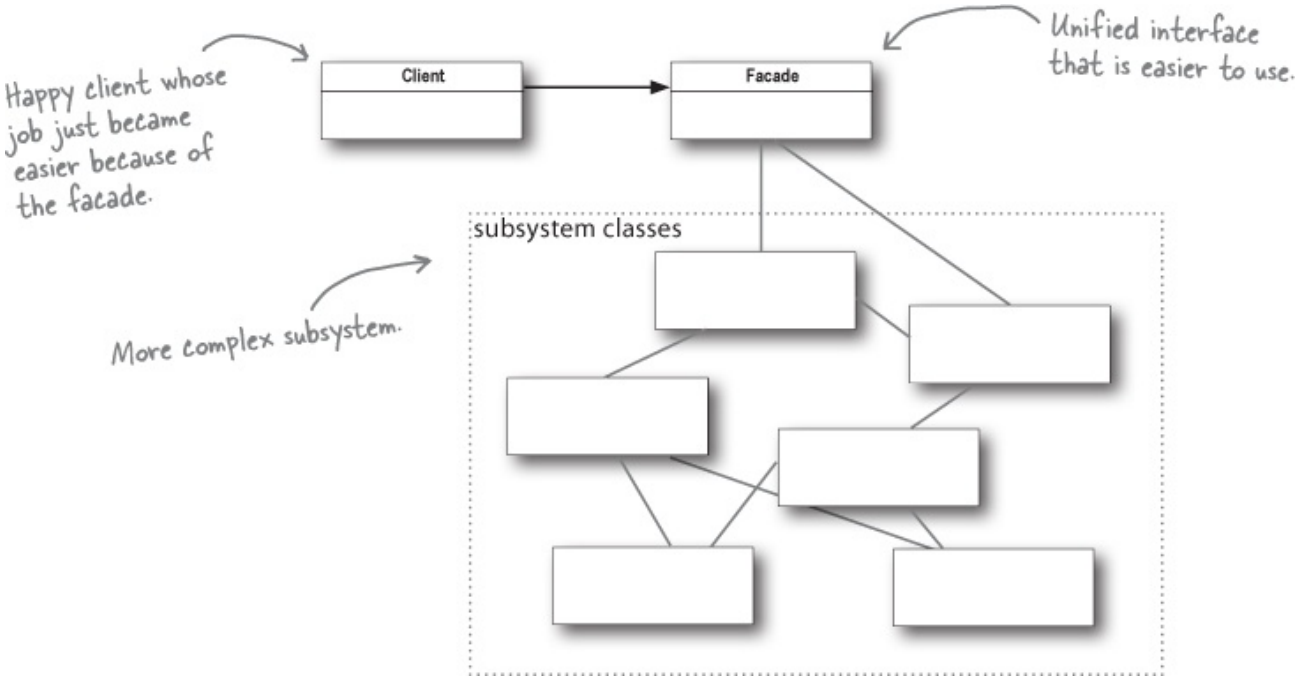
Here we're creating the components right in the test drive. Normally the client is given a facade; it doesn't have to construct one itself.

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.

The Façade Pattern defined

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



The Principle of Least Knowledge (PLK)

Talk only to your immediate friends

The principle tells us that we should only invoke methods that belong to:

- The object itself
- Objects passed in as a parameter to the method
- Any object the method creates or instantiates

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.

Keeping your methods calls in bounds …

```
public class Car {
        Engine engine;
        // other instance variables

        public Car() {
                // initialize engine, etc.
        }

        public void start(Key key) {
                Doors doors = new Doors();
                boolean authorized = key.turns();
                if (authorized) {
                        engine.start();
                        updateDashboardDisplay();
                        doors.lock();
                }
        }

        public void updateDashboardDisplay() {
                // update display
        }
}
```

Here's a component of this class. We can call its methods.

Here we're creating a new object; its methods are legal.

You can call a method on an object passed as a parameter.

You can call a method on a component of the object.

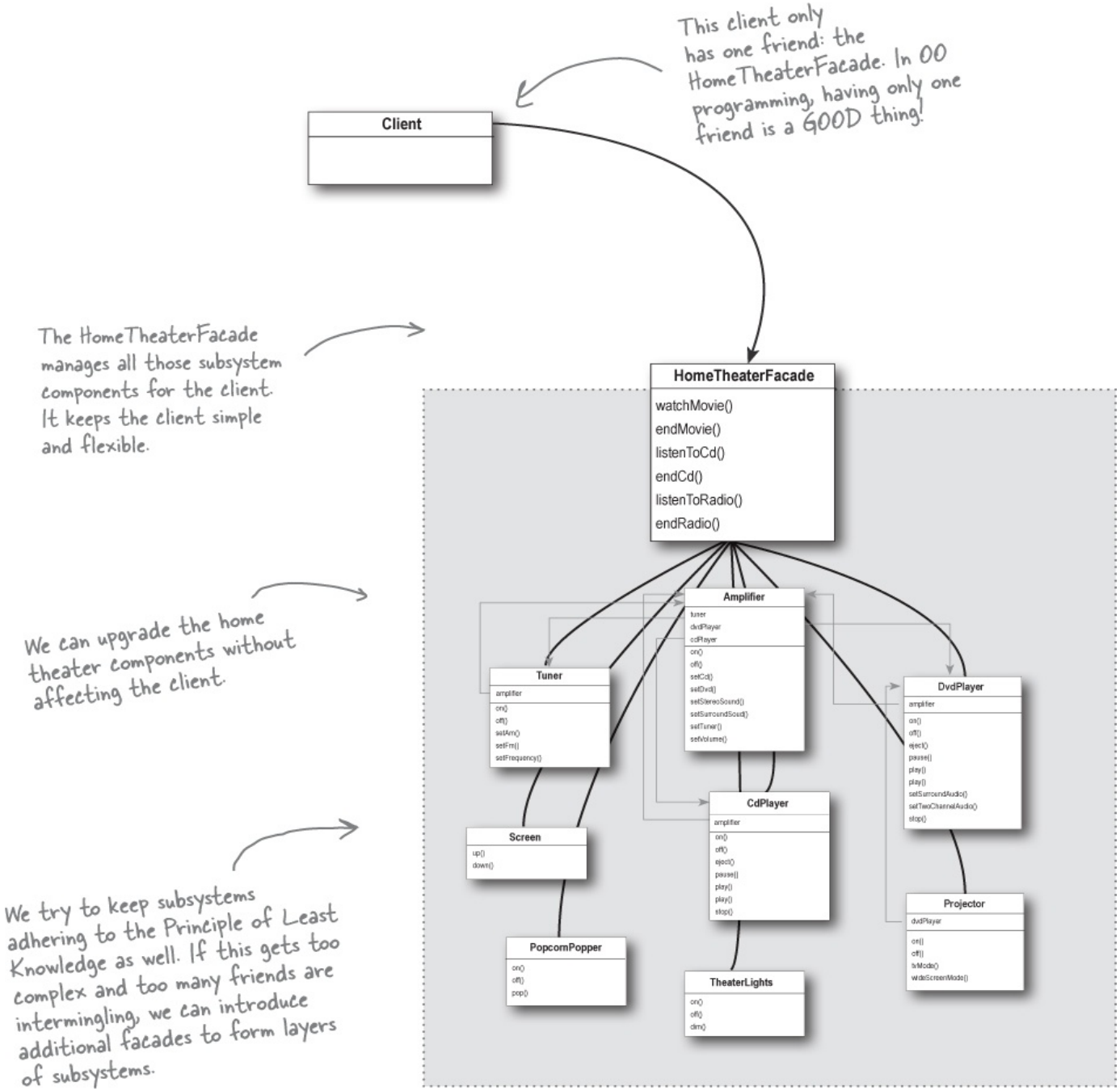
You can call a local method within the object.

You can call a method on an object you create or instantiate.

Do these classes violate PLK?

```
public House {
    WeatherStation station;

    // other methods and constructor

    public float getTemp() {
         return station.getThermometer().getTemperature();
    }
}
public House {
     WeatherStation station;

    // other methods and constructor

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

    public float getTempHelper(Thermometer thermometer) {
        return thermometer.getTemperature();
    }
}
```

Façade and PLK



Recap

A facade decouples a client from a complex subsystem.

When you need to simplify and unify a large interface or complex set of interfaces, use a facade.

Be anti-social, only talk to your immediate friends!