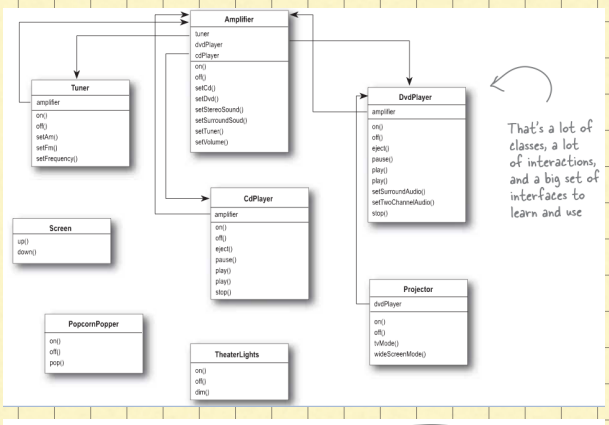
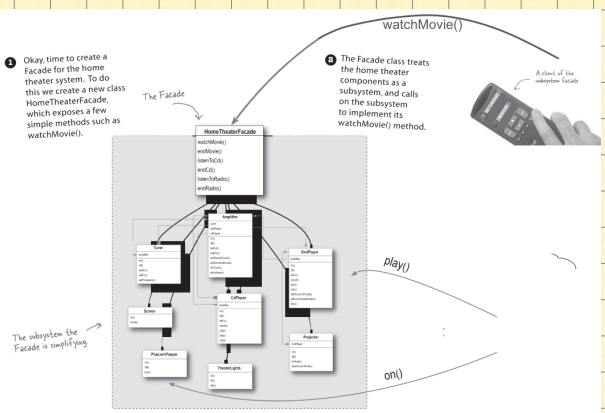
A Home Theater





Sidebar: Façade vs. Adapter

- A façade not only simplifies an interface, but it also decouples a client from a subsystem of components
- Facades and adapters may wrap multiple classes
  - A façade's intent is to **simplify**
  - An adapter's intent is to **convert** the interface into something different
- A façade does not encapsulate; it just provides a simplified interface

The 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...

```
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();
}
```

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.

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

The Façade Pattern

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

A Sidebar: The Principle of Least Knowledge

- As a general design principle, you should reduce the interactions between objects to just a few "close friends"
  - Be careful of the number of classes an object interacts with and also how it comes to interact with those classes
  - Prevents creating designs that have a very high degree of coupling among classes (these systems are much more fragile)
- General guidelines:
  - Only invoke methods that belong to
    - The object itself
    - Objects passed as parameters to the method
    - Any object the method creates or instantiates
    - Any components of the object
  - Do **not** invoke methods on objects that were returned from calling other methods!

Example

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.

Principle of Least Knowledge

- a.k.a. Law of Demeter
- Advantages
  - Reduces dependencies between objects, which has been demonstrated to reduce maintenance costs
- Disadvantages
  - Results in more "wrapper" classes to avoid method calls to other components
  - This can result in increased complexity and development time

Examples of Good Practice

```
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.

Façade and the Principle of Least Knowledge