COMP2511

Command and Facade Patterns

Prepared by

Dr. Ashesh Mahidadia

Design Patterns

Creational Patterns

- Factory Method
- Abstract Factory
- Builder
- Singleton

Structural Patterns

- Adapter
- Composite
- Decorator
- Façade

Behavioral Patterns

- Iterator
- Observer
- State
- Strategy
- Template
- Visitor
- Command Pattern

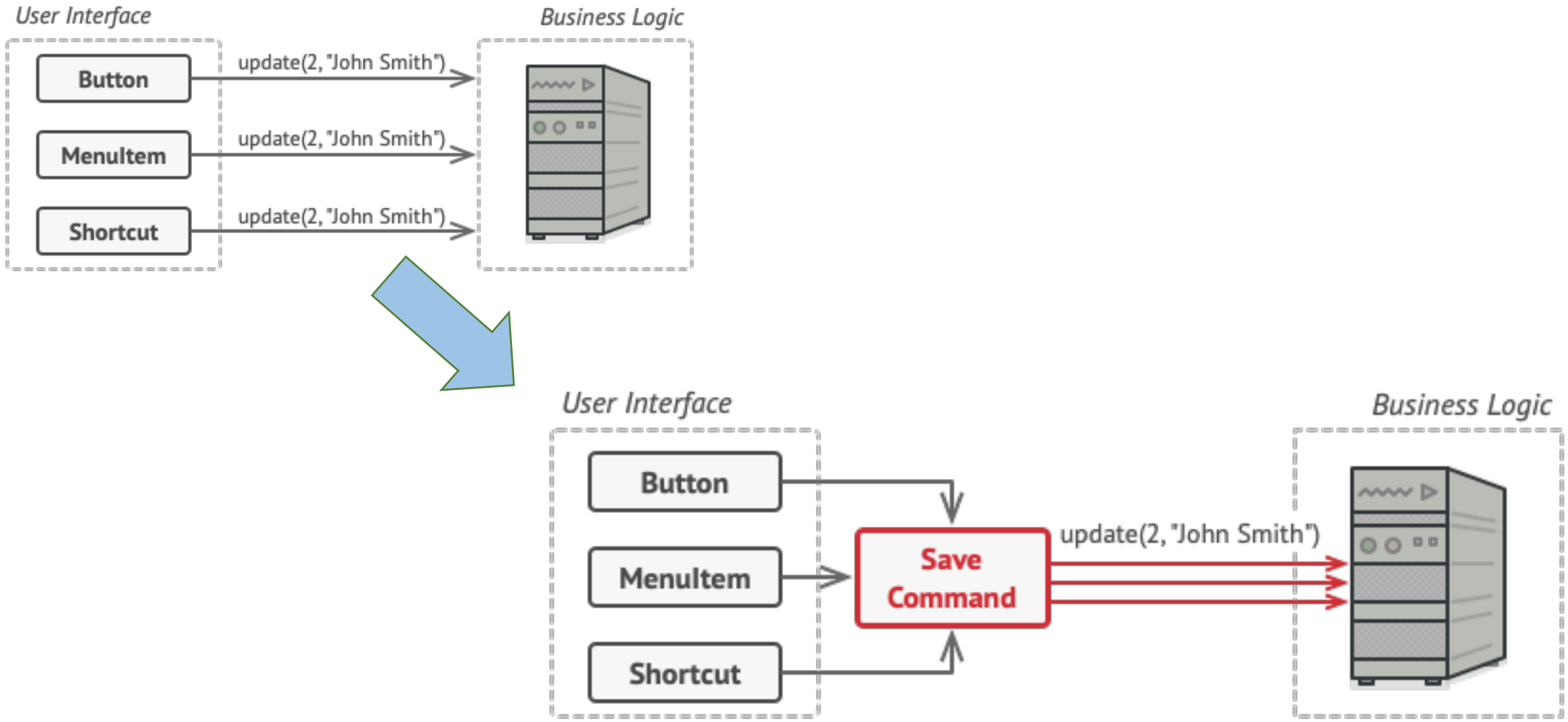
The lecture slides use material from the websites
https://refactoring.guru/design-patterns/
and the Head First Design Patterns reference book.

Command Pattern

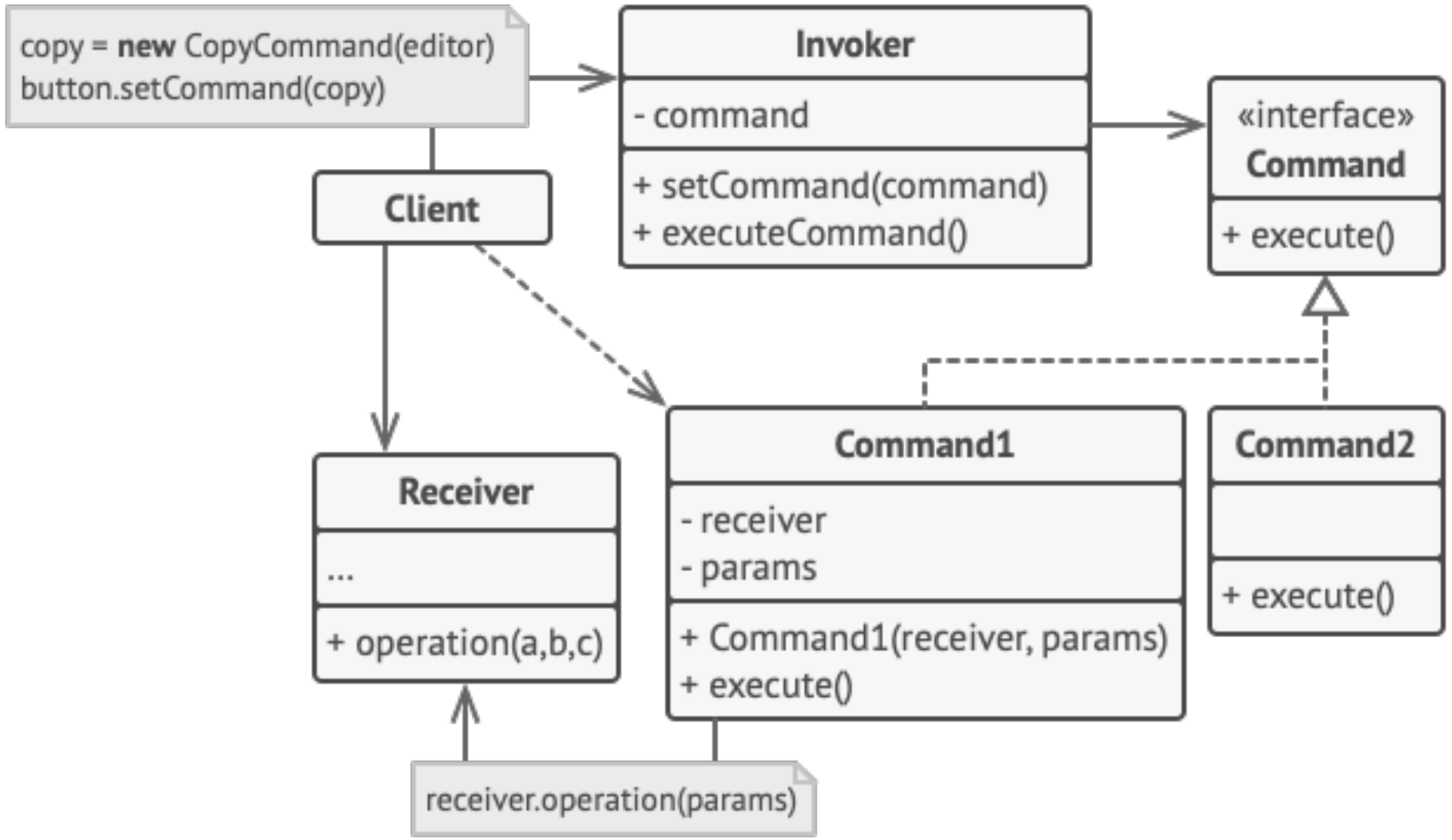
Command Pattern

- The Command Pattern allows you to decouple the requester of an action from the object that actually performs the action.
- A command object encapsulates a request (i.e., turn on light) on a specific object (say, the living room light object).
- A command object is associated with an invoker (say a button).
- An invoker executes a predefined method on a command object, that in turn performs actions as per the associated request.
- An invoker (say a button) is decoupled from the original request (turn on light).
- We can easily change / substitute a command object, resulting in a different action.
- Command pattern is a behavioral pattern, it transforms a request into an object, allowing it to be passed as method arguments, serialized it, log it, queue it for delayed execution, etc.

Command Pattern



Command Pattern



Command Pattern: Remote Control Example

```java
public interface Command {

    public void execute();

}
```

```java
public class LightOffCommand implements Command {
    Light light;

    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.off();
    }
}
```

```java
public class LightOnCommand implements Command {
    Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.on();
    }
}
```

```java
public class Light {
    String location = "";

    public Light(String location) {
        this.location = location;
    }

    public void on() {
        System.out.println(location + " light is on");
    }

    public void off() {
        System.out.println(location + " light is off");
    }
}
```

```java
public class StereoOnWithCDCommand implements Command {
    Stereo stereo;

    public StereoOnWithCDCommand(Stereo stereo) {
        this.stereo = stereo;
    }

    public void execute() {
        stereo.on();
        stereo.setCD();
        stereo.setVolume(11);
    }
}
```

Command Pattern: Remote Control Example

```
public class RemoteControl {
    // This is the invoker

    Command[] onCommands;
    Command[] offCommands;

    public RemoteControl() {
        onCommands = new Command[7];
        offCommands = new Command[7];

        Command noCommand = new NoCommand();
        for (int i = 0; i < 7; i++) {
            onCommands[i] = noCommand;
            offCommands[i] = noCommand;
        }
    }

    public void setCommand(int slot, Command onCommand, Command offCommand) {
        onCommands[slot] = onCommand;
        offCommands[slot] = offCommand;
    }

    public void onButtonWasPushed(int slot) {
        onCommands[slot].execute();
    }

    public void offButtonWasPushed(int slot) {
        offCommands[slot].execute();
    }
```

Command Pattern: Remote Control Example

Demo …..

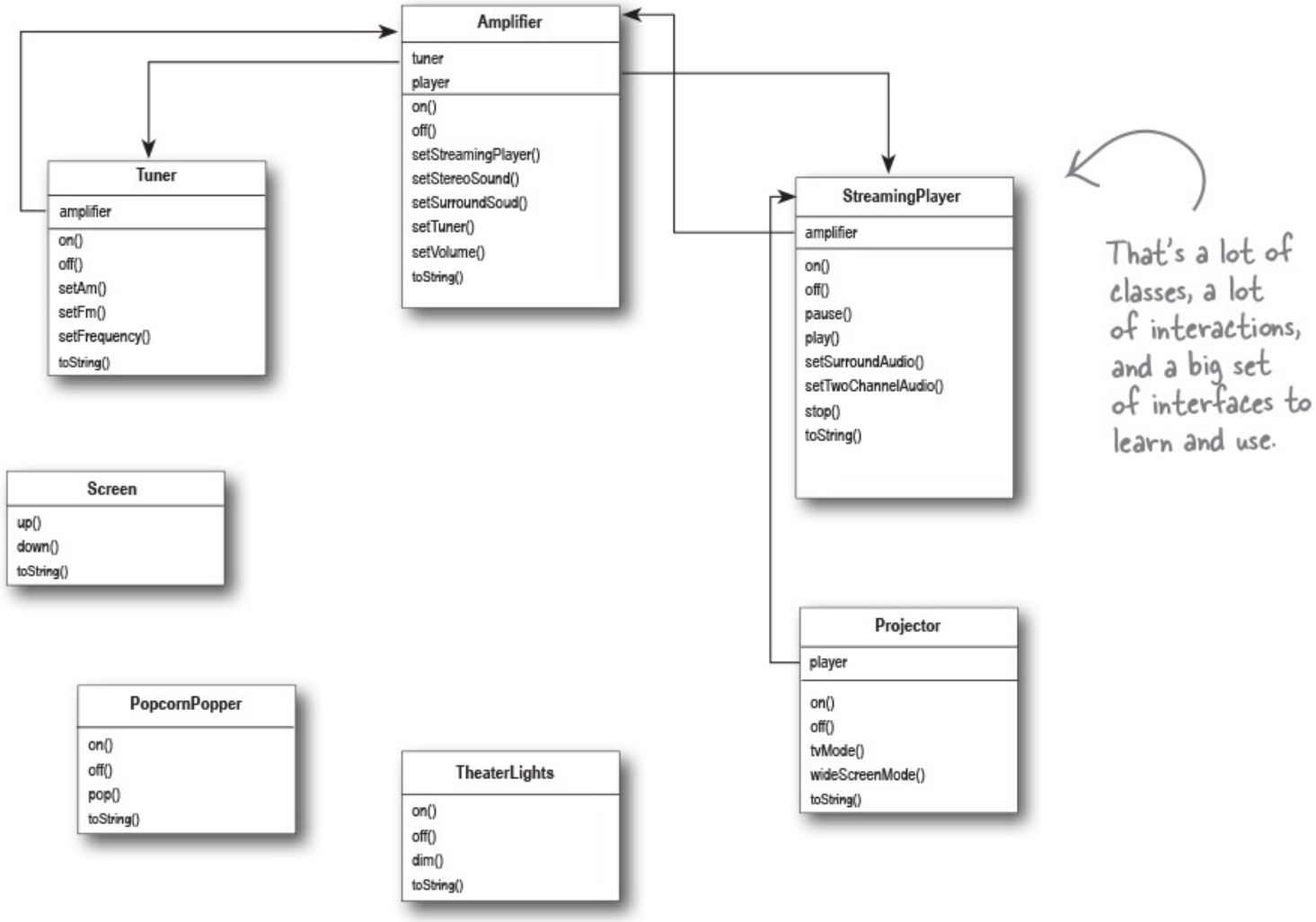
Façade Pattern

- Façade offers a simplified interface (façade) to hide all the complexity of one or more classes .
- Adapter Vs Façade Patterns:
  - *Adapter Pattern*: Converts one interface to another (one a client is expecting)
  - *Façade Pattern*: Makes an interface simpler to a *complex* class/classes (subsystem)
- Facades offers a simplified interface to the underlying class/classes.
- Importantly, facades do NOT "*encapsulate*" the subsystem classes.
- The underlying subsystem classes and their methods are still available for direct use by clients. For example, in the *Home Theatre* example, methods of a projector, amplifier, etc.

Example: Home Theatre

To watch a movie, you need to perform a few tasks:

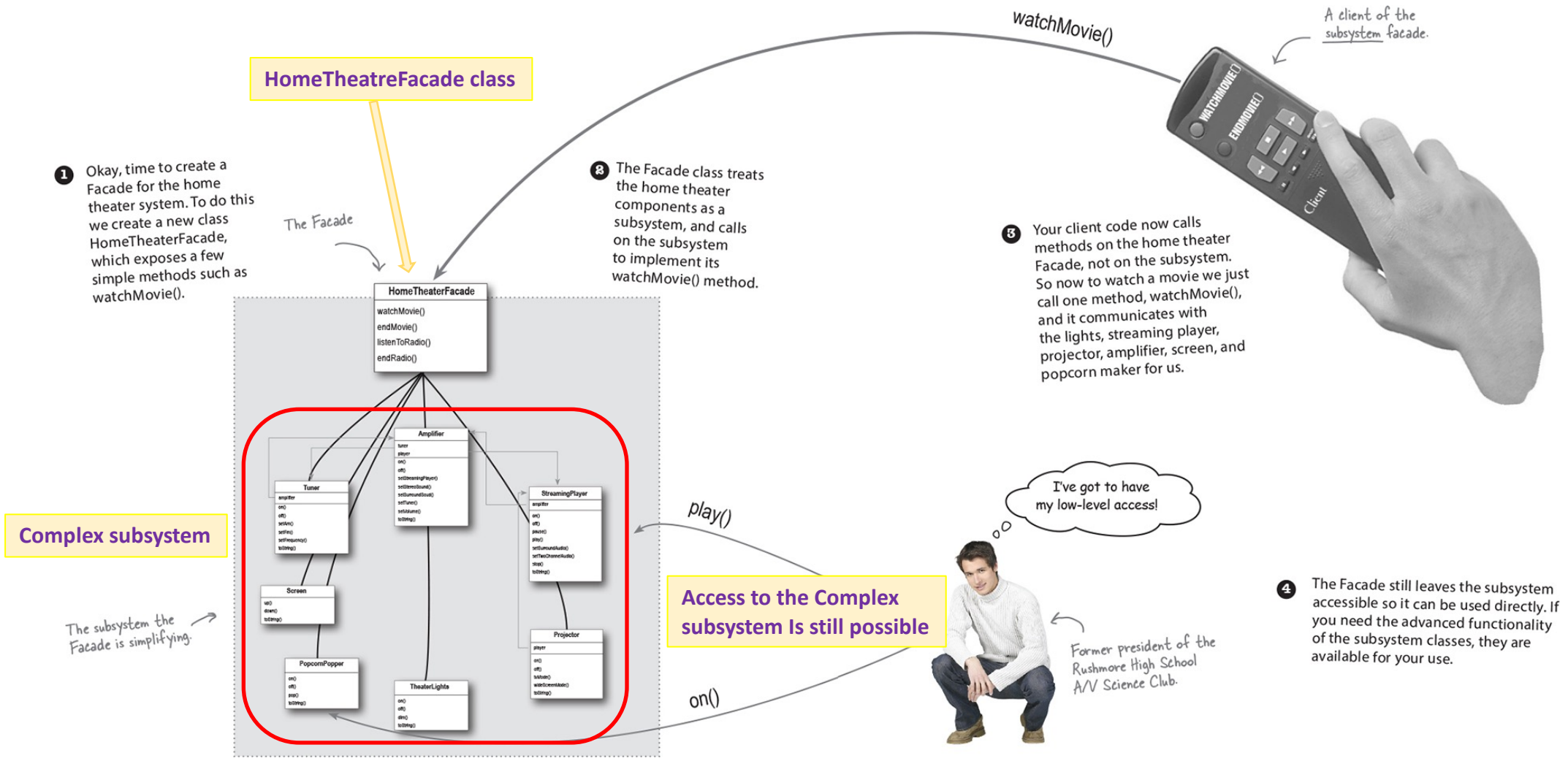
- Turn on the popcorn popper
- Start the popper popping
- Dim the lights
- Put the screen down
- Turn the projector on
- Set the projector input to streaming player
- Put the projector on widescreen mode
- Turn the sound amplifier on
- Set the amplifier to streaming player input
- Set the amplifier to surround sound
- Set the amplifier volume to medium (5)
- Turn the streaming player on
- Start playing the movie



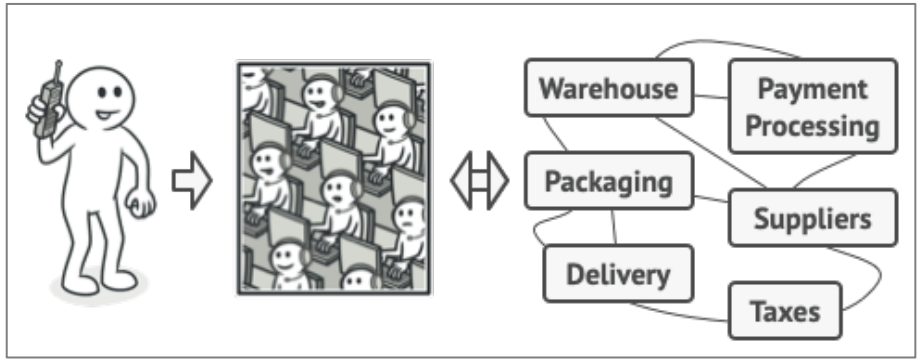
Lot of interfaces to deal with!

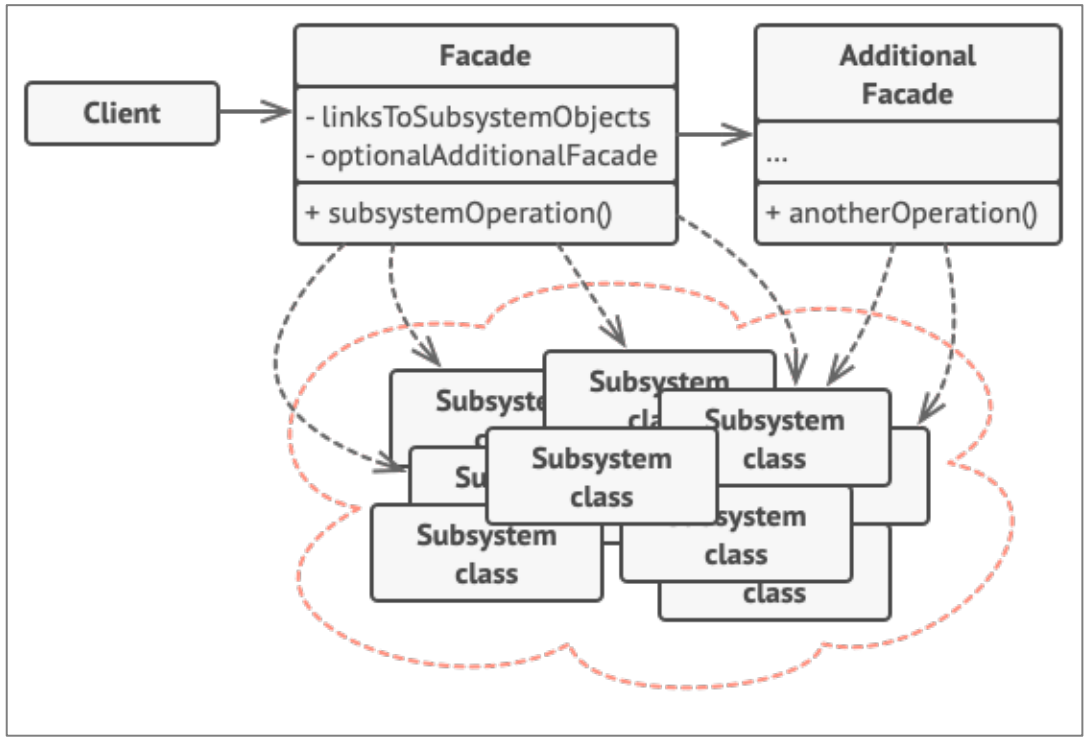
- Projector, Screen, Streaming Player, Theatre lights, Amplifier, Tuner, Theatre lights

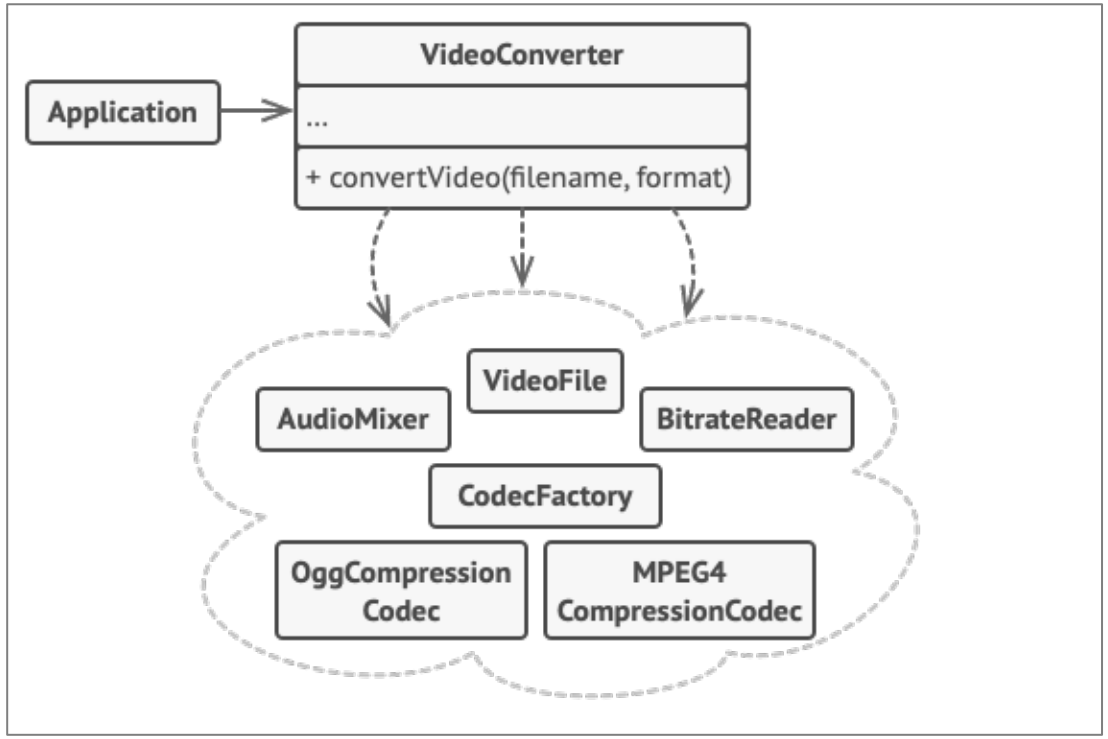
Example: Home Theatre with Façade class



Façade: Other Examples



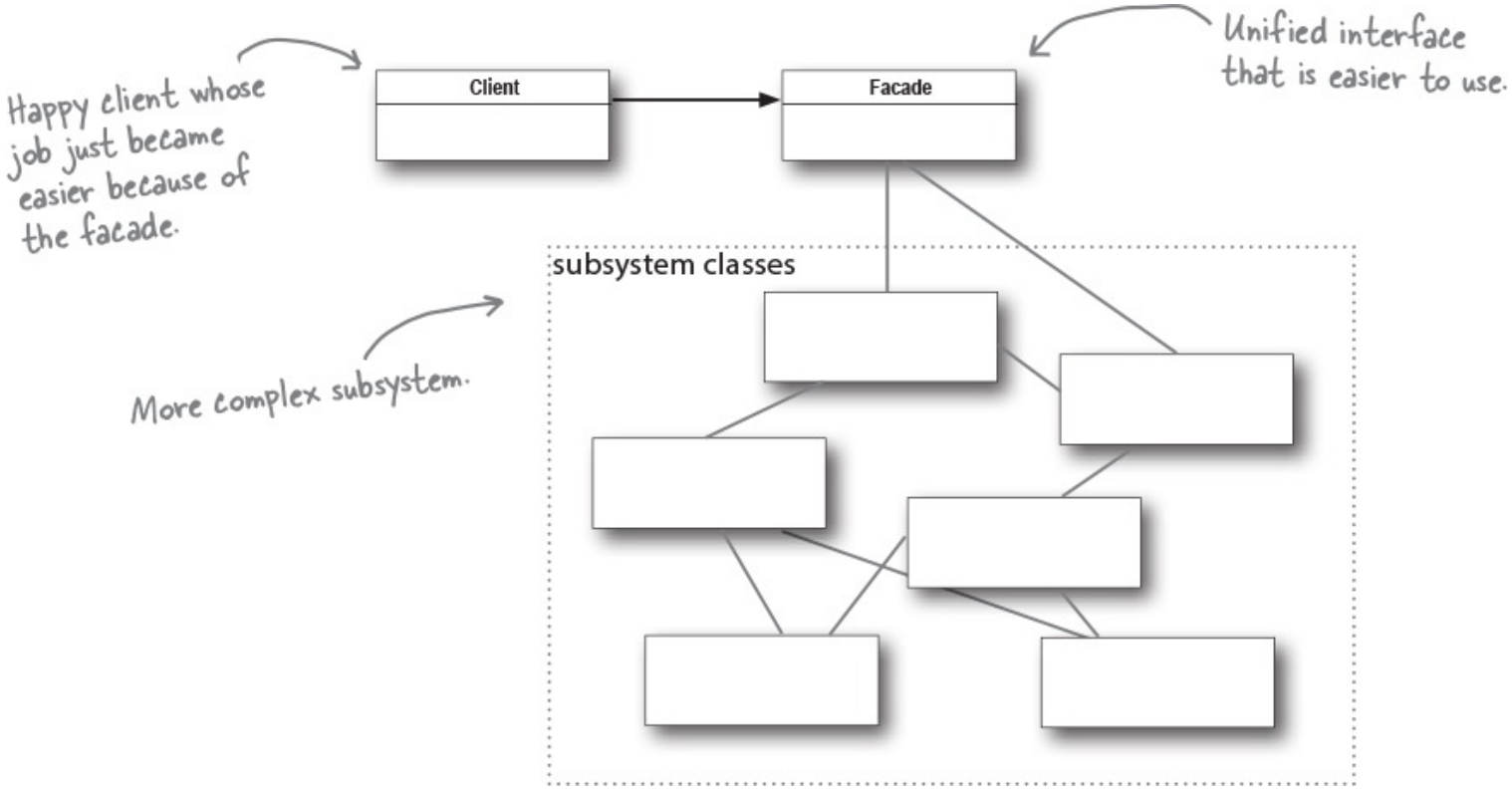




From https://refactoring.guru/design-patterns/facade

Façade Pattern

- ***Important:*** A facade can add domain knowledge to improve client experiences (i.e., set light intensity depending on a time of a day).
- A complex subsystem can have multiple facades, for different clients.
- The Facade Pattern decouples a client interface from any one of the subsystems. For example, you can change a type of your streaming player without changing the façade interface used by clients.



End