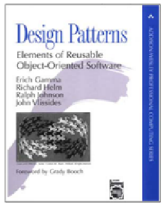


Design Pattern Definitions from the GoF Book

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.

Creational Patterns

- **The Factory Method Pattern**
  Defines an interface for creating an object, but lets subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- **The Abstract Factory Pattern**
  Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- **The Singleton Pattern**
  Ensures a class has only one instance, and provides a global point of access to it.
- **The Builder Pattern**
- **The Prototype Pattern**

Structural Patterns

- **The Decorator Pattern**
  Attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.
- **The Adapter Pattern**
  Converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- **The Facade Pattern**
- **The Composite Pattern**
- **The Proxy Pattern**
- **The Bridge Pattern**
- **The Flyweight Pattern**

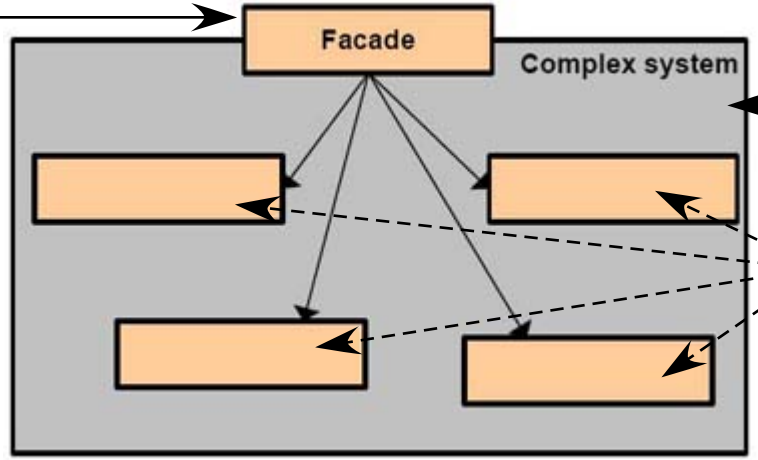
Behavioral Patterns

- **The Strategy Pattern**
  Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
- **The Observer Pattern**
  Defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatially.
- **The Command Pattern**
  Encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.
- **The Template Method Pattern**
- **The Iterator Pattern**
- **The State Pattern**
- **The Chain of Responsibility Pattern**
- **The Interpreter Pattern**
- **The Mediator Pattern**
- **The Memento Pattern**
- **The Visitor Pattern**

Design Patterns: The Facade
Quick Overview

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.

Defines the interface the client will actually use.

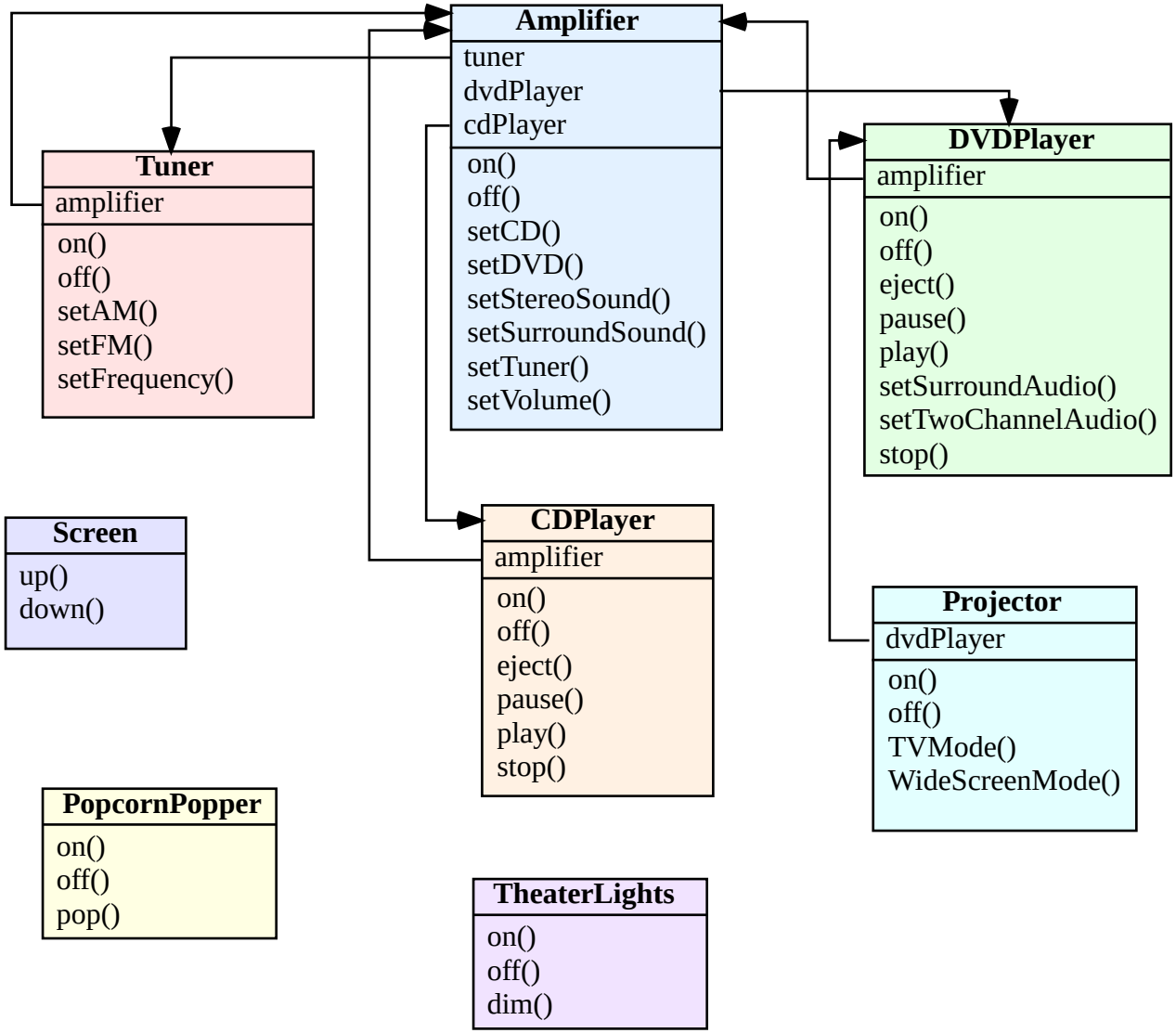


The system or set of objects that the client needs to use.

Interfaces (each different) to objects in the system.

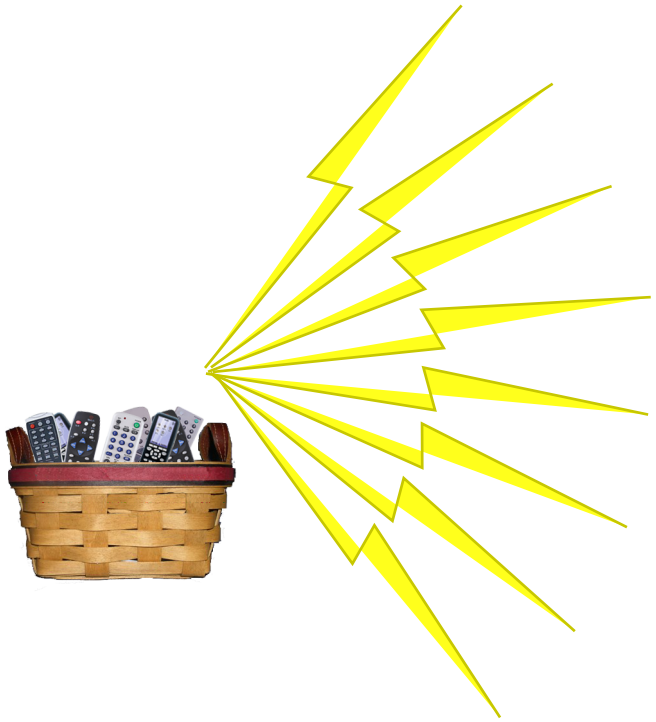
Design Patterns: The Facade

The Home Theater System



Design Patterns: The Facade

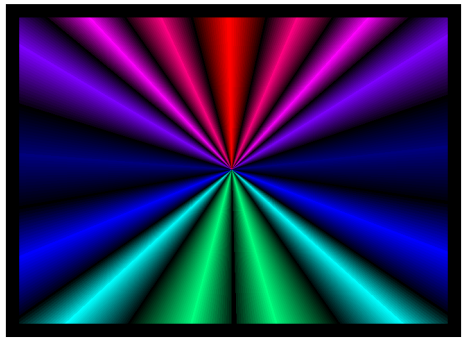
"I want to play a movie."

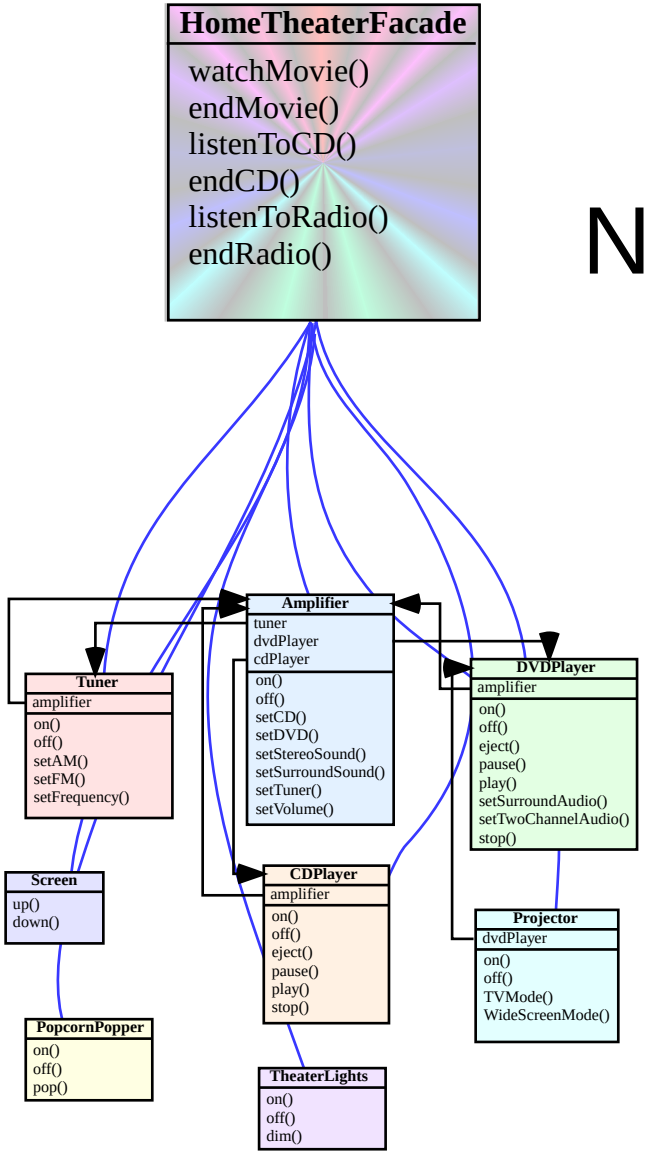


1. Turn on the popcorn popper.
2. Start the popper popping.
3. Dim the lights.
4. Put the screen down.
5. Turn the projector on.
6. Set the projector input to DVD.
7. Put the projector on wide-screen mode.
8. Turn the sound amplifier on.
9. Set the amplifier to DVD input.
10. Set the amplifier to surround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

And after the movie finishes playing you have to repeat everything, but in reverse order to turn the entire system off.

Design Patterns: The Facade





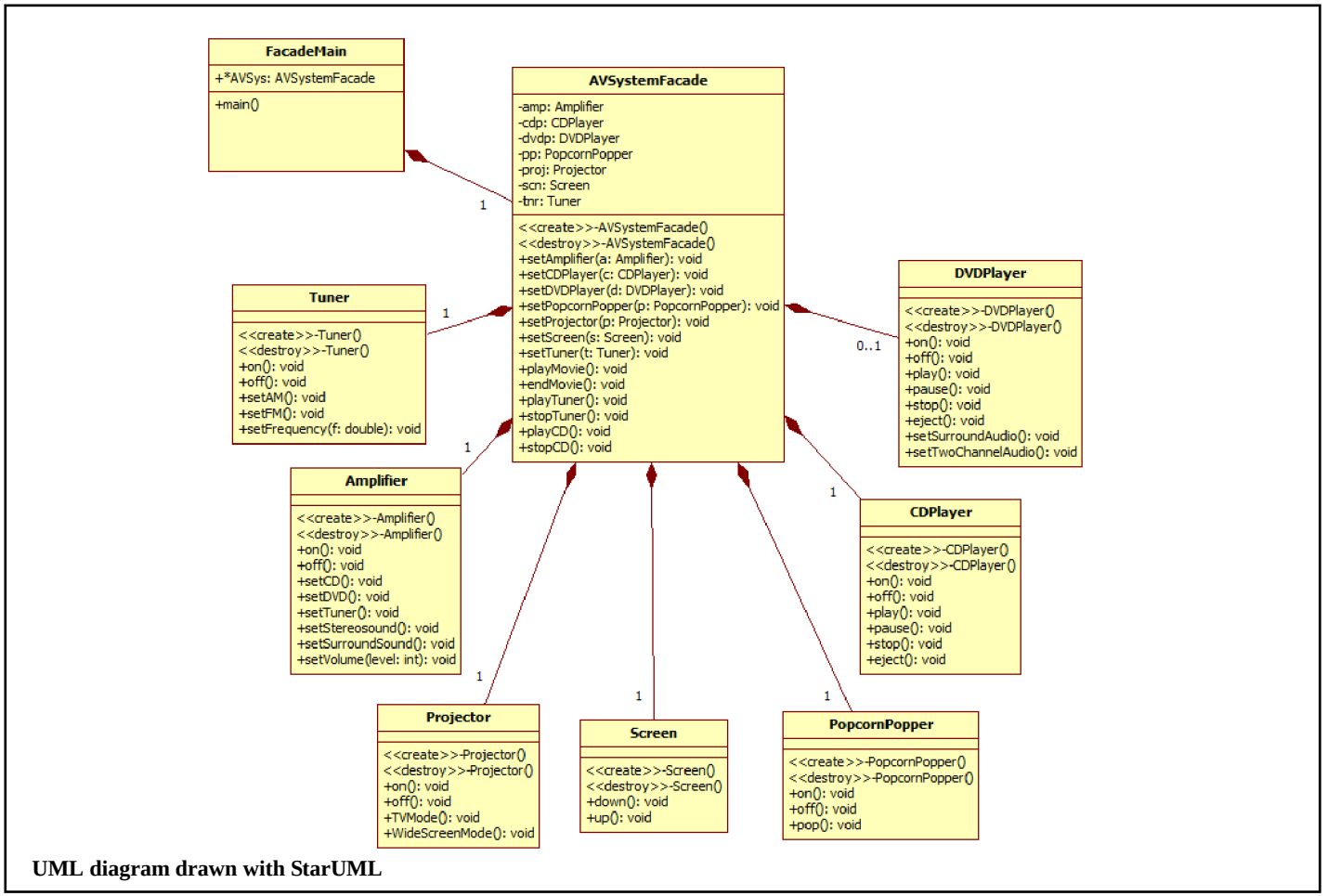
Now isn't that a whole lot easier?

Design Principles

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

Design Patterns: The Facade

Code Sample



UML diagram drawn with StarUML

FacadeMain

Demonstrates how to play a movie without the Facade.
Creates the home theater system.
Demonstrates how to play a movie with the Facade.
Demonstrates how to switch after the movie to play a CD.
Demonstrates how to play a CD with the Facade.

Let's look at the code and run the demonstration.