CMPT 276 Class 16: Software Design Patterns

Dr. Jack Thomas
Simon Fraser University
Fall 2020

Today’s Topics

- What are **Software Design Patterns**?
- Looking at the **Observer** and **Publisher-Subscriber** patterns.
- Using patterns in professional software engineering.

Software Design Patterns

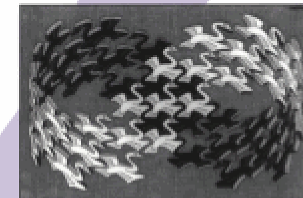
- A description of a **common software design problem** and the **essence of its solution**.
  - Allows discussion, implementation, and reuse of proven software designs.
- **Design Patterns, Elements of Reusable Object-Oriented Software (1994)**: A pioneering book on design patterns by four authours: Gamma, Helm, Johnson, and Vlissides (the "Gang of four")

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

What Do They Look Like?

- Design patterns are **best practices** for situations that commonly occur in **object-oriented programming**.
- They are **abstract**, closer to high-level ideas than specific code.
- They **do not have a single universal format** or documentation style.
- There is **no one comprehensive list** or authority – you can make up your own patterns if you want.

Design Pattern Example: The Singleton

- Already used in this course, and one of the original 23 from *Design Patterns*.
- **In brief**: the Singleton allows a class to be instanced only once.
- The pattern describes **how** to do this (having the class itself control when it is instanced) and also **why** (to coordinate between other objects).
- Instead of one specification, countless different tutorials, examples, and graphs exist online explaining it.

Classifying Software Design Patterns

- *Design Patterns* provided three categories.
- **Creational**: deals with how objects are created.
  - Ex: The Singleton pattern.
- **Structural**: deals with how objects are related to each other, particularly through inheritance and interfaces.
  - Ex: The Wrapper pattern.
- **Behavioral**: deals with how objects communicate and interact with each other.
  - Ex: The Observer and Publisher/Subscriber patterns.

The Observer Pattern: Motivation

- Imagine you are writing an **automatic day-planner**:
  - It reads in the user's interests, plus information about the world, and suggest what they should do.
- Possible design idea:
  - You want to use **different objects for cultural planning, sports planning, and sight-seeing**.
  - Some objects bring in information about the world; your planning-objects use these info objects.
- Challenge:
  - All of these objects need to know the weather.
  - Your weather object gets updates now and then.
  - **How do you tell all the objects new data is available?**

Possible Idea

- Have the weather object **call each info object**:

```
class Weather
void newDataUpdate() {
        String weatherData = ...;
        culturePlanner.update(weatherData);
        sportsPlanner.update(weatherData);
        sightseeingPlanner.update(weatherData);
        // Change here EVERY time you get a new planner.
}
```

- Bad because weather object **is tightly coupled to each planner!**
- Every new planner you get, you'll have to **change the weather object's code**, recompile, and re-run.

The Observer Pattern

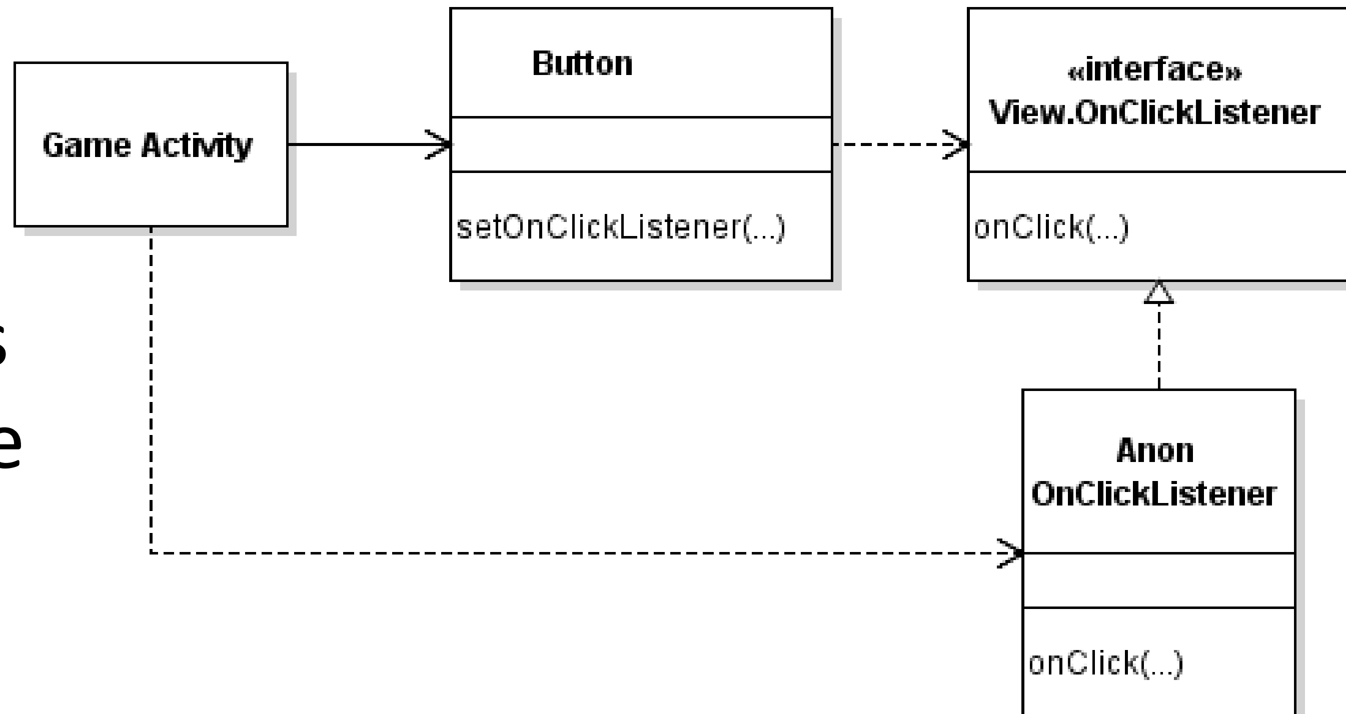
- An object, called the **subject**, is the source of **events.**
- One or more **observer** objects want to be notified when such an event occurs.
- **Solution**:
  1. Define an observer interface type.
  2. All concrete observers implement it.
  3. Subject maintains a collection of observers.
  4. Subject supplies methods for attaching and detaching observers.
  5. Whenever an event occurs, the subject notifies all observers.

The Observer Pattern

- This would allow objects to “register for updates” with another object at run-time.
- Produces a **one to many relationship**:
  - One object observed (called the **subject**)
  - Many objects observing (called the **observers**).
- Great because it **loosely couples** objects:
  - Object with something to report does not need a hard-coded list of who to tell; it simply looks up its observer list.

Observer Button Example

- **Button** knows of a click; **Game Activity** wants to know.
- Activity creates anonymous **OnClickListener**
    - Activity registers it with button as a listener.
- The benefit is decoupling: Button knows nothing of the program.



Publisher/Subscriber Pattern

- Another Behavioral-type pattern.
- A subject's list of observers is replaced with **brokers** that it **publishes messages** to.
- The observers can **subscribe** to these brokers and then read the messages.
- **Ex**: The **Robot Operating System** (ROS) uses **topics** like /velocity. High-level behaviour classes publish new movement orders to that topic, while other low-level classes that control the robot's hardware are subscribed to it and read the messages as they come in.

Observer vs. Subscriber/Publisher

- What is the material difference between these two patterns?
  - The **Subject** knows it has **Observers**, and the **Observers** know who they are observing. Both **Subscribers** and **Publishers** only know about **Brokers** and the **Messages** posted there.
- When would you use one over the other?
- Definitions are fuzzy, some believe Sub/Pub is just a sub-type of Observer.

Software Design Patterns and Professional Practice

- Well-known design patterns are **generalized** across almost all software development.
- A common fundamental of **technical interviews**.
- *Design Patterns* or *Code Complete* are good places to start, but look for other patterns popular in the field and in discussions online.

Domain-Specific Design Patterns

- Different sub-fields of software engineering create their own patterns as needed.
- **Ex**: Android (and mobile development) makes significant use of the Model-View-Controller (**MVC**) and Model-View-Presenter (**MVP**) patterns.
- Learning the design patterns common to a particular field is an important step toward professional specialization.

Recap: The Summary Pattern

- Software Design Patterns are a **collection of common best-practices** for object-oriented programming.
- The **Observer Pattern** is a behaviour where a subject class maintains a list of observers to notify whenever they update their state.
  - The **Subscriber/Publisher Pattern** is a variation where messages are posted to a middleman instead of directly from subject to observers.
- In professional software engineering, design patterns are a **basic part of your toolset** and often **tailored to the domain** you work with.