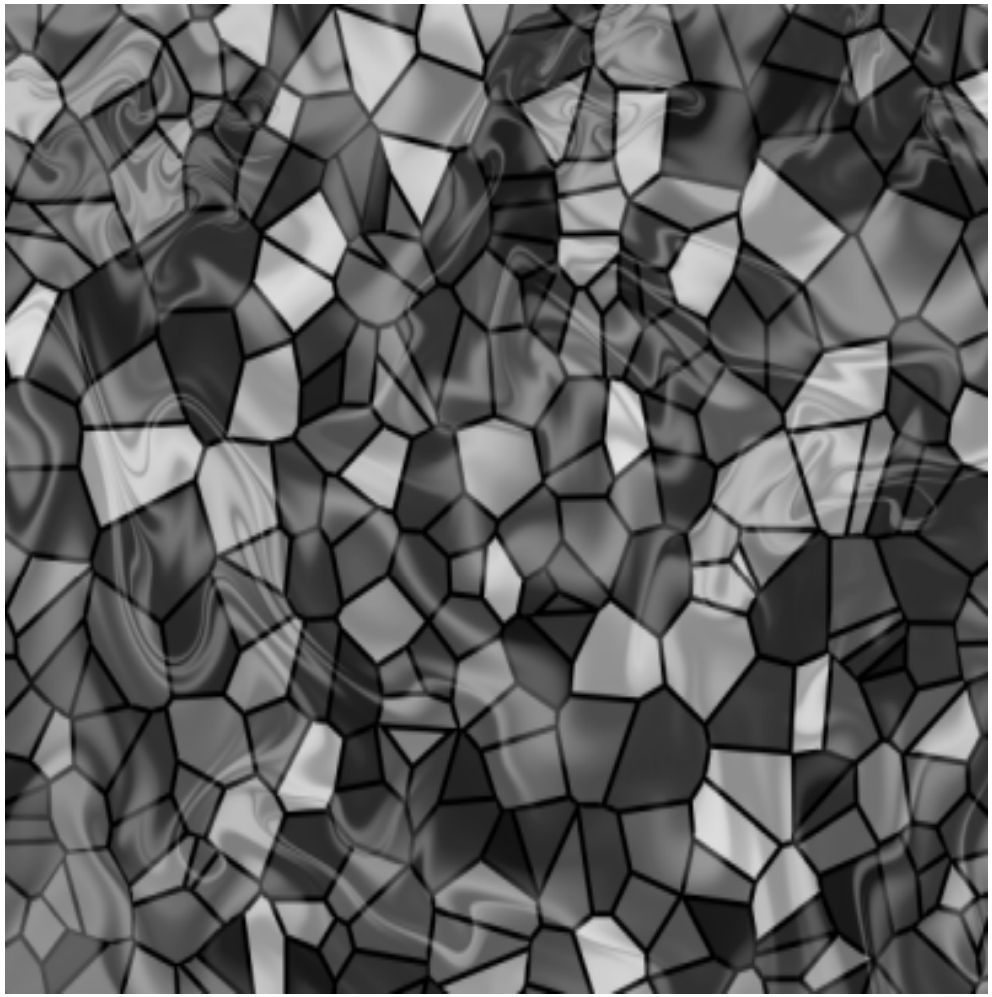
Observer Pattern

Photo by James Frid from Pexels

Topics

1) What is a software design pattern?
2) How can an object be notified when an event occurs in another object?



Software Design Patterns

OOD

- Object Oriented Design (OOD)

  ..
  - Coding techniques to arrange dependencies such that object can tolerate change
  - Prevents changes cascade from one module to another
- Ex: FPS video game tightly coupled to select-gun module
  - never refactored because it was core to the *structure* of the game
- Common Technique
  - Often decouple code by adding

    ..

    between modules (ex: adding an interface)

Design Patterns

- Design Patterns

  ..

- Not Code
  - Design patterns are design ideas
  - *Archetypes* (like the mysterious wizard, or hero) to help us understand the big picture quickly
- Common Language
  - Gives common language to more easily discuss complex solutions

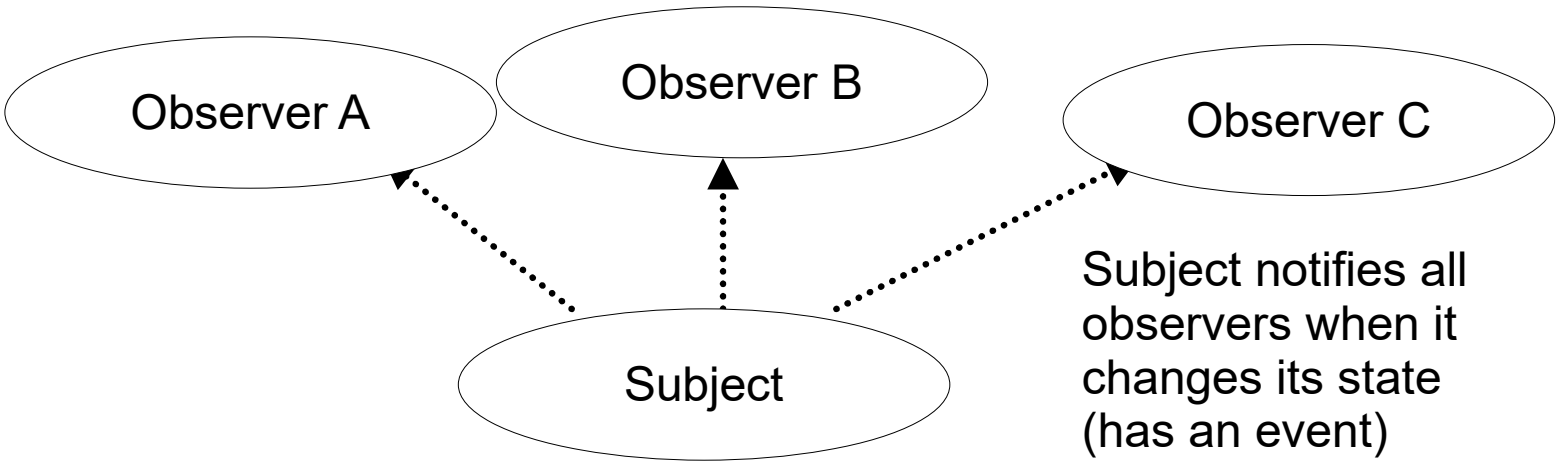
Design Patterns

- A design pattern addresses a common problem
  - We must learn..
    and..
    so we can apply it to future problems
- Use in incorrect situations is bad
  - it forces us to conform to a design which does not fit
  - we must adapt design pattern to our needs
  - Ex: CMPT 276 students wanting to apply Singleton to every class

Observer Pattern

Motivation

- Situation
  - An object (..) needs to notify any number of objects (..) to..
- Diagram
  (showing notification, not UML dependencies)

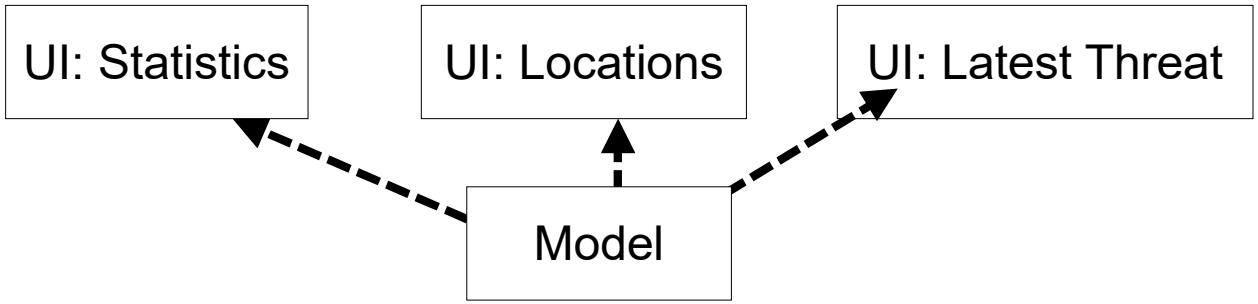


Motivation

- Ex: Network threat analysis
  - Model watches network traffic for threats
  - UI shows:
    - traffic sources & statistics
    - latest threat
    - highest priority threat
    - etc.
  - Model notifies UI components when updated analysis data is available

Idea 1: Model calls UI

- Idea 1: Model call methods in UI
  - Model knows which UI objects to call



Model depends on (has reference to) each individual part of UI

- Good: Works! Easy to understand
- Bad:
  - Couples the model to each parts of the UI (Couples the subject to each of the observes)
  - Since UI already holds reference to model, this would create a circular dependency

Idea 1: The code

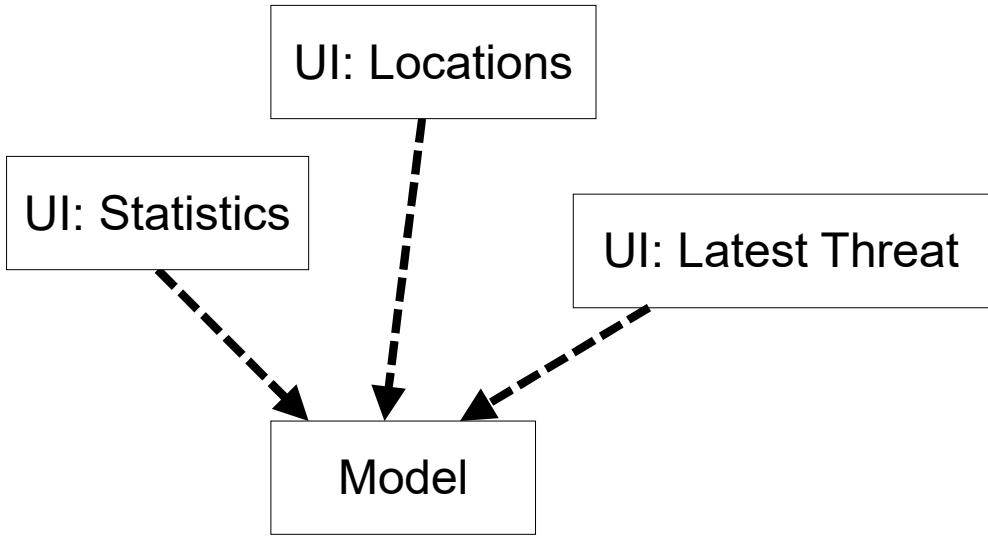
```
class ThreatModel {
    private UiThreatList uiList;
    private UiThreatLocations uiLocations;
    private UiThreatLatest uiLatest;

    public void notifyUiOfNewAnalysis() {
        uiList.notifyNewAnalysis();
        uiLocations.notifyNewAnalysis();
        uiLatest.notifyNewAnalysis();
    }
}
```

- How could model get references to UI?
  - Instantiate them? (Coupling!); DI (dependency injection)
- Model tightly coupled to UI:
  change to UI requires change to model
  - Violates the Open-Closed principle

Idea 2: UI Polls

- Each UI class polls model for updates
  - Often model replies that nothing has changed



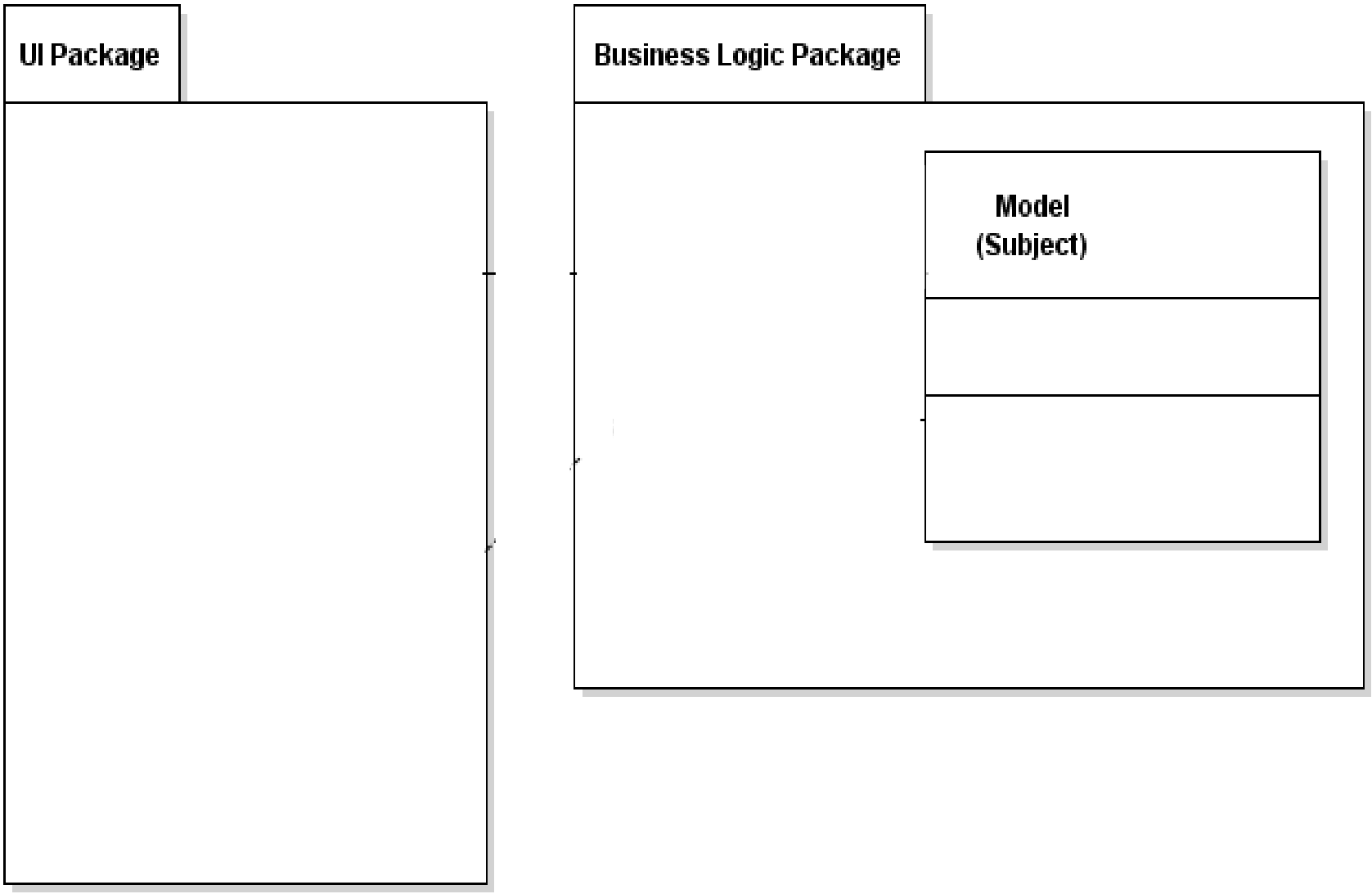
UI classes each reference model (good)
but repeatedly ask for changes (bad)

- Good?
  - ..
- Bad?
  - ..
  - Slow to update UI

Idea 3: Observer

- Observer Process
  - Model defines an Observer interface
  - UI classes create Observer objects (often anonymous classes)
  - UI register's its Observer object with the Model
  - Model calls notifyChangedState() (or the like) to inform Observers of an event
- When no longer needing notifications
  - UI unregisters its Observer object from the Model

Observer UML



Demo

- Open Patterns-Observer IntelliJ Project
- Without Observers
  - Package: no_observer.simple
  - Class: MainText
- With Observers
  - Package: no_observer.simple
  - Class: MainText
  - UI: MainUI

Observer Pros / Cons

- Advantages
  - ..

    Can change UI and not change the Model's code
  - Multiple observers can register with subject;
    all get updated for each change

    Ex: Multiple views in a UI
- Drawback
  - ..

    No compile time references to see what code will be executed when data changes

Adapt Observer Pattern

- Patterns are design ideas: not rigid
  - Just 1, or many observers?
  - Observer have one notify method, or several?
- Android Adapts Observer Pattern
  - Button click:
    myButton.setOnClickListener(...)
  - Text change in TextEdit:
    myEditText.addTextChangedListener(...)
- Java Swing
  - myJButton.addActionListener(...)

Summary

- Observer
  - Objects register an observer to a subject at runtime
  - Subject notifies all registered observers when an event occurs
  - Observers can unregister
- Benefits
  - Loosely couples subject to the observing object
  - Allows for efficient notifications