

Design Patterns

Observer Pattern

School of Computing
Clemson University
Clemson, SC 29634 USA

CPSC 2150: Wednesday, Mar. 29, 2023

Slide Sources:

- 1 Previous/Current CPSC 2150 Instructors
- 2 The lecture slides have also benefitted from instructors at Ohio State: Paolo Bucci, Wayne Heym, Paul Sivilotti, and Bruce Weide.

1 Design Patterns

- ▶ Introduction
- ▶ Interfaces and Abstract Classes
- ▶ Design Patterns and OOP Languages

2 Template Method Pattern

3 Factory Method Pattern

Homework Assignment

- 1 Programming assignment 4 is posted on your lab Canvas page.

Exam #2

1 Exam #2 - April 10, 2023

- ▶ Please complete the Lockdown Browser + Monitor Practice Quiz as part of your participation grade when it opens up. More information will be announced on Canvas.
- ▶ Will post review slides soon!

What Are Design Patterns?

“a general reusable solution to a commonly occurring problem within a given context in software design”

- ▶ Design patterns represent the collective experience of decades of object-oriented software development experience.

A Motivating Example

Suppose. . . you're writing a program that receives data from a network and performs various tasks when data is received (displays the message, sends response data, etc.). The tasks to be performed may change depending on the circumstances. How do you design object interactions to allow this flexibility?

A Motivating Example

Suppose...you're writing a program that receives data from a network and performs various tasks when data is received (displays the message, sends response data, etc.). The tasks to be performed may change depending on the circumstances. How do you design object interactions to allow this flexibility?

- ▶ You could come up with your own solution...but why reinvent the wheel?
- ▶ **Solution 1:** Use polling
- ▶ **Solution 2:** You look up behavioral design patterns (interactions) and find the **observer pattern**!

A Motivating Example

Terminology

- ▶ We want to monitor the network for **events**
- ▶ The machine on the network where the event occur is the **subject** of the interaction
- ▶ The objects in your program that need to do something in response to the events for a particular subject are called **observers** (or **listeners**) for that subject

A Motivating Example

Solution #1: Use Polling



subject



observer

- ▶ The main program (*the only observer*) continually **polls** each possible subject to ask whether any events have occurred
- ▶ This is considered cumbersome. . .
- ▶ This is **NOT** the observer pattern

A Motivating Example

Solution #1: Use Polling (Pseudocode)

```
while (true) {  
    if (s0 has experienced an event) {  
        if (event is e0) {  
            respond to it  
        } else if (event is e1) {  
            respond to it  
        } else ...  
    } else if (s1 has experienced an event) {  
        ...  
    }  
}
```

A Motivating Example

Solution #2: Use Callbacks (Observer Pattern)



- ▶ Each observer (*there may be many*) registers its interest in a subject's events, and then waits until the subject **calls it back**¹ to tell it that there has been an event

¹But how? What does this mean?

Observer Pattern

- ▶ Each subject expects each observer (listener) to **register**² itself with that subject if it is interested in the subject's events
- ▶ Each subject keeps track of its own **set of interested observers**³
- ▶ Whenever an event occurs, the subject invokes a specific **callback method**⁴ for each registered observer, passing an **event argument** that describes the **event**
- ▶ This is one of many *object-oriented design patterns* that address common OOP issues (often language deficiencies); most are considered **best practices**.

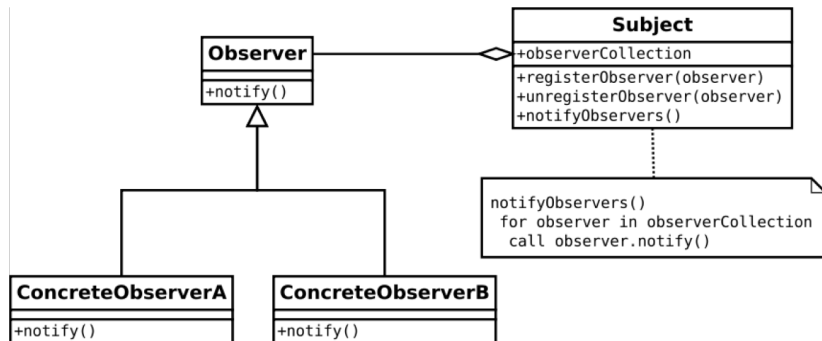
²Registering interest is done by calling a method of the subject; usually this is done once as part of set-up.

³The set of observers for a given subject can be kept in a **Set** variable, for example.

⁴This method is described in an **interface** that any potential observer must implement.

Observer Pattern

UML Diagram



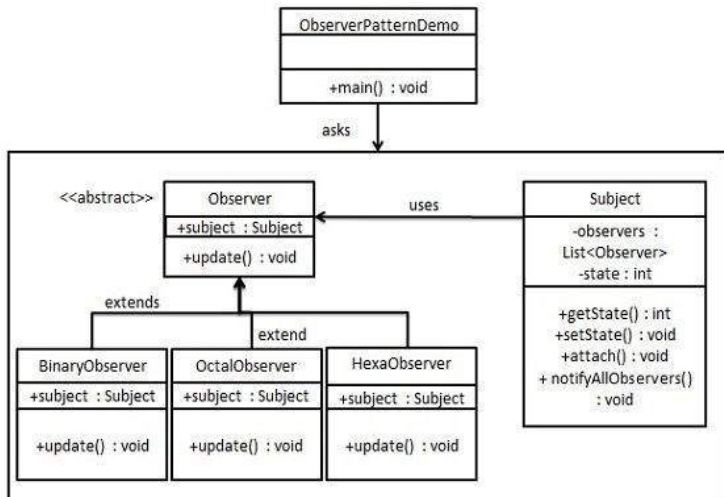
Source: By Gregorybleiker - Own work,

https://commons.wikimedia.org/wiki/File:Observer_w_update.svg

Observer Pattern Demo

- ▶ Use the starter code provided on Canvas and let's complete it to implement the Observer Pattern.
 - ▶ Modified from https://www.tutorialspoint.com/design_pattern/observer_pattern.htm
- ▶ **Goal:**
 - ▶ Every time we update `MyInteger`, we want to print out the binary, octal and hexadecimal equivalent.

Observer Pattern Demo



Source: https://www.tutorialspoint.com/design_pattern/images/observer_pattern_uml_diagram.jpg

Observer Pattern

Graphical User Interfaces (GUI)

- ▶ Another use is with Graphical User Interfaces (GUI)
- ▶ Every “widget” on the screen is an object in the code
 - ▶ Buttons, textboxes, etc
- ▶ We don't know what widget the user will interact with next
- ▶ The widgets are the subjects
- ▶ We create an observer for them

Observer Pattern

Graphical User Interfaces (GUI)

- ▶ Every “widget” on the screen is becomes a subject
 - ▶ Implements some **Subject** interface
- ▶ Every subject contains a set of interested observers
 - ▶ The observers will have to register with the subjects to be added to this list
- ▶ Every subject must provide a method for observers to register (takes in an **Observer** object)
- ▶ When someone interacts with the “widget”, it notifies its observers

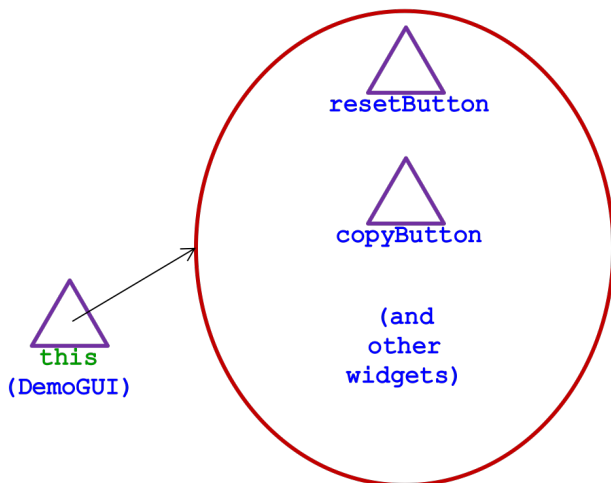
Observer Pattern

Graphical User Interfaces (GUI)

- ▶ Our `Screen` class will become the observer
- ▶ Every “widget” on the screen is declared as a `private` variable of the screen itself
 - ▶ Now the screen can access all the “widgets”
- ▶ The screen class needs to call the register method for each “widget”, and pass itself in.
- ▶ The `Screen` class will have to provide the call back method
 - ▶ The “widget” will call this method to let the screen know that the user interacted with it
 - ▶ In this method, the screen decides how to react to someone interacting with that “widget”

Observer Pattern

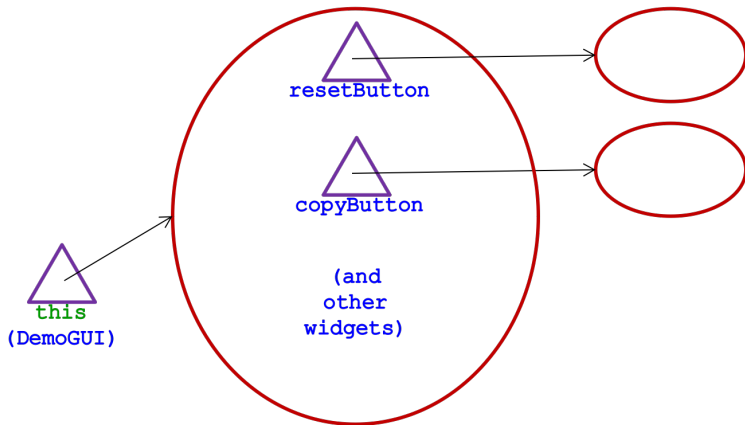
Illustrative Example



Set Up by `DemoGUI` Constructor

Observer Pattern

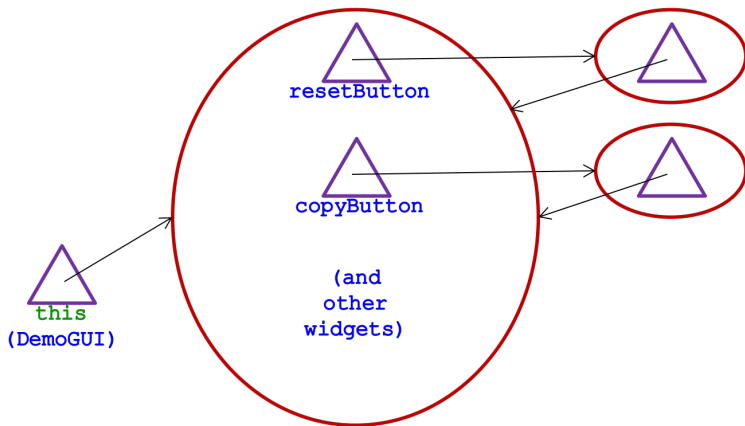
Illustrative Example



Before registration of `this` as an observer of the buttons.

Observer Pattern

Illustrative Example



After registration of `this` as an observer of the buttons.

Homework Assignment

- 1 Programming assignment 4 is posted on your lab Canvas page.

Exam #2

1 Exam #2 - April 10, 2023

- ▶ Please complete the Lockdown Browser + Monitor Practice Quiz as part of your participation grade when it opens up. More information will be announced on Canvas.
- ▶ Will post review slides soon!

Summary

- 1 A Motivating Example
- 2 Observer Pattern
 - ▶ UML Diagram
 - ▶ Demo
 - ▶ Graphical User Interfaces (GUI)
 - ▶ Illustrative Example