UMassAmherst | Manning College of Information & Computer Sciences

Programming Methodology

# Lab 11: Observables

Wednesday November 12, 2025

# Reminders

- HW 6 is released! You should have received an email last night from CATME with information about your assigned team
  - Reach out early and begin coordinating time to work together!
- Midterm 2 grades have been released
  - Regrade requests will be open until Wednesday 11/19
  - Request a regrade if you feel unsure about the grading or believe there was a mistake made in grading

# Today’s Goals

- Homework 6 setup
- Observables
- Observers

# Homework 6: Do these now!

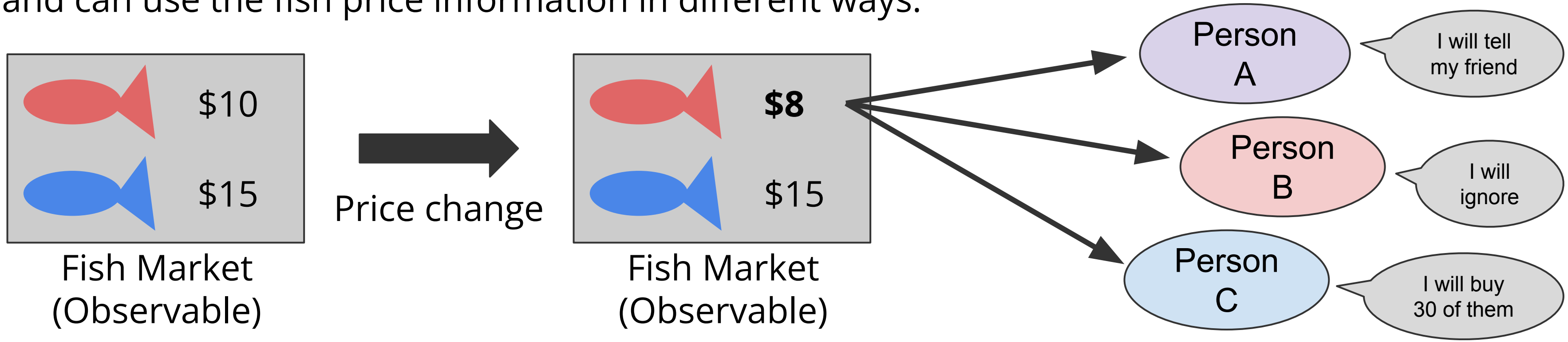
1. **PLEASE** disable the jest extension in VSCode if you are using it. An alternative extension would be the jest-runner extension. See the HW specs if you are curious why!
2. If you do not have a Github account under your UMass email:
    - Check your email for an invite to a Github repository. When you click it, you can login with an existing GitHub account, or you can create a new one
    - It is not required that your GitHub account uses your UMass email, personal accounts are acceptable
3. If you already have a Github account under your UMass email, you may need to check your notifications in Github for the invite
4. Please accept the Github invite before it expires!

# Observables Review

The Observer design pattern describes two main components, Observers and Observables.

Observables are like publishers, which broadcast information to their “audience”. The members of this audience are Observers. Observers subscribe to Observables in order to receive the broadcasted information.

For example, I could have an Observable for the price of different fish in a market. Each time the price of a fish changes, I will broadcast the new price to anyone who is interested. Specifically, people (Observers) can indicate their interest by subscribing to my Observable, and can use the fish price information in different ways.



# Observables Review

```
interface Observable {
  // adds the given subscriber to the set of subscribers
  subscribe(subscriber: Observer): void;

  // removes the given subscriber from the set of subscribers
  unsubscribe(subscriber: Observer): void;
}
```

The Observer design pattern is defined differently in many contexts, but in this class we will assume Observables have the above interface. Observers do not have a set interface, and will be defined based on the context.

We will typically assume both Observables and Observers to be objects, but there are instances where it can make more sense for Observers to be functions! See the lecture notes for more details.

# Observables Review

```
interface Observable {
  // adds the given subscriber to the set of subscribers
  subscribe(subscriber: Observer): void;

  // removes the given subscriber from the set of subscribers
  unsubscribe(subscriber: Observer): void;
}
```

Note: if you have started working on the extra credit homework, you will have noticed that Observables and Observers are defined differently in that assignment compared to lecture.

These definitions are not too distinct, but to avoid confusion, any lecture, lab, or exam questions on Observables will use the above interface and treat Observers as objects, unless otherwise stated.

The extra credit homework gives you a chance to explore a slightly different definition of Observables, and how you can essentially compose Observables in different scenarios.

# Context of Today's Exercises

You will be implementing an Observable and two Observers for handling different (computer) mouse events. Specifically, mouse clicks and mouse drags. Mouse clicks are described by the position (coordinate) clicked, and mouse drags are described by the start and end positions when the mouse is dragged.

```
interface MouseObserver {
  screen: number;

  // update observer with new mouse click coordinate
  update_coordinate(coord: Coordinate): void;

  // update observer with new mouse drag event
  update_drag(start: Coordinate, end: Coordinate): void;
}
interface Observable {
  // adds an Observer to the set of subscribers
  subscribe(subscriber: MouseObserver): void;

  // removes an Observer from the set of subscribers
  unsubscribe(subscriber: MouseObserver): void;
}
```

Note that MouseObservers have a public property screen that defines which screen they are observing (i.e. which monitor).

Assume screen numbers start at 0 and count upwards. I.e. if there are 3 screens total, the valid screen numbers would be 0, 1, 2.

Coordinate is defined as {x: number, y: number}. You can also assume that both x and y are non-negative.

# Exercise 1: Observables

- Create a class **MouseEvents** that is an Observable, implementing the following:
  - `constructor(num_screens: number)`
    - Initializes number of screens of MouseEvents and any other attributes
  - `subscribe(subscriber: MouseObserver): void`
    - Adds the given observer to a set of observers
  - `unsubscribe(subscriber: MouseObserver): void`
    - Removes the given observer if they are subscribed
  - `mouse_click(screen: number, coord: Coordinate): void`
    - Updates all observers for the given screen with the mouse click Coordinate
  - `mouse_drag(screen: number, start: Coordinate, end: Coordinate): void`
    - Updates all observers for the given screen with the start and end coordinates of the mouse drag
- Use the definitions at the top of the starter code! Try to reduce code duplication and think about how to broadcast updates efficiently.

# Exercise 2a: Observers

- Implement the MouseEventLogger Observer, with the following methods:
  - **`update_coordinate`**`(coord: Coordinate): void`
    - Takes the given coordinate and prints out the string "`Click: (x, y)`", with x and y replaced by the respective coordinates in `coord`.
  - **`update_drag`**`(start: Coordinate, end: Coordinate): void`
    - Takes the given coordinates and prints out the string "`Drag: (x1, y1) to (x2, y2)`", with x1 and y1 replaced by the start coordinates, and x2 and y2 replaced by the end coordinates.
- The constructor has already been implemented for you
- Your implementations of these methods should pretty much just be one line each!

# Exercise 2b: Observers

- Implement the MouseEventArea Observer, with the following methods:
  - **`constructor`**`(screen: number)`
    - Initializes the screen number of this MouseObserver, and any other attributes
  - **`update_coordinate`**`(coord: Coordinate): void`
    - Calculates the area of the smallest (not rotated) rectangle that contains all coordinates clicked on so far
    - Prints "`Coordinate area: A`", with A replaced by the calculated area.
    - Hint: consider the rectangle formed by (x_min, y_min) and (x_max, y_max) as opposite corners, where x_min, y_min correspond to the smallest x or y values seen so far
  - **`update_drag`**`(start: Coordinate, end: Coordinate): void`
    - Calculates the area of the rectangle formed by the start and end coordinates as opposite corners.
    - Prints "`Dragged area: A`", with A replaced by the calculated area.