

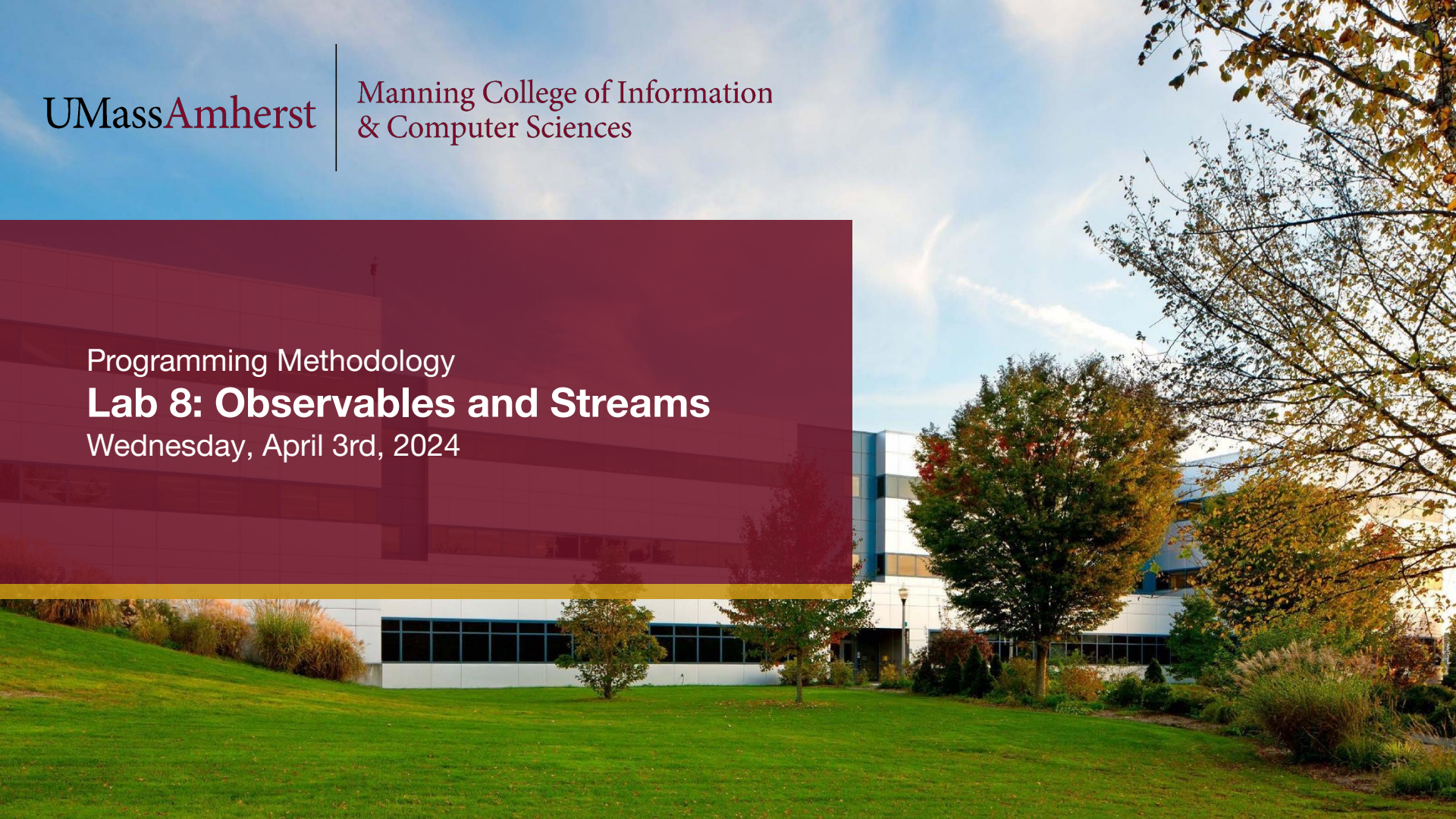
UMassAmherst

Manning College of Information
& Computer Sciences

Programming Methodology

Lab 8: Observables and Streams

Wednesday, April 3rd, 2024



Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
 - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

A View of the Week Ahead

WED Apr 3		SUN Apr 7		TUE Apr 9	WED Apr 10
We're here Observables EC Released	...	Exam Review Session from 7-9pm @ILC S140	...	Observables EC Due SI Review Session from 7-9pm @ILC N151	Exam 7-9pm @TBD More details to follow

CATEME due Friday!

Reminders

- Download the starter code.
- Homework 5 is due tonight at 11:59pm
 - Come to [office hours](#) for help!
- The observables extra credit assignment will be released
 - Due **Tuesday April 09** at midnight
- Complete the **CATME Survey** by next **Friday April 12** at midnight
- Midterm 2 is next week!
 - Start studying early.
 - Lab next week will be held as scheduled and attendance is required

Today's Goals

- Practice working with the observer pattern
- Practice working with streams

Observer Review

- What: A design pattern in which an observable subject automatically notifies dependent observers of any state changes
- Why: It's everywhere. E.g: GUI updates
- How: Reusable class

```
type Observer<T> = (x: T) => any;

class Observable<T> {
  private observers: Observer<T>[] = []; // Maintain a list of observers

  subscribe(f: Observer<T>) {           // Add an observer to the list
    this.observers.push(f);
  }

  update(x: T) {                         // Notify each observer of update
    this.observers.forEach(f => f(x));
  }
}
```

Observer Review

```
1  type Observer<T> = (x: T) => any;
2
3  class Observable<T> {
4      private observers: Observer<T>[] = []; // Maintain a list of observers
5
6      subscribe(f: Observer<T>) { // Add an observer to the list
7          this.observers.push(f);
8      }
9
10     update(x: T) { // Notify each observer of update
11         this.observers.forEach(f => f(x));
12     }
13 }
14
```

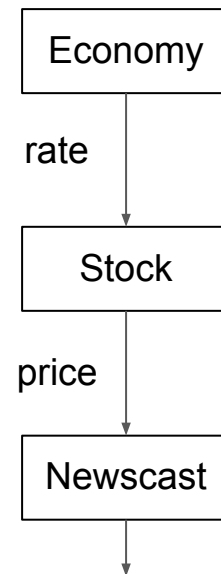
Exercise 1

- Model the stock market with 3 classes. Make sure to test!

```
// Should be "observable"
class Economy /* possibly extends something */ {
    updateRate(rate: number): void {} // Notify whoever cares about the economy
}

// Should observe Economy's rate, and be "observable"
class Stock /* possibly extends something */ {
    constructor(name: string, base: number, price: number) {}
    updatePrice(rate: number): void {} // Update price = base * rate
}

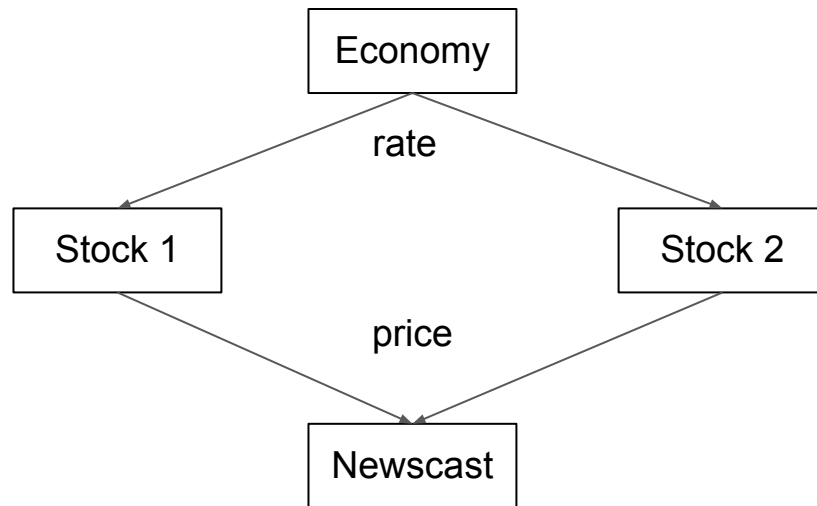
// Should observe and report Stock's price
class Newscast {
    constructor() {}
    report(name: string, price: number): void {
        console.log(`Stock ${name} has price ${price}.`)
    }
}
```



Print stock price

Exercise 2

- Add a function `observe(stocks: Stock[])` to `Newscast` so that it can observe any number of input stocks
- Make `Newscast` be an `Observable` that updates subscribers with the tuple `[stockName, stockPrice]` of type `[string, number]` whenever it reports



Stream Review

- What: A sequence of data made available over time
- Why: Useful abstraction for the paradigm where there's limited random data access and each data record can only be seen once*. E.g: Data reading, signal processing
- How: We implemented stream as a lazily constructed list with memoized tail

```
interface Stream<T> {  
  head: () => T;  
  tail: () => Stream<T>;  
  isEmpty: () => boolean;  
  toString: () => string;  
  map: <U>(f: (x: T) => U) => Stream<U>;  
  filter: (f: (x: T) => boolean) => Stream<T>;  
  reduce: <U>(f: (acc: U, e: T) => U, init: U) => Stream<U>; // This is new  
}  
  
reduce: (f, init) => snode(init, () => memoizedTail.get().reduce(f, f(init, head)))
```

Exercise 3: Maxima stream (in a previous exam!)

- Implement `maxUpTo(s: Stream<number>): Stream<number>`
- Input: A stream of numbers a_1, a_2, a_3, \dots ,
- Output: A stream of maxima of numbers up to the current one:
 $a_1 \Rightarrow \max(a_1, a_2) \Rightarrow \max(a_1, a_2, a_3) \Rightarrow \dots \Rightarrow \text{empty}$
- Example:
Input stream: $1 \Rightarrow 4 \Rightarrow 3 \Rightarrow 2 \Rightarrow 5 \Rightarrow 1 \Rightarrow \text{empty}$
Output stream: $1 \Rightarrow 4 \Rightarrow 4 \Rightarrow 4 \Rightarrow 5 \Rightarrow 5 \Rightarrow \text{empty}$