Flow of Control

\[
x = 2 \\
y = x * 2 \\
z = buzz(y) \\
\text{if } z < 10 \\
\quad w = \text{foo}(z) \\
\text{else} \\
\quad w = \text{bar}(z)
\]
Observer Pattern

Subject
- Attach(Observer)
- Detach(Observer)
- Notify()

ConcreteSubject
- GetState()
- Update()

Observer
- Update()

ConcreteObserver
- subject
- observerState

subject
observer A
observer B
- GetState()
- Update()

SetState()
Notify()
class Counter extends Observable {
    private int count = 0;

    public int value() { return count; }

    public void increase() {
        count++;
        setChanged();
        notifyObservers( "INCREASE" );
    }

    public void decrease() {
        count--;
        setChanged();
        notifyObservers( "DECREASE" );
    }
}
Java Observer

class IncreaseDetector implements Observer {
    public void update( java.util.Observable whatChanged, java.lang.Object message) {
        if ( message.equals( "INCREASE" ) ) {
            Counter increased = (Counter) whatChanged;
            System.out.println( " changed to " + increased.value());
        }
    }
}

public static void main(String[] args) {
    Counter test = new Counter();
    IncreaseDetector adding = new IncreaseDetector();
    test.addObserver(adding);
    test.increase();
}
Flow of Control

```java
public static void main(String[] args) {
    Counter test = new Counter();
    IncreaseDetector adding = new IncreaseDetector();
    test.addObserver(adding);
    test.increase();
}
```

Flow of control not explicit

Don’t see that `increase()` executes code in `IncreaseDetector`
class Counter extends Observable {
    private int count = 0;
    private IncreaseDetector observer = new IncreaseDetector();

    public int value() { return count; }

    public void increase() {
        count++;
        observer.update(this, null);
    }
}

But less flexible
    Only one observer
    Have to modify code to add more or change observer
Observer Pattern

Reduces coupling between subject & observers
   Subject can have any number of observers
   Subject does not know type of the observers

Flow of control is obscured
Observer Pattern - Basic Steps

Subject changes
   You write code to trigger notify to observers

Observer
   Get notified that subject changed
   You write code to react to the change
Java Listeners

You add a listener to an event source
The event source triggers the notification

You write code in listener to react to the event

```java
public class Beeper extends JPanel implements ActionListener {
    JButton button;

    public Beeper() {
        super(new BorderLayout());
        button = new JButton("Click Me");
        add(button, BorderLayout.CENTER);
        button.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }
}
```
(def counter (atom 0))

(defn counter-observer
    [key pointer old new]
    (when-not (== old new)
        (if (< old new)
            (println "Increase")
            (println "Decrease"))))

(add-watch counter :example counter-observer)

(swap! counter inc)

Like listener
We just write code to
React to event
Register for updates

Changing the atom automatically
calls the observer function
Listener - Basic Steps

Subject changes
  You write code to trigger notify to observers

Observer
  Get notified that subject changed
  You write code to react to the change
React-Clojure Example

(def click-count (atom 0))

(defn stateful-with-atom []
  [:div "Number of clicks " @click-count])

(defn clicker []
  [:div {:on-click #(swap! click-count inc)}
   "Click on me"])

(defn home-page []
  [:div [:h2 "Click Example"]
   [clicker]
   [stateful-with-atom]])
React-Clojure - Basic Steps

Subject changes
  You write code to trigger notify to observers

Observer
  Get notified that subject changed

  You write code to react to the change
Basic Idea of Reactive Programming

When you change the value of a variable

   All uses of that variable are automatically updated

Reduces observer pattern to just using a variable
## Common Example - Spreadsheets

$$=$$A$1 + $B$1

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>
Reactive Programming

Programming paradigm oriented around data flows and the propagation of change.

This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.

Wikipedia

General programming but often used in
  - GUI
  - Networking
Java

b = 1
c = 2
a = b + c
b = 3

what is a?

Spreadsheet

let $A$1 = 1
let $B$1 = 2
let $C$1 =$A$1 + $B$1

Now set $A$1 = 3

what is $C$1
Reactive Programming - Types

Imperative

Object-oriented

Functional

Examples

Elm - web
Rx

Microsoft
RxJS
RxJava (Netflix port of RxJS)
ReactiveCocoa
Objective-C, Swift
React
Facebook
Functional Reactive Programming (FRP)

1997 - Elliott & Hudak
Fran - reactive animations

FRP is about handling time-varying values like they were regular values.

FRP is a declarative way of modeling systems that respond to input over time.
Higher Order FRP

Elliott & Hudak’s work

Time is a first-class citizen

Modeled time as continuous

Synchronous

Has some practical limitations
First-Order FRP

Elm - http://elm-lang.org

the best of functional programming in your browser

jump : Keys -> Model -> Model
jump keys mario =
  if keys.y > 0 && mario.vy == 0
  then { mario | vy <- 6.0 }
  else mario

Event driven
Synchronous or Asynchronous
Asynchronous Data Flow

Reactive Extension (Rx)
  RxJS
  RxJava (Netflix)
ReactiveCocoa
Bacon.js

Event Driven

Asynchronous only

Netflix use RxJava, RxJS
  Network traffic
  Reactive API backend services
  GUI
RxJava & Clojure

Clojure Reactive Programming
Borges, March 2015

Covers Rx programming in Clojure

On-line from SDSU library

Chapter 1 - history of FRP
Source for previous slides
React - Facebook

React - Javascript
First release 2013

React Native - iOS & Android

One-way data flow

Virtual DOM

Server-side rendering (JavaScript isomorphism)
Facebook, Netflix, PayPal
React & Clojure

Om
   First release on Github Jan 2014
   Om.next - coming soon

Reagent
   Simpler than Om
   First release Dec 2013

Quiescent
   First release Feb 2014
Reagent Resources

https://github.com/reagent-project/reagent
  Github repository

http://reagent-project.github.io/
  Short tutorial

https://github.com/reagent-project/reagent-cookbook
  Examples
To Start a Reagent Project

lein new reagent projectname
Live Development Updates

Figwheel

  lein figwheel

Devcard

  lein figwheel devcards