

# Activity: Paper Prototype

- Prototyping
  - Layout (canvas, box)
  - Content (text, image)
  - Control (buttons, scrollbar)
  - Integration (making sure everything fit together)
- Testing Phase
  - Computer (simulate UI behaviors)
  - Facilitator (brief and guide the users)
  - Observer (write down the difficulties users encounter)
  - User (test a prototype of another applications)

# **Lecture 8:**

# **UI Architecture**

February 24

- **Layout**
- **Events**
- **MVC**

# Virtual STAMP Food Court

(creator: Tom Yeh)



### Order

Name

  
 Pickup Only  
Delievery location




### Menu

McDonald

-  Burger \$20
-  Coke \$10



### Menu

Sbarro


-  Pasta \$20
-  Pizza \$15
-  Coke \$5

### Menu

Panda

-  Beef \$20
-  Chicken \$10

### Details



Name: Burger  
Price: 20  
Restaurant: McDonald

### Cart

Coke \$10	McDonald	<input type="button" value="remove"/>
Pasta \$20	Sbarro	<input type="button" value="remove"/>
Pizza \$15	Sbarro	<input type="button" value="remove"/>
Burger \$20	McDonald	<input type="button" value="remove"/>

# Layout

# Defining UI layout

- Procedural
- Declarative
- Form designer

# Procedural layout syntax

- Examples:
  - Java Swing
  - ActionScript

```
var btn1:Button = new Button();  
btn1.label = "Button 1";  
var btn2:Button = new Button();  
btn2.label = "Button 2";  
var btn3:Button = new Button();  
btn3.label = "Button 3";
```



# Declarative layout syntax

- Examples:
  - HTML
  - MXML

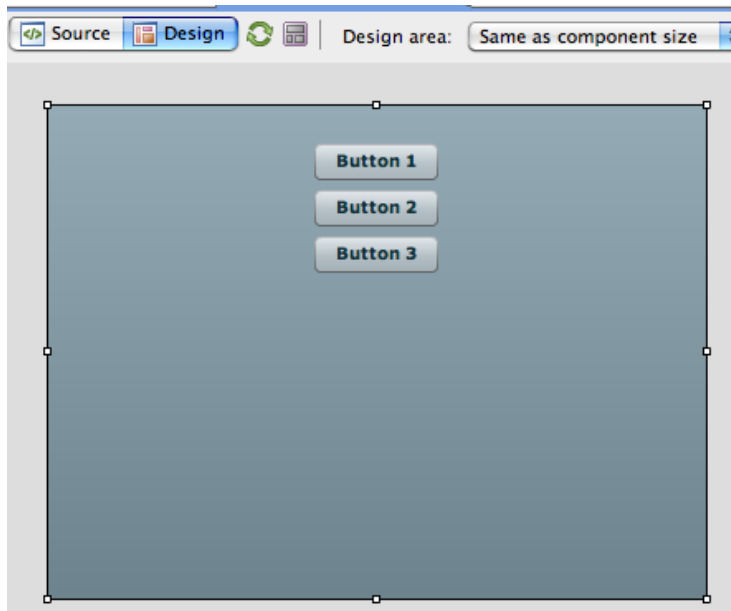
```
<mx:Button label="Button 1" />  
<mx:Button label="Button 2" />  
<mx:Button label="Button 3" />
```





# Form designer

- Front page
- Flex builder design view

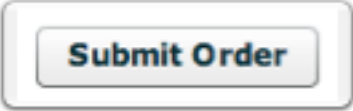




# Events

# Event-based programming

- Each user action generates an event
  - E.g., clicks on a button, types in a text field
- A change in system status can also generate an event
  - E.g., a download finishes
- The system responds to each event
  - E.g., displays some text, plays a sound clip, saves some data

# Event loop pattern

```
while True:  
    if the user clicks on X  
        if (X == ):  
            Submit()  
        else if (X == ):  
            Pickup()  
        else if (X == ):  
            Select()
```

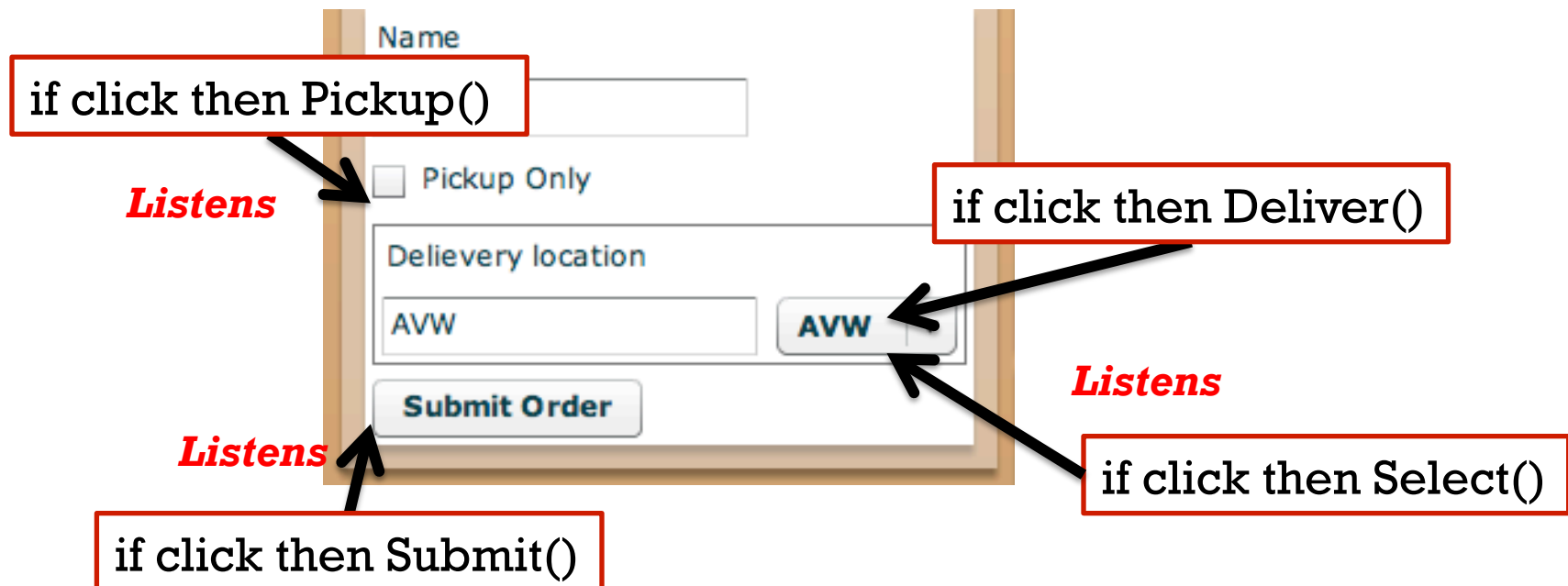
# Event listener pattern

A handler can register to listen for a particular event.



# Event listener pattern

Multiple handlers can listen for the same event



# Flex example

```
<mx:CheckBox ... click="Pickup" />
```

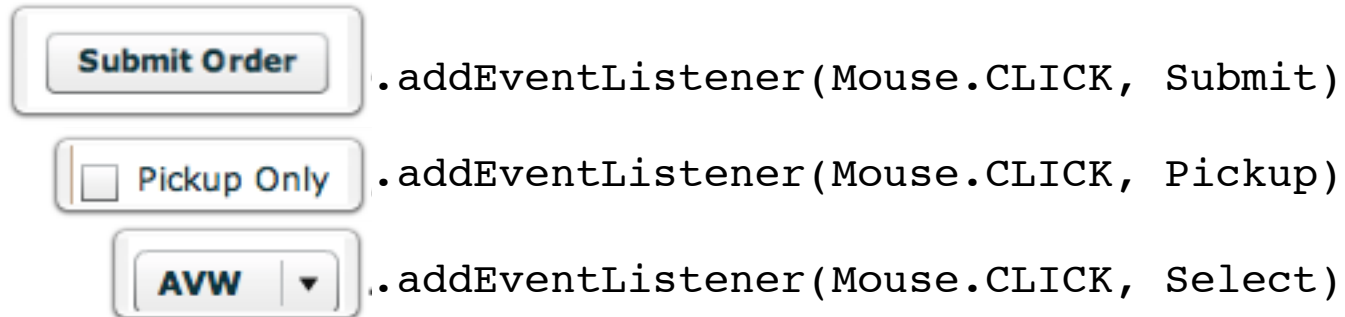
...

```
<mx:DropDown ... click="Select" />
```

...

```
<mx:Button ... click="Submit" />
```

# ActionScript example





# Component naming

- By ID

```
<mx:CheckBox id="pickupCheckBox"  
  click="Pickup" />
```

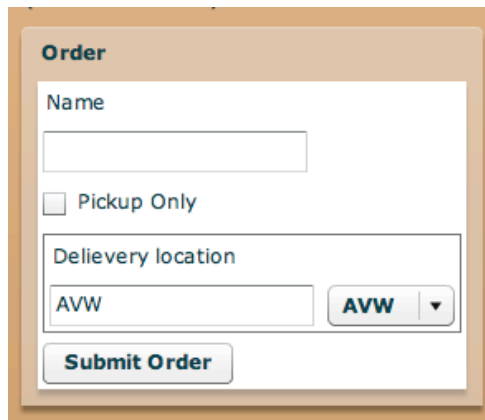
 .addEventListener(MouseEvent.CLICK, Pickup)



**pickupCheckBox**.addEventListener(MouseEvent.CLICK, Pickup)

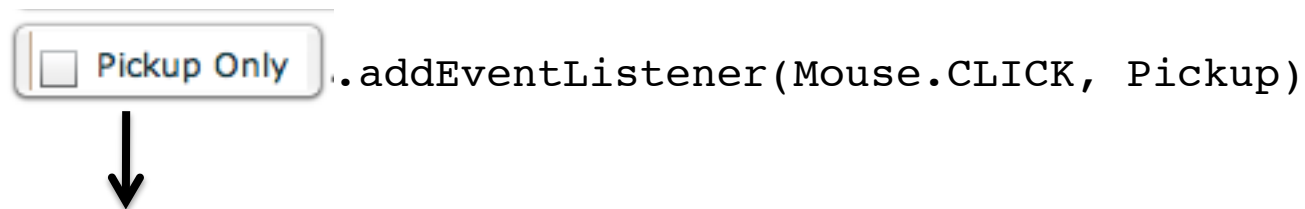
# Component naming

- By relationship



The screenshot shows a web form titled "Order". It contains a text input field for "Name", a checkbox labeled "Pickup Only", a text input field for "Delivery location" with a dropdown menu showing "AVW", and a "Submit Order" button.

```
<mx:Panel id="orderPanel" >  
...  
  <mx:CheckBox id="pickupCheckBox"  
    click="Pickup" />  
...  
</mx:Panel>
```



`Pickup Only`.addEventListener(MouseEvent.CLICK, Pickup)

```
orderPanel.children[2].addEventListener(MouseEvent.CLICK, Pickup)
```

# MVC

# Model-View-Controller pattern

- Origin: Smalltalk 80 interface
- Separate frontend and backend
  - Frontend: view and controller
  - Backend: model
- Separate responsibilities
  - Model: data
  - Controller: input
  - View: output

# Decoupling

- **Multiple models share the same view**
  - As long as the model implements the same interface
- **Multiple views exhibit the same model**
- **Multiple controllers manipulate the same model or different models**
  
- **Changes can be made independently**

# Model

- Responsible for data
- Maintain applications states
- Save mutable application data
- Notify view and controller on changes
- Backed by databases (e.g., SQL)

# View

- Responsible for output
- Fetch data from the model
- Draw data on the screen
- Listens for changes in the model and update the screen

# Controller

- Responsible for input
- Receive and process input events
  - Local keyboard, mouse events
  - Web requests
- Instruct the model to change



# MVC at multiple scales

- Widget
- Component (multiple widgets)
- Desktop GUI software
- Web application
  
- Paper prototype

# Widget

- Self-contained MVC
- E.g., Text-field
  - View: rendered text field
  - Model: string
  - Controller: handler for key events

# Web Application

- View: web pages
- Model: database tables
- Controller: CGI scripts
  
- Popular frameworks:
  - PHP: HTML, SQL, PHP scripts
  - Java: JSP, JDO, Java Servlets
  - RAILS: RHTML, SQL, Ruby scripts

# Paper prototype

E.g., To-do list

- View: paper UI
- Controller: simulated by hands
- Model: memorized in the head

# CMSC 434

- **View: Flex**
  - View: Flex components (I1)
  - Model: ActionScript objects, XML (I2, I3)
  - Controller: ActionScript (I2)
- **Model: Java Data Objects (JDO) (I4)**
- **Controller: Java Servlets (I4)**
  - Hosted by Google App Engine