# Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

# Today's Lecture

- JavaFX

- Event Handling
- Model View Controller

# JavaFX – Button/Event Handling

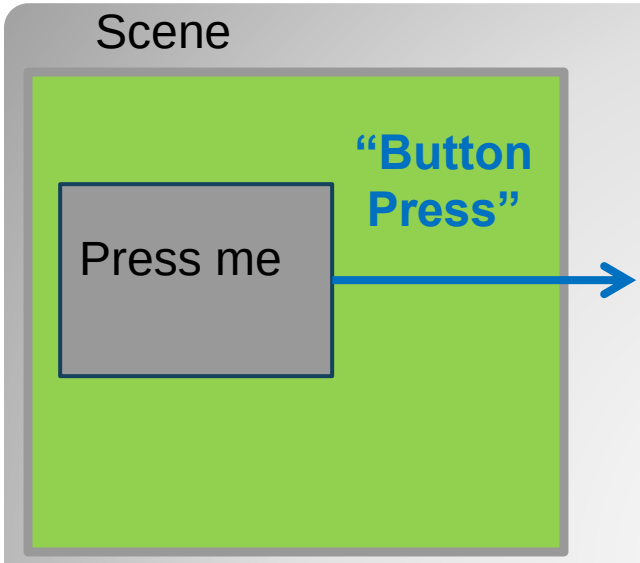
- A window with a Label was interesting but it didn't do a thing!
- Now let's put a button inside the window.
- To use a button in a window you need to do the following:

1. Create a button instance and associate it with the window.
2. Add an event handler for the button.

Event: Something that occurs

- Pressing the mouse button is an event.
- Pressing a key is an event.
- To make the program do something when the button is pressed we must "handle" the event of the button being pressed.

# JavaFX – Button/Event Handling




**When the button is pressed the handle method gets called:**

```
@FXML
protected void handleButtonAction(ActionEvent event) {
        System.out.println("button pressed");
}
```

The handleButtonAction method contains code that should run when the button is pressed.

Where does the event handler go? Which class is it in?

# JavaFX- EventHandler

# Model View Controller (MVC)

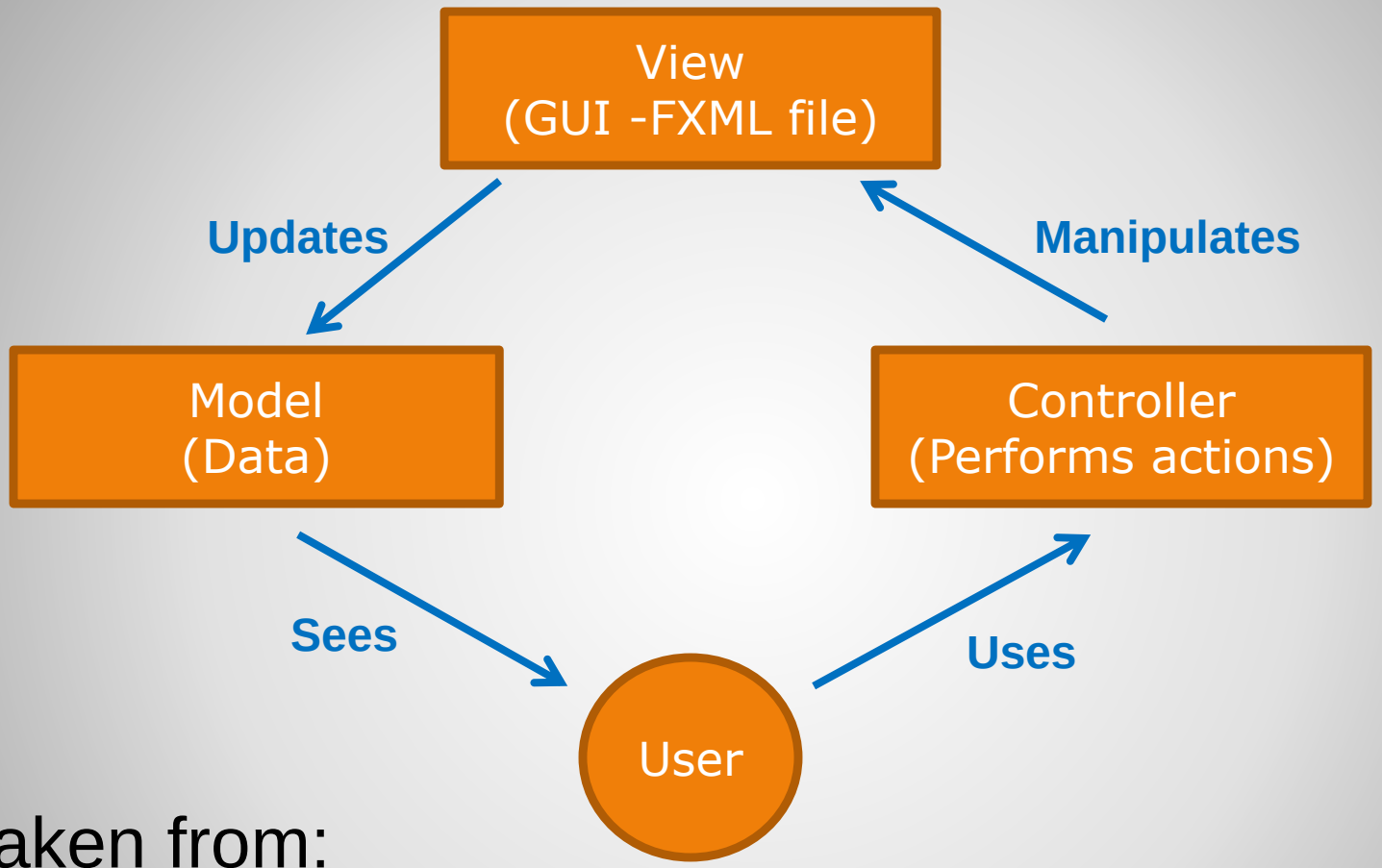
- MVC Pattern stands for Model-View-Controller Pattern.
- Separates an application's logic and its presentation.
- This separation is good because you can make changes to the view of the data and not effect the logic.

- **BEST SEPARATION**
  - Use only FXML for the display.
  - Use Java for the business logic.

- Some of above taken from:

http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

# Model View Controller Overview




Taken from:

http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

# Model View Controller Overview

- **Model** - Model represents an object or Java POJO carrying data. It can also have logic to update controller if its data changes.
- **View** - View represents the visualization of the data that model contains.
- **Controller** - Controller acts on both model and view. It controls the data flow into the model object and updates the view whenever data changes. It keeps the view and the model separate.

- Above taken from:

http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

Note: POJO stands for Plain Old Java Object

# Model View Controller Overview

- **Model** – Java classes that store data.
- **View** – FXML code. Defines the application display.
- **Controller** – Java classes that are responsible for putting data into the view and into the model.

# FXML and MVC

**MVC in Java with FXML**

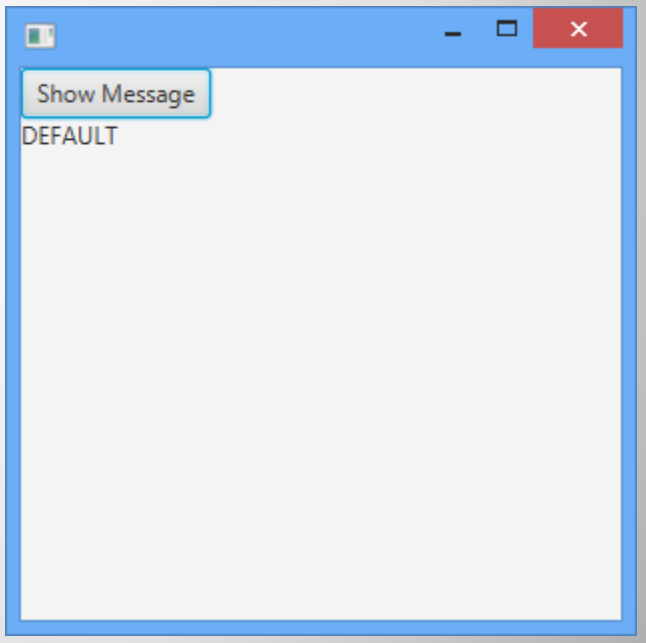
1. **FXML (the view)** – Define the view GUI using FXML code (.fxml files).
2. **Controller class** - Write a Controller class in Java that will handle events and associate its member variables with FXML controls.
3. **Model class** – Write Java classes for the model. Declare member variables of the model classes in the controller class. This holds the application's data.

# MVC in Java with FXML

- FXML code (sampleui.fxml):

**Full name of controller class (includes package)**

```
<VBox
    xmlns:fx="http://javafx.com/fxml"
    fx:controller="bcs345.practice.javafx.fxml.MyController"
    >
    <Button
        text="Show Message"
        onAction="#handleButtonAction"
        fx:id="messageButton"
        />
    <Label
        text = "DEFAULT"
        fx:id="messageOutputLabel"
        />
</VBox>
```




# FXML - View (MVC)

- FXML code (sampleui.fxml):

```
<VBox
    xmlns:fx="http://javafx.com/fxml"
    fx:controller="bcs345.practice.javafx.fxml.MyController"
    >
    <Button
        text="Show Message"
        onAction="#handleButtonAction"
        fx:id="messageButton"
        />
    <Label
        text = "DEFAULT"
        fx:id="messageOutputLabel"
        />
</VBox>
```

**This is the FULL name of the Java controller class (on next slide). It should use the package name FROM YOUR PROJECT.**

**Name of the method on the controller that should be called when button is pressed (must have a # prefix)**

**Id of the button. Needs to have the same exact name as the Button member on the controller**

**id of the label. Needs to have the same exact name as the Label member on the controller**

# FXML - View (MVC)

- FXML controller in Java (MyController.java) :

```
Package bcs345.practice.javafx.fxml;
public class MyController {

  @FXML
  private Button messageButton;

  @FXML
  private Label messageOutputLabel;

  @FXML
  protected void handleButtonAction(ActionEvent event) {
      System.out.println("Button pressed");
  }
}
```

Controller class definition

Id for FXML attribute must match the member variable name for the button (creates an association between the two)

Id for FXML attribute must match the member variable name for the label

FXML attribute makes it known that this will be used by FXML markup

Button event handler. The onAction attribute of the button definition in FXML has the name of this method

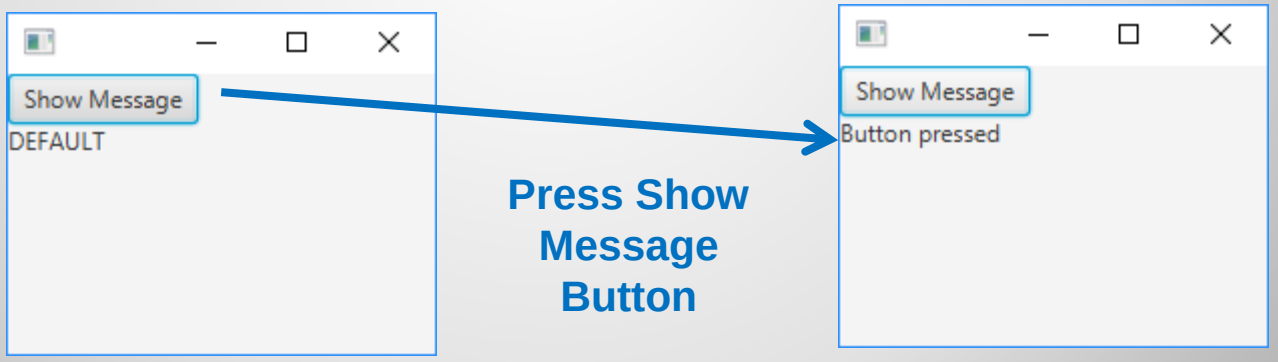
Displays the string “Button pressed " in the console window

# FXML – Controller (MVC)

# Change Label in Code

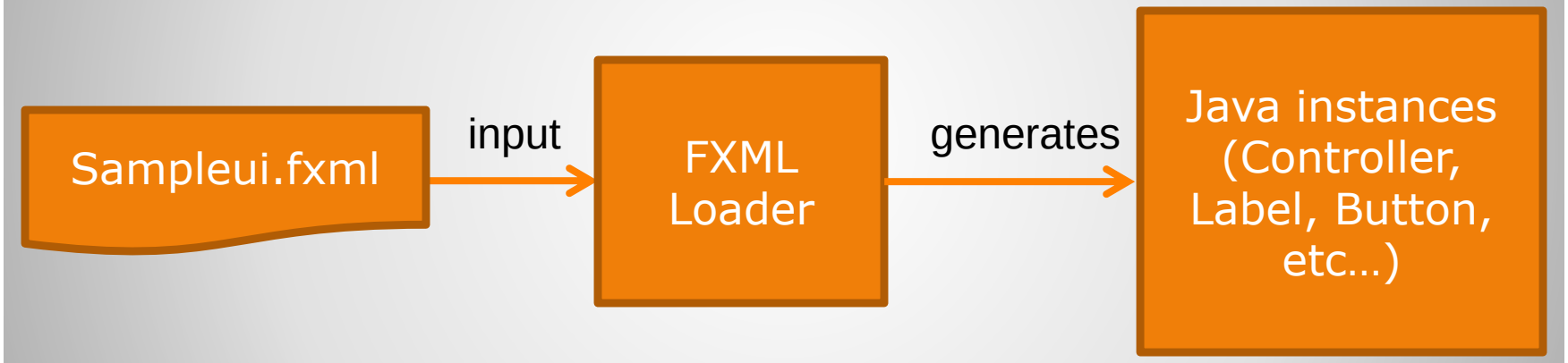
- When the previous application runs and the button is pressed it prints a message in the console window.
- We can update the code so that it also changes the contents of the label.
- Update the handleButtonAction method in the controller:

```
@FXML
protected void handleButtonAction(ActionEvent event) {
   System.out.println("Button pressed");
   messageOutputLabel.setText("Button pressed");
}
```



- The FXML loader generates instances for the Java equivalent of the FXML code (creates the scene graph and returns the root control of the scene graph).

- The FXML loader also creates the controller instance for you (calls new for the controller in the background).




**FXML Code**

```
<Label text = "DEFAULT"
fx:id="messageOutputLabel“ />
```

**Java Code That Gets Created Behind the Scenes**

```
messageOutputLabel = new Label(“DEFAULT”);

// Also calls new on MyController
```

# FXML Loader

- We did **NOT** add code to create an instance of MyController (no call to new).
- The **MyController** instance is created behind the scenes by the FXMLLoader!!!

# FXML – Controller (MVC)

- The previous simple examples did not use a model.
- We could add a model to the code.
- The model would be placed in the controller.
- Assume that we already wrote a Person class.

```
public class MyController {
   private Person[] person = new Person[100];

   @FXML
   private Button messageButton;
   @FXML
   private Label messageOutputLabel;
   @FXML
   protected void handleButtonAction(ActionEvent event) {
       // Code that uses the person array (the model) goes here...
   }
}
```

**Model**

**The person member is the model in this example.**

**You could have multiple members if you want.**

# Model (MVC)

# Get Controller Instance

- The instance load method version has the benefit of being able to retrieve the controller instance (cannot do this using the static load method).
- Use this if you need to directly access the controller.

- Example (get the controller):

```
// Create instance of FXMLLoader
FXMLLoader loader=new FXMLLoader(getClass().getResource("sampleui.fxml"));

// Call load on the loader
root = loader.load();

// Get the controller from the loader instance
MyController myController;
myController = loader.<MyController>getController();
```

**Creates controller instance behind the scenes**

**Get controller instance**

**Controller initialize()**

- You can add an initialize method to your controller class.
- The initialize method of the controller is useful for GUI initialization code.
- initialize() gets called automatically AFTER the constructor is called and AFTER all @FXML variables have been initialized.
- The constructor cannot be used for GUI initialization code because the @FXML variables have not been initialized yet when it runs

```
@FXML
public void initialize() {

   // Your GUI initialization code goes here…

}
```

# Controller initialize()

# End of Slides