CSC40232: SOFTWARE ENGINEERING

Professor: Jane Cleland-Huang
Lecture 4: Getting Started with Java FX
Wednesday, January 30th and February 1st
sarec.nd.edu/courses/SE2017

Resources

1. Introducing JavaFX 8 Programming, by Herbert Schildt, Oracle Press

https://www.eclipse.org/efxclipse/install.html#for-the-lazy

Some mac users complained on Stack Overflow that they had problems – and reinstalled the all-in-one solution (other mac users had no problem at all).
http://efxclipse.bestsolution.at/install.html#all-in-one
User Interface with Java

- **AWT**: Abstract Window Toolkit – provides only rudimentary support for GUI programming. Translates visual components into platform-specific equivalents. Different apps may have different look-and-feel.

- **Swing** (1997): Included in Java Foundation Classes.
  - Lightweight components i.e. written in java and do not rely on individual platforms.
  - Pluggable look-and-feel.
  - Supports MVC (but combines VC)

  More on MVC later in the course.

JavaFX

- A collection of classes and interfaces that define Java’s GUI.

- Can be used to create GUIs for PCs, Tablets, Web Applications, and mobile Apps.

- Provides a diverse set of controls (e.g. buttons, scroll panes, text fields, trees, tables etc)

- Supports animation.

- Streamlines creation of an app by simplifying the management of its GUI elements and the application’s deployment.
There are numerous javafx packages. Four main ones include:

- javafx.application
- javafx.stage
- javafx.scene
- javafx.scene.layout

**Today:**
- JavaFX Basics: Application class, Stage class, Scene class, Layout

**Wednesday:**
- Handling Events: Controls, Keys, Mouse

---

**Setting the Stage**

The primary JavaFX metaphor is the **stage**. A stage contains a **scene**.

The stage is a container for scenes.

A scene is a container for items that comprise the scene.

All JavaFX applications automatically get access to the **primary stage** – this is provided by the runtime system when a JavaFX application is started.

To set a scene – add elements to an instance of the Scene and then set the scene onto the stage.
Nodes and Scene Graphs

- The elements of a scene are called nodes. (e.g. a button)
  - Nodes can consist of groups of nodes
  - Nodes can have children
  - The collection of all nodes in a scene is referred to as a scene graph – and comprises a tree.
  - A scene-graph has one root node.

Layouts
A JavaFX application must extend Application class. Application defines three life-cycle methods which you can override:

1. init() – performs initializations but cannot be used to build a scene or stage!
2. start() – called after init and CAN be used to create a scene or stage. Start() is abstract and therefore MUST be overridden by your application!
3. stop() – called when the application terminates

Include this – although only needed in some cases.

Write compilable code for a HelloWorld JavaFX application.

Breaking it down:

1. Create a label – or some other kind of control to place into the scene

2. Add the label to the scene’s graph.
   Call getChildren() on the root node of the scene graph. It returns a list of the childnodes in the form of an ObservableList<Node>.
   Add the new control to the list.

An ObservableList is packaged in javafx.collections – inherits java.util.List and allows listeners to track changes when they occur.
Multiple Controls/Layout

```
Label myLabel = new Label("Hello World");
Label myLabel2 = new Label("Bonjour le monde");
Label myLabel3 = new Label("Hallo Welt");

AnchorPane root = new AnchorPane();
ObservableList<Node> obsList = root.getChildren();
obList.add(myLabel);
obList.add(myLabel2);
obList.add(myLabel3);
```

This works too!
```
ObservableList<Node> obsList = root.getChildren();
obList.addAll(myLabel, myLabel2, myLabel3);
```

Improve the following line of code.
```
FlowPane root = new FlowPane();
```

Rewrite the two lines of code shown above without creating obsList as a named variable.

YouTube Video: First JavaFX Application  https://www.youtube.com/watch?v=vMeG-lvIqQ8

Event Handling

Many JavaFX controls need to generate events in response to user input. E.g. buttons, check boxes, lists etc.

JavaFX uses the delegation event model.
Delegation Event Model

1. An event handler registers itself with the control.

2. The component (control) fires the event object (e.g. when user clicks on the button). It generates an ActionEvent object.

3. The ActionEvent object is passed to the event listener – which responds by executing appropriate code.

The EventHandler Interface

- Events are handled by implementing the EventHandler Interface:
  ```java
  Interface EventHandler<T extends event> {
    where T specifies the Type of event.
    T extends Event class
    T defines one method called handle()
  }
  ```

- Specify an event handler in one of two ways:
  ```
  addEventHandler() (defined by the Node class)  
  removeEventHandler()  
  Use the convenience method `setOn` (property)
  ```

- Events are processed by the event dispatch chain which is a path from the top element in the scene graph to the target of the event.
Anonymous Classes

Before we look at event handlers – let’s look at the notion of an anonymous class.

class ProgrammerInterview {
    public void read() {
        System.out.println("Programmer Interview!");
    }
}

class Website {
    /* This creates an anonymous inner class: */
    ProgrammerInterview pInstance = new ProgrammerInterview() {
        public void read() {
            System.out.println("anonymous ProgrammerInterview");
        }
    };
}

This is a typical class. It contains one method – read().

An instance of an anonymous class is being created here.

We define a method at the same time as creating an instance of the class. We are actually subclassing ProgrammerInterview and overriding read()!

Event handlers are often defined using anonymous classes.

Button Events

Button btnFirst = new Button("First");
Label myLabel = new Label("No button pressed");
btnFirst.addActionListener(new ActionListener() {
    @Override public void actionPerformed(ActionEvent e) {
        myLabel.setText("First button was pressed.");
    }
});

Try to answer these questions by looking at the code:

1. What is the control that initiates events?
2. What event object does it generate?
3. What is the name of the event handler instance?
4. How is the event handler instantiated?
5. What does this event handler instance actually do?
Another Control: CheckBox

```java
final String[] names = new String[] {"Burger", "Coke", "Fries"};
fTnal CheckBox[] cb = new CheckBox[names.length];
TextArea textarea = new TextArea("Your current order:\n");
textarea.setMinSize(150, 125);
textarea.setMaxSize(200, 125);

final VBox vbox = new VBox();
for (int i = 0; i < names.length; i++) {
    final CheckBox cb = new CheckBox(names[i]);
    // Add a listener for each individual checkbox item
    cb.selectedProperty().addListener((ChangeListener<Boolean>)
        (changed, oldValue, newValue) {
            if (cb.isSelected()) {
                String text = cb.getText();
                sb.append(text);
                textarea.appendText(sb.toString());
            }
    });
    vbox.getChildren().add(cb);
}
vbox.setSpacing(5);
stack.getChildren().add(vbox);
stack.getChildren().add(textarea);
```

Some initial setup. Define an array of Checkboxes and create a TextArea for displaying the order in.

For each checkbox in the array, create an event handler.

Everytime the order changes we rewrite the whole order into the TextArea.

Add Checkboxes to a Vbox (vertical box). Add Vbox and TextArea to the scene.
Another Control: CheckBox

1. What event does this new Listener listen for?

2. True or False?
   When the focus is on another control and an event is initiated, the event object will be sent to this Listener and then ignored.

3. The EventHandler for the button control was instantiated as:
   ```java
   new EventHandler<ActionEvent>()
   ```
   Why is the CheckBox event handler instantiated in a different way as:
   ```java
   new ChangeListener<Boolean>
   ```

Observer vs. Event Delegation

How do they differ and when should each one be used?
Key Events

When a key is pressed on the keyboard a **KeyEvent** is generated.

KeyEvents can be handled by instances of various classes including Node and Scene.

Node and Scene both define methods that make it easy to register an event handler for the three types of key events:

- `final void setOnKeyPressed(EventHandler<? super KeyEvent> handler)`
- `final void setOnKeyReleased(EventHandler<? super KeyEvent> handler)`
- `final void setOnKeyTyped(EventHandler<? super KeyEvent> handler)`

Key Events

**KEY_TYPED**

Call `getCharacter()` on the event to get the key-typed event.

<table>
<thead>
<tr>
<th>Key Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIGHT</td>
<td>The stand-alone RIGHT ARROW key</td>
</tr>
<tr>
<td>LEFT</td>
<td>The stand-alone LEFT ARROW key</td>
</tr>
<tr>
<td>KP_RIGHT</td>
<td>The RIGHT ARROW key on the number pad</td>
</tr>
<tr>
<td>KP_LEFT</td>
<td>The LEFT ARROW key on the number pad</td>
</tr>
<tr>
<td>F1</td>
<td>The f1 key</td>
</tr>
<tr>
<td>F10</td>
<td>The f10 key</td>
</tr>
<tr>
<td>ALT</td>
<td>The ALT key</td>
</tr>
<tr>
<td>CONTROL</td>
<td>The CTRL key</td>
</tr>
<tr>
<td>SHIFT</td>
<td>The SHIFT key</td>
</tr>
</tbody>
</table>

**KEY_PRESSED**

**KEY_RELEASED**
Example

```java
myScene.setOnKeyPressed(new EventListener<KeyEvent>() {
    @Override
    public void handle(KeyEvent ke) {
        showKey.setText("You pressed the " + ke.getKeyCode());
    }
});

myScene.setOnKeyTyped(new EventListener<KeyEvent>() {
    @Override
    public void handle(KeyEvent ke) {
        showKey.setText("You typed " + ke.getKeyCharacter());
    }
});
```

Capture other keys pressed.
Note: On my keyboard I had to press Shift-F10 and Shift-ALT to get those keys to work!!

What would happen if we didn’t include the “break” statements?

YouTube video: [https://www.youtube.com/watch?v=-9g7WLaQVIY](https://www.youtube.com/watch?v=-9g7WLaQVIY)

an aMAZing Example

But first we need to look at the code to:
- Draw a circle
- Draw a line
- Import and display an image.
Circles

- Import Color shape package and paint.Color if we want to fill or color the circle
  - import javafx.scene.paint.Color;
  - import javafx.scene.shape.Circle;
- Create an instance of the circle
  int x = 20; int y = 20; int radius = 50;
  Circle circle;
  // Create a circle with radius 50;
  circle = new Circle(20);
  circle.setCenterX(25);
  circle.setCenterY(25);
  OR: circle = new Circle(25,25,20);
- Moving the circle

Display an Image

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

Image myImage = new Image("images\myImage.png",50,50,false,false);
ImageView myImageView = new ImageView(myImage);
myImageView.setX(x);
myImageView.setY(y);
root.getChildren().add(myImageView);
Lines

 Constructors

<table>
<thead>
<tr>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line()</td>
</tr>
<tr>
<td>Creates an empty instance of Line.</td>
</tr>
<tr>
<td>Line(double startX, double startY, double endX, double endY)</td>
</tr>
<tr>
<td>Creates a new instance of Line.</td>
</tr>
</tbody>
</table>

```java
public class Line
extends Shape

This Line represents a line segment in (x,y) coordinate space. Example:
import javafx.scene.shape.*;

Line line = new Line();
line.setStartX(0.0f);
line.setStartY(0.0f);
line.setEndX(100.0f);
line.setEndY(100.0f);
}
```

Start using
https://docs.oracle.com/javase/8/javafx/api
To figure out how to construct and use java classes.

Designing the Maze Solution

- Write down a simple description of the problem.
- The application must generate a maze and add a target icon onto the target location. The seeker is placed onto the maze in the top left position. The users presses keys to move the seeker around the maze. When the target is reached, the icon changes to “won”. The game is played and displayed in a GUI.
- Underline key nouns.
- Identify candidate classes.

Maze  Seeker  Game
Class-responsibility-collaboration (CRC) cards are a brainstorming tool used in the design of object-oriented software. They were originally proposed by Ward Cunningham and Kent Beck as a teaching tool, but are also popular among expert designers and recommended by extreme programming supporters.

### Designing the Maze Solution

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maze</td>
<td>• Generate a Maze</td>
<td>• Game</td>
</tr>
<tr>
<td></td>
<td>• Build the Maze</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Specify the start point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Specify the end point</td>
<td></td>
</tr>
<tr>
<td>Seeker</td>
<td>• Move (Left, Right, Up, Down)</td>
<td>• Maze</td>
</tr>
<tr>
<td></td>
<td>• Return current position</td>
<td>• Game</td>
</tr>
<tr>
<td>Game</td>
<td>• Display and update the GUI elements of the game</td>
<td>• Maze</td>
</tr>
<tr>
<td></td>
<td>• Request new maze</td>
<td>• Seeker</td>
</tr>
<tr>
<td></td>
<td>• Manage key input</td>
<td></td>
</tr>
</tbody>
</table>

Code Review (Live in class). View on YouTube: https://www.youtube.com/watch?v=ZXa--WSCPbs