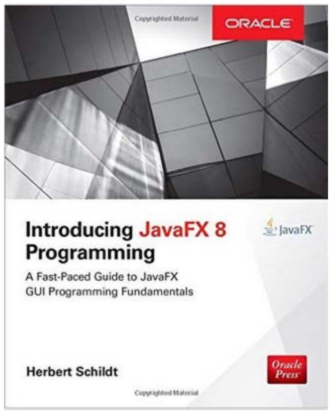
CSC40232: SOFTWARE ENGINEERING

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

Department of Computer Science and Engineering

Resources


Copyrighted Material
ORACLE
Introducing JavaFX 8 Programming
A Fast-Paced Guide to JavaFX GUI Programming Fundamentals
JavaFX
Herbert Schildt
Oracle Press
Copyrighted Material

1. Introducing JavaFX 8 Programming, by Herbert Schildt, Oracle Press
2. JavaFX Rich Desktop Applications, Rajmahendra Hegde, Presentation at Conference on Software Development. URL: http://www.slideshare.net/rajmahendra/javafx-2-rich-desktop-platform

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

2

User Interface with Java

- **<u>AWT</u>**: 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.
- **<u>Swing</u>** (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.

3

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.

4

JavaFX

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

5

Setting the Stage

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

The stage is a container for scenes.

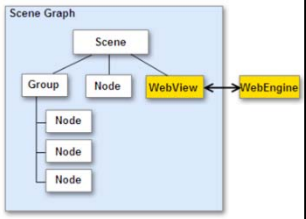
A scene is a container for items that comprise the scene.

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.

6

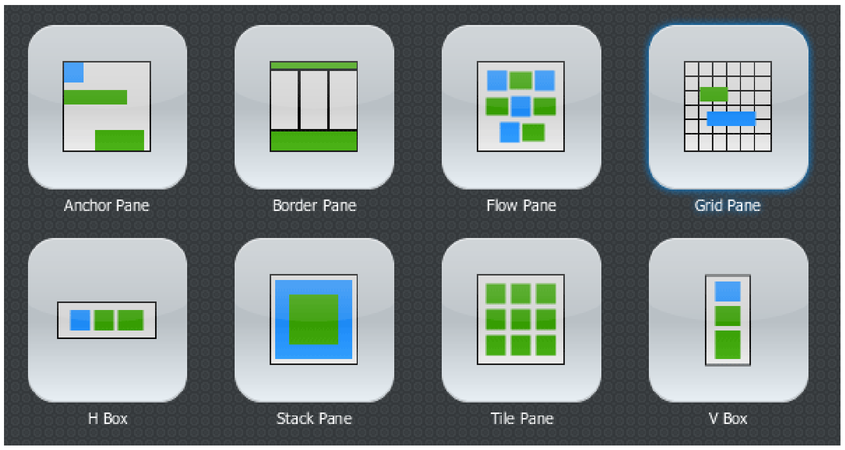
Nodes and Scene Graphs



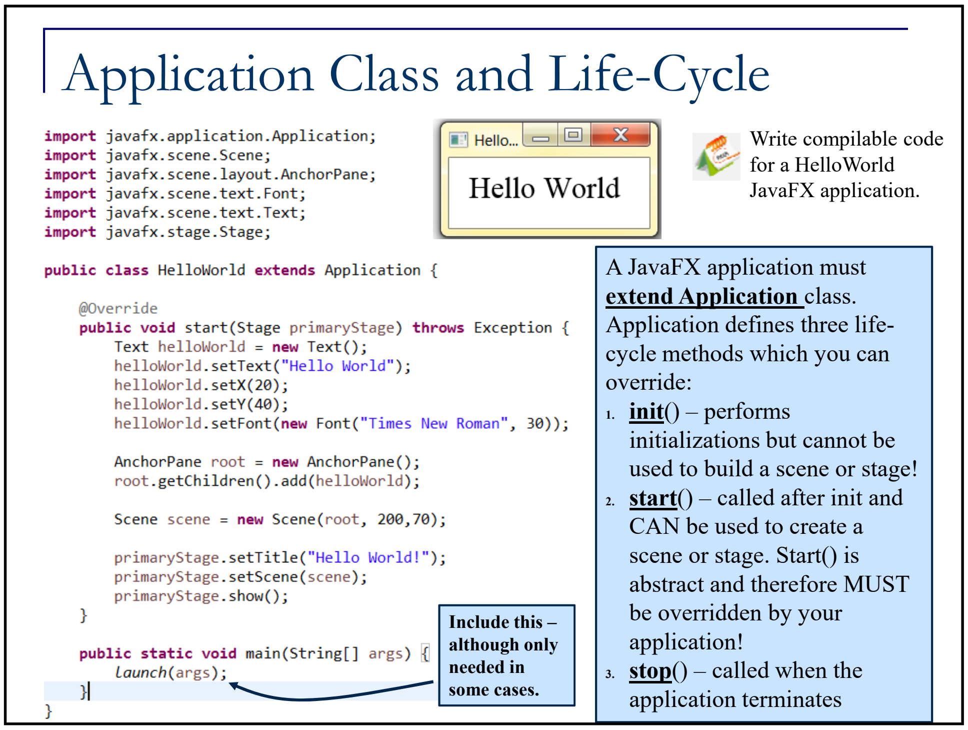
- The elements of a scene are called **nodes**. (e.g. a button)
  - Nodes can consist of groups of nodes
  - Nodes can have children
  - The collection of all nodes in a scene is referred to as a scene graph – and comprises a tree.
  - A scene-graph has one root node.

7

Layouts



8



Application Class and Life-Cycle

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Text helloWorld = new Text();
        helloWorld.setText("Hello World");
        helloWorld.setX(20);
        helloWorld.setY(40);
        helloWorld.setFont(new Font("Times New Roman", 30));

        AnchorPane root = new AnchorPane();
        root.getChildren().add(helloWorld);

        Scene scene = new Scene(root, 200,70);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Hello...

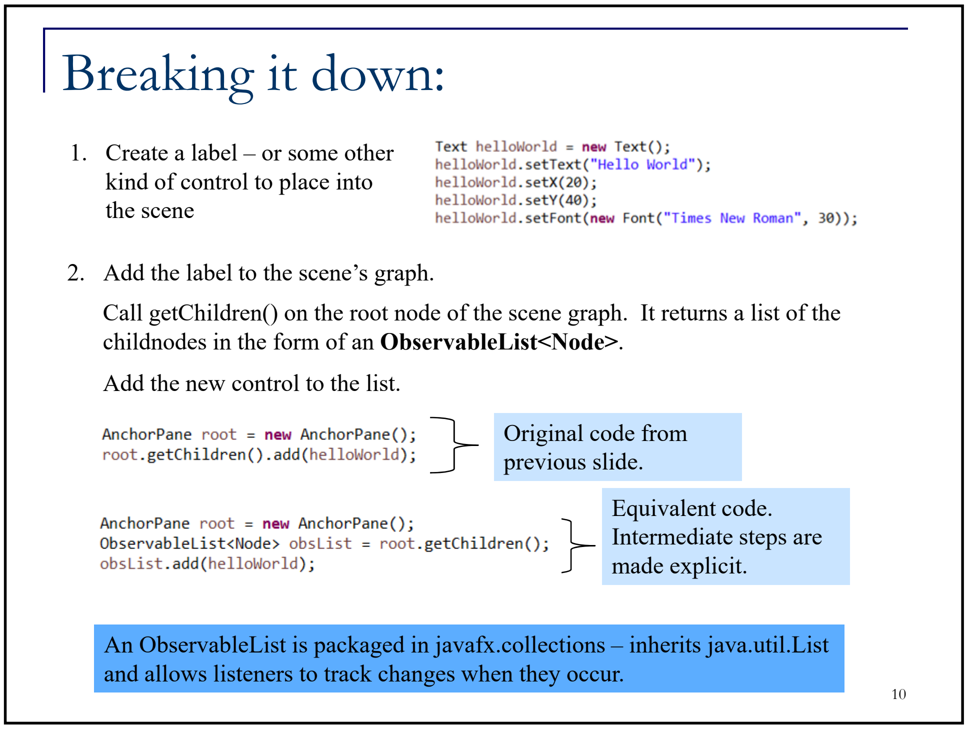
Hello World

Write compilable code for a HelloWorld JavaFX application.

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.



Breaking it down:

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

```java
Text helloWorld = new Text();
helloWorld.setText("Hello World");
helloWorld.setX(20);
helloWorld.setY(40);
helloWorld.setFont(new Font("Times New Roman", 30));
```

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.

```java
AnchorPane root = new AnchorPane();
root.getChildren().add(helloWorld);
```

Original code from previous slide.

```java
AnchorPane root = new AnchorPane();
ObservableList<Node> obsList = root.getChildren();
obsList.add(helloWorld);
```

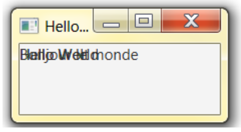
Equivalent code. Intermediate steps are made explicit.

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

10

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();
obsList.add(myLabel);
obsList.add(myLabel2);
obsList.add(myLabel3);
```



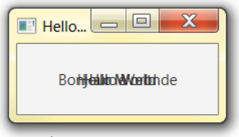
This works too!

```
ObservableList<Node> obsList = root.getChildren();
obsList.addAll(myLabel, myLabel2, myLabel3);
```

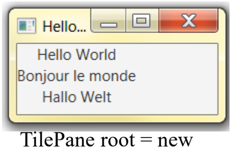
Rewrite the two lines of code shown above without creating obsList as a named variable.

Improve the following line of code. What OO Principle did you use?

```
FlowPane root = new FlowPane();
```

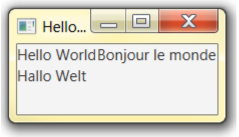


StackPane root = new StackPane()



Hello World
Bonjour le monde
Hallo Welt

TilePane root = new TilePane()



Hello World Bonjour le monde
Hallo Welt

FlowPane root = new FlowPane()

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

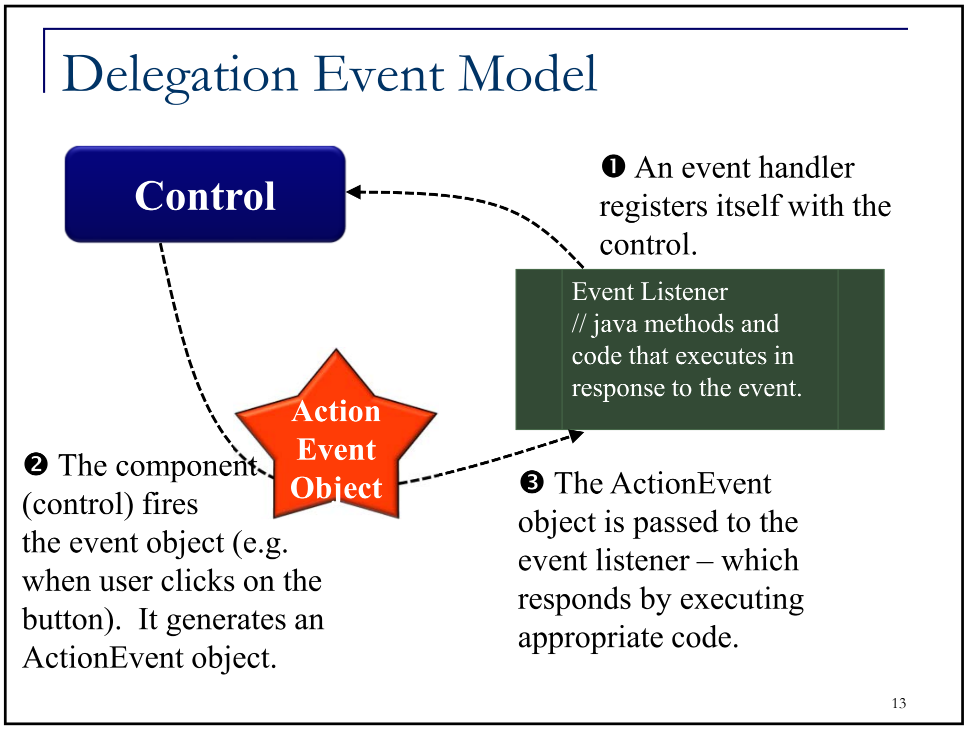
Event Handling

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.

Event Listener

12



Delegation Event Model

Control

❶ An event handler registers itself with the control.

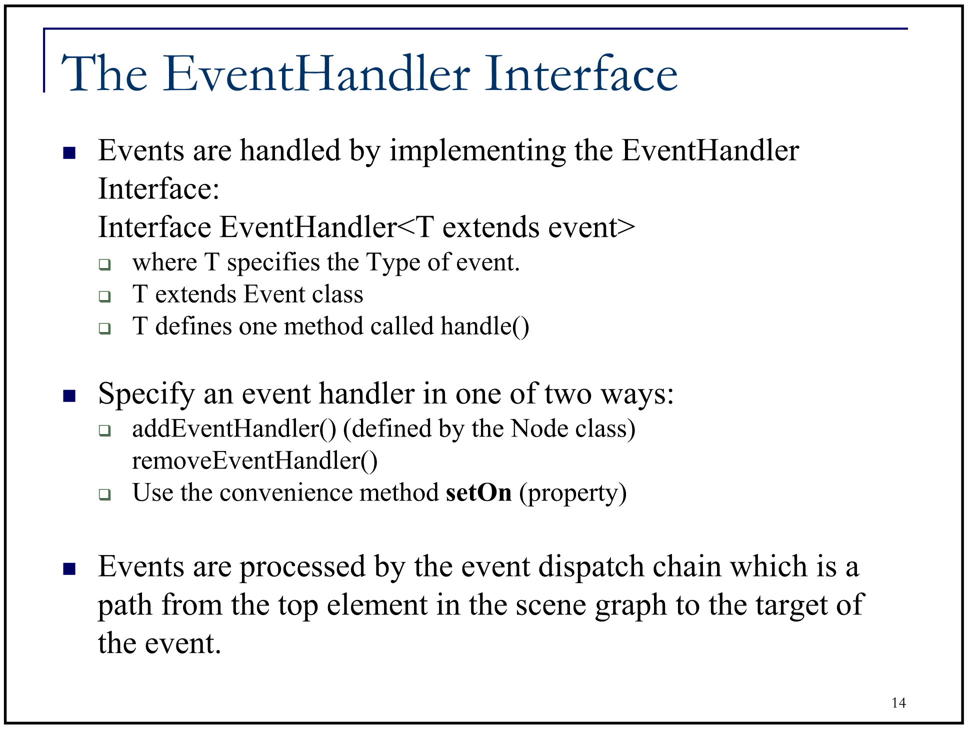
Event Listener
// java methods and code that executes in response to the event.

Action Event Object

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

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

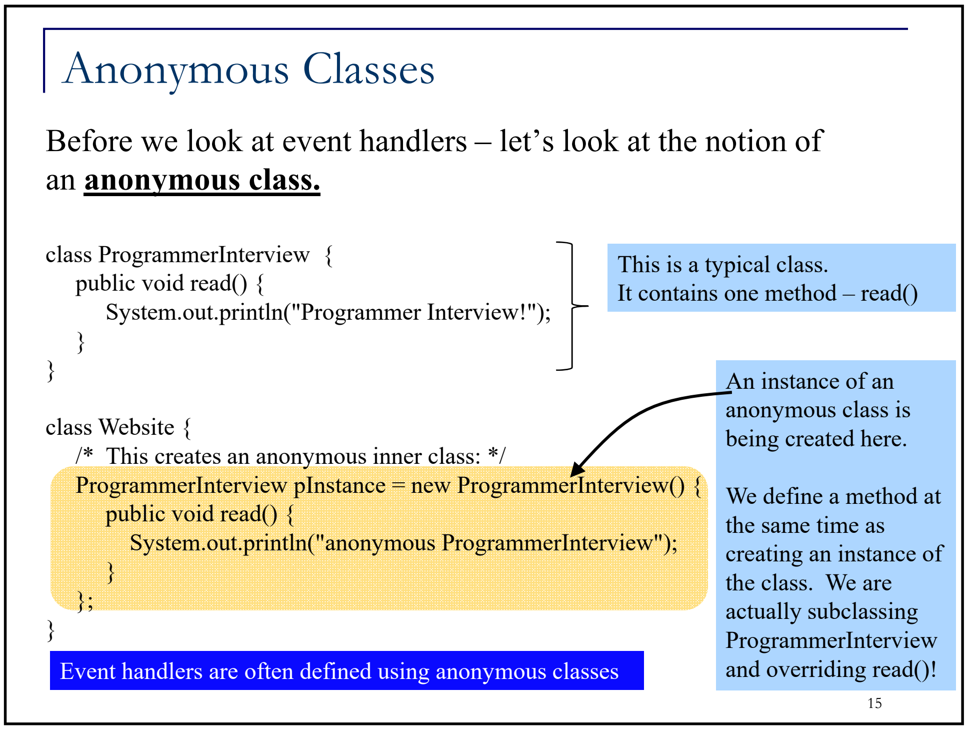
13



The EventHandler Interface

- Events are handled by implementing the EventHandler Interface:
  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.

14



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");
        }
    };
}
```

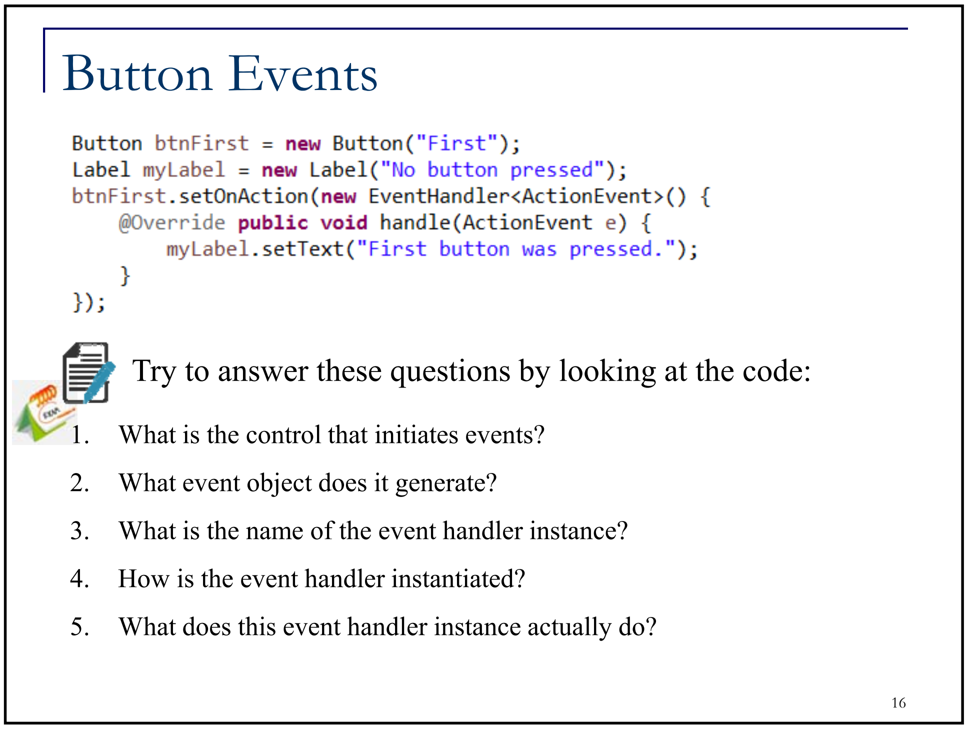
This is a typical class.
It contains one method – read()

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

15



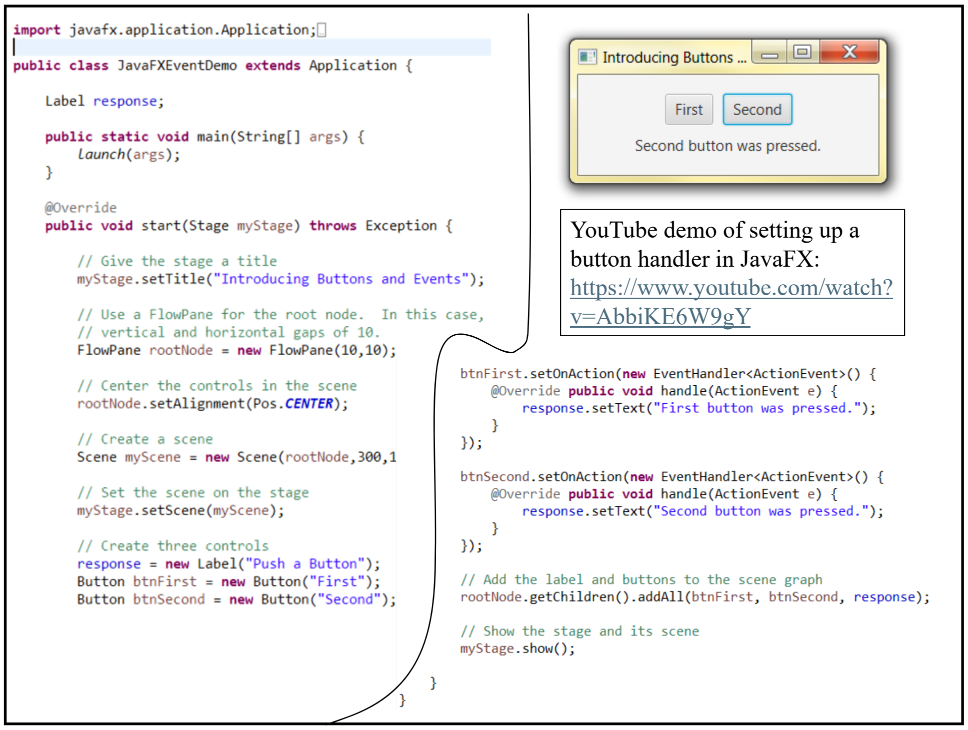
Button Events

```
Button btnFirst = new Button("First");
Label myLabel = new Label("No button pressed");
btnFirst.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(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?

16



```
import javafx.application.Application;

public class JavaFXEventDemo extends Application {

    Label response;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage myStage) throws Exception {

        // Give the stage a title
        myStage.setTitle("Introducing Buttons and Events");

        // Use a FlowPane for the root node.  In this case,
        // vertical and horizontal gaps of 10.
        FlowPane rootNode = new FlowPane(10,10);

        // Center the controls in the scene
        rootNode.setAlignment(Pos.CENTER);

        // Create a scene
        Scene myScene = new Scene(rootNode,300,1

        // Set the scene on the stage
        myStage.setScene(myScene);

        // Create three controls
        response = new Label("Push a Button");
        Button btnFirst = new Button("First");
        Button btnSecond = new Button("Second");
```

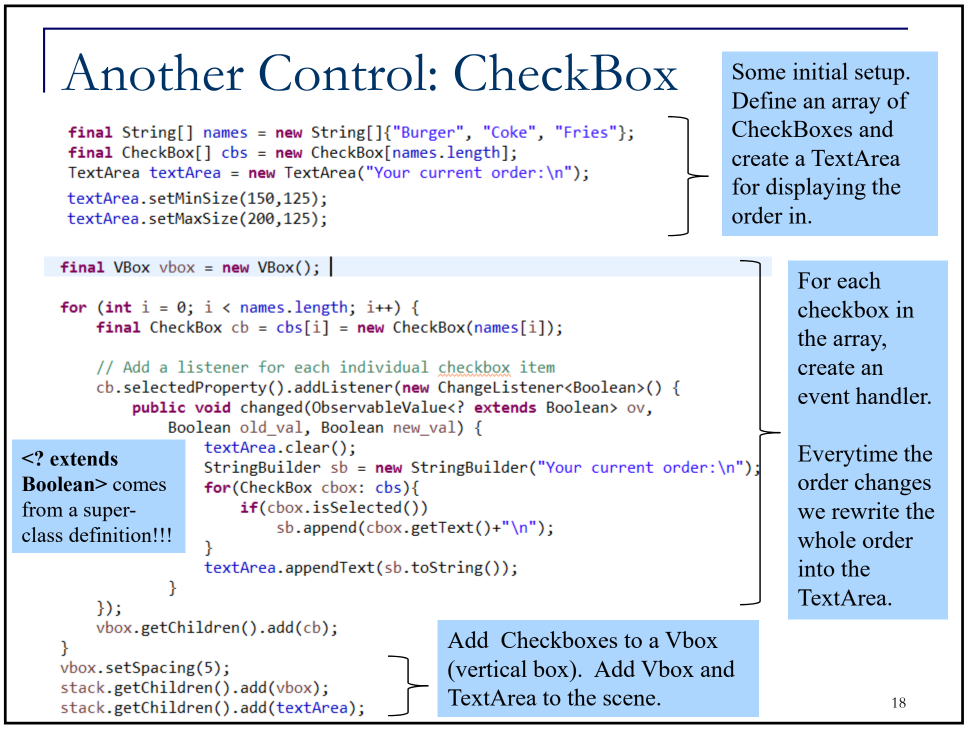
Introducing Buttons ...

First Second

Second button was pressed.

YouTube demo of setting up a button handler in JavaFX: https://www.youtube.com/watch?v=AbbiKE6W9gY

```
        btnFirst.setOnAction(new EventHandler<ActionEvent>() {
            @Override public void handle(ActionEvent e) {
                response.setText("First button was pressed.");
            }
        });

        btnSecond.setOnAction(new EventHandler<ActionEvent>() {
            @Override public void handle(ActionEvent e) {
                response.setText("Second button was pressed.");
            }
        });

        // Add the label and buttons to the scene graph
        rootNode.getChildren().addAll(btnFirst, btnSecond, response);

        // Show the stage and its scene
        myStage.show();

    }
}
```



Another Control: CheckBox

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

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

```
final VBox vbox = new VBox();

for (int i = 0; i < names.length; i++) {
    final CheckBox cb = cbs[i] = new CheckBox(names[i]);

    // Add a listener for each individual checkbox item
    cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
        public void changed(ObservableValue<? extends Boolean> ov,
            Boolean old_val, Boolean new_val) {
                textArea.clear();
                StringBuilder sb = new StringBuilder("Your current order:\n");
                for(CheckBox cbox: cbs){
                    if(cbox.isSelected())
                        sb.append(cbox.getText()+"\n");
                }
                textArea.appendText(sb.toString());
            }
    });
    vbox.getChildren().add(cb);
}
vbox.setSpacing(5);
stack.getChildren().add(vbox);
stack.getChildren().add(textArea);
```

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

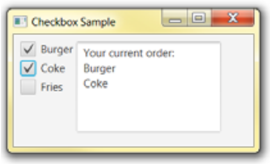
Everytime the order changes we rewrite the whole order into the TextArea.

<? extends Boolean> comes from a super-class definition!!!

Add Checkboxes to a Vbox (vertical box). Add Vbox and TextArea to the scene.

18

Another Control: CheckBox



```
Your current order:
Burger
Coke
```

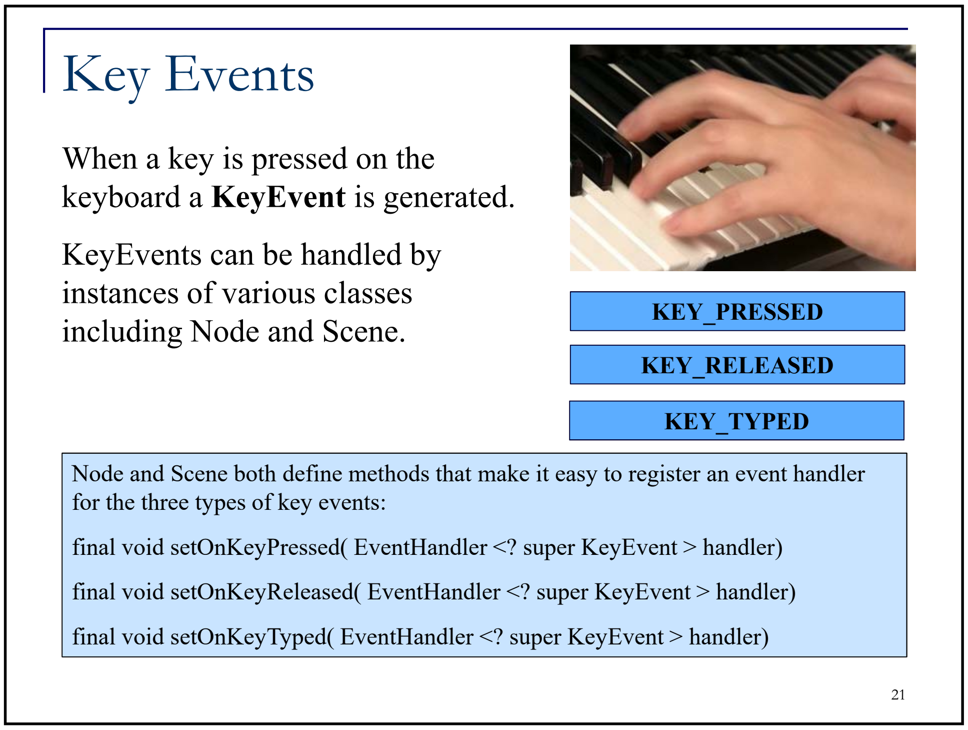
https://www.youtube.com/watch?v=FuhrwWVGO9A

```java
// Add a listener for each individual checkbox item
cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
    public void changed(ObservableValue<? extends Boolean> ov,
        Boolean old_val, Boolean new_val) {
            textArea.clear();
            StringBuilder sb = new StringBuilder("Your current order:\n");
            for(CheckBox cbox: cbs){
                if(cbox.isSelected())
                    sb.append(cbox.getText()+"\n");
            }
            textArea.appendText(sb.toString());
        }
});
```

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:
   new EventHandler<ActionEvent>()
   Why is the CheckBox event handler instantiated in a different way as:
   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.

KEY_PRESSED

KEY_RELEASED

KEY_TYPED

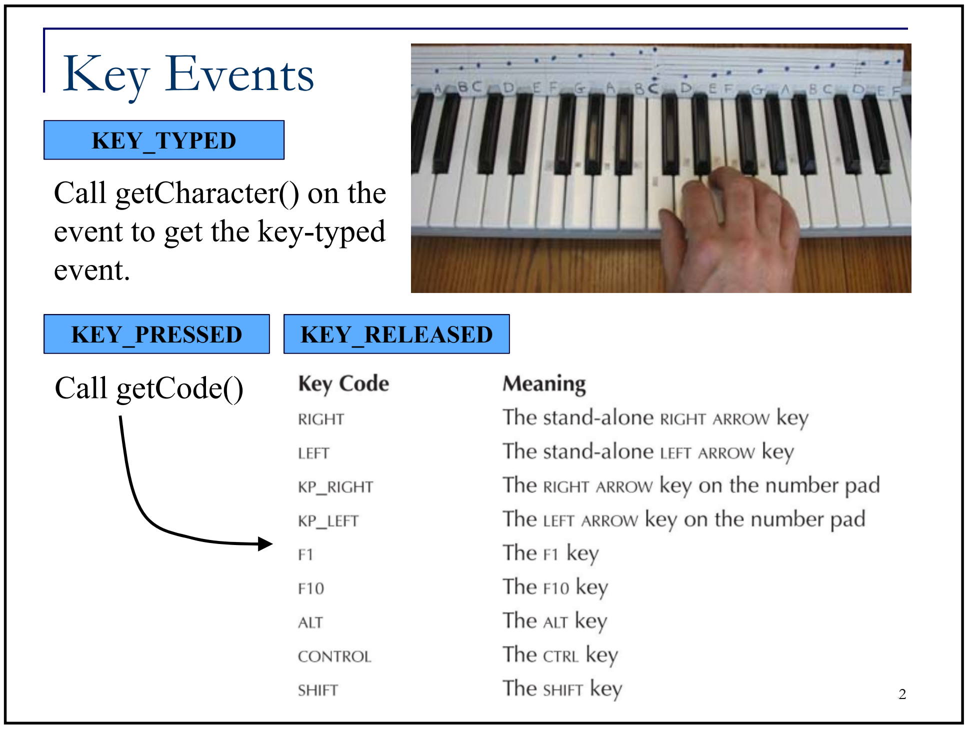
Node and Scene both define methods that make it easy to register an event handler for the three types of key events:

final void setOnKeyPressed(EventHandler <? super KeyEvent > handler)

final void setOnKeyReleased(EventHandler <? super KeyEvent > handler)

final void setOnKeyTyped(EventHandler <? super KeyEvent > handler)

21



Key Events

KEY_TYPED

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

KEY_PRESSED **KEY_RELEASED**

Call getCode()

Key Code	**Meaning**
RIGHT	The stand-alone RIGHT ARROW key
LEFT	The stand-alone LEFT ARROW key
KP_RIGHT	The RIGHT ARROW key on the number pad
KP_LEFT	The LEFT ARROW key on the number pad
F1	The F1 key
F10	The F10 key
ALT	The ALT key
CONTROL	The CTRL key
SHIFT	The SHIFT key

2

Example

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

myScene.setOnKeyPressed(new EventHandler<KeyEvent>() {
    @Override
    public void handle(KeyEvent ke) {
        switch(ke.getCode()){
        case RIGHT:
            showKey.setText("You pressed the Right Arrow");
            break;
        case LEFT:
            showKey.setText("You pressed the Left Arrow");
            break;
        case F10:
            showKey.setText("You pressed the F10 key");
            break;
        case ALT:
            showKey.setText("You pressed the ALT key");
            break;
        }
    }
});
```

Capture alpha-numeric keys typed.

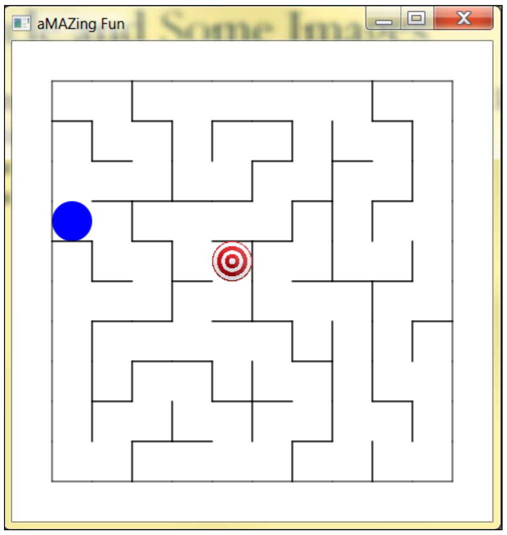
Capture other keys pressed.
Note: On my keyboard I had to press Shift-F10 and Shift-ALT to get those keys to work!!

What would happen if we didn't include the "break" statements?

YouTube video: https://www.youtube.com/watch?v=-9g7WLaQVlY

23

an aMAZing Example



But first we need to look at the code to:

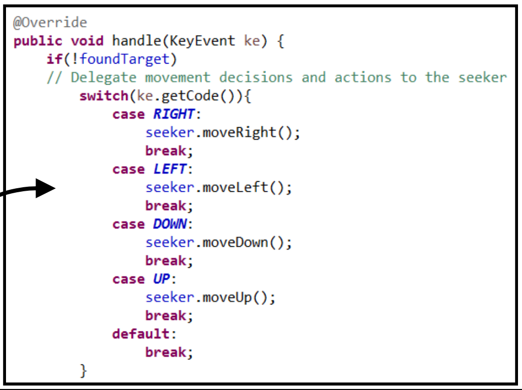
- Draw a circle
- Draw a line
- Import and display an image.

24

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



```
@Override
public void handle(KeyEvent ke) {
    if(!foundTarget)
    // Delegate movement decisions and actions to the seeker
        switch(ke.getCode()){
            case RIGHT:
                seeker.moveRight();
                break;
            case LEFT:
                seeker.moveLeft();
                break;
            case DOWN:
                seeker.moveDown();
                break;
            case UP:
                seeker.moveUp();
                break;
            default:
                break;
        }
```

Display an Image

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

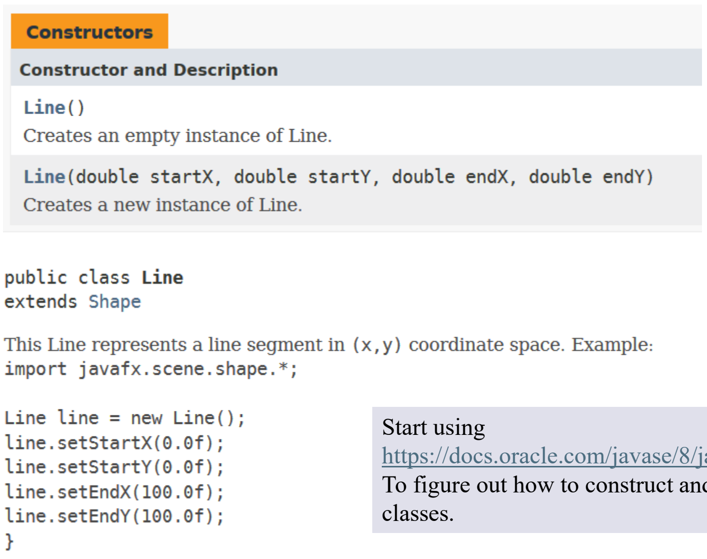
requested width
requested height
preserve ratio
Smoothing

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);

```
private void loadImages(){

    seekImage = new Image("images\\target.png",cellSize, cellSize, false, false);
    seekImageView = new ImageView(seekImage);
    seekImageView.setX(maze.getTargetPoint().x*cellSize);
    seekImageView.setY(maze.getTargetPoint().y*cellSize);

    // Load "found" target image
    foundImage = new Image("images\\win.gif",cellSize, cellSize, false, false);
    foundImageView = new ImageView(foundImage);
    foundImageView.setX(maze.getTargetPoint().x*cellSize);
    foundImageView.setY(maze.getTargetPoint().y*cellSize);

    // set current target image to "not found yet" image
    root.getChildren().add(seekImageView);
}
```

26

Lines



Constructors

Constructor and Description
`Line()` Creates an empty instance of Line.
`Line(double startX, double startY, double endX, double endY)` Creates a new instance of Line.

```
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);
}
```

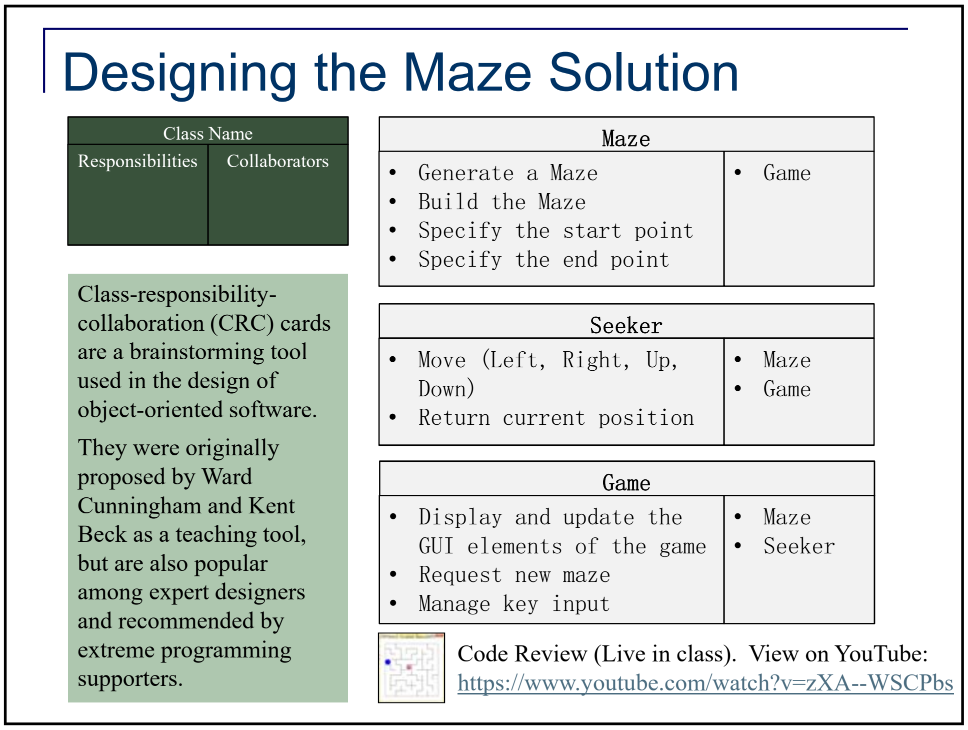
Start using
https://docs.oracle.com/javase/8/javafx/api
To figure out how to construct and use java classes.

27

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



Designing the Maze Solution

Class Name	
Responsibilities	Collaborators

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.

Maze	
• Generate a Maze • Build the Maze • Specify the start point • Specify the end point	• Game

Seeker	
• Move (Left, Right, Up, Down) • Return current position	• Maze • Game

Game	
• Display and update the GUI elements of the game • Request new maze • Manage key input	• Maze • Seeker

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