JavaFX

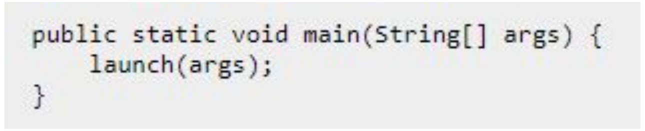
CTEC 260

Getting JavaFX

- Built into Java 8
- Must be imported in Java 11 and 13
  - https://openjfx.io/

Application

- javafx.application.Application
- An abstract class that defines the starting wrapper of your application
- For the simplest implementation, the class that has your main method should extend Application
- You can override the start() method to add functionality
- Call the static launch() method to start the magic



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

```java
public class HelloWorldFX extends Application {

    public void start(Stage stage) {

        Label message = new Label("First FX Application!");
        message.setFont( new Font(40) );

        Button helloButton = new Button("Say Hello");
        helloButton.setOnAction( e -> message.setText("Hello World!") );
        Button goodbyeButton = new Button("Say Goodbye");
        goodbyeButton.setOnAction( e -> message.setText("Goodbye!!") );
        Button quitButton = new Button("Quit");
        quitButton.setOnAction( e -> Platform.exit() );

        HBox buttonBar = new HBox( 20, helloButton, goodbyeButton, quitButton );
        buttonBar.setAlignment(Pos.CENTER);
        BorderPane root = new BorderPane();
        root.setCenter(message);
        root.setBottom(buttonBar);

        Scene scene = new Scene(root, 450, 200);
        stage.setScene(scene);
        stage.setTitle("JavaFX Test");
        stage.show();

    } // end start();

    public static void main(String[] args) {
        launch(args);  // Run this Application.
    }

} // end class HelloWorldFX
```

About Lambda Notation

- Everyone thinks Functional Programming is super cool right now
- Java wants in on the fun
    - But it’s strictly an Object Oriented Language
- Lambda Notation is a way for Java to fake First Class Functions
    - First Class Function - when Function Definitions can be passed around like any other value in a variable
- If an Interface only defines a single method, it’s considered a “Functional” Interface
    - As a shorthand for creating an Anonymous Object based on that interface, you can write a Lambda Function

Lambda Notation

```
helloButton.setOnAction(new ActionListener(){
        public void actionPerformed(ActionEvent e){
                message.setText(“Hello World!”);
        }
});
```

```
helloButton.setOnAction( e ->
message.setText(“Hello World”) );
```

Ok, back to JavaFX now

Stage

- Represents the GUI window
- Implicitly passed to start() when you launch()
- You can make others, but you get the primary window from start()

```
stage.setScene(scene);
stage.setTitle(“Hello”);
stage.show();
```

Scene

- The root container for all other components
  - You can toss buttons, labels, and other containers in here
- A Stage can only contain one Scene at a time
- All constructors for Scene require one “Parent” node
  - Parent Node - a container object in JavaFX that can contain other nodes

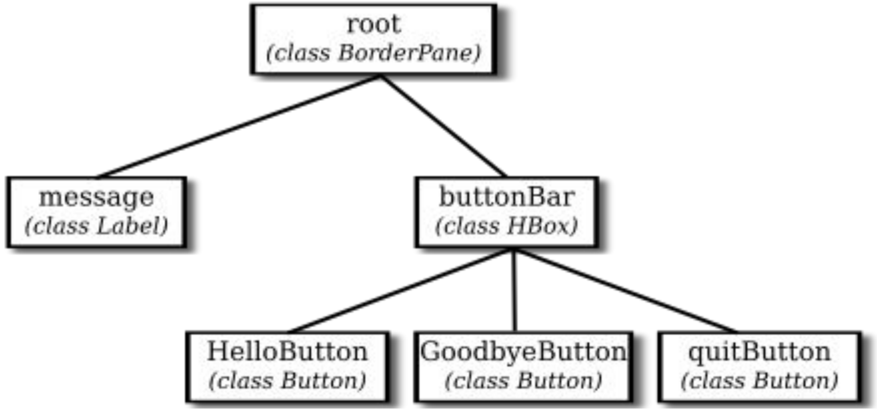
```
Scene scene = new Scene(myContainer);
stage.setScene(scene);
```

Nodes, Children, Parents and Containers

- Node - an object that can be part of the SceneGraph
- Parent - a subclass of Node, can contain other Nodes
- Children - Nodes that cannot contain other Nodes
  - Not a specific class

Unlike Swing, Containers are represented as different classes with different layout methods, rather than a generic class that can implement different layout managers

An Example SceneGraph



A Quick Word on Events

- Listeners and Events generally work the same for JavaFX as for Swing
- However, the Listener Interfaces and addListener methods are different
    - They’re generally simplified to use single method interfaces, thus allowing you to use Lambda Notation

Basic Classes

JavaFX Color

- Slightly different from Swing Color
- javafx.scene.paint.Color

```
Color myColor = new Color(r,g,b,a);
Color myOtherColor = Color.color(r,g,b,a);
Color myThirdColor = Color.color(r,g,b);
Color anotherColor = Color.rgb(r,g,b);
```

Fonts

- Javafx.scene.text

Font myFont = Font.font(family, weight, posture, size);

Font font = Font.font(“Times New Roman”, FontWeight.BOLD ,FontPosture.ITALIC, 12);

Image

- Javafx.scene.image
- Can be displayed by a graphicsContext or by other specialized components

```
Image myImage = new Image(“moose.jpg”);
```

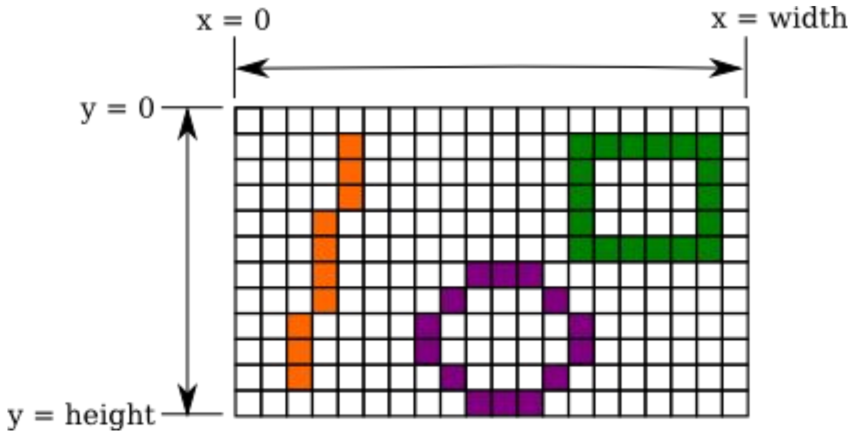
Canvas

- Unlike Swing, JavaFX has a specialized container for drawing
- Javafx.scene.canvas

```
Canvas canvas = new Canvas(width, height);
```

- Width and height here are actually of type Double
- Canvas by default is “filled” with a transparent black
    - So it will show whatever it’s in front of

Remember this?



GraphicsContext

- Every Canvas has its own Graphics Context
- You can access it via canvas.getGraphicsContext2D();

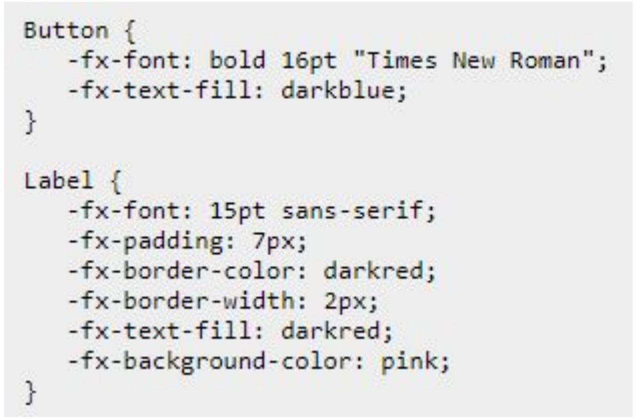
Some GraphicsContext Methods:

- g.strokeRect(x,y,w,h)
- g.strokeOval(x,y,w,h)
- g.fillRect(x,y,w,h)
- g.drawImage(image, x, y)
  - There are several drawImage overloads that allow you to draw only part of an image

CSS

- Cascading Style Sheets
  - … but a specialized version

scene.getStylesheets().add(“mystyle.css”));



```
Button {
    -fx-font: bold 16pt "Times New Roman";
    -fx-text-fill: darkblue;
}

Label {
    -fx-font: 15pt sans-serif;
    -fx-padding: 7px;
    -fx-border-color: darkred;
    -fx-border-width: 2px;
    -fx-text-fill: darkred;
    -fx-background-color: pink;
}
```

Events

MouseEvents and KeyEvents

- While Swing had Listener objects that handled multiple types of Events, JavaFX simplifies things by having a single EventHandler Interface with a single “handle” method
- An object will have several methods to hook up distinct handlers
    - setOnMousePressed(myEventHandler)
    - setOnMouseDragged(myEventHandler)
    - setOnMouseReleased(myEventHandler)
    - setOnKeyPressed(myEventHandler)
    - setOnKeyReleased(myEventHandler)

Note that each Handler will be using a different type of Event (MouseEvent, KeyEvent) based on the context

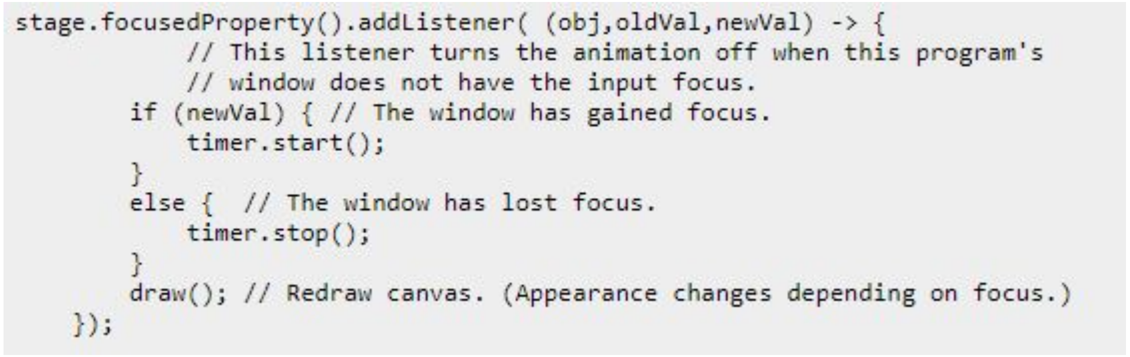
AnimationTimer

- javafx.animation.AnimationTimer
- More limited than a Swing Timer, but does what we generally need
- Runs specified logic every 60th of a second

```
AnimationTimer timer = new AnimationTimer(){
    public void handle(long time){
        //do stuff
    }
};
timer.start();
timer.stop();
```

Observable Values

- Some JavaFX classes are set up to raise events whenever their properties change
- You can use these properties to update different parts of the application when a value changes somewhere
    - We’ll come back to this later...



```
stage.focusedProperty().addListener( (obj,oldVal,newVal) -> {
            // This listener turns the animation off when this program's
            // window does not have the input focus.
        if (newVal) { // The window has gained focus.
            timer.start();
        }
        else {  // The window has lost focus.
            timer.stop();
        }
        draw(); // Redraw canvas. (Appearance changes depending on focus.)
    });
```

Wait, what’s a property?

- javafx.beans.property
- In practical terms, it’s just like an instance field
    - Instead of that field being a “boolean” or an “integer”, it’s a booleanProperty or an integerProperty
    - These Properties are just objects that inherit from the Property Class
    - Their purpose is to create values that store information like normal fields, but that automatically raise events do other cool stuff when they change

Basic Controls

ImageView

- A Node that can contain an image

```
Image moose = new Image(“moose.jpg”);
ImageView mooseView = new ImageView(moose);
```

The Control Class

- Javafx.scene.control
- Parent class to many other controls
- Some common methods:
    - control.setDisable(true);
    - control.setToolTipText(“Hello, I’m a tool tip!”);
    - control.setStyle(cssString);

Labeled

- A parent class that represents a control with text information
- Can contain text and graphics
- Subclasses:
    - Button
    - Label
    - CheckBox
    - And Many More!

Labels

```
Label message = new Label(“Hello World”);
```

Buttons

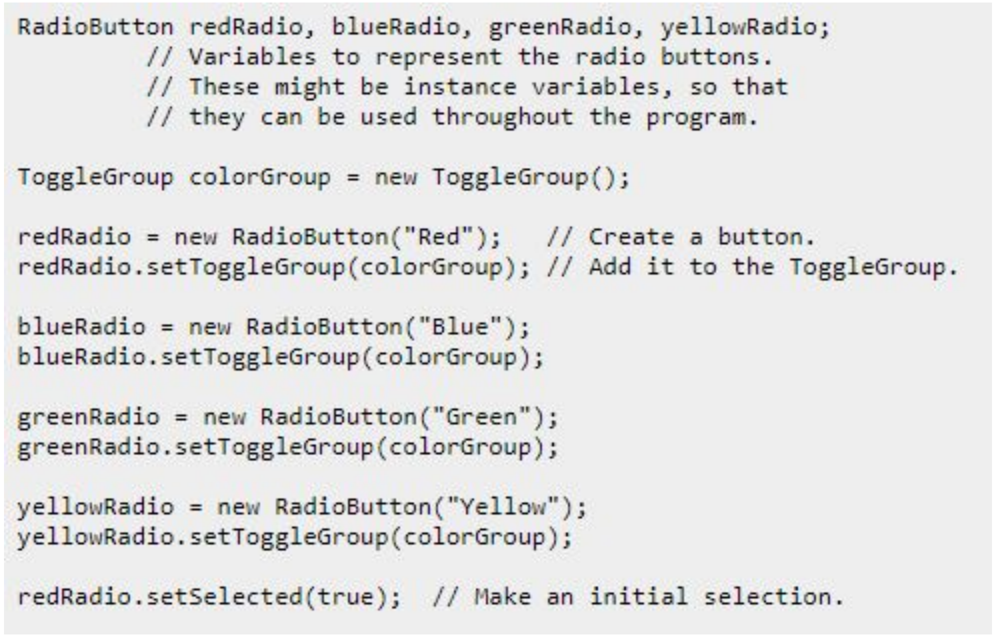
```
Button button = new Button(“Stop”);
button.setOnAction( e -> timer.stop());
```

CheckBox

CheckBox check = new CheckBox(“Check Me!”);

- Can set checked or unchecked with setSelected(true) and setSelected(false)
  - This won’t fire any events
- Can read with isSelected() method
- Can set a Handler with setOnAction()
- Can toggle with fire()
  - This will fire an event

RadioButton



```
RadioButton redRadio, blueRadio, greenRadio, yellowRadio;
         // Variables to represent the radio buttons.
         // These might be instance variables, so that
         // they can be used throughout the program.

ToggleGroup colorGroup = new ToggleGroup();

redRadio = new RadioButton("Red");   // Create a button.
redRadio.setToggleGroup(colorGroup); // Add it to the ToggleGroup.

blueRadio = new RadioButton("Blue");
blueRadio.setToggleGroup(colorGroup);

greenRadio = new RadioButton("Green");
greenRadio.setToggleGroup(colorGroup);

yellowRadio = new RadioButton("Yellow");
yellowRadio.setToggleGroup(colorGroup);

redRadio.setSelected(true);  // Make an initial selection.
```

TextField and TextArea

- TextField - for the small stuff
- TextArea - for the big stuff
- Methods
  - setText(“asdf”);
  - getText()
  - appendText(“asdf”);

Slider

```
Slider slider = new Slider(min, max, current);
```

Observable Properties

- Slider (and many other controls) have Observable Properties

```
slider.valueProperty().addListener( e -> doAThing() );
```

https://docs.oracle.com/javase/8/javafx/api/javafx/beans/value/ObservableValue.html

Layout

Pane

- A container with no predefined layout

```
Pane pane = new Pane();
pane.setPrefWidth(500);
pane.setPrefHeight(500);
pane.getChildren().addAll(myButton, myLabel);

Scene scene = new Scene(pane, 500,500);
```

relocate()

- Method of Node
- Sets the x and y position of the Node for whatever container it’s in

```
Button button = new Button(“Click me!”);
button.relocate(20,20);
pane.getChildren().add(button);
```

setManaged() and resize()

- Nodes can be “managed”
  - This allows them to automatically be resized based on content and context
  - This prevents you from directly controlling their size
- setManaged(false);
  - Disables management
- resize(width, height);
  - Manually sets the width and height

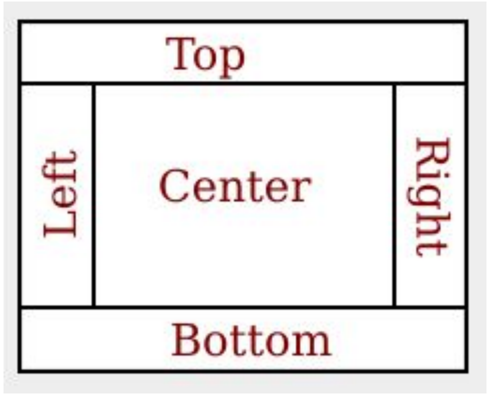
```
button.setManaged(false);
button.resize(100,20);
```

BorderPane

- A subclass of Pane that has 5 areas in which Nodes can be placed
- Nodes will attempt to fill their area
- You can also set behavior for when a child cannot fill their area

```
pane.setCenter(node);
pane.setTop(node);
pane.setRight(node);
pane.setBottom(node);
pane.setLeft(node);
```

```
BorderPane.setAlignment( child, position );
```



HBox and VBox

- Subclasses of Pane that layout elements in a straight line
  - HBox - elements laid out in a single row
  - VBox - elements laid out in a single column
- setSpacing(amount);
  - Sets the space between each child
- setHgrow(child, priority); and setVgrow(child, priority);
  - Makes the specified node grow to fill larger sizes based on the provided priority
  - Priority Values:
    - Priority.ALWAYS
    - Priority.NEVER
    - Priority.SOMETIMES
- setPrefWidth(width);
- setMaxWidth(width);

Menus

- The Menu class - a dropdown menu that contains items
- The MenuItem class - an item in a dropdown menu
- The MenuBar class - a bar that contains menus

Bindable Properties

Observable and Bindable Properties

- Remember, many of these classes have Properties that can raise events when they’re changed
  - A Property is just a value held on an object that works some magic in its setter and getter methods
- On many of these, you can also “bind” values
  - When you change a property on one object, it changes a property on another!

Bindable Properties

```java
Label message = new Label("Never Seen");
TextField input = new TextField("Type Here!");
message.textProperty().bind( input.textProperty() );
```

Bidirectional Bindings

```
cb2.selectedProperty().bindBidirectional( cb1.selectedProperty() );
```

JavaFX Concurrency

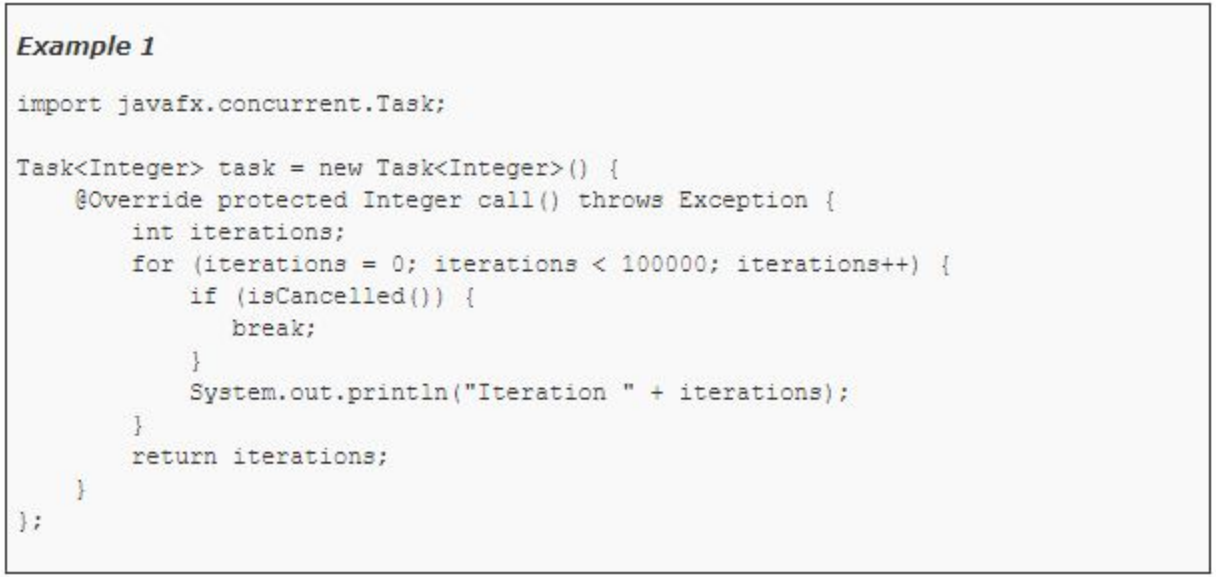
The Worker Interface

- Defines an object that performs work on a background thread
- Parts of the object are observable and usable from the JavaFX Application Thread
- Useful Properties:
    - totalWork
    - workDone
    - progress

The Task Class

- Inherits from Worker
- Built to do a job once
- Implements Runnable and can be provided to a Thread
- You can build a Task by creating a class that inherits from it
  - That class should override call()

Building for Cancellation

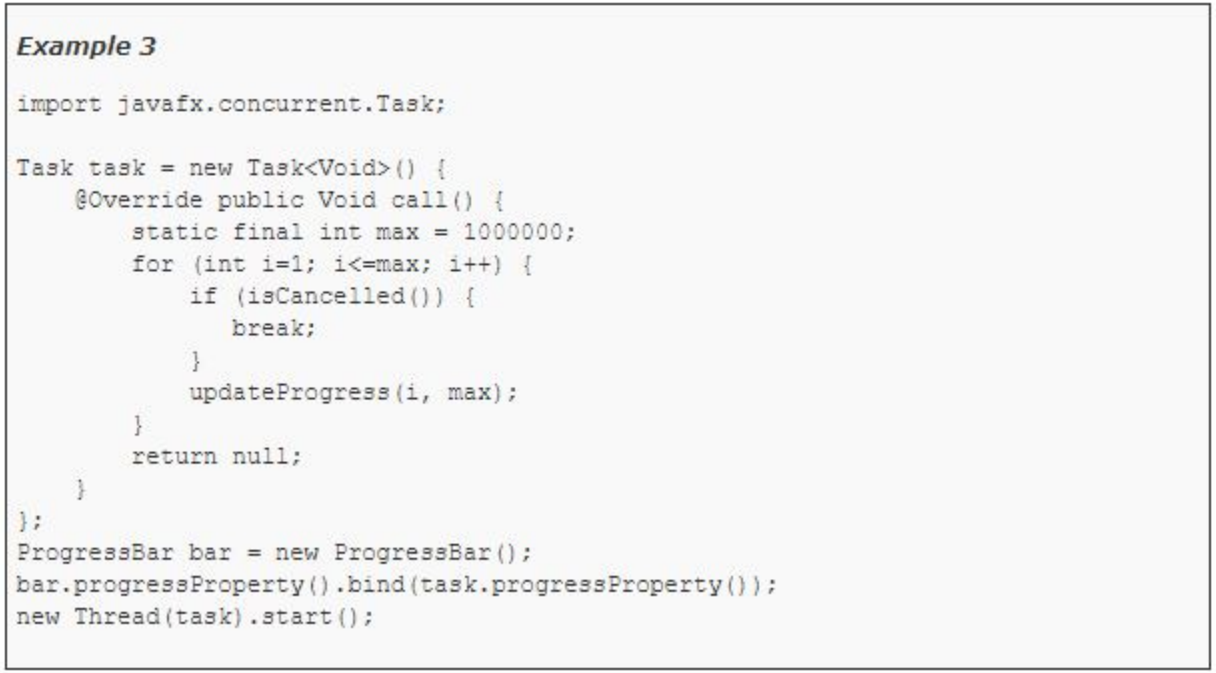


Example 1

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations;
        for (iterations = 0; iterations < 100000; iterations++) {
            if (isCancelled()) {
               break;
            }
            System.out.println("Iteration " + iterations);
        }
        return iterations;
    }
};
```

Showing Progress



Example 3

```
import javafx.concurrent.Task;

Task task = new Task<Void>() {
    @Override public Void call() {
        static final int max = 1000000;
        for (int i=1; i<=max; i++) {
            if (isCancelled()) {
               break;
            }
            updateProgress(i, max);
        }
        return null;
    }
};
ProgressBar bar = new ProgressBar();
bar.progressProperty().bind(task.progressProperty());
new Thread(task).start();
```