Brief Overview of JavaFX

ICS432

Concurrent and High-Performance Programming

Henri Casanova (henric@hawaii.edu)

Why Talk about this in ICS432?

- Recall that there are two main motivations for using concurrency: interactivity and performance
- We'll cover both uses in the semester, but we'll start with interactivity
- Interactivity is often needed in the context of Graphical User Interfaces (GUIs)
- So in this course you will be exposed to the principles behind using threads in GUIs
- The goal is not to develop anything fancy or particularly nice-looking, but to focus more on the inner workings of GUIs
  - See HCI courses for how to design interfaces

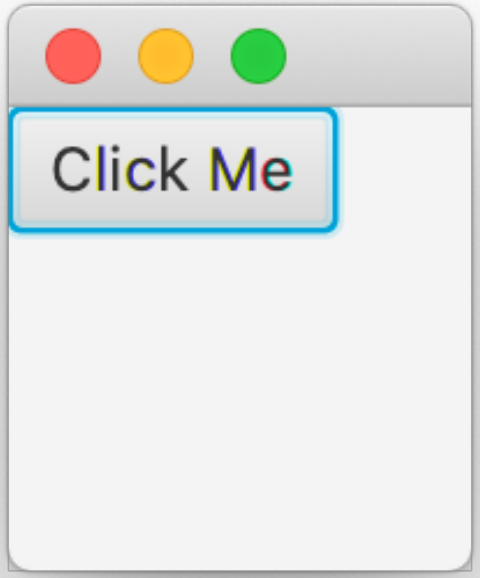
Java GUIs with JavaFX

- The way to develop GUIs in Java is JavaFX
  - Successor to Swing, itself successor to awt
  - Swing is still used by developers
- Show of hands: Who has done anything with JavaFX before?

- There are many on-line JavaFX tutorials and of course full Javadoc documentation
- These slides will be very light on JavaFX itself, and focus mostly on one multi-threading aspect
  - Besides, I'll give you starter code that showcases already quite a few JavaFX features
  - You will have very little JavaFX development to do, and when in doubt ask questions

The Hello Word JavaFX Program

- Program **`ButtonInFrameJavaFX.java`** on the course Web site



The source code

```
public class ButtonInFrameJavaFX extends Application {

    private Button button;

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

    public void start(Stage primaryStage) {
        button = new Button("Click Me");
        button.setOnAction(
                new EventHandler<ActionEvent>() {
                        public void handle(ActionEvent event) {
                                System.out.println("Clicked!");
                        }
                });
        FlowPane layout = new FlowPane();
        layout.getChildren().add(button);
        primaryStage.setScene(new Scene(layout, 100, 100));
        primaryStage.show();
    }
}
```

The source code

```
public class ButtonInFrameJavaFX extends Application {

    private Button button;

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

    public void start(Stage primaryStage) {
        button = new Button("Click Me");
        button.setOnAction(
                EventHandler<ActionEvent>() {
                        public void handle(ActionEvent event) {
                                System.out.println("Clicked!");
                        }
                });
        FlowPane layout = new FlowPane();
        layout.getChildren().add(button);
        primaryStage.setScene(new Scene(layout, 100, 100));
        primaryStage.show();
    }
}
```

Can be replaced with a lambda

The source code

```
public class ButtonInFrameJavaFX extends Application {

    private Button button;

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

    public void start(Stage primaryStage) {
        button = new Button("Click Me");
        button.setOnAction(event-> System.out.println("Clicked!"));
        FlowPane layout = new FlowPane();
        layout.getChildren().add(button);
        primaryStage.setScene(new Scene(layout, 100, 100));
        primaryStage.show();
    }
}
```

The source code

```
public class ButtonInFrameJavaFX extends Application {

    private Button button;

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

    public void start(Stage primaryStage) {
        button = new Button("Click Me");
        button.setOnAction(event-> System.out.println("Clicked!"));
        FlowPane layout = new FlowPane();
        layout.getChildren().add(button);
        primaryStage.setScene(new Scene(layout, 100, 100));
        primaryStage.show();
    }
}
```

Let's look at the real code and run it…

JavaFX Classes

- JavaFX has many, many classes
- They can be used to develop professional GUIs
  - FXML allows you to define GUIs in XML and then generate a lot of JavaFX code automatically
  - Very useful for complex GUIs, and not used in the ics432imgapp at all :)
- The objective in our upcoming assignments is not for you to create amazing GUIs
  - You will only add to an existing JavaFX application
- You just need basic understanding of JavaFX; going further it totally up to you
  - Some people really enjoy GUI development, some really do not…
  - I don't, so the ics432imgapp likely isn't as great as it could be

JavaFX is multi-threaded

- Even though your JavaFX application may not create any threads, it is multi-threaded
- In the code in the previous slides, there are two threads!
  - Your (main) thread
  - The JavaFX Application thread
- The JavaFX Application thread is in charge of “GUI stuff” (e.g., reacting for mouse clicks)
- The **`javafx.Platform.isFxApplicationThread()`** method returns true if it is called by the JavaFX Application thread, false otherwise
- Let’s add some of these calls to this program and see this is action…

Golden Rule

- A lot of the code you write in your JavaFX app is executed by the JavaFX Application thread
- And yet, this thread is in charge of bunch of stuff to make sure the GUI works as expected
- **Golden Rule:** once the GUI is visible, the JavaFX Application thread should only call methods that return quickly
- If you break that rule, you will have the dreaded "frozen GUI" problem
  - While the JavaFX Application thread is running your code, it cannot, e.g., respond to mouse clicks
- You should pay attention to the golden rule above always, but in particular when implementing `handle(ActionEvent event)` methods
- Let's add a second button to our application and showcase the frozen GUI problem…

Thread Proliferation

- What if I need to do something that takes time?
  - Say in some window there is a button that, when clicked, will do something that takes 20 minutes
  - I don’t want all the other windows to be frozen!
- The answer: do it in a thread!
- Whenever you’re writing code that the JavaFX Application thread will execute and that may take a while: Instead spawn a thread to do the work and have the JavaFX Application thread return quickly to its intended work of handling GUI events
- This may require some re-engineering of the application code, as we’ll see

More Thread Proliferations

- Sometimes, JavaFX will enforce that some code be executed by the JavaFX Application Thread
- In these cases, you'll get the exception: **`"java.lang.IllegalStateException: Not on FX application thread"`**
- The solution: use **`Platform.runLater()`** to do it in a thread!
- Note that if you are already in the JavaFX Application Thread and you call **`Platform.runLater()`**, that will run… well.. "later"
  - The JavaFX Application Thread maintains a "todo list" and goes through it in order
- So, yeah, plenty of threads
  - And as you add threads, you may encounter more of the above exception, leading to even more threads!!!
- There are already **`Platform.runLater()`** calls in the started ics432imgapp application...

Conclusion

- JavaFX is huge, but it's well-documented and tutorials are clear
- It's up to you how much of a JavaFX expert you want to become in this course
- The only thing that matters to us is concurrency
- We just saw one concurrency issue with JavaFX
- We'll see others!
- All GUI systems are pretty much alike, with a few differences, and the same concurrency issues occur
  - We're just using JavaFX because you all know Java and Java development is as easy as can be

- For now, make sure you read/understand the ics432imgapp JavaFX code
  - Coming to class with questions about the ics432imgapp code will help everyone and is highly encouraged