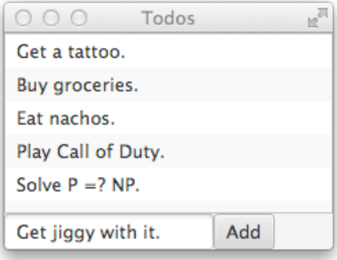
Introduction to Object-Oriented Programming

JavaFX GUIs

Christopher Simpkins
chris.simpkins@gatech.edu

Todo GUI

Today we'll make a GUI for todo lists



Get a tattoo.

Buy groceries.

Eat nachos.

Play Call of Duty.

Solve P =? NP.

Get jiggy with it.

Along the way we'll

- Review event handling
- Learn two new UI controls
- Learn about nested layouts
- See an example of MVC in JavaFX
- See a basic use of JavaFX's properties

The Application

Where do we start? The `Application`:

```
import javafx.application.Application;
import javafx.stage.Stage;

public class TodoList extends Application {

    @Override public void start(Stage stage) {

    }
}
```

And now we just follow our recipe:

- Create UI controls
- Add UI controls to a parent node in a scene graph
- Set the stage's scene graph and show

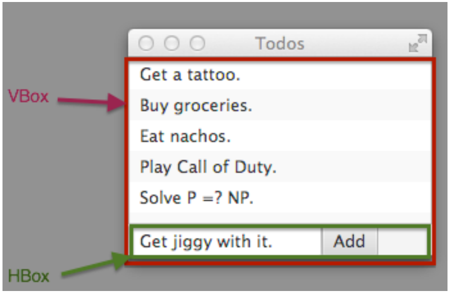
Create UI Controls

```
    @Override public void start(Stage stage) {
        ListView<String> listView = new ListView<String>();

        Button addButton = new Button();
        addButton.setText("Add");

        TextField inputField = new TextField();
    }
```

And, of course, we'll need to import these:

```
import javafx.scene.control.Button;
import javafx.scene.control.ListView;
import javafx.scene.control.TextField;
```

Add UI Controls to Parent Node - Layout



Get a tattoo.
Buy groceries.
Eat nachos.
Play Call of Duty.
Solve P =? NP.
Get jiggy with it.

To acheive the layout we want, we'll nest an `HBox` inside a `VBox`:

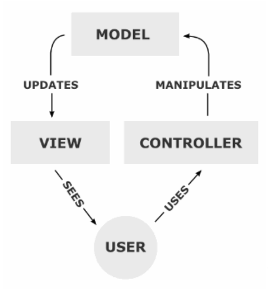
```
    @Override public void start(Stage stage) {
        HBox entryBox = new HBox();
        entryBox.getChildren().addAll(inputField, addButton);

        VBox vbox = new VBox();
        vbox.getChildren().addAll(listView, entryBox);
    }
```

Setting up and showing the stage

Although we’re not done with our UI controls, we go ahead and do the last step of our recipe so we can run the program:

```java
import javafx.scene.Scene;
// ...

public class TodoList extends Application {
    @Override public void start(Stage stage) {
        // ...
        Scene scene = new Scene(vbox);
        stage.setScene(scene);
        stage.setTitle("Todos");
        stage.show();
    }
}
```

The Model-View-Controller Design Pattern



[1]

- The *model* contains the data that is displayed by the *view*
- The *view* displays the data from the *model* on screen
- The *controller* gets input from the user and manipulates the model

In JavaFX the view and controller are typically combined.

[1] http://en.wikipedia.org/wiki/File:MVC-Process.png

ListView, ObservableList, and MVC

JavaFX provides model classes that work with UI controls. For our `ListView` we'll simply use an `ObservableList<String>`, which we obtain from `FXCollections.observableArrayList()`:

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;

public class TodoList extends Application {

    private ObservableList<String> todos;

    @Override public void start(Stage stage) {
        ObservableList<String> todos =
    FXCollections.observableArrayList();
        ListView<String> listView = new ListView<String>(todos);
        // ...
```

Handling Model Updates

- Whenever the `ObservableList<String> todos` is updated, the change is automatically reflected in the `ListView`.
- So all we have to do is add text from the `TextField` to the `todos` list whenever the add button is clicked.

```
    @Override public void start(Stage stage) {
        // ...
        addButton.setOnAction(e -> {
            todos.add(inputField.getText());
            inputField.setText("");
            inputField.requestFocus();
        });
        // ...
    }
```

Notice that after the text is added to the list we reset the text field and give it the focus again.

Properties

We don’t want to add empty strings from the TextField to the todos list, so let’s disable the Add button when the TextField is empty:

```java
import javafx.beans.binding.Bindings;
// ...

public class TodoList extends Application {

    private ObservableList<String> todos;

    @Override public void start(Stage stage) {
        // ...
        addButton.disableProperty()
            .bind(Bindings.isEmpty(inputField.textProperty()));
        // ...
    }
}
```

Properties play a big role in modern JavaFX programming. This is just a small taste.

Conclusion

Very simple app to get started with UI controls and MVC.

- GUI programming requires two things:
  - Knowledge of GUIs (widgets, how they work, how they're used)
  - Knowledge of a particular GUI framework (like JavaFX)
- The JavaFX classes you've seen make extensive use of OOP.
- GUI programs are straightforward, but get complex quickly.
- JavaFX's properties and the Model-View-Controller pattern help us deal with the complexity of GUI programming.

The full Todos example is online: TodoList.java.