Graphical User Interfaces

using JavaFX

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Agenda

- Controls
  - List controls
  - Progress indicators
  - Dialogues and Alerts
  - Charts
  - Menus
  - Tabs and the web
  - Multimedia
- Mouse and Keyboard
  - Events
  - Event Handling
  - Mouse events
  - Keyboard events
- Animation
  - Transitions
  - Translations
  - Timeline animation
  - AnimationTimer
- Wrapping up!

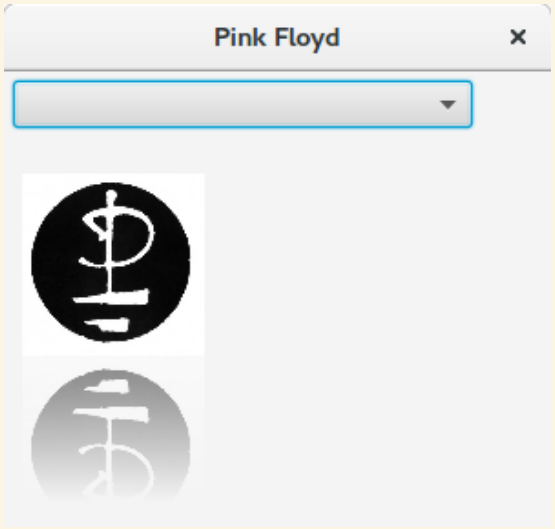
More controls

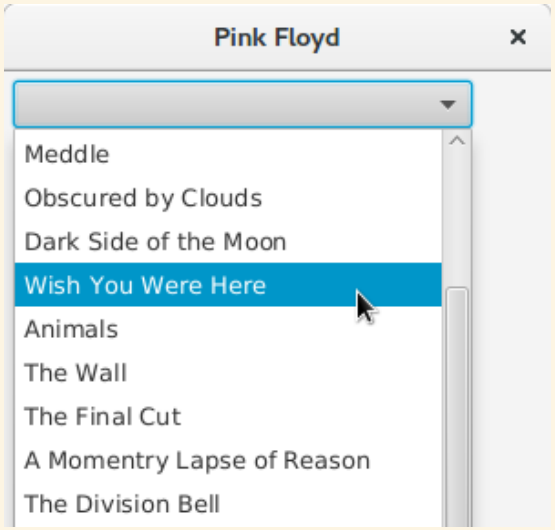
- The first lecture on JavaFX gave a quick look at what is available in JavaFX.
- In this lecture focus is on several more of the different controls available.
- It is not feasible to look at all controls so a selection has been made.
  - Also, after a while working of working with the controls it is obvious that they have been made to work more or less the same.
- Lastly, this lecture will also have a brief look at different animations and effects.

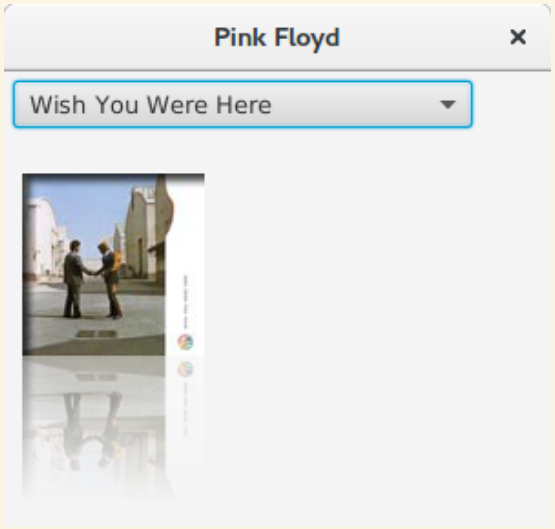
ComboBox

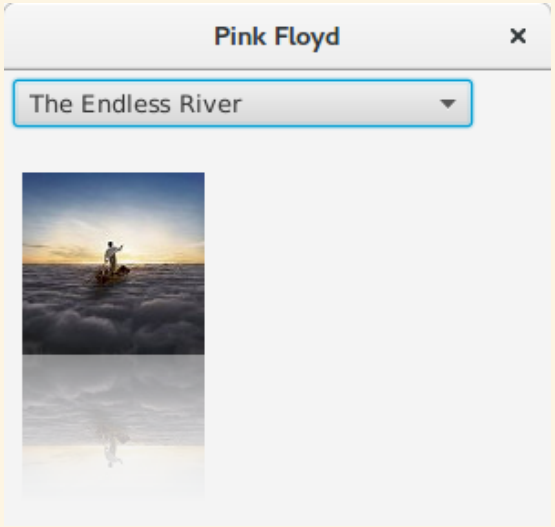
- A `ComboBox` is a *drop down box*.
  - A selection is made from a list of elements that are shown when clicking the box.
- This kind of list control is best suited for 7 ± 3 elements.
- Before version 2.1 of JavaFX a similar control was called `ChoiceBox`.
  - It was less advanced but there are still many examples of it on the Internet.
- Elements in the list can be editable, multiple selectable or just readable.
- If “too many” elements are in the list, a scroll bar will be shown.

Running program









Source code, part 1

```
public void start(Stage primaryStage) {
    VBox layout = new VBox();
    layout.setPadding(new Insets(5, 5, 5, 5));

    ComboBox<String> pinkFloyd = new ComboBox<>();
    pinkFloyd.getItems().addAll(
            "The Piper at the Gates of Dawn", "A Saucerful of Secrets", "More",
            "Ummagumma", "Atom Heart Mother", "Meddle", "Obscured by Clouds",
            "Dark Side of the Moon", "Wish You Were Here", "Animals", "The Wall",
            "The Final Cut", "A Momentry Lapse of Reason", "The Division Bell",
            "The Endless River"
    );

    VBox imageBox = new VBox();
    imageBox.setPadding(new Insets(25, 5, 5, 5));
    Image theAlbum = new Image("file:images/pf.jpg");
    ImageView showAlbum = new ImageView(theAlbum);
    showAlbum.setFitWidth(100.0);
    showAlbum.setPreserveRatio(true);
    Reflection refl = new Reflection();
    refl.setFraction(0.8);
    showAlbum.setEffect(refl);
    imageBox.getChildren().add(showAlbum);

    layout.getChildren().addAll(pinkFloyd, imageBox);
```

Source code, part 2

```
pinkFloyd.setOnAction((ActionEvent e) -> {
       if (pinkFloyd.getValue().equals("The Piper at the Gates of Dawn"))
           showAlbum.setImage(new Image("file:images/piper.jpg"));
       else if (pinkFloyd.getValue().equals("A Saurcerful of Secrets"))
           showAlbum.setImage(new Image("file:images/saucer.jpg"));
       else if (pinkFloyd.getValue().equals("More"))
           showAlbum.setImage(new Image("file:images/more.jpg"));

       // More albums have been cut due to space...

       else if (pinkFloyd.getValue().equals("The Wall"))
           showAlbum.setImage(new Image("file:images/wall.jpg"));
       else if (pinkFloyd.getValue().equals("The Final Cut"))
           showAlbum.setImage(new Image("file:images/final.jpg"));
       else if (pinkFloyd.getValue().equals("A Momentary Lapse of Reason"))
           showAlbum.setImage(new Image("file:images/mlor.jpg"));
       else if (pinkFloyd.getValue().equals("The Division Bell"))
           showAlbum.setImage(new Image("file:images/divbell.jpg"));
       else if (pinkFloyd.getValue().equals("The Endless River"))
                showAlbum.setImage(new Image("file:images/river.jpg"));
   });

   Scene scene = new Scene(layout, 300, 250);

   primaryStage.setTitle("Pink Floyd");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

ListView

- In the `ListView` a number of items are displayed in a list.
  - Which is a better choice than a `ComboBox` if more than seven items are to be displayed.
- It allows for single or multiple selection as well as automatic scroll bars.
- There are a number of methods to listen to:
  - `getSelectionModel().getSelectedIndex()` – returns the selected index
  - `getSelectionModel().getSelectedItem()` – returns the selected item
  - `getFocusModel().getFocusedIndex()` – returns the index of focused item
  - `getFocusModel().getFocusedItem()` – returns the currently focused item

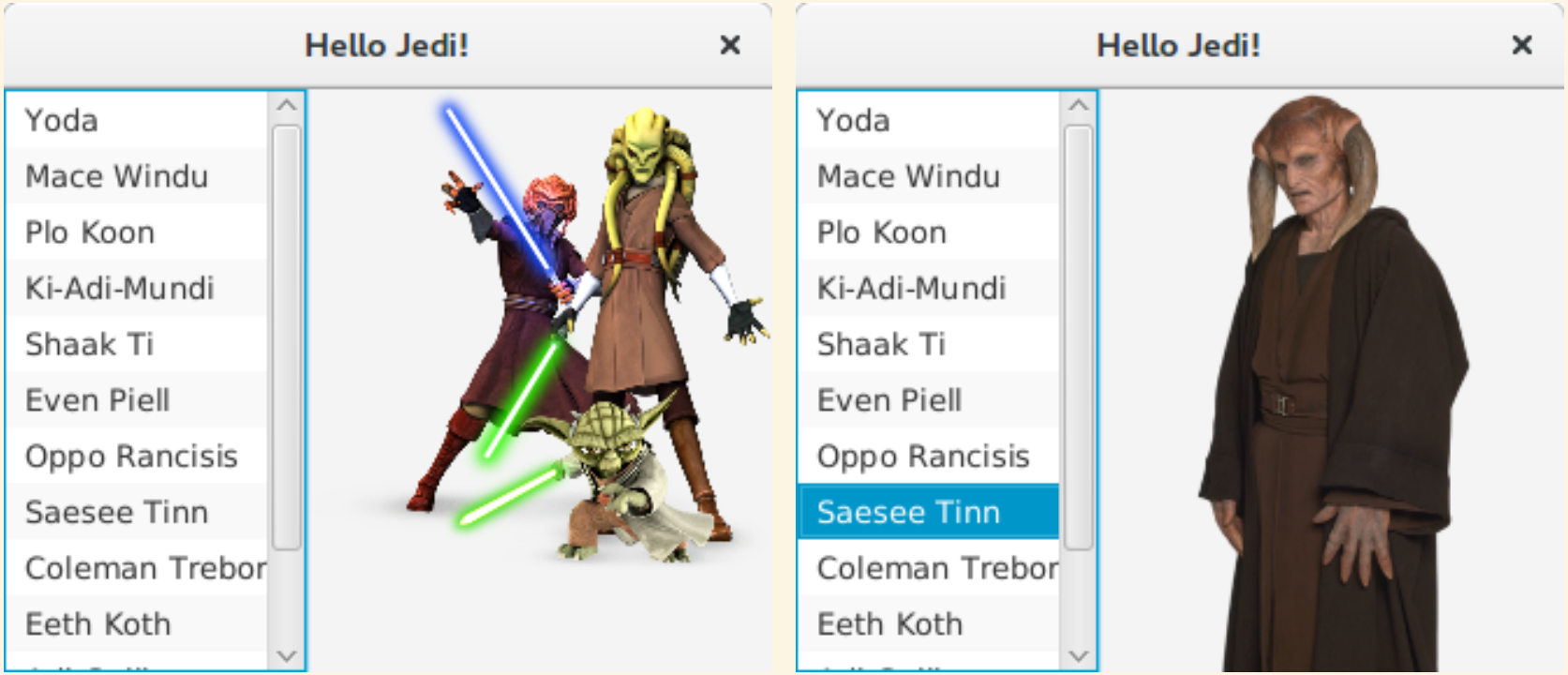
The code, part 1

```
public void start(Stage primaryStage) {
     HBox layout = new HBox();

     ListView<String> jediList = new ListView<>();
     jediList.setPrefWidth(150);

     jediList.getItems().addAll(
             "Yoda",
             "Mace Windu",
             "Plo Koon",
             "Ki-Adi-Mundi",
             "Shaak Ti",
             "Even Piell",
             "Oppo Rancisis",
             "Saesee Tinn",
             "Coleman Trebor",
             "Eeth Koth",
             "Adi Gallia",
             "Depa Billaba"
     );

     Image jediImage = new Image("file:images/jediorder.png");
     ImageView showJedi = new ImageView(jediImage);
     showJedi.setFitWidth(200);
     showJedi.setPreserveRatio(true);
```

The code, part 2

```
    final ArrayList<Image> manyJedi = new ArrayList<>();
    manyJedi.add(new Image("file:images/yoda.png"));
    manyJedi.add(new Image("file:images/macewindu.png"));
    manyJedi.add(new Image("file:images/plokoon.png"));
    manyJedi.add(new Image("file:images/kiadimundi.png"));
    manyJedi.add(new Image("file:images/shaakti.png"));
    manyJedi.add(new Image("file:images/evenpiell.png"));
    manyJedi.add(new Image("file:images/Oppo.jpg"));
    manyJedi.add(new Image("file:images/saeseetiin.png"));
    manyJedi.add(new Image("file:images/Coleman.jpg"));
    manyJedi.add(new Image("file:images/eethkoth.png"));
    manyJedi.add(new Image("file:images/adigallia.png"));
    manyJedi.add(new Image("file:images/Depa.jpg"));

    layout.getChildren().addAll(jediList, showJedi);

    // Add a listener to a selection
    jediList.getSelectionModel().selectedIndexProperty().addListener(ov -> {
        showJedi.setImage(manyJedi.get(jediList.getSelectionModel().getSelectedIndex()));
    });

    Scene scene = new Scene(layout, 330, 250);

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

In graphics


Hello Jedi!
Yoda
Mace Windu
Plo Koon
Ki-Adi-Mundi
Shaak Ti
Even Piell
Oppo Rancisis
Saesee Tinn
Coleman Trebor
Eeth Koth
Hello Jedi!
Yoda
Mace Windu
Plo Koon
Ki-Adi-Mundi
Shaak Ti
Even Piell
Oppo Rancisis
Saesee Tinn
Coleman Trebor
Eeth Koth

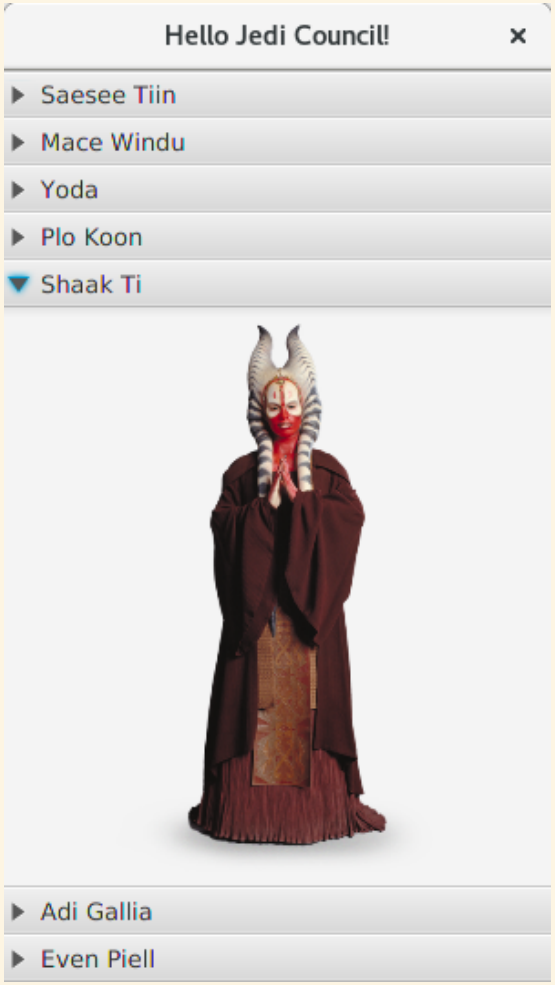
Accordion

- An interesting control for showing lists is the *accordion*.
- It works by having a space with buttons and when one button is pressed, it content is “dragged out”.
  - It was popularised by e-mail clients that use it to change beteen different activites like e-mail, calendar, to-do and so on.
- In JavaFX two classes are used together to perform an accordion effect.
  - `TiledPane` is the panel that can be maximised or minimised.
  - `Accordion` which is the accordion itself.
- Together they create a nice accordion control for displaying information.

Source code

```
public void start(Stage primaryStage) {
    TitledPane tiinPane = new TitledPane();
    tiinPane.setText("Saesee Tiin");
    ImageView theImage = new ImageView(new Image("file:images/saeseetiin.png"));
    theImage.setFitHeight(300);
    theImage.setPreserveRatio(true);
    tiinPane.setContent(theImage);

    TitledPane macePane = new TitledPane();
    macePane.setText("Mace Windu");
    ImageView imageMace = new ImageView(new Image("file:images/macewindu.png"));
    imageMace.setFitHeight(300);
    imageMace.setPreserveRatio(true);
    macePane.setContent(imageMace);

    // Several panes have been removed for space...

    Accordion dragspel = new Accordion();
    dragspel.getPanes().addAll(tiinPane, macePane, yodaPane, ploPane, tiPane, adiPane, piellPane);
    dragspel.setExpandedPane(tiinPane);

    StackPane root = new StackPane();
    root.getChildren().add(dragspel);

    Scene scene = new Scene(root, 300, 500);
    primaryStage.setTitle("Hello Jedi Council!");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Running program



TreeView

- A more advanced control for building lists like trees is `TreeView`.
- In JavaFX they are contructed in two steps, with two classes:
  - First the root element with children with `TreeItem`.
  - Then the view itself with `TreeView`.
- The elements can contain any type of node, most often either text or images, or both.
- To set a branch to be visible as default, use the method `setExpanded()`.
- A listener can be set to a selected element, as previously shown, to make something happen when selected.

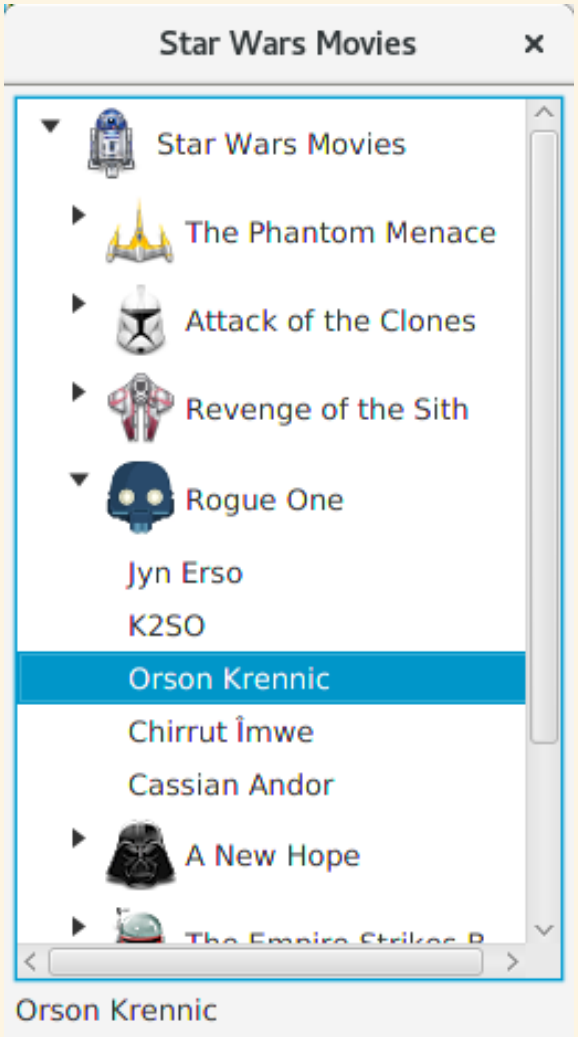
Source code, part 1

```
public void start(Stage primaryStage) {
    VBox layout = new VBox();
    layout.setPadding(new Insets(5, 5, 5, 5));
    layout.setSpacing(5);

    TreeItem<String> starWarsRoot = new TreeItem<>("Star Wars Movies",
            new ImageView(new Image("file:images/movies/R2-D2.png")));
    starWarsRoot.setExpanded(true);

    TreeItem<String> menaceRoot = new TreeItem<>("The Phantom Menace",
            new ImageView(new Image("file:images/movies/Naboo Starfighter.png")));
    menaceRoot.getChildren().add(new TreeItem<>("Anakin Skywalker"));
    menaceRoot.getChildren().add(new TreeItem<>("Qui-Gon Jinn"));
    menaceRoot.getChildren().add(new TreeItem<>("Obi-Wan Kenobi"));
    menaceRoot.getChildren().add(new TreeItem<>("Jar-Jar Binks"));
    menaceRoot.getChildren().add(new TreeItem<>("Queen Amidala"));

    TreeItem<String> attackRoot = new TreeItem<>("Attack of the Clones",
            new ImageView(new Image("file:images/movies/clone-1.png")));
    attackRoot.getChildren().add(new TreeItem<>("Jango Fett"));
    attackRoot.getChildren().add(new TreeItem<>("Cliegg Lars"));
    attackRoot.getChildren().add(new TreeItem<>("Count Dooku"));
    attackRoot.getChildren().add(new TreeItem<>("Poggle the Lesser"));
    attackRoot.getChildren().add(new TreeItem<>("Onaconda Farr"));

    // More elements...
```

Source code, part 2

```
        TreeItem<String> forceRoot = new TreeItem<>("The Force Awakens",
                new ImageView(new Image("file:images/movies/kylo.png")));
        forceRoot.getChildren().add(new TreeItem<>("Kylo Ren"));
        forceRoot.getChildren().add(new TreeItem<>("Rey"));
        forceRoot.getChildren().add(new TreeItem<>("Finn"));
        forceRoot.getChildren().add(new TreeItem<>("Poe Dameron "));


        starWarsRoot.getChildren().addAll(menaceRoot, attackRoot, revengeRoot,
                rogueRoot, hopeRoot, empireRoot, returnRoot, forceRoot);

        TreeView<String> starWars = new TreeView<>(starWarsRoot);

        Label selectedCharacter = new Label();

        starWars.getSelectionModel().selectedItemProperty().addListener(
                (ov, old_val, new_val) -> selectedCharacter.setText(new_val.getValue()));

        layout.getChildren().addAll(starWars, selectedCharacter);

        Scene scene = new Scene(layout);

        primaryStage.setTitle("Star Wars Movies");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

Running program



Progress indicator

- A progress indicator shows how far a piece of work as progressed.
  - This could be a download, a copying, number of calculations etc.
- In JavaFX there are two progress indicators:
  - `ProgressBar` that shows a line representing the progress.
  - `ProgressIndicator` showing a pie chart.
- Both take a value from 0.0 to 1.0 for the amount of progress.
  - A negative value represents an indetermined status.
- The progress is set with the method `setProgres()`.

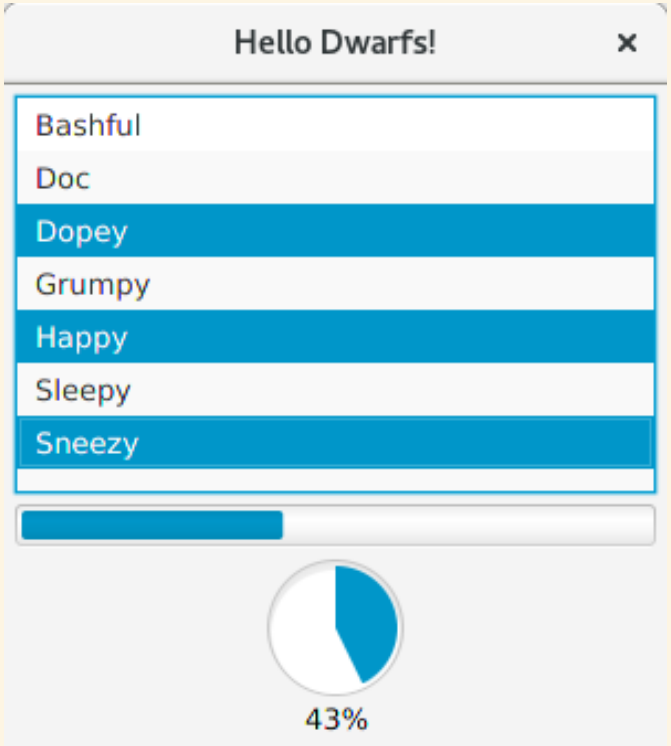
Source code, part 1

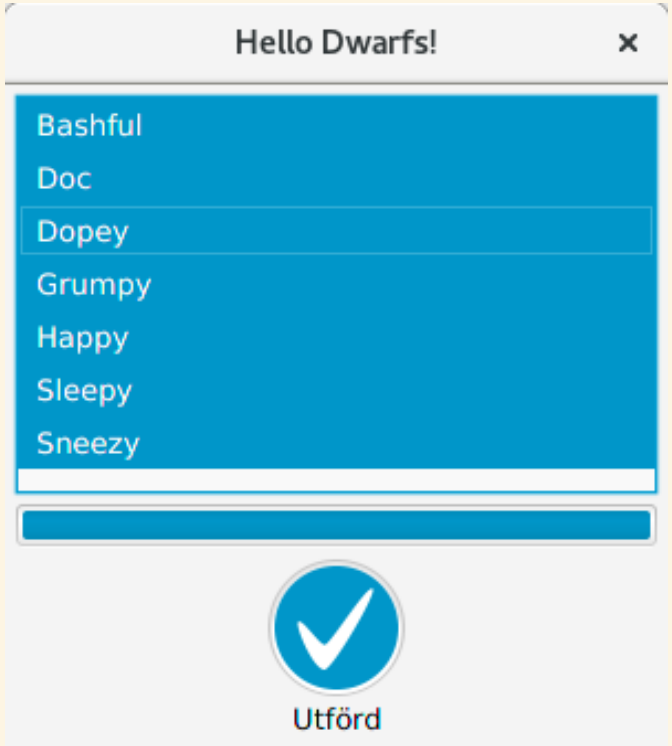
```
public void start(Stage primaryStage) {
    VBox layout = new VBox();
    layout.setPadding(new Insets(5, 5, 5, 5));
    layout.setSpacing(5);

    ListView<String> seven = new ListView<>();
    seven.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
    seven.setPrefHeight(200);
    seven.getItems().addAll(
            "Bashful",
            "Doc",
            "Dopey",
            "Grumpy",
            "Happy",
            "Sleepy",
            "Sneezy"
    );

    ProgressBar theProgress = new ProgressBar(0.0);
    theProgress.setPrefWidth(295);
    ProgressIndicator theIndicator = new ProgressIndicator(0.0);
    theIndicator.setPrefSize(100, 100);
```

Source code, part 2

```
    seven.getSelectionModel().selectedIndexProperty().addListener(
            (ov, old_val, new_val) -> {
                int numItems = seven.getSelectionModel().getSelectedItems().size();
                if(numItems<=7) {
                    theProgress.setProgress(numItems/7.0);
                    theIndicator.setProgress(numItems/7.0);
                }
            });

    layout.getChildren().addAll(seven, theProgress, theIndicator);

    Scene scene = new Scene(layout, 300, 300);

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

Running program





Dialogue boxes and Alerts

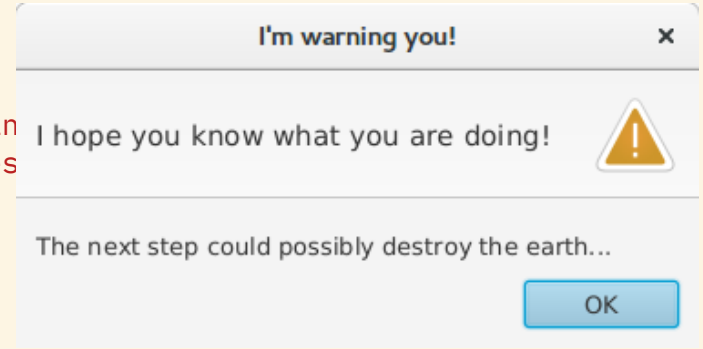
- Dialogue boxes is a, surprisingly, late addition to JavaFX – they first appeared in update 40 of JavaFX 8.
  - Before that there were versions of it in the FXControls library, and in addition to that, they are not too difficult to create by hand.
- They come in many shapes and forms, most of them very flexible.
- As a result of the late adding of dialogue boxes, they rely heavily on Java 8.
  - From Java 8 it takes the new data type called `Optional<T>` which serves as a safe reference to `T` which can be either a value or `null`.
  - It has a method called `isPresent()` that checks if a value is present or not.

Simple example of Alert

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("Dialogues");
    VBox layout = new VBox();
    layout.setPadding(new Insets(5, 5, 5, 5));
    layout.setSpacing(5);

    Button aButton = new Button("Press me");
    aButton.setOnAction(e -> {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Important Information");
        alert.setHeaderText("Your actions will have consequences");
        alert.setContentText("You will now destroy the earth.");

        alert.showAndWait();
    });


    layout.getChildren().addAll(aButton);
    Scene theScene = new Scene(layout, 300, 200);
    primaryStage.setScene(theScene);
    primaryStage.show();
```

Two more examples of Alert

```
aButton.setOnAction(e -> {
    Alert alert = new Alert(AlertType.WARNING);
    alert.setTitle("I'm warning you!");
    alert.setHeaderText("I hope you know what you are doin
    alert.setContentText("The next step could possibly des

    alert.showAndWait();
});

aButton.setOnAction(e -> {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("The earth is gone!");
    alert.setHeaderText("Now you did it!");
    alert.setContentText("Cannot continue because there is

    alert.showAndWait();
});
```

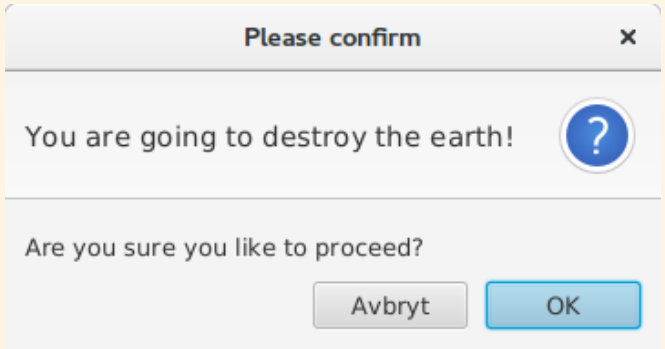


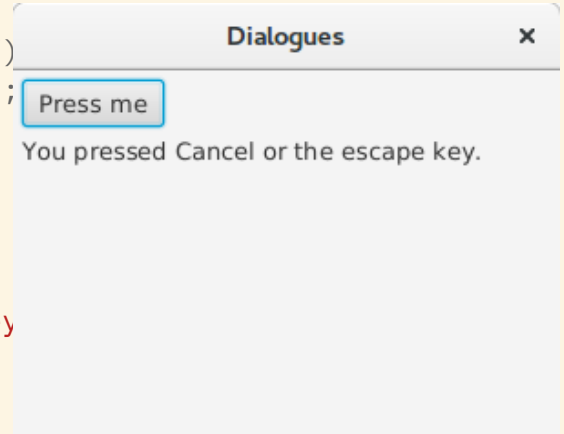
Getting feedback

- As mentioned previously, the new data type `Option<>` is used when interacting with dialogue boxes.
- Later we will use the class `Dialog` for the more advanced dialogues.
- The return value from a dialogue can be either the press of a button or an entered string.
  - There are some predefined buttons like `ButtonType.OK` but it is also possible to create custom buttons.
- When looking on examples on the Internet you will probably see many different ways of handling the response, we will use the more traditional way.

Example

```
primaryStage.setTitle("Dialogues");
VBox layout = new VBox();
layout.setPadding(new Insets(5, 5, 5, 5));
layout.setSpacing(5);

final Label response = new Label();

Button aButton = new Button("Press me");
aButton.setOnAction(e -> {
    Alert alert = new Alert(AlertType.CONFIRMATION);
    alert.setTitle("Please confirm");
    alert.setHeaderText("You are going to destroy the earth!");
    alert.setContentText("Are you sure you like to proceed?");

    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == ButtonType.OK){
        response.setText("You pressed the OK button!");
    } else {
        response.setText("You pressed Cancel or the escape key
    }
});

layout.getChildren().addAll(aButton, response);
Scene theScene = new Scene(layout, 300, 200);
primaryStage.setScene(theScene);
primaryStage.show();
```

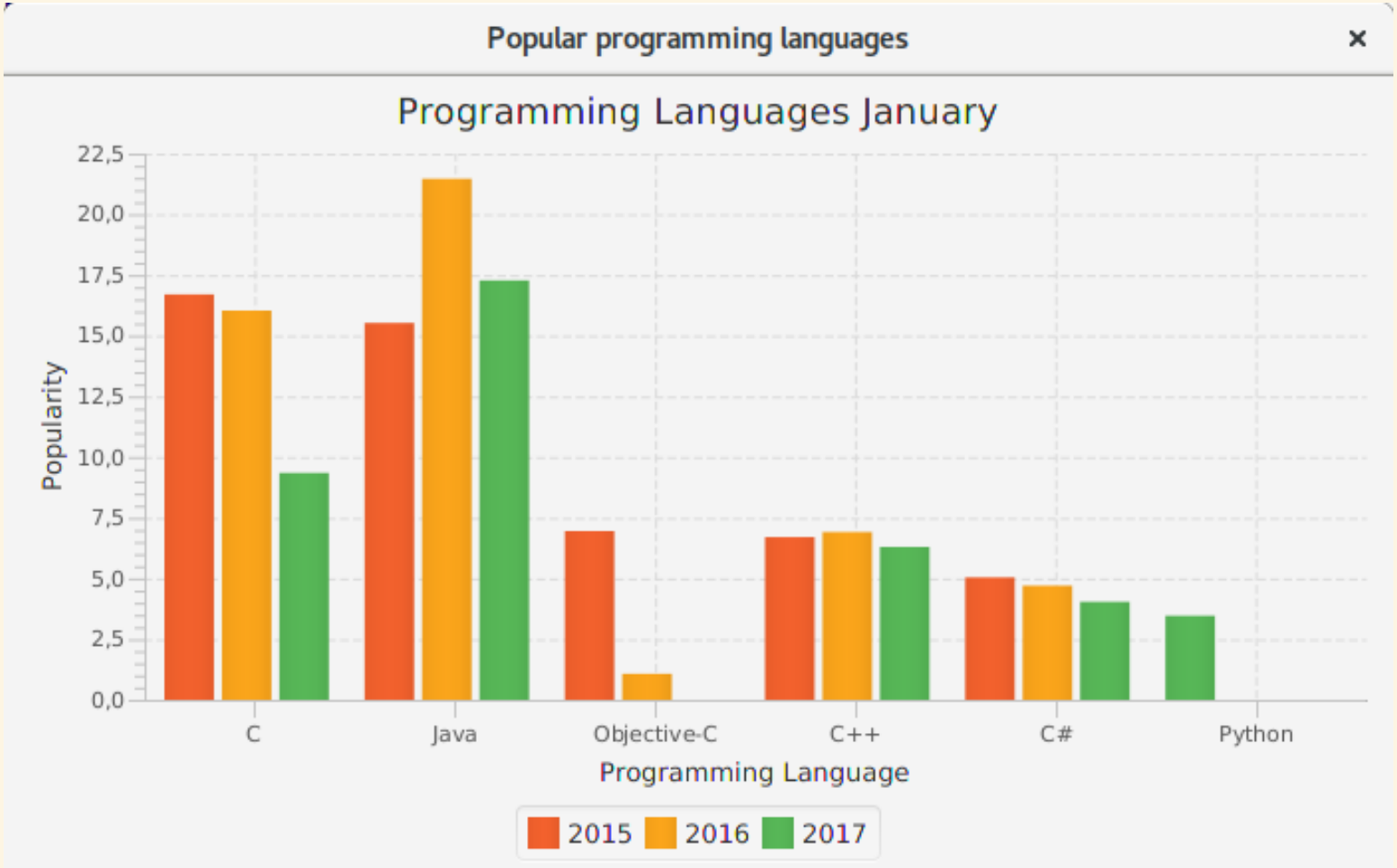




Charts

- A somewhat surprising addition to JavaFX when compared to Swing is native support for *charts*.
- This could be seen as support for developing enterprise applications for management, which often use chars.
  - This area of business is also quite large for Java.
- Currently six different charts are covered by JavaFX:
  - Pie Charts
  - Line Charts
  - Area Charts
  - Bubble Charts
  - Scatter Charts
  - Bar Charts
- They all have a similar foundation, with added behaviour for different charts.

Bar Chart example in graphics



```
primaryStage.setTitle("Popular programming languages");
CategoryAxis xAxis = new CategoryAxis();
NumberAxis yAxis = new NumberAxis();
BarChart<String, Number> prgLang = new BarChart<>(xAxis, yAxis);
prgLang.setTitle("Programming Languages January");
xAxis.setLabel("Programming Language");
yAxis.setLabel("Popularity");

XYChart.Series<String, Number> values17 = new XYChart.Series<>();
values17.setName("2017");
values17.getData().add(new XYChart.Data<String, Number>("Java", 17.278));
values17.getData().add(new XYChart.Data<String, Number>("C", 9.349));
values17.getData().add(new XYChart.Data<String, Number>("C++", 6.301));
values17.getData().add(new XYChart.Data<String, Number>("C#", 4.039));
values17.getData().add(new XYChart.Data<String, Number>("Python", 3.465));

XYChart.Series<String, Number> values16 = new XYChart.Series<>();
values16.setName("2016");
values16.getData().add(new XYChart.Data<String, Number>("C", 16.036));
values16.getData().add(new XYChart.Data<String, Number>("Java", 21.465));
values16.getData().add(new XYChart.Data<String, Number>("Objective-C", 1.074));
values16.getData().add(new XYChart.Data<String, Number>("C++", 6.914));
values16.getData().add(new XYChart.Data<String, Number>("C#", 4.707));

XYChart.Series<String, Number> values15 = new XYChart.Series<>();
values15.setName("2015");
// Cut for space

prgLang.getData().addAll(values15, values16, values17);
Scene scene = new Scene(prgLang, 700, 400);
primaryStage.setScene(scene);
primaryStage.show();
```

Menus

- Many programs today use *menus* as part of their GUI.
- Normally they are placed in the top of the application (or screen depending on OS), but there are also context menus that are activated via a right mouse click.
- In JavaFX three steps are needed to create a menu:
  1. Create the menu bar with `MenuBar`.
  2. Create the different menu options on the bar with `Menu`.
  3. Create the different options tied to each menu with `MenuItem`.
- For more flexible options the classes `RadioButtonItem` and `CheckMenuItem` can be used.
- It is also possible to create menus in many levels.

More on menus

- Each menu option can have an image associated with it.
- It is also possible to create keyboard short cuts to speed the use of the GUI.
- Seperation between option can be shown using the class `SeparatorMenuItem`.
- The behaviour for the options is created as usually with `setOnAction()`
- In the example an auxiliary method for reading from disc is also shown.

Source code, part 1 – auxiliary mehtod

```
private String readFile(String file) {
    File theFile = new File("images/skylanders/" + file);
    StringBuilder sbText = new StringBuilder();
    String text;

    try (Scanner scan = new Scanner(theFile)) {
        while(scan.hasNext()){
            text = scan.nextLine();
            sbText.append(text);
        }
    } catch (IOException ex) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error reading file!");
        alert.setHeaderText("The file could not be read!");
        alert.setContentText("Please check that the file is in the correct path.");

        alert.showAndWait();
    }
    return sbText.toString();
}
```

Source code, part 2

```
public void start(Stage primaryStage) {
    Label theText = new Label();
    theText.setWrapText(true);
    ImageView theView = new ImageView();
    Image theImage = new Image("file:images/skylanders/spyro.png");
    theView.setImage(theImage);

    MenuBar menuBar = new MenuBar();
    Menu menuFile = new Menu("File");
    Menu menuWater = new Menu("Water");
    Menu menuTech = new Menu("Technology");
    Menu menuEffects = new Menu("Effects");
    menuBar.getMenus().addAll(menuFile, menuWater, menuTech, menuEffects);

    MenuItem exitItem = new MenuItem("Exit", new ImageView(
        new Image("file:images/skylanders/application-exit-5.png")));
    exitItem.setOnAction(e -> {
        exit();
    });

    exitItem.setAccelerator(keyCombination("Ctrl+X"));

    MenuItem thumpback = new MenuItem("ThumpBack");
    thumpback.setOnAction(t -> {
        theView.setImage(new Image("file:images/skylanders/thumpback.png"));
        theText.setText(readFile("thumpback.txt"));
    });
```

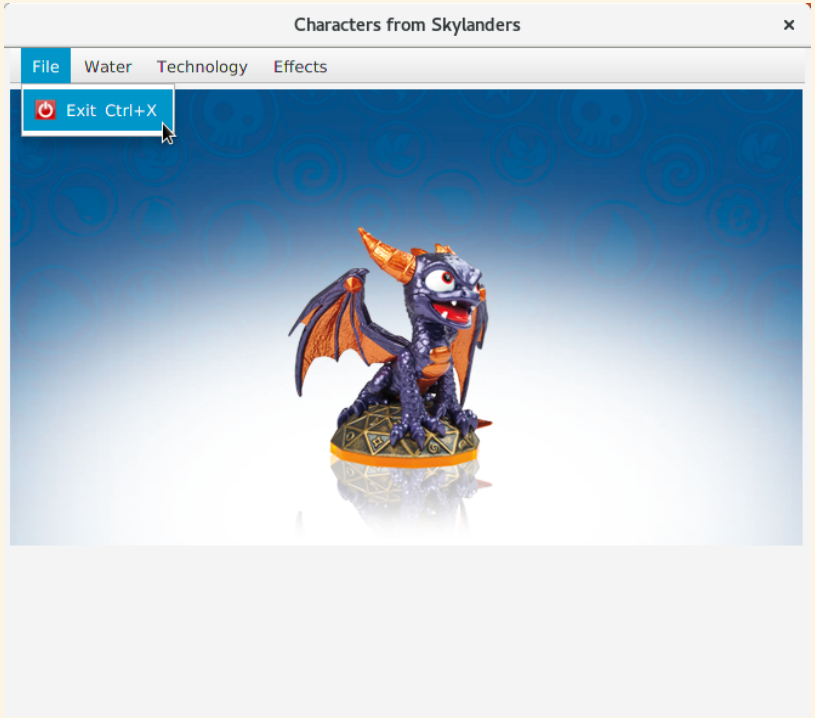
Source code, part 3

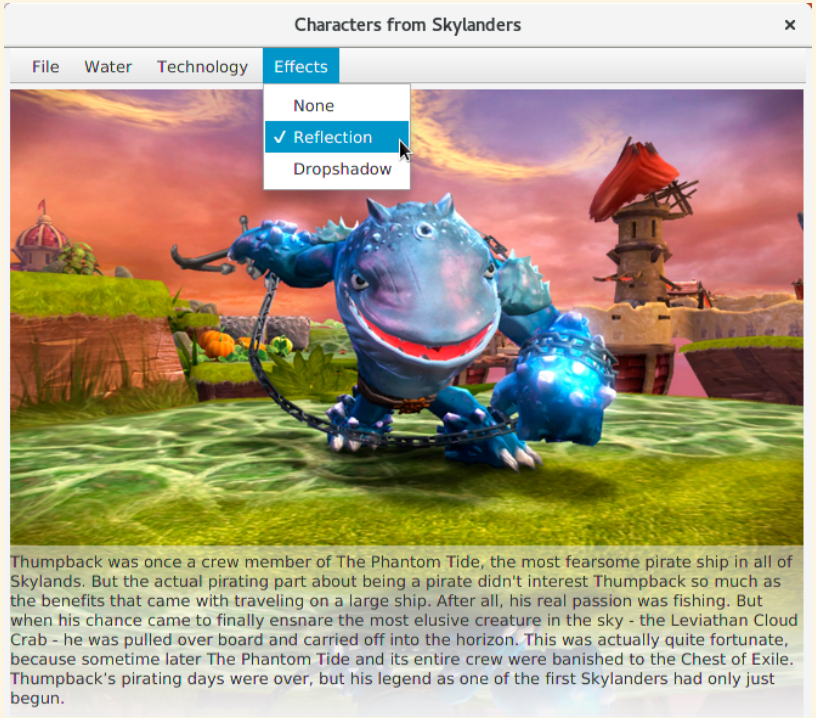
```
MenuItem trigger = new MenuItem("Trigger Happy");
trigger.setOnAction(t -> {
    theView.setImage(new Image("file:images/skylanders/triggerhappy.png"));
    theText.setText(readFile("triggerhappy.txt"));

});

ToggleGroup radioEffects = new ToggleGroup();
RadioMenuItem noEffects = new RadioMenuItem("None");
noEffects.setOnAction(e -> theView.setEffect(null));
noEffects.setToggleGroup(radioEffects);

Reflection refl = new Reflection();
refl.setFraction(0.4);

RadioMenuItem reflEffects = new RadioMenuItem("Reflection");
reflEffects.setOnAction(e -> theView.setEffect(refl));
reflEffects.setToggleGroup(radioEffects);
```

Source code, part 4

```
    DropShadow ds = new DropShadow();
    ds.setOffsetX(0.2);
    ds.setOffsetY(0.2);

    RadioMenuItem dropEffects = new RadioMenuItem("Dropshadow");
    dropEffects.setOnAction(e -> theView.setEffect(ds));
    dropEffects.setToggleGroup(radioEffects);

    menuFile.getItems().add(exitItem);
    menuWater.getItems().addAll(thumpback, chill, gill, slam);
    menuTech.getItems().addAll(bouncer, drill, sprocket, trigger);
    menuEffects.getItems().addAll(noEffects, reflEffects, dropEffects);

    VBox root = new VBox();
    root.setPadding(new Insets(0, 5, 5, 5));
    root.setSpacing(5);
    root.getChildren().addAll(menuBar, theView, theText);

    Scene scene = new Scene(root, 665, 550);

    primaryStage.setTitle("Characters from Skylanders");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Running program





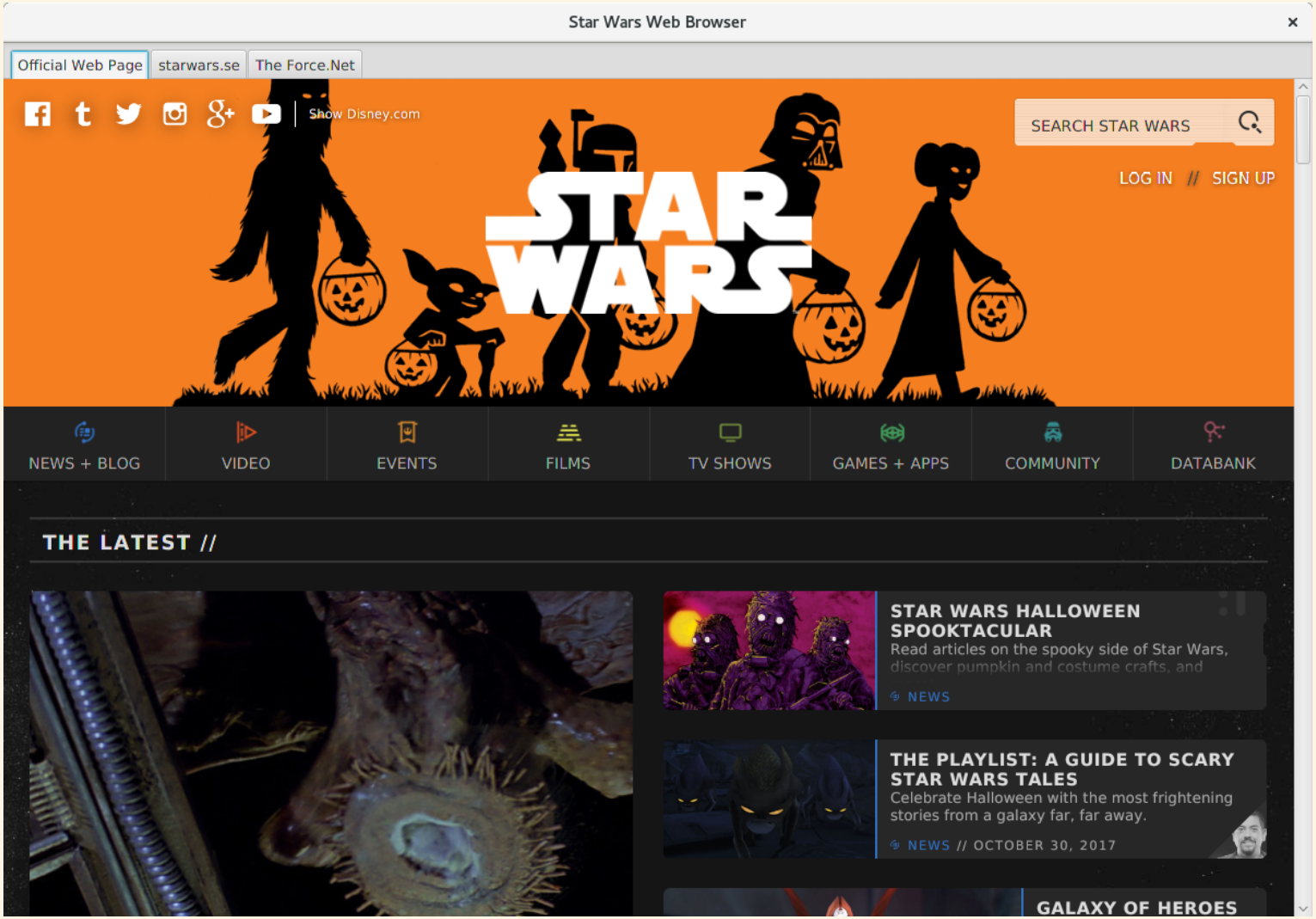
Tabs and the web

- Many modern programs also use *tabs* in addition to or instead of menus.
- JavaFX has support for creating tab based GUIs.
- Tabs are created using the class `TabPane` which is set to contain a number of `Tab` objects.
  - Each tab is given a content using the method `setContent()` and it can be any other node.
- The example will also show how to display web pages inside of a program using the `WebView` and `WebEngine` classes.
  - The underlying implementation for the web view is `WebKit`.
- The method `load()` reads a page to the engine and shows it in the view.

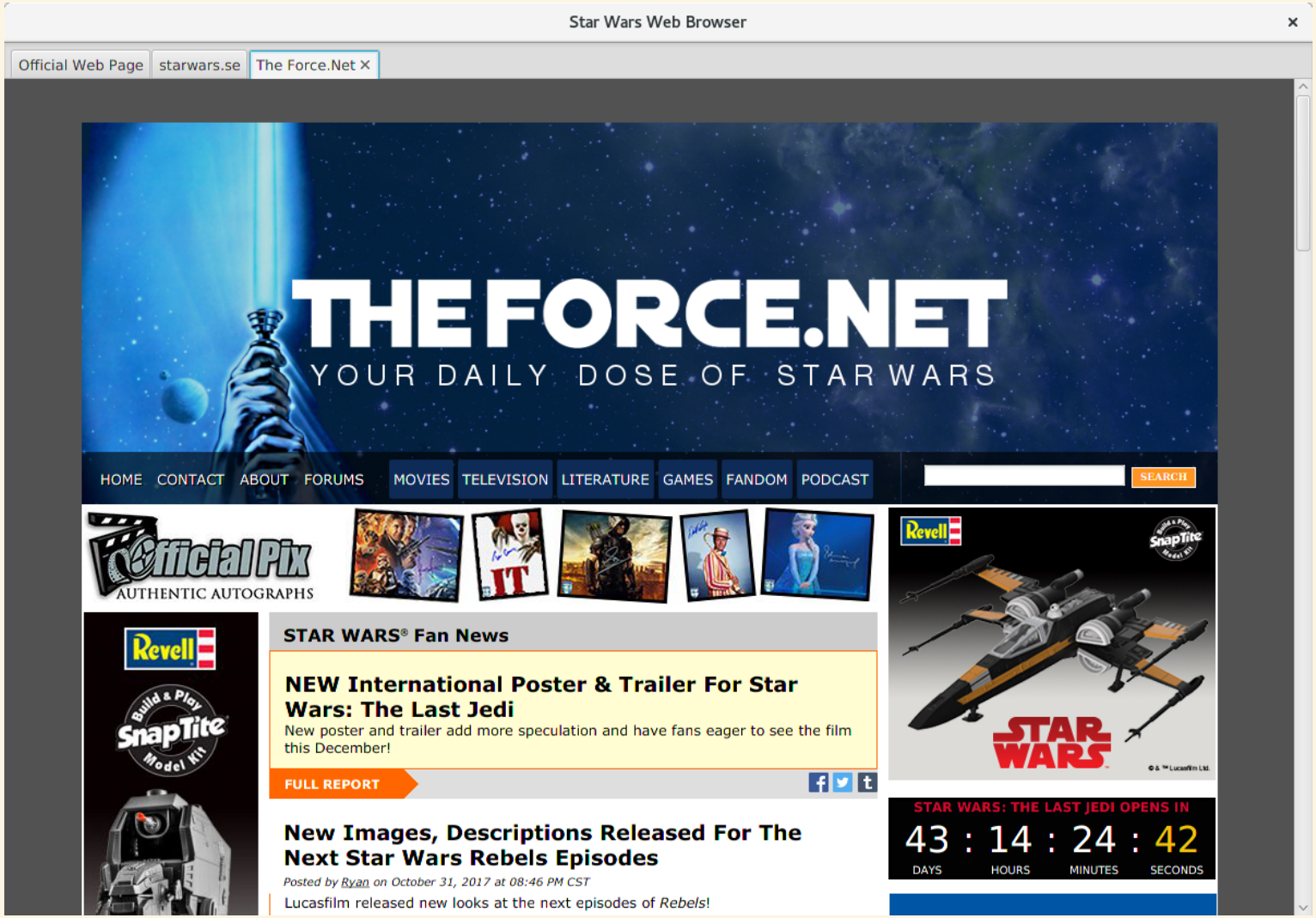
Source code

```
public void start(Stage primaryStage) {
    TabPane tabs = new TabPane();
    Tab official = new Tab("Official Web Page");
    official.setClosable(false);
    Tab swedish = new Tab("starwars.se");
    swedish.setClosable(true);
    Tab theForce = new Tab("The Force.Net");

    tabs.getTabs().addAll(official, swedish, theForce);

    WebView browserOfficial = new WebView();
    WebEngine showOfficial = browserOfficial.getEngine();
    showOfficial.load("http://www.starwars.com");
    official.setContent(browserOfficial);

    // In the same way for the other tabs

    StackPane root = new StackPane();
    root.getChildren().addAll(tabs);
    Scene scene = new Scene(root, 1200, 800);

    primaryStage.setTitle("Star Wars Web Browser");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

First tab


Star Wars Web Browser
Official Web Page
starwars.se
The Force.Net
Show Disney.com
SEARCH STAR WARS
LOG IN // SIGN UP
STAR WARS
NEWS + BLOG
VIDEO
EVENTS
FILMS
TV SHOWS
GAMES + APPS
COMMUNITY
DATABANK
THE LATEST //
STAR WARS HALLOWEEN SPOOKTACULAR
Read articles on the spooky side of Star Wars, discover pumpkin and costume crafts, and
NEWS
THE PLAYLIST: A GUIDE TO SCARY STAR WARS TALES
Celebrate Halloween with the most frightening stories from a galaxy far, far away.
NEWS // OCTOBER 30, 2017
GALAXY OF HEROES

Last tab


Star Wars Web Browser
Official Web Page
starwars.se
The Force.Net
THE FORCE.NET
YOUR DAILY DOSE OF STAR WARS
HOME
CONTACT
ABOUT
FORUMS
MOVIES
TELEVISION
LITERATURE
GAMES
FANDOM
PODCAST
SEARCH
Official Pix
AUTHENTIC AUTOGRAPHS
STAR WARS® Fan News
NEW International Poster & Trailer For Star Wars: The Last Jedi
New poster and trailer add more speculation and have fans eager to see the film this December!
FULL REPORT
New Images, Descriptions Released For The Next Star Wars Rebels Episodes
Posted by Ryan on October 31, 2017 at 08:46 PM CST
Lucasfilm released new looks at the next episodes of Rebels!
STAR WARS: THE LAST JEDI OPENS IN
43 : 14 : 24 : 42
DAYS
HOURS
MINUTES
SECONDS

Multimedia

- There has been quite a lot of work in JavaFX to get multimedia to work.
- Both in the use of hardware using OpenGL/DirectX and for support of different formats.
- For music the following formats are supported:
  - MP3; AIFF containing uncompressed PCM; WAV containing uncompressed PCM; MPEG-4 multimedia container with Advanced Audio Coding (AAC) audio
- For video the following formats are supported:
  - FLV containing VP6 video and MP3 audio; MPEG-4 multimedia container with H.264/AVC (Advanced Video Coding) video compression
- For both audio and video JavaFX support a large number of features like seeking and progressive download.

Play an MP3, part 1

```
public void start(Stage primaryStage) throws URISyntaxException {
     primaryStage.setTitle("Comfortably Numb");
     File path = new File("media/ComfortablyNumb.mp3");
     Media music = new Media(new File(path.getAbsolutePath()).toURI().toString());
     MediaPlayer musicPlayer = new MediaPlayer(music);

     Image pf = new Image("file:images/pfwall.jpg");
     ImageView pfView = new ImageView(pf);

     VBox layout = new VBox();
     layout.setPadding(new Insets(5));
     layout.setSpacing(5);

     HBox buttons = new HBox();
     buttons.setPadding(new Insets(5));
     buttons.setSpacing(10);
```

Play an MP3, part 2

```
    Button playPause = new Button("Play/Pause");
    playPause.setOnAction(event -> {
        Status status = musicPlayer.getStatus();

        if(status == Status.PAUSED || status == Status.STOPPED || status == Status.READY)
            musicPlayer.play();
        else
            musicPlayer.pause();
    });

    Button solo = new Button("To the solo!");
    solo.setOnAction(event -> {
        musicPlayer.seek(Duration.millis(270000));
    });

    buttons.getChildren().addAll(playPause, solo);

    layout.getChildren().addAll(pfView, buttons);

    Scene scene = new Scene(layout, 325, 370);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

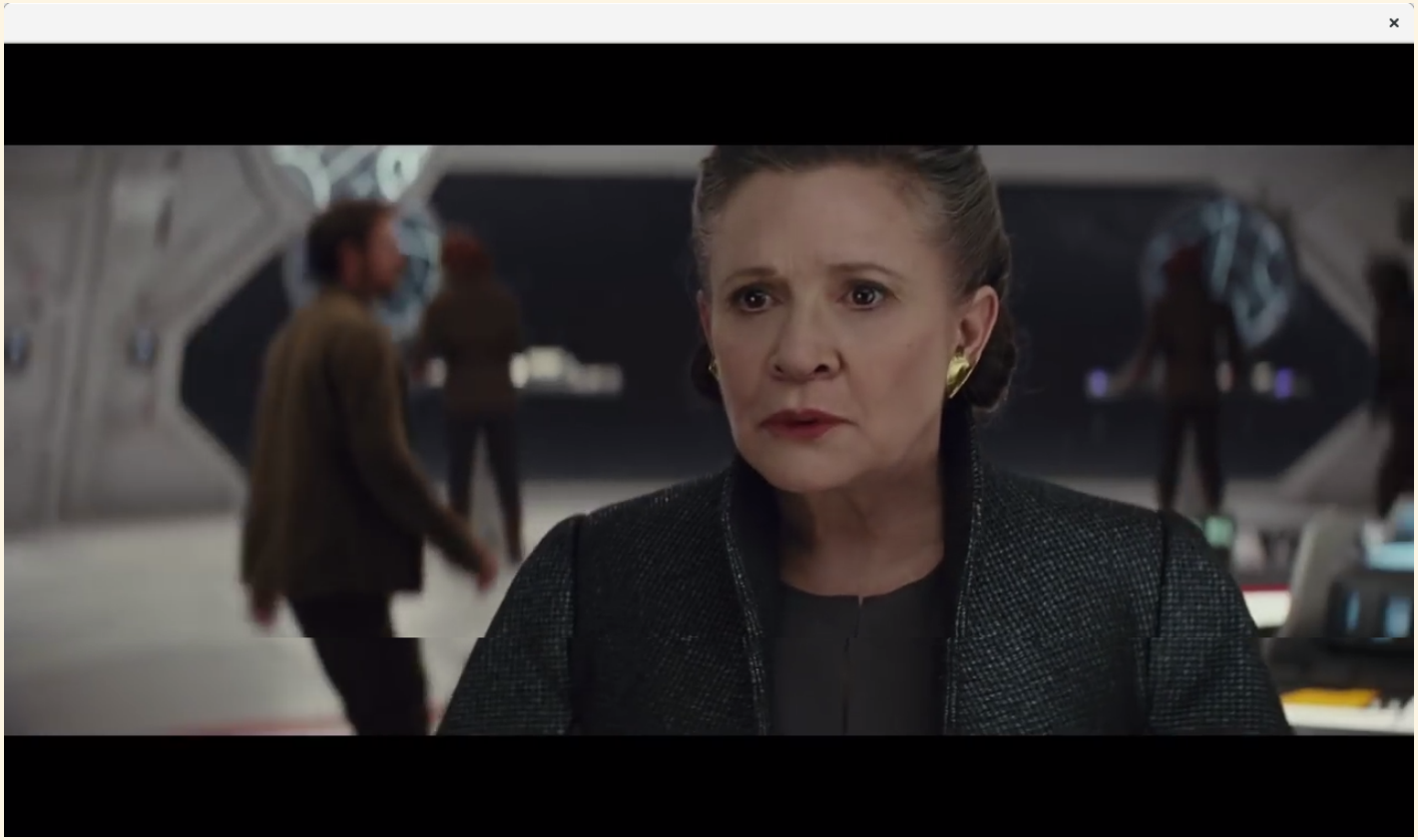
Running program

- It would have been better with sound...

Play a video

```
public void start(Stage primaryStage) {
    File path = new File("media/LastJediTrailer.mp4");
    Media swTrailer = new Media(new File(path.getAbsolutePath()).toURI().toString());
    MediaPlayer swMovie = new MediaPlayer(swTrailer);
    MediaView trailer = new MediaView(swMovie);
    swMovie.setAutoPlay(true);

    Group root = new Group();
    root.getChildren().addAll(trailer);
    Scene scene = new Scene(root, 1280, 720);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Running program



Mouse and Keyboard

More on events

- Events and listeners for events is an important part of programming GUI in Java.
- Previously we have seen the “click” event for the button.
  - Hidden in the `setOnAction()` method of the button.
- The button itself is called an *event source*.
- For anything to happen an `ActionEvent` is fired and managed by an `EventHandler`.
- It is more evident when using the old, pre-lambda, style of creating a method for a button:

```
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override public void handle(ActionEvent e) {
        label.setText("Pressed!");
    }
});
```

Register an event

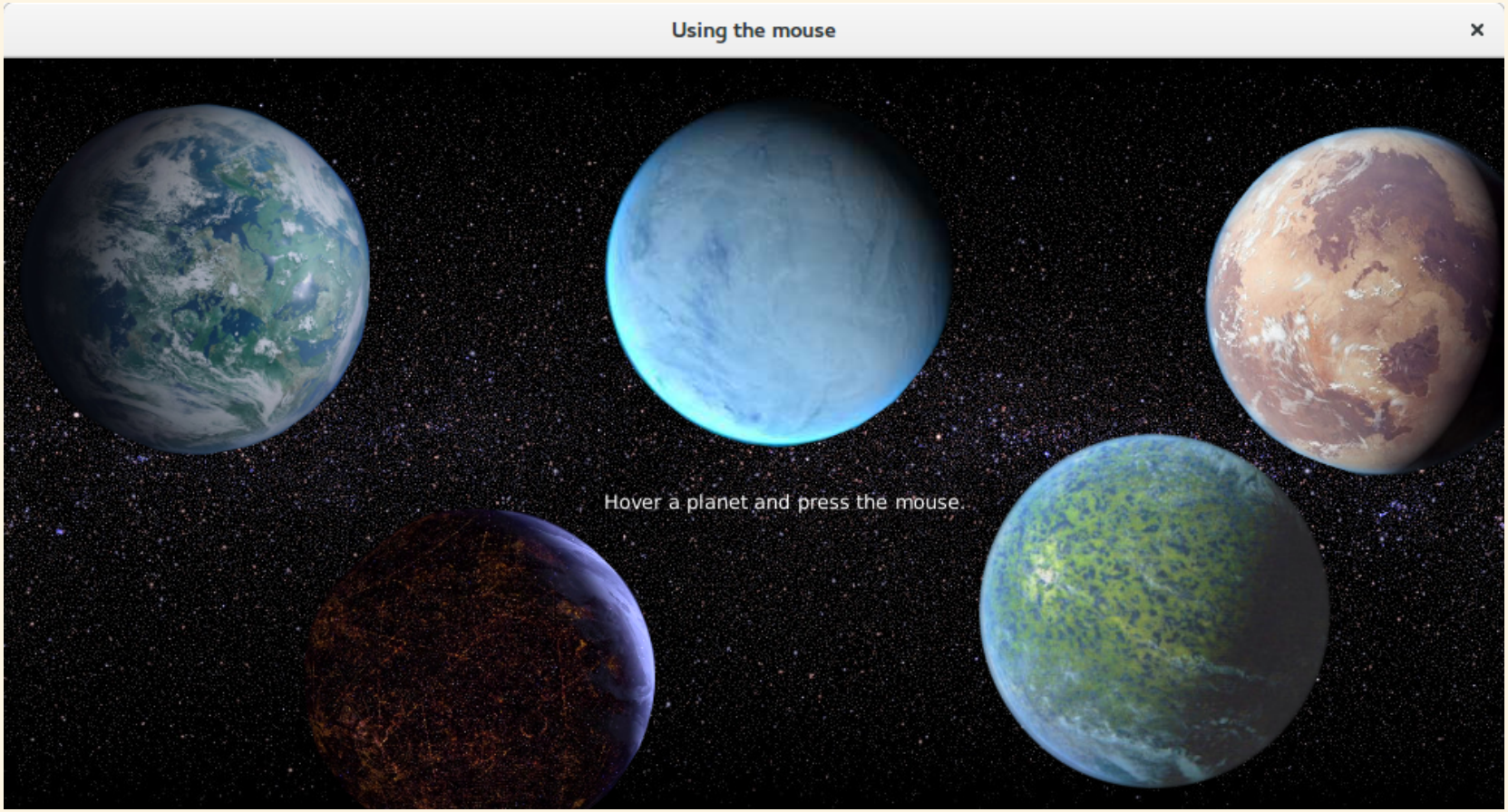
- A handler as seen before, must be registered with an event source (the button).
- It must also be of the appropriate type (an interface).
- In Java the event handler is also called a *listener*.
- Apart from the button, many other controls are possible as event sources, and many event sources can have several listeners attached to it.
- This is how Java works when reacting to other events as well, like mouse buttons and keyboard presses.

Event handling in Java

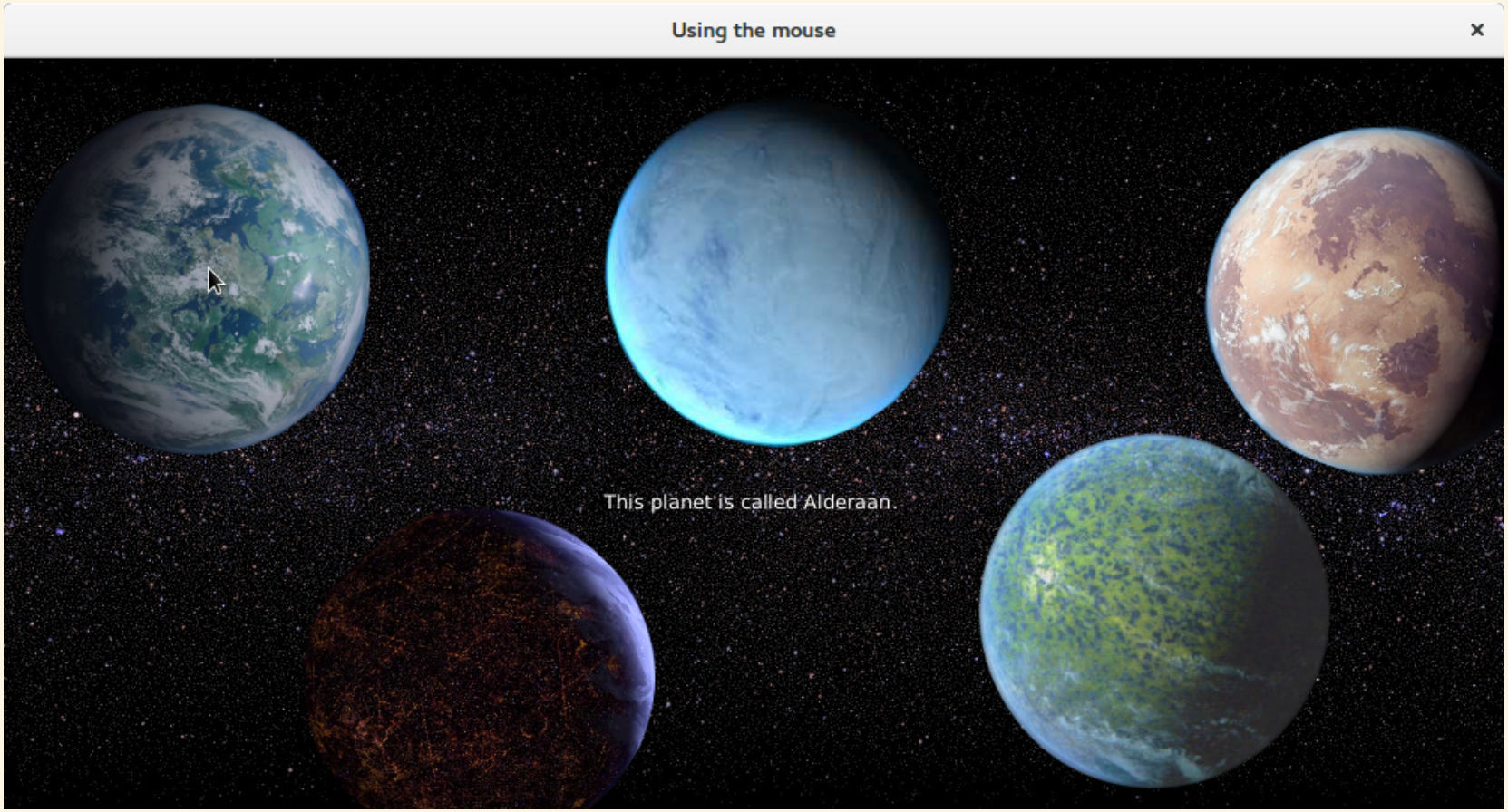
- Below is a list of *some* of the different combinations that are available.

User Action	*Source object*	*Event type fired*	*Event registration method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

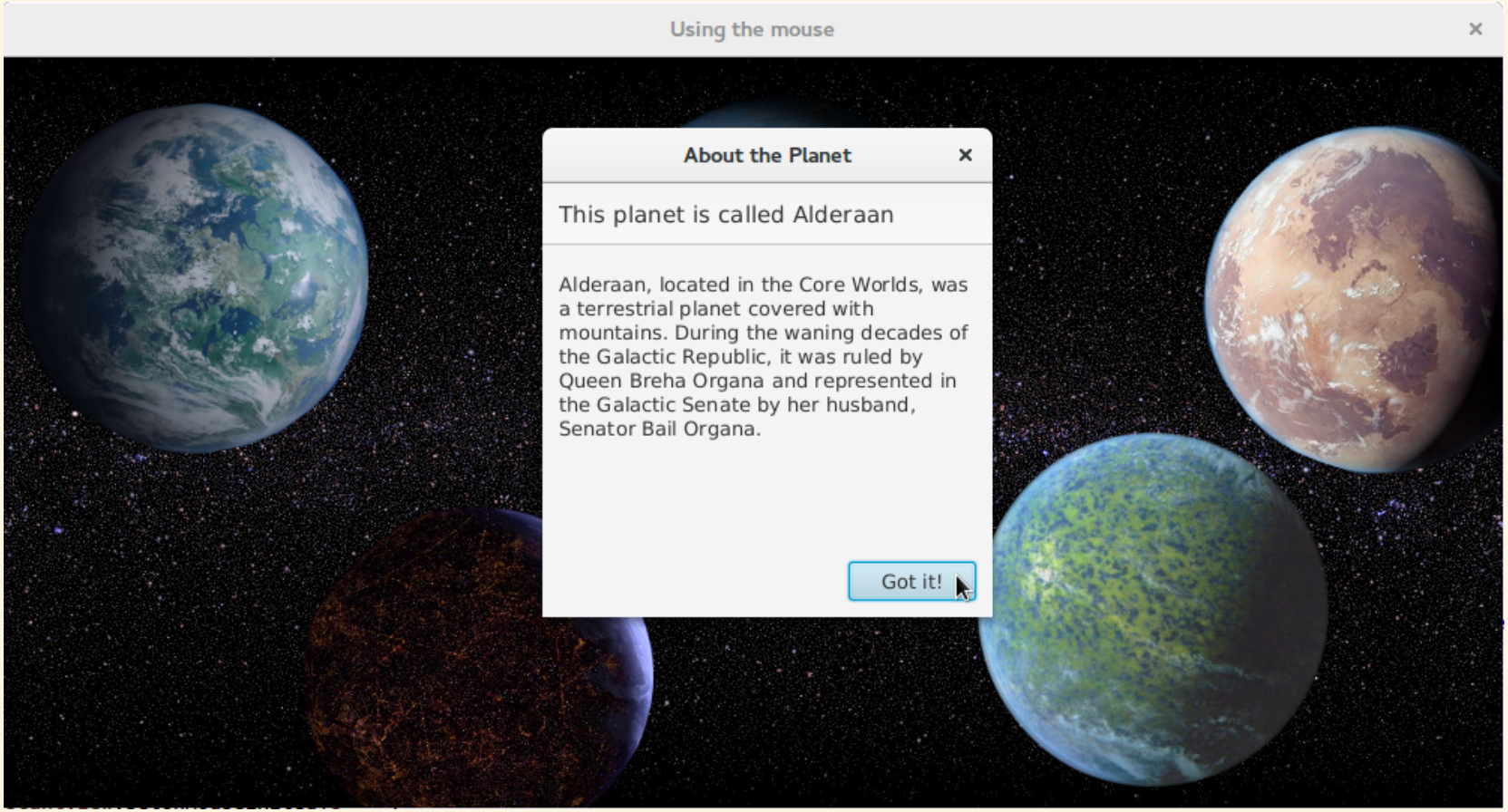
Example



Example



Example


Using the mouse
About the Planet
This planet is called Alderaan
Alderaan, located in the Core Worlds, was a terrestrial planet covered with mountains. During the waning decades of the Galactic Republic, it was ruled by Queen Breha Organa and represented in the Galactic Senate by her husband, Senator Bail Organa.
Got it!

Setting up the window

```
primaryStage.setTitle("Using the mouse");
Group layout = new Group();
Text thePlanet = new Text(400, 300, "Hover a planet and press the mouse.");
thePlanet.setFill(Color.WHITE);
Image alderaan = new Image("file:a.png");
Image coruscant = new Image("file:c.png");
Image hoth = new Image("file:h.png");
Image takodana = new Image("file:t.png");
Image tatooine = new Image("file:tat.png");
Image space = new Image("file:space.jpg");
ImageView theView = new ImageView(space);
ImageView alderaanView = new ImageView(alderaan);
alderaanView.setLayoutX(10);
alderaanView.setLayoutY(30);
ImageView coruscantView = new ImageView(coruscant);
coruscantView.setLayoutX(200);
coruscantView.setLayoutY(300);
ImageView hothView = new ImageView(hoth);
hothView.setLayoutX(400);
hothView.setLayoutY(25);
ImageView takodanaView = new ImageView(takodana);
takodanaView.setLayoutX(650);
takodanaView.setLayoutY(250);
ImageView tatooineView = new ImageView(tatooine);
tatooineView.setLayoutX(800);
tatooineView.setLayoutY(45);
```

Custom dialogue and mouse events

```
Dialog<String> dialogue = new Dialog<>();
dialogue.setTitle("About the Planet");
ButtonType okButton = new ButtonType("Got it!", ButtonData.OK_DONE);
dialogue.getDialogPane().getButtonTypes().addAll(okButton);

alderaanView.setOnMouseEntered(e -> {
    thePlanet.setText("This planet is called Alderaan.");
});
alderaanView.setOnMouseExited(e -> {
    thePlanet.setText("Hover a planet and press the mouse.");
});
alderaanView.setOnMouseClicked(e -> {
    dialogue.setHeaderText("This planet is called Alderaan");
    VBox spacing = new VBox();
    spacing.setPadding(new Insets(10));
    Text info = new Text("Alderaan, located in the Core Worlds, was a terrestrial planet covered with mo
    info.setWrappingWidth(300);
    spacing.getChildren().add(info);
    dialogue.getDialogPane().setContent(spacing);
    dialogue.showAndWait();
});
```

Keyboard events

- Reading the keys of the keyboard is done using the `KeyEvent` which has a method called `getCode()` that holds the key pressed.
- The returned key can be compared against constants in `KeyCode`, some examples are:

Constant	*Description*	*Constant*	*Description*
`KeyCode.HOME`	The Home key	`KeyCode.CONTROL`	The Control key
`KeyCode.TAB`	The Tab key	`KeyCode.ESCAPE`	The Esc key
`KeyCode.UP`	The up-arrow key	`KeyCode.DOWN`	The down-arrow key
`KeyCode.A`	The 'a' key	`KeyCode.9`	The '9' key

- It is also possible to combine keys, like CTRL-R and similar, but then use the class `KeyCodeCombination`.
  - It takes a main key as first parameter and any number of modifiers after that.

Responding to a press

- Just as with previous actions, it is possible to use Java 8 lambdas to reduce the amount of code.
- The sacrifice done then, is that it is hard to understand what is happening.
- The pre-Java 8 event handling was:

```
scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
    public void handle(KeyEvent ke) {
        if(ke.getCode() == KeyCode.UP)
            // Do something
    }
});
```

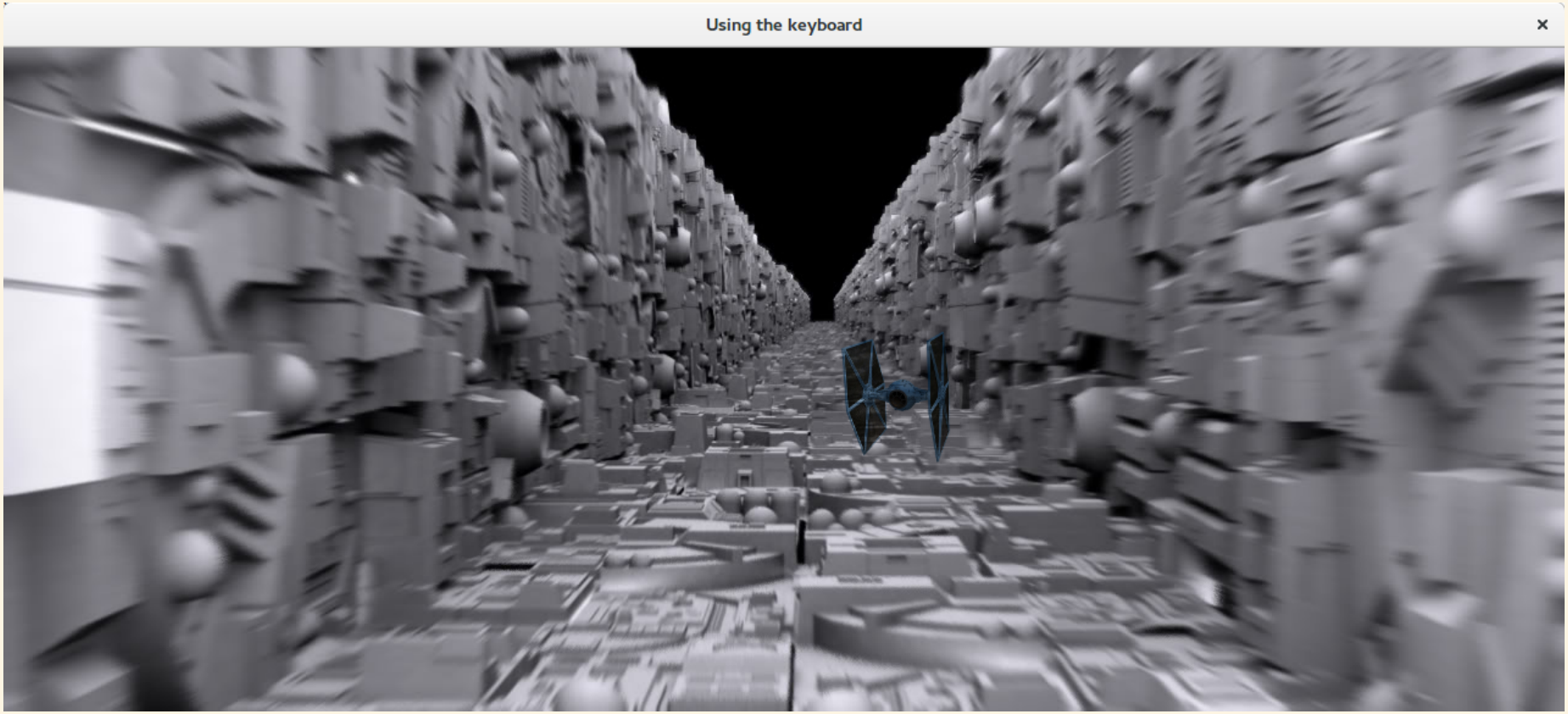
- The Java 8 way is the rather shorter:

```
scene.setOnKeyPressed(e -> {
     if(e.getCode() == KeyCode.UP)
        // Do something
});
```

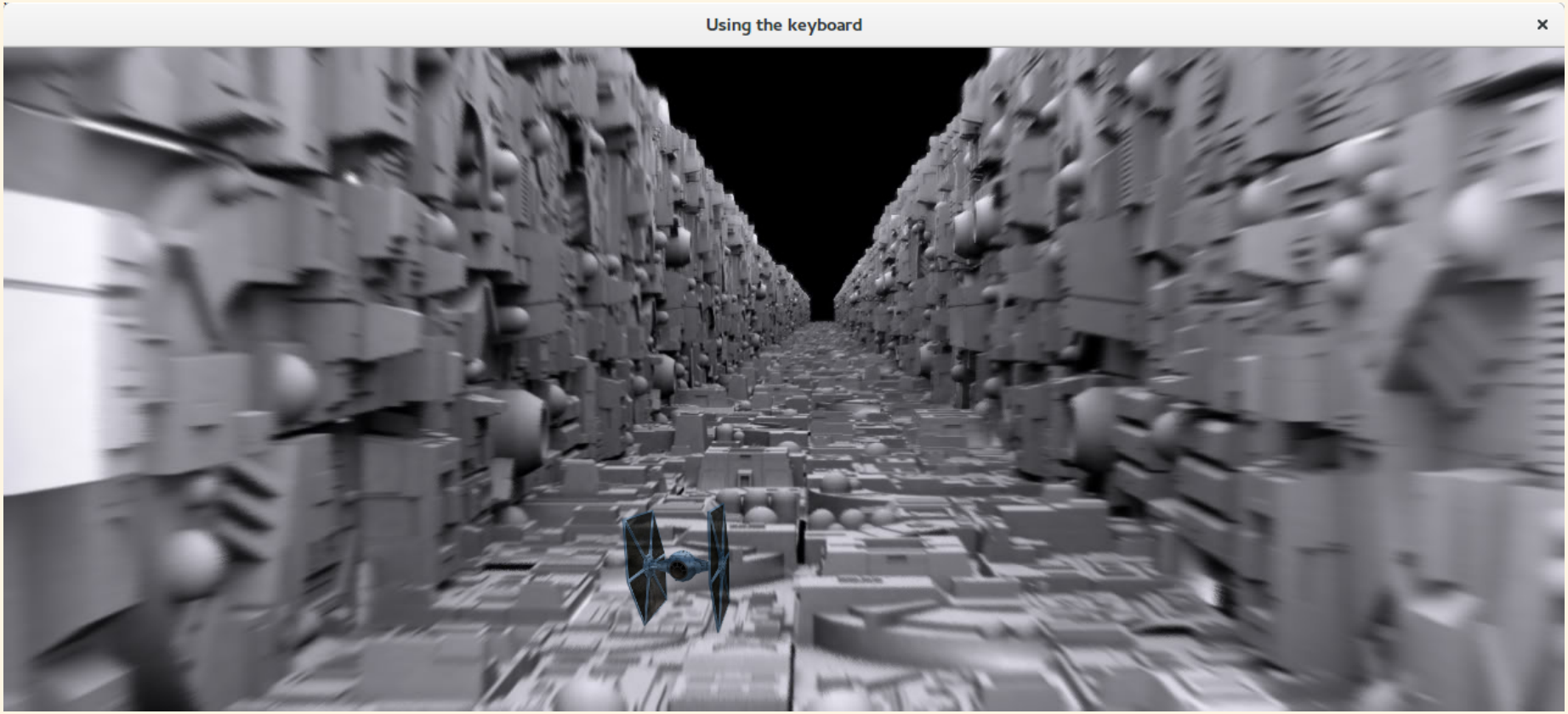
Example

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("Using the keyboard");
    Group layout = new Group();
    Image tie = new Image("file:images/tie.png");
    ImageView theTie = new ImageView(tie);
    theTie.setLayoutX(512);
    theTie.setLayoutY(350);
    Image space = new Image("file:images/trench.jpg");
    ImageView theView = new ImageView(space);
    layout.getChildren().addAll(theView, theTie);

    Scene scene = new Scene(layout, 1280, 544);

    scene.setOnKeyPressed(e -> {
        if(e.getCode() == KeyCode.UP)
            theTie.setLayoutY(theTie.getLayoutY()-10);
        else if (e.getCode() == KeyCode.DOWN)
            theTie.setLayoutY(theTie.getLayoutY()+10);
        else if (e.getCode() == KeyCode.LEFT)
            theTie.setLayoutX(theTie.getLayoutX()-10);
        else if (e.getCode() == KeyCode.RIGHT)
            theTie.setLayoutX(theTie.getLayoutX()+10);
    });

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

In graphics



In graphics

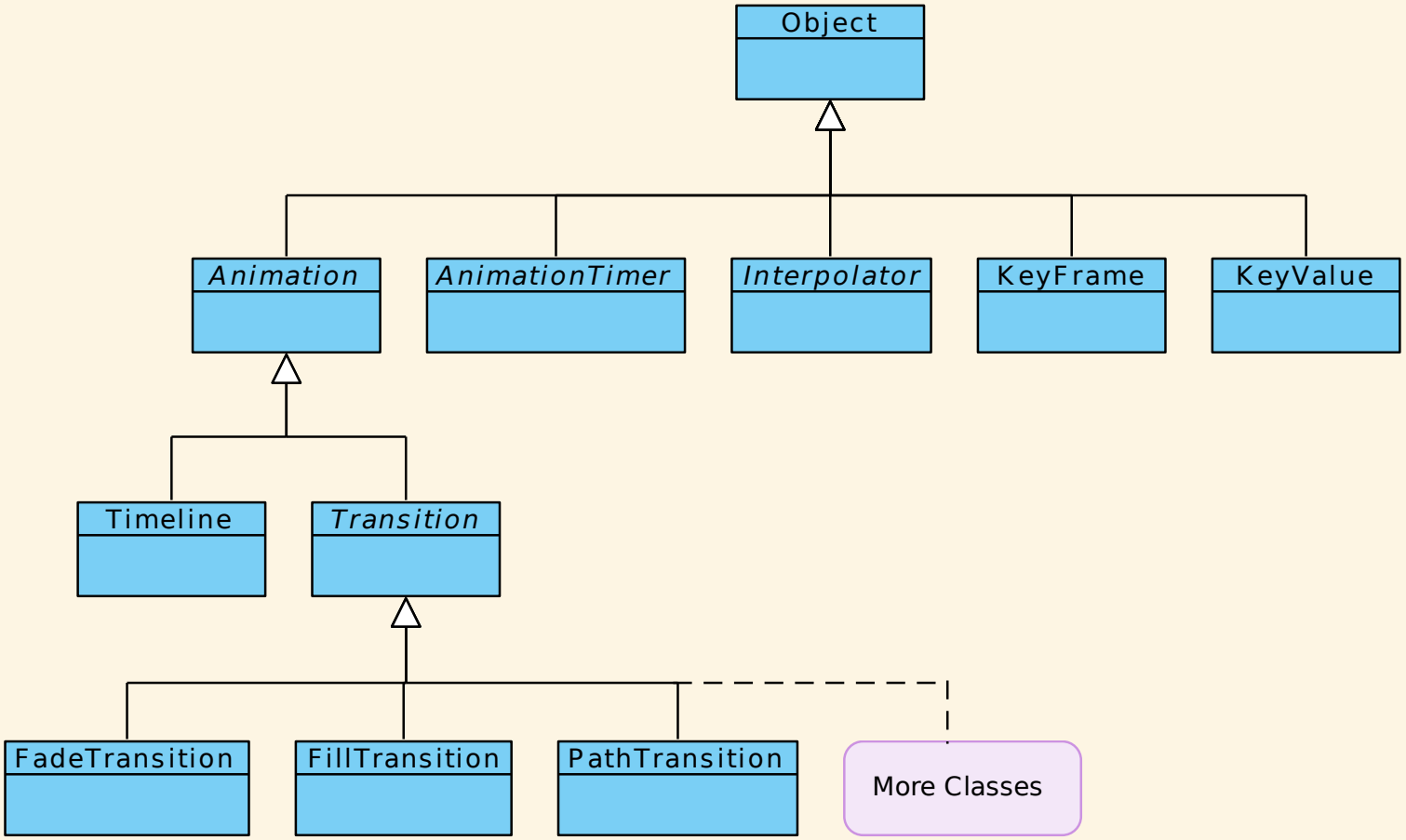


Animation

Animation in JavaFX

- One part of JavaFX that has been thoroughly updated and optimised is *animation*.
- With JavaFX hardware acceleration and dedicated animation classes were added.
- The abstract classes `Animation` and `AnimationTimer` encapsulate much of the functionality and they have several concrete subclasses:
  - `Timeline` defines a number of frames.
  - `Transition` with a number of subclasses:
    - `PathTransition` for following a predefined path.
    - `FadeTransition` will fade between different images, texts or other.
- For best control, it is also possible to create your own subclass from `Animation` or `AnimationTimer`.

The JavaFX Animation Hierarchy



Animation class

- The `Animation` class inherits directly from `Object` and not the `javafx` packages.
- In this class a number of fields are declared:
  - `autoReverse` defines if the animation should reverse when done.
  - `cycleCount` defines the number of times a animation should be played.
  - `rate` defines the speed and direction for the animation, negative for reverse.
  - `status` (read only) tells the status of the animation.
- The methods `start()`, `stop()` and `pause()` control the status of the animation.
- The easiest way of creating animations is through on of the `Transition` subclasses.

Transitions

- The main idea behind transitions is to have a change of state over time.
- This is done via an internal timeline, in contrast to other animations.
- The `Transition` class is abstract and has several concrete sub classes.
  - `FadeTransistion`
  - `RotateTransition`
  - `PathTransition`
- All of them work on `Nodes`, so most elements can be used.
  - Images, text and so on.
- All transitions set a *duration* for the internal timeline.

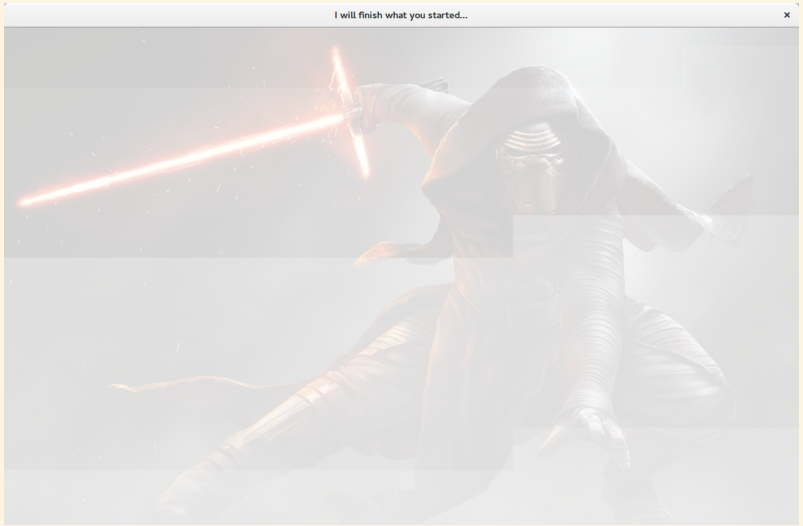
FadeTransistion

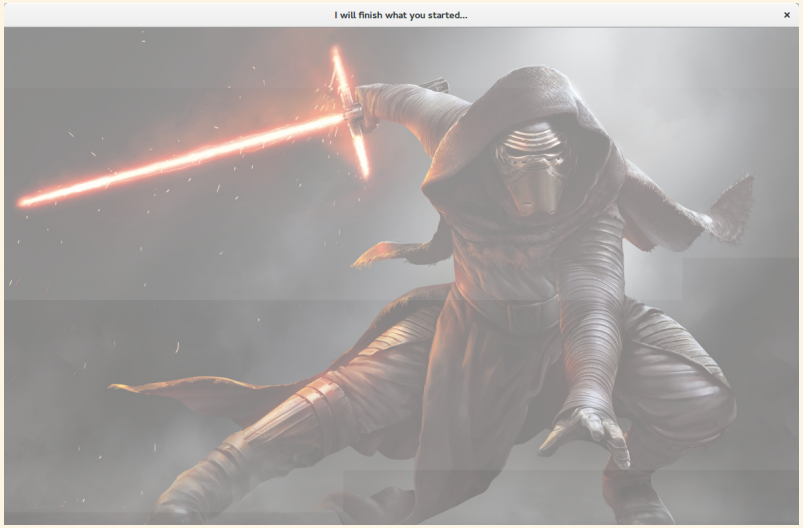
- The `FadeTransistion` makes it possible to fade a node.
- For the node to fade, the start and end values are set.
  - A `double` going from 0.0 (invisible) to 1.0 (fully visible).
- A duration is set for the entire fade, but also an increment for each step in the fade.
  - This is also a `double` from 0.0 to 1.0.
- It is also possible to set it to cycle and to reverse when at the end.
- When the transition is set, the `play()` method will start the animation.

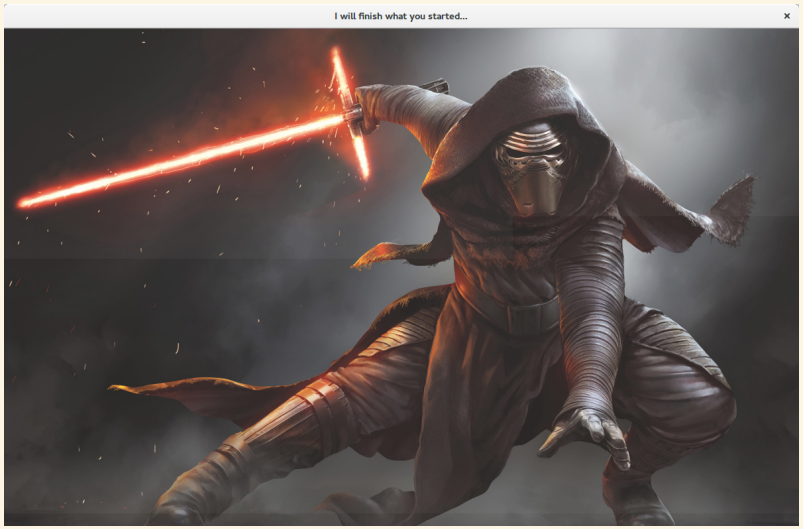
In code

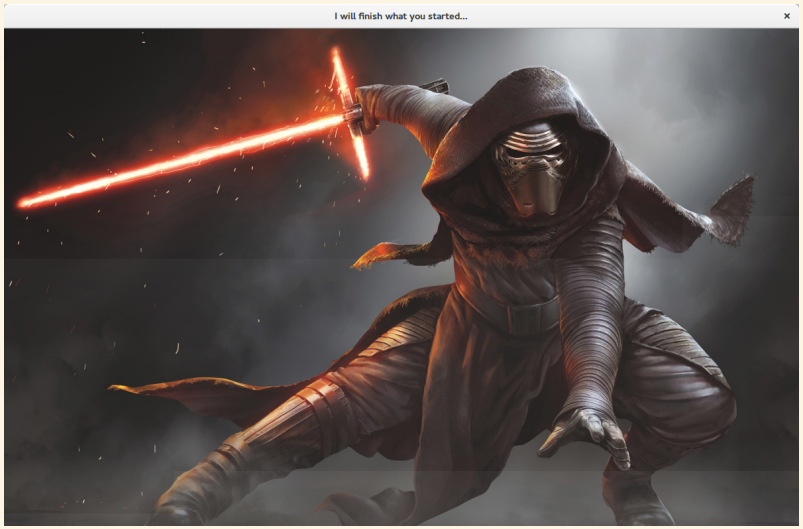
```
public void start(Stage primaryStage) {
    primaryStage.setTitle("I will finish what you started...");
    Image kylo = new Image(getClass().getResourceAsStream("kyloren.jpg"));
    ImageView kyloRen = new ImageView(kylo);

    FadeTransition fadeToBlack =
            new FadeTransition(Duration.millis(4000), kyloRen);
    fadeToBlack.setFromValue(0.0);
    fadeToBlack.setToValue(1.0);
    fadeToBlack.setByValue(0.3);
    fadeToBlack.setCycleCount(Animation.INDEFINITE);
    fadeToBlack.setAutoReverse(true);
    fadeToBlack.play();

    Scene scene = new Scene(new Group(kyloRen), 1200, 750);

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

In graphics


I will finish what you started...


I will finish what you started...


I will finish what you started...


I will finish what you started...

`RotateTransition`

- To create an animation of a rotation, it is possible to use the class `RotateTransition`.
- The object of `RotateTransition` is given values for:
  - Angle – the complete change from the initial state, 360 for a full circle (obviously).
  - A cycle count for the number of times it needs to be done.
- In the example we also set the *interpolation*.
- This can be done using either a separate class or as a method to the transition.
  - It decides the start and end movement of the transition.

The code

```
final Image itsMe = new Image("jag.png");
final ImageView showMe = new ImageView(itsMe);

RotateTransition snurr =
      new RotateTransition(Duration.millis(3000), showMe);
snurr.setByAngle(360);
snurr.setCycleCount(Animation.INDEFINITE);
snurr.setAutoReverse(true);
snurr.setInterpolator(Interpolator.EASE_BOTH);
snurr.play();

Scene scene = new Scene(new Group(showMe), 500, 400);

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

In graphics

PathTransition

- It is also possible to set up a transition over a path.
- The path will then be defined by using a number of path classes like `MoveTo`, `LineTo` and `CubicCurveTo`.
- In the example we only set a path and let our node move over it, put it is possible to make it follow a mouse click, a key press or anything else.
  - Also notice that the previous transition is still in effect.
- When the path is set, a `PathTransition` object must be created taking the path as a parameter.
  - As well as the node to animate.
- For the transition the orientation is set, in this case to `NONE` which means that it will only follow the path.
- The example is fairly complex and borrowed from a book.

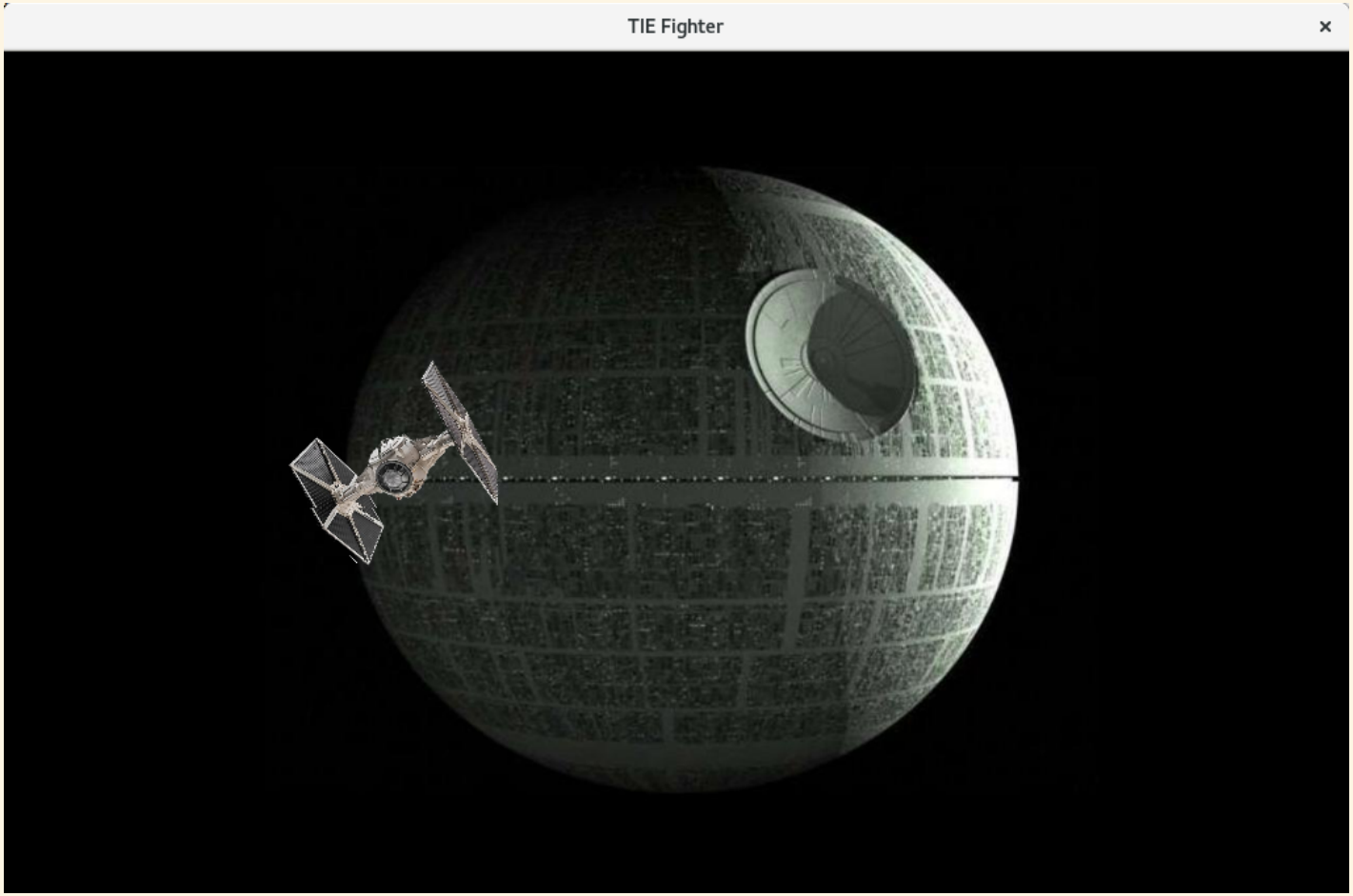
The code

```
PathTransition pathTransition = new PathTransition();

Path path = new Path();
path.getElements().add(new MoveTo(50, 50));
path.getElements().add(new LineTo(800, 400));
path.getElements().add(new LineTo(50, 600));
path.getElements().add(new CubicCurveTo(580, 0, 580, 120, 200, 120));
path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240));

path.setVisible(false);

ImageView tie = new ImageView(new Image("tiefighter.png"));
tie.setFitWidth(150.0);
tie.setPreserveRatio(true);
```

The code, cont.

```
pathTransition.setDuration(Duration.seconds(10));
pathTransition.setPath(path);
pathTransition.setNode(tie);
pathTransition.setOrientation(OrientationType.ORTHOGONAL_TO_TANGENT);
pathTransition.setCycleCount(Timeline.INDEFINITE);
pathTransition.setAutoReverse(true);

pathTransition.play();

primaryStage.setTitle("TIE Fighter");

StackPane root = new StackPane();
String image = "deathstar.jpg";
root.setStyle("-fx-background-image: url('" +
        image
        + "'); -fx-background-position: center center; -fx-background-repeat: stretch;");
root.setAlignment(Pos.TOP_LEFT);
root.getChildren().addAll(tie, path);
primaryStage.setScene(new Scene(root, 1024, 768));
primaryStage.show();
pathTransition.play();
```

In graphics



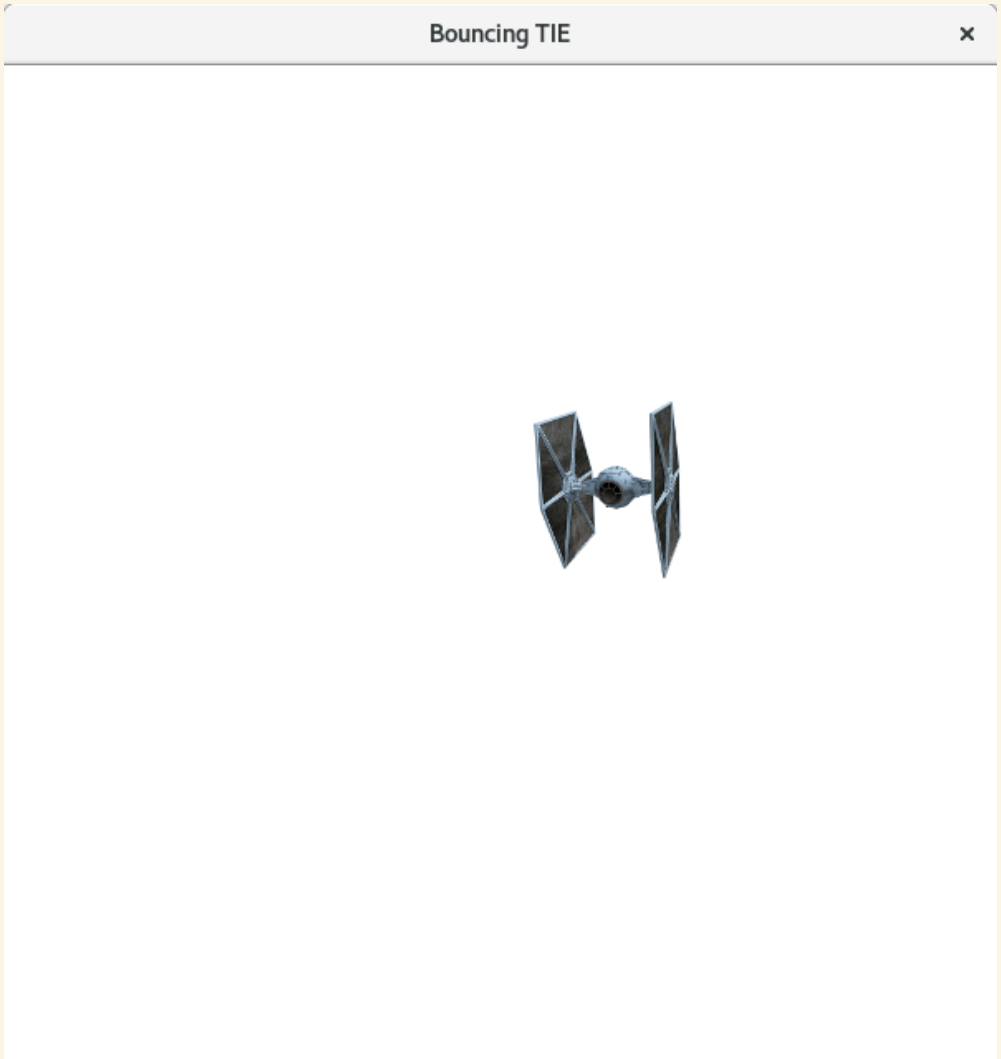
Translation animation

- The class `TranslateTransition` makes it possible to change position on pixel level.
  - *Translation* is often used as a term when saying that something should be moved.
- A part from that, the class works very much as the previous classes in that they define the time an animations should take.
  - As well as the start and end positions.
- The example uses the width of the image, and it must be set, as the “edge” and changes direction.

Source code

```
public void start(Stage primaryStage) {
    ImageView tie = new ImageView(new Image("file:images/tie.png"));
    tie.setFitWidth(100.0);
    tie.setPreserveRatio(true);

    Group root = new Group();
    root.getChildren().add(tie);

    Scene scene = new Scene(root, 600, 600);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Bouncing TIE");
    primaryStage.show();

    TranslateTransition t = new TranslateTransition(
            Duration.millis(2000), tie);
    t.setFromX(tie.getX());
    t.setToX(scene.getWidth() - tie.getFitWidth());
    t.setFromY(scene.getHeight() / 3);
    t.setToY(scene.getHeight() / 3);
    t.setCycleCount(Transition.INDEFINITE);
    t.setAutoReverse(true);
    t.setInterpolator(Interpolator.LINEAR);
    t.play();
}
```

Running program



Combining transitions and translations

- It is possible to combine different translations and transitions in an animation using JavaFX.
  - This can be needed in games, for example, when different things should be moving at the same time.
- There are two classes for combining:
  - `SequentialTransition` for running animations after each other.
  - `ParallelTransition` for running animations concurrently.
- Create animations as previously and send them to one of the above mentioned classes.
  - Use `play()` to start but be sure not to use `play()` on the individual transitions, only the combined.

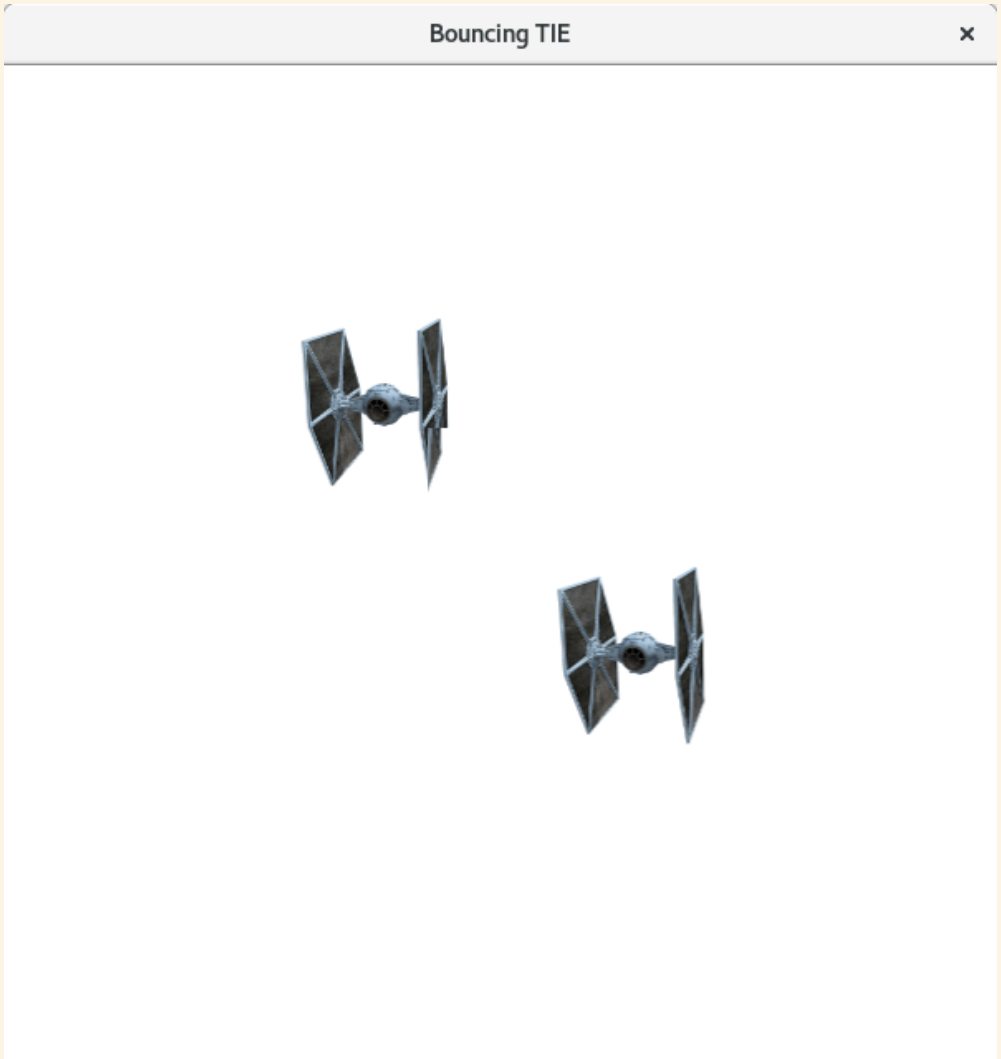
Source code, part 1

```
public void start(Stage primaryStage) {
     ImageView tie1 = new ImageView(new Image("file:images/tie.png"));
     tie1.setFitWidth(100.0);
     tie1.setPreserveRatio(true);

     ImageView tie2 = new ImageView(new Image("file:images/tie.png"));
     tie2.setFitWidth(100.0);
     tie2.setPreserveRatio(true);

     Group root = new Group();
     root.getChildren().addAll(tie1, tie2);

     Scene scene = new Scene(root, 600, 600);
     primaryStage.setScene(scene);
     primaryStage.setTitle("Bouncing TIE");
     primaryStage.show();
```

Source code, part 2

```
    TranslateTransition t1 = new TranslateTransition(
            Duration.millis(2000), tie1);
    t1.setFromX(tie1.getX());
    t1.setToX(scene.getWidth() - tie1.getFitWidth());
    t1.setFromY(scene.getHeight() / 4);
    t1.setToY(scene.getHeight() / 4);
    t1.setCycleCount(Transition.INDEFINITE);
    t1.setAutoReverse(true);
    t1.setInterpolator(Interpolator.LINEAR);

    TranslateTransition t2 = new TranslateTransition(
            Duration.millis(2000), tie2);
    t2.setFromX(scene.getWidth() - tie2.getFitWidth());
    t2.setToX(0.0);
    t2.setFromY(scene.getHeight() / 2);
    t2.setToY(scene.getHeight() / 2);
    t2.setCycleCount(Transition.INDEFINITE);
    t2.setAutoReverse(true);
    t2.setInterpolator(Interpolator.LINEAR);

    ParallelTransition pt = new ParallelTransition(t1, t2);
    pt.play();
}
```

Running the new program

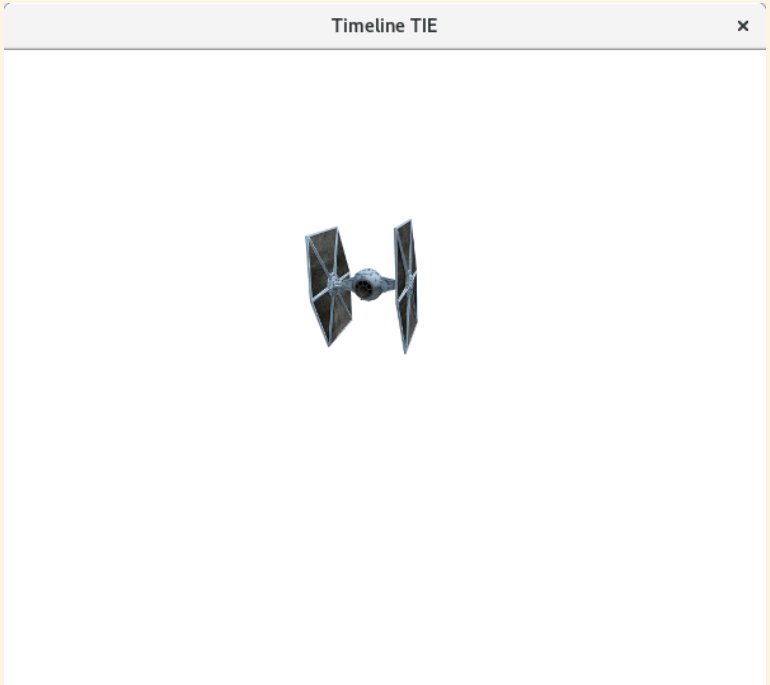


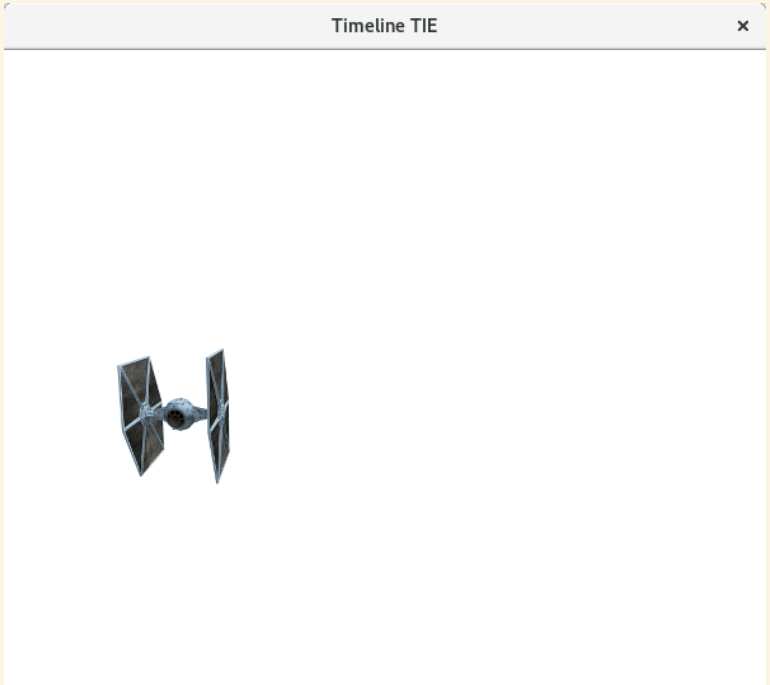
A more flexible animation

- To do more flexible animations, one of several options is to use a *timeline*.
- With a timeline it is, just as in movies, possible to define a number of *frames* that should be updated.
- Two classes are used:
  - `KeyFrame` which is the frame itself and defines what should be done.
  - `Timeline` that defines the sequence of frames.
- For each frame an update time as well as an event is defined.
  - An `ActionEvent` is performed for each frame.
- The timeline then starts the animation.

```
public void start(Stage primaryStage) {
    Group root = new Group();

    tie = new ImageView(new Image("file:images/tie.png"));
    tie.setFitWidth(100.0);
    tie.setPreserveRatio(true);
    tie.setX(100);
    tie.setY(100);

    root.getChildren().addAll(tie);
    Scene scene = new Scene(root, width, height);
    primaryStage.setTitle("Timeline TIE");
    primaryStage.setScene(scene);
    primaryStage.show();

    KeyFrame k = new KeyFrame(Duration.millis(10), event -> {
        tie.setX(tie.getX() + x_speed);
        tie.setY(tie.getY() + y_speed);

        if(tie.getX() <= 0 || tie.getX() >= width - tieSize){
            x_speed = -x_speed;
        }

        if(tie.getY() <= 0 || tie.getY() >= height - tieSize){
            y_speed = -y_speed;
        }
    });

    Timeline t = new Timeline(k);
    t.setCycleCount(Timeline.INDEFINITE);
    t.play();
}
```

Running program





Many TIEs!

- To combine everything to something more useful, the next example will show five TIEs bouncing not only the edges but also each others.
- The ships will now be in their own class which inherits from `ImageView`.
- Each ship will also have an array of all the other ships (not in any way optimal but works good for illustrative purposes).
- For each frame the ships are moved and collision detection is done.
  - Which will be done using the method `intersects()` which is available for all nodes.

Main program

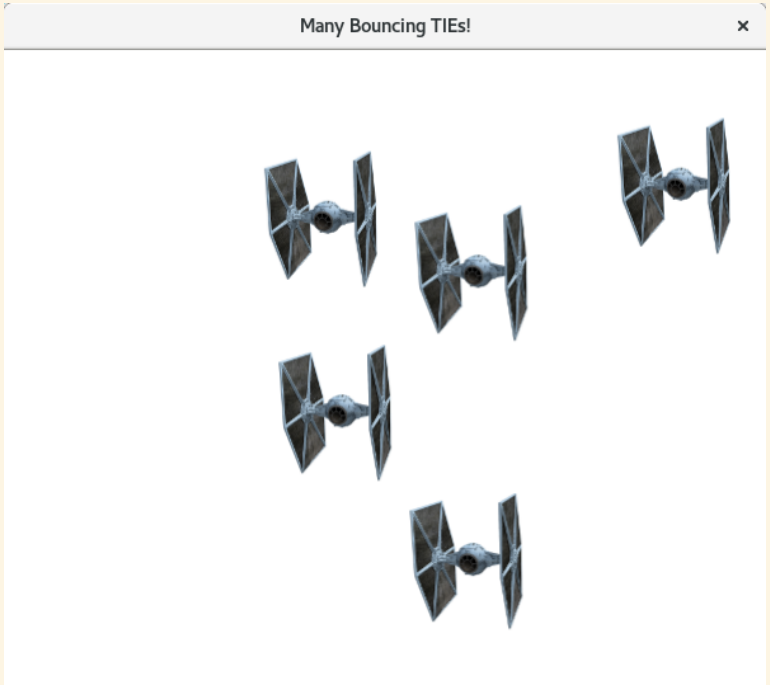
```
public void start(Stage primaryStage) {
    ArrayList<TieFighter> ties = new ArrayList<>();

    Group root = new Group();

    for(int i = 0; i < 5; i++){
        ties.add(new TieFighter(ties));
    }

    root.getChildren().addAll(ties);

    Scene scene = new Scene(root, 600, 500);
    primaryStage.setTitle("Many Bouncing TIEs!");
    primaryStage.setScene(scene);
    primaryStage.show();

    KeyFrame k = new KeyFrame(Duration.millis(10), event -> {
       for(TieFighter t: ties){
           t.move();
       }
    });

    Timeline t = new Timeline(k);
    t.setCycleCount(Timeline.INDEFINITE);
    t.play();
}
```

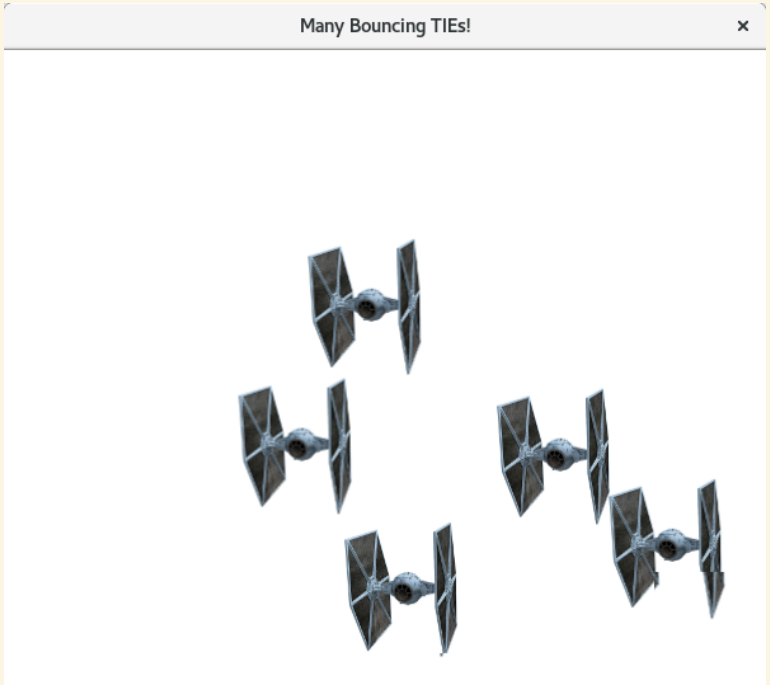
The ship class, part 1

```
public class TieFighter extends ImageView {
    public double x_speed;
    public double y_speed;
    private ArrayList<TieFighter> others;

    public TieFighter(ArrayList<TieFighter> others) {
        super();
        this.setImage(new Image("file:images/tie.png"));
        this.setPreserveRatio(true);
        this.setFitWidth(100);
        this.setFitHeight(113);
        this.others = others;

        super.setX(Math.random() * (600 - this.getFitWidth()) +1);
        super.setY(Math.random() * (500 - this.getFitHeight()) +1);
        this.x_speed = Math.random() * 5 + 1;
        this.y_speed = Math.random() * 5 + 1;
    }

    public void move() {
        super.setX(super.getX() + this.x_speed);
        super.setY(super.getY() + this.y_speed);
```

```
            if(super.getX() <= 0){
                super.setX(0);
                this.x_speed = -this.x_speed;
            }
            if(super.getX() >= 600 - this.getFitWidth()){
                super.setX(600- this.getFitWidth());
                this.x_speed = -this.x_speed;
            }
            if(super.getY() <= 0){
                super.setY(0);
                this.y_speed = -this.y_speed;
            }
            if(super.getY() >= 500 - this.getFitHeight()){
                super.setY(500 - this.getFitHeight());
                this.y_speed = -this.y_speed;
            }

            for(TieFighter t: others) {
                if(t != this && t.intersects(super.getLayoutBounds())) {
                    double tempx = this.x_speed;
                    double tempy = this.y_speed;

                    this.x_speed = t.x_speed;
                    this.y_speed = t.y_speed;

                    t.x_speed = tempx;
                    t.y_speed = tempy;
                    break;
                }
            }
        }
    }
```

Running program





Exact timing

- JavaFX has, as seen, a large number of auxiliary classes to make animation simple.
- However, to make exact timing it is necessary to subclass a timer class – called `AnimationTimer`.
  - The name is a bit confusing as it is actually a general timer class, but Java already have two of those (in `util` and `swing`) so the name became `AnimationTimer`.
- It is possible to get 60 frames per second (FPS) using `AnimationTimer` and as it is hardware accelerated, you will most likely get it.
  - It is, however, important to make the calculations in the timer as fast as possible in order not to make it a bottleneck.

The example

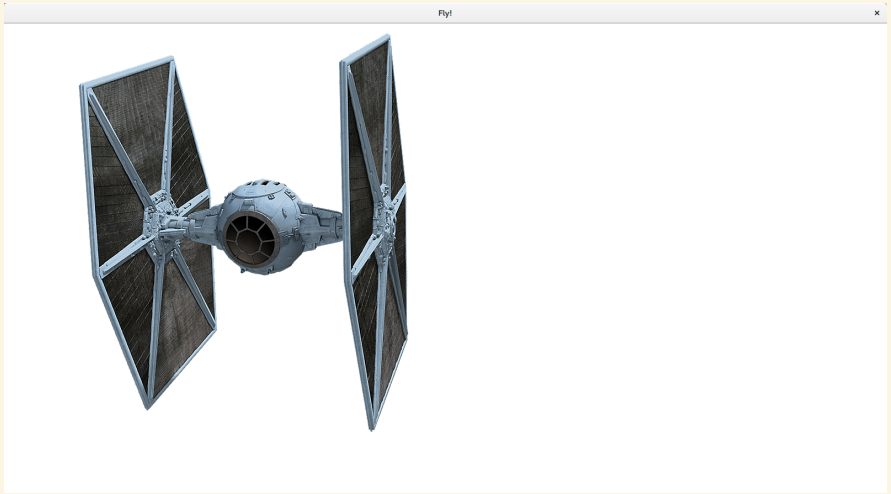
- This example uses a rather simple mechanism to add a timer, namely to create an anonymous inner class.
  - A better way, for a more robust program, it so create a separate class that derives from `AnimationTimer`
- When using the `AnimationTimer` the program needs to implement the method `handle()`.
  - This method is called as many times as possible, called a *pulse*, in the best of cases 60 times per second.
- Whatever is in the method is performed for every pulse, in this case calculating and updating a position of an image.
- The animation is started by calling the method `start()`, which comes from the superclass (`AnimationTimer`)

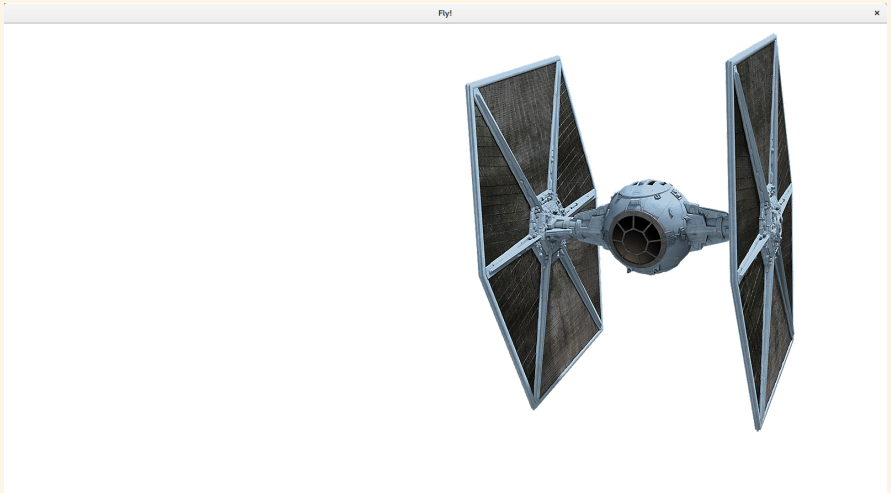
The code

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("Fly!");
    Image tie = new Image(getClass().
        getResourceAsStream("tie.png"));
    ImageView tieView = new ImageView(tie);
    tieView.setTranslateX(0);
    tieView.setTranslateY(0);

    Group layout = new Group();
    layout.getChildren().add(tieView);

    Scene scene = new Scene(layout, 1600, 850);
    primaryStage.setScene(scene);
    primaryStage.show();
```

```
    new AnimationTimer() {
        boolean right = true;
        int x = 0;
        @Override
        public void handle(long arg0) {
            if(right)
            {
                if(x <= 1000)
                    x += 10;
                else if(x >= 1000)
                    right=false;
            }
            else
            {
                if(x >= 0)
                    x -= 10;
                else if (x <= 0)
                    right = true;
            }
            tieView.setTranslateX(x);
        }
    }.start();
}
```

In graphics





Wrapping up!

- This concludes the lectures on JavaFX.
- As is probably noticable, JavaFX is really, really large!
- Even if much has been pushed into these two lectures, there is still quite a lot left.
  - Skinning using a CSS like syntax, several more controls, more on SceneBuilder and much more.
- JavaFX has really taken off these last years and much can be read on the Internet.