



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA

Lecture 03A

Maintainable GUI Development (1/2)

Introduction to GUI Programming in Java

COMP2013: Developing Maintainable Software
Week 4 – 4:00pm Monday – 16 October 2023



valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

Horia A. Maior and Marjahan Begum



valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14



Lecture 03A

Maintainable GUI Development (1/2)

Introduction to GUI Programming in Java

Horia A. Maior and Marjahan Begum

What's coming up ...



valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

Coursework 1 released

Coursework 1 Part 1 Deadline

Coursework 1 Part 2 Deadline

Teaching Week	Topic
1	Introduction to DMS
2	More Advanced Java Topic
3	Maintainable GUI Development (1/2)
4	Design Principles and Patterns
5	Maintainable GUI Development (2/2)
6	Coding and Repository Tools for DMS
7	UML for the Maintainer
8	Refactoring Skills.
9	Open Source
10	Guest Lecture
11	Revision and Exam Prep

Topics for this Week



valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

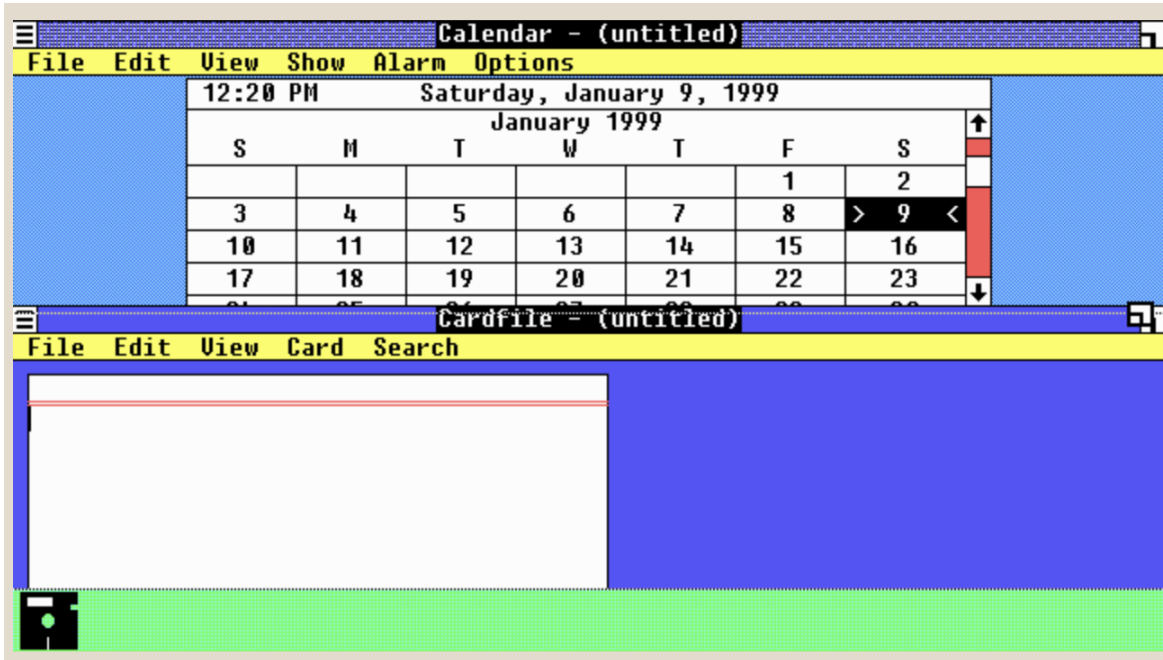
- Lecture 03A
 - Introduction to GUI programming in Java
 - GUIs; Swing vs JavaFX; JavaFX foundations
- Lab 03
 - Practice JavaFX GUI development (phone app)
- Lecture 03B
 - GUI development in practice
 - Lab reflection; MVC pattern revisited; JavaFX and FXML



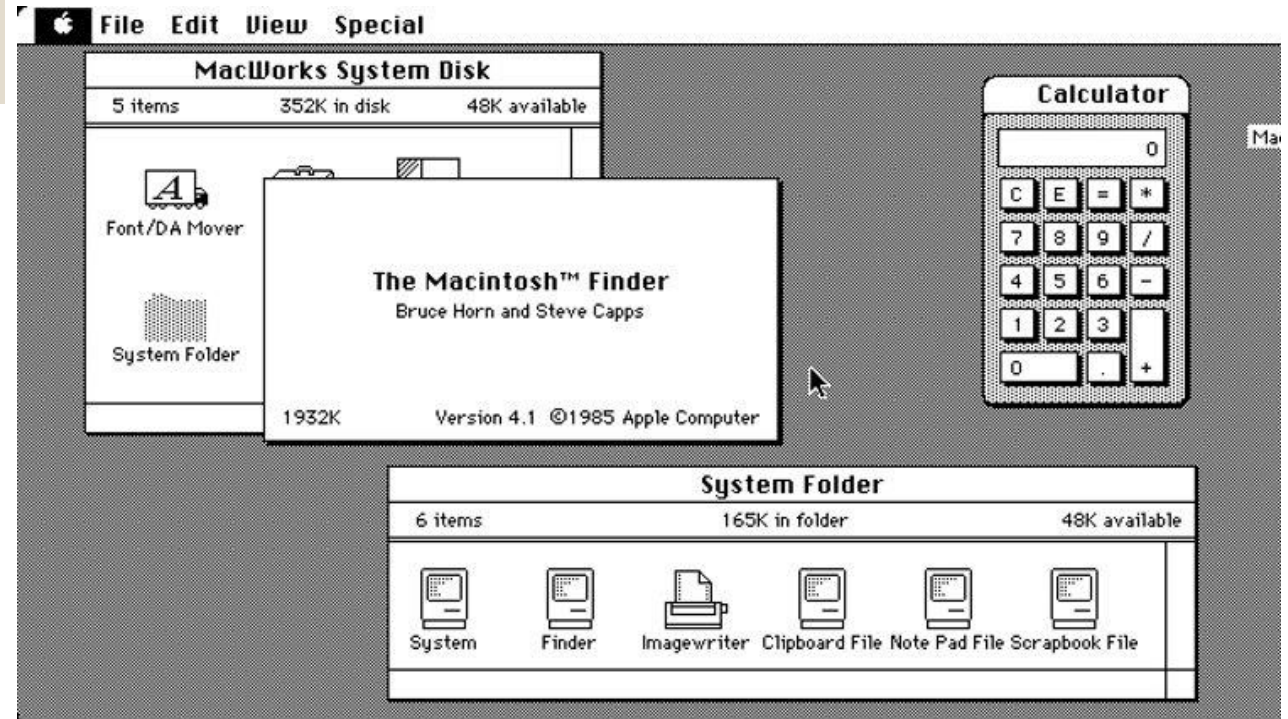
valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

Graphical User Interfaces (GUIs)





<http://www.catb.org/~esr/writings/taouu/html/ch02s05.html>



What are GUIs?

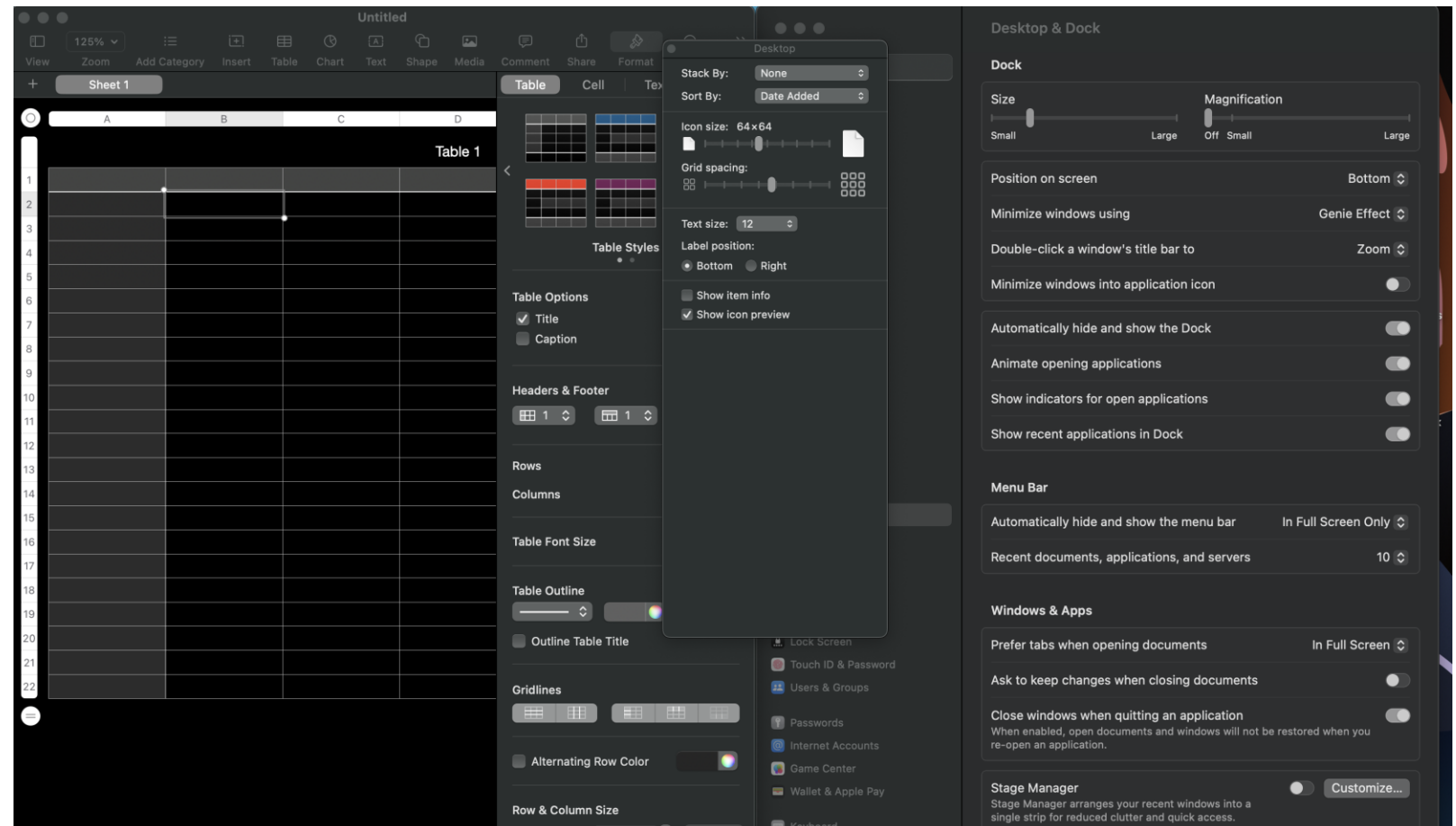


- What are Graphical User Interfaces (GUIs)?
 - GUIs are a form of user interface that allow users to interact with electronic devices through graphical icons [...] as primary notation, instead of text-based user interfaces, typed command labels or text navigation. [Wikipedia]
 - GUIs were introduced in reaction to the perceived steep learning curve of Command Line Interfaces (CLIs) which require commands to be typed on a computer keyboard. [Wikipedia]
- Why do I need to learn about them in this module?
 - Need to understand the link between a GUI and the code behind it in order to maintain it
 - Want to write new GUIs from scratch that are easy to maintain

What are GUIs



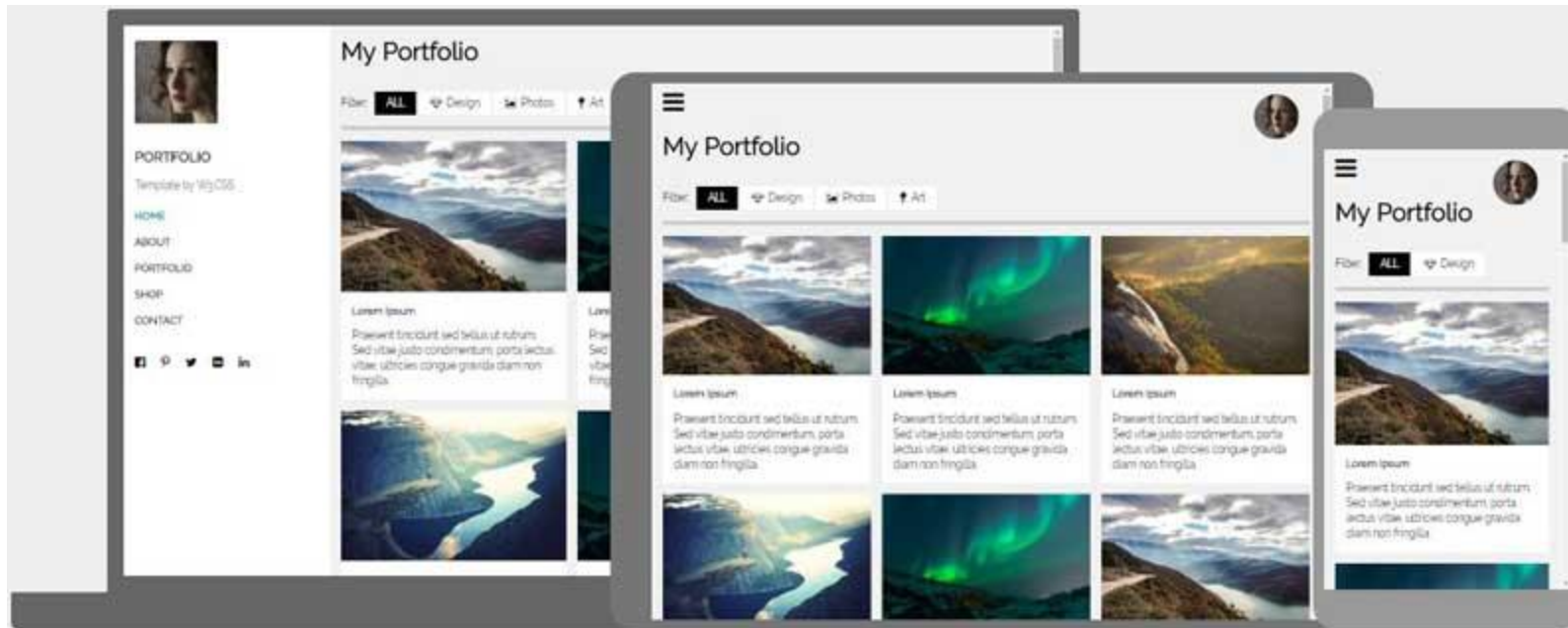
- GUIs can often be set up in different ways depending on personal needs and likes



What are GUIs?



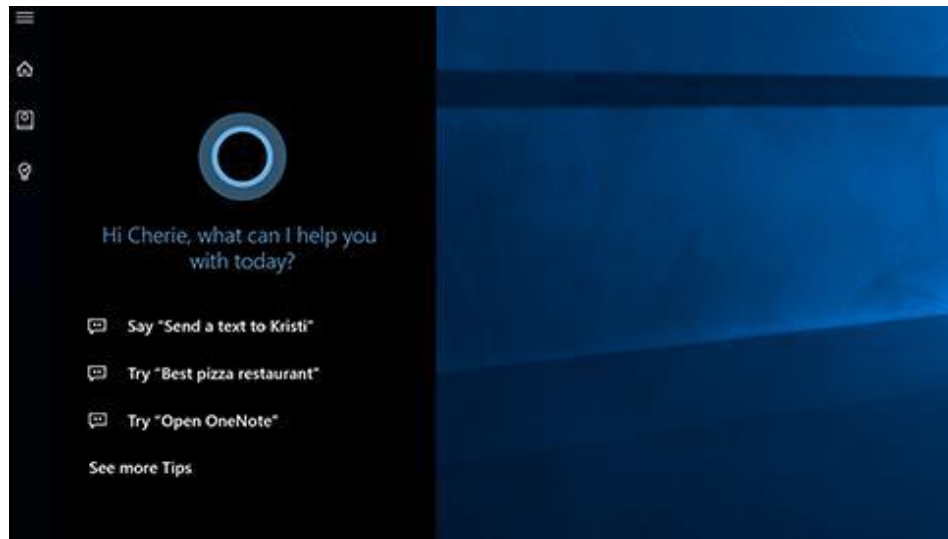
- Different devices require different GUI design styles: We need responsive GUIs



What are GUIs?



- Does the GUI have a future?




Standards and Guidance



ISO 9241 defines usability as effectiveness, efficiency, and satisfaction with which users accomplish tasks

<https://www.iso.org/obp/ui/#iso:std:iso:9241:-161:en>



Online Browsing Platform (OBP)

SearchISO 9241-161:2016(en) ×

ISO 9241-161:2016(en) Ergonomics of human-system interaction — Part 161: Guidance on visual user-interface elements

Table of contents

Foreword

Introduction

1 Scope

2 Normative references

3 Terms and definitions

4 Accessibility

5 Relationship of input methods and visu

6 States of visual user-interface element

7 Describing visual user-interface elem

8 Visual user-interface elements

8.1 Accordion

8.2 Analogue form element/slider

8.3 Carousel/Carrousel

8.4 Check box/check button

8.5 Collapsible container

8.6 Colour picker

8.7 Combination box/combo box

8.8 Cursor

8.9 Date picker

8.10 Dialogue box

8.11 Dropdown list box

8.12 Entry field/input field

8.13 Entry field with dialogue button

8.14 Geographical map

8.15 Group/group box

8.16 Handle

8.17 Hierarchical list/tree view/tree lis

8.18 Implicit designator

8.19 Instructive information

8.20 Input tokenizer

8.21 Label

8.22 Legend/chart key

8.23 List/grouped list

Available in: EN FR

Introduction


In different communities in the interactive system development ecosystem, the use, the names and the understanding of user-interface elements differs significantly. One of the results is that users have to cope with elements which differ in terms of keyboard entry and control, mouse behaviour, visual presentation of functionality and different options to control elements. **Consistent element behaviour, functionality and rendering is crucial for the usability of user interfaces.** This causes added efforts in all stakeholders in human-centred design activities, since this multitude needs to be managed in order to ensure high-quality collaboration of various specialists. Especially in the light of new emerging user-interface concepts and designs, a common definition of visual user-interface elements and the rationale for their selection, as well as their use can be regarded as an effort to sustain cooperation and ensure a sound basis for professional conversation. It is also of importance to state that this part of ISO 9241 of visual user-interface elements in no ways predetermines a visual style of the elements themselves, thus avoiding to impress determinants in creation, brand usage and style development. In addition, this part of ISO 9241 is laid out in an independent of platform specifics, so that no specific industrial user-interface styleguide, implementation technology or development process needs to be observed in order to be compliant with this part of ISO 9241.

This part of ISO 9241 aims to provide information on visual user-interface elements to help those responsible for managing software design and re-design processes, create user interface specifications, styleguides and visual concepts to identify, plan and design effective, efficient and satisfactory interactive systems.

Visual user-interface elements described in this part of ISO 9241 complements existing systems design approaches, methods or processes. They can be referenced in any kind of user interface strategy, regardless of the technology used for the user interface.

Table 1 — Overview of different visual user interface properties that are used to build a user interface design

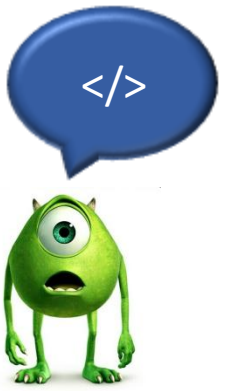
User Interface Design				
Interactive Properties		Informative Properties		Decorative Properties



University of
Nottingham
UK | CHINA | MALAYSIA

COMP2013

13



- Guiding principles for designing high quality GUIs?
 1. Keep it simple
 2. Create consistency and use common elements
 3. Be purposeful in page layout
 4. Strategically use colour and texture
 5. Use typography to create hierarchy and clarity
 6. Make sure that the system communicates what's happening
 7. Think about the defaults

<https://www.usability.gov/what-and-why/user-interface-design.html>

Heuristics and Guidelines



- <https://www.nngroup.com/articles/ten-usability-heuristics/>

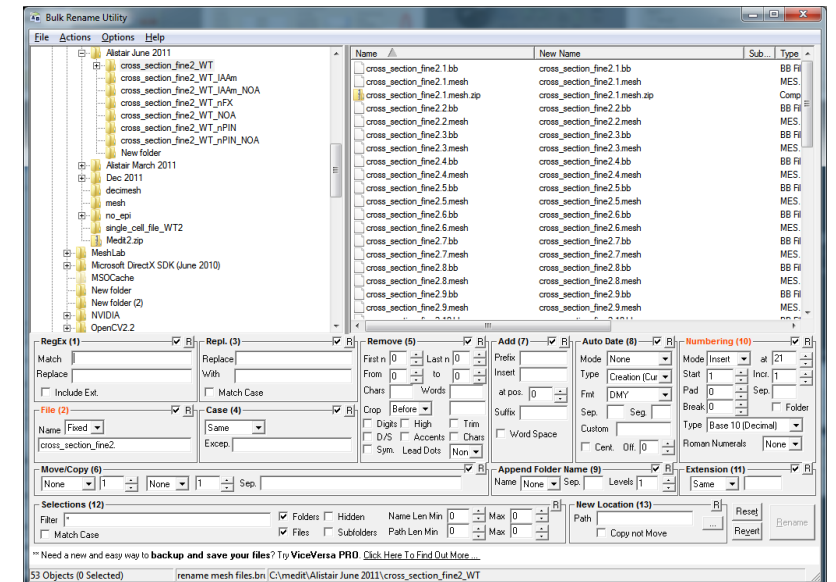
Nielsens (1993), 10 usability heuristics

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help recognize, diagnose, and recover from errors
10. Help and documentation

Examples of Bad UI Design



- Appearance over functionality
- What does the user need to do?
- Accessibility of controls
- Form field overload





basics of GUI programming

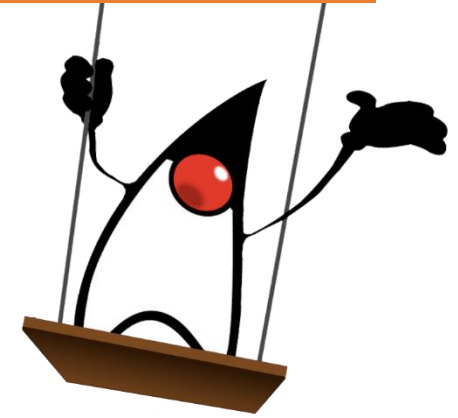
Swing and JavaFX

What do I need to know before getting started?



- Under the hood GUIs are event-driven
 - The user decides the order of execution depending whether they click a button, or select a drop-down, or choose a menu item etc.
 - The system is waiting for something to happen
- GUIs are implemented by using frameworks or libraries
 - Java examples: Swing + JavaFX
- GUIs can be programmatically designed or drawn in a graphical editor

Swing



- Java's first attempt at a 'modern' GUI approach
 - Came about in the late 1990's...
 - Superseded/built on the earlier AWT (Abstract Window Toolkit)
 - Swing still relies on AWT for some circumstances
 - You can spot a Swing component as it starts with a J, e.g. JFrame
 - Many existing Java GUIs use Swing, and it still has a strong following
 - But Swing is being slowly retired in favour of JavaFX

Java Swing Example

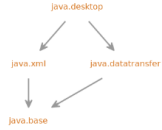


- Hello World GUI in Java and Swing
- Video Recording Demo Available:
 - [Demo Hello World Java Swing - HD 1080p.mov](#)

Module java.desktop

Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.

Module Graph:



<https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/module-summary.html>



DEMO

```
module-info.java (HelloWorldSwingGUI) ×  
1  module HelloWorldSwingGUI {  
2      opens com.COMP2013;  
3      requires java.desktop;  
4  }
```



DEMO

```
1 package com.COMP2013;
2 > import ...
6 ▶ public class HelloWorldSwingGUIApp extends JFrame {
    1 usage
7     public HelloWorldSwingGUIApp(){
8         initUI();
9     }
    1 usage
10    private void initUI(){
11        setSize( width: 600, height: 600);
12        setTitle("My first GUI in Java Swing. Hello world!");
13        setLocationRelativeTo(null);
14        setDefaultCloseOperation(EXIT_ON_CLOSE);
15        JPanel pannel = new JPanel(new GridBagLayout());
16        JButton button1 = new JButton( text: "Exit program");
17        button1.addActionListener(new ActionListener() {
18            @Override
19            public void actionPerformed(ActionEvent e) {
20                System.out.println("Now we will exit the program");
21                System.exit( status: 0);
22            }
23        });
24        pannel.add(button1);
25        this.add(pannel);
26    }
```

```
module-info.java (HelloWorldSwingGUI) x
1 module HelloWorldSwingGUI {
2     opens com.COMP2013;
3     requires java.desktop;
4 }
```




DEMO

```
1 package com.COMP2013;
2 > import ...
6 public class HelloWorldSwingGUIApp extends JFrame {
    1 usage
7     public HelloWorldSwingGUIApp(){
8         initUI();
9     }
    1 usage
10    private void initUI(){
11        setSize( width: 600, height: 600);
12        setTitle("My first GUI in Java Swing. Hello world!");
13        setLocationRelativeTo(null);
14        setDefaultCloseOperation(EXIT_ON_CLOSE);
15        JPanel pannel = new JPanel(new GridBagLayout());
16        JButton button1 = new JButton( text: "Exit program");
17        button1.addActionListener(new ActionListener() {
18            @Override
19            public void actionPerformed(ActionEvent e) {
20                System.out.println("Now we will exit the program");
21                System.exit( status: 0);
22            }
23        });
24        pannel.add(button1);
25        this.add(pannel);
26    }
```

```
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
```

```

1 package com.COMP2013;
2 > import ...
6 public class HelloWorldSwingGUIApp extends JFrame {
    1 usage
7     public HelloWorldSwingGUIApp(){
8         initUI();
9     }
    1 usage
10    private void initUI(){
11        setSize( width: 600, height: 600);
12        setTitle("My first GUI in Java Swing. Hello world!");
13        setLocationRelativeTo(null);
14        setDefaultCloseOperation(EXIT_ON_CLOSE);
15        JPanel pannel = new JPanel(new GridBagLayout());
16        JButton button1 = new JButton( text: "Exit");
17        button1.addActionListener(new ActionListener() {
18            @Override
19            public void actionPerformed(ActionEvent e) {
20                System.out.println("Now we will exit");
21                System.exit( status: 0);
22            }
23        });
24        pannel.add(button1);
25        this.add(pannel);
26    }

```

module-info.java (HelloWorldSwingGUI) ×

```

1 module HelloWorldSwingGUI {
2     opens com.COMP2013;
3     requires java.desktop;
4 }

```

```

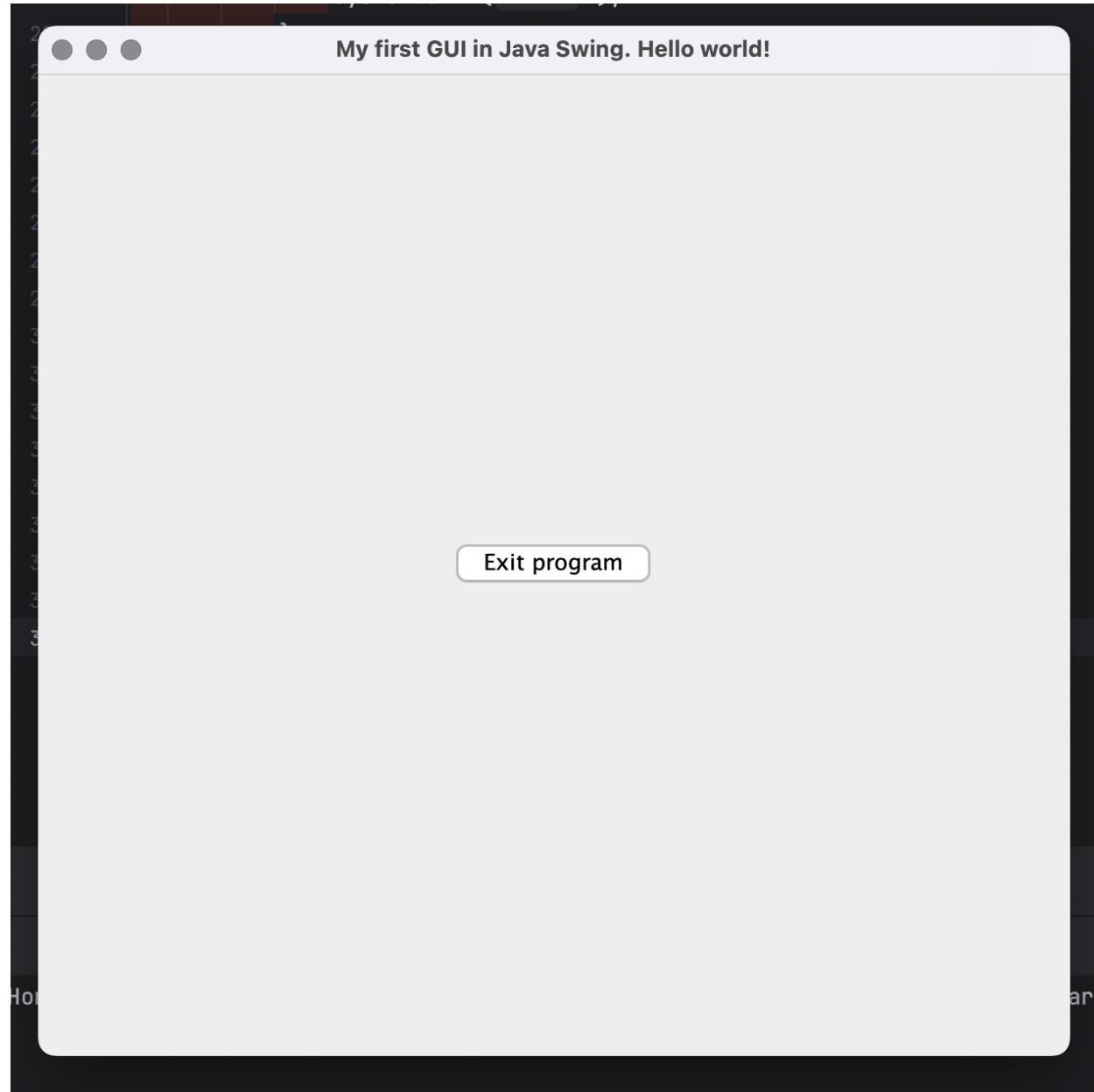
27
28
29 public static void main(String[] args) {
30     //main function
31     EventQueue.invokeLater(new Runnable() {
32         @Override
33         public void run() {
34             new HelloWorldSwingGUIApp().setVisible(true);
35         }
36     });
37 }
38

```



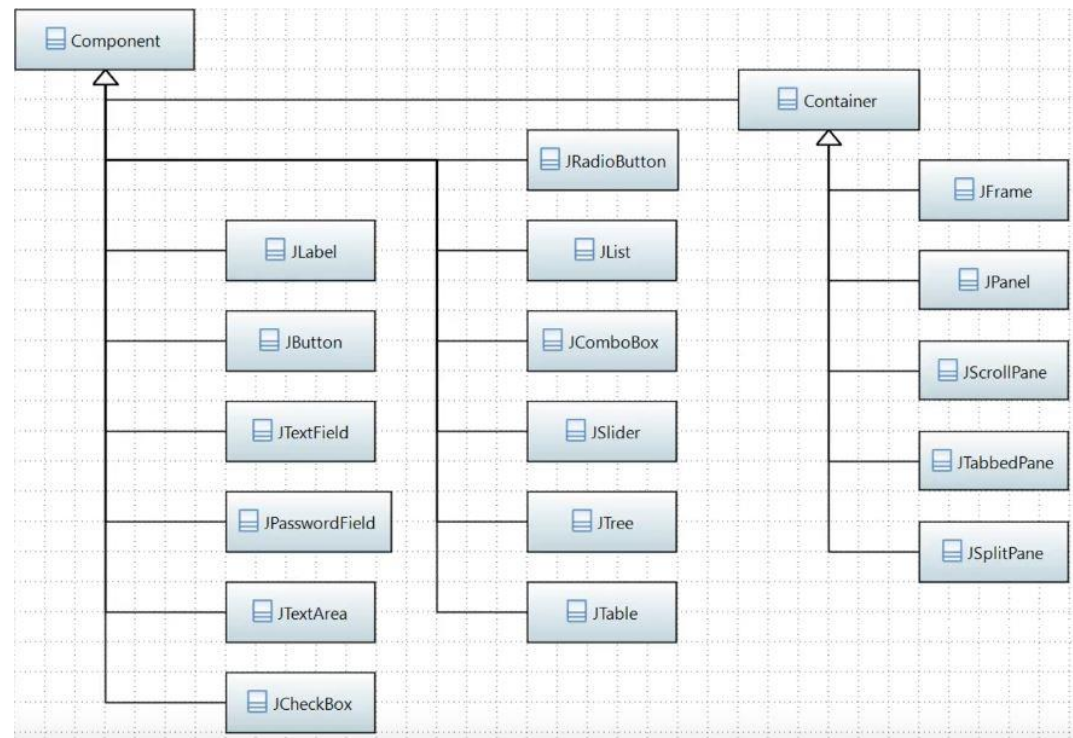


DEMO





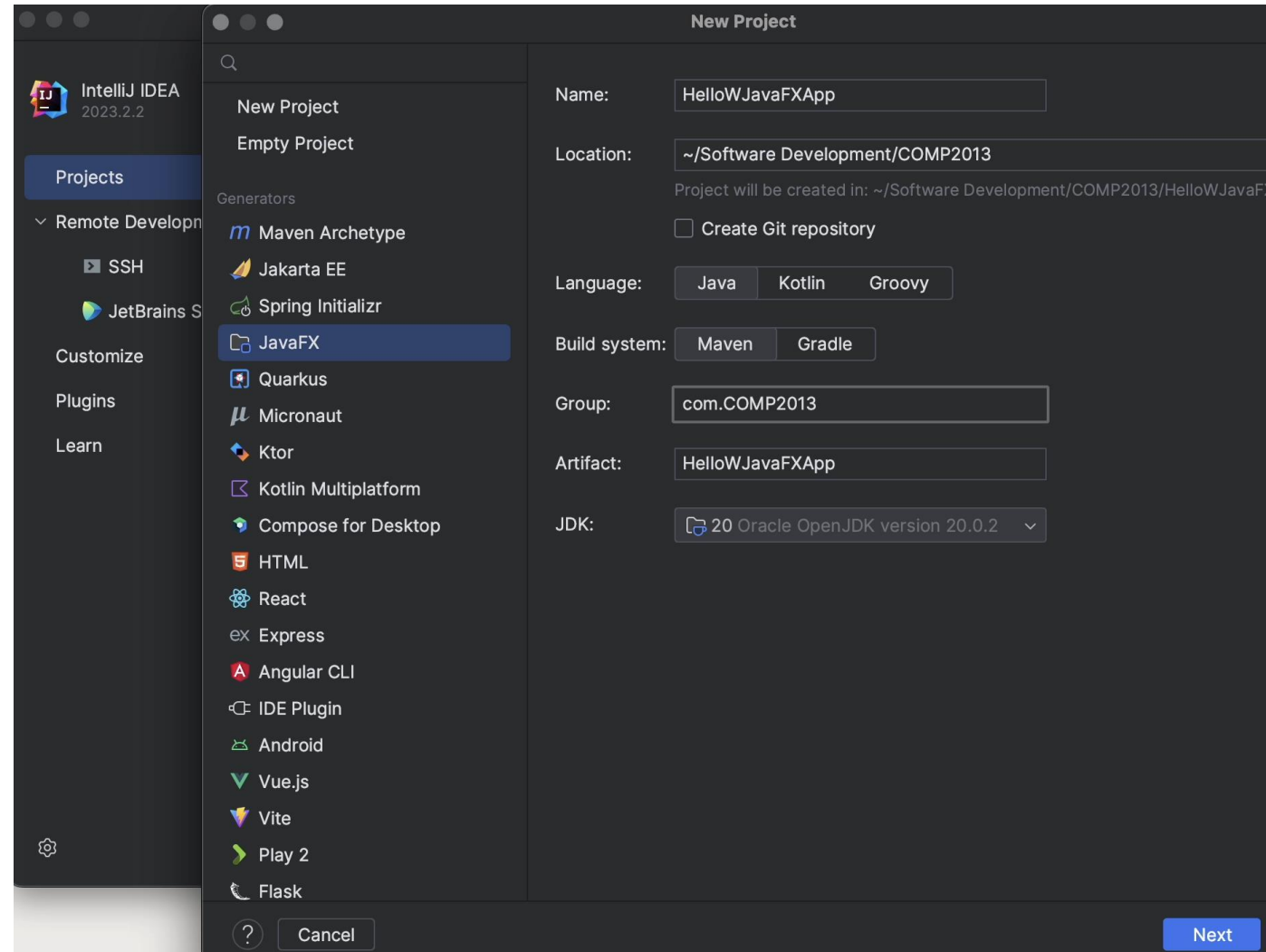
- For those of you not (so) familiar with Swing, here are some excellent tutorials:
 - <https://www.youtube.com/playlist?list=PLTMybUaeagJagT2qoftaf5CkCvgc3pBTn>





- More recently introduced as part of JDK/JRE (until Java 10) but then removed from the JDK/JRE (from version 11 onwards) and available as external library
 - Can deploy GUIs to tablet, phone, desktop etc.
 - Able to separate GUI code from program code using an XML description file (FXML)
 - Handling graphics by using a hardware-accelerated graphics pipeline to do the rendering job
 - Supports idea of properties (variables that represent the state of an instantiated object)
 - Formatting uses Cascading Style Sheets (CSS)
 - Special effects/animations supported
 - Comes with media player







DEMO

```
1  module com.comp2013 {  
2      opens com.comp2013;  
3      requires javafx.controls;  
4      requires javafx.graphics;  
5  }
```



DEMO

```
1 package com.comp2013;
2
3 > import ...
4
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     Ⓢ↑@ public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button(s: "Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             Ⓢ↑ public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, v: 600, v1: 600));
28         stage.show();
29     }
30 ▶ public static void main(String[] args) {
31     launch();
32 }
33 }
```

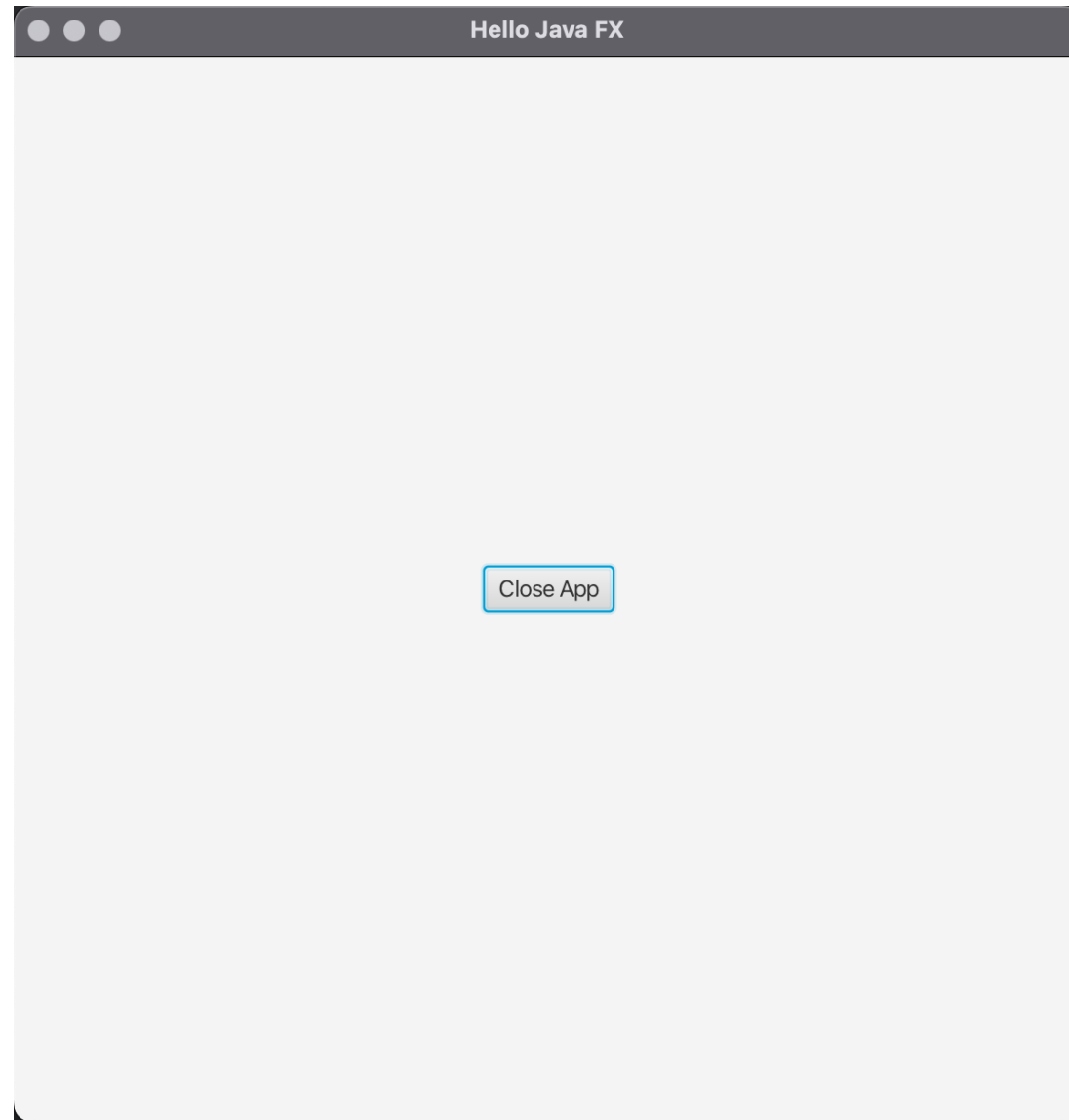


DEMO

```
1 package com.comp2013;
2
3 > import ...
12
13 public class HelloApplication extends Application {
14     @Override
15     public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button("Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, w: 600, h: 600));
28         stage.show();
29     }
30     public static void main(String[] args) {
31         launch();
32     }
33 }
```

```
3 import javafx.application.Application;
4 import javafx.event.ActionEvent;
5 import javafx.event.EventHandler;
6 import javafx.scene.Scene;
7 import javafx.scene.control.Button;
8 import javafx.scene.layout.StackPane;
9 import javafx.stage.Stage;
```





DEMO

Adding Framework Support ...



DEMO

The screenshot shows an IDE window with the following components:

- Project Explorer (Left):** Displays the project structure for 'HelloJavaFXWorld'. It includes folders for 'src' (main, test), 'resources', and 'test'. The 'test' folder is highlighted, showing 'HelloJavaFXWorldAppTest' and 'resources'.
- Editor (Center):** Displays the 'pom.xml' file. The XML content is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.siebers</groupId>
  <artifactId>hellojavafxworld</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <junit.version>5.8.2</junit.version>
    <javafx.version>17.0.2</javafx.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.10.1</version>
        <configuration>
          <source>${maven.compiler.source}</source>
          <target>${maven.compiler.target}</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.0.0-M6</version>
      </plugin>
      <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.8</version>
        <executions>
          <execution>
            <!-- Default configuration for running with: mvn clean javafx:run -->

```
- Maven (Right):** Displays the Maven lifecycle and plugins. The 'Lifecycle' section shows 'clean', 'validate', 'compile', 'test', 'package', 'verify', 'install', 'site', and 'deploy'. The 'Plugins' section lists various plugins, including 'maven-clean-plugin', 'maven-compiler-plugin', 'maven-deploy-plugin', 'maven-install-plugin', 'maven-jar-plugin', 'javafx-maven-plugin', 'javafx-jlink', 'javafx-run', 'maven-resources-plugin', 'maven-site-plugin', and 'maven-surefire-plugin'.



getting started with JavaFX



```
1 package com.comp2013;
2
3 > import ...
12
13 public class HelloApplication extends Application {
14     @Override
15     public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button(s: "Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, v: 600, v1: 600));
28         stage.show();
29     }
30     public static void main(String[] args) {
31         launch();
32     }
33 }
```



launch readies the application and then invokes **start**

Execution moves then to the JavaFX thread

JavaFX Theatre analogy



The whole play contains several *Scenes*



All the action *Scenes* takes place on a *Stage*

JavaFX's Stage and Scene



- Stage:
 - Think of it as an application window
 - Depending on OS, there may be only one
 - Equivalent to Swing's JFrame (or JDialog)
- Scene:
 - Equivalent to a content pane
 - Holds other objects (JavaFX Node objects)



```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @I↑@ public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button(s: "Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             @I↑ public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, v: 600, v1: 600));
28         stage.show();
29     }
30 ▶ public static void main(String[] args) {
31     launch();
32 }
33 }
```

So this is our window,
and we can set its text
in the title bar etc.



```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @i↑@ public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button(s: "Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             @i↑ public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, v: 600, v1: 600));
28         stage.show();
29     }
30 ▶ public static void main(String[] args) {
31     launch();
32 }
33 }
```

These are examples of
Node objects



```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @i↑@ public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button(s: "Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             @i↑ public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, v: 600, v1: 600));
28         stage.show();
29     }
30 ▶ public static void main(String[] args) {
31     launch();
32 }
33 }
```

Node objects can contain other **Node** objects



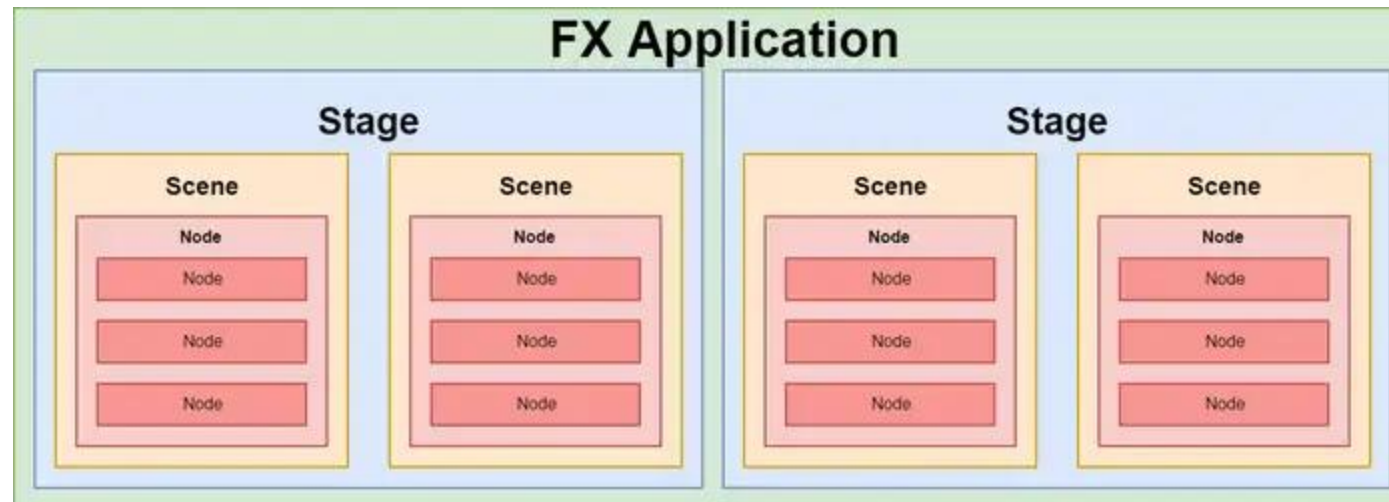
```
1 package com.comp2013;
2
3 > import ...
4
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @i↑@ public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button(s: "Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             @i↑ public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(status: 0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, v: 600, v1: 600));
28         stage.show();
29     }
30 ▶ public static void main(String[] args) {
31     launch();
32 }
33 }
```

Finally, we put it all together,
and make it visible

Summary: Stage / Scene / Node



- Summary
 - If the OS allows each application can have multiple stages - windows. Each stage can switch between multiple scenes. Scenes contain nodes - layout and regular components.



From: <https://www.vojtechruzicka.com/javafx-hello-world/>

Properties & Binding



- JavaFX properties (observable containers) are often used in conjunction with binding, a powerful mechanism for expressing direct relationships between variables
- When objects participate in bindings, changes made to one object will automatically be reflected in another object
- You can also add a change listener to be notified when the property's value has changed



DEMO

```
1 package com.COMP2013;
2 import ...
14 public class HelloApplication extends Application {
15     @Override
16     public void start(Stage stage) throws IOException {
17         stage.setTitle("Welcome to the PropertyBinding Demo!");
18         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
19         Text text = new Text(String.valueOf((int)slider.getValue()));
20         text.setFont(Font.font(s: "Verdana", v: 50));
21         StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
22         slider.valueProperty().addListener(new ChangeListener<Number>() {
23             @Override
24             public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
25                 text.setText(String.valueOf((int)slider.getValue()));
26             }
27         });
28         StackPane root = new StackPane();
29         root.getChildren().addAll(slider, text);
30         stage.setScene(new Scene(root, v: 500, v1: 500));
31         stage.show();
32     }
33
34     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```



DEMO

```
1 package com.COMP2013;
2 import ...
14 public class HelloApplication extends Application {
15     @Override
16     public void start(Stage stage) throws IOException {
17         stage.setTitle("Welcome to the PropertyBinding Demo!");
18         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
19         Text text = new Text(String.valueOf((int)slider.getValue()));
20         text.setFont(Font.font(s: "Verdana", v: 50));
21         StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
22         slider.valueProperty().addListener(new ChangeListener<Number>() {
23             @Override
24             public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
25                 text.setText(String.valueOf((int)slider.getValue()));
26             }
27         });
28         StackPane root = new StackPane();
29         root.getChildren().addAll(slider, text);
30         stage.setScene(new Scene(root, v: 500, v1: 500));
31         stage.show();
32     }
33
34     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```



DEMO

```
1 package com.COMP2013;
2 import ...
14 public class HelloApplication extends Application {
15     @Override
16     public void start(Stage stage) throws IOException {
17         stage.setTitle("Welcome to the PropertyBinding Demo!");
18         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
19         Text text = new Text(String.valueOf((int)slider.getValue()));
20         text.setFont(Font.font(s: "Verdana", v: 50));
21         StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
22         slider.valueProperty().addListener(new ChangeListener<Number>() {
23             @Override
24             public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
25                 text.setText(String.valueOf((int)slider.getValue()));
26             }
27         });
28         StackPane root = new StackPane();
29         root.getChildren().addAll(slider, text);
30         stage.setScene(new Scene(root, v: 500, v1: 500));
31         stage.show();
32     }
33
34     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```

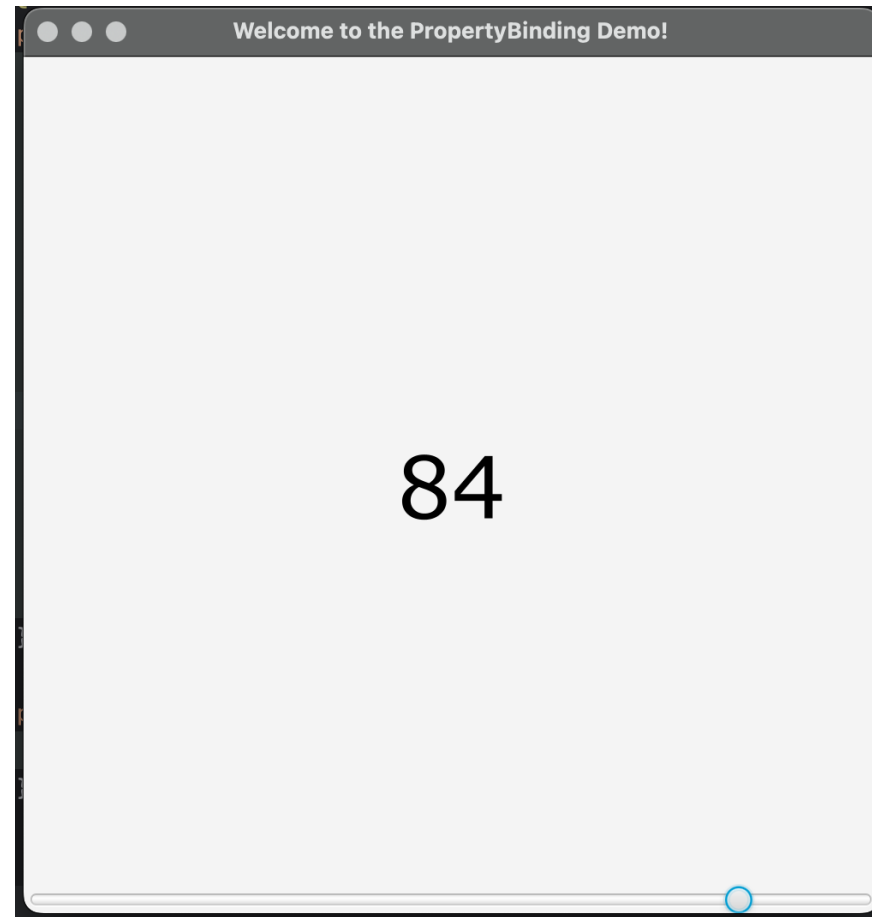


DEMO

```
1 package com.COMP2013;
2 import ...
14 public class HelloApplication extends Application {
15     @Override
16     public void start(Stage stage) throws IOException {
17         stage.setTitle("Welcome to the PropertyBinding Demo!");
18         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
19         Text text = new Text(String.valueOf((int)slider.getValue()));
20         text.setFont(Font.font(s: "Verdana", v: 50));
21         StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
22         slider.valueProperty().addListener(new ChangeListener<Number>() {
23             @Override
24             public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
25                 text.setText(String.valueOf((int)slider.getValue()));
26             }
27         });
28         StackPane root = new StackPane();
29         root.getChildren().addAll(slider, text);
30         stage.setScene(new Scene(root, v: 500, v1: 500));
31         stage.show();
32     }
33
34     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```



DEMO



The Node Class



More information:

https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1_intro.html

<https://stackoverflow.com/questions/54138287/getting-java-lang-reflect-invocationTargetException-while-adding-a-button-to-lay>

- Fundamental to JavaFX
- Used to represent controls, layouts, shapes, etc.
- Can apply effects (transform, translate etc.) to nodes

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH:

Module javafx.graphics

Package javafx.scene

Class Node

java.lang.Object[Ⓔ]
 javafx.scene.Node

All Implemented Interfaces:
Styleable, EventTarget

Direct Known Subclasses:
Camera, Canvas, ImageView, LightBase, MediaView, Parent, Shape, Shape3D, SubScene, SwingNode

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRE

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD |

Module javafx.controls

Package javafx.scene.control

Class Button

java.lang.Object[Ⓔ]
 javafx.scene.Node
 javafx.scene.Parent
 javafx.scene.layout.Region
 javafx.scene.control.Control
 javafx.scene.control.Labeled
 javafx.scene.control.ButtonBase
 javafx.scene.control.Button

All Implemented Interfaces:
Styleable, EventTarget, Skinnable

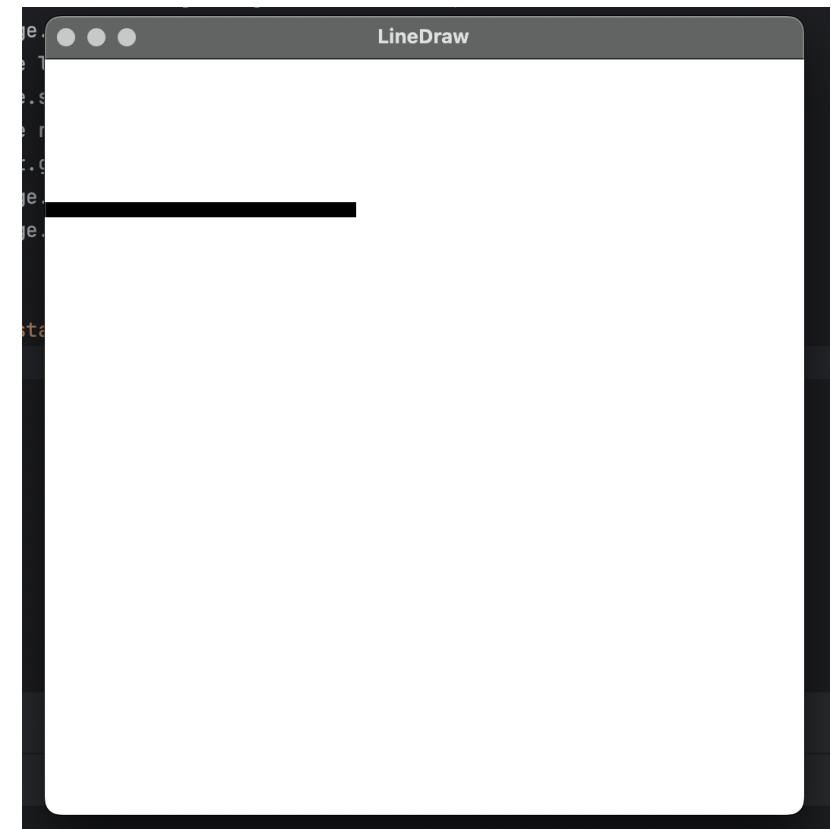
Screen Coordinates



Adding simple 2D Graphics



```
1 package com.comp2013;
2 import ...
9 public class LineApp extends Application {
10     @Override
11     public void start(Stage stage) throws IOException {
12         stage.setTitle("LineDraw");
13         Line line = new Line(v: 0, v1: 100, v2: 200, v3: 100);
14         line.setStrokeWidth(10);
15         Pane root = new Pane();
16         root.getChildren().add(line);
17         stage.setScene(new Scene(root, v: 500, v1: 500));
18         stage.show();
19     }
20
21     public static void main(String[] args) { launch(args); }
24 }
```

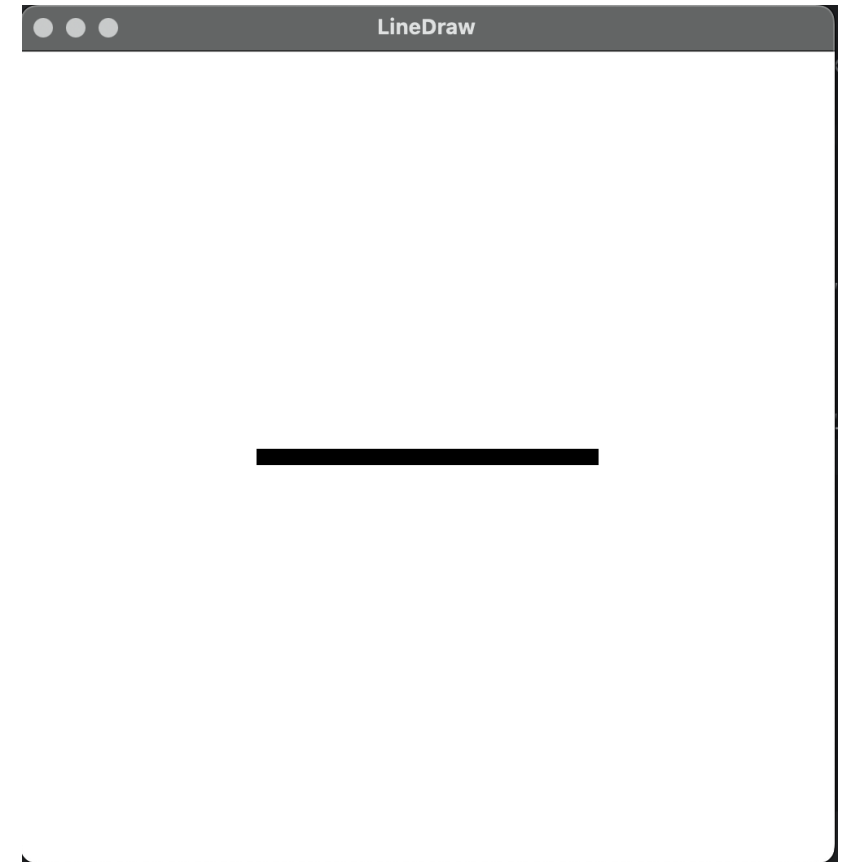


Adding simple 2D Graphics



© LineApp.java ×

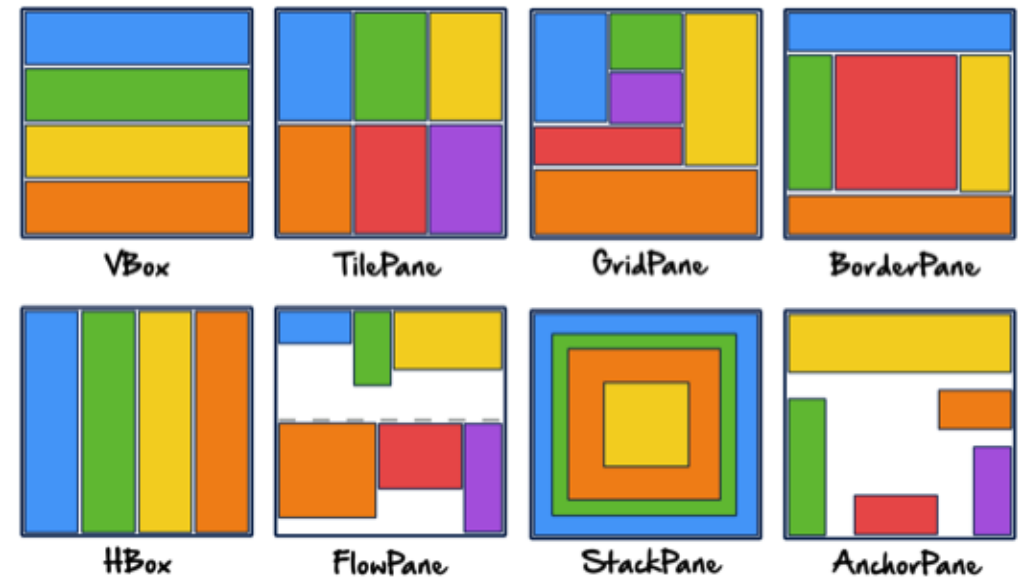
```
1 package com.comp2013;
2 > import ...
10 ▶ public class LineApp extends Application {
11     @Override
12     @
13     public void start(Stage stage) throws IOException {
14         stage.setTitle("LineDraw");
15         Line line = new Line(v: 0, v1: 100, v2: 200, v3: 100);
16         line.setStrokeWidth(10);
17         Pane root = new StackPane();
18         root.getChildren().add(line);
19         stage.setScene(new Scene(root, v: 500, v1: 500));
20         stage.show();
21     }
22 ▶ > public static void main(String[] args) { launch(args); }
25
```

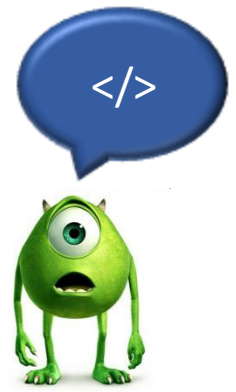
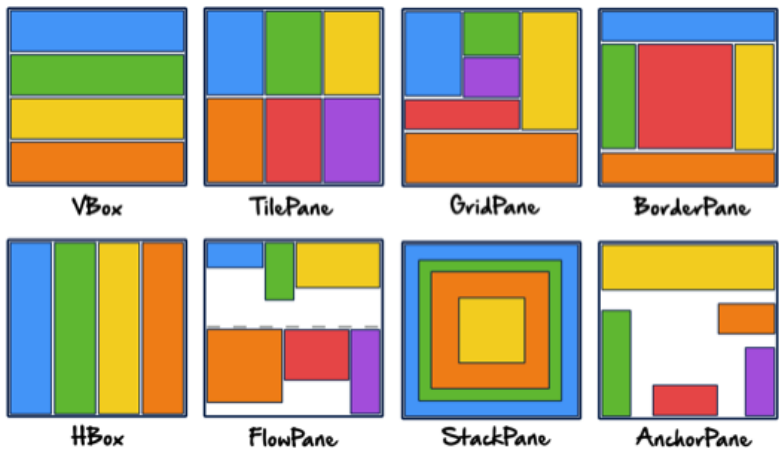


Built-in Layout Panes



- **Vbox:** Provides an easy way for arranging a series of nodes in a single row
- **TilePane:** Places all of the nodes in a grid in which each cell, or tile, is the same size. Nodes can be laid out horizontally (in rows) or vertically (in columns)
- **GridPane:** Allows to create a flexible grid of rows and columns in which to lay out nodes; nodes can be placed in any cell in the grid and can span cells as needed
- **BorderPane:** Provides five regions in which to place nodes: top, bottom, left, right, and centre
- **HBox:** Provides an easy way for arranging a series of nodes in a single column
- **FlowPane:** Nodes are laid out consecutively and wrap at the boundary set for the pane; nodes can flow vertically (in columns) or horizontally (in rows)
- **StackPane:** Places all of the nodes within a single stack with each new node added on top of the previous
- **AnchorPane:** Allows to anchor nodes to the top, bottom, left side, right side, or centre of the pane; as the window is resized, the nodes maintain their position relative to their anchor point.





The screenshot shows an IDE with a JavaFXPaneDemo application window on the left and its source code on the right. The application window displays a simple user interface with buttons labeled 'one', 'two', 'three', 'four', and 'five five five'. The code on the right defines the layout and logic for this application.

```

FlowPane root=new FlowPane();
HBox hbox=new HBox(btn1,btn2);
VBox vbox=new VBox();
vbox.getChildren().add(btn3);
vbox.getChildren().add(btn4);
root.getChildren().addAll(hbox,vbox,btn5);
//root.getChildren().addAll(btn1,btn2,btn3,btn4,btn5);
stage.setScene(new Scene(root, v: 500, v1: 500));
stage.show();
}

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

```



Linking JavaFX and Swing

Introduction


```

1  package com.comp2013;
2
3  > import ...
12
13  ▶ public class HelloApplication extends Application {
14      @Override
15      @
16      public void start(Stage stage) throws IOException {
17          stage.setTitle("Hello Java FX");
18          Button button1 = new Button(s: "Close App");
19          button1.setOnAction(new EventHandler<ActionEvent>() {
20              @Override
21              public void handle(ActionEvent actionEvent) {
22                  System.out.println("Bye Bye!");
23                  System.exit(status: 0);
24              }
25          });
26          StackPane root = new StackPane();
27          root.getChildren().add(button1);
28          stage.setScene(new Scene(root, v: 600, v1: 600));
29          stage.show();
30      }
31      ▶ public static void main(String[] args) {
32          launch();
33      }

```

```

1  package com.COMP2013;
2  > import ...
6  ▶ public class HelloWorldSwingGUIApp extends JFrame {
7      1 usage
8      public HelloWorldSwingGUIApp(){
9          initUI();
10     }
11     1 usage
12     private void initUI(){
13         setSize(width: 600, height: 600);
14         setTitle("My first GUI in Java Swing. Hello world!");
15         setLocationRelativeTo(null);
16         setDefaultCloseOperation(EXIT_ON_CLOSE);
17         JPanel pannel = new JPanel(new GridBagLayout());
18         JButton button1 = new JButton(text: "Exit program");
19         button1.addActionListener(new ActionListener() {
20             @Override
21             public void actionPerformed(ActionEvent e) {
22                 System.out.println("Now we will exit the program");
23                 System.exit(status: 0);
24             }
25         });
26         pannel.add(button1);
27         this.add(pannel);
28     }

```

Linking Swing to JavaFX





some final remarks ...