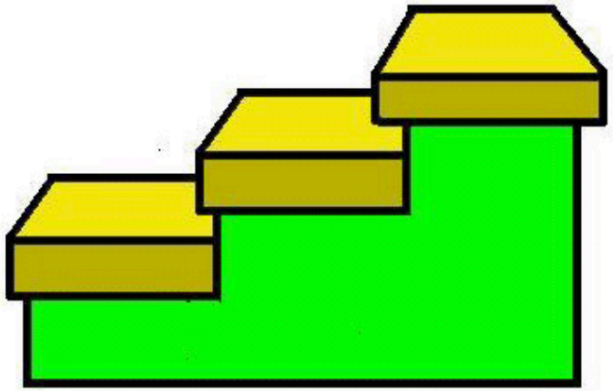
Week 5 Presentation 3

Graphical user Interfaces II
First Application

Making a GUI in Java

The previous presentation on GUIs (“gooey”) summarized the way Event Driven Programming is applied to Graphical User Interface Development

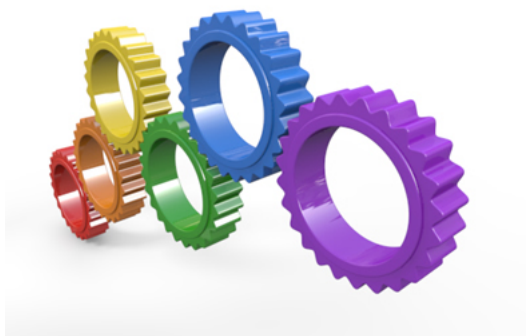
This presentation starts to describe how to implement all this in practice with JavaFX



Swing and AWT are replaced by the JavaFX platform for developing rich GUI applications.

The Basic Structure of a JavaFX Program

The `javafx.application.Application` *class defines the essential framework for writing JavaFX programs.*



A simple JavaFX displays a button in the window.

MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {                    extend Application
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {                      override start
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");                            create a button
11     Scene scene = new Scene(btOK, 200, 250);                   create a scene
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title  set stage title
13     primaryStage.setScene(scene); // Place the scene in the stage  set a scene
14     primaryStage.show(); // Display the stage                  display stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {                     main method
22     Application.launch(args);                                  launch application
23   }
24 }
```

Life of a javafx application

Step 1 - Create an instance of the Application class

The program class must extend Application

A JavaFX application derives from the Application class in the JavaFx package. It means that it automatically inherits standard attributes and methods.

A simple JavaFX displays a button in the window.

MyJavaFX.java

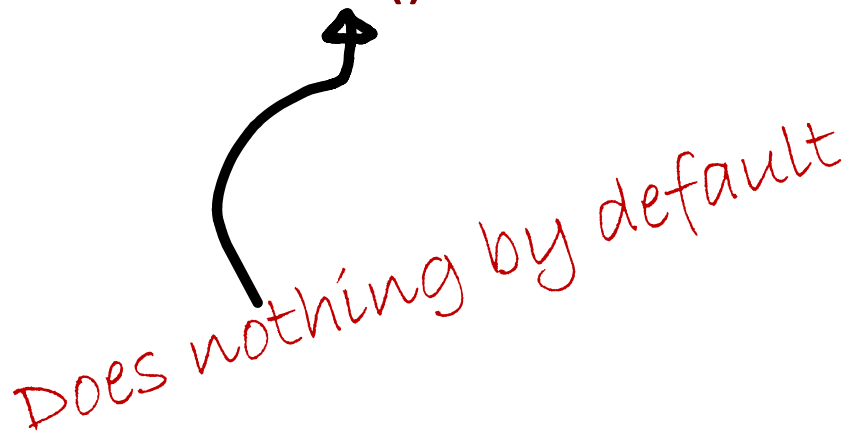
```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {                    extend Application
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {                      override start
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");                            create a button
11     Scene scene = new Scene(btOK, 200, 250);                   create a scene
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title  set stage title
13     primaryStage.setScene(scene); // Place the scene in the stage  set a scene
14     primaryStage.show(); // Display the stage                  display stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);
23   }
24 }
```

launch application

Life of a javafx application

Step 1 - Create an instance of the Application class

Step 2 - Call the init() method

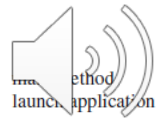


JavaFx will automatically call a function called init(). By default, this function does nothing. You can write your own version, and connect to a network or a database, or read a parameter file.

A simple JavaFX displays a button in the window.

MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {                    extend Application
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {                      override start
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");                            create a button
11     Scene scene = new Scene(btOK, 200, 250);                   create a scene
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title  set stage title
13     primaryStage.setScene(scene); // Place the scene in the stage  set a scene
14     primaryStage.show(); // Display the stage                  display stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);                                  launch application
23   }
24 }
```



Life of a javafx application

Step 1 - Create an instance of the Application class

Step 2 - Call the init() method

Must be rewritten!

Step 3 - Call the start(javafx.stage.Start) method

What you must write (and override) is a function called "start()" that takes a "Stage" (the name given to windows in JavaFx) as parameter. The function adds the widgets to the window and defines how it looks, and how widgets will react.

A simple JavaFX displays a button in the window.

MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {                    extend Application
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {                      override start
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");                            create a button
11     Scene scene = new Scene(btOK, 200, 250);                   create a scene
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title  set stage title
13     primaryStage.setScene(scene); // Place the scene in the stage  set a scene
14     primaryStage.show(); // Display the stage                  display stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);
23   }
24 }
```

Life of a javafx application

Step 1 - Create an instance of the Application class

Step 2 - Call the init() method

Step 3 - Call the start(javafx.stage.Stage) method

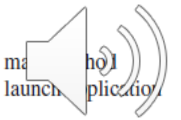
Step 4 - Wait for the application to finish:
the application calls Platform.exit()
or window closed

You must write the event handlers you need – *and nothing else* – JavaFx will run the application until it calls an exit routine (perhaps associated with a "Quit" button) or it receives the event "Window destroyed".

A simple JavaFX displays a button in the window.

MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {                    extend Application
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {                      override start
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");                            create a button
11     Scene scene = new Scene(btOK, 200, 250);                   create a scene
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title  set stage title
13     primaryStage.setScene(scene); // Place the scene in the stage  set a scene
14     primaryStage.show(); // Display the stage                  display stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);
23   }
24 }
```



Life of a javafx application

Step 1 - Create an instance of the Application class

Step 2 - Call the init() method

Step 3 - Call the start(javafx.stage.Stage) method

Step 4 - Wait for the application to finish:
the application calls Platform.exit()
or window closed

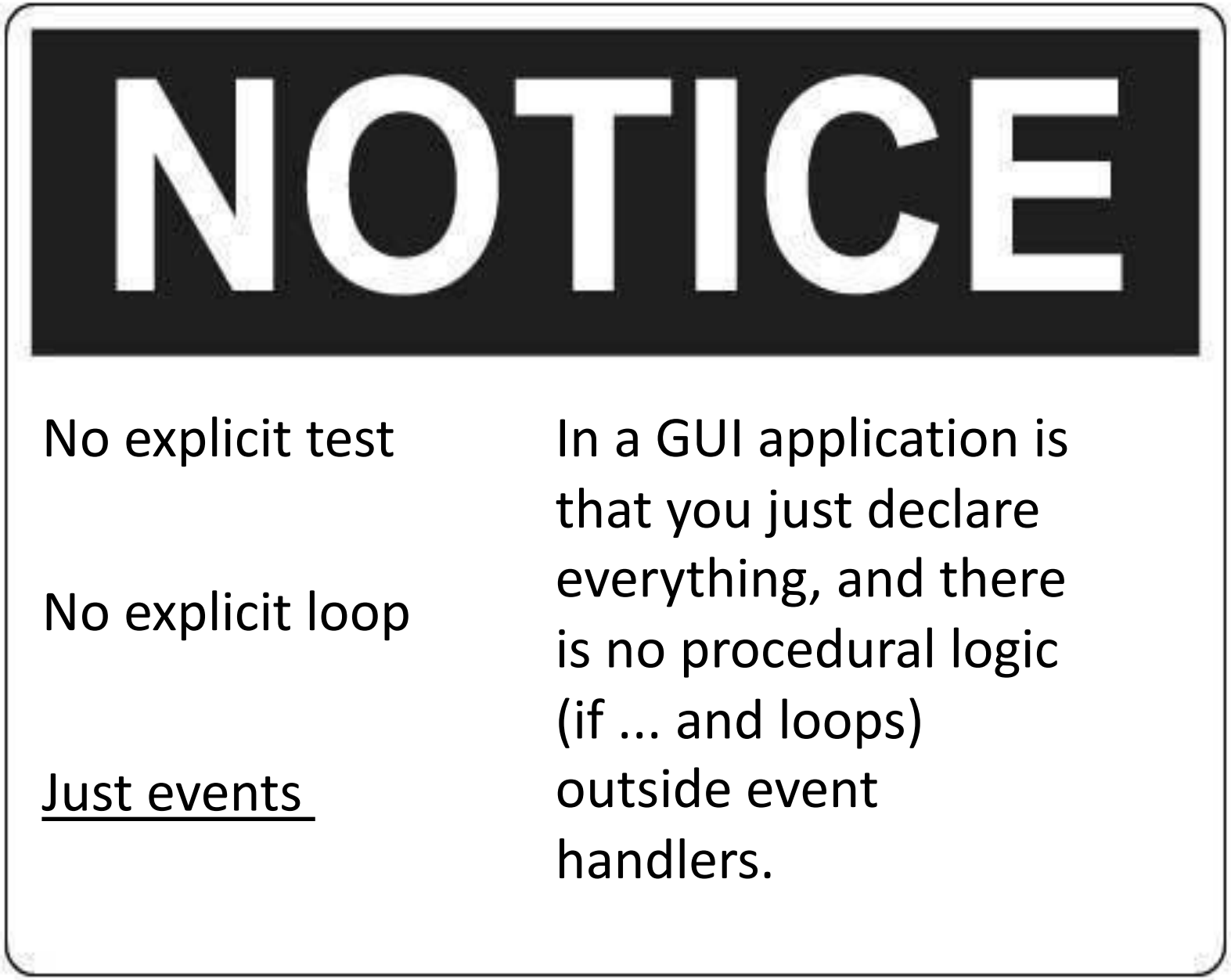
Step 5 - Call the stop() method

It will then call a stop() method where you can undo what you have done in init() – for example disconnect from a database or network. Like with init(), and exit() rewriting stop() is only done if you *need* to.

A simple JavaFX displays a button in the window.

MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {                    extend Application
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {                      override start
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");                            create a button
11     Scene scene = new Scene(btOK, 200, 250);                   create a scene
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title  set stage title
13     primaryStage.setScene(scene); // Place the scene in the stage  set a scene
14     primaryStage.show(); // Display the stage                  display stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {                     main method
22     Application.launch(args);                                  launch application
23   }
24 }
```


NOTICE
No explicit test
No explicit loop
Just events
In a GUI application is that you just declare everything, and there is no procedural logic (if ... and loops) outside event handlers.