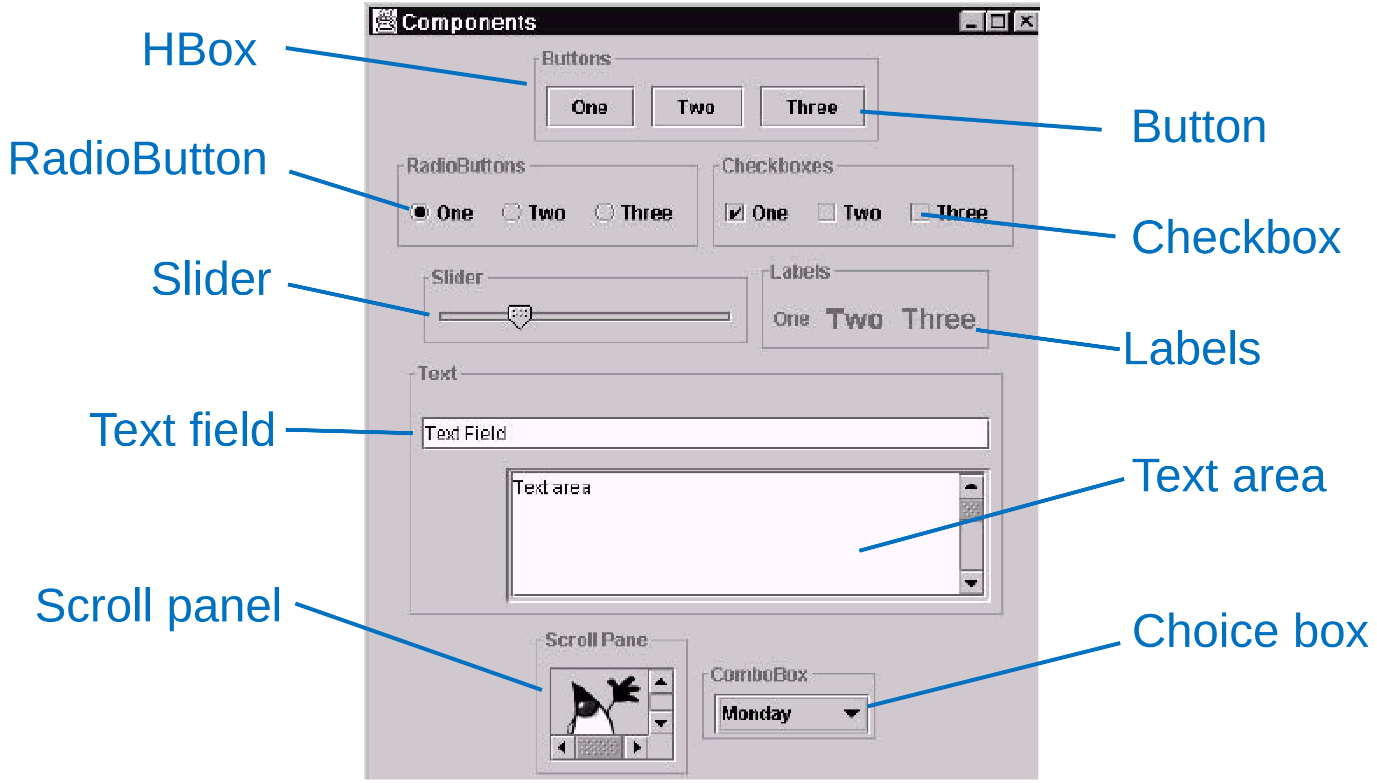
UNIVERSITY OF TWENTE.

GRAPHICAL USER INTERFACES (GUIs)

LECTURER: FAIZAN AHMED

GRAPHICAL USER INTERFACE

A CLASS FOR EACH GUI COMPONENT (TYPE)



GUI INGREDIENTS

COMPONENTS FOR A REACTIVE GUI

- Graphical components
  - Model-view-controller pattern
  - View components to display model state
  - Controller components for user input
- Layout managers: regulate how components are shown
- Reaction to user actions
  - Action listeners (Observer pattern)

A BIT OF HISTORY: AWT AND SWING

JAVA TOOLKITS FOR GUIS

- Abstract Windowing Toolkit (AWT, package `java.awt`)
  - In principle portable → should work in each platform!
  - Java wrappers around native platform libraries
  - Too much original behaviour → portability was lost!
- Swing (package `javax.swing`)
  - Since Java 1.2 (!) part of JFC (Java Foundation Classes)
  - Complete Java GUI → uses only native canvas, and draws Java (lightweight) components on it

CURRENT GUI TOOLKIT: JAVAFX

- JavaFX is a software platform for creating and delivering desktop applications
- Brings desktop applications closer to Internet Rich Applications (IRAs), which run on different devices
- Developed to replace Swing on the long run
- Not distributed with JDK 11 → download from https://gluonhq.com/products/javafx/ and define your own User Library
- Used in this lecture (and in Eck's book)

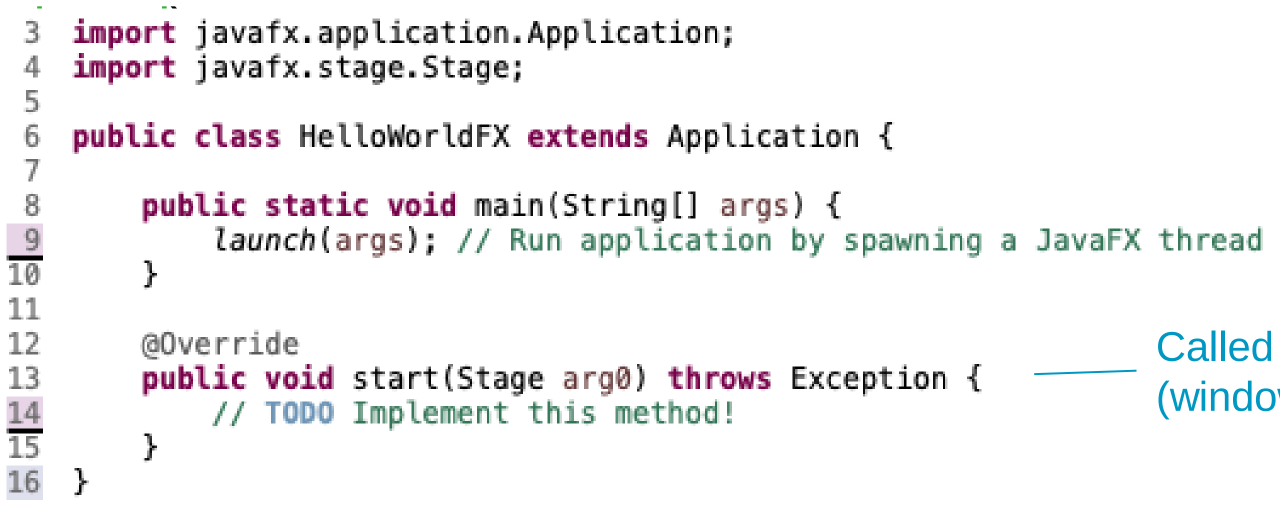
APPLICATION

CLASS TO REPRESENT A PROGRAM WITH A GUI

Strongly recommended!

- Programs should extend Application and implement main() method
- In main() method, application is launched

  → Objects are created and start method is called (JavaFX



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

public class HelloWorldFX extends Application {

    public static void main(String[] args) {
        launch(args); // Run application by spawning a JavaFX thread
    }

    @Override
    public void start(Stage arg0) throws Exception {
        // TODO Implement this method!
    }
}
```

Called with a Stage (window, from the OS)

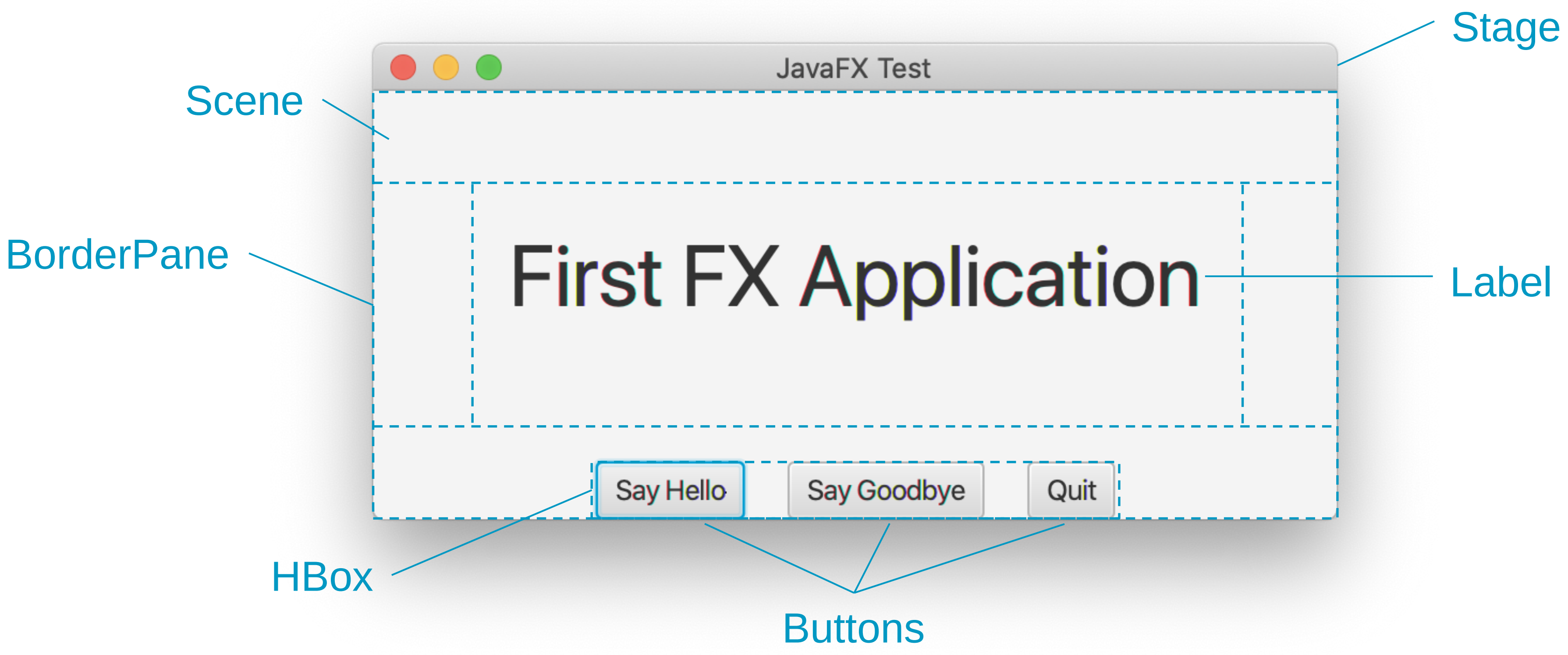
APPLICATION METHODS

- init() to initialise the application before calling start()
- stop() to shut down the application, e.g., releasing resources if necessary
- Both methods have a dummy (empty) default implementation

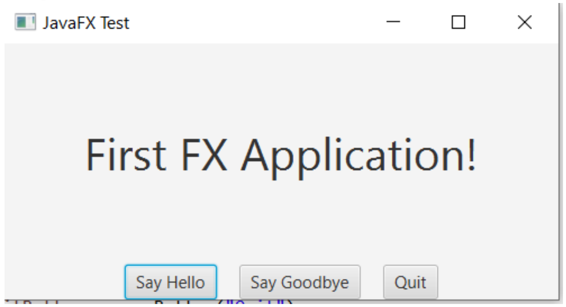
- Normally it is enough to implement start()

SIMPLE EXAMPLE

TO INTRODUCE THE MAIN CONCEPTS

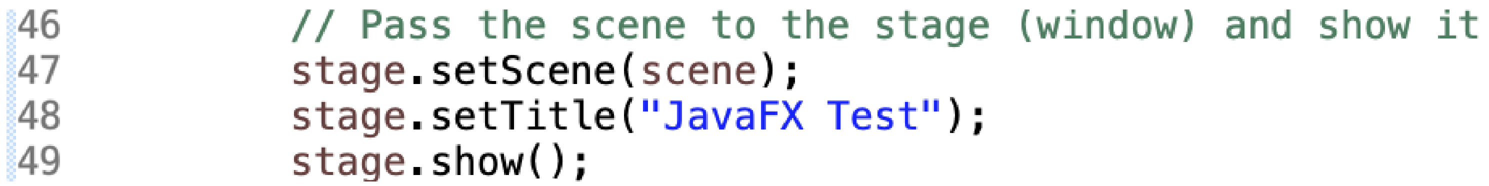


```
    public void start(Stage stage) {
        // Create a Label object
        Label message = new Label ("First FX Application");
        message.setFont(new Font(40));

        // Create three buttons
        Button helloButton = new Button("Say Hello");
        helloButton.setOnAction( e -> message.setText("Hello World!"));
        Button byeButton = new Button("Say Goodbye");
        byeButton.setOnAction( e -> message.setText("Goodbye!!"));
        Button quitButton = new Button("Quit");
        quitButton.setOnAction( e -> System.exit(0));

        // Aggregate the buttons in an HBox
        HBox buttonBar = new HBox (20, helloButton, byeButton, quitButton);
        buttonBar.setAlignment(Pos.CENTER);

        // Create a Border pane
        BorderPane root = new BorderPane();
        root.setCenter(message);
        root.setBottom(buttonBar);

        // Assign this pane to a scene
        Scene scene = new Scene (root, 450, 200);

        // Pass the scene to the stage (window) and show it
        stage.setScene(scene);
        stage.setTitle("JavaFX Test");
        stage.show();
    }
```



STAGE AND SCENE

- Application gets a Stage with the start() method, which represents a window on the screen (main window of a program)
- New windows can be created by creating more Stage objects
- Application needs to fill in and show the main window
- A Stage shows a Scene, which is a container for GUI components

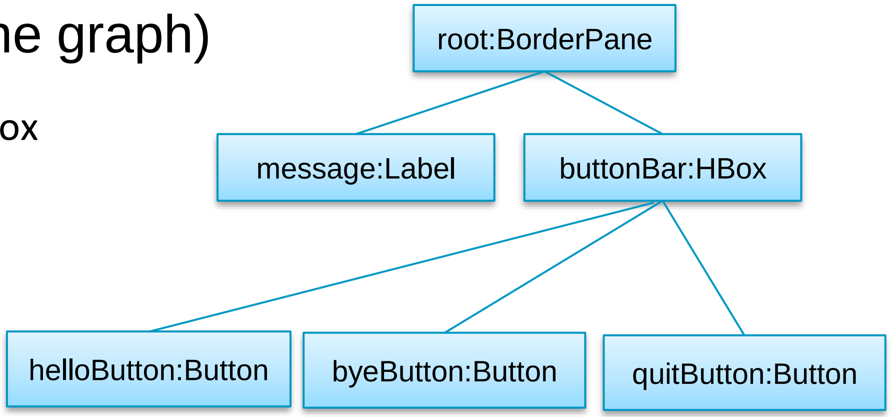


```
        // Pass the scene to the stage (window) and show it
        stage.setScene(scene);
        stage.setTitle("JavaFX Test");
        stage.show();
```

UNIVERSITY OF TWENTE.

GUI CONTAINERS

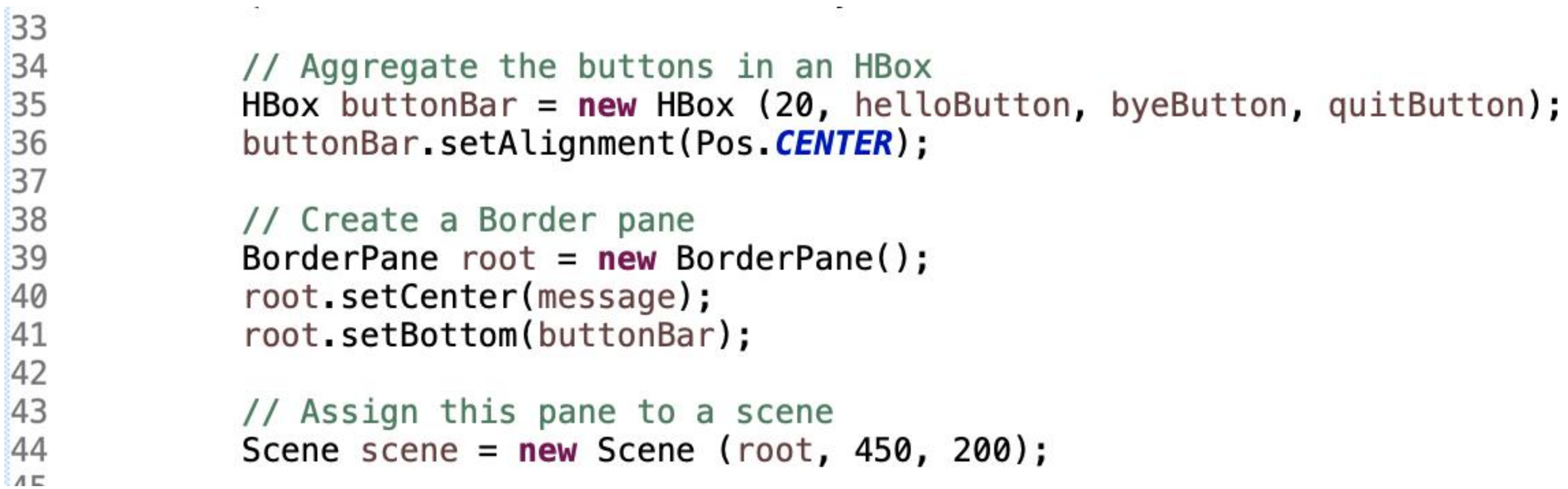
- A Scene is a container for GUI components, which can be itself also GUI containers, forming a so-called scene graph
- In the example:
  - Scene contains a BorderPane (root of the graph)
  - BorderPane contains a Label and an HBox
  - HBox contains three Buttons



SCENE GRAPH NODES

- To stress the scene graph structure, parts of a scene graph are nodes (subclasses of `javafx.scene.Node`)
- A graph object can only be a container if it is a subclass of `javafx.scene.Parent`
- Since `Parent` nodes can have children nodes, these must be somehow arranged on the screen (layout)
- Different `Parent` nodes may have different layout policies (e.g., `HBox` uses horizontal rows, `BorderPane` uses 5 regions, etc.)

PARENT NODES AND CONTAINMENT IN THE EXAMPLE



```java

        // Aggregate the buttons in an HBox
        HBox buttonBar = new HBox (20, helloButton, byeButton, quitButton);
        buttonBar.setAlignment(Pos.CENTER);

        // Create a Border pane
        BorderPane root = new BorderPane();
        root.setCenter(message);
        root.setBottom(buttonBar);

        // Assign this pane to a scene
        Scene scene = new Scene (root, 450, 200);
```

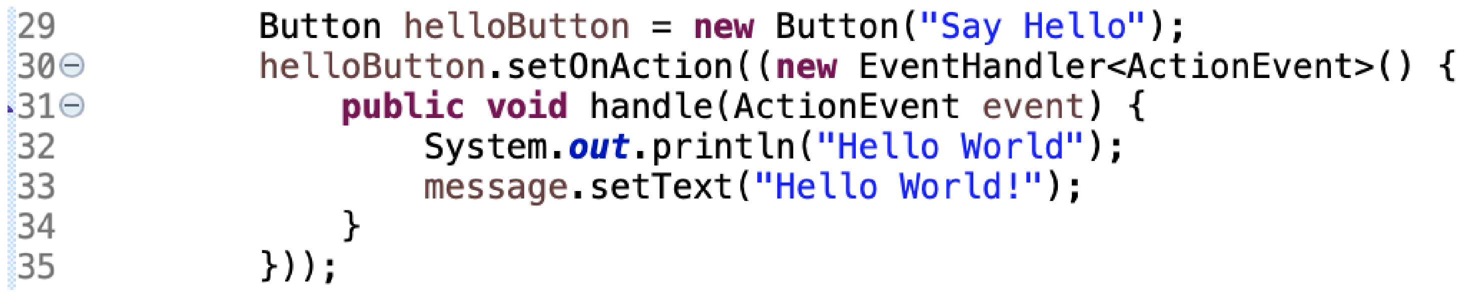
UNIVERSITY OF TWENTE.

EVENT-DRIVEN PROGRAMMING

HOW TO MAKE SOMETHING HAPPEN WHEN BUTTON IS PRESSED?

- Events like pressing a button have to be handled
- An Event contains information about what happened
- An EventHandler handles an Event

Example without lambda expression

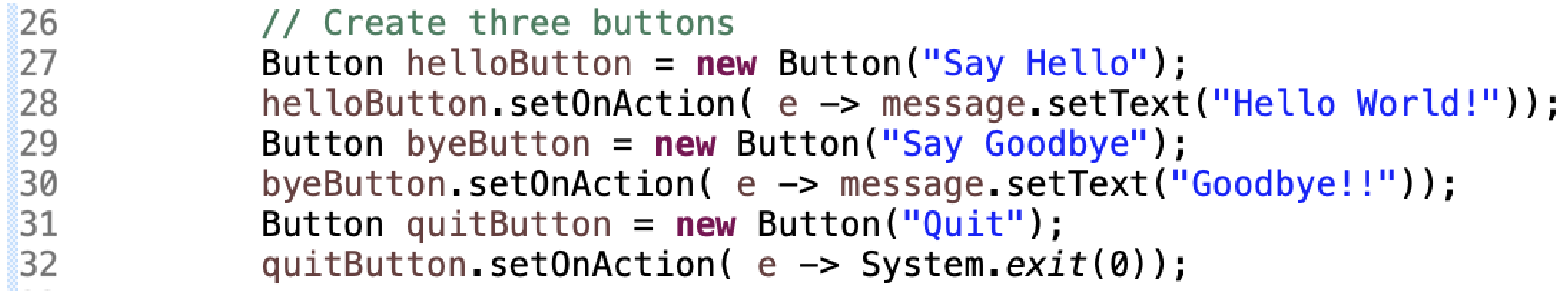


```
Button helloButton = new Button("Say Hello");
helloButton.setOnAction((new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        System.out.println("Hello World");
        message.setText("Hello World!");
    }
}));
```

UNIVERSITY OF TWENTE.

EVENTHANDLER
DEFINED WITH A LAMBDA EXPRESSION

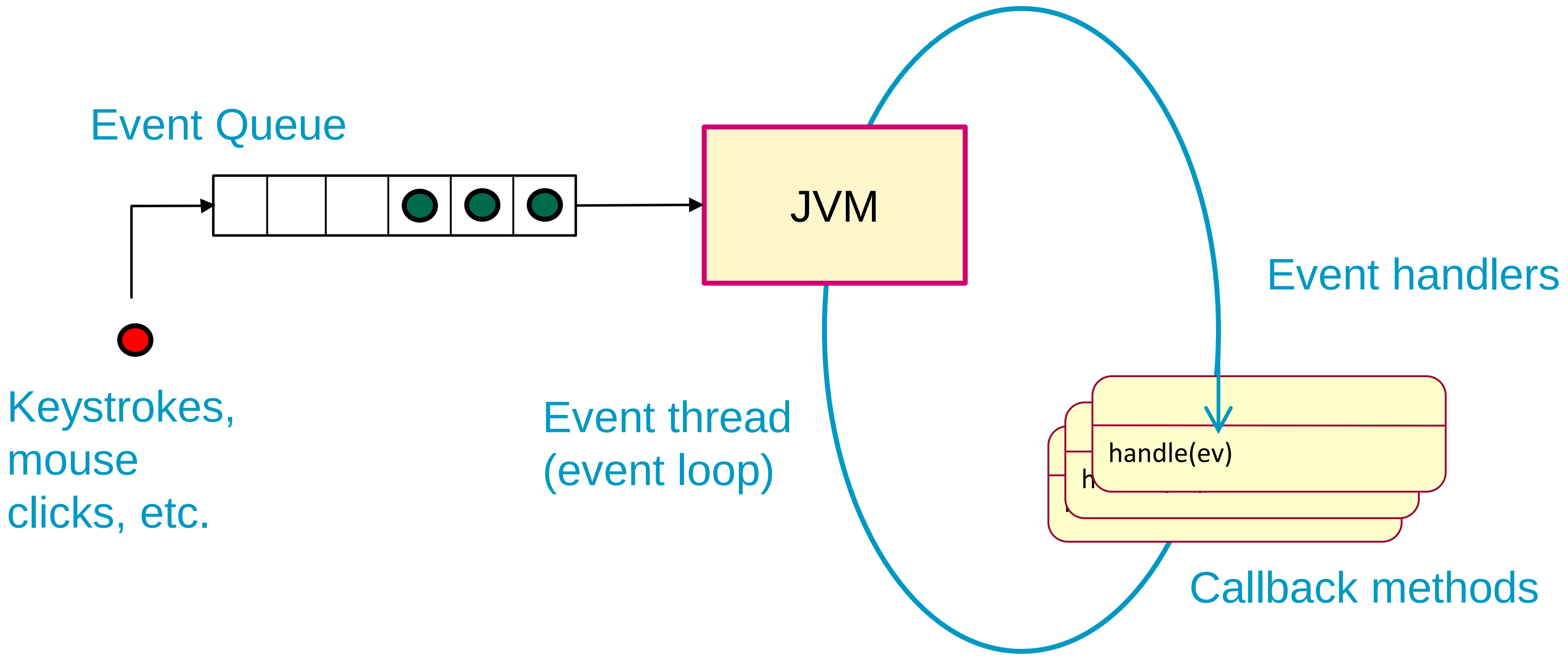
- EventHandler is a functional interface (defines a single function handle(Event e)), so it can be used as target for a lambda expression



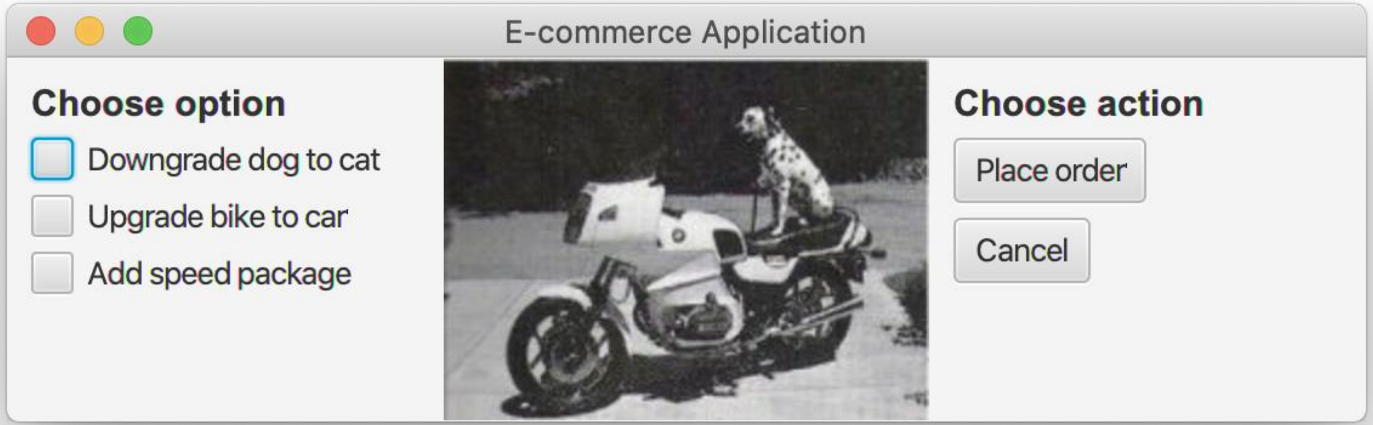
```
        // Create three buttons
        Button helloButton = new Button("Say Hello");
        helloButton.setOnAction( e -> message.setText("Hello World!"));
        Button byeButton = new Button("Say Goodbye");
        byeButton.setOnAction( e -> message.setText("Goodbye!!"));
        Button quitButton = new Button("Quit");
        quitButton.setOnAction( e -> System.exit(0));
```

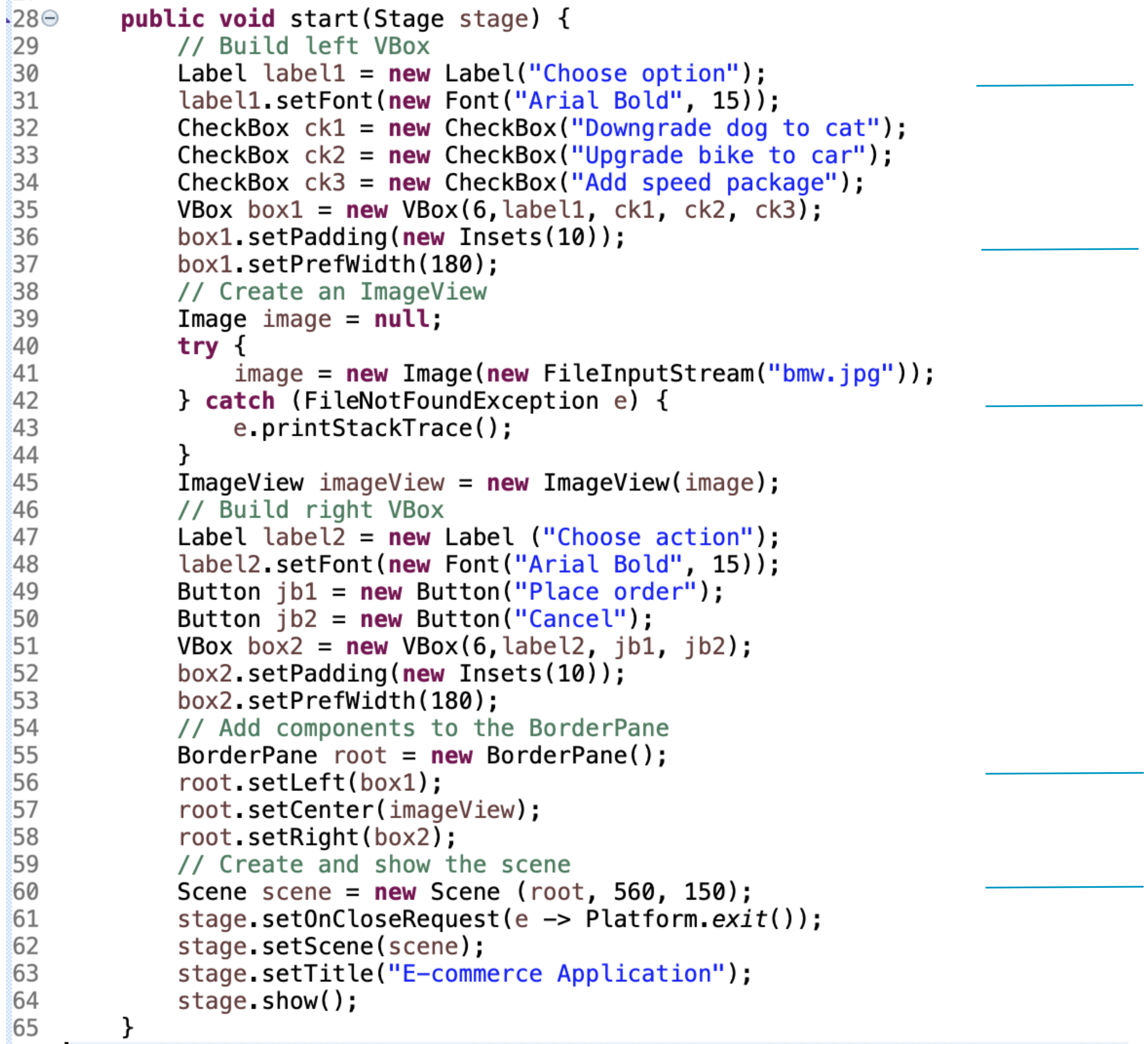
UNIVERSITY OF TWENTE.

EVENT-DRIVEN PROGRAMMING



YET ANOTHER EXAMPLE





```java
    public void start(Stage stage) {
        // Build left VBox
        Label label1 = new Label("Choose option");
        label1.setFont(new Font("Arial Bold", 15));
        CheckBox ck1 = new CheckBox("Downgrade dog to cat");
        CheckBox ck2 = new CheckBox("Upgrade bike to car");
        CheckBox ck3 = new CheckBox("Add speed package");
        VBox box1 = new VBox(6,label1, ck1, ck2, ck3);
        box1.setPadding(new Insets(10));
        box1.setPrefWidth(180);
        // Create an ImageView
        Image image = null;
        try {
            image = new Image(new FileInputStream("bmw.jpg"));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        ImageView imageView = new ImageView(image);
        // Build right VBox
        Label label2 = new Label ("Choose action");
        label2.setFont(new Font("Arial Bold", 15));
        Button jb1 = new Button("Place order");
        Button jb2 = new Button("Cancel");
        VBox box2 = new VBox(6,label2, jb1, jb2);
        box2.setPadding(new Insets(10));
        box2.setPrefWidth(180);
        // Add components to the BorderPane
        BorderPane root = new BorderPane();
        root.setLeft(box1);
        root.setCenter(imageView);
        root.setRight(box2);
        // Create and show the scene
        Scene scene = new Scene (root, 560, 150);
        stage.setOnCloseRequest(e -> Platform.exit());
        stage.setScene(scene);
        stage.setTitle("E-commerce Application");
        stage.show();
    }
```

Change font

Define padding

Create ImageView

Add components to BorderPane

Define what to do when window is closed

TAKE HOME MESSAGES

- GUI component classes: Button, Label, CheckBox, RadioButton, etc.
- Layout policies are used to position components in a parent node (e.g., a BorderPane)
- EventHandler: controller in the MVC pattern, to be added to GUI components (similar to Observer role)
- Many more facilities not discussed here!
- GUIs can get complex, so tools like Scene Builder can help!