Topic 6: GUIs, Events and FSMs

ICT373: Software Architectures

Recap

- Streams, and Persistence
- Development of Streams for i/o and data transfer.
- Use of streams to implement persistence of data and objects beyond the lifetime of a program.
- RTTI, Java parameter passing, aliasing, cloning and immutable objects.

Outline

- Events and FSMs
- Java Swing
- JavaFX
- Beans
- GUI design

Reading: Textbook (11th ed) Chapters 12, 13, 22

Murdoch UNIVERSITY

Objectives

- Describe the main considerations of a GUI designer.
- Explain the structure of a Finite State Machine and its use.
  - What is a State Transition Diagram?
  - How is an FSM represented in UML notation?
- Describe the Java AWT/JavaFX event model.
- Implement a GUI in Java.
- Implement event handling in java.
- Explain the purpose of JAR.
- Explain the purpose of Beans.
- What is meant by separating the GUI from the business logic.

Murdoch UNIVERSITY

GUIs, Events and JavaFX

- Graphical User Interfaces.
  - presents a user friendly mechanism for interacting with an app.
- GUIs which look nice and are easy to use are expected of most software these days.
- In good GUI design you will be concerned with:
  - nice looking components arranged nicely on the screen
  - correct, predictable and reasonable behaviour
  - separating the GUI implementation from the real computations underneath (i.e. **the *business logic***).

GUIs, Events and JavaFX

- The look of the GUI depends on what the programming language makes available
  - Java's Swing package (javax.swing)
  - JavaFX
- The behaviour of individual components also depends on what is available in the language but the more important question about overall GUI behaviour can be answered more generally.
- The interaction between the GUI and the business logic is also a question for the software architect.
- We will look first at the overall GUI behaviour.
  - A GUI will be able to be in several different ***states*** and will be able to undergo ***transitions*** from one state to another due to events in its ***environment,*** i.e., the user, internal computations or network connections.

Murdoch UNIVERSITY

States and Transitions

- A **State** represents a situation during the operation of a system (or during the life of an object) in which it satisfies some condition, performs an action, or waits for some event to occur. All states are given names.
- A **Transition** represents the relationship between two states indicating that a system (or an object) will perform an action to transfer from one state to another.
- In many applications in Computer Science and Electronic Engineering there is a need to represent, design, check, reason about or prove something about a system which can be in one of a (finite) number of different states and can undergo transitions from one state to another due to events in its environment.

States and Transitions

- Examples include coin-operated machines, VLSI chips, the traffic lights at a road intersection, the signals on the London Underground, an aeroplane, the electrons around an atom.
- Theoretical approaches to reasoning about such systems actually began with studies of groups of neurons in the brain.
- In all such examples, useful questions can be answered by abstracting away from the details of the situation and just considering the system from the point of view of which states it can be in and of which transitions between states can occur as a result of environmental events.
- State machine diagrams are a familiar technique for describing the behaviour of a system.

Finite State Machine

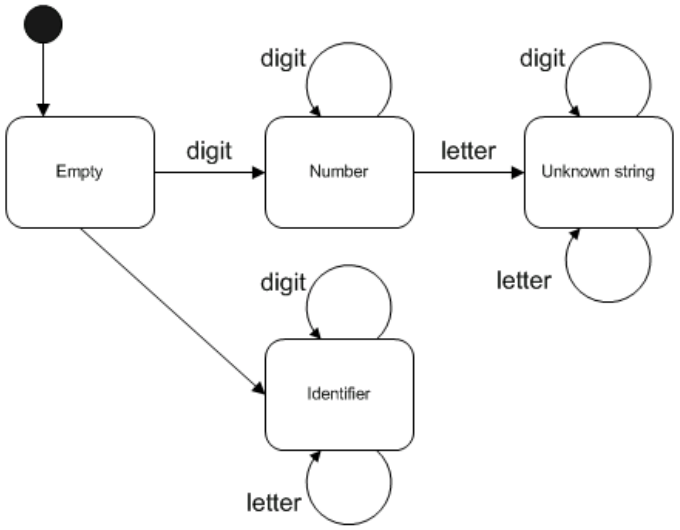
- A **Finite State Machine** (FSM) is a hypothetical machine (an abstract mathematical object).
- FSMs produce an overall system or component design, which can be checked, often mechanically, for correctness of its behaviour.
- Properties of FSM
  - FSM has a fixed finite set of states and
  - a fixed finite set of transitions.
  - It can only be in one of its states at any specific time.
  - Each transition has a source state (where it comes from), a destination state, an input (condition) and possibly an output.
  - In response to the input and its current state the machine generates an output and changes state.

FSM

- Variations are available.
- In more formal approaches, inputs will have to come from a precisely defined formal alphabet.
- In the rough plan for a GUI, e.g., input conditions can be natural language sentences describing user or network behaviour. But the more precise, the better the design.
- Some approaches allow/require:
  - specifying a start state
  - specifying end states (with no transitions out of them).

FSM

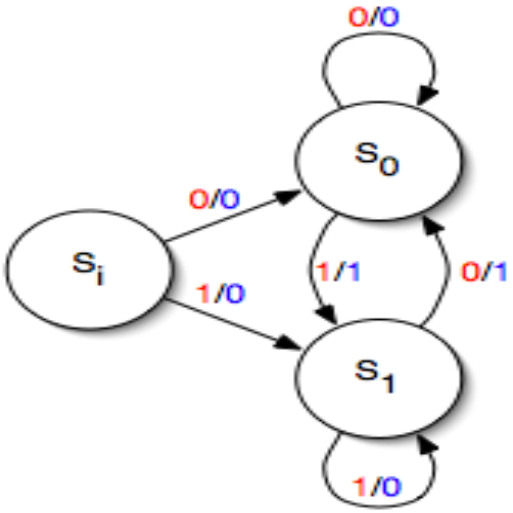
- In UML, states are denoted by rectangles with rounded corners, except for the initial state which is denoted by a small filled circle.
- A solid arrow with appropriate method invocation denotes a transition.

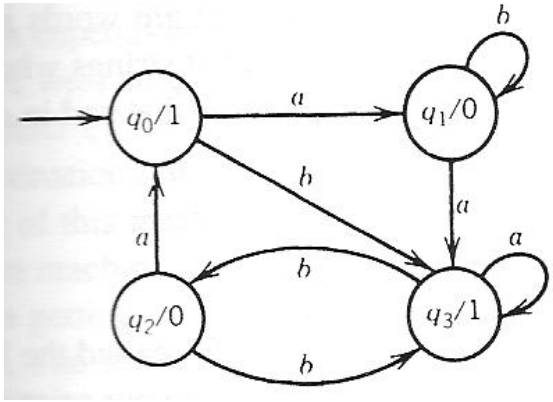


Murdoch UNIVERSITY

State Transition Diagrams (STDs)

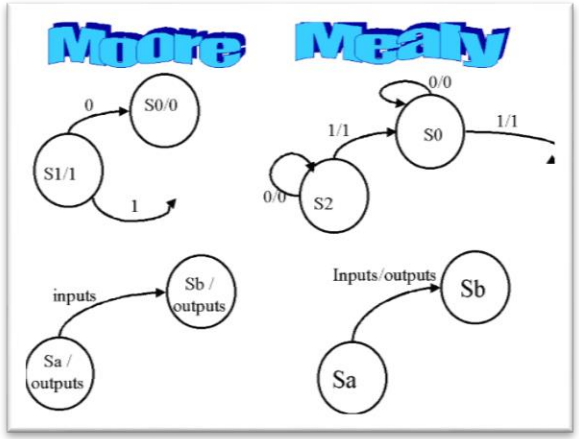
- A particular FSM can be defined by listing its states and then listing (usually in a table) its transitions.
- It is often useful to have a diagrammatic representation, e.g. an STD.
- States are shown as (e.g.) labelled circles and transitions as labelled arrows (directed arcs) between them. States are usually named but transitions do not usually need to be and are just labelled with input and output information in some standard format, e.g. input/output.





Murdoch UNIVERSITY

Things to Check (1/2)



Decide what type of FSM is appropriate for your situation,

1.Does output seem to correspond to states? eg, what is seen (or heard) from your system just depends on the state it is in and there is no extra events occurring on transitions (called a **Moore machine**).

- On Moore STDs, only the stimulus (input) name is on the arc, the response is coupled with the state in the circle.

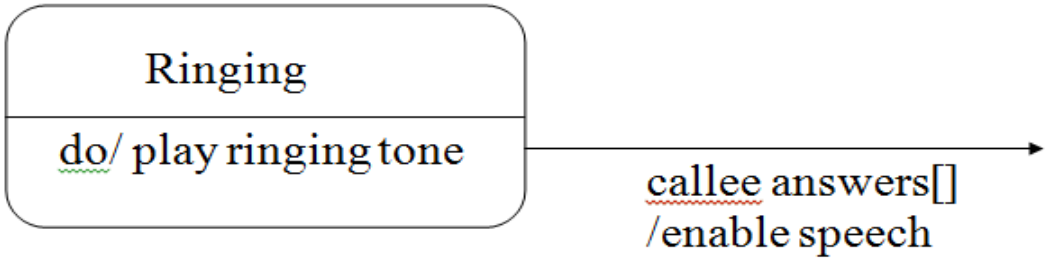
2.Does output seem to correspond to transitions (ie combination of state and input 'symbol')? eg, when there is a transition then something happens which is visible to the outside world (a **Mealy machine**).

Murdoch UNIVERSITY

Things to Check (2/2)

- Once the FSM is complete, then check for:
  - **consistency, determinism**: for any given initial state and any given input, is there only one state that the machine can move to?
  - **completeness**: for any given state and any given input is there an appropriate transition?
  - **reachability**: for any given state, is there a path to it from the start state and is there a path from it to an end state?
- Checking such things is easy to do on an STD and will help ensure that you have a valid design.

The UML version of STDs



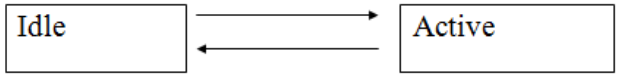
- STDs (of FSMs) are so useful in design that UML has its own version of STD notation. The idea is that they are used to represent the states that a particular object goes through during its lifetime or just during the execution of a particular method.
- The use of such STDs should not be confined to designing GUIs. Any object in an O-O program has a range of states which it goes through:
- these are just the collective values of all the component fields of the object. If your O-O program has an object which is designed to go through various transitions between a finite number of states then it might be useful to use an STD. This is true of many objects, even quite deeply hidden ones, as well as GUI objects.

Telephone Example

(Source: Figure 3.71, page 3-137 of OMG's UML1.5 docs)

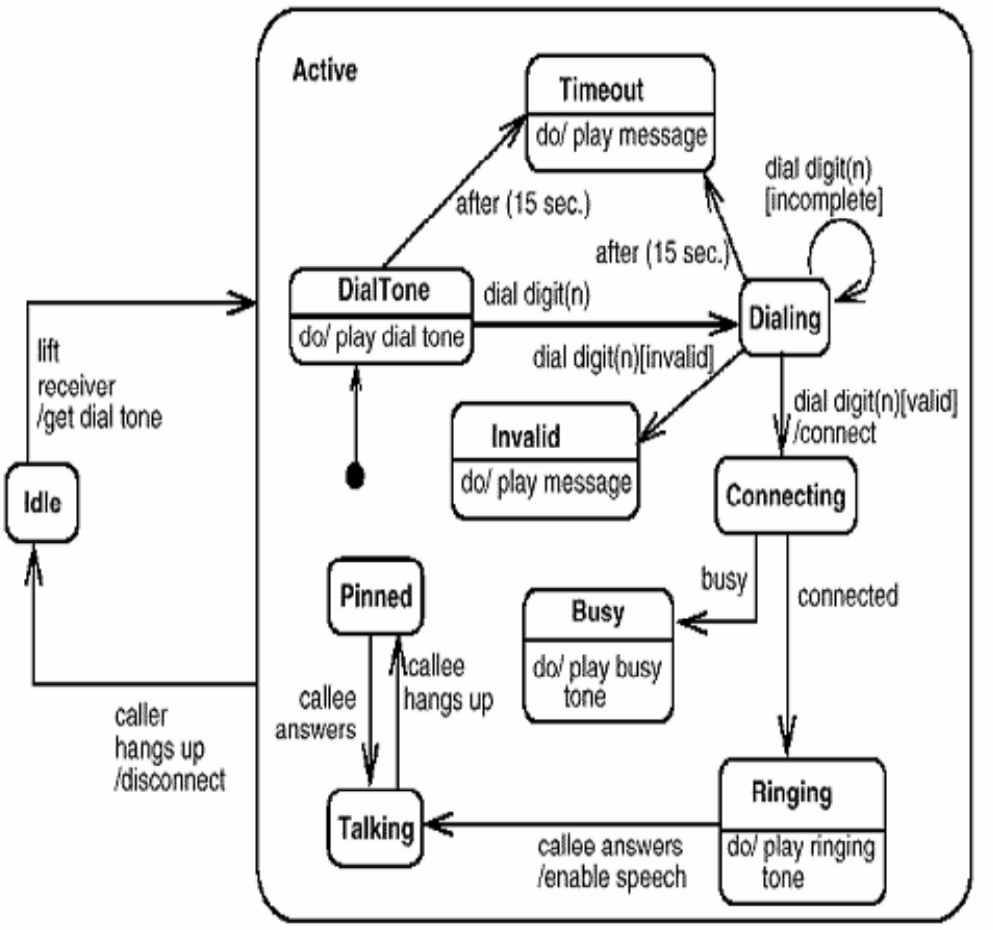
- Nested FSM.

The idea is that at a higher level of abstraction the system can be modelled as



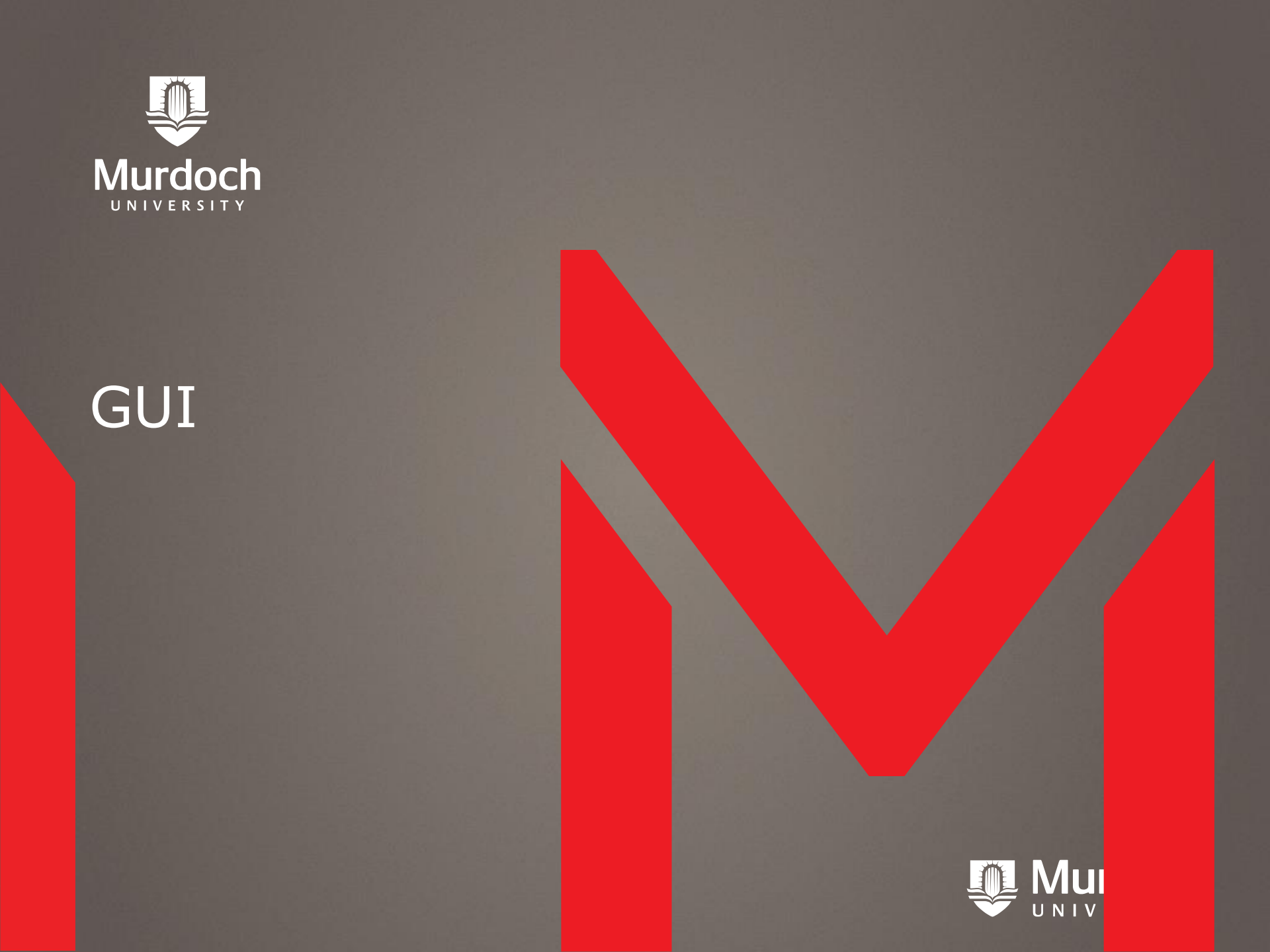
- "Active" states can really be subdivided, if we look with a finer granularity.

Figure 41. State diagram



Granularity and Abstraction

- If you consider some of the states in Active, eg “Dialling” then you might further regard them as whole FSM’s with their own internal states at an even finer granularity.
- Dealing with different granularities, i.e. different levels of abstraction, enables the designer to give an overall picture and hide details.


Murdoch
UNIVERSITY
GUI
Mu
UNIV

Swing vs JavaFX

- JavaFX may be a GUI toolkit for Java (GUI is brief for Graphical User Interface).
- JavaFX makes it easier to form desktop applications and games in Java.
- JavaFX has replaced Swing because of the suggested GUI toolkit for Java.

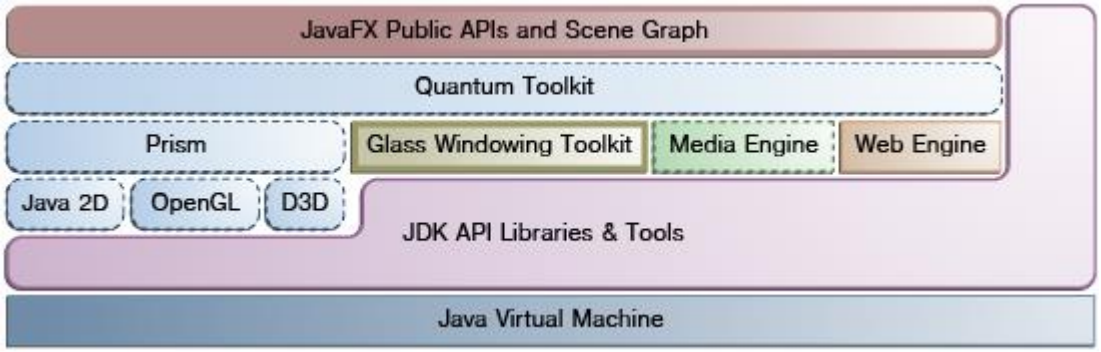
More: e.g., https://www.educba.com/javafx-vs-swing/

In this unit we will use JavaFX

JavaFX

- A *Graphical User Interface* (GUI) is a system of visible components (such as windows, menus, buttons, text fields) which allow a program to interact with a user.
  - Modern programs use these windowing interfaces to allow the user to make choices with a mouse.
- JavaFX is a software platform for creating and delivering desktop applications, as well as rich Internet applications (RIAs) that can run across a wide variety of devices.
  - JavaFX is actually a different language (similar, but different syntax), but it runs on a JVM and can use Java classes.
  - JavaFX requires a JVM; but just having Java doesn't mean you have JavaFX. Thus, JavaFX is not inherently a part of Java.
  - any platform that doesn't support Java would therefore not support JavaFX.

JavaFX Architecture



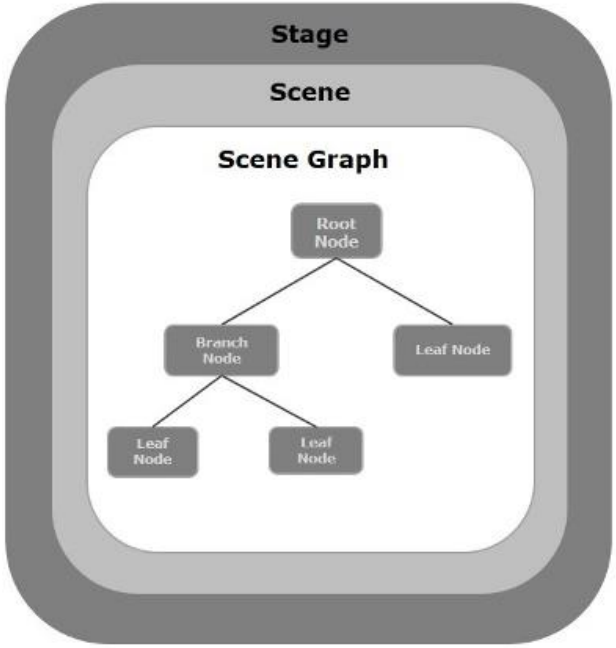
- The top layer of the JavaFX architecture provides a complete set of Java public APIs that support rich client application development.
- These APIs provide unparalleled freedom and flexibility to construct rich client applications.
- The JavaFX platform combines the best capabilities of the Java platform with comprehensive, immersive media functionality into an intuitive and comprehensive one-stop development environment.

https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#A1106498

https://www.tutorialspoint.com/javafx/javafx_architecture.htm

Murdoch UNIVERSITY

JavaFX Application Structure



A JavaFX application will have three major components: Stage, Scene and Node

Stage: A stage (a window) contains all the objects of a JavaFX application.

- It is represented by Stage class of the package *javafx.stage*.
- The primary stage is created by the platform itself.
- The created stage object is passed as an argument to the *start()* method of the Application class.
- A stage has two parameters determining its position namely *Width* and *Height*.
- Call *show()* method to display the contents of a stage.

https://www.tutorialspoint.com/javafx/javafx_application.htm

Murdoch UNIVERSITY

JavaFX Application Structure

Scene: A scene represents the physical contents of a JavaFX application.

- It contains all the contents of a scene graph.
- The class **Scene** of the package **javafx.scene** represents the scene object. At an instance, the scene object is added to only one stage.
- You can create a scene by instantiating the Scene Class.
- You can opt for the size of the scene by passing its dimensions (height and width) along with the **root node** to its constructor.

JavaFX Application Structure

Scene Graph and Nodes: A **scene graph** is a tree-like data structure (hierarchical) representing the contents of a scene.

- A **node** is a visual/graphical object of a scene graph.
- The **Node** Class of the package **javafx.scene** represents a node in JavaFX, this class is the super class of all the nodes.
- A node may include –
  - Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon.
  - UI Controls such as – Button, Checkbox, Choice Box, Text Area.
  - Containers (Layout Panes) e.g., Border Pane, Grid Pane, Flow Pane,.
  - Media elements such as Audio, Video and Image Objects.
- **Root Node** – The first Scene Graph is known as the Root node.
- **Leaf Node** – A node without child nodes is known as the leaf node.

Creating a JavaFX Application

To create a JavaFX application

- you need to instantiate the Application class and
- implement its abstract method **start()**.

The **Application** class of the package **javafx.application** is the entry point of the application in JavaFX.

- To create a JavaFX application, you need to inherit this class and implement its abstract method **start()**. In this method, you need to write the entire code for the JavaFX graphics.

In the **main** method, you have to launch the application using the **launch()** method.

- This method internally calls the **start()** method of the Application class as shown in the following program.

Murdoch UNIVERSITY

Creating a JavaFX Application

```
public class JavafxSample extends Application {
   @Override
   public void start(Stage primaryStage) throws Exception {
      /*
      Code for JavaFX application.
      (Stage, scene, scene graph)
      */
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

- Within the **start()** method, in order to create a typical JavaFX application, you need to follow the steps given below −
    - Prepare a scene graph with the required nodes.
    - Prepare a Scene with the required dimensions and add the scene graph (root node of the scene graph) to it.
    - Prepare a stage and add the scene to the stage and display the contents of the stage.

Murdoch UNIVERSITY

Creating a JavaFX Application

Preparing the Scene Graph

- As per your application, you need to prepare a scene graph with required nodes.
- Since the root node is the first node, you need to create a root node. As a root node, you can choose from the **Group, Region or WebView**. E.g.,

```
Group root = new Group();
```

- The **getChildren()** method of the **Group** class gives you an object of the **ObservableList** class which holds the nodes.

```
//Retrieving the observable list object
ObservableList list = root.getChildren();
//Setting the text object as a node
list.add(NodeObject);
```

Murdoch UNIVERSITY

Creating a JavaFX Application

- We can also add Node objects to the group, just by passing them to the **Group** class and to its constructor at the time of instantiation:

```
Group root = new Group(NodeObject);
```

- In a Group, you can instantiate a few other classes and use them as root nodes, e.g.,:

```
//Creating a Stack Pane
StackPane pane = new StackPane();
//Adding text area to the pane
ObservableList list = pane.getChildren();
list.add(NodeObject);
```

Creating a JavaFX Application

Preparing the Scene

- Create a Scene by instantiating this class. While instantiating, it is mandatory to pass the root object to the constructor of the scene class.

```
Scene scene = new Scene(root);
```

- You can also pass two parameters of double type representing the height and width of the scene as shown below.

```
Scene scene = new Scene(root, 600, 300);
```

Murdoch UNIVERSITY

Creating a JavaFX Application

Preparing the Stage

- An object of this class is passed as a parameter of the **start()** method of the Application class.
- Using this object, you can perform various operations on the stage. Primarily you can perform the following –
  - Set the title for the stage using the method setTitle().
  - Attach the scene object to the stage using the setScene() method.
  - Display the contents of the scene using the **show()** method as shown below.

```
primaryStage.setTitle("Sample application"); //Setting the title to Stage.

primaryStage.setScene(scene); //Setting the scene to Stage

primaryStage.show(); //Displaying the stage
```

- Example: DisplayingText; JavaFXTest; MultiStage

Murdoch UNIVERSITY

Lifecycle of JavaFX Application

The JavaFX Application class has three life cycle methods:

- **start()** – The entry point method where the JavaFX graphics code is to be written.
- **stop()** – An empty method which can be overridden, here you can write the logic to stop the application.
- **init()** – An empty method which can be overridden, but you cannot create stage or scene in this method.

In addition to these, it provides a static method named **launch()** to launch JavaFX application. launch() is static; need to call it from a static context (main generally). Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).

- An instance of the application class is created.
- Init() method is called.
- The start() method is called.
- The launcher waits for the application to finish and calls the stop() method.

Terminating the JavaFX Application

When the last window of the application is closed, the JavaFX application is terminated implicitly.

- You can turn this behavior off by passing the Boolean value “False” to the static method **setImplicitExit()** (should be called from a static context).
- You can terminate a JavaFX application explicitly using the methods **Platform.exit()** or **System.exit**(int).

Overview of GUI components

Every user interface considers the following three main aspects –

- **UI elements** – These are the core visual elements which the user eventually sees and interacts with.
- **Layouts** – They define how UI elements should be organized on the screen and provide a final look and feel to the GUI.
- **Behavior** – These are events which occur when the user interacts with UI elements. These are so called Event Handlers.

JavaFX provides several classes in **javafx.scene.control**.

- Each control is represented by a class; a control is created by instantiating its respective class.

Commonly used GUI controls

- **Label:** A Label object is a component for placing text.
- **Button:** This class creates a labeled button.
- **CheckBox:** A CheckBox is a graphical component that can be in either an on(true) or off (false) state.
- **RadioButton:** The RadioButton class is a graphical component, which can either be in a ON (true) or OFF (false) state in a group.
- **ListView:** A ListView component presents the user with a scrolling list of text items.
- **TextField:** A TextField object is a text component that allows for the editing of a single line of text.
- **Scrollbar:** A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- **FileChooser:** A FileChooser control represents a dialog window from which the user can select a file.

Examples

- Registration
- JavaFXSimpleAPP

Murdoch UNIVERSITY

JavaFX - Layout Panes(Containers)

- After constructing all the required nodes in a scene, we will generally arrange them in order.
- JavaFX provides several predefined layouts such as

HBox,	**VBox,**	**Border Pane,**
Stack Pane,	**Text Flow,**	**Anchor Pane,**
Title Pane,	**Grid Pane,**	**Flow Panel**

- A layout is represented by a class and all these classes belongs to the package **javafx.layout**.
- The class named **Pane** is the base class of all the layouts in JavaFX.

Murdoch UNIVERSITY

Creating a Layout

To create a layout, you need to –

- Create node.
- Instantiate the respective class of the required layout.
- Set the properties of the layout.
- Add all the created nodes to the layout.

Creating Nodes: Create the required nodes.

- Example: create a text field and two buttons namely, play and stop in a HBox layout

```
TextField textField = new TextField(); //Creating a text field
Button playButton = new Button("Play"); //Creating the play button
Button stopButton = new Button("stop"); //Creating the stop button
```

Murdoch UNIVERSITY

Creating a Layout

Instantiating the Respective Class:

- After creating the nodes (and completing all the operations on them), instantiate the class of the required layout.
- Example, to create a Hbox layout:

```
HBox hbox = new HBox();
```

Setting the Properties of the Layout:

- Example: If you want to set space between the created nodes in the HBox layout, then you need to set value to the property named spacing.

```
hbox.setSpacing(10);
```

Murdoch UNIVERSITY

Creating a Layout

Add all the created nodes to the layout:

```
hbox.getChildren().addAll(textField,playButton,stopButton);
```

Layout Panes

- **Hbox:** The HBox layout arranges all the nodes in our application in a single horizontal row.
- **Vbox:** The VBox layout arranges all the nodes in our application in a single vertical column.
- **BorderPane:** The Border Pane layout arranges the nodes in our application in top, left, right, bottom and center positions.
- **StackPane:** The stack pane layout arranges the nodes in our application on top of another just like in a stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it.
- **FlowPane:** The flow pane layout wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width.
- **GridPane:** The Grid Pane layout arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms using JavaFX.
    - Program Example: GridPaneExample

JavaFX - Event Handling

- Whenever a user interacts with the application (nodes), an event is said to have been occurred.
- Types of Events: two categories

Foreground Events

- require the direct interaction of a user.
- generated as consequences of a person interacting with the graphical components in a GUI.
- Example: clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page.

Background Events

- Those events that require the interaction of end user are known as background events.
- The operating system interruptions, hardware or software failure, timer expiry, operation completion are the example of background events.

Events in JavaFX

The class named **Event** of the package **javafx.event** is the base class for an event.

- An instance of any of its subclass is an event. JavaFX provides a wide variety of events.

Mouse Event – This is an input event that occurs when a mouse is clicked. It is represented by the class named **MouseEvent**.

- actions like mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target.

Key Event – This is an input event that indicates the key stroke occurred on a node. It is represented by the class named **KeyEvent**.

- actions like key pressed, key released and key typed.

Window Event – This is an event related to window showing/hiding actions. It is represented by the class named **WindowEvent**.

- actions like window hiding, window shown, window hidden, window showing.

Event Handling

- the mechanism that controls the event and decides what should happen, if an event occurs.
  - This mechanism has the code which is known as an event handler that is executed when an event occurs.
- JavaFX provides handlers and filters to handle events. Every event has:
  - **Target** – The node on which an event occurred. A target can be a window, scene, and a node.
  - **Source** – The source from which the event is generated will be the source of the event. E.g., mouse can be the source of the event.
  - **Type** – Type of the occurred event; in case of mouse event – mouse pressed, mouse released are the type of events.

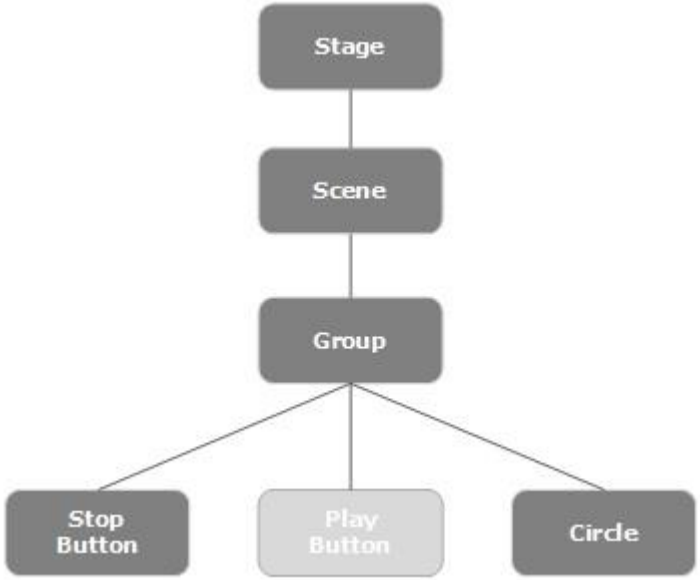
Event Handling

If you click on the play button, the source will be the mouse, the target node will be the play button and the type of the event generated is the mouse click.

Phases of Event Handling

- **Route Construction:** Whenever an event is generated, the default/initial route of the event is determined by construction of an **Event Dispatch chain**. It is the path from the stage to the source Node.
- **Event Capturing Phase:** After the construction of the event dispatch chain, the root node of the application dispatches the event. This event travels to all nodes in the dispatch chain (from top to bottom).

  If any of these nodes has a **filter** registered for the generated event, it will be executed.



Murdoch UNIVERSITY

Phases of Event Handling

Event Bubbling Phase

In the event bubbling phase, the event is travelled from the target node to the stage node (bottom to top). If any of the nodes in the event dispatch chain has a **handler** registered for the generated event, it will be executed.

Event Handlers and Filters

Event filters and handlers are those which contains application logic to process an event.

Filters:

- handles an event during the event capturing phase of event processing.
- A node can have one or more filters for handling an event.
- A single filter can be used for more than one node and more than one event type.
- filters enable the parent node to provide common processing for its child nodes or to intercept an event and prevent child nodes from acting on the event.

Phases of Event Handling

Handlers:

- Executes at the bubbling phase.
- EventFilter is executed before the EventHandler.
- Example: **DraggablePanelsExample**

Consuming of an Event

- Consuming the event in an event filter prevents any child node on the event dispatch chain from acting on the event.
- Consuming the event in an event handler stops any further processing of the event by parent handlers on the event dispatch chain.

Event Handling

To add an event filter to a node, you need to register this filter using the method **addEventFilter()** of the **Node** class.

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
   @Override
   public void handle(MouseEvent e) {
      System.out.println("Hello World");
      circle.setFill(Color.DARKSLATEBLUE);
   }
};
//Adding event Filter
Circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

To remove:

```
circle.removeEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

Murdoch UNIVERSITY

Event Handling

To add an event handler to a node, you need to register this handler using the method **addEventHandler()** of the **Node** class

```
//Creating the mouse event handler
EventHandler<javafx.scene.input.MouseEvent> eventHandler =
   new EventHandler<javafx.scene.input.MouseEvent>() {

   @Override
   public void handle(javafx.scene.input.MouseEvent e) {
      System.out.println("Hello World");
      circle.setFill(Color.DARKSLATEBLUE);
   }
};
//Adding the event handler
circle.addEventHandler(javafx.scene.input.MouseEvent.MOUSE_CLICKED, eventHandler);
```

To remove:

```
circle.removeEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

Murdoch UNIVERSITY

Using Convenience Methods for Event Handling

Some of the classes in JavaFX define event handler properties.

- By setting the values to these properties using their respective setter methods, you can register to an event handler.
- These methods are known as convenience methods.
- Most of these methods exist in the classes like Node, Scene, Window, etc., and they are available to all their sub classes.

```
playButton.setOnMouseClicked((new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
        System.out.println("Hello World");
        pathTransition.play();
    }
}));
```

Murdoch UNIVERSITY

Inputting and Outputting Numbers

- To get or input an `int` from a `TextField` or `TextArea`,

```
int n =
        Integer.parseInt(mytext.getText().trim());
```

  i.e., get a String using `getText`, trim leading and trailing spaces using `trim`, then convert the String to an `int` using `parseInt`.

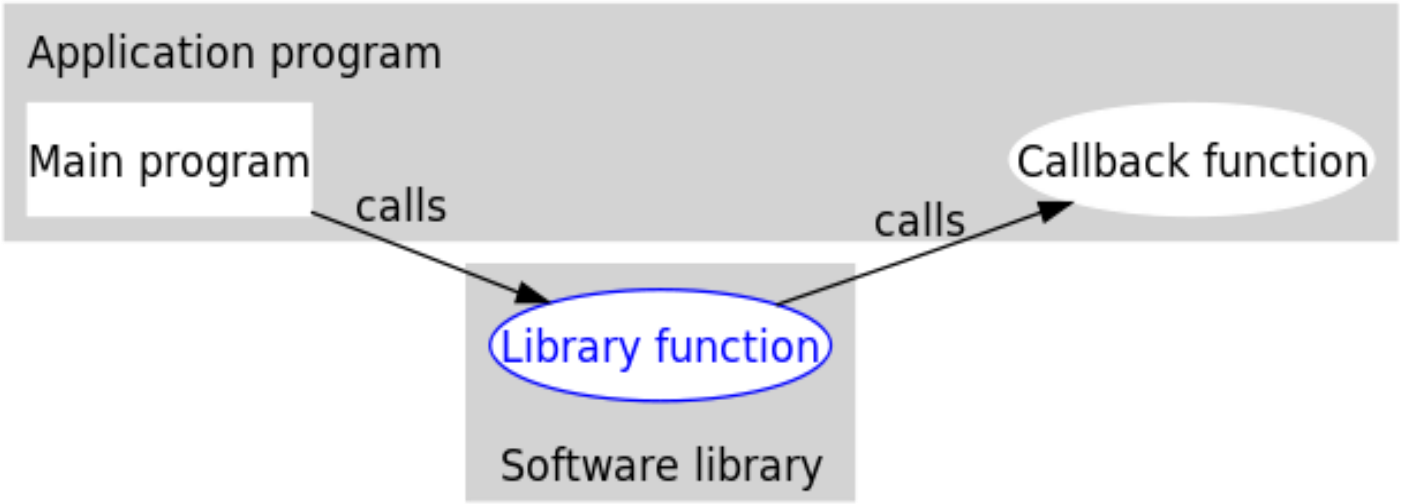
- To put an `int` into a `TextField` or `TextArea`, use:

```
mytext.setText(Integer.toString(n));
```

- Similarly, for other data types (float, double etc).

Callback function

Callbacks refer to a mechanism in which a library or utility class provides a service to clients that are unknown to it when it is defined.



Murdoch UNIVERSITY

Dialog Boxes

- = windows which pop up out of other windows (to deal with specific issues without cluttering the original window).
- These are normally used as a temporary window to receive additional information from the user or to provide notification that some event has occurred.
- javafx.scene.control.Dialog
- javafx.scene.control.Alert
- javafx.scene.control.TextInputDialog
- javafx.scene.control.ChoiceDialog

Using NetBeans tools

Visual Programming and Java Beans

- Java supports a Visual Programming Modular Assembly design style. This is a particular approach to GUI design which enables reuse and was popularized in Microsoft's VB and Borland's Delphi. The reusable objects are visual components such as buttons, menus, etc.
- The idea is for the GUI designer to run an "application builder tool" (eg, BDK or Bean Builder, also NetBeans and commercial versions of jBuilder, SceneBuilder).
- A selection of visual components (called Beans in Java) are made available on a palette and these can be chosen, dragged and dropped at desired locations on a form. A particular bean will have some properties (like colour) and some events (like Action). The designer configures the initial value of the properties. The application builder tool writes/creates code as you do this and that code will cause the component to be created in the running program.

Visual Programming and Java Beans

- Serialization allows the configured bean to be stored ready to appear in the right way when the form is used.
- Beans will have EventHandlers() which can be used by the rest of the code to allow things to happen in response to bean events.
- A combination of a standard method naming convention and reflection allow application builder tools to use a Java Bean got from anywhere.

Designing a GUI

- One of the best design decisions you can make when using a GUI is to separate the GUI code from the "business logic" i.e. the code that does the real work behind the GUI.
- Plan to allow the business code to be (re)used separately from the GUI and even for the GUI to be reused in other situations.
- The GUI components should use a public method of a business object to hand over all business domain computations to business objects.
- Eg, an event handler method (for a Button) should just hand over the straight text String from a TextField to one business public method for processing. The business method might compose an SQL query, send it across the internet, get an answer and format it for display.
- The eventhandler method can just put the formatted response in another textField. You are not helping with reuse if the eventhandler method does all these separate tasks by itself.

Murdoch
UNIVERSITY

JavaFX TreeView

Subtitle if required

Mu
UNIV

TreeView

- Builds tree structures in your JavaFX application,
- Add items to the tree views, process events, and customize the tree cells by implementing and applying cell factories.

- The TreeView class of the javafx.scene.control package provides a view of hierarchical structures.
- In each tree the highest object in the hierarchy is called the "root."
- The root contains several child items, which can have children as well.
- An item without children is called "leaf”.

Murdoch UNIVERSITY

Tree View

- typically need to instantiate the TreeView class,
- define several TreeItem objects,
- make one of the tree items the root,
- add the root to the tree view and other tree items to the root.
- Useful methods: getChildren(), add(), and addAll().
- More: https://docs.oracle.com/javafx/2/ui_controls/tree-view.htm

An example

Steps: http://tutorials.jenkov.com/javafx/treeview.html

- Create a TreeView
- Add TreeView to Scene Graph
- Add Tree Items to TreeView
- Add Children to a TreeItem

- JavaFXTreeViewexample

Murdoch
UNIVERSITY

Summary

Subtitle if required

Summary

- This topic looked at the design and implementation of Graphical User Interfaces (GUI).
- The behaviour of a GUI is determined by its current state and by the events, like a mouse click, that can occur. The states, events, and response actions can be modelled as a Finite State Machine (FSM) to describe the dynamic behaviour of the GUI.
- Events can be handled according to different event models which need to be understood by the designer.

Murdoch

UNIVERSITY