X01 JavaFX 1

Introduction to JavaFX

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

JavaFX

- Designed for rich client applications
  - Graphics, UI’s, video, audio, etc.
- Replaces Swing, AWT
- JavaFX HelloWorld

JavaFX

Extend `javafx.application.Application`

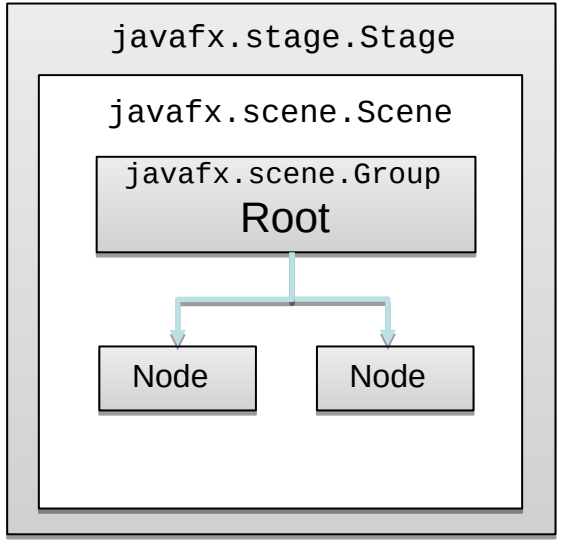
Override the `start()` method

`Stage`: the window

`Scene`: container for a scene graph

`Node`: object or group of objects in scene

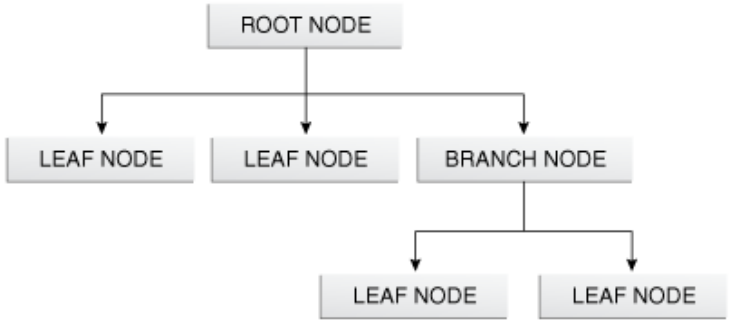
`Pane`: organizer of nodes in scene graph: `FlowPane`, `BorderPane`, `GridPane`, `HBox`, `VBox`, etc.

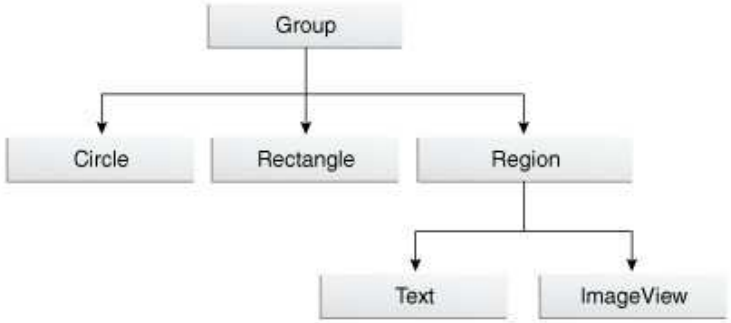


Java FX Scene Graph

Tree of nodes, with a single ‘branch’ at the root

- **branch** (may have children e.g. `Group`, `Region`)
- **leaf** (may not have children e.g. `Rectangle`, `Circle`)





Nodes and Properties

Can set node properties programmatically:

```
Text message = new Text("Hello");
message.setFont(Font.font("Tahoma", FontWeight.NORMAL, 40);
message.setFill(Color.RED);
```

or declaratively using FXML / CSS:

```
#text {
  -fx-font-family: Tahoma, sans-serif;
  -fx-font-style: normal;
  -fx-font-size: 40;
  -fx-fill: red;
}
```

X02 JavaFX 2

JavaFX and Event Handling

Event Handling

Event handling is another control flow construct.

- Branches (a conditional or switch selects control flow)
- Loops (a loop repeats control flow)
- Methods (a method call nests control flow)
- Events (the occurrence of event changes control flow)
  - Event handling in UIs
  - Exception handling (later)

Events and Passing Code in Java

An event handler executes some code when an event occurs.

Q: How do we pass code as an argument in Java?

A: We pass *objects*, which implement *interfaces* through *methods*.

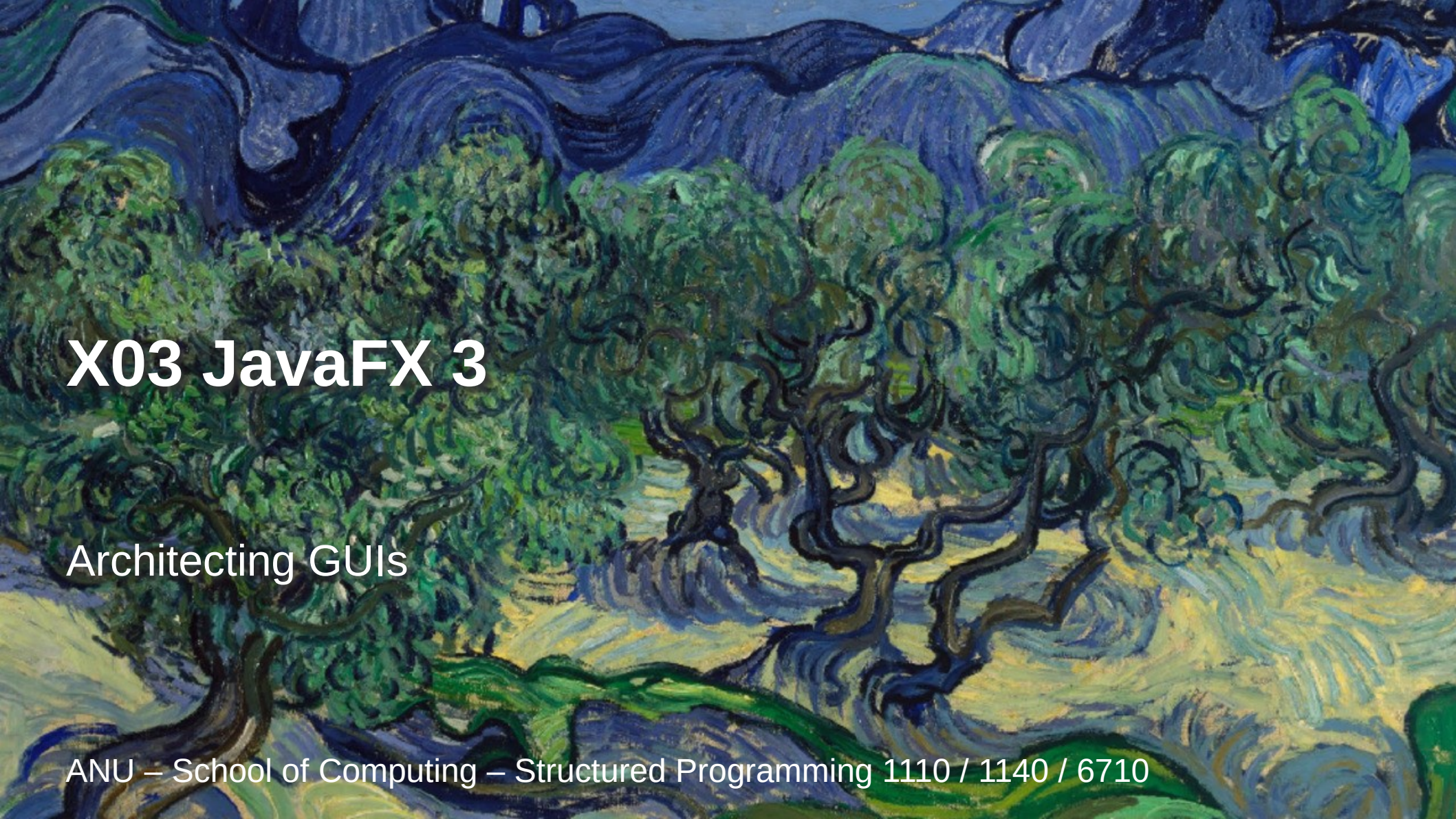
JavaFX event handler interfaces are functional, so we can use lambda expressions.

Events in JavaFX

- Events are instances of `javafx.event.Event`
  - Have Event type, Source, Target
- Event handlers implement `javafx.event.EventHandler`
  - Functional interface, with method `void handle(Event)`.
- An event handler can be created with a lambda expression, for example:

  ```
  scene.setOnKeyTyped(event -> { ...code... })
  ```

  (or a nested class, an anonymous inner class, ...)



X03 JavaFX 3

Architecting GUIs

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

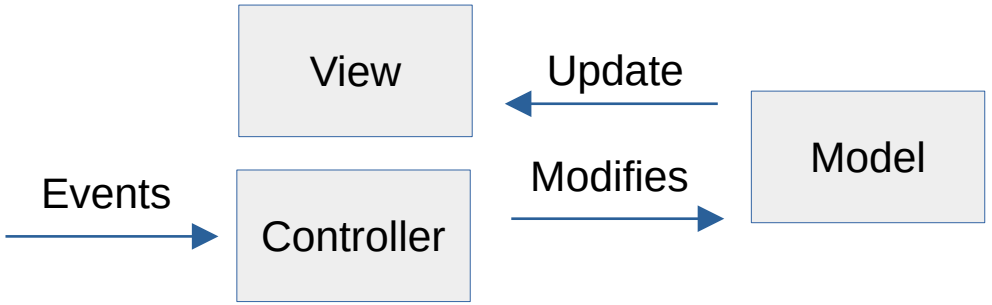
GUI Design Principles

- Separation of **data** and **domain logic** from its **presentation**.
  - Enable multiple ways to display the same data, either multiple GUIs can be developed or multiple views of the same thing within the same GUI.
  - Make it easier / possible to test the domain logic.
  - Separation of responsibilities, source of truth of state.

GUI Architectural Patterns

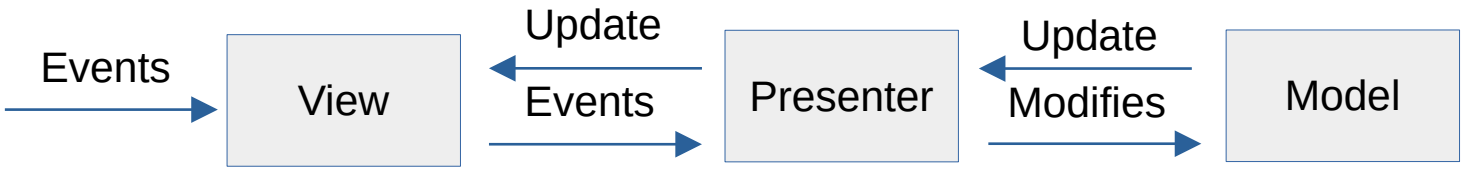
Model-View-Controller (MVC)

Separation into View and Controller



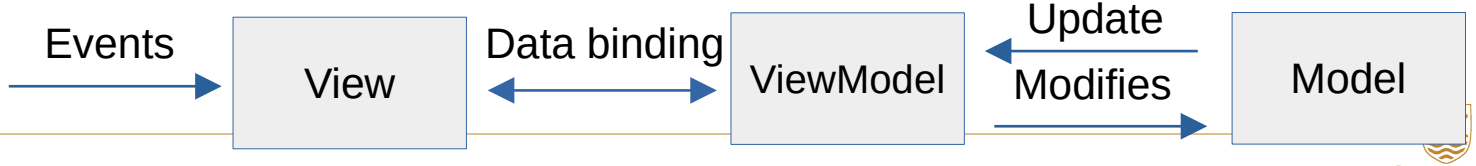
Model-View-Presenter (MVP)

Presenter is middle-man



Model-View-ViewModel (MVVM)

Leverage “data binding” to simplify event handling



Recommendations

- Separate GUI from game logic and state, **but keep it simple** (no need to try these architectural patterns).
- Don’t let the GUI and Model get out of sync, and always check with the model.
  - Event -> Modify model -> Update view based on model
- Purely GUI considerations should be kept out of the model (e.g., position of object being dragged).