Event-driven programming and GUI input
JavaFX GUI

- Output: what’s drawn on the screen
  - Nodes
    - E.g., Button, buttons, labels, lists, sliders, canvas
  - Parent nodes: contain other nodes, control layout
    - Pane, HBox, VBox, GridPane, StackPane, Group...

- Helper classes
  - E.g., Graphics, Color, Font, FontMetrics, Dimension

- Input: handling user interaction
  - Events
    - E.g., button-press, mouse-click, key-press
    - EventHandlers: an object that responds to an event
  - Properties
    - Listeners
    - Animation

http://docs.oracle.com/javase/8/javase-clienttechnologies.htm
UI Builder tool


- The JavaFX Scene Builder makes XML representations of UI node layouts.
  - Example: simple.fxml

```xml
<?xml version="1.0" encoding="UTF-8"?>
...
<AnchorPane id="AnchorPane" prefHeight="299.0" prefWidth="319.0" xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/2.2">
  <children>
    <HBox layoutX="0.0" layoutY="0.0" prefHeight="299.0" prefWidth="333.0">
      <children>
        <Button id="pressme" mnemonicParsing="false" text="Press me!" />
        <TextArea id="typeme" prefHeight="299.0" prefWidth="236.0" text="Type more text here." wrapText="true" />
      </children>
    </HBox>
  </children>
</AnchorPane>
```

- Can read XML into UI nodes with FXMLLoader.load(url)
Events

http://docs.oracle.com/javase/8/javafx/events-tutorial/events.htm#JFXED117

• GUI code responds to (and creates) events
  • E.g., mouse button, keyboard pressed, mouse motion, window exposed, …
  • All subclasses of javafx.ui.Event

• Some nodes already handle events on their own, generate new events, e.g.:
  ▪ Buttons: mouse press, release → ‘button clicked’
  ▪ Scrollbar: mouse clicks, motion → scrollbar value
  ▪ Multiple press/release events → ‘double-click’

• Application defines how to handle both ‘raw’ and synthesized events, can generate its own events.
Event handlers

• An Event<Handler<T> is an object that handles events of type T:

```java
interface Event<Handler<T> {
    void handle(T event);
}
```

• Event handlers can be registered with nodes that generate events:

```java
Button b = new Button("press me");
b.setOnAction(myButtonHandler);
Scrollbar s = new Scrollbar();
s.setOnScroll(myScrollEventHandler);
```

Note: there can be only one.
A brief example

class PrintIt implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent _) {
        System.out.println("Button was clicked");
    }
}

class Main extends Application {
    public void start(Stage stage) throws Exception {
        try {
            URL r = getClass().getResource("simple.fxml");
            if (r == null) ... // error
            Scene scene = new Scene(FXMLLoader.load(r));
            stage.setScene(scene);
            stage.sizeToScene();
            Button b = (Button) scene.lookup("#pressme");
            b.setOnAction(new PrintIt());
            stage.show();
        } catch ... // error
    }
}
Event types

- Different kinds of events represented by different event classes
- E.g., MouseEvent reports mouse position
Delegation Model

• Timeline for an event
  ▪ User (or program) does something to a component, event is generated.
  ▪ Event is passed down event dispatch chain to find handlers for event
    • Event dispatch chain determined by the event target the event is sent to (e.g., the window = Stage).
    • Event dispatch chain usually corresponds to chain of nodes in layout tree from root to leaf—can be overridden, but usually not necessary.
  ▪ Each event handler uses event to update application state appropriately.
    • handler can modify, consume event (so not seen by rest of chain), generate new events.
Accessing state from handler

class PrintIt implements EventHandler<ActionEvent> {
    Main main;
    PrintIt(Main m) { main = m; }
    @Override
    public void handle(ActionEvent _) {
        System.out.println(main.message);
    }
}

public class Main extends Application {
    String message = "Button was clicked";
    public void start(Stage stage) throws Exception {
        ...
        Button b = (Button) scene.lookup("#pressme");
        b.setOnAction(new PrintIt(this));
        ...
    }
}
public class Main extends Application
    implements EventHandler<ActionEvent> {
    String message = "Button was clicked";
    public void start(Stage stage) throws Exception {
        ...  
        Button b = (Button) scene.lookup("#pressme");
        b.setOnAction(this);
        ...
    }
    public void handle(ActionEvent _) {
        System.out.println(message);
    }
}
Event handler as inner class

```java
public class Main extends Application {
    String message = "Button was clicked";
    public void start(Stage stage) throws Exception {
        ...
        Button b = (Button) scene.lookup("#pressme");
        b.setOnAction(new PrintIt());
        ...
    }

    class PrintIt implements EventHandler<ActionEvent> {
        public void handle(ActionEvent _) {
            System.out.println(message);
        }
    }
}
```
…as anonymous inner class

```java
public class Main extends Application {
    String message = "Button was clicked";
    public void start(Stage stage) throws Exception {
        ...
        Button b = (Button) scene.lookup("#pressme");
        b.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent _) {
                System.out.println(message);
            }
        });
    }
}
```
Properties

- Another way to access dynamic behavior in JavaFX: node properties
- Node accessors correspond to property objects:

```java
boolean isDisabled()    BooleanProperty disabledProperty()
double getWidth(),      ReadOnlyDoubleProperty
    getHeight()       widthProperty(), heightProperty()
double getLayoutX(),    DoubleProperty layoutXProperty(),
    getLayoutY()      layoutYProperty()
Paint getTextFill()     ObjectProperty<Paint> textFillProperty()
String getText()        StringProperty getTextProperty()
```
Listening to properties

- Program actions can be triggered by changes to properties, by attaching listeners.

```java
TextArea t = …;
t.textProperty().addListener(new ChangeListener<String>() {
    @Override
    public void changed(ObservableValue<? extends String> obs,
                         String before, String after) {
        System.out.println("Changed from \\
         \"+ before + "\" to \"+ after + "\\n" + after + "\")
    }
});
```

- Any number of listeners can be attached
- Design pattern: Observer
Animations

http://docs.oracle.com/javafx/2/animations/jfxpub-animations.htm

- Properties can be controlled by animations
- Animation is defined by a sequence of **key frames**
- Each key frame has a time $T$ and defines the values of some set of properties
- JavaFX interpolates properties smoothly between key frames.
Creating an animation

• “Over the next 0.5 seconds, increase the Y position of the button by 10 pixels”

Timeline tl = new Timeline();
    tl.getKeyFrames().add(new KeyFrame(
        Duration.millis(500), "done",
        new KeyValue(b.layoutYProperty(), b.getLayoutY() + 10.0)));

tl.play();
Binding properties

• Properties can be bound to computations rather than to values.

Button b1 = new Button();
Button b2 = new Button();
DoubleProperty p = b2.getLayoutY();
p.bind(b1.layoutYProperty().add(
    new SimpleDoubleProperty(10.0)));

• Effect: b2’s Y position is recomputed and updated automatically as b1’s Y position changes.

http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm