CSE 215: Programming Language II Lab
Lab 17
Focusing on: Event Driven Programming, Lambda Expressions

In the last lab we saw the designing aspect of JavaFX. However, the UI elements we placed on to the stage did nothing reasonable when we performed any action, such as clicking a button. Today we will be covering on how to perform actions based on an event.

Event: An event is an object created from an event source. Firing an event means to create an event and delegate the handler to handle the event.

Whenever user performs an action, such as clicking a button, moving the mouse and pressing a key on the keyboard, an event occurs. The table below lists the user action, corresponding event type and what type of handler to use to handle the event.

TABLE 15.1 User Action, Source Object, Event Type, Handler Interface, and Handler

User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press Enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`RadioButton`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`CheckBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Mouse entered			`setOnMouseEntered(EventHandler<MouseEvent>)`
Mouse exited			`setOnMouseExited(EventHandler<MouseEvent>)`
Mouse moved			`setOnMouseMoved(EventHandler<MouseEvent>)`
Mouse dragged			`setOnMouseDragged(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

You must register a handler for the appropriate types of events in order to perform something via code when an event occurs.

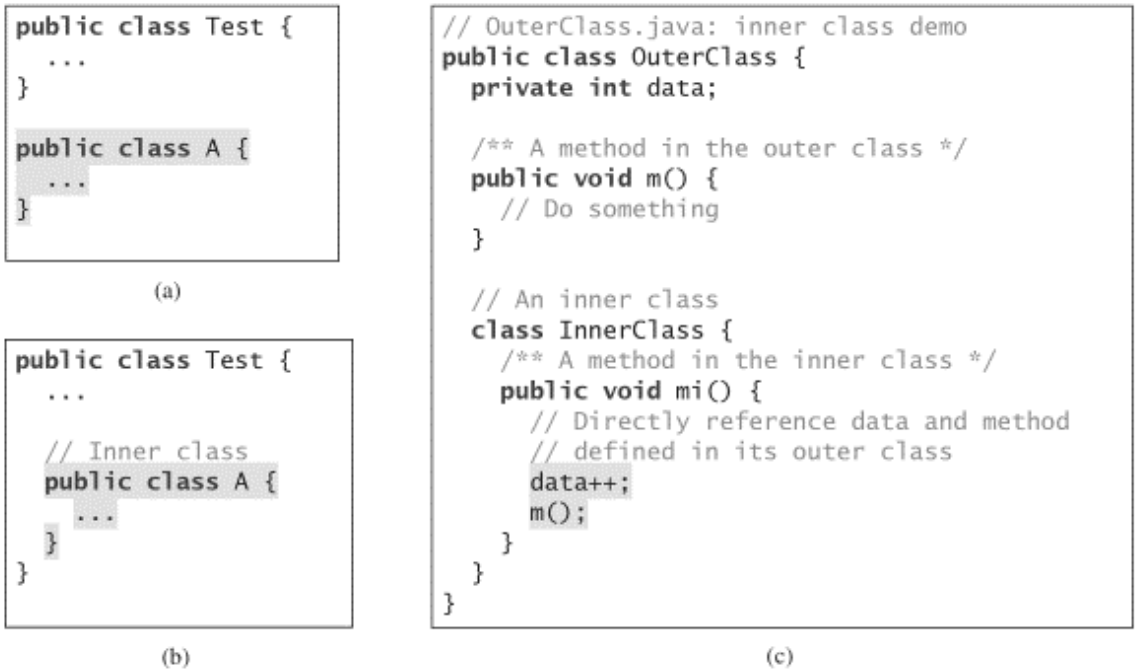
Before that, you need to know about two concepts.

A. **Inner class**
B. **Anonymous inner class**

A. Inner Class

An inner class, or nested class, is a class defined within the scope of another class. Inner classes are useful for defining handler classes. Inner classes combine dependent classes into the primary class.

You may only need a class inside another class. In that case, it would be redundant to declare the second class separately where every other class may access it.



B. Anonymous inner class

An anonymous inner class is a way of declaring an inner class without giving it a name. An anonymous inner class can extend a class or implement an interface. An example of it would be to create a thread like this, without extending the Thread class in the MyThread class:

```
class MyThread {
  public static void main(String[] args) {
    // define an anonymous thread and start it
    (new Thread() {
       public void run() {
         System.out.println("Child thread");
       }
    }).start();

    System.out.println("Main thread");
  }
}
```

You can use both of these approaches when designing event handlers. You can define either inner classes that implement EventHandler interface or you can also use anonymous inner classes.

Example:

Write a JavaFX program with a label and another node such that when the mouse cursor enters within the given node or the node is clicked, the x and y coordinates of the cursor is displayed in the label. Relative and absolute scene and screen coordinates must be displayed.

A. Using Inner classes

```
1  public class DynamicLabel extends Application {
2    private Label xyCoordinateLabel  = new Label();
3    @Override
4    public void start(Stage primaryStage) {
5      //set header
6      Label messageLabel = new Label("Move the cursor inside the text box to get X Y
7  coordinates. \nClick to get a surprise.");
8      messageLabel.setPadding(new Insets(10,10,10,10));
9      Font labelFont = Font.font("Helvetica",FontWeight.BOLD, 22f);
10     messageLabel.setFont(labelFont);
11
12     //set label for XY coordinates
13     Font xyCoordinateLabelFont = Font.font("Helvetica", FontPosture.REGULAR, 20f);
14     this.xyCoordinateLabel.setFont(xyCoordinateLabelFont);
15     this.xyCoordinateLabel.setStyle("-fx-text-fill: red;");
16
17     BorderPane parentLayout = new BorderPane();
18
19     VBox vericalLabelsLayout = new VBox(16);
20     vericalLabelsLayout.setPadding(new Insets(20,20,20,20));
21     vericalLabelsLayout.setAlignment(Pos.CENTER);
22     vericalLabelsLayout.getChildren().addAll(messageLabel,
23 this.xyCoordinateLabel);
24     parentLayout.setTop(vericalLabelsLayout);
25
26     TextField textField = new TextField();
27     textField.setMinSize(300, 300);
28     textField.setMinSize(300, 300);
29     textField.setPadding(new Insets(20,0,20,30));
30     VBox textFieldLayout = new VBox(16);
31     textFieldLayout.getChildren().add(textField);
32     textFieldLayout.setAlignment(Pos.CENTER);
33
34     textField.setOnMouseMoved(new MouseMoveHandler());
35     textField.setOnMouseClicked(new MouseClickHandler());
36     parentLayout.setCenter(textFieldLayout);
37
38     parentLayout.setPadding(new Insets(30, 30, 30, 30));
39
40     Scene primaryScene = new Scene(parentLayout, 800, 550);
41     primaryStage.setTitle("Cursor Movement with Anonymous Inner Class");
42     primaryStage.setScene(primaryScene);
43     primaryStage.show();
44   }
45
46   class MouseMoveHandler implements EventHandler<MouseEvent> {
47     @Override
48     public void handle(MouseEvent event) {
49       xyCoordinateLabel.setText("Relative  X  coordinate:  "+  event.getX()  +
50 "\t\t\tRelative Y coordinate: " + event.getY() + "\n" +
51       "Absolute Scene X coordinate: "+ event.getSceneX() + "\tAbsolute Scene Y
52 coordinate: " + event.getSceneY() + "\n");
53     }
```

```
54      }
55
56      class MouseClickHandler implements EventHandler<MouseEvent> {
57        @Override
58        public void handle(MouseEvent event) {
59          xyCoordinateLabel.setText("Absolute     Screen     X     coordinate:     "+
60    event.getScreenX() + "\tAbsolute Screen Y coordinate: " + event.getScreenY() +
61    "\n");
62        }
63      }
64
65      /**
66       * @param args the command line arguments
67       */
68      public static void main(String[] args) {
69          launch(args);
70      }
71    }
```

When using inner classes, we need to have the dynamic nodes as class attributes, otherwise the inner classes won't be able to find them.

B. Using anonymous inner classes

The given code is really…ugly. Let's fix that.

```
 1  public class DynamicLabel extends Application {
 2      @Override
 3      public void start(Stage primaryStage) {
 4        //set header
 5        Label messageLabel = new Label("Move the cursor inside the text box to get X
 6  Y coordinates. \nClick to get a surprise.");
 7        messageLabel.setPadding(new Insets(10,10,10,10));
 8        Font messageLabelFont = Font.font("Helvetica",FontWeight.BOLD, 22f);
 9        messageLabel.setFont(messageLabelFont);
10
11        //set label for XY coordinates
12        // move the label inside the start method
13        Label xyCoordinateLabel  = new Label();
14        Font  xyCoordinateLabelFont  =  Font.font("Helvetica",  FontPosture.REGULAR,
15  20f);
16        xyCoordinateLabel.setFont(xyCoordinateLabelFont);
17        xyCoordinateLabel.setStyle("-fx-text-fill: red;");
18
19        BorderPane parentLayout = new BorderPane();
20
21        VBox verticalLabelsLayout = new VBox(16);
22        verticalLabelsLayout.setPadding(new Insets(20,20,20,20));
23        verticalLabelsLayout.setAlignment(Pos.CENTER);
24        verticalLabelsLayout.getChildren().addAll(messageLabel, xyCoordinateLabel);
25        parentLayout.setTop(verticalLabelsLayout);
26
27        TextField textField = new TextField();
28        textField.setMinSize(300, 300);
29        textField.setMinSize(300, 300);
30        textField.setPadding(new Insets(20,0,20,30));
31        VBox textFieldLayout = new VBox(16);
32        textFieldLayout.getChildren().add(textField);
33        textFieldLayout.setAlignment(Pos.CENTER);
34
35        // anonymous inner class for mouse movement handler
```

```
36          textField.setOnMouseMoved(new EventHandler<MouseEvent>() {
37            @Override
38            public void handle(MouseEvent event) {
39              xyCoordinateLabel.setText("Relative  X  coordinate:  "+  event.getX()  +
40    "\t\t\tRelative Y coordinate: " + event.getY() + "\n" +
41              "Absolute Scene X coordinate: "+ event.getSceneX() + "\tAbsolute Scene Y
42    coordinate: " + event.getSceneY() + "\n");
43            }
44          });
45
46          // anonymous inner class for mouse click handler
47          textField.setOnMouseClicked(new EventHandler<MouseEvent>() {
48            @Override
49            public void handle(MouseEvent event) {
50              xyCoordinateLabel.setText("Absolute    Screen    X    coordinate:    "+
51    event.getScreenX() + "\tAbsolute Screen Y coordinate: " + event.getScreenY() +
52    "\n");
53            }
54          });
55
56          parentLayout.setCenter(textFieldLayout);
57
58          parentLayout.setPadding(new Insets(30, 30, 30, 30));
59
60          Scene primaryScene = new Scene(parentLayout, 800, 550);
61          primaryStage.setTitle("Cursor Movement with Anonymous Inner Class");
62          primaryStage.setScene(primaryScene);
63          primaryStage.show();
64        }
65
66        /**
67         * @param args the command line arguments
68         */
69        public static void main(String[] args) {
70            launch(args);
71        }
72      }
```

The changes are in lines 12-13 and 34-53. We got rid of the inner classes and replaced them with anonymous inner classes, which simplifies the code.

Using Lambda Expressions:

Lambda expressions are those expressions that implement a functional interface. A functional interface is an interface that only has one method defined in it. The `EventHandler` interface is a functional interface, with only one method `handle(Event e)` inside it. Lambda expressions can further simplify the above code.

Simply replace lines 34-53 above with the following:

```
33    textField.setOnMouseMoved(event ->
34            xyCoordinateLabel.setText("Relative  X  coordinate:  "+  event.getX()  +
35    "\t\t\tRelative Y coordinate: " + event.getY() + "\n" +
36            "Absolute Scene X coordinate: "+ event.getSceneX() + "\tAbsolute Scene Y
37    coordinate: " + event.getSceneY() + "\n");
38        );
39
40        textField.setOnMouseClicked(event ->
```

```
41          xyCoordinateLabel.setText("Absolute     Screen     X     coordinate:     "+
42  event.getScreenX() + "\tAbsolute Screen Y coordinate: " + event.getScreenY() +
43  "\n");
44      );
```

We end up with much cleaner code by using lambda expressions. Also, we no longer need to specify the exact event type e.g EventHandler<MouseEvent>. It is automatically deduced at runtime.

Home Assignment

1. Write a program that lets the user click on a pane to dynamically create and remove points. When the user left-clicks the mouse (primary button), a point is created and displayed at the mouse point. The user can remove a point by pointing to it and right- clicking the mouse (secondary button).

[Hint: You need to find a way to find out whether the user clicked left or right mouse button. Consider using javafx.scene.input.MouseButton]