CSE 215: Programming Language II Lab
Lab 19
Focusing on: Multithreaded JavaFX

In today's lab, we will be creating a simple BMI calculator that involves multiple windows or stages. In any GUI application, a particular window or stage can either be:

A. Blocking: The window blocks access to all other windows until it is closed. This is also known as a modal window.

B. Non-blocking: This type of window usually runs on a separate thread and does not block access to other windows. This type of windows is also known as non-modal windows.

For today's application, we will have four .java files:

BmiInputLayout.java: This class will only create the layout for the first window/stage of the BMI calculator app. It will set up all the buttons, labels and text fields with appropriate height, width and text.

BmiOutputLayout.java: The calculated BMI will be displayed in a separate window to demonstrate the technique of displaying a non-blocking window using threads.

BmiCalculator.java: This class will contain the business logic that actually calculates the BMI from the given height and weight.

BmiApp.java: This class will be the subclass of `javafx.application.Application` and the main JavaFX UI thread will run from here. It will also have a blocking error window to demonstrate the pros and cons of a blocking window/stage.

Before we proceed, we must introduce two new concepts:

a) Thread Pool

b) JavaFX Concurrency

a) Thread Pool: It is inefficient to create multiple threads for different tasks manually if you do not have adequate CPU cores. For example, if you have only 2 available CPU cores and you create 100 threads, this may actually cause slower performance than without threads. The better option would be to create a fixed number of threads and then use and reuse those threads for various tasks, when needed. Thread pools contain a defined number of worker threads that are assigned tasks when they become available. In Java, the `java.util.concurrent.ExecutorService` interface and `java.util.concurrent.Executors` class are used together to achieve this.

```
  public static void main(String[] args) {
    // Create a fixed thread pool with maximum three threads
    ExecutorService executor = Executors.newFixedThreadPool(3);

    // Submit runnable tasks to the executor
    executor.execute(new PrintChar('a', 100));
    executor.execute(new PrintChar('b', 100));
    executor.execute(new PrintNum(100));

    // Shut down the executor
    executor.shutdown();
  }
}
```

Fig: Creating a thread pool and executing separate methods using it

b) **JavaFX Concurrency:** In JavaFX, the GUI application runs on a single thread. Any changes to the UI in any window, e.g. updating a Label or changing the color of the text must be done on the main thread. As a result, it is not straightforward to setup a multithreaded JavaFX application. The prerequisites for doing so are:

i. `javafx.application.Platform` class: This class has a static method called `runLater()`, which takes in a Runnable task. Any UI updates that are to be done in a separate thread are defined within this Runnable task. Ideally, this is within a separate method inside the main JavaFX app class.

ii. Threading: Using thread pools is the best way to achieve multithreading, because the usual ways of implementing the Runnable interface or extending the Thread class won't work here. Instead, we need to use anonymous Runnable class and thread pool in conjunction.

The BMI App

We intend our BMI app to look like this.

Figure: BMI Input Layout

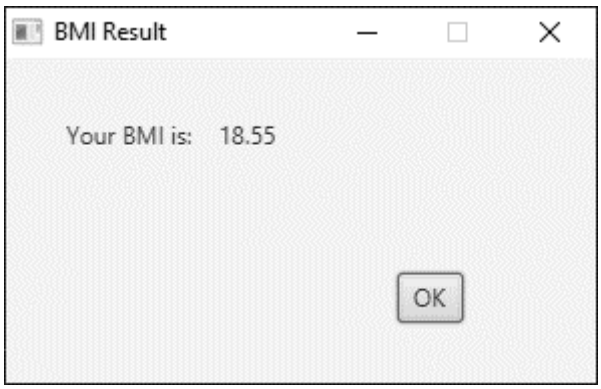


Figure: BMI Output Layout (non-blocking)

Figure: Error Window (blocking)

For simplicity, we have broken down our code into several files, otherwise it would have been a gigantic file with a lot of code.

BmiInputLayout.java

```
1   public class BmiInputLayout {
2       private Pane bmiInputLayout;
3
4       public BmiInputLayout() {
5           Button calculateBMIButton = new Button("Calculate BMI");
6           calculateBMIButton.setId("calculate-bmi-btn");
7
8           Button exitButton = new Button("Exit");
9           exitButton.setId("exit-btn");
10
11          Label weightTextFieldLabel = new Label("Your Weight (Kilograms): ");
12          Label heightTextFieldLabel = new Label("Your Height (Feet): ");
13
14          TextField weightTextField = new TextField();
15          weightTextField.setId("weight-text-field");
16
17          TextField heightTextField = new TextField();
```

```
18          heightTextField.setId("height-text-field");
19
20          // set up positioning of the UI components
21          HBox weightTextAndLabelLayout = new HBox();
22          weightTextAndLabelLayout.setSpacing(20);
23          weightTextAndLabelLayout.getChildren().addAll(weightTextFieldLabel,
24  weightTextField);
25
26          HBox heightTextAndLabelLayout = new HBox();
27          heightTextAndLabelLayout.setSpacing(50);
28          heightTextAndLabelLayout.getChildren().addAll(heightTextFieldLabel,
29  heightTextField);
30
31          HBox buttonLayout = new HBox();
32          buttonLayout.setSpacing(30);
33          buttonLayout.getChildren().addAll(calculateBMIButton, exitButton);
34
35          // parent pane
36          GridPane parentLayout = new GridPane();
37          parentLayout.setPadding(new Insets(20, 30, 20, 30));
38          parentLayout.setHgap(20);
39          parentLayout.setVgap(20);
40
41          ColumnConstraints gridLayoutColumn1 = new ColumnConstraints();
42          gridLayoutColumn1.setPercentWidth(60);
43          ColumnConstraints gridLayoutColumn2 = new ColumnConstraints();
44          gridLayoutColumn2.setPercentWidth(40);
45
46          parentLayout.getColumnConstraints().addAll(gridLayoutColumn1,
47  gridLayoutColumn2);
48
49          RowConstraints gridLayoutRow1 = new RowConstraints();
50          gridLayoutRow1.setPercentHeight(75.0 / 2);
51          RowConstraints gridLayoutRow2 = new RowConstraints();
52          gridLayoutRow2.setPercentHeight(75.0 / 2);
53          RowConstraints gridLayoutRow3 = new RowConstraints();
54          gridLayoutRow3.setPercentHeight(25);
55
56          parentLayout.getRowConstraints().addAll(gridLayoutRow1, gridLayoutRow2,
57  gridLayoutRow3);
58
59          parentLayout.add(weightTextAndLabelLayout, 0, 0);
60          parentLayout.add(heightTextAndLabelLayout, 0, 1);
61          parentLayout.add(buttonLayout, 1, 2);
62
63          this.bmiInputLayout = parentLayout;
64      }
65
66      public Pane getBmiInputLayout() {
67          return this.bmiInputLayout;
68      }
69
70  }
```

BmiOutputLayout.java

```
1   public class BmiOutputLayout {
2       private Pane bmiOutputLayout;
3
4       public BmiOutputLayout() {
5           Button okButton = new Button("OK");
6           okButton.setId("ok-btn");
7
8           Label bmiInfoLabel = new Label("Your BMI is: ");
9           Label bmiResultLabel = new Label();
10          bmiResultLabel.setId("bmi-result");
11
12          HBox bmiInfoandResultLayout = new HBox();
13          bmiInfoandResultLayout.setSpacing(10);
14          bmiInfoandResultLayout.getChildren().addAll(bmiInfoLabel, bmiResultLabel);
15
16          // parent pane
17          GridPane parentLayout = new GridPane();
18          parentLayout.setPadding(new Insets(10, 30, 20, 30));
19          parentLayout.setHgap(20);
20          parentLayout.setVgap(20);
21
22          parentLayout.add(bmiInfoandResultLayout, 0, 1);
23          parentLayout.add(okButton, 3, 4);
24
25          this.bmiOutputLayout = parentLayout;
26      }
27
28      public Pane getBmiOutputLayout() {
29          return this.bmiOutputLayout;
30      }
31  }
```

BmiCalculator.java

```
1   public class BmiCalculator {
2       private static final double FEET_TO_METRES_CONVERSION_FACTOR = 3.281;
3
4       BmiCalculator() {}
5
6       private static double feetToMetres(double heightInFeet) {
7           return heightInFeet / FEET_TO_METRES_CONVERSION_FACTOR;
8       }
9
10      public static double calculateBMI(double weight, double height) {
11          if (height <= 0 || weight <= 0) {
12              throw new IllegalArgumentException("Height or weight must be a
13  positive number!");
14          }
15          return weight / Math.pow(feetToMetres(height), 2);
16      }
17  }
```

BmiApp.java

```
1   public class BmiApp extends Application {
2     private Stage bmiOutputStage = null;
3     private ExecutorService threadExecutor = Executors.newFixedThreadPool(2);
4
5     @Override
6     public void start(Stage primaryStage) {
7       Pane bmiInputLayout = new BmiInputLayout().getBmiInputLayout();
8
9       Scene primaryScene = new Scene(bmiInputLayout);
10      primaryStage.setTitle("BMI Calculator");
11      primaryStage.setScene(primaryScene);
12      primaryStage.setMinHeight(200);
13      primaryStage.setMinWidth(700);
14      primaryStage.show();
15
16      ((Button) bmiInputLayout.lookup("#calculate-bmi-btn")).setOnAction(e -> {
17          Runnable processBMIRunnableTask = new Runnable() {
18            @Override
19            public void run() {
20              processBMI(bmiInputLayout);
21            }
22          };
23
24          this.threadExecutor.execute(processBMIRunnableTask);
25      });
26
27      ((Button) bmiInputLayout.lookup("#exit-btn")).setOnAction(event -> {
28        primaryStage.close();
29      });
30
31      primaryStage.setOnHiding(e -> this.threadExecutor.shutdown());
32    }
33
34    public static void main(String[] args) {
35      Application.launch(args);
36    }
37
38    public void showErrorWindow(Pane parentPane, String errorText) {
39      Button okButton = new Button("OK");
40      Label errorLabel = new Label(errorText);
41
42      VBox errorWindowLayout = new VBox();
43      errorWindowLayout.setAlignment(Pos.CENTER);
44      errorWindowLayout.setSpacing(20);
45      errorWindowLayout.setPadding(new Insets(20, 20, 20, 20));
46      errorWindowLayout.getChildren().addAll(errorLabel, okButton);
47
48      Stage errorStage = new Stage();
49      errorStage.initModality(Modality.APPLICATION_MODAL);
50      errorStage.initOwner(parentPane.getScene().getWindow());
51      errorStage.setScene(new Scene(errorWindowLayout));
52      errorStage.setTitle("Error!");
```

```
53        errorStage.setResizable(false);
54
55        okButton.setOnAction(event -> errorStage.close());
56
57        errorStage.show();
58      }
59
60      public void processBMI(Pane bmiInputLayout) {
61        Platform.runLater(new Runnable() {
62          @Override
63          public void run() {
64            TextField weightTextField = (TextField) bmiInputLayout.lookup("#weight-
65    text-field");
66            TextField heightTextField = (TextField) bmiInputLayout.lookup("#height-
67    text-field");
68
69            double weight, height;
70
71            try {
72              weight = Double.parseDouble(weightTextField.getText());
73              height = Double.parseDouble(heightTextField.getText());
74
75              Double bmi = BmiCalculator.calculateBMI(weight, height);
76              DecimalFormat bmiDecimalFormat = new DecimalFormat("##.##");
77              String bmiString = bmiDecimalFormat.format(bmi);
78
79              // the stage was created already
80              if (bmiOutputStage != null) {
81                ((Label) bmiOutputStage.getScene().getRoot().lookup("#bmi-
82    result")).setText(bmiString);
83
84                bmiOutputStage.show();
85              } else {
86                Pane bmiOutputLayout = new BmiOutputLayout().getBmiOutputLayout();
87                bmiOutputStage = new Stage();
88
89                ((Label) bmiOutputLayout.lookup("#bmi-result")).setText(bmiString);
90
91                ((Button) bmiOutputLayout.lookup("#ok-btn")).setOnAction(event -> {
92                  bmiOutputStage.close();
93                });
94
95                Scene outputScene = new Scene(bmiOutputLayout);
96                bmiOutputStage.setScene(outputScene);
97                bmiOutputStage.setTitle("BMI Result");
98                bmiOutputStage.setMinWidth(300);
99                bmiOutputStage.setResizable(false);
100               bmiOutputStage.show();
101             }
102           } catch (NumberFormatException invalidNumber) {
103             showErrorWindow(bmiInputLayout, "Invalid height/weight.");
104           } catch (IllegalArgumentException illegalArgument) {
105             showErrorWindow(bmiInputLayout, illegalArgument.getMessage());
106           }
107       }
```

```
108         });
109       }
110     }
```

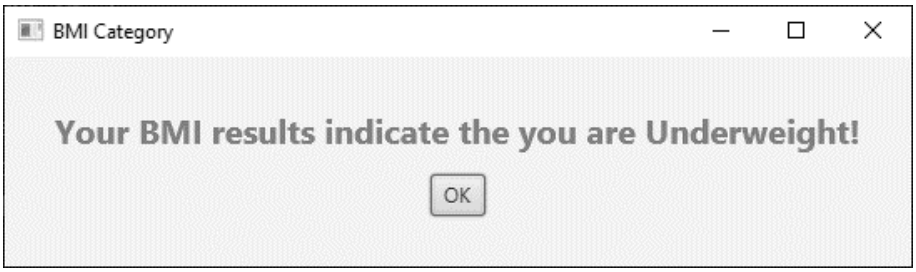
If we run the above code, we get to see the following phenomena:

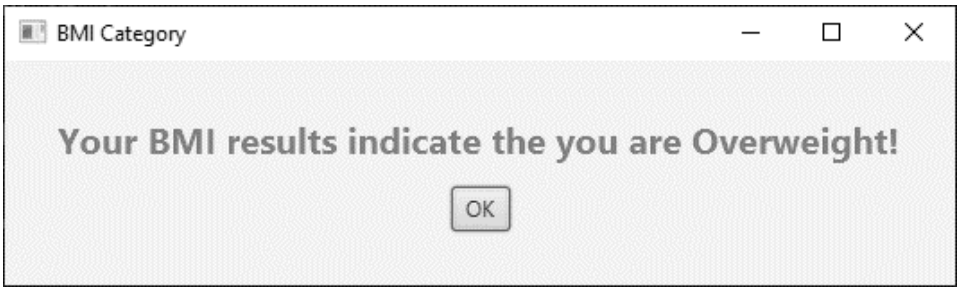
a) BMI Input Window shows up first.
b) BMI output window isn't created multiple times. If it is already open, the value of the BMI is merely updated. It is also non-blocking.
c) Once the user gives some erroneous input, the error window pops up, showing the cause of the error, and is blocking access to other windows.
d) BMI input and output windows are completely separate of each other. Closing one down doesn't close the other one. This is because they are running on individual threads.

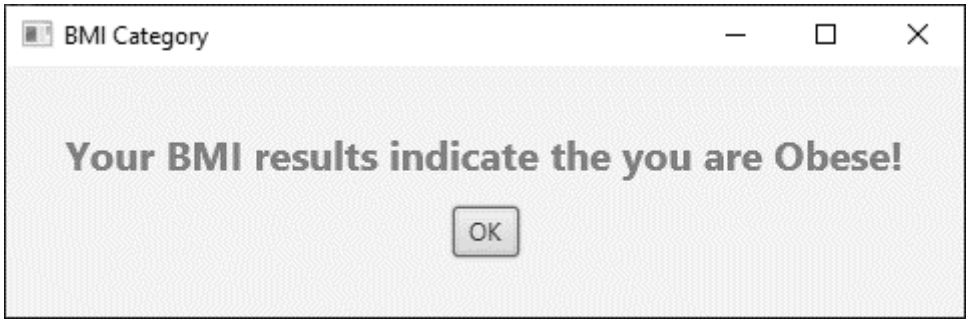
Home Task

Modify the above example to include a third window/stage.

a) It should contain a button and a label.
b) This window pops up if your BMI results fall in the categories Underweight, Overweight or Obese.
c) Make it non-blocking.
d) Use a separate thread from the thread pool to construct and display the window.
e) Use three different text colors for each of the categories. Consider using a HashMap to map the colors to the three categories.






BMI Category
Your BMI results indicate the you are Obese!
OK