COMS/SE 319: Software Construction and User Interface
Fall 2018
LAB Activity 4 – More Node.js & JavaFX

Task 1:Node.js

Objectives:

Learn to use node.js programming.

Warm-up:

NOTE 1: For installing node.js and running node.js codes, please refer to the previous lab activity [***LAB Activity 03_More JavaScript+ Node.js.zip***]

NOTE 2: Play with the given example - ***example.js***. Open using a text editor of your choice and modify to learn how the different instructions work.

Task:

Your lab activity task is to **create a simple binary calculator program by modifying the given *example.js*. You do not need to submit this file into Canvas just answer the Quiz question in Canvas.** You can start with the given warm-up example "***example.js***" and follow lab activity 3. You need to install **'readline-sync'** like here.

For a binary calculator,

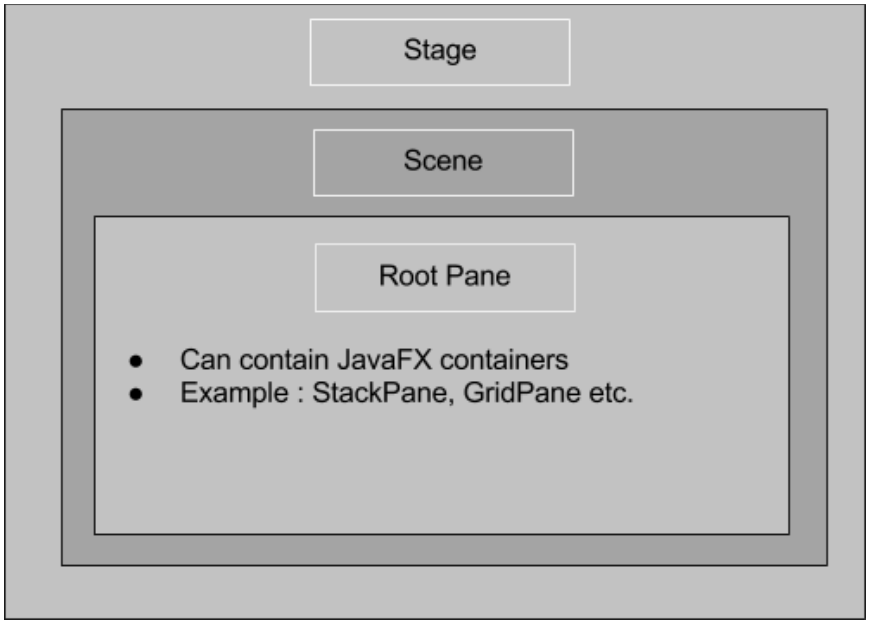
1. Note that for some operations on the binary calculator, it may be more convenient to convert the binary numbers to integers and then do the operation. (It is a suggestion, you can implement your own logic).
2. You can assume that only positive binary numbers are represented and used. For example, positive 9 is represented as 1001.
3. Binary operator "+" represents addition operation.
4. Binary operator "*" represents multiplication.
5. Binary operator "/" represents division.
6. Binary operator "%" represents mod or remainder (i.e. divide the first value by the second, what is remaining, only works on positive numbers).
7. Binary operator "&" represents AND (only works on positive numbers) e.g. (101 & 1011 gives 0001)
8. Binary operator "|" represents OR (only works on positive numbers) e.g. (101 | 1010 gives 1111)
9. Unary operator "~" represents not (i.e. invert each bit of the binary value, only works on positive numbers) e.g. (101 ~ gives 10).

You do not need to submit this file into Canvas just answer the Quiz question in Canvas Quiz.

Task 2: JavaFX

JavaFX is a set of graphics and media packages that enable developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms. See what JavaFX capable of in JavaFX Overview.

JavaFX Architecture



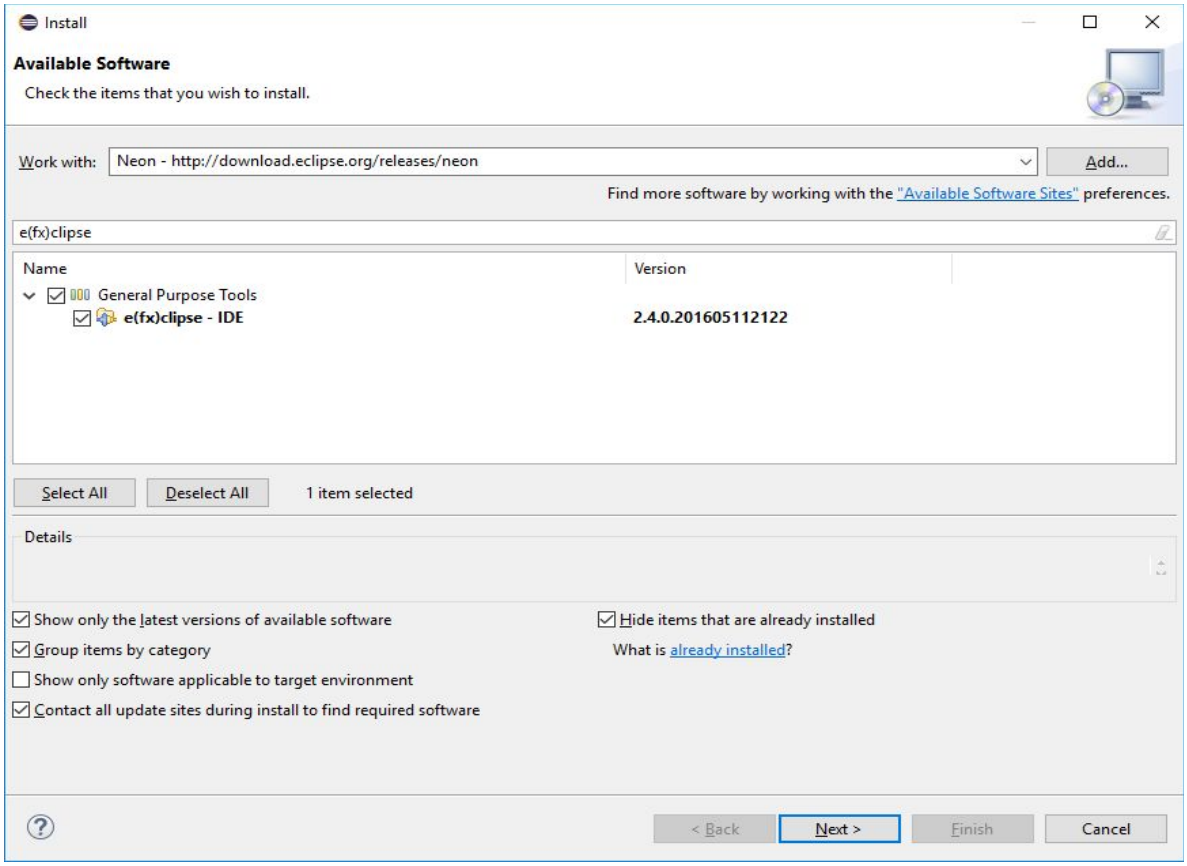
- **Stage**: This is the outermost container of the application and contains the entire program.

- **Scene**: The object directly contained by the stage. Just like a theatre, a scene can only exist inside a stage and a stage can have only one scene at any given time. When the story changes, the play goes to a NEW scene. In the program when you want to switch to another view you can also have the Stage change to a different Scene.

- **Root Pane/Container**: The container can contain other parts of the application like buttons, labels, text fields, etc. Depending on how we want to layout our apps, the root container can be represented by a number of JavaFX containers such as the StackPane, GridPane, BorderPane, FlowPane, etc. More about Root Pane can be found here.

Installation

1. Click Help->Install New Software.
2. Select Eclipse version to Work with.
3. In the filter enter e(fx)clipse.
4. Restart Eclipse after installation.



Task 2.1:

Let's check whether we properly installed JavaFX by running the following program.

1. Create a new JavaFX project (File ->New-> Project->JavaFX project)
2. Create **JavaFXExample1.java** and copy the following code.

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;


public class JavaFXExample1 extends Application {

   @Override
   public void start(Stage primaryStage) {
       Button bt = new Button("Click Me");
```

```
        StackPane frame = new StackPane();
        frame.getChildren().add(bt);

        bt.setOnAction(new EventHandler<ActionEvent>() {

            @Override
                public void handle(ActionEvent e) {
                System.out.println("javaFX says Hello !!");
             }

             });

        Scene scene = new Scene(frame, 300, 250);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }

}
```

After running the code please answer the following Attendance quiz question in Canvas.

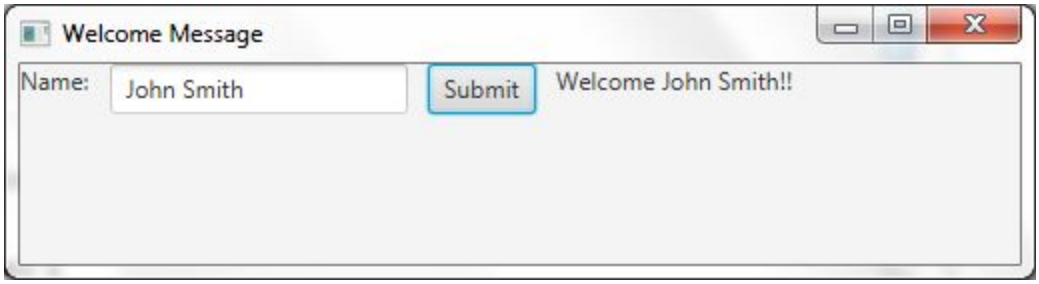
How to change the title of the JavaFXExample1 output stage to "Example 1"?

Task 2.2:

Complete EventHandler for **Submit** button click in provided **JavaFXExample2.java**

The program should print Welcome message if a name is provided in text box. Otherwise, print an error message "You have not entered name."

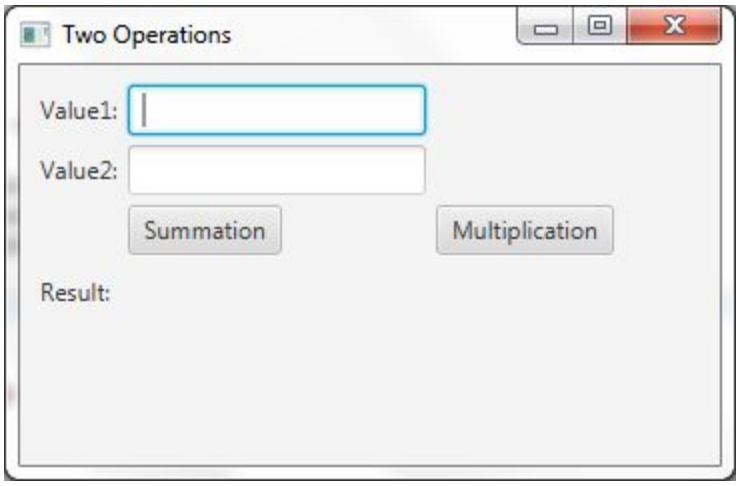
[You can read the text in text box using getText() and write to label using setText()]



setOnAction method sets the action details when that particular action is fired.

Task 2.3:

Complete the provided **JavaFXExample3.java.**
You can use "result" label to print the answer.



After running the code please answer the following Attendance quiz question in Canvas.

Will this code be sufficient for handling summation in the given JavaFXExample3?

```
sum.setOnAction(new EventHandler<ActionEvent>() {
                  @Override
                  public void handle(ActionEvent event) {
                        try {

result.setText((Double.parseDouble(value1.getText()) +
Double.parseDouble(value2.getText())) + "");
                        } catch (NumberFormatException e) {
                              result.setText("Invalid Inputs");
                        }
                  }
            });
```

You can find more about JavaFX UI Controls from JavaFX Documentation. Also there are other ways to build javaFX UI controls. Scene builder http://gluonhq.com/products/scene-builder/ is one of them.
