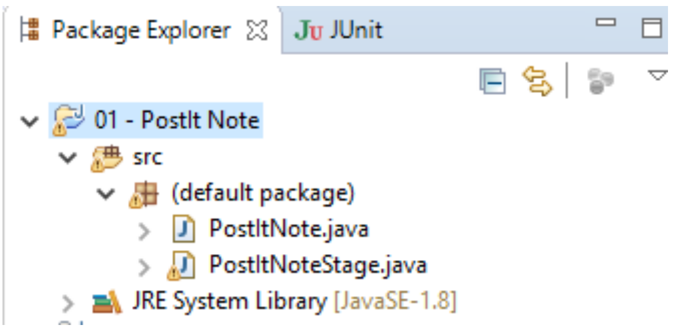
Practical Exercise (INFT 1004 Version .2 2nd May 2018)

Instructions

In this practical exercise you will be developing a Post-It note application using JavaFX. Your major assignment will also use JavaFX so this exercise will assist you with understanding some of the available JavaFX components that are used to create Graphical User Interfaces. You should use the Eclipse development environment and only use java and JavaFX (i.e. don't use netbeans or other tools). The practical is broken up into sections that increase in difficulty as you progress. **You should NOT consider this a step-by-step guide** – as you progress the instructions will reduce in detail and you should use you the skills you have developed to solve the steps. You should spend some time researching and exploring the API in order to fully implement all the requirements of the practical exercise.

You will add source files to the default package as you progress through the exercise.

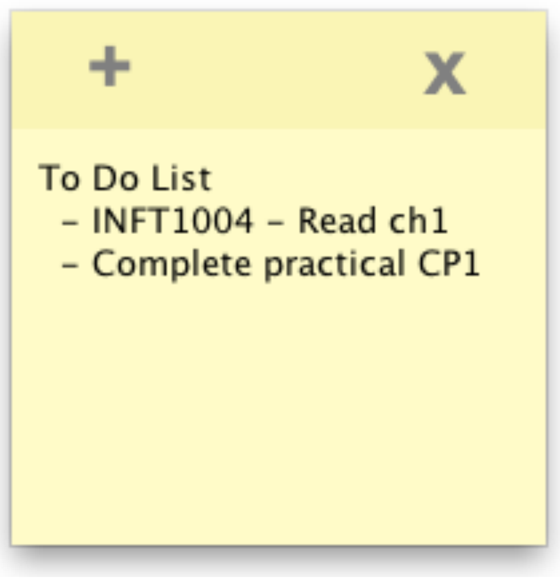


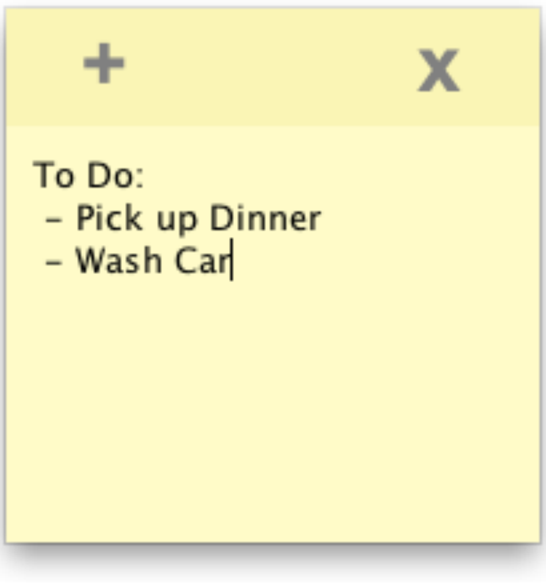
```
Package Explorer   JUnit
01 - PostIt Note
  src
    (default package)
      PostItNote.java
      PostItNoteStage.java
  JRE System Library [JavaSE-1.8]
```

Presentation and Comments

All code should be carefully commented using javadoc. You should clearly explain the code logic and structure so someone who is unfamiliar with your code can quickly understand the logic and why code choices have been selected.

JavaFX Exercises – Creating a "Post-it note" application





To Do:
- Pick up Dinner
- Wash Car

Task 0 – Setting Up JavaFX

To get started with JavaFX you can either create a new JavaFX Project or add the JavaFX library to the project.

To create a JavaFX Project in Eclipse,

1. Select **New**
2. Select **Other**
3. Find and open the **JavaFX** folder
4. Select **JavaFX Project**
5. Step through the usual Eclipse project creation

If you cannot find the JavaFX folder in the dialog. Close the dialog and follow the steps found at:

https://www.eclipse.org/efxclipse/install.html#for-the-lazy

Once the installation process is complete, follow the steps above to create your JavaFX project.

Task 1 – Creating a JavaFX Application

We will start by writing a skeleton application. Our main window has to be derived from the JavaFX stage class.

Let's start by creating the class **PostItNote.java**

Create a new java project in Eclipse called “01 Post-It Note” with Eclipse and create a file PostItNote.java in the src directory.

This class should be:

- public
- extend Application
- have a single constructor
- have a 'main' method
- have a ‘start’ method

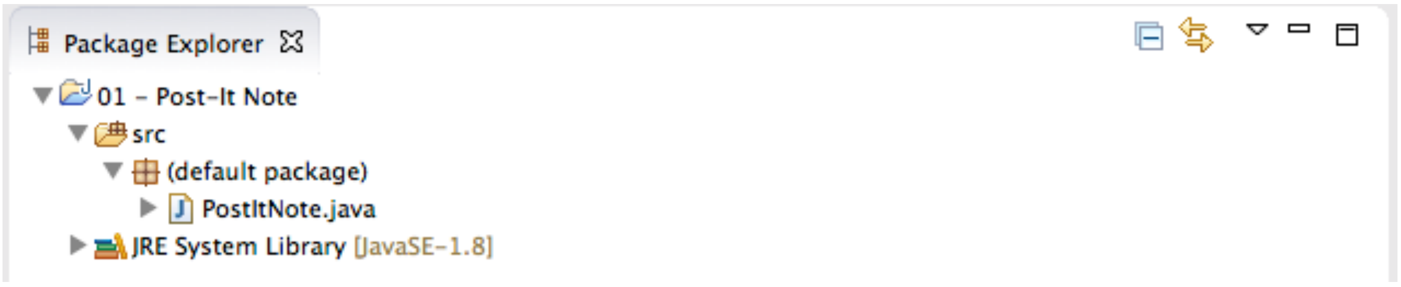
In the main method you should print the following lines to the console:

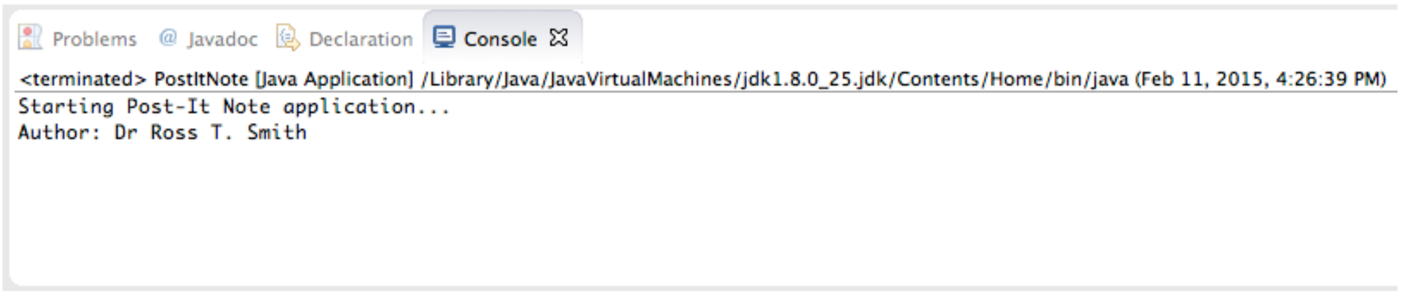
"Starting Post-It Note application..."
“Author: Dr Ross T. Smith” (Note you should use your own name here!)

Note: In order to launch the application you will need to add the following code to the main method.

```
launch(args);
```

Write the skeleton code for this class, compile to check for syntax errors and run the application.





```
<terminated> PostItNote [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Feb 11, 2015, 4:26:39 PM)
Starting Post-It Note application...
Author: Dr Ross T. Smith
```

Task 2 – Creating the Stage and our Scene

The next step is to create the Stage where our Post-It note application comes to life. We will create a new class called PostItNoteStage.

Create a new file called **PostItNoteStage.java** You should write code so your class has the following properties:

- has a constructor with four parameters (double sizeX, double sizeY, double positionX, double positionY)

Create a new Stage so we can add our scene to it and set the position of the stage.

```
        stage = new Stage();


        stage.setX();
        stage.setY();
```

Next we need to create a Scene that holds our GUI components. One way to think of the difference between a Stage and a Scene is to imagine that the stage is our empty canvas, and a we paint a scene on the canvas. When we add new visual elements, we add them into the scene, rather than the stage.

You may like to check the on-line documentation to read more about these methods at:

https://docs.oracle.com/javase/8/javafx/api/toc.htm

You do not need to import anything to begin with, you can import the JavaFX components as needed.

Note: Some JavaFX components share a name with Java Swing components. You must ensure you are importing the correct component from the javafx.* library.

To create a new Scene we need to set it's layout and its size.
For now we'll create a simply add a Pane layout to our Scene and set its size using the parameters we passed through in the constructor.

```
Scene scene = new Scene(new Pane(), sizeX, sizeY);
```

Add the scene to our stage

```
stage.setScene(scene);
```

Now compile your code and remove any syntax errors.

Task 3 – Making the Stage visible

The next step is to instantiate your new PostItNoteStage class from the PostItNote application and make your PostItNoteStage visible.

In your PostItNote.java file you should add to the constructor a line to instantiate the first PostItNoteStage. You should create a class variable in the PostItNote class called “mainWindow” and instantiate it in the start method.

```
//Create the main window for our first post it note
mainWindow = new PostItNoteStage(200, 200, 0, 0);
```

In PostItNoteStage you also need to make the stage visible.

```
//Make stage visible
stage.show()
```

Task 4 – Planning the layout

Now that you have prepared the basic skeleton application it is time to give some thought to the layout of the application. Your Post-It note application will have two buttons (create new note and delete note) at the top and a large text area at the bottom. This simple design is easily represented with a BorderPane layout using the Center and Top locations. One technique I find very useful to help you understand exactly what the layout manager is doing is to create a Pane for each of the main sections and change its colour so you can see the regions it is creating.

Create the following two class variable and instantiate them in the constructor of PostItNoteStage

```
BorderPane content;
BorderPane buttonArea;
```

Instantiate each in your constructor

Let's change our code so that our new layout is added to the scene instead of the default we set earlier. You should set it to use the BorderPane you have created

```
Scene scene = new Scene(content,sizeX,sizeY);
```

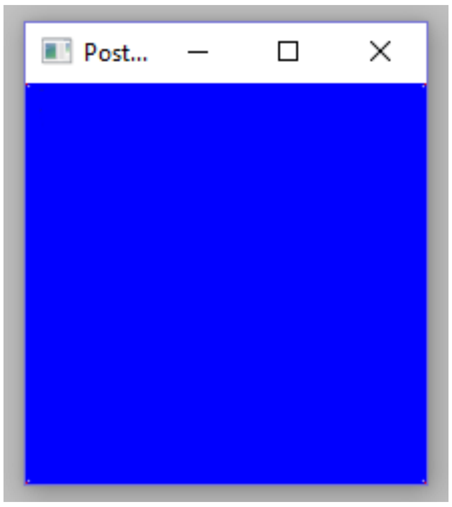
You should also set the color of "buttonArea" to red and "content" to blue. To do this we will need to use some CSS code.

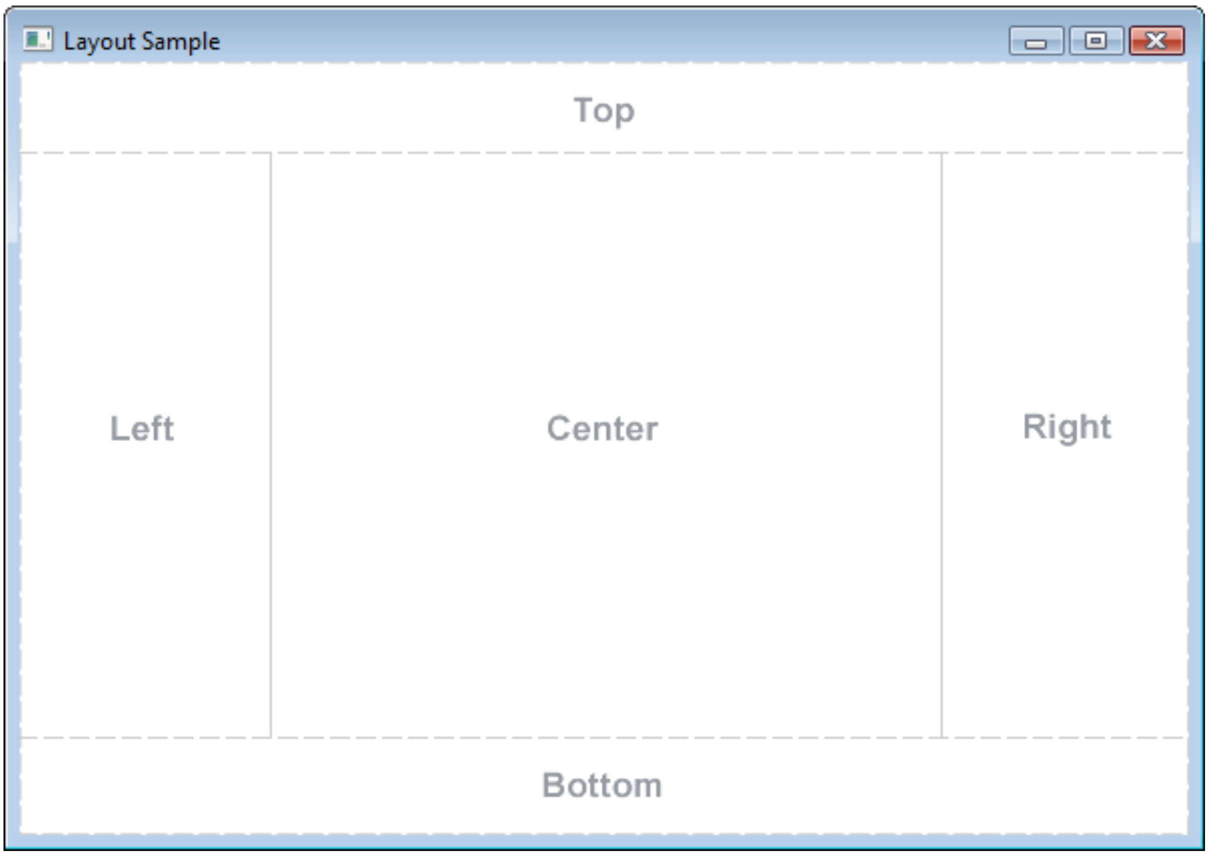
```
content.setStyle("-fx-background-color: blue");
buttonArea.setStyle("-fx-background-color: red");
```

Finally you should add the buttonArea BorderPane to the content pane.

```
content.setTop(buttonArea)
```

At the moment you will not be able to see the buttonArea because there is nothing in it.




Layout Sample
Top
Left
Center
Right
Bottom

Task 5 – Adding Buttons

Next you will add the two buttons add (+) and delete (x) to the button area you just created. For this you can use the Button class which provides all the functionality you need.

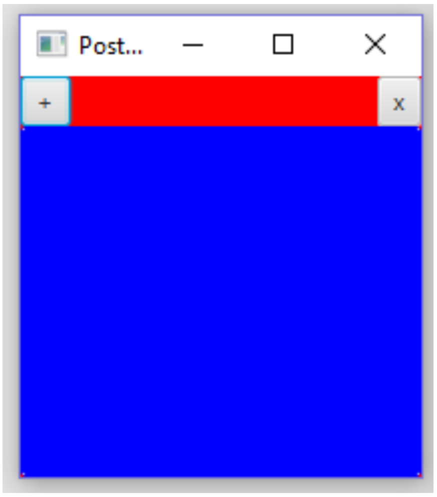
Create two class variables

```
Button newPostItNote;
Button deletePostItNote;
```

Instantiate them in your constructor and add them to the buttonArea BorderPane.

```
…= new Button("+");
…= new Button("x");
buttonArea.setLeft(newPostItNote);
…
```

You will notice the buttonArea BorderPane will expand to accommodate the two new buttons

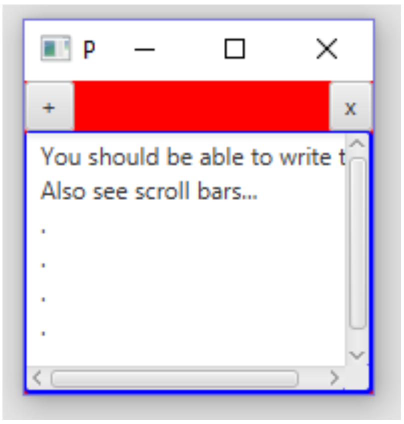


Task 6 – Creating a text area where we can write notes

You need to add a region where you can take notes. For this we will use a TextArea component

```
TextArea textArea;
```

Add this to the Center of our content BorderPane.



```
You should be able to write t
Also see scroll bars...
.
.
.
.
```

Task 7 – Modifying the appearance of the Post-It Note

At this point your Post-It note is hideous as it does not follow a typical color scheme. In this step you will be modifying the appearance of all the components to use various yellow and grey shades.

We will refer as everything located in the Top section to be the title and everything in the Center to be the body. There are many ways to specify a colour in CSS, in this example we will be using rgb values from 0-255 (0 being no colour and 255 being full colour).

For example:

```
content.setStyle("-fx-background-color: red”);
```

Is the same as writing

```
content.setStyle("-fx-background-color: rgb(255,0,0)”);
```

I have provided the RGB values used in my example, but try experimenting with your own colour schemes.

Title: `rgb(248, 247, 182)`
Content: `rgb(248, 253, 201)`

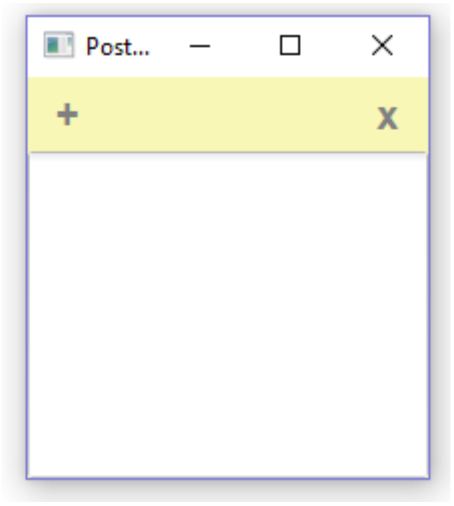
Now we will start off by modifying the title regions. Firstly, you can change the color of the buttonArea BorderPane that is currently set to RED. You should change it to use the Title rgb values.

The Buttons (newPostItNote and deletePostItNote) you created also need to have their appearance changed to match in with the yellow. This can be done the same way you changed the colours of the buttonArea and textArea.

Now you can alter the font size and colour so the button text is larger and grey in color. Create a new Font buttonFont class variable and change the button font color using the pre-defined Color.GREY.

```
buttonFont = Font.font("Arial", FontWeight.BOLD,20);
newPostItNote.setFont(buttonFont);
newPostItNote.setTextFill(Color.GREY);
```

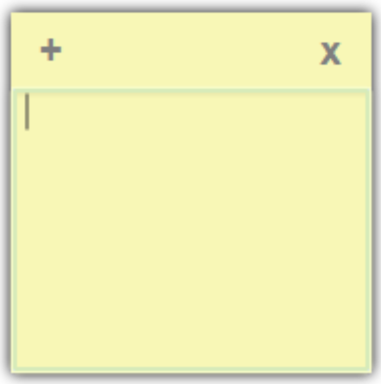
Do the same for the deletePostItNote button



Task 8 – Modifying the text area appearance

Now it's looking more like a Post-It Note, we should also change the text area to look similar but with a slightly lighter yellow color (using the RGB values provided earlier).

Unfortunately this gets a bit tricky as the default TextArea doesn't behave the way we want it even when we change the background colour and the inner colour via CSS.

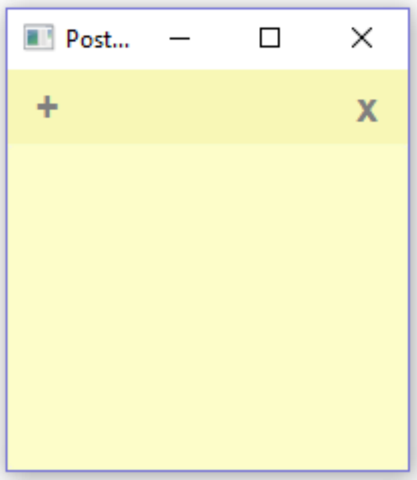


To do this we need to cast the inner part of our TextArea as a Region, which is the base class for Node-based UI elements and modify it that way.

```
Region region = (Region) textArea.lookup(".content");
region.setStyle( "-fx-background-color: rgb(253, 253, 201)" );
```

As the lookup function is using CSS, we need to make sure that the Stage is showing for this method to work properly, so make sure you place it **after** stage.show().

Now your Post-It Note should be looking something like this.



Task 9 – Event capture for the + (new) button

We have two buttons that currently do not operate. To make them operate we need to capture the events. For this we can implement the EventHandler class.

Add this code so we can create an action for our newPostItNote button.

```
EventHandler<ActionEvent> newButton = new EventHandler<ActionEvent>(){
        @Override public void handle(ActionEvent e){
                new PostItNoteStage(sizeX,sizeY,0,0);
        }
};
```

Add the event handler to your newPostItNote button.

```
newPostItNote.setOnAction(newButton);
```

Run your application and you should be able to create a new Post It Note by clicking the 'new' Button.

However when it is created it will be placed in the top left corner every time. So we need to adjust the position. To do this we will need to find both the current 'stage' position and check the screen dimensions to ensure there is enough room to fit your new note.

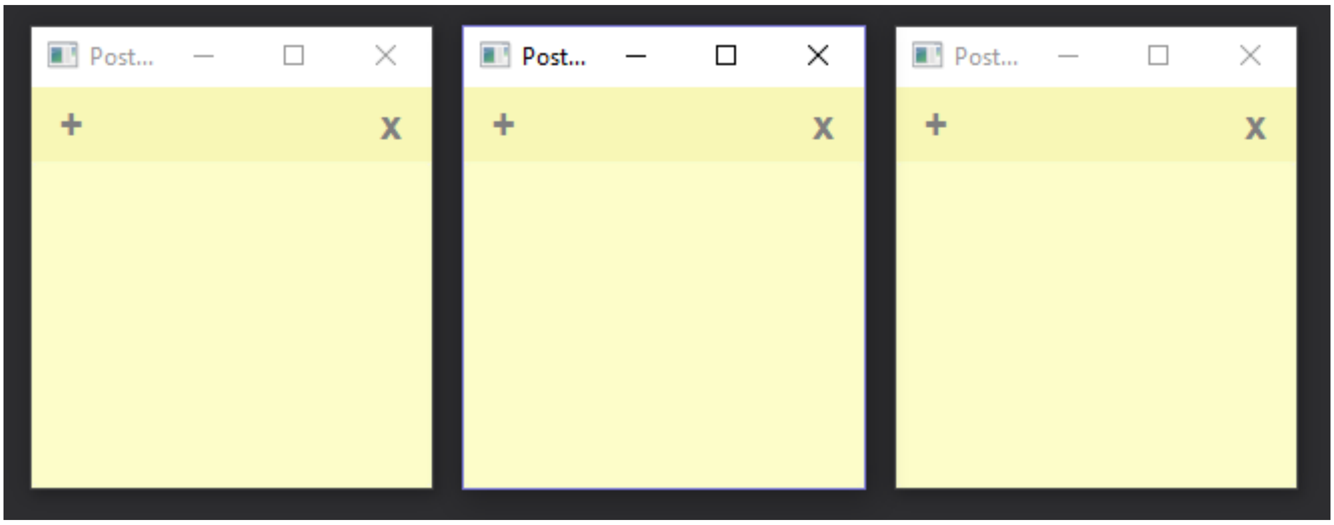
We can start by storing the x and y position of the current stage . The origin location is the top left corner of your postit note. Retrieve them with the following commands.

```
x = stage.getX()
y = stage.getY()
```

Assuming we would like the new note to be created immediately to the right of the original one we need to add the width to the start location.

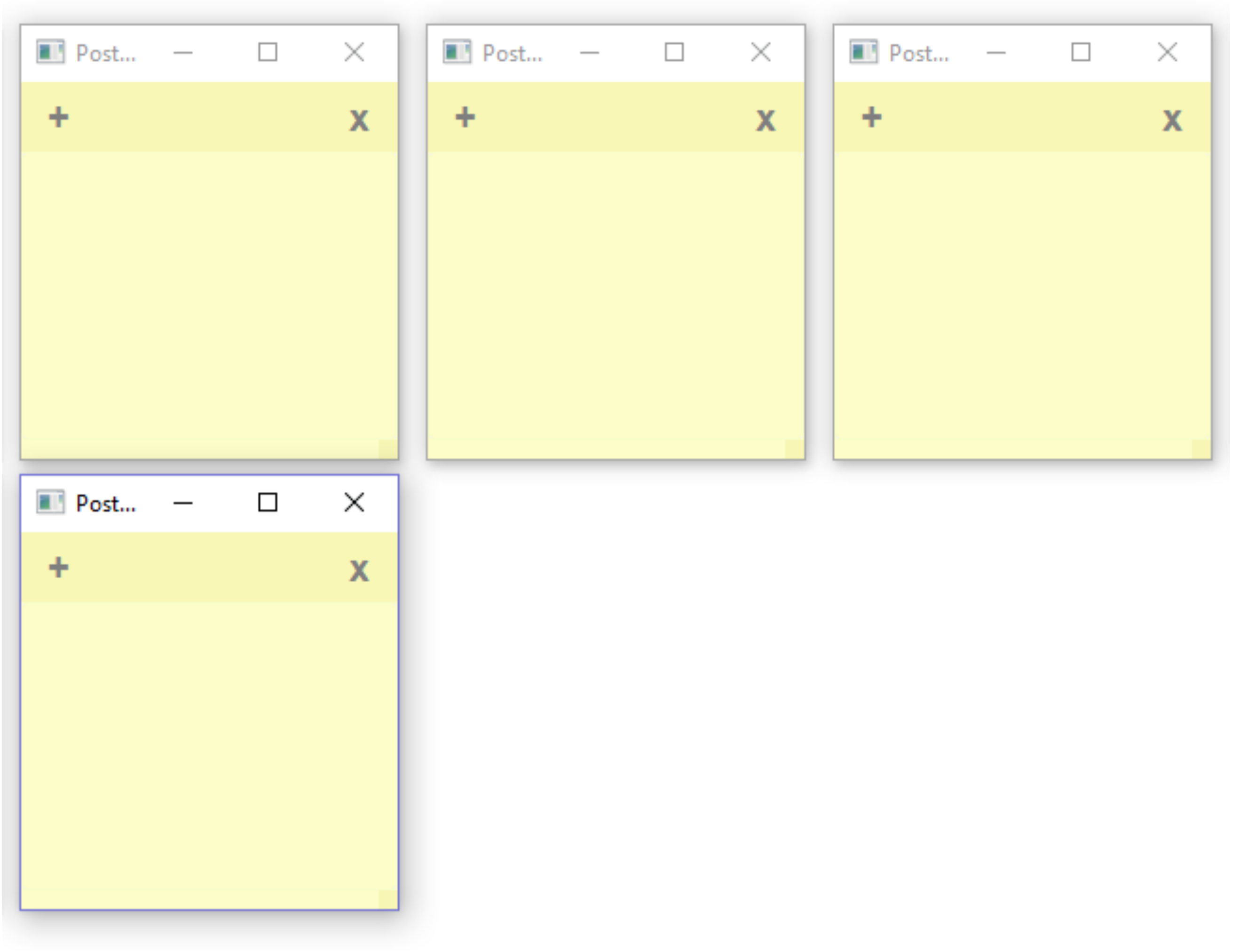
```
double newX = x + stage.getWidth()
double newY = y
```

Pass your new positions as parameters to your constructor and run your application. You should be able to create new postit notes that appear immediately to the right of the previous note.



However a problem occurs when you reach the width of your monitor. So one solution is to check if the new X position exceeds the width and if so move down and go back to the start of the left side of the screen. To achieve this we need to get the screen width and add some logic to calculate the new starting position.

```
//Check we fit within the screen x size
Rectangle2D screen = Screen.getPrimary().getVisualBounds();
if(stage.getX() + stage.getWidth()*2 > screen.getWidth()){
      newX = 0;
      newY = stage.getY() + stage.getHeight();
}
```



Task 10 – Close Button Event

Now we need to make the x (close) button work as expected. Given you can have many notes open simultaneously the x should only close the one note. Using your button event handling that captures the close event, change to dispose of the current window.

```
stage.close();
```

Run and test the function.

Task 11 – Right Click Menu

Currently if you right click in the textArea a context menu appears with most of the features we need. However, since this is a learning exercise, let's reinvent the wheel and make our own Right Click Menu with the items cut, copy, paste, about and exit.

Firstly we need to create a ContextMenu and name it rightClickMenu.

```
rightClickMenu = new ContextMenu();
```

Then we create the MenuItem to add to rightClickMenu

```
cut = new MenuItem("Cut");
rightClickMenu.getItems().add(cut);
```

Follow this template and add the rest of the items; copy, paste, about, and exit.

Now we need a way to display our menu instead of the default. To do this we add an EventFilter which "consumes" the event,

```
textArea.addEventFilter(ContextMenuEvent.CONTEXT_MENU_REQUESTED, Event::consume);
```

Once the EventFilter has gobbled up our unwanted event, we can detect a right click by using a MouseEvent EventHandler.

```
EventHandler<MouseEvent> rightClick = new EventHandler<MouseEvent>() {
              @Override public void handle(MouseEvent e) {
              if(e.getButton() == MouseButton.SECONDARY)  {
                  menu.show(content, e.getScreenX(), e.getScreenY());;
              }

              }});
```

Now add it to our content and textArea Panes.

```
content.setOnMouseClicked(rightClick)
```

Now implement the events for each of the MenuItems using the same template as used for the new and close buttons. Create a ClipboardContent object in order to set the content on the clipboard. You can access the clipboard with:

```
Clipboard.getSystemClipboard()
```

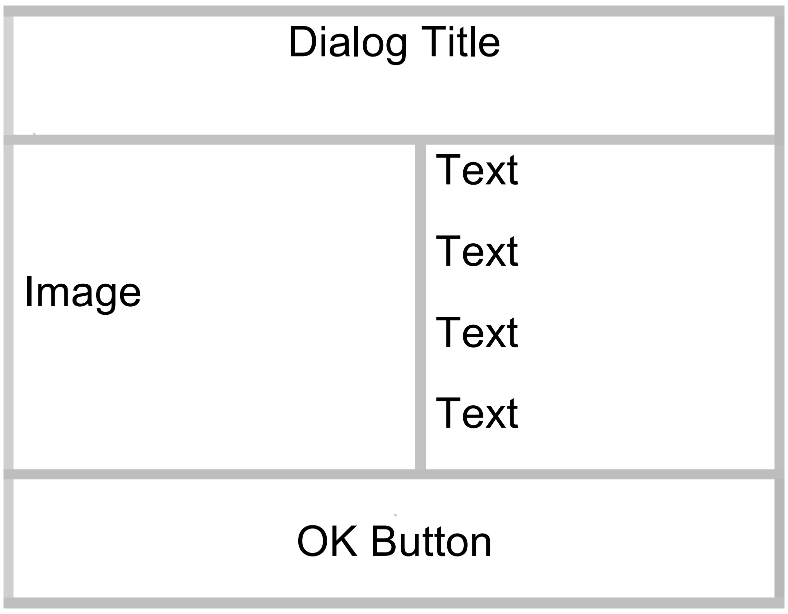
Note that you will need to check and define which data type you are either putting on or retrieving from the clipboard. Here is my code for the paste function.

```
if(Clipboard.getSystemClipboard().hasString()){
      textArea.appendText(
            (String)Clipboard.getSystemClipboard().getContent(DataFormat.PLAIN_TEXT));
}
```

Compile and run your application to test out the menu.

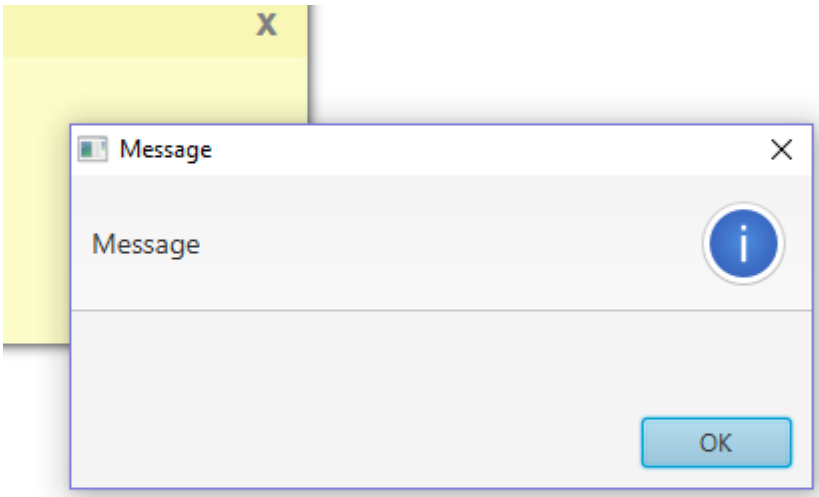
Task 12– About Dialog

We want to add an about dialog to provide some information about the functionality of the program, and author details. The dialog will display when the about item of the right click menu is selected, looking something like the following template:



Our Dialog box will be an Alert with AlertType.INFORMATION. Add the code to show the Alert to your "about" event.

If done correctly, when you click about you should see the following dialog.

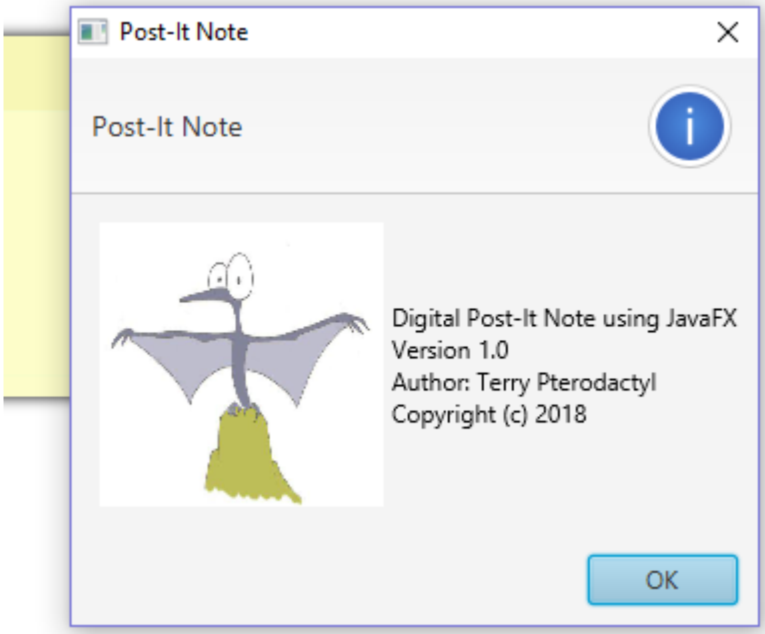


Your next tasks are to implement the about dialog in alignment with the template given above. You will need:

- A GridPane for the layout of the content
- A Text component for the text in the GridPane
- An ImageView for the picture

The picture should be a picture of you, saved in a common image format and place inside your project folder. You can either scale it to a suitable size manually, or do it programmatically. Initialise your Image variable as follows:

```
Image image = new Image("pic.jpg");
ImageView imageView = new ImageView(image);
```



Digital Post-It Note using JavaFX
Version 1.0
Author: Terry Pterodactyl
Copyright (c) 2018

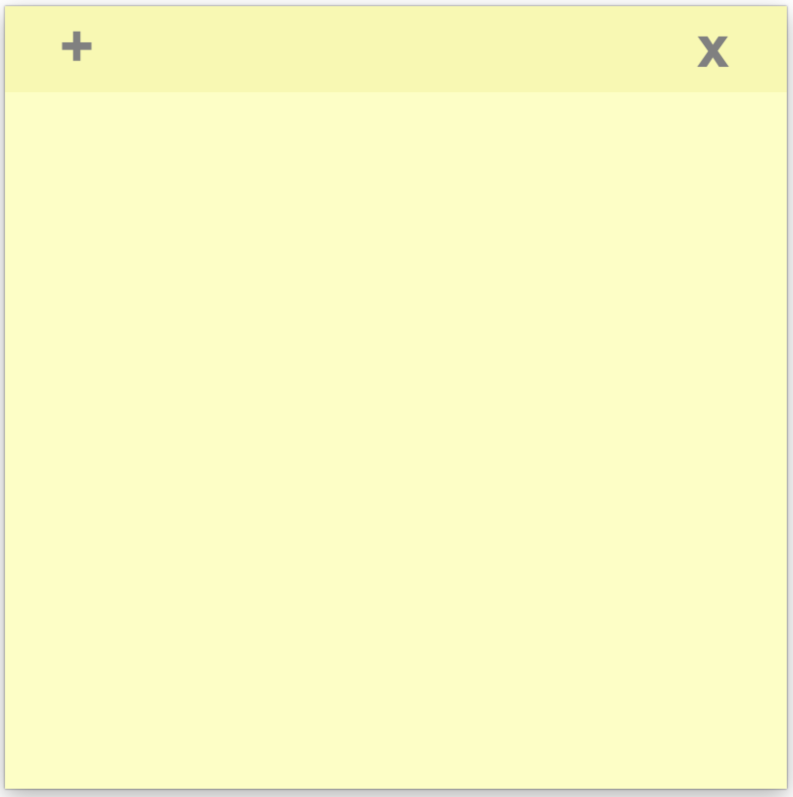
Task 13 – Undecorated and re-size

The final task is to make the application look like a more realistic Post-It note. The current limiter of this is the operating system window decoration. Windows and OS X add their own title bar to the window, which does limit the realism of our application.

To remove this, add one line to the constructor of your PostItNoteStage class.

```
stage.initStyle(StageStyle.UNDECORATED);
```

If you run the program now, you will see the note without any OS-specific decoration.



There is one problem, however. The OS decoration enabled the move and resize functions. Now that we have taken that away, the note cannot be dragged around or resized. We can add that functionality ourselves.

First, we need to add an .onMousePressed() event to our buttonArea.

Create a new EventHandler<MouseEvent> like we did for our rightClickMenu. There we can calculate an offset for the X and Y axes from where we clicked to the post it notes position on the screen.

This stops the Post It Note from jumping to the spot where we clicked, and the top left will always be the same distance away from the cursor while dragging.

Do this by subtracting the click position from the stage position and store these as doubles OUTSIDE of your class constructor.

Now we set the position of the stage in the .onMouseDragged() method. This is simply done by adding the offsets we stored in .onMousePressed() to the current screen coordinates of the cursor.

Run and debug your application. If you have correctly implemented the formula, you will be able to move the Post-It by dragging on the title bar.

The resize function is similar to moving the window, but instead of setting a new location, you need to set the size of the PostItNoteStage. There are many methods for implementing this, but one could include adding a panel to the bottom right of the application and activating a similar method to the drag method.

Try setting the width to the original width + the amount the cursor has moved on the x axis and repeat for the y.

Lastly, restrict the resizing to remain, at minimum, the original size.