Lab 15 – Creating Graphical User Interfaces

*Last updated: 2.27.2025*

* **To complete this lab, you must either be working:**

1. **in the CS Open Lab (2111 Nevins) and completed Lab 15a, or**
2. **on a personal device and completed Lab 15b.**

Contents

[1 Introduction 2](#_Toc191982695)

[2 Create Hello World Gui 4](#_Toc191982696)

[3 The *GridPane* Class 7](#_Toc191982697)

[4 The *HBox* Class 9](#_Toc191982698)

[5 The *VBox* Class & *CheckBox* Class 11](#_Toc191982699)

[6 Modularizing Gui Construction 13](#_Toc191982700)

[7 The *ComboBox* Control 15](#_Toc191982701)

[8 The RadioButton & ToggleGroup Controls 16](#_Toc191982702)

[9 The *ListView* Control 18](#_Toc191982703)

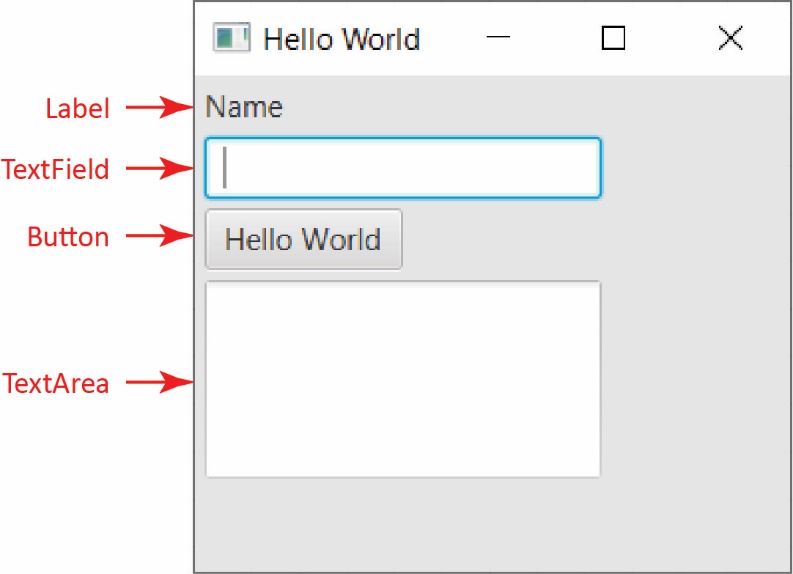
[10 Submission 19](#_Toc191982704)

[Appendix 1 n/a 19](#_Toc191982705)

# Introduction

This is a lab on writing *Graphical User Interfaces* (Gui).

**Read (no action required)**

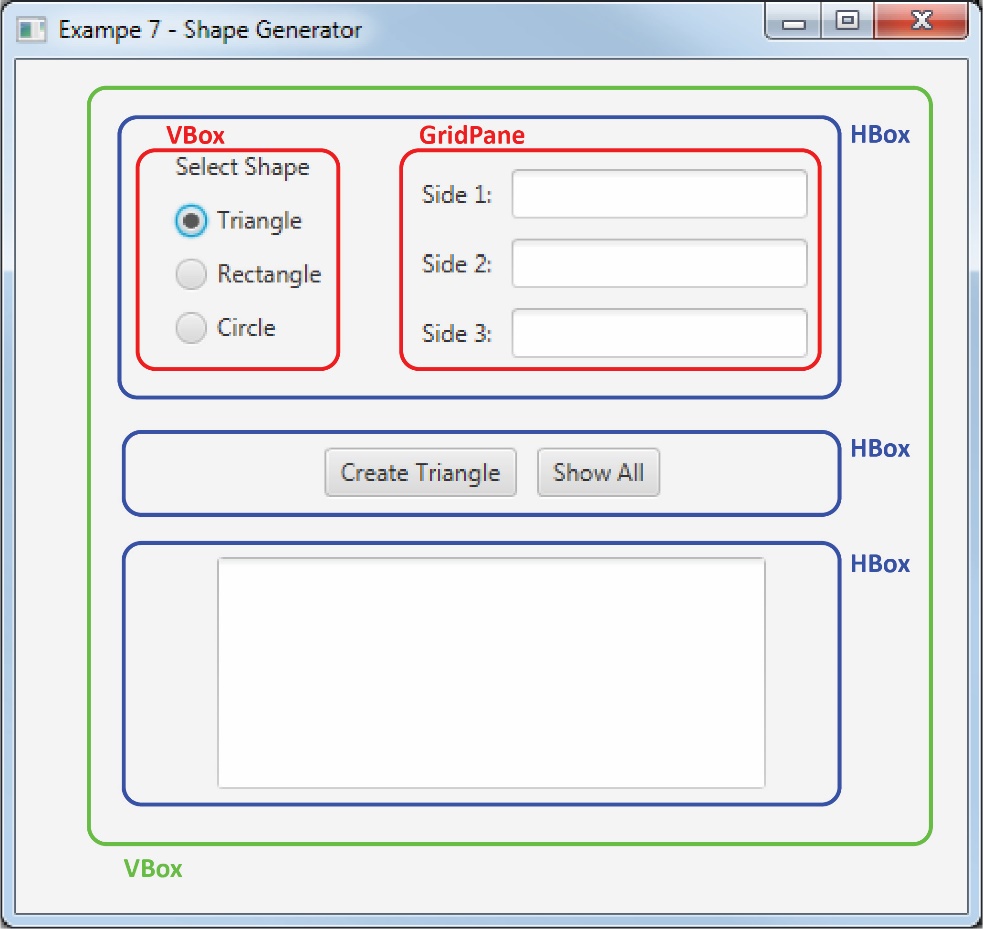
1. Some basic definitions:

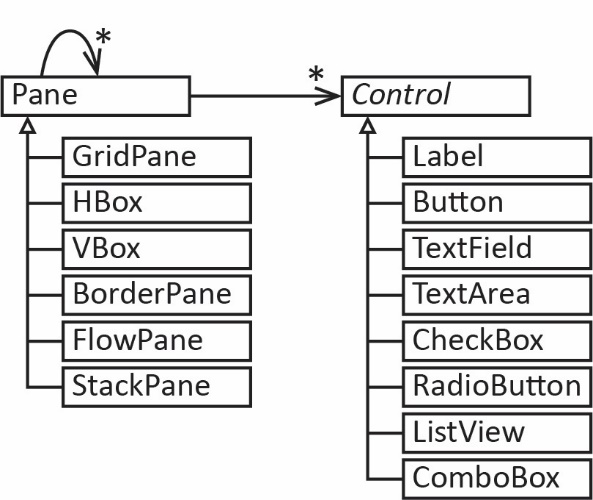
* ***Graphical User Interface* (GUI)** – A window that allows the user to interact with graphical icons (controls). A Gui is also called a *form* or *window.*
* ***Control*** – An element that is displayed on a GUI. For example, on the right we see the following controls: label, text field, button, text area. *Controls* are sometimes called *widgets*.
* ***Event Handler*** *–* An event handler is a piece of code that runs in response to a user interacting with certain controls. In the example on the right, when the button is pressed a *button event handler* is called to compute the monthly payment and total payment. Event handlers will be considered in the next lab.

1. To build a Gui application, there are two main tasks:

* Build the GUI.
* Write event-handlers.

This lab discusses how to build a Gui. The next lab discusses how to program event handlers.

1. Below are a few of the basic ideas and concepts that surround Gui construction in JavaFX. We will learn the details as we move through the chapter.

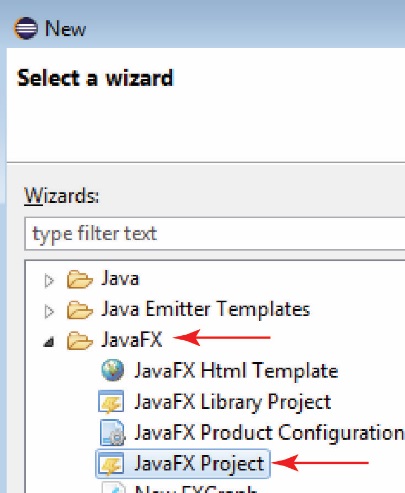
* [*JavaFX*](https://docs.oracle.com/javase/8/javase-clienttechnologies.htm) refers to a set of classes that we use to build a Gui.
* To build a Gui in JavaFX, you put *Controls* on a *Pane*.
* A *Pane* is a container that arranges *Controls* in a particular layout. For example, the red *VBox* pane (outlined for emphasis) contains a *Label,* and 3 *RadioButton*s.
* Panes can be nested. For example, the blue *HBox* at the top contains a *VBox* and a *GridPane.*
* A Gui must have exactly one *root* pane. The root pane in the Gui on the right is the green *VBox*. Best practice is to name the root pane, *root;* however, I’ve not been consistent with that in this Lab.
* *Control* is an abstract class in Java. Some common subclasses are shown in the diagram on the right. Some of the common controls are shown here:

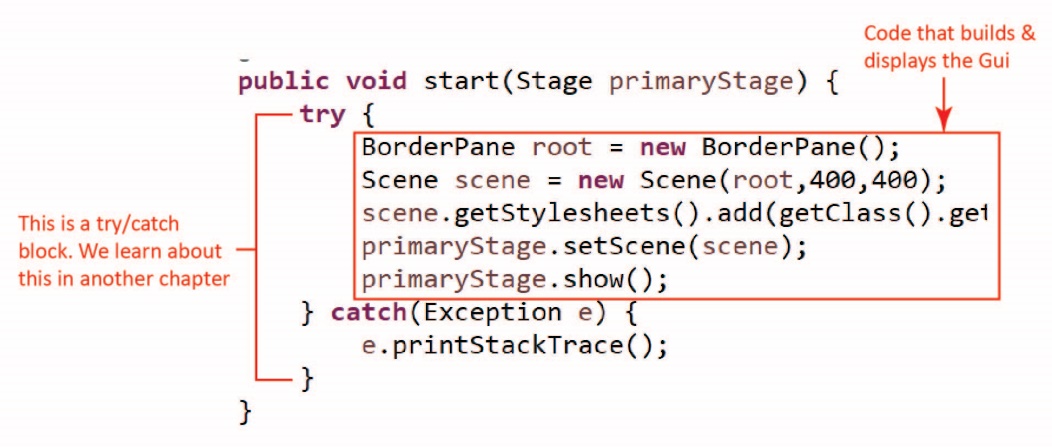
<https://docs.oracle.com/javafx/2/ui_controls/overview.htm>

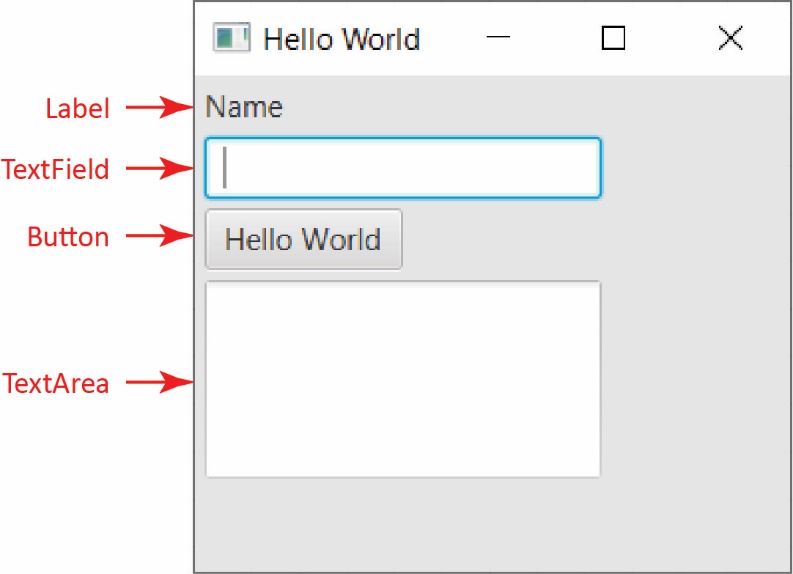
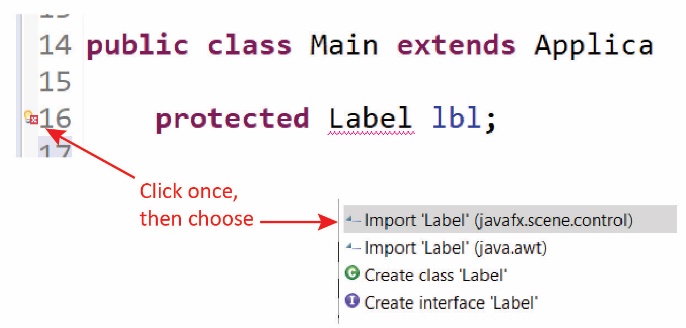
* *Pane* is a class in Java and is used as a container to hold *Controls* or nested *Panes*. Subclasses of *Pane* are used to layout controls in different ways. Some common subclasses are shown in the diagram on the right. For example, a *GridPane* arranges the controls in a grid (see Gui above), an *HBox* arranges controls horizontally in a single row. The diagram also shows that panes can be nested, *i.e.* a pane inside another pane. Scroll through this page to see examples of different types of *Pane*s:

<http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm>

# Create Hello World Gui

1. **Establish a Workspace** – Create a folder on your drive where you will put your lab or use an existing one.
2. **Run Eclipse** – As the program begins to run, it will ask you to navigate to the Workspace you want to use.
3. **Create a JavaFX Project** –
4. Choose: File, New, Other, JavaFX, JavaFX Project, Next (as shown on the right).
5. Supply a project name, *e.g. lab15\_gibson* and then choose: Finish
6. Attach the JavaFX libraries. See Lab 15a if necessary.
7. Expand the files in the Package Explorer. You will see the following items that were automatically created:
8. An *application* package.
9. A *Main* class which is where you will write your code.
10. A file named *application.css* which is a (empty) style sheet. Cascading Style Sheets (css) are very powerful and flexible technique for styling your GUI. This lab will use them, but the explanation will come in class.
11. A *JavaFX SDK* folder and a build file, *build.fxbuild* neither of which we will use directly (but don’t delete them).
12. Do the following:
13. Open *Main* and run the program and observe the empty window that is displayed. Thus, when you create a *JavaFX* project it comes with sample code that displays an empty Gui.
14. Look at the *start* method. It won’t make much sense yet, but the important point is that the code to build and display the Gui is within the *try* block. A *try/catch* block is a structure in Java that allows you to “try” to executre some code and if it fails, the *catch* block is executed. Our code will always be in the *try* block. We learn about *try/catch* blocks in another chapter.



1. **Create *Hello World* Application** – Open *Main* and do the following:
2. We are going to build a Gui that looks as shown on the right.
3. Add the instance variable (*i.e.* not in the *start* method) below which declares a *Label:*

**protected** Label lbl;

and then click on the red X as shown in the figure on the right (or hover your mouse over “Label”) and choose the import shown.

import ‘Label’ (javafx.scene.control)

**Warning: it may not be the first item in the list as shown in the figure above.**

**Note: you will have to do this at other places in this lab. For Gui components alwayschoose the ones in the *javafx…* package.**

1. Add these instance variables directly below the *Label*. You will need to import the corresponding packages as you did above.

**protected** TextField txfName;

**protected** TextArea txaMessage;

**protected** Button btnHelloWorld;

1. Identify the first two lines in the *try* block:

BorderPane root = **new** BorderPane();

Scene scene = **new** Scene(root,400,400);

And replace them with (continues next page):

// Create controls

lbl = **new** Label("Name");

txfName = **new** TextField();

txaMessage = **new** TextArea();

txaMessage.setPrefHeight(100);

txaMessage.setPrefWidth(200);

btnHelloWorld = **new** Button("Hello World");

// Create container for controls

GridPane root = **new** GridPane();

// Add controls to container

root.add(lbl, 0, 0);

root.add(txfName, 0, 1);

root.add(btnHelloWorld, 0, 2);

root.add(txaMessage, 0, 3);

// Set the size of window (pixels)

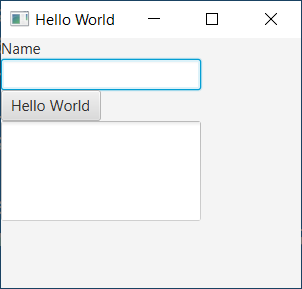
Scene scene = **new** Scene(root,300,250);

// Set the title for the window

primaryStage.setTitle("Hello World");

// Code to display the Gui

Note: you will need to resolve the compilation error for *GridPane*.

1. Run, and observe the Gui. Notice:

* There is no spacing between controls nor padding around the window. We will address that shortly by applying styling.
* You can type in the text field, and press the button, but nothing happens. Of course, what we need is an *event handler* for the button, which we address in the next lab.

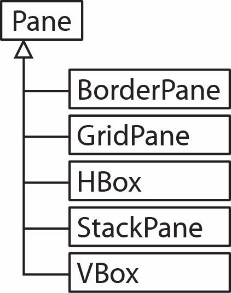
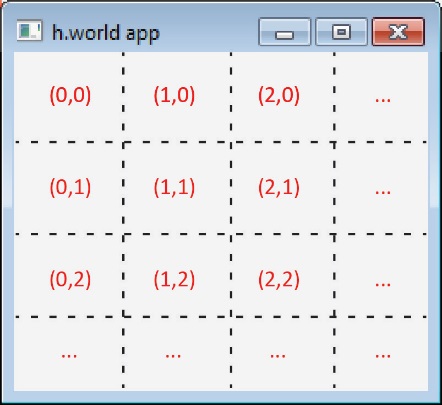
1. Go back and read the code and comments in the *try* block. We will explain more as we go along. Note that we are using a *GridPane* as our “root” container. We consider *GridPane* in the next stage of this lab.
2. Next, we will add some styling to put some space between the controls and padding around the window. In the Project Explorer, open, *application.css* and add these styles:

|  |  |
| --- | --- |
| .root {  -fx-padding: 10px;  }  **GridPane** {  -fx-hgap:10px;  -fx-vgap:10px;  } | There are two *style definitions* on the left: *.root* and *GridPane.*   1. The first, *.root* applies to the root container (which happens to be a *GridPane*). 2. The second, *GridPane* applies to all instances of a *GridPane* (in this case, there is only one). Since the *root* container is *GridPane*, the rules could be combined; however, we will keep them separate. |

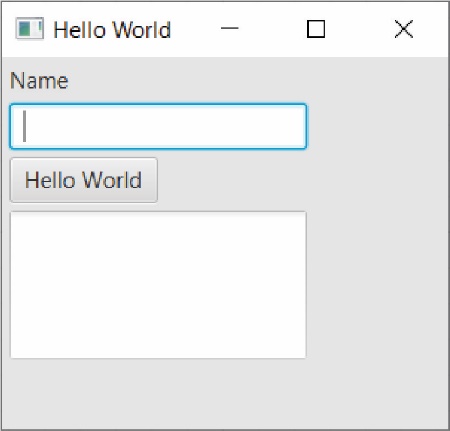
1. Run and observe the Gui. It should be similar to the one shown in step *e* above.

# The *GridPane* Class

In this stage we consider the *GridPane* class which is a way to organize the controls on a page.

1. **Read (no action required)** –
2. A *Pane* is a way to layout a Gui. A few common *Pane* classes are shown on the right. We will consider *GridPane* in this stage of the labe, and *HBox,* and *VBox* another*.*
3. A *GridPane* is a container that holds UI controls and arranges them in a grid with columns and rows.
   * A *GridPane* has regions as shown on the right. In general, the regions are not equally sized (as shown in the figure). The rules for sizing regions depend on the controls used, and their preferred sizes. This can be complicated and we will not consider sizing.
   * *GridPane* has an *add* method that accepts the control to add and the position on the grid to put the control (notice that the order is *column, row,* not the other way around!).

add(Control control, int column, int row)

1. For example, in the Gui we built above (and shown on the right), we used this code:

// Create container for controls

GridPane root = **new** GridPane();

// Add controls to container

root.add(lbl, 0, 0);

root.add(txfName, 0, 1);

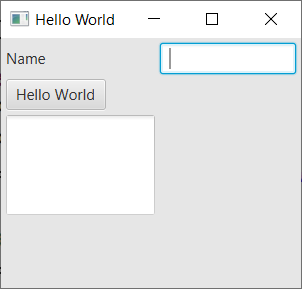
root.add(btnHelloWorld, 0, 2);

root.add(txaMessage, 0, 3);

**Verify the placement of each of the controls. You can see that we only used the first column.**

1. Save and close all open files. Then go to the Project Explorer and copy, *Main* and paste it in the *application* package. The dialog that appears will suggest giving it the name, *Main2*. Accept that.
2. Open *Main2,* in *start*, do the following:

|  |  |
| --- | --- |
| Replace this: | With this: |
| // Add controls to container  root.add(lbl, 0, 0);  root.add(txfName, 0, 1);  root.add(btnHelloWorld, 0, 2);  root.add(txaMessage, 0, 3); | // Add controls to container  root.add(lbl, 0, 0);  root.add(txfName, 1, 0);  root.add(btnHelloWorld, 0, 1);  root.add(txaMessage, 0, 2); |

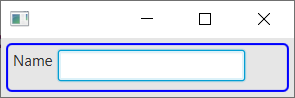
1. Run *Main2* and observe the Gui (shown on right). Verify from the code the positioning. It doesn’t look very nice! We will soon learn how to make it look at little better. However, as we said, positioning and sizing is challenging.

Feel free to experiment with positioning the controls in *Main2,* we won’t be using it anymore.

# The *HBox* Class

1. (Read, no action required)

* An *HBox* is a container that arranges the controls horizontally, one after the other, in a single row. For example, the code below renders as shown on the right (I have added a blue rectangle around the *HBox* for emphasis):

lbl = **new** Label("Name");

txfName = **new** TextField();

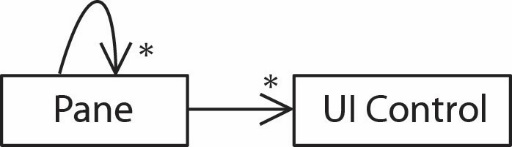
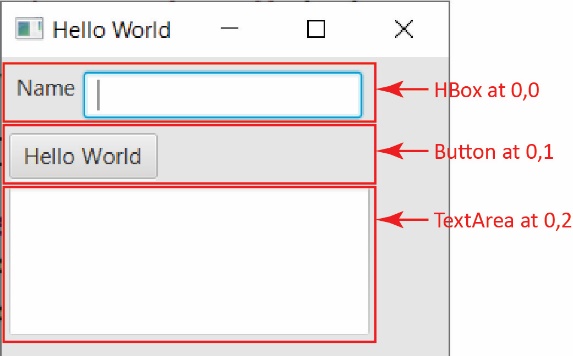
// Create HBox

HBox hBoxName = **new** HBox();

// Add controls to HBox

hBoxName.getChildren().addAll(lbl, txfName);

Note that the *HBox* has a non-obvious way to add the controls. There is not an *add* method. Instead, you call the *getChildren* method that returns a collection of the controls in the *HBox* (in this case, initially empty) and then you can call the *addAll* method to add the controls (or you can call the *add* method and add the controls one at a time.

* One very useful concept is that we can *nest panes*:we can put panes inside of other panes. This provides for infinitely more layout flexibility and organization. In other words, a *Pane* can contain any number of (sub) *Panes*, and each *Pane* can contain any number of controls as shown in the class diagram on the right.
* For example, the code below would render as shown on the right. The code below, surrounded by the box illustrates nesting the *HBox* at the top, inside the root (*GridPane*).

// Create controls

lbl = **new** Label("Name");

txfName = **new** TextField();

txaMessage = **new** TextArea();

txaMessage.setPrefHeight(100);

txaMessage.setPrefWidth(200);

btnHelloWorld = **new** Button("Hello World");

|  |
| --- |
| // Create HBox and add label and text field  HBox hBoxName = **new** HBox();  hBoxName.getStyleClass().add("hbox");  hBoxName.getChildren().addAll(lbl, txfName);  // Create root container for controls  GridPane root = **new** GridPane();  // Add HBox to Gridpane  root.add(hBoxName, 0, 0); |

// Add other controls to GridPane

root.add(btnHelloWorld, 0, 1);

root.add(txaMessage, 0, 2);

// Add GridPane (root) to Scene.

Scene scene = **new** Scene(root,300,250);

1. Next, we build the Gui shown above. Do the following:
2. Go to the Project Explorer and copy, *Main* (not *Main2*) and paste it in the *application* package. The dialog that appears will suggest giving it the name, *Main3*. Accept that.
3. Replace all code in the *try* block with (continues next page):

// Create controls

lbl = **new** Label("Name");

txfName = **new** TextField();

txaMessage = **new** TextArea();

txaMessage.setPrefHeight(100);

txaMessage.setPrefWidth(200);

btnHelloWorld = **new** Button("Hello World");

// Create HBox

HBox hBoxName = **new** HBox();

// Add style definition to HBox

hBoxName.getStyleClass().add("h\_or\_v\_box");

// Add controls to HBox

hBoxName.getChildren().addAll(lbl, txfName);

// Create root container for controls

GridPane root = **new** GridPane();

// Add HBox to Gridpane

root.add(hBoxName, 0, 0);

// Add other controls to GridPane

root.add(btnHelloWorld, 0, 1);

root.add(txaMessage, 0, 2);

// Add GridPane (root) to Scene.

Scene scene = **new** Scene(root,300,250);

// Set the title for the window

primaryStage.setTitle("Hello World");

// Code to display the Gui

scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

primaryStage.setScene(scene);

primaryStage.show();

1. Open *application.css* and add this style rule

.h\_or\_v\_box {

-fx-padding: 10px;

-fx-spacing: 10px;

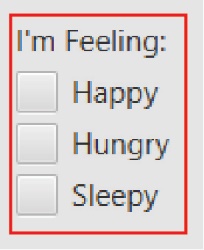
}

Note: see the highlighted line of code above. We are applying the style definition directly to a control. This contrasts with the earlier style definitions which were applied automatically. We will explain this in class.

1. Run and verify that the Gui appears as shown above.

# The *VBox* Class & *CheckBox* Class

1. (Read, no action required)

* A *VBox* is the same as an *HBox* except that the controls are arranged vertically, in a single column. For example, the code below would render as shown on the right (note that I have added a red border around the *VBox* for emphasis):

lblImFeeling = **new** Label("I'm Feeling:");

chkHappy = **new** CheckBox("Happy");

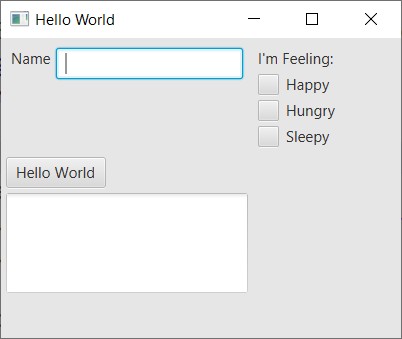
chkHungry = **new** CheckBox("Hungry");

chkSleepy = **new** CheckBox("Sleepy");

VBox vBoxFeeling = **new** VBox();

vBoxFeeling.getChildren().addAll(lblImFeeling, rbHappy, rbOk, rbSad);

* You can see that we are using the *CheckBox* class above.

1. Next, we add the *VBox* component above to the Gui we have been building and the result will be similar to that as shown on the right. Do the following:
2. Go to the Project Explorer and copy, *Main3* and paste it in the *application* package. The dialog that appears will want suggest giving it the name, *Main4*. Accept that.
3. Add these instance variables to the class:

**protected** Label lblImFeeling;

**protected** CheckBox chkHappy, chkHungry, chkSleepy;

1. Replace all code in the *try* block with the code below (continues next page). Carefully read the comments marked with “\*\*\*”.

// Create controls

lbl = **new** Label("Name");

txfName = **new** TextField();

txaMessage = **new** TextArea();

txaMessage.setPrefHeight(100);

txaMessage.setPrefWidth(200);

btnHelloWorld = **new** Button("Hello World");

// Create HBox and add label and text field

HBox hBoxName = **new** HBox();

hBoxName.getStyleClass().add("h\_or\_v\_box");

hBoxName.getChildren().addAll(lbl, txfName);

//\*\*\*LOOK at this carefully below

// Create VBox and check boxes and add

lblImFeeling = **new** Label("I'm Feeling:");

chkHappy = **new** CheckBox("Happy");

chkHungry = **new** CheckBox("Hungry");

chkSleepy = **new** CheckBox("Sleepy");

VBox vBoxFeeling = **new** VBox();

vBoxFeeling.getStyleClass().add("h\_or\_v\_box");

vBoxFeeling.getChildren().addAll(lblImFeeling, chkHappy, chkHungry, chkSleepy);

// Create root container for controls

GridPane root = **new** GridPane();

//\*\*\*LOOK at this carefully below

//\*\*\*Note how we add the hbox at 0,0, and

// the vbox at 1,0, right beside it

// Add HBox to Gridpane

root.add(hBoxName, 0, 0);

// Add VBox to Gridpane

root.add(vBoxFeeling, 1, 0);

// Add other controls to GridPane

root.add(btnHelloWorld, 0, 1);

root.add(txaMessage, 0, 2);

// Add GridPane (root) to Scene.

Scene scene = **new** Scene(root,400,300);

// Set the title for the window

primaryStage.setTitle("Hello World");

// Code to display the Gui

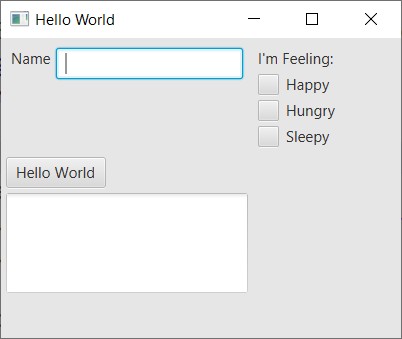
scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

primaryStage.setScene(scene);

primaryStage.show();

1. In the *start* method, change the size of the window to 400, 300 (make bigger or smaller as necessary):

Scene scene = **new** Scene(root,400,300);

1. Run and verify that the Gui appears as shown on the right.

# Modularizing Gui Construction

In this stage we consider modularizing the creation of the Gui.

1. **Read (no action required)** – Almost always there is a lot of code to build a Gui so a good idea is to modularize this code. For example, the very simple Gui above required 21 lines of code (excluding comments and blank lines). In real applications it will be hundreds to thousands of lines of code. Instead of building the Gui in the *start* method, we will have the *start* method call a method, *buildGui* that builds and returns the Gui. And, it will further modularize by calling helper methods to build various regions (Panes) of the Gui. . Here is an overview of how we will modularize:
2. We will write a helper method, *buildGui* that builds the entire Gui and returns a *Pane* object (which is the Gui)

**private** Pane buildGui() {

GridPane root = **new** GridPane();

...

**return** root;

}

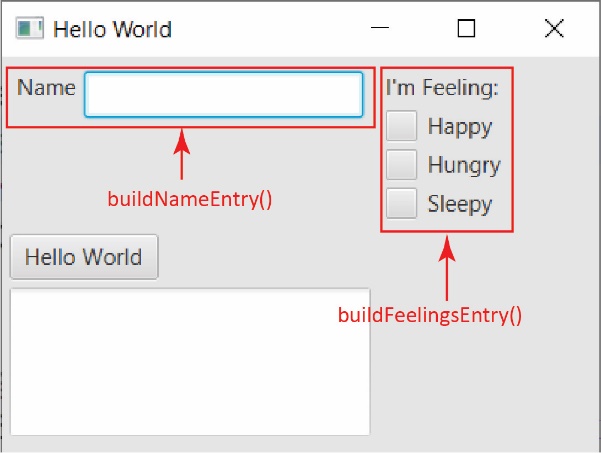
1. In *start*, we call this method and then pass the result to the *scene*.

// Create root container for controls

Pane root = buildGui();

// Add GridPane (root) to Scene.

Scene scene = **new** Scene(root,400,300);

1. As shown on the right, we modularize further by writing helper methods to build the name entry, *buildNameEntry()* and to build the check box component, *buildFeelingsEntry()*.
2. Then, in *buildGui*, we call the two methods:

**private** Pane buildGui() {

GridPane root = **new** GridPane();

Pane p = buildNameEntry();

root.add(p, 0, 0);

p = buildFeelingsEntry();

root.add(p, 1, 0);

...

**return** root;

}

1. Do the following:
2. Close all open files
3. Select *Main4.java,* copy, paste into the *application* package accepting the name, *Main5.*
4. Study the helper method below and then add it to *Main5*

**private** Pane buildNameEntry() {

lbl = **new** Label("Name");

txfName = **new** TextField();

// Create HBox and add label and text field

HBox hBoxName = **new** HBox();

hBoxName.getStyleClass().add("h\_or\_v\_box");

hBoxName.getChildren().addAll(lbl, txfName);

**return** hBoxName;

}

1. Study the helper method below and then add it to *Main5*

**private** Pane buildFeelingsEntry() {

// Create VBox and check boxes and add

lblImFeeling = **new** Label("I'm Feeling:");

chkHappy = **new** CheckBox("Happy");

chkHungry = **new** CheckBox("Hungry");

chkSleepy = **new** CheckBox("Sleepy");

VBox vBoxFeeling = **new** VBox();

vBoxFeeling.getStyleClass().add("h\_or\_v\_box");

vBoxFeeling.getChildren().addAll(lblImFeeling, chkHappy, chkHungry, chkSleepy);

**return** vBoxFeeling;

}

1. Study the top-level helper method below and then add it to *Main5*

**private** Pane buildGui() {

GridPane root = **new** GridPane();

Pane p = buildNameEntry();

root.add(p, 0, 0);

p = buildFeelingsEntry();

root.add(p, 1, 0);

// Create other controls

txaMessage = **new** TextArea();

txaMessage.setPrefHeight(100);

txaMessage.setPrefWidth(200);

btnHelloWorld = **new** Button("Hello World");

// Add other controls to GridPane

root.add(btnHelloWorld, 0, 1);

root.add(txaMessage, 0, 2);

**return** root;

}

1. Replace all the code in the *try* block with the code below. Study the code and note how simple it is to read. This is called *top-down* design: we push the details into helper methods.

// Create root container for controls

Pane root = buildGui();

// Add GridPane (root) to Scene.

Scene scene = **new** Scene(root,400,300);

// Set the title for the window

primaryStage.setTitle("Hello World");

// Code to display the Gui

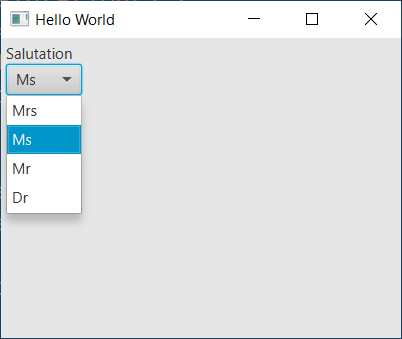
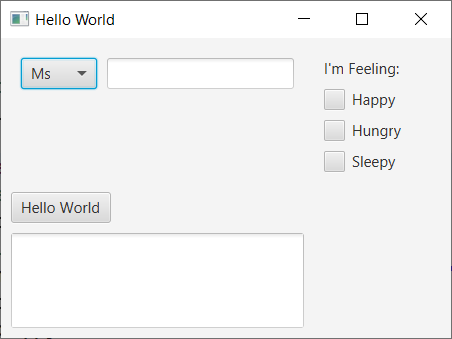
scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

primaryStage.setScene(scene);

primaryStage.show();

1. Run and observe that the Gui is the same as *Main4.*

# The *ComboBox* Control

1. **Read (no action required)** – In this stage, we are going to replace the *Label* in *buildNameEntry()* with a *ComboBox* as shown on the right. A *ComboBox* is what we frequently call a *drop-down list*: when selected it displays a list of options as shown below.
2. Do the following:
3. Close all open files.
4. Select *Main5.java,* copy, paste into the *application* package accepting the name, *Main6*.
5. In *Main6*, add the instance variable below to declare the *ComboBox,* which is a generic class, so we must specify the type of elements we will display. In this case, the generic type parameter, <String> indicates that the “list” will be displayed as strings (it could hold images or other things).

**protected** ComboBox<String> cmbSalutation;

1. Find the *buildNameEntry()* method and do the following:
2. Replace:

lbl = **new** Label("Name");

with:

cmbSalutation = **new** ComboBox<>();

cmbSalutation.getItems().addAll("Mrs", "Ms", "Mr", "Dr");

cmbSalutation.setValue("Ms");

Notes:

* The first highlighted line adds the items to display in the *ComboBox*.
* The second highlighted line sets the item in the list that is initially displayed.

1. Replace:

hBoxName.getChildren().addAll(lbl, txfName);

with:

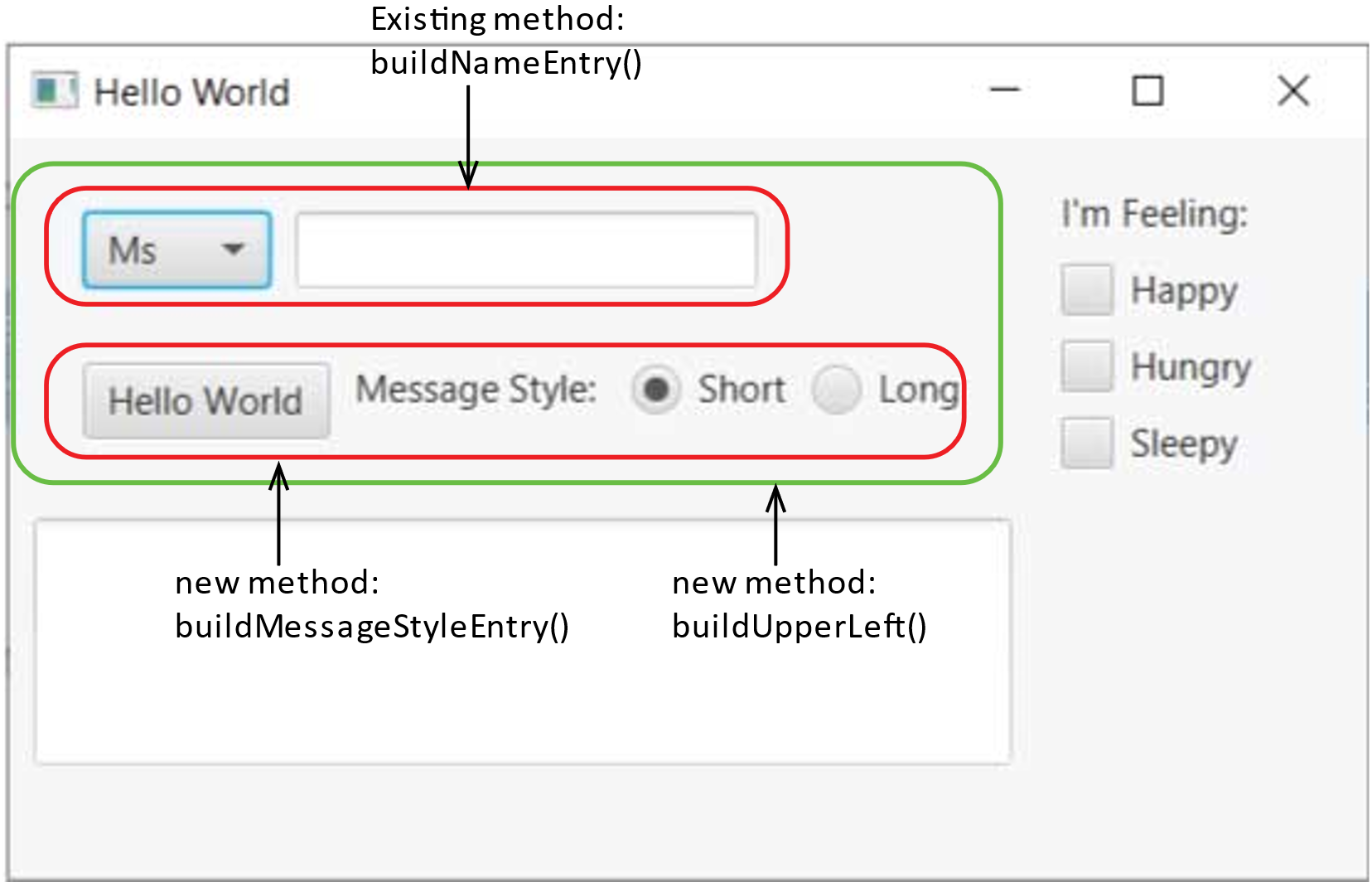
hBoxName.getChildren().addAll(cmbSalutation, txfName);

1. In the *start* method, change the size of the window to 450, 300:

Scene scene = **new** Scene(root,450,300);

1. Run and observe that the Gui is similar to what is shown above.

# The RadioButton & ToggleGroup Controls

1. **Read (no action required)** – In this stage, we are going to introduce *RadioButtons* and to our Gui as shown on the right. We will also utilize two new helper methods:

* *buildMessageStyleEntry* uses a *HBox* to place the button, label, and radio buttons.
* *buildUpperLeft* uses a *VBox* to hold the name entry and the message style entry. And, *buildUpperLeft* is placed at (0,0) in the root *GridPane*.

*RadioButtons* are a bit more complicated to deal with. Standard usage of radio buttons requires that only one button be allowed to be selected. To achieve this, we must attach a *ToggleGroup* to the *RadioButtons*.

1. Do the following:
2. Close all open files.
3. Select *Main6.java,* copy, paste into the *application* package accepting the name, *Main7*.
4. In *Main7*, add the instance variable below to declare the *ToggleGroup.* We could define the *RadioButtons* as instance variables, but we don’t have to. Instead, we will make the *RadioButtons* local variables in *buildMessageStyleEntry.* This will be explained in class.

**protected** ToggleGroup tGrpStyleChoice = new ToggleGroup();

1. Add the method below to *Main7*.

**private** Pane buildMessageStyleEntry() {

btnHelloWorld = **new** Button("Hello World");

Label lbl = **new** Label("Message Style: ");

RadioButton rbShort = **new** RadioButton("Short");

rbShort.setToggleGroup(tGrpStyleChoice);

rbShort.setSelected(**true**);

RadioButton rbLong = **new** RadioButton("Long");

rbLong.setToggleGroup(tGrpStyleChoice);

// Create HBox and add label and radio buttons

HBox hBoxName = **new** HBox();

hBoxName.getStyleClass().add("h\_or\_v\_box");

hBoxName.getChildren().addAll(btnHelloWorld, lbl, rbShort, rbLong);

**return** hBoxName;

}

Notes:

* The first highlighted line creates a *RadioButton* which displays the text: “Short”.
* The second highlighted line assigns the radio button to the *ToggleGroup*
* The third highlighted line selects this radio button for the initial display.

1. Add the method below to *Main7*.

**private** Pane buildUpperLeft() {

VBox vBox = **new** VBox();

vBox.getStyleClass().add("h\_or\_v\_box");

vBox.getChildren().addAll(buildNameEntry(), buildMessageStyleEntry());

**return** vBox;

}

1. Replace the *buildGui* method with the version below.

**private** Pane buildGui() {

GridPane root = **new** GridPane();

Pane p = buildUpperLeft();

root.add(p, 0, 0);

Pane p2 = buildFeelingsEntry();

root.add(p2, 1, 0);

// Create other controls

txaMessage = **new** TextArea();

txaMessage.setPrefHeight(100);

txaMessage.setPrefWidth(200);

root.add(txaMessage, 0, 1);

**return** root;

}

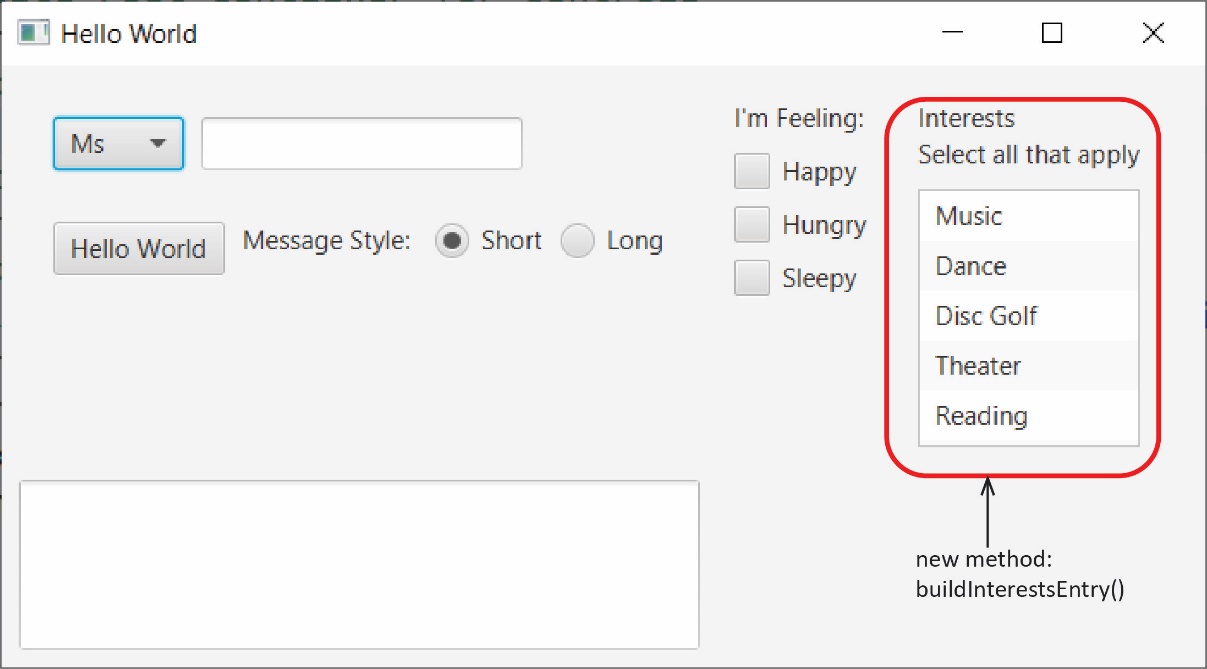
1. In the *start* method, change the size of the window to 550, 300:

Scene scene = **new** Scene(root,550,300);

1. Run and observe that the Gui is similar to what is shown above.

# The *ListView* Control

1. **Read (no action required)** – In this stage, we introduce the *ListView* control. A *ListView* displays a list of items vertically and allows for single selection, or for multiple selection.



1. Do the following:
2. Close all open files.
3. Select *Main7.java,* copy, paste into the *application* package accepting the name, *Main8*.
4. In *Main8*, add the instance variable below to declare the *ListView.*

**protected** ListView<String> lvwInterests;

1. Add the method below to *Main8*.

**private** Pane buildInterestsEntry() {

lvwInterests = **new** ListView<>();

lvwInterests.getSelectionModel().setSelectionMode(SelectionMode.***MULTIPLE***);

lvwInterests.getItems().addAll("Music", "Dance", "Disc Golf", "Theater", "Reading");

lvwInterests.setPrefHeight(150);

lvwInterests.setPrefWidth(120);

VBox vBox = **new** VBox();

vBox.getStyleClass().add("h\_or\_v\_box");

vBox.getChildren().add(**new** Label("Interests\nSelect all that apply"));

vBox.getChildren().add(lvwInterests);

**return** vBox;

}

Notes:

* The first highlighted line sets the *ListView* so that multiple items can be selected
* The second highlighted line assigns the items in the list.

1. Add these two lines to the *buildGui* method.

Pane p3 = buildInterestsEntry();

root.add(p3, 2, 0);

1. In the *start* method, change the size of the window to 700, 350:

Scene scene = **new** Scene(root,700,350);

1. Run and observe that the Gui is similar to what is shown above.

# Submission

1. Do the following
2. Zip all the folders (packages) under the *src* folder into a zip file named: *lab15\_lastname.zip*
3. Upload your zip file to the *lab15* dropbox in Blazeview.

**You are done!**

Appendix

1. n/a

.