Lab 1 – Java Editor

**Purpose:**

In this lab, you will learn to create a basic Java GUI. The following concepts will be covered:

1. Intro to Java programming
2. Java GUI development with Swing/AWT
3. Menus
4. Action Listener

**Step 1 – Create your frame**

Most GUI systems make extensive use of Object Oriented Programming. There are classes and subclasses, and you use instances of these classes to construct and manipulate your GUI. The Java GUI classes are in packages called awt and swing. We will be using parts of both of these packages.

For starters, we need a Frame – an outer most container that all your other components will reside in. You can create your Frame as follows:

JFrame myFrame = new JFrame();

myFrame.setSize(600, 400);

myFrame.setTitle("Phil Howard");

myFrame.setVisible(true);

The functions above should be fairly obvious in what they do. There is another method that you should call prior to setVisible. This function tells Swing what to do when you close the window. For our application, we want the program to end when we close the window. We do that with the following call:

myFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

Note: If your program has syntax errors, Eclipse can often fix them for you. Try clicking on the red box on the right side of the Eclipse editing window and see what options it gives you.

Run your program and experiment with different parameters to the methods to convince yourself that you know what the parameters do.

**Step 2: Create the drawing surface**

In general, JFrames are containers that hold JComponents. We would normally need a layout manager to control where the components go, but we are only going to use a single component, so we can skip the layout manager (we’ll do layouts next week).

We need an area in which we can enter and edit text. The JTextArea does this. However, we also want scroll bars, so we need to put the JTextArea inside a JScrollPane.

You can create your scrollable text area as follows:

JTextArea textArea = new JTextArea();

JScrollPane scrollPane = new JScrollPane(textArea);

textArea.setEditable(true);

We need to add the scrollPane to our frame so that it shows up when the frame is displayed. You can do this as follows:

myFrame.add(scrollPane);

Note: All of this should happen **before** the frame is made visible.

If you run your program, you should be able to enter text into the window. As your text grows (in width or length), scroll bars should be added to your frame to allow you to scroll through your text.

**Step 3 – add a menu**

We want to be able to change the font of our text as well as save and load files into our editor. We will use a menu to do this.

Create a class that extends JMenuBar. From Eclipse, select New->Class and in the dialog box, set the superclass to javax.swing.JMenuBar. We will use the constructor of our new class to create our menu. In Java, JMenuBars contain JMenus and JMenus contain JMenuItems. Let’s create our first menu and add it to the MenuBar:

JMenu menu;

menu = new JMenu(“File”);

add(menu);

Where is the add method defined? It is a member of the JMenuBar class. Since our class extends JMenuBar, the call to add is adding the menu to our menu bar.

We now need some items in the menu. You can add an item as follows:

JMenuItem menuItem;

menuItem = new JMenuItem(“Load”);

menu.add(menuItem);

menuItem = new JMenuItem(“Save”);

menu.add(menuItem);

Add menu items for “Save” and “Exit” as well. Note: You can re-use the menuItem variable. You do not need a separate variable for each menuItem.

Now let’s add our menu to our Frame. Go back to the class that defines the frame and add the following before the call to setVisible():

myFrame.setJMenuBar(new MyMenu());

You may have chosen a different name for your menu class. If so, use the name of your class.

You should now be able to run your program and see the menu in the frame. Once you see your menu, add another menu for Font and add menu items for Mono, Serif, and San-Serif. Add another menu for Style and add items for Plain, Italics, and Bold.

When you run your program, you should now be able to navigate all your menus (but selecting a menu item won’t actually do anything).

**Step 4: make the menus actually do something**

We have menus in our frame, and we can navigate through them, but they don’t actually do anything yet. To get them to do something, we need to attach an *action listener* to each menu item. An action listener is an object that “listens” for mouse (or keyboard) activity and responds to it.

You do not need to create new files for your action listeners. You can create the classes inside your menu class. You will need a class that implements the ActionListener interface. The interface requires a method with the following prototype:

public void actionPerformed(ActionEvent e)

For now, simply have your action listener print the parameter e.

Rather than creating a separate action listener class for each menu item, we could create an instance variable in our action listener and use the constructor to set the instance variable. Then, to attach the action listener, we could add the following immediately after the line the created the menu item (the following example is for the Plain menu item):

menuItem.addActionListener(new StyleListener(“Plain”));

where StyleListener is the name of the class you just defined. Now modify the actionPerformed method to also print out your instance variable so you know what menu item was selected. Add action listeners to each of your menu items and run your program. Each time you select a menu item, you should see output in the console window of Eclipse that indicates what menu item was selected.

Note: There are certainly other ways to accomplish this. If you find a different mechanism that your prefer, feel free to use it. I will give another hint a bit later.

Add ActionListeners to all items in your Font and Style menus. I’d suggest two ActionListener classes: one for the Font menu and another for the Style menu.

**Step 5: Connecting the menu to the JTextArea**

We now need to redo something that we did earlier. Instead of adding a JTextArea to your JScrollPane, create a subclass of JTextArea and add that to your JScrollPane. For now, the constructor of your new subclass should simply set the editable attribute to true.

Add protected instance variables for Font Family and Style (look at the documentation for java.awt.Font to determine the types of these variables). Add setters (you could also add getters, but we won’t be using them). Your setters should not only save the value into the appropriate instance variable, you should also call the setFont function (defined in JTextArea) to change the font for the text area.

Now we simply need to get our menus to call the correct setter and our font should change. First the hint I promised earlier: If the argument (and instance variable) for your ActionListener matches the type of the argument for the text area’s setter, then the action listener can simply pass it’s instance variable to the text area’s setter. For example, the constructor for you Style action listener should be capable of receiving the arguments: Font.PLAIN, Font.Italics, and Font.BOLD. Then your actionPerformed code can simply be:

{

textArea.SetStyle(m\_Style);

}

But that begs the question: How can the action listener know what text area to call SetStyle on? I will suggest the following solution: Place instance variables in the appropriate classes and have the constructors set them.

main calls the constructor for your menu and passes the text area

the menu constructor calls the constructors for your action listeners and passes the text area

You should now be able to get your Font and Style menus to work.

**The File Menu**

For the exit menu item, you need to simulate pressing the “X” button on the frame. This can be done with the following call:

frame.dispatchEvent(new WindowEvent(frame,

WindowEvent.WINDOW\_CLOSING));

Your Save and Load menu items should write/read the contents of the text area to a file. You can use a JFileChooser to select the file to load or to save to (see documentation for a tutorial on how to use a JFileChooser). You can use getText and setText to get and change the text of your text area. You will have to do a little research on how to do file I/O in Java.

**Completing your project**

Java has a standard way to bundle a project into a single file called a JAR (Java ARchive) file. From Eclipse, select File->Export->Jar file. Be sure the “Export Source Files” box is checked. Create your JAR file, make sure you can import it to Eclipse and that your source exists after your import. When you are sure you JAR file is correct, attach it to the Blackboard assignment.