CSCI 241 Data Structures
Winter 2018
Laboratory Exercise 6
Due: Saturday, March 3rd, (two weeks)

Objectives

1. Gain some familiarity with the development of a graphical user interface (GUI), using Java's Abstract Window Toolkit (AWT) and Swing components.

Getting Credit for Your Work

Submit your program via the Lab Exercise 6 submission on the course web site by 5:00 pm on Friday, December 8.

Developing a GUI with Java

One of the original objectives of the Java language was to provide a framework for development of application programs with a graphical user interface. One reason for Java's early popularity was the widespread use of small Java programs (applets) embedded within web pages, enabling implementation of *dynamic* web pages which could be automatically tailored to individual user's needs.

The first Java library package to support graphics was the Abstract Window Toolkit (AWT). AWT had limited capability and required heavy resource usage as it relied on the host operating system for rendering the graphical components. The Swing package was developed with all components implemented in Java and a wide range of versatile, *lightweight* components. GUI development in Java now uses both AWT and Swing components.

Developing a Simple Text Editor

Your task for this lab exercise is to develop a simple text editor. You will complete this task in seven stages, adding capability in each stage.

Stage 1

We start by developing a program that displays a window on the monitor screen. The Window has borders, a title bar and buttons to maximize, iconify (minimize) and close the window.

This simple window is provided by the Swing `JFrame` component.

Here are the steps for Stage 1:

1. Open an editor to develop the Java program `Lab6.java`.

2. For access to the Swing components:

```
import javax.swing.*;
```

3. The `Lab6` class must be a subclass of `JFrame`, so that inherits all the properties from that class.

   ```
   public class Lab6 extends JFrame
   ```

4. Write a constructor for the `Lab6` class and include the following:

   a. Call the constructor of the parent class to place a String in the title bar of the window

      ```
      super("CSCI 241 Lab6");
      ```

   b. Set the size of the window to 500 pixels across and 300 pixels high

      ```
      setSize(500, 300);
      ```

   c. Specify the operation that is to be taken when the window's close button is pressed. By default, the window just becomes invisible, but the program keeps running. We need the program to exit.

      ```
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      ```

   d. Make the window visible.

      ```
      setVisible(true);
      ```

5. Write a `main()` method which simply creates an instance of the `Lab6` class.

6. You should now be able to compile and run the program. You should find that all the standard window controls are there: you can maximize the window to fill the screen, minimize it to an icon and close it, causing the program to exit.

Stage 2

In this stage, you will add Copy, Cut and Paste buttons to a toolbar along the top edge of the window. Those buttons will not do anything until Stage 3, where we include some event-handling to specify what should happen when a button is pressed.

1. Add three private instance variables, with type `JButton`, to the `Lab6` class: `cutButton`, `copyButton` and `pasteButton`.

2. In the constructor, add a call to a method `initialize()`, which will be used to add graphical components to the window. A separate method is used simply to prevent the constructor from getting too large.

3. Write a new private void method `initialize()` and include the following:

a. Add a panel for the toolbar. The panel acts as a container for other components.

```
JPanel toolbar = new JPanel();
add(toolbar, BorderLayout.NORTH);
```

The `add()` method call adds the panel to the top part of the window (that's what `BorderLayout.NORTH` means).

Since the `BorderLayout` class is in the `java.awt` package, you need another import statement at the top of the file:

```
import java.awt.*;
```

4. Write another private void method called `addButton()` to add a button to a panel. This method will require three parameters:

   a. `JPanel` object, the `panel` to which the button is being added.

   b. `JButton` object, the `button` being added to the panel.

   c. `String` object, the `label` to appear on the button.

   Within the method:

   1. Create the button with the label:

      ```
      JButton button = new JButton(label);
      ```

   2. Add the button to the panel:

      ```
      panel.add(button);
      ```

5. In the `initialize()` method, add a call to `addButton()` for each of the three buttons.

6. Compile and run the program. You should now see three buttons at the top of the window, arranged in the order they were added to the toolbar. The buttons don't do anything yet, but that's for the next stage.

Stage 3

In this stage, you will add event-handling for clicks on those buttons. The event-handler will simply display some text in the terminal window from which you run the program, but we will enhance that later.

1. The event-handling resources are defined in the package `java.awt.event`, so add another import statement to the top of the program file:

   ```
   import java.awt.event.*;
   ```

2. In order for the program to handle button-click events, it must listen for them. To do this, the `Lab6` class must implement the `ActionListener` interface. You need to change the class heading accordingly:

   ```
   public class Lab6 extends JFrame implements ActionListener
   ```

3. When a class implements an interface, it must include a version of every method in that interface. The `ActionListener` interface has only one method:

   ```
   public void actionPerformed (ActionEvent e);
   ```

   This is the event-handler method which will be called whenever a button is pressed. The `ActionEvent` parameter contains information about the event. In particular, its `getActionCommand()` method returns the string label of the button.

   For this stage just display a message to `System.out`, indicating which button was pressed:

   ```
   System.out.println( e.getActionCommand() + " pressed");
   ```

4. To make the window listen for each button add the following line to the `addButton()` method:

   ```
   button.addActionListener(this);
   ```

   Here `button` refers to the `JButton` parameter to the `addButton()` method and `this` refers to the `Lab6` object being constructed.

5. Compile and run the program. You should see messages appear on `System.out` when you click the buttons.

Stage 4

In this stage, you will add a scrollable text area to the window.

1. Add two more instance variables to the `Lab6` class:

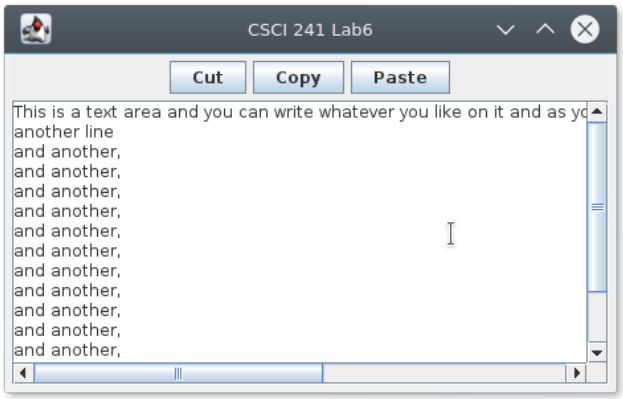
```
JScrollPane scroller;
JTextArea text;
```

The scroll pane will add horizontal and vertical scroll bars only when needed, depending on the size of the window and how much text is put in the text area.

2. In method `initialize()`, create and add the text area and scroll pane to the center of the window. The scroll pane's constructor is given the text area as a parameter, so that the text area is contained within the scroll pane.

```
text = new JTextArea();
scroller = new JScrollPane(text);
add (scroller, BorderLayout.CENTER);
```

3. Compile and run the program. You should be able to click the mouse in the text area and start typing. If you type a long enough line, a horizontal scrollbar will appear. If you type enough lines, a vertical scrollbar will appear.



Stage 5

In this stage you will make the Cut, Copy and Paste button actually work on text in the text area. This stage is very easy, because the text area has `cut()` `copy()` and `paste()` methods that work with the system clipboard.

1. In the `actionPerformed()` method use the appropriate method, depending on which button was pressed. For example:

```
if (e.getActionCommand() == "Cut")
    text.cut();
```

2. Compile and run the program. Type some text in the text area and select some of that text. You should now be able to cut, copy and paste selected text. Note: there are no shortcut keys, because your program does not know anything about such things.

Stage 6

In this stage you will enable the program to open a file and read the entire contents of that file into the text area. The file can be chosen using a standard dialog window.

1. For file input, we need to import some more packages. Add the following import statements to the top of the program file:

   ```
   import java.io.*;
   import java.nio.file.*;
   ```

2. Add an Open button to the toolbar

   a. Add another instance variable `openButton` to the `Lab6` class.

   b. Add another call to `addButton()` in `initialize()`.

   c. Add another option to the `actionPerformed()` method, so that if the Open button is pressed, method `readFile()` is called.

3. Write method `readFile()`. There are several ways to read the entire contents of a file, but the following is probably the shortest.

   a. Create a file chooser:

      ```
      JFileChooser chooser = new JFileChooser();
      ```

   b. Display the file chooser in *open-file* mode:

      ```
      int option = chooser.showOpenDialog(this);
      ```

   c. The return value, stored in `option`, indicates whether the user clicked the OK button of the dialog. If not, no further action is taken on this event:

      ```
      if (option == JFileChooser.APPROVE_OPTION)
      ```

   d. Within this if-statement, we can load the entire contents of the chosen file into the text area. Since any IO operation can throw an `IOException`, we need to use a try-catch statement:

      ```
      try {
          String filename = chooser.getName(chooser.getSelectedFile());
          text.setText(new
      String(Files.readAllBytes(Paths.get(filename))));
      }
      catch (IOException e) {
          System.out.println("Cannot read the file " + e);
      }
      ```

4. Compile and run the program. You should now be able to click the Open button, select a file and see it appear in the text area. You can edit the file, using the cut, copy and paste buttons, but cannot save your changes.

Stage 7

In this stage you will enable the program to save the entire contents of the text area to a file chosen using the standard file dialog.

1. Add a Save button to the program, along the same lines as the Open button in Stage 6.

2. Modify the `actionPerformed()` method so that when the Save button is clicked, method `writeFile()` is called.

3. Write method `writeFile()`.

    a. Create a file chooser and this time display it in *save-file* mode:

    ```
    int option = chooser.showSaveDialog(this);
    ```

    b. If the user clicks the OK button on the file dialog, write the entire contents of the text area to the chosen file:

    ```
    if (option == JFileChooser.APPROVE_OPTION) {
        try{
            String filename =
    chooser.getName(chooser.getSelectedFile());
            Files.write(Paths.get(filename),
    text.getText().getBytes());
       }
        catch (IOException e) {
            System.out.println("Cannot write to file " + e);
        }
    }
    ```

4. Compile and run the program. You should now be able to save the text from the text area. Congratulations! You have just written a text editor program.