Developing GUIs in Java and Networking Tools

Objectives:

1. To explore GUIs in Java.
2. To write Java applications that have windows, simple graphics, GUI components, and menus.
3. To use Java listeners for event handling.
4. To explore several Unix networking tools.

Preparation: Before Lab read the following chapters in *Java: How to Program* by Deitel and Deitel, sixth edition, Chapters 11 GUI Components: Part 1; 12 Graphics and Java 2D; 22 GUI Components: Part 2.

Laboratory Assignment:

This lab gets you started in writing window-based Java applications, i.e., GUIs. Also, the lab introduces several Unix networking tools.

1. The Java API:

Java is a smaller and cleaner language than C++. Chapters 1-10, 13, and 29 of the Java text cover most of the language except threads (Chapter 23). The reason why programmers like Java is the HUGE standard *Application Programming Interface* (API). Sun's API includes classes for developing *Graphical User Interfaces* (GUIs), multimedia, networking, web-based computing, database connectivity, distributed objects (RMI and CORBA), security and others. This is the fun part of programming in Java!

Take a few minutes and explore Sun's API for Java 2 (version 1.5) at URL:

`http://java.sun.com/j2se/1.5/docs/api/index.html`

Many other Java APIs are available from third party sources. You only need to search the Web with "java api".

2. A Window-based Java Application:

Below is the bare bones of a Java application that opens a window. The application extends **JFrame** which is part of the **swing** API. See pages 515 and on of Java text.

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

public class Lab3Ex2 extends JFrame {

    // A constructor
    public Lab3Ex2() {
        // super must be first line in constructor
```

```
        super("Window for Lab3Ex2");  // title for window

        // code to handle window event to allow proper "Close"
        addWindowListener(
                        new WindowAdapter() {
                          public void windowClosing(WindowEvent e) {
                             System.exit(0);
                          }
                        }
                        );

        // set size of window in pixels
        setSize(600, 400);
        // set window to visibile
        setVisible(true);
    }

    public static void main(String args[]) {

        // create object w and call the constructor
        Lab3Ex2 w = new Lab3Ex2();
    }
}
```

Since **JFrame** is the superclass to **Lab3Ex2**, the method **super()** passes the string to **JFrame**'s constructor. If you invoke a **super()** method, it **MUST** be the first line in a constructor.

Note that the structure of the program uses the common design pattern described in Lab 2. We told you we would use it!

Don't try to understand the **addWindowListener** code. It is needed for the window to close when the user selects "Close" from the standard CDE menu. If you must know, see page 1011 of Java text.

Copy the code to a file, compile and run it.

Note the use of `System.exit(0);`. Even if you don't have a window listener but use graphics in your Java program, you should have a `System.exit(0);`. Without the `System.exit(0);`, the window will close but the running java process will not quit.

The above skelton is an excellent start for any window-based Java application.

3. Adding GUI Components to the Window:

GUIs are built by adding *GUI components* to a window. Some possible components are text fields, labels, buttons, check boxes and radio buttons. See Chapter 11 of Java text.

Copy the file from Exercise 2 to a new file and in the constructor set the layout for the window to **FlowLayout**. To the window add two **JLabel** objects initialized to some text.

FlowLayout says to place the components one after the other until the components no longer fit across the window then start a new row. Layouts in Java take a little getting use to. The idea is that when a user resizes the window the components flow around to fit the new window size.

After displaying the two **JLabel** objects, adjust the size and shape of the window to see the behavior.

4. Adding JTextField and a Listener:

In this exercise we will add a **listener** to capture the text typed in a **JTextField**. **Listener**s are the way Java's API does event handling. See Chapter 11 of Java text.

Copy the file from Exercise 3 to a new file. Add a **JTextField** object of width of 20 characters to the container. Create a new **TextFieldHandler** object and add the **JTextField** object to **ActionListener**. See page 523 of Java text for information on **JTextField**. Create your own inner class to handle the event and call it **TextFieldHandler**. Display what is typed in the **JTextField** object in the shell window using **System.out.println()**.

5. Adding a Menu:

Menus are an important part of GUIs. See section 22.4 starting on page 1011 of Java text.

Copy the file from Exercise 4 into a new file. Add a menu bar with the label "File" that has an "Exit" item on it to quit the program.

When you run it, notice that the new menu bar shifts the **JLabel** and **JTextField** components down to make room.

6. Adding Simple Graphics to the Window:

Drawing lines, rectangles, and circles is easy in Java but a bit tricker if you want to draw as well as have other graphical components on the screen. See Chapter 12 of Java text.

Copy the file from Exercise 5 to a new file.

You should *avoid* the older approach of **AWT** which used the **paint()** method. We strongly urge you to use the newer and much improved **swing** approach which uses **paintComponent()**. For example, if you have a Java program with an animation, **paintComponent()** will refresh the screen automatically for you while **paint()** does not.

To use **paintComponent()**, you must create a **JPanel** object. A **JPanel** creates a drawing area for graphics. See pages 158-160 about **JPanel**. A good way to do this is to create a second file with a class that **extends** the **JPanel** class, e.g., **MyJPanel**. Inside this extended class insert your **paintComponent()** method. You will need to add **super.paintComponent(g);** as the first line of your **paintComponent()** method.

Add lines in the **paintComponent()** method to display a blue rectangle and some red text. See Chapter 12 of Java text for details.

BEFORE you create an object of **MyJPanel** and add it to the window, you need to be careful with your Java layout. In Java the default layout is **BorderLayout**, which has five regions, NORTH (top), SOUTH (bottom), EAST (right), WEST (left) and CENTER. If you don't specify when you add a component, it goes in the CENTER. If you add two components to the CENTER, the second overwrites the second. It is very easy to do this and find yourself cursing "Where in the Heck is my drawing?"

Since **JPanel**s and FlowLayout don't seem to get along, change your program to use **BorderLayout** and place the panel in the CENTER, and the two **JLabel**s and **JTextField** to NORTH, EAST, and SOUTH.

To create a line border around the panel, use the following line right after you create the panel.

```
panel.setBorder(BorderFactory.createLineBorder(Color.black));
```

When you run your **paintComponent()** method you override the **paintComponent()** method in the superclass **JPanel**. **JPanel** automatically calls the **paintComponent()** method after creating the window and after any *expose window event*. Your Java window receives an expose window event when the window is minimized (made an icon) and then maximized (icon opened). An expose event also

happens when the window is redrawn after another window has overlapped it. Try both of these situations to see what happens.

7. Unix Network Tools: Name _____

Unix has many useful networking tools. Two are **ping** and **traceroute**. Read the **man** pages for both.

ping

Try the following to **ping** the host **castor**.

```
ping castor
```

Use Control-c to stop it. What does it tell you? _____

Use **ping** to send 10 packets of 1000 bytes each. (Read **man** page for help!)

How many bytes are actually sent? _____

And how long does it take in seconds? _____

What is the smallest packet that can be sent? _____

And why? _____

What is the largest packet that can be sent? _____

Ping Bucknell's main web server (www.bucknell.edu).

What is the host name? _____

What is the IP address? _____

Ping www.acm.org For security reasons, many system administrators turn off **ping** on their servers.

traceroute

Try the following to trace the route to the host **castor**.

```
traceroute castor
```

The command **traceroute** prints the intermediate computers along the path to a remote host. Use it to trace the computers to www.acm.org. The stars mean the requests were blocked and they timed out. Unfortuantely, traceroute is not allowed off campus by ISR.

Determine the host name of a workstation next to you in the lab. Trace the route to it.

And how long does it take in seconds? _____

Note: Some network administrators disable protocols that **traceroute** and **ping** use to prevent others from obtaining detailed information about a host. Since many Denial of Service (DoS) attacks use traceroute, it is turned off.

Note: Both tools are part of the Windows Operating Systems, e.g. NT, but with different options. Type **ping** or **tracert** at the **command prompt** for the options. Note shortened spelling of **tracert**. For the **command prompt** select `Programs->Accessories->Command Prompt` from Window's **Start** menus.

Hand in

1. For Exercise 6, hand in the java code and a snapshot of the screen. Use the Linux tool `xv` to take a snapshot of the window. Print the java code using a2ps.
2. Hand in this page with answers to the questions of Exercise 7.