Labs

Lab 5.1: Enabling Messagepanel to Fire Actionevent

What is the purpose?

In this lab, you will study the example of an international clock on pages 1021-1024 and then create a program called internationalTime that displays the current date and time. The program will enable the user to select a local, time zone, date style, and time style from the combo boxes.

Here is a sample of the output:

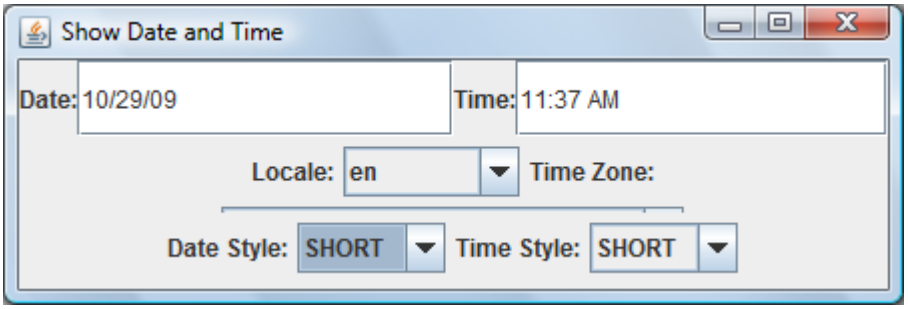

Figure 5-1-1: Sample Output 1

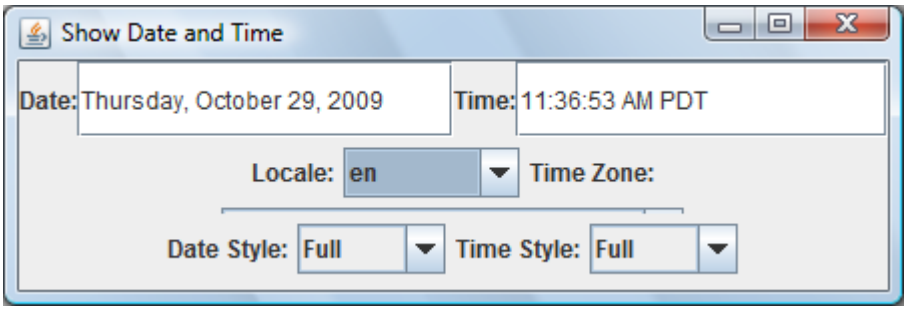

Figure 5-1-2: Sample Output 2

What are the steps?

- **Task 1**

  Procedure:

  1. Create a private method availableLocales in the Local class to return all the available locales supported in the system.
  2. Create a private method availableTimeZones in the TimeZone class to return all the available time zones supported in the system.
  3. Create a private method dateFormat in the DateFormat class to return all the available data formats: Full, Long, Medium, and Short.
  4. Create a private method timeFormat in the DateFormat class to return all the available time formats: Full, Long, Medium, and Short.
  5. Create the first subclass of JPanel to contain an instance of JTextField for displaying the date.

6. Create the second subclass of JPanel to contain an instance of JTextField for displaying the time.
7. Create the third subclass of JPanel to contain three instances of JComboBox for selecting locale, date style of time zone, and time style of time zone.
8. Complete the following sample code or create your own code:

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

public class internationalTime extends JApplet implements
ActionListener {
  private Locale[] availableLocales = Locale.getAvailableLocales();
  private String[] availableTimeZones = TimeZone.getAvailableIDs();
  private DateFormat dateFormat = DateFormat.getDateInstance();
  private DateFormat timeFormat = DateFormat.getTimeInstance();
  private Date date = new Date();
  private JTextField jtfDate = new JTextField(dateFormat.format(date));
  private JTextField jtfTime = new JTextField(timeFormat.format(date));
  private JComboBox jcboLocale = new JComboBox(availableLocales);
  private JComboBox jcboTimeZone = new JComboBox(availableTimeZones);
  private JComboBox jcboDateStyle = new JComboBox(
    new String[] {"Full", "LONG", "MEDIUM", "SHORT"});
  private JComboBox jcboTimeStyle = new JComboBox(
    new String[] {"Full", "LONG", "MEDIUM", "SHORT"});

  private Locale locale = Locale.US;
  private String timeZone = TimeZone.getDefault().getID();

  private int dateStyle = DateFormat.FULL;
  private int timeStyle = DateFormat.FULL;

  public internationalTime() {
    JPanel p1 = new JPanel(new GridLayout(1, 2));
    JPanel p1_1 = new JPanel(new BorderLayout());
    JPanel p1_2 = new JPanel(new BorderLayout());
    p1_1.add(new JLabel("Date:"), BorderLayout.WEST);
    p1_1.add(jtfDate, BorderLayout.CENTER);
    p1_2.add(new JLabel("Time:"), BorderLayout.WEST);
    p1_2.add(jtfTime, BorderLayout.CENTER);
    p1.add(p1_1);
    p1.add(p1_2);

    JPanel p2 = new JPanel();
    p2.add(new JLabel("Locale:"));
    p2.add(jcboLocale);
    p2.add(new JLabel("Time Zone:"));
    p2.add(jcboTimeZone);

    JPanel p3 = new JPanel();
    p3.add(new JLabel("Date Style:"));
    p3.add(jcboDateStyle);
```

```
    p3.add(new JLabel("Time Style:"));
    p3.add(jcboTimeStyle);

    this.setLayout(new GridLayout(3, 1));
    this.add(p1);
    this.add(p2);
    this.add(p3);

    jcboLocale.addActionListener(this);
    jcboTimeZone.addActionListener(this);
    jcboDateStyle.addActionListener(this);
    jcboTimeStyle.addActionListener(this);
  }

  public static void main(String[] args) {
    // Create a frame
    JFrame frame = new JFrame("Show Date and Time");

    // Create an instance of the applet
    JApplet applet = new internationalTime();

    // Add the applet instance to the frame
    frame.add(applet, BorderLayout.CENTER);

    // Display the frame
    frame.setSize(450, 150);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {


    dateFormat = DateFormat.getDateInstance(dateStyle, locale);
    timeFormat = DateFormat.getTimeInstance(timeStyle, locale);
    dateFormat.setTimeZone(TimeZone.getTimeZone(timeZone));
    timeFormat.setTimeZone(TimeZone.getTimeZone(timeZone));

    jtfDate.setText(dateFormat.format(new Date()));
    jtfTime.setText(timeFormat.format(new Date()));
  }
}
```

Figure 5-1-3

9. Submit your Java code and sample output to your instructor.

Did it work?

Were you able to:

- Make the interface display a different time zone?
- Make the interface display a different date style?
- Make the interface display a different time style?

Lab 5.2: Creating Custom Event Sets and Source Components

What is the purpose?

In this lab, you will develop a project by carrying out the following steps:

- Create a source component named MemoryWatch for monitoring memory. The component generates a MemoryEvent when the free memory space exceeds a specified highLimit or is below a specified lowLimit. The properties highLimit and lowLimit are customizable in MemoryWatch.

- Create the event sets named MemoryEvent and MemoryListener. MemoryEvent simply extends java.util.EventObject and contains two methods, freeMemory() and totalMemory(), which turn the free memory and the total memory of the system, respectively. The MemoryListener interface contains two handlers, sufficientMemory() and insufficentMemory(). The sufficientMemory() method is invoked when the free memory space exceeds the specified high limit, and insufficientMemory() is invoked when the free memory space is less than the specified low limit. The free memory and total memory in the system can be obtained using the code:

```
Runtime runtime = Runtime.getRuntime();
    runtime.freeMemory();
    runtime.totalMemory();
```

- Develop a listener component that displays the free memory and the total memory and indicates whether the memory is sufficient or insufficient when a MemoryEvent occurs.

- Make the listener an applet with a main() method to run in stand-alone mode.

Here is a sample of the output:

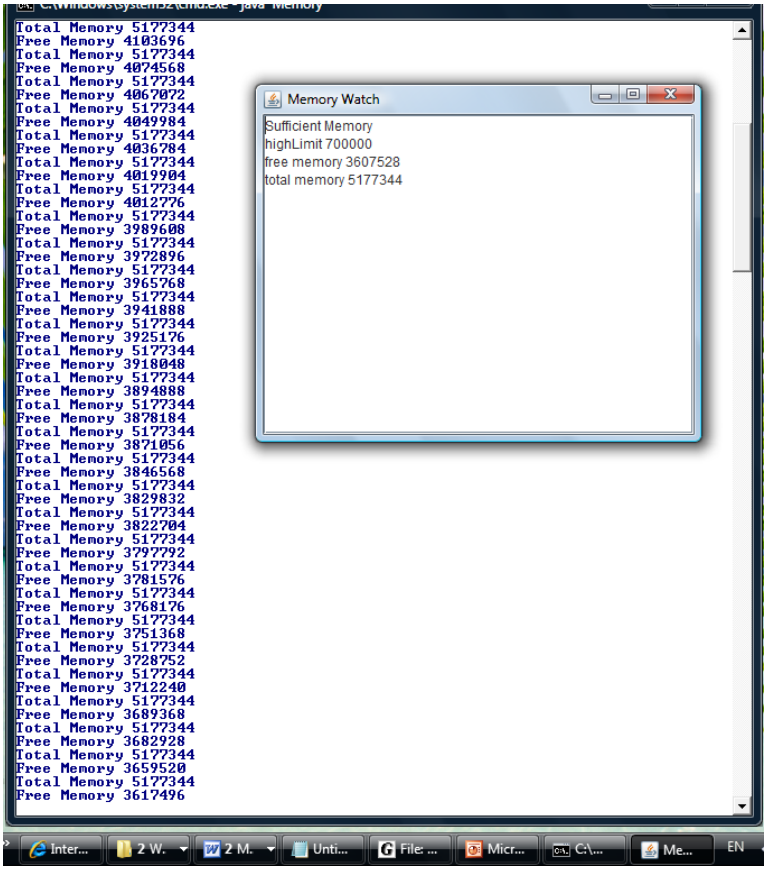

Figure 5-2-1

What are the steps?

- **Task 1**

  Procedure:

  1. Create an applet file named Memory.java that can display the free memory and the total memory of a computer.
  2. Complete the following code or create your own code:

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

public class Memory extends JApplet implements MemoryListener {
  boolean isStandalone = false;
  private MemoryWatch memoryWatch1 = new MemoryWatch();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTextArea jTextArea1 = new JTextArea();

  /** Initialize the applet */
  public void init() {
     this.setSize(new Dimension(400,300));
    jTextArea1.setText("jTextArea1");
```

```
  memoryWatch1.addMemoryListener(new MemoryListener() {
      public void sufficientMemory(MemoryEvent e) {
      }

      public void insufficientMemory(MemoryEvent e) {
        memoryWatch1_insufficientMemory(e);
      }
    });

    memoryWatch1.addMemoryListener(new MemoryListener() {
      public void sufficientMemory(MemoryEvent e) {
        memoryWatch1_sufficientMemory(e);
      }

      public void insufficientMemory(MemoryEvent e) {
      }
    });
    this.getContentPane().add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(jTextArea1, null);
  }

  //Main method
  public static void main(String[] args) {
    Memory applet = new Memory();
    applet.isStandalone = true;
    JFrame frame = new JFrame();
    frame.setTitle("Memory Watch");
    frame.getContentPane().add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    frame.setSize(400,320);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) / 2, (d.height
- frame.getSize().height) / 2);
    frame.setVisible(true);
  }

  public void sufficientMemory(MemoryEvent e) {
    jTextArea1.setText("Sufficient Memory\n");
    jTextArea1.append("highLimit " + memoryWatch1.getHighLimit()+"\n");
    jTextArea1.append("free memory " + e.freeMemory() + "\n");
    jTextArea1.append("total memory " + e.totalMemory() + "\n");
  }

  public void insufficientMemory(MemoryEvent e) {
    jTextArea1.setText("Insufficient Memory\n");
    jTextArea1.append("lowLimit " + memoryWatch1.getLowLimit()+"\n");
    jTextArea1.append("free memory " + e.freeMemory() + "\n");
    jTextArea1.append("total memory " + e.totalMemory() + "\n");
  }

 void memoryWatch1_insufficientMemory(MemoryEvent e) {
    insufficientMemory(e);
  }
```

```
  void memoryWatch1_sufficientMemory(MemoryEvent e) {
    sufficientMemory(e);
  }
}

interface MemoryListener extends java.util.EventListener {
  // Handler for sufficient memory
  public void sufficientMemory(MemoryEvent e);

  // Handler for insufficient memory
  public void insufficientMemory(MemoryEvent e);
}

class MemoryEvent extends java.util.EventObject {
  private Runtime runtime = Runtime.getRuntime();

  public MemoryEvent(Object o) {
    super(o);
  }

  public long freeMemory() {
    return runtime.freeMemory();
  }

  public long totalMemory() {
    return runtime.totalMemory();
  }
}

class MemoryWatch implements Runnable {
  private int highLimit = 700000;
  private int lowLimit = 200000;

  private Runtime runtime = Runtime.getRuntime();

  private Thread thread;
  private transient Vector memoryListeners;

  public MemoryWatch() {
    thread = new Thread(this);
    thread.start();
  }

  public void run() {
    while (true) {
      try {
        Thread.sleep(1000);
      }
      catch (InterruptedException e)
      {  }

      System.out.println("Total Memory " + runtime.totalMemory());
   System.out.println("Free Memory " + runtime.freeMemory());

      if (runtime.freeMemory() > highLimit) {
        MemoryEvent e = new MemoryEvent(this);
    fireSufficientMemory(e);
```

```
      }

      if (runtime.freeMemory() < lowLimit) {
        MemoryEvent e = new MemoryEvent(this);
        fireInsufficientMemory(e);
      }
    }
  }

  public static void main(String[] args) {
    MemoryWatch memoryWatch1 = new MemoryWatch();
  }

  public void setHighLimit(int newHighLimit) {
    highLimit = newHighLimit;
  }

  public int getHighLimit() {
    return highLimit;
  }

  public void setLowLimit(int newLowLimit) {
    lowLimit = newLowLimit;
  }

  public int getLowLimit() {
    return lowLimit;
  }

  public synchronized void removeMemoryListener(MemoryListener l) {


  }

  public synchronized void addMemoryListener(MemoryListener l) {


  }

  protected void fireSufficientMemory(MemoryEvent e) {


  }

  protected void fireInsufficientMemory(MemoryEvent e) {


  }
}
```

Figure 5-2-2

3. Submit your Java code and sample output to your instructor.

Did it work?

Were you able to cause the applet to display:

- Whether the system has sufficient or insufficient memory?
- The free memory of the system?
- The maximum memory limit of the system?
- The total memory of the system?