AP CS Unit 11: Graphics and Events

This packet shows how to create programs with a graphical interface in a way that is consistent with the approach used in the Elevens program.

Copy the following two classes and run the program.

```java
public class RunMyFrame1 {
    public static void main(String[] args) {
        MyFrame1 mf = new MyFrame1();
        mf.display();
    }
}
```

```java
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;

public class MyFrame1 extends JFrame {
    public MyFrame1(){
        setDefaultCloseOperation( EXIT_ON_CLOSE );
        setTitle( "HEY" );
        JPanel jp = new JPanel();
        jp.setBackground( Color.RED );
        jp.setPreferredSize( new Dimension( 300, 200 ) );
        jp.setLayout( null );
        getContentPane().add( jp );
        pack();
    }
    public void display() {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                setVisible(true);
            }
        });
    }
}
```

For an explanation about the code in the display method.
http://stackoverflow.com/questions/4908824/explain-what-the-following-code-does
Ex 1. Create a program that displays a green background that is 300 pixels long and 100 pixels wide. Make the frame non-resizable. This method is in a superclass of the JFrame class.

**JButtons and JLabels and Positioning.** Copy and run the following program. It consists of two files: RunDemoButton and DemoButtonFrame.

```java
public class RunDemoButton {
    public static void main(String[] args) {
        DemoButtonFrame f = new DemoButtonFrame();
        f.display();
    }
}

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

public class DemoButtonFrame extends JFrame {
    public DemoButtonFrame(){
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setTitle( "Button!" );
        JPanel jp = new JPanel();
        jp.setBackground( Color.WHITE );
        jp.setPreferredSize( new Dimension( 150, 150 ) );
        jp.setLayout( null );
        JButton button = new JButton( "Click Here" );
        button.setBounds( 25, 75, 100, 50 );  // x, y coordinates of the upper left
                                                  // followed by width, height (all in pixels)
        jp.add( button );
        getContentPane().add( jp );
        pack();
    }

    public void display() {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                setVisible(true);
            }
        });
    }
}
```

What is this? `jp.setLayout( null );`  
Java typically uses Layout Manager objects that are responsible for managing the positions and sizes of different components. For example, the GridLayout manager will keep all components the same size and in a grid arrangement even if the frame/panel is resized. Calling the `setLayout` method with an argument of null means there is no layout manager and we are responsible for positioning everything. Use the `setBounds` method to position and size any components added to a JPanel.
The **Coordinate System** used by java is similar (but not identical) to the one used in algebra and geometry.

Each panel has its own coordinate system where the upper-left hand corner is the origin.

The top edge of the panel is the positive x-axis and the left edge of the panel is the positive (not the negative) y-axis.

Coordinates can have negative values.

**Ex2.** Create the program shown to the right.

- The panel is initially 300 pixels wide and 400 pixels high.
- Button 1 is 100 by 100 pixels in size and has an orange background.
- Button 2 is 200 by 200 pixels in size and has a magenta background.
- The buttons should be positioned as shown. Button 2 is initially centered in the panel though if the panel is resized the panel does not stay centered.

After you complete the program, experiment with your code to answer the following questions.

a) Can the x and/or coordinates be negative?

b) What happens if the height or width of the button is not big enough to display the text?

c) What happens if the width and/or height is set to a negative value?

d) What happens if you position the buttons so that they overlap? Which one is displayed in front of the other? Does that order always stay that way?
Here’s a figure that shows some of the subclasses of JComponent. Any method that JComponent has, such as setBackground, are inherited by JButton, JPanel, and so on.

![Subclasses of JComponent](image)

**Ex 3.** Create the program shown to the right. There are two JLabels, two JTextFields, and one JButton inside one JPanel (which has a yellow background). You pick the sizes and positions to get your program to look as close to this as possible.

![Example Program](image)

**Ex 4.** Write a program that displays 10 panels inside of a white panel that is inside the frame. The 10 panels should be randomly positioned and have random widths and heights and random colors.

You can create a random color using the following code.
```java
int red = (int)(256*Math.random());
int green = (int)(256*Math.random());
int blue = (int)(256*Math.random());
Color c = new Color( red, green, blue );
```
Events. An event can broadly be defined as something that happens: For example, a mouse is clicked or a window is resized. In the java event model, there are generally three objects involved.

- Event Source. This is the object that was effected. For example, the button that is clicked or the window that was resized.
- Event Object. This object contains information about the event. For example, a MouseEvent contains information about where the mouse was at the time of the event.
- Event Listener. This is the object that is responsible for responding to the event.

Copy and run the following program.

```java
public class RunDemoAction {
    public static void main(String[] args) {
        DemoEventFrame f = new DemoEventFrame();
        f.display();
    }
}
```

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

```java
public class DemoEventFrame extends JFrame {
    private JLabel lbl = new JLabel( "Random Number" );
    private JButton btnEven = new JButton( "Display Random Even Number" );
    private JButton btnOdd = new JButton( "Display Random Odd Number" );

    public DemoEventFrame (){
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setTitle( "Buttons That Work" );
        JPanel jp = new JPanel();
        jp.setBackground( Color.WHITE );
        jp.setPreferredSize( new Dimension( 250, 150 ) );
        jp.setLayout( null );
        ButtonListener spy = new ButtonListener();
        btnEven.addActionListener( spy );
        btnOdd.addActionListener( spy );
        btnEven.setBounds( 10, 10, 230, 30 );
        btnOdd.setBounds( 10, 60, 230, 30 );
        lbl.setBounds( 10, 110, 150, 30 );
        jp.add( btnEven );
        jp.add( btnOdd );
        jp.add( lbl );
        getContentPane().add( jp );
        pack();
    }

    private class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if( e.getSource() == btnEven ) {
                int random = (int) (Math.random() * 100);
                if( random % 2 == 0 )
                    lbl.setText( "Random Even Number: " + random );
                else
                    lbl.setText( "Random Odd Number: " + random );
            } else if( e.getSource() == btnOdd ) {
                int random = (int) (Math.random() * 100);
                if( random % 2 != 0 )
                    lbl.setText( "Random Odd Number: " + random );
                else
                    lbl.setText( "Random Even Number: " + random );
            }
        }
    }
}
```

Something New!

spy is the object that will respond when either button is clicked.
public void display() {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            setVisible(true);
        }
    });
}

private class ButtonListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        if (e.getSource().equals(btnEven)) {
            int num = (int)( 5*Math.random() ) + 1;
            num = 2 * num;
            lbl.setText( num + "" );
        } else { // must be btnOdd
            lbl.setText( "I should write this" );
        }
    }
}

ButtonListener is an inner class. It is declared inside the DemoEventFrame class. Why do this? Because:
1) The only class that ever uses this ButtonListener class is the DemoEventFrame class. By limiting its scope, no one else can use it.
2) An inner class can directly access the private instance variables, and methods, of its enclosing class.

After you get this program running, modify the actionPerformed method so that when the user clicks the Odd Button that a random odd number between 1 and 9 appears.

In the above program, there are two Event Sources – btnEven and btnOdd. When they are clicked, the JVM generates ActionEvent objects which are sent to the one EventListener object that we instantiated.

To link the Event Sources to an Event Listener, we used these statements
    btnEven.addActionListener( spy );
    btnOdd.addActionListener( spy );
The addActionListener has one parameter of type ActionListener. ActionListener is an interface with one method: actionPerformed.
Ex 5. Create the program to the right. There are three JLabels, two JTextFields, and two JButtons. You will need to use the getText and setText methods.

To convert a string into an integer use Integer.parseInt.

Ex 6. Every time you click the button, the background color of the panel changes to another random color. Use the code from exercise 4 to generate the random color.

Ex 7. Create the program shown above. There is an array of 26 buttons, a label, and another button. Whenever one of the 26 top buttons are pressed, that letter is displayed in the label. The figures show the panel after five buttons have been pressed. Click the bottom button to clear the label.

Here’s an outline for the loop that creates the 26 buttons.

```java
ButtonListener ear = new ButtonListener();
for ( int k = 0; k < 26; k++ ){
    char ch = (char)(k + 65);
    String letter = ch + "";
    // create the button
    // calculate the x and y coordinates based on the value of k, then call setBounds
    // add the action listener
    // add the button to the panel
}
```

In the actionPerformed method you may want to do this:

```java
JButton b = (JButton) e.getSource(); // must cast because getSource returns an Object
String text = b.getText();
```
Simple Animation. Copy and run the following program. This uses a Timer object to generate action events at regular intervals. Explanations for certain statements can be found on the next page.

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

public class DemoTimerFrame extends JFrame {
    private JLabel lbl = new JLabel( "0" );
    private JButton btnStart = new JButton( "Start" );
    private JButton btnStop = new JButton( "Stop" );
    private JPanel jp = new JPanel();
    private int count = 0;
    private javax.swing.Timer timer;

    public DemoTimerFrame(){
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setTitle( "Timer" );
        jp.setBackground( Color.WHITE );
        jp.setPreferredSize( new Dimension( 150, 150 ) );
        jp.setLayout( null );
        btnStart.setBounds( 10, 5, 130, 30 );
        lbl.setBounds( 60, 50, 90, 30 );
        btnStop.setBounds( 10, 100, 130, 30 );
        Font f = new Font("SansSerif", Font.BOLD, 36);
        lbl.setFont( f );
        btnStart.addActionListener( new Respond() );
        btnStop.addActionListener( new Respond() );
        timer = new javax.swing.Timer( 100, new TimerListener() );
        jp.add( btnStart );
        lbl.add( btnStop );
        getContentPane().add( jp );
        pack();
    }

    public void display() {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                setVisible(true);
            }
        });
    }

    // continued on the next page
```
private class TimerListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        count++;
        lbl.setText( count + "" );
    }
}

private class Respond implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if ( e.getSource().equals( btnStart ) )
            timer.start();
        else
            timer.stop();
    }
}

Explanations.

A private javax.swing.Timer timer;

Java has two different Timer classes; one is in the java.util package and one is in the javax.swing package. This syntax makes it clear to the compiler that we want the Timer class from the javax.swing package. We will not use the other Timer class. If you are curious why Java has two Timer classes: http://stackoverflow.com/questions/15556646/why-are-there-two-timer-classes-in-javaone-under-javax-swing-one-under-java-ut

B Font f = new Font("SansSerif", Font.BOLD, 36);
    lbl.setFont( f );

This demonstrates how to change the font type of a JLabel. You could use the same approach on the buttons. Look up the Font class for more information.

C new javax.swing.Timer( 100, new TimerListener() );

The constructor for the Timer class has two parameters. First, the amount of time, in milliseconds, between events. Second, an object that will respond to the Action Events that a Timer object generates.

D private class TimerListener implements ActionListener

A Timer object generates Action events at regular intervals therefore the object that responds to those events must implement the ActionListener interface. I used a separate inner class for the timer events because I didn’t think the same object should handle action events from buttons and action events from a timer.

E timer.start();
    timer.stop();

The start method causes the Timer object to start generating action events at the specified interval. The stop method causes the Timer object to stop generating action events.
Ex 8. Create the program shown to the right. Start and stop the timer with the two buttons. Whenever the timer generates an action event, the program should generate three random colors:
- one color for the background of the panel
- one color for the background of the two buttons
- one color for the text on the buttons (use the setForeground method)

Also, change the font type on the buttons and use the following method to the class that handles the action events from the timer.

```java
private Color random(){
    int red = (int)(256*Math.random());
    int green = (int)(256*Math.random());
    int blue = (int)(256*Math.random());
    return new Color( red, green, blue );
}
```

Ex 9. Create a program which simulates a simple slot machine. The idea is the user starts with 100 tokens. If they get three matching numbers then they win 30 more tokens. If any two numbers match they get 10 tokens. If the numbers are all different, they lose 10 tokens.

Comments/Hints.
- There are two buttons and four labels.
- Create an instance variable that represents how many tokens the user currently has.
- Clicking the Start button causes the timer object to start generating action events.
- The object that responds to the timer generated action events should display random integers (1 to 7 inclusive) in the labels.
- When the Stop button is clicked, stop the timer, determine if the user won or lost, update the number of tokens, and display an appropriate message.
- If the user clicks on the Stop button at the very start of the program, before any numbers have been displayed in the labels, there will be an error.
- The number of tokens may go negative.
Ex 10. Write a program that displays a message that scrolls across the window from the left to the right. When it is completely off the right side, it should slide in from the right.

Comments/Hints.
- There is one JLabel in the JPanel; there are no JButton objects.
- The animations should start as soon as the program starts. The animation does not stop.
- If the size of the window is changed, the program should still work correctly.
- Create an instance variable that represents the x coordinate of the label’s upper left hand corner.
- Every time the timer fires off an action event, increase the x coordinate of the label’s upper left hand corner. Then call setBounds again.
- There is a method, getWidth(), that returns the current width of a component. Use this to figure out if the label has gone past the right side of the panel. If so, reset the x coordinate so that the label will reappear on the left side.

Ex 11. In this program that is an array of labels. Each label contains a single letter. As soon as the program starts these letters start dropping down and then reappear at the top of the window and drop again. Each letter drops at its own random speed.

Comments/Hints.
- Your array of labels should have a length of at least five elements.
- Create an array of ints to store the current y coordinate of each label’s corner.
- Create a second array of ints that stores random numbers between 5 and 10. This array should be the same length as the first two arrays.
- Every time the timer fires off an action event, increase the y coordinate of the label’s upper left hand corner using the random numbers in the second array of ints. Then call setBounds again.
- There is a method, getHeight(), that returns the current width of a component. Use this to figure out if a label has gone below the bottom of the panel. If so, reset the y coordinate so that the label will reappear on the left side.