Java Programming Tutorial

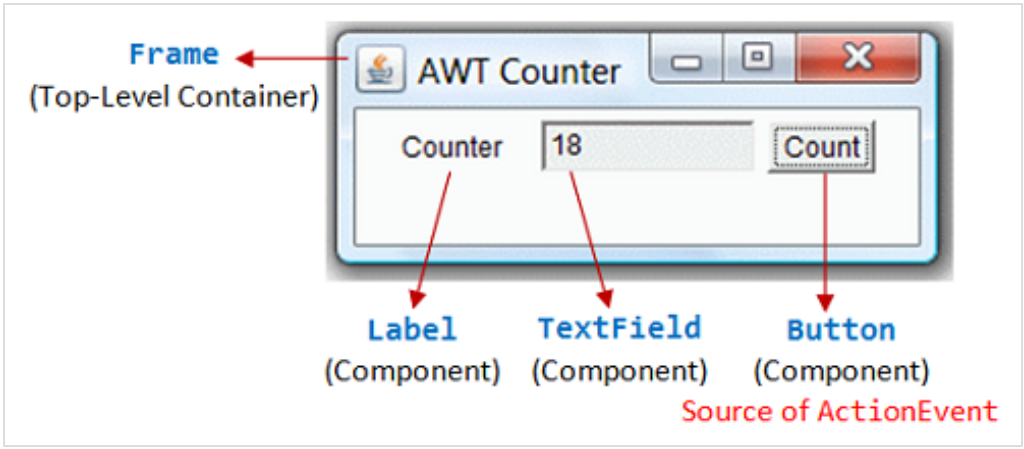
Graphics Programming Exercises

TABLE OF CONTENTS (HIDE)

1. AWT GUI Applications/Applets

1.1 Exercise: AWTCounter

Write an AWT GUI application (called AWTCounter) as shown in the Figure. Each time the "Count" button is clicked, the counter value shall increase by 1.



The program has three components:

1. a Label "Counter";
2. a non-editable TextField to display the counter value; and
3. a Button "Count".

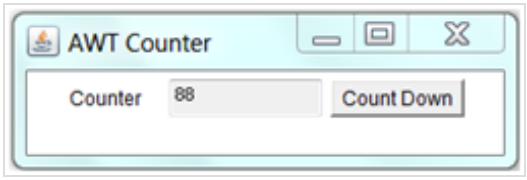
The components are placed inside a container Frame, arranged in FlowLayout.

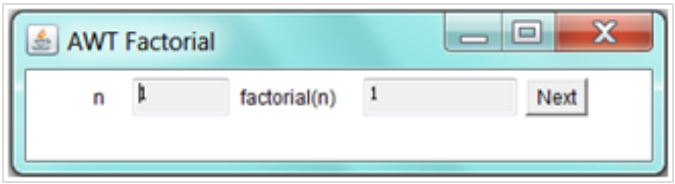
```
1  import java.awt.*;          // Using AWT containers and components
2  import java.awt.event.*;    // Using AWT events and listener interfaces
3
4  // An AWT GUI program inherits the top-level container Frame
5  public class AWTCounter extends Frame implements ActionListener {
6     private Label lblCount;     // declare component Label
7     private TextField tfCount;  // declare component TextField
8     private Button btnCount;    // declare component Button
9     private int count = 0;      // counter's value
10
11    // Constructor to setup UI components
```

```
12      public AWTCounter () {
13         setLayout(new FlowLayout());
14            // "this" Frame sets layout to FlowLayout, which arranges
15            //  Components from left-to-right, then top-to-bottom.
16
17         lblCount = new Label("Counter"); // allocate Label instance
18         add(lblCount);                   // "this" Frame adds Label
19
20         tfCount = new TextField(count + "", 10); // allocate
21         tfCount.setEditable(false);       // read-only
22         add(tfCount);                     // "this" Frame adds tfCount
23
24         btnCount = new Button("Count");   // allocate Button instance
25         add(btnCount);                    // "this" Frame adds btnCount
26         btnCount.addActionListener(this);
27            // btnCount is a source that fires ActionEvent when clicked.
28            // The source add "this" object as a listener, which provides
29            //  the ActionEvent handler called actionPerformed().
30            // Clicking btnCount invokes actionPerformed().
31
32         setSize(250, 100);       // "this" Frame sets initial size
33         setTitle("AWT Counter"); // "this" Frame sets title
34         setVisible(true);        // show "this" Frame
35      }
36
37      // ActionEvent handler - Called back when the button has been clicked.
38      @Override
39      public void actionPerformed(ActionEvent evt) {
40         ++count;                     // incrase the counter value
41         tfCount.setText(count + ""); // display on the TextField
42                                      // setText() takes a String
43      }
44
45      // The entry main() method
46      public static void main(String[] args) {
47         // Invoke the constructor by allocating an anonymous instance
48         new AWTCounter();
49      }
50   }
```

You have to use control-c, or "close" the CMD shell, or hit the "terminate" button on Eclipse's Console to terminate the program. This is because the program does not process the `WindowEvent` fired by the "window-close" button.

TRY:

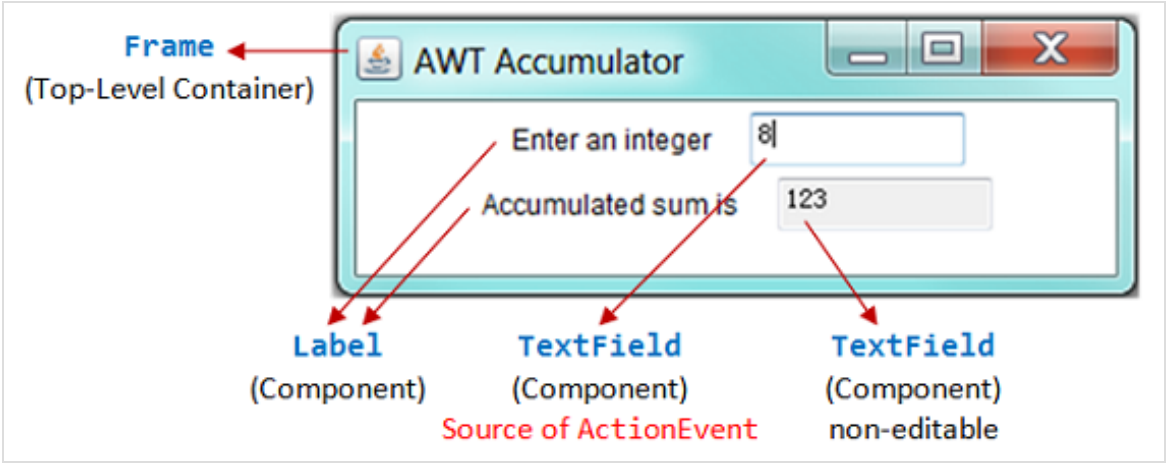




1. Modify the program (called `AWTCounterDown`) to count down, with an initial value of 88, as shown.
2. Modify the program (called `AWTFactorial`) to display n and factorial of n, as shown. Clicking the "Next" button shall increase n by 1. n shall begin at 1.

1.2 Exercise: `AWTAccumulator`

Write an AWT GUI application called



`AWTAccumulator`, which has four components:

1. a `Label` "Enter an integer and press enter";
2. an input `TextField`;
3. a `Label` "The accumulated sum is", and
4. a protected (read-only) `TextField` for displaying the accumulated sum.

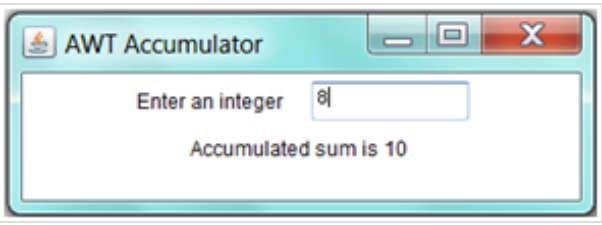
The four GUI components are placed inside a container `Frame`, arranged in `FlowLayout`. The program shall accumulate the numbers entered into the input `TextField`, and display the accumulated sum on the display `TextField`.

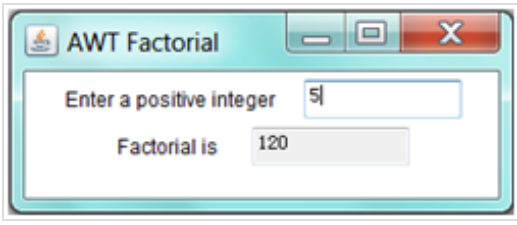
```
1  import java.awt.*;       // Using AWT containers and components
2  import java.awt.event.*; // Using AWT events and listener interfaces
3
4  // A GUI program inherits the top-level Container Frame
5  public class AWTAccumulator extends Frame implements ActionListener {
6     private Label lblInput;     // declare input Label
7     private Label lblOutput;    // declare output Label
8     private TextField tfInput;  // declare input TextField
9     private TextField tfOutput; // declare output display TextField
10    private int numberIn;  // the number entered
11    private int sum = 0;   // the accumulated sum, init to 0
12
13    // Constructor to setup the UI
14    public AWTAccumulator() {
15       setLayout(new FlowLayout()); // "this" Frame sets to FlowLayout
16
17       lblInput = new Label("Enter an integer"); // allocate
18       add(lblInput);  // "this" Frame adds the Label
19
20       tfInput = new TextField(10);  // allocate
21       add(tfInput);   // "this" Frame adds the TextField
22
23       tfInput.addActionListener(this);
24          // tfInput is a source that fires ActionEvent when entered.
25          // The source add "this" object as a listener, which provides
26          //  an ActionEvent handler called actionPerformed().
27          // Hitting enter key on tfInput invokes actionPerformed().
28
29       lblOutput = new Label("Accumulated sum is");  // allocate
30       add(lblOutput);  // "this" Frame adds Label
31
32       tfOutput = new TextField(10); // allocate
33       tfOutput.setEditable(false);  // read-only
34       add(tfOutput);   // "this" Frame adds TextField
```

```
35
36          setTitle("AWT Accumulator"); // "this" Frame sets title
37          setSize(350, 120);    // "this" Frame sets initial size
38          setVisible(true);     // "this" Frame shows
39       }
40
41       // The entry main() method
42       public static void main(String[] args) {
43          // Invoke the constructor by allocating an anonymous instance
44          new AWTAccumulator();
45       }
46
47       // ActionEvent handler - Called back when enter key was hit on TextField.
48       @Override
49       public void actionPerformed(ActionEvent evt) {
50          numberIn = Integer.parseInt(tfInput.getText());
51                   // get the String entered, convert to int
52          sum += numberIn;      // accumulate numbers entered into sum
53          tfInput.setText("");  // clear input TextField
54          tfOutput.setText("" + sum); // display sum on the output TextField
55       }
56    }
```

TRY:

1. Modify the program (called `AWTAccumulatorLabel`) to display the sum using a `Label` instead of a protected `TextField`, as shown.



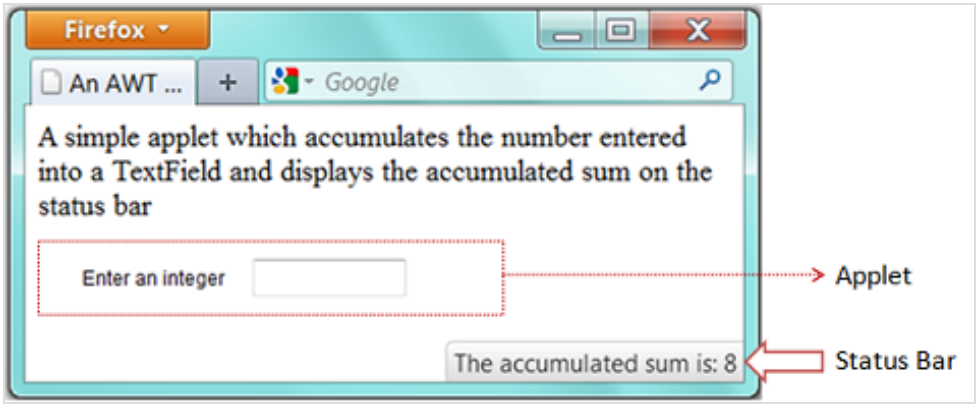


2. Modify the program (called `AWTFactorialTextField`) to display the factorial of the input number, as shown.

1.3 Exercise: `AWTAccumulatorApplet`

An Java *applet* is a graphics program run inside a browser. Write a Java applet (called `AWTAccumulatorApplet`) which contains:

1. a `label` "Enter an integer:",
2. a `TextField` for user to enter a number.
3. The applet shall accumulate all the integers entered and show it on the status bar of the browser's window.




```
1  import java.applet.Applet;
2  import java.awt.*;          // Using AWT containers and components
3  import java.awt.event.*;    // Using AWT events and listener interfaces
4
```

```
// An applet extends java.applet.Applet
public class AWTAccumulatorApplet extends Applet implements ActionListener {
   private TextField tfInput;   // input TextField
   private int number;   // number entered
   private int sum = 0;  // accumulated sum

   // init() runs when the applet is loaded. Setup the UI.
   public void init() {
      add(new Label("Enter an integer"));

      tfInput = new TextField(10);
      add(tfInput);
      tfInput.addActionListener(this);
         // Hitting enter key on tfInput invokes actionPerformed()
   }

   // ActionEvent handler - Called back when enter key was hit on TextField.
   public void actionPerformed( ActionEvent evt) {
      number = Integer.parseInt(evt.getActionCommand());
         // getActionCommand() returns the String entered.
      sum += number;
      tfInput.setText("");  // Clear input TextField
      showStatus("The accumulated sum is: " + sum);
         // show the sum on the status bar of the browser's window
   }
}
```

Note that:

- An applet extends from `java.applet.Applet`, whereas a standalone GUI application extends from `java.awt.Frame`. You cannot `setTitle()` and `setSize()` on Applet.
- Applet uses `init()` to create the GUI, while standalone GUI application uses the constructor (invoked in `main()`).

HTML codes: `AWTAccumulatorApplet.html`

Applet runs inside a web browser. A separate HTML script (says `AWTAccumulatorApplet.html`) is required, which uses an <applet> tag to embed the applet as follows:

```
<html>
<head>
  <title>An AWT Applet</title>
</head>
<body>
  <p>A simple applet which accumulates the number entered into
  a TextField and displays the accumulated sum on the status bar</p>
  <applet code="AWTAccumulatorApplet.class" width="300" height="60">
  </applet>
</body>
</html>
```

TRY:

1. Modify the applet to run the "Counter" application (as in `AWTCounter`).
2. Modify the applet to run the "Factorial" application (as in `AWTFactorial`).

2. Event-Handling

2.1 Exercise: Window-Close

Modify the `AWTCounter` program (called `AWTCounterWithClose`) to process the "close-window" button.

```
public class AWTCounterWithClose extends Frame
      implements ActionListener, WindowListener {
   ......

   // Constructor
   public AWTCounterWithClose () {
      ......
      addWindowListener(this);
        // "this" Frame fires WindowEvent.
        // "this" class is also the WindowEvent listener
      ......
   }
   ......

   // WindowEvent handlers
   @Override
   public void windowClosing(WindowEvent e) {
      System.exit(0);  // terminate the program
   }

   @Override
   public void windowOpened(WindowEvent e) { }

   @Override
   public void windowClosed(WindowEvent e) { }

   @Override
   public void windowIconified(WindowEvent e) { }

   @Override
   public void windowDeiconified(WindowEvent e) { }

   @Override
   public void windowActivated(WindowEvent e) { }

   @Override
   public void windowDeactivated(WindowEvent e) { }
}
```

3. Inner Class - Named and Annoymous

Compared with the `AWTCounter`, the following programs `AWTCounterNamedInnerClass` and `AWTCounterAnonymousInnerClass` use "named inner classes" and "anonymous inner classes", respectively, as the `ActionEvent` listener instead of "`this`" object.

A named inner class as the event listener:

`AWTCounterNamedInnerClass.java`

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class AWTCounterNamedInnerClass extends Frame {
5     // This class is NOT the listener, hence, it does not implement ActionListener
6
7     private TextField tfCount;
```

```
   private int count = 0;

   // Constructor to setup the UI
   public AWTCounterNamedInnerClass () {
      setLayout(new FlowLayout());  // "this" Frame sets to FlowLayout
      add(new Label("Counter"));    // anonymous Label
      tfCount = new TextField(count + "", 10);
      tfCount.setEditable(false);   // read-only
      add(tfCount);                 // this Frame adds tfCount

      Button btnCount = new Button("Count");
      add(btnCount);                // this Frame adds btnCount

      // Construct an anonymous instance of inner class BtnListener as
      //  listener to the source btnCount
      btnCount.addActionListener(new BtnListener());

      setSize(250, 100);
      setTitle("AWT Counter");
      setVisible(true);    // show it
   }

   public static void main(String[] args) {
      new AWTCounterNamedInnerClass();
   }

   // A named inner class to be used as listener of ActionEvent
   // This inner class can access private variables of the outer class
   private class BtnListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent e) {
         ++count;
         tfCount.setText(count + "");
      }
   }
}
```

Explanation

- An inner class called `BtnListener` is defined, to be used as listener for the `ActionEvent` fired by the `Button btnCount`. Since `BtnListener` is an `ActionEvent` listener, it has to implement `ActionListener` interface and provide implementation to the `actionPerformed()` method declared in the interface.
- Although instance variables `tfCount`, `count` are `private`, the inner class `BtnListener` has access to them. This is the sole reason why an inner class is used instead of an ordinary outer class.
- An anonymous instance of `BtnListener` is constructed via statement "`new BtnListener()`". The `Button btnCount` registers this anonymous instance as a listener to its `ActionEvent` via `btnCount.addActionListener(new BtnListener())`.

An anonymous Inner class as the event listener:

AWTCounterAnonymousInnerClass.java

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

public class AWTCounterAnonymousInnerClass extends Frame {
   // This class is NOT the listener, hence, it does not implement ActionListener

   private TextField tfCount;
```

```
    private int count = 0;

    // Constructor to setup the UI
    public AWTCounterAnonymousInnerClass () {
       setLayout(new FlowLayout());  // "this" Frame sets to FlowLayout
       add(new Label("Counter"));    // anonymous Label
       tfCount = new TextField(count + "", 10);
       tfCount.setEditable(false);   // read-only
       add(tfCount);                 // this Frame adds tfCount

       Button btnCount = new Button("Count");
       add(btnCount);                // this Frame adds btnCount

       // Construct an anonymous instance of an anonymous class as
       //  listener to the source btnCount
       btnCount.addActionListener(new ActionListener() {
          @Override
          public void actionPerformed(ActionEvent e) {
             ++count;
             tfCount.setText(count + "");
          }
       });

       setSize(250, 100);
       setTitle("AWT Counter");
       setVisible(true);   // show it
    }

    public static void main(String[] args) {
       new AWTCounterAnonymousInnerClass();
    }
 }
```

Explanation

- An anonymous instance of an anonymous inner class is defined via

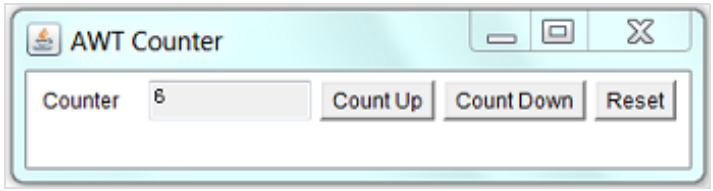
```
new ActionListener() { ... }
```

- The compiler creates an anonymous inner class called `$n` (where `n` is a running number of inner classes) as follows:

```
class $n implements ActionListener() { .... }
new $n()
```

Notes: Observe the output files produced by the Java compiler. Named inner class is named "`OuterClassName$InnerClassName.class`" and anonymous inner class is named "`OuterClassName$n.class`".

TRY:



1. Modify all the earlier programs to use (i) a named inner class; (ii) an anonymous inner class as the `ActionEvent` listener.
2. Modify `AWTCount` (called `AWTCounter3Buttons`) to include two additional buttons for counting down and reset the count value. Use (i) "`this`" class as listener for all the 3 buttons; (ii) use one named inner class as listener for all the 3 buttons; (iii) use an anonymous inner class as listener for each button.

Hints for (i) and (ii): `You can use event.getActionCommend()` to retrieve the label of the button that has fired the event.

```
@Override
public void actionPerformed(ActionEvent e) {
   String btnLabel = e.getActionCommand();
       // event.getActionCommand() returns the button's label
   if (btnLabel.equals("Count Up")) {
      ......
   } else if (btnLabel.equals("Count Down")) {
      ......
   } else {
      ......
   }
   ......
}
```

4. Swing GUI Applications

4.1 Exercise: Converting from AWT to Swing

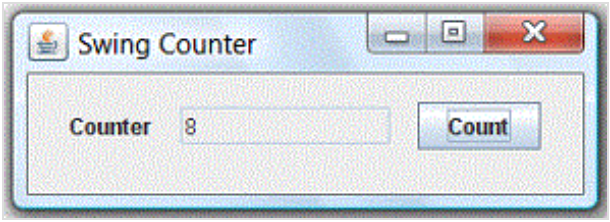
Convert all the previous AWT exercises (`AWTCounter`, `AWTAccumulator`, `AWTFactorial`, etc.) to Swing applications (called `SwingCounter`, `SwingAccumulator`, `SwingFactorial`, etc.).

Notes:

- Swing Components are kept in package `javax.swing`. They begin with a prefix "`J`", e.g., `JButton`, `JLabel`, `JFrame`.
- Swing Components are to be added onto the `ContentPane` of the top-level container `JFrame`. You can retrieve the `ContentPane` via method `getContentPane()` from a `JFrame`.

```
Container cp = getContentPane();  // of JFrame
cp.setLayout(......);
cp.add(......);
```

For example, `SwingCounter.java`:




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;  // Using Swing components and containers
4
5  // A Swing application extends javax.swing.JFrame
6  public class SwingCounter extends JFrame {
7     private JTextField tfCount;
8           // Use Swing's JTextField instead of AWT's TextField
9     private int count = 0;
10
11    public SwingCounter () {
12       // Get the content pane of top-level container Jframe
13       // Components are added onto content pane
14       Container cp = getContentPane();
15       cp.setLayout(new FlowLayout());
```

```
16
17          cp.add(new JLabel("Counter"));
18          tfCount = new JTextField(count + "", 10);
19          tfCount.setEditable(false);
20          tfCount.setHorizontalAlignment(JTextField.RIGHT);
21          cp.add(tfCount);
22
23          JButton btnCount = new JButton("Count");
24          cp.add(btnCount);
25          btnCount.addActionListener(new ActionListener() {
26            @Override
27             public void actionPerformed(ActionEvent e) {
28                ++count;
29                tfCount.setText(count + "");
30             }
31          });
32
33          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34             // Exit program if Jframe's close-window button clicked
35          setSize(300, 100);
36          setTitle("Swing Counter");
37          setVisible(true);    // show it
38       }
39
40       public static void main(String[] args) {
41          // Recommended to run the GUI construction in
42          //  Event Dispatching thread for thread-safet operations
43          SwingUtilities.invokeLater(new Runnable() {
44             @Override
45             public void run() {
46                new SwingCounter(); // Let the constructor does the job
47             }
48          });
49       }
50    }
```

4.2 Exercise: `SwingAdder`

Write a Swing application called `SwingAdder` as shown. The "ADD" button adds the two integers and display the result. The "CLEAR" button shall clear all the text fields.

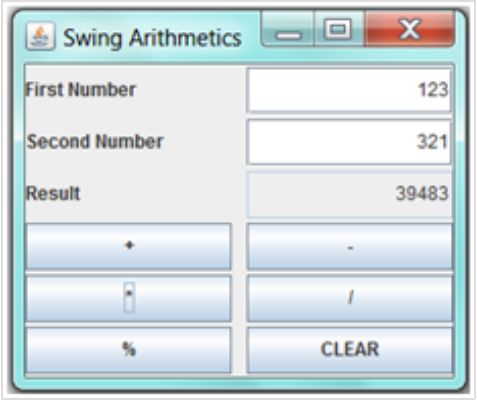
Hints: Set the content-pane to 4x2 GridLayout. The components are added from left-to-right, top-to-bottom.

```
import javax.swing.*;   // Swing components
import java.awt.*;
import java.awt.event.*;

// Swing application extends from javax.swing.JFrame
public class SwingAdder extends JFrame {

   private JTextField tfNumber1, tfNumber2, tfResult;
   private JButton btnAdd, btnClear;
   private int number1, number2, result;

   public SwingAdder() {       // Constructor
      // Swing components must be added to the ContentPane.
      Container cp = getContentPane();
      // Set this Container to grid layout of 4 rows and 2 columns
      cp.setLayout(new GridLayout(4, 2, 10, 3));

      // Components are added from left-to-right, top-to-bottom
      cp.add(new JLabel("First Number "));     // at (1, 1)
```

```
      tfNumber1 = new JTextField(10);
      tfNumber1.setHorizontalAlignment(JTextField.RIGHT);
      cp.add(tfNumber1);                          // at (1, 2)
      .......
      .......

      btnAdd = new JButton("ADD");
      cp.add(btnAdd);                            // at (4, 1)
      btnAdd.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent evt) {
            number1 = Integer.parseInt(tfNumber1.getText());
            ......
         }
      });


      btnClear = new JButton("CLEAR");
      cp.add();                                   // at (4, 2)
      btnClear.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent evt) {
            ......
         }
      });

      setDefaultCloseOperation(EXIT_ON_CLOSE); // for the "window-close" button
      setTitle("Swing Adder");
      setSize(300, 170);
      setVisible(true);
   }

   // The entry main() method
   public static void main(String[] args) {
      // For thread safety, use the event-dispatching thread to construct UI
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            new SwingAdder(); // Let the constructor does the job
         }
      });
   }
}
```

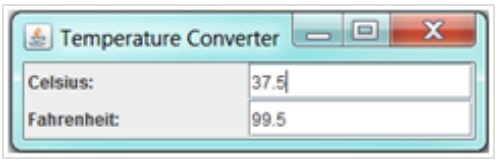
TRY:

1. Modify the above exercise (called `SwingArithmetics`) to include buttons "`+`", "`-`", "`*`", "`/`", "`%`" (remainder) and "`CLEAR`" as shown.



4.3 Exercise: `SwingTemperatureConverter`

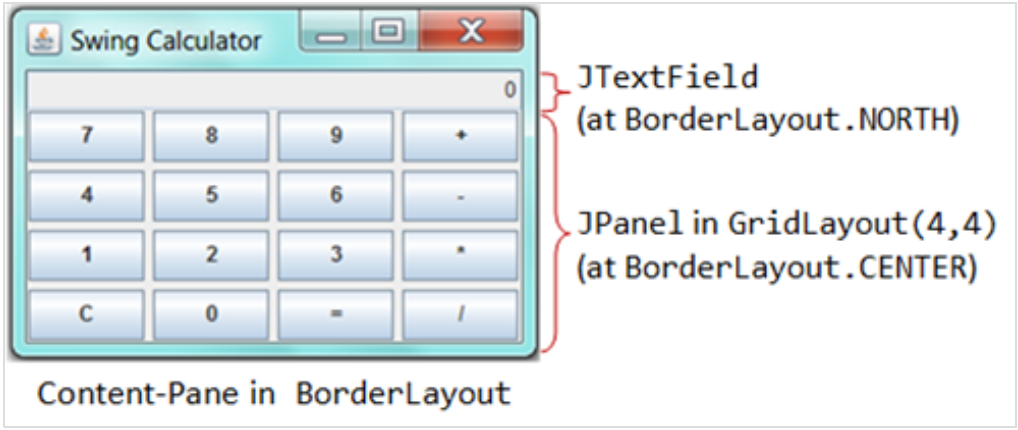
Write a GUI program called `SwingTemperatureConverter` to convert temperature values between Celsius and Fahrenheit. User can enter either the Celsius or the Fahrenheit value, in floating-point number.



Hints: To display a floating-point number in a specific format (e.g., 1 decimal place), use the `static` method `String.format()`, which has the same form as `printf()`. For example, `String.format("%.1f", 1.234)` returns String "`1.2`".

4.4 Exercise: `SwingCalculator`

Implement a simple calculator (called `SwingCalculator`) as shown.



Hints:

- Set the `ContentPane` to `BorderLayout`. Add a `JTextField` (`tfDisplay`) to the `NORHT`. Add a `JPanel` (`panelButtons`) to the `CENTER`. Set the `JPanel` to `GridLayout` of 4x4, and add the 16 buttons.
- All the number buttons can share the same listener as they can be processed with the same codes. Use `event.getActionCommand()` to get the label of the button that fires the event.
- The operator buttons "`+`", "`-`", "`*`", "`/`", "`%`" and "`=`" can share a common listener.
- Use an anonymous inner class for "`C`" button.
- You need to keep track of the *pervious* operator. For example in "1 + 2 =", the current operator is "=", while the *previous* operator is "+". Perform the operation specified by the previous operator.

```
import javax.swing.*;   // Using Swing components
import java.awt.*;
import java.awt.event.*;

// Swing application extends from javax.swing.JFrame
public class SwingCalculator extends JFrame {

   private JTextField tfDisplay;
   private int result = 0;          // the result so far
   private String numberInStr = ""; // the number entered as String
   private char previousOpr = ' ';  // the previous operator
   private char currentOpr = ' ';   // the current operator

   // Constructor to setup the UI
   public SwingCalculator() {
      // TODO: Setup the UI
      // ......
   }

   // Number buttons listener (inner class)
   class NumberBtnListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent evt) {
         numberInStr += evt.getActionCommand();
```

```
            tfDisplay.setText(numberInStr);
         }
      }

      // Operator buttons listener (inner class)
      class OprBtnListener implements ActionListener {
         @Override
         public void actionPerformed(ActionEvent evt) {
            previousOpr = currentOpr;  // save
            currentOpr = evt.getActionCommand().charAt(0);
            // TODO: Processing logic
            // ......
         }
      }
   }
```

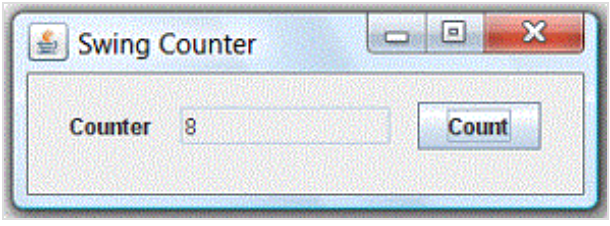
4.5 Exercise: Sudoku

Produce the display for a Sudoku game, as shown. It consists of 9x9 `JTextField`s arranged in a `GridLayout`.

Hints: Use the following variables.

```
// Game board
int[][] board = new int[9][9];
JTextField[][] tfDisplays = new JTextField[9][9];

// Puzzle to solve - 0 are the empty cells
int[][] puzzle = {{5, 3, 4, 6, 7, 0, 9, 1, 2},
                  {6, 7, 2, 1, 9, 5, 3, 4, 0},
                  {1, 9, 0, 3, 4, 2, 5, 6, 7},
                  {0, 5, 9, 7, 6, 1, 4, 2, 3},
                  {4, 2, 6, 0, 5, 3, 7, 9, 1},
                  {7, 1, 3, 9, 2, 4, 0, 5, 6},
                  {9, 6, 1, 5, 3, 7, 2, 0, 4},
                  {2, 0, 7, 4, 1, 9, 6, 3, 5},
                  {3, 4, 5, 2, 0, 6, 1, 7, 9}};
```

4.6 Exercise: Writing a GUI program using NetBeans GUI Builder



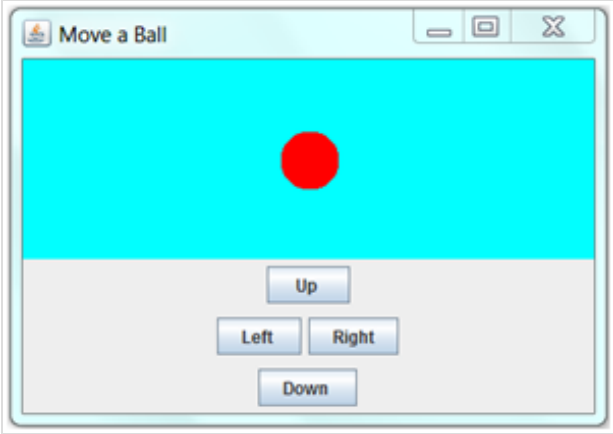
Write the `SwingCounter` program using NetBeans' GUI builder. Read "Writing Java GUI (AWT/Swing) Application in NetBeans".

Study the code generated by NetBeans, in particularly, the `initComponents()` method.

5. Custom Graphics

5.1 Exercise: `MoveABall`

Study the "Move-a-line" program. Modifying the program to move a ball in response to up/down/left/right buttons, as well as the 4 arrow keys, as shown.



REFERENCES & RESOURCES

Latest version tested: JDK 1.7.3_03
Last modified: May, 2012

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg)
| HOME