Java Programming Tutorial

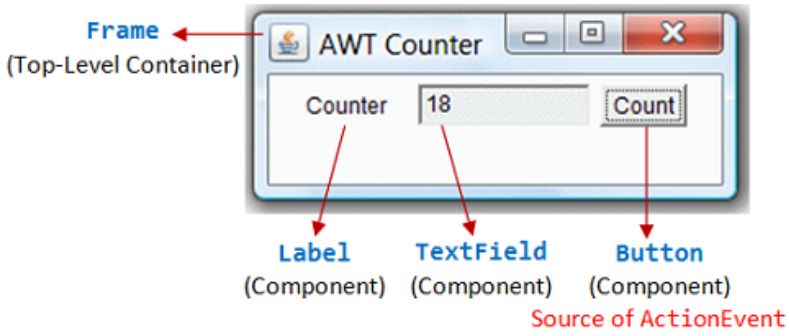
Graphics Programming Exercises

TABLE OF CONTENTS (HIDE)

1. AWT GUI Applications
   - 1.1 Ex: AWTCounter
   - 1.2 Ex: AWTAccumulator
   - 1.3 Ex: AWTAccumulatorApplet (
2. Event-Handling
   - 2.1 Ex: WindowEvent and Windowl
3. Inner Class - Named and Anony
   - 3.1 Ex: Named Inner Class
   - 3.2 Ex: Anonymous Inner Class
4. Swing GUI Applications
   - 4.1 Ex: Converting from AWT to S
   - 4.2 Ex: SWingAdder
   - 4.3 Ex: SwingTemperatureConver
   - 4.4 Ex: SwingCurrencyConverter
   - 4.5 Ex: SwingNumberGuess
   - 4.6 Ex: SwingPhoneApp
   - 4.7 Ex: SwingCalculator
   - 4.8 Ex: SwingLoginPanel
   - 4.9 Ex: SwingLock
   - 4.10 Ex: Using Eclipse/NetBeans G
5. Custom Graphics
   - 5.1 Ex: MoveABall
   - 5.2 Ex: BouncingBall

1. AWT GUI Applications

1.1 Ex: 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 `java.awt.Label` "`Counter`";
2. a non-editable `java.awt.TextField` to display the counter value; and
3. a `java.awt.Button` "`Count`".

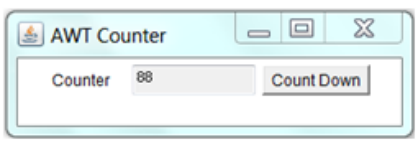
The components are placed inside the top-level AWT container `java.awt.Frame`, arranged in `FlowLayout`.

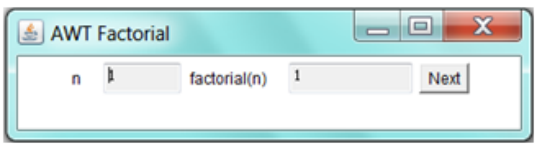
```
1  import java.awt.*;        // Using AWT container and component classes
2  import java.awt.event.*;  // Using AWT event classes and listener interfaces
3
4  // An AWT program inherits from the top-level container java.awt.Frame
5  public class AWTCounter extends Frame {
6     private Label lblCount;    // Declare a Label component
7     private TextField tfCount; // Declare a TextField component
8     private Button btnCount;   // Declare a Button component
9     private int count = 0;     // Counter's value
10
11    // Constructor to setup GUI components and event handlers
12    public AWTCounter () {
13       setLayout(new FlowLayout());
14          // "super" Frame, which is a Container, sets its layout to FlowLayout to arrange
15          // the components from left-to-right, and flow to next row from top-to-bottom.
16
17       lblCount = new Label("Counter");  // construct the Label component
18       add(lblCount);                    // "super" Frame container adds Label component
19
20       tfCount = new TextField(count + "", 10); // construct the TextField component with initial text
```

```
21          tfCount.setEditable(false);       // set to read-only
22          add(tfCount);                     // "super" Frame container adds TextField component
23
24          btnCount = new Button("Count");   // construct the Button component
25          add(btnCount);                    // "super" Frame container adds Button component
26
27          BtnCountListener listener = new BtnCountListener();
28          btnCount.addActionListener(listener);
29             // "btnCount" is the source object that fires an ActionEvent when clicked.
30             // The source object adds an instance of BtnCountListener as an ActionEvent listener,
31             //   which provides an ActionEvent handler called actionPerformed().
32             // Clicking "Count" button calls back actionPerformed().
33
34          setTitle("AWT Counter");  // "super" Frame sets its title
35          setSize(300, 100);        // "super" Frame sets its initial window size
36
37          // For inspecting the Container/Components objects
38          // System.out.println(this);
39          // System.out.println(lblCount);
40          // System.out.println(tfCount);
41          // System.out.println(btnCount);
42          setVisible(true);         // "super" Frame shows
43          // System.out.println(this);
44          // System.out.println(lblCount);
45          // System.out.println(tfCount);
46          // System.out.println(btnCount);
47       }
48
49       // The entry main() method
50       public static void main(String[] args) {
51          // Invoke the constructor to setup the GUI, by allocating an instance
52          AWTCounter app = new AWTCounter();
53             // or simply "new AWTCounter();" for an anonymous instance
54       }
55
56       // Define an inner class to handle the "Count" button-click
57       private class BtnCountListener implements ActionListener {
58          // ActionEvent handler - Called back upon button-click.
59          @Override
60          public void actionPerformed(ActionEvent evt) {
61             ++count; // Increase the counter value
62             // Display the counter value on the TextField tfCount
63             tfCount.setText(count + ""); // Convert int to String
64          }
65       }
66    }
```

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.

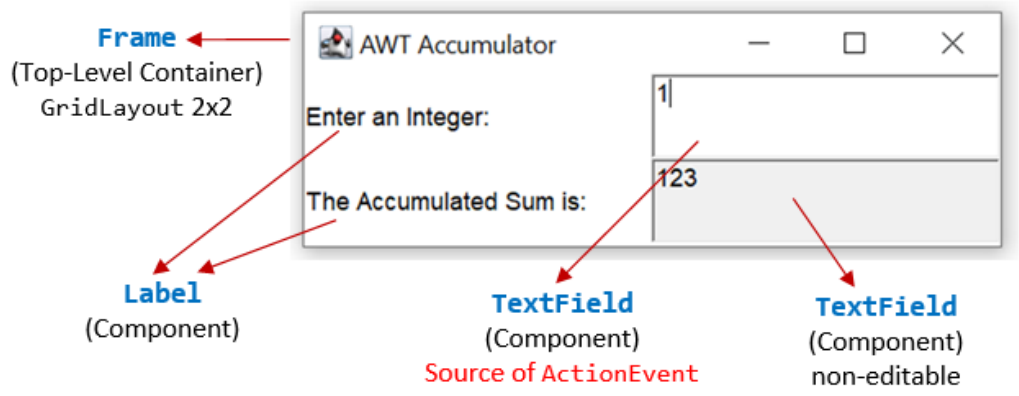
TRY:





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 Ex: `AWTAccumulator`



Write an AWT GUI application called `AWTAccumulator`, which has four components:

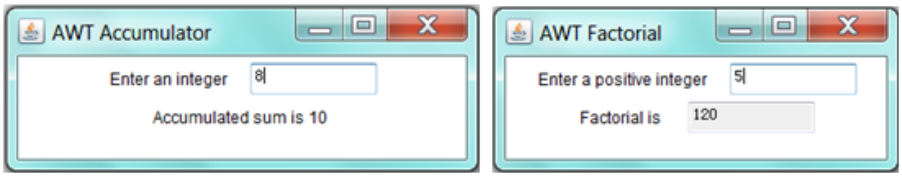
1. a `java.awt.Label` "Enter an integer and press enter";
2. an input `java.awt.TextField`;
3. a `java.awt.Label` "The accumulated sum is", and
4. a protected (read-only) `java.awt.TextField` for displaying the accumulated sum.

The four GUI components are placed inside a container `java.awt.Frame`, arranged in `GridLayout` of 2 rows 2 columns. 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 container and component classes
2  import java.awt.event.*;  // Using AWT event classes and listener interfaces
3
4  // An AWT GUI program inherits (customized) from the top-level container
5  //   java.awt.Frame
6  public class AWTAccumulator extends Frame {
7     //private Label lblInput;   // Declare input Label (to use anonymous)
8     //private Label lblOutput;  // Declare output Label (to use anonymous)
9     private TextField tfInput;  // Declare input TextField
10    private TextField tfOutput; // Declare output TextField
11    private int sum = 0;        // Accumulated sum, init to 0
12
13    // Constructor to setup the GUI components and event handlers
14    public AWTAccumulator() {
15       setLayout(new GridLayout(2, 2));
16          // "super" Frame (Container) sets layout to GridLayout of 2 rows 2 columns.
17
18       add(new Label("Enter an Integer: "));  // "super" Frame adds an anonymous Label
19
20       tfInput = new TextField(10); // Construct TextField
21       add(tfInput);                // "super" Frame adds TextField
22
23       tfInput.addActionListener(new TFInputListener());
24          // "tfInput" is the source object that fires an ActionEvent upon entered.
25          // The source add an anonymous instance of TFInputListener as an ActionEvent
26          //   listener, which provides an ActionEvent handler called actionPerformed().
27          // Hitting "enter" on tfInput invokes actionPerformed().
28
29       add(new Label("The Accumulated Sum is: "));  // "super" Frame adds an anonymous Label
30
31       tfOutput = new TextField(10); // allocate TextField
32       tfOutput.setEditable(false);  // read-only
33       add(tfOutput);                // "super" Frame adds TextField
34
35       setTitle("AWT Accumulator");  // "super" Frame sets title
36       setSize(350, 120);  // "super" Frame sets initial window size
37       setVisible(true);   // "super" Frame shows
38    }
39
40    // The entry main() method
41    public static void main(String[] args) {
```

```
      // Invoke the constructor to setup the GUI, by allocating an anonymous instance
      new AWTAccumulator();
   }

   // Define an inner class to handle the input TextField.
   // An ActionEvent listener must implement ActionListener interface.
   private class TFInputListener implements ActionListener {
      // ActionEvent handler - Called back upon hitting "enter" key on TextField
      @Override
      public void actionPerformed(ActionEvent evt) {
         // Get the String entered into the TextField tfInput, convert to int
         int numberIn = Integer.parseInt(tfInput.getText());
         sum += numberIn;      // Accumulate numbers entered into sum
         tfInput.setText("");  // Clear input TextField
         tfOutput.setText(sum + ""); // Display sum on the output TextField
                                     // convert int to String
      }
   }
}
```

TRY:



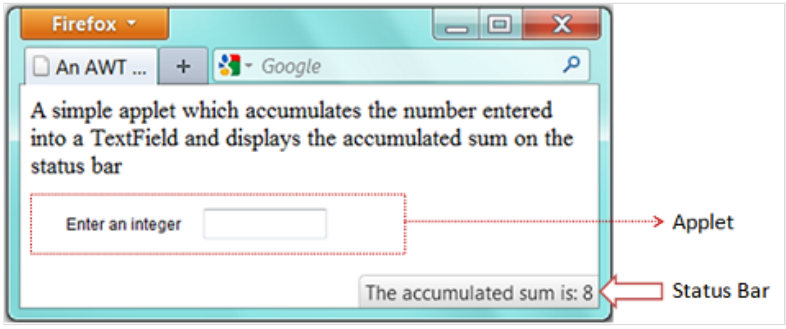
1. Modify the program (called `AWTAccumulatorLabel`) to display the sum using a `Label` instead of a protected `TextField`, as shown (using `FlowLayout`).
2. Modify the program (called `AWTFactorialTextField`) to display the factorial of the input number, as shown (using `FlowLayout`).

1.3 Ex: `AWTAccumulatorApplet` (Obsolete)

NOTE: Most browsers today do not support Java applets anymore. Keep this section for *nostalgia*.

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.




```
import java.applet.Applet;
import java.awt.*;        // Using AWT's containers and components
import java.awt.event.*;  // Using AWT's event classes and listener interfaces

// An applet extends java.applet.Applet
public class AWTAccumulatorApplet extends Applet implements ActionListener {
   private TextField tfInput;   // The input TextField
   private int sum = 0;         // The accumulated sum, init to 0

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

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

```
19
20      // ActionEvent handler - Called back when enter key was hit on TextField.
21      public void actionPerformed( ActionEvent evt) {
22         int numberIn = Integer.parseInt(evt.getActionCommand());
23            // getActionCommand() returns the String entered.
24         sum += numberIn;
25         tfInput.setText("");  // Clear input TextField
26         showStatus("The accumulated sum is: " + sum);
27            // show the sum on the status bar of the browser's window
28      }
29   }
```

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:

```
1   <html>
2   <head>
3     <title>An AWT Applet</title>
4   </head>
5   <body>
6     <p>A simple applet which accumulates the number entered into
7     a TextField and displays the accumulated sum on the status bar</p>
8     <applet code="AWTAccumulatorApplet.class" width="300" height="60">
9     </applet>
10  </body>
11  </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 Ex: `WindowEvent` and `WindowListener`

Modify the `AWTCounter` program (called `AWTCounterWithClose`) to process the "Window-Close" button. See the main article.

3. Inner Class - Named and Anonymous

Similar to `AWTCounter`, the following programs `AWTCounterNamedInnerClass` and `AWTCounterAnonymousInnerClass` use "named inner classes" and "anonymous inner classes", respectively, as the `ActionEvent` listener.

3.1 Ex: Named Inner Class

A named inner class as the event listener: `AWTCounterNamedInnerClass.java`

```
1   import java.awt.*;        // Using AWT's components and containers
2   import java.awt.event.*;  // Using AWT's event classes and listener interfaces
3
4   public class AWTCounterNamedInnerClass extends Frame {
5
6      private TextField tfCount;
7      private Button btnCount;
8      private int count = 0;
```

```
9
10      // Constructor to setup the UI components and event handlers
11      public AWTCounterNamedInnerClass () {
12         setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
13         add(new Label("Counter"));    // anonymous Label
14         tfCount = new TextField(count + "", 10);
15         tfCount.setEditable(false);   // read-only
16         add(tfCount);                 // "super" Frame adds tfCount
17
18         btnCount = new Button("Count");
19         add(btnCount);                // "super" Frame adds btnCount
20
21         // Construct an anonymous instance of inner class BtnListener as
22         //  listener to the source btnCount.
23         btnCount.addActionListener(new BtnListener());
24
25         setSize(250, 100);
26         setTitle("AWT Counter");
27         setVisible(true);    // show it
28      }
29
30      public static void main(String[] args) {
31         new AWTCounterNamedInnerClass();
32      }
33
34      // A named inner class to be used as listener of ActionEvent
35      // This inner class can access private variables of the outer class, such as count and tfCount.
36      private class BtnListener implements ActionListener {
37         @Override
38         public void actionPerformed(ActionEvent evt) {
39            ++count;
40            tfCount.setText(count + "");
41         }
42      }
43   }
```

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())`.

3.2 Ex: Anonymous Inner Class

An anonymous Inner class as the event listener: `AWTCounterAnonymousInnerClass.java`

```
1   import java.awt.*;       // Using AWT's components and containers
2   import java.awt.event.*; // Using AWT's event classes and listener interfaces
3
4   public class AWTCounterAnonymousInnerClass extends Frame {
5
6      private TextField tfCount;
7      private Button btnCount;
8      private int count = 0;
9
10     // Constructor to setup the UI components and event handlers
11     public AWTCounterAnonymousInnerClass () {
12        setLayout(new FlowLayout());  // "super" Frame sets to FlowLayout
13        add(new Label("Counter"));    // anonymous Label
14        tfCount = new TextField(count + "", 10);
15        tfCount.setEditable(false);   // read-only
16        add(tfCount);                 // "super" Frame adds tfCount
```

```
17
18          Button btnCount = new Button("Count");
19          add(btnCount);                // "super" Frame adds btnCount
20
21          // Construct an anonymous instance of an anonymous class as
22          //  listener to the source btnCount
23          btnCount.addActionListener(new ActionListener() {
24             @Override
25             public void actionPerformed(ActionEvent evt) {
26                ++count;
27                tfCount.setText(count + "");
28             }
29          });
30
31          setSize(250, 100);
32          setTitle("AWT Counter");
33          setVisible(true);   // show it
34       }
35
36       public static void main(String[] args) {
37          new AWTCounterAnonymousInnerClass();
38       }
39    }
```

Explanation

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

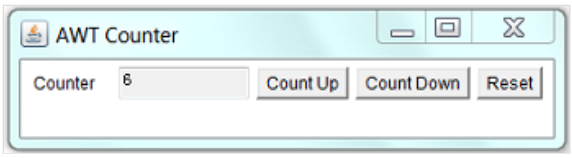
```
new ActionListener() { ... }
```

- 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 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 evt) {
   String btnLabel = evt.getActionCommand();
       // event.getActionCommand() returns the button's label
   if (btnLabel.equals("Count Up")) {
      ......
   } else if (btnLabel.equals("Count Down")) {
      ......
   } else {
      ......
   }
   ......
}
```

4. Swing GUI Applications

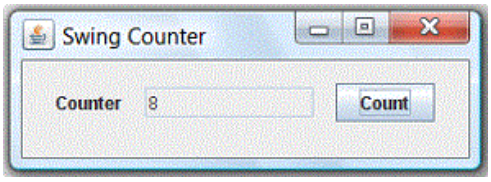
4.1 Ex: Converting from AWT to Swing

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(......);
```



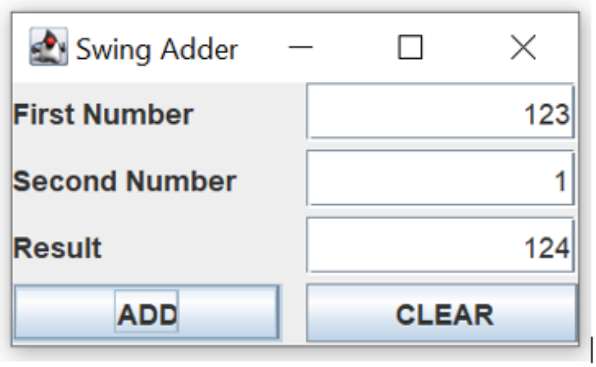
For example, `SwingCounter.java`:

```
1  import java.awt.*;       // Using AWT's layouts
2  import java.awt.event.*; // Using AWT's event classes and listener interfaces
3  import javax.swing.*;    // Using Swing components and containers
4
5  // A Swing application extends javax.swing.JFrame (instead of java.awt.Frame)
6  public class SwingCounter extends JFrame {
7     private JTextField tfCount;
8           // Use Swing's JTextField instead of AWT's TextField
9     private JButton btnCount;
10          // Using Swing's JButton instead of AWT's Button
11    private int count = 0;
12
13    public SwingCounter() {
14       // Get the content-pane of top-level container JFrame
15       // Components are added onto content pane
16       Container cp = getContentPane();
17       cp.setLayout(new FlowLayout());
18
19       cp.add(new JLabel("Counter"));
20       tfCount = new JTextField(count + "", 10);
21       tfCount.setEditable(false);
22       tfCount.setHorizontalAlignment(JTextField.RIGHT);
23       cp.add(tfCount);
24
25       btnCount = new JButton("Count");
26       cp.add(btnCount);
27       btnCount.addActionListener(new ActionListener() {
28         @Override
29          public void actionPerformed(ActionEvent evt) {
30             ++count;
31             tfCount.setText(count + "");
32          }
33       });
34
35       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
36          // Exit program if JFrame's close-window button clicked
37       setSize(300, 100);
38       setTitle("Swing Counter");
39       setVisible(true);    // show it
40    }
41
42    public static void main(String[] args) {
43       // Recommended to run the GUI construction in
44       //  Event Dispatching thread for thread-safe operations
45       SwingUtilities.invokeLater(new Runnable() {
46          @Override
47          public void run() {
```

```
            new SwingCounter(); // Let the constructor does the job
         }
      });
   }
}
```

4.2 Ex: 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.



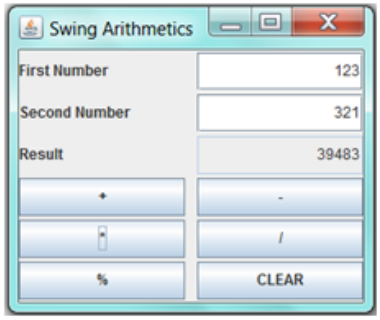
Hints: Set the content-pane to 4x2 GridLayout. The components are added from left-to-right, top-to-bottom.

```
import java.awt.*;        // Using AWT's layouts
import java.awt.event.*;  // Using AWT's event classes and listener interfaces
import javax.swing.*;     // Using Swing's components and container

// A 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;

   // Constructor to set up UI components and event handlers
   public SwingAdder() {
      // Swing components should be added to the content-pane of the JFrame.
      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(btnClear);                          // at (4, 2)
      btnClear.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent evt) {
            ......
         }
      });

      setDefaultCloseOperation(EXIT_ON_CLOSE); // for the "window-close" button
```

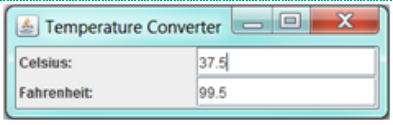
```
      setTitle("Swing Adder");
      setSize(300, 170);
      setVisible(true);
   }

   // 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.



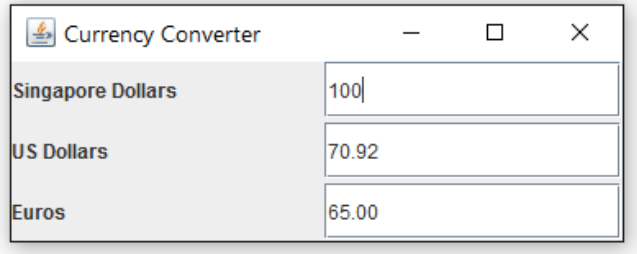
4.3 Ex: SwingTemperatureConverter



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.

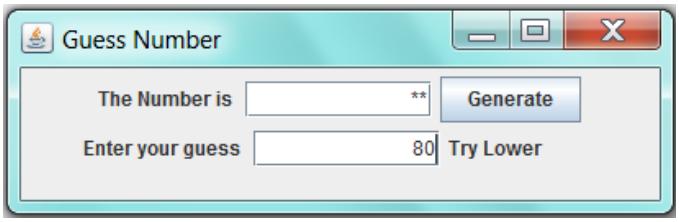
Hints: To display a floating-point number in a specific format (e.g., 1 decimal place), use the `static` method `String.format()`, which has the same form as `printf()`. For example, `String.format("%.1f", 1.234)` returns String "`1.2`".

4.4 Ex: SwingCurrencyConverter



Write a simple currency converter, as shown in the figure. User can enter the amount of "Singapore Dollars", "US Dollars", or "Euros", in floating-point number. The converted values shall be displayed to 2 decimal places. Assume that 1 USD = 1.41 SGD, 1 USD = 0.92 Euro, 1 SGD = 0.65 Euro.

4.5 Ex: SwingNumberGuess

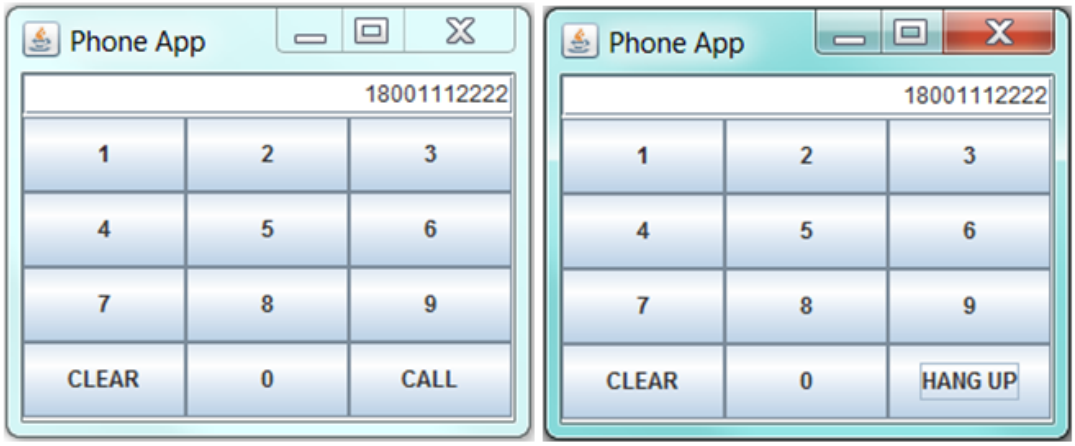


Write a number guessing game in Swing (as shown in the Figure). The program shall generate a random number between 1 to 100. It shall mask out the random number generated and output "Yot Got it", "Try Higher" or "Try Lower" depending on the user's input.

Hints:

- You can use `Math.random()` to generate a random number in `double` in the range of `[0.0, 1.0)`.

4.6 Ex: SwingPhoneApp



Write a Software Phone App using Java Swing as illustrated in the figure. The user enters the phone number and pushes the "CALL" button to start a phone call. Once the call is started, the label of the "CALL" button changes to "HANG UP". When the user hangs up, the display is cleared.

Assume that the following 2 methods are available for handling phone call:

```
public void call(String phoneNumber);  // to make a phone call with the phoneNumber
public void hangup();  // to terminate the existing call
```

Hints:

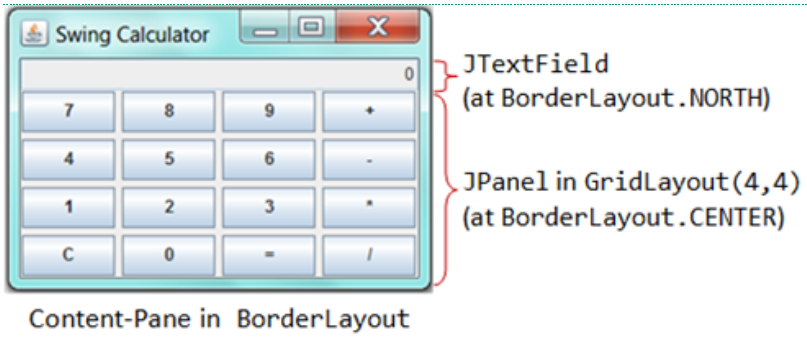
- Use a 10-element `JButton` array to hold the 10 numeric buttons. Construct a common instance of a named inner class as the `ActionListener` for the 10 numeric buttons.
- Use a `boolean` flag (says `isCalling`) to keep track of the status.

```
1  import java.awt.*;       // Using AWT's layouts
2  import java.awt.event.*; // Using AWT's event classes and listener interfaces
3  import javax.swing.*;    // Using Swing components and containers
4
5  // A Swing application extends javax.swing.JFrame
6  public class SwingPhoneApp extends JFrame {
7     private JButton[] btnNumbers;  // array
8     private JButton btnCall;
9     private JButton btnClear;
10    private JTextField tfDisplay;
11    private String strPhoneNumber = "";
12    private boolean isCalling = false;
13
14    public SwingPhoneApp() {
15       // Get the content-pane of top-level container JFrame
16       // Components are added onto content pane
17       Container cp = getContentPane();
18       cp.setLayout(new BorderLayout());
19
20       // Set up the top display panel
```

```
21          JPanel pnlDisplay = new JPanel(new FlowLayout());
22          cp.add(pnlDisplay, BorderLayout.NORTH);
23
24          // Set up the display text field
25          tfDisplay = new JTextField(30);
26          tfDisplay.setEditable(false);
27          tfDisplay.setHorizontalAlignment(JTextField.RIGHT);
28          pnlDisplay.add(tfDisplay);
29
30          // Set up the center button panel
31          JPanel pnlButtons = new JPanel(new GridLayout(4, 3, 3, 3));
32          cp.add(pnlButtons, BorderLayout.CENTER);
33
34          // Set up a common listener for all number buttons
35          MyNumberButtonListener lis = new MyNumberButtonListener();
36
37          // Allocate the JButton array for number buttons
38          btnNumbers = new JButton[10];
39          // Allocate each of the item (JButton) of the array
40          for (int i = 0; i < btnNumbers.length; ++i) {
41             btnNumbers[i] = new JButton(i + "");
42             // Add the common listener to each of the JButton
43             btnNumbers[i].addActionListener(lis);
44          }
45
46          // Add the JButtons in the right order
47          for (int i = 1; i <= 9; ++i) {
48             pnlButtons.add(btnNumbers[i]);
49          }
50
51          btnClear = new JButton("CLEAR");
52          pnlButtons.add(btnClear);
53          btnClear.addActionListener(new ActionListener() {
54             @Override
55             public void actionPerformed(ActionEvent e) {
56                strPhoneNumber = "";
57                tfDisplay.setText(strPhoneNumber);
58             }
59          });
60
61    //    JDK 8 Lambda Expression
62    //    btnClear.addActionListener(e -> {strPhoneNumber = ""; tfDisplay.setText(strPhoneNumber);});
63
64          pnlButtons.add(btnNumbers[0]);
65
66          btnCall = new JButton("CALL");
67          pnlButtons.add(btnCall);
68          btnCall.addActionListener(new ActionListener() {
69             @Override
70             public void actionPerformed(ActionEvent e) {
71                if (isCalling) {
72                   isCalling = false;
73                   btnCall.setText("CALL");
74                   // call(strPhoneNumber)
75                } else {
76                   isCalling = true;
77                   btnCall.setText("HANG UP");
78                   // hangup()
79                }
80             }
81          });
82
83          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
84             // Exit program if JFrame's close-window button clicked
85          setSize(300, 250);
86          setTitle("Swing Counter");
87          setVisible(true);    // show it
```

```
    }

    // Named inner listener class for the number buttons
    private class MyNumberButtonListener implements ActionListener {
       @Override
       public void actionPerformed(ActionEvent e) {
          strPhoneNumber += e.getActionCommand();
          tfDisplay.setText(strPhoneNumber);
       }
    }

    public static void main(String[] args) {
       // Recommended to run the GUI construction in
       //  Event Dispatching thread for thread-safe operations
       SwingUtilities.invokeLater(new Runnable() {
          @Override
          public void run() {
             new SwingPhoneApp(); // Let the constructor does the job
          }
       });
    }
}
```

4.7 Ex: SwingCalculator



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

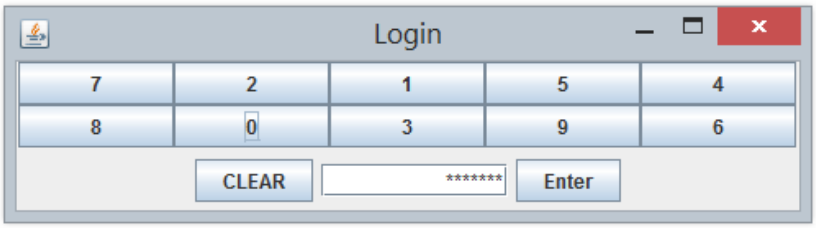
Hints:

- Set the `ContentPane` to `BorderLayout`. Add a `JTextField` (`tfDisplay`) to the `NORTH`. 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 *previous* operator. For example in "1 + 2 =", the current operator is "=", while the *previous* operator is "+". Perform the operation specified by the previous operator.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;   // Using Swing's components and containers

// A 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 components and event handlers
   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
         // ......
      }
   }
}
```

4.8 Ex: SwingLoginPanel

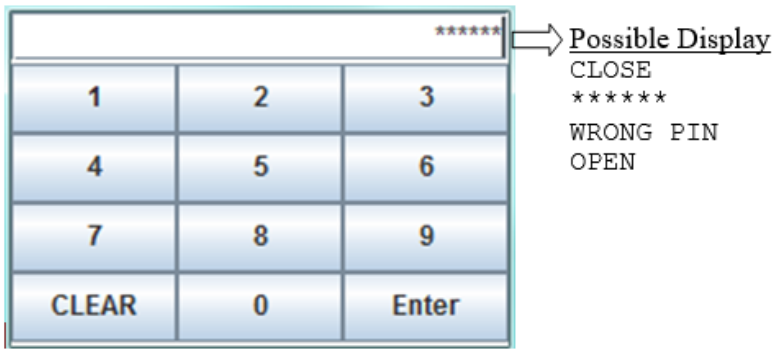


A Java Swing application has a login page as shown in the Figure. Users are required to enter the correct passcode to start the application. The system uses a scramble keypad with a randomly allocated set of numbers from 0 to 9. The display shall show "Enter passcode" initially, and show an asterisk (*) for each number entered. Upon pushing the "Enter" button, the system verifies the passcode. If the passcode is correct, the system invokes a method called `startApp()` to start the application. Otherwise, it displays "Wrong passcode". The "Clear" button shall clear the display.

Assume that the following methods are available:

```
public String getPasscode(); // return the passcode
public void startApp();      // Start the application
public void shuffleArray(int[] array)
   // Shuffle (Randomize) the given int array, e.g.,
   // int[] numbers = {1, 2, 3, 4, 5};
   // shuffleArray(numbers); // randomize the elements
```

4.9 Ex: SwingLock



Write a Java Swing application for an electronic lock as shown in the figure. The display shall show the state of either "CLOSE" or "OPEN". In the "CLOSE" state, the user types his PIN followed by the "Enter" key to unlock the system. The display shall show an asterisk (*) for each number entered. The display shall show "WRONG PIN" if the PIN is incorrect. The "Clear" button clears the number entered (if any), locks the system and sets the display to "CLOSE".

Assume that the following methods are available:

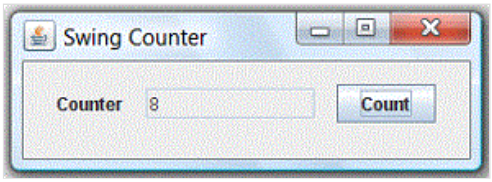
```
public boolean checkPIN(String PIN);  // return true for correct PIN
public void unlock();   // Unlock the system
public void lock();     // Lock the system
```

Hints:

- Use a 10-element `JButton` array to hold the 10 numeric buttons. Construct a common instance of a named inner class as their `ActionListener`.
- Use a `boolean` flag (says `isLocked`) to keep track of the status.

4.10 Ex: Using Eclipse/NetBeans GUI Builder

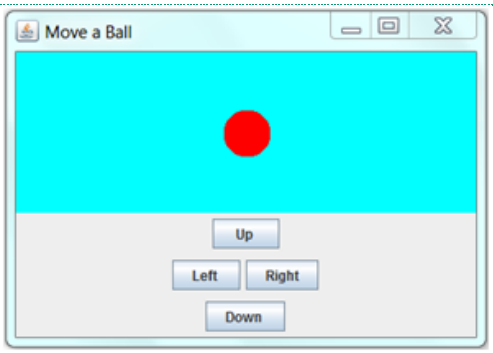
Write the `SwingCounter` using Eclipse/NetBeans' GUI builder. Read the respective section in "Eclipse How-To" or "NetBeans Hot-To".



Study the code generated by Eclipse/NetBeans.

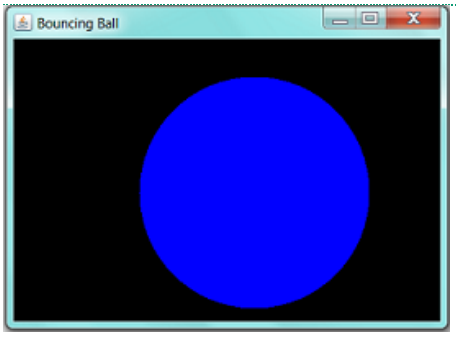
5. Custom Graphics

5.1 Ex: 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.

5.2 Ex: BouncingBall



Latest version tested: JDK 15.0.2
Last modified: April, 2021