

Special Topics No. 8: Swing Classes—Part One

This document is the first part of a two-part series on Swing classes. In this part, we will rewrite the sample programs from Chapter 13 using the Swing classes that replace the standard `java.awt` component classes. In the second part, Special Topics No. 9, we will describe additional capabilities and introduce new classes unique to Swing.

Introduction

In this document, we will show you how to rewrite most of the sample programs in Chapter 13 by using the Swing counterparts. We will introduce the Swing components that replace the java.awt GUI components described in Chapter 13 and highlight some key differences in using the Swing components instead of the java.awt components. Swing counterparts such as JLabel, JMenuItem, and others have many additional capabilities not found in the AWT classes. In this document, we will restrict the discussion to simply converting the AWT sample programs in Chapter 13 to Swing programs. We will leave the discussion of additional capabilities of the Swing classes mentioned here and introduction of other Swing classes in the companion document Special Topics No. 9.

1 Motivation Behind Lightweight Swing Components

Although it is possible to mix the AWT and Swing components in a single program, the rules for mixing the two correctly are not that simple. As such, we recommend not to mix the two.



Don't mix the AWT and Swing components in the same program.

2 The MyFirstJFrame Class

In this section, we will rewrite the MyFirstFrame class. The Swing-based MyFirstFrame class is named MyFirstJFrame. We follow the naming convention for the Swing classes of prefixing the class name with the letter J, such as JLabel, JButton, and so forth, in naming the sample classes in this document.

Many standard AWT components have counterpart Swing components, so converting the AWT-based programs to Swing-based programs mainly involves replacing the AWT classes with their Swing counterparts. Table 1 lists the AWT classes discussed in Chapter 13 (some of them first appeared in Chapter 5) and their Swing counterparts.

TABLE 1

AWT classes discussed in Chapter 13 and their Swing counterparts (some components first appeared in Chapter 5).

AWT Classes	Swing Counterparts
Label	JLabel
TextField	JTextField
Button	JButton
Frame	JFrame
Dialog	JDialog
Menu	JMenu
MenuItem	JMenuItem
MenuBar	JMenuBar

As noted in the Special Topics No. 2 document, a simple textual substitution, such as changing

```
private Label prompt;
...
prompt = new Label( "Please enter your name:" );
```

to

```
private JLabel prompt;
...
prompt = new JLabel( "Please enter your name:" );
```

is all you need to do to use Swing components. Since the Swing counterparts support the same methods supported by the AWT component classes, no further changes are necessary in the statements that call the components' methods. The Swing components are defined in the `javax.swing` package.

Unlike the AWT components, the Swing components are not placed directly on the `Frame` object. Instead, they are placed on the *content pane* of the `Frame` object. In the constructor of `MyFirstFrame`, we add a `Button` components as

```
...
okButton = new JButton("OK");
okButton.setBounds(70, 125, BUTTON_WIDTH, BUTTON_HEIGHT);
add(okButton);
...
```

This adds **okButton** to the **MyFirstFrame** object.

The corresponding constructor for `MyFirstJFrame` is

This adds `okButton` to the content pane of the `MyFirstJFrame` object.

```
...
Container contentPane = getContentPane( );
...
okButton = new JButton("OK");
okButton.setBounds(70, 125, BUTTON_WIDTH, BUTTON_HEIGHT);
contentPane.add(okButton);
...
```

Because we are placing objects on the `contentPane` container, the layout manager we want to use must be attached to `contentPane`. In this example, we are not using any layout manager, so we set the layout manager of `contentPane` as null in the constructor:

```
contentPane.setLayout( null );
```

Since the absolute positioning of components is relative to the content pane, we need to adjust the values we pass to the `setBounds` method of the buttons. Figure 1 shows a sample placement of a button that is relative to the content pane.

In the `MyFirstFrame` class, we used a `ProgramTerminator` object, whose task is to terminate the program when a window closing event occurs, as the frame's window event listener. We did this to make the termination of a program easy. With the Swing `JFrame` class, the `ProgramTerminator` is no longer necessary because we can set a default closing operation of a `JFrame` by using its `setDefaultCloseOperation` method. We call the method as

```
//set 'Exit upon closing' as the default close operation
setDefaultCloseOperation( EXIT_ON_CLOSE );
```

The constant `EXIT_ON_CLOSE` is defined in the `JFrame` class. Other possible values are defined in the `WindowConstants` class (e.g., `DO_NOTHING_ON_CLOSE`, `HIDE_ON_CLOSE`, and `DISPOSE_ON_CLOSE`). We still have to define a special class such as `ProgramTerminator` if we want some application-specific actions to take place. However, if all we want is a generic closing operation, such as hiding a window or terminating a program, we can use the `setDefaultCloseOperation`. We will use this method in this and other programs in this document.

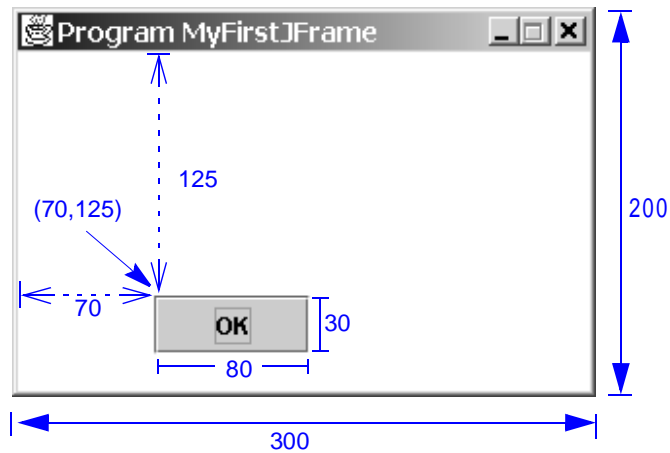
Action event handling remains the same so this part of the code requires no changes. We are now ready to list the complete `MyFirstJFrame` class.

FIGURE 1The diagram illustrating how the **setBounds** method works.

```

frame = new Frame();
frame.setSize(300,200);
contentPane = frame.getContentPane();
contentPane.setLayout( null );
okButton = new Button("OK");
okButton.setBounds(70, 125, 80, 30);
contentPane.add(okButton);

```



File: MyFirstJFrame.java

```

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

/**
 * MyFirstJFrame class
 *
 * <p>
 * A sample frame to illustrate the placing of GUI objects and event
 * handling with the Swing JFrame.
 */
class MyFirstJFrame extends JFrame implements ActionListener
{
    //-----
    //    Data Members
    //-----

    /**

```

```

    * Default frame width
    */
private static final int FRAME_WIDTH    = 300;

/**
 * Default frame height
 */
private static final int FRAME_HEIGHT   = 200;

/**
 * X coordinate of the frame default origin point
 */
private static final int FRAME_X_ORIGIN = 150;

/**
 * Y coordinate of the frame default origin point
 */
private static final int FRAME_Y_ORIGIN = 250;

/**
 * Default width for buttons
 */
private static final int BUTTON_WIDTH   = 80;

/**
 * Default height for buttons
 */
private static final int BUTTON_HEIGHT  = 30;

/**
 * The Swing button for Cancel
 */
private JButton cancelButton;

/**
 * The Swing button for OK
 */
private JButton okButton;

//-----
// Constructors
//-----

/**
 * Default constructor
 */
public MyFirstJFrame()
{
    Container contentPane = getContentPane( );

    //set the frame properties
    setSize      ( FRAME_WIDTH, FRAME_HEIGHT );
    setResizable ( false );
    setTitle     ( "Program MyFirstJFrame" );
    setLocation  ( FRAME_X_ORIGIN, FRAME_Y_ORIGIN );

    //set the content pane properties
    contentPane.setLayout( null );
    contentPane.setBackground( Color.white );

```

```

        //create and place two buttons on the frame's content pane
        okButton = new JButton("OK");
        okButton.setBounds(70, 125, BUTTON_WIDTH, BUTTON_HEIGHT);
        contentPane.add(okButton);

        cancelButton = new JButton("CANCEL");
        cancelButton.setBounds(160, 125, BUTTON_WIDTH, BUTTON_HEIGHT);
        contentPane.add(cancelButton);

        //register this frame as an action listener of the two buttons
        cancelButton.addActionListener(this);
        okButton.addActionListener(this);

        //set 'Exit upon closing' as the default close operation
        setDefaultCloseOperation( EXIT_ON_CLOSE );
    }

    //-----
    //      Public Methods:
    //
    //      void      actionPerformed      (      ActionEvent      )
    //
    //-----

    /**
     * Standard method to respond the action event.
     *
     * @param event the ActionEvent object
     */
    public void actionPerformed(ActionEvent event)
    {
        JButton clickedButton = (JButton) event.getSource();

        if (clickedButton == cancelButton) {
            setTitle("You clicked CANCEL");
        }
        else { //the event source must be okButton
            setTitle("You clicked OK");
        }
    }
}

```

Here are some of the key points in converting AWT-based programs to Swing-based programs.



Things to remember in writing Swing-based programs:

- 1. Use Swing components such as JButton, JLabel, and so forth.***
- 2. Add components to the content pane (Container) of the frame.***
- 3. Values passed to the setBounds method are relative to the content pane.***

- 4. Set the content pane's background color to `Color.white` *explicitly* if this is the color you want to use (default is blue-gray).**

3 The JMenuBar Class

We will rewrite the `MenuFrame` class from Section 13.4. Following our naming convention, the new class is named `JMenuBar`. The main task is replacing the AWT class `MenuBar`, `Menu`, and `MenuItem` with `JMenuBar`, `JMenu`, and `JMenuItem`, respectively. Unlike other Swing components, Swing menu objects are associated to the frame, not to its content pane. So this part of code requires no changes, except for the method associate a `JMenuBar` object to a frame is `setJMenuBar` (it was `setMenuBar` in the `MenuFrame` class).

Here is the `JMenuBar` class:

File: [JMenuBar.java](#)

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

/**
 * class MenuFrame
 *
 * This frame includes one MenuBar, two Menu objects File and Edit,
 * and eight MenuItem objects. When a menu item is selected, a string
 * showing which menu choice is selected will appear on the frame.
 *
 * @author Dr. Caffeine
 */

class JMenuBar extends JFrame implements ActionListener
{
    //-----
    // Data Members
    //-----

    /**
     * Default frame width
     */
    private static final int FRAME_WIDTH = 450;

    /**
     * Default frame height
     */
    private static final int FRAME_HEIGHT = 300;

    /**
     * X coordinate of the frame default origin point
     */
    private static final int FRAME_X_ORIGIN = 150;

    /**
     * Y coordinate of the frame default origin point
     */
}
```



```

    */
    private static final int FRAME_Y_ORIGIN = 250;

    /**
     * Text shown in response to the menu selection
     */
    private JLabel    response;

    /**
     * File menu group
     */
    private JMenu     fileMenu;

    /**
     * Edit menu group
     */
    private JMenu     editMenu;

//-----
//    Constructors
//-----

    /**
     * Default constructor
     */
    public JMenuBar()
    {
        Container contentPane;

        //set the frame properties
        setTitle      ("Testing Swing Menus");
        setSize       (FRAME_WIDTH, FRAME_HEIGHT);
        setResizable  (false);
        setLocation   (FRAME_X_ORIGIN, FRAME_Y_ORIGIN);

        contentPane = getContentPane( );
        contentPane.setLayout(null);
        contentPane.setBackground( Color.white );

        //create two menus and their menu items
        createFileMenu();
        createEditMenu();

        //and add them to the menubar
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        menuBar.add(fileMenu);
        menuBar.add(editMenu);

        //create and position response label
        response = new JLabel("Hello, this is your menu tester." );
        response.setBounds(100, 100, 250, 50);
        contentPane.add(response);

        //set 'Exit upon closing' as the default close operation
        setDefaultCloseOperation( EXIT_ON_CLOSE );
    }

//-----

```

```

//      Public Methods:
//
//      void      actionPerformed      (      ActionEvent      )
//
//-----

/**
 * Standard method to respond the action event.
 *
 * @param event the ActionEvent object
 */
public void actionPerformed(ActionEvent event)
{
    String  menuName;

    menuName = event.getActionCommand();

    if (menuName.equals("Quit")) {
        System.exit(0);
    }
    else {
        response.setText(menuName + " is selected.");
    }
}

//-----
//      Private Methods:
//
//      void      createFileMenu      (      )
//      void      createEditMenu      (      )
//
//-----

/**
 * Create File menu and its menu items
 */
private void createFileMenu( )
{
    JMenuItem      item;

    fileMenu = new JMenu("File");

    item = new JMenuItem("New");           //New
    item.addActionListener( this );
    fileMenu.add( item );

    item = new JMenuItem("Open...");       //Open...
    item.addActionListener( this );
    fileMenu.add( item );

    item = new JMenuItem("Save");           //Save
    item.addActionListener( this );
    fileMenu.add( item );

    item = new JMenuItem("Save As...");    //Save As...
    item.addActionListener( this );
    fileMenu.add( item );
}

```

```

        fileMenu.addSeparator();           //add a horizontal separator line

        item = new JMenuItem("Quit");      //Quit
        item.addActionListener( this );
        fileMenu.add( item );
    }

    /**
     * Create Edit menu and its menu items
     */
    private void createEditMenu( )
    {
        JMenuItem    item;

        editMenu = new JMenu("Edit");

        item = new JMenuItem("Cut");        //Cut
        item.addActionListener( this );
        editMenu.add( item );

        item = new JMenuItem("Copy");       //Copy
        item.addActionListener( this );
        editMenu.add( item );

        item = new JMenuItem("Paste");      //Paste
        item.addActionListener( this );
        editMenu.add( item );
    }
}

```

4 The MyJSketchPad Class

Mouse listeners and mouse movement listeners work the same in the Swing class, so this part requires no changes. The only change we have to make in the `MySketchPad` class is the way to erase the contents when the right mouse button is clicked. In the `MySketchPad` class, we used the `clearRect` method of the `Graphics` class. This worked fine in the original because the default background color was white.

We make the changes here by setting the background color of the content pane to white and use the `fillRect` method to paint the whole content pane with the set background color when the right mouse button is clicked. We cannot use `clearRect` here because it will restore the frame's blue-gray color, not the white background we want. The portion that erases the drawing is changed to the following:

```

Graphics g = getGraphics();
Rectangle r = getBounds();
g.setColor( getContentPane().getBackground() );
g.fillRect(0, 0, r.width, r.height);

```

```
g.dispose();
```

Here is the MyJSketchPad class:

File: [MyJSketchPad.java](#)

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

/**
 * class MySketchPad
 *
 * An alternative implementation of the SketchPad class
 * from the javabook package.
 */
class MyJSketchPad extends JFrame implements
    MouseListener, MouseMotionListener
{
    //-----
    // Data Members
    //-----

    /**
     * Default frame width
     */
    private static final int FRAME_WIDTH = 300;

    /**
     * Default frame height
     */
    private static final int FRAME_HEIGHT = 200;

    /**
     * X coordinate of the frame default origin point
     */
    private static final int FRAME_X_ORIGIN = 150;

    /**
     * Y coordinate of the frame default origin point
     */
    private static final int FRAME_Y_ORIGIN = 250;

    /**
     * X coordinate of the last drag point
     */
    private int last_x;

    /**
     * Y coordinate of the last drag point
     */
    private int last_y;

    //-----
    // Constructors
    //-----

    /**
     * Default constructor
     */
}
```

```

public MyJSketchPad()
{
    //set frame properties
    setTitle      ("SketchPad For YOur Doodle Art");
    setSize       (FRAME_WIDTH, FRAME_HEIGHT);
    setResizable  (false);
    setLocation   (FRAME_X_ORIGIN, FRAME_Y_ORIGIN);

    getContentPane().setBackground( Color.white );

    setDefaultCloseOperation( EXIT_ON_CLOSE );

    last_x = last_y = 0;

    addMouseListener( this );
    addMouseMotionListener( this );
}

//-----
//      Public Methods:
//
//      void    mousePressed    (    MouseEvent    )
//
//      void    mouseDragged    (    MouseEvent    )
//
//-----

/*****
    Mouse Event Handling
    *****/

/**
 * Standard method to respond MousePressed events.
 *
 * @param e the MouseEvent object
 */
public void mousePressed(MouseEvent e)
{
    if (e.isMetaDown()) {
        //erase the content if it is a rightbutton

        Graphics g = getGraphics();
        Rectangle r = getBounds();
        g.setColor( getContentPane().getBackground() );
        g.fillRect(0, 0, r.width, r.height);
        g.dispose();
    }
    else {
        //reset for a new mouse drag
        last_x = e.getX();
        last_y = e.getY();
    }
}

public void mouseReleased(MouseEvent e) { }
public void mouseClicked(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }

```

```

/*****
    Mouse Motion Event Handling
*****/

/**
 * Standard method to respond MouseDragged events.
 *
 * @param e the MouseEvent object
 */
public void mouseDragged(MouseEvent e)
{
    //process the close box event
    int x = e.getX();
    int y = e.getY();

    if (!e.isMetaDown()) { //don't process right button drag
        Graphics g = getGraphics();
        g.drawLine(last_x, last_y, x, y);
        last_x = x;
        last_y = y;
        g.dispose();
    }
}

public void mouseMoved(MouseEvent e) { }
}

```

5 Sample Program: A Simple Calculator

We have covered everything you need to know to convert the Calculator class to a Swing-based class. As a review of what we have covered so far, we will list the new JCalculator class in its entirety here.

File: [JCalculator.java](#)

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javabook2.*; //we use the Swing version of javabook

/**
 * Program Calculator
 *
 * <p>
 * This program provides a simple four function calculator with
 * two input fields to enter the left and right operands of an
 * arithmetic operation.
 */

class JCalculator extends JFrame implements ActionListener
{
    //-----
    // Data Members

```

```
//-----
/**
 * For entering the left operand
 */
private JTextField    leftOperand;

/**
 * For entering the right operand
 */
private JTextField    rightOperand;

/**
 * Add operator button
 */
private JButton        plusButton;

/**
 * Subtract operator button
 */
private JButton        minusButton;

/**
 * Multiplication operation button
 */
private JButton        multiplyButton;

/**
 * Division operation button
 */
private JButton        divideButton;

/**
 * Clear entry button
 */
private JButton        clearButton;

/**
 * For displaying error messages
 */
private MessageBox    errorMessageBox;

//-----
// Constructors
//-----

/**
 * Default constructor
 */
public JCalculator()
{
    //set window properties
    super    ( "Calculator" );

    setSize    ( 250, 175    );
    setResizable ( false    );
    setLocation ( 300, 200    );

    getContentPane().setLayout( null );

    //create and layout components

```

```

        initComponents();

        setDefaultCloseOperation( EXIT_ON_CLOSE );

        errorMessageBox = new MessageBox(this);
    }

//-----
//      Public Methods:
//
//      void      actionPerformed    (      ActionEvent      )
//-----

/**
 * Performs the specified calculation and display
 * the result.
 *
 * @param event the ActionEvent object
 */
public void actionPerformed (ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if (clickedButton == clearButton) {
        clearEntries();
    }
    else {
        //an operator button is clicked, so compute
        compute( clickedButton );
    }
}

//-----
//      Private Methods:
//
//      void      clearEntries        (                )
//      void      compute             ( JButton        )
//      void      initComponents      (                )
//
//      void      displayResult       (                )
//
//      boolean   isNumber            ( JTextField     )
//      boolean   isOperandValid     ( JTextField     )
//-----

/**
 * Clears the left and right operand JTextField objects
 */
private void clearEntries()
{
    leftOperand.setText("");
    rightOperand.setText("");
}

/**
 * Determines which operation is requested and perform
 * the appropriate computation.

```



```

*
* @param operatorButton one of the operator buttons
*                        clicked by the user
*/
private void compute (JButton operatorButton)
{
    float    leftOperandValue,
             rightOperandValue,
             result = 0f;

    //are values in text fields valid numbers?
    if ( isOperandValid(leftOperand) &&
        isOperandValid(rightOperand) ) {

        //okay, so get the two operands
        leftOperandValue = getNumberFrom(leftOperand);
        rightOperandValue = getNumberFrom(rightOperand);

        //compute the result
        if ( operatorButton == plusButton ) {
            result = leftOperandValue + rightOperandValue;
        }
        else if ( operatorButton == minusButton ) {
            result = leftOperandValue - rightOperandValue;
        }
        else if ( operatorButton == divideButton ) {
            result = leftOperandValue / rightOperandValue;
        }
        else if ( operatorButton == multiplyButton ) {
            result = leftOperandValue * rightOperandValue;
        }
        displayResult(result);
    }
    else { //invalid operands
        errorMessageBox.show("Error: Invalid Data");
    }
}

/**
 * Creates and initializes GUI components. Absolute positioning
 * is used.
 */
private void initGUIComponents( )
{
    Container contentPane = getContentPane( );
    contentPane.setBackground( Color.white );

    //create buttons
    leftOperand    = new JTextField();
    rightOperand   = new JTextField();

    plusButton     = new JButton("+");
    minusButton    = new JButton("-");
    multiplyButton  = new JButton("X");
    divideButton   = new JButton("/");
    clearButton    = new JButton("CLEAR");

    //position buttons
    leftOperand.setBounds ( 20, 40, 85, 25);

```

```

        rightOperand.setBounds ( 20, 75, 85, 25);

        plusButton.setBounds (110, 40, 50, 30);
        minusButton.setBounds (170, 40, 50, 30);
        multiplyButton.setBounds (110, 75, 50, 30);
        divideButton.setBounds (170, 75, 50, 30);
        clearButton.setBounds (110, 110, 110, 30);

        //add this calculator as an action listener
        //to all five buttons
        plusButton.addActionListener( this );
        minusButton.addActionListener( this );
        multiplyButton.addActionListener( this );
        divideButton.addActionListener( this );
        clearButton.addActionListener( this );

        //add buttons to the calculator frame
        contentPane.add(leftOperand);
        contentPane.add(rightOperand);

        contentPane.add(plusButton);
        contentPane.add(minusButton);
        contentPane.add(multiplyButton);
        contentPane.add(divideButton);

        contentPane.add(clearButton);
    }

    /**
     * Displays the argument in the left operand JTextField
     *
     * @param number the number to display
     */
    private void displayResult(float number)
    {
        leftOperand.setText(Convert.toString(number));
        rightOperand.setText("");
    }

    /**
     * Retrieves the number from one of the operand JTextField
     *
     * @param operand the operand to retrieve the entered value
     *
     * @return the value entered in the designated operand field
     */
    private float getNumberFrom(JTextField operand)
    {
        return Convert.toFloat(operand.getText());
    }

    /**
     * Determines whether the given number is valid or not
     *
     * @param operand one the operand JTextField
     *
     * @return true if valid; else return false
     */
    private boolean isOperandValid(JTextField operand)

```

```
{
    boolean status;

    try {
        float number = Convert.toFloat(operand.getText());
        status = true;
    }
    catch (NumberFormatException e) {
        status = false;
    }

    return status;
}
}
```