Swing

1. Swing

Java Swing is a lightweight Graphical User Interface (GUI) toolkit that includes a rich set of widgets. It includes package let we make GUI components for our Java applications, and it is platform independent.

Swing API is a set of extensible GUI Components to ease the developer's life to create Java based Front End/GUI Applications. It is built on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls.

> Note:
> API (application programming interface) is a set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication among various components.
> We can see the API specification for the Java Platform, Standard Edition 8 from https://docs.oracle.com/javase/8/docs/api/index.html.

To display the GUI component, we have to hang the component on the tree in a containment hierarchy, where the container is a Top-Level Hierarchy. A Container in AWT is a component itself and it provides the capability to add a component to itself. In Swing, the three basic top level containers are prepared; JFrame, JDalog and JApplet.

Since Swing offers a wide variety of configuration methods, this document will introduce about JFrame. We use a three-level structure consisting of top-level containers, intermediate containers and atomic components (individual parts).

2. Top-Level Container

An example program that realized the top-level container using JFrame is shown below.

```
JFrame frame = new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400, 300);
frame.setVisible(true);
```

For classes that inherit JFrame, the first line and "frame." are unnecessary.

The constractor and the main methods of JFrame are introduced below.

Constructor & Description of JFrame
JFramel() Constructs a new frame that is initially invisible.
JFrame(String title) Creates a new, initially invisible Frame with the specified title.

Method & Description of JFrame, Frame and Window
Container getContentPane() Returns the contentPane object for this frame.
Graphics getGraphics() Creates a graphics context for this component. This method will return null if this component is currently not displayable.
void setBackground(Color bgColor) Sets the background color of this window.
void setBounds(int x, int y, int width, int height) Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.
void setDefaultCloseOperation(int operation) Sets the operation that will happen by default when the user initiates a "close" on this frame. If the operation is EXIT_ON_CLOSE (defined in JFrame) , exits the application using the System exit method.
void setSize(int width, int height) Resizes this component so that it has width width and height height.
void setTitle(String title) Sets the title for this frame to the specified string.
void setVisible(boolean b) Shows or hides this Window depending on the value of parameter b.

The getContentPane method can provide a Container class that can manipulate top-level containers that contain intermediate containers.

Figure 1 shows the relationship between superclass and subclass provided by Swing. For example, the superclass of the Container is the Component class and the superclass of the Component class is the Object class. Therefore, the Container class can use methods of the Component class and the Object class.

In the JPanel class used by the intermediate container, and the JLavel class, the JButton class and the JTextField class which are the atomic component, methods of

the JComponent class, the Container class, the Component class and the Object class cane be used.

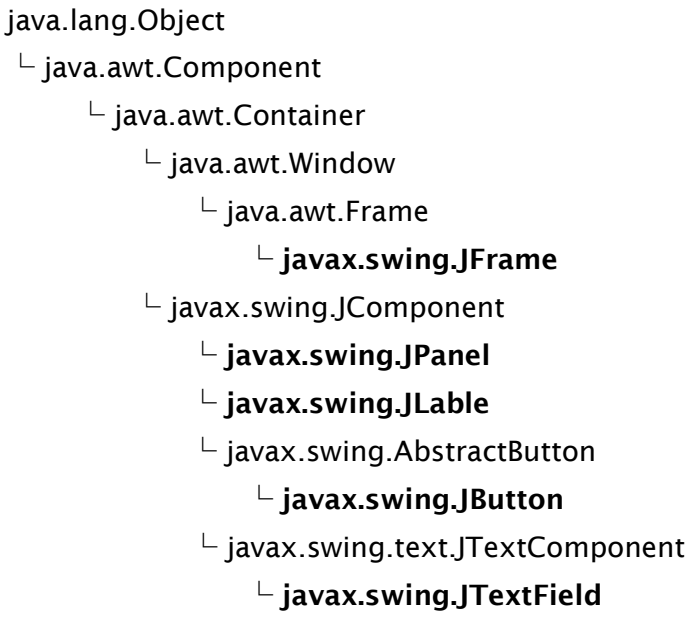

Figure 1. Relationship of classes related to Swing

The main methods of JComponent, Container and Component are introduced below.

Method & Description of JComponent class
void setBackground(Color color) Sets the background color of this component.
void setFont(Font font) Sets the font for this component.
void setForeground(Color color) Sets the foreground color of this component.
void setVisible(boolean flag) Makes the component visible or invisible. if flag is true, make the component visible; othrewise, make it invisible.

Method & Description of Container class
Component add(Component comp) Appends the specified component to the end of this container.
Component add(Component comp, int index) Adds the specified component to this container at the given position by index.

Method & Description of Component class
void addMouseListener(MouseListener ml) Adds the specified mouse listener to receive mouse events from this component.
void addMouseMotionListener(MouseMotionListener ml) Adds the specified mouse motion listener to receive mouse motion events from this component.
void repaint() Repaints this component. If this component is a lightweight component, this method causes a call to this component's paint method as soon as possible.
void setLayout(LayoutManager mgr) Sets the layout manager for this container.

3. Intermediate Container

An intermediate container is a container for placing other Swing components. In this document, the JPanel is used. How to use JPanel is introduced in Section 5.1.

4. Atomic Component

Swing offers many classes that can be used as atomic components. Some of them will be introduced in this chapter.

4.1 JLabel

The class **JLabel** can display either text, an image, or both. Label's contents are aligned by setting the vertical and horizontal alignment in its display area. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

(A) Constructors

Constructor & Description of JLabel class
JLabel() Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String text) Creates a JLabel instance with the specified text.
JLabel(String text, int horizontalAlignment) Creates a JLabel instance with the specified text and horizontal alignment.

(B) Methods

Method & Description of JLabel class
int getHorizontalAlignment() Returns the alignment of the label's contents along the **X** axis.
int getText() Returns the text string that the label displays.
void setHorizontalAlignment(int alignment) Sets the alignment of the label's contents along the **X** axis.
void setText(String text) Defines the single line of text this component will display.

4. 2 JButton

The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can also have an Image.

(A) Constructors

Constructor & Description of JButton class
JButton() Creates a button with no set text or icon.
JButton(String text) Creates a button with the text.
JButton(String text, Icon icon) Creates a button with an initial text and an icon.

(B) Methods (including methods of super classes)

Method & Description of JButton class and AbstractButton class
void addActionListener(ActionListener al) Adds an ActionListener to the button.
void doClick() Programmatically perform a "click". This does the same thing as if the user had pressed and released the button.
String getText() Returns the text string that the label displays.
void setActionCommand(String command) Sets the action command for this button.
void setEnabled(Boolean *b*) If *b* is true, enable the button, otherwise disable the button.
void setText(String text) Defines the single line of text this component will display.

4. 3 JTextField

The class **JTextField** is a component which allows the editing of a single line of text.

(A) Constructors

Constructor & Description of JTextField class
JTextField() Constructs a new TextField.
JTextField(int columns) Constructs a new empty TextField with the specified number of columns.
JTextField(String text) Constructs a new TextField initialized with the specified text.
JTextField(String text, int columns) Constructs a new TextField initialized with the specified text and columns.

(B) Methods (including methods of super classes)

Method & Description of JTextField class and JTextComponent class
int getHorizontalAlignment() Returns the alignment of the label's contents along the **X** axis.

int getText() Returns the text string that the label displays.
void setActionCommand(String command) Sets the command string used for action events.
void setFont(Font font) Sets the font for this component.
void setHorizontalAlignment(int alignment) Sets the alignment of the label's contents along the **X** axis.
void setText(String text) Defines the single line of text this component will display.

5. Layouts

Layout refers to the arrangement of components within the container. In another way, it could be said that layout is placing the components at a particular position within the container. The task of laying out the controls is done automatically by the Layout Manager.

Java provides various layout managers to position the controls. But, the most frequently used are BorderLayout class and FlowLayout class.

5. 1 BorderLayout

The class BorderLayout arranges the components to fit in the five regions: east, west, north, south and center. Each region can contain only one component and each component in each region is identified by the corresponding constant EAST, WEST, NORTH, SOUTH and CENTER which are provided in Field of BorderLayout class.

An example program using JButton and BorderLayout class and the result are shown in Figure 2 and Figure 3, respectively.

JButton variables are sometimes referred to in several places, so it has better to declare them in the field.

The five JButton objects are placed on the five JPanel objects respectively (lines 18-20, 22-24, 26-28, 30-32, 34-36). In Example71.java, five panels are prepared in five areas (lines 38 to 42). Finally, we use BordarLayout on the contentPane obtained by getContentPane method of JFrame to arrange panels in each area. In the result shown in Figure 3, the west, center and east buttons are located relatively upper. This is because they are placed directly under the north region. If nothing is placed in the north area, they are displayed at the top.

The south button is placed at the bottom of the window.

Notice that setVisible(true) method needs to be executed after setting buttons.

Example71.java

```java
/*
 * An example of JButton and BorderLayout
 * by Hitoshi Ogawa on November 8th, 2021
 */
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Example71 extends JFrame {
    JButton button1, button2, button3, button4, button5;

    public Example71() {
        setSize(400,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Example");

        button1 = new JButton("north");
        JPanel panel1 = new JPanel();
        panel1.add(button1);

        button2 = new JButton("east");
        JPanel panel2 = new JPanel();
        panel2.add(button2);

        button3 = new JButton("west");
        JPanel panel3  = new JPanel();
        panel3.add(button3);

        button4 = new JButton("south");
        JPanel panel4 = new JPanel();
        panel4.add(button4);

        button5 = new JButton("center");
        JPanel panel5 = new JPanel();
        panel5.add(button5);

        getContentPane().add(panel1, BorderLayout.NORTH);
        getContentPane().add(panel2, BorderLayout.EAST);
        getContentPane().add(panel3, BorderLayout.WEST);
        getContentPane().add(panel4, BorderLayout.SOUTH);
        getContentPane().add(panel5, BorderLayout.CENTER);
        setVisible(true);
    }

    public static void main(String[] args) {
        new Example71();
    }
}
```

Figure 2. An Example program of JButton and BorderLayout.

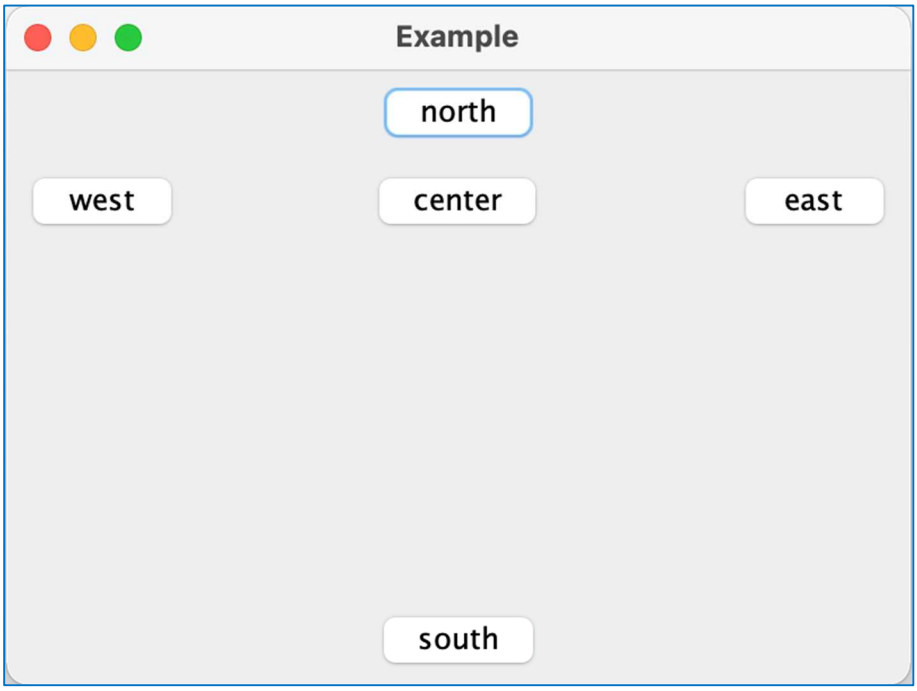

Figure 3. The Result of program of JFrame and BorderLayout.

5. 2 FlowLayout

FlowLayout is the common used layout. It is default layout used by JPanel. It is used to arrange components in a line or a row (for example from left to right). It arranges components in a line, if no space left remaining components goes to next line. Align property determines alignment of the components as center, left and right by the corresponding constant CENTER, LEFT and RIGHT which are provide in the Field of FlowLayout class.

An example program using FlowLayout class and the result are shown in Figure 4 and Figure 5, respectively.

In order to realize a flexible layout, it is recommended to set FlowLayout to panel and BorderLayout to Top-Level Container. For example, “button1” and “button2” are placed on the left in panel “p1” using setLayout method of Component class, and “p1” is placed in the top of the window (see lines 18 to 24). The buttons (“button3” and “button4”) are placed on the center in panel “p2”, and “p2” is placed in the center of the window (see lines 26 to 32). The buttons (“button5” and “button6”) are placed on the right in panel “p3”, and “p3” is placed in the south of the window (see lines 34 to 40).

Notice that setVisible(true) method needs to be executed after setting buttons.

Example72.java

```java
/*
 * An example of FlowLayout
 * by Hitoshi Ogawa on November 8th, 2021
 */
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Example72 extends JFrame {
    JButton button1, button2, button3, button4, button5, button6;

    public Example72() {
        setSize(400,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        button1 = new JButton("Button1");
        button2 = new JButton("Button2");
        JPanel p1 = new JPanel();
        p1.setLayout(new FlowLayout(FlowLayout.LEFT));
        p1.add(button1);
        p1.add(button2);
        getContentPane().add(p1, BorderLayout.NORTH);

        button3 = new JButton("Button3");
        button4 = new JButton("Button4");
        JPanel p2 = new JPanel();
        p2.setLayout(new FlowLayout(FlowLayout.CENTER));
        p2.add(button3);
        p2.add(button4);
        getContentPane().add(p2, BorderLayout.CENTER);

        button5 = new JButton("Button5");
        button6 = new JButton("Button6");
        JPanel p3 = new JPanel();
        p3.setLayout(new FlowLayout(FlowLayout.RIGHT));
        p3.add(button5);
        p3.add(button6);
        getContentPane().add(p3, BorderLayout.SOUTH);

        setVisible(true);

    }

    public static void main(String[] args) {
        new Example72();
    }
}
```

Figure 4. An Example program of FlowLayout.

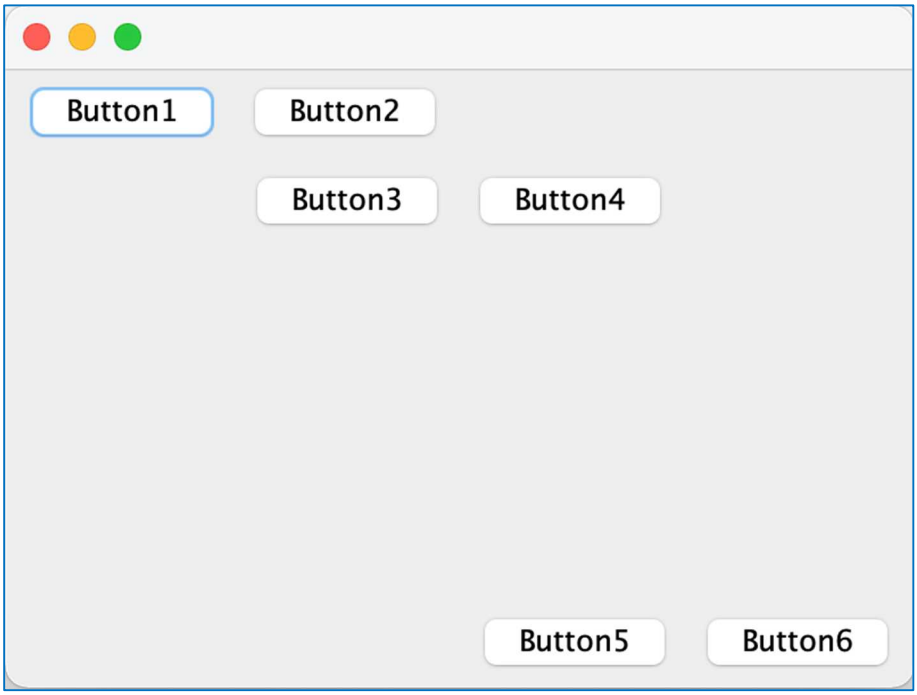

Figure 5. The Result of program of FlowLayout.

6. Event Processing

This chapter introduces button manipulation methods using event handling introduced in “AWT Event Handling” (The sixth day).

6. 1 Changing label of button

In order to process when a button is clicked or pressed, the “addMouseListener” method is used to register the event processing in the button. The text of the button is rewritten for each press. The results of an example are shown in Figure 6 (a) to (c). The program source of an example program is shown in Figure 7.

In the initial state shown in Figure 6 (a), when the button is pressed once, it becomes the state shown in Figure 6 (b). Furthermore, pressing the second time makes the state shown in Figure 6 (c).

(a) (b) (c)

Figure 6 (a) initial state, (b) the state after the button is pressed once, (c) the state after the button is pressed twice.

Example73.java

```java
/*
 * An example of button and event processing
 * by Hitoshi Ogawa on November 8th, 2021
 */
import java.awt.BorderLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Example73 extends JFrame {
    JButton button1;
    int count = 0;

    public Example73() {
        setSize(400,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        button1 = new JButton("initial state");
        button1.addMouseListener(new ButtonListener1());
        JPanel panel1 = new JPanel();
        panel1.add(button1);
        getContentPane().add(panel1, BorderLayout.NORTH);

        setVisible(true);
    }

    public static void main(String[] args) {
        new Example73();
    }

    class ButtonListener1 extends MouseAdapter{
        public void mousePressed(MouseEvent e){
            if(++count > 1) {
                button1.setText("Press " + count + " times");
            } else {
                button1.setText("One press");
            }
        }
    }
}
```

Figure 7. An Example program of Button Event Processing.

6. 2 Changing textfield

In Section 6.1, the display of the button was changed every press. In this section, an example program of displaying in the textfield instead of displaying on the button is shown in Figure 8. The result of the program is shown in Figure 9.

Example74.java

```java
/*
 * An example of button and textfield
 * by Hitoshi Ogawa on November 8th, 2021
 */
import java.awt.BorderLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Example74 extends JFrame {
    JButton button1;
    JTextField text1;
    int count = 0;

    public Example74() {
        setSize(400,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        button1 = new JButton("button1");
        button1.addMouseListener(new ButtonListener1());
        text1 = new JTextField("textfield", 10);
        JPanel panel1 = new JPanel();
        panel1.add(button1);
        panel1.add(text1);
        getContentPane().add(panel1, BorderLayout.NORTH);

        setVisible(true);
    }

    public static void main(String[] args) {
        new Example74();
    }

    class ButtonListener1 extends MouseAdapter{
        public void mousePressed(MouseEvent e){
            if(++count > 1) {
                text1.setText("Press " + count + " times");
            } else {
                text1.setText("One press");
            }
        }
    }
}
```

Figure 8. An example program of displaying in the textfield.

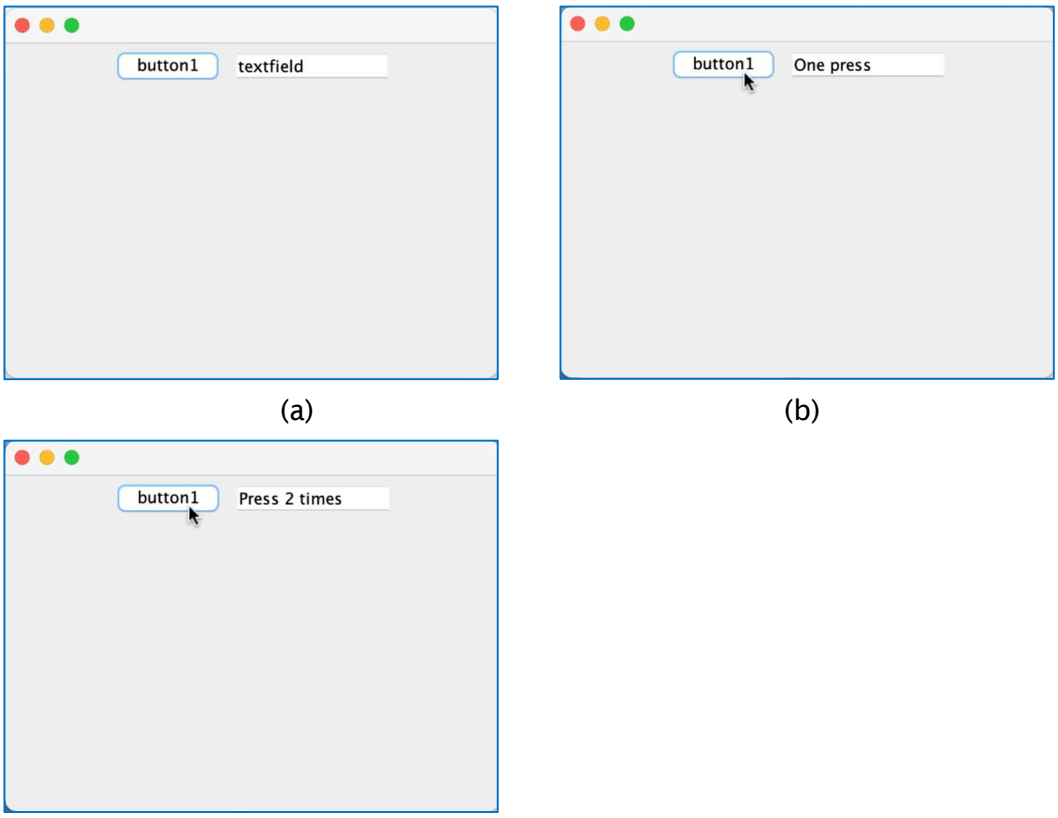

Figure 9. (a) initial state, (b) the state after the button is pressed once, (c) the state after the button is pressed twice.

6. 3 Reading textfield

A program example of displaying a character string described in a text area on a window is shown in Figure 10.

When the button is pressed, a string is acquired from the textfield and substituted into the variable str (line 43). The getGraphics method is used to get Graphics object (line 44). The font is defined (lines 46 and 47) and color is set (line 48). Then the content of str are displayed on the window (line 49). See Figure 11.

Example75.java

```java
/*
 * An example of reading from textfield
 * by Hitoshi Ogawa on November 8th, 2021
 */
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Example75 extends JFrame {
    JButton button1;
    JTextField text1;

    public Example75() {
        setSize(400,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        button1 = new JButton("button1");
        button1.addMouseListener(new ButtonListener1());
        text1 = new JTextField("", 20);
        JPanel panel1 = new JPanel();
        panel1.add(button1);
        panel1.add(text1);
        getContentPane().add(panel1, BorderLayout.NORTH);

        setVisible(true);
    }

    public static void main(String[] args) {
        new Example75Key();
    }

    class ButtonListener1 extends MouseAdapter{
        String str;

        public void mousePressed(MouseEvent e){
            str = text1.getText();
            Graphics g = getGraphics();
            g.clearRect(0, 70, 400, 100);
            Font font1 = new Font("Dialog", Font.PLAIN, 30);
            g.setFont(font1);
            g.setColor(Color.blue);
            g.drawString(str, 50, 100);
        }
    }
}
```

Figure 10. An example of displaying a character string described in a text area.

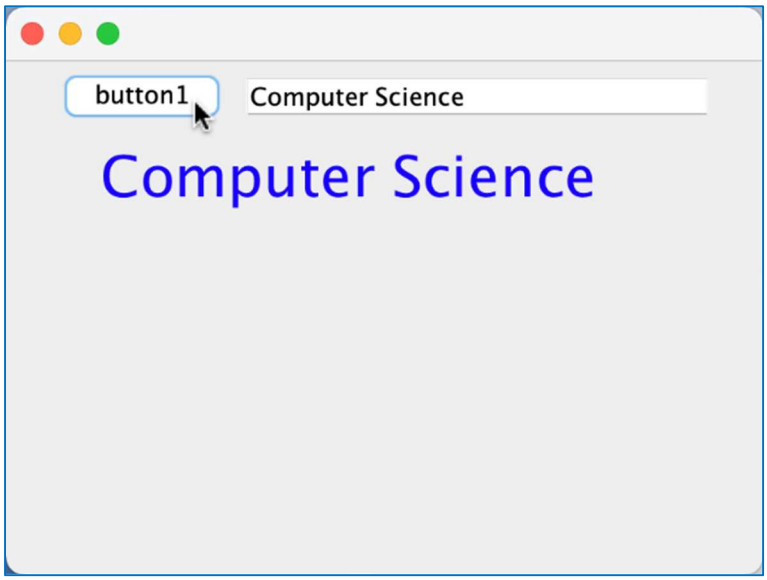

Figure 11. The result of Example75.java