HANDOUT ON THE VISUAL STACK APPLET FOR CSE 373

Steve Tanimoto
University of Washington

1 Introducing the Visual Stack Applet

Let us now take a look at a computer program that illustrates the operation of a stack. First, we'll try operating the program. Then, we'll examine the Java code for it.

Although one purpose of this applet is to make it clear what a stack does, a more important goal is to show how the program itself works to implement and illustrate a data structure. The program follows a "visual applet" methodology that offers a combination of pedagogical power and implementation simplicity that we'll use often in this book.

2 Using The Visual Stack Applet

You can start up the Visual Stack Applet by firing up any Java-enhanced browser (such as Mozilla, Netscape, or Microsoft Internet Explorer) and going to the web page: `http://ole.cs.washington.edu/EDSA/VisStackApplet.html`.

You interact with the applet by entering textual commands into the command box and clicking the Execute button. The applet loads up with an example sequence of commands that you can run immediately.

The commands are probably self-explanatory, but here is a brief description. There are essentially two kinds of commands: commands that correspond to operations on the data structure itself, such as PUSH and POP, and commands to control the presentation or the reporting. For example, the DELAY command sets the number of milliseconds that the applet will wait after performing one operation before starting the next. The STATS command causes some performance information to be printed into the history window. If you would like to see the contents of the history window, click on the History button.

3 How it Works

Let us now examine the Java code for the Visual Stack Applet in detail. This code can be easily edited to create visual applets for many other data structures and algorithms. There are just a few slighly tricky Java features used here that might not be completely obvious if you are new to them. These include the use of a ScrollPane object for the main graphics area, and the use of a separate Thread object for handling the animation of the data structure.

The source file for this program is available in machine-readable form at the following URL:

`http://ole.cs.washington.edu/EDSA/VisStackApplet.java`.

All the code is shown below, but it is broken up by explanatory text. You can tell the difference between the code and the text by the fonts in which they are printed: the code is in `Courier` while the explanatory text is in Times Roman.

The file starts off with some Java import commands. The program uses classes in the Swing, AWT, AWT Event, and Utilities libraries.

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

Next is some documentation, in the JavaDoc format. Following that is the beginning of the definition of the VisStackApplet class itself, which is defined to be a subclass of JApplet.

```
/*
 * VisStackApplet is
 * an applet that can be modified to produce an interactive
 * demonstration for a data structure. Written for CSE 373,
 * it is currently set up to show a stack.
 * It takes textual commands in a TextArea and when the user
 * clicks on the Execute button, it processes the commands,
 * updating the display as it goes.
 *
 * @Version of June 14, 2004
 * @author Steve Tanimoto, Copyright, 2004.
 *
 */
public class VisStackApplet extends JApplet
  implements ActionListener, Runnable {

  ScrolledPanel visPanel; //Where to paint graphics
  MyScrollPane msp;
  Button executeButton;
  Button historyButton;
  TextArea userInputText;
  TextArea history;
  JFrame historyFrame;
  JTextField statusLine;
  MyStack theStack; // The data structure being demonstrated
  Font stackFont;
  int cellHeight = 20; // For drawing the stack.
  int cellWidth = 200; // How wide to plot pink rectangles
  int cellGap = 4; // vertical space between successive cells
  int topMargin = 25; // Space above top of stack.
  int fontSize = 16; // Height of font for displaying stack elements.
  int leftMargin = 20; // x value for left side of cells
  int bottomMargin = 10; // Minimum space betw. bot. of
    // visPanel and bot. of lowest cell.
  int leftOffset = 5; // space between left side of cell
    // and contents string.
  int delay = 300; // default is to wait 300 ms between updates.
  Thread displayThread = null;
```

The applet initialization method is defined next. Most of the code for it creates the various graphical user interface components such as the scrolled panel and the two buttons.

```
public void init() {
  setSize(300,300); // default size of applet.
  visPanel = new ScrolledPanel();
  visPanel.setPreferredSize(new Dimension(400,400));
  msp = new MyScrollPane(visPanel);
  msp.setPreferredSize(new Dimension(400,200));

  Container c = getContentPane();
  c.setLayout(new BorderLayout());
  c.add(msp, BorderLayout.CENTER);
  JPanel buttons = new JPanel();
  buttons.setLayout(new FlowLayout());
  JPanel controls = new JPanel();
  controls.setLayout(new BorderLayout());
  executeButton = new Button("Execute");
  executeButton.addActionListener(this);
  buttons.add(executeButton);
  historyButton = new Button("History");
  historyButton.addActionListener(this);
  buttons.add(historyButton);
  userInputText = new TextArea(";Enter commands here.");
  statusLine = new JTextField();
  statusLine.setBackground(Color.lightGray);
  controls.add(buttons, BorderLayout.WEST);
  controls.add(userInputText, BorderLayout.CENTER);
  controls.add(statusLine, BorderLayout.SOUTH);
  controls.setPreferredSize(new Dimension(400,100));
  c.add(controls, BorderLayout.SOUTH);
  c.validate();

  theStack = new MyStack();
  stackFont = new Font("Helvetica", Font.PLAIN, 20);
  history = new TextArea("VisStackApplet history:\n", 20, 40);
}
```

The scrolled panel itself will be responsible for painting the data structure after each operation. The browser will call the applet's `paintComponent` method, which in turn will call the scrolled panel's `paintComponent` method, which is defined here. It first paints its background in the standard JPanel way, and then it calls `paintStack` to actually render the stack data structure that is featured in the program.

```
class ScrolledPanel extends JPanel {
```

```
    public void paintComponent(Graphics g) {
      super.paintComponent(g);
      paintStack(g);
    }
  }
```

The following code customizes the built-in class `JScrollPane` so that it will always show scroll bars.

```
  class MyScrollPane extends JScrollPane {
    MyScrollPane(JPanel p) {
      super(p,
        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    }
  }
```

The stack is implemented internally using Java's `Vector` class. This is convenient, but the Vector is a somewhat heavyweight class to use for implementing a stack.

```
  class MyStack extends Vector {

    int n; // number of elements in the stack
    int npushes; // number of PUSH operations so far.
    int npops; // number of POP operations so far.

    void init() {
      n = 0; npushes = 0; npops = 0;
    }

    void push(Object elt) {
      add(n, elt);
      n++;
      npushes++;
    }

    Object pop() {
      if (n == 0) { return null; }
      Object o = lastElement();
      n--;
      npops++;
      remove(n);
      return o;
    }
  }
```

The following method, `actionPerformed`, provides an implementation of the `ActionListener` interface, and it tells the Java environment what to do if the user clicks on either of the two buttons.

```
  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("Execute")) {
      displayThread = new Thread(this);
      displayThread.start();
      return;
    }
    if (e.getActionCommand().equals("History")) {
      if (historyFrame == null) {
        historyFrame = new JFrame("History of the VisStackApplet");
        historyFrame.getContentPane().add(history);
        historyFrame.setSize(new Dimension(300,300));
      }
      historyFrame.show();
      System.out.println("Should have displaals("SIZE")) {
      String stats = "Current number of elements: " +
                     th -= (cellHeight + cellGap);
    }
  }
```

Next is a method that examines the number of elements in the stack and figures out whether the whole stack will fit on the scrolled panel. If not, it enlarges the scrolled panel by just the right amount.

```
/**
 * The following computes the height of the display area needed by the
 * current stack, and if it won't fit in the scrolled panel, it enlarges
 * the scrolled panel.  In the current implementation, the panel never
 * gets smaller, even if the stack becomes empty.  This could easily be
 * changed.
 */
  void checkScrolledPanelSize() {
    int heightNeeded = topMargin + theStack.n * (cellHeight + cellGap)
                       + cellHeight+ bottomMargin;
    Dimension d = visPanel.getPreferredSize();
    int currentHeight = (int) d.getHeight();
    int currentWidth = (int) d.getWidth();
    if (heightNeeded > currentHeight) {
      visPanel.setPreferredSize(new Dimension(currentWidth,
                                              heightNeeded));
      visPanel.revalidate(); // Adjust the vertical scroll bar.
    }
  }
}
```

The source file also includes, for the developer's convenience, an actual command sequence that can be copied and pasted into the command box of the applet to test it. When developing an applet like this, it is helpful to create a test sequence like this before actually writing the code, so that it can help guide the implementation and testing from the very beginning.

```
 /*
A sample command sequence for the applet is given below.
 PUSH John
 PUSH Mary
 SIZE
 POP
 STATS
 RESET
 ; This is a comment - we are beginning with a new stack.
 DELAY 500  ; from here wait only 500 ms between updates
 PUSH 3.14159
 PUSH 25
 PUSH 13
 STATS
 */
```

4 Exercises

1. What is the purpose of the history window in the Visual Stack Applet?

2. Name at least three different statistics that could be reported for a session involving the use of a stack.

3. Give a mathematical formula that tells how much space is required to display a stack containing n data elements. Assume that the space is described by two values: width and height, both of which are given in units of pixels.

4. Modify the Visual Stack Applet so that it can be used as a calculator for mathematical expressions given in a sort of postfix form. For example, the formula $(7 + 2) * (15 - 9)$ would be written

   ```
   PUSH 7
   PUSH 2
   DO +
   PUSH 15
   PUSH 9
   DO -
   DO *
   POP
   ```

and the result returned would be 54. The DO command pops two arguments off the stack, performs the requested arithmetic and pushes the result back on the stack.

5. Modify the Visual Stack Applet so that the stack is displayed horizontally instead of vertically, and make the top of the stack be at its right end.

6. Do the previous exercise except make the top of the stack be at its left end.

7. Modify the Visual Stack Applet so that the stack's top is located at the bottom of the display.

8. Modify the Visual Stack Applet so that it contains two stacks, A and B. Each command should then take one more argument than in the original applet. For example,

   ```
   PUSH A 15
   ```

   should push the value 15 onto the first stack. The scrolled display panel should always be high enough that the larger of the two stacks is accommodated. That way, both stacks are fully displayed.