Java - Applications

C&G Criteria: 5.3.2, 5.5.2, 5.5.3

Java applets require a web browser to run independently of the Java IDE. The Dos based console applications will run outside the IDE but have no access to the Java awt controls. Java can produce Windows applications. The code required is very similar to that for applets.

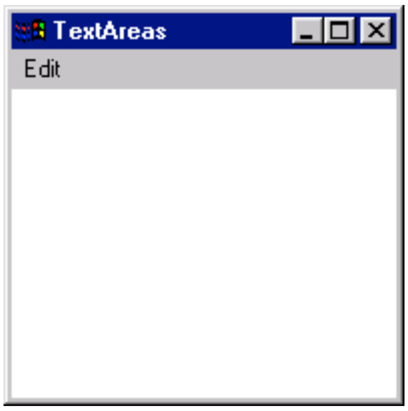
There are the following key differences

- No access to the Applet Class, removing some methods such as getCodeBase and getImage.
- Code has to be written to close the application window - clicking the little 'x' is not enough.
- Applications allow menus, applets do not.
- The default layout is Borderlayout for an application and FlowLayout for an applet
- The dreaded public static void main(String[] args) is back.

The following code sets up an application with a drop down menu, as yet the menu does not do anything.

```
import java.awt.*;
import java.awt.event.*;  //can't import applet

public class Paste extends Frame implements WindowListener, ActionListener
{ //WindowListener is used to respond to window events

        private MenuItem copy, cut, paste;

        public static void main(String [] args)
        {  //main is inside the Class - the Class is created as a new Frame Object
                Frame f = new Paste();
                f.setSize(200,200);
                f.setVisible(true); //create somewhere to show the class, size 200, 200
        }
        public Paste()  //the constructor class holds the data that would be in init()
        {       //of an applet
                setTitle("TextAreas");  //caption of the application
                setLayout(new FlowLayout()); //the default layout is BorderLayout

                this.addWindowListener(this); //need to register window actions
                                              //compare to mouse actions
                MenuBar menubar = new MenuBar(); //menu heading
                Menu edit = new Menu("Edit");  // a single drop down

                copy = new MenuItem("Copy");//add individual menu items
                edit.add(copy);
                copy.addActionListener(this);
                //add a cut and a paste item as well
```

```
		menubar.add(edit); //add edit to the MenuBar
		setMenuBar(menubar); //set up the MenuBar

	}
	public void actionPerformed(ActionEvent event)
	{
		//event handling code to put here
	}
	public void windowClosing(WindowEvent event)
	{//allow closing of window
		System.exit(0);
	}
	//like the mouse actions these are required for Java events that might not be used
	public void windowIconified(WindowEvent event){}
	public void windowOpened(WindowEvent event){}
	public void windowClosed(WindowEvent event){}
	public void windowDeiconified(WindowEvent event){}
	public void windowActivated(WindowEvent event){}
	public void windowDeactivated(WindowEvent event){}
}
```

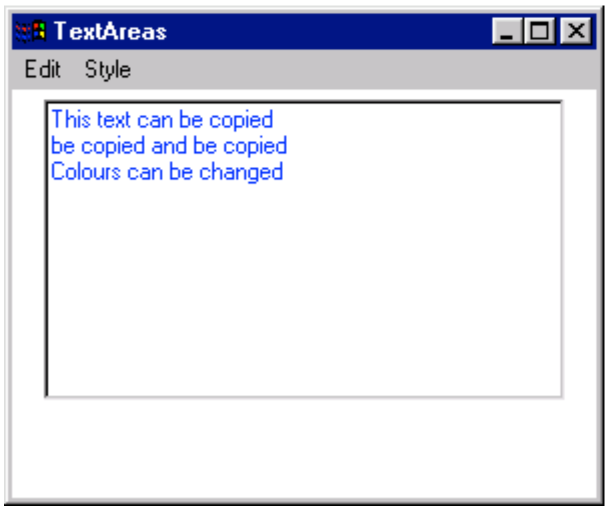


Enhancements

1. Add a TextArea to the application, this will hold text to try out the copy and pasting code. The new code will be in the actionPerformed events for the menus. Although a TextArea is set up with a number of columns and rows for the text Java allows any text entered to scroll off the viewing area.

The editing will depend on some of the methods of the TextArea, getSelectedText(), getSelectionStart(), getSelectionEnd(),replaceRange(). The string to be cut, copied or pasted will have to be remembered between menu clicks so will be declared at the top of the class with the awt controls.

2. The colour of the text could be changed from another menu drop down. This requires a similar line to Menu edit = new Menu("Edit"); that will add the new menu header. New MenuItems are added to that Menu. Their ActionEvents will change the font colour in the TextArea through - setForeground().



This text can be copied
be copied and be copied
Colours can be changed

3. Handling font styles and sizes from a menu is less than ideal as the user would prefer to pick from a range of font styles and sizes. This will be achieved with the drop down control that Java calls Choice. At this stage the Choice controls will be set up but they will not do anything.

Adding the Choice controls is the same as adding a Button, declare then create an object with new. The application will have 2 Choices and a Label for the size Choice. It will soon be evident that the style Choice is not going to need a Label. To tidy up the application the layout will use flow and BorderLayout with the TextArea in Center and the Choices in North with the Label.

So the new controls will be part of a new Panel that is destined to be at the BorderLayout position North.

```
Panel p = new Panel();    //create new Panel
p.setLayout(new FlowLayout()); //the new controls flow on the Panel
styles = new Choice(); //create new controls
p.add(styles); //add the new controls to the panel

add(p, BorderLayout.NORTH); //add panel to top
add(input, BorderLayout.CENTER); // add TextArea to centre
```

The new Choice controls will be populated using for loops. The font size is straightforward as any suitable range of numbers can be used. Have the loop generate a range of integers, convert these to Strings and add to the Choice:

```
myChoice.add(myStringFontSize);
```

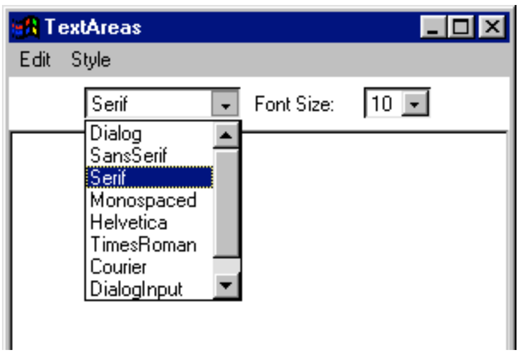
There is a cunning trick for the font style as Java knows what fonts it has access to and we can tell it go and get them. This requires use of Toolkit object, this is accessed in the following fashion:

```
Toolkit tk = Toolkit.getDefaultToolkit();
```

The resulting font names can then be assigned to a String array

```
String[] fontList = tk.getFontList();
```

A for loop will add these Strings to a Choice control, fontList.length can be used to stop the loop without going out of bounds of the array.

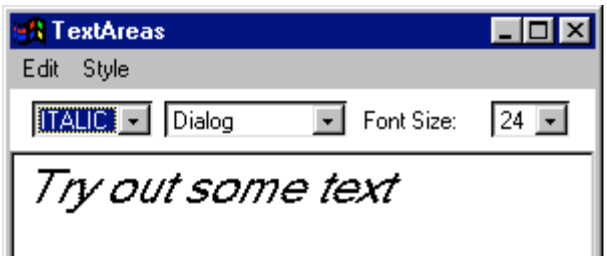


4. Next the fancy new controls need to act on the TextArea. The Choice controls do not use actionPerformed(ActionEvent event) but itemStateChanged(ItemEvent e). This requires an additional implements for the class, ItemListener and the associated listener for the controls such as sizes.addItemListener(this).

As the selection of either Choice will have the same effect, a change in font a single if statement can cover both e.getSource() possibilities. The aim is to get the font name and size from both Choices. The font name is a String as is the text in both Choices. For the font size the String selected will have to be changed to an integer. When declaring a font the weight (bold, normal) is also required; use Font.PLAIN to fill in the specification.

5. Allow the user to set up font weights with another Choice control. The value used to set a font weight is an integer, plain is 0, bold is 1 and italic is 2. Take advantage of this by setting up the Strings in the weight Choice in that order. The font weight can then be found from:

int iType=chWeight.getSelectedIndex();



File I/O

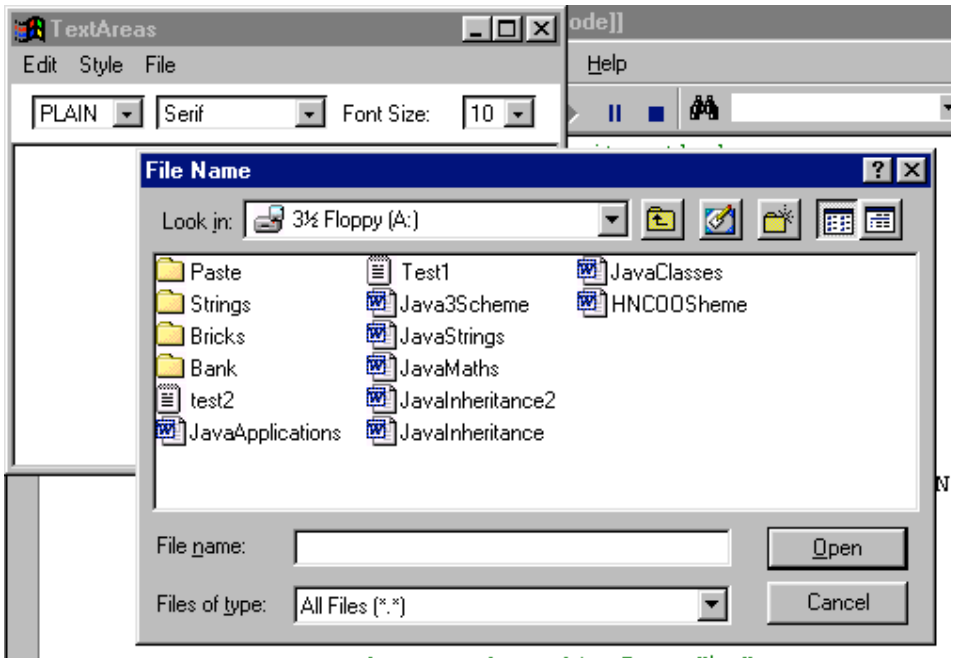
The TextArea can be used to display the contents of a text file or the data in the TextArea could be saved to a file. To trigger these events the application will need another Menu (file) with MenuItems for opening and saving files.

Java has an awt component called the FileDialog that is just dandy for inputting file names for later use. The FileDialog has to be declared like any other component:

private FileDialog getName;

It can then be created and used in the appropriate actionPerformed event. At this stage the file name is passed to a String getName for future use.

```
if(event.getSource()==open)
{
        getName=new FileDialog(this,"File Name",FileDialog.LOAD);
        // FileDialog.SAVE for saving a file name
        getName.setVisible(true);       //no drop downs for these but code works
        fileName=getName.getFile();
}
```



Unfortunately the getFile method only returns the name of the file. As long as the file is in the same directory as the java program is running from this does not matter. For more useful file browsing the directory will have to be found and appended to the file name.

```
String dir = getName.getDirectory();
fileName=dir + getName.getFile();
```

To work with files requires importing java.io above the class header:

```
import java.io.*;
```

As messing with files is a useful operation all the file handling can be part of a method that accepts a String containing the file name and path and returns a String with the file contents.

```
public String openFile(String name)
{
        BufferedReader inputFile;//for file i/o
        String result="";//must initialise
        try
        { //FileReader reads the file into BufferedReader so 2 new objects
```

```
                inputFile=new BufferedReader(new FileReader(name));
                String lineIn;
                lineIn= inputFile.readLine();
                //the BufferedReader method to read the file line by line
                while(lineIn!=null)  //while lineIn still reads data
                {
                        result= result + lineIn + "\n";   //add a new line to the string
                        lineIn= inputFile.readLine();  //read the next line
                }
                inputFile.close();  //close BufferedReader
        }
        catch(IOException e)  //catch cock ups, such as file as not found
        {       //print error
                result="Error " + name + e.toString();
                //System.exit(1);  //use to quit program
        }
        return result;
}
```

It is worth implementing this code and opening a text file that has been created in Notepad before creating code to save files.

Saving files should also be set up as a method. This will involve less code as writing to disk does not have to move down line by line. It chucks the lot in and picks up the \n symbols as it goes along. There are some key differences.

- Instead of BufferedReader, use PrintWriter

```
PrintWriter Reader outputFile=new PrintWriter(new FileWriter(fileName),true);
```

FileWriter is created with the file name and the option true to flush the system buffer.

- A single line will assign all the required text to the PrintWriter object.

```
outputFile.print(text);
```

The method to save files will have to accept the name of the file to be saved and the data to be saved. These are both Strings. The method need not return anything but it would be useful if it returned any error messages generated by failed write operations. The following is a suitable file saving method title:

```
public String saveFile(String fileName, String text)
```

Go Objects

Create a new Class to save or open files. The 2 methods created above can be cut and pasted into the Class. The return String result, the BufferedReader and PrintWriter objects can all be private, java.io.*; will move to the new Class. Create an object of the new Class in the original application and use its public methods to save and open files.