Java – String Manipulation

C&G criteria: 2.1.3

The String Class has a number of methods that allow manipulation of the elements within a String object. It cannot do everything but useful methods are available within other Classes such as StringBuffer. The required operation can be a method of the StringBuffer Class with the result assigned to a String object.

String Class

Strings are objects but they also have some of the characteristics of simple variables in that new is not required to initialise a String (optionally it can be used) and Strings can be added together almost like numbers.

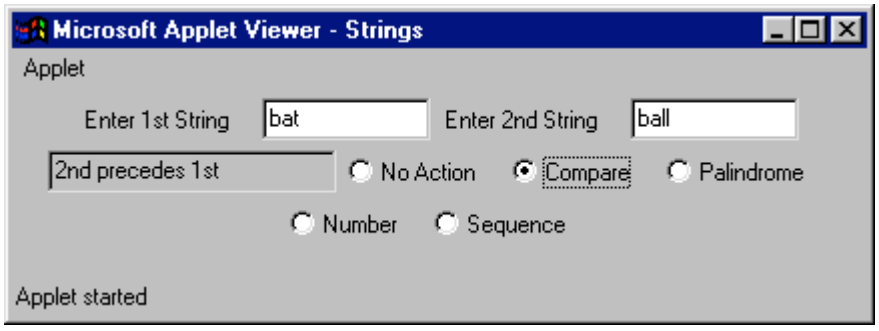
```
String sName = "Java";  // initialise new String with data
String sUse;    //String has no value
sUse = " is useful";
sName = sName + sUse;  //sName now equals "Java is useful"
```

To compare Strings = = will not work, this operator looks at the names of the String Objects

```
sName = = sName; //true
sName = " is useful";
sName = = sUse; //false although they hold the same data
```

Methods of the String Class can be used to overcome this problem

- equals // does what = = ought to do
- equalsIgnoreCase // ignore upper or lower case
- compareTo // returns 0 for equal Strings, <0 for 1 String alphabetically before the other // >0 for 1 String alphabetically after the other.



The above applet allows various String manipulation methods to be tested. The Checkbox clicked defines which operation is performed. No action is used as a default. Initially code the applet with active Compare and No Action Checkboxes. The grey TextField does not allow data to be edited by the user. Unlike a Label the output String can be changed at run time, unlike a default TextField the user cannot overtype this output.

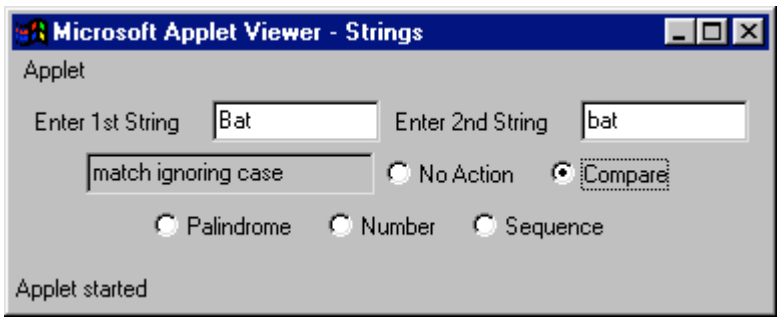
```
output.setEditable(false);  //TextField is called output
```

This is an outline of the user interface code.

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import java.util.*;//for StringTokeniser – used later

public class Strings extends Applet implements ActionListener, ItemListener
{
	private TextField input1, input2, output;
	private Label info1,info2;
	private CheckboxGroup options;
	private Checkbox none,compare;  // add other objects later

	public void init()
	{//set up awt components
		info1 = new Label("Enter 1st String");
		add(info1);
		input1 = new TextField(10);
		add(input1);
		input1.addActionListener(this);
		//input2 and info2 are much the same
		output = new TextField(20);
		add(output);
		output.addActionListener(this);
		output.setEditable(false);//don't allow changes

		options = new CheckboxGroup();
		none = new Checkbox("No Action",options,true);
		add(none);
		none.addItemListener(this);
		compare= new Checkbox("Compare",options, false);
		add(compare);
		compare.addItemListener(this);
		// other controls to add
	}
	public void actionPerformed(ActionEvent e)
	{ // code to test interface
		if(e.getSource()==input2)//fires on enter/return of 2nd TextField
			output.setText("Test data");
	}

	public void itemStateChanged(ItemEvent e)
	{
		String source = input1.getText();
		String comp = input2.getText();
		if(e.getSource()==compare)
		{
			int order = source.compareTo(comp);
			if(order==0)
```

```
                    output.setText("Strings are the same");
                if(order<0)
                    output.setText("1st precedes 2nd");
                if(order>0)
                    output.setText("2nd precedes 1st");
            }
            // handle other events here

}
```

The code above will compare 2 Strings input by the user. Modify the code to allow for case sensitive comparisons as well as the alphabetical order. Note that compareTo will not return 0 if the 2 Strings contain the same text but differ in case.

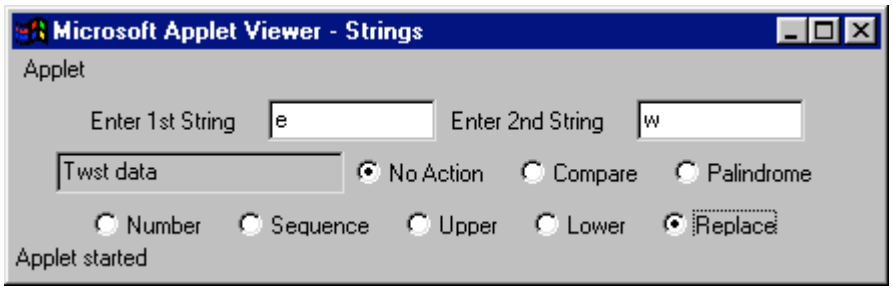


Create CheckBoxes to manipulate the data output. These will be methods acting on the TextField output.

- Set the output String to upper case – String result = String.toUpperCase();
- Set the output String to lower case – String result = String.toLowerCase();
- Replace all instances of 1 character with another – String.replace(char,char);

The 1st 2 cases are relatively simple. The 3rd will use characters from the TextFields input1 and input2. These TextFields will contain Strings not characters, even a single letter will be interpreted as a String. This can be overcome by the String method charAt(int) with the 1st position of a String being 0.

char c1 = source.charAt(0);



The palindrome action checks to see if 1 String is the same backwards and forwards (Bolton and Notlob). This action only requires the use of 1 TextField. The input String is taken apart 1 element at a time from the back and these characters are assigned to another String. If this String and that originally input are the same we have a palindrome.

The length of a String is found from String.length(). As the initial element of a String is 0 the final element will be String.length()-1. A for loop can start at this final element and assign each element in turn to the temporary String for comparison.

The number option assumes that the String represents an integer or a float. The key difference will be a '.' In some part of the float alone. Only a single TextField is used for data entry. The code uses a for loop to cycle through the String 1 character at a time. If a single '.' Is found the data is identified as a float. Otherwise an integer is present.

StringTokenizer Class

This Class is useful where a long String needs to be broken up into smaller Strings such as breaking up the data in a single database field into several distinct fields. The StringTokenizer requires an initial String and a delimiter, some character that is used to show where 1 part of the String begins and another ends. Possible delimiters are commas or spaces although any repeating character can be used. The following might be the contents of a single database field address;

House, street, town, post code

As long as all the address fields contain the same groups of data it is relatively easy to convert the String address to distinct Strings house, street, town, postcode.

```
String address =  "House, street, town, post code";
String field;
StringTokenizer source = new StringTokenizer(address," ,");
while(source.hasMoreTokens())
{
        field = source.nextToken();
        // pass field to other variables
}
```

Use the StringTokenizer Class to add up the contents of input1 where the data entered is a series of figures and + or – signs.

1 + 2 + 3 or 3 + 2 – 2

The token to be used is the " " between figures or symbols. The symbol will decide on the mathematical operation. As the figure involved occurs before the symbol it will be necessary to record the last token as well as the current token. The result can be sent to the TextField output.

StringBuffer Class

The StringBuffer Class allows much the same data manipulation as the String Class with some extra bells and whistles. To use the Class a String must be converted to a StringBuffer. Having finished manipulation the StringBuffer will have to be converted back to a String to be much use. Part of the StringBuffer can also be converted to a distinct String.

StringBuffer has been beefed up in Java 2 allowing a number of actions that VJ++ will not run as it supports Java 1.1. For example this code to replace part of a String with another String will only work in Java 2.

StringBuffer bTemp = new StringBuffer(text);//data from TextArea
bTemp.replace(2,4,"new word"); //places the new String between position 2 and 4
//data at position 3 is lost
field.setText(bTemp.toString()); //set String back to TextArea field

After a brief digression to Java 2, there are some things that 1.1 can do with StringBuffer. The Class allows 1 String to be appended to the existing StringBuffer. This is not a great deal more exciting than appending 1 String onto another.

StringBuffer bTemp = new StringBuffer(text);//data from TextArea
bTemp.append(" add some data");
field.setText(bTemp.toString());

SetLength will set a maximum length for the StringBuffer. Any additional characters are chopped off.

bTemp.setLength(3); //"A String" becomes "A S"

Add a TexTArea to the Strings Class and use it to test the use of the append StringBuffer method.