Applets in Java using NetBeans as an IDE (Part 1)

C.W. David
Department of Chemistry
University of Connecticut
Storrs, CT 06269-3060
Carl.David@uconn.edu

We are interested in creating a teaching/testing tool for the WWW using applet technology implemented using Java. You can see from Illustration 1 what we are aiming at:

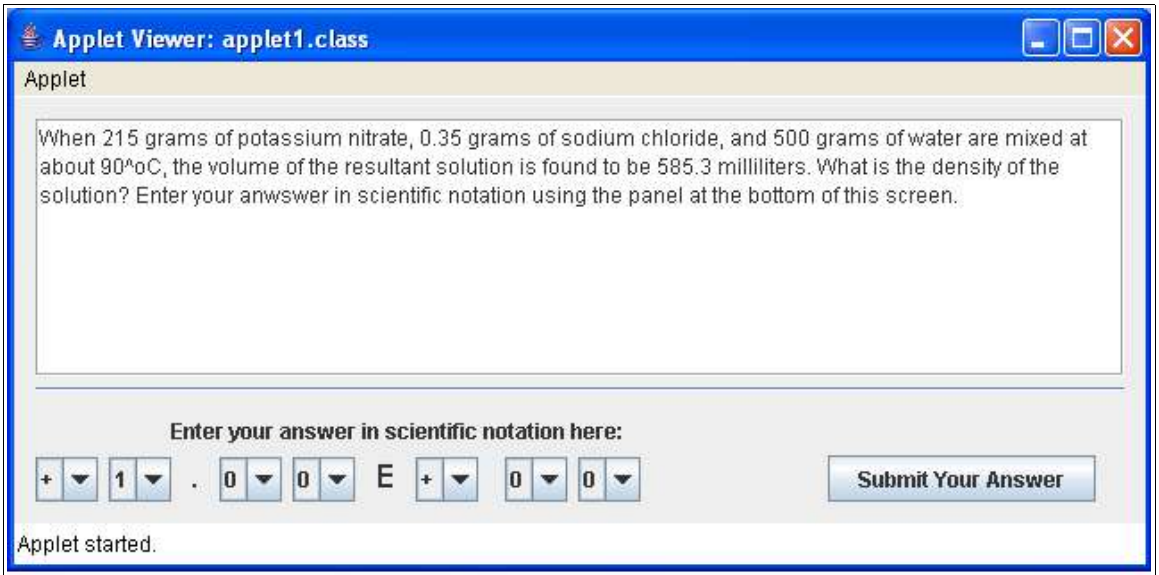

Illustration 1: An applet showing constructed responses in an intuitive mode. Note the misprint!

Clearly, we are aiming for what is called a constructed response, not a multiple choice, method for answering. Notice that the text contains 90ºC which failed (using LaTeX notation), and 90^o fails also. This means that whatever we've done, *vide ante*, we are not interpreting HTML properly (more on this later). In all that follows, we

will be using NetBeans 1.5, and Java 1.5. There are other IDE's available, but this one has freeform GUI construction, which turns out to be incredibly intuitive.

If you find any errors or omissions, or want clarifications added, please e-mail me at Carl.David@uconn.edu, and I will make the appropriate adjustments.

Criteria for the Project:

Our criterion for ultimate success in this project is to use HTML properly, so that chemical subscripts and superscripts can be included (as well as the degree symbol, etc.). In this first example, the text is embedded in the question, and our second goal is to obtain the text from a web site in a public_html directory accessible to teachers who are willing to edit HTML text for their questions. Of course, we need to know (or generate) the correct answer, and compare it to the student answer, and we need to create a strategy for what to do when the student submits an incorrect answer. But first, we analyze this example as best we can.

Creating an Applet:

We start with creating an applet:

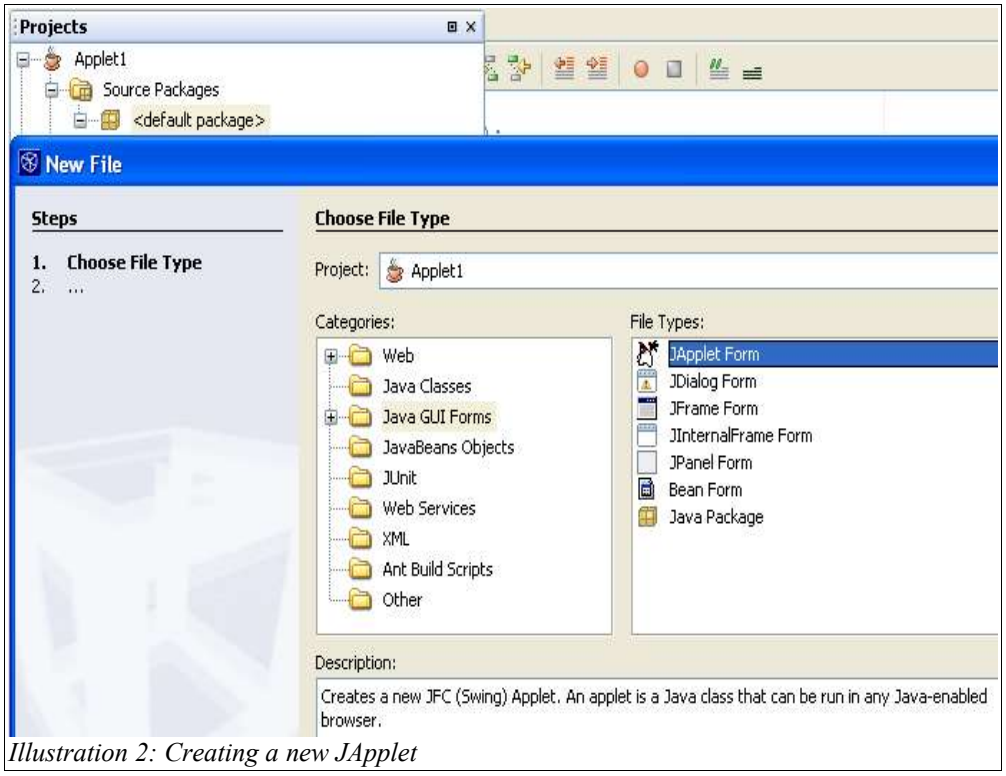

Illustration 2: Creating a new JApplet

we've started with an existing Project, and found the sources folder, and right clicked there to get choice, and as novices, we try new, which allows us to see a Description field

for any choice to be made, helping us.

We choose a JApplet, and give it a name, i.e., follow the menus and choices as they appear. IN the code snippets which follow, lots will have been removed for the sake of clarity. It is most important, when a symbol from the API's can't be found, that you check the import statements, which, for me, grow as I work, i.e., I don't start with

```
import javax.swing.JTextArea;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.math.*;
import java.util.Formatter;
```

but this is what I ended up with after coding (there is a clean up of imports in the IDE, which one can employ later.

To continue, we go to the Design menu and start adding components from the Palette. This generates code which occurs in the hidden, uneditable, part of the source, and most of the time you needn't look at it. We start with the ± symbol (we will need two of these, one for the 3 digit part and one for the power of ten part).

First, we have to set up the layout, and the easiest in NetBeans1.5 is the free form one. Right clicking on the brown form in the design mode one gets:

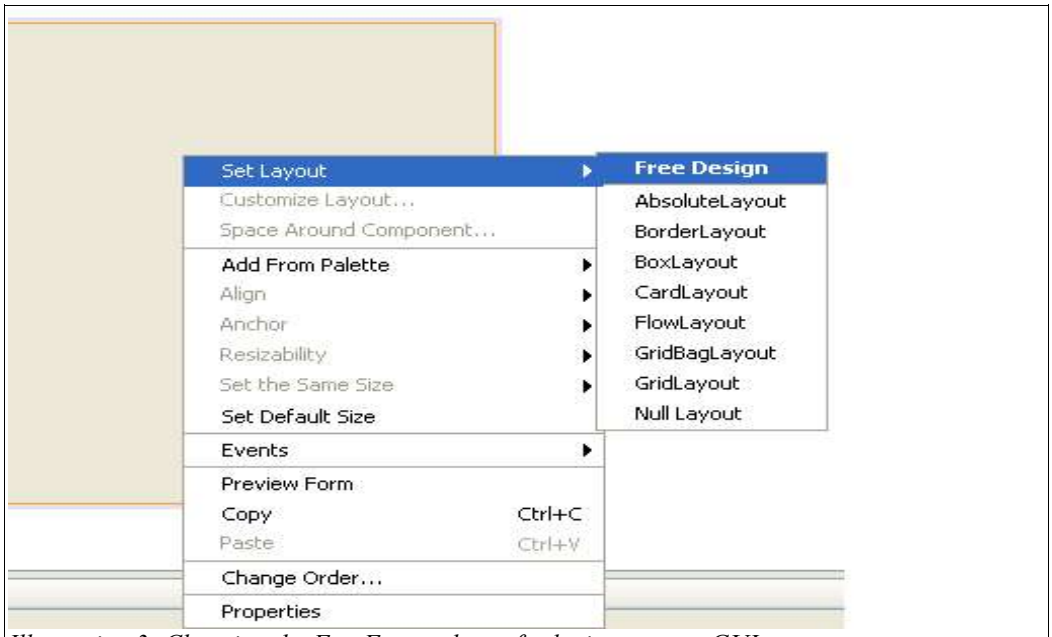

Illustration 3: Choosing the FreeForm scheme for laying out our GUI

and we choose Free Design, which is superb!

Next, we click on a JComboBox in the palette and then click on the emerging form. Lo and behold, a JComboBox appears; its temporary name is jComboBox1. We

see:

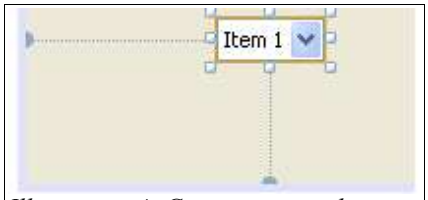

Illustration 4: Creating a graphic element from the Palette

and a little experimentation will yield that we can move this around to where we want it. Right click on it, and choose Properties (at the bottom of the pop-up menu). You will see:

Illustration 5: Editing the items to be shown in the JComboBox

which allows you to edit its **model** "Item 1, Item 2, Item 3, Item 4" into "+ -". Wonderful. We repeat the procedure, adding a second JComboBox, obtaining:

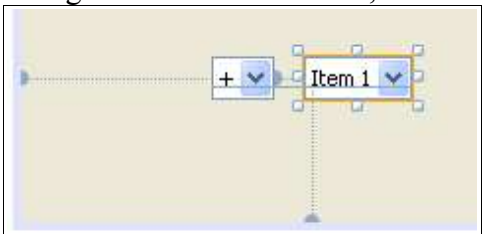

Illustration 6: Adding a second element after editing the first element

Notice that we've aligned the second JComboBox with the first, and we can not edit it's **model** to read :1,2,3,4,5,6,7,8,9. Notice that we've omitted zero, since we want to force the student to use scientific notation, where the first digit must be non-zero.

Next, we add a label, whose text is ".", and then two more JComboBoxes so that a 3 significant figure result can be seen. Continuing with the rest of the boxes and such is obvious, and omitted here.

When we're all said and done, we've got these JComboBoxes and Labels, and we can align them together using point and click, and we can group them and move them where we like. The entire process is just plain incredible.

We need to add a JeditorPane, sized to fill the upper part of the form, and a submit button which will tell the computer to grade the student's answer once it is completely entered. Notice that we do not want to process the individual JComboBox changes as they occur, we want to process them when the appropriate time comes.

To enter the question, we are forced to code under the properties of the JEditorPane ,which shows up in the compressed code as:

```
jTextArea1.setText("When 215 grams of potassium nitrate, 0.35 grams of sodium chloride, and 500 grams of water are mixed at about 90^oC, the volume of the resultant solution is found to be 585.3 milliliters. What is the density of the solution?");
```

This means that we are going to have a major problem with this kind of implementation, since we don't want our teachers editing Java code!

So, for the time being, we will live with this implementation, and try a different scheme later. Right now, we want to focus on how to grade the student's response!
In the design view, we now right click on the submit button, and look at its Events. Just clicking on the empty field next to "Action Performed" leads to insertion of code into our source, which we add to under the TODO rubric.

We start with the plus/minus sign, and write

```
JComboBox pm = (JComboBox)getContentPane().getComponent(2);
      String pm_s = (String)pm.getSelectedItem();
```

You may ask, how did I know that it was component #2, and I will tell you, pure and simple trial and error. Phooey! We will return to this inadequate method after we've finished. Any way, we obtain pm, and then convert it to a string so that we can do other things with it in a little while.

After retrieving the contents of the other JComboBoxes, items 3, 5 and 6, we execute the most important statement:

```
 String mantissa = (String)pm_s+i1.getSelectedItem()+"."+i2.getSelectedItem()
+i3.getSelectedItem();
```

which creates a number of the kind "3.14" with the appropriate sign pre-pended. Next we convert this string to a number (double in our case):

```
double mant = Double.valueOf(mantissa).doubleValue();
      //we have just gotten the x.yz (signed) part of a student's answer!
```

We then repeat the coding for the power of ten. After obtaining the power of ten, we create the final student answer:

```
student_answer = Math.pow(10,exponent)*mant;
```

which we report by changing the label which asked for the student's answer to contain his/her last answer as well, i.e.,

```
 String answer_formatted = String.format(Double.toString(student_answer),"%9.2g");
      myLabel1.setText("Enter your answer in scientific notation here; your prior answer was:"+answer_formatted);
```

Here is the code in its entirety:

```
   private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

      JComboBox pm = (JComboBox)getContentPane().getComponent(2);
      String pm_s = (String)pm.getSelectedItem();

      JComboBox i1 = (JComboBox)getContentPane().getComponent(3);
```

```
      JComboBox i2 = (JComboBox)getContentPane().getComponent(5);
      JComboBox i3 = (JComboBox)getContentPane().getComponent(6);
      String mantissa = (String)pm_s+i1.getSelectedItem()+"."+i2.getSelectedItem()
+i3.getSelectedItem();
      double mant = Double.valueOf(mantissa).doubleValue();
      //we have just gotten the x.yz (signed) part of a student's answer!
      JComboBox pm2 = (JComboBox)getContentPane().getComponent(8);
      String pm_s2 = (String)pm2.getSelectedItem();
      JComboBox i4 = (JComboBox)getContentPane().getComponent(9);
      JComboBox i5 = (JComboBox)getContentPane().getComponent(10);
      String exp = (String)pm_s2+i4.getSelectedItem()+i5.getSelectedItem();
      double exponent = Double.valueOf(exp).doubleValue();
      //we have just gotten the exponent of the power of ten.
      student_answer = Math.pow(10,exponent)*mant;
      JLabel myLabel1 = (JLabel)getContentPane().getComponent(1);
      String answer_formatted = String.format(Double.toString(student_answer),"%
9.2g");
      myLabel1.setText("Enter your answer in scientific notation here; your prior answer
was:"+answer_formatted);
   }
```

In order to test our method of judging the student response, we here give some pseudo code which illustrates our attack:

```
correct_answer = 1.22;// this is just for testing purposes!

      if(answer_formatted.equals(correct_formatted)){
         myLabel.setForeground(java.awt.Color.GREEN);
         myLabel.setText("Enter your answer in scientific notation here; your prior
answer,"+answer_formatted+", was RIGHT!");
      }else{
         myLabel.setForeground(java.awt.Color.RED);
         myLabel.setText("Enter your answer in scientific notation here; your prior
answer,"+answer_formatted+", was WRONG!");
      }
```

so that we can compare the student's answer to our own (locally coded for the time being). We can (and perhaps will) tell the student if his last digit is off, i.e., that he is approximately right, but that is another subject, and we will defer that for the time being.

The Next Step:

We have several alternatives now, which we need to address. Here is a list of them:

1. How to get HTML interpreted correctly.
2. How to get the HTML to contain the answer desired (*vide infra*),
3. How to get random numbers into the question's HTML so that each student gets a different version.

4. How to include SureMath in the HTML so that a SureMath presentation can be included in each question for those who need help.

We will answer these an other questions in succeeding sections of this presentation.