JUIE v2.0

Java User Interface Editor

By: Ammar Muqaddas

Thanks to: Dr. Shakeel Lagari , Dr.Kassem Saleh, Jasim Al-Bazzaz

Note: This document was written way back in 1997-1998. I was a young undergraduate student, so please pardon my immature English. Also, some of the material and classes that I refer too here are obsolete.

♦ Introduction

There are many problems which programmers face when they program. As you know programming is not an easy job. It needs time, experience, resources, decision making and patience.

With the new ideas of Object Oriented Programming many of the problems of programming are solved.
One of the problems that Java programmers may face is how to design a **user interface** that fits their needs with a small amount of work and little time.
You may say: "*There is no problem in this. Just Add your components giving the x and y. and every thing will be OK"*.
Well do you know how much trials it takes to fit a button just in the place you want ?
Ask my friend Jasim about this. He suffered a lot.

Sun Microsystems tried to make things easier for the programmer. They added the LayoutManagers to Java. Layout managers take care of the positioning of the UI components. You just give small information and leave the rest on Java. For example: in a GridLayout you give the number of columns and rows.

But what if your user interface is not just a simple grid ?. What if it was really complex ? In this case you should add panels and have different Layouts for each panel. Which may be harder than the x , y style.
And this is true. I have seen some students (Working on H-ATM) confused of those layouts. Specially this GridBag Layout which until now I don't know how does it work precisely.

Why going far ?, Jasim was working on layouts. When he knew about the NULL layout (which is x,y positioning). He changed all his code to NULL Layout saying that it is much better.

As a result we thought that it is useful to do something that helps. The idea of the JUIE is the following:

Instead of writing the User Interface by lengthy complex code. Why don't we just edit the interface by mouse and leave the rest to a program that will convert this UI into a Java program.

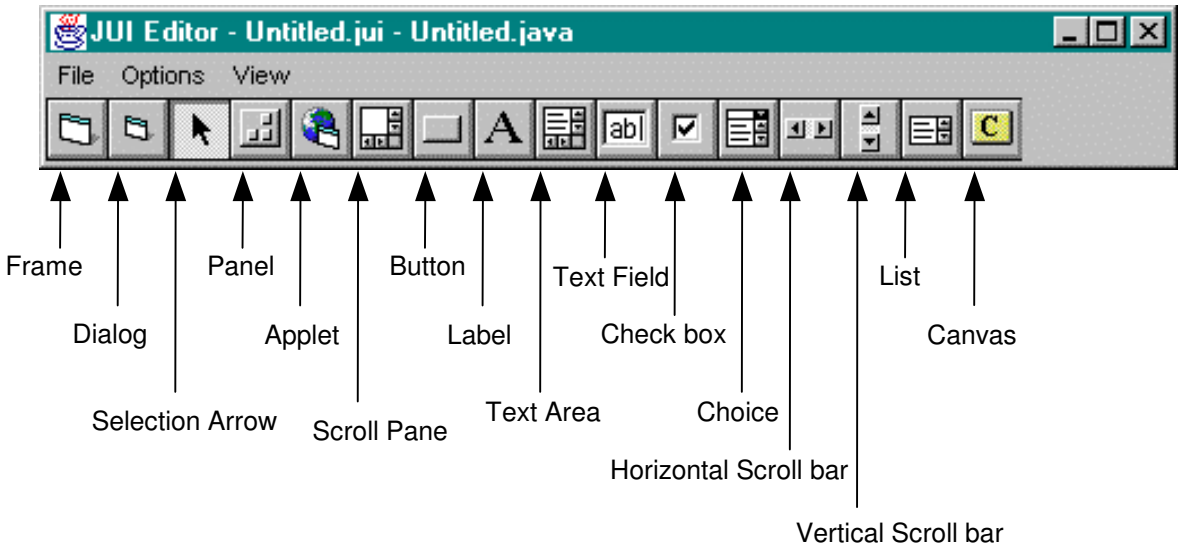
This is the job of the JUIE.

How to use JUIE

JUIE is made to be self explanatory. you don't need so much time to learn how to use it. As usual you use the Java interpreter like this:

```
C:\java Ed
```

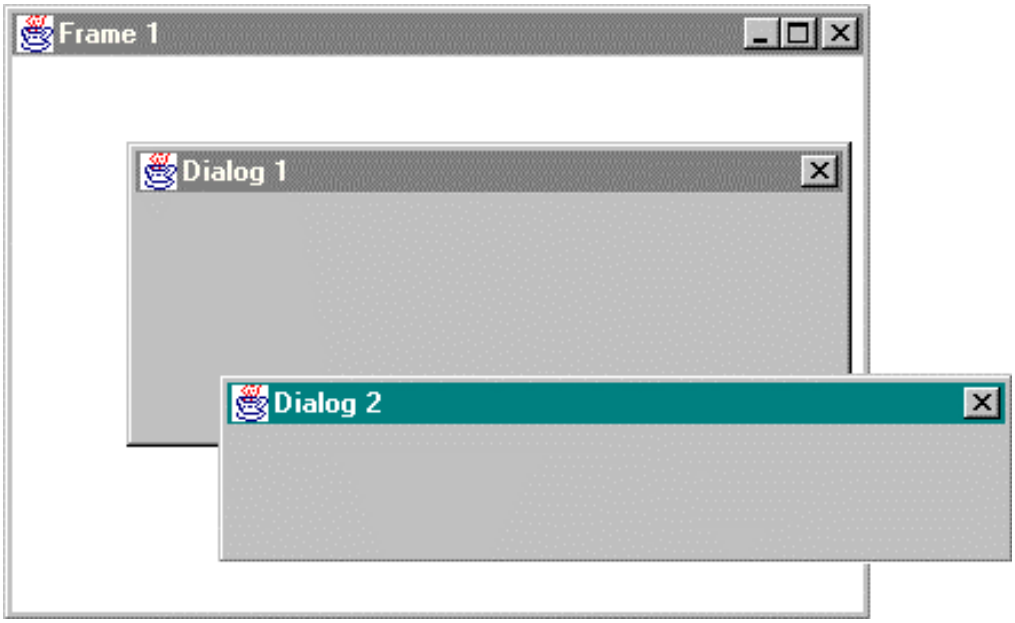
Were Ed is the main class name (Ed.class). The following window will appear :



The JUIE supports 15 components from Java's AWT. Each component has it's own button as indicated above. JUIE is not an MDI (Multiple-Document Interface). That is, it can only view ONE user interface at a time.

1. Adding Frames & Dialogs:

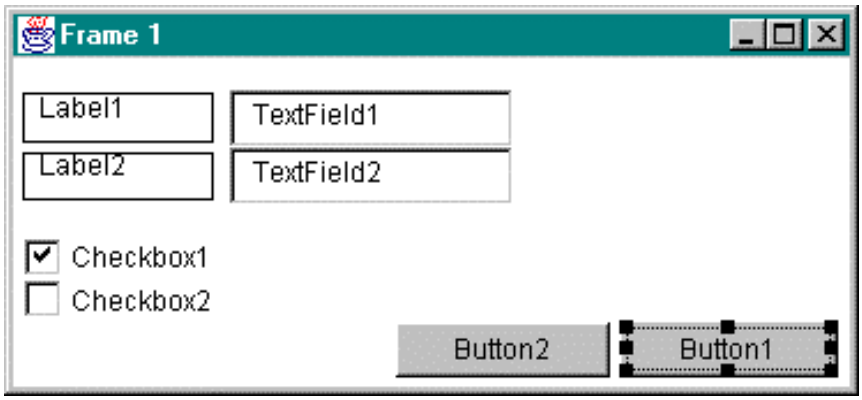
Click on the **Frame**  or **Dialog**  buttons to add them to your user Interface. They will appear to you in a fixed place with fixed size. You can then resize and reposition as usual.



2. Adding Other AWT Components:

Buttons after the **Selection arrow** all work the same way.
Click on the button corresponding to the component you want to add (**Button, ScrollBar,,,** etc.). And then go to a **window** and drag the component by mouse. It will appear directly.

This is an example:



3. Selecting, Positioning, Resizing, Deleting, Copying, Cuting and Pastnig components.

Selection:
You can select a component by just clicking on it. When selected a component will appear having six black squares on it's border.
BUT there is a problem. There are components which are containers (Like **Panel, Applet, ScrollPane**). When you click on these containers other components gets added instead of selecting the containers !

The solution is the **Selection Arrow**  . You just press it and then you can select any Component, whether it was a container or not.

Positioning:
There are two ways of positioning:

1. ***Mouse Dragging:*** You just Select the component and drag it by mouse. Unfortunately only **Panel, Applet, ScrollPane, Canvas** can be repositioned using the mouse. (That is because other components does not have a DRAG EVENT in Java).
2. ***Arrow keys:*** Select the component and use the arrow keys to reposition.

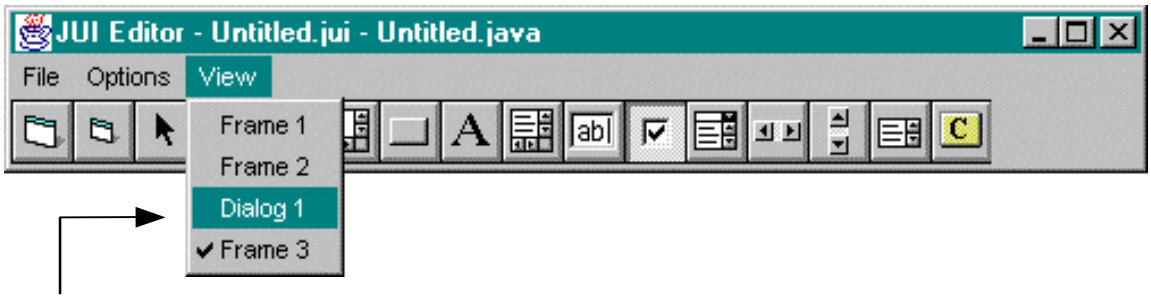
Resizing:
Select the component and while Pressing SHIFT use the arrow keys to resize.

Deleting :
Select the component and press the DELETE key. Or click on the **Options | Delete** menu. To Delete a window Click on that window and click on the **Options | Window Delete** menu.

Copying, Cuting, Pasting: ***(new)***
You are undoubtly familiar with these in windows. To Copy, Cut or Paste any component or any container that contains components use the **Options | Copy, Options | Cut** and **Options | Paste**.
If you copy or paste a container all it's sub components will be copied reserving the full hierarchy. Note that JUIE does not copy or paste into the windows global clipboard. It uses a special clipboard of it's own.

4. Hiding and Showing Windows:

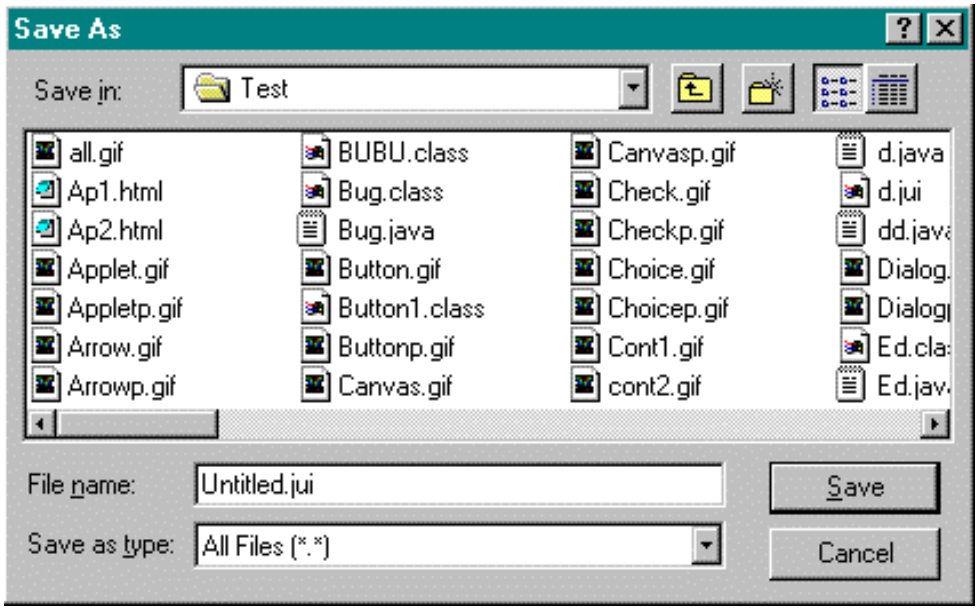
You can hide a window by clicking on it's **close** button.
You can show a window by clicking on it's name in the View menu as shown below:



You can show the Window from here

5. Saving and opening a user interface.

Saving: After you have edited a user interface. Most probably you want to save it as a file. You can do this by clicking on the **File | Save** menu or the **File | Save As** menu. If you choose **save As** A File dialog box will appear asking you for a file name as the following:

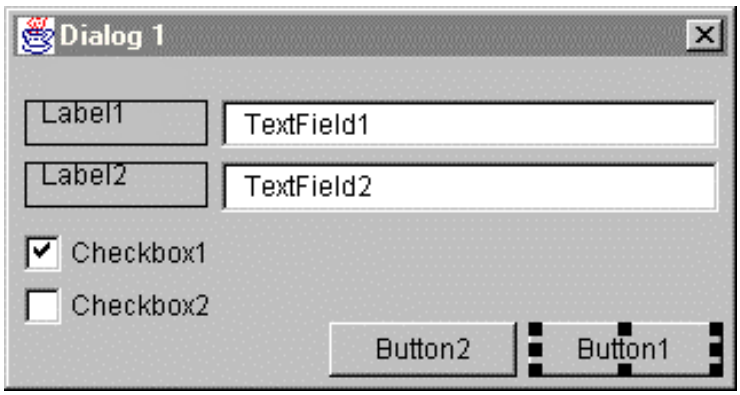


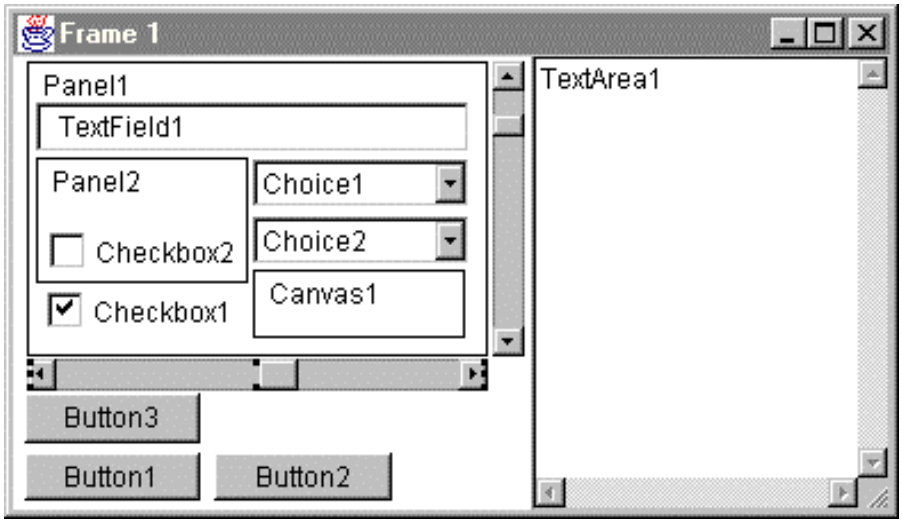
Opening: Use the **File | Open** menu to open a jui file. As above a file dialog box will appear asking you for the file name.

You can also start a new UI clicking on the **File | New** menu

6. Java Code Generation:

The Most important thing in JUIE is the ability to Generate a Java program that when run produces the edited UI. Now suppose that we've edited the following UI: (Frame and Dialog)





Now we want to Generate Java code for it.
Just go to the **File | Save Java Program** or **File | Save Java Program As** And choose a file name with an extension .JAVA. Then JUIE will apply complex methods and generate a text file Java program.

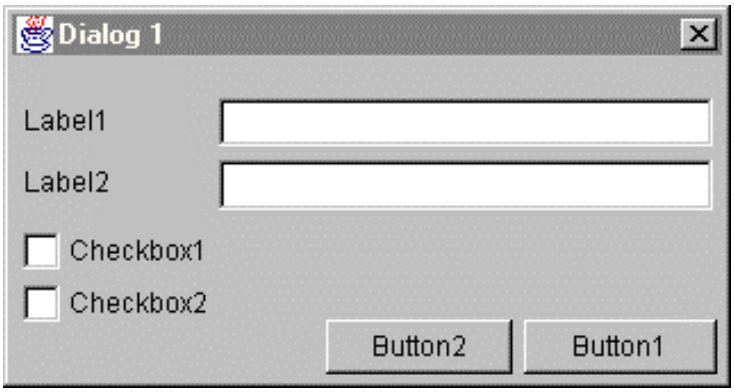
Here is the code Generated by the JUIE:

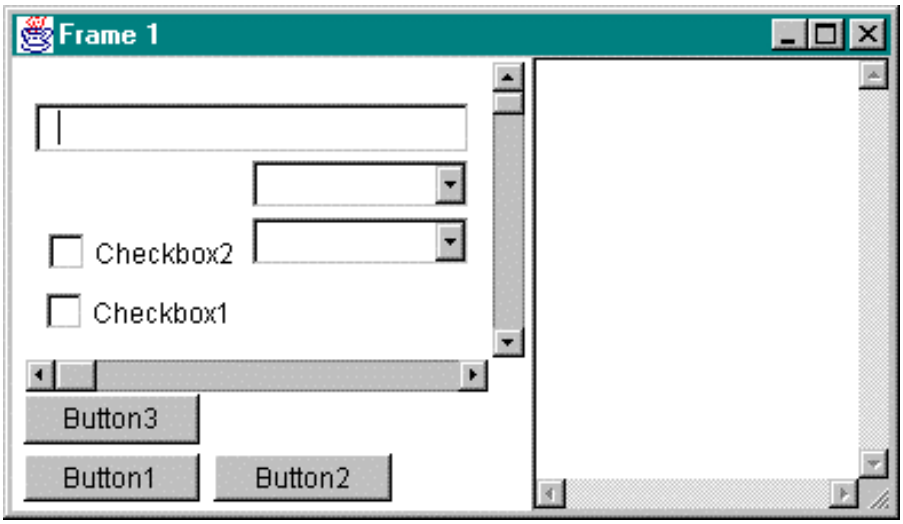
```
import java.awt.*;
import java.applet.Applet;

public class Testing
{ public static void main(String args[])
  { Component temp;

    Frame v1 = new Frame("Frame 1");
    v1.setLayout(null);
    v1.setBounds(81,83,392,225);

    Dialog v2 = new Dialog(v1,"Dialog 1");
    v2.setLayout(null);
    v2.setBounds(314,316,320,168);

    temp = new Button("Button1");
    v1.add(temp);
    temp.setBounds(9,196,77,21);

    temp = new Button("Button2");
    v1.add(temp);
    temp.setBounds(92,196,78,21);

    temp = new Button("Button3");
    v1.add(temp);
    temp.setBounds(9,170,77,22);

    temp = new TextArea();
    v1.add(temp);
    temp.setBounds(231,22,158,200);

    Panel v3 = new Panel();
    v1.add(v3);
    v3.setLayout(null);
    v3.setBounds(10,25,202,129);

    temp = new Scrollbar(1);
    v1.add(temp);
    temp.setBounds(214,25,14,129);

    temp = new Scrollbar(0);
    v1.add(temp);
    temp.setBounds(10,155,202,14);

    temp = new TextField();
    v3.add(temp);
    temp.setBounds(4,18,190,22);

    temp = new Choice();
    v3.add(temp);
    temp.setBounds(99,43,95,21);

    temp = new Choice();
    v3.add(temp);
    temp.setBounds(99,68,95,21);
```

```
    temp = new Canvas();
    v3.add(temp);
    temp.setBounds(99,91,93,30);

    temp = new Checkbox("Checkbox1");
    v3.add(temp);
    temp.setBounds(9,100,83,19);

    Panel v4 = new Panel();
    v3.add(v4);
    v4.setLayout(null);
    v4.setBounds(4,42,93,55);

    temp = new Checkbox("Checkbox2");
    v4.add(temp);
    temp.setBounds(6,31,84,21);

    temp = new Button("Button1");
    v2.add(temp);
    temp.setBounds(229,138,85,24);

    temp = new Button("Button2");
    v2.add(temp);
    temp.setBounds(142,138,82,24);

    temp = new TextField();
    v2.add(temp);
    temp.setBounds(95,41,217,21);

    temp = new TextField();
    v2.add(temp);
    temp.setBounds(95,68,217,22);

    temp = new Label("Label1");
    v2.add(temp);
    temp.setBounds(9,41,80,20);

    temp = new Label("Label2");
    v2.add(temp);
    temp.setBounds(9,68,80,20);

    temp = new Checkbox("Checkbox1");
    v2.add(temp);
    temp.setBounds(9,99,117,18);

    temp = new Checkbox("Checkbox2");
    v2.add(temp);
    temp.setBounds(9,121,115,20);

    // The following show()s are just for illustration. you can
remove them
    v1.show();
    v2.show();
  }
}
```

How much time it would take you to do this by hand ?

After compiling and running the code. The result is the following UI:





You may be wondering now : "*Where are the two panels and canvas which were in Frame1 ?*"
Well. They actually exist in the same place but they don't show because **Panels, Applets, and Canvases** do not have a visual look !
"Then how did it show when it was created by JUIE ?"
It showed because JUIE automatically draws a border for it so that the user can see it.

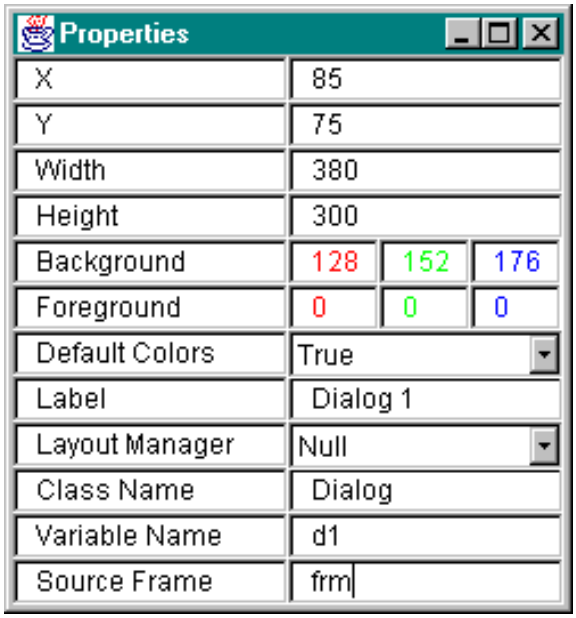
7. New features In JUIE 2

The following are the major new features we included in this release:

- Properties Window

In the previous release of JUIE you could add any components you want. You can resize them and reposition them. But not more than this. You can not for example, change the color, label, Text.
But now with JUIE 2 you can. The properties window will view all the properties of any component you select. You may change these properties as you wish. And those changes will be reflected directly in the code generated for your user interface.

The properties window looks like this:



The left column is the name of the property, and the right side is it's value.
Here is a list of all the properties with a brief description. Note that not all components have all properties, for example: **Source Frame** property is only for dialogs.

1. X, Y, Width and Height
These are obvious properties that determine the position and size of any component. Note that the X,Y of a component are relative to the nearest enclosing container. That is to the parent container which the return value of the JAVA built in function **getParent( ).**

2. Background and Foreground colors.
These determine the color of any component. In the properties window you'll see three colored numbers as values for Background, and foreground. These are a combination of Red, Blue and Green colors. When they mix they give you the color you want. Each one should be a value between 0 up to 255.
Note that these properties will only work if you set the Default Color property to False.

3. **Default Colors.**
   This is a true or false value that will Enable or Disable using the system color for the selected component. If you set it to true the code generator will ignore the colors you put for Background and Foreground and it will use the system color for the selected component. You may ask *"Why this, I can just leave the default color unchanged, and there is no need for this property."*
   This will work for your system only if you don't change your system color. If you change it the program generated will use the old color. Something else, How could you know the system color of somebody else. This will be impossible to handle without this property. Note that when you set this property to true, the background and foreground properties will be disabled. Also the system color may not show directly on the component. Don't worry about this. It is guaranteed to show in executing the generated code.

4. **Label**
   The label is the text shown on a component when it is displayed.

5. **Variable name**
   In the previous release the code generator automatically generates variable names for your component. You have no control of what variable names to be used in the program generated. But now with the **variable name** property you can specify a variable name for any component. If you don't specify a variable name for a component the program will automatically generate a name for it if it needs it to be named. (Components need to be named only if they were containers of other components).
   There are three restrictions for naming variables, First variable names should be unique. Second, you should not use a name of a format as **v#,** I mean don't use **v1, v2, v3, , , ,** because the generator uses those names. Third, You should not use the name **temp.** it is also used by the generator for temporary variables.

6. **Class name**
   In the previous release you are restricted to using predefined classes of Java. If you want to define your own classes you must do it your own by modifying the generated program. But you don't have to do that. If you want any component to be of your own defined classes just type the name of the class you want in this **class name** property. The code generator will automatically define this class for you, and the component will be of type this class. Then you can just modify the class definition if it doesn't fit your needs.

7. **Layout Manager**
   This property is available only for containers (Components that contain components). Java supports a number of layout managers to choose from. A layout manager manages the positions and sizes of all components in a container ignoring the x,y and sizes of the components. The values you set for x,y, width and height for components inside a container will only work if the layout manager property of that container is set to **Null**, which is the default value. The code generator will only define the container to have the layout manager you selected, It is your responsibility to configure the layout manager as you want.

8. **Source Frame**
   This is only available for Dialogs. For some reason in Java, A Dialog will not be created unless you give it a frame. That is all the constructors of a dialog need a reference to a frame in their declaration. For this reason, when you create a dialog you have to put the **Variable name** of any frame in this property, (Of course the frame has to be created). If you don't give it a frame the code generator will choose an arbitrary frame, or it will secretly create a new hidden frame and set **Source frame** property to it.

9. **Window On Click**
   The JUIE 2 is now able to make some interactivity with the user. If you want a component to respond to a mouse click and display a window for example, It can be done using this property. Go to the component you wish and set **Window On Click** to the **Variable name** of a Frame or Dialog (Window). Then generate the code and compile and run the program. Press the component, you will see the window you selected has appeared in front.
   Note: The effect will not be clear if the program generated shows for all windows from the beginning. Remove the **show** method from the window you want to see it apear clearly. Or hide the window by clicking close  . This will work only when you create a new class for the corresponding window.

New features summary:

1. Copy, Cut and Paste menus
2. Controlling the looking of a component, By its Background, Foreground, Label, Setting the Layout manager and using System default color.
3. Custom Variable Names mixed with automatic naming.
4. Class Auto Creation.
5. Showing Windows on click events.

Important notes:

JUIE is written for programmers. This means you should carefully follow programming rules, If you don't JUIE won't complain but the code generated may not compile. You are totally responsible of wrong variable names and classes. You should be familiar with programming in Java.

The following are examples of wrong things that you should avoid:

1. If you give two objects the same variable name, There may be a conflict. The code may not compile. Give each object (Component) a unique name.

2. If you have two components or more of different base class types, (For example a button and textbox). Don't give them the same class name because it is impossible for a class in Java to be of many kinds. Or have several base classes. You are allowed to give them the same class name only if they were the same type (That is from the same base class). Like three buttons, or 2 Frames.

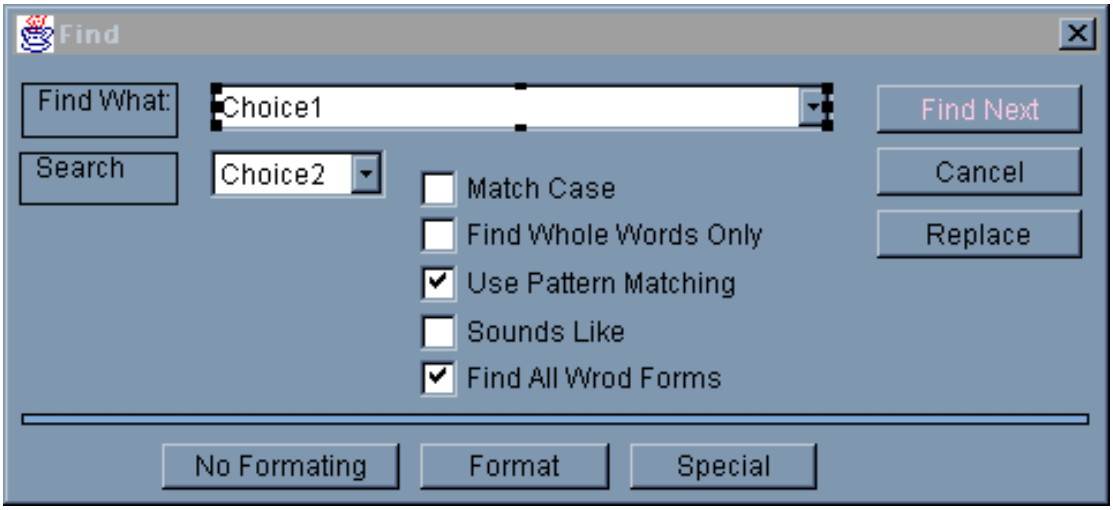
3. You will realize that when you run a generated code the close button does not work for Windows. This because java does not automatically define a Event procedure for the close Event, The programmer should define it. The JUIE can do it for you but you have to define a class (Non-standard class) for the window which you want it to work for.

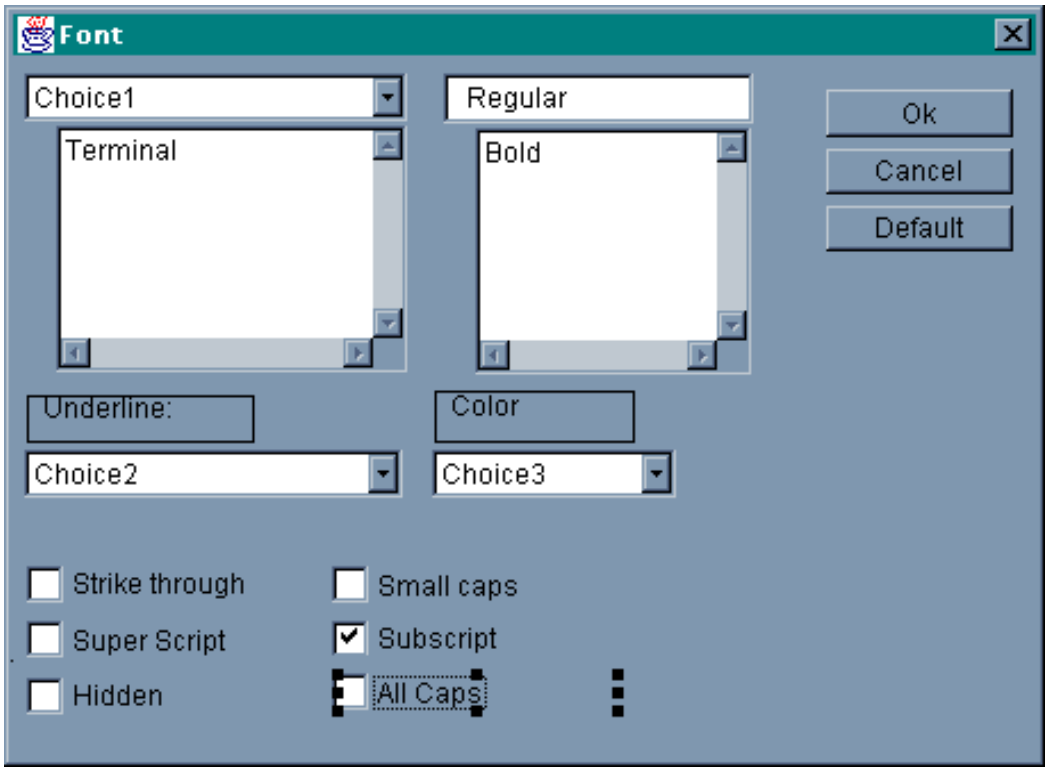
   Note: A non-standard class means a new class of your own that is not defined originally by java. The classes that appear by default in the **Class Name** property are all standard. You should change it to something else if you want the close to work for a frame.

   (See new class in the following)

An Example using the new features:

Here is a interface that is done by the JUIE. It saved to a file named Example1.jui (It's on the disk).





The interface is fore three dialog boxes. First the main window, second a search dialog, third a font choosing dialog.
The purpose of main window is to show the other windows by pressing the corresponding button. As you see there are two buttons in the main window, when **Choose font** is pressed the font dialog will be shown (If it was hidden). When the **find button** is pressed, the find dialog should be shown.
The properties of the interface are set just to do so. You can easily load Eaxmple1.jui and see the properties of each component, it better than listing them here.

- **Code Generation**

The program generated for this interface is:

```
import java.awt.*;
import java.applet.Applet;

public class Example1
{
  public static void main(String args[])
  { Component temp;

    Frame v1 = new Frame("Main");
    v1.setLayout(null);
    v1.setBounds(229,89,228,130);
    v1.setForeground(new Color(0,0,0));
    v1.setBackground(new Color(170,200,220));

    FDialog Find = new FDialog(v1,"Find");
    Find.setLayout(null);
    Find.setBounds(65,235,480,217);

    FDialog Font = new FDialog(v1,"Font");
    Font.setLayout(null);
    Font.setBounds(87,160,457,334);

    temp = new MButton("Find");
    v1.add(temp);
    temp.setBounds(12,75,203,37);
    temp.setForeground(new Color(255,255,0));
    temp.setBackground(new Color(128,152,176));
    ((MButton) temp).clickwindow = Find;

    temp = new MButton("Choose Font");
    v1.add(temp);
    temp.setBounds(12,35,203,38);
    temp.setForeground(new Color(255,255,0));
    temp.setBackground(new Color(128,152,176));
    ((MButton) temp).clickwindow = Font;

    temp = new Label("Find What:");
    Find.add(temp);
    temp.setBounds(8,34,68,24);

    temp = new Choice();
    Find.add(temp);
    temp.setBounds(90,34,270,21);
```

```
NewButton findnext = new NewButton("Find Next");
Find.add(findnext);
findnext.setBounds(379,35,89,21);
findnext.setForeground(new Color(255,200,230));
findnext.setBackground(new Color(128,152,176));

temp = new NewButton("Cancel");
Find.add(temp);
temp.setBounds(379,62,89,21);

temp = new Label("Search");
Find.add(temp);
temp.setBounds(7,64,69,23);

temp = new Choice();
Find.add(temp);
temp.setBounds(90,63,76,21);

temp = new Button("Replace");
Find.add(temp);
temp.setBounds(379,89,89,21);

temp = new Checkbox("Match Case");
Find.add(temp);
temp.setBounds(181,70,194,20);

temp = new Checkbox("Find Whole Words Only");
Find.add(temp);
temp.setBounds(181,90,193,20);

temp = new Checkbox("Use Pattern Matching");
Find.add(temp);
temp.setBounds(181,111,193,20);

temp = new Checkbox("Sounds Like");
Find.add(temp);
temp.setBounds(181,132,193,20);

temp = new Checkbox("Find All Word Forms");
Find.add(temp);
temp.setBounds(181,152,193,20);

temp = new Button("Format");
Find.add(temp);
temp.setBounds(181,190,82,20);

temp = new Button("Special");
Find.add(temp);
temp.setBounds(272,190,81,20);

temp = new Button("No Formating");
Find.add(temp);
temp.setBounds(69,190,103,20);

temp = new Canvas();
Find.add(temp);
temp.setBounds(8,177,463,5);
temp.setForeground(new Color(0,0,0));
```

```
temp.setBackground(new Color(128,172,220));

temp = new Button("Ok");
Font.add(temp);
temp.setBounds(361,37,82,21);

temp = new Button("Cancel");
Font.add(temp);
temp.setBounds(361,63,82,20);

temp = new Button("Default");
Font.add(temp);
temp.setBounds(361,88,82,20);

temp = new TextArea("Terminal");
Font.add(temp);
temp.setBounds(23,53,154,108);
temp.setForeground(new Color(0,0,0));
temp.setBackground(new Color(255,255,255));

temp = new Choice();
Font.add(temp);
temp.setBounds(9,30,168,21);

temp = new TextField("Regular");
Font.add(temp);
temp.setBounds(193,30,136,22);
temp.setForeground(new Color(0,0,0));
temp.setBackground(new Color(255,255,255));

temp = new TextArea("Bold");
Font.add(temp);
temp.setBounds(207,54,121,108);
temp.setForeground(new Color(0,0,0));
temp.setBackground(new Color(255,255,255));

temp = new Choice();
Font.add(temp);
temp.setBounds(9,195,166,21);

temp = new Label("Underline:");
Font.add(temp);
temp.setBounds(10,171,100,21);

temp = new Choice();
Font.add(temp);
temp.setBounds(188,195,107,21);

temp = new Label("Color");
Font.add(temp);
temp.setBounds(189,169,88,23);

temp = new Label("Label3");
Font.add(temp);
temp.setBounds(2,287,2,1);

temp = new Checkbox("Strike through");
Font.add(temp);
temp.setBounds(10,243,112,22);
```

```
    temp = new Checkbox("Super Script");
    Font.add(temp);
    temp.setBounds(10,267,106,23);

    temp = new Checkbox("Hidden");
    Font.add(temp);
    temp.setBounds(10,291,84,24);

    temp = new Checkbox("Small caps");
    Font.add(temp);
    temp.setBounds(144,245,115,19);

    temp = new Checkbox("Subscript");
    Font.add(temp);
    temp.setBounds(144,270,121,15);

    temp = new Checkbox("All Caps");
    Font.add(temp);
    temp.setBounds(144,291,128,21);

    // The following show()s are just for illustration. you can
remove them
    v1.show();
    Find.show();
    Font.show();
  }
}

class FDialog extends Dialog
{ public FDialog(Frame fr,String title)
  {  super(fr,title);
  }

  public boolean handleEvent(Event evt)
  {
    if (evt.id==Event.WINDOW_DESTROY)
    { this.setVisible(false);
      return true;
    } else
    return false;
  }
}

class MButton extends Button
{ public Window clickwindow = null;

  public MButton(String label)
  {  super(label);
  }

  public boolean handleEvent(Event evt)
  {
    if ((evt.id==Event.MOUSE_DOWN || evt.id==1001) &&
evt.target==this && this.clickwindow!=null)
    { this.clickwindow.show();
      return true;
    } else
    return false;
```

```
  }
}

class NewButton extends Button
{ public NewButton(String label)
  {  super(label);
  }
}
```

If you compile this program and run it, the two dialogs will be shown from the beginning !
This is because the program automatically shows them. Just hide them using the close button. And press the buttons on the main window, you will see that the dialogs show again. As I explained earlier, the close will not work unless you put a new class (using the **class name** property) for the window you want the close to work with.

Object Oriented Design

- Initialization:

As you know JUIE is an application. So it needs the **main** method. The **main** is located in the Ed class. It first initializes some variables.

```
public static void main(String args[]) {
  // Initializing
  NFrame.maxx=0;NFrame.maxy=0;NFrame.minx=0;NFrame.miny=0;

NDialog.maxx=0;NDialog.maxy=0;NDialog.minx=0;NDialog.miny=0;
```

Then loads the 16 Images (used as buttons). Then Instantiates the **Ed** class and calls it **edit.**
edit is the Editor main window. So it has to be **static** because every module in the program is going to use it.

- Classes Used

Instead of confusing you by displaying all the methods and variables. I prefer to show the names of the classes we have in the JUIE with a brief description:

Ed *extends Frame:*
The main Editor Frame which takes events from the ToolButtons and drives the menus.

MsgBox *extends Dialog:*
A reusable class that is responsible of displaying Error messages, showing a maximum of 4 button and returning a value indicating the button pressed by the user. The value is stored in the **pressed** variable.

ToolButton *extends Canvas:*
Represents the Image buttons on the main **edit** Frame. With some modifications to it you can turn it into a truly reusable Imaged Button which Java does not support (for the 1.1.4). Note that it extends Canvas. This is because the **Button** class has a built in visual look. But **Canvas**es don't have. So canvases will look exactly the way you want them without the need to destroy the old looking. If you extend the button you may find it difficult to draw over the old (Non imaged) looking.

NFrame *extends Frame:*
This is the class that the JUIE instantiates from it the frames of your user interface. I mean when you press the **Frame**  button a new instance of **NFrame** will be instantiated. This class has a very big **handleEvent** method that receives all the events for the **NFrame** or any AWT Components inside it. No Events are handled by the Components them selves.
Actually we spent nearly ½ the time of the project to build this method !. Note that it violates good cohesiveness.

NDialog *extends Frame:*
This is nearly a copy of the **NFrame** class except that it works for **Dialog**s. Nearly all the operations done by **NFrame** is done by **NDialog.**
You may be asking *"Why don't you just make two classes using the same general methods. specially if the* ***handleEvent*** *is so large ? Why writing the same thing two times ?"*
We tried really hard to make them this way but it appeared to be very lengthy. (Ask us about this).

The following classes are very similar so we will describe them together:

NButton, NPanel, NApplet, NScrollBar, NLabel, NTextArea, NTextField, NCheckbox, NChoice, NScrollbar, NList, NCanvas:
These classes are the ones used to instantiate AWT components of your edited user interface. That is the JUIE create an instance of any one of them when you place it by mouse on an NFrame or an NDialog. All of them have paint method. Some of them have exceptional behavior like the **NCanvas**, it has an **islabel** attribute.

Labels don't respond to the MOUSE_DOWN Event in Java !. We have to give a solution for this because if labels didn't respond to mouse events we could not even *select* them !!
We solved this by *tricking the JUIE !*. Instead of putting a Label why don't we put a Canvas and label it. The user will not see the difference.
BUT when we do Java Code Generation the JUIE will convert those Canvases to true Labels.

You may say: "*Ok. But how does the JUIE knows that a Canvas is Label not a real Canvas ?"*

The answer is the **islabel** attribute in class **NCanvas**. It is a Boolean that tells the JUIE whether or not the Canvas is meant to be a Label.

JConv extends Object
At last. this is the class used to Generate Java code. It first generates the code putting it in the **static String jav.** Then after the generation is done it saves this **String** to a text file. Using the method **saveText** which is defined in this class.

This class ***was*** more power full than what the editor expects. It can not only generate the code for the user interface but also create new classes and have manual variable naming by the user. In the previous release the editor was not yet complete to use these futures. (BUT KNOW WE HAVE DONE THIS)

This class may be modified to be used by other editors also. Although this is not easy because the code in this class is really complex. It traverses the components in your user interface recursively. The most confusing thing when we developed it was keeping track of **auto named** variables.

This class has been modified to support the new features of JUIE 2. Such as responding to click events.

Properties extends Frame ***(New class)***
This is the class used to manage the properties window described earlier. It contains methods to update the properties window or update the component selected and also to bind a selected component with properties window.

- Reusable methods and Classes in JUIE

Reusable Methods:

1. **Method createViewMenu** in class Ed

```
public static void createViewMenu(Vector wins,Menu me)
{ //Deleting the items
  me.removeAll();

  for (int i=0; i<wins.size(); i++)
  {  if (wins.elementAt(i) instanceof Frame)
       me.add(new CheckboxMenuItem(((Frame) wins.elementAt(i)).getTitle(),false));
     else me.add(new CheckboxMenuItem(((Dialog) wins.elementAt(i)).getTitle(),false));
  }
}
```

This method is highly reusable. You give it a Vector **wins** of Dialogs and Frames and a menu **me**.
Then it will automatically add **MenuItem**s to the menu **me** for each Frame or Dialog.
The View menu of the JUIE is created using this method.

2. **Method sumParentsXY** in class NFrame

```
public Point sumParentsXY(Component com)
// Summing all the x,y of all the parent containers of the com Component (including x,y for the component)
{ Point a1=new Point(0,0);
  Point a2;
  Component comp=com;
  while (comp!=this)
  { a2=((Component) comp).getLocation(); a1.translate(a2.x,a2.y);
    comp=((Component) comp).getParent();
  }
  return a1;
}
```

You give this method a component **com.** The method gets the position of **com** with respect to *this* window.

In other words. If you give this method a component. It will take *the x of this component + the x of the parent component + the x of the Parent of the Parent +.. +.. +.. Stop when you reach* ***this*** window. And same thing is done for y.

You may wonder *" Is this method really important ?"*
Well. it is important for the JUIE in the handleEvent method in NFrame.

3. **Method doborder** is class NFrame

```
public static void doborder(Component comp)
// Makes visual 6 squares around any component
{ Graphics g= comp.getGraphics();
  g.setColor(Color.black);
  int sx=0; int sy=0; int siz=6; int min;
  min=Math.min(comp.getBounds().width,comp.getBounds().height);
  if (min<=23) { siz=Math.max((int) (min*0.25),3); }
  if ((comp instanceof List) || (comp instanceof TextField)) { sx=4; sy=4; }
  if ((comp instanceof ScrollPane) || (comp instanceof TextArea)) { sx=20; sy=20; }
  g.fillRect(0,0,siz,siz); g.fillRect(comp.getBounds().width-siz-sx,0,siz,siz);
  g.fillRect(0,comp.getBounds().height-siz-sy,siz,siz); g.fillRect(comp.getBounds().width-siz-
sx,comp.getBounds().height-siz-sy,siz,siz);

  g.fillRect((int) ((comp.getBounds().width-sx-siz)/2),0,siz,siz);
  g.fillRect((int) ((comp.getBounds().width-sx-siz)/2),comp.getBounds().height-siz-sy,siz,siz);
  g.fillRect(0,(int) ((comp.getBounds().height-sy-siz)/2),siz,siz);
  g.fillRect(comp.getBounds().width-siz-sx,(int) ((comp.getBounds().height-sy-siz)/2),siz,siz);
} //doborder
```

Give this method a component. And it draw 6 filled squares around the component.

4. **Method boxin** in class NFrame

```
public static void boxin(Component comp)
{ Graphics g= comp.getGraphics();
  g.setColor(Color.black);
  g.drawRect(0,0,comp.getBounds().width-1,comp.getBounds().height-1);
}
```

This method draws Rectangle around any component to give it.
It is used in some paint methods in the program. This way the box will be permanent.

5. **Method getClassName** in class JConv.

```
public static String getClassName(Component com)
{ String classname;
  try
  { classname=(String) (com.getClass().getDeclaredField("classname").get(com));
    if (classname.equals(""))
    {  classname=com.getClass().getSuperclass().getName();
       if (com.getClass().getName().equals("NCanvas"))
          if (((NCanvas) com).islabel) classname="Label";
    }
  } catch (Exception e) { classname=com.getClass().getSuperclass().getName();
                          if (com.getClass().getName().equals("NCanvas"))
                            if (((NCanvas) com).islabel) classname="Label";
                        };
  classname=("."+classname).substring(("."+classname).lastIndexOf(".")+1);
  return classname;
}
```

Not as you expect. This method does not return the name of the class of that component.
This method searches in the **com** component for a String attribute called **classname.**
If it finds it then the return value is the same as **classname**. If it doesn't find it the method will return the name of the super class of the class of **com.**

6. **Method saveText** in class JConv

```
public static void saveText(String filename, String text) throws Exception
{ try
  { FileOutputStream o= new FileOutputStream(filename);
    o.write(text.getBytes());
    o.flush();
    o.close();
  } catch (Exception e) { throw e; };
}
```

This method is useful. Suppose you want to save a String as a text file without going to the details of FileOutputStream. Give this method a String and a filename and it will do the job.

There are other Reusable methods that need some modifications to be used in your programs. Like **sourceName, getVariable, declare**.

Reusable classes:
The two truly reusable Classes are **MsgBox, ToolButton.** (See description above in the classes part).