A JPython Guide

Gerald M. Guglielmo*
(FNAL CD/ODS/OSP)

March 26, 1999
Version 0.3

Abstract

This is a short document on getting started with JPython and Swing (JFC) in JPython. The document contains at the moment how to install and use JPython on a RedHat 5.0 Linux system using the Swing 1.0.3 Beta version as well as some examples on how to use some of the Swing classes in JPython.

*gug@fnal.gov

Fermilab Standard Government Sponsorship Notice

This work was performed at Fermi National Accelerator Laboratory, which is operated by Universities Research Association, Inc., under contract DE-AC02-76CHO3000 with the U.S. Department of Energy.

TERMS AND CONDITIONS

This software has been made freely available with a request for collaborative input subject to standard Fermitools Terms and Conditions given in full below.

SUPPORT

Help with the installation or use of FermiPython is available by sending mail to FermiPython-support@fnal.fnal.gov

which is auto-forwarded to the group in charge of FermiPython support at Fermilab.

This software is furnished by Fermilab on an "as is" basis as a service to the wider community with the goal of stimulating collaborative use and further development.

Provision of the software implies no commitment of support by Fermilab.

The Fermilab Computing Division is open to discussing other levels of support for use of the software with responsible and committed users and collaborators.

Use of the software in experiments at Fermilab will be assured of a level of support as agreed to through the standard Memorandum Of Understanding process.

Users may send mail to the mail address given above, with the expectation that it will be read and noted.

TERMS AND CONDITIONS IN FULL

Fermitools Software Legal Information - October 16, 1997

Terms and Conditions

When a User distributes or otherwise obtains a software package included in the Fermilab Software Tools Program, the user agrees to abide by the Terms and Conditions of the Program below:

- Any redistribution of the software shall be accompanied by this README file.
- The User shall acknowledge the origin of the software as set forth below: "This work was performed at Fermi National Accelerator Laboratory, operated by Universities Research Association, Inc., under contract DE-AC02-76CH03000 with the U.S. Department of Energy."
- The user is asked to feed back problems, benefits, and/or suggestions about the software to the Fermilab Software Providers and/or Fermitools management.
- Any distribution of the software available from this server shall be at no charge except for nominal media and distribution costs. To obtain a license to commercialize any of the software programs available from this server, contact FNAL's Office of Research and Technology Applications, P.O. Box 500, MS-200, Batavia, IL 60510-0500.

Indemnification by User of Third Party Claims and Disclaimer of Liability

The User, his/her directors, officers, employees, and agents hereby release and waive all claims against Universities Research Association, Inc. (URA) operator of Fermi National Accelerator Laboratory, its trustees, overseers, directors, officers, employees, agents, subcontractors, successors and assigns, for any and all liability and damages arising from the reproduction, use or other disposition of the software. The User shall indemnify URA and the U.S. Government for all damages, costs or expenses, including attorney's fees, arising from the utilization of the software, including, but not limited to, the making, using, selling or exporting of products, processes or services derived from the Software. The User agrees to indemnify, hold harmless and defend URA, its trustees, overseers, directors, officers, employees, agents, subcontractors, successors and assigns, against any and all liability, damage, loss, cost, charge, claim, demand, fee or expense of every nature and kind which may at any time hereafter, be sustained by URA by reason of claims of third parties arising out of alleged acts or omissions of the User in the reproduction, use or other disposition of the Software.

The User agrees that URA, its trustees, overseers, directors, officers, employees, agents, subcontractors, successors and assigns shall not be liable under any claim, charge, or demand, whether in contract, tort, criminal law, or otherwise, for any and all loss, cost, charge, claim, demand, fee, expense,or damage of every nature and kind arising out of, connected with, resulting from or sustained as a result of the use of the software programs available from this server. In no event shall URA be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, fees or expenses of any nature or kind.

Disclaimer of Warranties

The software is provided on an "as is" basis only. URA makes no representations, express or implied. URA MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that the Software will not infringe any privately held patent, copyright, or trademark. The entire risk as to the results and the performance of this software is assumed by the User.

Disclaimer of Endorsement

Reference herein to any specific commercial products, process, or service by tradename, trademark, manufacturer or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the United States Government, U.S. Department of Energy or URA. The views and opinions of individuals expressed herein do not necessarily state or reflect those of the United States Government, U.S. Department of Energy or URA and shall not be used for advertising or product endorsement purposes.

Liabilities of the Government

This software is provided by URA, independent from its Prime Contract with the U.S. Department of Energy. URA is acting independently from the Government and in its own private capacity and is not acting on behalf of the U.S. Government, nor as its contractor nor its agent. Correspondingly, it is understood and agreed that the U.S. Government has no connection to this software and in no manner whatsoever shall be liable for nor assume any responsibility or obligation for any claim, cost, or damages arising out of or resulting from the use of the software available from this server.

Contents

1 Installation of JDK (RedHat 5.0 Linux)

mkdir /usr/products/jdk

chmod o+w /usr/products/jdk

tar xvzf jdk_1.1.6-v5-glibc-x86.tar.gz

2 Install JPython (RedHat 5.0 Linux)

mkdir /usr/products/jpython chmod o+w /usr/products/jpython

download JPython103.class via the http://www.python.org/jpython/ web page. Then build the program.

../jdk/jdk116_v5/bin/java JPython103

>>>> /usr/products/jpython/JPython-1.0.3

download pylib151a.jar via the http://www.python.org/jpython/ web page. Next build the python classes.

JPython-1.0.3/jpython -jar pylib151a.jar

3 Install python v1_5 (RedHat 5.0 Linux)

Note: you cannot remove older version of python from system because of dependencies from other products the system needs. Instead install a separate version (via tar not via rpm). From fnkits, you can ftp python_v1_5_Linux+2.tar.

tar xvf python_v1_5_Linux+2.tar

4 Using Tkinter with CPython v1_5

Set up the environment so that JPython can find version 1_5 of python properly.

PYTHONHOME=/usr/products/python/v1_5

export PYTHONHOME

TCL_LIBRARY=/usr/lib/tcl8.0; export TCL_LIBRARY

5 Configure JPython

modify registry JPython-1.0/registry:

python.path=.:/usr/products/python/v1_5/lib/python1.5/: \
/usr/products/python/v1_5/lib/python1.5/lib-tk/

modify jpython script JPython-1.0/jpython to give full path:

```
exec /usr/products/jdk/jdk116_v5/bin/jre \
-Dinstall.root=/usr/products/jpython/JPython-1.0 -classpath \
/usr/products/jpython/JPython-1.0/jpython.jar:$CLASSPATH:\
/usr/products/jdk/jdk116_v5/lib/classes.zip org.python.core.jpython
"$@"
```

6 Using JPython

Add the directory with the executables to your path (java or javac or appletviewer) :

```
PATH=$PATH:/usr/products/jdk/jdk116_v5/bin
```

Now one can run JPython or python version 1_5 by calling them with full path names,

```
/usr/products/jpython/JPython-1.0.3/jpython
$PYTHONHOME/bin/python
```

or aliases can be defined to make life easier. Here is an example script:

```
#!/bin/sh echo "Setting environment for Python and JPython on
fndapg (Linux)" PYTHONHOME=/usr/products/python/v1_5;
export PYTHONHOME
JPYTHONHOME=/usr/products/jpython/JPython-1.0.3;
export JPYTHONHOME
TCL_LIBRARY=/usr/lib/tcl8.0;
export TCL_LIBRARY
alias python15="$PYTHONHOME/bin/python"
alias jpython="$JPYTHONHOME/jpython"
PATH=$PATH:/usr/products/jdk/jdk116_v5/bin
CLASSPATH=/usr/products/swing/swing-1.0.3/swingall.jar
export CLASSPATH
```

7 Prepare JPython Applets Demo

```
cd /usr/products/jpython/JPython-1.0.3/demo/applet
../../jpythonc -deep -core -jar appletdemo.jar *.py
cp appletdemo.jar html
```

Note: to view applets with browser or appletviewer, the program must be enclosed within <applet> and </applet> tags.

8 JPython Default Examples

jpython /usr/products/jpython/JPython-1.0.3/demo/awt/Graph.py

jpython /usr/products/jpython/JPython-1.0.3/demo/awt/Colors.py

jpython /usr/products/jpython/JPython-1.0.3/demo/awt/simple.py

9 JPython

JPython is a scripting language that may be as useful as CPython has become for rapid prototype development and basic script level programming. The term CPython comes from the more familiar name of Python which is written in C. By analogy JPython is a version of Python written in Java. Where CPython is a logical scripting language to use coupled with C++, JPython now becomes the logical choice to use with Java.

The main difference between the two scripting languages is in how support for graphical user interfaces is incorporated. CPython uses TKinter which is a wrapper to provide an object oriented interface to the Tcl/Tk language. JPython, on the other hand, uses the graphical user interface support that is built into Java and works with the latest Swing toolset. In my experience so far with both TKinter and Swing, I have found Swing a lot easier to use. The documentation for using TKinter and Swing in JPython is very weak to non-existent. I suspect that documentation for Swing will soon become widespread and that the mapping between Swing in Java and Swing in JPython will be much more consistent than I found between Tcl/Tk and TKinter.

9.1 The Basics of the Script

JPython uses the same basic structure for creating scripts as CPython. For example, here is the import statements used in the imagelabel1.py example:

```
import sys
```

The reader who is familiar with CPython will recognize that the format for the include statements above is the same as in CPython. This is not a coincidence since JPython is essentially CPython re-written in Java. Another important feature in JPython is that it can import and use many of the modules written for CPython even though they are written in C and not Java. I suspect that eventually most of the modules for CPython will be re-written in Java, but in the interim this turns out to be a very useful feature and aids in compatibility between the two scripting languages. If the reader is unfamiliar with the formatting of CPython scripts, it is best to read documentation on the basics of that language since a complete discussion at that level is beyond the scope of this document. One comment I will make on the very fundamental structure of CPython is that code blocks are delineated by indentation and not be sets of braces (as in C and C++) or other sets of markers.

The basic structure of a Python script (CPython or JPython) has three sections. The first section is a set of import statements which can be considered a combination of include and library statements for languages like C (where definitions are in header files and included at compile time, and the object code for linking pointed to by library statements on the link line). The second section is where all of the class definitions are held and usually most of the main execution code. The final section is the basic initialization section which contains the statements to begin execution analogous for the most part to the "main()" section in a C program. For purposes of simplicity in short examples, the second section can be omitted and the code normally there placed in the third section.

9.1.1 The First Section of a Python Script (Imports)

The structure of the import statements section was shown above, but will be repeated here and explained briefly. Here is the import statements used in the imagelabel1.py example:

```
from pawt import swing
from java import awt
from pawt import colors
from string import *
import java
import sys
from javax.swing import event
```

Just as in CPython, modules can either have sections imported using "from sys import path" or have the whole module imported using "import sys". Using the former syntax removes the namespace from the path to the classes, so one could type "path". In the later case, one would have to type sys.path. Here is what this looks like in an interactive JPython session for the first case:

```
>>> from sys import path
>>> path
['.', '/usr/products/jpython/JPython-1.1a3/Lib']
```

And in the second case:

```
>>> import sys
>>> sys.path
['.', '/usr/products/jpython/JPython-1.1a3/Lib']
```

The second method reduces namespace pollution but requires more typing and perhaps a bit less friendly looking code. Therefore one must decide when it is better to use "from xxx import yyy" and when it is better to use "import xxx".

9.1.2 The Second Section of a Python Script(Body)

The second section is where the class definitions and global functions in the script are usually defined. Here is a very basic definition of a global function called exit that calls the system exit member function:

```
def exit(e):
  java.lang.System.exit(0)
```

The "def" syntax indicates that a function is being defined and the semi-colon indicates that the next line should be indented and is first line of code in the function. If we wanted to define a class here we could do that here too. Here is a capture of an interactive session in Jpython which shows a class being defined, instantiated and then used:

```
>>> class row:
...   def __init__(self):
...     self.x=1
...   def xprint(self):
...     print self.x
...
>>>
>>> a=row()
>>> a.xprint()
1
>>>
```

Once again I want to emphasize that if this code could equally have been typed in the CPython interpreter it would have run there too because the syntax is the same.

9.1.3 The Third Section of a Python Script(Mainloop)

The third section is where you normally put the code that tells the interpreter how to begin executing the code and is like the "main()" body in a C program. If the script you are developing will not be run by itself, but is only meant to provide a foundation for other scripts (like a library of classes), then this section can be omitted. In this section there would be an instantiation of some class and then perhaps a call to run one of it's member scripts. For example, let's consider that a class defining a house has been written called HouseClass. In this class there is a member function for adding rooms to the house. The member function for adding rooms takes one argument which specifies the number of rooms to add. Then the third section of code might look like this:

```
if __name__== '__main__':
  house=HouseClass()
  house.addRooms(1)
```

In this example a instance of HouseClass is instantiated and 1 room is added. The key to this section of the script is the line “if __name__== '__main__':“ which signals that this is the main body of the script (think again about “main()” in a C program).

9.2 Where to Find Basic Python Information

The best place to start looking is on the Python web page. There one can find some useful documents on the basics of Python scripting. There is also an reasonable book called *Programming Python* by Mark Lutz (published by O'Reilly & Associates, Inc.). I also wrote a short note on using Python with TKinter that describes briefly a bit of the very basics of Python. This note is called *Notes on Using Python/TKinter* and is archived as DØNote 3377. The note can also be obtained in postscript format from the author (see email address for the author in the footnotes).

10 Swing

The Swing classes will eventually be part of the JDK package, and may already be on your platform. On Linux when I first started writing this they had to be downloaded separately. There may also be a later version of these classes, this is a beta version from before the 1.2 release of JDK was officially announced, so the user should download the most up to date version possible (and thus more things will work). Anyway, here is what I did:

1. download swing 1.0.3 from http://java.sun.com/products/jfc/
2. mkdir swing
3. copy tar file to swing
4. tar -xvf swing103.tar
5. JAVA_HOME=/usr/products/jdk/jdk116_v5
6. export JAVA_HOME

Note: you need to "unset" CLASSPATH to get Swing demos to work using java and you need to have it set have jpython swing demos working. Note CLASSPATH may not be necessary when you are reading this (hopefully that syntax will no longer be needed), but for the moment I will mention it here.

11 Swing in JPython

The Swing classes can be imported into JPython scripts and allow for some fairly fancy graphical structure to be developed without a lot of coding. The complete demo application is listed in section 12.

11.1 JFrame

The JFrame class is a container class that provides a top-level window including a title and a border. It is generally with a frame and within containers placed within the frame that the building of the visible application is done. Here is an example of a creating a JFrame object where the window title is set to "Image Label Example":

```
mainFrame=swing.JFrame('Image Label Example',visible=1,
  windowClosing=lambda event: sys.exit(0))
```

In the above code, the "windowClosing" specifies what function to call when the window is closed. In the example above the function is the "exit(0)" from the "sys" library. Alternatively, one could specify there own function which contains the call to "sys.exit(0)" if other actions are desired before the system exit.

11.2 JPanel

Inside of a JFrame object on can build up the structure of widgets desired for the appearance of the application. Inside a JFrame object a JPanel object can be placed with a layout specified. The simplest layout, and the one recommended, is to user a border layout. The border layout allows widgets to be placed within the panel at the North, South, East, West and Center positions. If more structure is need, which is often the case, JPanel objects can be placed within JPanel objects to build an increasingly more elaborate structure. Here is an example of a call to create a JPanel object with a border layout:

```
panel1 = swing.JPanel(layout=awt.BorderLayout())
```

That is the simplest way I know of for creating the JPanel with a border layout. If you would like a fancier look to the panel, there are configuration options available to accomplish this. For example, one could change the color of the background and add a border with a label around the panel. This is one way to do it with the panel created above:

```
panel1.setBackground(awt.Color(0,0,100))
greenLine = swing.BorderFactory.createLineBorder(awt.Color(0,200,0))
panel1BorderTitle = swing.BorderFactory.createTitledBorder(greenLine,
  'JLabel')
panel1.setBorder(panel1BorderTitle)
panel1BorderTitle.setTitleColor(awt.Color(200,0,0))
```

This may seem like it is complicated, but it really isn't. Let's look at the code line by line. The first line requests that the background color be set to blue (awt.Color is a class that takes red, green, weights). The background color has now been changed. The second, third and fourth lines could have been combined into one, it is just a little easier to read as three lines. The second line says create a Color object that represents a shade of green. The third line creates a border with a title and color , "JLabel" and a shade of green respectively. The fourth line adds sets the border for the panel to the border we created. Finally, the color of the border title is changed to a shade of red. Notice that the "p.setBorder(pBorderTitle)" call came before the call to set the title color in pBorderTitle. This indicates that the border is being set to the actual border object we created and not a copy of that object. This is a bit of a suprise to me at first, but makes sense when considering how panels and frames seem to work.

11.3 Image Labels with JLabel

The JLabel widget allows images or test to be displayed, and can be used as a simple way of displaying a logo in the window frame.

```
panel1Image1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
panel1Label2 = swing.JLabel(panel1Image1)
```

In the two lines of code above, an ImageIcon instance is created and then used in the creation of a JLabel instance. The first line shows how an ImageIcon instance can be created using the gif file fermilogo.gif. The second line shows how the a JLabel object can be created using the ImageIcon object. Instead of using an ImageIcon, the JLabel could also have been created using a string of ASCII text.

11.4 Adding Widgets to a JPanel

Adding a widget to a JPanel object is easy, especially if the border layout was specified. To accomplish this task you use the "add" member function of the JPanel class. With a border layout you should pass in the widget object you wish to add and where it should be placed (North, South, East, West or Center). For example, with a JPanel object "p" and a JLabel object "label1" the label can be added to the center of the panel with:

```
panel1.add(panel1Label2, 'Center')
```

Other objects can then be added to the panel, including JPanel objects, at the other positions if desired. Note that you should only add one object to any one position.

11.5 Adding a JPanel to a JFrame

Adding a panel to a JFrame object works differently that adding an object to a JPanel object. Instead of using the "add" member function, to add a panel to a JFrame object you use the "add" member function of the "contentPane" data member object of the JFrame. In other words, JFrame is a holder of the object that we really want to add the panel. In other words, we use the "add" member function of "contentPane" to add the JPanel object to it and "contentPane" happens to be a data member of JFrame. The code is simpler than the explanation:

```
mainFrame.contentPane.add(panel1)
```

11.6 Packing a JFrame Object

The packing of a JFrame object should be familiar to people who have use Tk or Tkinter. There is a member function of the JFrame class called "pack", and all one needs to do is call this member function. Assume that there is a JFrame object created call "mainFrame", then to pack the frame just write:

```
mainFrame.pack()
```

11.7 JScrollPane

Creating a scrollable pane is fairly easy once you know how to do it. There are several different ways to accomplish this task depending on what is needed. If the only requirement is to have a scrollable region displayed in the frame, then the task is easy. The trick is to create a JScrollPane object passing in a panel which will serve as the region to be scrolled. Once the JScrollPane object is created this way, then one needs to add the JScrollPane object to the contentPane of the JFrame.

```
sPaneN=swing.JScrollPane(panelN)
frameN=swing.JFrame('ScrollPane Example',visible=1)
frameN.contentPane.add(sPaneN)
```

By default, jpython will try to open the window large enough to fit the whole panel in and thus the scroll bars will not be seen unless the frame is resized. One can force the frame size to be smaller after the "pack" call is made to allow the scroll bars to be seen by default. Again assume a JFrame object called frame. Also assume that the panel would be larger than 100 by 100 (units are not important at the moment). Then the forced resizing can be accomplished by using the "setSize" member function:

```
frameN.setSize(apply(awt.Dimension, (100,100)))
```

The apply call above is converting the (100,100) into an object of type awt.Dimension required by the "setSize" member function.

If, on the other hand, the scrollable pane needs to be part of a larger display within the frame, the previously described method will not work. In this case, one needs to use the the viewport functionality.

```
panel9Image1=swing.ImageIcon('celticKnot.jpg', 'Celtic Knot')
panel9Label1=swing.JLabel(panel9Image1)
sPane3=swing.JScrollPane(panel9)
sPane3.getViewport().setView(panel9Label1)
panel9.add(sPane3, 'Center')
panel9Label2= swing.JLabel('Celtic Knot')
panel9.add(panel9Label2, 'South')
```

Once again a JScrollPane object is created. This time however, the the viewport of the JScrollPane object has its view set to a JLabel object. Next the JScrollPane object is added to a JPanel object which can be added to the main frame or another type of container (like JPanel for instance). In the above example, the JScrollPane object is added to a panel along with a JLabel object.

11.8 JTabbedPane

The JTabbedPane class is a nice way to develop pages which can be navigated by selecting a tab along one of the edges. This prevents the need to add specific buttons or menu items for navigating between pages. To accomplish this task one needs to create a JTabbedPane object, and that to a JPanel object and then add JPanel objects to the JTabbedPane. Each one of the JPanel objects added to the JTabbedPane will then have a tab associated with it for navigation between the panels. Here is a small piece of code for doing this:

```
tabPane=swing.JTabbedPane( )
mainPanel= swing.JPanel(layout=awt.BorderLayout())
mainPanel.add(tabPane, 'Center')
tabPane.addTab('JLabel', None, panel1)
```

The JTabbedPane object is created on the first line. The second line creates a JPanel object and then the JTabbedPane object is added to it on the third line. The fourth line adds "panel1" to the JTabbedPane object. The tab that will be created for the JPanel object will have the text 'JLabel'. One could have used a picture instead of words for the tab by creating an ImageIcon object and passing that in instead of the text. Assume that image4 is an ImageIcon object. Then instead of the "addTab" call structure above, one would use:

```
tabPane.addTab("", image4, panel1)
```

11.9 JButton

The JButton Class is useful for adding clickable buttons to a panel. The buttons can be configured to call a function to perform the desired action. You can also change the appearance of the button is you like, such as the colors for the foreground and the background. In the next piece of code a JButton object is created with the text "Random Bkg Color", both the background and foreground colors are set and the button is configured to call the function "randomColor":

```
panel2Button1 = swing.JButton('Random Bkg Color',
    actionPerformed=self.randomColor,
  foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
```

In the above code self.randomColor is a user defined member function that takes one argument (in addition to 'self') which is the instance of the event, in this case the button being clicked. If one wanted to call a global function instead of a member function, the actionPerformed statement would be:

```
actionPerformed=randomColor
```

The definitions of the randomColor functions in these two cases would be:

```
def randomColor(self,e):
```

for the member function case, and

```
def randomColor(e):
```

for the global function case. The difference between these two definitions is the parameter 'self', which Python automatically passes in for member functions.

11.10 Callbacks

In CPython I believe that callbacks are only allowed to take one argument (apart from 'self' for member functions), which turns out to be the instance of the event which causes the callback to be invoked. For the case of a button that would be an instance of the event of the button being pressed. Therefore one needs to find another way to get the needed parameters into the function when it is used as a callback. One option is to use global objects. In CPython there was also a mechanism called a lambda function which allowed one to get around this obstacle (see also section on JOptionPane). What the lambda function allows you to do is require additional arguments which have default values specified for them. For example:

```
frameN=swing.JFrame('Second Window Example',
  windowClosing=lambda event,a=self.mainFrame:
  swing.JOptionPane.showMessageDialog(a,
```

```
"Warning: A secondary window has been deleted.", "Warning",
swing.JOptionPane.WARNING_MESSAGE),
visible=1)
```

In this example, an additional parameter "a" is defined with a default value of "self.mainFrame". Then this additional parameter is specified as an input parameter inside the function call. Thus the one argument limit has been side-stepped. Additional arguments can also be specified with their default values in the comma separated list preceding the ":" character.

Here is an example of what the definition of a function that will be used as a simple callback might look like for a member function:

```
def randomColor(self,e):
  cr=r.nextInt()
  red=(0xff0000&cr)>>16
  green=(0xff00&cr)>>8
  blue=(0xff&cr)
  dotColor=awt.Color(red,green,blue)
  self.panel2.setBackground(dotColor)
```

The key points are that the function looks the same as a regular function definition does, except that it takes one and only one argument whereas normal functions do not have this requirement imposed.

11.11 Spawning a New Window

It is often useful to be able to spawn another window to hold a different interface of picture. This also seems to be fairly straight-forward to to in JPython with the Swing widgets. One way of doing this would be to create a button that when pressed would create the new window. This has the same for as the JButton object we created earlier. Let's assume we have already defined the member function that will create the new window and it was called "newWindow". Then the JButton call could look like this:

```
winButton = swing.JButton('New Window',
  actionPerformed=self.newWindow,
 foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
```

Now all we need to do is discuss how to write "newWindow". First since it is a callback we should define it to take only one argument (the instance of the event), and since it is a member function we need to hold a place for 'self' which Python will automatically pass in. Next we create a new JFrame object, place some widgets in it and then pack it. Here is an example:

```
def newWindow(self,e):
# create a second window — do not exit program if window is
deleted
  frameN=swing.JFrame('Second Window Example',
    windowClosing=lambda event,a=self.mainFrame:
    swing.JOptionPane.showMessageDialog(a,
    "Warning: A secondary window has been deleted.", "Warning",
    swing.JOptionPane.WARNING_MESSAGE),
    visible=1)
  panelN=swing.JPanel(layout=awt.BorderLayout())
  panelN.setBackground(awt.Color(0,200,0))
 blueLine = swing.BorderFactory.createLineBorder(awt.Color(0,0,200))
  panelNBorderTitle = \
swing.BorderFactory.createTitledBorder(blueLine,
    'Second Window')
  panelN.setBorder(panelNBorderTitle)
  panelNBorderTitle.setTitleColor(awt.Color(200,0,0))
  panelN = swing.JPanel(layout=awt.BorderLayout())
  panelNImage1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
  panelNLabel1 = swing.JLabel(panelNImage1)
  panelN.add(panelNLabel1)
  frameN.contentPane.add(panelN)
  frameN.pack()
```

For now Ignore the JOptionPane related stuff, it is for popping up a dialog box and will be described later. Apart for that, the syntax and calls look just like any other example for creating and filling a JFrame object.

11.12 JOptionPane

The JOptionPane can be used to pop up pre-defined dialog boxes for passing along simple information or requesting simple input from the user. The syntax in the following example looks a bit more complex than the usual JFrame object creation lines we looked at before. The complication is the "windowClosing" definition. Since callbacks can only take one argument (the instance of the event that caused the callback to be invoked) and we want to pass in some parameters we define (mainFrame in this example), the lambda function is used to get around the problem.

```
panel5Button3 = swing.JButton('ERROR Pane',
  actionPerformed=lambda event, a=mainFrame:
  swing.JOptionPane.showMessageDialog(a,
  "Error: This is a Error Option Pane", "Error",
  swing.JOptionPane.ERROR_MESSAGE),
 foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
```

For the purposes of this section it is better to just examine the actual JOptionPane object creation.

```
swing.JOptionPane.showMessageDialog(a,
  "Error: This is an Error Option Pane", "Error",
  swing.JOptionPane.ERROR_MESSAGE)
```

Here a JFrame object is passed for the dialog box to be associated with (a is set through the lambda function mechanism to be 'self.mainFrame'). Next is the message to be displayed followed by a title for the window ("Error: This is an Error Option Pane." and "Error" respectively). Finally, the type of message box is given which is a ERROR_MESSAGE type in this example.

11.13 JList

The JList widget can be combined with a scrollable pane to provide an scrolled list. This widget can be used to provide a list of options that a user can select from. By default the JList widget only allows a single item to be selected at one time, but this is configurable (setSelectionMode member function).

```
months=['January','February','March','April','May','June',
  'July','August','September','October','November','December']
panel6List1 = swing.JList(months)
sPane2=swing.JScrollPane(panel6)
sPane2.getViewport().setView(panel6List1)
panel6.add(sPane2, 'Center')
listButton= swing.JButton('Print Selected Item',
  actionPerformed=self.listSelect)
panel6.add(listButton, 'South')
```

Assume that a JPanel object called panel6 has been created. First "months" is created and is a list of items that will be displayed by the JList object. The creation of the JList object "panel6List1" uses "months" as a parameter. Next a JScrollPane object is created and the view is set to "panel6List1". Following the adding of the JScrollPane to the panel, a button is created that will call a function to print out the selected item when pressed. The callback for printing the selected item gets the value for the selected list item and passes this value to the print command

```
def listSelect(self,e):
  print self.panel6List1.getSelectedValue()
```

11.14 JCheckBox

The JCheckBox provides a useful means of selecting from a number of options where the number selected can range from none to all and selected items may not be adjacent items. One useful feature of the JCheckBox widget is that actions can be performed when the box is checked to select or de-select. In the code below assume that a JPanel object called "panel7" exists.

```
panel7CheckBox1 = swing.JCheckBox('I am Selected',
actionPerformed=self.checkAction)
panel7CheckBox1.setSelected(1)
panel7.add(panel7CheckBox1, 'South')
```

First a JCheckBox object is created with the default label of "I am Selected". Next the JCheckBox object is set to a value of selected by calling the member function "setSelected" with a value of 1 (a value of 0 should unselect). Finally, the JCheckBox object is added to the panel.

In this example a callback function is defined that will toggle the text of the label for the JCheckBox between "I am Selected" and "I am Unselected". This example shows how a JCheckBox object can be used to initiate actions.

```
def checkAction(self,e):
  if(panel7CheckBox1.isSelected()):
    panel7CheckBox1.setText('I am Selected')
  else:
    panel7CheckBox1.setText('I am Unselected')
```

11.15 JRadioButton

Unlike JCheckBox objects JRadioButton widgets are designed so that the can be grouped into sets of mutually exclusive options (any one option selected from the group forces all the others to be set to unselected). The additional step of having to group the JRadioButton objects allows for more than one set of options to be defined. In this example only one group of JRadioButton objects will be defined, but building multiple groups of JRadioButton objects should be easy starting with the template the following example provides.

```
panel8Radio1= swing.JRadioButton('Fermi',
  actionPerformed=self.setFermi)
```

```
panel8Radio2= swing.JRadioButton('Duke',
  actionPerformed=self.setDuke)
panel8Group1 = swing.ButtonGroup()
panel8Group1.add(panel8Radio1)
panel8Group1.add(panel8Radio2)
panel8Radio2.setSelected(1)
```

In the above code, the first few lines create the JRadioButton objects. Notice that each one has a different function passed in for "actionPerformed". This feature allows separate actions to be performed when a JRadioButton object is selected with a click of the mouse. It is important to note that these functions are not called when a JRadioButton is selected in the code with a "setSelected" member function call, but only when an X event initiates the selection. On the following lines a ButtonGroup instance is created, then the JRadioButton objects are added to the ButtonGroup object, and finally one of the JRadioButton objects is set to be selected.

The callback functions for the above example change the image displayed in the panel to reflect the JRadioButton object selected.

```
def setFermi(self,e):
  self.panel8Image1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
  self.panel8Label1.setIcon(self.panel8Image1)
  print "Fermi selected"
def setDuke(self,e):
 self.panel8Image1=swing.ImageIcon('dukeWave.gif', 'Duke Waves')
  self.panel8Label1.setIcon(self.panel8Image1)
  print "Duke selected"
```

11.16 JTextField

The JTextField class provides functionality for text field type objects and can be used as entry boxes.

```
panel10Text1 = swing.JTextField('North Message',
  actionPerformed=self.printText1)
panel10Text2 = swing.JTextField('South Message',
  actionPerformed=self.printText2)
panel10Label1= swing.JLabel('Your Text Here')
panel10.add(panel10Text1, 'North')
panel10.add(panel10Label1, 'Center')
panel10.add(panel10Text2, 'South')
```

The member functions used for the callbacks in this example do two simple tasks. First they each print out the text of the JTextField object associated with it. Second they change the text of the label object at the center of the panel to be the text of the JTextField object. To get the text the JTextField member function “getText” is used.

```
def printText1(self,e):
  print "The North Text is <", self.panel10Text1.getText(), ">"
  self.panel10Label1.setText(self.panel10Text1.getText())

def printText2(self,e):
  print "The text is <", self.panel10Text2.getText(), ">"
  self.panel10Label1.setText(self.panel10Text2.getText())
```

11.17 JMenuBar

This section talks about how to create a simple menubar add include it in a JFrame object. First one needs to build menus that can be added to the menubar and this can be done with the JMenu and JMenuItem classes. JMenuItem objects are used to build up options in a JMenu object. Each JMenuItem can have an action associated with it to be performed when the item is selected. In the example below, two menus are built and then added to a JMenuBar object.

```
exitMenu=swing.JMenu('Exit')
helpMenu=swing.JMenu('Help')
exitItem1=swing.JMenuItem('exit', actionPerformed=exit)
helpItem1=swing.JMenuItem('help', actionPerformed=version)
exitMenu.add(exitItem1)
helpMenu.add(helpItem1)
menuBar1=swing.JMenuBar()
menuBar1.add(exitMenu)
menuBar1.add(helpMenu)
```

The first two lines create the menus with the text “Exit” and “Help” displayed for the exitMenu and helpMenu JMenu instances respectively. Next the JMenuItem objects are created, each has a different callback to a global function. Then the JMenuItem objects are added to their respective menus with the JMenu member function “add”. Now the menubar is created as a JMenuBar object and the JMenu objects are added using the “add” member function of JMenuBar. All that remains to be done is to add this to a main panel. This is done with the following line:

```
mainPanel.add(menuBar1, 'North')
```

The two global functions used as callbacks and one global object in the example are:

```
myVersion='v00.02.00'
def exit(e):
  java.lang.System.exit(0)

def version(e):
  print myVersion
```

The global object "myVersion" is just a string representing the version of the script. The "exit" function calls for the program to be exited and the version function prints the version of the script.

12 A complete Example Script

This section contains the source code for a complete JPython script which has been used in part for the examples in this section. The script creates a JTabbed-Pane instance which holds panes for all of the different examples discussed in this document.

12.1 The Text of the Script "summary.py"

```
# summary.py
# JPython examples in a tabbed pane
# Created:  2 March 1999 Gerald M. Guglielmo
#
##### section 1
from pawt import swing
from java import awt
from pawt import colors
from string import *
import java
import sys
from javax.swing import event
from java.util import Random

###### section 2
myVersion='v00.02.00'
```

```
def exit(e):
  java.lang.System.exit(0)

def version(e):
  print myVersion

class tabbedExample:
#initialization
  def __init__(self):

# create main frame
    mainFrame=swing.JFrame('Image Label Example',visible=1,
      windowClosing=lambda event: sys.exit(0))

### create menus
    exitMenu=swing.JMenu('Exit')
    helpMenu=swing.JMenu('Help')

    exitItem1=swing.JMenuItem('exit', actionPerformed=exit)
    helpItem1=swing.JMenuItem('help', actionPerformed=version)
    exitMenu.add(exitItem1)
    helpMenu.add(helpItem1)

### create menubar
    menuBar1=swing.JMenuBar()
    menuBar1.add(exitMenu)
    menuBar1.add(helpMenu)

### create and configure JLabel Example

    panel1 = swing.JPanel(layout=awt.BorderLayout())
    panel1.setBackground(awt.Color(0,0,100))
    greenLine = \
swing.BorderFactory.createLineBorder(awt.Color(0,200,0))
```

```
    panel1BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
      'JLabel')
    panel1.setBorder(panel1BorderTitle)
    panel1BorderTitle.setTitleColor(awt.Color(200,0,0))
    panel1Image1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
    panel1Label1=swing.JLabel('Here is an Image Label')
    panel1Label2 = swing.JLabel(panel1Image1)
    panel1.add(panel1Label1, 'North')
    panel1.add(panel1Label2, 'Center')

### create and configure JButton Example
    panel2 = swing.JPanel(layout=awt.BorderLayout())
    panel2.setBackground(awt.Color(50,0,50))
    panel2BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
      'JButton')
    panel2.setBorder(panel2BorderTitle)
    panel2BorderTitle.setTitleColor(awt.Color(200,0,0))
    panel2Button1 = swing.JButton('Random Bkg Color',
      actionPerformed=self.randomColor,
    foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
    panel2.add(panel2Button1, 'Center')

### create and configure the Second Window Example
    panel3 = swing.JPanel(layout=awt.BorderLayout())
    panel3.setBackground(awt.Color(50,0,50))
    panel3BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
      'SecondWindow')
    panel3.setBorder(panel3BorderTitle)
    panel3BorderTitle.setTitleColor(awt.Color(200,0,0))
    panel3Button1 = swing.JButton('New Window',
      actionPerformed=self.newWindow,
    foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
```

```
    panel3Label1=swing.JLabel('Create a Second Window')
    panel3.add(panel3Label1, 'North')
    panel3.add(panel3Button1, 'Center')

### create and configure the JScrollPane Example
# do this by launching a new window:
    panel4 = swing.JPanel(layout=awt.BorderLayout())
    panel4.setBackground(awt.Color(50,0,50))
    panel4BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
       'JScrollPane')
    panel4.setBorder(panel4BorderTitle)
    panel4BorderTitle.setTitleColor(awt.Color(200,0,0))
    panel4Button1 = swing.JButton('Scroll Window',
      actionPerformed=self.newScrollWindow,
    foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
    panel4Label1=swing.JLabel('Launch for Example')
    panel4.add(panel4Label1, 'North')
    panel4.add(panel4Button1, 'Center')


### create and configure JOptionPane Example
    panel5 = swing.JPanel(layout=awt.BorderLayout())
    panel5.setBackground(awt.Color(200,200,200))
    panel5BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
      'JOptionPane')
    panel5.setBorder(panel2BorderTitle)
    panel5BorderTitle.setTitleColor(awt.Color(50,0,0))
    panel5Button1 = swing.JButton('WarningPane',
      actionPerformed=lambda event, a=mainFrame:
      swing.JOptionPane.showMessageDialog(a,
      "Warning: This is a Warning Option Pane", "Warning",
      swing.JOptionPane.WARNING_MESSAGE),
    foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
```

```
    panel5Button2 = swing.JButton('Info Pane',
      actionPerformed=lambda event, a=mainFrame:
      swing.JOptionPane.showMessageDialog(a,
      "Info: This is a Information Option Pane", "Information",
      swing.JOptionPane.INFORMATION_MESSAGE),
    foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
    panel5Button3 = swing.JButton('ERROR Pane',
      actionPerformed=lambda event, a=mainFrame:
      swing.JOptionPane.showMessageDialog(a,
      "Error: This is an Error Option Pane", "Error",
      swing.JOptionPane.ERROR_MESSAGE),
    foreground=awt.Color(0,120,0), background=awt.Color(200,255,200))
    panel5.add(panel5Button1, 'North')
    panel5.add(panel5Button2, 'Center')
    panel5.add(panel5Button3, 'South')


### create and configure JList Example
    panel6 = swing.JPanel(layout=awt.BorderLayout())
   greenline = swing.BorderFactory.createLineBorder(awt.Color(0,200,0))
   panel6BorderTitle = swing.BorderFactory.createTitledBorder(greenline,
       'List Example')
    panel6.setBorder(panel6BorderTitle)
    panel6BorderTitle.setTitleColor(awt.Color(100,0,100))
    months=['January','February','March','April','May','June',
      'July','August','September','October','November','December']

# need to use a viewport to get a scrollable list
    panel6List1 = swing.JList(months)
    sPane2=swing.JScrollPane(panel6)
    sPane2.getViewport().setView(panel6List1)
    panel6.add(sPane2, 'Center')

    listButton= swing.JButton('Print Selected Item',
       actionPerformed=self.listSelect)
```

```
    panel6.add(listButton, 'South')


### create and configure JCheckBox Example
    panel7 = swing.JPanel(layout=awt.BorderLayout())
    panel7.setBackground(awt.Color(50,0,50))
    panel7BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
       'JCheckBox')
    panel7.setBorder(panel7BorderTitle)
    panel7BorderTitle.setTitleColor(awt.Color(200,0,0))
    panel7CheckBox1 = swing.JCheckBox('I am Selected',
      actionPerformed=self.checkAction)
    panel7CheckBox1.setSelected(1)
    panel7.add(panel7CheckBox1, 'Center')


### create and configure JRadioButton Example
    panel8 = swing.JPanel(layout=awt.BorderLayout())
    panel8.setBackground(awt.Color(50,0,50))
    panel8BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
       'JCheckBox')
    panel8.setBorder(panel7BorderTitle)
    panel8BorderTitle.setTitleColor(awt.Color(200,0,0))
    panel8Radio1= swing.JRadioButton('Fermi',
      actionPerformed=self.setFermi)
    panel8Radio2= swing.JRadioButton('Duke',
      actionPerformed=self.setDuke)

    panel8Group1 = swing.ButtonGroup()
    panel8Group1.add(panel8Radio1)
    panel8Group1.add(panel8Radio2)
    panel8Radio1.setSelected(1)
```

```
    panel8Image1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
    panel8Label1=swing.JLabel(panel8Image1)

    panel8.add(panel8Radio1, 'North')
    panel8.add(panel8Label1, 'Center')
    panel8.add(panel8Radio2, 'South')


### create and configure JScrollPane and setView Example
    panel9 = swing.JPanel(layout=awt.BorderLayout())
   redLine = swing.BorderFactory.createLineBorder(awt.Color(200,0,0))
   panel9BorderTitle = swing.BorderFactory.createTitledBorder(redLine,
      'JScrollPane & setView')
    panel9.setBorder(panel9BorderTitle)
    panel9BorderTitle.setTitleColor(awt.Color(0,100,100))
    panel9Image1=swing.ImageIcon('celticKnot.jpg', 'Celtic Knot')
    panel9Label1=swing.JLabel(panel9Image1)

# need to use a viewport to get a scroll to work here
    sPane3=swing.JScrollPane(panel9)
    sPane3.getViewport().setView(panel9Label1)
    panel9.add(sPane3, 'Center')

    panel9Label2= swing.JLabel('Celtic Knot')
    panel9.add(panel9Label2, 'South')


### create and configure JTextField Example
    panel10 = swing.JPanel(layout=awt.BorderLayout())
    panel10.setBackground(awt.Color(200,200,200))
    panel10BorderTitle = \
swing.BorderFactory.createTitledBorder(greenLine,
      'JTextField')
    panel10.setBorder(panel7BorderTitle)
    panel10BorderTitle.setTitleColor(awt.Color(200,0,0))
```

```
    panel10Text1 = swing.JTextField('North Message',
      actionPerformed=self.printText1)
    panel10Text2 = swing.JTextField('South Message',
      actionPerformed=self.printText2)
    panel10Label1= swing.JLabel('Your Text Here')
    panel10.add(panel10Text1, 'North')
    panel10.add(panel10Label1, 'Center')
    panel10.add(panel10Text2, 'South')

### create and configure JTextArea and setView Example
    panel11 = swing.JPanel(layout=awt.BorderLayout())
  redLine = swing.BorderFactory.createLineBorder(awt.Color(200,0,0))
  panel11BorderTitle = swing.BorderFactory.createTitledBorder(redLine,
      'JTextArea')
    panel11.setBorder(panel11BorderTitle)
    panel11BorderTitle.setTitleColor(awt.Color(0,100,100))

    panel11TextArea1=swing.JTextArea()
    panel11Label1=swing.JLabel('Type in Text')

    panel11Text1 = swing.JTextField('Add to TextArea',
      actionPerformed=self.printText3)

# need to use a viewport to get a scroll to work here
    sPane4=swing.JScrollPane(panel11)
    sPane4.getViewport().setView(panel11TextArea1)
    panel11.add(sPane4, 'Center')
    panel11.add(panel11Label1, 'North')
    panel11.add(panel11Text1, 'South')


### create and add the tabbed pane, also notice here we add the
pane to the
### frame.
    tabPane=swing.JTabbedPane( )
```

```
    mainPanel = swing.JPanel(layout=awt.BorderLayout())
    mainPanel.add(tabPane, 'Center')
    tabPane.addTab('JLabel', None, panel1)
    tabPane.addTab('JButton', None, panel2)
    tabPane.addTab('New', None, panel3)
    tabPane.addTab('JScrollPane', None, panel4)
    tabPane.addTab('JOptionPane', None, panel5)
    tabPane.addTab('JList', None, panel6)
    tabPane.addTab('JCheckBox', None, panel7)
    tabPane.addTab('JRadioButton', None, panel8)
    tabPane.addTab('JScrollPane 2', None, panel9)
    tabPane.addTab('JTextField', None, panel10)
    tabPane.addTab('JTextArea', None, panel11)

    mainPanel.add(menuBar1, 'North')
    mainFrame.contentPane.add(mainPanel)
    mainFrame.pack()

    mainFrame.setSize(apply(awt.Dimension, (200,300)))

# make some objects data members for this class
    self.mainFrame=mainFrame
    self.mainPanel=mainPanel
    self.tabbedpane=tabPane
    self.panel1=panel1
    self.panel2=panel2
    self.panel3=panel3
    self.panel4=panel4
    self.panel5=panel5
    self.panel6=panel6
    self.panel7=panel7
    self.panel8=panel8
    self.panel9=panel9
    self.panel10=panel10
    self.panel11=panel11
```

```
    self.panel6List1=panel6List1
    self.panel7CheckBox1=panel7CheckBox1
    self.panel8Radio1=panel8Radio1
    self.panel8Radio2=panel8Radio2
    self.panel8Label1=panel8Label1
    self.panel8Image1=panel8Image1
    self.panel10Text1=panel10Text1
    self.panel10Text2=panel10Text2
    self.panel10Label1=panel10Label1

    self.panel11TextArea1=panel11TextArea1
    self.panel11Text1=panel11Text1

### other member functions
  def randomColor(self,e):
    cr=r.nextInt()
    red=(0xff0000&cr)>>16
    green=(0xff00&cr)>>8
    blue=(0xff&cr)
    dotColor=awt.Color(red,green,blue)
    self.panel2.setBackground(dotColor)

### create a second window —
### do not exit program if window is deleted
### shows the lambda trick of getting extra parameters in by mak-
ing a list
### with default values supplied.
  def newWindow(self,e):
    frameN=swing.JFrame('Second Window Example',
      windowClosing=lambda event,a=self.mainFrame:
      swing.JOptionPane.showMessageDialog(a,
     "Warning: A secondary window has been deleted.", "Warning",
      swing.JOptionPane.WARNING_MESSAGE),
      visible=1)
```

```
    panelN=swing.JPanel(layout=awt.BorderLayout())
    panelN.setBackground(awt.Color(0,200,0))
   blueLine = swing.BorderFactory.createLineBorder(awt.Color(0,0,200))
    panelNBorderTitle = \
swing.BorderFactory.createTitledBorder(blueLine,
     'Second Window')
    panelN.setBorder(panelNBorderTitle)
    panelNBorderTitle.setTitleColor(awt.Color(200,0,0))
    panelN = swing.JPanel(layout=awt.BorderLayout())
    panelNImage1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
    panelNLabel1 = swing.JLabel(panelNImage1)
    panelN.add(panelNLabel1)
    frameN.contentPane.add(panelN)
    frameN.pack()

### create a second window (scrollable) —
### do not exit program if window is deleted
  def newScrollWindow(self,e):
    panelN=swing.JPanel(layout=awt.BorderLayout())
    panelN.setBackground(awt.Color(0,200,0))
   blueLine = swing.BorderFactory.createLineBorder(awt.Color(0,0,200))
    panelNBorderTitle =
swing.BorderFactory.createTitledBorder(blueLine,
     'JScrollPane')
    panelN.setBorder(panelNBorderTitle)
    panelNBorderTitle.setTitleColor(awt.Color(200,0,0))
    panelNImage1=swing.ImageIcon('celticKnot.jpg', 'CelticKnot')
    panelNLabel1 = swing.JLabel(panelNImage1)
    panelN.add(panelNLabel1, 'Center')

# create and add the scrollable pane, also notice here we add the
pane to the
# frame.
    sPaneN=swing.JScrollPane(panelN)
    frameN=swing.JFrame('ScrollPane Example',visible=1)
```

```
    frameN.contentPane.add(sPaneN)
    frameN.pack()
    frameN.setSize(apply(awt.Dimension, (100,100)))

# list item print
  def listSelect(self,e):
    print self.panel6List1.getSelectedValue()

# toggle label on checkbox
  def checkAction(self,e):
    if(self.panel7CheckBox1.isSelected()):
      self.panel7CheckBox1.setText('I am Selected')
    else:
      self.panel7CheckBox1.setText('I am Unselected')

# radiobutton toggle image functions
  def setFermi(self,e):
    self.panel8Image1=swing.ImageIcon('fermilogo.gif', 'Fermi logo')
    self.panel8Label1.setIcon(self.panel8Image1)
    print "Fermi selected"

  def setDuke(self,e):
   self.panel8Image1=swing.ImageIcon('dukeWave.gif', 'Duke Waves')
    self.panel8Label1.setIcon(self.panel8Image1)
    print "Duke selected"

# textfield functions
  def printText1(self,e):
    print "The North Text is <", self.panel10Text1.getText(), ">"
    self.panel10Label1.setText(self.panel10Text1.getText())

  def printText2(self,e):
    print "The text is <", self.panel10Text2.getText(), ">"
    self.panel10Label1.setText(self.panel10Text2.getText())
```

```
# textarea functions
  def printText3(self,e):
    self.panel11TextArea1.append(self.panel11Text1.getText())
    self.panel11TextArea1.append('\n')


##### section 3
if __name__ == '__main__':
  r=Random()
  x=tabbedExample()
```

Index