**Lab 12: Some GUI Programming**

For this lab, you will do some work on a GUI program. Most of the work is related to actions, buttons, menus, and toolbars.  
  
When you turn in your work for this lab, you should paste the code for your **executable jar file** of your program, in addition to all the files from your Eclipse project.  See the last section on this page for information about how to create executable jar files.  
  
**To begin the lab, create a new project in Eclipse and copy-and-paste both directories – guidemo and resources -- from the *code* directory into your Eclipse project.** The guidemo directory is a package that contains the Java code of the program. The resources directory has several sub-directories that contain image and sound resources used by the program. You will be working only on the code in the *guidemo* package.  
  
The *code* directory also contains an executable jar file of the completed program, GuiDemoComplete.jar. You can run this jar file to see how the program should look when it's done. In the incomplete source code that you will be working on, the main class is in GuiDemo.java, and you can execute that file to see what the program does so far.

**Make a Menu**

The program has a toolbar at the bottom of the window. The buttons in this toolbar have icons. If you click one of the icons, you can then use the mouse to "stamp" copies of the icon onto the picture. (Note that a short sound is played when you stamp.) If you click the button "DEL", you can use the mouse to erase icon images from the picture.  
  
The toolbar is made by adding *Actions* to a *JToolBar* object. This is done in the class *IconSupport*, which contains a list of actions and a *createToolbar* method to create a toolbar from the actions.  
  
**Write a method *createMenu()* in the IconSupport class to create and return a menu with the same functionality as the toolbar.** That is, all the *Actions* from the *actions* arraylist should be added to the menu, just as they were added to the toolbar. (Remember that a menu is represented by an object of type *JMenu*, which can be created, for example, with: JMenu stamper = new JMenu("Stamper");)  
  
Once you have a method to create the menu, find the place in the constructor of the *GuiDemo* class where the menu bar is created. Use the method to create a menu, and add that menu to the menu bar. Test to make sure that it is working.

**Make a Toolbar**

There is room at the top of the window for another toolbar. **Make a *JToolBar* that duplicates the functionality of the "Background" menu, except for the *JCheckBox* that is at the bottom of the menu.** You can do all the work for this in the *GuiDemo* class. That class already has a *makeBackgroundMenu* method to create the "Background" menu. Add a *makeToolbar* method to create the toolbar. You can add *Actions* of type *ChooseBackgroundAction* to the toolbar, just as is done for the menu in *makeBackgroundMenu*. (You can create new action objects; you don't have to re-use the ones that are created in *makeBackgroundMenu*.)  
  
Find the place in the constructor of the *GuiDemo* class where the icon toolbar is added to the window. Add a background toolbar in the same way, but placing it in the *BorderLayout.NORTH* position instead of the *BorderLayout.SOUTH* position.  
  
You can also add the existing actions *newPictureAction* and *saveImageAction*to the toolbar, as I did in my program.

**Adding an Icon to an Action**

The "Custom..." and "Color..." actions don't look so good in the toolbar. Since they don't have icons, their text is shown instead. It would be nice to use icons instead; and, actually, it would be nice for them to have icons in the "Background" menu as well.  
  
**Use the resource "resources/action\_icons/fileopen.png" to make an icon for the "Custom..." action.** Add this icon to the action where it is created in the constructor of the ChooseBackgroundAction class (at the bottom of GuiDemo.java). Icons are already created there for the predefined image backgrounds, so you can see how it's done.

**ImageIcon from Scratch**

The "fileopen" icon seems OK for the "Custom..." action. But what about the "Color..." action? This action lets the user use a solid background color as the background, so perhaps an icon showing a plain patch of color would be OK. This is a chance to make an icon by creating a *BufferedImage* from scratch. **Create a BufferedImage, use it to make an ImageIcon, and use that icon for the "Color..." action.**  
  
To make the image, create a new *BufferedImage* with width 32, height 32, and type *BufferedImage.TYPE\_INT\_RGB.* Create a *Graphics* for drawing on the image, and use it to draw the image. For example, fill the entire image with dark gray, then fill a smaller square with light gray (or make some other drawing, such as the red/green/blue bars in my program). If you need an example of something like this, take a look at how *NoIconAction* is created in *IconSupport*.

**Add Radio Buttons in a Submenu**

Multi-line text can be *left-justified, right-justified*, or center-justified. The program has the code to do text justification, but there is no way for the user to select the kind of justification to be used.  
  
**Ad a submenu with a group of three buttons that can be used to control the text justification. You can do the work in the file *TextMenu.java*.**  
  
Create a new *JMenu* named "Justify." To this menu, you will add three *JRadioButtonMenuItems* named "Left", "Right", and "Center". To make the radio buttons into an actual group, create a *ButtonGroup* object, and add each radio button to the button group. Add the *JMenu* to the menu that is created by the constructor of the *TextMenu* class. You might want to use a *makeJustifyMenu* method, similar to (but simpler than) the makeFontNameSubmenu method that is already defined in *TextMenu*.  
  
Since the text justification has to change when the user selects one of the buttons, you will have to add an ActionListener to each of the buttons. The action listener for the "Left" button should do:  
  
panel.getTextItem().setJustify(TextItem.LEFT);  
panel.repaint();  
  
With similar code for the other action listeners, using *TextItem.RIGHT* and TextItem.CENTER for the values of the justify property.  
  
There is one subtle problem here. When the user selects the "New" command, a default text item is placed in the picture. This text item has "Left" justification, but the radio group might still indicate that the justification is "Right" or "Center," from its setting in the previous picture. There was a similar problem with the "Bold" and "Italic" checkboxes in the text menu. To solve it, a *setDefaults()* method was added to *TextMenu* which resets the checkboxes to their default state (unselected). **You need to do the same thing with the radio group.** Add a line to the *setDefaults* method in *TextMenu.java* to select the "Left" radio button. (To make this work, the "Left" *JRadioButtonMenuItem* will have to be an instance variable in the TextMenu class.)

**Use an Input Dialog**

Another text feature that the user might want to adjust is the *line-height* -- the distance from the baseline of one line of text to the baseline of the next line below it. A font comes with a default line-height, but the default is not always appropriate.  
  
The *TextItem* class has a *lineHeightMultiplier*property to address this problem. The font's default line-height is multiplied by the value of this property. By default, the property has value 1.0, but it can be set to any positive *double* number.  
  
**Add support for changing the value of the *lineHeightMultiplier* to *TextMenu.java*.** You can use an input dialog to get the new value from the user. The same thing is already done in *TextMenu* for the "Set Size..." command, which is created in the *TextMenu* constructor. You can imitate what is done there.

**Personalize Your Program**

The constructor in *GuiDemo.java* sets up the background and text that appear in the picture when the program starts. **Personalize the program by using your own saying and choice of background.** I want to be able to tell different people's programs apart! (And I want to see what kind of sayings you come up with.)

**Make a "jar" file**

A JAR (Java ARchive) file can hold Java classes and other resources. It is possible to designate one of the classes in a jar file as the "main class." The main class must contain a main program. Double-clicking on a jar file that has a main class should execute the *main()* routine in that main class. A jar file with a main class is called an "executable jar file". (If double-clicking does not work on Linux, try right-clicking and using "Open With...".)  
  
It's easy to create an executable JAR file in Eclipse. The hardest part is to remember to select a main class. Right-click on the project name and choose "Export..." from the pop-up menu. In the dialog box, choose "Jar file" under "Java" as the type of object that you want to export. Click "Next." On the next screen, set the name of the Jar file; use the "Browse" button to select the directory and file where the jar file is to be saved. Then, click "Next" **twice**. The last screen has an input box where you can specify the main class; click the "Browse" button next to this box, and select the main class from the list. Finally, click "Finish." You are likely to get a message such as "Jar export completed with warnings." Don't worry about the warnings -- this just refers to compiler warnings in your code, and they are not errors.  
  
(You can see section 6.6.4 of the textbook for more detailed information on jar files.)