CST8221 – Java Application Programming
Hybrid Activity #5
Dialogs - Talk to the User

Terminology

A ***dialog*** is an independent window meant to carry brief temporary notice apart from the main application window. In general dialogs present an error message or warning to a user, but dialogs can be used for simple input, and can present images and just about anything compatible with the main application that manages them.

For convenience, several Swing component classes can directly instantiate and display dialogs. To create simple, standard dialogs, you should use the ***JOptionPane*** class. The ***ProgressMonitor*** class can put up a dialog that shows the progress of an operation. Two other classes, ***JColorChooser*** and ***JFileChooser***, also supply standard dialogs. If you want to create dialog which is different from the standards dialogs offered by the four classes above you have to create a custom dialog, using the JDialog class. For example, all dialogs created by ***JOptionPane*** are **modal**. Modal means that the dialog window blocks the main application and the user must respond and close the dialog before they can work with the main application. If you want a modeless dialog you must create a custom dialog using directly ***JDialog*** (see the reference link for demo examples).

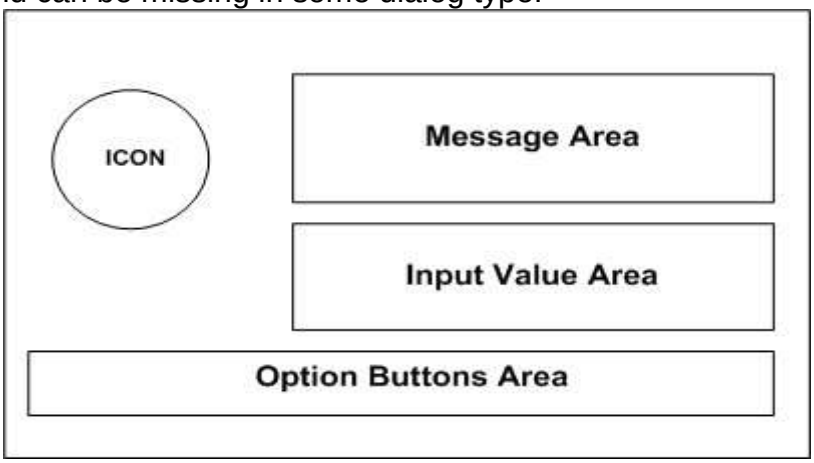
In this activity you will use dialogs created with ***JOptionPane*** class.

The Nature of Things

The ***JOptionPane*** class makes it very easy create and display a set of standards dialogs. The class contains many *static* methods but almost all practical uses of this class are one-line calls to four static methods: ***showConfirmDialog(), showMessageDialog(), showOptionDialog(),*** and ***showInputDialog()***. Those methods are overloaded, so that different flavors of dialogs can easily be created. JOptionPane provides similar methods for creating internal frames which are holding the dialog box.

Instead of using the static methods you can use the JOptionPane class constructors to instantiate a *JOptionPane* and ask it to put itself into a *JDialog* or *JInternalFrame*, which you then display. The basic trade-off is this: using the static methods is a bit easier, but using a constructor allows you to hold on to and reuse the *JOptionPane* instance, a useful feature if the pane is fairly complex, and you expect to be displaying it frequently (if you use the static methods, the option pane gets recreated each time you call). The significance of this difference depends largely on the complexity of the pane.

The basic appearance of the dialogs is similar to the picture shown below. Each of the elements is optional and can be missing in some dialog type.



The parameters the methods take are: *parentComponent, message, messageType, optionType, options, icon, title, initialValue*. The *parentComponent* defines the container that is to be the parent of this dialog box. It could be the *frame* of the main application, or it could be *null*. If it is *null*, a default frame is used, and the dialog will be centered on the screen. Do not overuse null because it creates a new frame every time the dialog pops up. For more details explaining the parameters please visit the ***JOptionPane*** API documentation.

All dialog boxes (except Message dialogs) return some information to the application. Here is a quick summary of what the returned values mean.

Input Dialogs: The versions that do not take an array of selection values return a String. This is the data entered by the user. The methods that do take an array of selection values return an Object reflecting the selected option. In any case, if the user presses the "Cancel" button, null is returned.

Confirm Dialogs: These methods return an *int* reflecting the button pressed by the user. The possible values are YES_OPTION, NO_OPTION, CANCEL_OPTION, and OK_OPTION. CLOSED_OPTION is returned if the user closes the window without selecting anything.

Message Dialogs: These methods have void return types.

Option Dialogs*:* If no options are specified, this method returns one of the constant values YES_OPTION, NO_OPTION, CANCEL_OPTION, and OK_OPTION. If options are explicitly defined, the return value gives the index into the array of options that matches the button selected by the user. CLOSED_OPTION is returned if the user closes the window without selecting anything. Getting a value from a JOptionPane you've instantiated directly is also very simple. The value is obtained by calling the pane's *getValue()* method. This method returns an Integer value using the same rules as those described above for Option Dialogs with two small variations. Instead of returning an Integer containing CLOSED_OPTION, *getValue()* returns null if the dialog is closed. Also, if you call getValue() before the user has made a selection (or before displaying the dialog at all, for that matter), it will return UNINITIALIZED_VALUE. To get the value of user input (from a *JTextField*, *JComboBox*, or *JList*), call *getInputValue()*.

References - Swing

Textbook 1 – Chapter 12.22;
Java Swing, second edition.
Links:
http://download.oracle.com/javase/tutorial/uiswing/components/dialog.html

When JavaFX API was released it did not provide a standard API similar to Swing API for creating dialogs. The application developer had to create their own dialogs – see the hybrid activity ***CustomDialogFX.java*** code example. There are also third party APIs providing convenient sets of dialog controls. For more information, please visit the following links:

http://code.makery.ch/blog/javafx-8-dialogs/
http://fxexperience.com/controlsfx/

With the release of Java 1.8.40 the problem with the standard dialogs has been solved. Four specialized classes have been introduced Alert, Dialog, TextInputDialog, and ChoiceDialog. Those classes provide the same flexibility as the corresponding Swing classes and much more – see the hybrid activity ***StandardDialogFX.java*** code example. You can find more example at this location:

http://code.makery.ch/blog/javafx-dialogs-official/

References - JavaFX
Textbook 2 – Chapter 16, 14;

Code Examples

The code examples demonstrate the workings of the static methods of ***JOptionPane*** and how to create dialogs in JavaFX.

You will find the examples in ***CST8221_HA05_code_examples.zip***.

Exercise
Download, compile, and run the code examples. When running the Swing example, you must have an open console to see the results from the interaction with the dialogs. Once you see how they work, explore very carefully the code. Open the *JOptionPane* API documentation and try to make some changes. Visit the reference link provided above. You will find a demo exploring all possible combinations of Dialog boxes.
Examine the JavaFX examples and visit the reference links to learn more about the standard JavaFX dialogs.

Note: If you use Eclipse IDE to compile and run the examples, make sure that you copy the image file (*smile.gif*) used in the example in the project folder and the ***bin*** folder of the project. If you use NetBeans IDE, copy the image file in the source files folder (***src***) of the project.
If you use Command Window or Terminal to compile and run manually, the image file must be in the same folder as your class files.

Questions

Q1. Are all dialogs created by *JOptionPane* modal?
Q2. Can you have more than one value returned by the Input Dialogs?
Q3. Do you need a parent frame in order to create a dialog?
Q4. Does JavaFX currently provide a standard dialog API?
Q5. Are the standard JavaFX dialogs modal?

Submission
No submission is required for this activity.

Marks
No marks are allocated for this activity, but remember that understanding how *Dialog* works is essential for building an user friendly GUI.

And do not forget that:

"Conversation without a dialog is like a program without input." Anonymous

but also never forget that*:*

"Never miss a good chance to shut up." Will Rogers