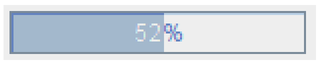
CST8221 – Java Application Programming
Hybrid Activity #6
Progress bars and Progress Monitors

Terminology

Nobody wants to be frozen in time waiting for something to happen. If you want that, go to a horror movie. Progress Bars and Progress Monitors are very important GUI components which are used to show the progress of a relatively slow activity. A ***JProgressBar*** is a Swing component that indicates progress of program activity. A progress A ***ProgresMonitor*** is a dialog box that contains a progress bar and some additional information. A ***ProgressMonitorInputStream*** displays automatically a progress monitor dialog while the input stream is read.

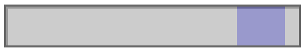
The Nature of Things

A progress bar is a very simple component. It is a horizontal or vertical rectangle filled with some color bar which indicates the progress of an activity or operation. By default, the progress is indicated growing color bar, but it can also display a string “N%” showing the progress in percents like in the figure below.



Different Look and Feels have different filling styles. Some use solid fill like the one above. Windows Look and Feel uses an LED-type style. The bar is filled with adjacent rectangles separated with some space.

Sometimes the length of a long-running task cannot be determined in advance, or the task might stay at the same state of completion for a long period of time. To show work without measurable progress you can switch the progress bar in *indeterminate mode.* A progress bar in indeterminate mode displays animation to indicate that work is occurring. That is the kind of progress you usually see in your browser (it is usually done with an animated gif file). In the default Java Look and Feel, the progress bar moves a colored rectangle back and forth:



As soon as the progress bar can display more meaningful information, it should be switched back into its default, determinate mode.

To create a progress bar is very simple task. You have to construct a progress bar by providing the minimum and maximum value and an optional orientation.

```
JProgressBar progressBar = new JProgressBar(0, 1000);
JProgressBar progressBar = new JProgressBar(StringConstants.VERTICAL,0, 1000);
```

You can also set the minimum and maximum with *setMinimum()* and *setMaximum()* methods. The progress bar does not react to user actions. Your program **must** call the *setValue()* method in order to update it. If you call *setStringPainted(true)*, the progress bar calculates the completion percentage and automatically displays a sting “N%”. If you want to show a different string you can change it with the *setString()* method:

```
if(progressBar.getValue() > 500) progressBar.setString(“Half done”);
```

To see how all this works run the ***ProgressBarDemo*** example included in the code examples.

To use a progress bar you have to place it in some kind of container. In contrast, a ***ProgresMonitor*** is a complete dialog box (see HA#5) that contains a progress bar. Usually it contains a Cancel button. If it is clicked, the monitor dialog box will be closed. You can check if the progress has been canceled by calling *isCanceled()* method of the ***ProgressMonitor*** object and terminate the monitored task.

To construct progress monitor you should supply the following parameters:

- The parent component over which the dialog box should pop-up. If null, the dialog box will pop-up in the center of the screen.
- An object which should be a string, icon, or some other component) that is displayed on the dialog box.
- A note to display below the object. It null, no note is displayed.
- The minimum and maximum values.

The progress monitor cannot measure progress or cancel an activity by itself. You still need to set the progress value by calling the *setProgress()* method (This is equivalent to the *setValue()* method of the ***JProgressBar*** class). When the monitored task has concluded, the dialog box will close. If the click the Cancel button the event handler of the button must call the *close()* method to dismiss the dialog box.

By default, a progress monitor waits a minimum of 500 milliseconds before deciding whether to pop up the dialog. It also waits for the progress to become more than the minimum value. If it calculates that the task will take more than 2000 milliseconds to complete, the progress dialog appears. To adjust the minimum waiting period, invoke *setMillisToDecidedToPopup()*. To adjust the minimum progress time required for a dialog to appear, invoke *setMillisToPopup()*.

The biggest problem with using progress monitor dialog box is that the handling of the cancelation request. You cannot attach an event handler to the Cancel button. Instead, you need in some other event handler to periodically call the *isCanceled()* method to see if the user has clicked the Cancel button.

To see how all this works run the ***ProgressMonitorDemo*** example included in the code examples.

You should use progress bar if you want more control over the configuration of the progress bar or intend to reuse it. A progress bar can be reused; a progress monitor cannot - a new progress monitor must be created.

The swing API contains one more useful progress monitor – **ProgressMonitorInputStream**. This monitor allows the programmer to monitor the amount of data from an input stream. It includes a reference ***progressMonitor*** to a ***ProgressMonitor*** object inside itself that the programmer can access to update the progress of the reading. For the most part the ***ProgressMonitorInputStream*** class contains many of the methods you can find in the *InputSteram* class.

When it is created, the ***ProgressMonitorInputStream*** attempts to read the amount of data available and updates automatically the progress monitor. Since it uses progress monitor, the file should be big enough in order to see the progress dialog.

Note: Because the progress monitor stream uses the *available()* method of the *InputStream* class to determine the total number of bits in the stream, it works well for files and HTTP URLs but it does not work well with streams which allow blocking.

References

Textbook – Chapter 23, p. 992
Java Swing, second edition.
Links:
http://en.wikipedia.org/wiki/Deadlock

JavaFX offers similar controls (*ProgressBar* and *ProgressIndicator*) but there is no equivalent to *ProgressMonitor*. To see a demo and learn how to use those JavaFX control visit the following link:

http://docs.oracle.com/javafx/2/ui_controls/progress.htm

Code Examples

You will find the examples in ***CST8221_HA06_code_examples.zip***.

Exercise

Download, compile, and run the code examples. You must have an open console to see the output from the ***ProgressMonitorInputDemo*** application. Once you see how they work, explore very carefully the code. Try to make the suggested modifications. Try to display the results from the ***ProgressMonitorInputDemo*** in a ***JTextArea*** component.

Visit the reference link provided above. You will find more demos exploring the progress related classes.

Questions

Q1. Can you change the appearance of a progress bar?
Q2. Can you change the text of the Cancel button of the progress monitor?
Q3. Can you reuse a progress monitor?
Q4. Can a progress bar show work without measurable progress?
Q5. Does JavaFX provide a ProgressMonitor control?

Submission

No submission is required for this activity.

Marks

No marks are allocated for this activity, but remember that understanding how *progress bars* and *monitors* works is essential for building a user friendly GUI.

And do not forget that:

"We call progress everything we do not understand." Anonymous

but also never forget that*:*

"Nobody can stop progress!" Another Anonymous