Design of user interfaces

Comprimir archvos en formato ZIP

2018/19 course - Practice 8

David Bohmann
Petr Lukašík
November 3, 2018

Contents

1. Assigment

It is time to see how we can build user interfaces for applications that do intense background tasks. In these cases, the User Interface must not lose control of the execution, it must keep the User informed at all times of what the status is, and level of progress, in which the task is located, as well as being able to stop it if the User considers it necessary.

Java Swing offers us the SwingWorker class to put our processes in the background. In this way, the User Interface is not blocked until the end of a process and can continue to be updated, informing the User, or interacting with it (for example, to receive the request for cancellation of the process).

Another class that we should know is JProgressBar, which allows to show the user a Progress Bar of the task.

The joint use of the two previous classes, and of all the knowledge that we have already acquired in the previous practices (especially in Practice 5 with the use of the JFileChosser class), will allow us to realize the User Interface of an Application that compresses to a zip file, all the files that are in a certain folder.

To generate the zip file we will use the *java.util.zip* package of classes provided by java, (an example of using these classes is included in the "Implementation Notes" section 1.1).

The functionality of the developing application in this practice must contain the following:

- Allow the user to select an existing folder (use of the JFileChosser class), which will be the folder that contains all the files to be compressed.
- Allow the user to select an existing output folder (use of the JFileChosser class), which will be the folder where the folder.zip file is generated.
- Once the folders have been selected and verified that they are the different folder (for obvious reasons), the application must allow using a button (JButton) to start the task of compressing the files in the background (using of the SwingWorker class).
- The progress of the task that is compressing the files must be shown through a Progress Bar (use of the JProgressBar class).

- The application must allow the user to cancel the generation of the zip file by means of a Cancel button (JButton), which will only be enabled when the compressed zip file is being generated.

1.1 Implementation notes

We have already said before that Java allows you to compress files in zip format by using the classes that are in the package java.util.zip. An example of how you can compress all files that exist in a List (use of the List class) is shown below:

Code for compressing file

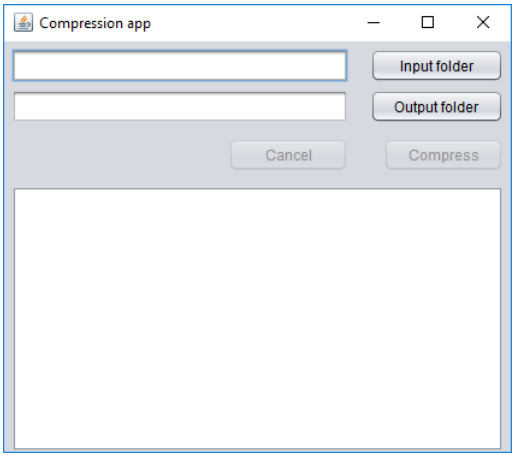
```
import java.util.zip.*;
import java.util.List;
...
 List<String> files = new ArrayList<String>();
 files.add("c:\\archivo_1.txt");
 files.add("c:\\archivo_2.txt");
 files.add("c:\\archivo_3.txt");
...
 files.add("c:\\archivo_n.txt");
 try
 {
 // Objeto para referenciar a los archivos que queremos
    comprimir
 BufferedInputStream origin = null;
 // Objeto para referenciar el archivo zip de salida
 FileOutputStream dest = new FileOutputStream("c:\\folder.zip");
 ZipOutputStream out = new ZipOutputStream(new
    BufferedOutputStream(dest));
 // Buffer de transferencia para mandar datos a comprimir
 byte[] data = new byte[BUFFER_SIZE];
 Iterator i = files.iterator();
 while(i.hasNext())
 {
 String filename = (String)i.next();
 FileInputStream fi = new FileInputStream(filename);
 origin = new BufferedInputStream(fi, BUFFER_SIZE);

 ZipEntry entry = new ZipEntry( filename );
 out.putNextEntry( entry );
 // Leemos datos desde el archivo origen y los mandamos al
    archivo destino
 int count;
```

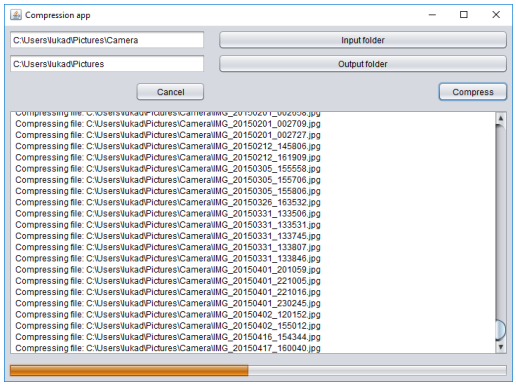
```
 while((count = origin.read(data, 0, BUFFER_SIZE)) != -1)
 {
 out.write(data, 0, count);
 }
 // Cerramos el archivo origen, ya enviado a comprimir
 origin.close();
 }
 // Cerramos el archivo zip
 out.close();
 }
 catch( Exception e )
 {
 e.printStackTrace();
 }
...
```

2. Implementation

Our application (shown in pic 2.1a) is based on assigment and contains two main areas. First area is filled with button settings in the upper side of application. It contains buttons for choosing the input and output folder (using the FileChooser class). The input and output folder cannot be the same and if user chooses to do so, he is notified by error dialog. Both folder paths are shown in noneditable Textfield area on the left from the buttons. After both paths are chosen, compress button is enabled and user can start compressing which starts in another thread (SwingWorker class). After compressing starts, Cancel button is enabled and user can cancel the compressing operation at any time. If he does it, he is notified about it by dialog window (as shown in pic 2.2b). Second area is underneath in the center/bottom position containing TextArea as output console. If compressing operation starts, it shows all output like preparation and compression of chosen files at current time. As well when compressing starts it is shown a progress bar under textArea (pic 2.1b). When compression is done, it is shown success dialog box (pic 2.2a) and a folder.zip file is created in the output folder. The zip file contains only files in the input folder and not whole path with folders.

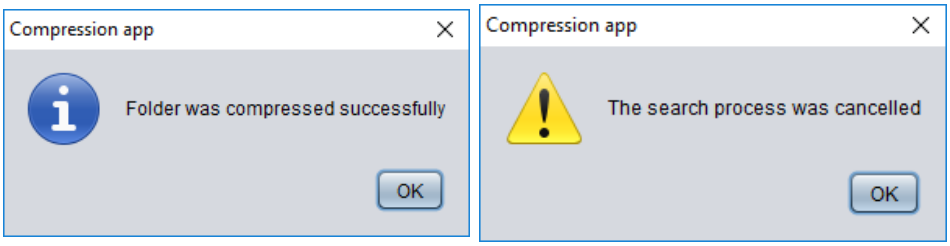


(a) State of application on the start



(b) State with compressing in progress

Figure 2.1: Compression application



(a) Detail on success

(b) Detail on cancellation

Figure 2.2: Different compression finished state

3. Shneiderman and Plaisant principles

- **Consistency**
  User can differ settings area in upper part of application from output area in center/bottom side. From previous applications user should be already acknowledged with using of FileChooser.
- **Universal usability**
  We did not implement any shortcuts or hidden functions to ease the using of the application because there are not that many interactions for this to be neccessary.
- **Informative feedback**
  When choosing folder, its path is shown in TextFields. Compression progress is shown in output TextArea and progress bar. Success or failure is shown with dialog boxes.
- **Design dialogue to yield closure**
  Progress of compression is shown by writing the currently compressed file in the output TextArea.
- **Simple error handling.**
  Cancallation of progress or choosing the wrong output folder (same as input or without permission) is shown by error dialog box.
- **Permit easy reversal of actions**
  There is no setting to reverse actions because its not neccessary.
- **Support internal locus of control**
  The user is in full control of choosing input and output folders. User controls compression and can cancel it anytime. If not, he is informatively notified why.
- **Reduce short-term memory load**
  For such a small application our application is fast enough for human needs.