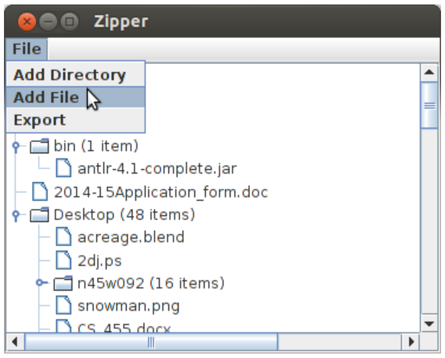
CS 245 Homework 2
Zipper

1 Overview

In the 1980s, University of Wisconsin–Milwaukee graduate Phil Katz formed PKWARE, Inc. His first major product was PKARC, a shareware clone of an existing tool that combined and compressed multiple files into a single archive file of reduced size. Computer networks at the time were considerably slower than they are today, so compression tools like PKARC were vital if one wanted to share files. PKWARE swam in profits. Katz hired his mother to help manage the business. However, Katz was soon sued for copyright and trademark infringement by the developers of the tool that he cloned. After settling with the plaintiff, Katz developed a new compression tool called PKZIP that implemented a new and fast ("zippy") compression protocol. In 2000, Katz was found dead in his hotel room. The medical examiner cited the cause of death as pancreatic bleeding brought on by chronic alcoholism. Katz was 37.

Katz's legacy lives on in the pervasive and free ZIP standard. Your task is to write a graphical tool for constructing ZIP files. Users may add entire directories and individual files through entries in the tool's File menu. The added contents are displayed using a `JTree`. When a user is ready to create the ZIP file, she selects File / Export. Base your interface off of the following screenshot:



2 Requirements

Specifications do not tell you how to solve a problem—just what pieces may be used. The classes and methods described below will need to be thought about and pieced together using your own good mind. You will likely need to read this section many times.

Your solution is to meet the following specification:

1. Write all code in your fork of the class Bitbucket project, in package `hw2`.

2. Use the Java's library for creating ZIP files. Consider this short example, which creates an archive that when unpacked produces two directories and two files:

```
File outFile = new File("archive.zip");
ZipOutputStream zos = new ZipOutputStream(new FileOutputStream(outFile));
zos.putNextEntry(new ZipEntry("dir-a/file1.txt");
zos.write("this text is in dir-a/file1.txt".getBytes());
zos.putNextEntry(new ZipEntry("dir-b/file2.txt");
zos.write("this text is in dir-b/file2.txt".getBytes());
zos.close();
```

Each file is written as a `ZipEntry`. Generally in ZIP files, we do not explicitly add directories. They are implied by paths sent to the `ZipEntry` constructor. Note that in this example we fake the contents of the two files. In your work, you'll need to explicitly open a file and read in its contents.

3. Write a class `Utilities` with the following specification:

   (a) A `static` method `slurp` that accepts a `File` parameter. It reads in the entire contents of the specified file and returns the contents as a `byte` array. Consider how this may be useful, given the example code above. If you use the `RandomAccessFile` file class, this problem can be solved in just a few lines of code. You may assume the file's size is less than `Integer.MAX_VALUE`.

   Test before moving on.

4. Write an `abstract` class `ZipperNode` with the following specification:

   (a) Is a subclass of `DefaultMutableTreeNode`. This relationship means we can make a `JTree` out of `ZipperNode`s.

   (b) Has a constructor accepting a `File` parameter identifying the file whose contents are to be zipped.

   (c) Has a getter named `getFile` for the `File` this node represents.

   (d) Has an `abstract` method `enzip` that appends the node's contents to a ZIP file being written. It accepts a `String` parameter for the output path of its parent and a `ZipOutputStream` to write to.

   (e) Has a `static` method `generateHierarchy` that creates a node hierarchy rooted at a given file and returns it as a `ZipperNode`. It accepts the root of the hierarchy as a `File` parameter. If the file is a directory, then a `ZipperDirectory` is returned. Otherwise, a `ZipperFile` is returned.

5. Write a class `ZipperFile` with the following specification:

   (a) Is a subclass of `ZipperNode`.

   (b) Has a constructor accepting a `File` parameter identifying the file whose contents are to be zipped.

   (c) Has a `toString` method that returns the name of this node's file. No parent directory information is included.

   (d) Has a method `enzip` that overrides the superclass version to write a new `ZipEntry` out to `ZipOutputStream`. If the node has no parent directory—meaning the output path of the parent is the empty string—then the `ZipEntry` label is just the name of this file. Otherwise, the label is of the form `output-path-of-parent/name-of-file`. Writing the file requires two steps: 1) starting a new entry and 2) writing out the file's contents. If the writing fails, let an `IOException` be thrown. We don't know how to handle that error at this level.

Test before moving on.

6. Write a class `ZipperDirectory` with the following specification:

   (a) Is a subclass of `ZipperNode`.

   (b) Has a constructor accepting a `File` parameter identifying the directory whose contents are to be zipped. Add its contents as children of this node by traversing through the files inside this directory, treating each item as the root of hierarchy, and adding the hierarchy to this node using the inherited `add` method.

   (c) Has a `toString` method that returns the name of the directory followed by a description of the number of items it contains in the form " (# items)", correctly pluralized [e.g., `dir1 (1 item)` or `dir2 (3 items)`]. No parent directory information is included.

   (d) Has a method `enzip` that overrides the superclass version to add all this directory's children to the ZIP file being written. This can be done recursively by `enzip`ping all the child nodes. One can iterate through the children with the inherited `getChildCount` and `getChildAt` methods. If the directory has no parent directory—meaning the output path of the parent is the empty string—then this directory's output path is just its name. Otherwise, the output path is of the form `output-path-of-parent/name-of-directory`. If the writing fails, let an `IOException` be thrown. We don't know how to handle that error at this level.

Test before moving on.

7. Write a class `Zipper` with the following specification:

   (a) Is a subclass of `JFrame`.

   (b) Has a file menu with entries to add a directory, add a file, and export the added items to a ZIP file. I suggest you use a `JFileChooser` for prompting the user to select a file. This class has options for limiting the selection to directories or choosing between opening a file for reading or writing.

   (c) Shows the added files in a `JTree`. When a file or directory is added through the menu options, create a `ZipperNode` hierarchy for the selected file and add it to the tree. Many independent file hierarchies may be added. However, `JTree`s must have exactly one root. Create a fake, non-`ZipperNode` and invisible root to which all the `ZipperNode`s may be added with:

   ```
   DefaultMutableTreeNode root = new DefaultMutableTreeNode("Nodes");
   JTree tree = new JTree(root);
   tree.setRootVisible(false);
   tree.setShowsRootHandles(true);
   ```

   Then you can insert the `ZipperNode` hierarchies you make and update the display with:

```
ZipperNode node = /* code to generate a hierarcy for selected file */
root.add(node);
((DefaultTreeModel) tree.getModel()).nodeStructureChanged(root);
```

(d) Has a default constructor that adds to the frame the menu and the initial `JTree` wrapped up inside a `JScrollPane`. It also sets the default close operation to exit, gives the frame some non-zero size, and makes the frame visible.

(e) Has a method `add` that accepts a `File` parameter. It adds the `File`—which may be either a directory or a plain old file—to the `JTree` and updates the display.

(f) Has a method `export` that accepts a `File` parameter. It creates a new `ZipOutputStream` for the given file, iterates through all the top-level `ZipperNode`s in the `JTree` (the children of the fake root node), and `enzip`s each one. None of the top-level nodes has a parent, so the initial output path you send to `enzip` should be the empty string. If an exception occurs, catch it and show the exception's message with `JOptionPane.showMessage`.

8. Write a class `Main` with the following specification:

   (a) Has a `main` method. What it does is not specified, but I suggest you use it to test your code. Relying exclusively on the SpecChecker to test things will rob your brain of some neurons that are in your best interest to grow.

3 Submission

This homework is a regular assignment and is graded by hand and with help from the SpecChecker. To submit your work for grading:

1. Put the SpecChecker for this homework in your Build Path.
2. Run the SpecChecker as a Java Application (not a JUnit Test) and fix problems until all tests pass.
3. Commit and push your work to your repository. If you are resubmitting an earlier assignment, email me. The time of your email will determine the submission week.

The SpecChecker cannot check everything. Your assignment is also expected to fully meet the requirements above and the following:

- Variable names should be meaningful and accurate.
- Non-obvious parts of your code should be commented.
- Code should be cleanly formatted and indented.
- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. (If you find violators of this rule, please let me know.)
- Work must be submitted according to course policies on deadlines. To be eligible for later-week submission, you must have at minimum the skeletons for all specified classes and methods in your repository by the homework deadline.