Project 4: Password Cracker

NOTE: *If you turn in a solution to this problem based on something from a pay-for-answers website such as Chegg, CourseHero, etc., you will receive an F in this course (not just on this assignment). Start early and seek help from a lab assistant, the help room, or your professor if you need it. If you don't have time to do that, it is better to take the zero than to cheat.*

Background

`.zip` files can be protected with passwords, which are intended to make their contents inaccessible unless you know the password with which the `.zip` was encrypted. There is no limit to the number of times you can try different passwords however, so someone who did not know the password could theoretically try every possible password until they find the correct one. This is the idea behind a brute-force password cracker, which is the focus of this project.

For this assignment, you have been provided (on Pilot) with an external library called `zip4j-1.3.2.jar`, which contains some handy classes that will help you interface with `.zip` files. You will need to unzip (i.e. expand) this file before you can use it. On most operating systems you can do this just by double-clicking on the file. You have also been provided with some starter code called `Example.java` that tries a single password on an encrypted zip file. You can use the following commands to compile and run this code:

Compiling:
```
javac -cp zip4j-1.3.2.jar Example.java
```

Running:
On Windows:
```
java -cp .;zip4j-1.3.2.jar Example
```
On Mac and Linux:
```
java -cp .:zip4j-1.3.2.jar Example
```

Download the file `example.zip` (also available on Pilot) and put it in the same directory as the `zip4j-1.3.2.jar` and `Example.java` files. Compile and run the code, and you should see the message "Wrong password." Edit `Example.java` to change the value of the password variable from `bad` to `good` and re-compile and run the program, and you should see the message "File decrypted." ***Be sure to do this right away*** so that if you have trouble you can attend office hours or visit the CSE Help Room.

Part 1

You have been provided with a file called `protected3.zip` (this file is on Pilot) which is protected with a password. You are to write a program that tries all possible passwords until it finds the correct one. Your program should display the correct password and the time it took to find it (`System.currentTimeInMillis()` gets the current system time).

The password you are looking for is exactly three characters long and contains only lowercase letters. No numbers, no symbols. Note that trying to figure out all possible three-character strings over an alphabet is a great place to use recursion!

Part 2

You have also been provided with another file, called `protected5.zip`. As the name implies, this file is protected by a five-character password, still only made of lowercase letters. Because this password has more characters, your approach from Part 1 will take a long time to finish. Therefore, for this step you need to multithread your original implementation. This is required – you will not receive credit for programs that break this password using only a single thread.

Your program must include a variable called `numThreads` that contains a hard-coded integer value to control the number of child threads spawned to look for the password. You can tweak this setting to your liking while testing your program – just make sure it is named as requested so that we can easily do the same.

The multi-threaded version of your program should still quit as soon as any of the threads finds the password. To accomplish this, you will need a way to enable the main thread to detect when any of its child threads successfully finds the password and, if/when that happens, terminate the remaining child threads and quit. This is a great place to use a volatile static variable.

When timing the amount of time the program takes, include both the time to generate the passwords and the time for the threads to run. Run your program with three and then with four threads and put the number of milliseconds it took to get the password as a comment at the top of your driver class (i.e. the one with `main` in it).

Important: Because all of your threads will be trying to decrypt the file, if they were to all use the same file, that would create a bottleneck. To avoid this, each of your threads needs to make its own copy of the file to work on. Also, the method `extractAll()`, in addition to testing the password you set using `setPassword()`, leaves behind a directory (called `contents` in the starter code) that contains the results of its attempt at decrypting the `.zip` file. Since now all of the child threads will be using the `extractAll()` method, you'll need a way to specify a different place for each one to write its results to by changing "contents" to something unique for each thread like "contents-0", "contents-1", etc. Your threads should delete their copy of the target file and their contents directory when they finish. To receive full credit, this must all be done programmatically – you cannot manually make copies of the protected file before your program runs.

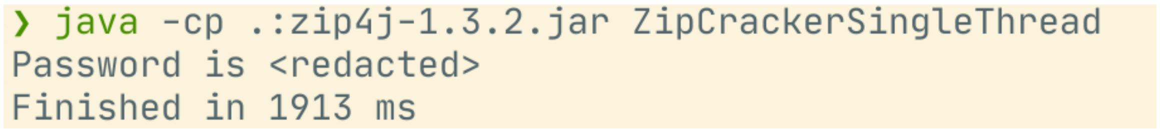
As a reminder, here is some code that copies / deletes files:

```
import java.nio.file.Files;
import java.nio.file.Path;

Files.copy(Path.of(srcFilename), Path.of(destinationFilename));
Files.delete(Path.of(filename));
```

Keep in mind that to delete a directory, you must first delete any files inside it.

Examples



```
❯ java -cp .:zip4j-1.3.2.jar ZipCrackerSingleThread
Password is <redacted>
Finished in 1913 ms
```

Turn In Instructions

If you have a multi-threaded version working that will also accept numThreads = 1 to run in a single thread, you only need to turn in that code. If you are not confident your multi-threaded version is working correctly, please turn in both your single-threaded version and the multi-threaded version so that you can get as much partial credit as possible.

Rubric

Note: Code that does not compile will receive a zero.

[30 pts] The program can crack any three-character password that contains only lower-case letters (not just the specific sample given) and reports the time it takes to do so.

[5 pts] The number of worker threads is controlled by a variable called numThreads.

[15 pts] Each worker thread works on its own files and cleans them up before the program terminates.

[15 pts] Each thread works independently on a different (non-overlapping) part of the password search space.

[15 pts] The program terminates when any thread finds the correct password.

[5 pts] Timing information for three versus four threads is included as a comment at the top of the driver program.

[15 pts] The program is clearly organized, commented, and follows standard coding practices, including variable and class names.