CS-1181 Project 4: Password Cracker

PURPOSE: The main objective of this project is to develop a deeper understanding of concurrency in Java. Multi-threading allows a program to handle multiple tasks at the same time which can have many benefits including faster execution. Completing this project will not only strengthen concurrency comprehension but also demonstrate the advantages of multi-threading in real time.

PROBLEM: .zip files can be protected with passwords, which make their contents inaccessible unless you know the password with which the .zip was encrypted. However, there is no limit to the number of times you can try different passwords, so someone could theoretically try every possible password until they find the correct one. This is the idea behind a brute-force password cracker.

For this assignment, you have been provided with an external library called `zip4j-1.3.2.jar` (on Pilot), 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:

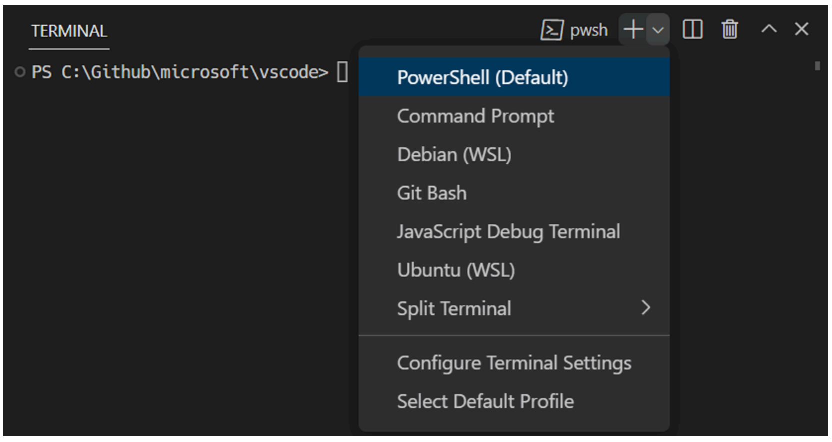
```
java -cp .:zip4j-1.3.2.jar Example
```

Note that you must be in the same directory as your .java files for these commands to work. To check this, type `dir` on Windows or `ls` on Mac and make sure your java files are in the list. If they are not, use the `cd` command to change to the directory (folder) where your files are. For example, if my .java files were on my desktop, I would type: `cd Desktop`.

For those using Visual Studio Code on Windows:

Go to `Terminal -> New Terminal`

Next to the + sign in the upper right of the terminal window, choose Command Prompt.

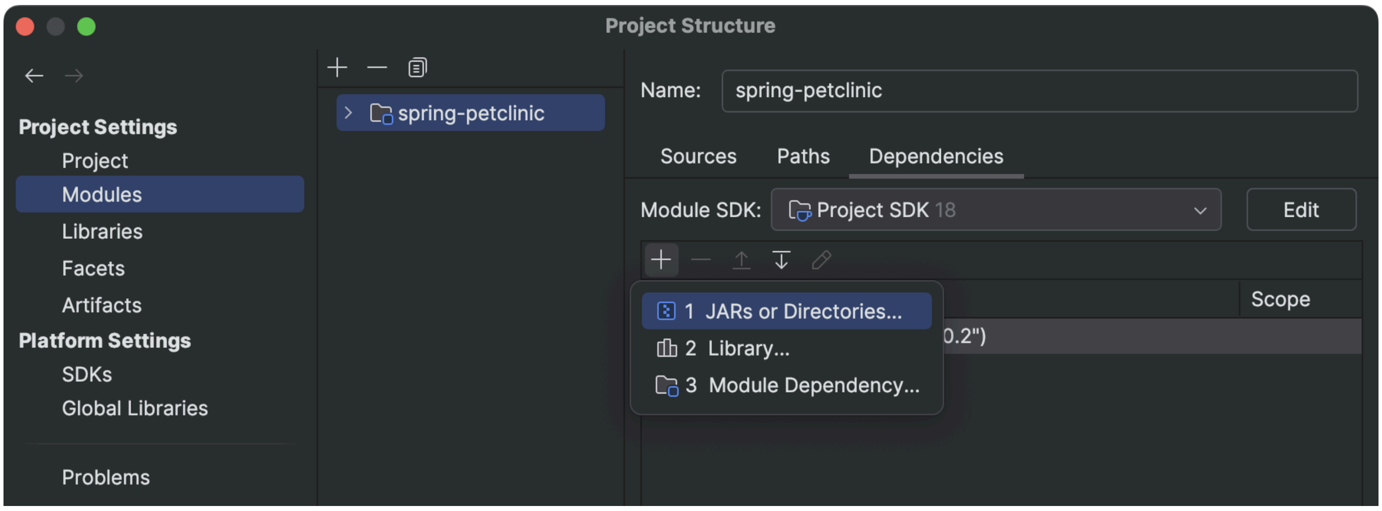


```
PS C:\Github\microsoft\vscode>
```

Download the file `example.zip` (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. Then, re-compile and run the program. You should see the message “File decrypted.” Be sure to do this right away, so if you have trouble you can attend office hours or visit the Help Room.

If you would like to avoid running the code via the command line, you can also add the .jar file as a dependency in your project.

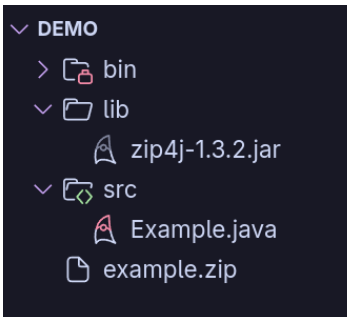
IntelliJ: To add a dependency in IntelliJ, navigate to `File > Project Structure` (Ctrl+Alt+Shift+S). Then click on `Modules` and `Dependencies`. Click on the + and select `JARs or Directories`. Finally, select the location of your .jar file.



Name: spring-petclinic

Module SDK: Project SDK 18

VSCode: To add a dependency in VSCode, navigate to the left-hand panel and find the `lib` folder. Then, drag and drop your .jar file into the folder.



Part 1

You have been provided with a file called `protected3.zip` (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. There are no numbers or 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 multi-thread 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's named as requested, so we can 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 for 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 how long the program takes, include both the time to generate the passwords and the time for the threads to run. Run your program with three, then four threads, and put the number of milliseconds it took to get the password as a comment at the top of your driver class.

Important: Because all 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 `extractAll()` method leaves behind a directory (called `contents` in the starter code) that contains the results of its attempt at decrypting the .zip file. Since all of the child threads will be using the `extractAll()` method, you'll need to specify a different place for each thread 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 (do not manually make copies of the protected file before your program runs).

As a reminder, below is some code that copies / deletes files. Keep in mind that to delete a directory, you must first delete any files inside it.

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

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

Example:

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

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 you can get as much partial credit as possible.

RUBRIC:

(85 pts) **Base Functionality**

Part 1

[25] The program can crack ***any*** three-character password, containing only lower-case letters

[5] The program reports the time it takes to crack the three-character password

Part 2

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

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

[15] Each thread works independently on a different/non-overlapping password search space part

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

[5] Timing information for three versus four threads is in a comment at the top of the driver program

(15 pts) **Style**

[10] Code is clearly written and follows standard conventions (variable names, indentation, etc.)

[5] The code is meaningfully commented

IMPORTANT NOTES:

- **Submissions that do not compile will receive a zero**
- **Submissions with improperly cited AI will receive a zero and an academic integrity violation**
- **Submissions that are partially or fully copied from another submission will receive a zero and an academic integrity violation**