Homework 13 - Ls-Cat

Authors: Carl Schriever, Shishir Bhat, Ryan Miles, Tushna Eduljee, Emma Barron, Allan Nguyen, Manny Sonubi, Peter Dong, Lauren Prinn

Problem Description

Still stuck in the future, you're starting to enjoy it. You've helped space explorers improve their data infrastructure, and briefly got launched into the world of Pokémon where you've left Professor Oak and the Pokemon Center eternally grateful for your services.

Unfortunately, you learn that "Georgia Tech University" was really just a front for a school named "Georgia Tech"! Professor Oak reveals himself to be the professor of a class named CS 1331 and tells you to get back to your homework and to stop reading pointless problem descriptions.

Solution Description

For this assignment, create the following files:

- `FileHelper.java`
- `CatException.java`
- `Tester.java`

FileHelper Class:

FileHelper will contain static helper methods that have functionality similar to the `ls` and `cat` unix commands.

Methods:

- `public static void printLs(String path)`
  - This will print out all of the files and folders that are contained within the folder located at `path`.
  - You must use the `printLsHelper` method below.
  - You may assume that the `path` parameter contains the path to a folder.
- `private static void printLsHelper(String path, int indentationLevel)`
  - **Must recursively call itself**
  - For every indentationLevel, print 4 spaces at the beginning of the line before printing anything else.
  - If the path that is passed in is a path to a file, print out the name of the file.
  - If the path that is passed in is the path to a folder, print out the name of the folder and then, indented a level, print out the contents of the folder (Hint: recursion).
  - This problem must be solved recursively or you will receive a heavy deduction (Hint: if you are not making at least one recursive call per folder, you are probably doing it wrong).
  - See the example below in testing your code for formatting to follow!
- `public static void printCat(String path)`
  - If `path` isn't a path to a real file or is a directory, throw a CatException. The message should be `"Path invalid."` and pass in the `path`.
  - Otherwise
    - For each line in the file, print out `"{line number}: {line from text file}"`
    - See the example below in testing your code for formatting to follow!

CatException Class:

- Make this custom Exception be a Checked Exception
- Take a `String message` and `String problemFile` into the constructor. Pass `message` to super and store `problemFile` in a private final field called `problemFile` to keep track of the path to the file.
- Override getMessage() to include the original message (Hint: use super's getMessage()) plus the absolute path to the `problemFile` in the following format: `"{original message}. Problem with file {absolute path}!"`
  - `problemFile` may be an absolute path or a relative path. Make sure to print out an absolute path.

Tester Class:

- Have a main method that takes in a file path in a command line argument. Print the output of `ls` of the current directory and then print the output of `cat` with the passed in file.
  - **Make sure your path to the current directory is a relative path and not a hard-coded or absolute path.**
  - **Do not use backslashes ("\") when indicating the current directory. Use forward slashes instead.**
  - Your main method should not handle any Exceptions that could occur when calling printLs or printCat and instead propogate them to the caller of the main method.

Testing your code:

The below examples will use the folder structure

```
Foo
|- Bar
|  |- Biz
|  |  |- Hi.txt
|  |  |- Shishi.png
|  |- Lambda.java
|- Bat
|  |- Scary.gif
|- Stop.txt
|- Spooky.txt
|- FileHelper.java
|- FileHelper.class
|- CatException.java
|- CatException.class
|- Tester.java
|- Tester.class
```

Assume that Spooky.txt contains:

```
In the cool of the evenin' when ev'rything is gettin' kind of groovy
I call you up and ask you if you'd like to go with me and see a movie
First you say "no", you've got some plans for the night
And then you stop, and say, "all right"
Love is kinda crazy with a spooky little girl like you
```

and Stop.txt contains:

```
In Defense of Christmas Music in October.
Ah, Christmas music! The warm tunes that remind so many of us of
```

```
huddling around a fire surrounded by family.
The sounds that remind of pine scents, mistletoe, and holiday cheer.
There's so much good feeling attached to Christmas music that it's nice to dip
into the holiday playlists early and get excited about what's to come.
```

`printLs("Foo")` would print out

```
Foo
    Bar
        Biz
            Hi.txt
            Shishi.png
        Lambda.java
    Bat
        Scary.gif
    Stop.txt
    Spooky.txt
    FileHelper.java
    FileHelper.class
    CatException.java
    CatException.class
    Tester.java
    Tester.class
```

`printCat("Foo/Spooky.txt")` would print out

```
1: In the cool of the evenin' when ev'rything is gettin' kind of groovy
2: I call you up and ask you if you'd like to go with me and see a movie
3: First you say "no", you've got some plans for the night
4: And then you stop, and say, "all right"
5: Love is kinda crazy with a spooky little girl like you
```

Running `java Tester ./Stop.txt` would output the following assuming that you are currently in the Foo directory

```
Foo
    Bar
        Biz
            Hi.txt
            Shishi.png
        Lambda.java
    Bat
        Scary.gif
    Stop.txt
    Spooky.txt
    FileHelper.java
    FileHelper.class
    CatException.java
    CatException.class
    Tester.java
    Tester.class
1: In Defense of Christmas Music in October.
2: Ah, Christmas music! The warm tunes that remind so many of us of
3: huddling around a fire surrounded by family.
4: The sounds that remind of pine scents, mistletoe, and holiday cheer.
5: There's so much good feeling attached to Christmas music that it's nice to dip
6: into the holiday playlists early and get excited about what's to come.
```

Depending on how you implement it Foo may just be `.` or `./` in all of the above examples and that is okay.

Allowed Imports

- You are only allowed to import the following classes:
  - `java.util.Scanner`
  - `java.io.*`
  - `java.nio.file.*`

Rubric

- [55] FileHelper.java
  - [2.5] printLs()
    - [2.5] Correct method header
  - [35] printLsHelper()
    - [2.5] Correct method header
    - [12.5] Calls itself recursively
    - [10] Output has correct spacing for each indentation level
    - [10] Prints file name when appropriate
  - [17.5] printCat()
    - [2.5] Correct method header
    - [10] Cat command has correct output
    - [5] Throws CatException
- [25] CatException.java
  - [5] Overrides getMessage() with correctly formatted message
  - [5] Is a checked exception
  - [2.5] Call to super in constructor
  - [2.5] Call to super in getMessage()
  - [5] Correct instance fields
  - [5] Constructor with two parameters correctly assigned
- [20] Tester.java
  - [10] Takes in command line arguments
  - [10] Handles exceptions thrown when calling printLs or printCat

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
* @author George P. Burdell
```

```
 * @version 1.0
 */

public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */

    public Dog() {
    ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */

    public int add(int a, int b) {
    ...
    }
}
```

A more thorough tutorial for Javadocs can be found here. Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here. To run Checkstyle, put the `jar` file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **100 points**. This means that up to 100 points can be lost from Checkstyle errors.

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment AT THE TOP of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `FileHelper.java`
- `CatException.java`
- `Tester.java`

Make sure you see the message stating "HW13 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.