CS 1103-01 Programming 2 - AY2023-T1

Lab 2: Exceptions/Sorting

This lab has two parts, with no real connection between them. The first part of the lab asks you to find out how long it takes to sort a large array using both a sorting method that you write and using Java's built-in sorting method; this part of the lab is part of this course's introduction to the analysis of algorithms. In the second part of the lab, you will write a program that does some simple networking and file I/O. Since these operations can generate checked exceptions, you have to use try..catch in your program to handle the potential exceptions.

You will probably want to start a new Eclipse project for this lab. However, note that a project can contain any number of programs, so you don't necessarily need a new project for every small program that you write.

For this assignment, you will write two programs for this lab. You will generate some data using the first program -- you should include this data in a comment at the beginning of the program file.

Part 1: Benchmarking Sorting Algorithms

The same task can take vastly different amounts of time, depending on the algorithm that is used to perform the task. You are familiar with simple sorting algorithms such as insertion sort and selection sort. (See Section 7.4 in the textbook.) While these methods work fine for small arrays, for larger arrays they can take an unreasonable amount of time. The question is whether we can do any better.

Java has some built-in sorting methods. They can be found in the class named Arrays in the package java.util. The one that you will use in this lab is Arrays.sort(A), which sorts the entire array A into increasing order. (Actually, there are different methods for different array base types, but all the methods have the same name and are used in the same way. You will be using an array of ints in this lab.)

You should write a program that does the following:

- Create two arrays of type int[]. Both arrays should be the same size, and the size should be given by a constant in the program so that you can change it easily.
- Fill the arrays with random integers. The arrays should have identical contents, with the same random numbers in both arrays. To generate random integers with a wide range of sizes, you could use (int)(Integer.MAX_VALUE * Math.random()).
- Sort the first array using either Selection Sort or Insertion Sort. You should add the sorting method to your program; you can copy it from Section 7.4 if you want. (It is a good idea to check that you got the sorting method correct by using it to sort a short array and printing out the result.)
- Time how long it takes to sort the array and print out the time.

- Now, sort the second (identical) array using Arrays.sort(). Again, time how long it takes, and print out the time.

You should run your program using array sizes of 1,000, 10,000, and 100,000. Record the sort times. **Add a comment to the top of the program that reports the times.** (You might be interested in applying Arrays.sort() to a million-element array, but don't try that with Selection Sort or Insertion Sort!)

Note: The general method for getting the run time of a code segment is:

```
long startTime = System.currentTimeMillis();
doSomething();
long runTime = System.currentTimeMillis() - startTime;
```

This gives the run time in milliseconds. If you want the time in seconds, you can use runTime/1000.0.

Part 2: Programming with Exceptions

In this part of the lab, you will write a program that fetches the information stored at a given URL on the web and saves that data to a file. This will also include networking and file operations and partly an exercise in using exceptions.

For doing I/O, Java has a pair of nice abstractions: InputStream and OutputStream. These are abstract classes in the package java.net. An InputStream is a place from which you can read data; an OutputStream is a place to which you can write data. For this lab, you will use an InputStream to represent the data read from the Web URL, and you will use an OutputStream to represent the file where you want to save a copy of the data. Once you have the streams, the data can be copied just by calling the following method, which you can copy into your program:

```
private static void copyStream(InputStream in, OutputStream out)
throws IOException {
int oneByte = in.read();
while (oneByte >= 0) { // negative value indicates end-of-stream
out.write(oneByte);
oneByte = in.read();
}
}
```

Aside from this method, you should have a main routine that does the following:

- Declare variables to represent the InputStream and the OutputStream. It would be a good idea to initialize them to null to avoid uninitialized variable errors.

- Read the URL and the file name as strings from the user.

- To connect to the web, you need a variable -- say url -- of type URL (from package java.net). You can create the URL object with the constructor call url = new URL(urlString), where urlString is the string provided by the user. This constructor will throw a MalformedURLException if the string is not a legal URL. (Note: the string must be a complete URL, beginning with "http://".)

- To get the input stream, you can simply call url.openStream(), which returns a value of type InputStream. This can throw an IOException, for example, if the web address that you are asking for does not exist.

- To get the output stream, you can use the constructor new FileOutputStream(fileName), where fileName is the file name that was input by the user. This can throw a FileNotFoundException if it is not possible to open the specified file for reading (for example, if the user is trying to create a new file in a directory where they don't have write permission). **Warning:** If a file of the same name already exists, the old file will be erased and replaced by the new one, without giving the user any notice!

- Now, copy the data from the web into the file by calling the above method. Note that this can throw an IOException.
- Lasta, use a *finally* to clause to make sure that both streams are closed (if they were successfully opened). Both InputStream and OutputStream have a close() method for closing the stream. Note that you can test whether the stream was opened by testing whether the value of the variable is still null.

Note that an exception should not crash your program. You should catch the exception and print out a reasonable error message before ending the program. It would be nice if the error message depends on the type of error that occurred (which means using several catch clauses).

Last modified: Monday, 28 June 2021, 8:07 AM

UoPeople Clock (GMT-5)

All activities close on Wednesdays at 11:55 PM, except for Learning Journals/Portfolios which close on Thursdays at 11:55 PM **always following the clock at the top of the page.**

Due dates/times *displayed* in activities will vary with your chosen time zone, however you are still bound to the **11:55 PM GMT-5** deadline.

◄ Lab 1 Unit 1

Jump to...

Code ►

Resources
- UoPeople Library
- Orientation Videos
- LRC
- Syllabus Repository
- Career Resource Center
- Honors Lists

Links
- About Us
- Policies
- University Catalog
- Support
- Student Portal

Instructors