National University of Singapore
School of Computing
CS2105: Introduction to Computer Networks
Semester 1, 2015/2016

Assignment 0
Warm-up

Due: 31 August 2018, 23:59
Total marks: 3

Overview

This assignment is a warm-up for you to familiarize yourself with some Java classes and programming skills that will be useful for the later assignments.

This programming assignment is worth 3 marks. All work in this assignment should be done **individually**.

Testing

You are free to use any editor/IDE that you are familiar with. However, your program should compile and run correctly on the SoC `sunfire` server. This is because we will test and grade your program on `sunfire`.

To test your program, you should log in to `sunfire` using your SoC UNIX id and password. If you do not have an SoC UNIX account, please create it at: `https://mysoc.nus.edu.sg/~newacct`. If you forgot your password, please reset it at `https://mysoc.nus.edu.sg/~myacct/iforgot.cgi`.

For Mac or Linux users, SSH should be available from the command line. You can simply type `ssh <username>@sunfire.comp.nus.edu.sg` to log in.

For Windows 10 users, the Fall Creators Update included SSH as a Feature-On-Demand. Installation instructions can be found at `https://blogs.msdn.microsoft.com/powershell/2017/12/15/using-the-openssh-beta-in-windows-10-fall-creators-update-and-windows-server-1709/`

For other Windows user, you will have to have an SSH client installed. If not, you can download PuTTY from `http://www.putty.org/`.

Copying files to sunfire

For Mac, Linux or Windows 10 (with OpenSSH installed), you can use `scp` in the terminal:

```
scp -r <source_dir> <unix_id>@sunfire.comp.nus.edu.sg: <dest_dir>
```

Otherwise, you can transfer your program with an SFTP client like WinSCP to connect to `sunfire` and transfer your files.

Test suite

We provide an autotest script to check that your code is properly zipped and meets the requirements. Note that passing the test cases does not guarantee that you will get full marks. It just means that the autotester has managed to compile and run your program with the sample test cases. Additional private test cases will be used during grading, so you are advised to test your program thoroughly. You may assume all input data is valid so there is no need to input validation, unless otherwise stated in the question.

To use the autotester, log in to `sunfire` and run the following command:

```
~cs2105/autotest/a0 a0-a123456.zip
```

where `a0-a123456.zip` is the name and path to the zip file to be submitted.

Submission

You will submit your programs on IVLE. Zip your files into a single zip file and name it "`a0-<matric number>.zip`" and upload to the Assignment 0 folder. For example, if your matric number is A0123456X, then you should name your file "`a0-a0123456x.zip`".

The dateline for submission is at 31 August 2018, 23:59. Late submission will be penalized by 1 mark.

Grading

Each program is worth 1 mark and will be graded with an auto-grader. **There is NO manual grading**. As such, please ensure your programs and classes are named correctly and follow the given sample runs exactly as the grading scripts are unable to award partial marks.

Please take note of these common issues:

- For questions that require output to screen, it should be sent to Standard Out, NOT Standard Error.
- For questions that require output to screen, you should not have any extra output that might be interpreted as the incorrect answer.

Warning against plagiarism

You are free to discuss this assignment with your friends or on the forum. But no posting of solution is allowed on the forum. You should ultimately write your own code from scratch. Writing your code while referring to a friend's solution is still considered as plagiarism. The same goes for submitting code found from public sources like GitHub or Stackoverflow.

We employ zero-tolerance policy against plagiarism. If a suspicious case is found, student would be asked to explain his/her code in person to the evaluator. Confirmed breach may result in zero marks for the assignment and further disciplinary action from the school. This also applies to the student who allowed his/her code to be copied. So do not post your solutions in any public domain on the Internet.

Languages for Submission

Java

We will use the Java 9 on `sunfire` to run your code. The location of the Java executables on `sunfire` is at `/usr/local/java/jdk/bin/`.

As with Java requirements, the `.java` file has to have the same as the class. Any extra files that you use should be properly referenced and zipped in the proper directory structure.

Before you start, you might wish to refresh yourself on how to use command-line arguments: http://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html

Python

We will use Python 3.7 on `sunfire` to run your code. The location of the Python executables on `sunfire` is at `/usr/local/Python-3.7/bin/python`. Note this is NOT the default `python` that is on the path.

For each of the exercises, simply use the same name but with a `.py` extension. e.g., `IPAddress.py`, `Calculate.py`, etc. When evaluating your program, we will run `python3 <program>.py` on `sunfire`. Your program should produce the same behaviour as the Java counterpart. The zip file that you submit should only contain `.py` files. If any `.java` files are found, we will only evaluate the Java version instead.

Before you start, you might wish to refresh yourself on how to use command-line arguments: https://docs.python.org/3.6/library/sys.html

Exercise 1 — IPAddress

This exercise is a refresher on converting a binary string input to integers.

An IP address is a 32-character long bit sequence (a bit is either 1 or 0). You are to write a program that reads an IP address from **interactive user input as a string** and then convert it to a dotted decimal format. A dotted decimal format for an IP address is formed by grouping 8 bits at a time and converting the binary representation into decimal representation.

For example, IP address 00000011100000001111111111111110 will be converted into dotted decimal format as: 3.128.255.254. This is because:

1. the 1st 8 bits 00000011 will be converted to 3,
2. the 2nd 8 bits 10000000 will be converted to 128,
3. the 3rd 8 bits 11111111 will be converted to 255 and
4. the last 8 bits 11111110 will be converted to 254.

Name your program `IPAddress.java` which contains only one class called `IPAddress`.

Sample runs are shown below, with the input in **bold**:

Sample run #1:

```
$java IPAddress
00000011100000001111111111111110
3.128.255.254
```

Sample run #2:

```
$java IPAddress
11001011100001001110010110000000
203.132.229.128
```

Exercise 2 — Checksum

This exercise will familiarize you how to use Java CRC32 (or Python zlib) class to compute a checksum.

Checksums can be used to detect data corruption and ensure integrity when data is transmitted over a network. You are to write a program that takes in one filename as an command-line argument, and output the CRC-32 checksum of that file.

Java

You may use the `CRC32` class in `java.util.zip` package to compute the CRC-32 checksum. First, you will need to read all the bytes of the file into a byte array. `Paths` and `Files` classes from `java.nio.file` package can be used to do this easily.

Next, invoke the `update()` method of the `CRC32` class with the byte array as a parameter. Finally, calling the `getValue()` method of the `CRC32` class will return the checksum.

For example:

```
byte[] bytes = Files.readAllBytes(Paths.get("doge.jpg"));
CRC32 crc = new CRC32();
crc.update(bytes); // compute checksum
System.out.println(crc.getValue()); // print checksum
```

Python

You may use the `crc32` function in the `zlib` package to compute the CRC-32 checksum. First, you will need to read all the bytes of the file into a buffer. Next, invoke the `zlib.crc32()` function with the buffer as a parameter. The checksum can be passed back into the function to compute the running checksum if you read the file into several buffers.

```
with open("doge.jpg", "rb") as f:
    data = binary_file.read() # Read the whole file at once
    print(zlib.crc32(data))
```

See https://docs.python.org/3/library/zlib.html#zlib.crc32 for details.

Name your program `Checksum.java` which contains only one class `Checksum`. You may assume that the arguments are valid and the file exists.

Sample run:

```
$java Checksum doge.jpg
4052859698
```

Exercise 3 - Copier

This exercise is a refresher on binary file operations.

You are to write a program that makes a copy of an existing file. The program should take in two command-line arguments: `src` and `dest`, and will copy the contents of `src` into a new file `dest`. If no path is given, you may assume the files are in the same directory as the program. Otherwise, you should follow the given path.

To test your program on `sunfire`, you can use the `cmp` command to compare the two files, e.g., "`cmp src dest`". If their contents are identical, nothing will be reported.

Java

There are several Java classes for reading and writing files. The `InputStream` class is an abstract class that represents a sequence of bytes that can be read. `FileInputStream` class is a special type of `InputStream`, which represents a sequence of bytes from a file.

As `FileInputStream` performs raw input from files, it is not efficient to read each byte directly from disk. To make the disk reading more efficient, we may wrap it with the `BufferedInputStream` class. The `BufferedInputStream` class reads a batch of data from the hard disk each time and keeps the data in memory for later use. For example:

```
byte[] buffer = new byte[1024];
FileInputStream fis = new FileInputStream("a.pdf");
BufferedInputStream bis = new BufferedInputStream(fis);
itn numBytes = bis.read(buffer); // returns num of bytes read
```

The counterparts of the classes for writing to files are `OutputSteam`, `FileOutputStream` and `BufferedOutputStream`.

```
FileOutputStream fos = new FileOutputStream("b.pdf");
BufferedOutputStream bos = new BufferedOutputStream(fos);
bos.write("Hello World".getBytes());
```

Remember to close the files at the end of your program.

You can find more details in the Java API documentation.

Python

With Python things are more straightforward. Just use `read()` and `write()` functions on an opened binary file.

Name your program `Copier.java` which contains only one class called `Copier`. You may assume that the input arguments are valid. The program will copy the file, overwriting any existing `dest` and output "`<src> successfully copied to <dest>`" where `<src>` and `<dest>` are the names of the source and destination files.

Remember to close all file variables at the end of your program. Your program should work with both text and binary files, and for small to large files of over hundreds of megabytes.

As this is a practice to read and write bytes, which you will need to do in future assignments, avoid using system utilities like `FileUtil.copyFile`.

Sample run #1:

```
$java Copier doge.jpg puppy.jpg
doge.jpg successfully copied to puppy.jpg
```

Sample run #2:

```
$java Copier ../files/Test.java ./out/puppy.jpg
../files/Test.java successfully copied to ./out/puppy.jpg
```