CS2105 **Assignment 0** Semester 2 AY17/18

Submission Deadline

1 Feb 2018 (Thu), 1pm sharp. 1 point penalty will be imposed on late submission (Late submission refers to submission or re-submission after the deadline).

Objectives

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

This programming assignment is worth 3 marks. All the work in this assignment shall be completed **individually**.

Testing Your Programs

You are free to write your programs on any platform/IDE that you are familiar with. However, you are responsible to ensure that your programs run properly on **sunfire** server because **we will test and grade your programs on sunfire**.

To test your programs, you may log on to **sunfire** using your **SoC UNIX id and password**.

a. If you don't have your SoC UNIX account, please create it here:

https://mysoc.nus.edu.sg/~newacct.

b. If you forget your password, please reset it here:

https://mysoc.nus.edu.sg/~myacct/iforgot.cgi.

c. If you are using a UNIX-based machine (e.g. Mac), SSH should be available from the command line and you may simply type:

```
ssh <SoC UNIX id>@sunfire.comp.nus.edu.sg
```

To copy files to **sunfire**, use **scp**. For example,
```
scp -r <source_directory>
<SoC_UNIX_id>@sunfire.comp.nus.edu.sg: <target_directory>
```

d. If you are using a Window-based machine, you may need to install an SSH client (e.g. "SSH Secure Shell Client", downloadable from IVLE Files -> Assignments folder) if your machine does not already have one installed. You may use the "SSH Secure File Transfer Client" software (bundled with "SSH Secure Shell Client") to upload your programs to the **sunfire** server for testing. A brief guide on how to connect to **sunfire** and upload programs can be found in IVLE Files -> Assignments folder.

For your convenience, we provide a simple test suite in which each exercise has a simple test case. Please copy your programs, the three test cases and the folder named "tests" to **sunfire** and make sure your programs and three **.sh** files are in the same directory. To test your program for exercise 1, run the following command:

```
bash test_ipaddress.sh
```

Take note that passing the given test suite doesn't guarantee that you will get full marks. You are advised to test your solutions thoroughly before submission. You may assume all input data to your programs are valid and hence there is no need for you to perform input data validation.

If you have any question or encounter any problem with the steps above, please post it on IVLE forum or consult the teaching team.

Program Submission

Please submit your programs to **CodeCrunch** website: https://codecrunch.comp.nus.edu.sg. A briefing guide on how to use **CodeCrunch** can be found in IVLE Files -> Assignments folder.

Note that **CodeCrunch** is just used for program submission and no test case has been mounted on it. Therefore you should ignore the automatic feedback from **CodeCrunch** regarding the quality of your programs (e.g. can improve). Just make sure that you have submitted the correct programs to **CodeCrunch**. We will collect your programs and test them on **sunfire** when the deadline is over.

You are not allowed to post your solutions in any public domain in the Internet.

Grading

Each program is worth 1 mark. Your programs will be graded according to their correctness using grading scripts. There is no manual grading. Please make sure your programs follow given sample runs exactly as the grading scripts are unable to award partial marks.

In addition, we will deduct 1 mark for every type of failure to follow instructions (e.g. wrong program name, wrong class name). Please take note of the following common issues:

1. Please use **System.out.println()** to print output on the screen.
2. Please don't have excessive output messages not shown in sample runs.

Plagiarism Warning

You are free to discuss this assignment with your friends. But, ultimately, you should write your own program. We employ zero-tolerance policy against plagiarism. If a suspicious case is found, student will be asked to explain his/her program to the evaluator in face. Confirmed breach may result in zero mark for the assignment and further disciplinary action from the school.

For your information, about 40 plagiarism cases were caught last year.

Question & Answer

If you have any doubts on this assignment, please post your questions on IVLE forum or consult the teaching team. The teaching team is not supposed to debug programs for you. However, we may help clarify misconceptions or give you necessary directions.

Exercise 1

Before you start, you may read the following online article on how to use command-line argument: http://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html.

An IP address is a 32 characters long bit sequence (a bit is either 1 or 0). Read an IP address as a command-line argument and convert it to a dotted decimal format. A dotted decimal format of 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.

To convert binary numbers to decimal numbers, remember that both are positional numerical systems, whereby the first 8 positions of the binary systems are:

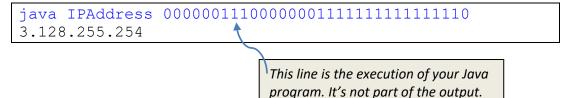| 2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Therefore, 00000011 = (0*128) + (0*64) + (0*32) + (0*16) + (0*8) + (0*4) + (1*2) + (1*1)

$\qquad\qquad\quad$ = 3

Name your program as **IPAddress.java** that contains only one class called **IPAddress**.

Two sample runs are shown below, with user input highlighted in **blue**.

Sample run #1:

```
java IPAddress 00000011100000001111111111111110
3.128.255.254
```

This line is the execution of your Java program. It's not part of the output.

Sample run #2:

```
java IPAddress 11001011100001001110010110000000
203.132.229.128
```

Exercise 2

Checksum can be used to detect if data is corrupted during network transmission (e.g. a bit flips from 0 to 1). Write a program **Checksum.java** that contains only one class called **Checksum** to calculate the checksum for a file **<src>** entered as command-line argument. File **<src>** should be placed in the same folder as **Checksum.java**.

You may use the **CRC32** class in **java.util.zip** package to calculate the CRC-32 checksum. Firstly, you will need to read all the bytes from a file and store them into a byte array. **Paths** and **Files** classes from **java.nio.file** package can help you achieve it easily. Subsequently, invoke the **update()** method of **CRC32** class giving the byte array as a parameter. Finally, call the **getValue()** method of **CRC32** class to obtain the checksum.

An example is shown below.

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

Note: you may encounter a **java.lang.OutOfMemoryError** if you attempt to load a huge file (e.g. hundreds of MBs) into a byte array as above. However, in this exercise we assume the file we process is not that big.

Sample run:

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

Exercise 3

Write a program **Copier.java** that contains only one class called **Copier** to make a copy of an existing file. Your program should take in two command-line arguments: **<src>** and **<dest>**, and then copy the content of **<src>** into the new file **<dest>**. Both **<src>** and **<dest>** should be placed in the same folder as **Copier.java**. You need to ensure **<dest>** has exactly the same content as **<src>**. On **sunfire**, you may use the following command to compare the contents of two files - if their contents are identical, nothing will be reported:

```
cmp <src> <dest>
```

There are several important Java classes for reading and writing files in Java. Let's consider reading first. The **InputStream** class is an abstract class that represents a sequence of bytes that can be read. **FileInputStream** is a special type of **InputStream**, which represents a sequence of bytes from a file.

FileInputStream performs raw input from files. To make the disk reading more efficient, we may wrap it up using **BufferedInputStream**. The **BufferedInputStream** class reads a bunch of data from the hard disk each time and keeps the data in memory for later use. Below is an example code snippet.

```
byte[] buffer = new byte[1024];

FileInputStream fis = new FileInputStream("a.pdf")

BufferedInputStream bis = new BufferedInputStream(fis);

int numBytes = bis.read(buffer); //return num of bytes read
```

For writing, there is an output-counterpart for **InputStream**, called **OutputStream**. Likewise, classes **FileOutputStream** and **BufferedOutputStream** exist and shall be used. The typical code to write something to a file is:

```
FileOutputStream fos = new FileOutputStream("b.pdf");

BufferedOutputStream bos = new BufferedOutputStream(fos);

bos.write("Hello World".getBytes());
```

Remember to close **bis** and **bos** at the end of your program. Your program should work for both text and binary files, and for both small files and large files (hundreds of MBs).

Sample run #1:

```
java Copier doge.jpg mime.jpg
doge.jpg is successfully copied to mime.jpg
```

Sample run #2:

```
java Copier Checksum.java Backup.java
Checksum.java is successfully copied to Backup.java
```