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

Assignment 0
Warm-up

Release date: 17 August 2015
Due: 4 September 2015, 23:59
Total marks: 6

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.

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

To upload your program, you can use an SFTP client like WinSCP to connect to `sunfire` and transfer your files.

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 4 September 2015, 23:59 hrs. Late submission will be penalized accordingly.

Grading

Each program is worth 1 mark and will be graded with an auto-grader. As such, please ensure your programs and classes are named correctly.

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. 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.

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 - Calculate

This exercise is a refresher on command-line arguments and exception handling.

You are to write a program to implement 5 basic arithematic operations: addition (+), subtraction (-), multiplication (*), division (/) and exponent (**).

Your program will read the inputs from 3 command-line arguments in the form `<operand> <operator> <operand>`, where the operands are integers. Examples of valid integers are 23, 0, -23, and invalid integers are 23.5 and 1.0.

If the inputs are valid, the respective operation is carried out and the result is displayed and the program terminates. Otherwise, an error message is displayed (on standard output) instead. The error message to display would be as follows:

- "`Incorrect number of arguments`" should be displayed if the number of arguments is incorrect.
- "`Invalid inputs`" should be displayed if the operands are not integers or the operators are not valid.
- "`Division by zero`" should be displayed if the expression causes a division by zero.

Note that as the grading will be automated, the displayed message should be **exactly** as shown without the surrounding quotes. You may also assume that the operands and the results will be within the range of the `int` type.

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

Sample run #1:

```
$java Calculate 30 - 7
23
```

Sample run #2:

```
$java Calculate 30 / 7
4
```

Sample run #3:

```
$java Calculate 2 ** 4
16
```

Sample run #4:

```
$java Calculate 30 / 0
Division by zero
```

Sample run #5:

```
$java Calculate 2 +
Incorrect number of arguments
```

Note: to test '`java Calculate 30 * 5`' on `sunfire`, you need to put quotes around the `*` like '`java Calculate 30 '*' 5`' as `*` will be interpreted as a UNIX alias.

Exercise 3 - Copier

This exercise is a refresher on binary file operations in Java.

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.

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.

The counterparts of the classes for writing to files are `OutputSteam`, `FileOutputStream` and BufferedOutputStream. You can find more details in the Java API documentation.

Name your program `Copier.java` which contains only one class called `Copier`. You may assume that the input arguments are valid. If no such `src` file is found, the program should output "`File not found`", otherwise it 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.

Sample run #1:

```
$java Copier dog.jpg puppy.jpg
dog.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
```

Sample run #3:

```
$java Copier bad.file good.file
File not found
```

Exercise 4 - Checksum

This exercise will familiarize you how to use Java CRC32 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.

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. 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.

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 dog.jpg
4052859698
```

Exercise 5 - Fields

This exercise is to familiarize yourself with reading a series of input strings from a stream and extracting useful information.

You are to write a program that takes several lines of text from the interactive user-input (i.e., standard input). Each line will be of the format "`<Field>: <Value>`". The series of fields will end with a blank line, thereafter the next few lines of input will be a string. If a string matches one of the `Fields` that was input earlier, the program should print the corresponding `Value` followed by a newline. Otherwise, it should print "`Unknown field`" followed by a newline. Your program should terminate whenever "`quit`" is entered. The headers are not case sensitive but the values should match exactly as entered.

You are to name your program `Fields.java` which contains only one class `Fields`. You may assume that the inputs are all valid. But note that the fields and values can contain spaces.

Sample run:

```
$java Fields
Name:  John Smith
Age:  21 years
Year of matriculation:  2014
CAP: 3.76

Name
John Smith
Year
Unknown field
cap
3.76
year of matriculation
2014
quit
```

Exercise 6 - TimePrinter

This exercise is to familiarize you with Timers and multithread execution.

You are to write a program that takes the following command-line arguments: `<outputString>` `<startTime>` `<interval>`. You program should print `outputString` after `startTime` seconds, and thereafter every `interval` seconds. The program will stop when the character "`q`" is entered by the user.

To schedule a task to be invoked periodically at a later time, you may use the `Timer` and `TimerTask` classes from `java.util` package. The `Timer` class schedules a task to be executed periodically after a certain delay and the `TimerTask` class implements the code to be executed.

Hint: You can extend your class with `TimerTask` and implement the `run()` method. The you can schedule the `run()` method to be executed as follows:

```
Timer timer = new Timer();
timer.schedule(new TimePrinter(...), <start>, <interval>);
```

where `start` and `interval` are in milliseconds. Remember to call `Timer.cancel()` to cancel the task before your program terminates.

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

Sample run:

```
$java TimePrinter Hello 5 10
(5-second pause)
Hello
(10-second pause)
Hello
(10-second pause)
Hello
q
(program terminates)
$
```

Note: the text in brackets are explanatory notes and not part of the output.