SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Computer System Engineering

Week 1: Lab 1 (25 marks)

Objective: Create a Shell Interface using Java or C

This lab exercise is to write a Java or C program to build an user shell interface. The interface accepts user commands and then create new process to execute the inserted command.

A shell interface provides the user with a prompt, after which the user enters the next command. The example below illustrates the prompt *jsh>* for java (*csh>* for C) and the user's next command: cat Prog.java. This command displays the file *Prog.java* on the terminal using the UNIX cat command.

jsh> cat Prog.java
csh> cat Prog.c

Perhaps the easiest technique for implementing a shell interface is to have the program first read what the user enters on the command line (here, *cat Prog.java*) and then create a separate external process that executes the command.

The Java and C program that provides the basic operations of a command-line shell is given in starting code. The *main()* method presents the prompt *jsh>* (for java shell) or *csh>* (for C shell) and waits to read input from the user.

This lab is organized into three questions: (1) creating the external process and executing the command in that process, (2) modifying the shell to allow changing directories, and (3) adding a history feature.

Q1: Creating an External Process (5 marks)
The first part of this project is to modify the main() method of the given program so that an external process is created and executes the command specified by the user. Initially, the command must be parsed into separate parameters and passed to ProcessBuilder object. For example, if the user enters the command

jsh> cat Prog.java

the parameters are (1) cat and (2) Prog.java, and these parameters can be passed through
.command() in ProcessBuilder class. For example:

ProcessBuilder pb = new ProcessBuilder();
pb.command(yourcommand);
Process p = pb.start();

If the user enters an invalid command, the *start()* method in the *ProcessBuilder* class throws an *java.io.IOException*. If this occurs, your program should output an appropriate error message and resume waiting for further commands from the user.

Suggested Function JAVA:

Function	Class which function belongs to
.command()	*ProcessBuilder*
.start()	*ProcessBuilder*
.getMessage()	*Exception*
.getInputStream()	*Process*
.readline()	*BufferedReader*

Test Command JAVA:
    jsh> ls -l
    jsh> mkdir sampleFolder
    jsh> ls
    jsh> wronginput

Suggested Function C:

Function	Use for What	Needed headers
main()	*the core of every program and it is required in each c program*	`#include <stdio.h>` `#include<stdlib.h>`
fget	*To take input from user*	
system	*Execute command*	

Test Command C:
    csh> ls -l
    csh> mkdir sampleFolder
    csh> ls
    csh> wronginput

Test output example:

```
jsh>ls -l
total 392
drwxr-xr-x@ 7 1001369  staff     238 Jan 26 17:59 CSE1
-rw-------@ 1 1001369  staff  179052 Jan 25 14:45 Lab2_Notes_for_teaching.docx
-rw-r--r--@ 1 1001369  staff     828 Jan 26 20:59 SimpleShell.class
-rwxrwxrwx@ 1 1001369  staff     699 Jan  1 12:28 SimpleShell.java
-rw-r--r--@ 1 1001369  staff    3264 Jan 26 21:39 SimpleShell3.class
-rw-r--r--@ 1 1001369  staff    3818 Jan 25 20:01 SimpleShell3.java
drwxr-xr-x@ 2 1001369  staff      68 Jan 26 21:45 sampleFolder
jsh>mkdir sampleFolder
jsh>ls
CSE1
Lab2_Notes_for_teaching.docx
SimpleShell.class
SimpleShell.java
SimpleShell3.class
SimpleShell3.java
sampleFolder
jsh>wronginput
Cannot run program "wronginput" (in directory "/Users/Jie/Dropbox/test"): error=2, No
  such file or directory
jsh>
```

Q2: Changing Directories (10 marks)

The next task is to modify the program in Q1 so that it changes directories. In UNIX systems, we encounter the concept of the *current working directory (pwd)*, which is simply the directory you are currently in. The *cd* command allows a user changing current directory. Your shell interface must support this command. For example, if the current directory is */usr/tom* and the user enters *cd music*, the current directory becomes */usr/tom/music*. Subsequent commands relate to this current directory. For example, entering *ls* will output all the files in */usr/tom/music*.

When the start() method of a subsequent process is invoked, the new process will use this as the current working directory. For example, if one process with a current working directory of */usr/tom* invokes the command *cd music*, subsequent processes must set their working directories to */usr/tom/music* before beginning execution; then if the new directory invokes the command *cd ..* , the working directory will back to its parent directory */usr/tom*. It is important to note that your program first make sure the new path being specified is a valid directory. If not, your program should output an appropriate error message.

If the user enters the command *cd*, change the current working directory to the user's home directory.

For JAVA:

The ProcessBuilder class provides the following method for setting the working directory:

ProcessBuilder pb = new ProcessBuilder();
pb.command(yourcommand);
pb.directory(newDirectory);
Process p = pb.start();

The home directory for the current user can be obtained by invoking *.getProperty()* method in the System class as follows:

System.getProperty("user.home");

Suggested Function:

Function	Class which function belongs to
.directory()	*ProcessBuilder*
.getProperty()	*System*
.isDirectory()	*File*
.getAbsolutePath()	*File*
.getParent()	*File*
.separator	*File*

Test Command:

> *jsh> pwd*
> *jsh> cd ..*
> *jsh> pwd*
> *jsh> cd myfolder*
> *jsh> pwd*
> *jsh> cd unknowfolder*

For C:
The C language provides the following function to change the current working directory:
*int chdir(const char *path);*
Suggested Function:

Function	Use for	Needed headers
strtock	*Parsing the command*	`#include<string.h>`
chdir	*Change directory*	`#include<unistd.h>`

Test Command:
>   *csh> pwd*
>   *csh> cd ..*
>   *csh> pwd*
>   *csh> cd myfolder*
>   *csh> pwd*
>   *csh> cd unknowfolder*

Test output example:

```
jsh>pwd
/Users/Jie/Dropbox/test
jsh>cd ..
jsh>pwd
/Users/Jie/Dropbox
jsh>cd test
jsh>pwd
/Users/Jie/Dropbox/test
jsh>cd unknowfolder
error: new directory is invalid!/Users/Jie/Dropbox/test/unknowfolder
jsh>
```

Q3: Adding a History Feature (10 marks)

Many UNIX shells provide a *history* feature that allows users to see the history of commands they have entered and to rerun a command from that history. The history includes all commands that have been entered by the user since the shell was invoked. For example, if the user entered the history command and saw as output:

0 pwd
1 ls -l
2 cat Prog.java

The history would list pwd as the first command entered, *ls -l* as the second command, and so on. Modify your shell program so that commands are entered into a history.
Your program must allow users to rerun commands from their history by supporting the following three techniques:
1. When the user enters the command history, you will print out the contents of the history of commands that have been entered into the shell, along with the command numbers.
2. When the user enters *!!*, run the previous command in the history. If there is no previous command, output an appropriate error message.
3. When the user enters <integer value *i*>, run the *i*th command in the history. For example, entering *4* would run the fourth command in the command history.

Make sure you perform proper error checking to ensure that the integer value is a valid number in the command history.

Suggested Function JAVA:

Function	Class which function belongs to
.insertElementAt()	*Vector<>*
.addElementAt()	*Vector<>*
.substring	*String*

Test Command:
> *jsh> ls -l*
> *jsh> pwd*
> *jsh> history*
> *jsh> !!*
> *jsh> 1*
> *jsh> 10*

Suggested Function C:

Function	Use for	Needed headers
malloc	*Dynamic memory allocation*	
strcmp	*Compare two strings*	
strcpy	*Copy between strings*	
atoi	*Convert string to integer*	#include<ctype.h>

Test Command:
> *csh> ls -l*
> *csh> pwd*
> *csh> history*
> *csh> !!*
> *csh> 1*
> *csh> 10*

Test output example:

```
jsh>ls -l
total 392
drwxr-xr-x@ 7 1001369  staff      238 Jan 26 17:59 CSE1
-rw-------@ 1 1001369  staff   179052 Jan 25 14:45 Lab2_Notes_for_teaching.docx
-rw-r--r--@ 1 1001369  staff      828 Jan 26 20:59 SimpleShell.class
-rwxrwxrwx@ 1 1001369  staff      699 Jan  1 12:28 SimpleShell.java
-rw-r--r--@ 1 1001369  staff     3265 Jan 26 22:10 SimpleShell3.class
-rw-r--r--@ 1 1001369  staff     3904 Jan 26 22:10 SimpleShell3.java
drwxr-xr-x@ 2 1001369  staff       68 Jan 26 21:45 sampleFolder
jsh>pwd
/Users/Jie/Dropbox/test
jsh>history
0 ls -l
1 pwd
jsh>!!
/Users/Jie/Dropbox/test
jsh>1
/Users/Jie/Dropbox/test
jsh>10
invalid command history search!
jsh>
```

Notes for Java:

- **Rename the "SimpleShell_startingCode.java" as "SimpleShell.java"**
- **How to compile your java code?**

    javac SimpleShell.java

- **How to execute?**

    java SimpleShell

Example:

```
jsh>ls
SimpleShell - starting code.java
SimpleShell1.class
SimpleShell1.java
SimpleShell2.java
SimpleShell3.class
SimpleShell3.java
jsh>ls -l
total 56
-rwxrwxrwx@ 1 1001369  staff   699 Jan  1 12:28 SimpleShell - starting code.java
-rw-r--r--@ 1 1001369  staff  1530 Jan  7 15:07 SimpleShell1.class
-rw-r--r--@ 1 1001369  staff  2332 Jan  1 23:31 SimpleShell1.java
-rw-r--r--@ 1 1001369  staff  2260 Jan  1 23:31 SimpleShell2.java
-rw-r--r--@ 1 1001369  staff  3254 Jan 11 10:52 SimpleShell3.class
-rw-r--r--@ 1 1001369  staff  3921 Jan 11 10:52 SimpleShell3.java
jsh>pwd
/Users/Jie/Dropbox/Phd Courses/TA_CSE/OS_project/Lab1/referenceCode
jsh>history
0 pwd
1 ls -l
2 ls
jsh>1
total 56
-rwxrwxrwx@ 1 1001369  staff   699 Jan  1 12:28 SimpleShell - starting code.java
-rw-r--r--@ 1 1001369  staff  1530 Jan  7 15:07 SimpleShell1.class
-rw-r--r--@ 1 1001369  staff  2332 Jan  1 23:31 SimpleShell1.java
-rw-r--r--@ 1 1001369  staff  2260 Jan  1 23:31 SimpleShell2.java
-rw-r--r--@ 1 1001369  staff  3254 Jan 11 10:52 SimpleShell3.class
-rw-r--r--@ 1 1001369  staff  3921 Jan 11 10:52 SimpleShell3.java
jsh>
```

Notes for C:

- **To build and compile your code using gcc, you need to use the command as the following:**

    gcc yourCode.c –o yourCode

- **Then for the execution:**

    ./yourCode

Example:

```
osboxes@osboxes:~/Desktop/TA$ gcc Q1.c -o Q1
osboxes@osboxes:~/Desktop/TA$ ./Q1
csh>ls
Q1  Q1.c  Q1.c~  QQ1  QQ1.c  QQ1.c~  sampleFolder  Untitled Document
csh>ls -l
total 40
-rwxrwxr-x 1 osboxes osboxes 7740 Jan  9 08:04 Q1
-rw-rw-r-- 1 osboxes osboxes 3470 Jan  9 07:48 Q1.c
-rw-rw-r-- 1 osboxes osboxes 3466 Jan  9 07:27 Q1.c~
-rwxrwxr-x 1 osboxes osboxes 9640 Jan  9 07:05 QQ1
-rw-rw-r-- 1 osboxes osboxes  471 Jan  9 07:05 QQ1.c
-rw-rw-r-- 1 osboxes osboxes  454 Jan  9 07:04 QQ1.c~
drwxrwxr-x 2 osboxes osboxes 4096 Jan  7 21:01 sampleFolder
-rw-rw-r-- 1 osboxes osboxes    0 Jan  7 20:50 Untitled Document
csh>pwd
/home/osboxes/Desktop/TA
csh>history
true
at [0] the command is ls

at [1] the command is ls -l

at [2] the command is pwd

at [3] the command is history

csh>
```

Lab1 submission：

Submit your Java/ C source code (modified from the starting code) which contains all the functions in Q1&Q2&Q3 and a doc/pdf file (illustrate your interface result for in Q1&Q2&Q3) to eDimension before next Lab.