



## **LABORATORY MANUAL**

**CPE102 / CSC102 : Introduction to Programming  
Computing Lab I (Location: N4-B1b-10)**

**SESSION 2009/2010  
SEMESTER 1  
COMPUTER ENGINEERING COURSE  
COMPUTER SCIENCE COURSE**

**SCHOOL OF COMPUTER ENGINEERING  
NANYANG TECHNOLOGICAL UNIVERSITY**

## **OVERVIEW**

### **1. LAB EXPERIMENTS**

The objective of laboratory experiments is to re-enforce your understanding on the course material. There are 8 lab. sessions altogether and the lab. sessions are conducted in the Computing Lab (N4-B1-10) in SCE. The working programming environment runs on the Linux operating system.

- Lab 1 – Linux Environment and My First Java Program
- Lab 2 – Java Fundamentals and Branching
- Lab 3 – Looping
- Lab 4 – Methods
- Lab 5 – Arrays
- Lab 6 – Array of Objects
- Lab 7 – Strings and 2-Dimensional Arrays
- Lab 8 – File Processing and Exception Handling

In doing these experiments, you do not need to submit any lab. report, but you are required to prepare a **Log Book** of your own to record the results of *your own* lab experiments. Here you need to record your own programs and the program outputs in the Log book.

Furthermore, the lab. supervisor may interview you and inspect your **Log book** during the lab. sessions. You may be asked to run the program and explain the purpose of the program code. The objective of this oral interview is to make sure that you do the work on your own. Marks could be deducted if students are found not doing the lab. experiments properly. This **Log book** inspection contributes to your final grade in this course. Therefore, make sure you bring your **Log book** to the lab. sessions.

### **2. LAB ASSIGNMENTS**

In addition to the above 8 lab. experiments and log book, you are also required to submit **two lab. assignments** (see the course webpage for detail). For each lab. assignment, you are required to submit *your own* lab. report. These assignments will also contribute to your final grade in this course.

The marking of your report is generally based on the following criteria:

1. Structure/organization/presentation of your Java codes;
2. Documentation of codes (whether the program is well commented to aid understanding);
3. Correctness of the program;
4. Thoroughness of the testing of the program;
5. Your understanding of the program;
6. User-friendliness of your program (how easy is it to use your program).

To achieve the above, you need to pay attention to:

1. Document your programs - This includes adding comments to your programs at appropriate locations, adding comments to state the purpose of the program, who the author is, and the time the program is written; in addition, you should also add comments on what is the purpose of each class and method.
2. Write reasonably indented codes and blank lines to enhance the readability of your programs.
3. Check all the test cases given to you to test the correctness of your programs.

Detail requirements for each assignment can be found on the website once the assignment is posted.

### **3. COURSE ASSESSMENT**

10% Lab. Experiments and Log book inspection, 40% Lab. Assignments, and 50% Final exam

### **4. RULES OF CONDUCT**

Please be reminded that **PLAGIARISM** (or copying part of/complete assignment) is considered as **CHEATING**, which is strictly prohibited. *Both the copier and code provider* may be **EXPELLED**. Note that lab. experiments and lab. assignments are *individual work*.

## **LAB 1: LINUX ENVIRONMENT AND MY FIRST JAVA PROGRAM**

### **1. OBJECTIVE**

The objectives of Lab 1 are (1) to learn the basic Linux commands; (2) to edit a program and (3) to compile and execute a program. This aims to help students to familiarize themselves with the working programming environment.

### **2. LABORATORY**

Lab 1 is conducted in Computing Lab 1 (N4-B1b-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation running under the *LINUX* environment.

Software: text editor e.g. *xemacs*, *gedit*, *vim* and *vi*, JGRASP

Java 2 Platform

The following diagram shows three basic components, i.e. monitor, mouse and keyboard, of a computer.



**Figure 1**

### **4. INTRODUCTION**

Imagine that you have just written a Java program on paper and want to run it on one of the workstations in the Computing Lab 1. The following steps will be involved.

- (1) Log into the workstation.
- (2) An **editor** will be used to enter and modify your Java program into the computer system.
- (3) A Java **compiler** will be invoked to translate your Java program into a bytecode file. If your program has grammatical errors, the compiler will tell you and you need to go back to step (2) to correct the errors.
- (4) Run the program contained in the bytecode file on your workstation.

In other words, to run a program on the workstation, both computer hardware and software will play their parts. The operating system is a collection of programs which accepts your commands from the

keyboard or the mouse and coordinates the operation of hardware and software to get things done. The operating system we will use is the Red Hat Linux operating system.

In this lab experiment we will go through the steps above and learn through practice.

## 5. EXPERIMENT

This experiment is divided into two parts. The first part is the *LINUX* experience which introduces the basic *LINUX* commands. The second part consists of editing, compiling and executing of a simple Java program.

### 5.1 Basic Linux Commands

In this section, we will first learn how to log onto the workstation. On every workstation in the Computing Lab 1, the screen should display a dialog box similar to that in *Figure 2*.

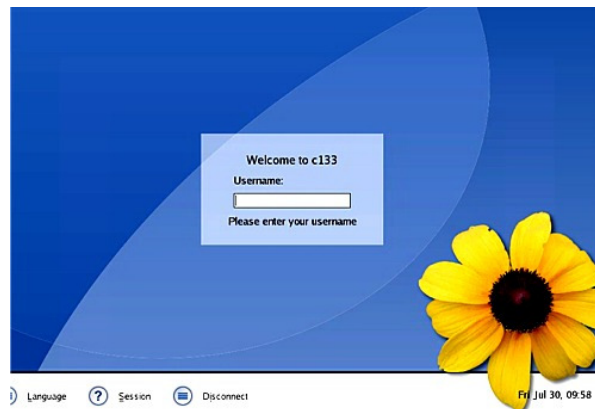


Figure 2

This dialog box is the gate through which you become authorized to use your workstation (hence, you need an account!). Your *Username* and *Login Password* will be issued during your first lab session in Computing Lab 1. Then you can settle down at your workstation, enter your *username* and *password* to login. You will see the Linux desktop as shown in *Figure 3* if you can log in successfully.

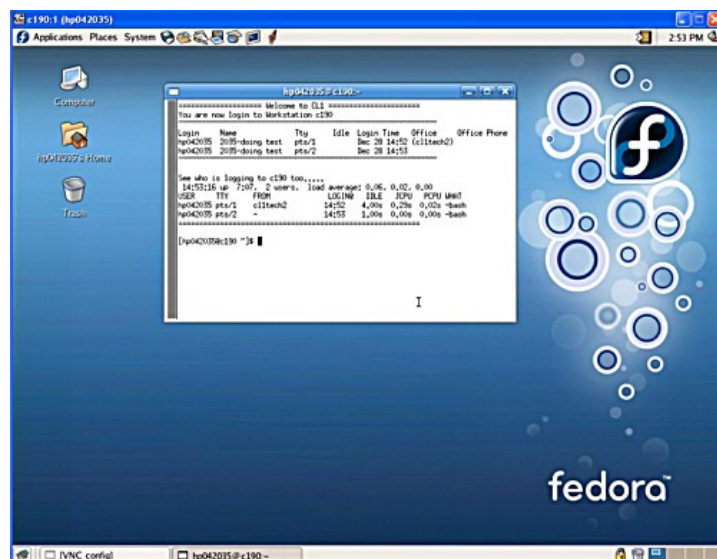
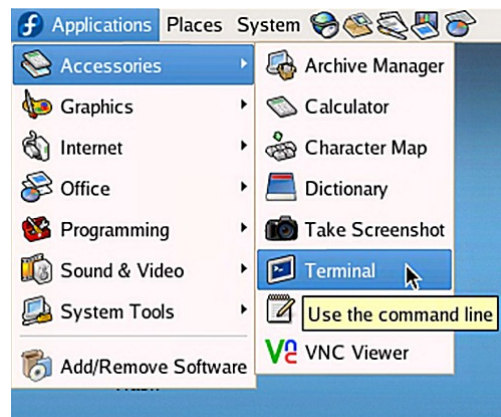


Figure 3

**To open a terminal window**

On the desktop, click on *Applications*, *Accessories* and follow by *Terminal* (in *Figure 4*) to open a terminal window.



**Figure 4**

The terminal window is a window for you to access the shell prompt. When you open a new terminal window, you will see a \$ sign at the top left corner of the window area. This \$ sign is the system prompt to the user (i.e. you) for a command. You need this window to execute Linux commands for opening application programs, and compile and run the programs.

For new account, all students within the same lab group are issued with a common password. Hence, you must change your login password by the end of your first lab session :

**Step 1:** You must confirm you are the rightful owner for the username issued to you, using command *finger*. Your name will be shown in the user information.

**Step 2:** Then change your login password using command *passwd*.

The new password must be more than 8-characters, with at least one non-word character (!@#%&^\*[( ){}). (Note: the cursor will not move when you entering the password.)

*Figure 5* gives an example of confirming your user information and changing your password successfully.

```
[hp042020@c133 hp042020]$ finger hp042020
Login: hp042020
Directory: /home/user12/hp042020
Never logged in.
No mail.
No Plan.
```

**Step 1 = Confirm you are the rightful owner**

```
[hp042020@c133 hp042020]$ passwd
Changing password for user hp042020.
Changing password for hp042020
(current) UNIX password:
New password:
Retype new password:
```

**Step 2 = Change your login password**

```
The password has been changed on cl1_s2.
passwd: all authentication tokens updated successfully.
[hp042020@c133 hp042020]$
```

**Figure 5**

## Linux File System

A file system is a collection of files. A file is usually stored on a disk. Files can be grouped into directories and organized into a hierarchical directory structure. The top of this hierarchical directory structure is called the root directory. The *home directory* is the directory assigned to you. The *path name* of a file enables you to identify a file uniquely to the Unix file system.

### Examples of Commands that Work with Files

To list the files in a directory: \$ ls (\$ - command prompt)  
 To examine the contents of a file: \$ cat file1 or \$ more file1  
 To make a copy of a file: \$cp file1 file1.backup  
 To rename a file: \$mv file1 newFile1  
 To remove a file: \$rm rubbish

### Examples of Commands that Work with Directories

To display your working directory: \$pwd  
 To create a directory: \$mkdir lab1  
 To change directory:  
     \$cd lab1  
     \$cd /home/user1/hp123456/lab2  
     \$cd ..  
 To copy a file from one directory to another: \$cp program.java lab1  
 To move a file from one directory to another: \$mv program.java lab1  
 To remove a directory: \$rmdir dir1

## Learning Linux Commands

In the terminal window, you may type any commands some of which are described below. When the system has finished the work requested by a command, the \$ sign will appear at the beginning of the next line and you may enter the next command.

To learn the Linux commands one can do a

**man <command> <RET>**

to find out what each of the command does. *<command>* stands for any Linux commands and *<RET>* stands for the ENTER or RETURN key on the keyboard. For example, **man cp <RET>** will have the computer displaying what the command **cp** will do and the format of using this command. Table 1 gives a basic set of commands and those in bold are perhaps the most frequently used ones. You are asked to refrain from overloading the printer for printing these manual pages **unless absolutely** necessary. The *<RET>* is used to mark the end of the current command and place the cursor to the first position on the next line. If a command is not ended by the *<RET>*, the command will not be executed.

To get started, you may do the following

1. Find out (by doing **man <command>**) what each of the bold faced command in Table 1 can do.
2. Try out some of the options, e.g. *-c*, *-C*, *-b*, *-r*, *-cb*, ..., etc., that go with each command. Illustrate with examples (from your try-out) of how useful some of the options are. Report your finding with at least two options.

<b>alias</b>						
cat	<b>cd</b>	chmod	clear	compress	<b>cp</b>	
date						

find	finger	ftp				
grep						
help	history					
kill						
ln	login	lp	ls	logout		
man	mkdir	more	mv			
passwd	ps	pwd				
rm	rmdir					
script	scp	sftp	ssh			
talk	time	tar	tee	telnet		
uptime	uname					
vi	vim	vncserver	vncviewer			
w	write	whereis	who	whoami		
zip						

Table 1

You may capture what appears on the monitor screen by using the command *script*. Use the command ***man script*** to find out how to do it. You will need this command to capture the tests you do on your programs and include them in your submissions for the assignments.

In Computing Lab 1, you need to create the proper directories to store your lab work. The lab staff may need to transfer your data to the account of lab supervisor, for grading purposes. All students must follow the guidelines below to create the directory in their home directories:

1. Create a subject directory in the home directory. For this subject, the subject directory should be cpe102 or csc102.
2. Create sub-directories in the subject directory to store individual lab works e.g. lab1, lab2, lab3, lab4 and so on.
3. All directory naming must be in lower case, with no space or non-word characters in between. The Linux system is case-sensitive, thus, naming such as 'LAB1', 'Lab1', 'LAb1', 'lab 1', 'lab\_1', 'lab-1' and 'lab#1' are not acceptable.

As an exercise, use the *script* command to capture the information on the monitor screen and fill in the following Table 2:

Step	Commands you type in	Write down what you see
1	cd	<i>Change the current directory path to your home directory.</i>
2	script -a TT	
3	pwd	
4	ls	
5	mkdir <b>xxx102</b>	( depending on your Subject Code, change <b>xxx102</b> to <b>cpe102</b> or <b>csc102</b> )
6	ll	
7	cd <b>xxx102</b>	
8	mkdir <b>lab0</b>	
9	ll	
10	cd <b>lab0</b>	

11	pwd	
12	touch myprogram	
13	ls -l	
14	rm myprogram	
15	ll	
16	cd ..	
17	cd	
18	ls -a	
19	(press <i>Ctrl-D</i> to exit from script command)	
20	ll -a	
21	more TT	

Table 2

## 5.2 My First Java Program

In this lab, one is going to write his/her first Java program by typing in a sample Java program *MyFirstProgram.java*. First make sure your current directory is lab1. Go through the following steps to complete this section.

### 5.2.1 Create a Java Source File

In this lab, you will write your first Java program using the text editor available in the Linux system. You can access the *Text Editor* by clicking on the *Applications*, followed by *Accessories* as shown in *Figure 6*.

Type in the following program *MyFirstProgram.java* in the editor window and then save the program in your home directory *.../cpe102/lab1/*. Note that the program must be saved as *MyFirstProgram.java*.

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello! This is my first program.");
        System.out.println("Bye Bye!")
    }
}
```



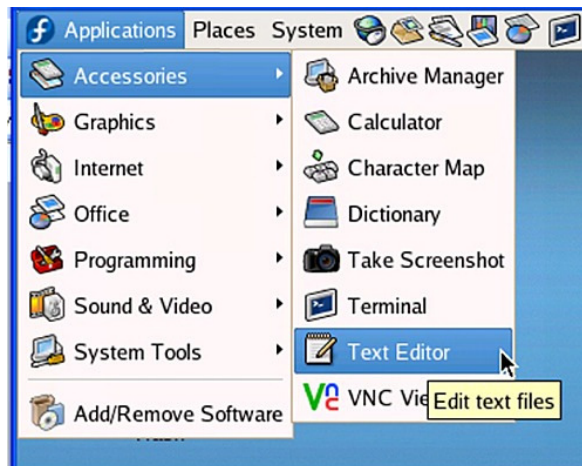


Figure 6

### 5.2.2 Compile the Source File

For compiling and executing the source file, you need to customize your system so that you are able to compile and execute the command without typing the full path. You will get the error message “*command not found*” if you fail to do so.

You need *Java 2 Platform (Standard Edition)* to compile and execute your java program. You have to find out the actual path where the above software is located in the workstation.

In Computing Lab 1, the software is stored in `/opt` as shown in Figure 7.

Next, use the text editor to open your system file `.bash_profile` and input the path

`/opt/jdk/bin`

as shown in Figure 8 then save and close the file.

For the change to take effect, type the following command in the terminal window:

`source .bash_profile`

Or logout from your account and log in again.

```

hp042035@c135:/opt
File Edit View Terminal Tabs Help
[hp042035@c135 ~]$ cd /opt
[hp042035@c135 opt]$ pwd
/opt
[hp042035@c135 opt]$ ll
total 60
drwxr-xr-x  2 1000 users 4096 Aug  9 01:28 install_flash_player_7_linux
drwxr-xr-x  7 root root 4096 Dec 13 14:03 javadocs
lrwxrwxrwx  1 root root   11 Dec 13 13:16 jdk -> jdk1.5.0_09
drwxr-xr-x  9 root root 4096 Oct 13 04:01 jdk1.5.0_09
drwxr-xr-x 10 root root 4096 Dec 13 14:03 jgrasp-1.8.4
drwxr-xr-x 10 root root 4096 Dec 13 14:03 jgrasp-1.8.5.b2
drwxr-xr-x 10 root root 4096 Dec 13 13:20 netbeans-5.0
drwxr-xr-x 10 root root 4096 Dec 13 13:23 netbeans-5.5
[hp042035@c135 opt]$

```

Figure 7

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

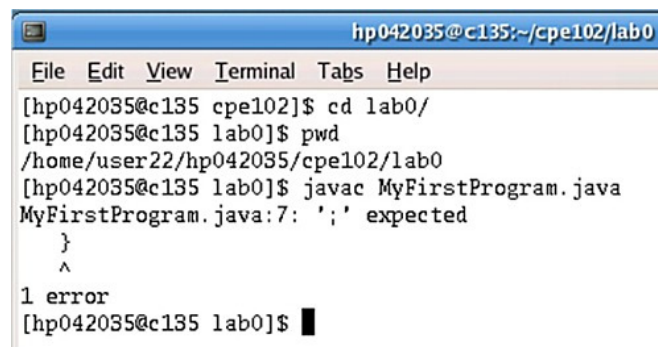
PATH=$PATH:$HOME/bin:/opt/jdk/bin:/usr/share:$HOME/seti;

export PATH
unset USERNAME
```

Figure 8

You may now compile the program. Compile the program using the following command:

```
javac MyFirstProgram.java
```



The screenshot shows a terminal window titled 'hp042035@c135:~/cpe102/lab0'. The user has navigated to the 'lab0' directory and attempted to compile 'MyFirstProgram.java' using the 'javac' command. The compiler reports an error: 'MyFirstProgram.java:7: ';' expected', pointing to a closing curly brace '}' on line 7. The error message indicates that a semicolon is missing at the end of the line.

```
hp042035@c135:~/cpe102/lab0
File Edit View Terminal Tabs Help
[hp042035@c135 cpe102]$ cd lab0/
[hp042035@c135 lab0]$ pwd
/home/user22/hp042035/cpe102/lab0
[hp042035@c135 lab0]$ javac MyFirstProgram.java
MyFirstProgram.java:7: ';' expected
    }
    ^
1 error
[hp042035@c135 lab0]$
```

Figure 9

The compiler will generate some error messages (in Figure 9). This is because of the grammatical error that ';' is missing at the end of the line

```
System.out.println("Bye Bye!")
```

Use the editor to add the ';' in and save the corrected version of the program. Then compile the program again and make sure that no more error messages are given out.

### 5.2.3 Run the Program

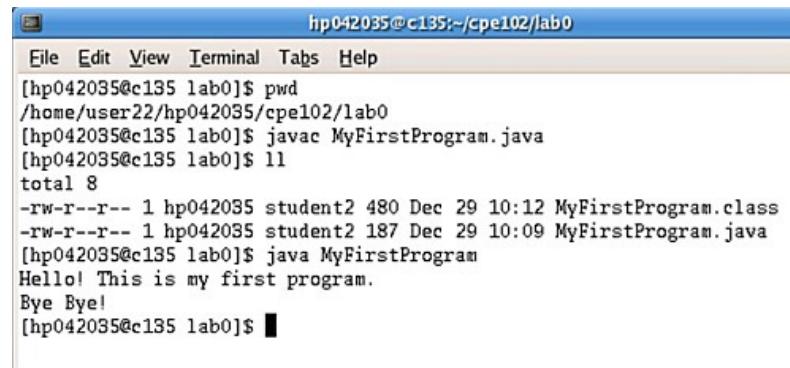
In the same directory, type the following command:

```
java MyFirstProgram
```

Now, you should see

```
Hello! This is my first program.
Bye Bye!
```

are displayed on the screen as shown in Figure 10.



```

hp042035@c135:~/cpe102/lab0
File Edit View Terminal Tabs Help
[hp042035@c135 lab0]$ pwd
/home/user22/hp042035/cpe102/lab0
[hp042035@c135 lab0]$ javac MyFirstProgram.java
[hp042035@c135 lab0]$ ll
total 8
-rw-r--r-- 1 hp042035 student2 480 Dec 29 10:12 MyFirstProgram.class
-rw-r--r-- 1 hp042035 student2 187 Dec 29 10:09 MyFirstProgram.java
[hp042035@c135 lab0]$ java MyFirstProgram
Hello! This is my first program.
Bye Bye!
[hp042035@c135 lab0]$

```

Figure 10

## 6. JGRASP - INTEGRATED DEVELOPMENT ENVIRONMENT

For those students who would like to use an integrated development environment (IDE) for developing their programs, you may try to use the JGRASP integrated development environment for developing MyFirstProgram. The JGRASP IDE has been installed on your PC.

In the workstation, you can run *JGrasp* in

- ❖ The Terminal Window - by typing  
`./opt/jgrasp-1.8.4/bin/jgrasp`
- ❖ The Desktop – access via the launcher (shortcut). On the desktop, click the right mouse button to open a pull-down menu as shown in *Figure 11* to create the launcher. In the window as shown in *Figure 12*, enter the name, command and then click on *No Icon* to select an icon for your launcher.

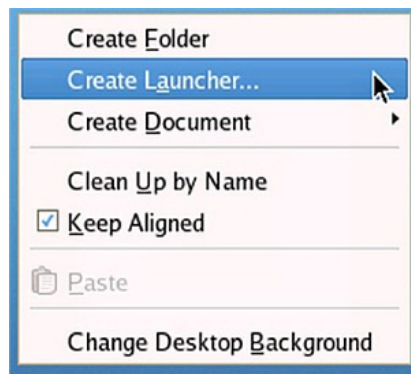


Figure 11



Figure 12

## 7. REPORT

Write a summary and record your results into your log book. There is no submission for this lab experiment.

## 8. REFERENCES

- [1] Computing Lab 1 Student FAQ. Available at <http://www.ntu.edu.sg/sce/labs/cpl/faqmain.htm>.
- [2] The textbook for CPE102/CSC102.
- [3] Any Linux operating system reference books.

- [4] Linux Online. Available at <http://www.linux.org/>.
- [5] Linux faq (frequently asked questions). Available at <http://www.linux.org/info/faq1.html#faq.q1>.
- [6] Tutorials for JGRASP Integrated Development Environment. Available at <http://www.jGRASP.com/>.

## **LAB 2: JAVA FUNDAMENTALS AND BRANCHING**

### **1. OBJECTIVE**

The objectives of Lab 2 are (1) to know the basic data types and operators; (2) to learn how to use the branching control structures.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-b1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP

Java 2 Platform

### **4. INTRODUCTION**

Java has a rich set of data types and operators. The best way to learn them is through manipulating different data types using various operators in programs. We will also practise on the branching control structure in this Lab exercise.

### **5. EXPERIMENT**

First create a sub-directory called *lab2* in your home directory. Do all the programming in this *lab2* sub-directory and leave all the programs in this *lab2*. Use the test cases suggested to test your program and understand why these test cases are used. You do not need to do (extra) error checking on input.

- 5.1** Write a program that prints the quadrant number of a point (x,y) on a plane. Recall that points in quadrant 1 have positive x and y values, points in quadrant 2 have a negative x value and a positive y value, points in quadrant 3 have negative x and y values, and the remaining points are in the quadrant 4. If a point is on an axis, choose the quadrant with the lower quadrant number.

**Important:** Remember to name the source code of this program as **P1.java** and name the compiled class code as **P1.class** (this is automatically done after compilation) inside the sub-directory *lab2*.

**Test cases:** 4 points : each from one of the four quadrants, e.g., (1,1), (-1, 1), (-1, -1), (1, -1).

**Expected outputs:** (1,1) – quadrant 1; (-1, 1) – quadrant 2; (-1, -1) – quadrant 3; (1, -1) – quadrant 4.

- 5.2** Write a program that reads a character from the user and then uses a *switch* statement to achieve what the following *if* statement does.

```
if ((choice == 'A') || (choice == 'a'))
    printf("Action movie fan\n");
else if ((choice == 'C') || (choice == 'c'))
    printf("Comedy movie fan\n");
else if ((choice == 'D') || (choice == 'd'))
    printf("Drama movie fan\n");
else
    printf("Invalid choice\n");
```

**Important:** Remember to name the source code of this program as **P2.java** and name the compiled class code as **P2.class** inside the sub-directory *lab2*.

**Test cases:** 'a', 'A', 'c', 'C', 'd', 'D', 'b', 'B'.

**Expected outputs:** 'a', 'A' – Action movie fan; 'c', 'C' – Comedy movie fan; 'd', 'D' – Drame movie fan; 'b', 'B' – Invalid choice.

**5.3** The salary scheme for a company is given as follows:

Salary range for grade A: \$700 - \$899

Salary range for grade B: \$600 - \$799

Salary range for grade C: \$500 - \$649

A person whose salary is between \$600 and \$649 is in grade C if his merit points are below 10, otherwise he is in grade B. A person whose salary is between \$700 and \$799 is in grade B if his merit points are below 20, otherwise, he is in grade A. Write a program to read in a person's salary and his merit points, and displays his grade.

**Important:** Remember to name the source code of this program as **P3.java** and name the compiled class code as **P3.class** inside the sub-directory *lab2*.

**Test cases:** (1) salary : \$500, merit : 10; (2) salaray : \$610, merit : 5; (3) salary : \$610, merit : 10; (4) salary : \$710, merit : 15; (5) salary : \$710, merit : 20; (6) salary : 800, merit : 30.

**Expected outputs:** (1) salary : \$500, merit : 10 – Grade C; (2) salaray : \$610, merit : 5 – Grade C; (3) salary : \$610, merit : 10 – Grade B; (4) salary : \$710, merit : 15 – Grade B; (5) salary : \$710, merit : 20 – Grade A; (6) salary : 800, merit : 30 – Grade A.

## **6. REPORT**

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## **7. REFERENCES**

[1] The textbook for CPE102/CSC102.

## **LAB 3: LOOPING**

### **1. OBJECTIVE**

The objectives of Lab 3 are (1) to know the basic data types and operators; (2) to learn how to use the branching and looping control structures.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-b1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP

Java 2 Platform

### **4. INTRODUCTION**

Java has a rich set of data types and operators. The best way to learn them is through manipulating different data types using various operators in programs. We will also practise on the branching and looping control structures in this Lab exercise.

### **5. EXPERIMENT**

First create a sub-directory called *lab3* in your home directory. Do all the programming in this *lab3* sub-directory and leave all the programs in this *lab3*. Use the test cases suggested to test your program and understand why these test cases are used. You do not need to do (extra) error checking on input.

- 5.1** Write a program to read in a character repeatedly (using a looping construct) from the user, and for each input character print a message to say whether the character is a vowel, a digit or neither. You may define any character as the sentinel for terminating the loop (e.g. '#').

**Important:** Remember to name the source code of this program as **P1.java** and name the compiled class code as **P1.class** inside the sub-directory *lab3*.

**Test cases:** a, d, 4, !, #

**Expected outputs:** a – vowel; d – neither; 4 – digit; ! – neither; # – program terminated.

- 5.2** Write a program to generate tables of currency conversions from Singapore dollars to US dollars. Use title and column headings. Assume the following conversion rate:

$$1 \text{ US dollar(US\$)} = 1.82 \text{ Singapore dollars (S\$)}$$

Allow the user to enter the starting value, ending value and the increment between lines in S\$. The starting value, ending value and the increment are all integer values. Generate three output tables using the following loops with the same input data from the user:

1. Use a *for* loop to generate the first table;
2. Use a *while* loop to generate the second table; and
3. Use a *do/while* loop to generate the third table.

Place all the codes in the main() method.

**Important:** Remember to name the source code of this program as **P2.java** and name the compiled class code as **P2.class** inside the sub-directory *lab3*.

**Test cases:** (1) starting : 1, ending : 5, increment : 1; (2) starting : 0, ending : 40, increment: 5; (3) starting : 40, ending : 0, increment: 5 (treat this case as an error).

**Expected outputs:**

(1) starting : 1, ending : 5, increment : 1;

S\$	US\$
1	1.82
2	3.64
3	5.46
4	7.28
5	9.1

(2) starting : 0, ending : 40, increment: 5;

S\$	US\$
0	0.0
5	9.1
10	18.2
15	27.3
20	36.4
25	45.5
30	54.6
35	63.7
40	72.8

(3) starting : 40, ending : 0, increment: 5 (treat this case as an error) – Error input!!

**5.3** Write a program that reads the length of a square and a character. It then displays at the left margin of the screen a solid square whose side has the specified length and inside filled with the character. For example, if the user enters 4 for length and '\*' for the character, the following solid square is displayed:

```
****
****
****
****
```

**Important:** Remember to name the source code of this program as **P3.java** and name the compiled class code as **P3.class** inside the sub-directory *lab3*.

**Test cases:** use 0, 2, 4 as the side lengths and at least two different characters in the three tests. For example, use '+' when the side length is 2 and use '\*' when the side length is 4.

**Expected outputs:** (1) length = 0 – Error input!!

(2) length = 2

```
++
++
```

(3) length = 4 – same as the sample square output

**5.4** Write a program that reads the height from a user and prints a pattern with the specified height. For example, when the user enters height = 3, the following pattern is printed:

```
AA
BBAA
AABBAA
```



If the height is 7, then the following pattern is printed:

```
AA
BBAA
AABBAA
BBAABBAA
AABBAABBAA
BBAABBAABBAA
AABBAABBAABBAA
```

**Important:** Remember to name the source code of this program as **P4.java** and name the compiled class code as **P4.class** inside the sub-directory *lab3*.

**Test cases:** 0, 3, 7

**Expected outputs:** (1) height = 0 – Error input!! (2) height = 3 & (3) height = 7 – same as the sample patterns.

- 5.5 Write a program that reads a value x (with data type double), and prints the result of the following series with the input value x:

$$x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots - \frac{x^{19}}{19}$$

**Important:** Remember to name the source code of this program as **P5.java** and name the compiled class code as **P5.class** inside the sub-directory *lab3*.

**Test cases:** 0.9, 0, -0.9

**Expected outputs:** x = 0.9 – result = 0.7298154560111497; x = 0 – result = 0;  
x = -0.9 – result = -0.7298154560111497.

## 6. REPORT

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## 7. REFERENCES

- [1] The textbook for CPE102/CSC102.

## **LAB 4: METHODS**

### **1. OBJECTIVE**

The objectives of Lab 4 are to practise on Java methods.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-b1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP  
Java 2 Platform

### **4. INTRODUCTION**

A method is an independent collection of source code designed to perform a specific task. By dividing a problem into sub\_problems and solve the sub\_problems by methods, we obtain a program which is better structured, easier to test, debug and modify. We will practise on writing methods in Java in this lab.

### **5. EXPERIMENT**

First create a sub-directory, *lab4*, under your home directory. All the programming for this lab experiment will be done in this *lab4* directory. Put the source code into the file **Lab4p.java**, and generate the compiled class code as **Lab4p.class**.

You may use the program template in Figure 1 to test your methods developed in this lab. The program contains a **main()** which includes a switch statement so that the following methods can be tested by the user. Write the code for each method and use the suggested test cases to test your code for correctness.

```
import java.util.Scanner;
public class Lab4p {
    public static void main(String[] args)
    {
        int choice;
        Scanner sc = new Scanner(System.in);
        do {
            System.out.println("Perform the following methods:");
            System.out.println("1: multiplication test");
            System.out.println("2: quotient using division by subtraction");
            System.out.println("3: remainder using division by subtraction");
            System.out.println("4: count the number of digits");
            System.out.println("5: position of a digit");
            System.out.println("6: extract all odd digits");
            System.out.println("7: quit");
            choice = sc.nextInt();

            switch (choice) {
```

```

        case 1: /* add mulTest() call */
            break;
        case 2: /* add divide() call */
            break;
        case 3: /* add modulus() call */
            break;
        case 4: /* add countDigits() call */
            break;
        case 5: /* add position() call */
            break;
        case 6: /* add extractOddDigits() call */
            break;
        case 7: System.out.println("Program terminating ....");
    }
} while (choice < 7);
}

/* add method code here */

}

```

Figure 1: Program template for Lab 4.

- 5.1** Write a method that is to test students ability to do multiplication. The method will ask a student 5 multiplication questions one by one and checks the answers. The method prints out the number of correct answers given by the student. The method *random()* from the *Math* class of the Java library can be used to produce two positive one-digit integers (i.e. 1,2,3,4, ...) in each question. A sample screen display when the method is called is given below:

```

How much is 6 times 7? 42
How much is 2 times 9? 18
How much is 9 times 4? 36
.....
4 answers out of 5 are correct.

```

The input which is underlined is the student's answer to a question. The method header is:

```
public static void mulTest()
```

**Test cases:** (1) give 5 wrong answers; (2) give 1 correct answer; (3) give more than 1 correct answer.

**Expected outputs:** straightforward.

- 5.2** Write the method *divide()* which does division by subtraction and returns the quotient of dividing *m* by *n*. Both *m* and *n* are positive integers (i.e. 1,2,3,4,...). Division by subtraction means that the division operation is achieved using the subtraction method. For example, *divide(12,4)* will be performed as follows: 12-4=8, 8-4=4, and then 4-4=0, and it ends and returns the result of 3 as it performs three times in the subtraction operation. No error checking on the parameters is required in the method. The method header is given below:

```
public static int divide(int m, int n)
```

**Test cases:** (1) *m* = 4, *n* = 7; (2) *m* = 7, *n* = 7; (3) *m* = 25, *n* = 7.

**Expected outputs:** (1) 4/7 = 0; (2) 7/7 = 1; (3) 25/7 = 3.

- 5.3** Write the method `modulus()` which does division by subtraction and returns the remainder of dividing `m` by `n`. Both `m` and `n` are positive integers. No error checking on the parameters is required in the method. The method header is given below.

```
public static int modulus(int m, int n)
```

**Test cases:** (1) `m = 4, n = 7` (2) `m = 7, n = 7` (3) `m = 25, n = 7`.

**Expected outputs:** `4 % 7 = 4`; (2) `7 % 7 = 0`; (3) `25 % 7 = 4`.

- 5.4** Write a method to count the number of digits for a positive integer (i.e. 1,2,3,4,...). For example, 1234 has 4 digits. The method `countDigits()` returns the result. The method header is given below:

```
public static int countDigits(int n)
```

**Test cases:** (1) `n : -12` (give an error message); (2) `n : 123`; (3) `n : 121456`;

**Expected outputs:** (1) `n : -12 - Error input!!` (2) `n : 123 - count = 3`; (3) `n : 121456 - count = 6`.

- 5.5** Write the method `position()` which returns the position of the first appearance of a specified digit in a positive number `n`. The position of the digit is counted from the right and starts from 1. If the required digit is not in the number, the method should return -1. For example, `position(12315, 1)` returns 2 and `position(12, 3)` returns -1. No error checking on the parameters is required in the method. The method header is given below:

```
public static int position(int n, int digit)
```

**Test cases:** (1) `n : 12345, digit : 3`; (2) `n : 123, digit : 4`; (3) `n : 12145, digit : 1`;

**Expected outputs:** (1) `position = 3`; (2) `position = -1`; (3) `position = 3`.

- 5.6** Write a method `extractOddDigits()` which extracts the odd digits from a positive number `n`, and combines the odd digits sequentially into a new number. The new number is returned back to the calling method. If the input number `n` does not contain any odd digits, then returns -1. For examples, if `n=1234567`, then 1357 is returned; and if `n=28`, then -1 is returned. The method header is given below:

```
public static long extractOddDigits(long n)
```

**Test cases:** (1) `n : 12345`; (2) `n : 54123`; (3) `n : 246`; (4) `n : -12` (give an error message)

**Expected outputs:** (1) `oddDigits = 135`; (2) `oddDigits = 513`; (3) `oddDigits = -1`; (4) `oddDigits = Error input!!`

## 6. REPORT

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## 7. REFERENCES

[1] The textbook for CPE102/CSC102.

## **LAB 5: ARRAYS**

### **1. OBJECTIVE**

The objective of Lab 5 is to get familiar with one dimensional arrays.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-B1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP

Java 2 Platform

### **4. INTRODUCTION**

An array is composed of a series of elements of the same data type. Arrays are often used to store data needed for a program. To access elements in an array, an individual element's subscript number is used. We will practise on writing arrays in Java in this lab.

### **5. EXPERIMENT**

First create a sub-directory called *lab5* in your home directory. Do all the programming in this *lab5* sub-directory.

The following methods work on an array which can be used to store up to 11 integers. From the second element to the last element of the array, they are used to store the integer data. The first element of the array is used to store the number of integer data that are stored in the array. Therefore, the array should be created to contain at most 11 elements. The array structure is shown below:

[0]	Number of data stored
[1]	int data
[2]	int data
[3]	int data
[4]	...
[5]	...
[6]	...
[7]	...
[8]	...
[9]	...
[10]	int data

Write the code for the following methods and then a main() method to test them. In the main() method, a menu that can support the following methods should be given: (1) initialize; (2) insert; (3) remove; (4) display; (5) quit. Then, the user selects an option from the menu. After the user has selected an option, the corresponding method will then be executed. If option (5) is not selected, then the menu will be repeated, and the user can select another option for execution.

```

import java.util.Scanner;
public class Lab5p
{
    public static void main(String[] args)
    {
        int choice;
        int[ ] array = new int[11];
        // create a Scanner object here

        do {
            System.out.println("Perform the following methods:");
            System.out.println("1: initialize");
            System.out.println("2: insert");
            System.out.println("3: remove");
            System.out.println("4: display");
            System.out.println("5: quit");

            // read user input
            switch (choice)
            {
                case 1: /* add initialize(array) call */
                    break;
                case 2: /* add insert() call */
                    break;
                case 3: /* add remove() call */
                    break;
                case 4: /* add display() call */
                    break;
                case 5: System.out.println("Program terminating ...");
            }
        } while (choice != 5);
    }

    /* add method code */
    ...
}

```

Figure 1: Program template for Lab 5.

The methods are defined as follows:

**public static void initialize(int[] ar)**

/\* read in up to 10 integers and USE **insert()** to store them in *ar*. The method reads in user input data, and stores them into the array *ar*. The method will first read in the number of integers to be initialized from the user, then it will read in the input integers into *ar*. *ar* should be sorted and stored as an array of integers in ascending order. **Recall that the first element of the array *ar* is used to store the number of integers stored in the array.** \*/

**public static void insert(int[] ar, int n)**

/\* insert the number *n* into the array *ar*. Before and after the method call, *ar* is an array of integers in ascending order. In addition, the updated value on the number of integers stored in *ar* should be updated into the first element of *ar*. The method should issue an error message and no insertion will be done if the number of integers stored in *ar* is equal to 10 before insertion. \*/

**public static void remove(int[] ar, int n)**

```
/* remove the first appearance of the number n from the array ar. Before and after the method call, ar
is an array of integers in ascending order. If the number of integers stored in ar is equal to zero or n
does not appear in ar, the method should issue an error message. */
```

```
public static void display( int[] ar)
```

```
/* print the numbers stored in ar. Recall that the first element of the array ar is used to record the
number of integers stored in the array. */
```

**Important:** Remember to name the source code of this program as **Lab5p.java** and name the compiled class code as **Lab5p.class** inside the sub-directory *lab5*.

**Test cases:**

- (a) **Insert:** (1) when the array is not full: insert a number which is smallest (e.g. 3); (2) insert a number which is the largest (e.g. 15); (3) insert a number which is neither the smallest nor the largest (e.g. 8); and (4) insert when the array is already full (i.e. display an error message). Display the contents in the array after each operation.
- (b) **Remove:** (1) when the number to be removed is not in the array; (2) when the number to be removed is in the array: when it is the first one in the array, last one in the array, somewhere in the middle. Display the contents in the array after each operation.

**Expected outputs:** (a) Insert – quite straightforward, and display an error message when the array is full. (b) Remove – also quite straightforward, and display an error message when the specified number is not in the array. Remember to use the display() method to display the contents in the array after each operation.

## 6. REPORT

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## 7. REFERENCES

- [1] The textbook for CPE102/CSC102.

## **LAB 6: ARRAY OF OBJECTS**

### **1. OBJECTIVE**

The objective of Lab 6 is to practise on processing array of objects.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-B1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP  
Java 2 Platform

### **4. INTRODUCTION**

Very often a program needs to process information in an array of objects in several ways. We will practise on writing array of objects in this lab.

### **5. EXPERIMENT**

The Colossus Airlines fleet consists of one plane with a seating capacity of 12. It makes one flight daily. In this experiment, you are required to write a seating reservation application program. The problem specification is given below:

- A. Write a class *PlaneSeat* that has the following features. Each *PlaneSeat* object should hold a seat identification number (*seatId*), a marker (*assigned*) that indicates whether the seat is assigned and the customer number (*customerId*) of the seat holder. The class diagram is given below:

PlaneSeat
- seatId: int - assigned: boolean - customerId: int
+ PlaneSeat(seat_id: int) + getSeatID(): int + getCustomerID(): int + isOccupied(): boolean + assign(cust_id: int): void + unAssign(): void

where

PlaneSeat() - is the constructor for the class.

getSeatID() – a get method that returns the seat number.

getCustomerID() – a get method that returns the customer number.

isOccupied() – a method that returns a boolean on whether the seat is occupied.

assign() – a method that assigns a seat to a customer.

unAssign() – a method that unassigns a seat.

Implement the class *PlaneSeat*.



- B. Write a class *Plane* that comprises 12 seats. The class should create an array of 12 objects from the class *PlaneSeat*.

The class diagram is given below:

Plane
- seat: PlaneSeat[ ] - numEmptySeat: int
+ Plane() + sortSeats(): void + showNumEmptySeats(): void + showEmptySeats(): void + showAssignedSeats(): void + assignSeat(): void + unAssignSeat(): void

where

seat – instance variable containing information on the seats in the plane. It is declared as an array of 12 seat objects.

numEmptySeat – instance variable containing information on the number of empty seats.

Plane() – a constructor for the class Plane.

sortSeats() – a method to sort the seats according to ascending order.

showNumEmptySeats() – a method displays the number of empty seats.

showEmptySeats() – a method displays the list of empty seats.

showAssignedSeat() – a method displays the assigned seats.

assignSeat() – a method that assigns an empty seat.

unAssignSeat() – a method that unassigns a seat.

Implement the class Plane.

- C. Write an application class *PlaneApp* that implements the seating reservation program.

The class *PlaneApp* should be able to support the following:

- (1) Show the number of empty seats
- (2) Show the list of empty seats
- (3) Show the list of customers together with their seat numbers in the order of the seat numbers
- (4) Assign a customer to a seat
- (5) Remove a seat assignment

The menu should also contain option (6) (i.e. quit) for terminating the program. After the user selects a particular option, the corresponding operation will be executed. If the selected option is not (6), then the program shows the menu for user selection again. This application does not need to save data into a file between runs.

### Important:

Remember to do all the programming inside the sub-directory *lab6* and name the source codes as **PlaneSeat.java**, **Plane.java** and **PlaneApp.java** and name the compiled codes as **PlaneSeat.class**, **PlaneSeat.class** and **PlaneApp.class**.

### Test Data:

Test your application program with the following data:

1. Assign a customer to a seat with SeatID=10, CustomerID = 10001.
2. Assign a customer to a seat with SeatID=12, CustomerID = 10002.
3. Assign a customer to a seat with SeatID=8, CustomerID = 10003.
4. Show the the list of customers together with their seat numbers in the order of the seat numbers.
5. Show the number of empty seats.
6. Show the list of empty seats.
7. Assign (attempt) a customer to a seat with any existing CustomerID, and SeatID. (Should give a warning message!)
8. Remove the seat assignment with SeatID=10.

9. Assign (attempt) a customer to a seat with SeatID = 12.
10. Remove the seat assignment with SeatID=12.
11. Show the list of customers together with their seat numbers in the order of the seat numbers.
12. Show the number of empty seats.
13. Show the list of empty seats.
14. Quit

**Expected outputs:**

- (1) Show number of empty seats
- (2) Show the list of empty seats
- (3) Show the list of seat assignments
- (4) Assign a customer to a seat
- (5) Remove a seat assignment
- (6) Exit

```
Enter the number of your choice: 4
Assigning Seat ..
Please enter SeatID: 10
Please enter Customer ID: 10001
Seat Assigned!
```

```
Enter the number of your choice: 4
Assigning Seat ..
Please enter SeatID: 12
Please enter Customer ID: 10002
Seat Assigned!
```

```
Enter the number of your choice: 4
Assigning Seat ..
Please enter SeatID: 8
Please enter Customer ID: 10003
Seat Assigned!
```

```
Enter the number of your choice: 3
The seat assignments are as follow:
SeatID 8 assigned to CustomerID 10003.
SeatID 10 assigned to CustomerID 10001.
SeatID 12 assigned to CustomerID 10002.
```

```
Enter the number of your choice: 1
There are 9 empty seats
```

```
Enter the number of your choice: 2
The following seats are empty:
SeatID 1
SeatID 2
SeatID 3
SeatID 4
SeatID 5
SeatID 6
SeatID 7
SeatID 9
SeatID 11
```

```
Enter the number of your choice: 4
Assigning Seat ..
Please enter SeatID: 8
Please enter Customer ID: 10004
Seat already assigned to a customer.
```

```
Enter the number of your choice: 5  
Enter SeatID to unassign customer from: 10  
Seat Unassigned!
```

```
Enter the number of your choice: 4  
Assigning Seat ..  
Please enter SeatID: 12  
Please enter Customer ID: 10005  
Seat already assigned to a customer.
```

```
Enter the number of your choice: 5  
Enter SeatID to unassign customer from: 12  
Seat Unassigned!
```

```
Enter the number of your choice: 3  
The seat assignments are as follow:  
SeatID 8 assigned to CustomerID 10003.
```

```
Enter the number of your choice: 1  
There are 11 empty seats
```

```
Enter the number of your choice: 2  
The following seats are empty:  
SeatID 1  
SeatID 2  
SeatID 3  
SeatID 4  
SeatID 5  
SeatID 6  
SeatID 7  
SeatID 9  
SeatID 10  
SeatID 11  
SeatID 12
```

```
Enter the number of your choice: 6
```

## 6. **REPORT**

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## 7. **REFERENCES**

[1] The textbook for CPE102/CSC102.

## **LAB 7: STRINGS AND 2-DIMENSIONAL ARRAYS**

### **1. OBJECTIVE**

The objective of Lab 7 is to get familiar with using strings and 2-dimensional arrays.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-B1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP

Java 2 Platform

### **4. INTRODUCTION**

Ciphers are used to produce messages which can be understood only by the sender and authorized receivers, so ciphers can be used for secret correspondence in situations where the message may fall into the wrong hands. The process of converting a readable message (the plaintext) to an unreadable form (the ciphertext) is called “encryption”; converting ciphertext to plaintext is called “decryption”. The Vigenere cipher was developed by Biais de Vigenere around 1586, for French diplomatic and military communications. The Vigenere cipher uses a keyword and a 26-by-26 matrix of letters to substitute plaintext letters with ciphertext letters, and vice versa. The message sender and receiver must both use the same matrix and keyword. For this experiment, the matrix will be:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

The matrix contains the (uppercase) ciphertext letters. The lowercase letters are the row and column indices.

To encrypt a plaintext letter:

1. The plaintext letter is matched with a keyword letter, as shown below.
2. The keyword letter determines the ROW of the matrix. Find the row that begins with the keyword letter.
3. The plaintext letter determines the COLUMN of the matrix.
4. The ciphertext letter is found at the intersection of ROW and COLUMN.

For example, if the keyword is “calculus” and the plaintext is “beameupscotty”:

Keyword:           calculus  
 Plaintext:         beammeupscotty

The ciphertext will be “DELOGPOHUCZVNJ”.

The first plaintext letter, ‘b’, is matched with the first keyword letter, ‘c’, so the first ciphertext letter, ‘D’, is found in the matrix at row ‘c’, column ‘b’. The second ciphertext letter, ‘E’, is found at row ‘a’, column ‘e’, etc.

To decrypt a message, the process is reversed. The ciphertext is matched with the same keyword as was used in the encryption. As before, a keyword letter determines a row of the matrix. To decrypt a ciphertext letter:

1. Match the ciphertext letter to a keyword letter.
2. Search the row corresponding to the keyword letter, then locate the ciphertext letter.
3. Find the index of the column in which the ciphertext letter appears. The column index is the plaintext letter that corresponds to the given ciphertext letter.

## 5. **EXPERIMENT**

First create a sub-directory called *lab7* in your home directory. Do all the programming in this *lab7* sub-directory and leave all the programs in this *lab7*.

Write a class *Ciphers* that supports the following:

- Allow the user to enter a keyword of 20 characters or less.
- Encrypt a plaintext message of 80 characters or less, using the method described above with the keyword supplied by the user.
- Decrypt a ciphertext message of 80 characters or less, using the method described above with the keyword supplied by the user.

The class diagram for *Ciphers* is given below:

Ciphers
- lookupTable: char[ ][ ] - keyword: String
+ Ciphers() + encrypt(plaintext: String): String + decrypt(ciphertext: String): String + readInKeyword(): void + encryptText(): void + decryptText(): void

Where

lookupTable – an instance variable used to store the 2-dimensional matrix.

keyword – an instance variable to store the keyword.

Ciphers() - is the constructor for the class.

encrypt() – a method that returns the encrypted string.

decrypt() – a method that returns the decrypted string.

readInKeyword() – a method that reads in the keyword.

encryptText() – a method that performs encryption using encrypt().

decryptText() – a method that performs decryption using decrypt().

Implement the class *Ciphers*.

Write an application class *CiphersApp* that should repeatedly print a menu and prompt the user until the user chooses to quit. Your program should print a short introduction for the user and ask for a keyword at the start of the run. The introduction should briefly state the purpose of the program and list

any limitations on the input (e.g. only lowercase letters may be used for plaintext). The application program should print a short exit message after the user chooses to quit.

**Example:**

Enter a keyword: computerscience

You have the following options:

- (1) encrypt a plaintext message
- (2) decrypt a ciphertext message
- (3) quit the program

Enter the number of your choice => 1

Enter a line of text: thequickbrownfoxjumpedoverthelazydogs

plaintext = thequickbrownfoxjumpedoverthelazydogs

ciphertext = VVQFOBGBTTWAAHSZXGBJXHFNGZXUGPCNKSIZW

You have the following options:

- (1) encrypt a plaintext message
- (2) decrypt a ciphertext message
- (3) quit the program

Enter the number of your choice => 2

Enter the ciphertext: VVQFOBGBTTWAAHSZXGBJXHFNGZXUGPCNKSIZW

ciphertext = VVQFOBGBTTWAAHSZXGBJXHFNGZXUGPCNKSIZW

plaintext = thequickbrownfoxjumpedoverthelazydogs

**Assumptions:**

- The user will always make valid menu choices – no error checking is needed.
- Keywords will consist entirely of lowercase letters, 20 or less, no blanks.
- Plaintext will consist entirely of lowercase letters, 80 or less, no blanks.
- Ciphertext will consist entirely of uppercase letters, 80 or less, no blanks.
- Your program should define at least two functions, such as encrypt() and decrypt(). You may use more than two functions.
- You should print plaintext in lowercase letters.
- You should print ciphertext in uppercase letters.

**Test cases:**

- Use the keyword “zygote”, encrypt “zoology” and “aardvark” (separately).
- Use the same keyword “zygote”, decrypt “YSIQAMMG” and “ZPIVTINNZSKCW”.
- Exit the program.
- Use the keyword “apple”, encrypt “zeppelin” and decrypt “ZNBFBVGN”.

**Expected outputs:**

Enter a keyword: zygote

You have the following options:

- (1) encrypt a plaintext message
- (2) decrypt a ciphertext message
- (3) quit the program

Enter the number of your choice => 1

Enter a line of text: zoology

plaintext = zoology

ciphertext = YMUZHKX

Enter the number of your choice => 1

Enter a line of text: aardvark

plaintext = aardvark

ciphertext = ZYXROEQI

Enter the number of your choice => 2

Enter a ciphertext: YSIQAMMG

ciphertext = YSIQAMMG

```
plaintext = zucchini
Enter the number of your choice => 2
Enter a ciphertext: ZPIVTINNZSKCW
ciphertext = ZPIVTINNZSKCW
plaintext = archaeopteryx
Enter the number of your choice => 3
Program terminating ...
```

```
Enter a keyword: apple
You have the following options:
(1) encrypt a plaintext message
(2) decrypt a ciphertext message
(3) quit the program
Enter the number of your choice => 1
Enter a line of text: zeppelin
plaintext = zeppelin
ciphertext = ZTEAILXC
Enter the number of your choice => 2
Enter a ciphertext: ZNBFVGN
ciphertext = ZNBFVGN
plaintext = zymurgy
Enter the number of your choice => 3
Program terminating ...
```

**Important:** Remember to name the source programs as **Ciphers.java** and **CiphersApp.java**, and name the compiled class codes as **Ciphers.class** and **CiphersApp.class** inside the sub-directory *lab7*.

## 6. REPORT

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## 7. REFERENCES

[1] The textbook for CPE102/CSC102.

## **LAB 8: FILE PROCESSING AND EXCEPTION HANDLING**

### **1. OBJECTIVE**

The objective of Lab 8 is to practise on file input/output processing and exception handling.

### **2. LABORATORY**

This lab is conducted in the Computing Lab 1 (N4-B1-10) in SCE.

### **3. EQUIPMENT**

Hardware: Workstation/PC running under the *Linux* environment.

Software: text editor, jGRASP  
Java 2 Platform

### **4. INTRODUCTION**

Java provides a set of input/output streams to store and retrieve data from files. A typical file processing task normally involves reading data from some input file, performing computation using the data, and writing the results of computation to one or more output files. A database is a collection of data organized in a way that allows efficient access to the data. Databases are sometimes organized as a collection of records, with each record containing all the related data items for one entity (such as an employee, department, etc.). We will practise on writing file I/O processing in this lab.

### **5. EXPERIMENT**

Assume you have a file named “names.txt” which contains single names separated by newline characters and no blank lines. For example:

**names.txt**

Alex Beckham Giggs Blanc Neville Scholls Keane McDonald KennyRoger BurgerKing AandW PHut
---

Write a Java program (with the necessary I/O exception handling code) which will sort the names into two files based on their length. Names with 5 letters or less should be written to a file named “small.txt”. Names with more than 5 letters should be written to a file named “big.txt”. Maintain the relative order of the names as they appear in “names.txt”. Each name should be followed by a newline character. Neither file should have any blank line. Given the above “names.txt” file, your program should produce the following two files:



**small.txt**

```
Alex
Giggs
Blanc
Keane
AandW
PHut
```

**large.txt**

```
Beckham
Neville
Scholls
McDonald
KennyRoger
BurgerKing
```

Create the “names.txt” file in your directory. For the names in “names.txt”, you may assume that:

- Each name is less than 20 characters (not counting the newline character).
- Each line contains a valid name (all characters, no space).
- Each name is terminated by a newline character.
- There is at least one name with 5 or less than 5 characters.
- There is at least one name with more than 5 characters.
- There is no blank space or blank line at the end of the file.

**Important:**

Remember to do all the programming inside the sub-directory *lab8* and name the source code of this program as **Lab8p.java** and name the compiled class code as **Lab8p.class**.

**Test Data:**

Use the “names.txt” as your test data.

**Expected outputs:** as shown from the sample data.

## 6. **REPORT**

There is no submission for this lab experiment. However, you do need to record the followings into your log book for inspection:

- program listings;
- the typescript file(s) to show the results of the testing of your programs using the test cases specified in this lab manual only.

## 7. **REFERENCES**

[1] The textbook for CPE102/CSC102.