Worksheet 2: Programming Basics

The objectives of this practical are to:

- learn how to design a programming solution with pseudocode;
- learn how to implement a solution using Java code;
- learn how to choose variable types and use them in solutions;
- learn how to retrieve user input in various formats.

1. Introduction to pseudocode

Recall from the lecture that pseudocode is written to help us design the solution to a problem, without having to consider issues with particular programming language syntax (such as for Java). The idea of pseudocode is that you can focus on solving the problem at hand correctly without being interrupted with syntax issues.

Pseudocode can be written in any structured format you wish; there is no one 'right answer' as it is not a programming language. The below example is just one way of writing pseudocode but if you are consistent with your language, do not use syntax similar to a programming language and can fully explain the logic of your program, then any syntax can be used for pseudocode.

We call this the 'CLUCC' principle; that is, your pseudocode is:

(a) **c**lear – is it described in the simplest way possible, or close to it?

(b) **l**ogical – does it make sense and cover the full logic of the solution, but does not include language-specific features such as initialising **Scanner**s?

(c) **u**nderstandable – could another programmer use this to implement a solution in Java or another programming language?

(d) **c**orrect – does the pseudocode solve the problem at hand?

(e) **c**onsistent – is user input is described in the same manner each time?

Don't forget that **it is its own language** – pseudocode should allow implementation in any language, so syntax based off existing languages such as Java or Python is not allowed.

Your tutor will now go through an example of pseudocode to explain the different parts and what they mean, just to reinforce the concepts. Remember that the style used is only an example; as detailed above, you can use your own style if it meets the three requirements.

It is okay if you don't remember the concepts of pseudocode fully; now is your chance to ask as many questions as possible before you write your own!

In the terminal, make sure you're in your **~/Documents/PDI/P02** folder. Ask your tutor if you are unsure about where you currently are or how to get to this directory.

> **Note:** The **pwd** command may be useful to you here.

To get started on your pseudocode, you must first create a **.txt** file in Vim. Over the semester, you will create two main types of files: **.txt** (for pseudocode and word processing style documents) and **.java** (for Java source code).

In your terminal, issue the command:

```
[user@pc]$ vim Calculator.txt
```

This will create a blank text file for you in Vim to type your pseudocode into. Below is some pseudocode you will need to type into this text file.

> **Note:** Remember to press `I` to enter **edit** mode. While you can copy and paste this pseudocode, **please** type it out. Your tutor will instruct you to do so if you haven't.

```
BEGIN Calculator
    MAIN:
        numOne <- ASK USER WITH PROMPT "Please enter the first number: "
        numTwo <- ASK USER WITH PROMPT "Please enter the second number: "

        answer <- numOne + numTwo

        PRINT "The answer is " numOne " + " numTwo " = " answer
    END MAIN
END
```

Make sure you copy everything exactly as it is above, including spacing and with the same capitalisation. However, do not worry that the colours don't match – Vim won't do syntax highlighting of pseudocode. However, take note of certain words that are in all capital letters. Consider the following:

(a) What do the capitalised words mean?

(b) Why are those words in capital letters?

(c) What does the content next to the **PRINT** statement do?

Once you have written everything that is in that file into your file, press `Esc` to enter command mode followed by **:wq** to save and exit your file.

2. Write your own pseudocode

Now that you have an example of pseudocode, it's time to write your own!

Your task is to design a simple program, using pseudocode, that converts a value input by the user from degrees Celsius to degrees Fahrenheit.

Your program should follow the steps below:

(a) Ask the user to input the temperature, as a whole number, in degrees Celsius (**c**);

(b) Convert the temperature to degrees Fahrenheit (**f**);

(c) Output the result of the conversion, as a whole number, to the user.

Ensure your user input and output are user friendly, however do not worry about the degrees symbol (°).

To assist you in making this program, you may wish to consult the unit materials, such as the lectures and worksheets, for examples of pseudocode. If you're having trouble writing the pseudocode, or don't know where to start, ask your tutor for help and they will happily guide you in the right direction.

Note: For your reference, the equation for conversion from Celsius to Fahrenheit is:

$$\frac{9}{5} * c + 32 = f$$

where c is the value in degrees Celsius and f is the value in degrees Fahrenheit.

Open a file named **CelsToFaren.txt**. Design your algorithm within this file, using pseudocode. Rather than Vim, you may wish to investigate **gvim** as an alternative code editor. Keep in mind, you will be unable to use this editor under a solely terminal-based interface (such as if you remotely access the lab machines using SSH).

Once you have completed your pseudocode, **please** get your tutor to have a look at it first before moving onto the next step.

Note: Fixing problems early on can save you a whole lot of time later down the track!

3. Introduction to Java

Now that you've designed a solution in pseudocode, it's time to implement it by writing the Java equivalent. Firstly, however, we will go through an example of creating and compiling a Java code file. Make a new Java file with the command:

```
[user@pc]$ vim Calculator.java
```

This will create a new blank file for you to write your Java code into and open it with Vim.

Note: As with the example pseudocode above, please type this out using Vim rather than copying-and-pasting. In the same manner as above, your tutor will instruct you to manually copy it out if you aren't doing so. The ↩ indicates that the line continues below.

```java
import java.util.*;

public class Calculator
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int numOne, numTwo, answer;

        System.out.print("Please enter the first number: ");
        numOne = sc.nextInt();

        System.out.print("Please enter the second number: ");
        numTwo = sc.nextInt();

        answer = numOne + numTwo;

        System.out.println("The answer is: " + numOne + " + " + numTwo + " = " + ↩
            answer);
        sc.close();
    }
}
```

Just like the pseudocode, make sure you copy everything including the spacing. Don't worry if the colours don't match exactly. As long as there is some syntax highlighting, everything is okay. Consider the following questions.

(a) What are the differences between the pseudocode you wrote and this Java code?

(b) If they are equivalent, why are there more lines here that we didn't write earlier?

(c) What things are similar between the two?

Make sure you save and close your file using Vim when you are done. Now that you've written both pseudocode and Java code, it's time to compile and run your program!

Java is (to a degree) a compiled language; you will need to compile your `.java` source code files that you wrote into `.class` files so they can be run by your computer using the Java Virtual Machine. You will need to do this each time you change your source code, if you want to run the changes you've made.

Make sure that you saved your file and exited Vim. Whenever you run a program, you must first always compile your code - otherwise, you will be unable to run it. Even if you make a small change, you must compile your program again for that change to be reflected when it is run.

> **Note:** You do not need to (nor is it really possible to) compile your pseudocode, as it is only a template for your Java code. Only your Java code will be compiled and run.

In your terminal, issue the command:

```
[user@pc]$ javac Calculator.java
```

The **javac** command followed by the name of the Java file will compile that Java source code file. This will create a new file in your directory called **Calculator.class**. This file contains the bytecode (built from your source code) that the Java Virtual Machine will use to run your program. It is also why you must always compile your program before running it as it will update your **.class** file with your changes.

You may see some messages output – these are error messages. If so, double-check you have copied the above code identically. If you have and are still having issues, ask your tutor. Only once there are no error messages output will a valid **.class** file be generated.

To run your compiled Java program, in your terminal, issue the command:

```
[user@pc]$ java Calculator
```

The **java** command followed by the name of the **.class** file tells Java to run that class file. When you run the program, follow along with what the output says and press enter when responding to the program's outputs. Consider the following questions:

(a) What sort of outputs did you get from the program?

(b) What did it ask you to enter?

(c) What did it do with the input you gave it?

(d) What was the purpose of this program?

(e) Can you follow along with what the code says, to match your assertions?

Don't stress if you can't answer all of these questions straight away. They are just some things you can start thinking about before next week. This file is a good example of the style of Java code you are expected to write in this unit. Feel free to use the file as a reference for when you are writing your own Java code.

4. Converting pseudocode to Java

Now that you've written the pseudocode for your temperature converter program, the pseudocode can now be used to write Java code that implements the temperature . Pseudocode and Java may map one-to-one (one line of pseudocode is equivalent to a line of Java), but this is not always the case.

To assist you with this, there is is a document on Blackboard named **"Java Coding Standard"**. This shows you the expected formatting and syntax required for Java code written in this unit.

Use your pseudocode as a guide to implement a solution in Java. Using your knowledge from what your tutor has demonstrated and what was covered in the lecture, alongside the resources in Blackboard, take each component of your pseudocode and write the corresponding Java code.

Once you have written your Java code, compile it and fix any errors, indicated by the messages that appear when you attempt to compile your code. If you are unsure as to what they mean, try to look at the line numbers and the error text (indicated on the first line in the error below). Can you tell what the cause of this compiler error is?

```
MyComplexProgram:171: error: cannot find symbol
    someValue = sc.nextInt();
    ^
symbol:   variable someValue
location: class MyComplexProgram
```

If you're still unsure, ask your tutor to walk you through what the errors are and how to fix them.

> **Note:** A helpful hint – keep a journal of different error messages you see and how you fixed them. This will save you a lot of time in the future when writing other programs, as you can refer back to your previous work!

Feel free to refer back to your first practical if you have forgotten how to compile and run your program. Once you have fixed any errors, you can now compile your program again (successfully) and run it.

> **Warning:** Just because your program compiles, that does not mean it works properly! This is why we must test our program every time. If your program compiles, it means that the Java syntax is correct. However, it doesn't mean what you have written calculates the right answer.

If your program works as intended, **please** get your tutor to have a look at your Java code before moving onto the next step.

5. Putting it together: ASCII converter

After completing this practical, you should be familiar with designing a basic program using pseudocode and implementing it with Java, as well as evaluating Java expressions. Your last task for this practical is to complete the following **in your own time.**

Create a sub-directory in your **P02** directory called **AssignmentTask** and move into it.

Your task is to create an ASCII converter; you will need to gather an uppercase character input from the user, and convert it to the ASCII value it represents. Then, you will need to display that back to the user, as well as the lowercase equivalent.

You must design this first in pseudocode, then implement it in Java and then test it with a copy of the test plan, modified for this purpose.

Name your files **CharConverter.txt** and **CharConverter.java** accordingly.

Open a file called **WhyIsThisImportant.txt** and answer the following question: **Why would we need to use ASCII values?** Think about the differences between characters and integers.

> **Note:** You may assume that all inputs are valid uppercase characters.

An example of the output expected is as follows, noting that the user provides input on the first line:

```
Please enter an uppercase character: A
The ASCII value of 'A' is: 65
The lowercase value of 'A' is: 'a'
```

> **Note:** ASCII is a code for representing English characters as numbers. Each letter of the English alphabet is assigned a number ranging from **0** to **127**. For example, the ASCII code for uppercase **P** is **80**.
>
> You may need to have a look at the difference between uppercase and lowercase characters in order to do the conversion. An ASCII table has been included below, at the end of this document, for your reference.

You will need to read the lecture notes and watch this weeks' lecture in order to work out how the conversion can be achieved. The solution involves the use of the **char** and **int** data types alongside typecasting. If you are stuck for ideas as to how to represent the process of typecasting, consider the following as an example of casting an integer to a floating point ('real') number:

```
floatValue <- intValue AS Real
```

> **Note:** **Scanner** does not provide a **nextChar()** function to read in a character; recall the lecture demonstration to see how to read in a single character.

If you are an internal student at Bentley and having trouble, feel free to ask the Senior Tutor; they will be more than willing to guide you in the right direction.

ASCII Table

Decimal	Hex	Binary	Octal	Char	Decimal	Hex	Binary	Octal	Char
48	30	110000	60	0	96	60	1100000	140	‘
49	31	110001	61	1	97	61	1100001	141	a
50	32	110010	62	2	98	62	1100010	142	b
51	33	110011	63	3	99	63	1100011	143	c
52	34	110100	64	4	100	64	1100100	144	d
53	35	110101	65	5	101	65	1100101	145	e
54	36	110110	66	6	102	66	1100110	146	f
55	37	110111	67	7	103	67	1100111	147	g
56	38	111000	70	8	104	68	1101000	150	h
57	39	111001	71	9	105	69	1101001	151	i
58	3A	111010	72	:	106	6A	1101010	152	j
59	3B	111011	73	;	107	6B	1101011	153	k
60	3C	111100	74	<	108	6C	1101100	154	l
61	3D	111101	75	=	109	6D	1101101	155	m
62	3E	111110	76	>	110	6E	1101110	156	n
63	3F	111111	77	?	111	6F	1101111	157	o
64	40	1000000	100	@	112	70	1110000	160	p
65	41	1000001	101	A	113	71	1110001	161	q
66	42	1000010	102	B	114	72	1110010	162	r
67	43	1000011	103	C	115	73	1110011	163	s
68	44	1000100	104	D	116	74	1110100	164	t
69	45	1000101	105	E	117	75	1110101	165	u
70	46	1000110	106	F	118	76	1110110	166	v
71	47	1000111	107	G	119	77	1110111	167	w
72	48	1001000	110	H	120	78	1111000	170	x
73	49	1001001	111	I	121	79	1111001	171	y
74	4A	1001010	112	J	122	7A	1111010	172	Z
75	4B	1001011	113	K	123	7B	1111011	173	{
76	4C	1001100	114	L	124	7C	1111100	174	|
77	4D	1001101	115	M	125	7D	1111101	175	}
78	4E	1001110	116	N	126	7E	1111110	176	~
79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
80	50	1010000	120	P					
81	51	1010001	121	Q					
82	52	1010010	122	R					
83	53	1010011	123	S					
84	54	1010100	124	T					
85	55	1010101	125	U					
86	56	1010110	126	V					
87	57	1010111	127	W					
88	58	1011000	130	X					
89	59	1011001	131	Y					
90	5A	1011010	132	Z					
91	5B	1011011	133	[					
92	5C	1011100	134	\					
93	5D	1011101	135	]					
94	5E	1011110	136	^					
95	5F	1011111	137	_					

End of Worksheet