Worksheet 4: Looping

Updated: 26th March, 2020

The objectives of this practical is to:

- Practice using loops
- Learn how to use arrays
- Implement bounds checking

Note: You must submit this practical worksheet by:

Sunday, 5th April 2020, 5pm (local time)

And have it signed off during your next **registered** practical session.

Submissions must be done **electronically** through Blackboard. You are only required to submit once for this practical (your **entire** `P04` directory in **`.tar.gz`** format). You must submit everything you have completed in this practical, both in class and at home.

You must also not modify the files in `~/Documents/PDI/P04` until after your submission has been marked. During your practical your tutor will check the modified dates of the files using `ls -l` (el) before marking your work.

To create a gzipped tarball use the following command from your **`PDI`** folder:

```
[user@pc]$ tar -cvzf <studentID>_P04.tar.gz P04
```

For more information on what each argument of the above command does use:

```
[user@pc]$ man tar
```

To get your **`.bash_history`** into a submittable format, first close all terminals down, using <ctrl>-d. Then open a new terminal, and type this command from anywhere:

```
[user@pc]$ history >~/Documents/PDI/P04/BashHistoryP04.txt
```

Your submission will be marked during your next **registered** practical session via an online interview (in `Blackboard Collaborate Ultra`) from your tutor, comprised of a few questions.

Please note that the questions asked in the interview may cover the entirety of the worksheet, not just the material in your Assignment Task. Your submitted work will be assessed and marks will be allocated accordingly.

1. Preamble

As we are moving to a fully online solution, there will be no physical presence of the tutors. However, they will still be available in the same capacity, but on Blackboard Collaborate Ultra[1]. If you have not read your announcements recently regarding the current situation, please read them and make yourself aware of how we intend to teach you.

If you are having trouble adapting to this solution, please contact your Lecturer/Unit Coordinator for additional assistance.

There are also other avenues of assistance available to you to continue your studies:

- **Study Support**
- **Communicating Online Guide**
- **Curtin Online Students**

Rest assured your teaching staff are here to support you in this new mode of study, so please **do** reach out with any concerns you might have.

2. Looping The Menu

Last week you created a Menu for the user to select what they would like to do, from a list of 6 options. Your task for this week is to loop this menu so that the program only exits (or stops running) when the user chooses to exit the program.

> **Note:** Remember the "Zero Rules" slide. This **must** be followed.

Again, copy your **`Menu.txt`** and **`Menu.java`** files over from your `P03` folder into your `P04` folder.

Your menu should include this option from last week:

```
> 0. Exit
```

Currently, this should only be printing a goodbye message if selected by the user. Now, we're going to use it for another task. When the user selects this option, we're going to stop the program. (But it stops anyway doesn't it?) Well now, in other cases, we're going to change that!

Using what you've learnt in the lecture about loops, you need to loop your menu, so that it loops **one or more times**.

The menu must keep printing to the screen (and thus the menu should loop) until and only until the user selects `0`.

> **Note:** Think of how you could use a variable to achieve this.

[1] Under Blackboard, select PDI and on the Left pane "Collaborate Ultra", near the bottom.

3. User Input Looping

Now that you've given the user the option to exit the program, let's make sure they're inputing the correct value.

User input validation (or sanitising your input) is crucial for any program, as it stops the program attempting to use "unclean" data. We assume the user doesn't know what they are doing and we cannot guarantee that if we ask them to enter in a value between 1 and 12, that they actually will do it accurately.

This is an example of the pseudo code for generic user input we expect to see in PDI:

```
prompt := "Please enter a value: "
errorMsg := "Error: the value must be between <MIN> and <MAX>"
out := prompt
DO
        OUTPUT out
        INPUT value
        out = errorMsg + prompt
WHILE((value > MAX) OR (value < MIN))
```

This loops the bit of code that prompts the user and stores user input. This continues until the user inputs a value that is within the specified range of **`MIN`** and **`MAX`**. We use a do-while loop because we need this bit of code to run at least **one or more times**.

The error message and output string are initialised before the loop starts so we can choose what we want to output and when. For example: when the user is first prompted to enter a value, they don't need to see the error message yet because they haven't entered in anything incorrect yet. Only when the user inputs an incorrect value, we display the error message along with the original prompt.

> **Note:** MIN and MAX are placeholder values that you must fill in with your own values.

Your task is to make sure every user input in your **`Menu.txt`** program is validated. **The format shown above is only currently applicable for integer inputs**. If the user has to input a character, then it is up to you to validate whether the user's input is one of the specified choices.

For example: if the user selects **`'C'`** for Celsius, that is a valid choice. If the user were to enter **`'R'`**, that would not be a valid option thus the user should be presented with an error message and asked to input a response again until their choice is valid.

> **Note:** Think of how you could utilise the **`ELSE`** and **`DEFAULT`** in your **`IF`** and **`CASE`** respectivley.

Make sure you follow this structure carefully because we'll be expanding on this in later weeks!

4. Arrays - The 1st Dimension

Hopefully by now you would be well acquainted with arrays from the lectures. Now it's time to learn how to implement them!

The variables that we have seen so far represent a single item, but we also often work with sets or groups of a similar data type. For example, a set of student marks, rather than each mark stored in a singular variable, we can store them in all in the same variable (an Array!). Arrays are the simplest kind of data structure for storing these sets of data. Instead of just one element, an array is a variable that contains many elements of the **same data type**.

The variable itself is a reference to the first element of the array. In Java, the "array variable" also knows how large the array is (meaning you must declare the length of the array when you declare the array itself), as they are a contiguous block of memory.

Your task is to design an algorithm that will:

- Take in an `Integer` from the user, for the length of the array, this must be between `1` and `20` (inclusive). Ensure you check for edge cases.
- Assign a random number to every element in the array (see Note 1).
- Take in another user input for an integer (must be greater than 0)
- Search the array for that number (See Note 2)
- Output either "`Found: <Array index number>`" or "`Not found: <Search Int>`" to the user depending on which one is true

As always, please refer to the **`Pseudo Code Style Guide`** for syntax in pseudocode and Java.

Note: 1. Random should be used in a similar manner to Scanner. Below is an example of how to use Random in java:

```
Random rand = new Random();
int number;

number = rand.nextInt(...);  // Replace the dots with actual values
```

So just as you declare a **`Scanner`**, you do the same for **`Random`** and just as you would take in an int via a **`Scanner`**, the same is done with **`Random`**.

Note: Once of the "Zero Rules" states that you cannot use **`break`** in a loop. Consider how searching works in a real world application. For example, searching your house for your lost keys (you don't keep looking once you have found them). Apply this logic to your programming. Think about how loops work, and how you can apply this to iterate over your array to search for your value.

5. Arrays - The 2nd Dimension

Now that you've learnt how to implement a one dimension array, now let's implement a two dimension array. Your task is to fill in the 2D array with the Times Tables:

- Take in an `Integer` from the user, for the number of **`rows`** in your table
- Take in another `Integer` from the user, for the number of **`columns`** in your table
- Populate each element in the array with the respective times tables (Assume we start from 1)
- Output the entire table back to the user with headings (see the sample output).

Note: For formatting's sake, you may assume that the user will not input a number greater than **`9`** for **`rows`** or **`columns`**, but you may try implement dynamic formatting for a challenge.

A sample output:

```
Please Enter Rows: 3
Please Enter Columns: 5
```

```
The 3 x 5 Times Tables:

  | 1  2  3  4  5
-------------------
1 | 1  2  3  4  5
2 | 2  4  6  8  10
3 | 3  6  9  12 15
```

Note: Similarly to 1D arrays, you need to declare the columns and rows. For example:

```
// Some declaration of rows and columns

int[][] twoD = new int[rows][columns];
for(int y = 0; y < twoD.length; y++)
{
    for(int x = 0; x < twoD[y].length; x++)
    {
        // This iterates through every element in the 2D array
        twoD[y][x] = x - y;
    }
}
```

The 'x - y' is an arbitrary value, fill that in with your own values

6. Errors, Redundancies and Inefficiencies

The algorithm below contains at least 3 errors, inefficiencies or redundancies. Describe three of them. You must state WHY each is a problem.
Note 1: The purpose of the algorithm is not relevant.
Note 2: There are more than 3 issues, you only need to provide 3.

```
1  found := TRUE
2  numElements := 0
3
4  OUTPUT "Please enter the length you'd like your array to be:"
5  INPUT numElements (Integer)
6  CREATE array OF INT, SIZE OF numElements // Assume the array is filled here
7
8  OUTPUT "Please enter the integer you would like to search for in the array:"
9  INPUT searchInt (Integer)
10
11 FOR ii := 0 TO numElements INC BY 1
12     foundAt := 0
13     IF array[ii] EQUALS numElements THEN
14         foundAt := ii
15         found = TRUE
16         BREAK
17     END IF
18 END FOR
19
20 OUTPUT "Integer found at" foundAt
```

Note: To get the most out of this activity, it is strongly advised you at least attempt to find at least one error in **each** category. Before you ask your tutor what is an error, redundancy or efficiency, please read the definitions below.

- **What is an error?**
  - An **error** is a piece of code that is incorrect, or has incorrect logic or does not align with the coding standards.
- **What is a redundancy?**
  - A **redundancy** is a piece of code that 'does nothing', meaning there is no point having it in the code either because it has no functionality or it is repeating the functionality of something else.
- **What is an inefficiency?**
  - An **inefficiency** is a piece of code that *could* be re-written in a different way to do the same job as the original piece of code, but better and more efficient.

7. Assignment Task: Matrix Multiplication

Your last task for this practical is to complete the following task **in your own time**.

Warning: This question goes towards your portfolio/assignment mark and thus any collusion will be dealt with as per university policy.

Now that we have learnt the basics of 1D and 2D Array's, we can apply them in a practical, real world example.

Consider a matrix A of size $M \times N$ and a matrix B size $N \times K$. Let $A_{i,j}$ and $B_{i,j}$ respectively be the entry of matrix A and B at row i and column j. Assume each entry is an integer value. Let C be the matrix product of A and B, i.e., $C = A \times B$ recall that as the result, matrix C has M rows and K columns. Further, if $C_{i,j}$ is the entry of matrix C at row i and column j, we calculate:

$$C_{i,j} = \sum_{r=1}^{N} A_{i,r} \times B_{r,j}$$

An example of the multiplication process is shown below.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \times \begin{bmatrix} u & v \\ w & x \\ y & z \end{bmatrix} = \begin{bmatrix} au + bw + cy & av + bx + cz \\ du + ew + fy & dv + ex + fz \end{bmatrix}$$

Note: We can only multiply matrices if the number of columns in the first matrix is the same as the number of rows in the second matrix. To check if the number of columns and rows you may use the length of the arrays. (Hint: think of this as your input validation for the matrices.

You are required to design an algorithm that multiplies two matrices together. Your file should be called **`MatrixMultiplication.txt`**. You will be getting the matrices from a static class (it is on Blackboard **`Matrix.java`**). Ensure that this Static Class is located within the directory that **`MatrixMultiplication.java`** is in.

You do not need to understand how these classes work just yet, however, you need to implement a way to retrieve the Array's from this class. Similar to the Math class, when accessing PI, we can access our Array's in the Static Class with "**`Matrix.ARRAY_A;`**". As the **`ARRAY_A`** and **`ARRAY_B`** in the Static class are stored as **`final`**, they cannot be changed. Keep that in mind when you are creating Array "C".

You are required to complete the loops in the above algorithm. You should start with pseudocode then convert it to Java once you have hand tested your algorithm. Your output must print the matrix in correct matrix form, ie: with the correct number of rows and columns. this means you will need to utilise a nested loop to access each element individually.

Note: Think about what type of loop you should be using. Do you know how many times you want to loop?

Once the multiplication is complete, and stored in a new array (**`matrixC`**) you will need to print out the resulting matrix to the user, ensure that you print it with spaces in between each element and new lines on new columns.

You will need to fill in a testing file **`MatrixMultiplication_Test.txt`** for this activity. Use the test results files you have in your `PDI` folder, for your design. Do not forget to actually test your algorithm and fill out the test results.

End of Worksheet