Worksheet 5: Modularity

Updated: 5th April, 2020

The objectives of this practical is to:

- Practice using loops
- More array implementation
- Implement bounds checking
- Test-Style questions

Note: You must submit this practical worksheet by:

Sunday, 26th April 2020, 5pm (local time)

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

You will have ∼3 weeks to complete this practical, so it is quite big. Ensure you spend your time wisely.

Submissions must be done **electronically** through Blackboard. You are only required to submit once for this practical (your **entire** `P05` 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/P05` 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>_P05.tar.gz P05
```

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/P05/BashHistoryP05.txt
```

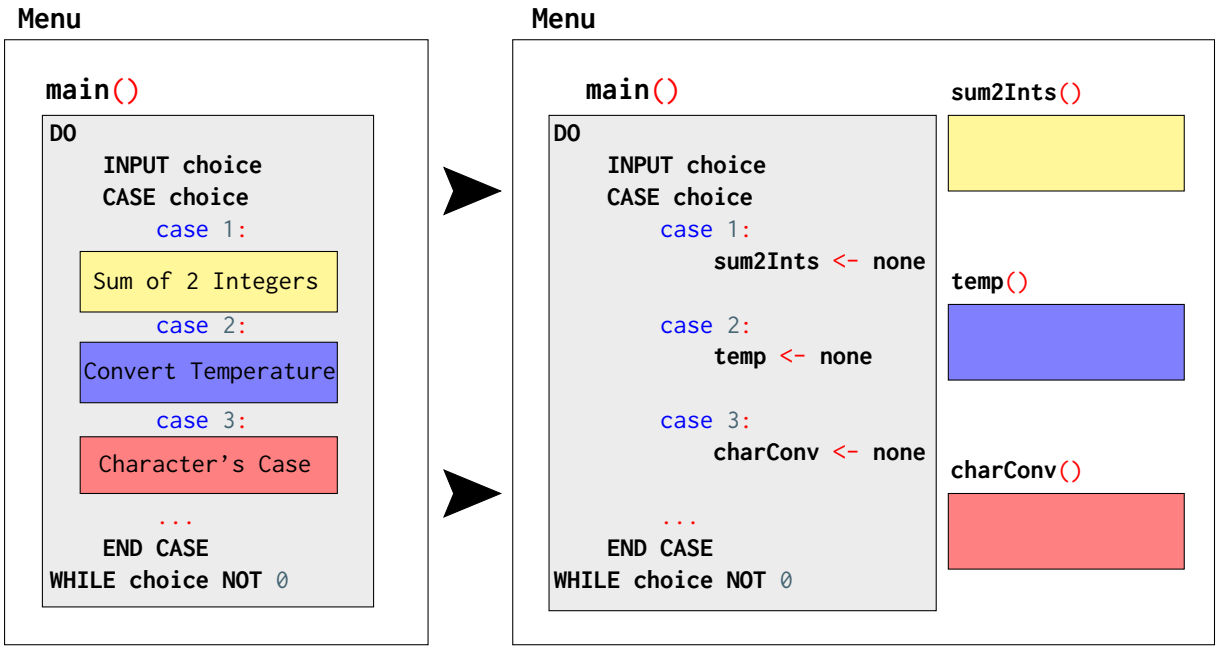
Your submission will be marked during your next **registered** practical session via an online interview (in `Blackboard Collaborate Ultra`) from your tutor, comprised of a few questions.

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. Introduction to Submodules

Up until now, our entire **Menu** has gotten quite large, and is contained all within one submodule, **main()** (yes, **main()** is a submodule). Depending on how you approached the problem, the structure of your code should be similar to that shown on the left.

Now its time to modify and adapt your **Menu**, so that it becomes more modular and makes effective use of submodules.



Modify your pseudo code by separating the main into several submodules, one for each "sub task". The structure should now resemble that shown on the right. Your main will still contain the menu input and selection, but now each menu option should just call the relevant submodule. Remember to follow the naming conventions and all good practises outlined in the lectures.

Note: Readability of your pseudo code is just as important as that of actual code. For now, this is a good set up to use for your submodules, you should maintain indentation in pseudo code too.

```
SUBMODULE: moduleName
IMPORT: variableName1 (DataType), variableName2  (DataType), ... etc. or none
EXPORT: singleVariableOnly (DataType) OR none
ASSERTION: "To retrieve some sort of result, or to do some task."
ALGORITHM:
    ...
    ...
```

Separating distinct tasks into their own submodules is one purpose of modularisation. However, if you examine these new submodules they still perform multiple tasks: input, calculation, and output.

We will deal with the easiest of these first, the calculation. Go ahead and create a submodule for each of the calculations you do in your **Menu**, each submodule should

import the necessary values, perform the calculation, and *export* the result. These submodules seem fairly basic due to the scale of this program, but you will *need* to learn how to make your code modular for future algorithms.

> **Note:** You may ignore the INPUT for now, as we will cover how to make that Modular in the next part of this worksheet.

By now, our **`Menu`** is quite large, and *hopefully* is correctly split up into its own tasks.

Now its time to copy all of the work that we did last week (Arrays 1^{st} Dimension, Arrays 2^{nd} Dimension, and Matrix Multiplication) into our **`Menu`** as well.

> **Note:** Ensure that we are keeping to our modular theme, by breaking up each task into smaller parts and turning them into submodules.

Your **`Menu`** algorithm should now be broken up into many submodules each doing one specific task. Ensure that if you have any repetition of code in your submodules (excluding single line print statements) that they are put into their own submodules now.

Its a good idea to check with your tutor, with regards to the amount and complexity of each submodule. Those of you taking Introduction to Software Engineering (`ISAD1000`) should reflect on the coupling and cohesion of each submodule.

Also, the name of our class (**`Menu`**), although correct, is a bit lacking when it comes to being descriptive. Our task now is to change the name of our **`Menu`** to **`PDIPortfolio`** to showcase our work that we have created.

2. More Modular Menu Modifications

Now go back to our newly created submodules, have a look at their algorithms. Are there any consecutive lines that are very similar? What about all the times we require INPUT from the user? Again, those lines should look fairly repetitive. "But I output different prompts!", that is fine. If you think about it your output string is just an argument you are passing to a method (eg., `System.out.println()`).

> **Note:** In pseudo code we do not usually worry about data types, we just ran into one of the few situation where it matters.
>
> **DivisionChecker** requires an Integer from the user, whereas **TempConverter** requires a Real and **CharConverter** requires a Character
>
> This means you need three submodules, one for inputting an Integer, one for inputting a Real and one for inputting a Character.

The input submodules are another example of code re-usability, the input of every Real number is now dealt with in a single method. The input submodules should contain the loops necessary to ensure the value is **valid**, meaning that the EXPORT of the submodule is guaranteed to be valid.

Similar to last week, below is a (slightly modified) example of an Input submodule.

```
SUBMODULE: inputX
IMPORT: prompt (String), min (X), max (X)
EXPORT: value (X)
ALGORITHM:
    value = min - 1
    error = "ERROR value must be between " + min + "and " + max
    outStr = prompt

    DO
        OUTPUT outStr
        INPUT value
        outStr = error + NEWLINE + prompt
    WHILE (value < min ) OR (value > max)
```

We will build on these INPUT submodules next week when we cover Exceptions. This will allow us to stop the user "crashing" our program when they enter the wrong datatype to our Scanner.

3. Scope Example

```
public class ScopeImage
{
    public static void main(String[] args)
    {
        int var1;
        if(booleanExpression)  // Assume this is declared
        {
            //var1 is accessible here
            int myVar;

        }//myVar ceases to exist here

        //myVar does not exist here
        //var1 is still accessible here

    }//var1 ceases to exist here

    public static void submodule1()
    {
        //var1 cannot be accessible here
    }

    public static void submodule2()
    {
        //var1 from main cannot be accessed here either

        /*this is a different var1 to the one declared in main
          ie: it is stored at a different location in memory*/
        int var1;
    }
}
```

This is what the scoping looks like for a class with submodules. Notice how local variables declared in the different submodule's scope cannot be seen/accessed by other methods. Each "scope" is defined in a literal sense, by the two curly braces surrounding it. If you need something to be accessed in a different scope, either pass it around or declare it in a higher scope.

Warning: Be careful about declaring variables in too high of a scope, once they leave a method they are considered as a "Global Variable" - this leads to 0 marks being applied to the entire assessment. FYI **`main()`** is also a method.

Note: When you get to creating objects this becomes slightly less restrictive.

4. Pond Storage

Given the following description you are required to design an algorithm. Create a subdirectory in this week `P05` directory and call it **`PondCalculator`**.

> **Note:** Think about what submodules you can reuse!

Three acquaintances, Joey, Cory, and Rachel, have decided to start keeping aquatic animals in their ponds. They require help figuring out the volume of water they need in order to completely fill their ponds. For each person you are required to, based on the `depth`, `length` and, `width` of their ponds, calculate the `volume` of water they require. The `depth`, `length` and `width` of the ponds should be input by the user as metres.

Each person has two options for what animal they can keep, your program should prompt the user to select one of the two options, and loop to ensure a valid option is selected. Then based on the volume of water and the table below find out how many animals Joey, Cory, or Rachel could store in their ponds. You do not have to take into account the displacement of the animals in the pond. Output the number of animals rounded to the lowest whole number.

Your algorithm should contain a looping menu that allows the user to pick whose pond they wish to calculate each repeat, and then which animal that person should keep (you can only keep one species in the pond at once).

Person	Animal	Animals Per m3
Joey	Sting Rays	0.5
	Arowana	0.4
Cory	Koi	0.6
	Puffer Fish	0.8
Rachel	Turtles	1.2
	Frogs	4.5

The output should look similar to this

```
Joey can store 200 sting rays in his 400m^3 pond.
```

OR

```
Cory can store 687 puffer fish in his 550m^3 pond.
```

OR

```
Rachel can store 182 turtles in her 152m^3 pond.
```

Once you have completed the pseudocode have a go at converting it to Java.

As always you are required to test your code. Ensure you are filling out your Testing file. It is **OK** if your code for this question fails some tests, you are still learning, the important thing is to **always** test what you have done so you can understand what has/hasn't worked.

5. Fibonacci

Create a subdirectory in this week `P05` directory and call it **`Fibonacci`**.

Your next task is to create a sequence generator, the program should be able to generate a sequence similar to fibonacci, but have the ability to specify the first two values and to decide if addition or subtraction should be used to generate the sequence. Your algorithm will need to perform the following steps:

(a) Prompt the user for the mode of generation (subtraction or addition).

(b) Prompt the user for the number of elements that should be generated, an integer between 5 and 50 (inclusive).

(c) Prompt the user for the first two digits of the sequence, integer between -100 and 100 (inclusive).

(d) Create an array of the appropriate size and fill it according to the above information. The first and second elements of the array should be the two digits entered in step 2. Then either add or subtract the two together to create the third element, apply the same calculation to the second and third elements to generate the fourth element.

(e) Provide a looping menu to the user where they can view a specific element of the array, or have the entire array displayed to them.

For example: if the user selects subtraction, 10 elements, and starting digits 2 and 3 then the pattern generated would be `2,3,-1,4,-5,9,-14,23,-37,60`. If the user then specifies the fourth element should be printed then -2 should be printed.

> **Note:** While arrays in most languages are 0 indexed, asking an end user with no programming knowledge to understand and use a 0 based index is extremely detrimental to usability. Hence the user wanting to see element 1, is actually the first element in the array, that is index 0.

You must make good use of submodules in your algorithm, make sure you pay attention to code reuse and don't forget to utilise constants.

Once your pseudocode design is completed you can convert your design to java and complete the test results document.

6. Eulers Constant

Create a subdirectory in this week `P05` directory and call it **`Euler`**.

Euler's constant can be calculated as the sum of the infinite series:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots$$

Note: 0! = 1

Design an algorithm which will:

- Input a number of terms to approximate e. Your algorithm should repeat the input until the number of terms input is between 6 and 100 (including 6 and 100).
- For each term, calculate the value of 1/x!, storing it in an array.
- After all of the individual terms have been calculated, calculate the final value of e, storing it in the last element of the array.
- Upon completion of all the calculations, the algorithm should output each value in the array to the user.

Convert this algorithm into a complete Java **program**.

7. Estimation of π

Create a subdirectory in this week `P05` directory and call it **`PI`**.

Write a pseudo code algorithm that approximate the number pi (π) to a certain level of precision. This can be calculated as follows:

$$\pi = \sum_{k=0}^{num} \frac{\sin(\theta * (2k+1))}{2k+1}$$

and can be calculated as a series of:

$$\frac{\sin(\theta * (2(0)+1))}{2(0)+1} + \frac{\sin(\theta * (2(1)+1))}{2(1)+1} + \frac{\sin(\theta * (2(2)+1))}{2(2)+1} + \dots$$

Where **`num`** is the number of iterations to calculate.

Your algorithm should perform the following steps:

- Input a number of terms to approximate π. Your algorithm should repeat the input until the number of values input is between 20 and 100 (including 20 and 100).
- Input a number θ between 0 and 360. This value will then need to be converted to radians, you may assume that a function called **`radians()`** exists which takes in a real value (θ) and converts it to its radian equivalent.
- Calculate the value of each term, storing it in an array.
- After **all** of the individual terms have been calculated, calculate the final value of π, storing it in the last element of the array.
- Upon completion of all the calculations, the algorithm should output each value in the array to the user.

Convert this algorithm into a complete Java **program**.

Note: This Math API may be useful to you:

- **`double r = Math.radians(double degrees);`**
  Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
- **`double s = Math.sin(double radians);`**
  Returns the trigonometric sine of an angle in radians.

8. Permutations

Create a subdirectory in this week `P05` directory and call it **`Permutations`**.

Write a pseudo code algorithm that determines the number of permutations (without repetition) when selecting a set number of elements from a set. This can be calculated as follows:

$$^{n}P_{r} = \frac{n!}{(n-r)!}$$

Where **`n`** is the number of elements in the set and **`r`** is the number of elements you want to select.
Your algorithm should perform the following steps:

- Prompt the user for the number of elements in the set (**`n`**), your algorithm should loop until this number is between 5 and 50 (inclusive).
- Prompt the user for the maximum number of elements that should be selected (**`rMax`**), an integer value between 2 and **`n`** (inclusive);
- Create an array of size (**`rMax - 1`**).
- Calculate the number of permutations, for each **`r`** value in the range 2 to **`rMax`** (inclusive). The result of each calculation should be stored in the array.
- Once all **`r`** values have been generated output the array to the user.

For Example:

```
INPUT                OUTPUT
n=5, rMax=4          [20, 60, 120]
n=6, rMax=3          [30, 120]
n=8, rMax=8          [56, 336, 1680, 6720, 20160, 40320, 40320]
```

Convert this algorithm into a complete Java **program**.

9. Assignment Task: Convolutions

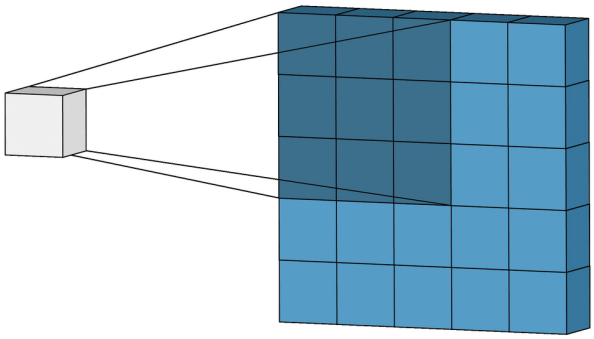
Your last task for this practical is to complete the following task **in your own time**.

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

Create a subdirectory in this week's `P05` directory and call it **`AssignmentTask`** your files should be called **`Convolutions.txt`** and **`Convolutions.java`** respectivley.

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel "slides" over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

This graphic is an easy way to understand how a Convolution operation would work on a (5 x 5) Matrix with a (3 x 3) Kernel, converted to a (3 x 3) Matrix.



If it doesn't play click here.

> **Note:** The reason why this Convolution is so important, is that it allows us to compare and contrast one side of the Kernel to the other for each operation. For example, If we were convoluting an image of grayscale values, with a Kernel of:
>
> $$K = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$
>
> It would allow us to detect the transition from a "light" side (some high value, representing colour) to a "dark" side (some low value, representing colour). Therefore showing us a line or edge in the resulting image.
>
> AI/Image Processing, combined with Machine Learning can then use this to "understand" and image and detect objects within it.

Below is an exaggerated example of how this can detect a line. The left of the **`Convolute.MATRIX_A`** is considered very bright (a high value) and the right is dark (a low value). Using a vertically biased Kernel (**`Kernel.VERTICAL`**), you are able to produce a line in the **`resultArray`**.

`Convolute.MATRIX_A`

	0	1	2	3	4	5
0	10	10	10	0	0	0
1	10	10	10	0	0	0
2	10	10	10	0	0	0
3	10	10	10	0	0	0
4	10	10	10	0	0	0
5	10	10	10	0	0	0

(6 x 6)

$*$

`Kernel.VERTICAL`

	0	1	2
0	1	0	-1
1	1	0	-1
2	1	0	-1

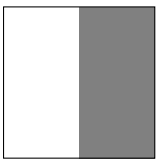
(3 x 3)

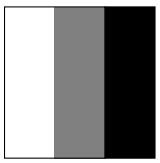
$=$

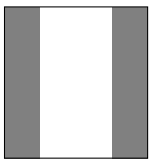
`resultArray`

	0	1	2	3
0	0	30	30	0
1	0	30	30	0
2	0	30	30	0
3	0	30	30	0

(4 x 4)







Your task is to take a Kernel (k x k) (located in **`Kernel.java`** supplied on Blackboard) and perform a convolution operation on a 2D Matrix (n x m) (located in **`Convolute.java`** supplied on Blackboard) to produce a $(n-k+1)$ x $(m-k+1)$ sized Matrix. The calculation for element $(0,0)$ if the Kernel is a (3 x 3) Matrix (K) and the image is a (n x m) Matrix (C) in the array would be:

$$(C_{0,0} * K_{0,0}) + (C_{0,1} * K_{0,1}) + (C_{0,2} * K_{0,2})$$
$$+(C_{1,0} * K_{1,0}) + (C_{1,1} * K_{1,1}) + (C_{1,2} * K_{1,2})$$
$$+(C_{2,0} * K_{2,0}) + (C_{2,1} * K_{2,1}) + (C_{2,2} * K_{2,2})$$

The general expression of a convolution is:

$$g(n,m) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} (x_{(n-i)(m-j)} * \omega_{(i)(j)})$$

where $x(n,m)$ is the original image, $\omega(k,k)$ is the filter kernel and $g((n-k+1),(m-k+1))$ is the filtered image.

Think about how we might use loops and submodules to calculate this. Make sure you are using a Modular approach, where each task that is needed is broken up into

Once you have performed a Convolution on a (n x m) Matrix you are required to print out each of the Matricies to the screen (Convolute, Kernel and Result).

In the later weeks we will learn how to read in a **`.png`** image and perform this operation on an array that is (>500,>500) so ensure that you do not hardcode anything for this task (including the Kernel size).

You are to perform operations with **`Kernel.VERTICAL`** and **`Kernel.HORIZONTAL`** on **each** of the Matrices in **`Convolute.java`**. You may also provide your own matrices and test those aswell.

It is a good idea to hand test your algorithm and hand test the output from each Convolution, to ensure that you are accurate.

A sample output of the Convolutions above: (Remember you will need to perform it on each one in the file)

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

```
Convolute:
---------------------------
| 10 | 10 | 10 | 0 | 0 | 0 |
---------------------------
| 10 | 10 | 10 | 0 | 0 | 0 |
---------------------------
| 10 | 10 | 10 | 0 | 0 | 0 |
---------------------------
| 10 | 10 | 10 | 0 | 0 | 0 |
---------------------------
| 10 | 10 | 10 | 0 | 0 | 0 |
---------------------------
| 10 | 10 | 10 | 0 | 0 | 0 |
---------------------------
Kernel:
--------------
| 1 | 0 | -1 |
--------------
| 1 | 0 | -1 |
--------------
| 1 | 0 | -1 |
--------------
Result:
-------------------
| 0 | 30 | 30 | 0 |
-------------------
| 0 | 30 | 30 | 0 |
-------------------
| 0 | 30 | 30 | 0 |
-------------------
| 0 | 30 | 30 | 0 |
-------------------
```

As always, ensure that you test this algorithm *thoroughly* (**`Convolutions_Test.txt`**) to ensure that you haven't missed any edge cases.

End of Worksheet