Challenge:	***#5***
Marks:	5% of module marks
Title:	**Explore Arrays**
Objectives:	• Explore arrays. • Explore array parameter passing to methods and return from methods. • Use **javadoc** to document each method.
Support:	Support will be available at the Week 10 and 11 laboratories. You should first complete the course work in the Coursework 5 worksheet before completing the challenge below.
Submission:	Submit to Sulis Assignments by Fri Week 12, 18 Dec, 23:00 (no late extension) Submit only the **ExploreExp.java** to Sulis. **Ensure that an appropriate file header comment is included in the java source file with: a short class description, name(s), id number(s), last date of modification.** Include also in the file header comment an example of the program output when executed, the answers to the questions in part k.
Notes:	*All module handouts and laboratory/challenge work should be maintained in an accessible file storage device.* *The work completed should be available in a folder named* ***Challenge5****.* ***Reminder: Maintain regular backups of all your work.***

1. *Exercise: arrays and methods (complete one solution per group)*

Euler's number, **e = 2.718 281 828 459 045 235 360**... is an important constant and it is the base of the natural logarithm **ln** x or **log$_e$** x.

e^x or **exp(x), the exponential function,** can be calculated from the Taylor series (aka Maclaurin):

$$e^x \text{ or } \mathbf{exp}(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

e^x can be approximated using the finite series: $\sum_{n=0}^{nMax} \frac{x^n}{n!}$ resulting in $nMax + 1$ terms.

The problem below requires you to be familiar with arrays, methods, repetition statements, and Javadoc comments.

a) Create a new class **ExploreExp**.

b) Create a **main** method and print the value of **Math.E** and then print the value of **e^1** by calling the method Math.exp(1.0). Print both to 18 decimal places e.g.:

```
System.out.printf( "Math.E is %20.18f\n", Math.E );
System.out.printf( "Math.exp(1.0) is %20.18f\n", Math.exp( 1.0 ) );
```

c) Calculate and display the partial sums for **e^1**: i.e. **e^x** with **x**=**1,** for **nMax = 0 to 20**:.

using the finite series to approximate e^x : $\sum_{n=0}^{nMax} \frac{x^n}{n!}$.

Complete in the **main** method, use a **for** statement and display the calculated **exp** value with a **precision** of 18 decimal places: use **%20.18f** in the format string of a **System.out.printf** method.

You can use your **factorial()** method and **power()** methods from your **MyMath** class to calculate **n!** and **x^n** by copying both methods into the **ExploreExp** class (or you can calculate the terms otherwise e.g. incrementally to improve the efficiency). If using your **factorial()** method, modify to calculate and return using **double** which is precise to **22!** and will calculate to **170!**

Table of partial sum estimates for e^1 where nMax = 0 to 20	nMax	Taylor's series exp(1)
	0	1.000000000000000000
	1	2.000000000000000000
	2	2.500000000000000000
	3	2.666666666666666500
	4	2.708333333333333000
		...
	20	2.718281828459045500

d) Based on the code in **c)** above (do not remove the existing code), create a separate **calcExp** method,

public static double calcExp(double x, int nMax)

calcExp accepts the exponent **x** to raise **e** to, and **nMax** the maximum value for n, as parameters and ***returns*** the estimate of **e^x**. The **calcExp** method must not call or print out anything. Include a **javadoc** comment.

e) Implement a single fully documented test case (***Name, Purpose, Input Parameter, Expected Return***) in the **main** method to test the **calcExp** method, and print whether the test passed or failed.

f) In the **main** method, use another **for** statement to calculate the first 21 partial sum estimates for **e^1** using **calcExp** and store in an array **expEstimates**. Create the array as follows:

double[ ] expEstimates = new double[21];

g) Print a table of estimates from the array. Use, or otherwise, the **printTable** method from the course work class **MyExploreArrays** (in part j), call **printTable("exp(1) estimates", expEstimates);**

h) Based on the code in **f)** above, create a method with a **javadoc** comment that generates and returns an array of the **exp** estimates:

public static double[] genExpEstimates (double x, int nEstimates)

genExpEstimates has two parameters: the exponent **x** and **nEstimates** the number of partial sum estimates, and ***returns*** the values calculated in an array.

The **genExpEstimates** method must not print out anything.

i) Print a table for **e^2** using **printTable("exp(2) estimates", genExpEstimates(2.0, 11));**

j) **Optional:** Print a table showing the error difference between using **Math.exp(x)** and **calcExp(x, 10)** for **x** between **0** and **100** in steps of **5**.

You decide how best to present the error.

k) **Include in the file header comment** an example of the program output and the answers to the following Q's:

i. How many digits versus precise **e** are **Math.E**, **Math.exp(1.0),** and **calcExp(1, 20)** accurate to?
ii. Which value of **nMax** in the **part c)** table first best matches the value of **Math.exp(1.0)**?
iii. What is the absolute error (difference) between **Math.exp(2.0)** and **calcExp(2.0, 10)**?