



# CSA 1017

## Data Structures and Algorithms 1

### Assignment

matrices Mark Said Camilleri  
B.Sc. (Hons) (Computing Science)  
University of Malta

Friday 4th May 2016

# Contents

	Page
<b>Statement of Completion</b>	<b>2</b>
<b>1 Arabic to Roman Numeral Converter</b>	<b>3</b>
1.1 The source code for Task 1 . . . . .	4
<b>2 Reverse Polish Notation Evaluator</b>	<b>6</b>
2.1 The source code for Task 2 . . . . .	7
2.1.1 Stack Class . . . . .	7
2.1.2 Question 2 Main Class . . . . .	8
<b>3 Prime Numbers</b>	<b>12</b>
3.1 Prime Number Checker using divisibility check . . . . .	12
3.1.1 The source code for 3.1 . . . . .	13
3.2 Prime Number Checker using The Sieve of Eratosthenes . . . . .	15
3.2.1 The source code for 3.2 . . . . .	16
<b>4 Shell Sort</b>	<b>19</b>
4.1 The source code for Task 4 . . . . .	20
<b>5 Approximation of a Square Root</b>	<b>23</b>
5.1 The source code for Task 5 . . . . .	24
<b>6 Matrix Multiplication</b>	<b>26</b>
6.1 The source code for Task 6 . . . . .	27
<b>7 Maximum Number in an Array</b>	<b>30</b>
7.1 The source code for Task 7 . . . . .	33
<b>8 Sine or Cosine Approximation</b>	<b>35</b>
8.1 The source code for Task 8 . . . . .	36
<b>9</b>	<b>40</b>

# Statement of Completion

The questions below were the ones that have been attempted:

**Question 1** This question has been successfully completed.

**Question 2** This question has been successfully completed.

**Question 3** This question has been successfully completed.

**Question 4** This question has been successfully completed.

**Question 5** This question has been successfully completed.

**Question 6** This question has been successfully completed.

**Question 7** This question has been successfully completed.

**Question 8** This question has been successfully completed.

**Question 9** This question has been successfully completed.

---

Signature

---

Date

# Task 1

## Arabic to Roman Numeral Converter

Input	Expected Output	Actual Output
1	I	I
2	II	II
4	IV	IV
5	V	V
6	VI	IV
9	IX	IX
10	X	X
20	XX	XX
40	XL	XL
49	XLIX	XLIX
50	L	L
60	LX	LX
90	XC	XC
99	XCIX	XCIX
100	C	C
150	CL	CL
400	CD	CD
499	CDXCIX	CDXCIX
500	D	D
600	DC	DC
900	CM	CM
999	CMXCIX	CMXCIX
1000	M	M
1024	MXXIV	MXXIV
2000	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:
abc	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:

## 1.1 The source code for Task 1

```
1  /**
2   * @author Mark Said Camilleri
3   * @version 20160509
4   */
5
6
7  import java.util.InputMismatchException;
8  import java.util.Scanner;
9
10
11 public class Question1 {
12
13     public static void main(String args[]) {
14
15         //Initialize Scanner object
16         Scanner in = new Scanner(System.in);
17         in.useDelimiter("\n");
18
19         /*===== WELCOME MESSAGE TO USER
20         =====*/
21         System.out.println("
22         -----");
23         System.out.println("|      CSA 1017 - Data Structures and
24 Algorithms 1      |");
25         System.out.println("
26         |-----|");
27         System.out.println("|      Submission by Mark Said Camilleri
28         |");
29         System.out.println("|      Task 1: Arabic to Roman Numeral
30 Converter      |");
31         System.out.println("
32         |-----|");
33         System.out.print("| Please enter a number between 1 and 1024: ")
34 ;
35
36         int toConvert = 0; //value to be converted.
37         boolean isError; // temporary boolean value used for error
38         checking of the input.
39         do {
40             isError = false;
41             try {
42                 toConvert = in.nextInt();
43             } catch (InputMismatchException e) {
44                 isError = true;
45                 in.next(); //To clear the buffer
46             }
47             /*===== Makes sure input is a number is between 1 and
48 1024 =====*/
49             if (isError || toConvert < 1 || toConvert > 1024) {
50                 /*===== OUTPUT ERROR MESSAGE TO THE USER
51                 =====*/
52                 System.out.println("
53                 -----|");
54                 System.out.println("|The input was not a valid number
```

```
        between 1 and 1024|");
44         System.out.print("|Please try again and enter a number
to convert: ");
45     }
46     } while (isError || toConvert < 1 || toConvert > 1024);
47
48     System.out.printf("| %4d = %-24s in Roman Numerals |", toConvert
, convert(toConvert));
49
50 }
51
52 /**
53  * Takes an int decimal value and outputs a string of the same value
in Roman Numerals.
54  *
55  * @param toConvert the decimal value to be converted to Roman
Numerals
56  * @return The roman numeral equivalent of the input parameter
57  */
58 private static String convert(int toConvert) {
59
60     //Defining the decimal and roman counterparts
61     final int dec[] = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500,
900, 1000};
62     final String rom[] = {"I", "IV", "V", "IX", "X", "XL", "L", "XC"
, "C", "CD", "D", "CM", "M"};
63
64     /* Begins by checking the input parameter against the largest
roman numeral/numeral pair.
65     * and works its way down to the unit numeral.
66     */
67     for (int i = dec.length - 1; i >= 0; i--) {
68         //If the value is larger, then the output is concatenated
with the output of the difference.
69         if (toConvert >= dec[i])
70             return rom[i] + convert(toConvert - dec[i]);
71     }
72     return ""; //What to return at 0, the base case.
73 }
74 }
```

## Task 2

# Reverse Polish Notation Evaluator

Input	Expected Output	Actual Output
4 5 +	9.0	9.0
12 6 -	6.0	6.0
3 2 /	1.5	1.5
7 2 *	14.0	14.0
23 85 + 92 *	9936.0	9936.0
43.5 3.2 - 4.5 * 3.24 +	184.59	184.59
matrices 34.8 62.11 * -76 /	-28.43984211	-28.439842105263157
282 -56 * 1024 - -55.6 *	934969.6	934969.6
Test	Your expression contained invalid characters. For input string "T" Your expression is invalid. Evaluation failed	Your expression contained invalid characters. For input string "T" Your expression is invalid. Evaluation failed
1 +	Stack is Empty. Your expression is invalid. Evaluation failed.	Stack is Empty. Your expression is invalid. Evaluation failed.
3 64 6 +	The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed.	The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed.
4 0 /	Infinity	Infinity

## 2.1 The source code for Task 2

### 2.1.1 Stack Class

```
1 import java.util.ArrayList;
2 import java.util.Collection;
3 import java.util.EmptyStackException;
4
5 /**
6  * Created by mark on 14/02/16.
7  *
8  * A stack implemented as an ArrayList to have it dynamically increase
9  * its size.
10 */
11 public class Stack<E> extends ArrayList {
12
13     /**
14      * Default constructor. Calls the ArrayList default constructor
15      */
16     public Stack() {
17         super();
18     }
19
20     /**
21      * Initialised a stack with the contents of the Collection in the
22      * parameter.
23      *
24      * @param c the contents to initialise the stack with.
25      */
26     public Stack(Collection<? extends E> c) {
27         super(c);
28     }
29
30     /**
31      * Pushes the data onto the stack
32      *
33      * @param data data to be pushed on the stack
34      * @throws IndexOutOfBoundsException if not successfully pushed
35      */
36     public void push(E data) throws IndexOutOfBoundsException {
37         int prevSize = this.size();
38         this.add(data);
39
40         //Condition to check if the data has been successfully added.
41         if (!(this.size() == prevSize + 1))
42             throw new IndexOutOfBoundsException("Failed to push to stack
43 ");
44     }
45
46     /**
47      * Pops the topmost item from the stack.
48      * @return the data from the top of the stack is not empty.
49      * @throws EmptyStackException if the stack is empty.
50      */
51     public E pop() throws ArrayIndexOutOfBoundsException {
52         if (this.size() == 0) throw new EmptyStackException();
53     }
54 }
```



```
50         else return (E) this.remove(this.size() - 1);
51     }
52
53     /**
54     * Returns the data at the top of the stack without popping it.
55     * @return the data if the stack is not empty. null if it is empty.
56     */
57     public E peek() {
58         if (this.size() == 0) return null;
59         else return (E) this.get(this.size() - 1);
60     }
61
62     /**
63     * Returns a string representation of the contents of the stack.
64     * @overrides toString() in class AbstractCollection<E>
65     * @return The string representation of the ArrayList if not empty.
66     * "Stack is empty" if it is empty.
67     */
68     public String toString() {
69         if (this.size() == 0) return "Stack is empty";
70         else return super.toString();
71     }
72 }
```

## 2.1.2 Question 2 Main Class

```
1 import java.util.EmptyStackException;
2 import java.util.Scanner;
3
4 /**
5  * Created by mark on 09/02/16.
6  * This answer assumes the RPN input is correct.
7  */
8 public class Question2 {
9     public static void main(String args[]) {
10
11         //Initialize Scanner object
12         Scanner in = new Scanner(System.in).useDelimiter("\n");
13
14         /*===== WELCOME MESSAGE TO USER
15         =====*/
16         System.out.println("
17         -----");
18         System.out.println("|      CSA 1017 - Data Structures and
19         Algorithms 1      |");
20         System.out.println("
21         |-----|");
22         System.out.println("|      Submission by Mark Said Camilleri
23         |");
24         System.out.println("|      Task 2: Reverse Polish Notation
25         evaluator      |");
26         System.out.println("
27         |-----|");
28         System.out.println("|      Note: This program can only do +,-,*
29         and /      |");
30         System.out.print("| Please enter an expression to evaluate: ");
31         //prompt for user input. Assumes it is rpn.
```

```
23
24     /* Initialized a stack object (using the stack defined here).
25      * Note, no importing of the Stack class.
26      */
27     Stack<Double> nums = new Stack<>();
28
29     //Reads user input. Must be a valid RPN expression.
30     String expression = in.next();
31     System.out.println("
|-----|");
32     System.out.println("| Contents of the stack at each step:
|"); //some message to user.
33
34     boolean exceptionRaised = false; //used for error checking.
35     try {
36         //Iterates through the string inputted by the user.
37         for (int i = 0; i < expression.length(); i++) {
38             char cChar = expression.charAt(i);
39
40             /* If the current character is a space,
41              * nothing needs to be done.
42              */
43             if (Character.isWhitespace(cChar)) continue;
44
45             /* If it's a '+', then 2 numbers are popped, added and
46              * the answer is pushed onto the stack.
47              */
48             else if (cChar == '+') {
49                 double num1 = nums.pop();
50                 double num2 = nums.pop();
51
52                 nums.push(num2 + num1);
53             }
54             /* If it's a '-', then 2 numbers are popped,
55              * subtracted and the answer is pushed onto the stack.
56              * The conjunction is to make sure that it's not
57              * detecting a negative number. The disjunction is
58              * true if the '-' is at the end of the string or
59              * there is a space after it. These both make sure
60              * that the '-' does not belong to a negative number
61              */
62             else if (cChar == '-' && (i == expression.length() - 1
|| Character.isWhitespace(expression.charAt(i + 1)))) {
63                 double num1 = nums.pop();
64                 double num2 = nums.pop();
65
66                 nums.push(num2 - num1);
67             }
68             /* If it's a '*', then 2 numbers are popped,
69              * multiplied and the answer is pushed onto the stack.
70              */
71             else if (cChar == '*') {
72                 double num1 = nums.pop();
73                 double num2 = nums.pop();
74
75                 nums.push(num2 * num1);
76             }
77             /* If it's a '/', then 2 numbers are popped,
```

```

78         * divided and the answer is pushed onto the stack.
79     */
80     else if (cChar == '/') {
81         double num1 = nums.pop();
82         double num2 = nums.pop();
83
84         nums.push(num2 / num1);
85     }
86     /*
87     * Otherwise, assuming it's inputted correctly, the
88     * character must be a number. In which case it is
89     * converted to a double (allowing for any real number
90     * to be inputted) and pushed onto the stack.
91     */
92     else {
93         int start = i++;
94         while (Character.isDigit(expression.charAt(i)) ||
95 expression.charAt(i) == '.')
96             i++;
97
98         nums.push(Double.parseDouble(expression.substring(
99 start, i)));
100     }
101     System.out.printf("| %-49s |\n", nums.toString());
102 }
103 if (nums.size() != 1) {
104     System.out.println("| The stack has not been emptied.
105 There are too    |");
106     System.out.println("| many operands in your expression.
107 |");
108     exceptionRaised = true;
109 }
110 } catch (IndexOutOfBoundsException ioobe) {
111     System.out.printf("| %-49s |\n", ioobe.getMessage());
112     exceptionRaised = true;
113 } catch (EmptyStackException ese) {
114     System.out.println("| Stack is Empty.
115 |");
116     exceptionRaised = true;
117 } catch (NumberFormatException nfe) {
118     System.out.println("| Your expression contained invalid
119 characters.    |");
120     System.out.printf("| %-49s |\n", nfe.getMessage());
121     exceptionRaised = true;
122 } finally {
123     if (exceptionRaised) {
124         System.out.println("| Your expression is invalid.
125 Evaluation failed.    |");
126         System.out.println("
127 ");
128         System.exit(1);
129     }
130 }
131 }
132 /*When the above iteration is complete, there should only be
133 * one item on the stack which is the answer.
134 */

```

```
127         System.out.println("
|-----|");
128         System.out.printf("| Answer of Evaluation = %-26s |\n", nums.pop
().toString());
129         System.out.println("

");
130     }
131 }
```

# Task 3

## Prime Numbers

### 3.1 Prime Number Checker using divisibility check

Input	Expected Output	Actual Output
1	1 is not a prime number	1 is not a prime number
2	2 is a prime number	2 is a prime number
3	3 is a prime number	3 is a prime number
4	4 is not a prime number	4 is not a prime number
-2	-2 is not a prime number	-2 is not a prime number
a 677	677 is a prime number	677 is a prime number
34939	34939 is a prime number	34939 is a prime number
188737204	188737204 is not a prime number	188737204 is not a prime number
381165334	381165334 is not a prime number	381165334 is not a prime number
947396057	947396057 is a prime number	947396057 is a prime number
9223372036854775807	9223372036854775807 is not a prime number	9223372036854775807 is not a prime number
-9223372036854775808	-9223372036854775808 is not a prime number	-9223372036854775808 is not a prime number
9223372036854775808	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.
Test	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.

### 3.1.1 The source code for 3.1

```

1 import java.util.InputMismatchException;
2 import java.util.Scanner;
3
4 /**
5  * @author Mark Said Camilleri
6  * @version 20160511
7  */
8
9 public class Question3a {
10
11     public static void main(String args[]) {
12
13         //Initialize Scanner object
14         Scanner in = new Scanner(System.in).useDelimiter("\n");
15
16         /*===== WELCOME MESSAGE TO USER
17         =====*/
18         System.out.println("
19         -----");
20         System.out.println("|      CSA 1017 - Data Structures and
21 Algorithms 1      |");
22         System.out.println("
23         -----|");
24         System.out.println("|      Submission by Mark Said Camilleri
25 |");
26         System.out.println("|      Task 3.1: Prime Number Checker
27 |");
28         System.out.println("
29         -----|");
30         System.out.print("| Please enter an integer to check: ");
31
32         try {
33             //The number to check whether or not it's prime
34             long prime = in.nextLong();
35
36             System.out.println("
37             -----|");
38             if (isPrime(prime))
39                 System.out.printf("| %31d is a prime number |\n", prime)
40 ;
41             else
42                 System.out.printf("| %27d is not a prime number |\n",
43 prime);
44
45             System.out.println("
46
47 ");
48         } catch (InputMismatchException ime) {
49             System.out.println("| Your input was not accepted. Please
50 restart the      |");
51             System.out.println("| program and try again with a valid
52 input.          |");
53             System.out.println("
54
55 ");
56             System.exit(1);
57

```

```
41     }
42 }
43
44 /**
45  * This method checks the input parameter to see if it is a prime
46  * number or not.
47  *
48  * @param prime The number to check if it is prime
49  * @return true if the number is prime, false if it isn't
50  */
51 private static boolean isPrime(long prime) {
52     /*Firstly, if the number is 1, 0 or negative then it's not
53     * prime
54     */
55     if (prime <= 1) return false;
56     //If the number is 2, 2 is prime.
57     else if (prime == 2) return true;
58     //If the number is even and not 2, then it's not prime.
59     else if (prime % 2 == 0) return false;
60     /* Otherwise, divide this number by all odd numbers till the
61     * square root of the number. If one is divisible then it's
62     * not prime.
63     */
64     else {
65         double root = Math.sqrt(prime);
66         for (int i = 3; i <= root; i += 2) {
67             if (prime % i == 0) return false;
68         }
69         return true;
70     }
71 }
72 }
```

## 3.2 Prime Number Checker using The Sieve of Eratosthenes

For this an `int` variable had to be used, due to it being the length of an array. As a result the range of numbers that can be checked was smaller.

Input	Expected Output	Actual Output
1	1 is not a prime number	1 is not a prime number
2	2 is a prime number	2 is a prime number
3	3 is a prime number	3 is a prime number
4	4 is not a prime number	4 is not a prime number
-2	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input
677	677 is a prime number	677 is a prime number
34939	34939 is a prime number	34939 is a prime number
188737204	188737204 is not a prime number	188737204 is not a prime number
381165334	381165334 is not a prime number	381165334 is not a prime number
947396057	947396057 is a prime number	947396057 is a prime number
2147483646	2147483646 is not a prime number	Requested array size exceeds VM limit Please restart the program and try again with a smaller input <sup>1</sup>
-2147483646	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input	Sieve of Eratosthenes only works on +ve integers. Please restart the program and try again with a valid input
9223372036854775807	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.
Test	Your input was not accepted. Please restart the program and try again with a valid input.	Your input was not accepted. Please restart the program and try again with a valid input.

<sup>1</sup>This error depends on the amount of memory allocated for the program



### 3.2.1 The source code for 3.2

```

1 import java.util.Arrays;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4
5 /**
6  * @author Mark Said Camilleri
7  * @version 20160511
8  */
9 public class Question3b {
10
11     public static void main(String args[]) {
12
13         //Initialize Scanner object
14         Scanner in = new Scanner(System.in).useDelimiter("\n");
15
16         /*===== WELCOME MESSAGE TO USER
17         =====*/
18         System.out.println("
19         -----");
20         System.out.println("|      CSA 1017 - Data Structures and
21 Algorithms 1      |");
22         System.out.println("
23         -----");
24         System.out.println("|      Submission by Mark Said Camilleri
25         |");
26         System.out.println("|      Task 3.2: Sieve of Eratosthenes
27         |");
28         System.out.println("
29         -----");
30         System.out.print("| Please enter a positive integer to check: ")
31 ;
32         try {
33             /*
34              * Value to check if it's prime. Note that this is now an int
35              * since the Sieve of Eratosthenes requires an array of this
36              * amount of elements.
37              */
38             int prime = in.nextInt();
39
40             System.out.println("
41             -----");
42             if (sieve(prime))
43                 System.out.printf("| %31d is a prime number |\n", prime)
44 ;
45             else
46                 System.out.printf("| %27d is not a prime number |\n",
47 prime);
48             System.out.println("
49
50 ");
51         } catch (InputMismatchException ime) {
52             System.out.println("| Your input was not accepted. Please
53 restart the      |");
54             System.out.println("| program and try again with a valid
55 input.          |");
56         }
57     }
58 }

```

```

41         System.out.println("
42     ");
43         System.exit(1);
44     } catch (IllegalArgumentException iae) {
45         System.out.printf("| %-49s |\n", iae.getMessage());
46         System.out.println("| Please restart the program and try
again with a |");
47         System.out.println("| valid input.
|");
48         System.out.println("
49     ");
50         System.exit(1);
51     } catch (OutOfMemoryError oome) {
52         System.out.printf("| %-49s |\n", oome.getMessage());
53         System.out.println("| Please restart the program and try
again with a |");
54         System.out.println("| smaller input.
|");
55         System.out.println("
56     ");
57         System.exit(1);
58     }
59 }
60
61 /**
62  * Runs the Sieve of Eratosthenes algorithm to check if the input
63  * is a prime number or not.
64  *
65  * @param prime The number to check if it is prime or not.
66  * @return true if the input is prime, false if it isn't.
67  * @throws IllegalArgumentException if the input is <= 0
68  */
69 private static boolean sieve(int prime) throws
IllegalArgumentException, OutOfMemoryError {
70     //Checks whether the input is valid or not
71     if (prime <= 0)
72         //Throws exception if input is not valid.
73         throw new IllegalArgumentException("Sieve of Eratosthenes
only works on +ve integers.");
74     else {
75         //Each ith element is true if i-1 is prime. False otherwise.
76         boolean[] nos = new boolean[prime];
77
78         //We begin by assuming all the values are prime.
79         Arrays.fill(nos, true);
80
81         //Then, 1 is crossed out since it is a square.
82         nos[0] = false;
83
84         /*
85          * After which each value is checked. i is assumed to be prime
86          * unless marked as not prime. Multiples of i are not prime.
87          * Therefore all the multiples of i are marked as not prime.
88          * This keeps going on until the inputted value is set to not
89          * prime or reached and still set to prime.

```

```
87         */
88         for (int i = 2; i <= prime; i++) {
89             //Checks if i is marked as prime.
90             if (nos[i - 1]) {
91                 //If it is, all multiples of i are marked as not
prime
92                 for (int j = i + i; j <= prime; j += i) {
93                     nos[j - 1] = false;
94                 }
95             }
96         /*
97          * If the inputted value is marked as prime, we can stop
98          * there since we found out whether it's prime or not.
99          * Otherwise, this program keeps going on till the end
100          * (i = prime).
101          */
102         if (!nos[prime - 1]) break;
103     }
104
105     return nos[prime - 1];
106 }
107 }
108 }
```

# Task 4

## Shell Sort

Since the requirement of this task was for the program to generate an array containing 16,384 elements, all of which are random numbers, and sort them, it is impossible to write the contents of the array for a reliable amount of tests and keep this document to a reasonable size. As a result, the program itself checks whether the array that has been sorted, is in fact sorted. This is done using the algorithm `checkSortedAscending` as given in Algorithm 1 (also included in the source code).

```
1 Algorithm: Check if sorted in ascending order
2 matrices Input: int array
   Output: boolean
3 for  $i = 0 \dots \text{length of array} - 1$  do
4   | if element at  $i$  > element at  $i+1$  then
5   | | return false
6   | end if
7 end for
8 return true
```

**Algorithm 1:** `checkSortedAscending(int[] array)`

## 4.1 The source code for Task 4

```

1 import java.util.Arrays;
2
3 /**
4  * @author Mark Said Camilleri
5  * @version 20160516
6  */
7 public class Question4 {
8
9     public static final int SIZE = 16384;
10
11     public static void main(String[] args) {
12
13         /*===== WELCOME MESSAGE TO USER
14         =====*/
15         System.out.println("
16         -----");
17         System.out.println("|      CSA 1017 - Data Structures and
18         Algorithms 1      |");
19         System.out.println("
20         |-----|");
21         System.out.println("|      Submission by Mark Said Camilleri
22         |");
23         System.out.println("|      Task 4: Shell Sorting Algorithm
24         |");
25         System.out.println("
26         |-----|");
27         System.out.println("| The array to be sorted is:
28         |");
29
30         //The array, which will eventually be sort
31         int[] arr = new int[SIZE];
32
33         //Populating the array with random data.
34         for (int i = 0; i < arr.length; i++) {
35             arr[i] = (int) (Math.random() * SIZE);
36         }
37
38         //Displays the unsorted array to the user.
39         System.out.println("Unsorted Array: " + Arrays.toString(arr));
40
41         //Sorts the array
42         int[] sorted = shellSort(arr);
43
44         //Displays the sorted array to the user
45         System.out.println("Sorted Array:  " + Arrays.toString(sorted))
46         ;
47
48         //Displays whether the array is indeed sorted or not.
49         System.out.println("Array is sorted: " + checkSortedAscending(
50         sorted));
51     }
52
53     /**
54     * Sorts the inputted array using the shell sort algorithm.
55     */

```

```

46     * @param unsorted The array to be sorted
47     * @return The sorted version of the array
48     */
49     private static int[] shellSort(int[] unsorted) {
50
51         /*
52          * First, the program begins by making a deep copy of the
53          * array to make sure the original one isn't affected.
54          */
55         int[] sorted = Arrays.copyOf(unsorted, SIZE);
56
57         /*
58          * This loop begins with the iterator being set at half the
59          * size of the array, - 1. This then iterates downwards by
60          * incrementing the iterator, dividing it by 2 and subtracting
61          * 1 from it, until it performs its final iteration at a
62          * value of 1.
63          */
64         for (int sep = (unsorted.length / 2) - 1; sep > 0; sep = (sep +
1) / 2 - 1) {
65             /*
66              * This for loop goes through the items between sep and
67              * the end of the array. To then use an bu sort
68              * algorithm to switch it
69              */
70             for (int i = sep; i < SIZE; i++) {
71                 /*
72                  * The algorithm begins by storing the value in a
73                  * temporary variable.
74                  */
75                 int temp = sorted[i];
76
77                 //iterator for next for loop. Used also outside loop.
78                 int j;
79                 /*
80                  * Finds good position of temp in steps of sep, sort
81                  * of like insertion sort.
82                  */
83                 for (j = i; j >= sep && sorted[j - sep] > temp; j -= sep
) {
84                     /*
85                      * Moves the (j-sep)th by a step of sep to
86                      * leave space for temp at its proper place.
87                      */
88                     sorted[j] = sorted[j - sep];
89                 }
90                 //Once found temp is stored in its proper place.
91                 sorted[j] = temp;
92             }
93         }
94         return sorted;
95     }
96 }
97
98 /**
99  * Method that checks whether the elements in the array in the
100  * parameter are all in ascending order.
101  */

```

```
102     * @param array The array to be checked.
103     * @return true if all elements are in ascending order, false
otherwise.
104     */
105     public static boolean checkSortedAscending(int[] array) {
106
107         /* Iterates through all elements except for the last one.
108         * If an unsorted element is found for loop stops.
109         */
110         for (int i = 0; i < array.length - 1; i++) {
111             /*
112             * If the ith element is larger than the i-1th element
113             * then the array is not sorted. Method ends returning
114             * false
115             */
116             if (array[i] > array[i + 1])
117                 return false;
118         }
119         /* If the condition is never satisfied, then the method returns
120         * true.
121         */
122         return true;
123     }
124 }
```

# Task 5

## Approximation of a Square Root

For this task, the Newton-Raphson algorithm was used, with the formula shown in Equation 5.2, which is basically a specific case of the general formula for the Newton-Raphson algorithm which is given in Equation 5.1.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (5.1)$$

$$x_{n+1} = x_n - \frac{x_n^2 - \text{input}}{2x_n} \quad (5.2)$$

	Input	Expected Output	Actual Output
	1	1	1
	2	1.414213562	1.414213562373095
	4	2	2
	100	10	10
	1000	31.6227766	31.622776601683793
	9999	99.99499987	99.99499987499375
	29535083406040468068234	$1.718577418 \times 10^{11}$	$1.7185774176929147 \times 10^{11}$
	45.3	6.730527468	6.730527468185536
	5.6	2.366431913	2.3664319132398464
	0.25	0.5	0.5
	0.125	0.3535533906	0.35355339059327373
	-4	Your input was not accepted since it was $\geq 0$ Please enter a POSITIVE number to find its square root:	Your input was not accepted since it was $\geq 0$ Please enter a POSITIVE number to find its square root:
	0	Your input was not accepted since it was $\geq 0$ Please enter a POSITIVE number to find its square root:	Your input was not accepted since it was $\geq 0$ Please enter a POSITIVE number to find its square root:
	Test	Your input was not accepted. Please try again:	Your input was not accepted. Please try again:



## 5.1 The source code for Task 5

```
1 import java.util.InputMismatchException;
2 import java.util.Scanner;
3
4 /**
5  * @author Mark Said Camilleri
6  * @version 20160516
7  */
8 public class Question5 {
9     public static void main(String[] args) {
10
11         /*===== WELCOME USER
12         =====*/
13         System.out.println("
14         -----");
15         System.out.println("|      CSA 1017 - Data Structures and
16         Algorithms 1      |");
17         System.out.println("
18         |-----|");
19         System.out.println("|      Submission by Mark Said Camilleri
20         |");
21         System.out.println("|      Task 5: Approximation of a Square
22         Root      |");
23         System.out.println("
24         |-----|");
25         System.out.print("| Please enter a positive number to find its
26         square root: ");
27
28         //variable to state whether input should be accepted or not.
29         boolean isWrong = false;
30
31         //input variable. This shall be x1 and the first value of xn,
32         double input = 1;
33         do {
34             try {
35                 //accpets user input.
36                 input = new Scanner(System.in).nextDouble();
37
38                 /*
39                  * If the value is not positive, then it will raise an
40                  * error
41                  */
42                 if (input <= 0) {
43                     isWrong = true;
44                     System.out.println("| Your input was not accepted
45                     since it was \u2264 0");
46                     System.out.print("| Please enter a POSITIVE number
47                     to find its square root: ");
48                 }
49
50                 //It will also raise an error if it's not a number.
51             } catch (InputMismatchException imme) {
52                 isWrong = true;
53                 System.out.print("| Your input was not accepted. Please
54                 try again: ");
55             }
56         }
```

```
45         //This is repeated until a correct value was inputted.
46     } while (isWrong);
47
48     //Setting the first value for x_n
49     double curr_x = Math.random() * input + 1;
50     //Initializing x_(n-1)
51     double prev_x;
52
53     //Counts the number of steps taken.
54     int steps = 0;
55
56     do {
57         steps++;
58         prev_x = curr_x;
59
60         //Newton-Raphson formula on y=sqrt(x)
61         curr_x = prev_x - (Math.pow(prev_x, 2) - input) / (2 *
prev_x);
62         /*
63          * This will keep on, either until there has been no change
64          * or until it has done Integer.MAX_VALUE number of steps.
65          */
66     }
67     while (curr_x != prev_x && curr_x != Double.NaN && steps <
Integer.MAX_VALUE);
68
69     System.out.println("
|-----|");
70
71     /*
72     * If the value is not the same as the sqrt computed by the
73     * built-in sqrt method, then it's approximately equal to,
74     * which is denoted by \u2248.
75     */
76     char sign = (curr_x == Math.sqrt(input)) ? '=' : '\u2248';
77
78     System.out.println("The square root of " + input + " " + sign +
" " + curr_x);
79     System.out.println("This was done in " + steps + " steps");
80 }
81 }
```

# Task 6

## Matrix Multiplication

The methods used in this programme were created to work for any  $n \times n$  matrices, with  $n$  being specified by a constant in the program called `SIZE`. Since writing the tests for  $32 \times 32$  matrices in this document would be too cumbersome,  $3 \times 3$  matrices shall be tested instead.

It is worth noting that the matrices used to test the output are the matrices outputted by the program. Since the program rounds each matrix (matrices **A**, **B** and **AB**) to 2 d.p. (and the expected output is also being rounded to 2 d.p) the expected output and the actual output will not be exactly the same, but they should be approximately equal to each other.

A	B	Expected AB	Actual AB
$\begin{bmatrix} 6.72 & 1.99 & 1.73 \\ 3.85 & 0.00 & 3.82 \\ 2.02 & 8.81 & -2.13 \end{bmatrix}$	$\begin{bmatrix} -1.92 & 2.21 & -2.42 \\ -0.18 & -5.64 & 0.03 \\ -0.70 & -3.86 & -6.60 \end{bmatrix}$	$\begin{bmatrix} -14.47 & -3.05 & -27.62 \\ -10.07 & -6.24 & -34.53 \\ -3.97 & -37.00 & 9.43 \end{bmatrix}$	$\begin{bmatrix} -14.48 & -3.01 & -27.66 \\ -10.07 & -6.24 & -34.60 \\ -3.96 & -36.99 & 9.43 \end{bmatrix}$
$\begin{bmatrix} 0.98 & 8.13 & -1.21 \\ 2.36 & -3.08 & -3.51 \\ 9.54 & -7.90 & -8.48 \end{bmatrix}$	$\begin{bmatrix} 3.67 & -0.20 & -7.32 \\ -4.25 & -7.69 & 9.94 \\ 8.91 & 3.91 & -0.75 \end{bmatrix}$	$\begin{bmatrix} -41.74 & -67.45 & 74.55 \\ -9.52 & 9.49 & -45.26 \\ -6.97 & 25.69 & -142.00 \end{bmatrix}$	$\begin{bmatrix} -41.75 & -67.48 & 74.53 \\ -9.47 & 9.52 & -45.27 \\ -6.97 & 25.64 & -141.96 \end{bmatrix}$
$\begin{bmatrix} -5.62 & 0.99 & 4.84 \\ 3.69 & 2.17 & 9.55 \\ -5.38 & -3.36 & -6.33 \end{bmatrix}$	$\begin{bmatrix} -6.78 & 6.98 & 3.41 \\ 8.55 & 3.79 & -1.39 \\ -8.44 & -1.03 & 4.98 \end{bmatrix}$	$\begin{bmatrix} 5.72 & -40.46 & 3.56 \\ 25.36 & 76.56 & 77.40 \\ 49.00 & 137.66 & 149.57 \end{bmatrix}$	$\begin{bmatrix} 5.79 & -40.48 & 3.51 \\ -87.04 & 24.11 & 57.07 \\ 61.22 & -43.76 & -45.17 \end{bmatrix}$
$\begin{bmatrix} -9.31 & 8.19 & 2.23 \\ -2.87 & 7.49 & 1.03 \\ 9.20 & -5.98 & 7.96 \end{bmatrix}$	$\begin{bmatrix} -1.38 & -4.24 & -7.42 \\ 0.23 & 1.28 & 9.99 \\ 8.33 & 5.48 & 6.01 \end{bmatrix}$	$\begin{bmatrix} 33.31 & 62.18 & 164.30 \\ 14.26 & 27.40 & 102.31 \\ 52.24 & -3.04 & -80.16 \end{bmatrix}$	$\begin{bmatrix} 33.29 & 62.25 & 164.22 \\ 14.25 & 27.42 & 102.29 \\ 52.21 & -3.09 & -80.15 \end{bmatrix}$
$\begin{bmatrix} -6.14 & 7.85 & 6.00 \\ -8.71 & 3.36 & -1.96 \\ -4.86 & 8.48 & 4.84 \end{bmatrix}$	$\begin{bmatrix} -0.47 & -2.13 & 3.90 \\ -4.35 & -5.06 & 6.65 \\ -3.94 & -2.29 & -8.39 \end{bmatrix}$	$\begin{bmatrix} -54.90 & -40.38 & -22.08 \\ -2.80 & 6.04 & 4.82 \\ -53.67 & -43.64 & -3.17 \end{bmatrix}$	$\begin{bmatrix} -54.91 & -40.38 & -22.12 \\ -2.77 & 6.07 & 4.74 \\ -53.71 & -43.67 & -3.19 \end{bmatrix}$
$\begin{bmatrix} -4.13 & 4.79 & -8.23 \\ -6.02 & 0.78 & -1.33 \\ -5.98 & -2.59 & 5.39 \end{bmatrix}$	$\begin{bmatrix} 6.72 & 5.18 & 0.65 \\ 7.19 & 6.84 & 2.82 \\ 6.72 & 8.95 & -2.87 \end{bmatrix}$	$\begin{bmatrix} -48.62 & -62.29 & 34.44 \\ -43.78 & -37.75 & 2.10 \\ -22.59 & -0.45 & -26.66 \end{bmatrix}$	$\begin{bmatrix} -48.66 & -62.29 & 34.42 \\ -43.76 & -37.74 & 2.07 \\ -22.59 & -0.51 & -26.67 \end{bmatrix}$

## 6.1 The source code for Task 6

```

1  /**
2   * @author Mark Said Camilleri
3   * @version 20160517
4   */
5  public class Question6 {
6
7      /**
8       * Stores the size of the matrices
9       */
10     public static final int SIZE = 32;
11
12     public static void main(String[] args) {
13
14         /*===== WELCOME USER
15         =====*/
16         System.out.println("
17         -----");
18         System.out.println("|      CSA 1017 - Data Structures and
19         Algorithms 1      |");
20         System.out.println("
21         |-----|");
22         System.out.println("|              Submission by Mark Said Camilleri
23         |");
24         System.out.println("|              Task 6: Matrix Multiplication
25         |");
26         System.out.println("
27         |-----|");
28
29         /**
30          * The 2 matrices. Matrix A will be multiplied with matrix B
31          * to form matrix AB.
32          */
33         double[][] matrixA = new double[SIZE][SIZE];
34         double[][] matrixB = new double[SIZE][SIZE];
35
36         //Put random values from 1 to 10 in matrixA
37         for (int i = 0; i < SIZE; i++) {
38             for (int j = 0; j < SIZE; j++) {
39                 matrixA[i][j] = Math.random() * 20 - 10;
40             }
41         }
42
43         //Put random values from 1 to 10 in matrixB
44         for (int i = 0; i < SIZE; i++) {
45             for (int j = 0; j < SIZE; j++) {
46                 matrixB[i][j] = Math.random() * 20 - 10;
47             }
48         }
49
50         //Multiplied matrix
51         double[][] result = new double[SIZE][SIZE];
52
53         /**
54          * Transposes MatrixB. This is done to make the multiplication

```

```
49     * easier since the method would accept 2 arrays, one for the
50     * row and the other for the column.
51     */
52     double[][] matBtranspose = matTranspose(matrixB);
53
54     /*
55     * Multiplies each row of Matrix A with each column of Matrix B
56     * and adds them up to get the value of each cell.
57     */
58
59     for (int i = 0; i < SIZE; i++) {
60         for (int j = 0; j < SIZE; j++) {
61             result[i][j] = cellMult(matrixA[i], matBtranspose[j]);
62         }
63     }
64     //Outputs Matrix A
65     System.out.print("Matrix A = ");
66     displayMatrix(matrixA);
67
68
69     System.out.println();
70     System.out.print("Matrix B = ");
71     displayMatrix(matrixB);
72
73     System.out.println();
74     System.out.print("  A * B =  ");
75     displayMatrix(result);
76
77 }
78
79 /**
80  * Performs a matrix multiplication for 1 cell only.
81  *
82  * @param row The row of the first matrix to be multiplied
83  * @param col The column of the second matrix to be multiplied
84  * @return The value of the cell that resides in both the row and
column that were inputted, for the resultant matrix.
85  */
86 public static double cellMult(double[] row, double[] col) {
87     //Result to be returned when finished.
88     double result = 0;
89
90     for (int i = 0; i < SIZE; i++) {
91         result += (row[i] * col[i]);
92     }
93
94     return result;
95 }
96
97 /**
98  * Transposes a square matrix of n x n size
99  *
100  * @param matrix The matrix to be transposed
101  * @return matrix transposed
102  */
103 public static double[][] matTranspose(double[][] matrix) {
104
105     double[][] transposed = new double[matrix.length][matrix[0].
```

```
length];
106
107     for (int i = 0; i < transposed.length; i++) {
108         for (int j = 0; j < transposed[0].length; j++) {
109             transposed[i][j] = matrix[j][i];
110         }
111     }
112
113     return transposed;
114 }
115
116 /**
117  * Displays the matrix to the user.
118  *
119  * @param matrix The matrix to be displayed to the user.
120  */
121 public static void displayMatrix(double[][] matrix) {
122     for (int i = 0; i < matrix.length; i++) {
123
124         for (int j = 0; j < matrix[i].length; j++) {
125             System.out.printf("%7.2f ", matrix[i][j]);
126         }
127         System.out.println();
128         System.out.print("          ");
129     }
130 }
131 }
```

# Task 7

## Maximum Number in an Array

### Test 1

Input 1

Expected Output 1

Actual Output 1

### Test 2

Input 1 1

Expected Output 1

Actual Output 1

### Test 3

Input 1 2

Expected Output 2

Actual Output 2

### Test 4

Input 2 1

Expected Output 2

Actual Output 2

### Test 5

Input 1 2 1

Expected Output 2

Actual Output 2

**Test 6**

**Input** 1 1 1 1 1

**Expected Output** 1

**Actual Output** 1

**Test 7**

**Input** 1 2 3 4 5 6

**Expected Output** 6

**Actual Output** 6

**Test 8**

**Input** 2 0 -3 10 3

**Expected Output** 10

**Actual Output** 10

**Test 9**

**Input** -9 -10 -5 -2 5

**Expected Output** 5

**Actual Output** 5

**Test 10**

**Input** -70210333 620267078 832297266 -242136666 -759786803 89498046 390942259  
-616830953 707423547 190501876

**Expected Output** 832297266

**Actual Output** 832297266

**Test 11**

**Input** 680932039 798137906 1362068 -482386095 976166649 54797596 264771471  
906347353 -674021294 -491689938

**Expected Output** 976166649

**Actual Output** 976166649



**Test 12**

**Input** 463847872 970322047 994450928 475574164 639319683 -942622068 -319105910  
-453518765 -167026456 579529276

**Expected Output** 994450928

**Actual Output** 994450928

**Test 13**

**Input** 1 2 3 4 e

**Expected Output** There was an error processing your input. Please try again.

**Actual Output** There was an error processing your input. Please try again.

**Test 14**

**Input** Test

**Expected Output** There was an error processing your input. Please try again.

**Actual Output** There was an error processing your input. Please try again.

## 7.1 The source code for Task 7

```

1 import java.util.ArrayList;
2 import java.util.Arrays;
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 /**
7  * @author Mark Said Camilleri
8  * @version 20160517
9  */
10 public class Question7 {
11     public static void main(String[] args) {
12
13         /*===== WELCOME USER
14         =====*/
15         System.out.println("
16         -----");
17         System.out.println("|      CSA 1017 - Data Structures and
18         Algorithms 1      |");
19         System.out.println("
20         |-----|");
21         System.out.println("|              Submission by Mark Said Camilleri
22         |");
23         System.out.println("|              Task 7: Largest Number
24         |");
25         System.out.println("
26         |-----|");
27
28         Scanner in = new Scanner(System.in);
29
30         //This will store the numbers before formatted as numbers.
31         String[] text_nos;
32         int[] numbers = {0};
33
34         /* This is to ask the user to input the numbers again if the
35         * user has provided us with a bad input.
36         */
37         boolean inputIsCorrect;
38         do {
39             try {
40                 inputIsCorrect = true;
41
42                 //Ask user for input.
43                 System.out.println("| Please enter a list of integers.
44                 Seperate them |");
45                 System.out.println("| with a space. Press Enter at the
46                 end of the list: |");
47
48                 //Reads input and splits it to string array.
49                 text_nos = in.nextLine().split("\\s");
50
51                 //Creates int array to store them.
52                 numbers = new int[text_nos.length];
53
54                 //Parses the strings in the string array to integers.
55                 for (int i = 0; i < text_nos.length; i++) {

```

```

47         numbers[i] = Integer.parseInt(text_nos[i]);
48     }
49     } catch (NumberFormatException nme) {
50
51         //Flags program to ask again for input.
52         inputIsCorrect = false;
53
54         //Tells user there was an error
55         System.out.println("
|-----|");
56         System.out.println("|       There was an error processing
your input.   |");
57         System.out.println("|                               Please try again.
|");
58         System.out.println("
|-----|");
59     }
60     } while (!inputIsCorrect);
61
62     System.out.println("
|=====|");
63     System.out.printf("| The maximum number from your list was: %10d
|\n", findMax(numbers, Integer.MIN_VALUE));
64     System.out.println("
|=====|");
65 }
66
67 /**
68  * Recursive function to find the maximum number in an array
69  *
70  * @param nums The array to return it's max value
71  * @param max Current maximum value. This should start with
72  *             Integer.MIN_VALUE.
73  * @return the maximum value.
74  */
75 public static int findMax(int[] nums, int max) {
76
77     //Base case. If array is empty.
78     if (nums.length == 0) {
79         return max;
80     } else {
81
82         /* If the element at startIndex > max, then it is the
83          * current maximum value.
84          */
85         if (nums[0] > max) {
86             max = nums[0];
87         }
88
89         /* Recursive call. Checks this max with the array starting
90          * from the next element
91          */
92         return findMax(Arrays.copyOfRange(nums, 1, nums.length), max
93     );
94 }
95 }
96 }

```

# Task 8

## Sine or Cosine Approximation

*N.B.: All these tests have been carried out with an approximation of 100 terms*

Program	Input		Output
	Mathematical	Expected	Actual
s0.5pir	$\sin(\frac{\pi}{2} \text{ rad})$	1.000000	1.000000
s270d	$\sin(270^\circ)$	-1.000000	-1.000000
c0.25pir	$\cos(\frac{\pi}{4} \text{ rad})$	0.707107	0.707107
s45d	$\sin(45^\circ)$	0.707107	0.707107
s100pid	$\sin(100\pi^\circ)$	-0.717406	-0.717406
c1r	$\cos(1 \text{ rad})$	0.540302	0.540302
c2pir	$\cos(2\pi \text{ rad})$	1.000000	1.000000
s8348pir	$\sin(8348\pi \text{ rad})$	0.000000	0.000000
1r	1 rad	Please enter the input with the function character	Please enter the input with the function character
100	100	Please enter the input with the function character	Please enter the input with the function character
test	-	Please enter the input with the function character	Please enter the input with the function character
s180	$\sin(180)$	Please enter the input with the unit character	Please enter the input with the unit character
crest	-	Please enter the input with the unit character	Please enter the input with the unit character
sd	$\sin(^{\circ})$	Your input could not be read. Please start over and input the expression again	Your input could not be read. Please start over and input the expression again

## 8.1 The source code for Task 8

```

1 import java.util.InputMismatchException;
2 import java.util.Scanner;
3
4 /**
5  * Created by mark on 25/04/16.
6  */
7 public class Question8 {
8     public static void main(String[] args) {
9         System.out.println("
10         -----");
11         System.out.println("|      CSA 1017 - Data Structures and
12         Algorithms 1      |");
13         System.out.println("
14         |-----|");
15         System.out.println("|      Submission by Mark Said Camilleri
16         |");
17         System.out.println("|      Task 8: Sine or Cosine
18         Approximation      |");
19         System.out.println("
20         |-----|");
21
22         /*===== ASK USER FOR EXPRESSION INPUT
23         =====*/
24         System.out.println("| Please enter \'s\' or \'c\' to compute the
25         sine or      |");
26         System.out.println("| cosine function respectively, followed by
27         the          |");
28         System.out.println("| value to compute it on. Use \"pi\" for pi.
29         This        |");
30         System.out.println("| should be followed by \'r\' or \'d\' for
31         radians or   |");
32         System.out.println("| or degrees respectively.
33         |");
34         System.out.println("
35         |-----|");
36         System.out.println("| For example, sin(2pi radians) is entered
37         as s2pir |");
38         System.out.println("|      or cos(180 degrees) is entered as
39         c180d      |");
40         System.out.println("
41         |-----|");
42
43         /*===== ACCEPT EXPRESSION INPUT
44         =====*/
45         Scanner in = new Scanner(System.in).useDelimiter("\n");
46
47         String exp = ""; //Default expression is empty string.
48
49         /*===== CHECK IF INPUT IS CORRECT
50         =====*/
51         boolean isWrong; //Used to ask user to input again
52         do {
53             isWrong = false; //Assumes input will be correct.
54             exp = in.nextLine(); //Takes user input
55
56
57

```

```

38         //Incorrect input if first character is not c or s (function
39     )
40         if (exp.charAt(0) != 's' && exp.charAt(0) != 'c') {
41             isWrong = true;
42             System.out.println("| Please enter the input with the
function character|");
43             //Incorrect input if last character is not d or r (unit)
44             } else if (exp.charAt(exp.length() - 1) != 'd' && exp.charAt
(exp.length() - 1) != 'r') {
45                 isWrong = true;
46                 System.out.println("| Please enter the input with the
unit character |");
47             } else {
48                 try {
49                     /*===== PARSE USER INPUT
=====*/
50                     /* First character determines which function to
51                     * evaluate
52                     */
53                     char fn = exp.charAt(0);
54
55                     //Last character determines the units
56                     char units = exp.charAt(exp.length() - 1);
57
58                     double angle; //Where angle is stored
59                     //Multiplies the value by pi if "pi" is inputted.
60                     if (exp.contains("pi")) {
61                         //If a number isn't present, then it's 1*PI
62                         //Throws a NumberFormatException
63                         angle = ((exp.length() == 4) ? 1 : Double.
valueOf(exp.substring(1, exp.length() - 3))) * Math.PI;
64                     } else {
65                         //Throws a NumberFormatException
66                         angle = Double.valueOf(exp.substring(1, exp.
length() - 1));
67                     }
68
69                     /*
70                     * Converts to radians since Maclaurin's expansion
71                     * works for radians
72                     */
73                     if (units == 'd') angle = Math.toRadians(angle);
74
75                     /* Reduces the angle to between -2pi and 2pi. This
76                     * makes it easier to compute. Sine and Cos are
77                     * periodic functions.
78                     */
79                     angle %= (2 * Math.PI);
80
81                     /*===== ASK USER HOW MANY TERMS TO
EVALUATE =====*/
82                     System.out.println("| Please enter the number of
terms to evaluate |");
83                     System.out.println("| the approximation of the
expression inserted |");
84
85                     //Number of terms to be evaluated for the series
86                     int n = in.nextInt();

```

```
86
87         //Sine and Cosine are local methods
88         double result = (fn == 's') ? sine(angle, n) :
cosine(angle, n);
89
90         System.out.printf("| The answer is: %-34f |", result
);
91
92         /* Catches an Exception if the input is not in the
93         * correct form.
94         */
95         } catch (NumberFormatException | InputMismatchException
except) {
96             System.out.println("| Your input could not be read.
Please start over |");
97             System.out.println("|               and input the
expression again |");
98
99             in.nextLine();
100
101             isWrong = true;
102         }
103     }
104     } while (isWrong);
105 }
106
107 /**
108  * Approximates the sine using Maclaurin's series expansion
109  *
110  * @param angle Angle in radians to calculate sin for
111  * @param terms Number of terms to calculate
112  * @return An approximation to sin(angle)
113  */
114 public static double sine(double angle, int terms) {
115     double result = 0;
116     //Adds the nth term to result
117     for (int n = 0; n <= terms; n++) {
118         result += (Math.pow(-1, n) * Math.pow(angle, 2 * n + 1)) /
fact(2 * n + 1);
119     }
120     return result;
121 }
122
123 /**
124  * Approximates the cosine using Maclaurin's series expansion
125  * @param angle Angle in radians to calculate cos for
126  * @param terms Number of terms to calculate
127  * @return An approximation to cos(angle)
128  */
129 public static double cosine(double angle, int terms) {
130     double result = 0;
131     //Adds the nth term to result
132     for (int n = 0; n <= terms; n++) {
133         result += (Math.pow(-1, n) * Math.pow(angle, 2 * n)) / fact
(2 * n);
134     }
135     return result;
136 }
```

```
137
138  /**
139   * Caluclates the factorial of n
140   * @param n number to calculate the factorial of
141   * @return n!
142   */
143 private static double fact(double n) {
144     //If n = 0 or 1, n! defaults to 1
145     double result = 1;
146     //Otherwise i is multiplied to result until n is reached.
147     for (int i = 2; i <= n; i++) {
148         result *= i;
149     }
150
151     return result;
152 }
153 }
```



## Task 9