Problem 51 Write the Java method called `sandwich` that takes as argument an array called `myArr` of integers. Assume the length of `myArr` is at least 3. If there is an index t in `myArr` such that $1 \leq t \leq$ `(myArr.length - 2)` and the value in `myArr`[t] equal to the product of the values `myArr`[t – 1] and `myArr`[t +1], then return any such index t. Otherwise, return the value –1.

```
public static int sandwich(int [] myArr)  {...}
```

Problem 52 Write the Java method called `maxOfMins` that takes a 2-dimensional array of integers called `table` and returns the maximum of the minimum values in each row. For example, in the array below, the minimum values of the rows are 20, 33, 15, and 28. Since the largest of these values 33, `maxOfMins` would return value 33 for this the array below.

	0	1	2	3	4
0	50	20	80	90	100
1	77	555	33	44	66
2	888	15	302	90	123
3	31	29	30	32	28

```
public static int maxOfMins(int [][] table)  {...}
```

Problem 53 Write the Java method `highestPow` which takes a positive integer n and returns the highest power of 2 (e.g. 1, 2, 4, 8, 16, 32, …) that goes evenly into n. Here are some examples:

`highestPow(30)` would return the value 2, since 2 goes evenly into 30 but 4 does not.
`highestPow(107)` would return the value 1, since 1 goes evenly into 107 but 2 does not.
`highestPow(36)` would return the value 4, since 4 goes evenly into 36 but 8 does not.
`highestPow(80)` would return the value 16, since 16 goes evenly into 80 but 32 does not.
`highestPow(40)` would return the value 8, since 8 goes evenly into 40 but 16 does not.

```
public static int highestPow(int n)  {...}
```

Problem 54 Write the Java method `firstSquare` which takes an integer array `a` and returns the smallest index k such that `a`[k] < k^2. If no such index, return the value -1. For example, for the array below, the value 4 would be returned since `a`[4] = 15, which is less than 4^2 = 16, and 4 is the smallest index for which this is true.

0	1	2	3	4	5	6	7
10	5	100	20	15	33	29	20

```
public static int firstSquare(int [] a)  {...}
```

Problem 55 Write a program that prompts the user to enter a positive integer *n*. If the user does not enter a positive integer, the program keeps prompting and reading user input until the user does enter a positive integer. The program then prompts the user to enter *n* integers, after which the program outputs the largest value entered, the number of odd values entered, and the sum of all values entered. Here is a sample run:

```
Enter positive integer n:  -6
Enter positive integer n:  0
Enter positive integer n:  -3857
Enter positive integer n:  5
Now enter 5 integers:  15  11  20  100  35
Largest value entered:  100
Number of odd values:  3
Sum of all values:  181
```

```
public static void main(String[] args)  {...}
```

Problem 56 Write a Java program that prompts the user to enter a positive integer *a*, and then a positive integer *d*. Assume the user enters valid input. The program then outputs the quotient of *a* divided by *d* and the remainder of *a* divided by *d* . The program then does the same thing again, and keeps doing it until the quotient and remainder are the same. Here is a sample output:

```
Enter pos. integer a:  49
Enter pos. integer d:  5
The quotient is 9 and the remainder is 4.

Enter pos. integer a:  80
Enter pos. integer d:  7
The quotient is 11 and the remainder is 3.

Enter pos. integer a:  45
Enter pos. integer d:  8
The quotient is 5 and the remainder is 5.
Bye!
```

```
public static void main(String[] args)  {...}
```

Problem 57 Write the Java method `sumOfDigits` which takes as a positive integer *n* as a parameter, and return the sum of the digits in *n*. For example, if *n* = 35042, the method would return the value 14, since 14 = 3 + 5 + 0 + 4 + 2.

```
public static int sumOfDigits(int n)  {...}
```

Problem 58 Write the Java method `innerOuter` which takes a 2-dimensional array of integers called `table`, which has the same number of rows as columns. We define the "outer sum" of such an array to be the sum of all elements in the 0th row and column and the last row and column. We define the "inner sum" to be the sum of all the other elements in the array. The method returns the larger of the array's inner sum and outer sum.

For example, the array below has an outer sum of 39 (note, the corner elements are counted only once in this sum) and an inner sum of 27, so the method would return the value 39 for this example.

	0	1	2	3	4
0	5	3	0	0	2
1	0	10	10	-5	8
2	1	0	-5	10	7
3	1	3	4	0	0
4	1	0	6	4	1

```
public static int innerOuter(int [][] table)  {...}
```

Problem 59 Write the Java method `mostSmallerValuesToLeft` which takes an array of integers called `myNums` as its parameter. There are no repeated values in `myNums`. The method returns the value in `myNums` which has the most smaller values to its left. For example, in the array below, the value 60 has 3 values to its left that are smaller than 60. Since no other value in the array has more than 3 smaller values to the left, the return value would be 60 for this example.

	0	1	2	3	4	5	6
myNums	10	50	80	70	35	60	40

```
public static int mostSmallerValuesToLeft (int [] myNums)  {...}
```

Problem 60 Write a Java program that prompts the user to enter an integer *n*. Assume the integer entered is at least 2. The program than prompts the user to enter *n* integers. The program computes the sum of these values and the product of these values. If the sum is bigger, output the word "SUM" to the console window. If the product is bigger, output the word "PRODUCT" to the console window. If they are equal to each other, output the word "TIE". Here is a sample run:

```
Enter positive integer n:  4
Enter 4 integers:  2  1  3  1
SUM
```

```
Enter positive integer n:  3
Enter 3 integers:  2  1  3
TIE
```

```
public static void main(String[] args)  {...}
```

Problem 61 Write the Java method `createArray` which takes as parameters integers *m* and *n* (both positive). The method then prompts the user to enter *n* values and reads them in. Then it prompts for another *n* values and reads them in. This happens a total of *m* times. The method returns a reference to a new 2-dimensional array that is initialized to the user's input.

The following is a sample interaction of `createArray` for *m* = 3 and *n* = 5:

```
Enter 5 integers for row 0:  40  70  0  55  100
Enter 5 integers for row 1:  10  300  22  200  9
Enter 5 integers for row 2:  999  33  444  2  8
```

The method would return a reference to a newly created 2-dimensional array initialized as follows:

	0	1	2	3	4
0	40	70	0	55	100
1	10	300	22	200	9
2	999	33	444	2	8

```
public static [][] int createArray (int m, int n)  {...}
```

Problem 62 Write a Java program that prompts the user to enter integers *n* and *k*. If both *n* and *k* are positive, the program outputs to the console window the number *n* repeated *k* times in a row, and then prompts the user for new values of *n* and *k*. Again, if *n* and *k* are both positive, the program outputs to the console window the new value of *n* repeated the new value of *k* times in a row, and then prompts the user for new values of *n* and *k*. The program continues doing this until either *n* is negative or *k* is negative, or both. Two sample runs:

```
Enter n and k:  50  4
50  50  50  50
Enter n and k:  8  10
8  8  8  8  8  8  8  8  8  8
Enter n and k:  0  3
0  0  0
Enter n and k:  3  0
Enter n and k:  -5  50
Bye!
```

```
Enter n and k:  222  3
222  222  222
Enter n and k:  5000  1
5000
Enter n and k:  60  -7
Bye!
```

```
public static void main(String[] args)  {...}
```

Problem 63 Write the Java method `rowWithMostZeros` which takes a two-dimensional array of integers called `table` and returns the index of the row which has the most zeros in it. If two or more rows tie for the most number of zeros, return the highest index of the rows that tie. For example, if the array `table` was the one given below, the method would return the value 3, since both rows 1 and 3 have three zeros, which is more than any other row, and 3 is the largest index of all rows that tied.

	0	1	2	3	4	5
0	5	3	0	0	2	2
1	0	10	0	-5	8	0
2	1	0	-5	10	7	7
3	1	3	4	0	0	0
4	1	0	6	4	1	1

```
public static int rowWithMostZeros (int [][] table)  {...}
```

Problem 64 Write the Java method `lastDigits` which takes an array of integers called `myNums` as its parameter. The method returns a reference to a newly created integer array of length 10, where cell 0 is initialized to the number of values in `myNums` whose last digit is 0, cell 1 is initialized to the number of values in `myNums` whose last digit is 1, cell 2 is initialized to the number of values in `myNums` whose last digit is 2, and so on. For example, if `myNums` were the following array:

	0	1	2	3	4	5	6	7
myNums	41	38	75	98	26	51	118	13

then the method would return a reference to a newly created array initialized as follows:

0	1	2	3	4	5	6	7	8	9
0	2	0	1	0	1	1	0	3	0

```
public static [] int lastDigits (int [] myNums)  {...}
```

Problem 65 Write a Java program that has the user enter a number of years, weeks, and days, and then outputs the number of days in that time span. For this problem, there are always 365 days in a year, and 7 days in a week. Here is a sample run:

```
Enter the number of years, weeks, and days:  2  11  6

2 years, 11 weeks, and 6 days is the same as 813 days.
```

```
public static void main(String[] args)  {...}
```

Problem 66 Write a Java program that prompts the user to continue entering integers until the user enters a negative integer. Assume that the user's input contains no repeated values, and assume that the user enters at least three values. Then output the sum of all the integers excluding the largest one entered and the smallest one entered (do not count the terminal negative value as the smallest value entered). Here is a sample run:

```
Enter a positive integer:  30
Enter a positive integer:  5
Enter a positive integer:  60
Enter a positive integer:  100
Enter a positive integer:  20
Enter a positive integer:  -999

The sum of these, not counting the lowest and highest, is 110.
```

```
public static void main(String[] args)  {...}
```

Problem 67 Write the Java method `playGuessingGame` which takes an integer *n* as its parameter. The method then plays "guess the number" using *n* as the target value: the user is prompted for a guess *g*, then method outputs to the console window "CORRECT" if $g = n$, "TRY HIGHER" if $g < n$, or "TRY LOWER" if $g > n$. The method keeps prompting the user for guesses and gives hints until the user guesses the number n. The method then outputs to the console window the number of guesses it took. Here is a sample output of `playGuessingGame` when it is passed the value $n = 43$:

```
Guess a number:  50
TRY LOWER.  Guess a number: 0
TRY HIGHER.  Guess a number: 30
TRY HIGHER.  Guess a number: 40
TRY HIGHER.  Guess a number: 45
TRY LOWER.  Guess a number: 44
TRY LOWER.  Guess a number: 43
It took you 7 guesses!
```

```
public static void playGuessingGame(int n) {...}
```

Problem 68 Write the Java method `sumTarget` which takes as parameters a two-dimensional array of integers called `table` (which has the same number of rows as columns) and an integer called `target`. The method returns the smallest index *k* such that the sum of the elements in the row and column *k* is equal to `target`. If no such row exists, the method returns the value –1. For example, if `table` were the array given below, and `target` had the value 50, `sumTarget` would return the value 2, since that is the smallest (and only in this example) index for which the elements in that row and column sum to 50. (The values in row 2 are 3, 2, 5, 0, and 20, and the other values in column 2 are 10, 1, 9, and 0. These values all sum to 50: 3 + 2 + 5 + 0 + 20 + 10 + 1 + 9 + 0 = 50).

	0	1	2	3	4
0	40	0	10	5	0
1	1	2	1	4	7
2	3	2	5	0	20
3	6	10	9	4	8
4	0	20	0	5	0

```
public static int sumTarget (int [][] table, int target)  {...}
```

Problem 69 Write the Java method `longerTF` which takes a boolean array called `guess` as its parameter. The method returns `true` if the longest run of `true` values is longer than the longest run of `false` values, otherwise the method returns `false`. A "run" of `true` values is a set of adjacent cells all of which contain the value `true`. For example, if `guess` were the array below, the length of the longest run of `true` values is 3, and the length of the longest run of `false` values is 4, so `longerTF` would return `false` for this example.

0	1	2	3	4	5	6	7	8	9
true	false	false	false	false	true	false	true	true	true

```
public static boolean longerTF (boolean [] guess)  {...}
```

Problem 70 Write an integer array method named fibNumbers that accepts one integer parameter n and creates an array containing the first n Fibonacci numbers. Each Fibonacci number is determined by adding together the previous two numbers in the sequence. The sequence starts with 0 and 1. Assume n is non-negative. Here are some examples:

If n = 3 then fibNumbers(n) returns a reference to an array containing the values {0, 1, 1, 2}

If n = 6 then fibNumbers(n) returns a reference to an array containing the values {0, 1, 1, 2, 3, 5, 8}

If n = 0 then fibNumbers(n) returns a reference to an array containing the values {0}

```
public static int[] fibNumbers(int n)  {...}
```

Problem 71 Write a method named switchColumns that accepts three parameters: a two-dimensional integer array named a and two integer values named x and y. The method makes a new array containing the values from a but has the columns x and y switched. Assume x and y are valid column indexes for a. Under no circumstances should you return a reference to the original array. The method returns a reference to the newly created array. Here are two examples:

```
Example 1:   a = {{3, 6, 7, 2},   and x = 0, y = 3  return a reference to  {{2, 6, 7, 3},
                  {1, 0, 5, 9}}                                              {9, 0, 5, 1}}

Example 2:   a = {{31,  3, 17},   and x = 2, y = 1  return a reference to  {{31, 17,  3},
                  { 7, 40,  5},                                              { 7,  5, 40},
                  {12,  8, 89}}                                              {12, 89,  8}}
```

```
public static int[][] switchColumns(int[][] a, int x, int y) {...}
```

Problem 72 Write a Java program that asks the user to enter numbers until the sum of all of the numbers that are entered is exactly 10. Then the program displays how many numbers were entered. Here are three sample runs:

```
Enter a number: 1.5
Enter a number: 3.5
Enter a number: 5
3
```

```
Enter a number: 20
Enter a number: -12.25
Enter a number: 0
Enter a number: 2.25
4
```

```
Enter a number: 9.1
Enter a number: -1.5
Enter a number: 1
Enter a number: -10
Enter a number: 0
```

```
public static void main(String[] args)  {...}
```

Problem 73 Write a method named consolidate that takes two character arrays `a` and `b` and returns a reference to a new array that consolidates the contents of `a` and b as follows: The new array first contains all of the elements from the even positions of array `a`, followed by all of the elements from the odd positions of array `b`, followed by all of the elements from the odd positions of array `a`, followed by all of the elements from the even positions of array `b`. Here are some examples:

Argument Array *a*	**Argument Array *b***	**New Array**
x f q	s d g	x q d f s g
m s t r	a c x z	m t c z s r a x
g h g h	i j	g g j h h i
x	v t w p	x t p v w
a b	c	a b c

```
public static char[] consolidate(char[] a, char[] b)  {...}
```

Problem 74 Write the Java method below which takes an integer array `a` of even length and returns the sum of the products of adjacent pairs of elements in `a` . Your method must work for any even length array—you may assume the length of array `a` is at least 2. For example, if `a` were this array:

	0	1	2	3	4	5	6	7
a	10	5	100	3	6	2	30	20

your method would compute and return the value 962, obtained by this computation:

```
a[0]*a[1] + a[2]*a[3] + a[4]*a[5] + a[6]*a[7]  =  10(5) + 100(3) + 6(2) + 30(20)
                                               =   50  +  300  +  12  +  600
                                               =   962.
```

```
public static int sumOfProducts(int [] a) {...}
```

Problem 75 Write the Java method `plusInSquare` that takes a positive integer n, which you should assume is odd. The method returns a reference to a new two-dimensional array of characters with n rows and n columns initialized as follows:

- all cells on the outer border of the array contain the character 'X',
- all cells in the middle row and the middle column contain the character 'X', and
- all other cells contain the period character: '.'

Here is an example of the array that is returned when `plusInSquare` is called with *n* = 5:

	0	1	2	3	4	5	6
0	X	X	X	X	X	X	X
1	X	.	.	X	.	.	X
2	X	.	.	X	.	.	X
3	X	X	X	X	X	X	X
4	X	.	.	X	.	.	X
5	X	.	.	X	.	.	X
6	X	X	X	X	X	X	X

```
public static char[][] plusInSquare(int n)  {...}
```