Question 1 – Multiple choice – Question-ID: 118828 (5 points)

A variable:

A Can hold values

B Has a name which represents a position in the memory

C Can change during the execution of the code

D All the other options are valid

Question 2 – Open-ended – Question-ID: 118819 (6 points)

Answer the following questions:

a. What does the following message mean and how is it generated?

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

b. Write TWO Java statements that can lead to this error. Each statement must include a different possible source for this error to be valid.

See grading criteria below

Question 3 – Hotspot: 5 hotspots – Question-ID: 118798 (5 points)

Mark the **compilation** errors in the code below:

```
public static String exam2022_2023(int a, b) {
    boolean bool;
    String c = a+b;

    if(c < a > 0)
        System.out.println("b is negative");
    else
        bool = false

    return bool;
}
```

Question 4 – Fill in (multiple) – Question-ID: 118799 (5 points)

Fill the gaps to complete a piece of code that prints all the odd numbers from `a` to `b` (both inclusive) but backwards. You must consider that both numbers are positive and that `a<=b`.

```
public static void fillGaps(int a, int b) {
    for(int i=b ; i>=a ; i-- ) {
        if(i%2 !=0 )
            System.out.println( i );
    }
}
```

Question 5 – Multiple choice – Question-ID: 118826 (4 points)

What will be the outcome of the following code snippet?

```
1  String a = "1", b="2";
2
3  if(a!=b)
4      System.out.println("Both numbers are different");
5      System.out.println("Both numbers are equal");
```

A Both numbers are different

B None of the other options

C Will not compile. There is a syntax error in line 1

D Both numbers are equal

E Will not compile. There is a syntax error in line 3

Question 6 – Fill in – Question-ID: 118789 (5 points)

Which would be the array returned by the following method when called as `createArray(10)`?

```
public static int[] createArray(int numElements) {
    int[] array = new int[numElements];

    for(int i=0; i<array.length; i=i+2) {
        if(i%2!=0)
            array[i] = i;
        else
            array[i] = i+1;
    }
    return array;
}
```

Note: In the solution, use square brackets, separate the elements of the array with commas and do not leave any spaces between elements such as in this example: [1,1,1,1]

[1,0,3,0,5,0,7,0,9,0]

Question 7 – Fill in – Question-ID: 118792 (5 points)

Given the recursive method below, which is the expected output when called as `recursion(10)`?

```
public static String recursion(int a) {
    if(a<=0)
        return "0";
    else if(a%3==0)
        return a+","+recursion(a+1);
    else
        return a+","+recursion(a-2);

}
```

10,8,6,7,5,3,4,2,0

Question 8 – Open-ended – Question-ID: 118800 (15 points)

Shifting vectors
A vector (or 1D array) is just a list of elements. In this exercise, our vector contains an indeterminate number of integers in a disordered matter. You must implement a method that, given two vectors, calculates the number of shiftings to the left (see example below) needed to transform the second one (defined as `vector`) to the first (also called `target`). If this is not possible, the method must return -1. Some examples are provided below:

***Example 1**: Given `target = [0,1,2]` and `vector = [2,0,1]`, the method must return `1` since, while moving all the elements to the left just once (and making the element in position 0 the last one in the resulting vector), both vectors are equal. You can see it visually as:*

***Example 2**: Given target = [0,1,2,3] and vector = [1,2,3,0], the method must return 3 since we need three siftings (moving all the elements to the left and moving the element in position 0 to the last position AFTER EACH SHIFTING) to make both vectors equal. This time, the steps would be, visually:*

***Example 3**: Given target = [0,1,2] and vector = [3,0,1], the method must return -1 since, vector cannot be equal to target, regardless the number of shiftings performed (different elements in each vector)*

***Example 4**: Given target = [0,1,2] and vector = [2,2,0,1], the method must return -1 since, vector cannot be equal to target, regardless the number of shiftings performed (size is different).*

Use the following signature for the method to be implemented:

```
public static int shiftedEqual (int[] target, int[] vector)
```

Question 9 − Open-ended − Question-ID: 118803 (15 points)

Additions
In this exercise, the method to be implemented must calculate the minimum number of additions needed to reach a specific value from a given one. However, there are some restrictions. In order to reach the target value, only values equal to or less than the initial one can be used. See the following examples for clarification.

> ***Example 1****: Given an* `initial` *value of 3 and a* `target` *value of 7, we would need a minimum of 2 addition operations. The first one can use the initial number (3+3=6). However, for the second operation, we must use a number which is lower, 1 in this case (3**+**3**+**1=7). Therefore, the minimum number of additions is 2. Notice that, as you cannot use a number greater than the initial one, 4 cannot be used (3+4=7 would lead to 1 operation but this operation is not valid).*
>
> ***Example 2****: Given an initial value of 4 and a target of 15, the output of the method must be 3 (4**+**4**+**4**+**3=15)*

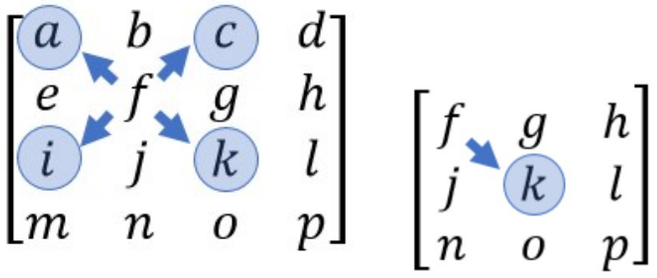
Use the following signature for the method to be implemented:

```
public static int additions(int initial, int target)
```

Question 10 − Open-ended − Question-ID: 118801 (15 points)

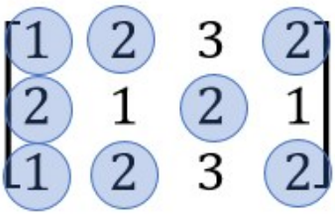
Checking diagonal
We have a program that works with 2D arrays of integers. To improve this program, we want to implement a new method able to calculate how many elements in the array meet a condition. The condition is that the element has all its neighbouring elements in the diagonal direction equal to it. Notice that each element has, at most, four neighbouring elements in its diagonals, namely left-up, left-down, right-up and right-down (see figure below). The input array can have any dimension and shape (different numbers of rows and columns may apply).

For clarification, the following images visualize the "neighbouring elements in the diagonal direction" of the element with value `f` in two different cases. In the first one, the element has four "neighbouring elements in the diagonal direction" while in the second example it has only one.

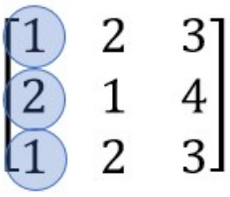


Some examples of the expected behaviour of the method are given below.

Example 1*: With the given 2D array, the method must return 6 (see the elements in a circle). Notice that, for elements in the borders, no all four diagonals exist*



Example 2*: In this second example, only the elements in the first column meet the requirement.*



Use the following signature for the method to be implemented:

```
public static int equalDiagonals(int[][] array)
```

End of test 2022-2023-KEN1120-RegularExam

Question 2 – Open-ended

Grading criteria:

- Description (maximum 2 points): The description of the error is accurate
- Operation division (maximum 2 points): One of the statements includes a division
- Operation module (maximum 2 points): One of the statements includes the calculation of the module (which implies a division)

Question 8 - Shifting vectors

```java
public static int shiftedEqual (int[] target, int[] vector) {
    int shiftings = -1;
    int numElements = target.length;
    if(numElements == vector.length) {
        for(int i=0; i<numElements; i++) {
            int[] c = new int[numElements];
            for(int j=0; j<numElements; j++) {
                if((j+i)<numElements)
                    c[j] = vector[j+i];
                else
                    c[j] = vector[(j+i)-numElements];
            }
            if(Arrays.equals(c,target))
                if(shiftings!=-1)
                    shiftings = Math.min(shiftings, (i+1));
                else
                    shiftings = i;
        }
    }
    return shiftings;
}
```

Question 9 - Additions

```
1     public static boolean validPosition(int a, int b, int aLength, int bLength) {
2         boolean result = true;
3         if(a<0 || b<0 || a>aLength-1 || b>bLength-1)
4             result = false;
5         return result;
6     }

7     public static int equalDiagonals1(int[][] array) {
8         int result = 0;
9         int width = array.length;
10        int height = array[0].length;

11        for(int i=0; i<width; i++)
12            for(int j=0; j<height; j++) {
13                int number = array[i][j];
14                if(((validPosition(i-1, j-1, width, height) && array[i-1][j-1]==number) ||
(!validPosition(i-1, j-1, width, height))) &&
15                ((validPosition(i-1, j+1, width, height) && array[i-1][j+1]==number) ||
(!validPosition(i-1, j+1, width, height))) &&
16                ((validPosition(i+1, j-1, width, height) && array[i+1][j-1]==number) ||
(!validPosition(i+1, j-1, width, height))) &&
17                ((validPosition(i+1, j+1, width, height) && array[i+1][j+1]==number) ||
(!validPosition(i+1, j+1, width, height))) )
18                    result++;
19            }
20        return result;
21    }
```

Question 10 - Checking diagonal

```
// “Expected” solution
1    public static int additions(int initial, int target) {
2        int result = 0;
3        int tmp = initial;
4        int addition = initial;
5        while(tmp<target) {
6            if((target-tmp)>=addition) {
7                tmp+=addition;
8                result++;
9            }
10           else
11               addition--;
12       }
13       return result;
14   }
```

```java
// Shorter version
1    public static int additions(int initial, int target){
2        if(target%initial == 0)
3            return ((target/initial)-1);
4        else
5            return(target/initial);
6    }
```

```java
// Recursive solution
1    public static int additions(int initial, int target){
2        // Base case
3        if( target<=0 )
4            return -1;
5        return 1 + additions(initial, target-initial);
6    }
```