Question 2: (20 points)

When a square matrix is transposed, the rows and columns are swapped, i.e., *all pairs of entries across the main diagonal* of the matrix are swapped. Write a MATLAB function **`transposeNegative`** that swaps pairs of entries across the main diagonal *only when the pair of entries are both negative*. For example, given the matrix below on the left, function **`transposeNegative`** returns the matrix on the right. The main diagonal of the matrix is shaded grey below. In this example, there are only two pairs of negative entries across the main diagonal: -9 and -3 is one pair; -13 and -4 is the other pair. Therefore these are the only two pairs of entries that are swapped by function **`transposNegative.`**

-1	2	-3	-4
5	6	-7	8
-9	10	11	12
-13	14	15	16

-1	2	-9	-13
5	6	-7	8
-3	10	11	12
-4	14	15	16

Function **`transposeNegative`** takes one square matrix **`M`** as the input argument and returns the (possibly) altered matrix **`M`**. Within the function, perform the special transpose operation described above *in-place* (do not create a new matrix). Do not use the MATLAB transpose operator (**'**).

Question 3: (20 points)

Write a MATLAB program to simulate a *random walk*. A random walk begins at a randomly generated location—random x- and y-coordinates. Each subsequent step, of length one, must be in the x- or y-direction, i.e., no diagonal steps are allowed. Therefore there are four *equally* possible choices for a step: +1 in the x-direction, -1 in the x-direction, +1 in the y-direction, or –1 in the y-direction. There are no boundaries for the walk. Your program starts by prompting the user for the number of steps to simulate. The program then generates two random integers, both in the range of -10 to 10, as the starting coordinates. After each step, display the step number and the new location. Simulate the requested number of steps.

Question 4: (15 points)

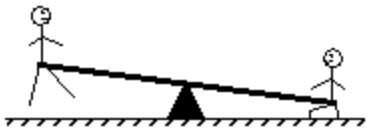
Write a Java program segment that reads a positive integer **`n`** from user input and prints asterisks (**`*`**) to form a square with **`n`** asterisks on each side with **`n`** asterisks along the main diagonal. For example, if the input value is **`5`**, the program should produce the following output:

```
*****
**   *
* * *
*  **
*****
```

Note: Your code should be general—works for *any* positive integer input. For input values of 1, 2, or 3, the printed figure will be a solid square, which is ok. *Do not use arrays*.

Question 5: (25 points)

Complete classes **`Seesaw`** and **`Person`** below. The program simulates the operation of a see-saw, as shown below. Two people push each other up and down, which causes them to burn calories. Assume the board starts in the bottom position on the right. One cycle of operation consists of the board going up and down again, which consumes **15** calories per person.



In one session, the people will play for **25** cycles, unless one or both of the people get bored, which happens about **10%** of the time. Check for boredom at the beginning of each cycle, including the beginning of the session. *Determine and display the number of calories the people burn in one session*.

Specification for class **`Person`**:

- Instance integer variables: **`calories`** is number of calories burned so far; **`calsPerCycle`** is number of calories burned per cycle
- Instance method **`wannaStop`** returns a **`boolean`** value to indicate if a **`Person`** wants to stop (with 10% probability)
- Instance method **`addCalories`** increases the number of calories burned so far by a **`Person`**
- Instance method **`getCalories`** returns the number of calories burned so far by a **`Person`**
- *Do not define any other instance or class variables/methods*

Specification for class **`Seesaw`**, a client class of **`Person`**: Class **`Seesaw`** has a single method **`main`**:

- Creates **`Person`**s to play
- Run the simulation
- Display the total number of calories burned by both people at the end of the session

For full credit, you must use an object-oriented approach with encapsulation and use all variables and methods specified above. *Do not use inheritance*. Read through both incomplete classes before you start writing.

```
/* Class Seesaw */
public class Seesaw {
  public static void main(String[] args) {
```

```
  }
} //class Seesaw
```

```
/* Class Person (Question 5, continued) */
public class Person {
```

```
} //class Person
```

Bonus Question:

Bonus points are worth *less* than core points. Work on this question *after* you have completed the five core questions and checked your work.

Question B1: (1 point) Write MATLAB code to use function **`transposeNegative`** (Question 2) on a randomly generated 4-by-4 matrix. The random floating point numbers should be in the range of -1 to 1.