Assignment #4 – Tower of Hanoi Game

1 Introduction

On this assignment, you will create a program that simulate the Tower of Hanoi game. If you are not familiar with this game, please take a look at the following sites:

https://www.cs.sfu.ca/~tamaras/recursion/Rules_Towers_Hanoi.html.

Your program will basically behave like the online game found at https://www.mathsisfun.com/games/towerofhanoi.html, however we will not do the GUI version (yet). The movements will be requests from the terminal window instead of mouse dragging and drop.

As shown in the online version of this game:

- the player can choose how many disks they want to play with.
- All the disks are in the 1st pole at the beginning of the game
- All the disks must be in the 3rd pole to win the game
- The number of movements is monitored and shown in the screen
- The minimum number of moves is calculated based on the number of disks

You will implement all the features above and few more in this assignment.

2 Implementation Design

Think about objects. What are the objects of this game???

- Disks are objects
- Poles are objects
- The whole game (with disks and poles) is an object too

Each of those objects will be created from their respective classes (blue prints), i.e., there will be three classes in your program: Disk, Pole, and Hanoi. Details about each of them later on.

Besides these classes, you will have the main class, which will play the game (the Hanoi object), by creating the Hanoi game with a given number of disks, and then requesting to move disks from one pole to another.

3 Class-by-Class Specification

In this Section we will discuss each individual class mentioned in the previous section in enough detail for you to implement them.

3.1 The Disk Class

3.1.1 Disk Class Fields

```
private int size;
private char shape;  // where shape can be hardcoded to '*'
```

3.1.2 Disk Constructor

The Disk class shall have one non-default constructor that accepts its size (such as 3, or 5, etc.). Assume that the size provided by whoever is calling the constructor is valid. Therefore, no validation is necessary from your constructor.

3.1.3 Disk Methods

3.1.3.1 The getSize() method

Method Name:	getSize()
Method input:	none
Method output:	an integer corresponding to the size of the given disk (such as 3, or 5, etc.)
Method functionality:	nothing much, just get the size defined during the disk creation (constructor)

3.1.3.2 The toString() method

Method Name:	toString()
Method input:	none
Method output:	a String containing the "drawing" representation of the given disk. Example: if the disk has size 7, this method shall return 7 asteriscs (along with a new line at the end). "*******"
Method functionality:	Use StringBuilder and a FOR loop to create the required output

3.2 The Pole Class

3.2.1 Pole Class Fields

```
private final int maxDiskCapacity;     // max number of disks
private final String poleChar = "|";   // character representing the pole
private int currentNumberOfDisks;      // current number of disks
private final Disk[] disks;            // array of the current Disks
                                       // in the pole
private final int poleWidth;           // pole width
private final int poleHeight;          // pole height
```

3.2.2 Pole Constructor

The Pole class shall have one non-default constructor that accepts the maximum number of disks (maxDiskCapacity). It is not how many disks it has in a given time, but it is the maximum number of disks that it can support. Example, for a Hanoi of 5 disks, the maxDiskCapacity should be 5.

Inside the constructor you shall:

- Set the class field maxDiskCapacity
- Set the currentNumberOfDisks to zero (you will add disks later on)
- Allocate memory for the disks objects (the 1D array of Disks) based on the max number of disks allowed
- Set the poleHeight as 1 unit higher that the number of disks so we can always see the tip of the pole even if all the disks are in that pole
- Set the poleWidth based on the following description:

  The smallest disk size should be 3 (three asterisks). Each disk shall have an even width, i.e., if the game has three disks, the disks will have the following sizes: 3, 5, and 7. The pole width shall be slightly wider than the largest disk (just one unit in the left and right size of the biggest disk). Therefore, in this case, the pole width should be set to 9.

3.2.3 Pole Methods

3.2.3.1 The getDisks() Method

Method Name:	getDisks()
Method input:	none
Method output:	returns an array of Disks objects found in the pole. The disk at the bottom of the pole is the first disk in the array. This array of disks changes over time.
Method functionality:	Nothing much to say here

3.2.3.2 The getNumberOfDisks() Method

Method Name:	getNumberOfDisks()
Method input:	none
Method output:	returns the current number of disks in this pole. This number changes over time.
Method functionality:	Nothing much to say here

3.2.3.3 The addDisk() Method

Method Name:	addDisk()
Method input:	the Disk object to be added to the pole
Method output:	returns a boolean value that represents if the disk was added or not to the pole. True if the disk was successfully added to the pole, otherwise false.
Method functionality:	Before adding the Disk to the pole, this method shall check if the size of this pole is smaller than the size of the topmost disk. If disk is successfully added to the pole, the variables currentNumberOfDisks and the disks array shall be updated accordingly. If the disk cannot be moved to this pole, this method prints a meaningful message in the terminal window.

3.2.3.4 The peekTopDisk () Method

Method Name:	peekTopDisk ()
Method input:	None
Method output:	returns a Disk object that represents the topmost disk in the given pole, or null if the pole currently has no disks.
Method functionality:	Based on the number of current disks and the Disk[] array, this method shall return the top most Disk object. Hint: this method is handy when adding disks to a given pole.

3.2.3.5 The removeDisk () Method

Method Name:	removeDisk ()
Method input:	None
Method output:	None
Method functionality:	Based on the current number of disks and the Disk[] array, this method shall set the top most Disk object to null in that array, what mimics the disk being removed from the pole. Do not forget to update the current number of disks variable accordingly.

3.2.3.6 The toString() Method

Method Name:	toString()
Method input:	None
Method output:	Visual representation of the current pole state in the form of a String.
Method functionality:	Based on the current number of disks and the Disk[] array, pole height, etc, this method shall create a String representation of the pole. The output of this method will be used later by the Hanoi object to print the poles in the terminal window. The pole shall be represented by the symbol "|", the disks by the symbol "*", and the base with "=" Note that the base is 1 unit larger than the maximum disk in the game in each side. Use StringBuilder to create the String representation of this pole. Do an interne search on StringBuilder in Java for details on how to use it. Examples:

```
    |
   ***
  *****
 *******
=========
Hanoi of 3 disks
```

```
     |
    ***
   *****
  *******
 *********
***********
 *********
  *******
   *****
    ***
     |
=================
Hanoi of 7 disks
```

3.3 The Tower of Hanoi Class

The Tower of Hanoi class will be used to create the actual game (hardware). The game has three poles, a given number of disks, and some methods to move disks here and there.

3.3.1 The Tower of Hanoi Class Fields

```
private final Pole[] poles = new Pole[3];
private final int maxNumberOfDisks;
```

3.3.2 The TowerOfHanoi Constructor

Method Name: TowerOfHanoi ()
Method input: Maximum number of disks.
Method functionality: The constructor shall initialize the class field maxNumberOfDisks, create the poles based on the maximum number of disks, initialize the game by placing all the disks in the 1st pole, and printing the game (the three poles) as shown below.

```
public TowerOfHanoi(int maxNumberOfDisks) {
    this.maxNumberOfDisks = maxNumberOfDisks;
    createPoles(maxNumberOfDisks);
    initializeGame();
    printGame();
}
```

3.3.3 The TowerOfHanoi Methods

3.3.3.1 The createPoles() Method

Method Name: createPoles()
Method input: None
Method Output: None
Method functionality: This method creates 3 Pole objects. These objects are placed in the global variable containing the array of poles (Section 3.3.1).

3.3.3.2 The initializeGame() Method

Method Name: initializeGame ()
Method input: None
Method Output: None
Method functionality: This method creates all the disks defined in the constructor and place them into the 1st pole. Remember, the smallest pole is 3 units wide and the others go up is size by 2 units. If 4 disks are defined, the disk sizes will be 3, 5, 7, and 9. Try to make a programming logic to calculate the disk sizes.

3.3.3.3 The moveDisk() Method

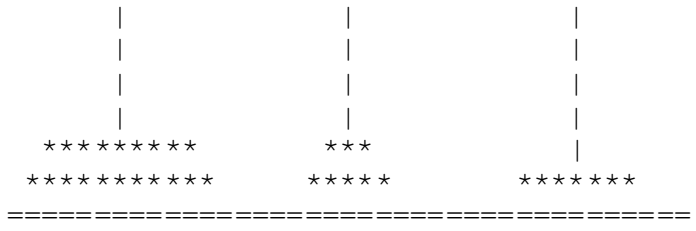
Method Name:	moveDisk()
Method input:	two integer numbers: the first designates from what pole you are moving the disk and the second designating to what pole to move the disk to.
Method Output:	None
Method functionality:	This method tries to move a disk from one pole to another by using the Pole addDisk() and removeDisk() methods. Hint: use the returning value of the Pole addDisk() method to certify that the disk could or not be moved.

3.3.3.4 The getPoles() Method

Method Name:	getPoles()
Method input:	None
Method Output:	An 1D array of Pole objects.
Method functionality:	Nothing much to say about it.

3.3.3.5 The printGameCurrentStatus() Method

Method Name:	printGameCurrentStatus ()
Method input:	None
Method Output:	None
Method functionality:	Prints the game in its current state in the terminal window. The three poles should be printed side-by-side, what will require some though on how to do it. I would consider using again a StringBuilder object to create a long string that would do a sort-of concatenation between the three pole strings (Pole toString() method).



3.4 The TowerOfHanoiGame Class

Up to Section 3.3 we have been creating the actual (hardware per say) game. Now we need a class that will interact with the hardware, asking to move disks and so on. You should not need to create a new class for this: the TowerOfHanoiGame class is your project class, whatever the name you gave to it when you have created your NetBeans project for this assignment. You do not need to rename it to TowerOfHanoiGame; use the name that you have already created.

3.4.1 TowerOfHanoiGame Class Fields

```
private int minimumNumberOfMoves;
private int currenNumberOfMoves = -1;
private int numberOfDisks;
```

(you might have more class fields, but the above ones must be there too.)

3.4.2 The TowerOfHanoiGame Constructor

Method Name: TowerOfHanoiGame()
Method input: None
Method functionality: The constructor shall be what we call in programming as the main driver. It will basically have all the necessary calls to:

- Display a welcome screen
- Ask the player how many disks they want to play with
- Create a TowerOfHanoi object with that many disks
- Display instructions on how to play
- Start the game

```
public static void main(String[] args) {
    new TowerOfHanoiPuzzle();
}

private TowerOfHanoiPuzzle() {
    displayWelcomeMessage();
    numberOfDisks = getNumberOfDisks();
    TowerOfHanoi hanoi = new TowerOfHanoi(numberOfDisks);
    displayHowToPlayInstructions();
    playGame(hanoi);
}
```

3.4.3 The TowerOfHanoiGame Methods

3.4.3.1 The TowerOfHanoiGame displayWelcomeMessage()

Method Name: displayWelcomeMessage()
Method input: None
Method Output: None
Method functionality: Prints a welcome message in the terminal window, such as the name of the game, the copyright (year), name of the developer (your name), and the famous statement "all rights reserved".

3.4.3.2 The TowerOfHanoiGame getNumberOfDisks()

Method Name: getNumberOfDisks()
Method input: None
Method Output: An integer representing the number of disks that the Tower of Hanoi will have.
Method functionality: This method uses the terminal window to request the player to provide the number of disks that the player wants to play with. Valid numbers are between 3 and 64 disks. This method must be bullet proof, i.e., it must:

- It must validate the number (from 3 up to 64)
- (Bonus) if the user enters 3a by mistake, the method should not abort
- If not valid, this method should keep asking.

3.4.3.3 The TowerOfHanoiGame displayHowToPlayInstructions()

Method Name: displayHowToPlayInstructions()
Method input: None
Method Output: None
Method functionality: This method shall display the instructions on how to play the game and also the minimum number of moves that the player would need to complete the game. This number is computed as

$$numberOfDisks^2 - 1$$ (example: for 5 disks, the number would be 24)

An example of the how to play message is given below:

```
The goal is to move all 5 disks from pole 1 to 3
The least number of moves for 5 disks is 24
Hit Enter to start
```

3.4.3.4 The TowerOfHanoiGame playGame()

Method Name: playGame()
Method input: TowerOfHanoi Object
Method Output: None
Method functionality: This method shall allow the player to play the game. Please place the following code inside this method. Your responsibility is to create the methods shown below yourself. Hanoi moveDisk() method has been discussed in Section 3.3.3.3. isGameOver() method is straight forward. The getMoveRequest() method is discussed in the next section.

```
private void playGame(TowerOfHanoi hanoi) {
    while (true) {
        int[] moveRequest = getMoveRequest();
        hanoi.moveDisk(moveRequest[0], moveRequest[1]);
        if (isGameOver(hanoi)) {
            System.out.println("Congratulations");
            System.exit(0);
        }
    }
}
```

3.4.3.5 The TowerOfHanoiGame getMoveRequest()

Method Name: getMoveRequest()
Method input: None
Method Output: An integer array containing 2 numbers: the first representing what pole the disk must be removed from and the second representing what pole the disk must be added to.
Method functionality: This method shall:

- Tells how many movements you have done so far
- Tells how to enter the requested movement
- Check if the movement is valid (this should be a method. See Section 3.4.3.6 for instructions on this method)
- (Bonus) try to prevent your program to crash if the user makes a typo, such as 3b

3.4.3.6 The TowerOfHanoiGame isMoveRequestValid()

Method Name: isMoveRequestValid ()
Method input: String Array containing a String representation of the poles from and to.
Method Output: True is the movement request is valid, otherwise false.
Method functionality: This method shall check the following things:

- There are two items inside the String array
- If the pole numbers are different
- If the pole numbers are valid (1, 2, or 3)
- If the inputs are numbers

4 Running Example

```
Welcome to the Tower of Hanoi Puzzle.
Please enter a number of disks (1 - 64): 3

    |         |         |
   ***        |         |
  *****       |         |
 *******      |         |
============================

The goal is to move all 3 disks from pole 1 to 3
The least number of moves for 3 disks is 8
Hit Enter to start
Current number of moves: 0 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 1 3

    |         |         |
    |         |         |
  *****       |         |
 *******      |        ***
============================
```

```
Current number of moves: 1 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 1 2

    |         |         |
    |         |         |
    |         |         |
 *******   *****     ***
============================

Current number of moves: 2 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 3 2

    |         |         |
    |         |         |
    |       ***       |
 *******   *****       |
============================

Current number of moves: 3 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 1 3

    |         |         |
    |         |         |
    |       ***       |
    |      *****   *******
============================

Current number of moves: 4 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 2 1

    |         |         |
    |         |         |
    |         |         |
   ***     *****   *******
============================

Current number of moves: 5 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 2 3

    |         |         |
    |         |         |
    |         |      *****
   ***        |     *******
============================

Current number of moves: 6 (goal: 8)
Enter <from><space><to> to move a disk (0 0 to quit): 1 3

    |         |         |
    |         |       ***
    |         |      *****
    |         |     *******
============================

Congratulations
```