I have now finished the challenge.
I found this to be a very peculiar challenge. It very much reminds me of the rounding population challenge in which the tweaks were strategic and served massive value.
However I think the difficulty level on this was extremely tough.

During the code, I was trying to manipulate my variables to be the same values as those expected (as part of my design phase). I consider this was the only way to remember logic, since variables seemed so arbitrary.

I managed to initially get the logic to hold for the nth term 1 and 2.
But then, it was mighty difficult.
Since I knew the code could not continue in the driver main method:
it left me using alternative techniques to shift the variable values for calculating number rows below and above the dissector.

This was extremely challenging without even using Collection.
I also suspect the difficulty was enhanced much more opting for a recursive approach to this challenge.
Also, as part of the recursive exercises I had a clear if statement (such as if variable is less than 1, return value. And stay clear of the main logic). But with this exercise, I just had no idea.

Once again on my research, it stated that there are suitable scenarios for iterative vs recursive.
I am slightly caught in two minds if there are any trade offs in the performance.
But I am very sure that if I had not created a design, it was almost close to impossible for me.

From all the coding undertaken to current date, I had to refer to design and code the most.

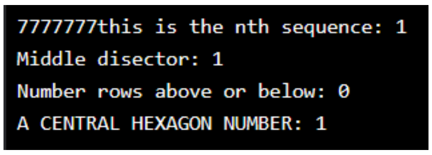
I am now ready to test this properly, although I feel I have obtained the correct onscreen values for nth term   1 => 5

Once I am satisfied that the code functions, I can utilise collections to enhance my data seeking.
I will maintain a new major version of the code.

And it would be interesting to see if there are performance differences. I suspect there will be for large centered hexagonal numbers!

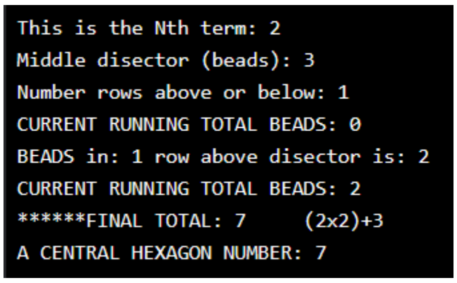
*** OUTPUT *****************

<u>TEST CASE 1:   n =1</u>



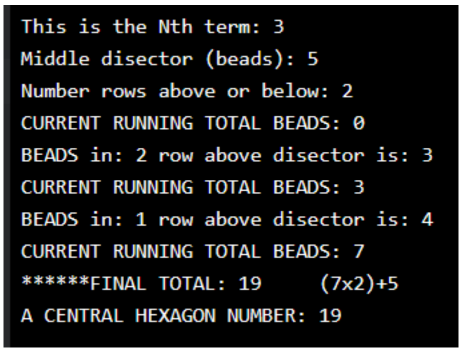
```
7777777this is the nth sequence: 1
Middle disector: 1
Number rows above or below: 0
A CENTRAL HEXAGON NUMBER: 1
```

<u>TEST CASE 2:   n =7</u>



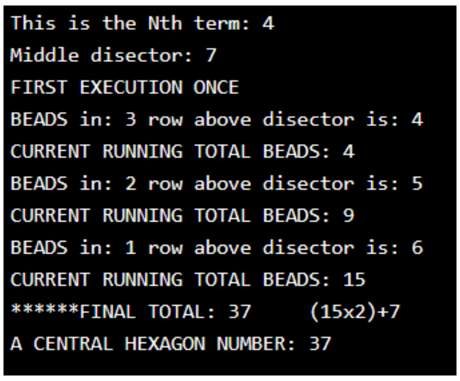
```
This is the Nth term: 2
Middle disector (beads): 3
Number rows above or below: 1
CURRENT RUNNING TOTAL BEADS: 0
BEADS in: 1 row above disector is: 2
CURRENT RUNNING TOTAL BEADS: 2
******FINAL TOTAL: 7     (2x2)+3
A CENTRAL HEXAGON NUMBER: 7
```

<u>TEST CASE 3:   n =19</u>



```
This is the Nth term: 3
Middle disector (beads): 5
Number rows above or below: 2
CURRENT RUNNING TOTAL BEADS: 0
BEADS in: 2 row above disector is: 3
CURRENT RUNNING TOTAL BEADS: 3
BEADS in: 1 row above disector is: 4
CURRENT RUNNING TOTAL BEADS: 7
******FINAL TOTAL: 19     (7x2)+5
A CENTRAL HEXAGON NUMBER: 19
```

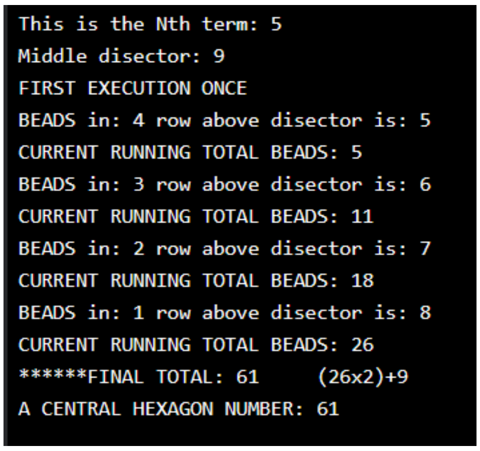
<u>TEST CASE 3:   n =37</u>



```
This is the Nth term: 4
Middle disector: 7
FIRST EXECUTION ONCE
BEADS in: 3 row above disector is: 4
CURRENT RUNNING TOTAL BEADS: 4
BEADS in: 2 row above disector is: 5
CURRENT RUNNING TOTAL BEADS: 9
BEADS in: 1 row above disector is: 6
CURRENT RUNNING TOTAL BEADS: 15
******FINAL TOTAL: 37     (15x2)+7
A CENTRAL HEXAGON NUMBER: 37
```

As described above, the code does not return to the Main class.
The above are all the test cases presented.
I will now try the next nth term 5. I am hoping to get 61 beads...

<u>TEST CASE 4: n =61</u>

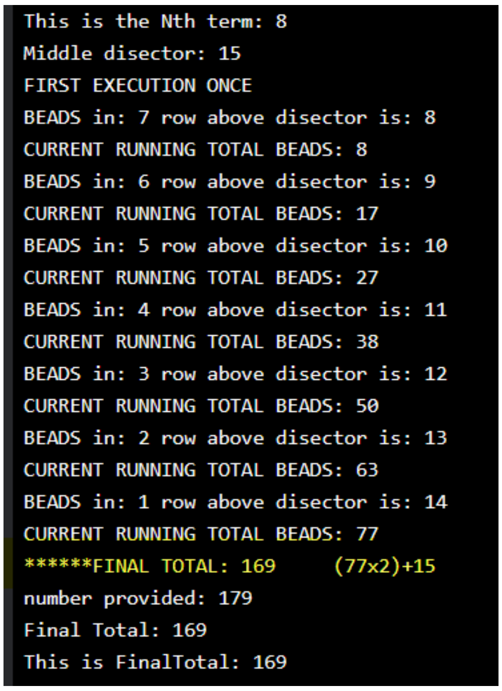


```
This is the Nth term: 5
Middle disector: 9
FIRST EXECUTION ONCE
BEADS in: 4 row above disector is: 5
CURRENT RUNNING TOTAL BEADS: 5
BEADS in: 3 row above disector is: 6
CURRENT RUNNING TOTAL BEADS: 11
BEADS in: 2 row above disector is: 7
CURRENT RUNNING TOTAL BEADS: 18
BEADS in: 1 row above disector is: 8
CURRENT RUNNING TOTAL BEADS: 26
******FINAL TOTAL: 61     (26x2)+9
A CENTRAL HEXAGON NUMBER: 61
```

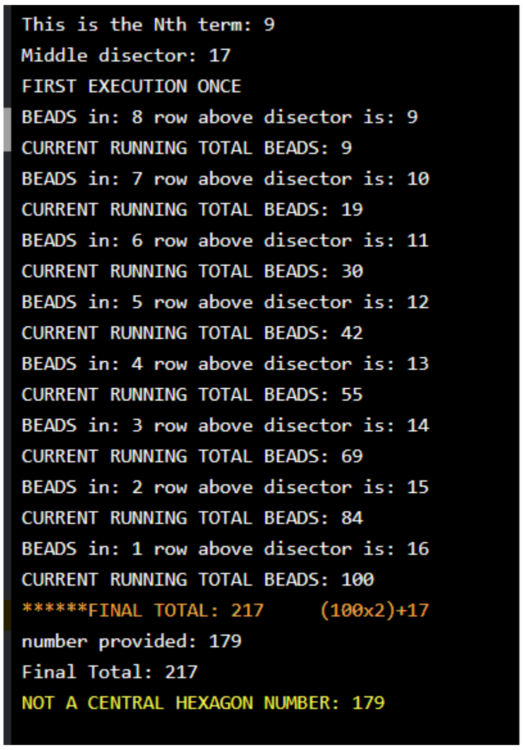
Now it is worth exploring a few more scenarios. Note my code is designed so if the calculated centered hexagon number exceeds n, it will terminate

<u>TEST CASE 4: n =179</u>

It can be seen that 169 is below 179, so it continues to run

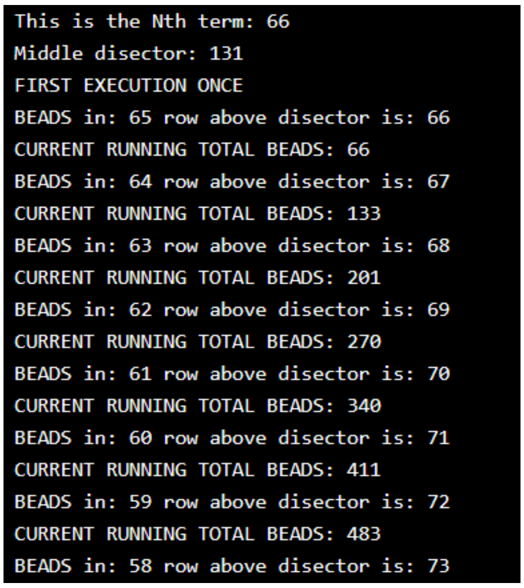


```
This is the Nth term: 8
Middle disector: 15
FIRST EXECUTION ONCE
BEADS in: 7 row above disector is: 8
CURRENT RUNNING TOTAL BEADS: 8
BEADS in: 6 row above disector is: 9
CURRENT RUNNING TOTAL BEADS: 17
BEADS in: 5 row above disector is: 10
CURRENT RUNNING TOTAL BEADS: 27
BEADS in: 4 row above disector is: 11
CURRENT RUNNING TOTAL BEADS: 38
BEADS in: 3 row above disector is: 12
CURRENT RUNNING TOTAL BEADS: 50
BEADS in: 2 row above disector is: 13
CURRENT RUNNING TOTAL BEADS: 63
BEADS in: 1 row above disector is: 14
CURRENT RUNNING TOTAL BEADS: 77
******FINAL TOTAL: 169     (77x2)+15
number provided: 179
Final Total: 169
This is FinalTotal: 169
```

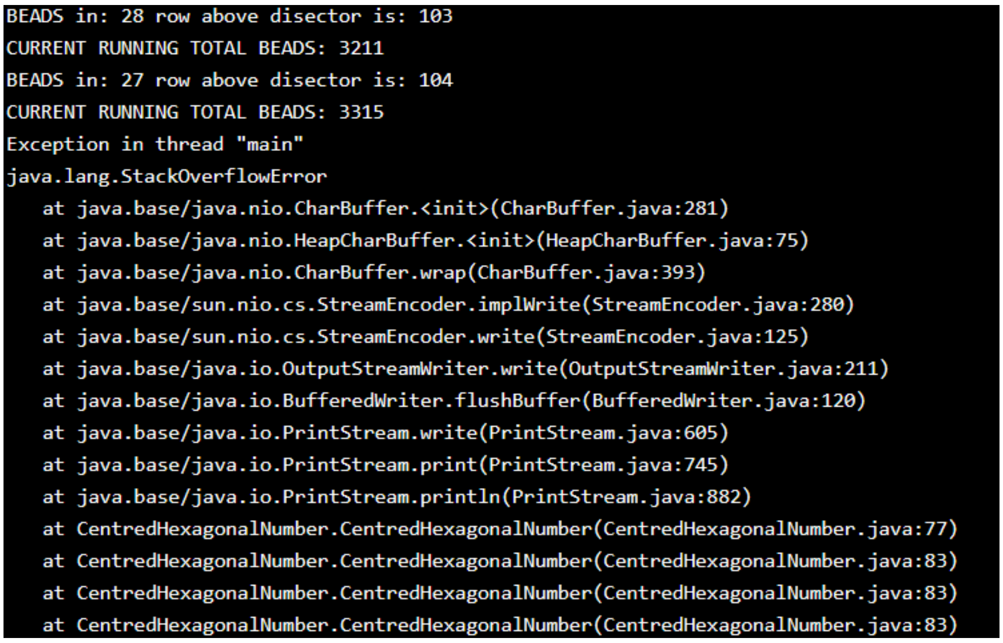

```
This is the Nth term: 9
Middle disector: 17
FIRST EXECUTION ONCE
BEADS in: 8 row above disector is: 9
CURRENT RUNNING TOTAL BEADS: 9
BEADS in: 7 row above disector is: 10
CURRENT RUNNING TOTAL BEADS: 19
BEADS in: 6 row above disector is: 11
CURRENT RUNNING TOTAL BEADS: 30
BEADS in: 5 row above disector is: 12
CURRENT RUNNING TOTAL BEADS: 42
BEADS in: 4 row above disector is: 13
CURRENT RUNNING TOTAL BEADS: 55
BEADS in: 3 row above disector is: 14
CURRENT RUNNING TOTAL BEADS: 69
BEADS in: 2 row above disector is: 15
CURRENT RUNNING TOTAL BEADS: 84
BEADS in: 1 row above disector is: 16
CURRENT RUNNING TOTAL BEADS: 100
******FINAL TOTAL: 217     (100x2)+17
number provided: 179
Final Total: 217
NOT A CENTRAL HEXAGON NUMBER: 179
```

And of course with recursion will be the StackOverFlow error
I performed the following test case:

TEST CASE 4: n = 999179

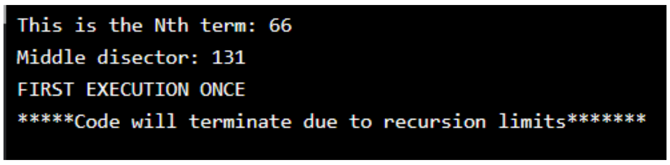


```
This is the Nth term: 66
Middle disector: 131
FIRST EXECUTION ONCE
BEADS in: 65 row above disector is: 66
CURRENT RUNNING TOTAL BEADS: 66
BEADS in: 64 row above disector is: 67
CURRENT RUNNING TOTAL BEADS: 133
BEADS in: 63 row above disector is: 68
CURRENT RUNNING TOTAL BEADS: 201
BEADS in: 62 row above disector is: 69
CURRENT RUNNING TOTAL BEADS: 270
BEADS in: 61 row above disector is: 70
CURRENT RUNNING TOTAL BEADS: 340
BEADS in: 60 row above disector is: 71
CURRENT RUNNING TOTAL BEADS: 411
BEADS in: 59 row above disector is: 72
CURRENT RUNNING TOTAL BEADS: 483
BEADS in: 58 row above disector is: 73
```



```
BEADS in: 28 row above disector is: 103
CURRENT RUNNING TOTAL BEADS: 3211
BEADS in: 27 row above disector is: 104
CURRENT RUNNING TOTAL BEADS: 3315
Exception in thread "main"
java.lang.StackOverflowError
    at java.base/java.nio.CharBuffer.<init>(CharBuffer.java:281)
    at java.base/java.nio.HeapCharBuffer.<init>(HeapCharBuffer.java:75)
    at java.base/java.nio.CharBuffer.wrap(CharBuffer.java:393)
    at java.base/sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:280)
    at java.base/sun.nio.cs.StreamEncoder.write(StreamEncoder.java:125)
    at java.base/java.io.OutputStreamWriter.write(OutputStreamWriter.java:211)
    at java.base/java.io.BufferedWriter.flushBuffer(BufferedWriter.java:120)
    at java.base/java.io.PrintStream.write(PrintStream.java:605)
    at java.base/java.io.PrintStream.print(PrintStream.java:745)
    at java.base/java.io.PrintStream.println(PrintStream.java:882)
    at CentredHexagonalNumber.CentredHexagonalNumber(CentredHexagonalNumber.java:77)
    at CentredHexagonalNumber.CentredHexagonalNumber(CentredHexagonalNumber.java:83)
    at CentredHexagonalNumber.CentredHexagonalNumber(CentredHexagonalNumber.java:83)
    at CentredHexagonalNumber.CentredHexagonalNumber(CentredHexagonalNumber.java:83)
```

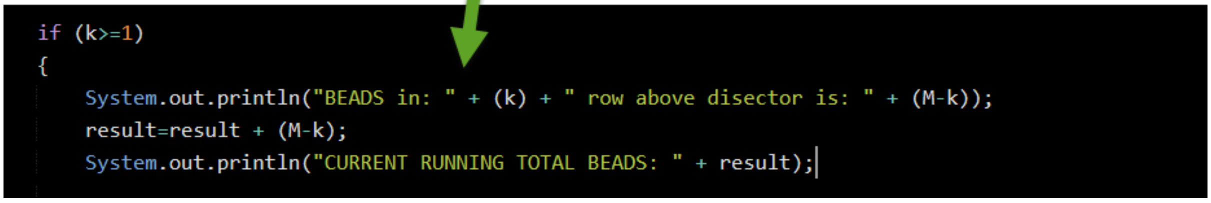
So I will prevent the code from going into NthTerm=66



```
This is the Nth term: 66
Middle disector: 131
FIRST EXECUTION ONCE
*****Code will terminate due to recursion limits*******
```

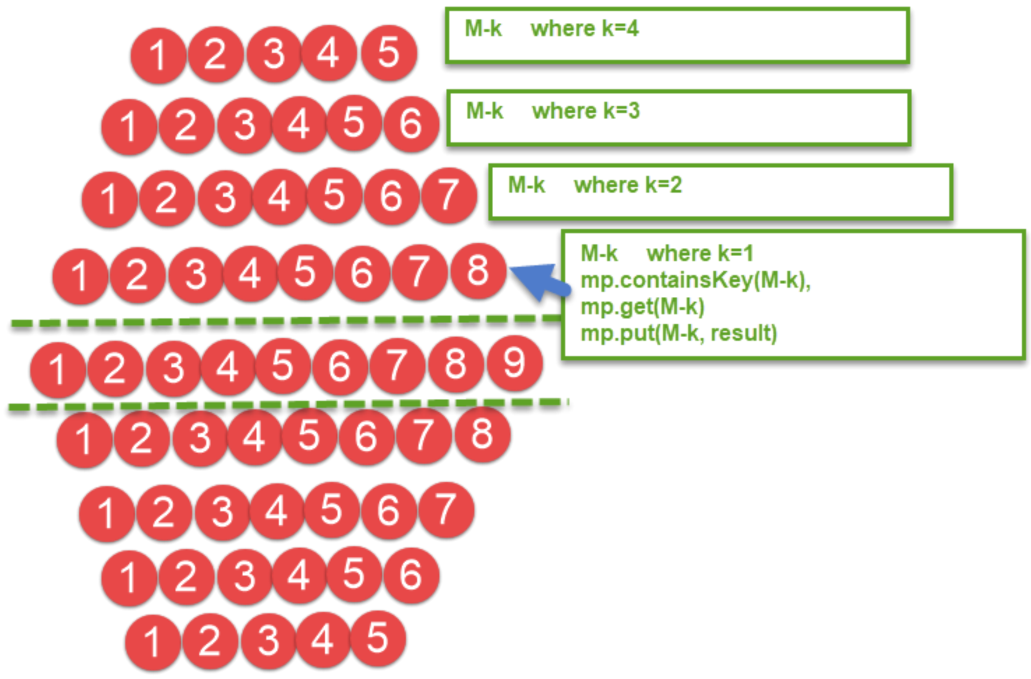
Now I will maintain the code for the above.
My attention will span towards using collection and recalling values from the Hashmap since it will alleviate the recursion issues...

This would be best position to conduct mp.get()   mp.put()   mp.containsKey()



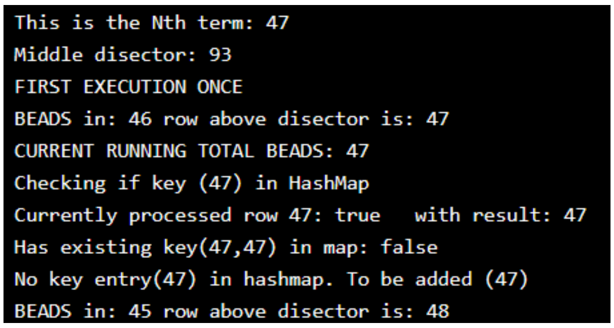
```
if (k>=1)
{
    System.out.println("BEADS in: " + (k) + " row above disector is: " + (M-k));
    result=result + (M-k);
    System.out.println("CURRENT RUNNING TOTAL BEADS: " + result);
```

Just to ensure I am completing the correct lookup:



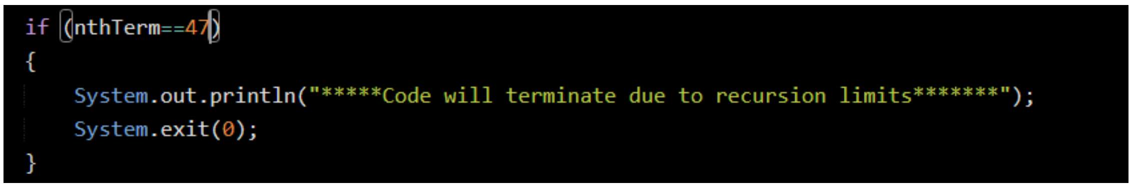
I am now looking at my rationale for using map.
I know I have tackled the

I have completed the following implementation, it was an absolute headache. Infact it added more code as expected, and for some reason it caused the Stack overflow much earlier (on nth term 47 as oppose to 66). So this meant it has performed more recursion! Which I am also unsure about at this stage.



```
This is the Nth term: 47
Middle disector: 93
FIRST EXECUTION ONCE
BEADS in: 46 row above disector is: 47
CURRENT RUNNING TOTAL BEADS: 47
Checking if key (47) in HashMap
Currently processed row 47: true   with result: 47
Has existing key(47,47) in map: false
No key entry(47) in hashmap. To be added (47)
BEADS in: 45 row above disector is: 48
```

For now, I have introduced this bit and adjusted value....



```
if (nthTerm==47)
{
    System.out.println("*****Code will terminate due to recursion limits*******");
    System.exit(0);
}
```

On first instance, it appears that all my results are still correct. But I can see that there are so many more transactional outputs on the screen. But this is unrelated to the StackOverFlow   since the session has not been killed due to memory exhaustion..

It also meant I introduced more repeat code... I was aware of this occurring even before I implemented the collection... However I opted to allow duplication as oppose to introducing methods. This was simply because the challenge stated to write a function. Infact using a single method was out of the question given that I tackled this via recursion and required static variables to hold state.

I will complete the official testing of the collection based challenge at later point and remediate the code!!