Lab 8: Memoizing the Binomial Coefficient

You must implement a class which calculates the binomial coefficient (sometimes called the "choose" operation), using memoization. The class should be called `BinomialCalculator`.

Remember that the binomial coefficient $\binom{n}{k}$ signifies the number of ways we may choose k objects out of n total objects, where order is irrelevant. It is usually defined as follows:

$$\binom{n}{k} = \begin{cases} \dfrac{n!}{k!\ (n-k)!} & \text{if } 0 \leq k \leq n \\ 0 & \text{otherwise} \end{cases}$$

However, this can cause great difficulty if either n or k is more than 20, since the factorials get so large. An alternate formula is:

$$\binom{n}{k} = \begin{cases} 1 & \text{if } n \geq 0 \text{ and } k = 0 \text{ or } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{if } 0 < k < n \\ 0 & \text{otherwise} \end{cases}$$

Even though this will yield the exact same answer, it is far more convenient. This is because the numbers being used are much smaller, and it can easily be *memoized*.

You should make the following static function:

- `public static long calcBinomialCoefficient(int numItems, int numChosen)`, which calculates the binomial coefficient. The variables `numItems` and `numChosen` are n and k, respectively.

In addition there should be a `main()` function, which calculates coefficients unitl the user hits enter:

```
Testing the binomial coefficient calculator.
Enter two numbers, or hit enter to quit: 5 3
  The binomial coefficient is 10.
Enter two numbers, or hit enter to quit: -1 -1
  The binomial coefficient is 0.
Enter two numbers, or hit enter to quit: 50 25
  The binomial coefficient is 126410606437752.
Enter two numbers, or hit enter to quit:
Goodbye!
```

You may of course add any private functions that you think are appropriate. (It may help to make a debugging function that prints out the table.) Also as before, make sure that your program doesn't throw any uncaught exceptions.

The table in which the coefficients are stored should be an `ArrayList` of `long[]` objects. (Remember that arrays are themselves complex types. Thus, there is nothing wrong with making an `ArrayList` of arrays.) The length of each array should be 1 more than its index (e.g. row 4 will have a length of 5). Thus, this data structure will resemble Pascal's triangle as shown:

		k								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
	0	1								
	1	1	1							
	2	1	2	1						
	3	1	3	3	1					
n	**4**	1	4	6	4	1				
	5	1	5	10	10	5	1			
	6	1	6	15	20	15	6	1		
	7	1	7	21	35	35	21	7	1	
	8	1	8	28	56	70	56	28	8	1
						⋮				

When the function is called, it will do the following:

1. It will first check if the table is large enough to contain the row based on n.
2. If not, it needs to grow the table:
   (a) It will add one or more arrays to the `ArrayList` until it is big enough.
   (b) Each row should have all the proper values filled in immediately, as calculated from the previous row.
3. Finally, since the table now *must* contain the proper row, it will return the nth array's kth element without problem.

Because of memoization, the algorithm will usually be very efficient!