Recursion Practice:

1. Write a recursive method that has one parameter which is an int value called \( x \). The method prints \( x \) asterisks, followed by \( x \) exclamation points. Do NOT use any loops. Do NOT use any variables other than \( x \).

2. Implement the following method. Do not use any local variables or loops.

   ```java
   public static void pattern(int n)
   // Precondition: n > 0;
   // Postcondition: The output consists of lines of integers. The first line
   // is the number n. The next line is the number 2n. The next line is
   // the number 4n, and so on until you reach a number that is larger than
   // 4242. This list of numbers is then repeated backward until you get back
   // to n.
   /* Example output with n = 840:
   840
   1680
   3360
   6720
   6720
   3360
   1680
   840
   */
   ```

3. Write a recursive method with two int parameters, \( m \) and \( n \). The precondition requires \( 0 \leq m \) and \( m \leq n \). The method prints a line of \( m \) asterisks, then a line of \( m+1 \) asterisks, and so on up to a line of \( n \) asterisks. Then the same pattern is repeated backward: a line of \( n \) asterisks, then \( n-1 \), and so on down to \( n \). The only loop allowed in your implementation is a loop to print a line of \( m \) asterisks. You may have two copies of this loop in different places of the implementation.

4. Write a method with one positive int parameter called \( n \). The method will write \( 2^n - 1 \) integers. Here are the patterns of output for various values of \( n \):

   - \( n=1 \): Output is: 1
   - \( n=2 \): Output is: 1 2 1
   - \( n=3 \): Output is: 1 2 1 3 1 2 1
   - \( n=4 \): Output is: 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

   And so on. Note that the output for \( n \) always consists of the output for \( n-1 \), followed by \( n \) itself, followed by a second copy of the output for \( n-1 \).
5. This question involves a game with teddy bears. The game starts when I give you some bears. You can then give back some bears, but you must follow these rules (where \( n \) is the number of bears that you have):

- If \( n \) is even, then you may give back exactly \( n/2 \) bears.
- If \( n \) is divisible by 3 or 4, then you may multiply the last two digits of \( n \) and give back this many bears. (By the way, the last digit of \( n \) is \( n\%10 \), and the next-to-last digit is \( ((n\%100)/10) \).
- If \( n \) is divisible by 5, then you may give back exactly 42 bears.
- The goal of the game is to end up with EXACTLY 42 bears.

For example, suppose that you start with 250 bears. Then you could make these moves:
--Start with 250 bears.
--Since 250 is divisible by 5, you may return 42 of the bears, leaving you with 208 bears.
--Since 208 is even, you may return half of the bears, leaving you with 104 bears.
--Since 104 is even, you may return half of the bears, leaving you with 52 bears.
--Since 52 is divisible by 4, you may multiply the last two digits (resulting in 10) and return these 10 bears. This leaves you with 42 bears.
--You have reached the goal!

Write a recursive method to meet this specification:

```java
public static boolean bears(int n)
{
    // Postcondition: A true return value means that it is possible to win
    // the bear game by starting with n bears. A false return value means that
    // it is not possible to win the bear game by starting with n bears.
    // Examples:
    //   bear(250) is true (as shown above)
    //   bear(42) is true
    //   bear(84) is true
    //   bear(53) is false
    //   bear(41) is false

    // Hint: To test whether n is even, use the expression ((n % 2) == 0).
```