Question 1: A Mathematician's Christmas Tree

a) The idea of my solution.

First we initialize important constants (minimal and maximal value for input tree height and strings to optimize memory usage). Then we print out description of the program and request text for input tree height value. In while loop we check is the value out of bounds and if yes, continuously ask user to input correct value until he will enter correct. Then we calculate offset value using maximal value of number in tree plus one extra symbol for space. To optimize calculation we create array where we will store powers of number two. Then in cycle from iteration 2 to height of tree we calculate new power of number two and print triangles with correspondent to iteration number height. For each triangle we use cycle again and first number two we print with offset multiplied by height of the tree minus number of the row in triangle. Then we print out other numbers in the row.

b) Source code

```
package uebung4.question1;

import io.Input;

/**
 * @author Andrii Dzhyrma
 */
public class ChristmasTree {

  // Bounds for input height
  private static final int MIN = 2;
  private static final int MAX = 15;
  // Constant string with description of the program
  private static final String PROGRAM_DESCRIPTION_STRING = "Christmas Tree Printing
Service\n-------------------------------\n";
  // Constant request string for height
  private static final String HEIGHT_REQUEST_FORMAT_STRING = "Height of Base Pyramid:
";
  // Constant error format string
  private static final String OUT_OF_BOUNDS_ERROR_FORMAT_STRING = "Height of Base
should be in range [%d, %d]. Please try again: ";

  /**
   * @param args
   *             - no arguments will evaluate
   */
  public static void main(String[] args) {
    // Printing out the description
    System.out.println(PROGRAM_DESCRIPTION_STRING);

    // Requesting the height of Base Pyramid
    System.out.print(HEIGHT_REQUEST_FORMAT_STRING);
    int height = Input.readInt();
    while (height < MIN || height > MAX) {
      System.out.printf(OUT_OF_BOUNDS_ERROR_FORMAT_STRING, MIN, MAX);
      height = Input.readInt();
    }
    System.out.println();

    // Calculation of the offset and power of 2 array initialization
    int offset = String.valueOf((int) Math.pow(2, height)).length() + 1;
    int[] powerOf2 = new int[height];
```

```
    powerOf2[0] = 2;

    for (int i = 2; i <= height; i++) {
      // Store calculated numbers into the array
      powerOf2[i - 1] = (int) Math.pow(2, i);
      // Print out rows of the tree
      for (int j = 0; j < i; j++) {
        System.out.printf("%" + (height - j) * offset + "d", 2);
        for (int k = 1; k <= j; k++)
          System.out.printf("%" + offset + "d", powerOf2[k]);
        for (int k = j - 1; k >= 0; k--)
          System.out.printf("%" + offset + "d", powerOf2[k]);
        System.out.println();
      }
    }
  }
}
```

c) Test plan

Aim	Input	Expected output	Program output

Common case	2	`Height of Base Pyramid: 2`
		
		`  2`
		`2 4 2`
			`Height of Base Pyramid: 2`
			
			`  2`
			`2 4 2`

Common case	5	`Height of Base Pyramid: 5`
		
		`  2`
		`2 4 2`
		`  2`
		`2 4 8 4 2`
		`  2`
		`2 4 8 4 2`
		`2 4 8 16 8 4 2`
		`  2`
		`2 4 8 4 2`
		`2 4 8 16 8 4 2`
		`2 4 8 16 32 16 8 4 2`
			`Height of Base Pyramid: 5`
			
			`  2`
			`2 4 2`
			`  2`
			`2 4 8 4 2`
			`  2`
			`2 4 8 4 2`
			`2 4 8 16 8 4 2`
			`  2`
			`2 4 8 4 2`
			`2 4 8 16 8 4 2`
			`2 4 8 16 32 16 8 4 2`

Out of bounds	16	**Display information about the value that it is out of bounds.**	`Height of Base Pyramid: 16` `Height of Base should be in range [2, 15]. Please try again:`
Out of bounds (less than zero)	-5	**Display information about the value that it is out of bounds.**	`Height of Base Pyramid: -5` `Height of Base should be in range [2, 15]. Please try again:`
Letters	a	**Display information that it is not an integer value**	`Height of Base Pyramid: a` `Not a valid int, please try again:`

Question 2: Mini-Calculator

a) The idea of my solution.

First we initialize all important constants (strings for output to optimize memory usage). Then we write print method to output our equation each time when calculation was done. We print the description of the program. Then we try to parse each number to int on each even position and on catch just print error and finish the program. After that we check all operators on odd positions of args, if we will find something else, then print an error and finish the program. At the end of checking we check for odd length of args array to be sure, that last argument is number. The whole array of args we store to the list to make deleting element process easier. While length of new list is bigger than one we will do next: go through operators and try to find multiplication sign with storing positions of signs plus and minus. If multiplication sign was not found, we use minus, and if that was not found, we use plus. Using switch statement we make correspondent operation to neighbor values and store result to left neighbor. After that we delete two next elements from the list and print out the new equation.

b) Source code

```
package uebung4.question2;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * @author Andrii Dzhyrma
 */
public class MiniCalc {

  // Constant string with description of the program
  private static final String PROGRAM_DESCRIPTION_STRING = "MiniCalc\n========\n";
  // Error message for incorrect integer input
  private static final String NUMBER_FORMAT_ERROR_STRING = "Incorrect integer input";
  // Error message for incorrect operator input
  private static final String OPERATOR_EXISING_ERROR_STRING = "Only +, -, x operators
are allowed";
  // Error message for incorrect equation format
  private static final String WRONG_EQUATION_FORMAT_ERROR_STRING = "Incorrect
equation format";

  /**
   * Print out equation in given by assignment format
   *
   * @param equation
   *            - equation to print
   */
  private static void printEquation(List<String> equation) {
    for (String string : equation) {
      System.out.print(' ');
      System.out.print(string);
    }
    System.out.println();
  }

  /**
   * @param args
   *            - no arguments will evaluate
   */
```

```
  public static void main(String[] args) {
    // Printing out the description
    System.out.println(PROGRAM_DESCRIPTION_STRING);

    // Checking for valid numbers
    try {
      for (int i = 0; i < args.length; i += 2)
        Integer.parseInt(args[i]);
    } catch (NumberFormatException e) {
      System.out.println(NUMBER_FORMAT_ERROR_STRING);
      System.out.println(e.getMessage());
      return;
    }

    // Checking for valid operators
    for (int i = 1; i < args.length; i += 2)
      if (args[i].length() != 1 || (args[i].charAt(0) != 'x' && args[i].charAt(0) !=
'+' && args[i].charAt(0) != '-')) {
        System.out.println(OPERATOR_EXISING_ERROR_STRING);
        return;
      }

    // Checking for valid equation (can't be a situation with operator only
    // at the end)
    if (args.length % 2 == 0) {
      System.out.println(WRONG_EQUATION_FORMAT_ERROR_STRING);
      return;
    }

    // Making a list with all numbers and operators
    List<String> argsList = new ArrayList<String>(Arrays.asList(args));
    // Printing out all the equation
    printEquation(argsList);

    // While there is more then one element, means there are still some
    // operators inside
    while (argsList.size() > 1) {
      // Finding positions of the operators in equation
      int plusPos = argsList.size();
      int minusPos = argsList.size();
      int signPos;
      for (signPos = 1; signPos < argsList.size() && argsList.get(signPos).charAt(0)
!= 'x'; signPos += 2)
        if (signPos < plusPos && argsList.get(signPos).charAt(0) == '+')
          plusPos = signPos;
        else if (signPos < minusPos && argsList.get(signPos).charAt(0) == '-')
          minusPos = signPos;
      // Choosing more important operator
      if (signPos >= argsList.size())
        signPos = (minusPos == argsList.size()) ? plusPos : minusPos;

      // Calculating result of chosen sign operator and neighbor numbers
      String result;
      switch (argsList.get(signPos).charAt(0)) {
      case 'x':
        result = String.valueOf(Integer.parseInt(argsList.get(signPos - 1)) *
Integer.parseInt(argsList.get(signPos + 1)));
        break;
      case '+':
        result = String.valueOf(Integer.parseInt(argsList.get(signPos - 1)) +
Integer.parseInt(argsList.get(signPos + 1)));
        break;
```

```
      case '-':
        result = String.valueOf(Integer.parseInt(argsList.get(signPos - 1)) -
Integer.parseInt(argsList.get(signPos + 1)));
        break;
      default:
        result = "";
        break;
      }

      //Replacing previous number with the new one
      argsList.set(signPos - 1, result);
      // Deleting calculated sign and number after it
      argsList.remove(signPos);
      argsList.remove(signPos);

      // Printing out new equation
      printEquation(argsList);
    }
  }
}
```

c) Test plan

Aim	Input	Expected output	Program output
Common case	1 + 2 x 3 x 4 - 5	MiniCalc ======== 1 + 2 x 3 x 4 - 5 1 + 6 x 4 - 5 1 + 24 - 5 1 + 19 20	MiniCalc ======== 1 + 2 x 3 x 4 - 5 1 + 6 x 4 - 5 1 + 24 - 5 1 + 19 20
Common case	1 x 2 x 3 x 4 x 5 x 6 x 7	MiniCalc ======== 1 x 2 x 3 x 4 x 5 x 6 x 7 2 x 3 x 4 x 5 x 6 x 7 6 x 4 x 5 x 6 x 7 24 x 5 x 6 x 7 120 x 6 x 7 720 x 7 5040	MiniCalc ======== 1 x 2 x 3 x 4 x 5 x 6 x 7 2 x 3 x 4 x 5 x 6 x 7 6 x 4 x 5 x 6 x 7 24 x 5 x 6 x 7 120 x 6 x 7 720 x 7 5040
Operator at the end	1 + 2 +	**Output an error about equation**	MiniCalc ======== Incorrect equation format
Two operators in a row	1 + + 2	**Output an error about equation**	MiniCalc ======== Incorrect integer input For input string: "+"
One of the values is float	1.2 + 3	**Output an error about values**	MiniCalc ======== Incorrect integer input For input string: "1.2"

One of the operator is not exists	1 / 2	**Output an error about operators**	MiniCalc ======== Only +, -, x operators are allowed
String instead of equation	Hello	**Output an error about equation**	MiniCalc ======== Incorrect integer input For input string: "Hello"
Only one number	5	MiniCalc ======== 5	MiniCalc ======== 5
Only one operator	+	**Output an error about equation**	MiniCalc ======== Incorrect integer input For input string: "+"
Out of integer max value	999999999999999 + 1	**Output an error about value**	MiniCalc ======== Incorrect integer input For input string: "999999999999999"
Out of integer result value	2147483647 + 1	MiniCalc ======== 2147483647 + 1 -2147483648	MiniCalc ======== 2147483647 + 1 -2147483648