Topic: Arithmetic expression

Jonathan is a freshman student at college. He must write a simple Java program to evaluate arithmetic expressions. These expressions consist of numbers, additions, and multiplications; like this:

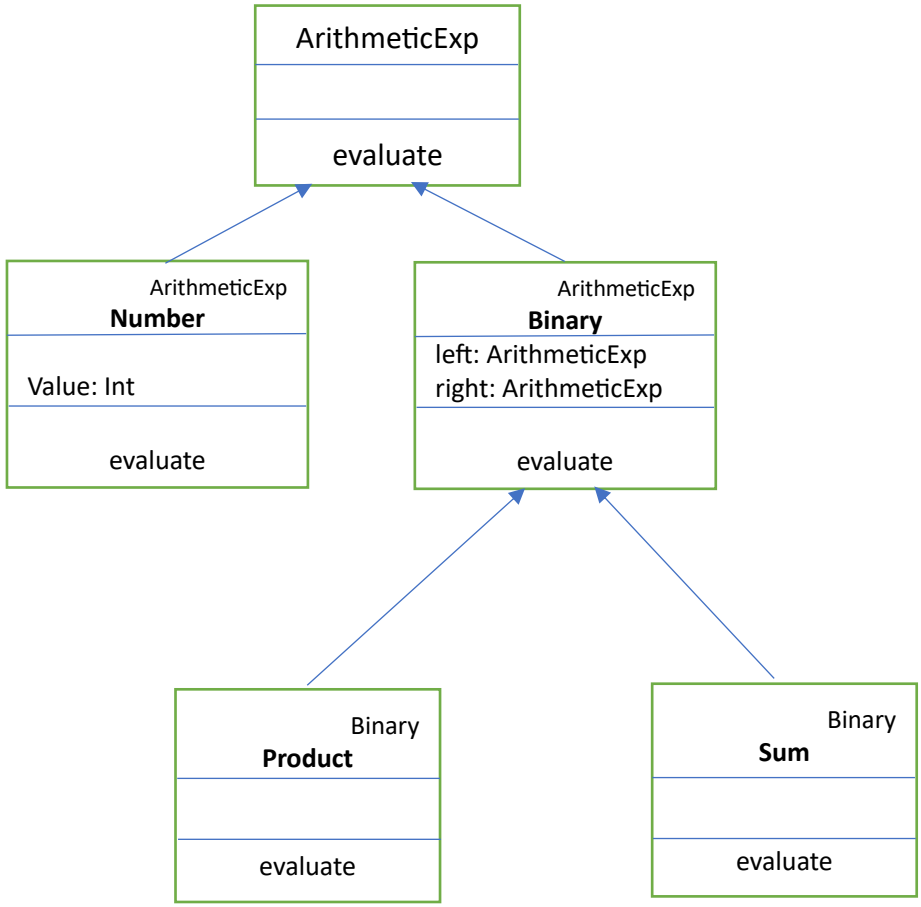
*2+5*3*

He produced the following solution which is not an OOP solution to the problem.

```
/** Data structures for the expressions*/
class ArithmeticExp {
      public int type;
      public int value;
      public ArithmeticExp left;
      public ArithmeticExp right;

      public ArithmeticExp(int type, int value, ArithmeticExp left, ArithmeticExp right) {
        this.type = type;
        this.value = value;
        this.left = left;
        this.right = right;
      }
}
/** main class */
class ArithmeticOperation {

      /** Constants for representing the types*/
      public static final int TYPE_NUMBER = 1;
      public static final int TYPE_SUM = 2;
      public static final int TYPE_PROD = 3;

      public static void main(String [] args) {
        // constructing the expression 3 + 2 * 5
        ArithmeticExp term = new ArithmeticExp(TYPE_SUM, 0,
                new ArithmeticExp(TYPE_NUMBER, 3, null, null),
                new ArithmeticExp(TYPE_PROD, 0,
                        new ArithmeticExp(TYPE_NUMBER, 2, null, null),
                        new ArithmeticExp(TYPE_NUMBER, 5, null, null)));

        System.out.println(evaluate(term));
      }
      /** Evaluating the expression recursively */
      public static int evaluate(ArithmeticExp term) {
        switch (term.type) {
                case TYPE_NUMBER:
                        return term.value;
                case TYPE_SUM:
                        return evaluate(term.left) + evaluate(term.right);
                case TYPE_PROD:
                        return evaluate(term.left) * evaluate(term.right);
                default:
                        return 0;       //error, should never happen

        }
      }
}
```

program works but is written in a "procedural" style rather than object oriented. Think about his solution and try to explain why it is not a good approach and how it could be improved.
Write an improved version based on the following guideline:

- Define an abstract class ArithmeticExp that will declare an evaluate abstract method.
- Create separate classes for each type of expression (Number, Sum, Product) and inherit from ArithmeticExp.
- You can avoid unnecessary duplication of code by creating a Binary intermediate abstract class that will contain the *left* and *right* attributes representing the left and right operands of a binary expression, respectively. Inherit Sum and Product from Binary.
- Adapt the main method of the main class to your new structure.



In the next step, try to improve your code in the following way:

- Add the ability to generate the representation of your expression as a string. For example, if your expression is the number 13, this function should return the string "13". If it is a Sum with 12 and 13 as left and right respectively, the function in question will have to return the string "12 + 13".
- Add a new type of binary expression: Modulo representing binary expressions whose evaluation returns the remainder of the entire left division by right.

Take a critical look at your new version. If you had to copy and paste portions of your code to add these features, then there are still some design issues. Try to make sure there is no unnecessary duplication of code.

Submission:

- Commit your work and push it on GitHub, provide the repo link on Moodle.
- Upload each .java file to the Moodle or send a zip file of your whole work to the Moodle.
    - Make the zip file as your first name and your student id:
        - *FirstName_studentId_FT.zip*

Deductions:

1. Did not follow the provided instruction for submission = -20%
2. Academic dishonesty = 0 marks
3. Late submissions = -10% for every 30 minutes late submission.