PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

SCHOOL OF APPLIED SCIENCES
--TLEMCEN--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

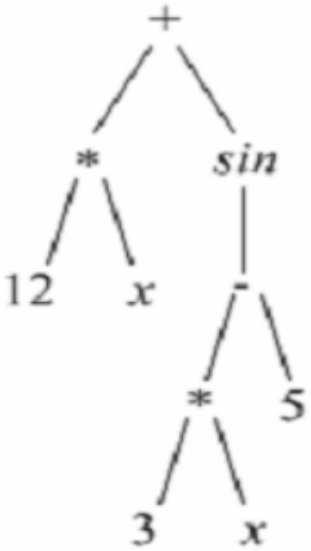
Course : Object Oriented Programming
Practicle Work N°4 : Interface, Abstract classes and ArrayList

In this exercise, you are asked to define a set of classes to represent functions of a single variable, composed of constants, occurrences of the variable x, the four arithmetic operations (+, –, ×, /), and calls to standard functions such as sin, cos, exp, log, etc. For example:

```
12 x + sin(3x – 5)
```

In a program, an expression like this can be efficiently represented by a tree structure, organized as shown in the adjacent figure. It is made up of:

- **Leaves** (constants and variables),
- **Nodes with two "children"** (binary operators), and
- **Nodes with one child** (functions of a single variable).



The classes that need to be defined are intended to represent the nodes of such a tree. There are several types of nodes:

- A **node representing a constant** holds a number — the value of the constant.
- A **node representing an occurrence of the variable x** holds no additional information.
- A **node representing an addition, subtraction, multiplication, or division** holds two pieces of information: the expressions that are its operands.
- A **node representing a function call** holds one piece of information: the expression that is its argument.

Define the following classes and interfaces:

Expression – an interface representing what all arithmetic expressions have in common (i.e., all types of nodes in our tree structure). This interface consists of a single method:

public double value(double x);

This method returns the value of the expression for a given value of x. Of course, all **concrete classes** in this hierarchy must provide an implementation of the value method. They should also provide a meaningful override of the method:

public String toString();

Constant – a concrete class that implements the Expression interface, where each instance represents an occurrence of a constant. This class has one member: the value of the constant.

Variable – a concrete class that implements the Expression interface, where each instance represents an occurrence of the variable x. This class does not require any members.

BinaryOperation – an abstract class that implements the Expression interface and gathers what all binary operators have in common. It therefore has two instance members of type Expression, representing the two operands.
Addition, Subtraction, Multiplication, Division – concrete classes that inherit from the BinaryOperation class to represent the binary operations. These classes provide the appropriate implementation of the value method promised by the Expression interface.

UnaryOperation – an abstract class gathering what all unary operators have in common. It has one instance member of type Expression, representing the single operand.
Sin, Cos, Log, Exp – concrete classes that inherit from the UnaryOperation class to represent the standard functions. These classes provide the appropriate implementation of the value method promised by the Expression interface.

You are **not** required to solve the (difficult) problem of "parsing" such a tree—that is, constructing it from text, for example read from the console. However, you **must** demonstrate that your structure is well suited for calculating the value of the expression for a given value of the variable x.
To do this, write a test class to evaluate the following expression:

```
f(x) = 2 * sin(x) + 3 * cos(x)
```

For different values of x stored in an ArrayList