Objective: To implement a mini arithmetic expression compiler and processor for a programming language called Lisp. It uses simple linear data structures.

OVERVIEW:

1. Define class Token.
2. Define class ExpressionEvaluator.
3. Define class ExpressionEvaluator2.
4. Download and use the tester module to ensure that your program is correct.

LISP is a family of programming languages with a long history and a distinctive, fully parenthesized prefix notation. Originally specified in 1958, LISP is the second-oldest high-level programming language in widespread use today. Lisp (LISt Processor) was originally created as a practical mathematical notation for computer programs, influenced by the notation of lambda calculus. It quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, Lisp pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, etc. Lisp is an expression-oriented language. A Lisp expression is written with its elements separated by whitespace, and surrounded by parentheses. For arithmetic expressions, each of the four arithmetic operators (+,-, *, /) appears before an arbitrary number of operands, which are separated by spaces and enclosed in parentheses. For example, (+ 1 2) is an expression whose elements are the three atoms +, 1, and 2.

The use of parentheses is Lisp's most immediately obvious difference from other programming language families. As a result, students have long given Lisp nicknames such as Lost In Stupid Parentheses.

The following are the behaviours of the four arithmetic operators that we are going to consider for this task:

- **Addition operator (+)**
  (+ a b c d ...) - returns the sum of all operands (a, b, c, d,...).
    - (+) - returns 0.
- **Subtraction operator (-)**
  (- a b c d ...) returns a - b - c - d - ....
    - (- a) - returns -a. the minus operator must have at least one operand.
- **Multiplication operator (*)**
  (* a b c d ...) returns the product of all operands (a, b, c, d,...).
    - (*) - returns 1.
- **Division operator (/)**
  (/ a b c d ...) returns a / b / c / d / ....
    - (/ a) - returns 1/a. The divide operator must have at least one operand.

You can form more complex expressions by combining these basic arithmetic expressions. For example, the following is a valid Lisp expression:

- (+ (- 12) (* 4 2 3) (/ (+4) (*) (- 2 3 1)))
- The expression is evaluated as:
- (+ (- 12) (* 4 2 3) (/ 4 1 -2))

- = (+ -12 24 -2)
- = 10

Rules

1. You may import Java built-in libraries (java.util.*)
2. All data fields should be declared private or protected
3. You may add your own helper methods or data fields, but they should be private or protected. You may add additional constructors which are public.
4. When commenting code, use JavaDoc-style comments.
   Include @param and/or @return tags to explain what is passed in or returned wherever it isn't obvious. Any other JavaDoc tags are optional.

TOKEN CLASS:

(*10 pts*) This class represents an individual token within a Lisp expression. A Token can be an operator or a variable. For example, in the expression (+ a 2), we have the following tokens: +, a, and 2, where + is an operator and 2 and a are operands. Define a class called Token. The class has the following attributes:

- operator - a Character object that represents an operator.
- operand - a Double object that represents operands.
- isOperator - a boolean variable that indicates whether a token is an operator or not.

The class also has the following methods:

- public Token(Character operator) - a constructor that initializes an operator with the given value and an operand to 0.0 and isOperator to true.
- public Token(Double operand) - a constructor that initializes an operand with the given value and an operator to ' ' (space character) and isOperator to false.
- Double applyOperator(Double value1, Double value2) - a method that applies the operator to the two values and return the result.
- public Double getIdentity() - a method that returns the identity value of an operator. For example, x + 0 = x. Therefore 0 is the identity for + and it will be associated with the expression (+), i.e., if a Lisp expression that contains (+) will be evaluated to 0. The method will return the identity for +, -, *, /.
- public boolean takesNoOperands() - this method decides if an operator can be used without an operand in an expression. Please, take a look at the description of Lisp expression above to decide which arithmetic operator can be used without an operand. The method returns true if the operator can be used without an operand, false otherwise.
- public boolean isOperator() - returns true if the token is an operator, false otherwise.
- Getter methods for the operand called getValue and the operator called getOperator.

EXPRESSIONEVALUATOR:

(*55 pts*) This class has static methods that will help us evaluate Lisp arithmetic expressions. Before a List expression is evaluated, we have to make sure that it is a valid expression. The following expressions show valid and invalid Lisp arithmetic expressions.

Valid Expression

```
(+ (- 5)
   (* 4 4 5)
   (/ 3 2 2)
   (* 4 4)
)
```

Invalid Expressions

```
(+ (-)
   (* 4 4 5)
   (/ 3 8 l1)
   (* 4 4)
)
(+ (- 9)
   (* 4 4 5)
   (* (/ 3 6 7)
   (* 1 1)
)
(+ (- 2)
   (* 4 4 5)
   (/ 3 2 1))
   (* c 6)
```

```
(+ (- 2)
         (* 3 3 4)
    ((* (/ 3 1 l)
    (* a b)
            )
```

The following are the behaviors of the class:

- (*20pts*) public static boolean isBalanced(String expr) - this method accepts an expression and returns true if it is balanced, false otherwise. A Lisp expression is considered balanced if it is properly parenthesized. The following table gives examples of correctly/incorrectly parenthesized Lisp expressions.

Expression	Result	Remark
(* r r)	balanced	
(* (/ 3 w l)	not balanced	missing closing parenthesis (should be (* (/ 3 w l))
(* (/ 3 w l)(* r r))	balanced	
(* (/ 3 w l)) (* (r r)))	not balanced	extra closing)
(* (/ 3 w l) * (r r))	balanced	However, the expression is not valid Lisp expresssion
((* (/ 3 w l) (* r r))	not balanced	Extra opening (

- (*35 pts*) public static Double evaluate(String expr) - a method that accepts a Lisp expression and returns the result. The method returns a RunTimeException if the accepted expression is not a valid Lisp expression. The following can be reasons for the method to throw the exception:
  - An empty expression ( ).
  - Unbalanced expression.
  - Invalid expression. See the following examples of invalid expressions.
  - If the expression does not adhere to the rules outlined in the "Overview" section, it is invalid.

  In this task, if an expression contains operators other than the basic four arithmetic operators it will be considered as an operand. An expression can contain single-letter variables as operands. If an expression contains variable operands, it should prompt the programmer for the values of the operands and use the values for the evaluation. For the sake of simplicity, only integer values are going to be accepted from the user. Note that if an expression contains operands such as var, it will consider it as three separate operand tokens v, a, r and prompt the programmer for the three values. Take a look at the examples given below.

The following table gives examples of invalid/valid Lisp expressions

Expression	Result	Remark
(* 3255)	valid	
(-)	invalid	- operator needs at least one operand
( )	invalid	Expression cannot be empty (just for this project)
(/ * w l)	invalid	two operators one after the other
(a + b + c + d)	invalid	not prefixed
(* (/ 3 w l) (* (r r))	invalid	not balanced
(* (/ 3 w l * r r))	invalid	two operators without proper parethesis
(*(/355)(*2k5))	valid	Your program should be smart enough to recognize tokens even if there is no space
(* (/ 3 w l) (/) (* r r))	invalid	division operator needs at least one operand

Example:

```
> import java.util.*;
> String test1 = "(+ (- 6 7) (* 234 455 256) (/ (/ 3) (*) (-2 3 1)))";;
> result = ExpressionEvaluator.evaluate(test1);
> System.out.println(result)
> 2.7256318833333332E7


> String test2 = "(+ (- 632) (* a 3 b c) (/ (+ 32) (*) (- c 3 d)))"
> result = ExpressionEvaluator.evaluate(test2);<
> What is the value of 'a'?
> 5 // accepts 5 from the user
> What is the value of 'b'?
> 6 // accepts 6 from the user
> What is the value of 'c'? // accepts 22 from the user
> What is the value of 'c'? // accepts 10 from the user
> What is the value of 'd'?// accepts 9 from the user

> System.out.println(result)
> 1332.0


> String test3 = (+ (- 6) (* abb3c4bc))";
> result = ExpressionEvaluator.evaluate(test3);<
> What is the value of 'a'? // accepts 1 from the user
> What is the value of 'b'? // accepts 6 from
> What is the value of 'b'? // accepts 0 from the user
> What is the value of 'c'? // accepts 10 from the user
> What is the value of 'b'?// accepts 19 from the user
> What is the value of 'c'?// accepts 95 from the user
> System.out.println(result)
> -6.0
```

EXPRESSIONEVALUATOR2:

- *(25 pts.)* Extend the <expressionevaluator< code="">class and override the evaluate( ) method of the class. The method is modified in such a way that the operands can be variable names that are string of letters. For example, the following are some examples of such valid expressions:

  Valid Expression

  ```
  (+ (- height)
     (* 4 4 5)
     (/ 3 width length)
     (* radius radius)
  )
  ```

  Your evaluate () method should prompt the user for each variable in an expression and return the evaluated value based on the accepted data from the user.

  Define a class ExpressionEvaluator2 with the above modification. Note that the Token class is not modified.

  Example:

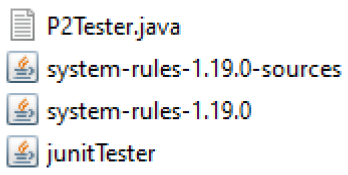
```
> import java.util.*;
> String test1 = "(+ (- 6 7) (* 234 455 256) (/ (/ 3) (*) (-2 3 1)))";;
> result = ExpressionEvaluator2.evaluate(test1);
> System.out.println(result)
> 2.7256318833333332E7
```

```
> String test2 = (+ (- height) (* 3 3 4) (/ 3 width length) (* radius radius))";
> result = ExpressionEvaluator2.evaluate(test2);
> What is the value of 'height'?
> 10 // accepts 10 from the user
> What is the value of 'width'?
> 5 // accepts 5 from the user
> What is the value of 'length'? // accepts 6 from the user
> What is the value of 'radius'? // accepts 6 from the user
> What is the value of 'radius'?// accepts 6 from the user
> System.out.println(result)
> 62.1
```

```
> String test3 = (+ (- 6) (* var3variable4bc))";
> result = ExpressionEvaluator2.evaluate(test3);<
> What is the value of 'var'? // accepts 20 from the user
> What is the value of 'variable'? // accepts 69 from
> What is the value of 'bc'? // accepts 65 from the user
```

```
> System.out.println(result)
> 1076394.0
```

Testing:



P2Tester.java
system-rules-1.19.0-sources
system-rules-1.19.0
junitTester

The additional files are for some of the test cases, and should be copied into your working directory.

RUBRIC:

The following are the Rubric for the grading:

- Proper documentaion (javaDoc) for the classes - (5 pts).
- Application of object-oriented programming - (3 pts).