HW #1: Polynomial Caculator

The assignment should be submitted in groups of two students.

1 General Description

It is highly recommended to read the entire assignment before starting any kind of work.

The assignment goal is to practice the following concepts

- Class design
- Collections
- Inheritance and substitution

In this assignment, you will implement a Polynomial Calculator. The Calculator will allow the user to compute polynomial addition, polynomial multiplication, and other services. The design should be general, and allow easy modification for any type of scalar. To this end, you need to implement the following classes. (Some of the classes may be abstract or interface. You are allowed to provide additional methods if you wish.)

- **Scalar**
  You may not add data members to this class.

  This class should support the services:

  - **Scalar add(Scalar s)** accepts a scalar argument and returns a new scalar which is the sum of the current scalar and the argument.
  - **Scalar mul(Scalar s)** accepts a scalar argument and returns a new scalar which is the multiplication of the current scalar and the argument.
  - **Scalar neg()** returns a new scalar which is the multiplication of the current scalar with (-1).
  - **Scalar power(int exponent)** accepts a scalar argument and returns a new scalar which is the scalar raised to the power of the exponent argument.
  - **int sign()** returns 1 for positive scalar, -1 for negative, and 0 for 0.
  - **boolean equals(Object o)** returns true if *o* is of type *Scalar* and the numeric value of both scalars is equal (not a part of the actual interface, but is overridden for each subclass).

  The following classes extend/implement Scalar:

  - **Integer**: For integer numbers
    The only data member of this class is:

    private int number;

The class should support the operation:

String toString() returns a string that represent the Integer number.

– **Rational**: For rational numbers

The only data member of this class:

* **private int numerator;**
* **private int denominator;**

Note: It is also possible to create a class *Pair*, composed of two integers, instead.

The class should support the services:

* **Rational reduce()** returns a new *Rational* which is rational in its lowest form [1].
* **String toString()** returns a string that represents the rational number according to the following rules:
  Assuming the numerator and denominator be a and b respectively.
  - If $\frac{a}{b}$ is an integer number, *toString()* returns a string representing the value of $\frac{a}{b}$, otherwise returns a string representing the rational in its simplest form (with no spaces).
  - If $\frac{a}{b} < 0$, *toString()* returns the rational value (in its simplest form) with a leading '-'.

  Examples:

  For a rational scalar with a=12 and b=-4, *toString* method should return "-3".

  For a rational scalar with a=-12 and b=-5, *toString* method should return "12/5".

  For a rational scalar with a=12 and b=-8, *toString* method should return "-3/2".

Note: We will not teest invalid rationals, e.g. denominator = 0.

- **Monomial**

  This class represents a single polynomial term. The Monomial is represented by a coefficient (Scalar) and an exponent (nonnegative integer).

  The only data members of this class are :

  – **private int exponent;**
  – **private Scalar coefficient;**

  The class should support the following services :

  – **Monomial add(Monomial m)** accepts a Monomial argument and returns a new Monomial that is the result of adding the current Monomial to the argument m. The Monomials must have the same exponent, otherwise, the method should return *Null*.
  – **Monomial mul(Monomial m)** accepts a Monomial argument and returns a new Monomial that is the result of multiplication of the current Monomial with the argument m.
  – **Scalar evaluate(Scalar s)** accepts a Scalar argument and evaluates the current Monomial using s, and returns the new resulting Scalar. For example, evaluating $2x^2$ on the scalar 2/3 yields $2 * (2/3)^2 = 8/9$.
  – **Monomial derivative()** returns a new Monomial which is the derivative of the current monomial
  – **int sign()** returns 1 for a positive coefficient, -1 for a negative one, and 0 for zero.
  – **boolean equals(Object o)** returns whether o is of type *Monomial* and the coefficient and the exponent of both monomials are equal.

[1] the numerator and denominator have no common factor other than 1

– **String toString()** returns a string that represents the Monomial. A coefficient of 1 of a Monomial with an exponent greater than zero, should not be included, if it equals -1, only the sign should be included. The exponent should be omitted if it is equal to 1.

Examples:

For a Monomial with coefficient 1 and exponent 3, *toString()* should return `x^3`.

For a Monomial with coefficient 4 and exponent 1, *toString()* should return `4x`.

For a Monomial with coefficient -1 and exponent 7, *toString()* should return $-$`x^7`.

For a Monomial with coefficient -5 and exponent 4, *toString()* should return $-$`5x^4`.

For a Monomial with any coefficient and exponent 0, *toString()* should return the coefficient(as a string).

- **Polynomial**

This class represents a polynomial. A polynomial is defined by its sequence of Monomials. A Polynomial does not include two terms with the same exponent.

The only data member of this class is :

private [Collection] monomials;.

Where Collection can be any standard Java Collection or Map

The class should support the *static* function

static Polynomial build(String input)

that receives a string input and returns the corresponding Polynomial. The input is a string of coefficients (numbers) separated by spaces. You may assume legal input, but you may also assume several spaces between the values. We will call this function from our main function, so do not change it's signature.

Examples:

The string "1 2 3" will result in the polynomial $1 + 2x + 3x^2$.
The string "0 1 2 3" will result in the polynomial $x + 2x^2 + 3x^3$.
The string "0 0 0 0 0 0 7" will result in the polynomial $7x^6$.
The string "5" will result in the polynomial $5x^0 = 5$.
The string "1 -2 3" will result in the polynomial $1 - 2x + 3x^2$. (note that the "minus" sign should be attached to the number, i.e., no spaces)
The string "0 1/2 3 -5/3" will result in the polynomial $1/2x + 3x^2 - 5/3x^3$.

In addition, the class supports the following services:

– **Polynomial add(Polynomial p)** receives a polynomial and returns a new polynomial resulting from adding the current polynomial with the argument p.

– **Polynomial mul(Polynomial p)** receives a polynomial and returns a new polynomial resulting from multiplying the current polynomial with the argument p.

– **Scalar evaluate(Scalar s)** receives a scalar and returns a new scalar, which assesses the polynomial on the scalar argument.

– **Polynomial derivative()** returns a new polynomial, the current polynomial derivative.

– **boolean equals(Object o)** returns whether o is of type *Polynomial* and both contains the same monomials.

– **String toString()** returns a friendly representation sorted by increasing the power of the Monomials. The string has to be in a valid form, i.e., no exponent will be shown more than once, exponents appear in ascending order, and terms with a coefficient of zero do not appear. For example $1 +$ `x` $-$ `x^2` $+$ `5x^3` $+$ `7/3x^8` $+$ `24x^24`.

Note: If necessary, every class can include getters, setters, and clone(). You can also add more methods to any of the above classes. Remember that all the above methods must stand on their own and could be called from our code (examiner's code). Data members cannot be added to the classes.

2 Requirements

- Design an appropriate **class diagram**.
- Do not use *casting* in any part of your code. (Except for double-to-int or casting in *equals*.).
- **Do not use *instanceof* in any part of your code.** (Unless you are overriding *equals*).
- Write **unit tests** for each class in your code, i.e., for each method in that class. The amount of tests for each of the methods is up to you. (at least 1 test).

3 Submission instructions

We will provide you with a main file that you will need to add to your project. You must adjust the required imports so that this main will communicate with your project correctly. When running the jar file, this main is the file we want to run.
When submitting the assignment, you must register for a group of two students, and the file must be submitted as a group to Moodle. The file should be a single archive file (.zip or .tar.gz) in the format [student1ID]-[student2ID].zip with the following contents:

1. hw1.pdf - should contain your class diagram modeling the code.
2. hw1.jar - should contain the **source code**, **tests** and **compiled class files**.
3. Create packages for your classes. Do not use the default package.

Note: We will run your code using the latest JDK, version 20. Be sure to create your jar file using this JDK version. You can download the latest JDK in the following link: `https://www.oracle.com/java/technologies/downloads/`. We highly recommend testing the execution of the jar file before submitting it. (using the command we saw at the first practical session)