Principles of Object Oriented Programming
Spring 2021
Assignment 1

Published: 11/4/2021 **Due:** 1/5/2021

The assignment should be submitted in groups of two students.

1 General Description

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 scalar which is the sum of the current scalar and the argument.
  - **Scalar mul(Scalar s)** accepts a scalar argument and returns a scalar which is the multiplication of the current scalar and the argument.
  - **Scalar neg()** returns the scalar multiplied by (-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.

The following classes extends/implements 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;**

  The class should support the services:

  * **Rational reduce()** returns a new *Rational* which is the 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".

- **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 :

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

- **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 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. 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 negative one, and 0 for zero.
- **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 `xˆ3`.

  For a Monomial with coefficient -1 and exponent 7, *toString()* should return $-$`xˆ7`.

  For a Monomial with coefficient -5 and exponent 4, *toString()* should return $-$`5xˆ4`.

  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 represented 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 not assume number of 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 $5x^0 = 5$.

In addition the class supports the following services:

- **Polynomial add(Polynomial p)** receives a polynomial and returns a new polynomial which is the result of adding the current polynomial with the argument p.
- **Polynomial mul(Polynomial p)** receives a polynomial and returns a new polynomial which is the result of multiplying the current polynomial with the argument p.
- **Scalar evaluate(Scalar s)** receive a scalar and returns a scalar which is the evaluation of the polynomial on the scalar argument.
- **Polynomial derivative()** returns a polynomial which is the derivative of the current polynomial.
- **String toString()** returns a friendly representation sorted by increasing 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 coefficient zero do not appear. For example $1 + \texttt{x} - \texttt{x\^{}2} + \texttt{5x\^{}3} + \texttt{7x\^{}8} + \texttt{24x\^{}24}$.

- **Calculator**
  The Calculator class will contain our main function. you will be given a calculator file, you can run in order to further test your implementation. You can add any additional runs you want to test.

Note: Every class can include getters, setters and clone() if necessary. You can also add more methods to any of the above classes. Keep in mind that all 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**.

- Use the template files associated with this assignment to implement the classes/interfaces *Scalar, Rational, Integer Monomial*, and *Polynomial*.
- Do not use *casting* in any part of your code. (double-to-int casting is allowed).
- **Do not use *instanceof* in any part of your code.**
- Write **unit tests** for each class in your code.

3 Submission instructions

Submit to the Moodle a single archive file (.zip or .tar.gz) in the format [student1ID]-[student2ID].zip with the following contents:

1. classDiagram.pdf - should contains your class diagram modeling the code.
2. hw1.jar - should contain the **source code**, **tests** and **compiled class files**.
3. Use the coding conventions given on the web site.
4. Create packages for your classes. Do not use the default package.