Principles of Object Oriented Programming
Spring 2018
Assignment 2

The assignment should be submitted in groups of two students.

General Description

The assignment goal is to practice the following concepts

- Object-oriented 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 evaluation, and other services. The design should be general, and allow easy modification for any type of scalar. You need to implement the following classes (It may be abstract or interface). You can provide additional methods, if you wish.

1. Scalar

This class mandatorily extends Object, it does not have data members and you may not add any
This class should support the following services

- **boolean isMatch(Scalar s)**:
  Receives a Scalar and returns true if the argument Scalar s is of the same type as the current Scalar, otherwise return false.
- **Scalar add(Scalar s)**:
  Accepts a scalar argument and returns a new scalar which is the sum of the current scalar and the argument. If the Scalar s does not match the current object return NULL;
- **Scalar mul(Scalar s)**:
  Accepts a scalar argument and returns a new scalar which is the multiplication of the current scalar and the argument. If the Scalar s does not match the current object return NULL.
- **Scalar mul(int i)**: (Needed for derivation)
  Accepts an integer argument and returns a new scalar which is the multiplication of the current scalar and the argument.
- **Scalar power(int exp)**:
  Accepts a scalar argument and returns a new scalar which is the current object raised to the power of the exp argument.
- **int sign()**:

Returns 1 for a positive scalar and -1 for a negative one.

The following two classes extend/implement Scalar, they cannot extend any other class.

- **RealScalar:** For Real numbers

  The ONLY data member of this class is
  - **private double v;**

  You may not add more data members

  This class should also implement
  - **String toString()**:

    Returns a string that represents the real number with up to 3 digits after the decimal point. It is highly recommended to use Java formatting functions.

- **RationalScalar**: For Rational numbers. A rational number is a number a/b, where a and b are integers, $b \neq 0$. It should be represented by using two integer fields.

  This ONLY data members of this class are
  - **private int a;**
  - **private int b;**

  You may not add more data members

  This class should also implement
  - **String toString()**:

    Returns a string that represents the rational number.
    1. If a/b is an integer number returns a string representing the value of a/b, otherwise returns the string "(a/b)" (with no spaces and no ") where a and b are the RationalScalar integers.
    2. If a/b < 0 return the rational value with a leading '-'
    3. Do not reduce (לצמצם) the common factor of a rational scalar when printed.
    Example:
    The rational scalar with a=12 and b=-4 should be printed as -3.
    The rational scalar with a=-12 and b=-5 should be printed as (12/5).
    The rational scalar with a=12 and b=-8 should be printed as (-12/8).
    (5/15) * (2/4) should be saved as a=10, b=60 and not a=1, b=6.
    (1/2) + (1/4) should be saved as a=6, b=8 ((4*1+2*1)/(2*4)) and not a=3, b=4

    See the Calculator section for more details.

2. Monomial

This class represents a single polynomial term. The Monomial is represented by a coefficient (Scalar) and an exponent (nonnegative integer). The type of the Monomial is derived from its coefficient, i.e Real or Rational. The polynomial: $4x^2 + 3x - 7$, has 3 Monomials:

1. $4x^2 - coefficient$: 4, exponent: 2
2. $3x - coefficient$: 3, exponent: 1
3. $-7 - coefficient$: -7, exponent: 0

This class mandatorily extends Object
The ONLY data members of this class are

- **private Scalar coe;**
- **private int exp;**

You may not add more data members

This class should support the following services:

- **boolean isMatch(Monomial m)**:
  Receives a Monomial and returns true if the argument Monomial m is of the same type as the current Monomial, otherwise return false.
- **boolean canAdd(Monomial m)**:
  Receives a Monomial and returns true if the argument Monomial m can be added to the current Monomial, otherwise returns false. Two Monomials can be added if they match and have the same exponents.
- **Monomial add(Monomial m)**:
  Receives a Monomial m and returns a new Monomial which is the result of adding the current Monomial and the argument m.
  It is an error to add two Monomials that cannot be added, if the Monomial m does not match the current object return NULL;
- **Monomial mul(Monomial m)**:
  Receives a Monomial m and returns a new Monomial which is the result of multiplying the current Monomial and the argument m.
  It is an error to multiply two Monomials that do not match, if the Monomial m does not match the current object return NULL;
- **Scalar evaluate(Scalar scalar):**
  Evaluates the current Monomial by using the scalar argument.
  For example, $2x^2$ with scalar 2/3, result is $2 * (2/3)^2 = 8/9$.
  The type of the Scalar argument as well as the returned type should be the same as the Monomial coefficient type.
- **Monomial derivative()**:
  Returns a new Monomial which is the result of the derivation on the current Monomial ($2x^2$ will return $4x$). The returned type should be the

same as the current Monomial, i.e. the same coefficient type.

- **int sign()**:
  Returns 1 for a positive coefficient and -1 for a negative one.
- **String toString()**:
  Returns a string that represents the Monomial. If the coefficient of a Monomial with an exponent greater than zero evaluates to 1 it should be omitted, if it evaluates to -1 only the sign should be kept. If the exponent equals to 1 it should be omitted as well.
  Example:
  The polynomial 1-1x^1+2x^2 should be printed as 1-x+2x^2.
  See the Calculator section for more details.

3. 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. All of its Monomials are of the same type. The type of the Polynomial is derived from the type of its Monomials, i.e Real or Rational.

This class mandatorily extends Object

The ONLY data member of this class is

- **private <*Collection*> monomials;** //<*Collection*> can be any standard java Collection or Map

You may not add more data members

This class should support the following services:

- **Polynomial()** {}
  The default empty constructor ONLY!!
  Do not add any other constructor, our main function won't use it.
- **Polynomial build(char type, String input)**:
  Receives the type ('R'(real) or 'Q'(Rational)) and the input string and returns the corresponding Polynomial. The input string is a string of space(s) separated coefficients. 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. See the Calculator section and the attached Main.java file for more details.
- **boolean isMatch(Polynomial p)**:
  Receives a Polynomial and returns true if the Polynomial argument is of the same type as the current Polynomial, otherwise return false.
- **Polynomial add(Polynomial p)**:
  Receives a Polynomial and returns a new Polynomial which is the sum of the current Polynomial with the argument p. The two Polynomials must match, if the Plynomial p does not match the

current object return NULL. The returned Polynomial has to be of the same type as the current one.

- **Polynomial mul(Polynomial p)**:
  Receives a Polynomial and returns a new Polynomial that is the multiplication of the current Polynomial with the argument. The two Polynomials must match, if the Plynomial p does not match the current object return NULL. The returned Polynomial has to be of the same type as the current one.
- **Scalar evaluate(Scalar scalar)**:
  Evaluates the polynomial using the argument scalar. The type of the Scalar argument as well as the returned type should correspond to the Polynomial type. if the Scalar does not match the current object return NULL.
- **Polynomial derivative():**
  Returns the Polynomial which is the result of applying first order derivation ($P'(x)$) on the Polynomial. The returned Polynomial has to be of the same type as the current one.
- **String toString()**:
  Returns a friendly representation sorted by increasing power of the Monomials, for example $1+x-x^2+\cdots$. 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.

4. Calculator

This class will hold the main method and is provided with this specification for your convenience. You may add tests to it but do not change the function signatures since we are going to use a similar one for examining your code.

- A polynomial $a0 + a1x + a2x^2 + a3x^3+\ldots$ is represented by a string of coefficients "$a0$ $a1$ $a2$ $a3$ …" where the coefficient indices represent the power of the monomials.
  Examples:
  - "3 5 -6" is $3+5x-6x^2$
  - "0 0 0 -4 0 7 -2" is $-4x^3+7x^5-2x^6$
  - "2/5 0 -11/200" is $2/5-(11/200)^2$
  - "0 1.5 0 0 -5" is $1.5x-5x^3$
- You can assume valid coefficients string!
- The coefficients are separated by spaces. You can NOT assume any number of spaces between them
- The coefficient is a **Scalar** and can also have a negative value as you can see in the examples above.

- All coefficients of a single Polynomial are of the same type, either Real or Rational however an input line can contain an integer coefficient n even if the Polynomial is Rational or Real.
  If n=0 then the corresponding monomial does not exist in the Polynomial.
  If n ≠ 0 than n should be handled as a Scalar according to the type of the Polynomial.
- Do not reduce (לצמצם) the common factor in a rational number since we are going to check your work automatically by file cooperation.
- It is highly recommended to parse the input using java functions such as String.split(…), Double.parseDouble(…), Integer.decode(…) etc…

Finally, every class can include public getters, setters and clone() if needed. You are also free to add more methods or classes (not static) to any of the above classes but keep in mind that all above methods should stand by their own and might be called from our code (our code = examiner's code)

Running the program

From your command line run " java -jar hw2.jar". This will invoke Calculator.main() and start the program.

DO NOT ADD ANY PRINTS, even not error messages, to your submitted program. Once again, we will automatically compare your output to the expected one so any extra char, even space, will fails your code and result in a grade deduction.

Requirements

1. Design:
   a. Define the components that take part in the system and their responsibilities, excluding the Calculator.
   b. Design an appropriate **UML Class Diagram**.

   To be submitted in a file named hw2.pdf

2. Implementation:
   a. Implement the program according to your design (class diagram) and the instructions.
   b. Each of the classes above has to be in a separate file, we might replace Polynomial.java, Monomial.java or the Scalars files (all three Scalars file together) while checking your work so be sure to implement all the requested methods and respect their signature.
   c. While you are free to add methods you cannot add any new data

members to the classes.

d. The Polynomial, Monomial and Scalar classes, extend Object (by default) you may not add any supper class.
   RealScalar and RationalScalar classes can extend Scalar (or implement Scalar) and they cannot extend any other superclass besides Object.
e. You may not add static classes, global variables, etc…
f. Base your code on object oriented capabilities and avoid using casting, instaceof(), etc… We will accept those but it might cause a grade deduction.
g. Use package/s for your classes and not the default package.

3. Debug
   a. It is highly recommended that you implement classes from bottom to top (From Scalar to Polynomial) and check each class before moving on to the next one.
   b. It is also It is highly recommended to implement and debug the add() and the mul() functions before the evaluate() and the derivative() that can be kept to the end.
   c. In the assignment.zip file you'll find Calculator.java and the expected output. Use them to debug your code.

To be submitted in a file named hw2.jar.
Do not submit Calculator.java, we will bring one of our own.

Submission Instructions

Submit to the CS Submission System **a single archive file (.tar.gz)** with the following contents:

1. hw2.pdf
2. hw2.jar – should contain the **source code and compiled class files** (check that you can correctly run the file hw2.jar from the command line)

בהצלחה