CSC 317: Project 2

Polynomial Calculator II

Total: 100 points

In this project you will extend the Polynomial Calculator that you built for Project 1. The calculator will provide the same services as before, but now allow for two different types of Scalars. So make "Scalar" an **interface** that provides the following services:

1. **Scalar**: This **interface** should support the following services:

   a) Scalar add(Scalar s): accepts a scalar argument and returns a scalar which is the sum of the current scalar and the argument.

   b) Scalar mult(Scalar s): accepts a scalar argument and returns a scalar which is the multiplication of the current scalar and the argument.

   c) Scalar pow(int exponent): accepts an integer argument and returns a scalar which is the power of the current scalar by the exponent.

   d) Scalar neg(): returns a scalar which is the result of multiplying the current scalar by (-1).

   e) boolean equals(Scalar s): returns true if the argument Scalar and the current Scalar have the same numeric value.

   You should then implement this interface in the following ways to allow for two different types of scalars (see the slide 39 in classes & methods):

   - **RealScalar**: For Real numbers. You have already done this in the previous version of Polynomial Calculator, but you probably called it "Scalar". Now call it "RealScalar" (and "Scalar" will now be an interface).
   - **RationalScalar**: For Rational numbers. A rational number is represented as a/b, where a and b are integers, $b \neq 0$. It should be represented using two integer fields in the RationalScalar class. Note that the methods for addition, multiplication, etc., will have implementations that are different from RealScalars, e.g., the addition operation works like this: x/y + a/b = (xb+ya)/yb.

2. **PolyTerm**: You should already have implemented this class to represent a polynomial's term. The coefficient of a PolyTerm should be of the Scalar (now an interface) type. Note that as a client of Scalar, a PolyTerm should not need to know how the Scalar is implemented, i.e., whether it is a RealScalar or a RationalScalar. So, for instance, the `add` method of a PolyTerm may look like this:

```
PolyTerm add(PolyTerm pt) {
   if (this.canAdd(pt)) {
      PolyTerm result = new PolyTerm();
      result.coefficient =
      this.coefficient.add(pt.coefficient);
      result.exponent = this.exponent;//or pt.exponent
      return result;//notice immutability
   }
   else
      return null;
}
```

Notice that both `this.coefficient` and `pt.coefficient` are Scalars, and the Scalar interface indeed provides the `add` service that takes a `Scalar` as input and yields a `Scalar` output. Therefore, `result.coefficient = this.coefficient.add(..)` is a valid operation. The PolyTerm class does not need to know whether the coefficients are actually RealScalar or RationalScalar objects, since both must have implemented the `add` service (in their own ways). At run-time, `this.coefficient` and `pt.coefficient` may, in fact, be RationalScalars, but the principle of *polymorphism*—that allows pointers declared at compile-time to be of a general/interface (Scalar) type to point to actual objects of a special (RationalScalar) type at run-time—means that the above code for `add` will not need to be aware of this run-time fact. Therefore, the changes in this version of Polynomial Calculator should not affect the PolyTerm class much, except for how you handle the user's choice of the Scalar type.

3. **Polynomial**: Since this class is a client of the PolyTerm class (a Polynomial has an array of PolyTerms), not Scalars directly, this class should need almost no change as well. The only change may have to do with how you handle the user's choice of the Scalar type.
4. **Calculator**: This class, as before, will hold the main method. This class should also need almost no change, except for how you handle the user's choice of the Scalar type.

Input/Output

This should be almost same as in the previous version. For RationalScalar operations, there is no need to reduce the fractions when displaying the output (e.g., 4/6 does not have to be printed as 2/3), but if you can do this, then this will earn you **5 extra credit points**.

Running the program

The program starts with showing the menu:

```
Please select an operation:
     1. Addition
     2. Multiplication
     3. Evaluation
     4. Derivative
     5. Exit
```

The user selects the requested operation and then the following message will show up:

```
Please select the scalar field
```

The user will either enter "R" for Reals or "Q" for Rationals. The next step, the user will be asked to enter polynomials (two or one) in separate lines, and returns a result shown in a valid form (no exponent will be shown more than once, and terms with coefficient zero do not appear).

If option 5 is selected, the program exits. Otherwise, the menu is shown again, and the user can continue.

Sample Runs

```
Please select an operation:
     1.Addition
     2.Multiplication
     3.Evaluation
     4.Derivative
     5.Exit
> 1
Please select the scalar field Rational (Q) or Real (R)
> Q
Please insert the first polynomial
> 4+3x^1
Please insert the second polynomial
> 5+2x^1+4x^2
The solution is:
9+5x^1+4x^2


Please select an operation:
```

```
     1.Addition
     2.Multiplication
     3.Evaluation
     4.Derivative
     5.Exit
> 2
Please select the scalar field Rational (Q) or Real (R)
> Q
Please insert the first polynomial
> 1+3x^1
Please insert the second polynomial
> 5+x^2
The solution is:
5+15x^1+x^2+3x^3

Please select an operation:
     1.Addition
     2.Multiplication
     3.Evaluation
     4.Derivative
     5.Exit
> 4
Please select the scalar field Rational (Q) or Real (R)
> Q
Please insert the polynomial
> 5+15x^1+1/3x^2
The derivative polynomial is:
15+2/3x^1

Please select an operation:
     1.Addition
```

```
     2.Multiplication
     3.Evaluation
     4.Derivative
     5.Exit
> 4
Please select the scalar field Rational (Q) or Real (R)
> R
Please insert the polynomial
> 5+15x^1+0.75x^2
The derivative polynomial is: 15+1.5x^1
...
```

What to submit

Submit the following files (**all required**) on Canvas:

1. A .pdf file showing the UML class diagram for the entire project.
2. A .zip file containing the source code (.java files organized possibly in (sub-)directory structures)
3. A runnable .jar file. I should be able to run the .jar from command line like so: `java -jar P2.jar`

Grading Breakdown

20 pts	UML diagram
50 pts	Validity of output on test cases
30 pts	Code organization and readability