

Program #6: Polynomials

Due Date: March 5th, 2007, at 10:00 p.m. MST

Overview: In class, we have just finished talking about interfaces and are about to start talking about list data structures and their representation. Given what you know of Java at the moment, the only representation we can use to hold the members of a list is an array.

You should already be quite familiar with polynomials, thanks to years of math classes. We can consider a polynomial of one variable (x) to be a list of terms, with each term consisting of two parts: a coefficient and an exponent. For example, the polynomial $2x^3 - 1 + x^{-4}$ has three terms, $2x^3$, $-1x^0$, and $1x^{-4}$. In the first term, the coefficient is 2 and the exponent is 3; in the second, they are -1 and 0, respectively. Using an array of term objects as our polynomial representation, we can represent this sample polynomial with an array of just three elements.

As a polynomial is a collection of terms, a polynomial class is likely to have methods dealing with the general issue of term quantity; for example, the number of terms in the polynomial. And, lots of other classes are likely to need quantity-related methods, too. Sounds like a job for ... an interface!

Assignment: Implement the following four components *in separate files*:

1. The class `Term`. A `Term` object represents a term of a polynomial. The implementation details of `Term` are up to you, as only your `Polynomial` class will need to use it. (Of course, we expect that you will create a well-designed, well-implemented, and well-documented `Term` class.)
2. The interface `Quantity`. This interface specifies that these methods be defined by any class implementing it:
 - `boolean isEmpty()` — Returns true if the instance of the `Type` currently has no members, false otherwise.
 - `boolean isFull()` — Returns true if the instance of the `Type` currently has no available space for additional members, false otherwise.
 - `int holding()` — Returns the quantity of members currently held by the instance of the `Type`.
3. The class `Polynomial`, which implements the `Quantity` interface. It is to have just one constructor:
 - `Polynomial()` — Create an empty (termless) `Polynomial`.

The public methods of this class are:

- `Polynomial add(Polynomial p)` — Add the polynomial `p` to the current `Polynomial`, returning a new `Polynomial` object. If both `Polynomials` possess terms with matching exponents, sum those terms.
- `void addTerm(double c, int e)` — Add to the current `Polynomial` a term with coefficient `c` and exponent `e`. If a term with exponent `e` already exists, sum the terms.
- `Polynomial replicate()` — Create a new `Polynomial` that is a copy of the current `Polynomial`.
- `boolean equals(Polynomial p)` — Returns true if `p` has the same number of non-zero terms with the same coefficients and same exponents as this `Polynomial`, false otherwise.

(Continued ...)

- `double evaluate(double x)` — Evaluate this Polynomial on the value `x`. Return the result of the evaluation.
- `double getCoefficient(int e)` — Return the coefficient currently associated with the term with exponent `e`.
- `boolean isEmpty()` — Returns true if the Polynomial currently has no terms with non-zero coefficients, false otherwise.
- `boolean isFull()` — Returns true if the Polynomial currently has no available space for additional terms, false otherwise.
- `int holding()` — Returns the number of terms with non-zero coefficients that are currently in the Polynomial.
- `Polynomial negate()` — Creates a new Polynomial that has the same number of terms with the same exponents as does the current Polynomial, but the signs on the coefficients are switched (positive becomes negative and negative becomes positive).
- `void scalarMultiply(double s)` — Multiply each term of this Polynomial by the value `s`.
- `String toString()` — Create a String representation of the current Polynomial. The polynomial $2x^3 - 1 + x^{-4}$ would be represented by the string `"(2)x ^ (3) - (1)x ^ (0) + (1)x ^ (-4)"`.

4. The class `Prog06`, which will contain your main method. The purpose of `Prog06` is to test the correct operation of the Polynomial class. As usual, write a good set of tests.

Data: For this program, there will be no sample data. After you submit your programs, We will run our version of the class `Prog06` on your Polynomial class. The non-documentation portion of your grade on this assignment will be determined by how well your code passes our testing. (So, be sure that you do a really good job testing your classes!)

Output: Because the output is dependent upon the construction of the `Prog06` class, there is no specific output expected.

Turn In: Electronically submit all of your files (`Term.java`, `Quantity.java`, `Polynomial.java`, and `Prog06.java`) using the usual

[section leader first name]_S[your section #]_P[program #]-[your last name]-[your first name] naming convention.

Hints, Reminders, and Other Requirements:

- Example programs `T05n04.java` and `T05n05.java` use interfaces. Note that they use parameterized (a.k.a. generic) interfaces. You can parameterize `Quantity`, but because its methods don't accept or return `Polynomial` references, you don't need to.
- Because we can copy content from a small array to a larger array as necessary, a `Polynomial` is only "full" when no more memory is available. As we don't have to worry about that, the `isFull()` method will always return `false`. So why include it in the interface? Other kinds collections of quantities do have a maximum size, and for them such a method is useful. This sort of 'optional' interface method isn't unusual; we'll see this idea again when we talk about iterators.
- We recommend that you run your classes through the `Prog06` tests of one or two of your classmates, and use your `Prog06` to help them test their classes. This is permissible so long as you don't share any of the rest of your code. You're also encouraged to suggest testing ideas to one another. When a test uncovers a bug, be sure that you fix it yourself.
- We fully expect that you will have behavior questions about the methods listed; the descriptions are intentionally brief. Think about the issues, and ask questions about them in class, in section, or by email. And, keep an eye on the class announcements page. When a question is being asked repeatedly, we'll add the answer for all to see.