CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

Program #5: Polynomials Reconsidered

Due Date: October 15th, 2015, at 9:00 p.m. MST

Overview: You recently completed (we hope!) your own implementation of the `Polynomial` methods. In doing so, you encountered two significant problems. In this assignment, we'll reconsider the decisions that led to those problems, and will address them both.

- *Problem #1: Arrays.* The first problem you encountered was that you were required to use an array for the storage of the polynomials. Arrays, as we've discussed, don't natively support a lot of operations, which means that you had to do a lot of the dirty work yourself (e.g. shifting of data, if you kept the terms of the polynomials in order by exponent).

  An alternative to doing the implementation from scratch is to adapt another class. We learned about adaptation not too long ago. As a `Polynomial` object is really just a list of `Term` objects, we can employ one of Java's supplied list classes to represent a `Polynomial`'s collection of `Term` objects. This won't be a perfect example of adaptation, because we need operations such as `add()` and `scalarMultiply()` that have no corresponding operation in a general-purpose list class. Still, it should be easier than what you had to do in Program #4, once you become comfortable with the operations of the class being adapted.

  Java has multiple general–purpose list classes that we can adapt. Two of them, `ArrayList` and `Vector`, are very similar. A third, `LinkedList`, stores lists of data using a different internal representation, one that we'll talk about in detail soon. All three implement Java's `List` interface, making them mostly interchangeable.

- *Problem #2: Polynomial's Methods.* A second problem with Program #4 was the Polynomial class, which didn't include the methods needed to cleanly implement operations such as `add()` and `equals()`. Many of you resorted to "peeking" inside the Polynomial objects to get the information you needed. Doing so violates a key program design principle: *information hiding.* The programmer shouldn't ever need to "peek" inside of a data structure. We'll fix both problems in this assignment.

Assignment: Start by making two copies of your `Prog4.java` file; name them `Prog5A.java` and `Prog5B.java` (and adjust the class names within, too!). Also bring along `Quantity.java` and `Term.java`, which, if you did a good job implementing them for the last assignment, shouldn't need much (if any) updating to be reused here. You may update any or all of these files to add more tests, correct logic errors, etc.

Next, create two new polynomial classes in separate files (creatively named `PolynomialA.java` and `PolynomialB.java`). As the names suggest, `Prog5A.java` is to test `PolynomialA` and `Prog5B.java` is to test `PolynomialB`. Both sets of files are to use the same `Quantity` and `Term` classes.

`PolynomialA.java` will implement Program #4's `Quantity` interface plus its Polynomial methods by adapting *either* Java's `ArrayList` class *or* Java's `Vector` class (your choice). Thus, in `PolynomialA` you will be replacing the array object that held the `Term` objects with either an `ArrayList` object or a `Vector` object, and rewriting the implementations of the methods that rely on that representation.

`PolynomialB.java` will do a similar adaptation, using Java's `LinkedList` class instead (no choice here). We haven't talked about linked lists yet, but our coverage of list operations should enable you to use the `LinkedList` class for this task easily, even without knowing what a linked list is. This demonstrates good interface design.

(Continued ...)

So that your implementations of the old polynomial methods in both of these new polynomial classes can better follow the principle of information hiding, We have these additional requirements:

1. **All instance variables in all classes must be private.** No more peeking into other objects to access their state information directly! To make an instance variable private, just prefix its declaration with the keyword "private." A method can still access the private instance variables of its own object. they are private.
2. **Implement one new polynomial instance method: `exponentList()`.** In Program #4, it was hard to cleanly learn which terms another Polynomial object possessed. To learn the terms of another Polynomial object, we'll add this method so that we can ask it for its Term object's exponents:
   - `List exponentList()` — Returns a reference to a List-implementing object that contains Integer objects that hold the values of the exponents of the terms in this polynomial. For example, a polynomial object holding the polynomial $2x^3 - 1 + x^{-4}$ would return a List containing three Integer objects holding 3, 0, and -4.

   With this method and the already-existing `getCoefficient()` method, methods like `add()` should be more straight-forward to write, and able to follow the principle of information hiding. Note that both `ArrayList` and `Vector` implement the `List` interface.

Data: As with Program #4, for this program there will be no sample data. After you submit your programs, we will run our versions of the `Prog5A` and `Prog5B` classes on your new `PolynomialA` and `PolynomialB` classes. The non–documentation portion of your grade on this assignment will be determined by how well your code passes our testing. As usual, make a point of doing a really good job testing your classes (which, if you did a good job of it on the last assignment, should be easy because you can reuse (and augment!) them for this assignment. Reviewing your tests from Program #4 should help you think of new tests to try.

Output: Because the output is dependent upon the construction of the `Prog5A` and `Prog5B` classes, there is no specific output expected.

Turn In: Use the 'turnin' utility to electronically submit your `Term.java`, `Quantity.java`, `PolynomialA.java` `Prog5A.java`, `PolynomialB.java` and `Prog5B.java` files to the `cs127bsXp05` directory at any time before the stated due date and time. Of course, you can turn them in late if you still have late days to use, or don't mind losing 20% per day if your late days are exhausted.

Hints, Reminders, and Other Requirements:

- If you didn't complete the `Term.java` and `Quantity.java` files in Program #4, you'll need to get them functional, as they are reused here. If you had completed them, you should be able to reuse them here without modification . . . if you implemented them correctly!
- Be sure to note, in your external documentation for `PolynomialA.java`, from which class you chose to perform your adaptation: `ArrayList` or `Vector`.

  If you happen to already know how to use `ArrayList`, this is a good opportunity to try `Vector`, and vice-versa. If both are new to you, we recommend adapting `ArrayList`.
- Don't forget to revise your documentation as necessary for this assignment. If you did a good job documenting your Program #4 code, you shouldn't have to make many changes to document this one. (But be sure to make those changes – your SL will be looking at your documentation closely.)
- You're still welcome to share testing code (that is, your `Prog5A/B` classes) with your classmates.
- If you find that you have to extensively modify the files that this assignment shares with the last assignment (that is, `Term.java`, `Quantity.java`, and just about all of `Prog4.java`), take some time to reflect on your Program #4 design decisions. What could you have done differently that would have allowed you to avoid changing those files for reuse in this assignment? Remember, one of the advantages of object-oriented languages is their support for code reuse. Whenever possible, you should try to design your code to be easily reusable.
- You must not store polynomials in arrays in this assignment! The point of the assignment is to get you to work with list objects. We'll be looking at your code to make sure you're using lists.