1. **Goal**
The goal of this assignment is to gain some experience with Java generics, iterators, and some of the classes from the JCF.

2. **Problem Statement** [client’s statement of their need]
Our lads, Sponge Bob and Patrick, have a polynomial system to beat the stock market as long as they can get the maths right! That means adding and multiplying polynomials.

3. **Analysis** [What is the client’s problem?]
Q: (Hey client, ) **What** is a polynomial?
A: *Polynomial* comes from *poly-* (meaning “many”) and *-nomial* (meaning “term”) ... so it’s “many terms.” In this assignment a polynomial is the sum of powers of a variable multiplied by constant coefficients. For example, \(a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0\).

Q: (Hey client, ) **What** do you need to do with them?
A: Three things: add two polynomials, multiple two polynomials, and print out a polynomial.

Q: (Hey client, ) **What** does it mean to add two polynomials?
A: [While I assume you know the answer, if not check out http://www.purplemath.com/modules/polyadd.htm.]

Q: (Hey client, ) **What** does it mean to two multiply polynomials?
A: [While I assume you know the answer, if not check out http://www.purplemath.com/modules/polymult.htm.]

Q: (Hey client, ) **What** kind of interface do you want?
A: None. [For this assignment you can hard code a test driver.]

Q: (Hey client, ) **What** [can you think of any other questions for your client?]

4. **Design** [How will you as a software engineer solve this problem.]
Q: (ask yourself) **How** should I represent a polynomial (i.e., decide on the data structure that you will use.)
A: We will represent a polynomial as a list of terms.

Q: (ask yourself) **How** should this list or ordered?
A: Order the terms from high to low based on the value of their exponents. [Grok the pros and cons of this design decision. Such is midterm-able :) ]

Q: (ask yourself) **How** can I do the addition?
A: One approach is to traverse both lists and examine the two terms at the current iterator position. If the exponent of one is smaller than the exponent of the other, then insert the larger one into the result then advance that list’s iterator. If the exponents are equal, then create a new term with that exponent and the sum of the coefficients, then advance both iterators.

Q: (ask yourself) **How** is a term represented?
A: As its own class. This class should implement the Comparable interface to compare exponents as described above.
Q: (ask yourself) How can I do the multiplication?
A: One approach is to iterate through the terms of the first polynomial multiplying the second polynomial by each term, and then adding up all the products.

Q: (ask yourself) How are there other implementation details that you need to understand before coding?
A: Assignment Requirements [part of being a course rather than part of software development]

• You must use at least two classes from the JCF to store the terms of a polynomial. First write the code using ArrayList and then create a second version of your solution that uses LinkedList. If you what to have more fun! try creating solution that use other JCF classes. Or, if you like, create you own hand-carved collection.
• To traverse the terms of a polynomial you must use iterators. Better solution will use both the for loop version and the hasNext version.
• You must use generic types for the ArrayList and the LinkedList. You will also need generics to make terms Comparable and when iterating over a collection of terms.
• Create your code to be part of a package named asn3.
• For the test plan provide 5 tests including input, expected output, and rationale.
• Use clear documentation and careful formatting. Be consistent in indentation and alignment of braces. Each open brace “{” must be on its own line.
• Each source code file must start with
  // This is my code
  // <Your Name>
  // CS312

Finally, let’s do some empirical data analysis!

Determine the value \(K\) such that the ArrayList version take between 5-10 seconds to run the loop

\[
\text{for (int } i=0; i<K; i++) \text{ poly = poly + poly;}
\]

Record in README.md the actual time taken, and the time that the LinkedList version takes for the same value of \(K\).

Repeat this experiment using first \(\text{poly = poly * term and then poly = poly * poly.}\) Try to explain any differences between the times for ArrayList and LinkedList.

What to hand in

1. A well-formatted 2-up printout of your source code.
2. A GitHub repo that includes (you must use these names as the grading script assumes them!)
   - README.md with the sections plateau schedule and test plan,
   - mycode.pdf (ensure you are happy with the formatting by using xpdf), and
   - your Java source code with main in Tester.java.

Notes

• Consider using the testing tool JUnit (We will starting using JUnit in our fourth week).
• Also, consider using test-driven development: write a test, write code to pass it, repeat.
• Error checking is a plus.
• Reread the general notes regarding style, braces, and header comments.
• While not really the OO way, declare attributes private for now.
• Don’t comment the obvious :)
• Use a tab stop (indent) of either 2 or 4 spaces ... no more!
• You can get the amount of memory that Java is using with \(\text{long memoryUsed = Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()}\)
• You can compute the time taken using \(\text{System.currentTimeMillis()}\) before and after the code to be timed.