1. Goal
The goal of this assignment is to gain some experience with Java generics and iterators, and to have fun empirically comparing classes of 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?]
[By 6am Friday push a 1-2 paragraph summary of the requirements, and optionally a UML class diagram, OOA1, OOA2, and OOD1. I will comment on however much you have pushed.]
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 a polynomial in x and thus is the sum of powers of the variable x 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, multiply 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 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, Squidward has provided a hard coded test driver! (check the repo)]

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 a copy of the larger 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 required by the ordering 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 code using `ArrayList`. After you are satisfied with its performance, replace all occurrences of `ArrayList` with `LinkedList`. As necessary continue to work on the code until it works regardless of your using `ArrayList` or `LinkedList`.
- You must use generic types for the `ArrayList` and the `LinkedList`. You may also need to use generics to make terms `Comparable` (i.e., implement `Comparable<T>` and when iterating over a collection of terms.
- To traverse the terms of a polynomial you must use `iterators`. Better solutions will use the `for` loop version where possible and the `hasNext` version where necessary.
- Each source code file must start with
  ```
  /**
   * This is my code! It’s goal is to ....
   * CS 312 - Assignment 5
   * @author Your Name
   * @version 1.0 9/17/2019 (a version number followed by the date)
   */
  ```
- Your polynomial class must accept the message `degree()`, which returns the degree of the polynomial.
- Finally, let’s do some empirical data analysis! As part of your test driver work out values for $K_i$ such that the `ArrayList` version takes about 5 to 10 seconds per loop to run the loops
  ```java
  for(int i=0; i<K_1; i++)
  poly = poly + poly;
  for(int i=0; i<K_2; i++)
  poly = poly * poly;
  for(int i=0; i<K_3; i++)
  poly = poly * term;
  ```
  Record in README.md the $K_i$’s along with the actual time taken and the time that the `LinkedList` version takes for the same values of $K_i$. In a sentence or two, try to explain any differences between the times for `ArrayList` and `LinkedList`.

**5. 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, test plan, and data analysis*;
   - `UML-class-diagram.pdf` (or .png, but nothing else!),
   - `mycode.pdf` (ensure you are happy with the formatting by using `xpdf`); and
   - your Java source code with `main` in `Tester.java`.

**6. Notes**
- Consider using the testing tool JUnit to help with test automation.
- Reread the general notes regarding style, braces, and header comments.
- You can get the amount of memory that Java is using with `long memoryUsed = Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()`.
- You can compute the time taken using `System.currentTimeMillis()` before and after the code to be timed.
- The grading rubric extends that of past assignments to include correct use of generics and iterators.