



# **CS 315**

## **Software Design**

### **Homework 4**

### **Style Matters!**

### **Due: Oct. 6, 11:30 PM**

#### **Objectives**

- Gaining experience reading code you did not write
- Improving the design decisions concerning how methods are written to improve understandability
- Developing an appreciation for the aesthetics of well-designed software

#### **Style Matters**

It is not uncommon for a software developer to be given a piece of software to maintain that is not well documented, not well written, and where the original developer is unable to answer questions about the software. In that kind of situation, it is often best to start making small improvements while trying to understand the code. Starting with simple things, like checking that variables are declared in the most appropriate scopes and that Eclipse or other tools do not warn of poor style, one can gradually develop understanding of the software. Other ideas are to insert "assert" statements to test whether the hypotheses you are developing about the preconditions, postconditions and class invariants are correct. As you gain understanding, you may be able to insert comments, rename variables and methods and continue to improve the software.

In this assignment, I want you to apply these ideas to improve an existing program so that it is written with better style, exhibiting better design decisions in the way the methods are written. In particular, your submission should be derived directly from the version that I give you. You should not re-think the algorithms used from first principles, but gradually improve the design of the entire program in little steps as suggested by the design guidelines below. When you are done, you should be able to sit back and admire your work, knowing that it is truly a better piece of software, even if, functionally, it does exactly the same thing!

#### **Style Guidelines**

Here are the guidelines you should apply when improving this software:

1. Use a consistent coding style. Eclipse offers lots of support for defining your own code style. See Eclipse menu - Preferences - Java - Code Style, especially Formatter. Once your style is set, use the Format command in the Source menu to format your class for you.
  - Indentation - be consistent

- Only use ++ and -- alone in statements
  - File layout: constants, instance variables, constructors, methods, static methods
2. Use intention-revealing names
    - Proper capitalization.
      - Classes should begin with capital letters. Internal words should be capitalized. Example: `ClassName`
      - Variables and methods should begin with small letters. Internal words should be capitalized. Example: `methodName`
      - Constants should be in all capitals. Internal words should be separated with `_`. Example: `CONSTANT_NAME`
    - Class names should be nouns.
    - Names of methods that modify instance variables should be verbs
    - Methods that just return a value should begin with the word "get", unless the type of the return value is boolean. If boolean, the name should begin with "is". Examples: `getHeight`, `isVisible`.
    - Methods that simply set a single instance variable based on the value of a single parameter should begin with the word "set".
    - Method names should not reveal implementation. Think about the name from the caller's perspective.
    - Introduce constants (static final) to give names to literal values.
  3. Comments
    - Use `/* */` for multi-line non-javadoc comments and `//` for single line comments
    - Use Javadoc comments for every class and its public or protected members
    - Good to do javadoc for every method to be sure that all parameters and return values are documented
    - Only put comments inside a method if it is not obvious what the code is doing. Strive to write code that needs little commenting!
  4. Variables
    - Declare variables in the smallest scope possible.
  5. Methods
    - If a method is too long, it is hard to understand. Decompose it into smaller methods to improve readability.
    - Use private methods to reduce code duplication
    - Chain constructors together. Generally, a constructor with fewer parameters should call a constructor with more parameters, passing in the appropriate default values.
    - Keep all statements in a method at same level of abstraction. Introduce private methods if necessary.
    - Create methods at right level to allow overriding. It is bad if you want most of a method to be the same but only change parts of it in subclass.

## Assignment

An anagram is a rearrangement of the letters in a word or phrase that produces another word or phrase. Go to <http://wordsmith.org/anagram/> to play around with anagrams.

I am providing you with an implementation of a program to find anagrams. The style used in this program is not very good. Your job is to apply the style guidelines above to turn it into a program that is easier to understand and more aesthetically pleasing.

I strongly recommend that you keep the original version of the program unchanged and change just a copy of the program. That way, you can always compare the output your program is producing with the output of the original program.

The program consists of 3 classes:

- `AnagramList` – this is the main class. It gets the phrase to make an anagram of from the command line. It then reads a dictionary from a file and finds a list of anagrams based on that dictionary.
- `Word` – This class is used to hold a word or a phrase representing the original word, a word from the dictionary, or a candidate anagram.
- `WordList` – This class holds a list of words or candidate anagrams.

## Eclipse

Eclipse can provide help with this assignment in a number of ways.

- **Formatting** - As mentioned above, you can select the style you want to use and Eclipse will apply it consistently for you with the Format command.
- **Warnings** - Eclipse can check for many common programming problems for you. You can control what it checks for by opening the Eclipse menu, selecting Preferences. Then open the Java preferences by clicking on the triangle next to Java in the left column. Next open the Compiler preferences and select Errors/Warnings. I generally set all of these to Error or Warning, **except for**:
  - Unqualified access to instance field
  - Access to a non-accessible member of an enclosing type
  - Non-externalized strings
  - Serializable class without serialVersionUID
  - Boxing and unboxing conversions
  - Parameter is never read - only ignore in overriding and implementing methods
  - You can ignore everything in the Annotations section
- **Refactoring** - In the refactor menu, there are many useful commands. You might try to use some of these.
  - **Renaming.** If you want to rename a variable or method, select the rename command and enter the new name. Eclipse will ensure that all uses of that name are changed to the new name and nothing else! It's really much better than find/replace!
  - **Extracting methods.** If you want to decompose a method into smaller methods, select the statements you want to put into a separate method, then select the "extract method" command from the Refactor menu. Eclipse will figure out what parameters and return type the method needs, create the method and replace the statements with a call to the new method!
- **Local history** - Sometimes when making changes to programs, you may wish you could go back to an earlier version. If so, right-click on the class you would like to revert in the Package Explorer. Then select "Compare with..." and then "Local History". You will get a list of timestamps. Double-click on a timestamp and Eclipse will show you the current version side-by-side with the version from the selected timestamp. The leftmost icon above the 2 versions allows you to revert to the timestamped version. The 2<sup>nd</sup> icon allows you to apply just the selected change to the current version. This feature can be a life saver!

## Grading

You are receiving a working program. The program you submit should work identically to the original. That is, given the same input, it should produce identical output. The only changes should be to improve its style.

Consistency	10 points
Naming	10 points
Comments	10 points
Variables	10 points
Methods	10 points
Correctness	50 points

### **Turning in your work**

To turn in your work, create a jar file that contains your source code and the generated html documentation. Submit this on Ella.