Course: CSC 340.05 Toe

Student: Steven McHenry, SFSU ID: 916931878

Teammate: None

Assignment Number: 01

Assignment Due Date & Time: 09-15-2019 at 11:55 PM

Part A

Prompt: Please choose 5 guidelines and discuss them in depth. For each guideline, use at least half a page for your discussion.

Answer:

Guideline 1 – Cohesion:

The basis of cohesion is how well the features and methods of a class belong together in a single class. Do these various members of the class work together when given the objective of accomplishing a task? The goal, in terms of Java classes, it to sustain a high level of cohesion throughout your project. When you are thinking about your program, it is a good first step to decide which classes will need to be made. These classes should have as narrow a set of objectives as possible. If you develop a general "actions" class for example that holds the various methods that you will need to use throughout your program, you are looking at an extremely low level of cohesion. The various methods within this class will likely have little to do with one another and thus will be better off in their own separate classes. Cohesion is good within a program because it makes your code easier to understand and more module, which can allow you to re-use these classes in other programs. A cohesive class is much easier to maintain and can also be scaled up when needed for the program.

Guideline 2 – Encapsulation

The overall goal of encapsulation is to hide the direct state of an object instance and only allow the interaction of that instance through methods which are specifically designed within that class. There are multiple benefits to this paradigm, but the biggest one of all is securing your data from unintended change. The best way to encapsulate your variables within a class is with the "private" modifier, which indicates that the variable cannot be accessed outside of the class. This requires the addition of "getter" and "setter" methods which, understandably, get the intended data of the object instance and set the object instance's variables to a designated value. When referencing an instance's fields you may need to use the "this" keyword to indicate that you are indeed intending to reference the current instance of that object. Another large benefit to encapsulation is the reduction of coupling it adds to your program. Any given class should not rely on the implementation of another class, and encapsulation ensures just that.

Guideline 3 – Consistency

When designing and implementing a program it is extremely important to stay consistent across multiply aspects of your work. A good place to start, when determining how you want your consistency to look, is with a styling guide for your desired language. One might look at the original creator of the language for their styling suggestions, but in some cases there are other programmers who have developed an even better way of maintaining a languages code. Once you have decided on a good styling guide to go off, you want to stick to it and not change halfway through your program. It is much better to have a sub-optimally styled code that is consistent than one that is using the cutting edge of code etiquette, but with different implamentations throughout. A good example of consistency is with naming conventions. If you

are naming certain types of variables one way in your program, and then another in a different part, it can be very difficult to tell what those variables are supposed to be doing.

Guideline 4 – Instance vs. Static

When you are designing a class, one of the first things you should consider is what will be shared between all the instances of the class and what will need to be separate between instances. If you decide that a value will remain the same between all instances, such as a final variable value, that is considered a static variable. These can be useful for code readability as well as memory management since the variable will not have to be created with each new creation of the object. On the other hand, if you need a variable to be unique across different instances, such as the age of a "student" object, then that would be an instance variable. Instance and static are not only applicable states of variables, however. One can also create static methods which can be accessed through a class name without having to instantiate an object of that class. This can be extremely useful for creating simple helper methods that can be used across multiple different classes without messing up your memory with unneeded objects. One thing to keep in mind is that you should never invoke an instance method from a static one.

Guideline 5 – Interfaces vs. Abstract Classes

When first starting out with Java and learning the terminology, it may be confusing as to the difference between and interface and abstract class. At the end of the day they are both describing a class without fully implementing the functionality of that class. The major difference from my point of view is the fact that an abstract class will contain at least one abstract method while an interface will need to contain ALL abstract methods. An abstract method is simply a moth without a body, which will need to be implemented when it is inherited (or extended). Both can be seen as a contract, or blueprint, for a class that will be implemented

down the road. If you know that your subclasses will all need their own unique implementations of the abstract methods, it would probably best to use an interface for that situation.

Part B

1. Program Analysis to Program Design
    a. Prompt: Your analysis of the provided information and the provided sample output. Please think Clients and think Sales.

Answer

When I was first given the provided information for the program, I was a bit confused by the desired end-product. There was the clear goal of a dictionary type product and yet the output was quire limited and repeating unnecessarily in some cases. It was also difficult to wrap my head around the required enumeration and data loading aspects of the project. It was not clear why enums were necessary or how it would al be loaded immediately when the program loaded. After further, careful, reading I did end up getting a hang for the desired result, however.

    b. Prompt: What problem you are solving. Please explain it clearly then define it concisely. Please think Problem Solving.

Answer

The main problem at hand when considering the implementation of this project was how to determine what the user desired when they were entering information into the fields. There were multiple different ways one could enter the desired information with various validations that needed to be incorporated along the way. There was also the problem of creating a dictionary that could be expanded in the future. It is one thing to load static values into a giant list, but you must consider the future where further implementations may be requested.

c. Prompt: How you store data in enum objects. And why. Please think Data Structures and think Data Design.

Answer

You store data in enum objects by first naming an enum class and then indicating a list of static values that are usually in all uppercase characters. This list is, in essence, its own value type that can be used in your program to ensure that only those values are being used when you are created an object of the new enum type you have created. This gives many benefits to the person designing the program initially and people who would be adding on to it at a later date. First and foremost is program security. With an enum type you can be sure that the field it is used within will have one of the predetermined values. Furthermore, it can greatly improve readability when there is a narrower window of options that a variable value can have.

d. Prompt: Which data structure(s) you use/create for your dictionary. And why. Think Data Structures and think Data Design.

Answer

When creating the dictionary, the first step I took was deciding on the data structures I would be using. I finally came up with a combination of ArrayLists and Google Guava MultiMaps. ArrayLists were just so that I could store simple lists while utilizing the utility methods of the build in ArrayList class. MultiMaps were used to that I could create key value pairs while also allowing for a key to have multiple values. This came in handy when I was required to imbed multiple definitions into a single keyword.

2. Program Implementation

Answer

My program does indeed run properly according to the provided sample output and given tests. I did indeed have to slightly tweak the test to maintain consistent formatting across them and the provided sample output, but I made sure to leave the spirit of what they were testing for intact. There are certainly areas of my program that I would like to improve if I had more time. One example is how when you have previously given an expression to the input field and it is waiting for another search value, if you just hit enter it will run the same query from before, which feels like a bug. It also feels quite limited and hard coded when deciding what to do with CSC keywords. I was not able to figure out an elegant way to hand those cases of formatting.