Course: CSC 340.04
Student: Melissa Estrada, SFSU ID: 918591991
Teammate: none
Assignment Number: 01
Assignment Due Date & Time: February-24-2019 at 12:00 AM

Part A OPP Class Design Guidelines

Discussion #1: On five guidelines discussed in depth:

Cohesion in computer science is a very important concept because it is essentially the invisible glue that exists to keep the module together; since programs are composed of one or more independent modules that do not combine, until the program or programs are combined. There are a few forms of cohesion so to elaborate on. First we have functional cohesion a very essential component that acts as a single computation, it is also very functional because it performs the task and functions. Another form is when an element outputs some data that becomes input for another element that is called sequential cohesion and it occurs naturally. Then there is a form of cohesion happens when two elements operate on the same input data or they contribute towards the same output data. There is an important aspect of cohesion it is procedural order of execution. Procedural cohesion makes sure that our actions are connected. For temporal cohesion all of its elements have to execute at the same time. So basically cohesion means there are a lot of tasks being executed at the same time. Sometimes these elements are more so related logically than functionally. Finally some are elements that have no relation at all other than their source code, which makes this the worst form of cohesion. So in computer programing cohesion refers to the degree to which the elements of a module belong together making it a measure of how strongly related each piece of functionality expressed by the source code of its software module is. Cohesion is just an ordinary type of measurement with two distinct parts, having models with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability. Whereas low cohesion is associated with undesirable traits such as being difficult to maintain difficult to test difficult to reuse and even difficult to understand. Cohesion is a concept that was designed to with the idea to reduce maintenance and modification costs of structured design. In a highly cohesive system code readability and the likely hood of reuses is increased while the complexity is kept manageable. Cohesion is increased if the functionalists embedded in the class accessed through its methods have much in common, methods carry out a small number of related activities by avoiding coarsely or unrelated sets of data. Its advantages of high cohesion is the increased understanding of modules, increased ease of maintenance because the logical changes in the domain affect much fewer models and because changes in one module require fewer changes in others. Programers will be able to find or adjust components they need more easily among a cohesive set of operations provided by the module.

Consistency is one of the four properties of a data base, meaning that it should transform the data base from one consistent state to another. The system has a state which includes the

current values of the data and sometimes we are more interested in the consistency of clients caches and other replicas which leads to the connection between the system part and system contents. For example when an operation executes it is not instantaneous because it actually starts when the client/user initiates an operation and it finishes when the system responds. Another form of consistency is operation, which is the operation executed by the user since it contains the operations that indicate whether an operation returns the correct results, for example a search. By specifying an operation the client/user will receive the correct result through the execution of the program. So why is the memory consistency of a processor important for software development? Assuming we have two memory address A and B initially both are 0 and we also have two threads t1 and t2 concurrently operating on these memory address. T1 first stores the value 1 at address A and then loads address B in a local register. While thread t2 first stores 1 at address B and then loads address A again in a local register. Now what are the contents of the local registers r1 and r2 after the execution of these instructions? Well if thread 1 can execute before thread 2 can execute its instructions the contents should be 0 in r1and in r2 similarly if thread executes its instructions before thread 1 excites its instructions the contents should be one in r1 and 0 in r2. After the load it'll be evident that we will have r1 and r2 equal (0,1), (1,0),(1,1) there is no way r1 and r2 can have the value of 0 at the end of the execution but still in practice this can occur. This is the result of the memory consistency model of a processor and this can render an algorithm that is theoretically correct. The memory consistency model basically describes the instruction ordering across addresses that the processor may do. It is important to understand that this model deals with accesses to different addresses and whatever optimizations the processor will do it with disregard to concurrency. With sequential consistency all instructions are executed atomically and there're no reorderings between memory instructions making it the simplest but wont have the best performance.

Encapsulation simply means the action of enclosing something, in an object oriented programming language it is used to refer to one of two related but also very different concepts or ideas usually a combination of both. These features are called classes which refer to the bonding of data with methods that operate on that data. So we use encapsulations to hide the values or state of a structured data object inside a class, preventing others from direct access to them. From there we also have a publicly accessed methods that are generally provided in the class like for example the "getters" and the "setters" are the ones that get to access the values and of course other classes call these methods to retrieve and modify the values inside these objects. When we use encapsulation to hide data we use private methods which only its own methods can directly inspect it or change. But usually there is always a way to over ride this situation by using reflection. The capsule will hold the methods and the variables within the class which means they are hiding in side, wrapping of data and functions into a single unit. It is also known as information hiding concept because the data is not accessible to the outside world since you cannot see what is present and only those functions which are wrapped in the class can access it. Leading to one of the biggest clashes between inheritance and encapsulation, since inheritance breaks encapsulation.

Inheritance means the practice of passing on something may it an object or a situation its something you receive. In computer science it is a way to add functionality to a program basically when extending the class the specialization of the child class extends the functionality of the parent. The child can also redefine the way the parent class behaves by overriding it. Even code that is being used in many cases can be placed in the parent class to be reused. It represents the "is" relationship between all the different classes in a program since it just allows a class to have the same behavior as another class and extends its behavior to provide special actions for specific needs throughout the program. Since all code is part of some class all classes except one must inherit from exactly one other class. The implication of inheritance is that all classes are descendants of an Object. All the classes thus all the objects have a toString, equals, hashCode, clone, and getClass methods. The overridden methods are normally toString, equals, and clone the sub classes gain all of the behavior from methods and data regarding state instance variable of the super class and all of its ancestor classes. The sub classes can add new fields add new methods and override existing methods. The sub classes may not remove fields or remove methods. Even though an object may have instance variables from its parent they may not be accessible by the code of the child class if the fields are private. It is also good to create private and protected classes in the same package to access the data. Since inheritance has to do with related classes so in order to organize related classes of objects we can create a hierarchy of classes. Inheritance is when we write a class thats based on another class taking on their attributes and behaviors. The benefit being the concept of code sharing meaning you can reuse code and not have to rewrite it and the other is you can use them together. So from the superclass we go to the sub class that gets a copy of everything that was in the superclass you can always add more. To override a superclass's member function by writing a new version of that function in a subclass.

Abstract class is used to define a class to gather together behaviors but an object of that type never exist and can never be created or instantiated. A method may be declared abstract in its header, after have a visibility modifier with no body to the method, all derived class must eventually implement this method, any class with one or more abstract methods must be an abstract class. These abstract object are created by keeping common features or attributes to various concrete objects or systems of study. This process uses to data types to perform data abstraction to decouple usage form working representations of data structures within the programs. This concept has two types control abstraction(you don't care how it gets done as long as you get the result) and data abstraction (need to maintain and manipulate data). In a program with all the different modules if the only relation between them are based on the API essentially mean all the modules can be modified independently of each other as long as the API doesn't change modifying one module will not affect anything else in the program. Abstraction is basically saying we're taking something that is really complex and taking a step back to package it up in a way that is simpler from the top and bottom. Making whatever inside the program doesn't matter as long as you can input and receive and output which means we're hiding things.

Part B - Java Programming , Data Structures, and Data Design

Question #1: Analysis, problem, how, and which.

The analysis that I made after receiving the sample output was that I had to think in a much more simpler way. By that I mean, I had to break down my problem into steps that were more attainable and easier to tackle. At first I began my program by creating a package with three classes. One contained my enum data, another my user interface, and the other held my main. After trying to make that process work it turned out to be a much larger program with a lot of problems that was not executing. I had runtime errors which meant my logic was off I was able to use the LinkedList and TreeMap packages but I couldn't figure out how to take the third parameter to work in order to implement the distinct function.

From there I decided to try and make it much smaller I realized at this point that I had way to much going on I was doing to much. I decided to move on and try to break down my first class Dictionary.java and combine my VocabList class into it which held my data within enum. So instead of have a package with three classes I decided to go with one class that included everything. Including the ArrayList, Arrays, HashMap, and Scanner packages to optimize my code these imports would help me make my program simpler but yet comprehensive. I decided I needed to create an ArrayList which implements List, and all list operations that permits all elements , including null. It also allows me to manipulate the size of the array that is used internally to stop my list of data. That I included using enum that serves the purpose of representing a group of named constants. This is where I realized I had to include distinct as into my data instead of trying to hard code it into the program.

From there I implemented HashMap that provides a basic implementation of Map interface that allows me to store data in a Key, Value pair. To access a value all I had to do was know its key. Hashing is a technique of converting a large String to a small string that represents the same string which is exactly what I needed to make my program more efficient. Making a shorter value allowed me to have a faster search. I used the string split method that breaks a given string around matches of the given regular expression. That includes the parameters regex- a delimiting regular expression and Limit-the result of the threshold that throws an exception since the limit of parameters is three values it works perfectly in my program. Using the return this returns the current object instances in the object. I also used the asList so I could receive an output in a list.

Finally I knew I had to implement a user interface which had to know exactly what to pull and check. Using the equalsIgnoreCase method that compares the given strings on the basis of content of the string irrespective of case of the string if any character
is not matched, it returns a false otherwise it returns true. I also used the split method here as well to split the user input so it can be searched. I had to insert a string into another string, the task was to insert another string in between the given string in a particular index. In order to do this I gad to get the string and the index, create a new string, traverse the string till the specified ended and copy this into the new string, copy the strong to be inserted into this new string, copy the remaining characters of the first string into the new string, and return or print the new string.

I was able to minimize my final program to a single class where I used ArrayList and

HashMap to structure my data in a more cohesive way. By using enum I was able to make my data more readable and easily accessible to the rest of my program. It was quite difficult to incorporate the concept of having a distinct call. This part of the program was what had me the most confused and where I struggles the most. By using an array I was able to execute this action as well as using the asList within that implementation.

Part #2: Program Implementation

Analysis #1:
My program did finally work properly.
I would like to learn how to incorporate google gauava I did try to read more into it and did try to add it after I finished to see if it would clean up my program but it only caused bugs. I do want to make it a goal to try and figure that out. Maybe by using the HashBiMap<k,v> I could improve my program since it is backed by two hast tables that allows the implementation of null keys and values that guarantees insertion based iteration order of its keys. I could use the methods because its parameters are the number of expected entries and throws an exception as well. From there it also constructs a new blimp containing initial values from map, The bimap is created with an initial capacity sufficient to hold the mapping in the specified map just like the hashMap I originally used. The interesting part of this is that it provides a get in interface Map<k,v>. That is an object that maps keys to values and can not contain duplicate keys which is exactly what I need. This interface take the place of the Dictionary class which is a complete abstract class rater than an interface. This also provides a collection of three views that allow the mapping of contents to be views as a set of keys, collection of values, or set of key-value mapping. The more I read into it the more it beings to make sense with time I do hope to figure out how to implement it and make this program more condensed if possible and more efficient.