Course: CSC340.05

Student: Kyle Harvey, SFSU ID:

Teammate:

Assignment Number: 02

Assignment Due Date & Time: 09-21-2020 at 11:55 PM

Question **Part A**: Please choose 5 guidelines and discuss them in-depth. For each guideline, use at least one page for your discussion. It is OK to use code to help demonstrate your points. The code portion, if any, should not take up more than 1/3 of each guideline's discussion.

- Answer/Summary/Discussion:

. **Consistency** – When it comes to writing code, it is very important to have consistency a particular java programming style and when it comes to naming conventions. The reason for that is when you're going over the code that you've written it'll make it easier to follow along what each part of the code is doing. Along with naming conventions, it will help with not confusing what the classes, data fields, and methods are doing if it's given informative names. It is important to make the names as consistent as possible, it isn't good to practice choosing different names for similar operations as it will confuse you when calling the wrong operations. Generally, you should provide a public no-arg constructor for constructing a default instance if you start off with no parameters. If a class doesn't support a no-arg constructor, it is helpful to document the reason with a comment to the right of it. When no constructors are defined explicitly, it is assumed that it's a public default no-arg constructor with an empty body. If any case you want to

prevent users from creating an object for a class, you can make the constructor private in the class that way they can't use it. Whatever is made private can only be used within that class.

Clarity – Another thing to add on to the consistency of writing code, is to make sure it's clear as possible in the code that's being written. It goes along with cohesion and encapsulation for being good guidelines for achieving design clarity. When it comes to a class, it should have a clear contract that is easy to explain its purpose and easy to understand in case you need to know what that class is doing when calling it. With clarity users have the flexibility to incorporate classes in many different combinations, orders, and environments, that way it's easier to follow along the code. So if you design a class that doesn't have any restrictions on how or when the user can use it, make sure you design those properties in a way a that allows users to set them in any order and with any order and with any combination of values, and design methods that work independently of their order of occurrence. If methods are defined intuitively then it wouldn't cause any confusion when making the methods. For example:

```
*/

 class Method01 {

    public static void main(String[] args) {
        displayGreeting();
    }

    public static void displayGreeting() {
        System.out.println("Happy Thanksgiving!");
    }
}
/*
Happy Thanksgiving!
*/
```

It clearly says what it's doing as displayGreeting and what it does which is to print out "Happy Thanksgiving!".

Using *modifier* is optional and in most cases omitted. If it's for a valued method then *return type* will be used for returning the value in the method since it's that data type. Also *parameters* in the general form of methods are used to specify values and objects that are inputs. When it comes to declaring a data field, it shouldn't be derived from other data fields.

Instance vs. Static – When it comes to a variable or method that is dependent on a specific instance of the class must be an instance variable or method. So given the variables **jamie** and **friend** reference the same instance of **Name** because it's dependent on that specific instance. A variable that is shared by all the instances of a class should be declared static because it's a static method that does not belong to an object of any kind which is still a member of class and uses the class name instead of an object name to invoke the method. It's helpful to always reference static variables and methods from a class name in improve its readability, that way it avoids any errors. It's important to not pass a parameter from a constructor to initialize a static data field. Instead it's better to use a setter method to change the static data field because it opens to using assignment statements to initialize two variables as methods. When it comes to instance and static being integral parts of object-oriented programming, a data field or method is either instance or static. It is a common design error to mistakenly overlook static data fields or methods because when defining an instance method, it should have been static instead. Usually a constructor is always instance, since it is used to create a specific instance. For a static variable or method, it can be invoked from an instance method, but an instance variable or method cannot be invoked from a static method.

Interfaces vs. Abstract Classes – For both interfaces and abstract classes can be used to specify common behavior for objects, but depending on the situation in the program you would lean on using interfaces or abstract classes. For interfaces, if you expect unrelated classes you would implement in your interface it could be comparable and clone able that are implemented by many unrelated classes. Users that want to specify the behavior of a particular data type, but if they aren't concerned about who implements its behavior then interfaces are better for consideration. Also if they want to take advantage of multiple inheritance of type. If users want to share code among several closely related classes or expect that classes that extend their abstract class having many common methods or fields, or require access modifiers other than public like protected and privates. Also if they want to declare non-static or non-final fields. This enables users to define methods that can access and modify the state of the object to which they belong, in all those cases that apply then abstract classes are the way to go. It seems that Abstract classes and interfaces work opposite of each other. As abstract class can have abstract and abstract methods, interfaces can only have abstract methods and it can have default and static methods as well. Abstract class doesn't support multiple inheritance; interface supports multiple inheritance. Interface has only static and final variables, but abstract class can have final, non-final, static and non-static variables. Abstract class can provide the implementation of interface but interface can't provide the implementation of abstract class.

Encapsulation – It is one of the most fundamental concepts in OOP as it describes the idea of bundling data and methods that work on that data within one unit such as a class in Java. The idea behind it, is that it's often used to hide the internal representation, or state, of an object

from the outside. It is known as information hiding since it's taking an attribute that isn't visible from the outside of an object, and bundle it with methods that provide access to reading or writing it. With that you can hide specific information and control access to the internal state of the object. So when creating a class, it's preferred to use the *private* modifier to hide its data from direct access by clients. By doing so it makes it easier to maintain since *private* doesn't allow other classes to access it. If users are familiar with any object-oriented programming language, then they should know those methods as getter and setter methods. As the names shows, a getter method retrieves an attribute, and a setter method changes it. It depends on the methods that you implement, you can decide if an attribute can be read and changed, or it could be put as read-only or left as not visible at all. The setter method to implement additional validation rules to ensure that the object always has a valid state. You provide a getter method only if you want the data field to be readable, and provide a setter method only if you want the data field to be updateable.

Question **Part B** #1: In at least 1 full page, please explain the following in detail:

▪ Your analysis of the provided information and the provided complete sample output. Please think about Clients and Sales.

▪ What problem you are solving. Please explain it clearly then define it concisely. Please think about Problem Solving and Interviews.

▪ How you store data in enum objects. And why. Please think about Data Structures and Data Design.

▪ Which data structures you use/create for your dictionary. And why. Please think about Data Structures and Data Design.

- Answer/Summary/Discussion: For this program assignment, first I needed to understand what exactly were the requirements. This is important when it comes to the real world working with clients and selling a product, we need know what exactly what they want in order to provide the best service possible. For this program, the object for us to create an interactive dictionary by taking inputs from users and using the input as a search key to look up values associated with the key. There are some restrictions and requirements that we must follow as if a client have needs to meet. Those restrictions and requirements included, no hard coding, the data source need to be stored in a set of enum objects. It also requires an existing data structure or creating new data structures to store the dictionary's data. The program also starts and loads all the original data from the Data Source into our dictionary's data structure. The user interface needs to allow users to input search keys to find what they're looking for in the outputs. Last but not least there's an output that needs to be matched up as to meet the client's expectations. When looking at my program, there are some issues that don't entirely meet the outputs as expected containing errors. The issues lie within any input after the first word, it's supposed show other outputs regarding to the key word but it doesn't process or recognize what it is and ignores it. Although I have my enum set up to output many of those keywords and what its suppose to output, it seems within the loops or arrays it doesn't recognize or store it. There's mostly likely a way to in prove my program to having it working to its extent of producing the correct output if I was able to spend more time on it but as of now there are some issues. Another way this program could have been improved or better is if the program was worked on with a partner or in a real world scenario in a team to go over the program and fill what was missing or how it could have optimized. The data structure that I used for creating this program were lists, multimap, vector, strings, algorithm and constructors. Overall this assignment was a good way to become familiar with C++ by

recreating the same program output from the previous assignment still using data structure, data design, and all the available tools provided in C++

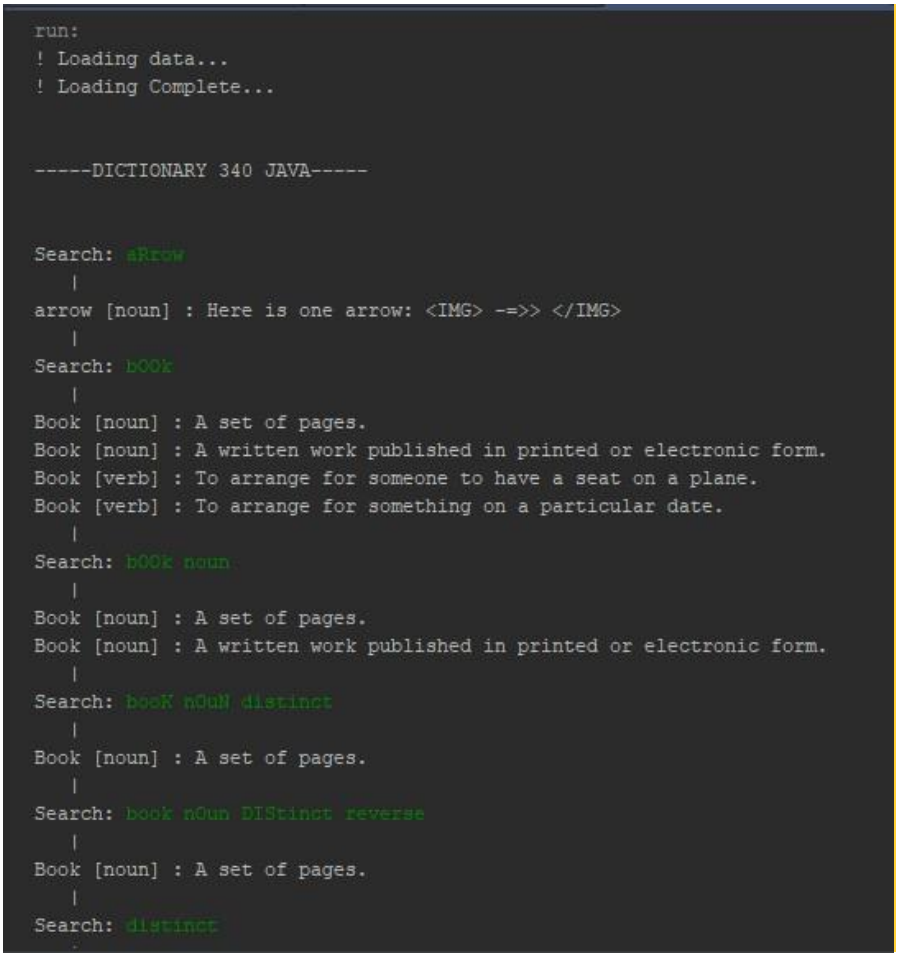
Question **Part B** #2: ▪ Implement your program to meet all the requirements.

▪ In your assignment report, demonstrate your program to your grader/client.

▪ Does your program work properly?

▪ How will you improve your program?

- Answer/Summary/Discussion:



```
   |
Distinct [adjective] : Familiar. Worked in Java.
Distinct [adjective] : Unique. No duplicates. Clearly different or of a different kind.
Distinct [adverb] : Uniquely. Written "distinctly".
Distinct [noun] : A keyword in this assignment.
Distinct [noun] : A keyword in this assignment.
Distinct [noun] : A keyword in this assignment.
Distinct [noun] : An advanced search option.
Distinct [noun] : Distinct is a parameter in this assignment.
   |
Search: distiNCT DISTinct
   |
Distinct [adjective] : Familiar. Worked in Java.
Distinct [noun] : A keyword in this assignment.
Distinct [adverb] : Uniquely. Written "distinctly".
   |
Search: distinct noun distinct
   |
Distinct [noun] : A keyword in this assignment.
   |
Search: distinct noun distinct REVERSE
   |
Distinct [noun] : A keyword in this assignment.
   |
Search: placeHOLDER
   |
Placeholder [adjective] : To be updated...
Placeholder [adjective] : To be updated...
Placeholder [adverb] : To be updated...
Placeholder [conjunction] : To be updated...
Placeholder [interjection] : To be updated...
Placeholder [noun] : To be updated...
Placeholder [noun] : To be updated...
```

```
Placeholder [noun] : To be updated...
Placeholder [preposition] : To be updated...
Placeholder [pronoun] : To be updated...
Placeholder [verb] : To be updated...
   |
Search: placeholder adjective reverse
   |
    <2nd argument must be a part of speech or "distinct">
   |
Search: reverse
   |
Reverse [adjective] : On back side
Reverse [adjective] : Opposite to usual or previous arrangement
Reverse [noun] : A dictionary program's parameter.
Reverse [noun] : A opposite.
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [verb] : Change something to opposite.
Reverse [verb] : Go back.
Reverse [verb] : Revoke ruling.
Reverse [verb] : To be updated...
Reverse [verb] : To be updated...
Reverse [verb] : Turn something inside out
   |
Search: reverse reverse
   |
    <2nd argument must be a part of speech or "distinct">
   |
   |
Reverse [adjective] : On back side
Reverse [adjective] : Opposite to usual or previous arrangement
```

```
Reverse [verb] : To be updated...
Reverse [verb] : To be updated...
Reverse [verb] : Turn something inside out
   |
Search: reverse noun ok
   |
    <2nd argument must be a part of speech or "distinct">
   |
Search: reverse noun ok distinct
   |
    <2nd argument must be a part of speech or "distinct">
   |
Search: reverse noun distinct ok
   |
Reverse [noun] : A dictionary program's parameter.
   |
Search: reverse ok ok ok
   |
    <2nd argument must be a part of speech or "distinct">
   |
   |
    <2nd argument must be a part of speech or "distinct">
   |
Search: reverse reverse ok ok
   |
    <2nd argument must be a part of speech or "distinct">
   |
   |
    <2nd argument must be a part of speech or "distinct">
   |
Search: reverse distinct reverse ok
   |
    <2nd argument must be a part of speech or "distinct">
```

```
Reverse [noun] : A dictionary program's parameter.
Reverse [noun] : A opposite.
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [verb] : Change something to opposite.
Reverse [verb] : Go back.
Reverse [verb] : Revoke ruling.
Reverse [verb] : To be updated...
Reverse [verb] : To be updated...
Reverse [verb] : Turn something inside out
   |
Search: reverse distinct reVERSE
   |
    <2nd argument must be a part of speech or "distinct">
   |
Search: reverse ok
   |
    <2nd argument must be a part of speech or "distinct">
   |
   |
Reverse [adjective] : On back side
Reverse [adjective] : Opposite to usual or previous arrangement
Reverse [noun] : A dictionary program's parameter.
Reverse [noun] : A opposite.
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [verb] : Change something to opposite.
Reverse [verb] : Go back.
Reverse [verb] : Revoke ruling.
```

```
    |
Search: reverse distinct reverse ok ok
    |
    <2nd argument must be a part of speech or "distinct">
    |
Search: reverse adverb
    |
    <Not Found>
    |
Search: facebook
    |
    <Not Found>
    |
Search: !q

-----Thank You-----
BUILD SUCCESSFUL (total time: 7 minutes 59 seconds)
```