Kayvaun Khoshkhou 920357344

Partner: Edreece Afridi 920159263

CSC340.03

Assignment 02

Due 9/20/2020 11:55 PM

Assignment 2 Report

Part A – OOP Class Design Guidelines

Cohesion

The more focused a class is on a task, the more cohesive it is. Cohesion is all about how a single class is designed and it is most closely associated with guaranteeing a class is designed with a single goal in mind. Classes that possess cohesion are much more easily maintained and managed amongst other classes. Sometimes these classes with precise cohesion are even usable in other programs because of their one-track goal-oriented task. Here is an example of very good cohesion within a class:

```
class Addition {

    int a = 3;

    int b = 7;

    public int add(int a, int b)

    {

        this.a = a;

        this.b = b;

        return a * b;

    }

}
```

```
class Output {

    public static void main(String[] args)

    {

        Addition i = new Addition();

        System.out.println(i.add(3, 7));

    }

}
```

This class is very straightforward and serves 2 purposes. It has hard-coded values in this scenario, but you could ask for user input if you needed to and remove the hard-coded values of a and b. The main thing to note is that it takes two values and adds them together. The next is that it prints it to the display. The cohesion displayed within this program is re-usable since just a few tweaks would allow you to use it many ways. The program itself is super simple so it isn't saying much for it to be reusable, but the point of it is to show cohesion.

One example of low cohesion is if there are many different tasks for a single class to complete. For example, if a class is created for signing up for an account on a website, there are many steps that could go into this process. Need for a check on username validity, password validity, email validity, email confirmation., etc. Each of these tasks can be split up into different classes in order to maintain organization and cohesion. However, a common mistake that you may see is the combination of all these tasks and them being put into one class.

Ultimately, the purpose of cohesion is to enable organization to its fullest potential. As well as to enable reusability with classes and to make programming easier for yourself in future work.

Consistency

Consistency is key in java. Without consistency your program will be sporadic and unreliable. There are many attributes that qualify whether a program is consistent or not. White space and spacing is important in readability, and without a proper consistency in those two

attributes, your program will become harder to read. Consistency is also important when it comes to naming conventions, this is one of the most important things as a programmer you want your work to be reliable and interchangeable among the many times it is used from creation and into the future. If your program doesn't have a rhythm or a flow to it, it's almost impossible for you to pick it back up after a few years and continue where you left off if you ever decide to take a break, let alone if someone else was to try and pick up your work from where you left off.

You also need a bare minimum level of consistency in order to have your program work in the first place. Once you set a name for a variable for example you need to consistently use that same name, if you use anything else your program will not compile. It's also not good practice to change names whilst in the middle of a program because you want to be final in your decisions when determining names. Otherwise down the line if you come across new variables you start to lose track of what is what. The rhythm is lost, and the flow of your program becomes halted. Similar operations should be grouped together with consistent naming conventions. Choosing different names for similar operations is bad practice.

With a consistent program, your code is much more predictable. If you use a construct within the code there's a good chance that you'll know why it's there, what it is, where to find it again, etc. This is due to the fact that you will be using it multiple times regardless, so the consistency becomes reliable.

You also want to keep a certain order throughout your entire program. If for example you begin with a certain rhythm of your program like: constants 1st, fields 2nd, constructors 3rd, methods 4th, inner classes 5th, etc, you want to maintain that rhythm and flow throughout the rest of your classes. Most programmers learn about this initially and keep a subconscious step-down rule for all their programs.

Ultimately the greater the consistency of your programming, the greater the readability rises of your program to your audience. A lot of people will lose interest or even disregard your program entirely if it is difficult to read. Therefore, consistency is so important because even if your program is amazing, the quality of your source code will greatly drop amongst your audience when they believe it is unorganized and inconsistent. One of the best traits a good programmer can possess in my opinion is good consistency.

Encapsulation

Encapsulation is extremely important in certain programs as privacy can be a huge issue for clients. Certain attributes of a program need to be kept private from direct access. Examples of these could be passwords or codes. Deciding early on what classes you want to make private or public will make the class easy to maintain as your program progresses. The manipulation of data access is a very important tool within java that every programmer needs to know if they wish to provide a basic level of security in their work. An example of how basic encapsulation can be achieved is by declaring all your variables in a class as private and writing your public methods in the class to set and get the values of those private variables. The data within a class is hidden from other classes and can be accessed only by member functions of their own class which they are declared in.

Some of the main advantages I will highlight from encapsulation are as follows:

Hiding data – Users will not be able to see the inner implementation of the private class. Storing values in the variables will be unseen and will only be able to access the values passed to a setter method. Variables can be initialized with the hidden values.

Reusability – This also falls under consistency and cohesion. The new requirements that come with encapsulation allow it to be used multiple times throughout the entire program.

Flexibility – There are quite a few ways to implement these processes, such as making the variables of a class read-only or write-only depending on what we're trying to accomplish.

If the programmer is aiming to make the variables read-only then you want to avoid setter methods like setSalary() or setWeight(). Or if you're aiming for the latter, and want to make them write-only then you would avoid getter methods like getSalary() or getWeight().

Clarity

When all the class design guidelines come together, in combination they provide excellent clarity. When everything is running smoothly within the program and everything is in working order, your program will maintain a very nice clarity to it. A clear, organized explanation of your

program is a good sign that it possesses clarity. The less restrictions and impositions that your classes place on each other the better clarity you will achieve. When a program is convoluted and filled with classes and methods that are enveloping each other and intertwining, the clarity of the program will decline heavily. Methods need to be defined intuitively without causing confusion. Data fields should not be declared when they can be derived from other data fields. There are many good practices that one can perform in order to improve clarity within a program. One practice I find useful is using enums instead of Boolean where appropriate, because it can bring a lot of clarity into your program. One of the things we have been going over heavily and are even using in this assignment is enums. Enums is a special class that represents a group of constants. In the English language, a word can be spelled the same and have several different meanings. For example, “plane” could be referencing an airplane, or it could be referencing an abstract level of existence (amongst other definitions but I’ll focus on these two). To avoid this in java, using enums defines that variable to a single definition, or value. If we defined plane as an airplane, then the 2nd definition would no longer exist within that enums class.

Take for example: (taken from https://www.david-merrick.com/2017/11/28/good-java-practices-for-clarity/)

```
public static class AccessControl {


   public enum Level {

       PUBLIC, PRIVATE

   };


   private Level access;


   public Level getAccessLevel(){

       return access;

   }

}
```

This program uses enums to set a specific value for public and private. Private will not grant access whereas public will grant access. These values are forever instilled throughout all the points in the program.

Instance vs. Static

An instance in java is a method that requires objects of their own class to be created prior to it being called. Static methods do not need their own objects to be called. The only reference they have is their own class names or the references to the objects of that class. Statics cannot be overwritten but can be overloaded. What that means is within those classes there can be multiple methods with the same name. Similar to constructor overloading which is when a class has more than one constructor with different argument lists. Static methods are made with the purpose to be shared among all objects created from that same class. Within instance methods each individual object that was created in the class has its own copy of the method of that class.

When trying to determine whether a variable should be static you need to ask the following questions. Does the variable describe a specific object? If no, and if it describes all the objects in the class, make it a static variable. If it doesn't describe all the objects, make it a local variable and if necessary, you can pass around as a parameter. However, if it does describe a specific object, then you need to ask if it would make sense to have more than one of these objects. If yes, then make it an instance variable. If no, then you have the option to make it static, but making it an instance variable wouldn't hurt.

As for methods, if you're trying to determine whether it should be static you should begin by asking if it uses any fields of a specific object. If it does you can go ahead and make it an instance method. If not, and it doesn't use any instance methods inside, then you can make it a static method. Otherwise, you can make it an instance method.

Here is an example of a class involving static methods I found on:
https://www.cis.upenn.edu/~matuszek/cit591-2006/Pages/static-vs-instance.html

```
class SomeOtherClass {

	void aStaticMethod() {

	SomeClass.myID = 5; // illegal

	SomeClass.nextID = 5;
```

```
        SomeClass.someStaticMethod(5);

        SomeClass.someInstanceMethod(5); //illegal

        SomeClass thing = new SomeClass();

        thing.myID = 5;

        thing.nextID = 5;

        thing.someStaticMethod(5);

        thing.someInstanceMethod(5);

        }

        void anInstanceMethod() {

          // same as in a static method

        }

}
```

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

1. Program Analysis to Program Design

When beginning this project, we first took a look at the given sample output. This helped us get an idea of how we wanted our program to finish, and we decided to work backwards from there. One of the first things we foresaw as a decisive issue was choosing which data structure to use. We could have gone with any of the following linkedlist, arrays, maps, lists. At first we messed around a little bit with different data structures but ultimately decided to go with Array List because it best served the purpose of storing our data and presenting it in the form of an online dictionary. Which leads into the main issue we had at hand, what are we working towards? We essentially wanted to create a dictionary-like database that would present a set of data based on user input, based on our parameters that we set. We store the data in enum objects by making a variable with 3 String parameters. With this data, we manipulate it into an Array List data structure then proceed to manipulate the data structure to get the output we desire. From a design aspect, we wanted our UI to as closely reflect the sample output as possible. One aspect of the job is to get it working in a raw technical aspect, but the other is to make it look just as

good too. We feel both are equally as important because one without the other doesn't quite simply make the cut. They both must go hand in hand to really have a presentable product. If we simply had a working program without proper readability or an interactive UI, it would not appeal to clients and we would have barely, if not, any sales at all.

2. Program Implementation

Implementation of the program is shown in the zip file. From top down, our program begins with

- Import of Java.util.*

- Enum filled with keywords, constructors, setters, getters, and a toString method.

```
public class CSC340Assignment2 {

    public enum enumKeywords{
        // Adding Keywords into an Enum

        // Arrow keyword with 3 String paramters
        Arrow("Arrow", "Noun", "Here is one arrow: <IMG> -=>> </IMG>"),

        // Four Book Keywords with 3 String parameters
        Book1("Book", "Noun", "A set of pages."),
        Book2("Book", "Noun", "A written work published in printed or electronic form."),
        Book3("Book", "Verb", "To arrange for someone to have a seat on a plane."),
        Book4("Book", "Verb", "To arrange something on a particular date."),

        // Eight Distinct Keywords with 3 String parameters
        Distinct1("Distinct", "Adjective", "Familiar. Worked in Java."),
        Distinct2("Distinct", "Adjective", "Unique. No duplicates. Clearly different or of a different kind."),
        Distinct3("Distinct", "Adverb", "Uniquely. Written 'distinctly'."),
        Distinct4("Distinct", "Noun", "A keyword in this assignment."),
        Distinct5("Distinct", "Noun", "A keyword in this assignment."),
        Distinct6("Distinct", "Noun", "A keyword in this assignment."),
        Distinct7("Distinct", "Noun", "An advanced search option."),
        Distinct8("Distinct", "Noun", "Distinct is a parameter in this assignment."),

        // Eleven Placeholder Keywords with 3 String parameters
        Placeholder1("Placeholder", "Adjective", "To be updated..."),
        Placeholder2("Placeholder", "Adjective", "To be updated..."),
        Placeholder3("Placeholder", "Adverb", "To be updated..."),
        Placeholder4("Placeholder", "Conjunction", "To be updated..."),
        Placeholder5("Placeholder", "Interjection", "To be updated..."),
        Placeholder6("Placeholder", "Noun", "To be updated..."),
        Placeholder7("Placeholder", "Noun", "To be updated..."),
        Placeholder8("Placeholder", "Noun", "To be updated..."),
        Placeholder9("Placeholder", "Preposition", "To be updated..."),
        Placeholder10("Placeholder", "Pronoun", "To be updated..."),
        Placeholder11("Placeholder", "Verb", "To be updated...");
```

```
// Fifteen Reverse Keywords with 3 String parameters
Reverse1("Reverse", "Adjective", "On back side."),
Reverse2("Reverse", "Adjective", "Opposite to usual or previous arrangement."),
Reverse3("Reverse", "Noun", "A dictionary program's parameter."),
Reverse4("Reverse", "Noun", "Change to opposite direction."),
Reverse5("Reverse", "Noun", "The opposite."),
Reverse6("Reverse", "Noun", "To be updated..."),
Reverse7("Reverse", "Noun", "To be updated..."),
Reverse8("Reverse", "Noun", "To be updated..."),
Reverse9("Reverse", "Noun", "To be updated..."),
Reverse10("Reverse", "Verb", "Change something to opposite."),
Reverse11("Reverse", "Verb", "Go back."),
Reverse12("Reverse", "Verb", "Revoke ruling."),
Reverse13("Reverse", "Verb", "To be updated..."),
Reverse14("Reverse", "Verb", "To be updated..."),
Reverse15("Reverse", "Verb", "Turn something inside out.");
```

Within the enum we filled out the necessary keywords to access basic implementation of the assignment.

Here you can also find the constructors that we created to access with the enum.

```
// Constructors for Enum
private String eKey;
private String ePOSpeech;
private String eDefinition;
```

Here are the setters and getters we created to allow the program to grab the user's request and mesh with the code to yield an output.

```
//Enum Method Setters
private enumKeywords(String eKey, String ePOSpeech, String eDefinition){
this.eKey = eKey;
this.ePOSpeech = ePOSpeech;
this.eDefinition = eDefinition;
}

// Enum Getters for the next 3 methods
public String geteKey() {
    return eKey;
}
```

```
// Enum Getters for the next 3 methods
public String geteKey() {
    return eKey;
}

public String getePOSpeech(){
    return ePOSpeech;
}

public String geteDefinition(){
    return eDefinition;
}
```

Here is the ToString method we used to output the user's request.

```
// ToString method for output
@Override
public String toString() {
    return this.eKey + ", [" + this.ePOSpeech + "] , " + this.eDefinition;
}
```

After this you will find our Main method where we begin printing out the enum Keywords using our Array List. At the end you can see we were testing the reverse method

```
    public static void main(String[] args) {

        // to make an array to take values from the enum
        enumKeywords[] wordArray = enumKeywords.values();

        // System.out.println(wordArray); Testing to see output, not important

        // to make an ArrayList with the enum in mind
        ArrayList<enumKeywords> wordArrayList = new ArrayList<enumKeywords>();

        // for variable "keys" in enumKeywords, fill with content from wordArray
        for(enumKeywords keys : wordArray){
            wordArrayList.add(keys);
        }
        System.out.println(wordArrayList);

        // Checking clear method to see if it works, erases all content inside ArrayList
        wordArrayList.clear();

        // Scanner Object to test user input
        Scanner s = new Scanner(System.in);
        String userinput = s.next();

        // For Loop to add content in array into ArrayList + comparing data at lower case
        for (enumKeywords keys : wordArray){
            if(keys.geteKey().toLowerCase().equals(userinput.toLowerCase())){
                wordArrayList.add(keys);
            }
        }

        // Testing reverse method
        System.out.println(wordArrayList);
        wordArrayList = CSC340Assignment2.reverse(wordArrayList);
        System.out.println(wordArrayList);
//        System.out.println(wordArrayList);

    }
```

Here is our distinct method where we created the necessary implementation to make the keyword distinct act accordingly in the dictionary.

```
// Making Distinct Method to get unique definition output
public static ArrayList<enumKeywords> distinct(ArrayList<enumKeywords> array){
    // Creating new ArrayList
    ArrayList<enumKeywords> checkDistinct = array;
    String aSpeech;
    String aDef;
    String bSpeech;
    String bDef;
    // test variable
    ArrayList<Integer> flag = new ArrayList<Integer>();

    // For loop to go through enum and get the second and third String
    for(int i = 0; i < checkDistinct.size() - 1; i++) {
        aSpeech = checkDistinct.get(i).getePOSpeech();
        aDef = checkDistinct.get(i).geteDefinition();
          // Same for loop
        for (int h = 1; i < checkDistinct.size(); h++) {
            bSpeech = checkDistinct.get(h).getePOSpeech();
            bDef = checkDistinct.get(h).geteDefinition();
            // If statement to compare
            if(aSpeech.equals(bSpeech) && aDef.equals(bDef)){
                flag.add(i);
                break;
            }
     }
    }
```

After this we implemented a for loop to take care of and remove any similar objects we would get as an input for our unique value.

```
    // For loop to remove similar objects to get unique value
    int counter;
    for(int c = flag.size() - 1; c >= 0; c--){
        counter = flag.get(c);
        checkDistinct.remove(counter);
    }
    return checkDistinct;
}
```

And Finally we have our reverse method where our keyword would be reiterated in a reverse fashion as required by the sample output. This one took a bit of tweaking to get working correctly. Calling it in the main required the main project class (See above).

```
// Reverse method, returns the called keywords backwards. Iterated backwards**
public static ArrayList<enumKeywords> reverse(ArrayList<enumKeywords> array){
    // make new ArrayList using enum
    ArrayList <enumKeywords> toReverse = new ArrayList<enumKeywords>();
    // transverse the arrayList and iterate it backwards
    for(int i = array.size() - 1; i >= 0; i--) {
        toReverse.add(array.get(i));
    }
    // easy return
    return toReverse;
}
```

- UI missing, could make an insane impact if we actually created one. We spent too much time on the hard parts (Distinct Method and manipulating data for reverse, etc) to actually implement the ui for cleaner code.

Our program does not operate at 100% due to the fact that our UI was incomplete. If we were able to get this working I'm sure our program would be functioning at maximum efficiency and capacity as the assignment required.

In order to improve our program, as said prior to this, our UI would need to be implemented properly. But hypothetically if we had gotten that to work,then the next step would be to improve the functionality and reach of the database in order to make a more realistic dictionary. We could also incorporate other methods to reach out and acquire information if we could implement other forms of programming such as web based programming. We would have access to a near infinitely large database of information from a variety of languages as well. Of course these are abstract and very far-reaching goals but I think in terms of programming it should be within our reach in a few years! If we were to ever work on a project similar to this, I'm sure we would refer back to this assignment to get things rolling at the beginning.