Assignment 2 Part A

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 require 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 over written 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 object, 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

}

}