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.