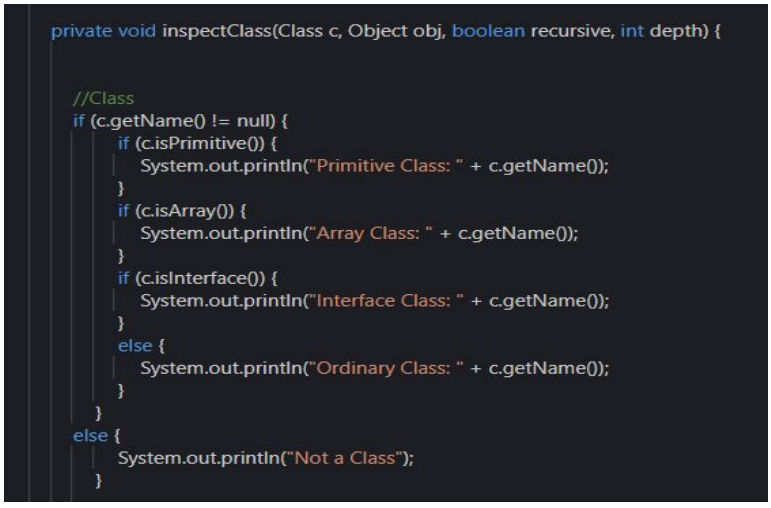
Brandon Sin 30012020, CPSC 501 Assignment2

Long Method:

Over the course of the assignment I see myself solely coding within the ‘inspectClass’ method. This method is long and difficult to read. As a result, I took out the logic for determining specific parts in the assignment, for example identifying Class, superclass, etc. The pictures below is the extraction for the logic of finding the Class. I removed the entire section of the code and placed it in a new method called ‘getClass’. The purpose of ‘getClass’ is solely on determining the entity and the name of the Class. This helps with readability and if I were to edit the method, I only need to focus on editing ‘getClass’ rather than the entire ‘inspectClass’ method. Therefore, reducing the chance of mistakably changing something that does not relate to identifying the Class. When it came to true or false for the recursive fields. I created another method solely to print the fields based on if the boolean is true or false. This reduces the inspectorClass.

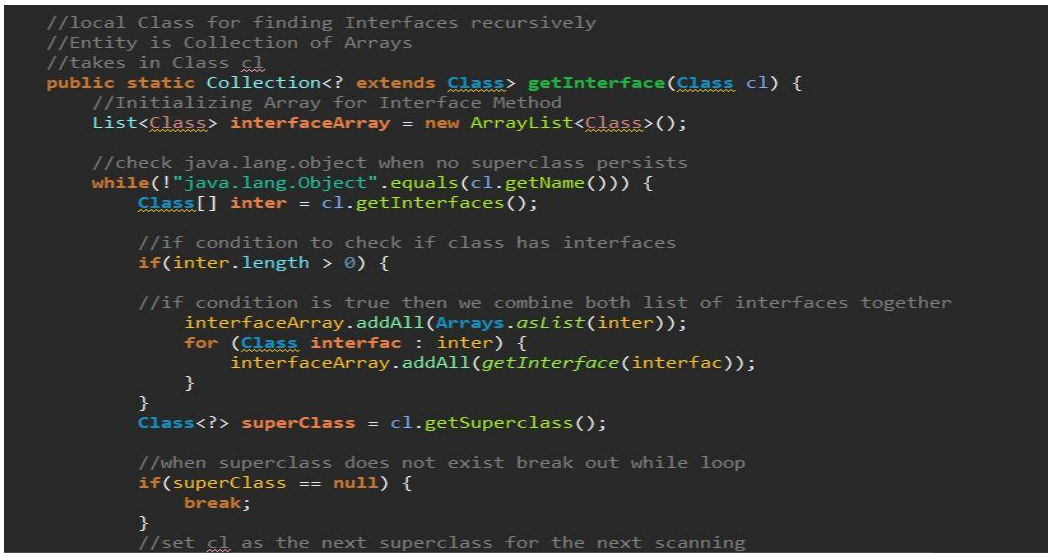
Altered Code



Original code

Comment:

Proper commenting and commenting at all is important as not everyone can understand each other. Some methods such as the recursions for interfaces and superclasses were more convoluted compared to other methods. These recursion functions require comments to help with understandability.



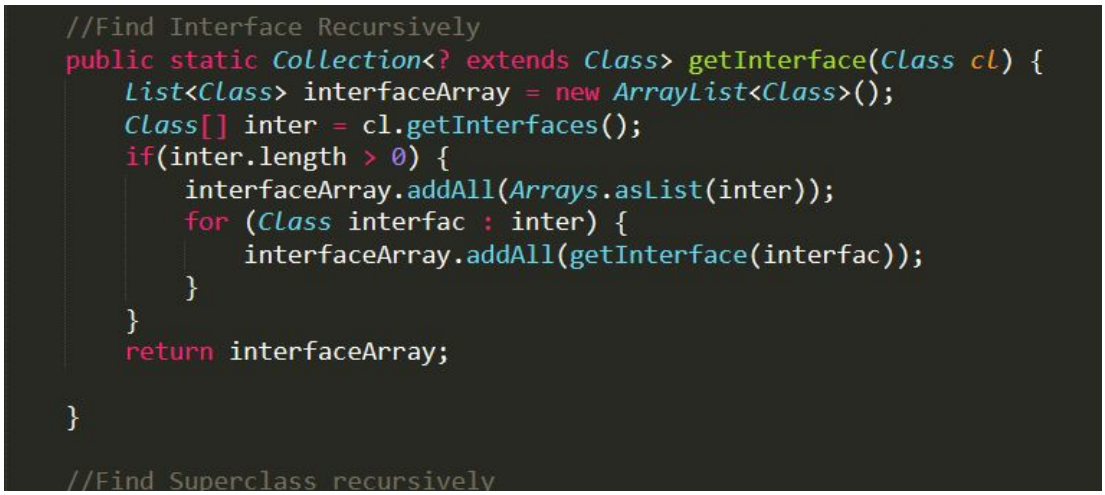
```
//local Class for finding Interfaces recursively
//Entity is Collection of Arrays
//takes in Class cl
public static Collection<? extends Class> getInterface(Class cl) {
    //Initializing Array for Interface Method
    List<Class> interfaceArray = new ArrayList<Class>();

    //check java.lang.object when no superclass persists
    while(!"java.lang.Object".equals(cl.getName())) {
        Class[] inter = cl.getInterfaces();

        //if condition to check if class has interfaces
        if(inter.length > 0) {

        //if condition is true then we combine both list of interfaces together
            interfaceArray.addAll(Arrays.asList(inter));
            for (Class interfac : inter) {
                interfaceArray.addAll(getInterface(interfac));
            }
        }
        Class<?> superClass = cl.getSuperclass();

        //when superclass does not exist break out while loop
        if(superClass == null) {
            break;
        }
        //set cl as the next superclass for the next scanning
```



```
//Find Interface Recursively
public static Collection<? extends Class> getInterface(Class cl) {
    List<Class> interfaceArray = new ArrayList<Class>();
    Class[] inter = cl.getInterfaces();
    if(inter.length > 0) {
        interfaceArray.addAll(Arrays.asList(inter));
        for (Class interfac : inter) {
            interfaceArray.addAll(getInterface(interfac));
        }
    }
    return interfaceArray;

}

//Find Superclass recursively
```