Lab 0

1. When lots of parameters are sent between methods, *data coupling* occurs. If the number of parameters in a method is more than three, we may need to consider creating objects of parameters and pass the object as a parameter to the method.

Please examine the Java code below (see **Figure 1**) and find the method that introduces *data coupling.*

```java
public class DataCoupling {
    int numberA = 1;
    int numberB = 2;
    int numberC = 3;
    boolean allNumbersSet = true;
    Printer printer = new Printer();
    Sum mySum = new Sum();

    public void firstCaller() {
        printer.print(numberA, numberB, numberC, allNumbersSet);
    }

    public void secondCaller() {
        mySum.sum(numberA, numberB);
    }

    public static void main(String[] args) {
        DataCoupling myDataCoupling = new DataCoupling();
        myDataCoupling.firstCaller();
        myDataCoupling.secondCaller();
    }
}

public class Printer {
    public void print(int numberA, int numberB, int numberC, boolean
allNumbersSet) {
        System.out.println("Number A: " + numberA);
        System.out.println("Number B: " + numberB);
        System.out.println("Number C: " + numberC);
        System.out.println("All numbers set? " + allNumbersSet);
    }
}

public class Sum {
      public void sum(int numberA, int numberB) {
             int result = numberA + numberB;
             System.out.println("sum: " + result);
      }
}
```

Figure 1: Example Java code that contains data coupling.

1.1. To remove the *data coupling,* do the following modifications in the Java code given in **Figure 1**:

(1) Create an object that contains the problematic method's parameters
(2) Modify all methods, so that each method takes the object as parameter

1.2. Do you think *data coupling* is removed in the resulting code you came up with after implementing the modifications described in **1.1**? Is there any other type of coupling introduced in the resulting code? Please briefly explain.

1.3. Start over modifying the code given in **Figure 1.** This time use the following alternative way: break the problematic method into smaller methods and pass lesser number of parameters to each method.

1.4. Do you think *data coupling* is removed in the resulting code that you came up with after implementing the modifications described in **1.3**? Is there any other type of coupling introduced in the resulting code? Please briefly explain.

2. *Content Coupling* is observed when one module or class accesses and modifies the data of another module or class. This type of coupling can be reduced by having private instance variables and using getters and setters.

In the Java code shown in **Figure 2**, Class `ContentCoupling` has an instance member `int age;` Class `CouplingExample` sets the value of `age` directly by `contentCoupling.age = 15;` This is *content coupling*.

2.1. Modify the code shown in **Figure 2** as follows: Rename variable `age` to `ageLived` or change the data type of `age` from `int` to `String`.

2.2. Modify Class `ContentCoupling` so that the code can execute.

2.3. Now imagine if `age` were used by 20 classes. We would need to make changes (such as the one you did in **2.2**) everywhere. To mitigate such situations, one solution is to use getters and setters. Do the following changes in the Java code shown in **Figure 2:**

- change type of `age` from `int` to `String` in Class `ContentCoupling;`
- to reduce content coupling:
  - implement getter and setter in Class `ContentCoupling;`
  - modify Class `CouplingExample` so that the resulting Java code can execute

```java
public class CouplingExample {
       public static void main(String[] args) {
              ContentCoupling contentCoupling = new ContentCoupling();
               contentCoupling.age = 20;
               System.out.println("age:"+contentCoupling.age);
       }
}
       class ContentCoupling {
           public int age;
       }
```

Figure 2: Example Java code that contains content coupling.