# SE (M) 2018-2019 Lab 8: Design Principles - SOLUTION

Revision: v2019a

## Identifying issues:

There might be more coupling in this code than listed here, but this is a start.

**`Course.java`**
Instance variables have the access modifier "public". Content coupling. Otherwise there's not much going on here.`

**`MastersCourse.java`**
As above. Content coupling. The includesGroupWork variable isn't set in the constructor. This leads to routine coupling in main. There's flag coupling at line 19. Routine coupling with these unnecessary methods at lines 31 to 39

**`Lecturer.java`**
There's import coupling on line 1. Then more content coupling with our instance variables (or at least scope for it). There's data coupling in the second constructor. This class appears to be logically cohesive.

**`Student.java`**
There's import coupling one line 1. Some scope for content coupling in the instance variables (name, courses). There's common coupling on line 15, there's no need for this counter to be global. Data coupling in the constructor. The countMastersCourses method uses this commonly coupled variable n. On line 55 we needlessly place a method in a utility class (does this count as type coupling?). More use of this commonly coupled variable n on line 57. Apart from placing this single method in a utility class this appears again to be (slightly?) logically cohesive.

**`Utils.java`**
Again import coupling on lines one and two. Line two is erroneous. Data coupling on line 6. This class is only coincidentally cohesive (not a term from the book!) since there's just random stuff chucked in here. This is our application wide common coupling dump.

**`Main.java`**
Import coupling on line 1. Type coupling when we use Student and MastersCourse objects. Data coupling in their constructors. Routine coupling on lines 16-17 and 19-20. Flag coupling on lines 17 and 20. Stamp coupling in both the helper methods in this class. Content coupling at lines 34, 36, 38, 40, 42 and 49. It might be argued that this class has procedural cohesion. Misc The Course and MastersCourse classes shouldn't be copy pasted like this. There isn't use of packages to group classes into a larger cohesive unit.

## Fixes and refactoring

Proposed solution: Do away with the utility class. Make a design choice between stamp and data cohesion. Move the helper methods from Main into Student. Use getters and setters instead of accessing class variables directly. Have MastersCourse extend an abstract class Course. Remove the routine cohesion involved in creating a MastersCourse object by including

a constructor which takes a boolean for the includesGroupWork variable. Just remove the flag coupling in this class. There's probably more.

So is it worth attempting to fix these all of the code issues or should we simply start over? Since this a simple example it is probably easier to start over and reimplement everything from scratch. Things are however rarely this simple and an actual refactoring, e.g. class by class, is often the only realistic solution.