COMP 345 Week 7 Modularity

h_lai@encs.concordia.ca
jyotsana.gupta@mail.concordia.ca

Scenario

1. assume you are asked to build a MyConcordia System
2. you are working on a task to provide a functionality of program CGPA calculation

Note:

1. a student may enroll in two or more programs (eg. qualifying program, master program)
2. we want to calculate the CGPA for each program separately (CGPA of master program)

A Possible Approach

1. take a student ID as input
2. get the student record (all the course he/she has taken) from database
3. filter the courses related to the targeted program
4. according to some formula, calculate the CGPA

It seems that all this code can be done in only one function.

```
int main( ) {

    id          = ask_from_user( );
    record   = get_student( id ).get_record( targeted_program );
    cpga      = cal_cgpa( record );

    return 0;
}
```

Good Design and Bad Design

There are various ways to implement a functionality, but we need to know which approach is good and which is not good.

The design of the CGPA calculation in the previous slide is NOT a good approach. Because it put everything together which we referred as high coupling. What happen if one day I need to add a new functionality.

For example, I want to calculate the total credits a student earn so far. You have to break down the previous design and add new codes inside of it.

Rule of good design: low coupling and high cohesion.

Modularity

Modular design, or "modularity in design", is a design approach that subdivides a system into smaller parts called modules or skids, that can be **independently created and then used** in different systems.

A modular system can be characterized by functional partitioning into discrete scalable, reusable modules; rigorous use of well-defined modular interfaces; and making use of industry standards for interfaces.

Object-oriented Programming (OOP)

Assign the behavior (method) to an object, objects can communicate with each other through message.

In our case, the CGPA should be an attribute of Program and Program should be assigned to Student.

Come back to the CGPA Calculator

Instead of putting everything in main(), let do it in an OOP way and logically separate code into sub module!

What kind of module (class, in this case, if can be package or sub project when we develop large software) we can have in that system?

1. Course
2. Program
3. Student

Course

- name
- credit

Maybe it has a bunch of concrete sub classes:

>> UgradCourse

>> GradCourse

>> ProCourse

Program

- List <Course> : a list of course that student has finished
- id
- name
- advisor
- … …
- finish_course(course)
- getCGPA(): return the CGPA of that program

Student

- List <Program> : a list of program the student enrolled
- id
- name
- getCGPA(Program)
- finish_course(program, course)

Driver (test environment)

Rule: In order to demo or for testing purpose (you have to show your functionality is implemented correctly), you should set up a scenario. You don't need to add any code inside the implementation part!

int main() {

1. create maybe two program (qualifying, master) objects
2. create a student object
3. create two lists of finished course, assign them to corresponding program using student.finish_course(program, course) method
4. invoke the student.getCPGA() method, compare the result return from that method and the expected result

}