NATIONAL UNIVERSITY OF SAN AGUSTIN
SYSTEM ENGINEER SCHOOL

TO - Lab 04 - Coupling and Cohesion

Christian E. Portugal-Zambrano

October 09, 2018

1 MODULARITY

A software construction method is modular, then, if it helps designers produce software systems made of autonomous elements connected by a coherent, simple structure [?] Chapter 3.

1.1 FIVE REQUIREMENTS

A design method worthy of being called "modular" should satisfy five fundamental requirements, explored in the next few sections:

- Decomposability.- if it helps in the task of decomposing a software problem into a small number of less complex subproblems, connected by a simple structure, and independent enough to allow further work to proceed separately on each of them.
- Composablity.- if it favors the production of software elements which may then be freely combined with each other to produce new systems, possibly in an environment quite different from the one in which they were initially developed.
- Understandability.- if it helps produce software in which a human reader can understand each module without having to know the others, or, at worst, by having to examine only a few of the others.
- Continuity.- if the software architectures that it yields, a small change in a problem specification will trigger a change of just one module, or a small number of modules.

- Protection.- it yields architectures in which the effect of an abnormal condition occurring at run time in a module will remain confined to that module, or at worst will only propagate to a few neighboring modules.

1.2 Five rules

- Direct Mapping.- The modular structure devised in the process of building a software system should remain compatible with any modular structure devised in the process of modeling the problem domain.
- Few Interfaces.- Every module should communicate with as few others as possible.
- Small Interfaces.- If two modules communicate, they should exchange as little information as possible (Weak Coupling).
- Explicit Interfaces.- Whenever two modules A and B communicate, this must be obvious from the text of A or B or both.
- Information Hiding.- The designer of every module must select a subset of the module's properties as the official information about the module, to be made available to authors of client modules.

1.3 Five principles

- The Linguistic Modular Unit.- Modules must correspond to syntactic units in the language used.
- Self-Documentation.- The designer of a module should strive to make all information about the module part of the module itself.
- Uniform Access.- All services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation.
- Open-Closed.
- Single choice.- Whenever a software system must support a set of alternatives, one and only one module in the system should know their exhaustive list.

Remember that all the theory and rules presented here are just guidelines!

2 Pre-requisites

For this practice you must have the implementation hierarchy of the structures SList, DList, CList, Stack, Queue and Tree, abstracts classes and interfaces necessary too. We need a tool for UML modeling, you can use StarUML.

3 Coupling and Cohesion

The problem of coupling and cohesion is based on where is focused(Class or Module) and the intensity identified (Low and High). The rule of thumb is to maintain Low coupling and High Cohesion but is necessary to specify where it must be used.

- **Inside** a class is necessary that all elements accomplish just one objective, so they must be high cohesioned.
- **Between** classes (Module) is necessary that they share as little information as needed, so they must be low coupled.

4 Warm up

Ok, may be you are a little confused, let's resolve these questions:

- Inside a class, when we have low cohesion?.
- Between classes, when we have high coupling?.
- Remember the code of a list from week01, what can you say about this?

```
class List{
private:
   void insert(T _data); //insert an element
   T at(int _pos); //get element at _pos

public:
   List(); // constructor
   List(const List& _list);
   ~List(); // destructor

   void erase(T _data);    //erase and element if exist
   void clear();  //clear all the list
   void remove(int _pos); //remove element at _pos
   void reverse(); //reverse the entire list

   bool isEmpty(); //true if is empty
   bool save(std::string); // save to a file
   bool load(std::string); // load from file
   unsigned int size(); //size of the list
   void show(); //show the list to console
   T operator [] (int); //overload []
   List<T>& operator << (T); // overload <<
private:
   Node<T>* proot;
   int Size;
};
```

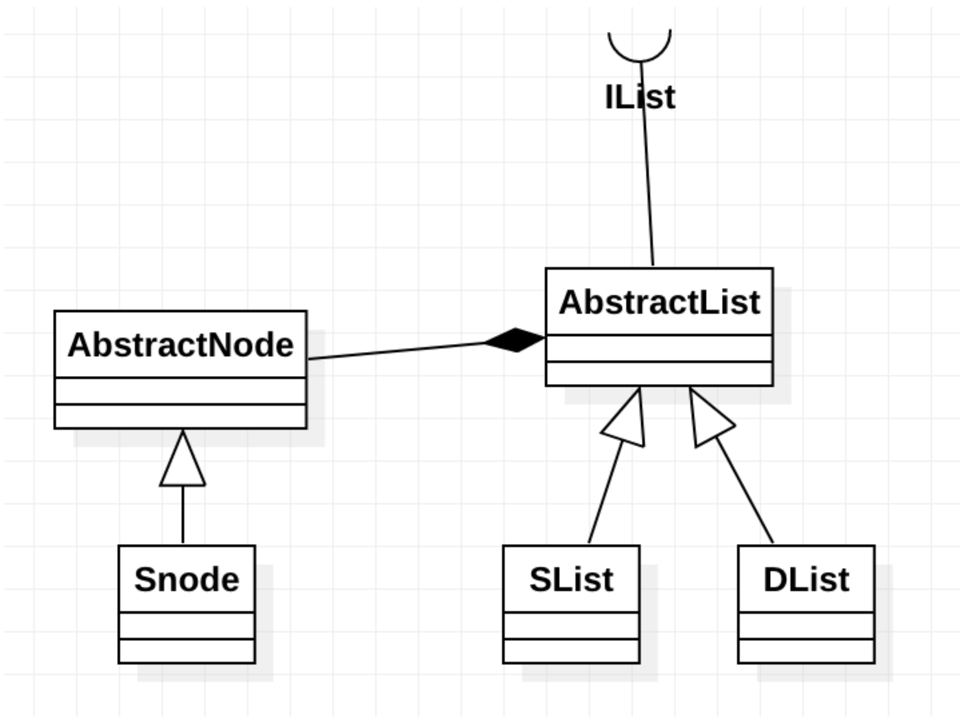

Figure 5.1: UML diagram for the data structures problem

5 To do

Implement other structures like Stacks, Queues, Trees, Hash and Sparse Matrix, additionally considering a AVL, RB-Tree or designing a new hierarchy based on arrays structures is a plus. Make a UML design before do any programming as shown in Figure 5.1, include interfaces, classes, abstract classes, attributes and methods, is necessary to focus in these details through all the design. Then make the implementation in C++ and try to ask these questions:

- Where is the code that shows up low coupling?
- Where is the code that shows up high cohesion?
- What improvements we can do to the code?
- What would it happen if two classes have zero coupling?
- What would happen if two classes have maximum cohesion?

6 Deep inside

This section is just for anyone who wants to make a deep inside into the theory and like challenges[1], you will have to prepare a presentation of just 5 minutes to explain and run the code, then you will have to defend yourself another five minutes of questions. I want that all students benefit from your presentation, remember that we are here to learn. If you want to do this please email to cportugalz@unsa.edu.pe

7 Deadline

This report will be qualified in the next class, consider the presentation of the UML design based on the 15 guidelines described before and a piece of code could be required too. Remember that plagiarism will be punished. All question and doubts must be done to the email.

[1]This must not be included into the report