B01 SE

ISE 1 Component II

Page No.
Date 08 04 23

Q1 What are different types of coupling and cohesion?

→

- Coupling is an inter module concept which captures the strength of interconnection between modules. The more tightly coupled the modules are, the more dependent they are on each other, and more difficult it is to modify them. Hence, low coupling is desirable for making the system for understandable and modifiable.
- In OO systems, three different types of coupling exists between modules, those are as follows:

1. Interaction coupling:
It occurs due to methods of a class invoking methods of other classes. In many ways, this situation is similar to a function calling another function and hence it is similar to coupling between functional modules. Like with functions, the worst ~~case~~ form of coupling here is if methods directly access internal parts of other methods.

2. Component coupling:
It refers to the interaction between two classes where a class has variables of other class. Three clear situations exist when this can happen. A class c can be component coupled with class c' if c has an instance variable of type c' or c has a method whose parameter is of type c' or if c has a method which has a local variable of type c'. Whenever there is component coupling, there is likely to be interaction coupling.

3. Inheritance coupling:
It is due to inheritance relationship between classes. Two classes are considered inheritance coupled if one class is a direct / indirect subclass of other. Within inheritance coupling there are some situations that are worse than others. The worst form is when a subclass modifies a signature of a method or deletes a method. Coupling is bad even when the signature is preserved but the implementation of a method is changed. The least coupling scenerio is when a subclass only adds instance variables and methods but does not modify any inherited ones.

- cohesion is an intra-module concept. It focuses on why elements of a module are together in the same module. In OO system, different types of cohesion are as follows:

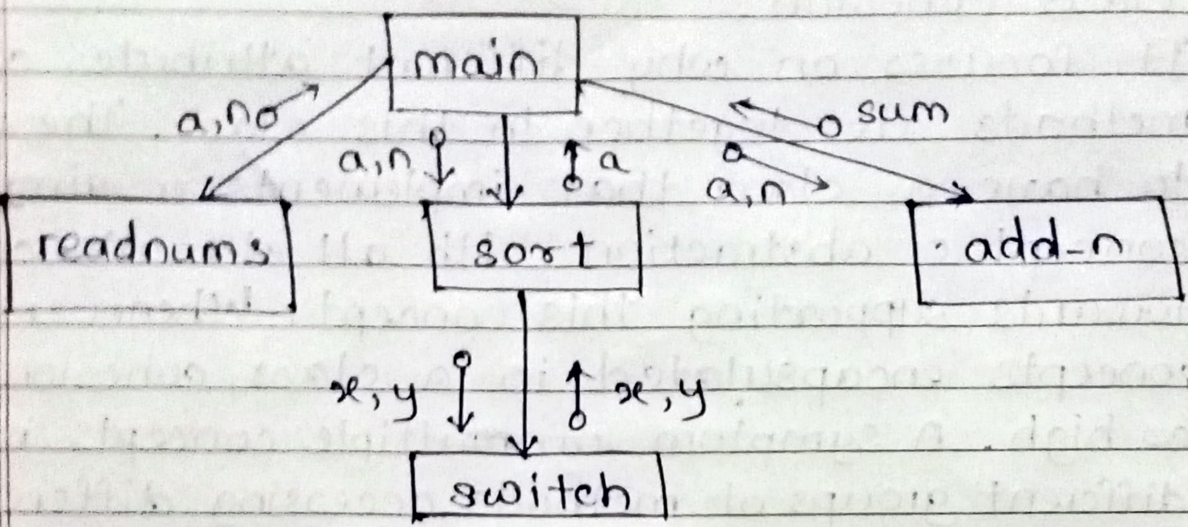
1. Method cohesion:
It focuses on why the different code elements of a method are together within the method. The highest form of cohesion is if each method implements a clearly defined function, and all statements in the method contribute to implementing this function. In general, with functionality cohesive methods, what the method does can be stated easily with a simple statement.

Page No.
Date

2. Class cohesion:
It focuses on why different attributes and methods are together in this class. The goal is to have a class that implements a simple concept or abstraction with all elements contributing towards supporting this concept: Whenever multiple concepts encapsulated in a class, cohesion is not as high. A symptom of multiple concepts is that different groups of methods accessing different subset of attributes.

3. Inheritance cohesion:
It focuses on why classes are together in an hierarchy. The two main reasons for inheritance are to model generalization - specialization relationship and for code reuse. Cohesion is considered higher if the hierarchy is for providing generalization - specialization.

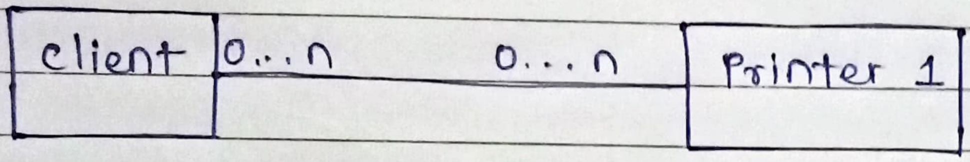
Q.2 Draw structure chart for sort program and state Open closed Principale with the example.
→ For a function oriented design, the design can be represented graphically by structure charts. Every program has a structure, and if it is given, it's structure can be determined.
The structure chart of the sort program:



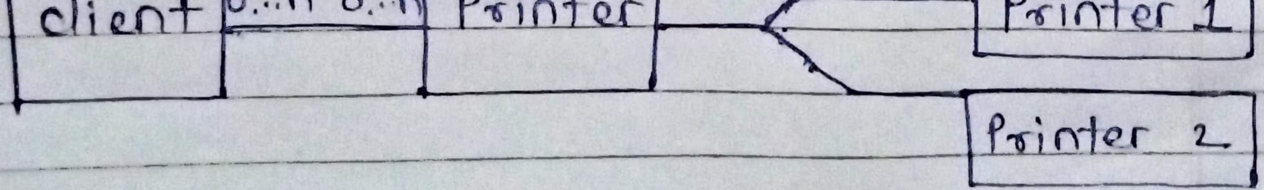
The structure chart of the sort program

- Besides cohesion and coupling open closed principle also helps in achieving modularity. The basic principle states that "software entities should be open for extension, but closed for modification".
- A module being "open for extension" means that its behaviour can be extended to accomodate new demands placed on this module due to changes in requirements and system functionality.
- The module being "closed for modification" means that the existing source code of the module is not changed when making enhancements.
- This principle restricts the changes to modules to extension only i.e. it allows addition of code, but disallows changing of existing code.
- It minimizes the risk of having existing ~~code~~ functionality stop working due to changes, which is a very important consideration while changing code.
- It is good for programmers, as they prefer writing new code rather than modifying old one

- In OO this principle is satisfied by using inheritance and polymorphism. Inheritance allows creating a new class to extend behaviour without changing the original class. This can be used to support the open-closed principle.
- Consider example of a client object which interacts with a printer object for printing, and calls the necessary methods for completing its printing needs. The class diagram for this will be:



- In this design, the client directly calls the method on the printer object1 ~~for~~. If another method of printer 2 is to be allowed, a new class printer 2 should be created, but the client will have to be changed if it ~~is~~ wants to use printer 2.
- The design of this system can be done in an alternative way using the open- closed Principle. The Printer 1 can be created as a subclass of a general printer class and for modification, add another subclass printer 2. The client does not need to be aware of this subtype as it interacts with objects of type printer.


client
0...n
0...n
Printer
Printer 1
Printer 2