LECTURE-10

Module Level Concepts

- Basic modules are classes
- During design key activity is to specify the classes in the system being built
- Correctness of design is fundamental
- But design should also be “good” – efficient, modifiable, stable, …
- Can evaluate a design using coupling, cohesion, and open-closed principle

Coupling

- Coupling is an inter-module concept, captures the strength of interconnection between modules
- More tightly coupled the modules, the more they depend on each other, more difficult to modify one
- Low coupling is desirable for making systems understandable and modifiable
- In OO, three types of coupling exists – interaction, component, and inheritance

Coupling…

- Interaction coupling occurs due to methods of a class invoking methods of other classes
  - Like calling of functions
  - Worst form if methods directly access internal parts of other methods
  - Still bad if methods directly manipulate variables of other classes
  - Passing information through temporary variables is also bad

Coupling…

- Least interaction coupling if methods communicate directly with parameters
  - With least number of parameters
  - With least amount of information being passed
  - With only data being passed
- I.e. methods should pass the least amount of data, with least number of parameters

Coupling…

- Component coupling – when a class A has variables of another class C
  - A has instance variables of C
  - A has some parameters of type C
  - A has a method with a local variable of type C
- When A is coupled with C, it is coupled with all subclasses of C as well
- Component coupling will generally imply the presence of interaction coupling also

Coupling…

- Inheritance coupling – two classes are coupled if one is a subclass of other
- Worst form – when subclass modifies a signature of a method or deletes a method
- Coupling is bad even when same signature but a changed implementation
- Least, when subclass only adds instance variables and methods but does not modify any

Cohesion

- Cohesion is an intra-module concept
- Focuses on why elements are together
  - Only elements tightly related should exist together in a module
  - This gives a module clear abstraction and makes it easier to understand
- Higher cohesion leads to lower coupling – many interacting elements are in the module
- Goal is to have higher cohesion in modules
- Three types of cohesion in OO – method, class, and inheritance

Cohesion…

- Method cohesion – why different code elements are together in a method (like cohesion in functional modules)
  - Highest form is if each method implements a clearly defined function with all elements contributing to implementing this function
  - Should be able to state what the module does by a simple statement

Cohesion…

- Class cohesion – why different attributes and methods are together in a class
  - A class should represent a single concept with all elements contributing towards it
  - Whenever multiple concepts encapsulated, cohesion is not as high
  - A symptom of multiple concepts – different groups of methods accessing different subsets of attributes

Cohesion…

- Inheritance cohesion – focuses on why classes are together in a hierarchy
  - Two reasons for subclassing
    - generalization-specialization
    - reuse
  - Cohesion is higher if the hierarchy is for providing generalization-specialization

Friday

Open-closed Principle

- Principle: Classes should be open for extension but closed for modification
  - Behavior can be extended to accommodate new requirements, but existing code is not modified
  - I.e. allows addition of code, but not modification of existing code
  - Minimizes risk of having existing functionality stop working due to changes – a very important consideration while changing code
  - Good for programmers as they like writing new code

Open-closed Principle…

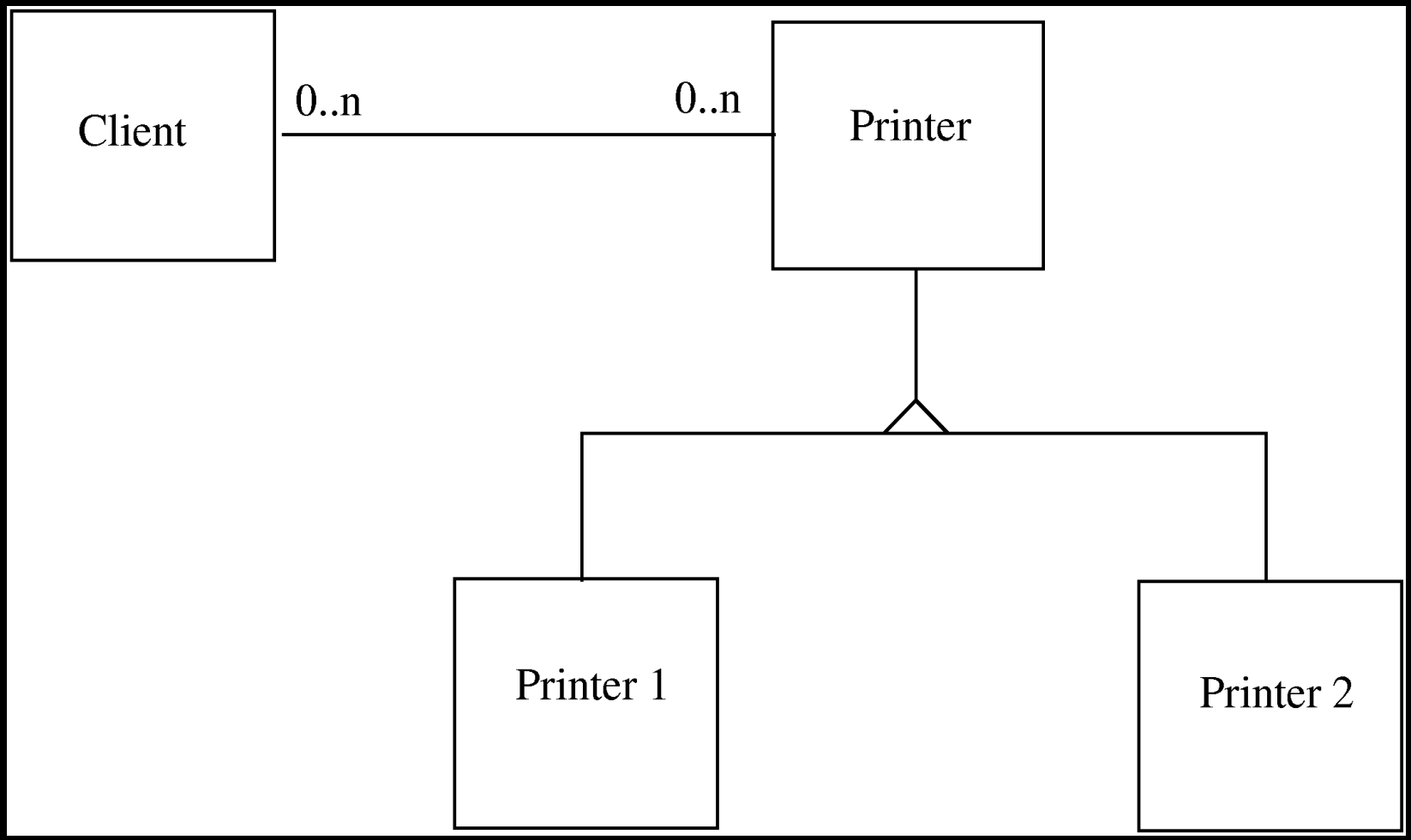
- In OO this principle is satisfied by using inheritance and polymorphism
- Inheritance allows creating a new class to extend behavior without changing the original class
- This can be used to support the open-closed principle
- Consider example of a client object which interacts with a printer object for printing

Example

Example..

- Client directly calls methods on Printer1
- If another printer is to be allowed
  - A new class Printer2 will be created
  - But the client will have to be changed if it wants to use Printer 2
- Alternative approach
  - Have Printer1 a subclass of a general Printer
  - For modification, add another subclass Printer 2
  - Client does not need to be changed

Example…



Liskov's Substitution Principle

- Principle: Program using object O1 of base class C should remain unchanged if O1 is replaced by an object of a subclass of C
- If hierarchies follow this principle, the open-closed principle gets supported