Design


2008
भारतीय प्रौद्योगिकी संस्थान भुवनेश्वर
INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR

Software Design

- Design activity begins with a set of requirements
- Design done before the system is implemented
- Design focuses on module view – i.e. what modules should be in the system
- Design of a system is a **blue print** for implementation
- Often has two levels – **high level** (modules are defined), and **detailed design** (logic specified)

Design…

- Design is a creative activity
- **Goal**: to create a plan to satisfy requirements
- Critical activity during system development
- Design determines the major characteristics of a system
- Has great impact on testing and maintenance
- Design document forms reference for later phases

- **Design methodology** – systematic approach for creating a design

Software Design Phase

- Different modules required to implement the design solution.
- Control **relationship** among modules (call/invocation relationship).
- Interface among different modules.

- Data structures of individual modules.
- Algorithms required to implement each individual module.

Characteristics: Good Software Design

- **Correctness**: Realize all the functionalities in the SRS document correctly.
- **Understandability**: Should be easily understandable.
- **Efficiency**: Should be efficient.
- **Maintainability**: Easily amenable to change.

Design Concepts

- Design is ***correct***, if it will satisfy all the requirements
- Of the correct designs, we want ***best design***
- We focus on **modularity** as the main criteria (besides correctness)

Modularity

- Modular system – in which modules can be built separately and changes in one have **minimum impact** on others
  - Supports independence of models
  - Enhances design clarity, eases implementation
  - Reduces cost of testing, debugging and maintenance
- Cannot simply chop a program into modules to get modularly
- Need some **criteria for decomposition** – coupling and cohesion are such criteria

Coupling

- Independent modules: if one can function completely without the presence of other
- Independence between modules is desirable
  - Modules can be modified separately
  - Can be implemented and tested separately
  - Programming cost decreases
- In a system all modules cannot be independent
- Modules must **cooperate** with each other
- **More connections** between modules
  - **More dependent** they are
  - More knowledge about one module is required to understand the other module.
- Coupling captures the notion of dependence

Coupling…

- **Coupling** between modules is the **strength of interconnections** between modules

- In general, the more we must know about module A in order to understand module B the more closely connected is A to B

- "Highly coupled" modules are joined by strong interconnection
- "Loosely coupled" modules have weak interconnections

Coupling…

- **Goal**: modules as loosely coupled as possible

- Where possible, have independent modules
- **Coupling** is decided during ***high level design***
- Cannot be reduced during implementation

- Major factors influencing coupling
  - Type of connection between modules
  - Complexity of the interface
  - Type of information flow between modules

Coupling – Type of connection

- Complexity of interfaces increase coupling
- **Minimize the number of interfaces** per module
- **Minimize the complexity of each interface**

- Coupling is minimized if
  - Only defined entry of a module is used by others
  - Information is passed exclusively through parameters

- Coupling increases if
  - Indirect and obscure interface are used
  - Internals of a module are directly used
  - Shared variables employed for communication

Coupling – interface complexity

- Coupling increases with complexity of interfaces eg., number and complexity of parms

- Interfaces are needed to support required communication

- Often more than needed is used eg. passing entire record when only a field is needed

- Keep the interface of a module as simple as possible

Cohesion

- Coupling characterized the ***inter-module*** bond
- Reduced by minimizing relationship between elts of different modules

- Another method of achieving this is by **maximizing relationship between elements of same module**
- **Cohesion** considers this relationship
- Interested in determining how closely the elements of a module are related to each other
- In practice both are used

Cohesion…

- Cohesion of a module represents how tightly bound are the elements of the module
- Gives a handle about whether the different elements of a module belong together
- High cohesion is the goal
- ***Cohesion*** and ***coupling*** are interrelated
- Greater cohesion of modules, lower coupling between module

Levels of Cohesion

- There are many levels of cohesion.
  - **Functional** (strongest form, focused on performing a single task)
  - **Sequential** (operations in a module occur in a specified order)
  - **Communicational** (not-related, but work on the same data)
  - **Temporal** (to happen around the same time)
  - **Logical** (similar kind of operations, but- not related)
  - **Coincidental** (weakest form of cohesion)

Open-closed Principle

- Besides *cohesion* and *coupling*, open closed principle also helps in achieving modularity

- **Principle**: A module 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

Open-closed Principle…

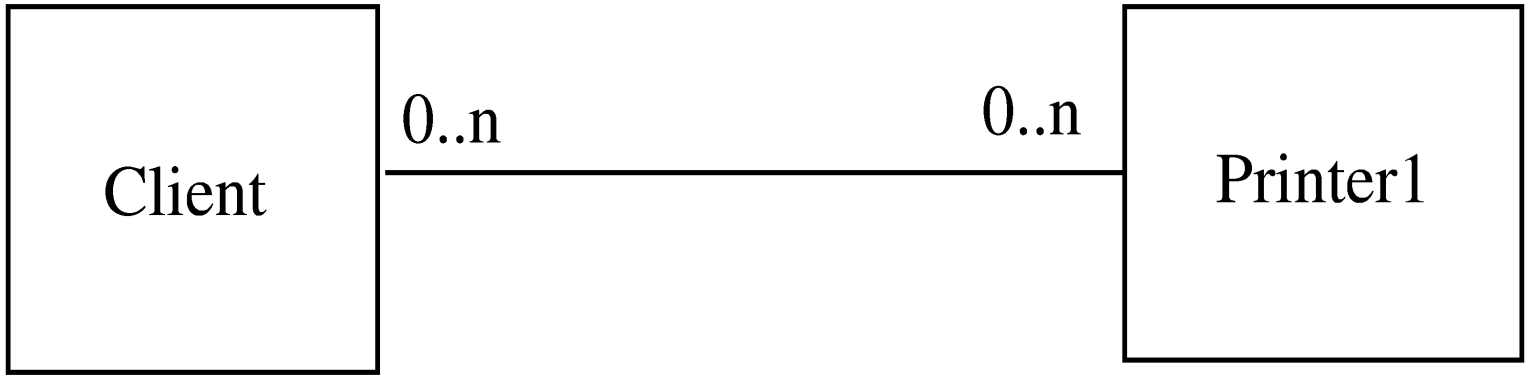
- In OO this principle is satisfied by using **inheritance** and **polymorphism**

- **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

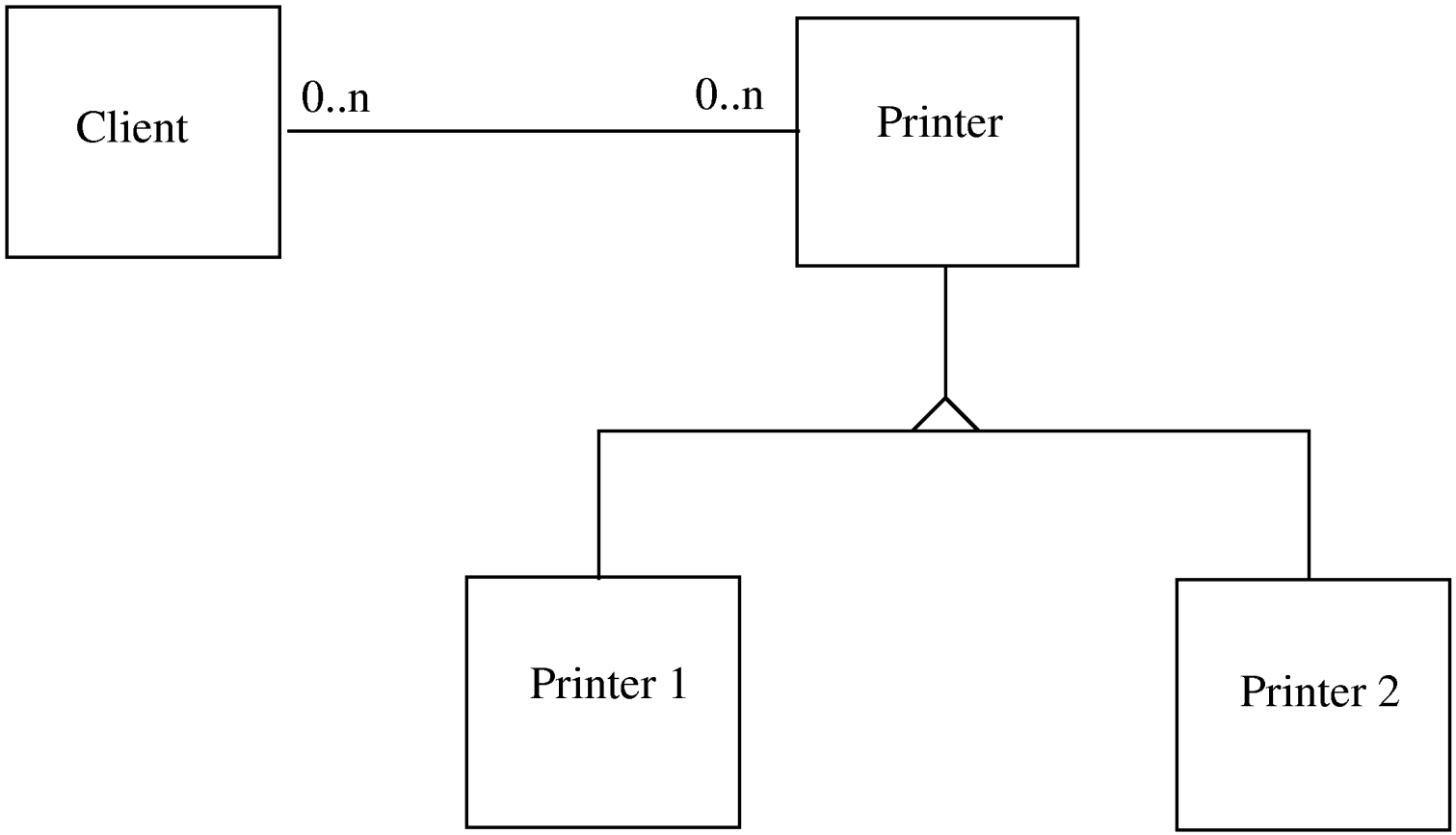
Example



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…



Summary

- Goal of designing is to find the best possible ***correct*** design
- **Modularity** is the criteria for deciding quality of the design
- Modularity enhanced by **low coupling**, **high cohesion**, and following **open-closed principle**