OOAD

System Design
(Part I)

Nguyễn Anh Hào
0913609730 – nahao@ptithcm.edu.vn

System Design

- System Design is the process of defining the architecture, components, interfaces and data of the system, to represent a ready-implement solution.
  - Design: the idea of the solution.
- Object-oriented design is to build the system using Object Oriented concepts
  - Real-world problems all have real-world solutions
  - Simulating reality is a way to design for real-world problems.

2015_Object-Oriented Analysis, Design and Implementation,2nd edition, Brahma Dathan&Sarnath Ramnath,Springer 2015 (chp 7, Page 159)

Component, SubSystem, Package

- Component
  - Each component of the system, in the sense of design, is the smallest unit (object class) of the system.
  - Each component only solves basic, common, and realistic problems.
- SubSystem
  - Is a set ot collaborating components
  - SubSystem solves a usecase's problem.
- Package (pre-written code library)
  - Is a container of implemented components (a file/folder).
  - Package solves common problems, and facilitate code-reuse of designed and implemented components.

Design objectives

A good design is the one that minimize the total cost of creating, using and upgrading the system over its lifetime.

1 Each component is easy to understand, no needs to refer to complementary documents.
  - Balancing between understandability and efficiency

2 Decompose the system to be built easily into components (or subsystems)

3 Necessary correction will be carried out in a small scope
  - Minimize the spread of damage from faulty components to other components.

4 Object-components after being made can be reused.

Coupling & Cohession

- **Coupling**: the degree of interdependence between different components of a system.
  - The higher the interdependency, the more likely changes in part of a design can cause changes to be required in other parts of the design
- **Cohesion**: the degree to which elements (within a component) are related and function together as a unified whole.
  - The components should be tightly integrated

Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Design criteria (p.286)

Coupling: Encapsulation

1 If a method refers to an attribute of its class, it is tied to the name of the attribute.

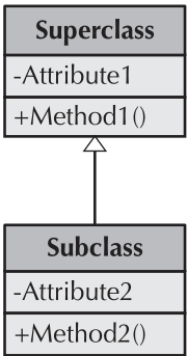
2 If a class has an attribute of type A, it is tied to the type of the attribute.

3 A class has an attribute in which a range of values has a semantic meaning: If the range would change, then every method that used the attribute would have to be modified.

- Example: sexuality={men,women} → {men, women, bisexual }, Room sevices should change.

Coupling: Inheritance

- When Subclass B inherits from Supperclass A: If B uses the inherited content from A, then B depends on this inherited content.
- Ensure inheritance is used only to support generalization/specialization semantics.
  - *Should a method defined in a subclass be able to call a method of the superclass ?*
  - *Should a method defined in a subclass refer to an attribute of the superclass ?*



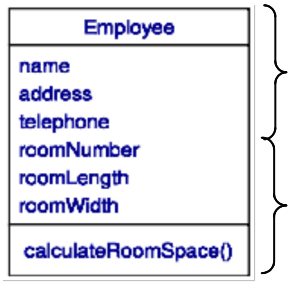
Coupling: Interaction

Interaction coupling deals with the coupling among methods and objects through message passing

1 Itself (an object sends message to itself): Calling method depends on called methods of its class
2 An object that is contained in an attribute of a class: Using this attribute depends on this object
3 An object that is passed as a parameter to the method: Using this parameter depends on this object
4 An object that is created by the method: Calling method depends on the object being returned by the called method.

Cohesion: Class

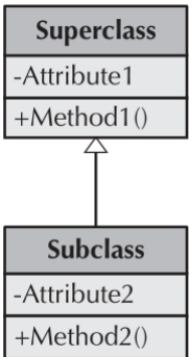
1 A class should represent one thing (e.g. person, car, department)
2 All attributes and methods of a class should be required for the class to represent the class
3 No redundant attributes or methods should exist
4 The cohesion of a class is the degree of cohesion between the attributes and the methods of a class



Low class cohesion

Cohesion: Inheritance

How are the classes in the inheritance hierarchy related ?
Are they related through a generalization /specialization semantics ?
To what degree a subclass actually needs the features it inherits ?

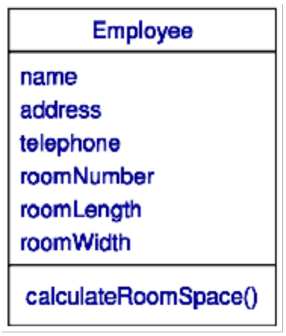


- **Liskov Substitution**: In a class hierarchy it should be possible to treat a specialized object as if it were a base object.

- All derived classes are necessary to satisfy the role /responsibility of the base class.

Cohesion: Method

1 A method should solve a single task.
  – A method performing multiple functions is more difficult to understand and reuse.
2 All elements of a method (data, code) are necessary for that method.



⟹ High method cohesion

The first 5 Principles of OO design

1. **S**ingle Responsibility: A class should have one and only one reason to change, meaning that a class should have only one job.
2. **O**pen/Closed: Classes should be open for extension, but closed for modification.
3. **L**iskov Substitution: every subclass or derived class should be substitutable for their base / parent class.
4. **I**nterface Segregation: A client shouldn't be forced to be dependent on data/functions that they do not use (avoid **fat interface**).
5. **D**ependency Inversion: The high-level module must not depend on the low-level module; they should depend on abstractions.

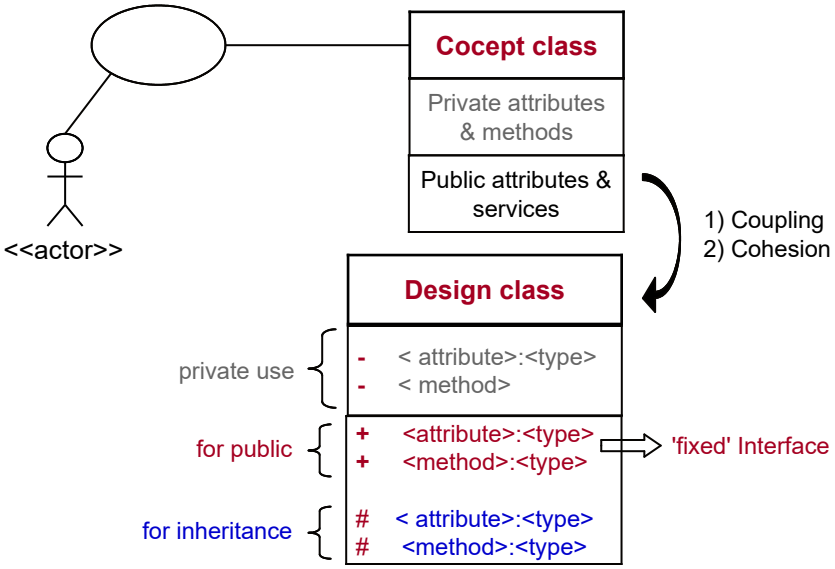
Dependency Inversion: Bad example


```
class Manager {
        Worker worker;
        public void setWorker( Worker w) { //or SuperWorker. How?
                worker = w;
        }
        public void manage() {   worker.work();
        }
  }
class Worker {
        public void work() {   // ....working
        }
  }
class SuperWorker {       // ADDED.
        public void work() {  //.... working much more
        }
  }
```

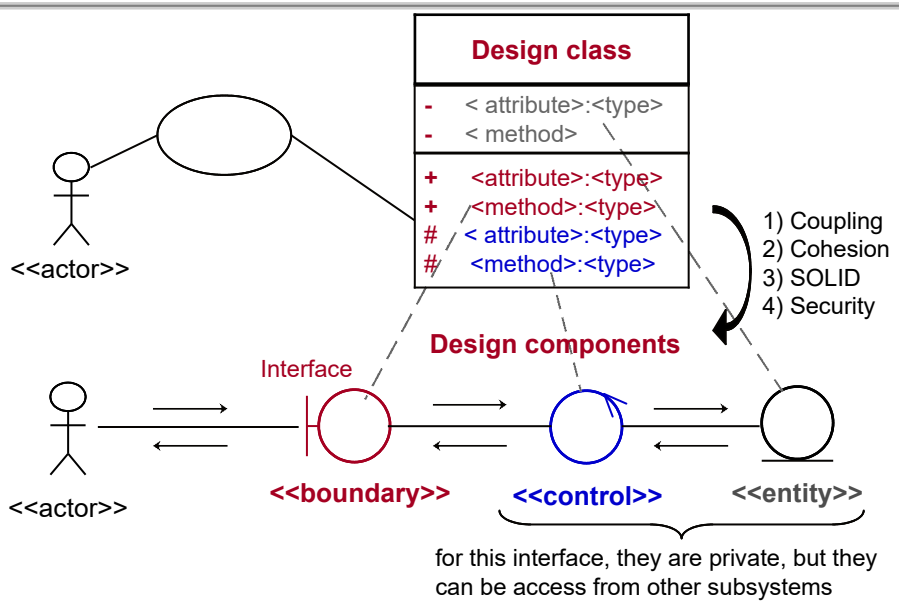
Dependency Inversion: Good example


```
class Manager {
        IWorker worker;
        public void setWorker( IWorker w) {     worker = w;   }
        public void manage() {    worker.work();   }
}
interface  IWorker {                    // Abstract interface
        public void work();
}

class Worker implements IWorker{
        public void work() { … }      // Concrete class 1
}

class SuperWorker implements IWorker{
        public void work() { … }      // Concrete class 2
}
```

From analysis to design: a Concept class

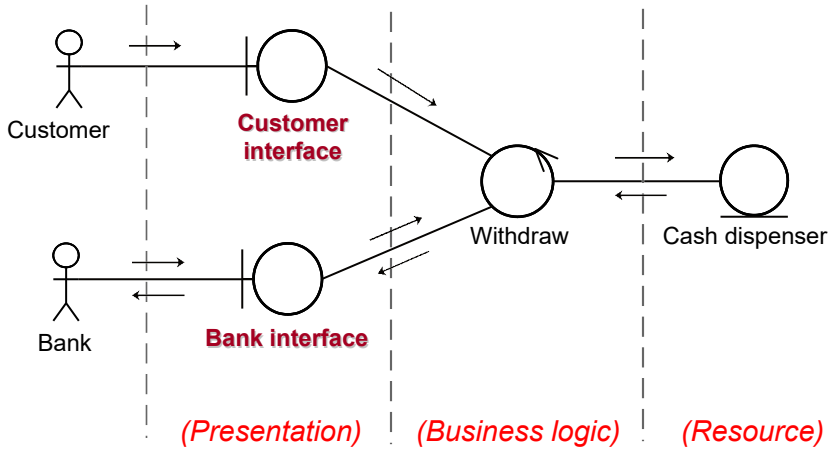


From analysis to design: a Design class



Design component example

ATM: withdraw money usecase



1.FACTORING

- Factoring is the process of separating out a module into standalone module(s) in and of itself, to simplify the overall design.
  - We may realize that some classes of the design share a similar definition (common attributes & methods)
  - In this case, it may make sense to factor out the similarities into a separate class.
  - Depending on whether the new class should be in a superclass to the existing classes or not, the new class can be related to the existing classes through generalization (A-Kind-Of) or aggregation (Has-Parts) relationship.
- *Abstraction* and *refinement* are two closely related processes to factoring.

Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Factoring (p.257)

Factoring: Abstraction & Refinement

- ***Abstraction*** deals with the creation of a "higher" level idea from a set of ideas.
  - Identifying the Employee class is an example of abstracting from a set of lower classes :Nurse, Administrative Staff, Doctor,..
  - In some cases, the abstraction process will identify abstract classes. Otherwise, it will identify additional concrete classes
- ***Refinement*** is the opposite of abstraction process.
  - It is possible to identify additional subclasses of the Administrative Staff class, such as Receptionist, Secretary,...
  - We would only add the new subclasses if there were sufficient differences between them.

2.PARTITIONING

- A partition is equivalent to a sub-system (minimize inter-relationships between subsystems).
- Potential partitioning is based on the collaborations that modeled in UML's communication diagrams for usecases
- The class diagram should be reviewed to see how different classes are related in their collaborations.
  - Combine the class diagram with the communication diagrams can be very useful to show what degree the classes are coupled.
  - The greater the coupling between classes, the more likely the classes should be grouped together into a partition.

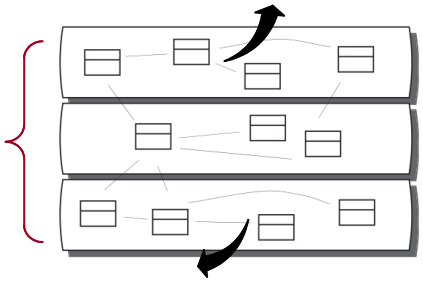
Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Partitions and Collaborations (p.258)

3.LAYERING

- Each layer is a cluster of collaborating objects, dependent on the facilities offered by the lower layers.
  - Help to reduce complexity
  - Increase the chance of reuse

Top layer represents the user interface (menus, dialogs, usability, intuitiveness,..)

Different layers have different focuses.

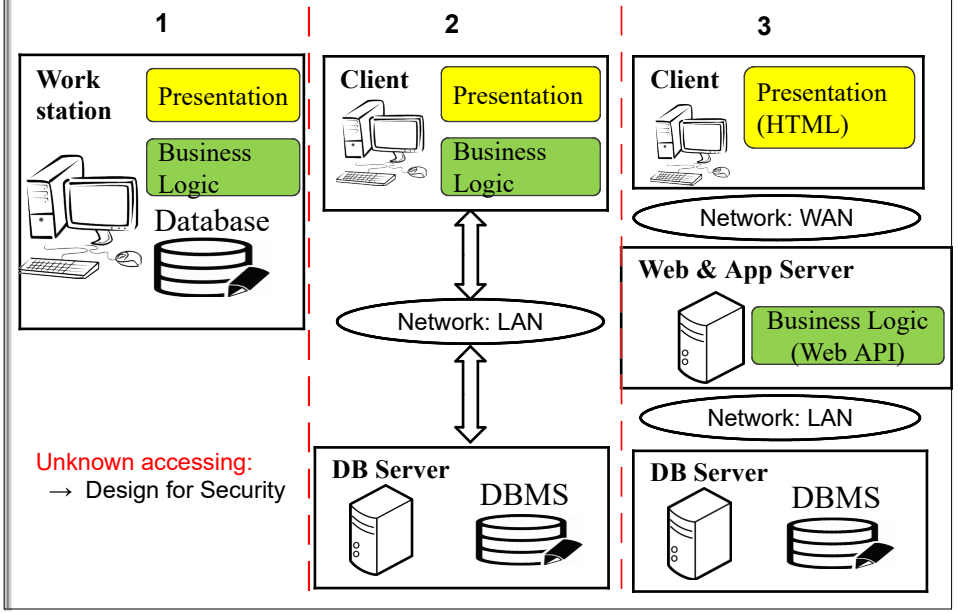


Middle layer: abstraction, attributes, data base, operations, polymorphism, reuse...

Bottom layer represents the operating system, or a network connection (protocols, band width and different types of hosts...)

Object Oriented Analysis And Design, Mike O'Docherty, Wiley 2005: Layer (P248)

Example: *"Thinner client, thicker server"*



4.SECURITY

1 **Privacy**: Information hiding, making it available only to those who are authorized to read or change it.

2 **Authentication**: Need to know where each piece of information came from, so that decide whether or not to trust it.

3 **Irrefutability**: This is the flip-side of authentication, ensuring that the originator of information can't deny that they're the source; this is helpful to us if anything goes wrong.

4 **Integrity**: Ensure that information hasn't been damaged, accidentally or maliciously, on its way from the source to the destination.

5 **Safety**: Control access to resources (such as machines, processes,databases and files). Safety is also known as authorization.

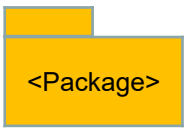
Object Oriented Analysis And Design,...:8.6 DESIGNING FOR SECURITY (P222)

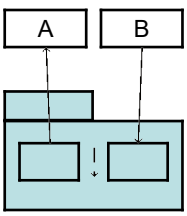
DESIGN PACKAGE

1 Is a mechanism for grouping basic design elements (classes, interfaces) into a physical structure ("module").

2 The contents of the Package can be:
- A library of reusable components;
- Classes that need to be installed together (they depend on each other)

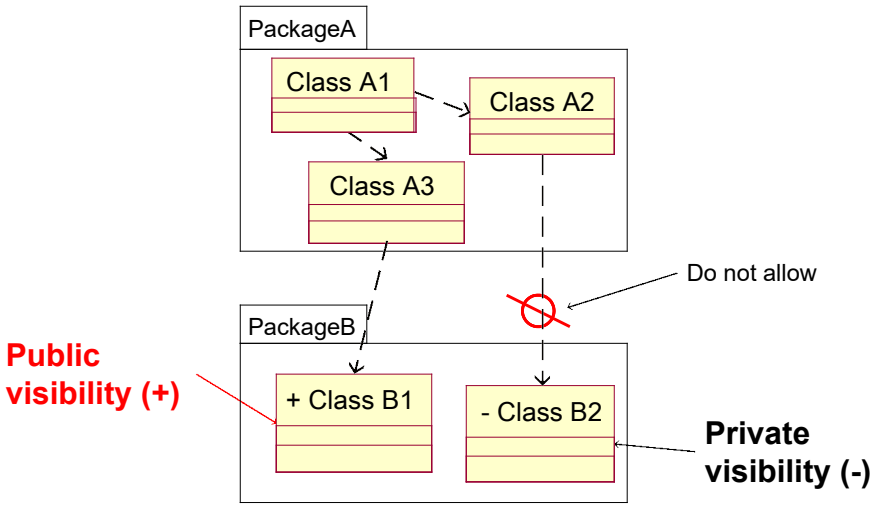
3 Package design: organize classes (code, data) into packages for easy development, reuse and maintenance.



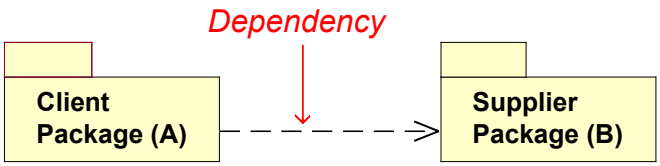


Systems Analysis and Design An Object-Oriented Approach with UML,5th edition, Alan Dennis,Wiley 2015: Package and package diagram (p.262)

Package interface: Visibility & Access



Package dependency



1 Package A depends on package B if there is a class in A that depends on a class in B.

2 The inevitable consequences are:
- Changes to the Supplier Package (B) may affect the Client Package (A)
- Client Package cannot be used separately, because it depends on Supplier Package

Guidelines for Creating Package Diagrams

1. Set the context for the package diagram as the problem domain layer (associate use case).
2. Group classes together when there is an inheritance, aggregation, or composition relationship between them or when the classes form a collaboration
   - The more the classes depend on each other, the more likely they belong together in a package
   - The direction of the dependency is typically from the subclass to the superclass, from the whole to the part, from the client to the server.
   - Each group is a package (partition)
3. Identify the dependency relationships among the packages
   - Packages should not be cross-coupled