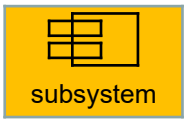
OOAD

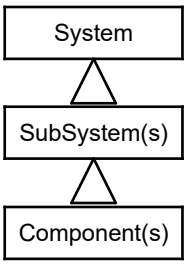
System Design
(Part II)

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

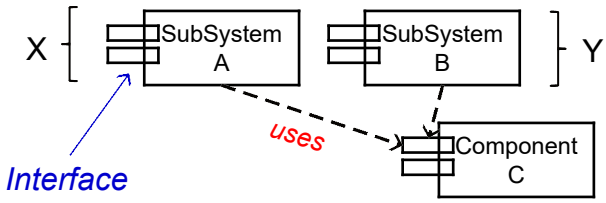
SUBSYSTEM

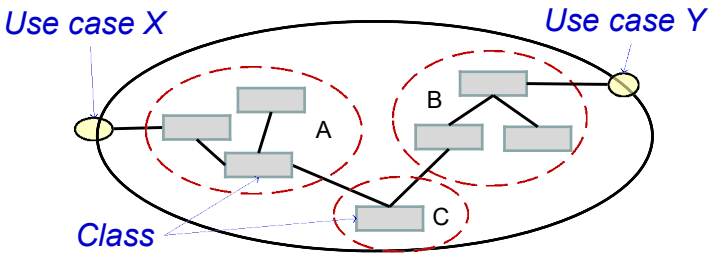
- Is a group of objects that cooperate to perform some system tasks
  - Participate in solving a use case
- SubSystem provides system services through its interfaces
- SubSystem Design: not intended for reuse (≠ Package)





SubSystem





1. Analysis model to Design model

1. Partitioning the system into SubSystems
   - Similar to an object class, each SubSystem is to handle a use case in the use case diagram.
   - Factoring: factor out the similarities into a superclass class, or the specific details into some subclasses
   - Partitioning: All the communication diagrams for the use case is used to indicate concept classes (mapped to design classes) of the SubSystem.

Analysis model to Design model

2. Interaction design for SubSystem
   - Only the boundary class is visible from the outside. All the internal classes of the SubSystem are implicitly hidden.
   - within the SubSystem, no internal classes depend on the boundary class.
   - SubSystem has at least 1 concrete class that implements the boundary class (=abstract class).
   - The data type of the message is also declared to the outside world.

3. Mapping classes to Implementation Languages
   - Methods & Attributes (contain persistent data)

INTERFACE: ROLE

1. Protect the system from unwanted impacts from the external environment
   - Security check
   - Detect errors
2. Filter out unnecessary input/output content
   - Interface segregation principle (4)
3. Encode and decode input/output content
   - Synthesize and convert data into necessary formats for the system or for other external objects.
4. Temporarily store input/output data using a buffer zone.

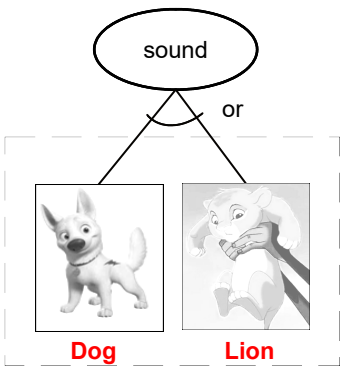
INTERFACE: DESIGN

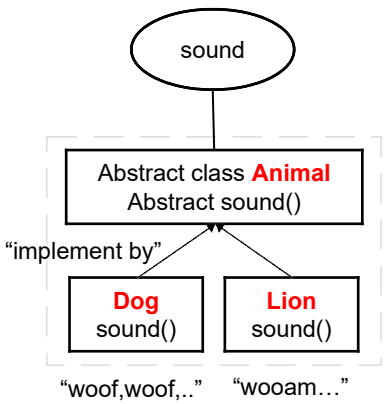
1. Only the interface (or boundary) of the system is visible from the external objects; all other internal classes are defined as invisible.
2. No inner classes depend on the boundary
3. Provides fixed or stable interfaces (prototypes).
4. The message data type is declared to be used for the external objects.
5. Complements the SOLID principle.

User Interface Modelling with UML.pdf

ABSTRACT INTERFACE

Abstract interface does not contain any concrete methods (methods with code), It only contains abstract methods: they do not have body; they just have name, parameters, and return type.





ABSTRACT INTERFACE benefits

1 Separate "what" from "how"
   – Abstract interfaces don't contain implementation logic. This allows multiple implementations to vary.
2 Enforce consistency
   – If a class implements an abstract interface, it must implement all of the required methods — the compiler enforces this.
3 Enable polymorphism
   – This allows different classes can agree to implement the same set of methods.
4 Support dependency inversion & testability
   – This depends on the interface rather than a concrete class, which makes swapping implementations easier.

ABSTRACT DATA TYPE

- The definition only mentions what operations are to be performed, not how these operations will be implemented.
  – It defines a common interface for its concrete classes.
  – It does not specify how data will be organized, and what algorithms will be used for implementing the operations.

- It gives an implementation-independent view, in order to
  – Enable code reuse
  – Create multiple versions of the component

MESSAGE

A message is a data content transferred between objects, so the content of the message needs to be understood between sender and receiver.

- The message content depends on its data structure (type, length,...).
- Sending & receiving messages are handled by many programmers, and they depends each other on this structure.
- In order to avoid dependents, message should be an object that provides processing method on its internal data structure, written by the sender.

Message example

An implementation of address in C++ is as follows:

```
public char Addr[40];
```

What difficulties will happen when developing software?

Programmers using this structure must be aware of:
1. Addr is a C zero string, ending with '\0', strlen ≤ 39 characters
2. Common commitment among programmers about the structure : “<num> <street> <ward> <district> <city>” …
3. How to change the structure of Addr without having to revise too much in many relate programs ?

Message Object


```
class Addr
{   private:            // information hidden
        string num, street, ward, district, city;
    public:             // accessing via public interface
        string GetAddr(); string Getnum(), string GetStreet();
…
        int SetNum(string sn); int SetStreet(string st);
        ….
};
```

So:

1. Accessing private data by using public methods only
2. Contents of the public methods and private properties are created and edited only by the sender

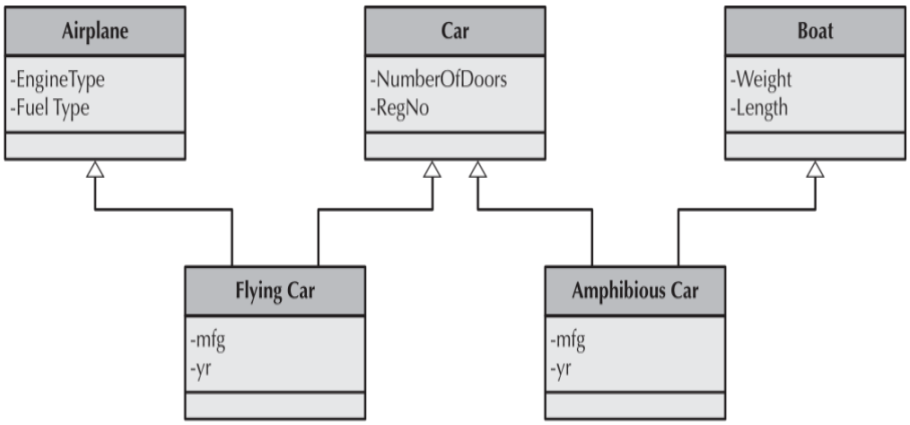
2. Persistent data into database

DATABASE TYPES

- Relational Database Management System (RDBMS)
  - Support referential integrity on simple data only
  - Cannot handle complex data, do not support for OO
- Object-Relational Database Management System (ORDBMS)
  - ORDBMSs are RDBMS with extensions to handle the storage of objects in the relational table structure.
  - ORDBMSs on the market still do not support all of the object-oriented features (e.g., **inheritance**)
- Object-Oriented Database Management System (OODBMS)
  - Able to handle complex data, direct support for OO
  - Technology is still maturing
- NoSQL data store
  - Able to handle complex data
  - Technology is still maturing

Example: Multi-Inheritance

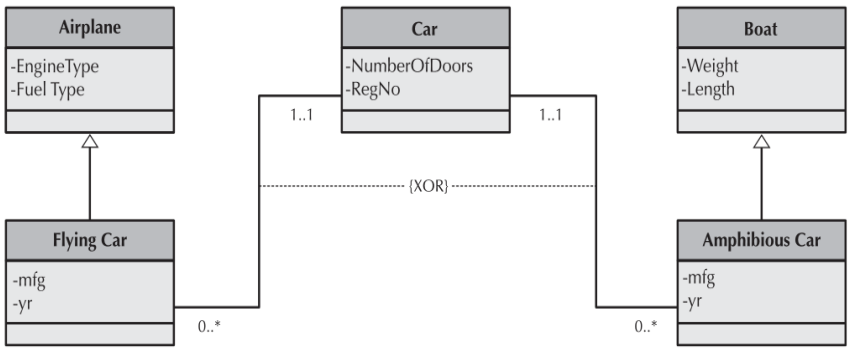
Simple example of multiple inheritance



(a)

Multi-Inheritance-ORDBMS: RULE 1a

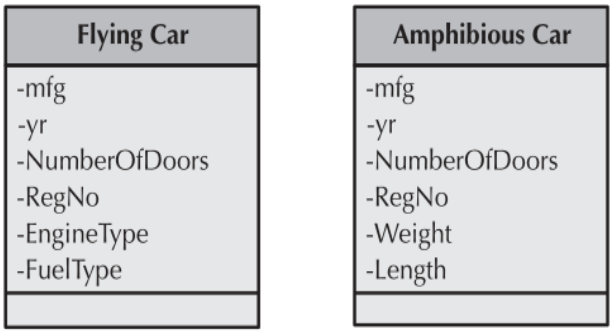
Convert the additional inheritance relationships to association relationships. The multiplicity of the new association from the subclass to the superclass should be 1..1, or 1..0 if superclasses are concrete.



(b)

Multi-Inheritance-RDBMS: RULE 1b

Flatten the inheritance hierarchy by copying the attributes and methods of the additional superclass(es) down to all of the subclasses and remove the additional superclass from the design



(c)

3.Domain Objects to RDBMS

1. All concrete classes must be mapped to tables in the RDBMS.
2. Single-valued attributes mapped to columns in RDBMS table.
3. Ensure that the primary key of the subclass instance is the same as the primary key of the superclass.
4. Methods should be mapped to either stored procedures or program modules
5. Single-valued (one-to-one) aggregation and association relationships are mapped to columns that can store the foreign keys of the related tables.
6. Multivalued attributes are mapped to new tables and create one-to-many relation
7. Multivalued aggregation and association relationships are mapped to new tables and create an associative table

Systems Analysis and Design An Object-Oriented Approach with UML (p.337)