Object Oriented Design

Design axioms - Designing Classes – Access Layer - Object Storage - Object Interoperability.

Design process and design axioms

- Analysis phase: Identify the classes, attributes and methods. We deal with how these real objects fix with real-world system. They are physical entities or business objects.
- Design Phase: Identified classes should be designed for implementation. We elevate the model into logical entities, which might relate more to the computer domain.

Activities of Design process

- Apply design axioms to design classes, their methods, attributes, associations, structures and protocols.
- Refine and complete the static UML class diagram.
  - Refine attributes
  - Design methods and protocols by utilizing UML activity diagram.
  - Refine associations between classes.
  - Refine class hierarchy and design with inheritance.
  - Iterate and refine again
- Design Access layer classes
  - Create mirror classes: For every business classes identified and created, create one access class.
  - Identify access layer class relationships.
  - Simplify classes and their r/ps-goal here is to eliminate
  - Redundant classes
  - Method classes – eliminate classes with only one or two methods
  - Iterate and refine again

Design process

- Design the view layer classes.
  - Design the macro level user interface, identifying view layer objects.
  - Design the micro level user interface.
    - Design the view layer objects
    - Build the prototype of the view layer interface.
  - Test the usability and user satisfaction
  - Refine and iterate
- Iterate and refine the whole process.

- Axioms: fundamental truth. It cannot be proven but can be invalidated by counter examples and exceptions.
- Theorem: is a proposition that may not be self-evident but can be proven from accepted axioms.
- Corollary: It is a proposition that follows from an axiom that has already proven.

Axiom1: deals with relationships between system components.

- The independence axiom
- Maintain the independence of components.
- Design of component must satisfy the requirements
- without affecting the other requirements

Axiom 2: deals with complexity of the design

- The information axiom
- Minimize the information content of the design.
- How to minimize complexity:
- Design should be simple
- reusability, use inheritance

Occam’s razor: “The best theory explains the known facts with a minimum amount of complexity and maximum simplicity and straight forwardness”.

Corollary 1: Uncoupled design with less information content. (High cohesive and less coupling)

(i) Coupling: Coupling is a measure of the strength of association between two objects.

- Interaction coupling: It involves the amount and complexity of messages between components.
  - content - Very high
  - Common - High
  - Control - medium
  - Stamp - low
  - Data - Very low
- Inheritance coupling: It is a form coupling between super and subclasses

(ii) cohesion: It deals with interaction between objects or software

- components.
- Method cohesion
- Class cohesion

Corollary 2. Single Purpose

- Every class should be clearly defined and necessary in the context of achieving the system’s goals.

Corollary 3 :Large Number of Simpler Classes

- More the classes are simple more it is reusable.
- The primary benefit of software reusablity is higher productivity.

Corollary 4 : Strong mapping

- There must be a strong association between the analysis's object and design's object
- OOA and OOD are based on the same model
- As the model progresses from analysis to implementation, more detailed is added.

Corollary 5 : Standardization

- Promote standardization by designing interchangeable components and reusing existing classes or components

Corollary 6 : Design with inheritance

- Common behavior (methods) must be moved to superclasses.
- The superclass-subclass structure must make logical sense
- Issues:
  - (i) Achieving Multiple inheritance in a single inheritance system.
  - It is done by making the object of another class as an attribute (aggregation)
  - (ii) Avoiding Inheriting Inappropriate Behaviors:
  - Design the subclass that inherit from appropriate superclass.

Designing Classes

- OCL is a specification language that uses simple logic for specifying the **properties of a system.**
- The expressions are meant to work on sets of values when applicable.

1.Item.selector—The selector is the name of an attribute in the item. The result is the value of the attribute.

– EX: john.age

– age is an attribute of the object john.

2.Item.selector[qualifier-value]—The selector indicates a qualified association that qualifies the item . The result is the related object selected by the qualifier.

– Ex: john.phone[3],assuming john has several phones.

- Set---select (boolean expression).
- The boolean expression is written in terms of objects within the set .

  ex: company.employee- >salary>50000.
- This represents employees with salaries over $50,000

- Apply design axioms to design classes , their attributes , methods , associations , structures , and protocols .
  - Refine and complete the static UML class diagram by adding details to that diagram.
    - Refine attributes
    - Design methods and the protocols by utilizing a UML activity diagram to represent the method’s algorithm.
    - Refine the associations between classes
    - Refine the class hierarchy and design with inheritance
  - Iterate and refine

Class visibility : Designing welldefined Public , Private and Protected protocols

- In designing methods or attributes for classes, we face 2 problems:
  - Protocol or interface to the class operations and its visibility, and
  - How it is implemented.

- In Private protocol (visibility) of theclass, includes messages that normally should not be sent from other objects; it is accessible only to operations of that class.
- In public protocol (visibility) ----it is accessible to all the class.
- In protected protocol (visibility)--- subclasses the can use the method in addition to the class itself.

Designing classes : refining attributes

- In analysis phase, the name of the attribute was sufficient.
- In design phase, detailed information must be added to the model (especially , that defining the class attributes and operations).

Attribute types

The three basic types of attributes are

- Single – value attributes---> It has only one value or state.
  - Ex : attributes such as name ,address , or salary are of the single-value type
- Multiplicity or multivalue attributes---->It can have a collection of many values at any point in time
  - Ex: if we want to keep track of the names of people who have called a customer support line for help , we must use the multivalues attributes .
- Reference to another object , or instance connection--->Instance connection attributes are required to provide the mapping needed by an object to fulfill its responsibilities.
  - Ex : an account can be assigned to one or more persons (ie.,joint account).

UML attribute presentation

The following is the attribute presentation suggested by UML :

visibility name : type – expression = initial – value

- Where visibility is one of the following :

- + public visibility (accessibility to all classes)
- # protected visibility (accessibility to subclasses and operations of the class)
- - Private visibility (accessibility only to operations of the class)
  - Example: +size:length = 100

Refining attributes for the bank objects

Refining attributes for the bankclient class

- During object-oriented analysis , we identified the following attributes
  - firstName :
  - lastName :
  - pinNumber :
  - cardNumber :
  - #firstName : String
  - #lastName : String
  - #pinNumber :String
  - #cardNumber : String
  - #account : Account (instance connection)

- Here is the refined list of attributes for the account class
  - #number : String
  - #balance : float
- The attributes for the transaction class are these
  - #transID : String
  - #transDate : Date
  - #transTime : Time
  - #transType : String
  - #amount : float
  - #postBalance : float

Access Layer: Object storage and object interoperability

- A Date Base Management System (DBMS) is a set of programs that enables the creation and maintenance (access, manipulate, protect and manage) of a collection of related data.
- The purpose of DBMS is to provide reliable, persistent data storage and mechanisms for efficient, convenient data access and retrieval.
- Persistence refers to the ability of some objects to outlive the programs that created them.
- Object lifetimes can be short for local objects (called transient objects) or long for objects stored indefinitely in a database (called persistent objects).

- Most object-oriented languages do not support serialization or object persistence, which is the process of writing or reading an object to and from a persistence storage medium, such as disk file.
- Unlike object oriented DBMS systems, the persistent object stores do not support query or interactive user interface facilities.
- Controlling concurrent access by users, providing ad-hoc query capability and allowing independent control over the physical location of data are not possible with persistent objects.

Object oriented databases versus Traditional Databases

- The responsibility of an OODBMS includes definition of the object structures, object manipulation and recovery, which is the ability to maintain data integrity regardless of system, network or media failure. The OODBMs like DBMSs must allow for sharing; secure, concurrent multiuser access; and efficient, reliable system performance.
- The objects are an “active” component in an object-oriented database, in contrast to conventional database systems, where records play a passive role. Another feature of object-oriented database is inheritance. Relational databases do not explicitly provide inheritance of attributes and methods.

Designing Access Layer Classes

- The main idea behind creating an access layer is to create a set of classes that know how to communicate with the place(s) where the data actually reside.
- Regardless of where the data reside, whether it be a file, relational database, mainframe, Internet, DCOM or via ORB, the access classes must be able to translate any data-related requests from the business layer into the appropriate protocol for data access.
- These classes also must be able to translate the data retrieved back into the appropriate business objects.
- The access layer’s main responsibility is to provide a link between business or view objects and data storage.
- Three-layer architecture is similar to 3-tier architecture. The view layer corresponds to the client tier, the business layer to the application server tier and the access layer performs two major tasks.