Module 43: Software Engineering

UML - Class Diagrams

Intructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

{*abir, sourangshu*}*@cse.iitkgp.ac.in*

Slides taken from NPTEL course on Object-Oriented Analysis & Design

by **Prof. Partha Pratim Das**

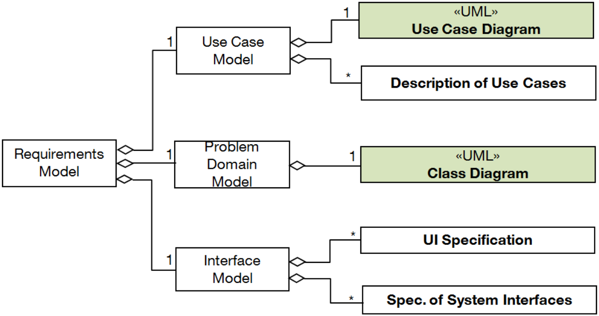
Module Objectives

- Understanding Class Diagrams

Module Outline

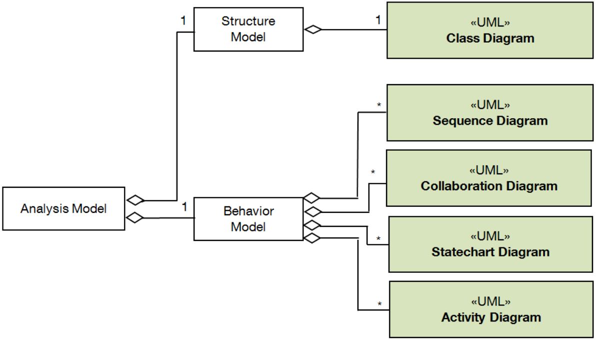
- What are Class Diagrams?
  - Class
  - Property (Attributes)
  - Operation (Methods)
  - Examples

Class Diagrams in SDLC phases: RECAP (Module 41)



- In the Requirements Phase, the class diagram is used to identify the major abstractions
- At this stage the attributes and operation of each abstraction may not be known
- Classes are identified as domain models

Class Diagrams in SDLC phases: RECAP (Module 41)



- After analysis of each abstraction, attributes and operation of each abstraction is known
- Hence the class diagram in the Analysis Phase is more detailed
- Classes are refined as domain models

Class Diagram

- Class diagram is UML structure diagram which shows structure of the designed system at the level of classes and interfaces, shows their features, constraints and relationships – associations, generalizations, dependencies, etc.
- Some common types of class diagrams are:
  - ○ Domain model diagram
  - ○ Diagram of implementation classes

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Features of a class

- **Non Static Features:** characterizes individual instances of class
- **Static Features:** represents some characteristic of the class itself
- **Structural Features (attributes):** is a typed feature of a class that specifies the structure of instances of the class
- **Behavioral Features (Methods):** is a feature of a class that specifies an aspect of the behavior of its instances

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Module 43

Objectives & Outline

Class Diagrams

Class

Property

Operations

Examples

Relationships

Association

Weak Aggregation

Strong Aggregation

Examples

Generalization

Constraints

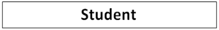
Examples

LMS Class Diagram

Summary

Notation for Class

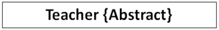
- Class name should be centered and in bold face inside a solid-outline rectangle, with the first letter of class name capitalized



Student

Class Student - details suppressed

- Abstract Classes (which cannot be instantiated) have the keyword abstract mentioned within { }



Teacher {Abstract}

Abstract Class Teacher - details suppressed

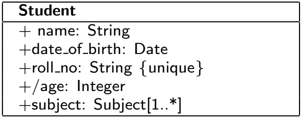
- A class has optional compartments separated by horizontal lines containing attributes and methods in order

Notation for Property (Attributes)

- **Property (Attributes) specification format:**

  Visibility PropertyName : Type [Multiplicity] = DefaultValue {Property string}

  - The visibility of the properties are denoted by +(public), #(protected) and –(private)
  - PropertyName is underlined if the Property is static
  - A property may be *Read Only*, *Static*, *Ordered*, *Unique* or *Optional* (to indicate allowable null value)
  - Property could have multiplicity. The multiplicity bounds constrain the size of the collection of property values. By default the maximum bound is 1
  - The default-value option is an expression for the default value or values of the property
  - A derived Property, designated by a preceding '/', is one that can be computed from other properties, but doesn't actually exist



Student

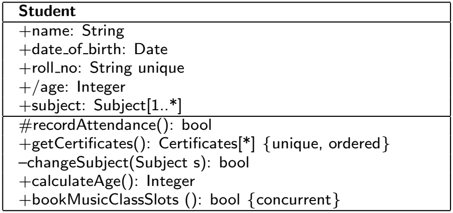
+ name: String
+date_of_birth: Date
+roll_no: String {unique}
+/age: Integer
+subject: Subject[1..*]

Notation for Operations (Methods)

- **Operation (Methods) specification format:**

  Visibility OperationName (ParameterName : Type) : ReturnType {*Property string*}

  - The visibility of the operations are denoted by +(public), #(protected) and –(private)
  - OperationName is underlined if it is Static, and is italic if it is Abstract
  - Return type is optional
  - An operation may be *Read Only*, *Static*, *Ordered*, *Unique*, *Abstract*, *Sequential*, *Guarded* or *Concurrent*



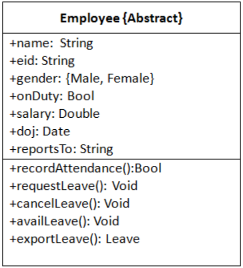
Student

+name: String
+date_of_birth: Date
+roll_no: String unique
+/age: Integer
+subject: Subject[1..*]

#recordAttendance(): bool
+getCertificates(): Certificates[*] {unique, ordered}
–changeSubject(Subject s): bool
+calculateAge(): Integer
+bookMusicClassSlots (): bool {concurrent}

Abstract Classes of LMS

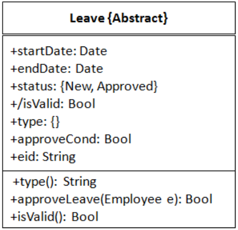
- We represent below the two abstract classes of LMS



Employee {Abstract}

+name: String
+eid: String
+gender: {Male, Female}
+onDuty: Bool
+salary: Double
+doj: Date
+reportsTo: String

+recordAttendance():Bool
+requestLeave(): Void
+cancelLeave(): Void
+availLeave(): Void
+exportLeave(): Leave

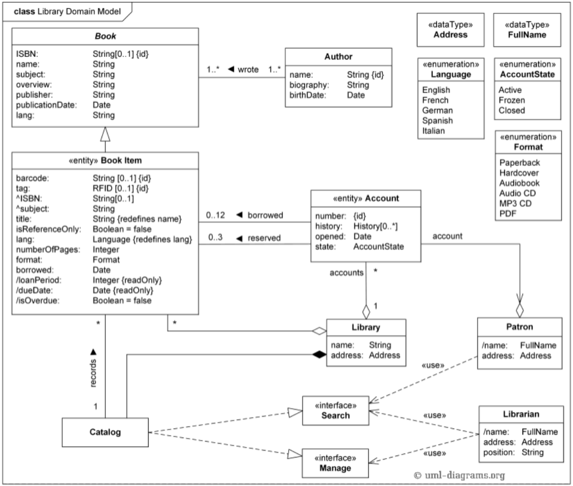


Leave {Abstract}

+startDate: Date
+endDate: Date
+status: {New, Approved}
+/isValid: Bool
+type: {}
+approveCond: Bool
+eid: String

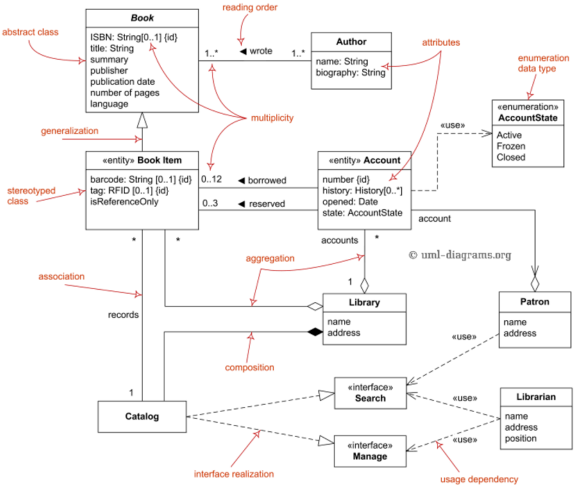
+type(): String
+approveLeave(Employee e): Bool
+isValid(): Bool

Library Domain Model



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Library Domain Model: Annotated



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Relationships of Classes: RECAP

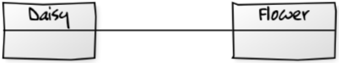
Class	**Relationship**
• A daisy is a kind of flower	Sharing connection – daisies and roses are both kinds of flowers – bright colored petals, fragrance, etc. Daisy IS_A Flower
• A rose is a (different) kind of flower	Sharing connection – daisies and roses are both kinds of flowers ... Rose IS_A Flower
• Red roses and yellow roses are both kinds of roses	Semantic connection – red roses and yellow roses are more alike than are daisies & roses Red Rose IS_A Rose, Yellow Rose IS_A Rose
• A petal is a part of both kinds of flowers	Semantic connection – daisies and roses are more closely related than are petals & flowers Flower HAS_A Petal
• Ladybugs eat certain pests such as aphids, which may be infesting certain kinds of flowers	Symbiotic connection – Ladybugs protect flowers from certain pests Semantic Dependency

Are Roses and Candles related? – Both decorate dinner tables

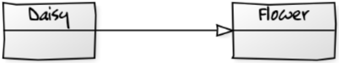
Source: *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007)

Association: RECAP

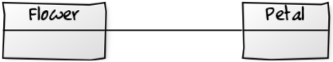
- Semantic Dependencies
  - Most general and most semantically weak
  - Bidirectional by default
  - Often refined over the analysis process



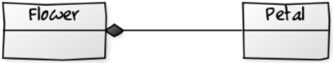
Early relationship



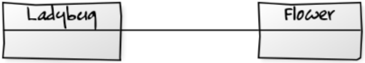
Refined to IS_A



Early relationship



Refined to HAS_A

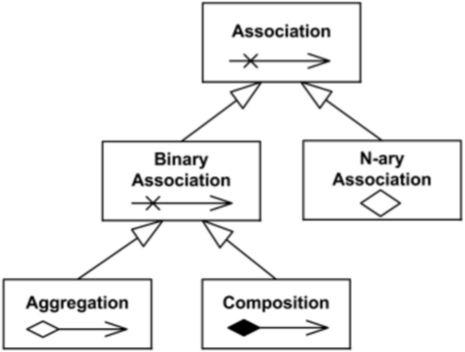


Early relationship

Refined to ?

Association: Notation

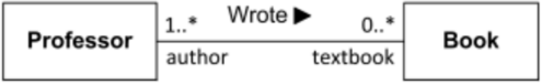
- An association icon (a line connector with label – association name) connects multiple classes and denotes a logical connection
- Associations can be binary of N-ary
- A class may have association to itself (Reflexive)



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Association: Notation

We show an association below between a Professor and a Book



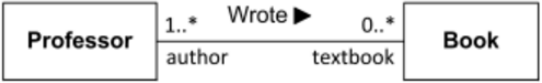
An association has three main concepts

- Association End
- Navigability
- Association Arity

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Association End

- Association end is a connection between the line depicting an association and the icon depicting the connected classifier
- The association end name is commonly referred to as role name
- The role name is optional and suppressible



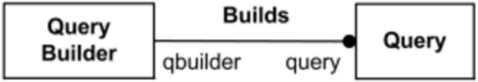
Professor "playing the role" of author is associated with textbook end typed as Book.

- Professor can have multiple roles, like author of some Books or an editor.

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Association End

- Association end could be owned either by end class or association itself
- Ownership of association ends by an associated classifier may be indicated graphically by a small filled circle (aka dot)



Association end query is owned by classifier QueryBuilder and association end qbuilder is owned by association Builds itself

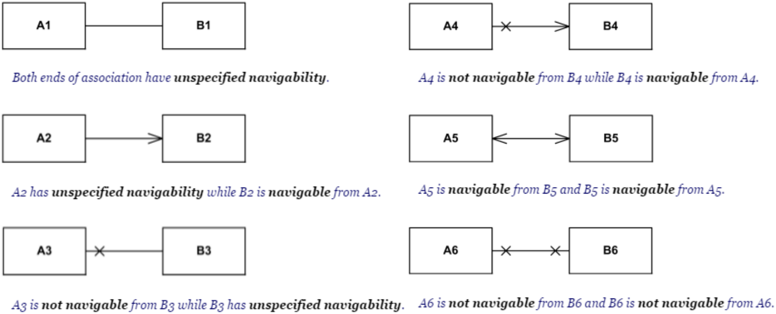
Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Navigability

- End property of association is navigable from the opposite end(s) of association if instances of the classes at this end of the link can be accessed efficiently at run-time from instances at the other ends of the link
- Navigable end is indicated by an open arrowhead on the end of an association
- Not navigable end is indicated with a small x on the end of an association

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Navigability



*Both ends of association have **unspecified navigability**.*

*A4 is **not navigable** from B4 while B4 is **navigable** from A4.*

*A2 has **unspecified navigability** while B2 is **navigable** from A2.*

*A5 is **navigable** from B5 and B5 is **navigable** from A5.*

*A3 is **not navigable** from B3 while B3 has **unspecified navigability**.*

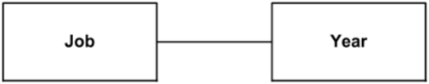
*A6 is **not navigable** from B6 and B6 is **not navigable** from A6.*

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Arity – Binary Association

Each association has specific arity as it could relate two or more classes

- **Binary association** relates two typed instances
- It is normally rendered as a solid line connecting two classifiers, or a solid line connecting a single classifier to itself (the two ends are distinct)
- The line may consist of one or more connected segments

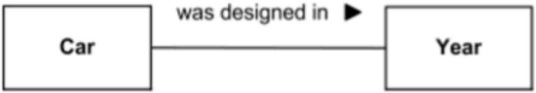


Job and Year classes are associated

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Arity – Binary Association

- A small solid triangle could be placed next to or in place of the name of binary association (drawn as a solid line) to show the order of the ends of the association
- The arrow points along the line in the direction of the last end in the order of the association ends

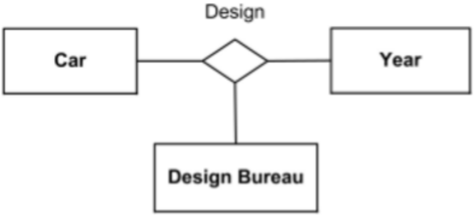


Order of the ends and reading: Car - was designed in - Year

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Arity – N-ary Association

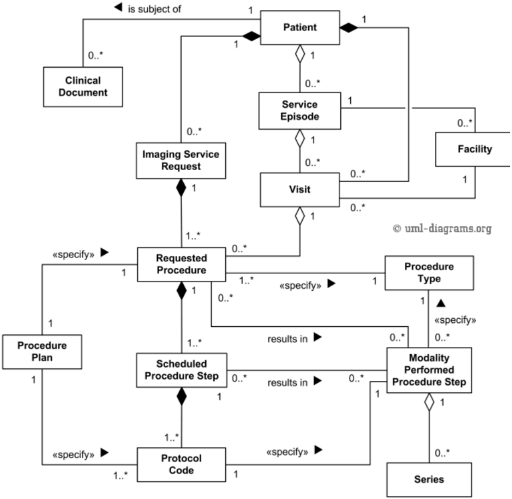
- **N-ary association** may be drawn as a diamond (larger than a terminator on a line) with a solid line for each association end connecting the diamond to the classifier that is the end's type
- N-ary association with more than two ends can only be drawn the following way



Ternary association Design relates three classes

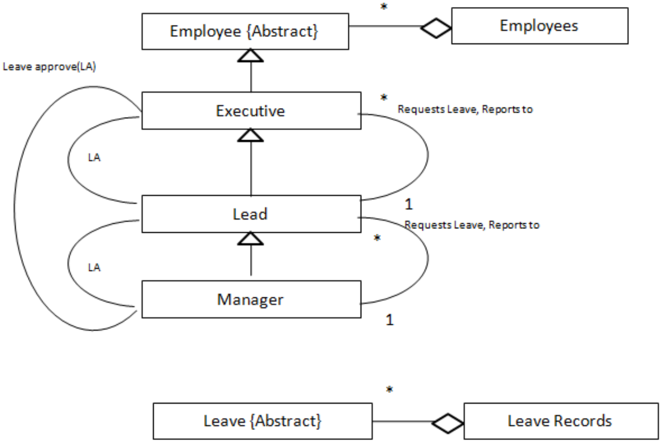
Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Health-care Organization Model



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

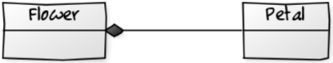
Associations in LMS



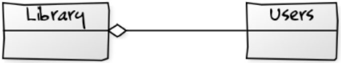
Associations in LMS

Aggregation (HAS_A): RECAP (Module 14)

- Whole / Part relationships
  - ○ Say, we model Flower HAS_A Petal
  - ○ Flower contains many Petals
  - ○ Flower is the Whole, Petal is the Part
  - ○ Depicted as:



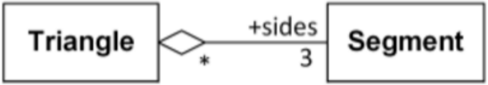
  - ○ Physical Containment – Composition / Strong Aggregation
- Member relationship
  - ○ Say, we model Library HAS Users
  - ○ Library enrolls many Users
  - ○ Library does not contain the Users
  - ○ Depicted as:



  - ○ Conceptual Containment – Weak Aggregation

Weak Aggregation

- Weak aggregation is depicted as an association decorated with a hollow diamond at the aggregate end of the association line



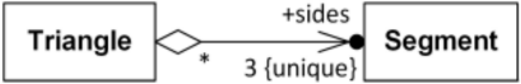
Triangle has 'sides' collection of three line Segments

Each line Segment could be part of none, one, or several triangles

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Weak Aggregation

- Weak aggregation could be depicted together with navigability and association end ownership

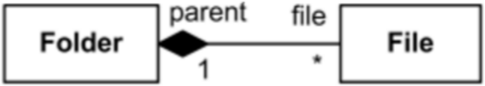


Triangle has 'sides' collection of three unique line Segments.
Line segments are navigable from Triangle.
Association end 'sides' is owned by Triangle, not by association itself

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Strong Aggregation (Composition)

- Strong aggregation (Composition) is depicted as a binary association decorated with a filled black diamond at the aggregate (whole) end.

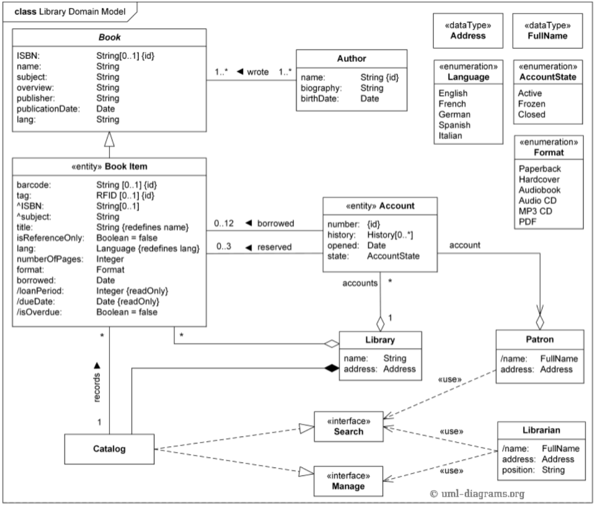


Folder could contain many files, while each File has exactly one Folder parent

If Folder is deleted, all contained Files are deleted as well

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Library Domain Model



class Library Domain Model

Book
ISBN: String[0..1] {id}
name: String
subject: String
overview: String
publisher: String
publicationDate: Date
lang: String

Author
name: String {id}
biography: String
birthDate: Date

«dataType» Address

«dataType» FullName

«enumeration» Language
English
French
German
Spanish
Italian

«enumeration» AccountState
Active
Frozen
Closed

«enumeration» Format
Paperback
Hardcover
Audiobook
Audio CD
MP3 CD
PDF

«entity» Book Item
barcode: String [0..1] {id}
tag: RFID [0..1] {id}
^ISBN: String[0..1]
^subject: String
title: String {redefines name}
isReferenceOnly: Boolean = false
lang: Language {redefines lang}
numberOfPages: Integer
format: Format
borrowed: Date
/loanPeriod: Integer {readOnly}
/dueDate: Date {readOnly}
/isOverdue: Boolean = false

«entity» Account
number: {id}
history: History[0..*]
opened: Date
state: AccountState

Library
name: String
address: Address

Patron
/name: FullName
address: Address

Librarian
/name: FullName
address: Address
position: String

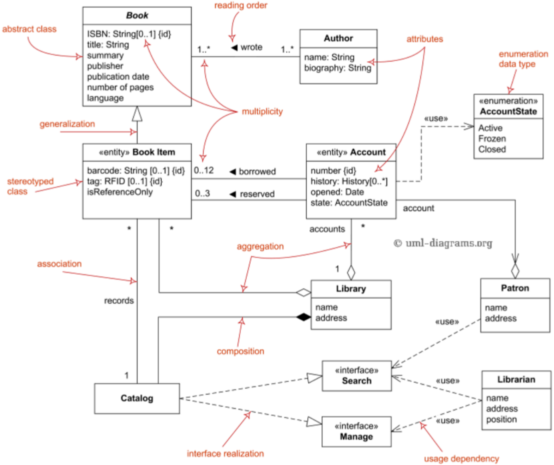
Catalog

«interface» Search

«interface» Manage

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Library Domain Model: Annotated

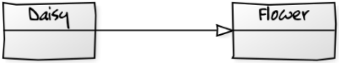


Domain diagram overview - classes, interfaces, associations, usage, realization, multiplicity.

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Inheritance (IS_A): RECAP (Module 14)

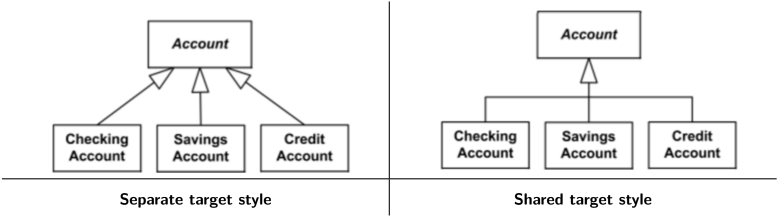
- Generalization / Specialization relationships
  - Say, we model Daisy IS_A Flower
  - Daisy will inherit the properties of Flower, and have some more of its own
  - Flower is the Generalization
  - Daisy is the Specialization
  - Depicted as:



- Semantically most interesting
- Can *delegate* behavior to related objects
- Comes in a number of flavors
  - Single / Multilevel / Hierarchical Inheritance
  - Multiple Inheritance
  - Hybrid Inheritance

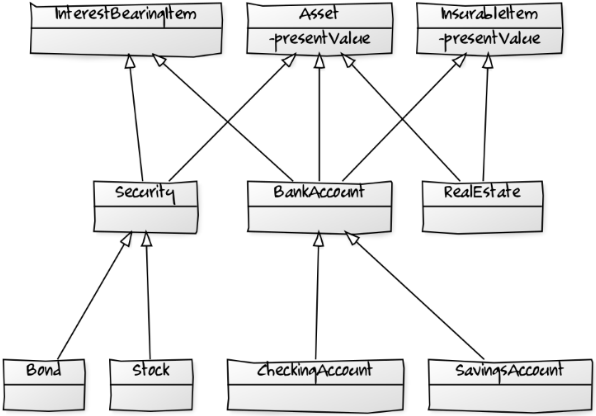
Generalization

- A generalization is shown as a line with a hollow triangle as an arrowhead



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (10-Aug-16)

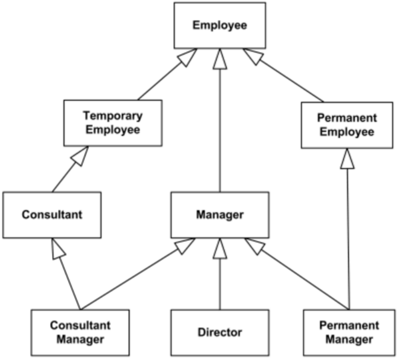
Multiple Inheritance: RECAP (Module 14)



- More than one superclass for a subclass
- RealEstate IS_A Asset, InsurableItem

Multiple Inheritance

- Multiple inheritance is implicitly allowed by UML standard, while the standard provides no definition of what it is.

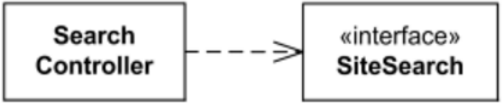


Multiple inheritance for Consultant Manager and Permanent Manager – both inherit from two classes

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (10-Aug-16)

Dependency

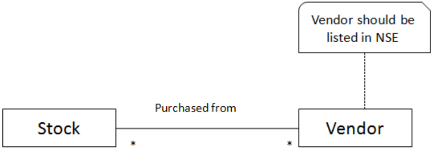
- Dependency is a directed relationship which is used to show that some UML element or a set of elements requires, needs or depends on other model elements for specification or implementation



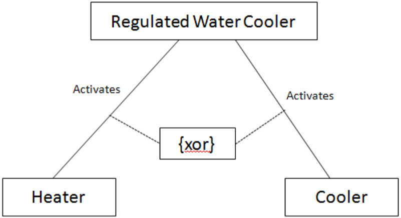
Class SearchController depends on (requires) SiteSearch interface

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Constraints

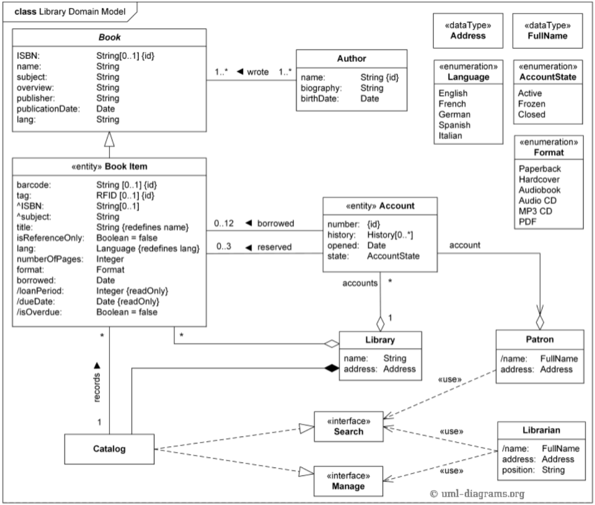


Constraint on Vendor List



Constraint on Activation of Heater and Cooler

Library Domain Model



class Library Domain Model

Book

ISBN: String[0..1] {id}
name: String
subject: String
overview: String
publisher: String
publicationDate: Date
lang: String

Author

name: String {id}
biography: String
birthDate: Date

1..* ◀ wrote 1..*

«dataType» **Address**

«dataType» **FullName**

«enumeration» **Language**

English
French
German
Spanish
Italian

«enumeration» **AccountState**

Active
Frozen
Closed

«enumeration» **Format**

Paperback
Hardcover
Audiobook
Audio CD
MP3 CD
PDF

«entity» **Book Item**

barcode: String [0..1] {id}
tag: RFID [0..1] {id}
^ISBN: String[0..1]
^subject: String
title: String {redefines name}
isReferenceOnly: Boolean = false
lang: Language {redefines lang}
numberOfPages: Integer
format: Format
borrowed: Date
/loanPeriod: Integer {readOnly}
/dueDate: Date {readOnly}
/isOverdue: Boolean = false

«entity» **Account**

number: {id}
history: History[0..*]
opened: Date
state: AccountState

0..12 ◀ borrowed

0..3 ◀ reserved

account

accounts *

1

Library

name: String
address: Address

Patron

/name: FullName
address: Address

*

*

records ▶

1

Catalog

«interface» **Search**

«interface» **Manage**

Librarian

/name: FullName
address: Address
position: String

«use»

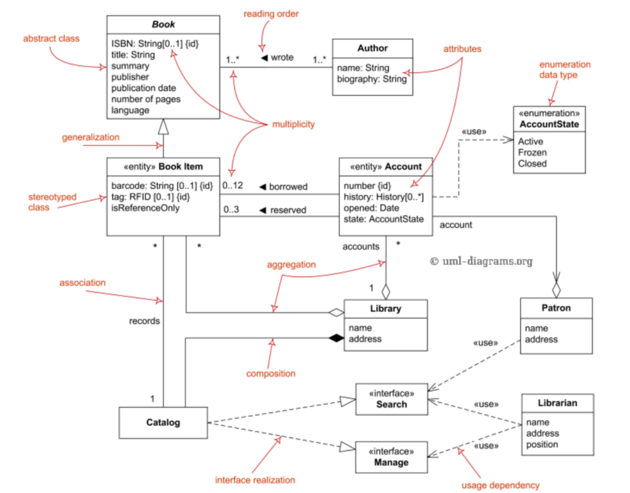
«use»

«use»

© uml-diagrams.org

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

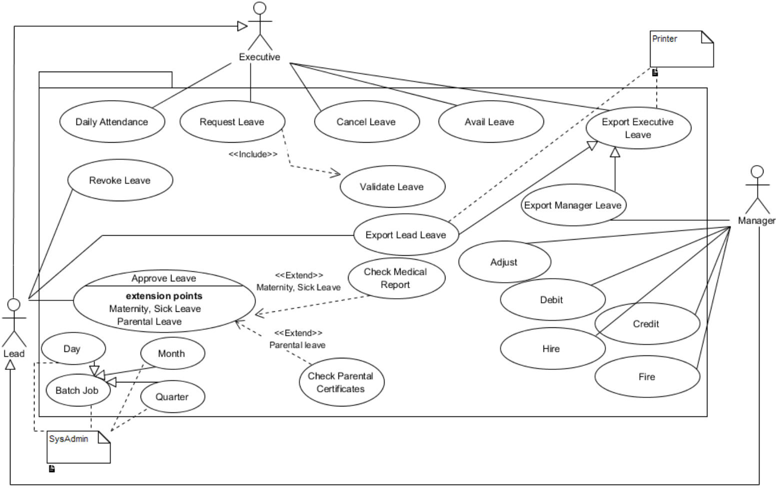
Library Domain Model: Annotated



Domain diagram overview - classes, interfaces, associations, usage, realization, multiplicity.

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

Use-Case Diagram for LMS RECAP (Module 25)



Not all use cases are shown in details

Class Diagram for LMS

We now derive the Class Diagram for LMS. The steps involved are:

- Identify Classes {Abstract Classes}
- Identify Properties and Operations
- Identify the Relationships among Classes
- Class Diagram

Identification of Classes {Abstract Classes}

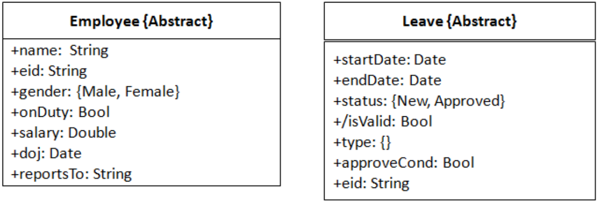
- Reading through the specification of the Leave Management System, we identify the various instances, that is, objects
- We categorize them into two abstract classes: Employee and Leave

Employee {Abstract}

Leave {Abstract}

Identification of Properties

Properties of the two abstract class of LMS



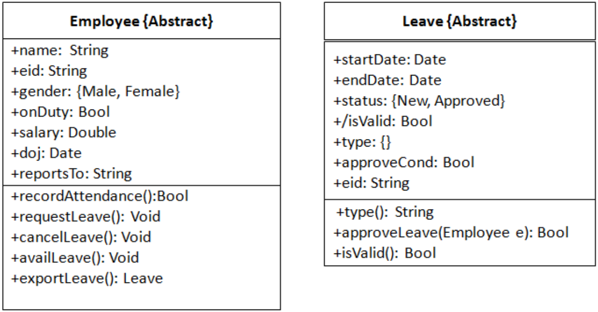
Employee {Abstract}

+name: String
+eid: String
+gender: {Male, Female}
+onDuty: Bool
+salary: Double
+doj: Date
+reportsTo: String

Leave {Abstract}

+startDate: Date
+endDate: Date
+status: {New, Approved}
+/isValid: Bool
+type: {}
+approveCond: Bool
+eid: String

Identification of Operations



Employee {Abstract}

+name: String
+eid: String
+gender: {Male, Female}
+onDuty: Bool
+salary: Double
+doj: Date
+reportsTo: String

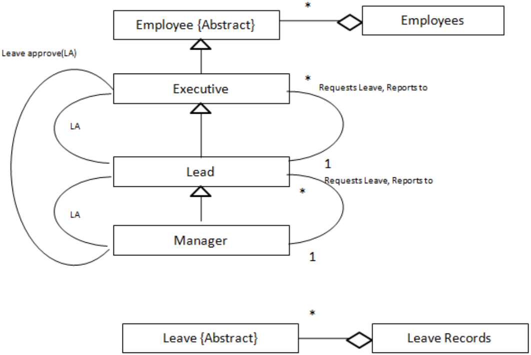
+recordAttendance():Bool
+requestLeave(): Void
+cancelLeave(): Void
+availLeave(): Void
+exportLeave(): Leave

Leave {Abstract}

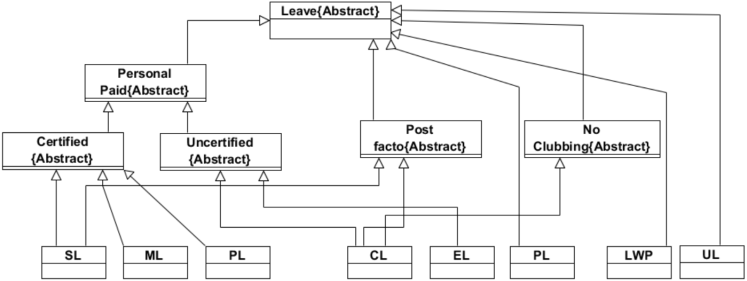
+startDate: Date
+endDate: Date
+status: {New, Approved}
+/isValid: Bool
+type: {}
+approveCond: Bool
+eid: String

+type(): String
+approveLeave(Employee e): Bool
+isValid(): Bool

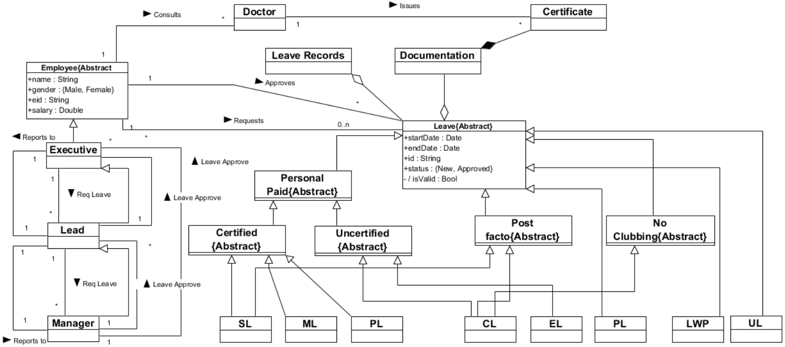
Identification of Associations



Identification of Generalizations



LMS Class Diagram (Partial)



Module Summary

- Class diagrams are introduced
- Representations for properties and operations are discussed
- An example is used for detailed illustration
- Association Relationships among classes are discussed
- Weak Aggregation and Strong Aggregation are important binary associations