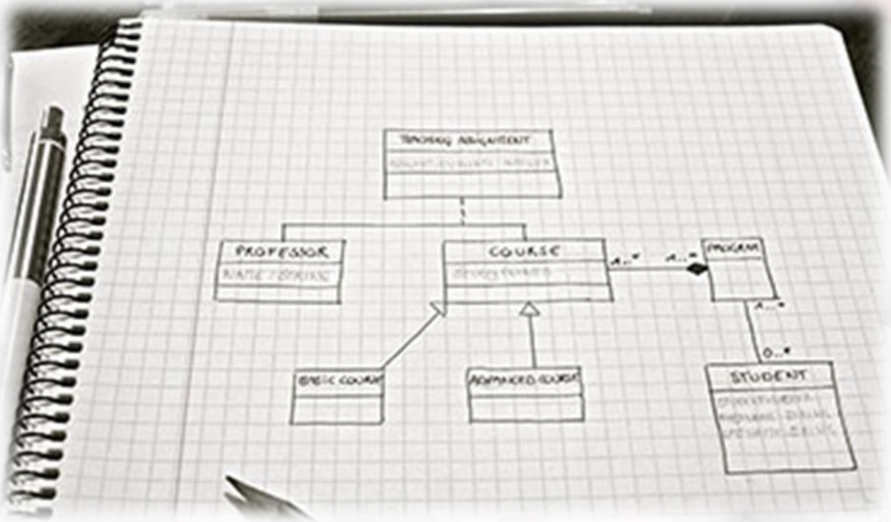
Object Oriented Analysis and Design ICT2133

Buddhika Gayashani Jayaneththi
Dept. of ICT
Faculty of Technology
University of Ruhuna

LECTURE 08 : Class Diagrams



Objectives

After successful completion of this lecture, students should be able to:

- Identify the difference between domain model vs. design model class diagrams.
- Draw class diagrams by analyzing a given scenario.

Analysis to Design

- A transition from "what" the system must do, to "how" the system will do it.

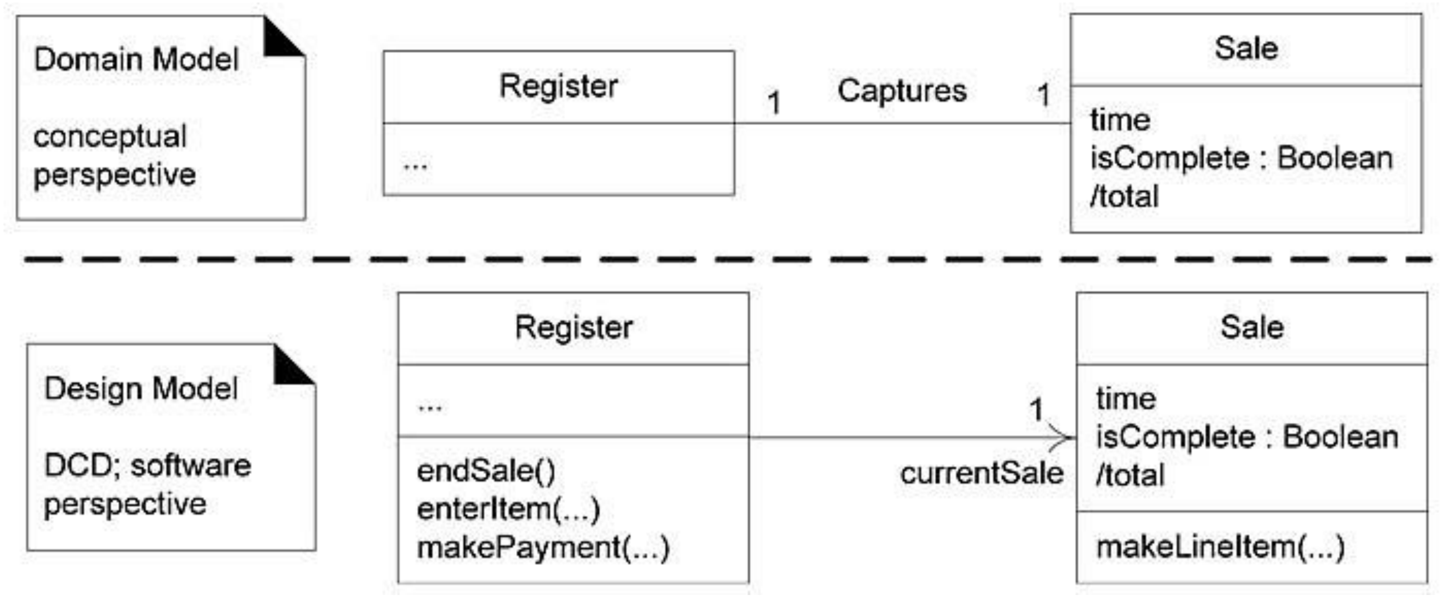
Two main models of design

1. Design class diagram
   – Design static view
2. Sequence diagram
   – Design dynamic view

- Domain models forms the basis for design class model.
- A useful domain model should capture essential abstractions in the modeling domain.

Domain Model vs Design Class Model

- A **domain class diagram** models the concepts in the problem space (just shows the name of the classes and helps model the vocabulary of the problem space).
- A **design class diagram** helps to model the actual design concepts (has more detail than the domain diagram).



Design Class Models

- Software classes with attributes & operations together with detailed relationships between the classes.
- Design class model extends the domain class model by refining attributes & relationships & by refining operations.

Class Diagrams

Class Diagrams

- ➢ Gives an overview of a system by showing its **classes** and the **relationships** among them.
- ➢ Also shows **attributes** and **operations** of each class.
- ➢ Good way to describe the overall architecture of system components.

Finding Classes

Two ways,

- – Finding classes using noun/verb analysis
- – Finding classes using CRC analysis

Finding classes using noun/verb analysis

- Noun/verb analysis has been used for many years and works well as it is based on a direct analysis of the language of the problem domain.

When a text is given,

- Nouns and noun phrases in the text indicate **classes** or **attributes** of classes.
- Verbs and verb phrases indicate **responsibilities** or **operations** of a class.
- Also find the “hidden” classes, which are intrinsic to the problem domain, but which might never be mentioned explicitly.

Note: Be very aware of synonyms and homonyms

Step 01 – Noun Phrase Identification

The Portfolio Manager shall be able to roll up portfolios on several levels.

A Trader shall be able to place orders, on behalf of a portfolio, that generate one or more trades.

A Portfolio Manager shall be able to select a pairoff method in conjunction with placing a sell order.

The entry of a trade shall generate forecasted cashflows associated with the given trade lot.

The system shall match up actual cashflows with forecasted cashflows.

The system shall automatically generate appropriate postings to the General Ledger.

The system shall allow an Assistant Trader to modify trade data and propagate the results accordingly.

Identify the nouns and noun phrases in this example system definition.

Step 0 – Noun Phrase Identification

The **Portfolio Manager** shall be able to roll up **portfolios** on several **levels**.

A **Trader** shall be able to place **orders**, on behalf of a **portfolio**, that generate one or more **trades**.

A **Portfolio Manager** shall be able to select a **pairoff method** in conjunction with placing a **sell order**.

The entry of a **trade** shall generate **forecasted cashflows** associated with the given **trade lot**.

The **system** shall match up **actual cashflows** with **forecasted cashflows**.

The **system** shall automatically generate appropriate **postings** to the **General Ledger**.

The **system** shall allow an **Assistant Trader** to modify **trade data** and propagate the **results** accordingly.

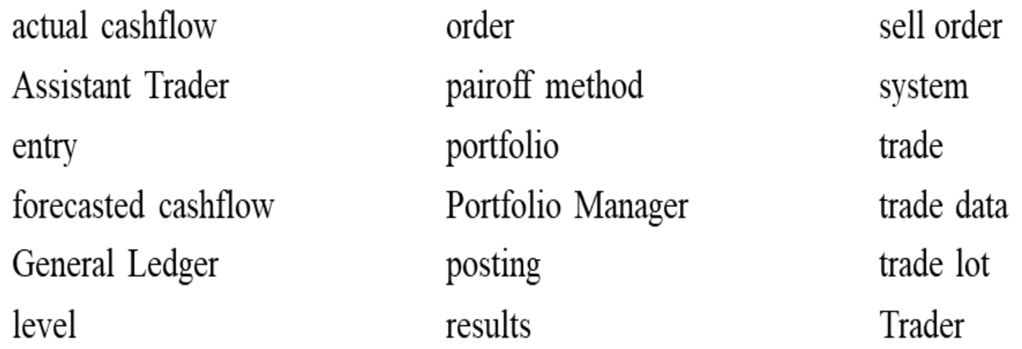
Identify the nouns and noun phrases in this example system definition.

Step 02 – Noun Phrase Consolidation

After identifying nouns and noun phrases,

- Make plural terms singular.
- Eliminate duplicate terms.
- Consolidate synonyms into single term.
- Place list in alphabetic order.

Step 02 – Noun Phrase Consolidation



actual cashflow	order	sell order
Assistant Trader	pairoff method	system
entry	portfolio	trade
forecasted cashflow	Portfolio Manager	trade data
General Ledger	posting	trade lot
level	results	Trader

After the consolidating noun phrases

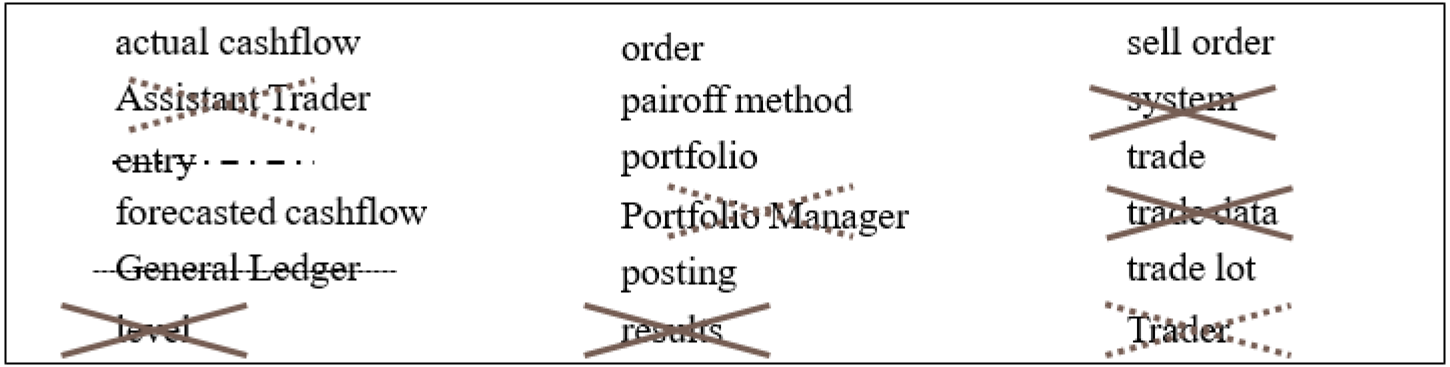
Step 03 – Noun Phrase Analysis

Now, remove:

- References to the system itself or to its context.
- Nouns which appear out of the scope of the project.
- Nouns which are too vague.
- Nouns which represent actions.

➢ Actors might be removed from our list of candidate classes, and placed instead on the use case diagrams.

Step 03 – Noun Phrase Analysis



actual cashflow	order	sell order
~~Assistant Trader~~	pairoff method	~~system~~
~~entry~~	portfolio	trade
forecasted cashflow	~~Portfolio Manager~~	~~trade data~~
~~General Ledger~~	posting	trade lot
~~level~~	~~results~~	~~Trader~~

⨯	actors
·–·–·–·–·	actions
------------	out of the scope of the project
✕	too vague

Step 4. Draw Initial Analysis Class Model

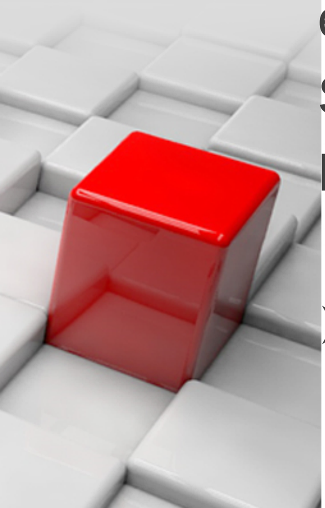
- Draw an initial analysis class model showing just class names, generalizations and associations.
- Just as noun phrase analysis helped identify candidate classes, verb phrase analysis can help identify class relationships.
- The result can be a table showing each candidate associations with your candidate classes.

e.g.,

Student **enrolls** in Course

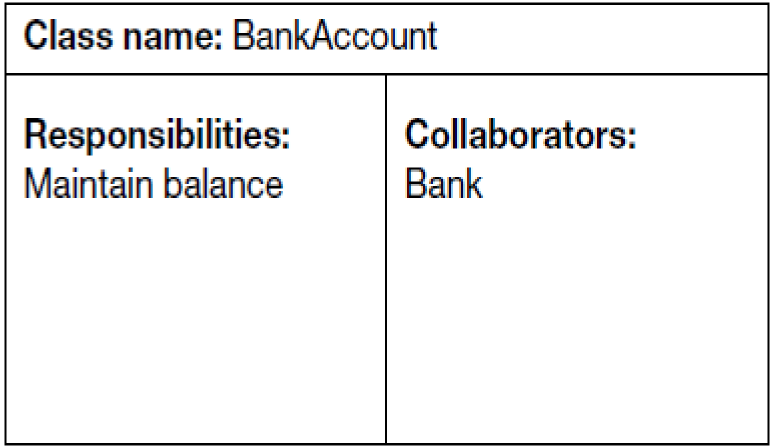
Professor **teaches** Course

- This gives you a first cut class model that you can refine by further analysis.

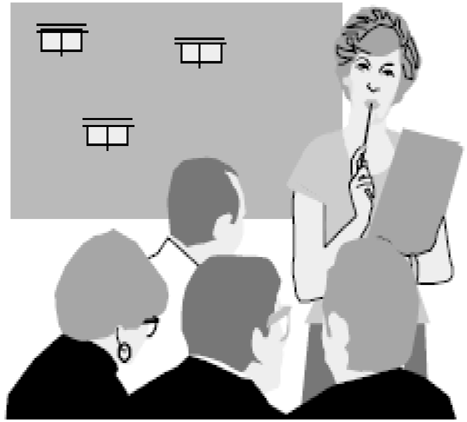


Finding classes using CRC analysis

- Stands for **class**, **responsibilities** and **collaborators**
- Uses the world's most powerful analysis tool, the **sticky note**!
- Begin by marking up some sticky notes as shown below.



Class name: BankAccount	
Responsibilities: Maintain balance	**Collaborators:** Bank



Finding classes using CRC analysis

- The note is divided into three compartments
  - Top compartment : Name of the candidate class
  - Left compartment: The responsibilities
  - Right compartment: The collaborators
- Collaborators are other classes that may collaborate with this class to realize a piece of system functionality.
- The collaborators compartment provides a way of recording relationships between classes.

CRC Analysis Procedure

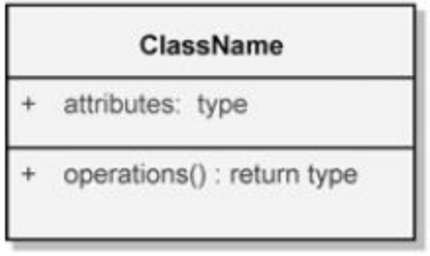
Two phase activity

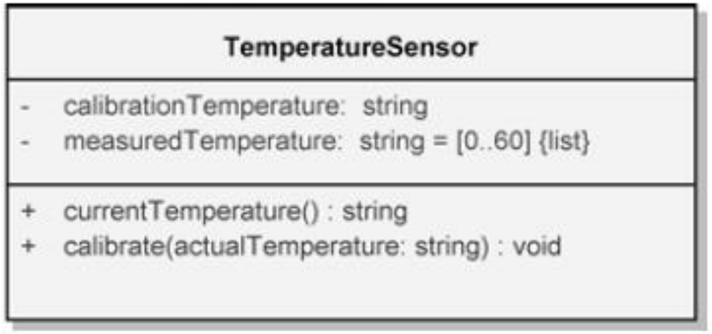
1. Brainstorm – gather the information
2. Analyze information

Class Diagram Notations

Classes

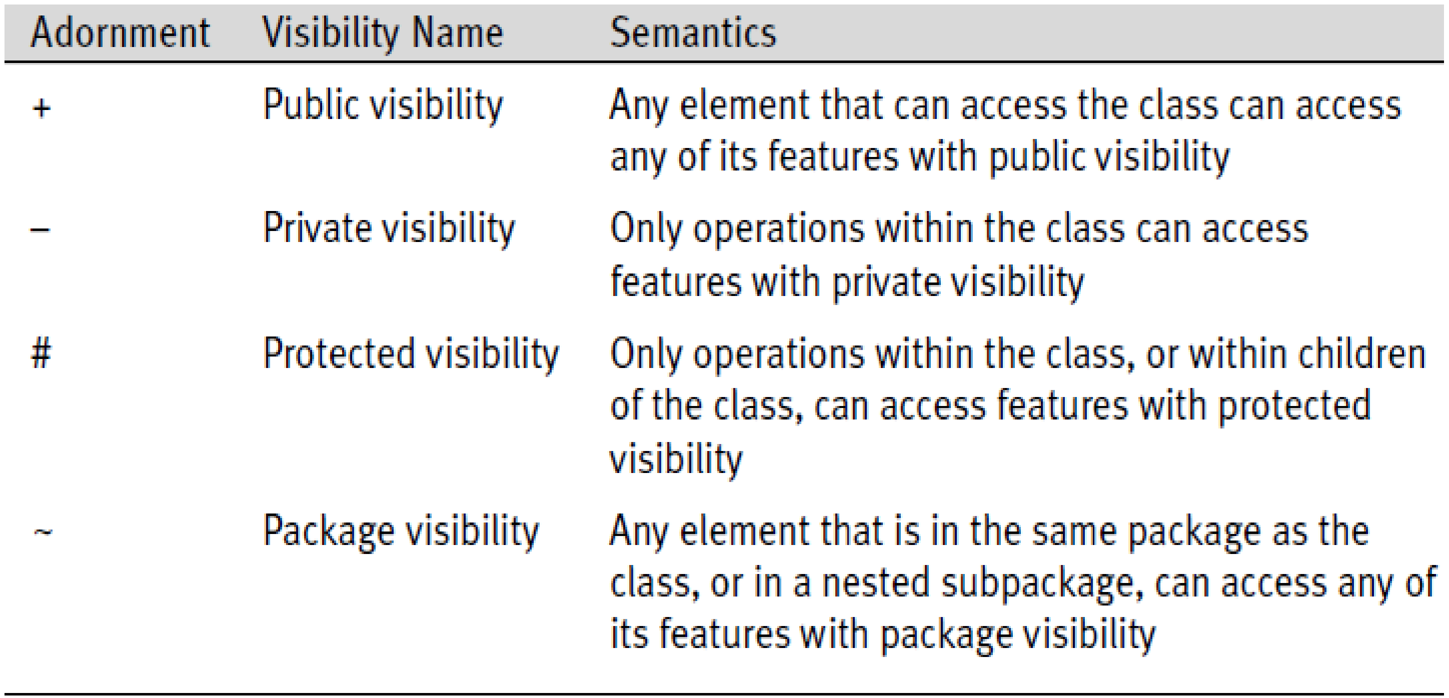
- **Class** : “The descriptor for a set of objects that share the same attributes, operations, methods, relationships and behavior”.
- Each class is represented by a rectangle, subdivided into three compartments.
  - Name
  - Attributes
  - Operations




TemperatureSensor
- calibrationTemperature: string
- measuredTemperature: string = [0..60] {list}
+ currentTemperature() : string
+ calibrate(actualTemperature: string) : void

Class

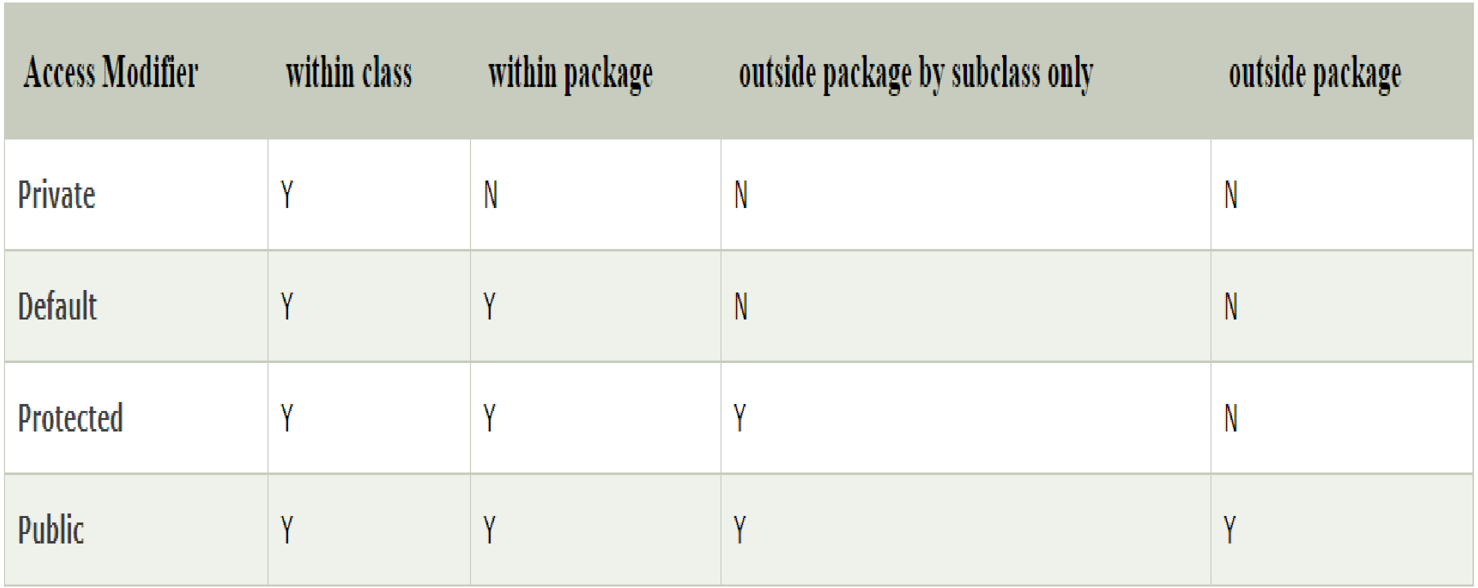
- Modifiers are used to indicate visibility of attributes and operations.



Adornment	Visibility Name	Semantics
+	Public visibility	Any element that can access the class can access any of its features with public visibility
–	Private visibility	Only operations within the class can access features with private visibility
#	Protected visibility	Only operations within the class, or within children of the class, can access features with protected visibility
~	Package visibility	Any element that is in the same package as the class, or in a nested subpackage, can access any of its features with package visibility

Class

- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.



Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Class Attributes & Methods

Attributes (fields, instance variables)

visibility name **: data_*type***

Example:

- balance : double

Operations / methods

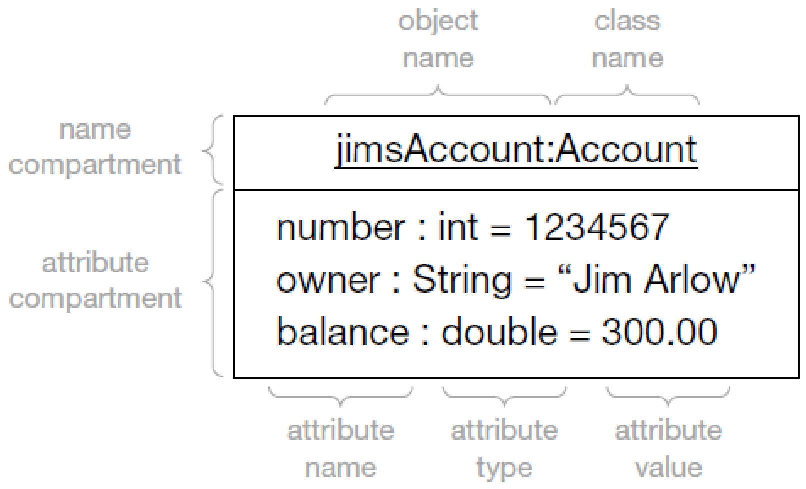
visibility name **(*parameters*) : *return_type***

Example:

+ calDistance(p1: double): double

Object

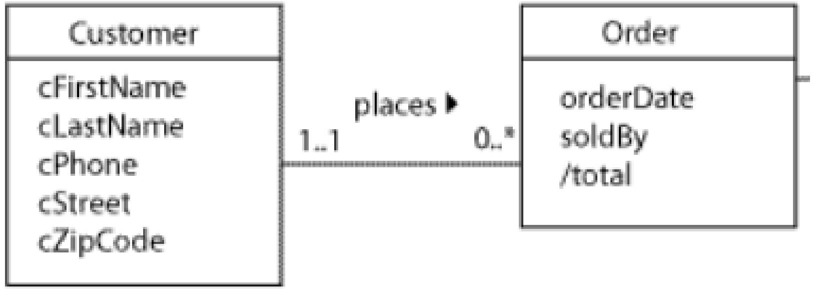
Object : An instance of a class

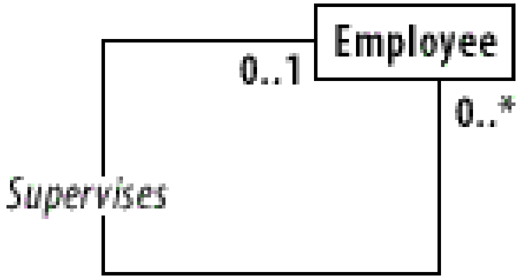


The top compartment contains the object identifier, which is *always* underlined.

Relationships- Association

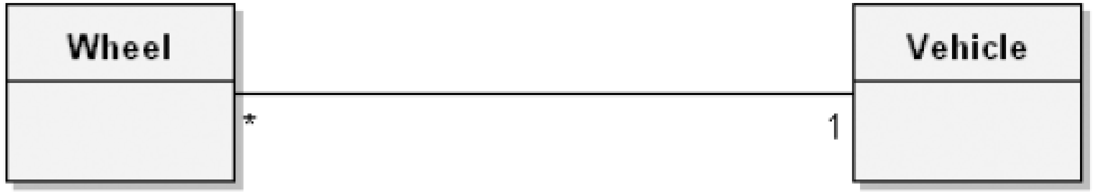
- Connects two classes and denotes a meaningful connection.
- Indicated by a straight line or arrow (direct association).
- A class may have an association to itself (called a ***reflexive association***).
- Associations may be further decorated with their multiplicity.





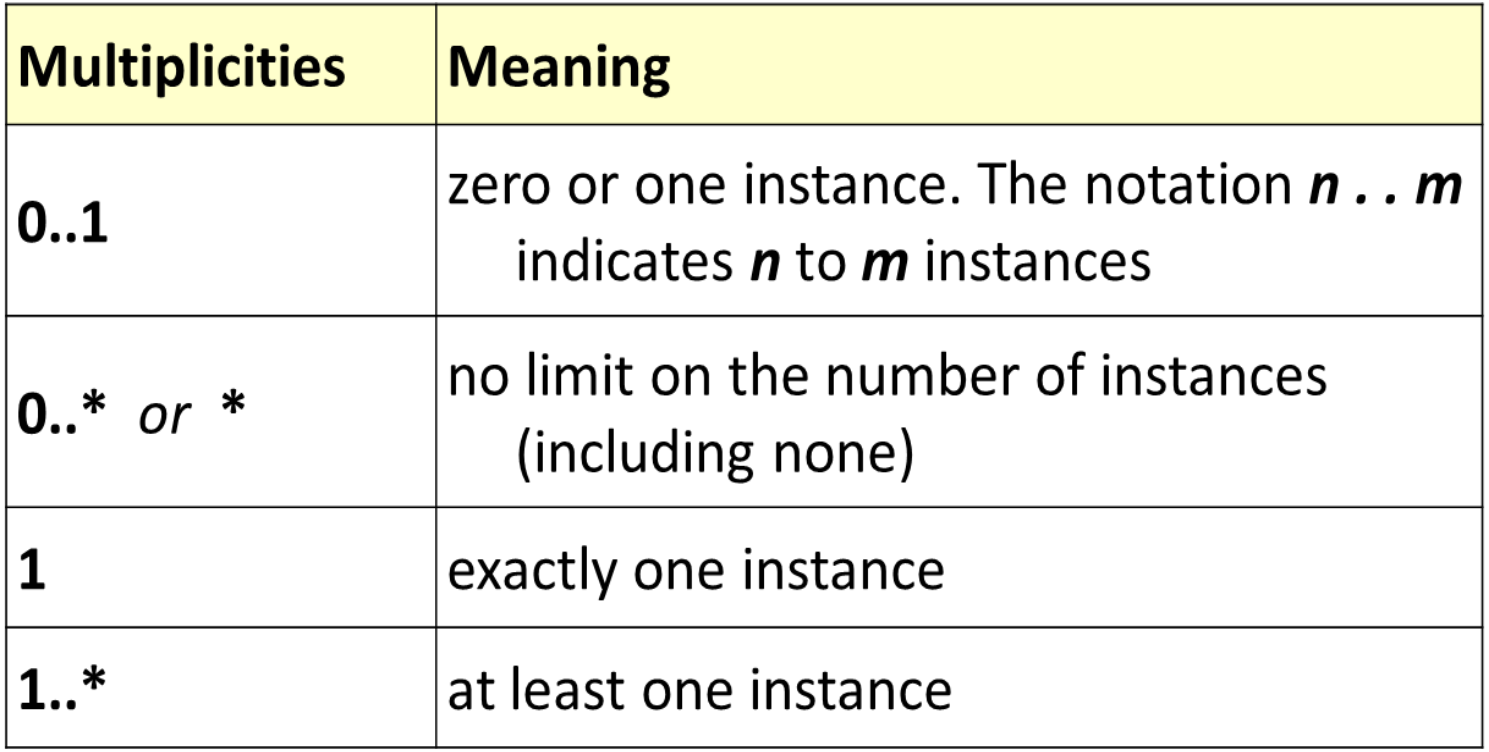
Multiplicity

- How many instances of class A can be associated with one instance of class B.
- A single instance of **<u>vehicle</u>** can be associated with many wheel **<u>instances.</u>**



- Multiplicity is applied to the target end of an association and denotes the number of links between each instance of the source class and instances of the target class.

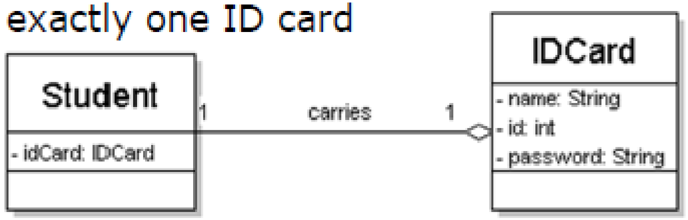
Multiplicity



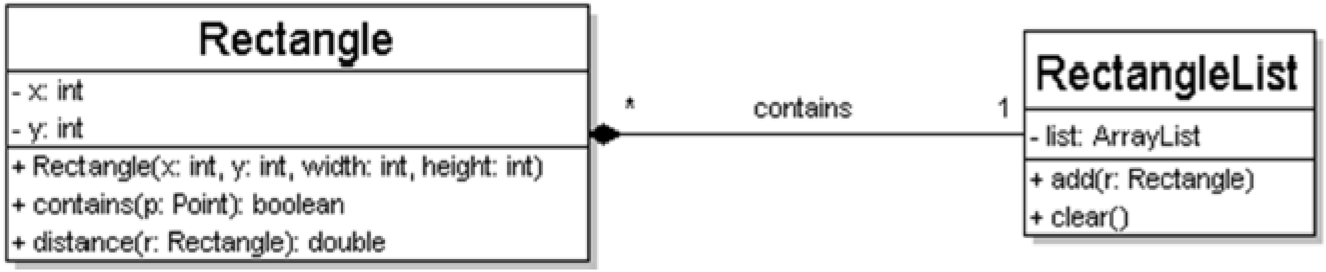
Multiplicities	**Meaning**
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances
0..* *or* *****	no limit on the number of instances (including none)
1	exactly one instance
1..*	at least one instance

Multiplicity

- one-to-one
  - each student must carry exactly one ID card

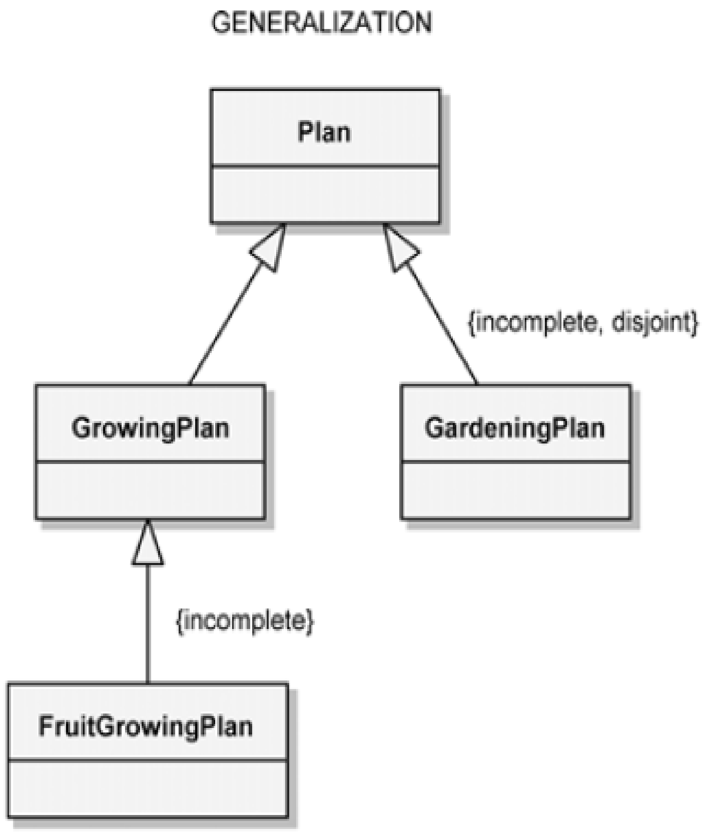


- one-to-many
  - one rectangle list can contain many rectangles


Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double
*
contains
1
RectangleList
- list: ArrayList
+ add(r: Rectangle)
+ clear()

Relationships- Generalization

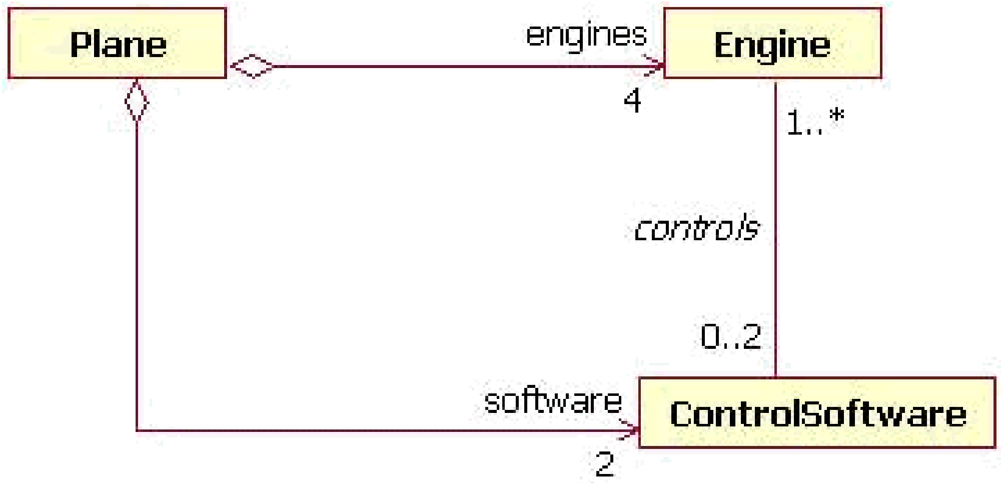
- The generalization icon denotes a generalization/ specialization relationship (the “**is a**” relationship).
- Appears as an association with a closed arrowhead.
- The arrow head points to the superclass, and the opposite end of the association designates the subclass.



Relationships- Aggregation

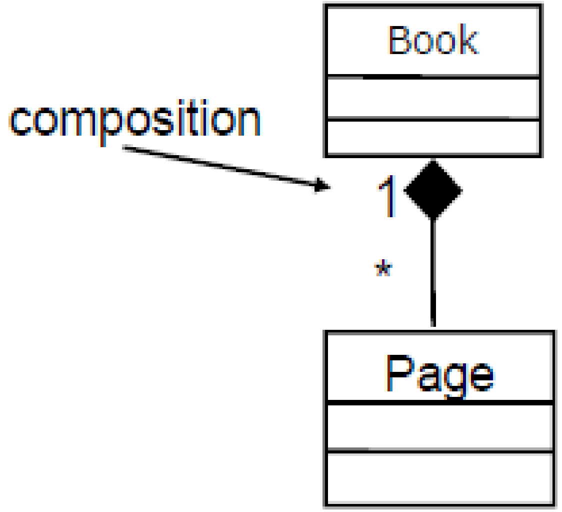
- A constrained form of the association relationship **“is part of” relationship.**
- An association where one class belongs to a collection.

  e.g. Instructor is part of Faculty
- Indicated by an empty diamond on the side of the collection to denote the aggregate (the whole).

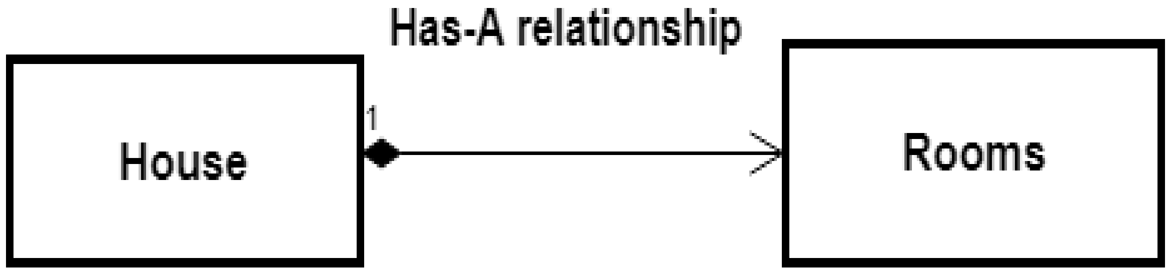


Relationships - Composition

- Strong form of Aggregation.
- Components cannot exist without the aggregate.
- Indicated by a solid diamond on the side of the collection.
- The multiplicity at this end is 1 because the parts are defined as having no meaning outside the whole, which owns the parts.
- **“is entirely made of” relationship**



Relationships - Composition

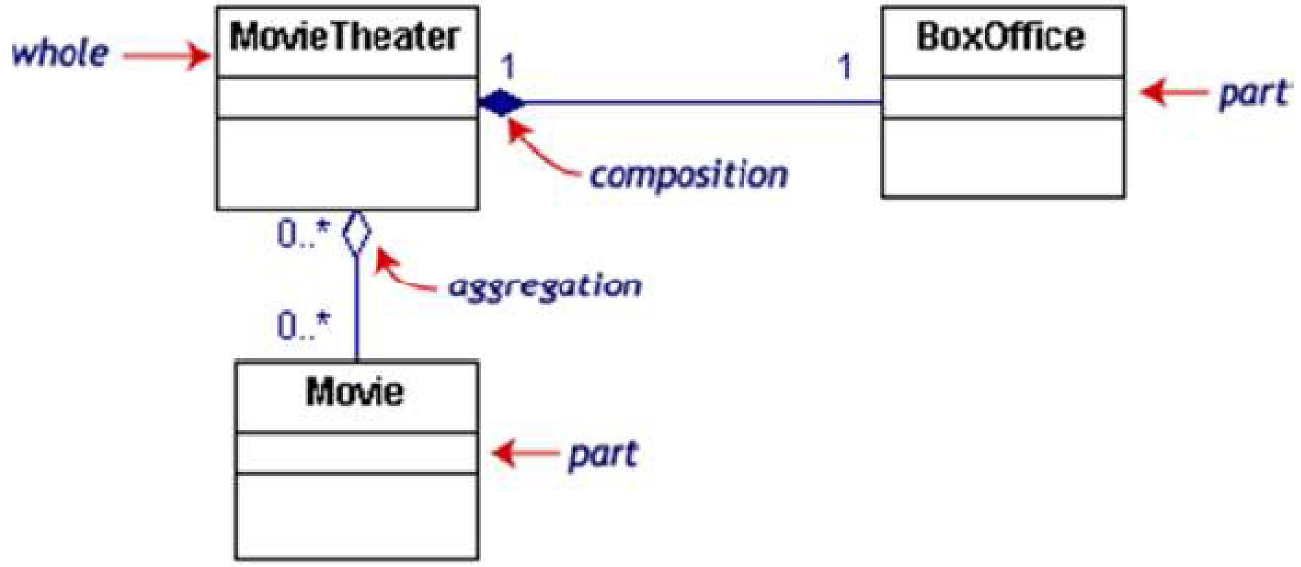


It represents part-of relationship

In composition, both the entities are dependent on each other

House is a whole and Rooms are parts. If an House is deleted then all corresponding Rooms for that House should be deleted

Composition / Aggregation Example



If the movie theater goes away
so does the box office => composition
but movies may still exist => aggregation

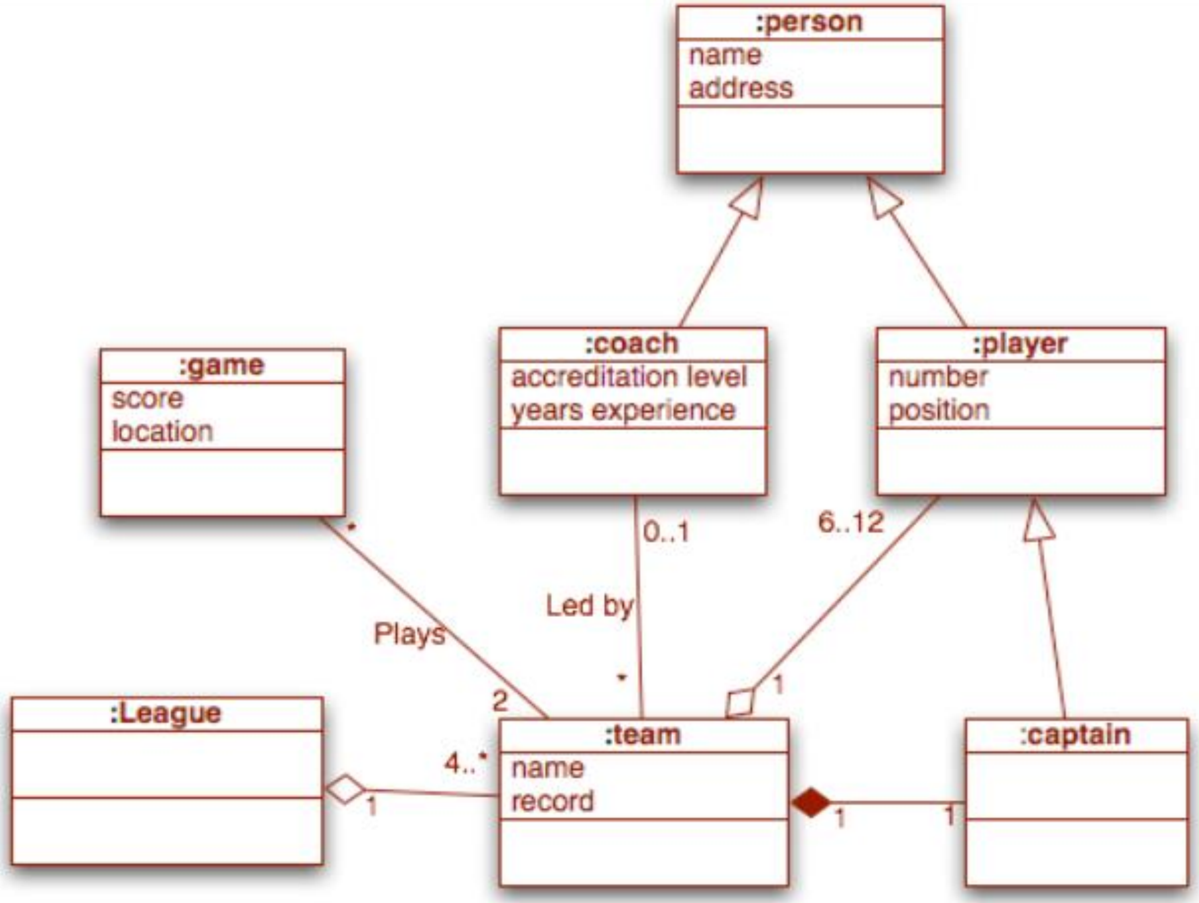
Exercise

Draw a UML Class Diagram representing the following elements from the problem domain of a hockey league. A hockey league is made up of at least four hockey teams. Each hockey team is composed of six to twelve players, and one player captains the team. A team has a name and a record. Players have a number and a position. Hockey teams play games against each other. Each game has a score and a location. Teams are sometimes lead by a coach. A coach has a level of accreditation and a number of years of experience, and can coach multiple teams. Coaches and players are people, and people have names and addresses. Draw a class diagram for this information, and be sure to label all associations with appropriate multiplicities.

Assumptions:

- each player only plays on one team
- each captain only captains one team
- each team only plays in one league

Answer


:person
name
address
:coach
accreditation level
years experience
:player
number
position
:game
score
location
:League
:team
name
record
:captain
*
Plays
2
0..1
Led by
*
6..12
1
4..*
1
1
1

THANK YOU!