CSE 223: Programming -2
02-OOA & UML (I)

Prof. Dr. Khaled Nagi

Department of Computer and Systems Engineering,

Faculty of Engineering, Alexandria University, Egypt.

Object Oriented Analysis (OOA)

Software Process

- Software Process defines the way to produce software. It includes
  - Software life-cycle model
  - Tools to use
  - Individuals building software
- Software life-cycle model defines how different *phases* of the life cycle are managed.
- Phases of Software Life-cycle
  - Requirements
  - Specification (Analysis)
  - Design
  - Implementation
  - Integration
  - Maintenance
  - Retirement

Requirements

- Assumption
  - The software being considered is considered economically justifiable.
- Concept exploration
  - Determine what the client needs, *not* what the client wants
- Document - Requirements Document

Specification (Analysis) Phase

- From the customer requirements identify *what* to build.
- Specifications must not be
  - Ambiguous
  - Incomplete
  - Contradictory
- Document – Specification Document

Design Phase

- From the specification identify *how* to build.
- Design involves two steps
  - Architectural Design – Identify modules
  - Detailed Design – Design each modules
- Document – Architecture Document, Design Document

Implementation Phase

- Implement the detailed design in code.
- Developer testing
  - Unit testing
  - Module testing
- Document – Commented source code

Integration Phase

- Combine the modules and test the product as a whole.
- Testing includes
  - Product testing
  - Acceptance testing
- Document – Test cases and test results

Maintenance Phase

- Any changes after the customer accepts the system.
- Maintenance phase is the most expensive
  - Lack of documentation
  - Regression testing
- Document – Documented Changes, Regression test cases

Retirement Phase

- Good software is maintained
- Sometimes software is rewritten from scratch
  - Software is now un-maintainable because
    - A drastic change in design has occurred
    - The product must be implemented on a totally new
    - hardware/operating system
    - Documentation is missing or inaccurate
    - Hardware is to be changed—it may be cheaper to rewrite
    - the software from scratch than to modify it
- True retirement is a rare event

Three Steps of OOA

1. Use-case modeling (“functional model”)
   - Determine how the various results are computed by the product (without regard to sequencing). Largely action oriented. Present in the form of a *use-case diagram* and associated *scenarios*
2. Class modeling (“object modeling”)
   - Determine the classes and their attributes. Then determine interactions among classes. Purely data-oriented. Present in the form of a *class diagram*
3. Dynamic modeling
   - Determine the actions performed by or to each class. Purely action-oriented. Present in the form of a *state diagram* **(for each class)**

Three steps effectively performed in parallel

Unified Modeling Language (UML)

Unified Modeling Language (UML)

- UML is a modeling language for
  - Specifying
  - Visualizing
  - Constructing
  - Documenting

object-oriented software

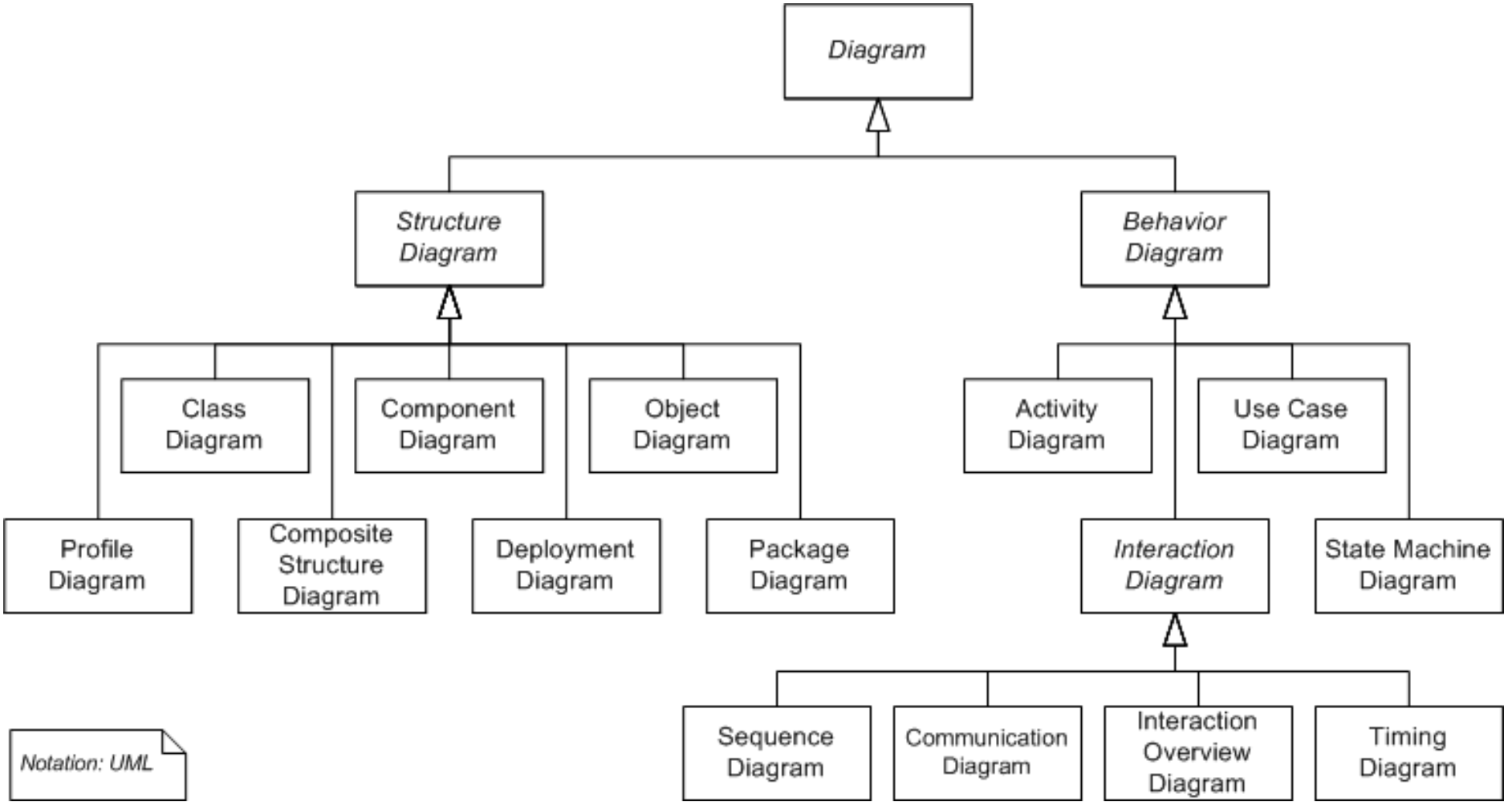
Why UML?

- Software growing larger & more complex
- We need a common language between analysts, designers, developers, and testers.
- UML provides standard graphic notations for modeling
- The standard is managed, and was created by, the Object Management Group (OMG)
- Describe software systems clearly and concisely
- Provide different levels of abstraction

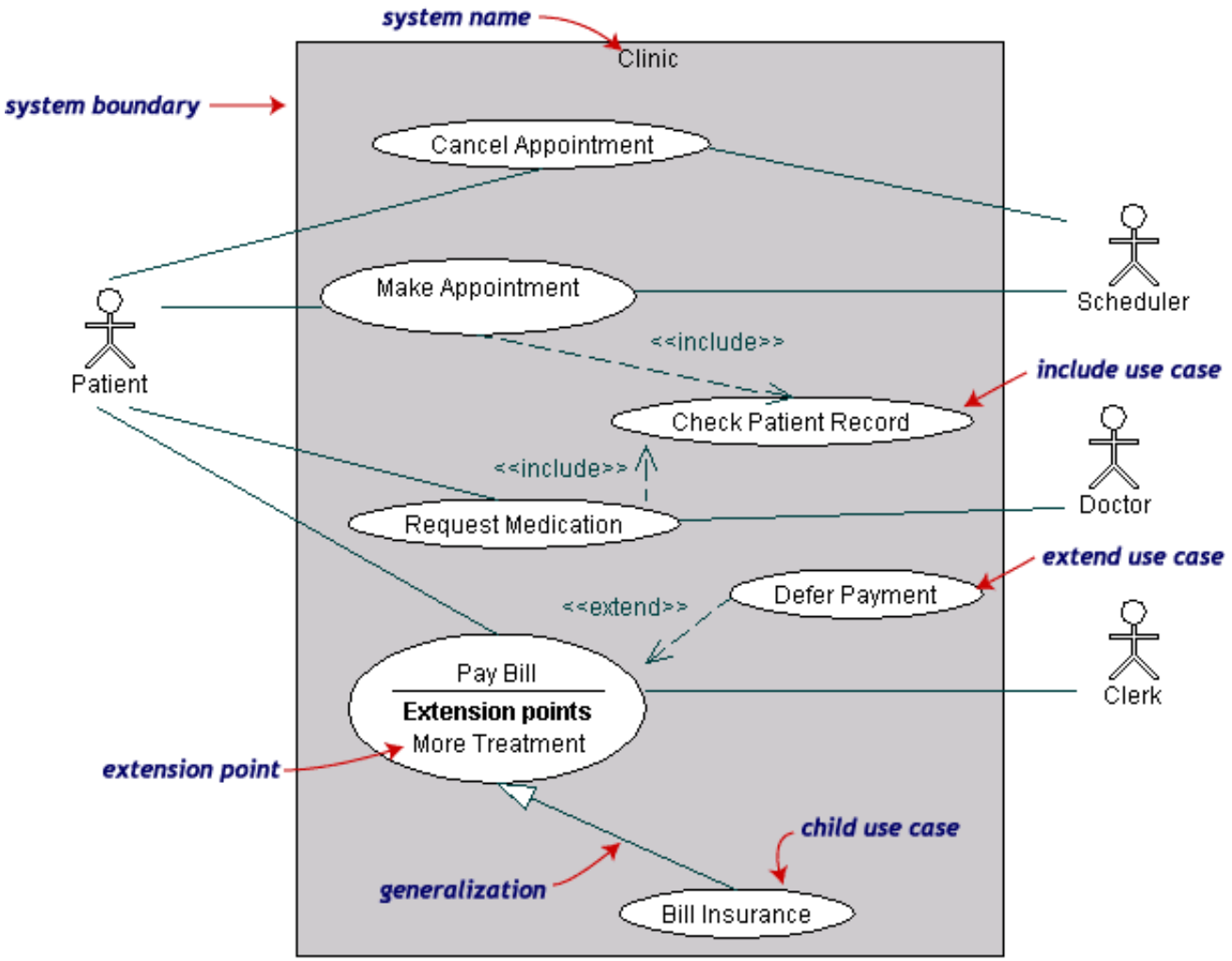
UML Diagrams

- UML diagrams represent two different views of a system model:
  - **Static** (or structural) view: Emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.
  - **Dynamic** (or behavioral) view: Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

Hierarchy of diagrams in UML 2.2



Use-case modeling
Use-Case Diagram of a Medical Clinic

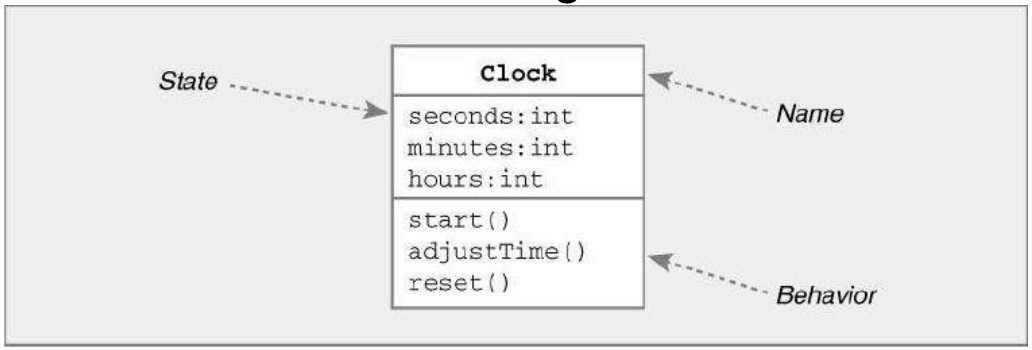


Use-case modeling
Use-Case Diagram of a Medical Clinic

- A use case **generalization** shows that one use case is simply a special kind of another
  - **Pay Bill** is a parent use case and **Bill Insurance** is the child
- **Include** relationships factor use cases into additional ones
  - Both **Make Appointment** and **Request Medication** include **Check Patient Record** as a subtask
- An **extend** relationship indicates that one use case is a variation of another
  - Base use case must declare certain “extension points”
  - The **extension point**, which determines when the extended case is appropriate, is written inside the base case
  - Extending use case may add additional behavior only at those extension points

Structure Diagrams: Class Diagram

- A Class Diagram shows a set of **classes**, **interfaces**, and **collaborations** and their **relationships**.
- Class Diagrams are the most **common** diagram found in modeling object-oriented systems.
- Class Diagrams address the **static** design view of a system.
- Class diagrams are **static** -- they display what interacts but not what happens when they do interact.
  - UML class notation is a rectangle divided into three parts: class name, attributes, and operations.
  - Names of abstract classes are in italics.
  - Relationships between classes are the connecting links.



UML Class Diagrams to Java Code

- Different representation of same information
  - Name, state, behavior of class
  - Relationships between classes
- Should be able to derive one from the other
- Motivation
  - UML → Java
    - Implement code based on design written in UML
  - Java → UML
    - Create UML to document design of existing code

UML ⇔ Java

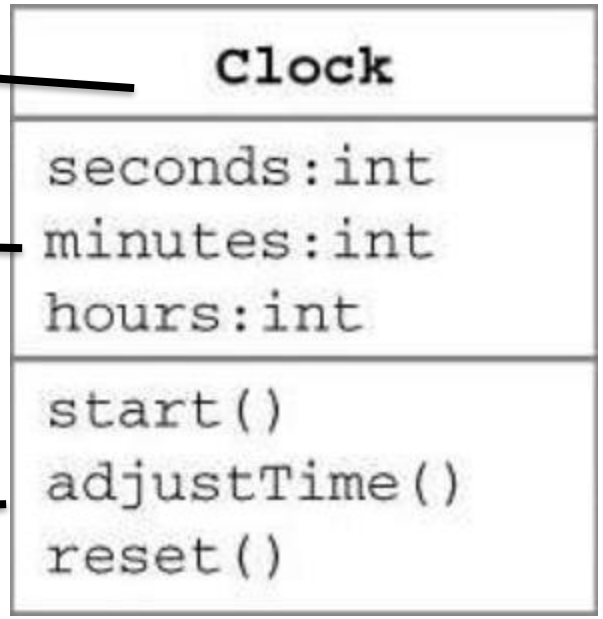
Java Code:

```
class Clock { // name

// state
int seconds;
int minutes;
int hours;

// behavior
void start();
void adjustTime();
void reset();

}
```



Class Elements: Type and Visibility

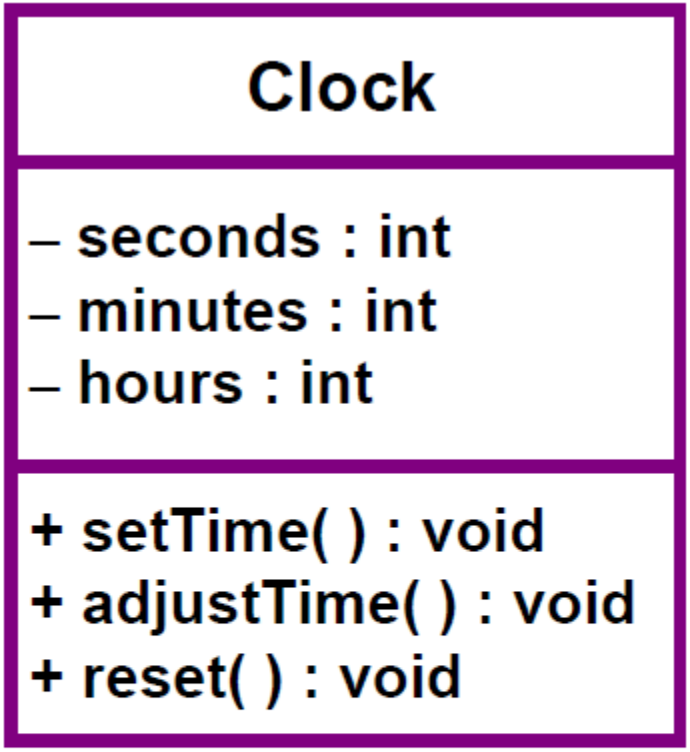
- Type/Return type: type name preceded by colon :
- Visibility: prefix symbol
  - + public
  - – private
  - # protected
  - ~ package
- Static: underline ($ in old notation)

UML ⇔ Java revisited

- Java

```
class Clock { // name
   // state
   private int seconds;
   private int minutes;
   private int hours;
   // behavior
   public void setTime( );
   public void adjustTime(int value);
   public void reset( );
}
```

Java Code

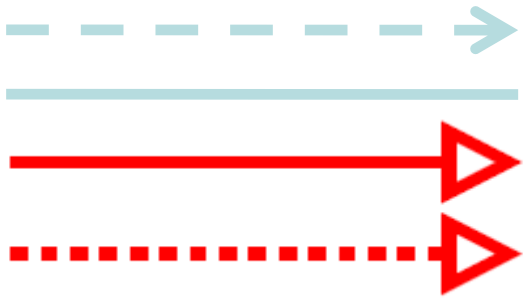


Class Diagram

Relationships in the Class Diagram

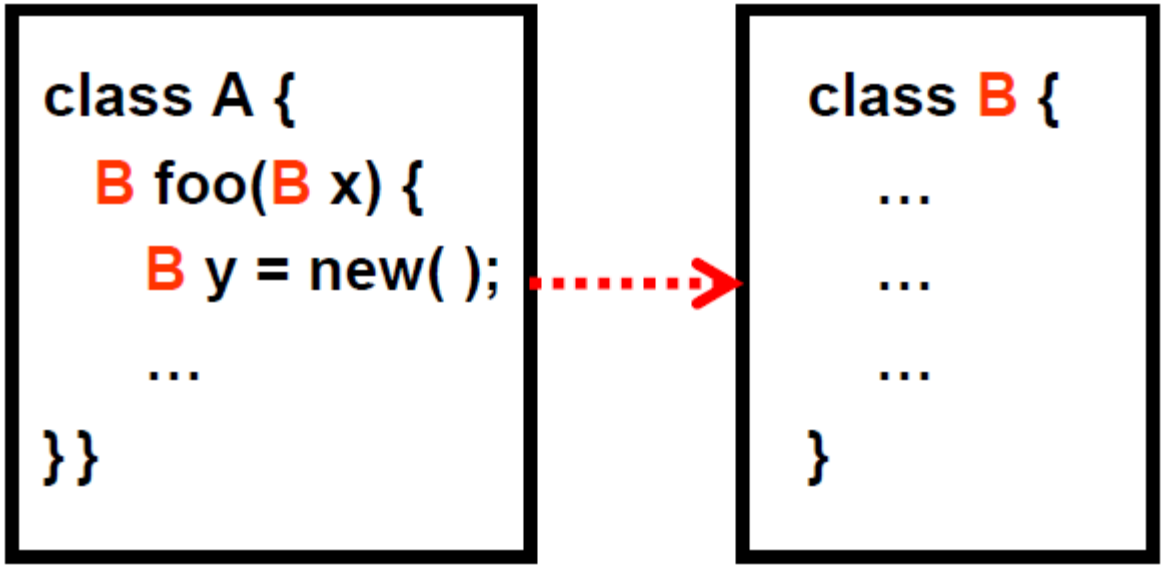
- Dependency
  - a semantic relationship between two things in which a change to one thing may affect the semantics of other thing.
- Association
  - a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
- Generalization
  - an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass.
- Realization
  - a semantic relationship between two things, wherein one thing specifies a contract that the other guarantees to carry out.

- Dependency
- Association
- Generalization
- Realization



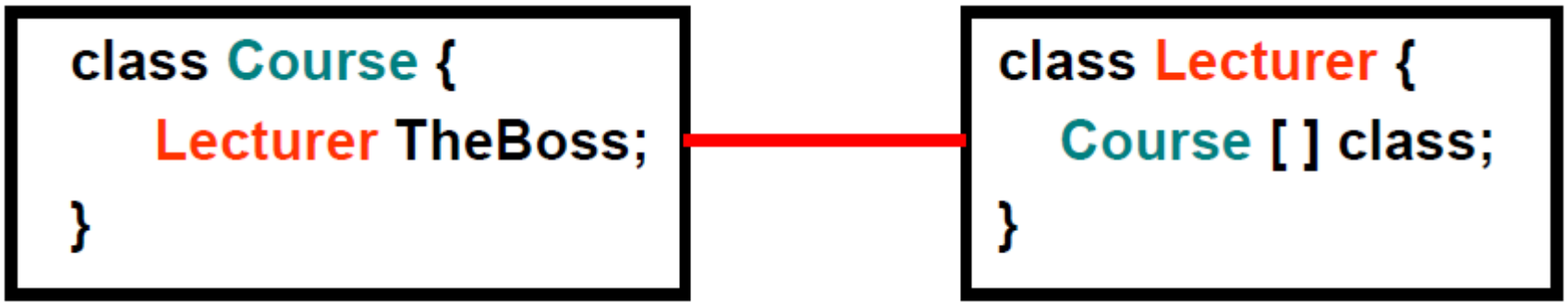
UML Relationships: Dependency

- Dependency may be caused by
  - Local variable
  - Parameter
  - Return value
- Example: Circle Area , PI



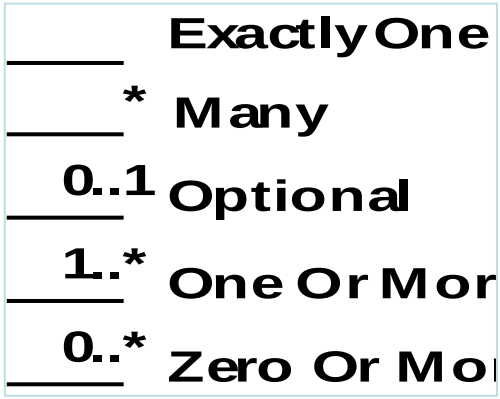
UML Relationships: Association

- Denotes interaction between two classes
- Example
  - Lecturer teaches course
    - Indicates relationship between Lecturer & Course



Navigability and Multiplicity

- A ***navigability*** arrow on an association shows which direction the association can be traversed or queried.
- The ***multiplicity*** of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers.

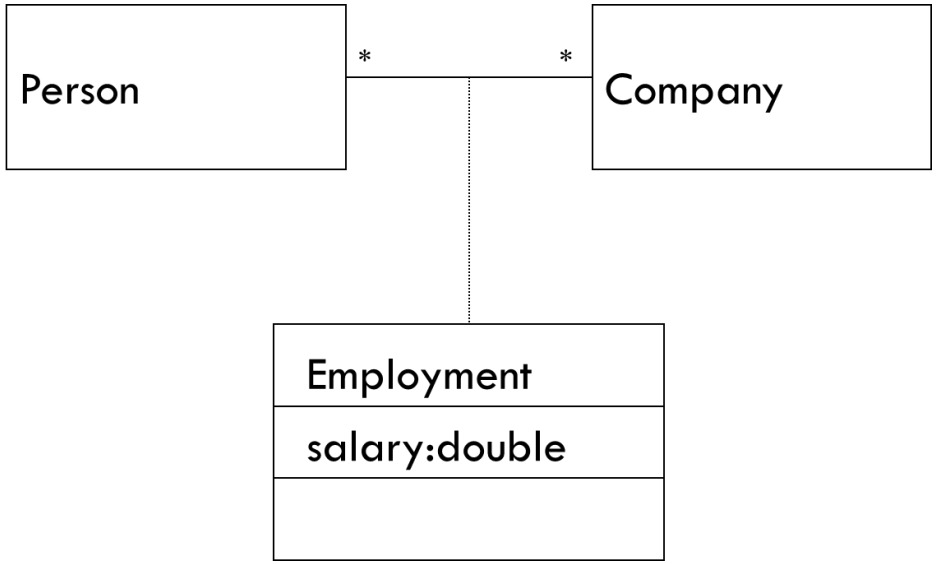


Ternary & High Order Associations

- Binary Association: Relates two classes
  - Woman is-wife-of Man
- Ternary Association : Relates three classes
  - Nancy is-daughter-of Susan and Robert
- *n*-ary Association : Relates *n* classes
- Higher Order Associations
  - complicated to draw, implement and think
  - try to avoid if possible

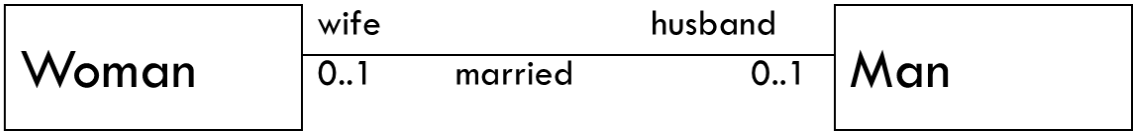
Link Attributes & Association Classes

- Attributes that belong to association of object rather than one object
  - Link Attributes belong in Association Classes
  - Ex: Salary received by Employee from Company
- In an one-to-one association you may try to make it attribute of one of the objects

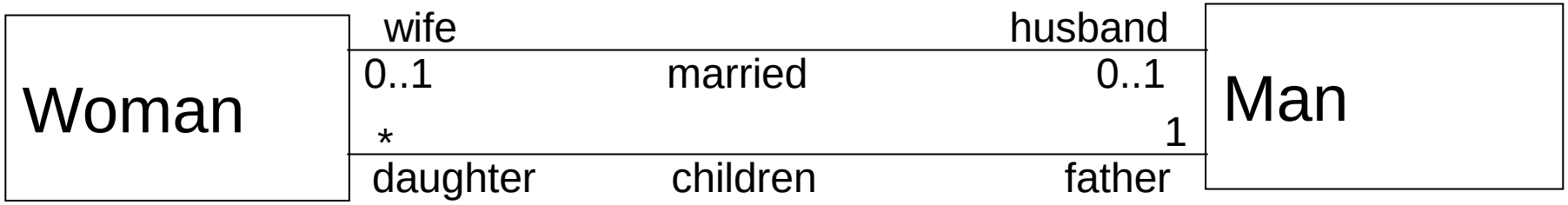


Role Names

- Name given to either end of an association
- Helps to navigate from one object to related objects



- Helps clarify when two classes have several associations between them



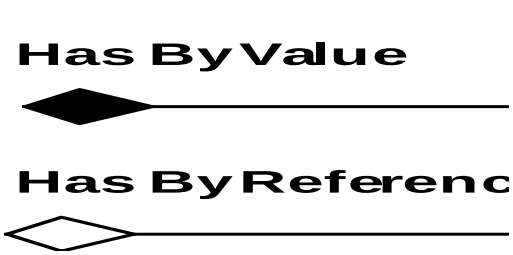
Aggregation Vs Composition

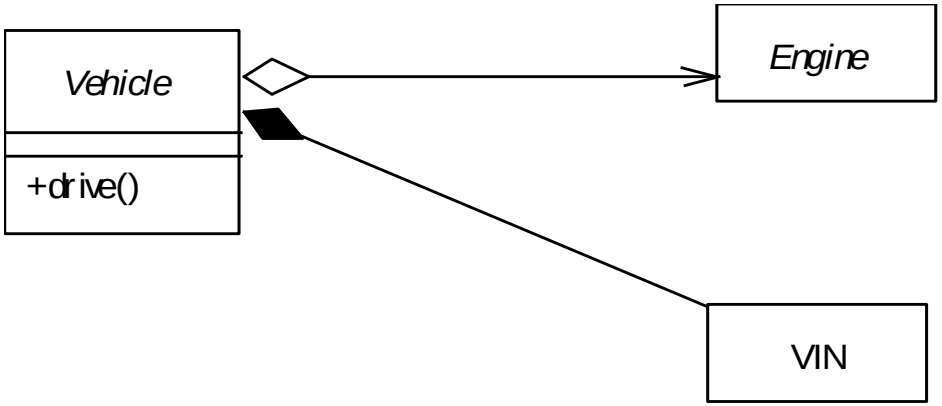
Aggregation

- A whole/part relationship (by reference)
- An aggregate object includes (has-a) other objects, each of which is considered to be a part of (part-of) the aggregate object.

Composition

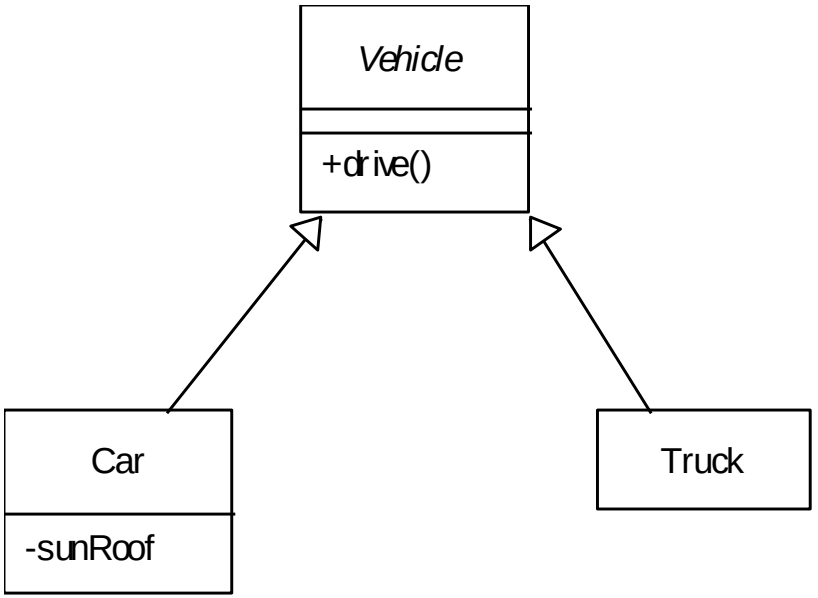
- A composition is a form of aggregation where the whole cannot exist without having the parts (by value)
- Part lives and dies with the whole





UML Relationships: Generalization

- Denotes inheritance between classes
  - Can view as “is a” relationship
- Example
  - Lecturer is a person (Lecturer extends Person class)



Abstract and Interface Classes

- Abstract Classes are represented by italicizing the name

Abstract class Shape

- Interfaces are prefaced with <<interface>>

Laptop implements DVDplayer interface

Class Diagram - Example

- Draw a class diagram for an information modeling system for a school.
  - School has one or more Departments.
  - Department offers one or more Subjects.
  - A particular subject will be offered by only one department.
  - Department has instructors and instructors can work for one or more departments.
  - Student can enroll in up to 5 subjects in a School.
  - Instructors can teach up to 3 subjects.
  - The same subject can be taught by different instructors.
  - Students can be enrolled in more than one school.

Class Diagram - Example

- School has one or more Departments.

- Department offers one or more Subjects.
- A particular subject will be offered by only one department.

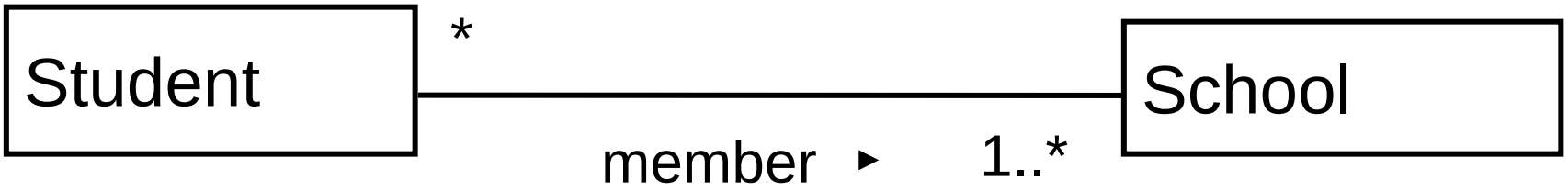
- Department has Instructors and instructors can work for one or more departments.

Class Diagram - Example

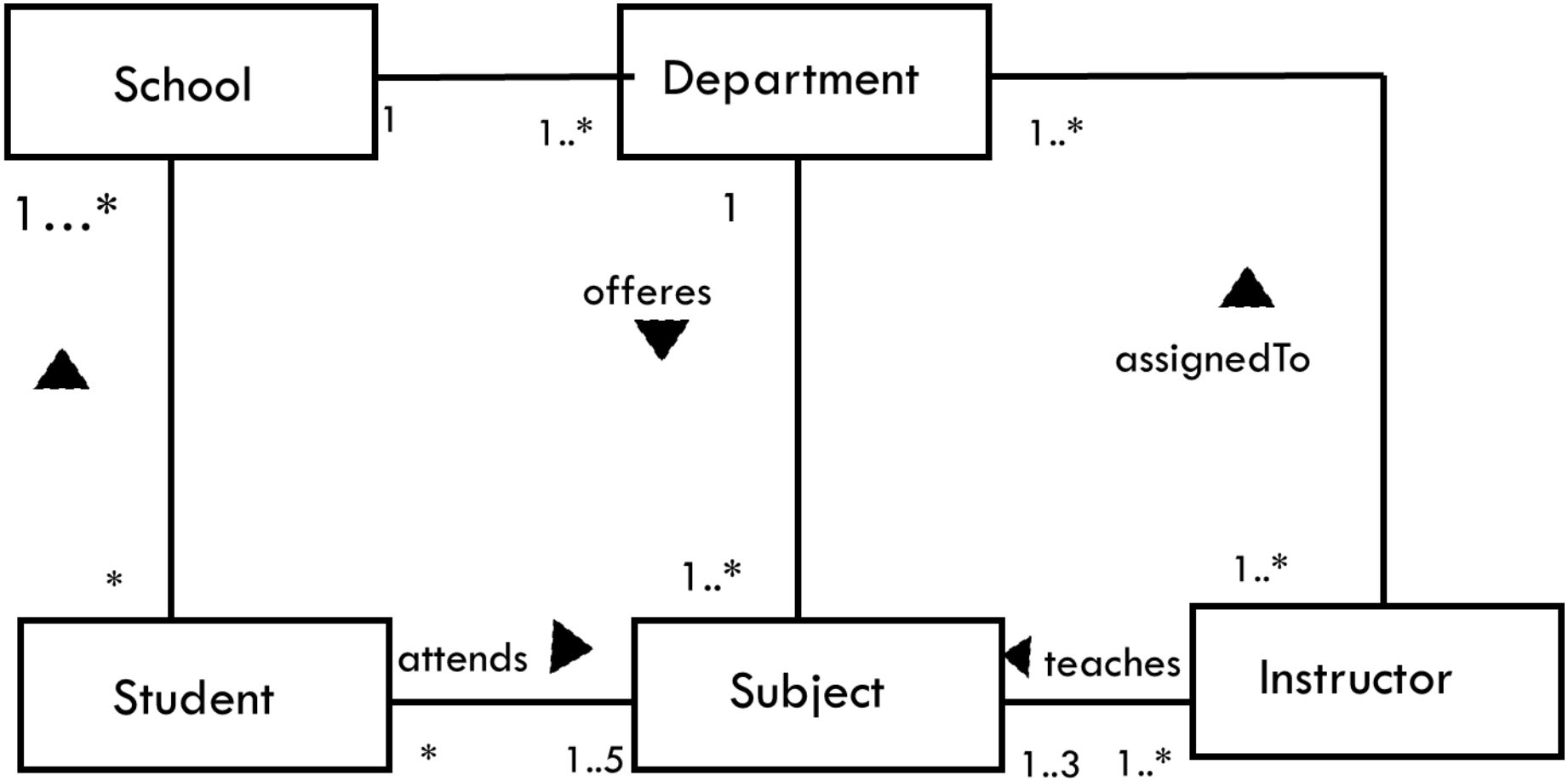
- Student can enroll in up to 5 Subjects.

- Instructors can teach up to 3 subjects.
- The same subject can be taught by different instructors.

- Students can be enrolled in more than one school.

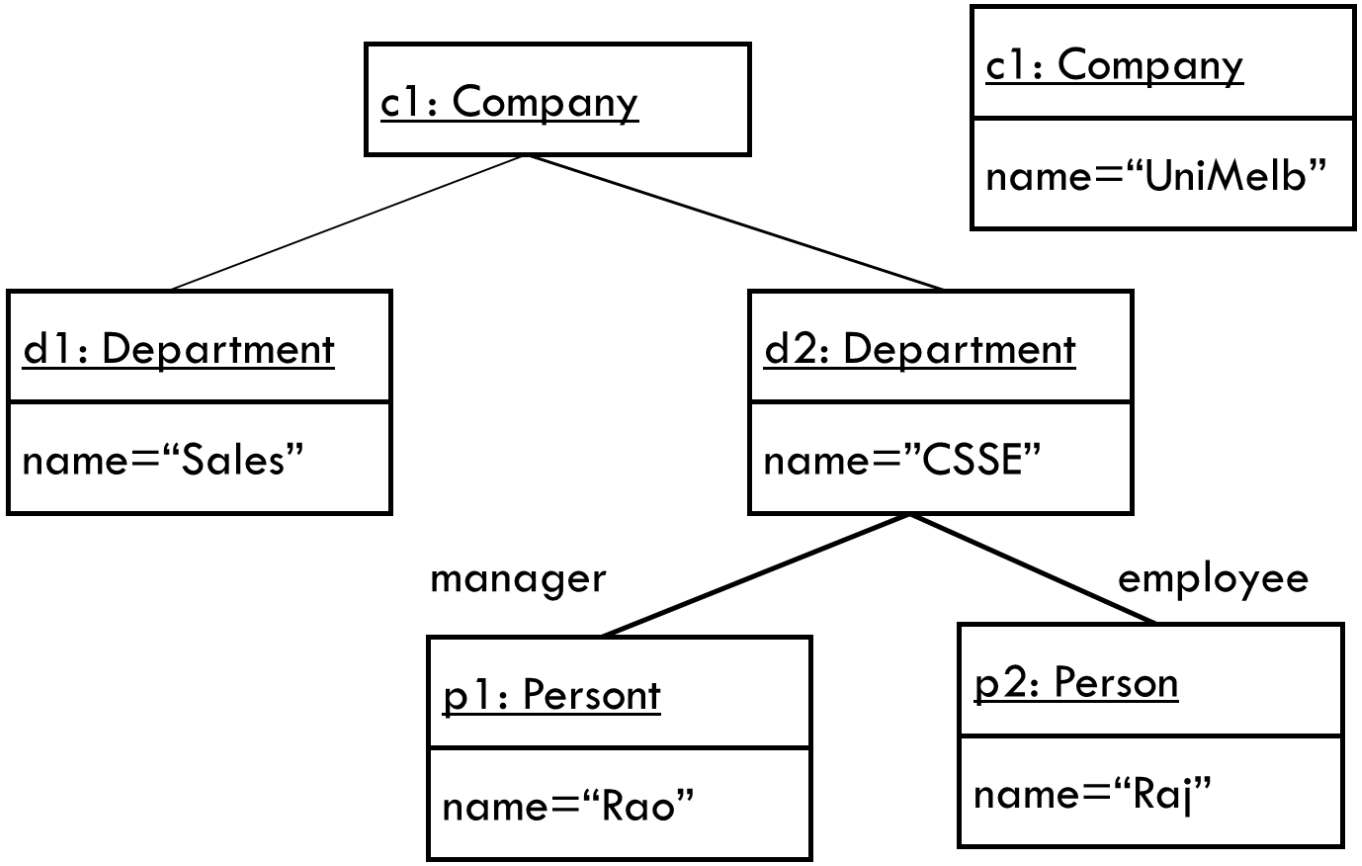


Class Diagram Example



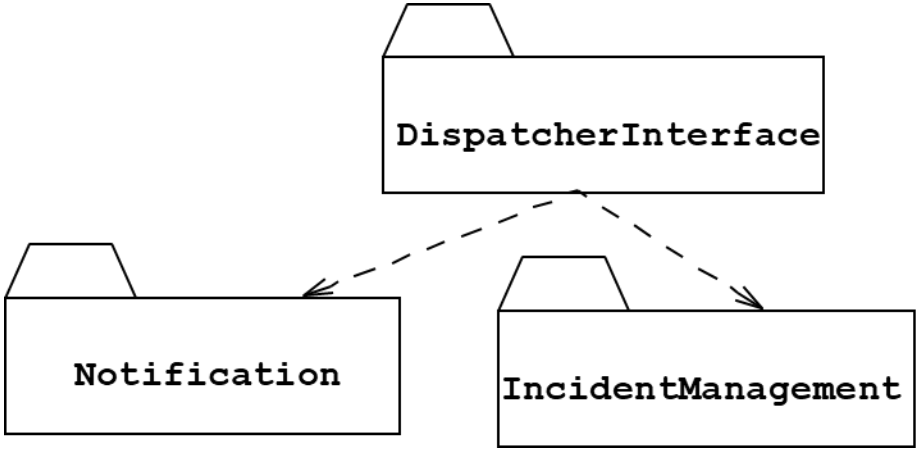
Object Diagram

- Object Diagram shows the relationship between objects.
- Unlike classes objects have a state.



Package Diagram

- A package is a UML mechanism for organizing elements into groups (usually not an application domain concept)
- Packages are the basic grouping construct with which you may organize UML models to increase their readability.
- A complex system can be decomposed into subsystems, where each subsystem is modeled as a package



Object Diagram (double check)

- Object Diagram shows the relationship between objects.