MAHARISHI UNIVERSITY of MANAGEMENT

Engaging the Managing Intelligence of Nature

Computer Science Department

CS401 MODERN PROGRAMMING PRACTICES (MPP)

PROFESSOR RENUKA MOHANRAJ

LECTURE 2: ASSOCIATIONS, MODELING RELATIONSHIPS WITH UML

Diversifying Self-Referral Relationship to the World of Objects

Wholeness Statement

In the real world, objects have relationships. These manifested relationships appear in many different ways. When these relationships are modeled in UML, those that reflect a permanent relationship are called *associations.*

At the most fundamental level every object is made out of the same essence – and is therefore (in a way) related to everything. An intellectual analysis or model of all these relationships is generally not practical.

A direct experience of the underlying reality of all of manifest creation and our relationship with all of nature is a result of our practice of Transcendental Meditation.

Overview

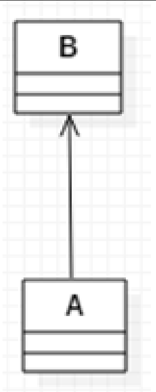
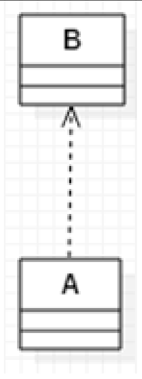
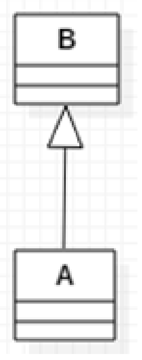
- Types of relationships between classes: association, dependency, inheritance
- Techniques for discovering associations
  - Identify verb phrases
  - Create an association matrix
- Association
  - Unidirectional and bidirectional
  - Aggregation
  - Composition
  - Reflexive
  - Association classes
  - Association "decorations": name, roles, multiplicities

Relationships Between Classes

In the OO paradigm, there are three fundamental types of relationships that can exist between classes

- Association
- Dependency
- Inheritance / Generalization (discussed in Lesson 3)

(continued)

		
Association from A to B	Dependency from A to B	A inherits from B

Associations

1. "Customer *has an* Account"
2. Permanent relationship
3. Association from A to B implies A keeps a reference to B
4. Association from A to B implies it is possible to navigate from A to B at runtime

Dependencies

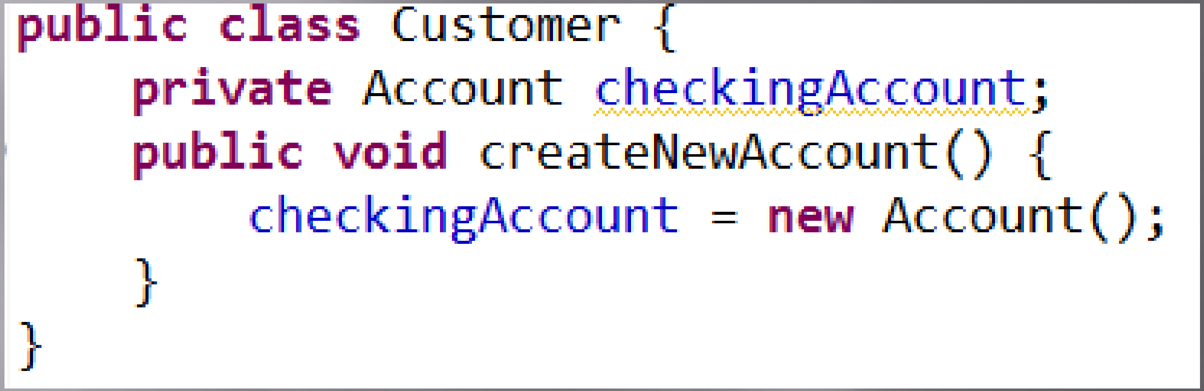
1. "Triangle *uses* Math" (see example)
2. Temporary relationship
3. Dependency from A to B implies A does *not* keep a reference to B

Dependency

- An Association always implies a *permanent* relationship, since an instance of the target class is stored in the source class.
- Sometimes, only a *temporary* relationship is needed – for instance, an instance of a target class may be needed in order to read or set some values inside a method call, but the relationship need not endure after the method returns.
- Temporary relationships are modeled as *dependencies.*
- When creating a class diagram, first identify associations. Later (during design), review your work to see if some of the associations really ought to be dependencies.

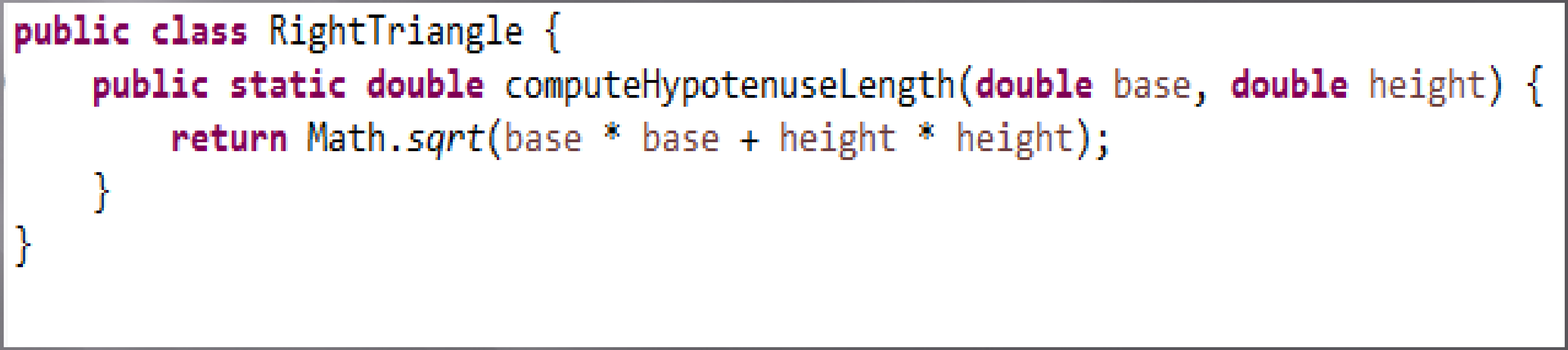
Two Examples

- Association



```java
public class Customer {
    private Account checkingAccount;
    public void createNewAccount() {
        checkingAccount = new Account();
    }
}
```

- Dependency



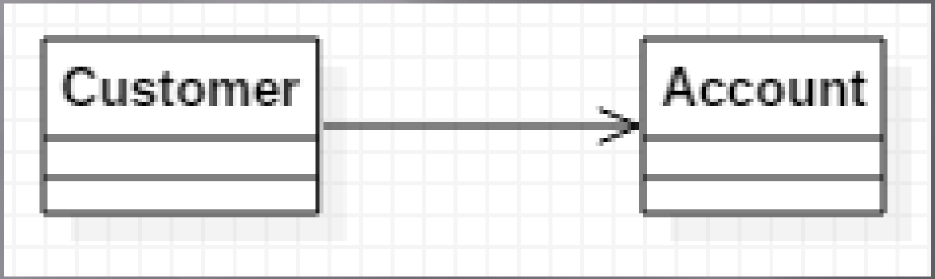
```java
public class RightTriangle {
    public static double computeHypotenuseLength(double base, double height) {
        return Math.sqrt(base * base + height * height);
    }
}
```

Dependency Example

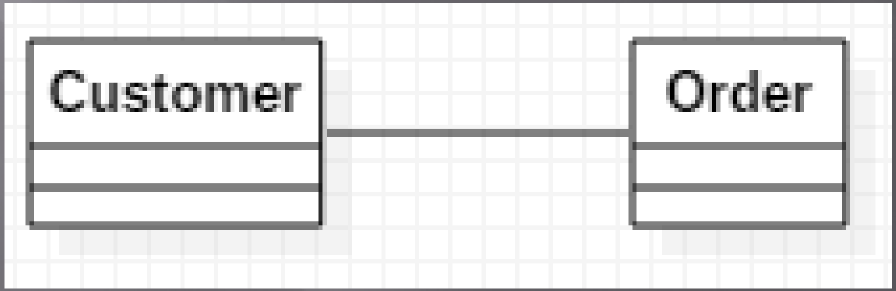
```
public class Email {
public void sendEmail(String subject, String message){
System.out.println("Subject : " + subject + " \n Message : "  + message);
}
}


public class Person {
public void greetFriend(Email ob){
ob.sendEmail("Hello", "Hello my friend :)");
 }
 }
```

One-way and Two-way Associations

- Sometimes should be able to navigate from A to B but not from B to A. This is a *one-way* or *uni-directional* association

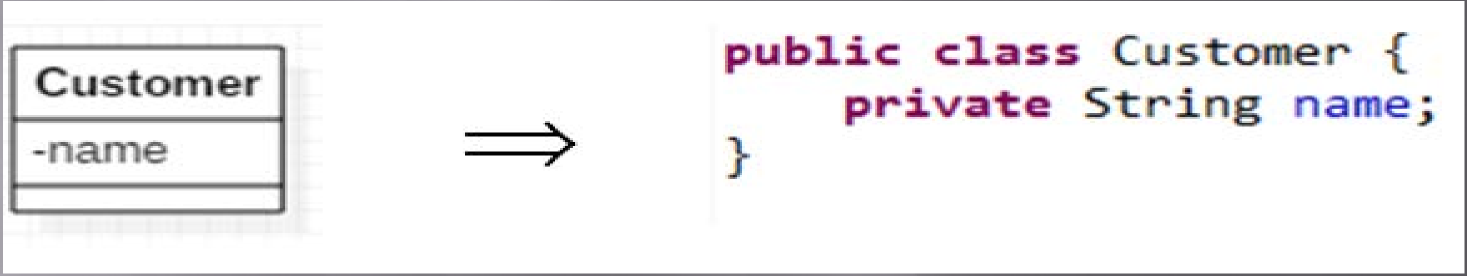


- Sometimes should be able to navigate from A to B and also from B to A. This is a *two-way* or *bi-directional* association



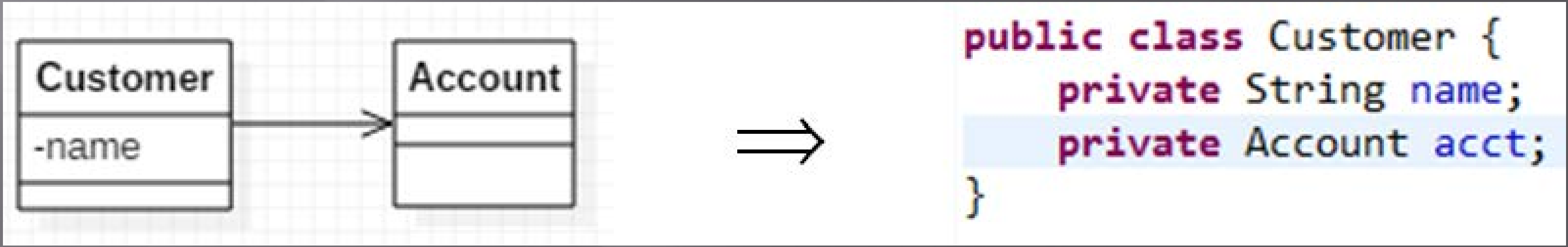
Properties as Attributes or Associations

- An *attribute* of a UML class indicates a variable that stores data, like `name` in `Customer`. A UML attribute is implemented in Java code as a *field* or *instance variable.*



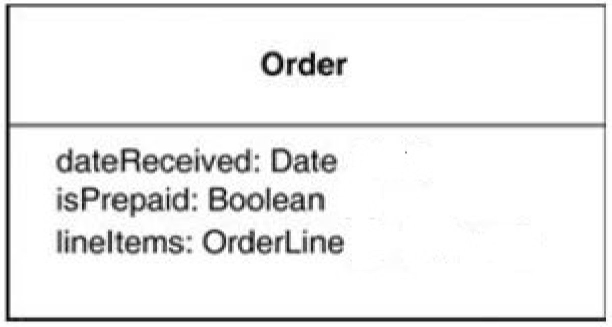
```
public class Customer {
        private String name;
}
```

- An association from one class to another is also implemented in Java code as a field or instance variable, like the association from `Customer` to `Account`

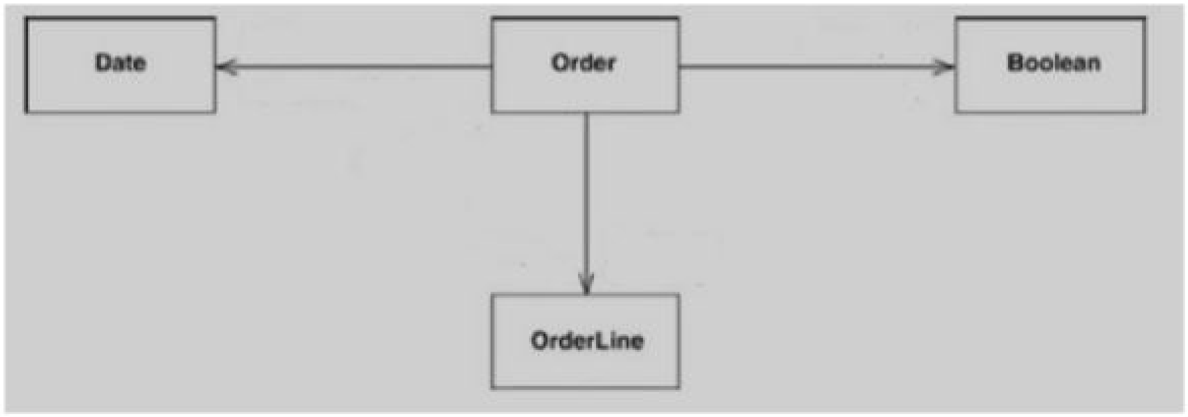


```
public class Customer {
      private String name;
      private Account acct;
}
```

- In the language of UML, both `acct` and `name` are called *properties* of `Customer`. The property `acct` is modeled as an association; the property `name` is modeled as an attribute.
- In UML, it is always possible to model properties either as attributes or as associations:



Properties of Order as *attributes*

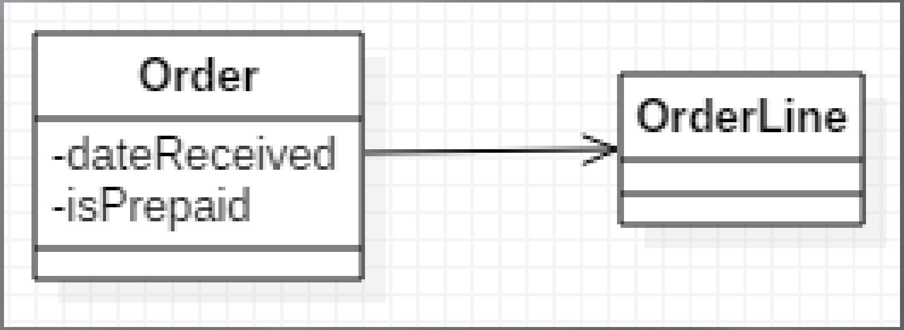


Properties of Order as *associations*

Which way is best – properties as *attributes* or *associations*?

- When a property has an internal structure, with its own properties, model it as an association; otherwise, as an attribute.

<u>Example</u>: In the previous example, an `Order` has many `OrderLines`, and each `OrderLine` has its own internal structure (cost, quantity, etc). So the `OrderLine` property should be modeled as an association; the others, as attributes



IMPORTANT: When a property is modeled as an association, it is not mentioned as one of the attributes of the class. Here, `Order` displays two attributes, one association. All three would appear as *fields* in a Java class implementation

Overview

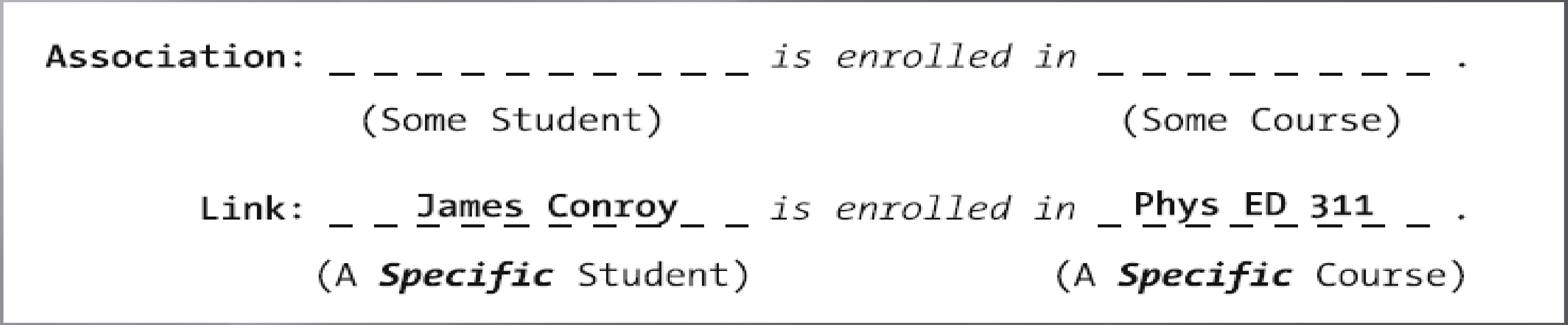
- Types of relationships between classes: association, dependency, inheritance
- **Techniques for discovering associations**
  - **Identify verb phrases**
  - **Create an association matrix**
- Types of association
  - Unidirectional and bidirectional
  - Aggregation
  - Composition
  - Reflexive
  - Association classes
  - Dependency
  - Association “decorations”: name, roles, multiplicities

Associations Specified by Verbs

Examples:

- Customer *has* an Account
- Professor *advises* a Student
- Student *enrolls in* a Section

An association is a template for creating links.



```
Association: _ _ _ _ _ _ _ _ _ _ is enrolled in _ _ _ _ _ _ _ _ .
               (Some Student)                  (Some Course)

       Link: _ _ James Conroy_ _ is enrolled in _Phys ED 311 _ .
             (A Specific Student)            (A Specific Course)
```

<u>*Strategy*</u>:

- Discover associations by finding verbs and verb phrases in the problem statement.
- Track the relationships in an *Association Matrix*

The Student Registration System

We have been asked to develop an automated Student Registration System (SRS) for the university. This system will enable students to register online for courses each semester, as well as track their progress toward completion of their degree.

When a student first enrolls at the university, he/she uses the SRS to create a plan of study that lists the courses he/she plans on taking to satisfy a particular degree program, and chooses a faculty advisor. The SRS will verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking.

Once a plan of study has been established, then, during the registration period preceding each semester, students are able to view the schedule of classes online, and choose whichever classes they wish to attend, indicating the preferred section (day of the week and time of day) if the class is offered by more than one professor. The SRS will verify whether or not the student has satisfied the necessary prerequisites for each requested course by referring to the student's online transcript of courses completed and grades received (the student may review his/her transcript online at any time).

Assuming that (a) the prerequisites for the requested course(s) are satisfied, (b) the course(s) meet(s) one of the student's plan of study requirements, and (c) there is room available in each of the class(es), the student is enrolled in the class(es).

If (a) and (b) are satisfied, but (c) is not, the student is placed on a first-come, first-served wait list. If a class/section that he/she was previously waitlisted for becomes available (either because some other student has dropped the class or because the seating capacity for the class has been increased), the student is automatically enrolled in the waitlisted class, and an email message to that effect is sent to the student. It is the student's responsibility to drop the class if it is no longer desired; otherwise, he/she will be billed for the course.

Students may drop a class up to the end of the first week of the semester in which the class is being taught.

HIGHLIGHTING VERB PHRASES IN THE SRS SPECIFICATION

We have been asked to develop an automated Student Registration System (SRS) for the university. This system will **enable students to register** online **for courses** each semester, as well as **track their progress toward completion of their degree**.

When a student first **enrolls at the university**, he/she uses the SRS to **set forth a plan of study** as to which **courses he/she plans on taking** to **satisfy a particular degree program**, and **chooses a faculty advisor**. The SRS will **verify whether or not the proposed plan of study satisfies the requirements of the degree that the student is seeking**.

Once a **plan of study has been established**, then, during the registration period preceding each semester, a student is able to **view the schedule of classes** online, and **choose whichever classes he/she wishes to attend, indicating the preferred section** (day of the week and time of day) if the **class is offered by more than one professor**. The SRS will **verify whether or not the student has satisfied the necessary prerequisites** for each requested course by **referring to the student's online transcript** of courses completed and grades received (the **student may review his/her transcript** online at any time).

Assuming that (a) the **prerequisites for the requested course(s) are satisfied**, (b) the **course(s) meet(s) one of the student's plan of study requirements**, and (c) **there is room available** in each of the class(es), the **student is enrolled in the class(es)**.

If (a) and (b) are satisfied, but (c) is not, the **student is placed on a first-come, first-served wait list**. If a **class/section that he/she was previously waitlisted for becomes available** (either because some other **student has dropped the class** or because the **seating capacity for the class has been increased**), the **student is automatically enrolled in the waitlisted class,** and an **email message** to that effect **is sent** to the student. It is the student's responsibility to **drop the class** if it is no longer desired; otherwise, **he/she will be billed for the course**.

Students may **drop a class** up to the end of the first week of the semester in which the **class is being taught**.

Association Matrix

	Section	Course	Plan of Study	Professor	Student	Transcript
Section				Is taught by		included in
Course						
Plan of Study						
Professor						
Student						
Transcript						

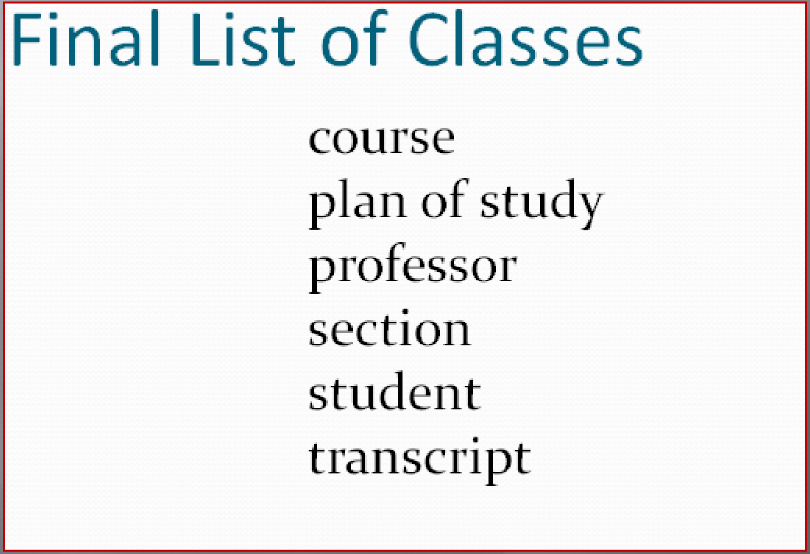
Associations– In class Exercise

In your small groups refer to our problem description and fill in the Association Matrix for the classes we have identified for the SRS system (handout)

	Section	Course	Plan of Study	Professor	Student	Transcript
Section		instance of		is taught by		included in
Course		prerequisite for	is called for by			
Plan of Study		calls for			observed by	
Professor	teaches				advises; teaches	
Student	registered for; waitlisted for; has previously taken	plans to take	observes	is advised by; studies under		owns
Transcript	Includes	Includes			belongs to	

Problem Description – In class Exercise

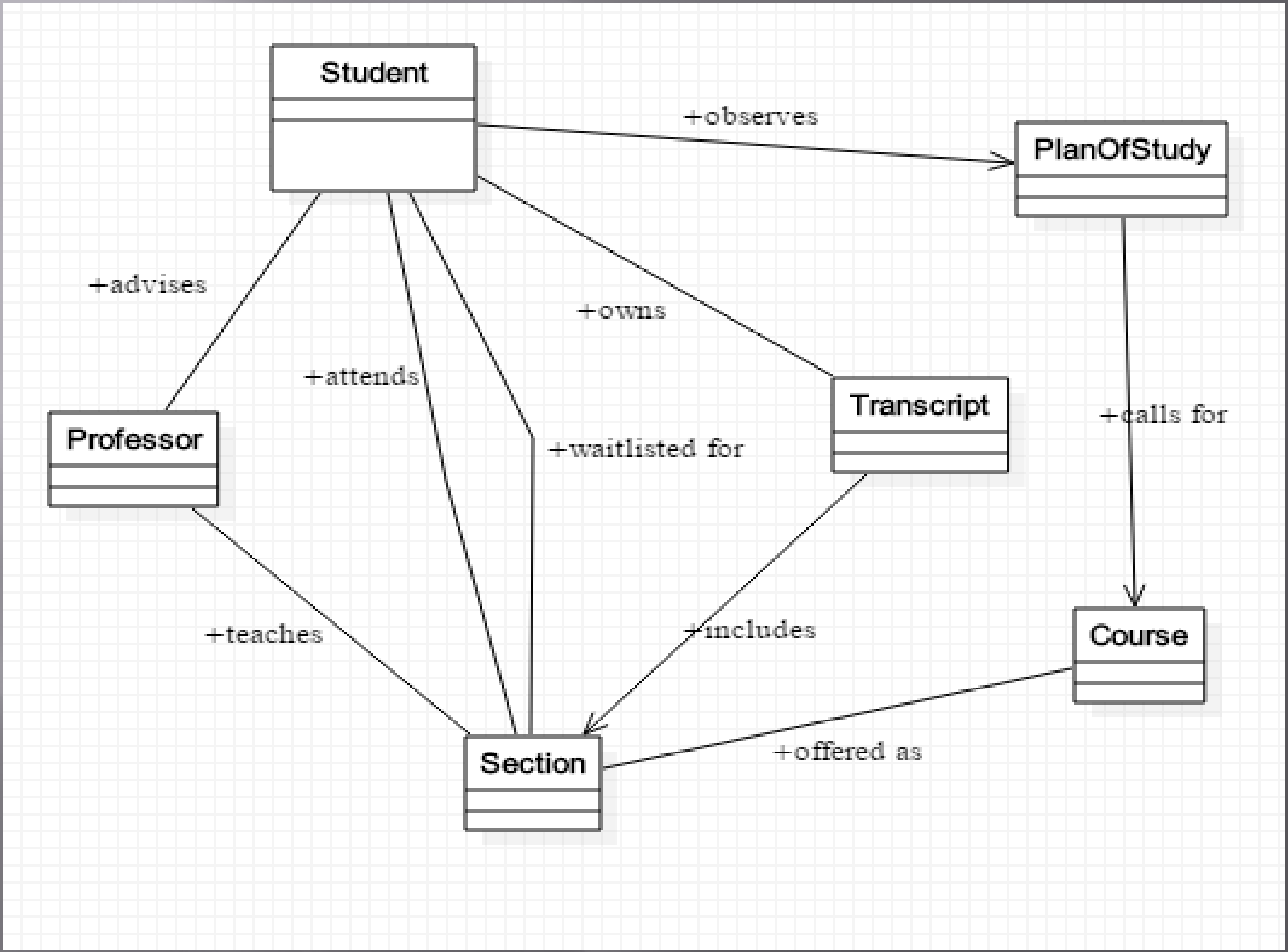
In your small group now try to create a diagram with all the classes and their labeled associations



Final List of Classes

course
plan of study
professor
section
student
transcript

Student Registration System



Main Point 1

Building a software system using OO principles involves an *analysis* step in which the problem is analyzed and broken into pieces as objects are discovered. The pieces are then refined and put together – in a step of *synthesis* – to give a picture of a unified system. This step of synthesis happens through the identification of relationships between classes, represented by *associations.*

This phenomenon is a characteristic of all knowledge – it arises through a combination of analysis and synthesis.

Quiz 1

1. Which one of the following indicates temporary relationship between the classes?

a. Inheritance

b. Association

c. Dependency

Overview

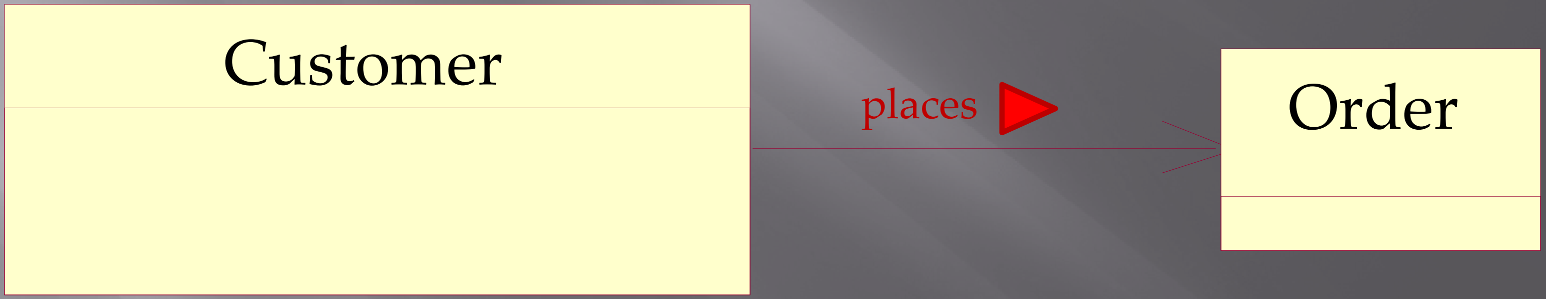
- Types of relationships between classes: association, dependency, inheritance
- Techniques for discovering associations
  - Identify verb phrases
  - Create an association matrix
- **Association**
  - **Unidirectional and bidirectional**
  - **Aggregation**
  - **Composition**
  - **Reflexive**
  - **Association classes**
  - **Dependency**
  - **Association "decorations": name, roles, multiplicities**

Describing Relationships

- Relationship between objects is referred as association.
- Relationships define how objects interact:
  - Student *takes* Sections
  - Professor *teaches* Sections
  - Client *pays* Vendor
  - Customer *places* Order
- Relationships are often two-way streets:
  - Student *takes* Section
  - Section *includes* Student

Association

- Unidirectional
  - Objects of a class have a reference to an object of another class.
  - First class keeps a reference to the second
  - Often has a name that describes the association, often with a direction indicator [some UML tools do not support direction indicators]

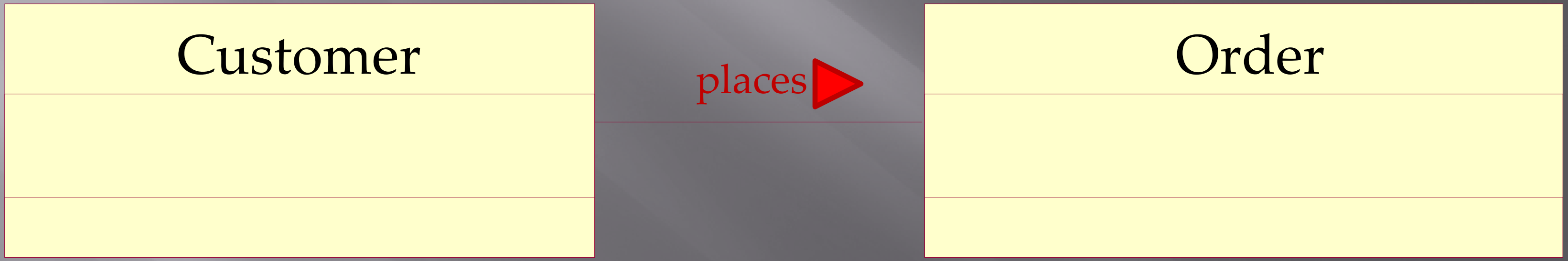


```
public class Customer {
    private Order order;
}
```

```
public class Order {

}
```

Association

- Bi-directional
  - Description direction is more important (still optional)



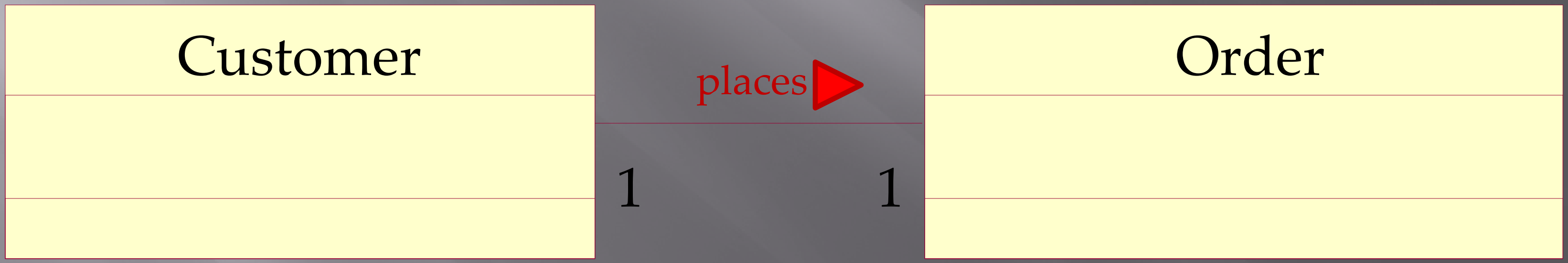
```
public class Customer {
    private Order order;
}
```

```
public class Order {
    private Customer customer;
}
```

Association: 1-1 Multiplicity

A 1-1 relationship between Customer and Order implies:

1. Each Customer has only ONE Order
2. A Customer cannot be created without associating an Order with that Customer



```
public class Customer {
    private Order order;
}
```

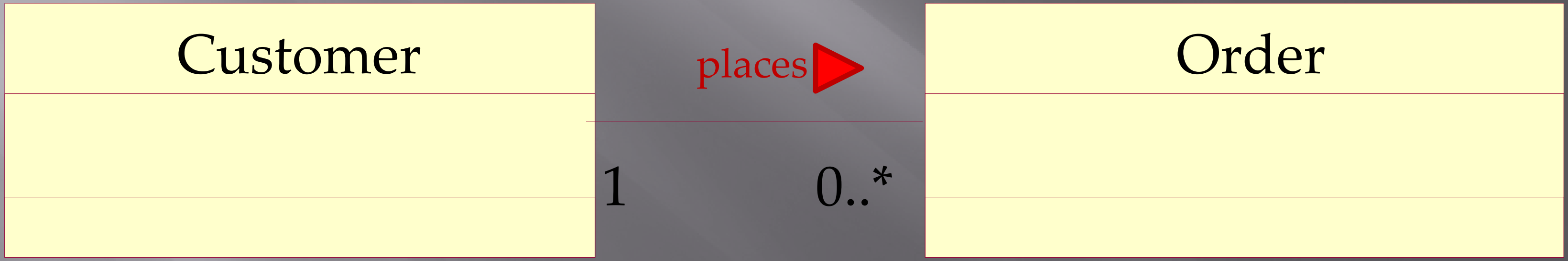
```
public class Order {
    private Customer customer;
}
```

This assignment of mulitplicities is not realistic…

Association: Other Multiplicities

- UML supports a variety of multiplicities

1	one (mandatory)
3	three (exactly)
*	many
0..*	zero or more (optional)
1..*	one or more
0..1	zero or one (optional)



```
public class Customer {
    private List<Order> orders;
}
```

```
public class Order {
    private Customer customer;
}
```

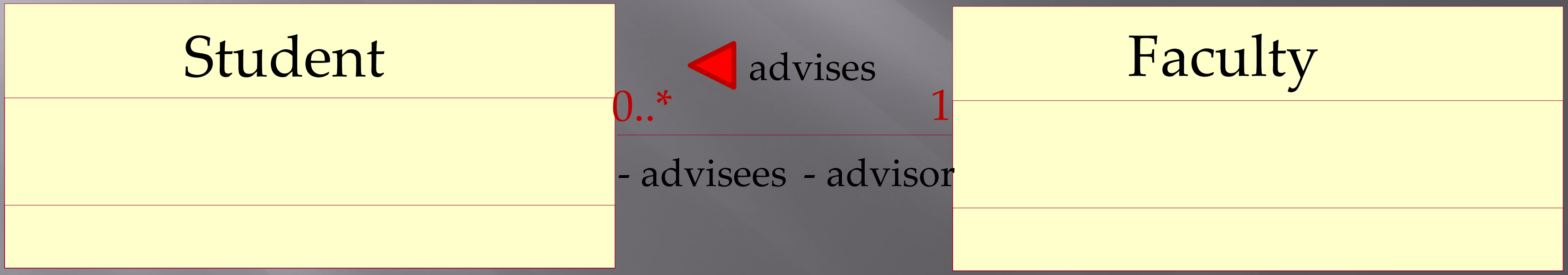
In this diagram, a Customer may have zero or more Orders. This is more realistic

Determining Multiplicity

- Ask: For a given instance of a class A and association involving A, B, how many instances of B must/may be associated with this instance of A?
- Optionality
  - Is the association required? If not, association will be "zero or more"; otherwise, it will be "one or more"
- Cardinality
  - How many instances are associated with a given instance? Could be 1:1, 1:2, 1:3, 1..* (for example)
- UML combines both ideas in the Multiplicity concept.

Association roles

- A role is a (noun) description placed on either side of the association to indicate the role(s) each object plays in the relationship. Roles are *optional.*

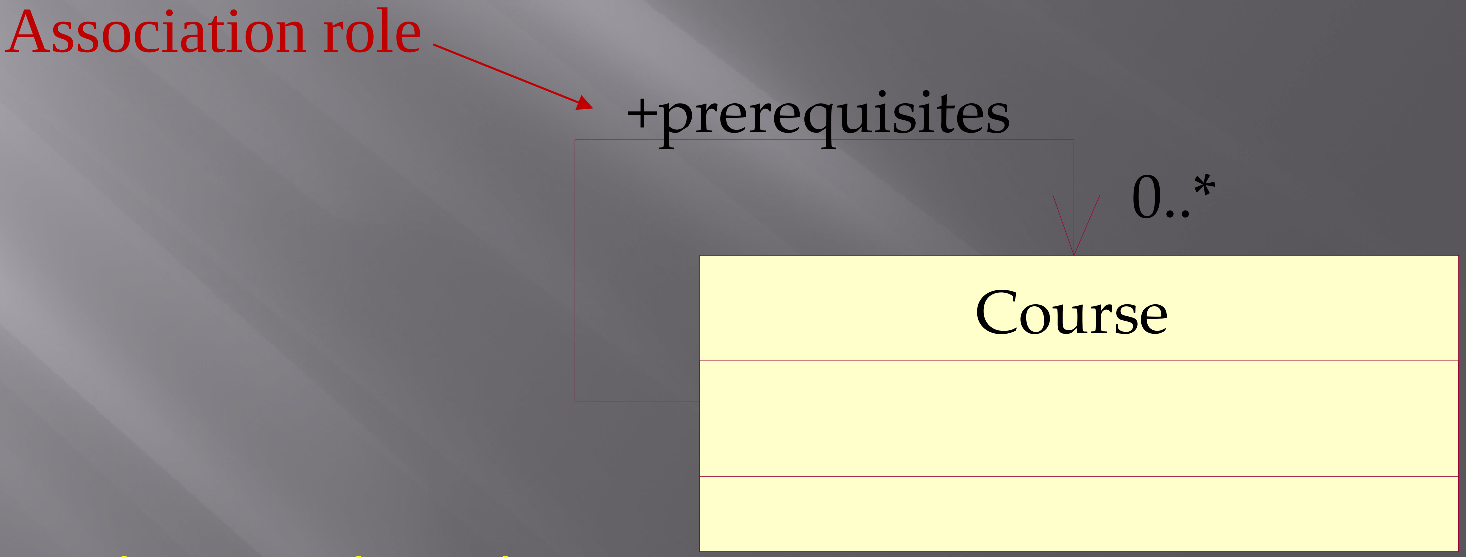


```java
public class Student {
    private Faculty advisor;
}
```

```java
public class Faculty {
    private List<Student> advisees;
}
```

Reflexive Association

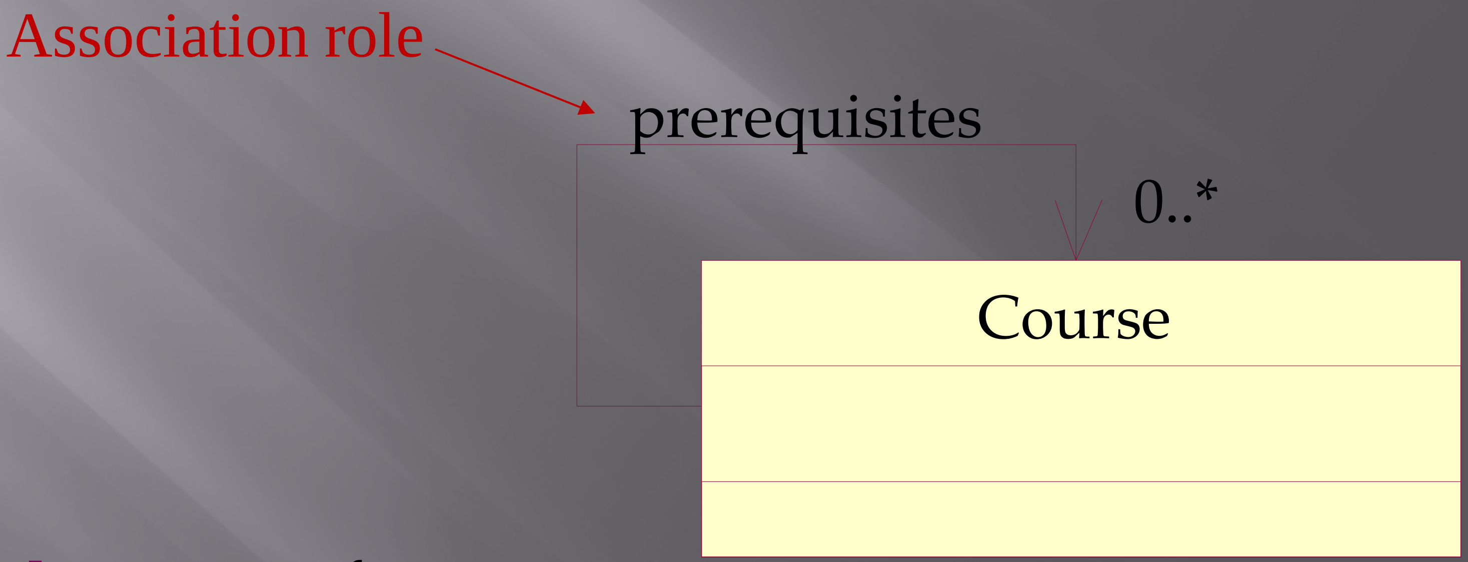
- Relationship between two or more objects of the same class.



How can this relationship be expressed in code?

Reflexive Association

- Relationship between two or more objects of the same class.



```
public class Course {
    private List<Course> prerequisites;
}
```

Main Point 2

Associations model the relationships that can exist between concepts. Simple (one-way) associations are modeled using an *arrow*; two-way associations are modeled using a *line segment*.

The association can have a *name* for ease of reading, and additional symbols to indicate direction and multiplicity.

The ends of an association arrow can also specify association roles.

The “simplest” association is the relationship of pure consciousness to itself; this can also be modeled with an arrow from pure consciousness to itself.

Quiz 2

Draw the association for the given problem with role and multiplicity.

1. "A Professor *works for* exactly one Department, but a Department *has* many (one or more) Professors as employees."

2. "A Student attends many Sections, and a Section is attended by many Students."

In-class Exercise

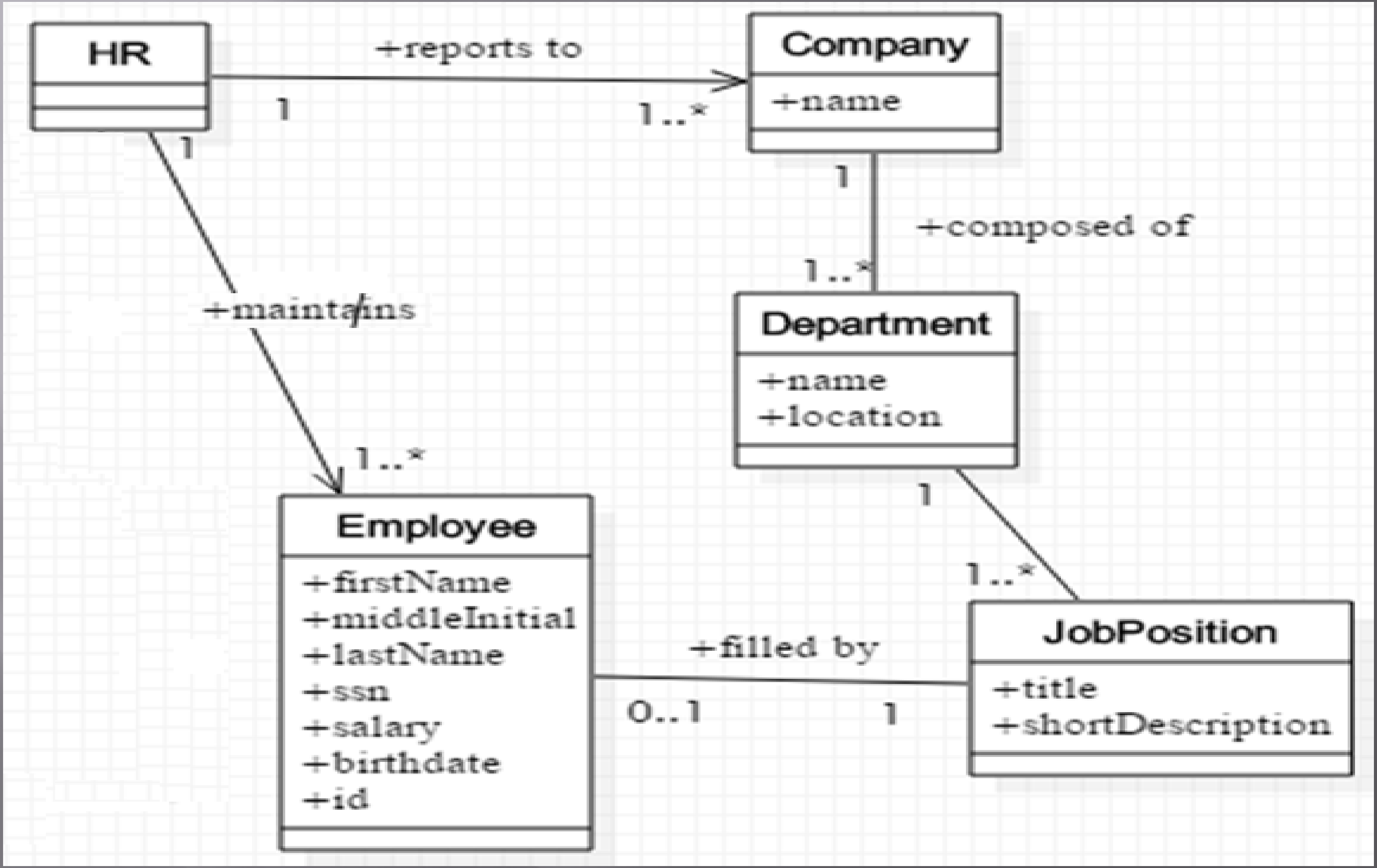
For the following problem statement:

- Work independently for 10 minutes to create the class diagram. Display the associations that are needed.
- Then review your diagrams for 10 minutes with your small group.

Association Exercise

A Human Resource (HR) department keeps track of employees for their several companies. Each company has a name and may consist of many departments. A department has a name and a location. Each department has one or more job positions. A position has a title and a short description. A position may be vacant. Otherwise, an employee is assigned to it. The HR personnel enter an employee's first name, middle initial, last name, birth date, Social Security Number (SSN), unique employee id and a salary.

Solution



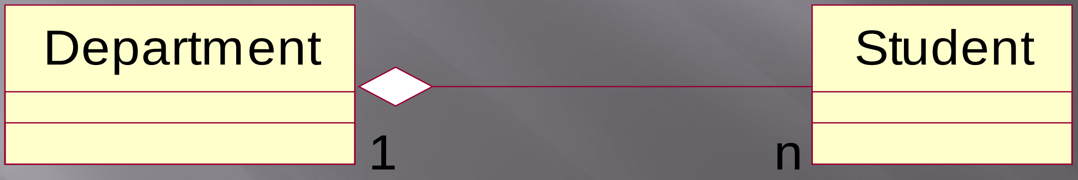
HR
+reports to
1
1..*
Company
+name
1
+composed of
1..*
Department
+name
+location
1
+maintains
1..*
Employee
+firstName
+middleInitial
+lastName
+ssn
+salary
+birthdate
+id
+filled by
0..1
1
1..*
JobPosition
+title
+shortDescription

Aggregation

- Represents a 'whole-part' relationship
  - 'contain'
  - 'is part of'

← Association name is implied

- Code looks the same as an association.

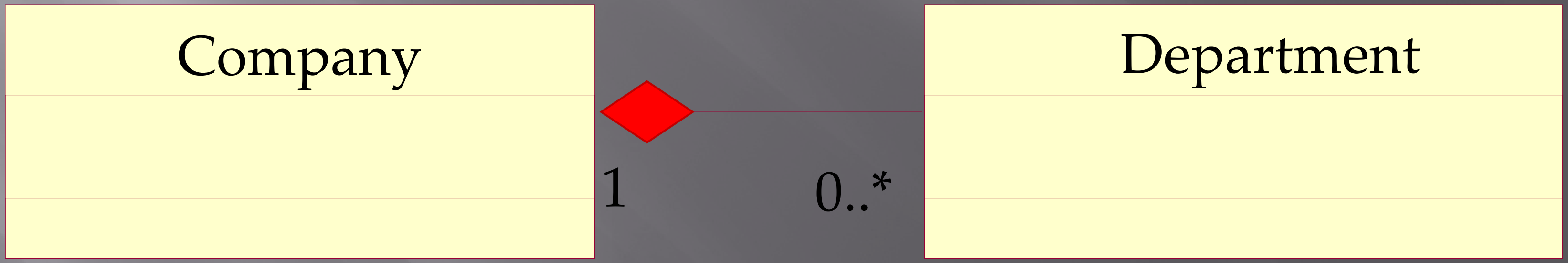


```
public class Department {
    private List<Student> students;
}
```

```
public class Student {
    private Department department;
}
```

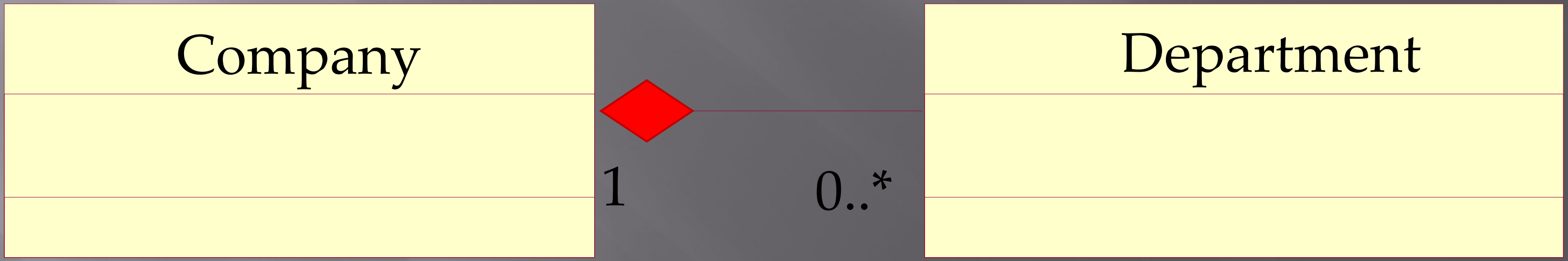
Composition

- Strong aggregation
  - 'whole-part' relationship – "is composed of"
  - If whole dies, parts also die.



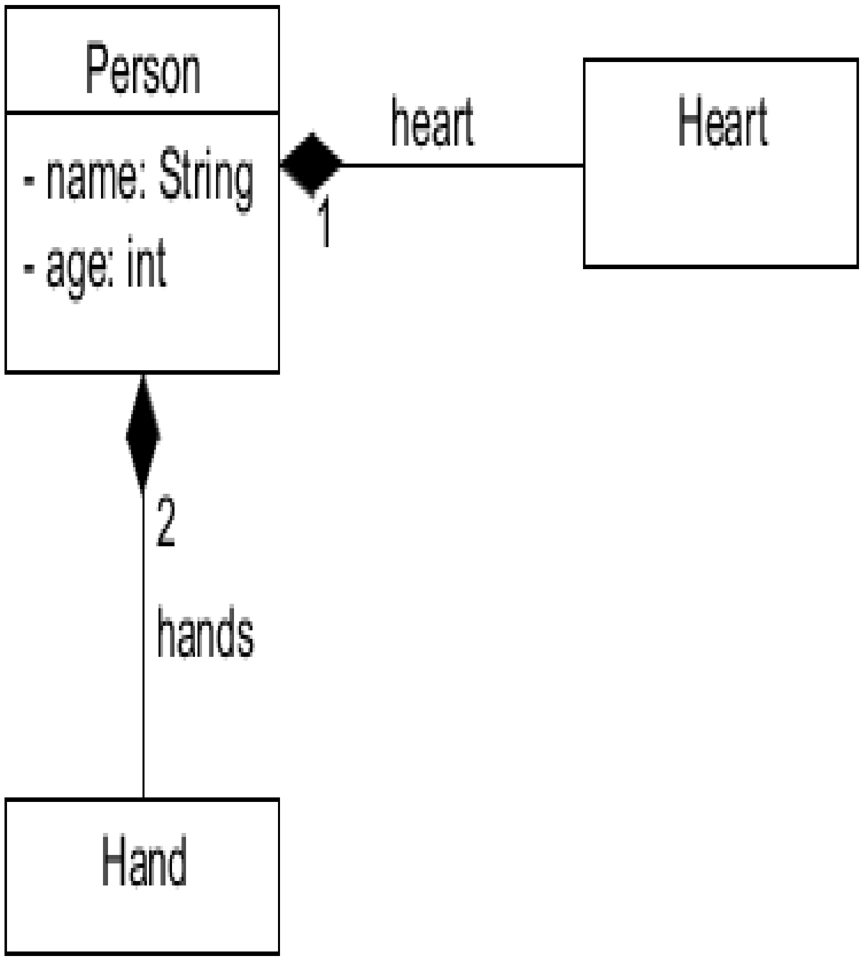
Composition

How to implement in code? There is no way (using Java) to ensure the composition relationship without using additional classes (beyond the two classes involved in the relationship). Inner class also useful to implement Composition.

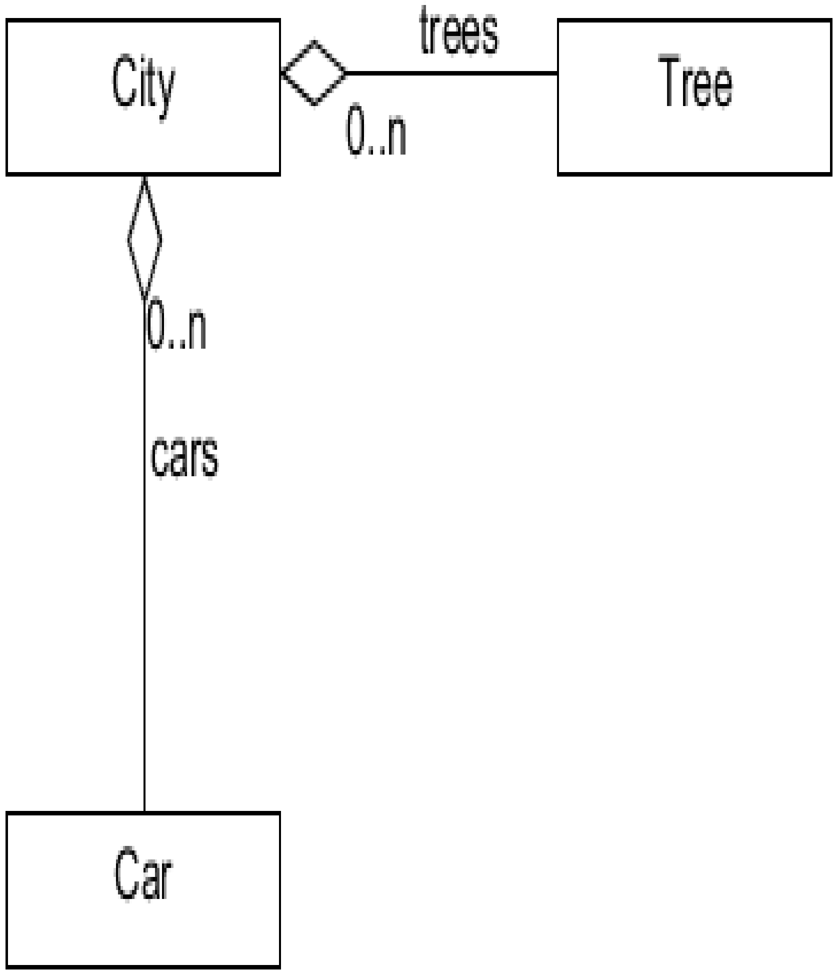


Composition and Aggregation

Composition

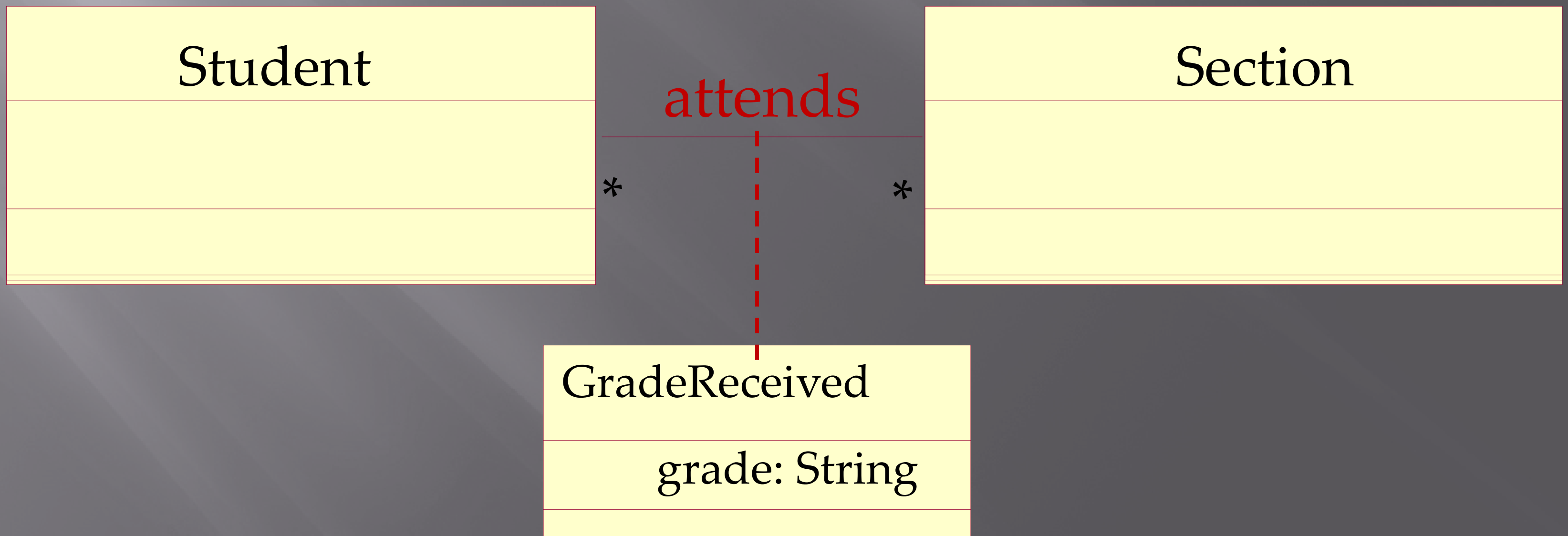


Aggregation



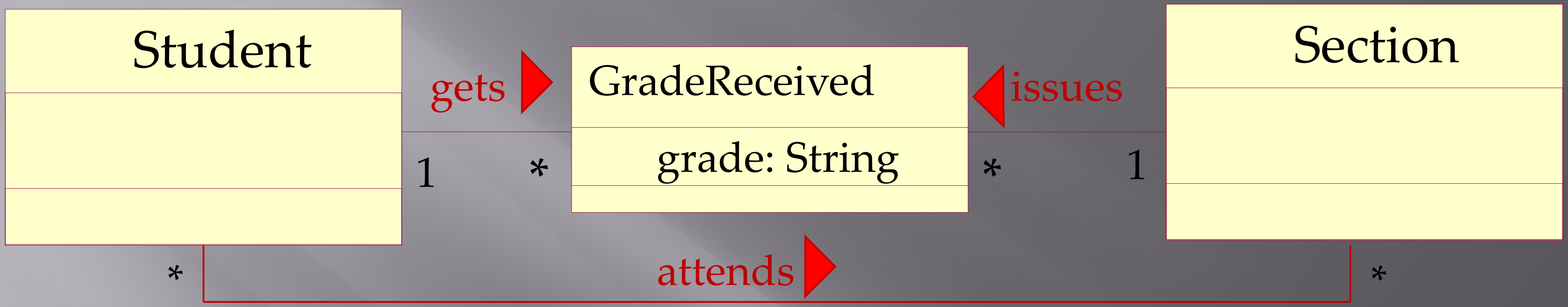
Association Classes

- We sometimes find ourselves in a situation where we identify an attribute that is critical to our model, but which doesn't seem to fit nicely into any one class.
- As an example, let's revisit the (many-to-many) association "a Student *attends* a Section,"
- Association Classes are useful to contain attributes of the link between objects.
- These are often modeled this way during analysis



Association Class - reworked

- During design, association classes are often re-worked to reflect the code instead of the concepts



```
public class Student {
List<Section> sectionsEnrolledIn;
List<GradeReceived> evaluationsReceived;
}
```

```
public class Section {
int sectionNumber;
List<Student> enrolledStudents;
List<GradeReceived> gradeSheets;

}
```

```
public class GradeReceived{
Student student;
Section section;
String grade;
}
```

Main Point 3

- There are several special forms of association, such as reflexive associations, aggregation, composition, and association classes, and dependency is a further refinement
- Although most of these have their own symbols, you could still model these relationships without them.
- The use of the symbols is to (easily) communicate additional information about the relationship.
- Even these additional symbols are still based on the simple concept of an *arrow*. This is an example of diversity on the basis of unity.

Exercise

- Objective: understand reflexive associations
- Task: Draw a UML class diagram for each of the following problem statements.

1. Doubly-Linked List:

   A LinkedList consists of zero or more ListItems. However, the LinkedList class only knows about the first ListItem. Each ListItem knows its previous and its next ListItem, if any.

2. Position Hierarchy:

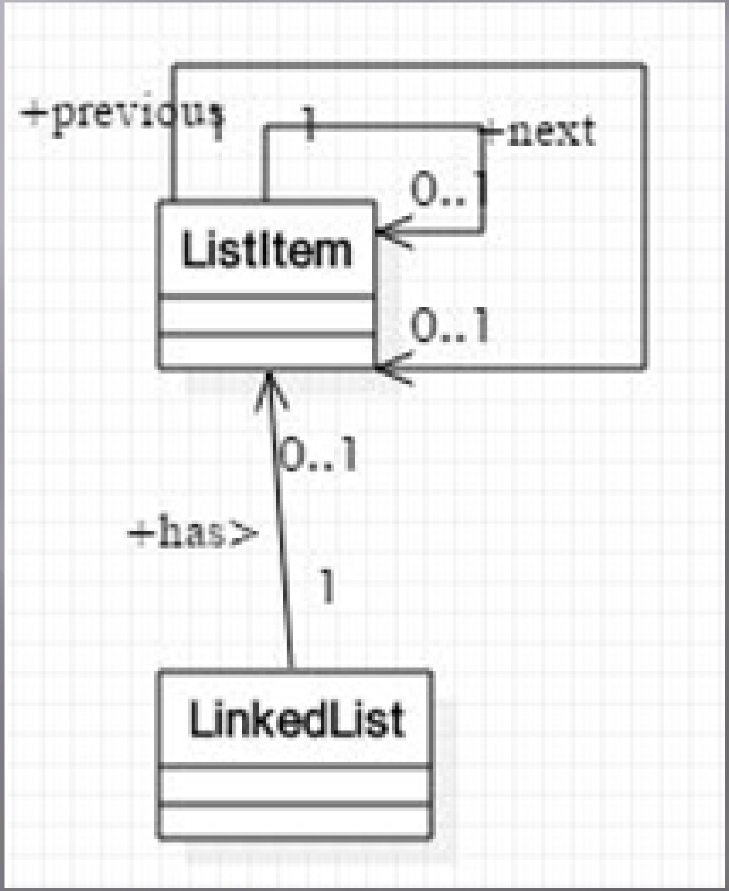
   A position may or may not be a managerial position. If it is a managerial position, then other positions report to this one.

3. Course Prerequisites:

   A university course may or may not require the student to have taken other courses first before taking this one. Any course can be a prerequisite course for any other course, except for itself.

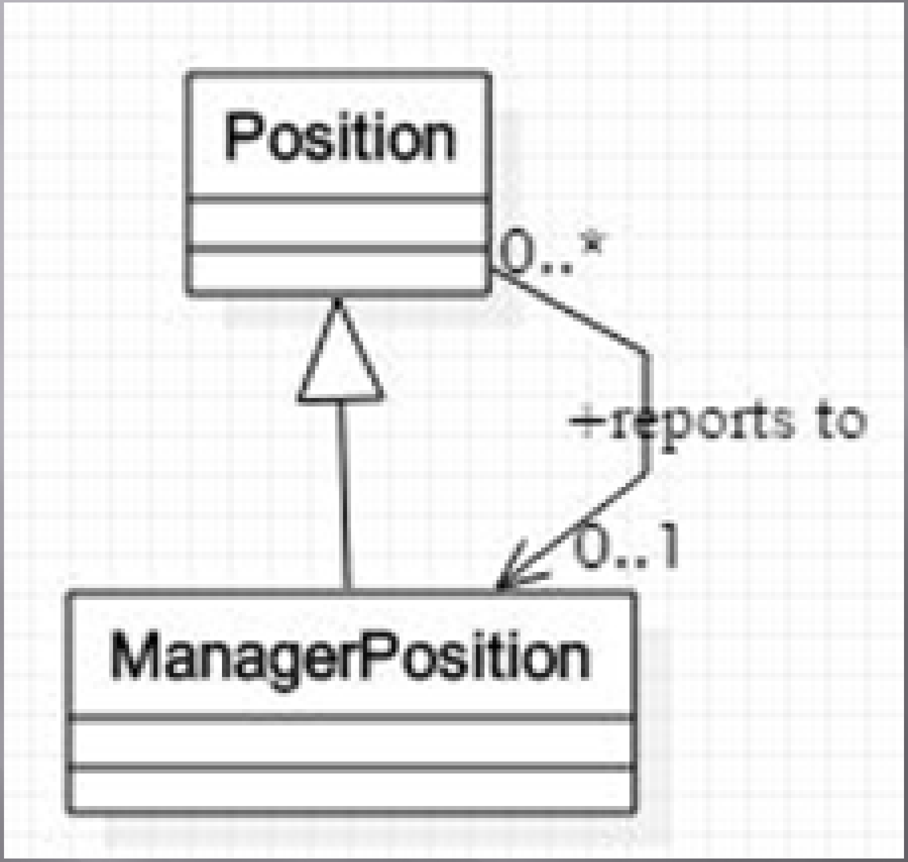
Solutions

Doubly Linked List



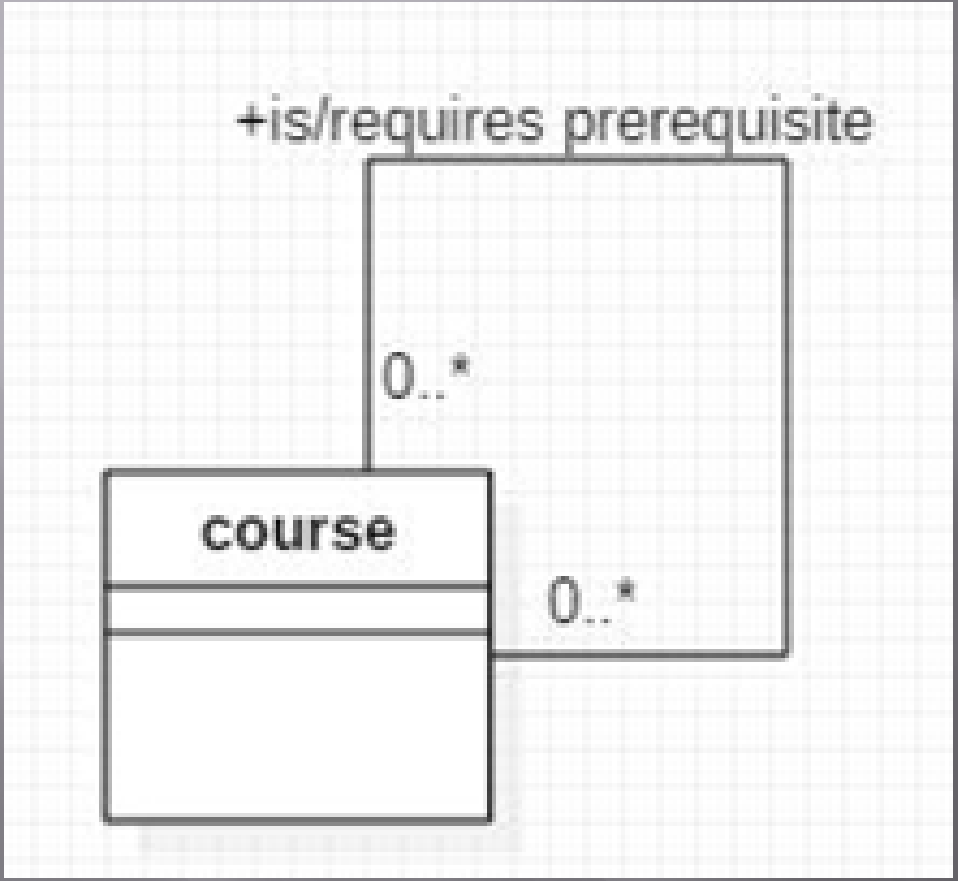
A LinkedList consists of zero or more ListItems. However, the LinkedList class only knows about the first ListItem. Each ListItem knows its previous
and its next ListItem, if any.

Position Hierarchy



A position may or may not be a managerial position. If it is a managerial position, then other positions report to this one.

Pre-requisites



A university course may or may not require the student to have taken other courses first before taking this one. Any course can be a prerequisite course for any other course, except for itself.

Summary

Modeling associations:

- We can use an association matrix to analyze what the relationships are between classes
- Associations are modeled with a line or arrow, and, optionally, a name describing the association, numbers on each side to indicate multiplicity, roles at either end of the association, and an arrow for directionality.
- Reflexive associations, aggregation, composition, association classes, and dependencies are further refinements of the concept of an association.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

1. Class diagrams are defined in terms of classes and their relationships (associations)

2. Although there are various special association forms (composition, aggregation, etc.), all are variations of the fundamental concept of an association from one object to another.

3. **Transcendental consciouness** is related to itself through its own self-referral dynamics.

4. **Wholeness moving within itself**: In Unity Consciousness, one recognizes that the relationship of the Self to the Self is not only fundamental, but is in reality the only relationship there is.