UML Class Diagrams

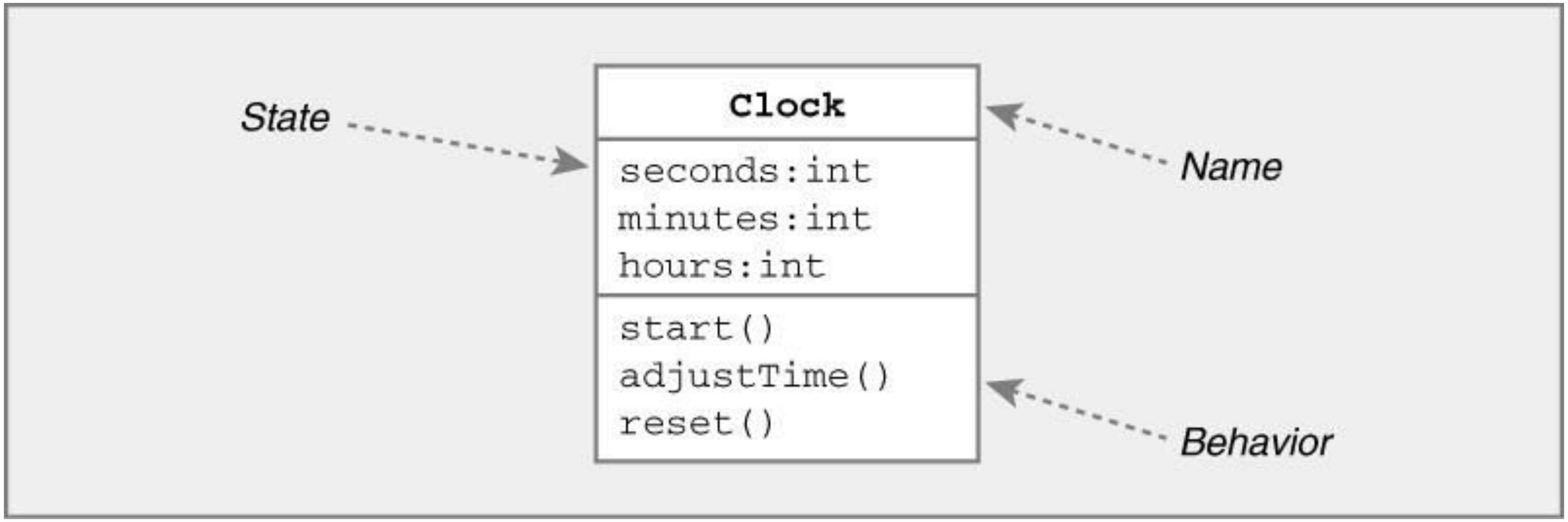
Class

- Template for object creation:
  - Instantiated into objects
  - An abstract data type (ADT)
- Examples: Employees, Books, etc.
- Sometimes not intended to produce instances:

UML Class Diagrams

- Represent the (static) structure of the system

General	In Java	In C++
Name	Name	Name
State	Variables	Members
Behavior	Methods	Functions



State

Clock

seconds:int
minutes:int
hours:int

start()
adjustTime()
reset()

Name

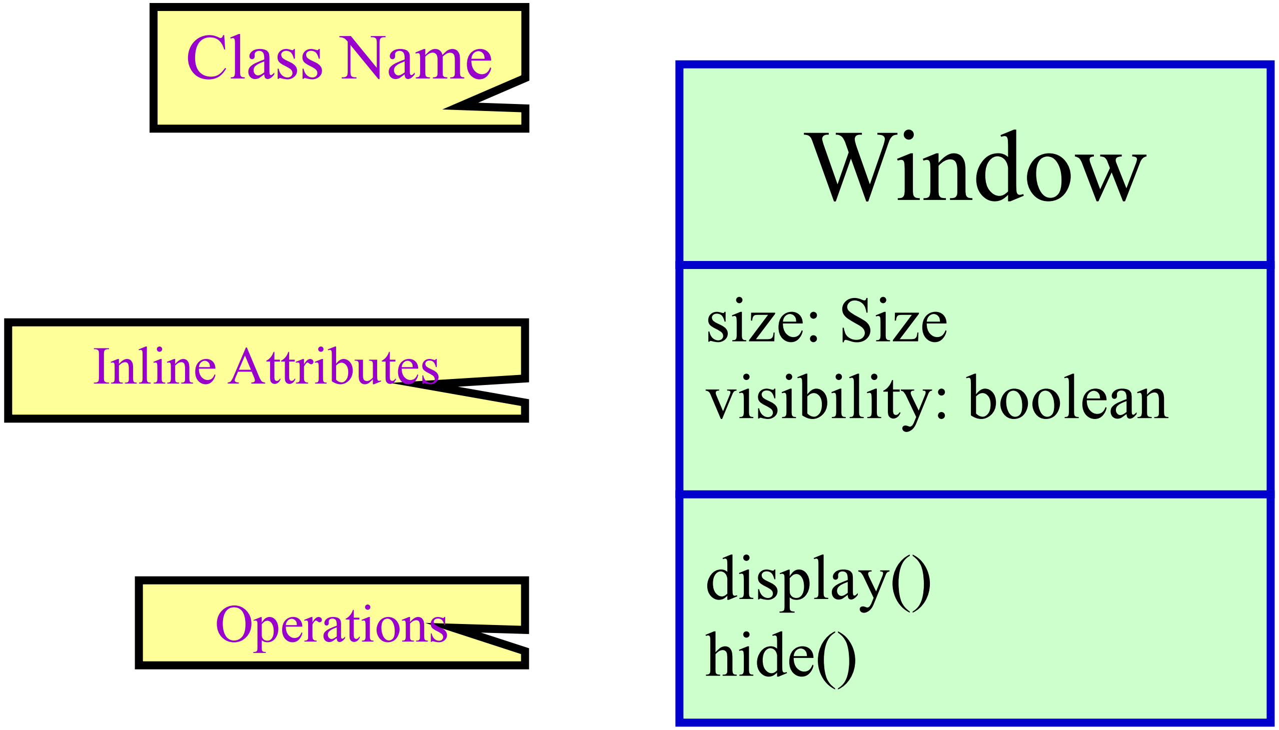
Behavior

Class Attribute Examples

Java Syntax	UML Syntax
Date birthday	Birthday:Date
Public int duration = 100	+duration:int = 100
Private Student students[0..MAX_Size]	-Students[0..MAX_Size]:Student

UML Class Representation

- A class represents a set of objects having similar attributes, operations, relationships and behavior.



Class Name

Inline Attributes

Operations

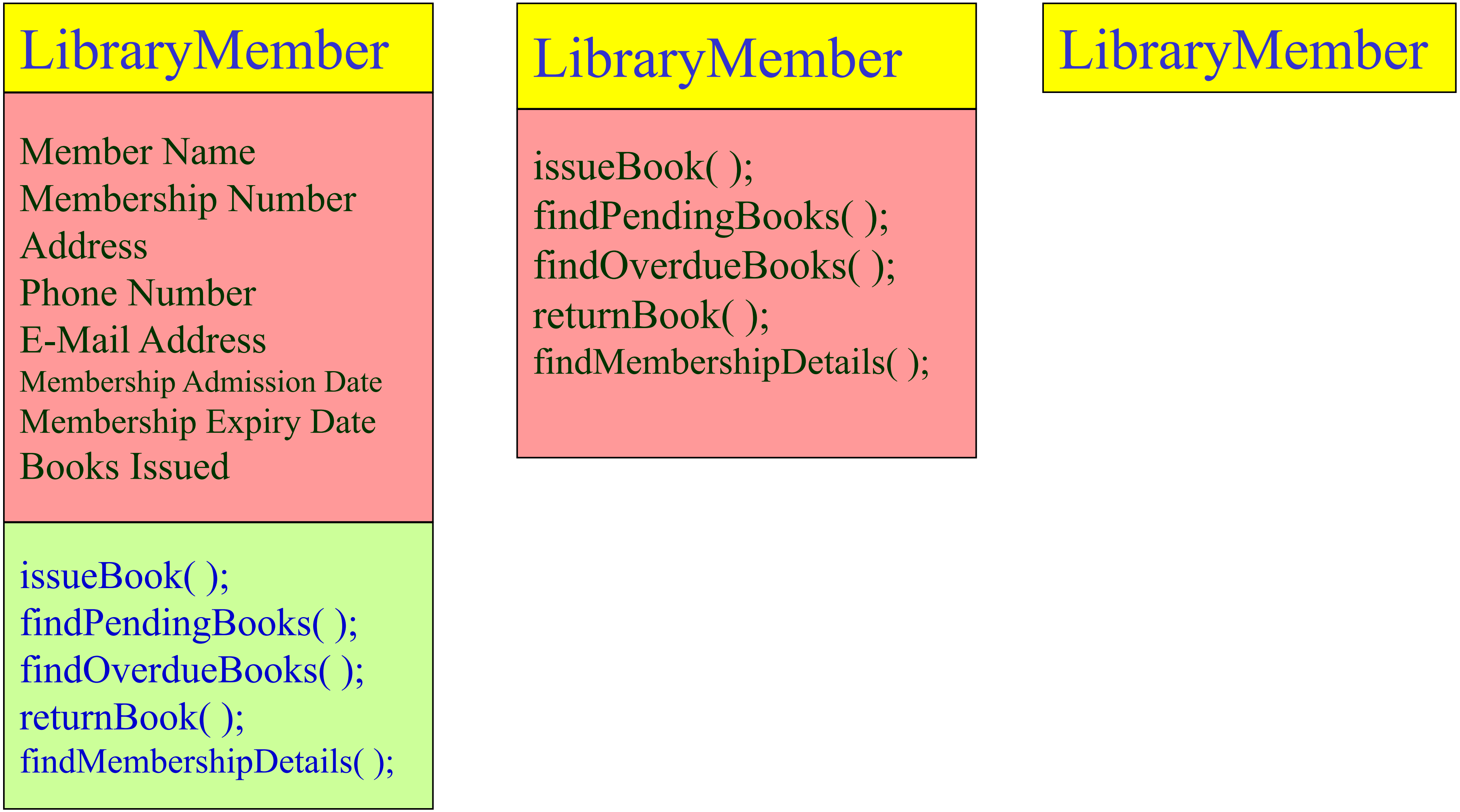
Window

size: Size
visibility: boolean

display()
hide()

A class can implicitly have a few association attributes

Example UML Classes



LibraryMember

Member Name
Membership Number
Address
Phone Number
E-Mail Address
Membership Admission Date
Membership Expiry Date
Books Issued

issueBook( );
findPendingBooks( );
findOverdueBooks( );
returnBook( );
findMembershipDetails( );

LibraryMember

issueBook( );
findPendingBooks( );
findOverdueBooks( );
returnBook( );
findMembershipDetails( );

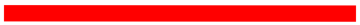
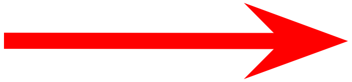
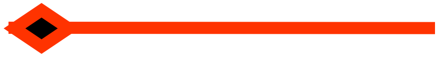
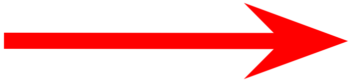
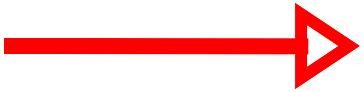
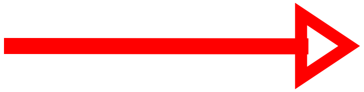
LibraryMember

Different representations of the LibraryMember class

Visibility Syntax in UML

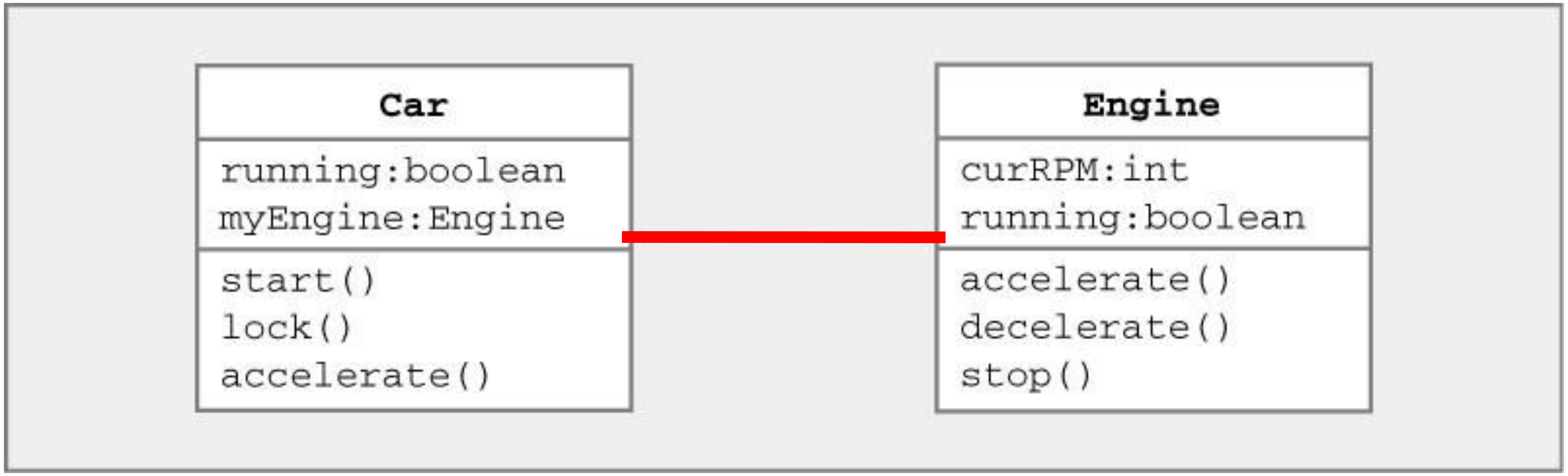
Visibilty	Java Syntax	UML Syntax
public	public	+
protected	protected	#
package		~
private	private	-

Relationships Between Classes

- Association  OR 
  - Permanent, structural, “has a”
  - Solid line (arrowhead optional)
- Aggregation 
  - Permanent, structural, a whole created from parts
  - Solid line with diamond from whole
- Dependency 
  - Temporary, “uses a”
  - Dotted line with arrowhead
- Generalization 
  - Inheritance, “is a”
  - Solid line with open (triangular) arrowhead
- Implementation 
  - Dotted line with open (triangular) arrowhead

Association

- Denotes permanent, structural relationship
- State of class A contains class B
- Represented by solid line (arrowhead optional)



Car

running:boolean
myEngine:Engine

start()
lock()
accelerate()

Engine

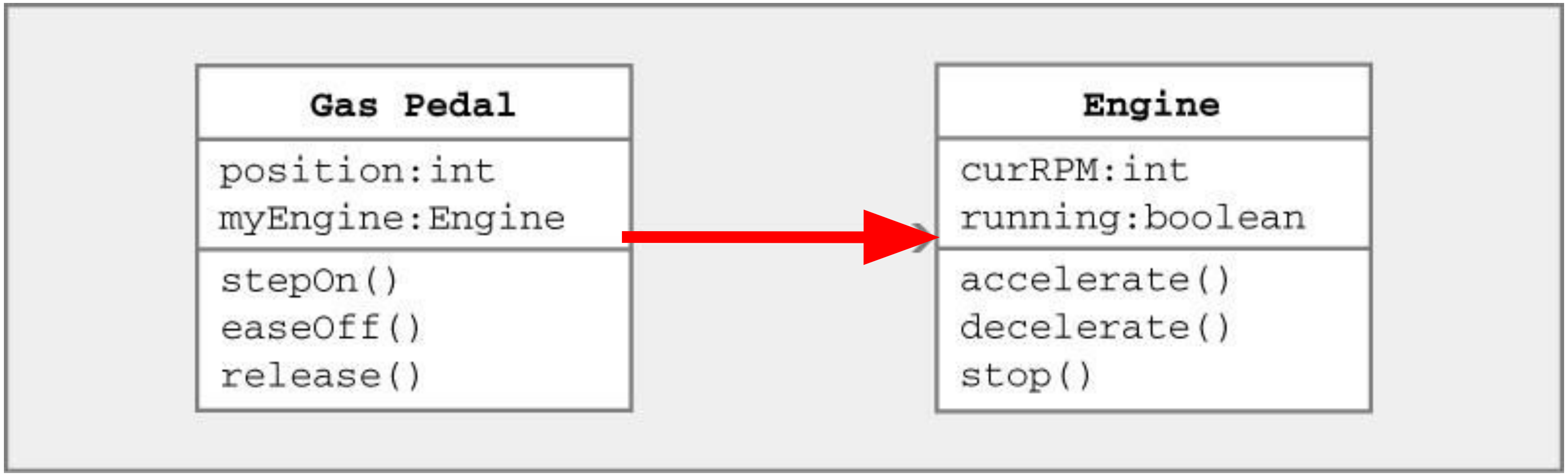
curRPM:int
running:boolean

accelerate()
decelerate()
stop()

Car and Engine classes know about each other

Associations w/ Navigation Information

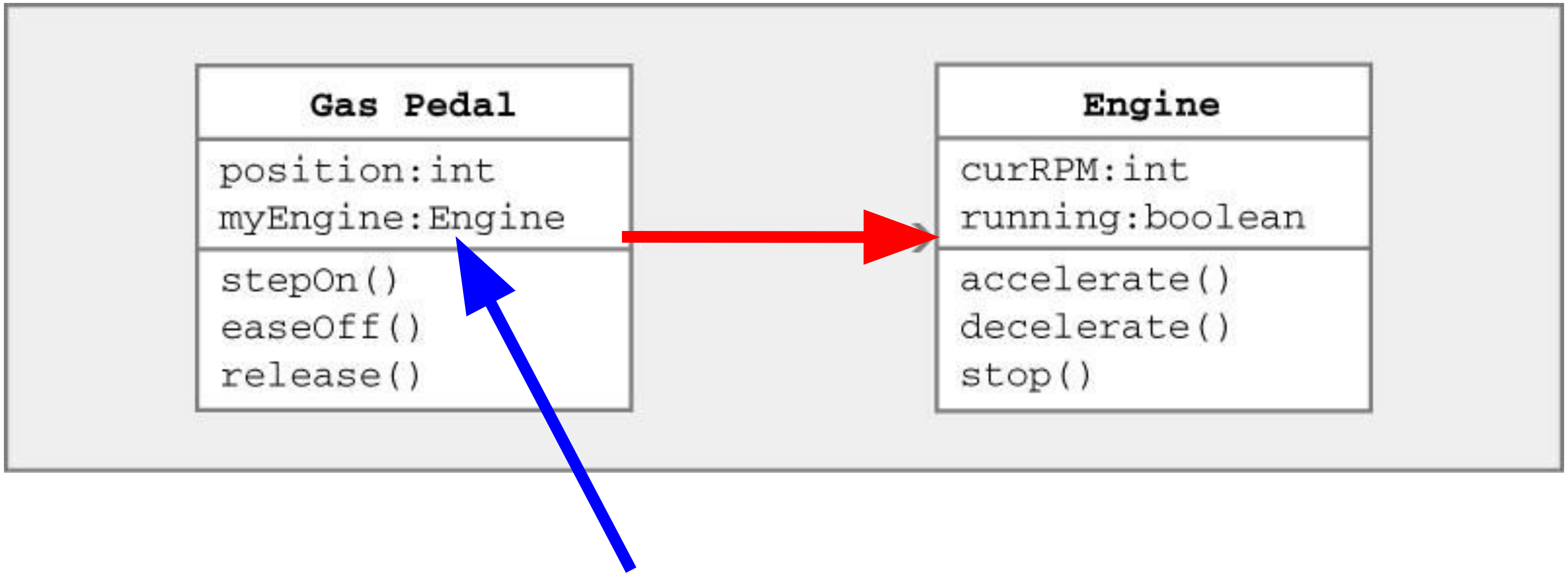
- Can indicate direction of relationship
- Represented by solid line with arrowhead



Gas Pedal class knows about Engine class
Engine class doesn’t know about Gas Pedal class

Associations w/ Navigation Information

- Denotes "has-a" relationship between classes
- "Gas Pedal" has an "Engine"

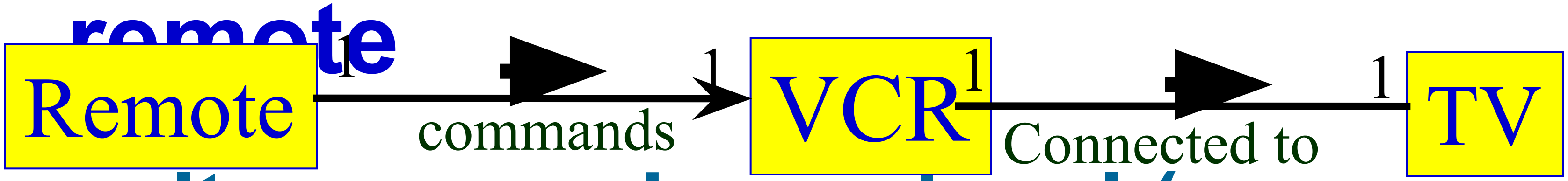


State of Gas Pedal class contains instance of Engine class ⇒ can invoke its methods

Association – example

- In a home theatre system,
  - A TV object has an association with a VCR object

    - It may receive a signal from the VCR
  - VCR may be associated with remote



Remote

1

commands

1

VCR

1

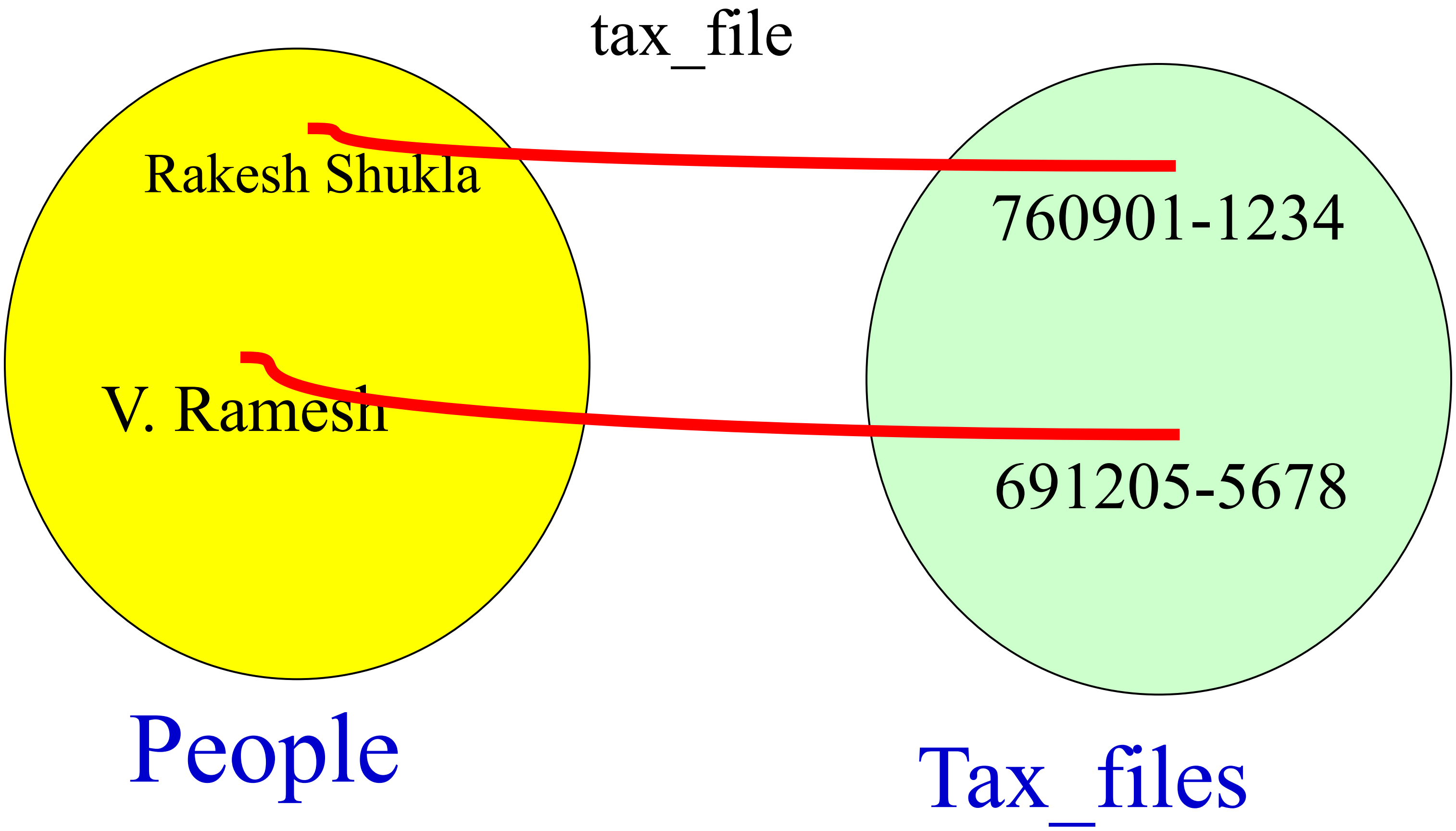
Connected to

1

TV

    - It may receive a signal (command)

1-1 Association – example



tax_file

Rakesh Shukla

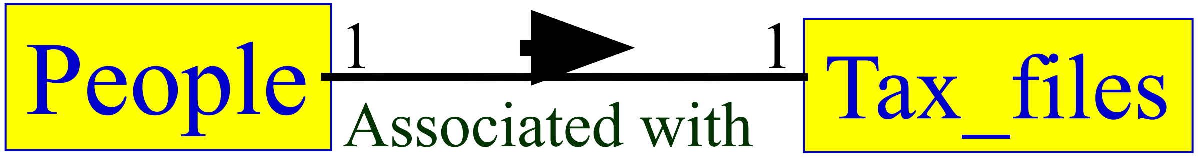
V. Ramesh

760901-1234

691205-5678

People

Tax_files



People

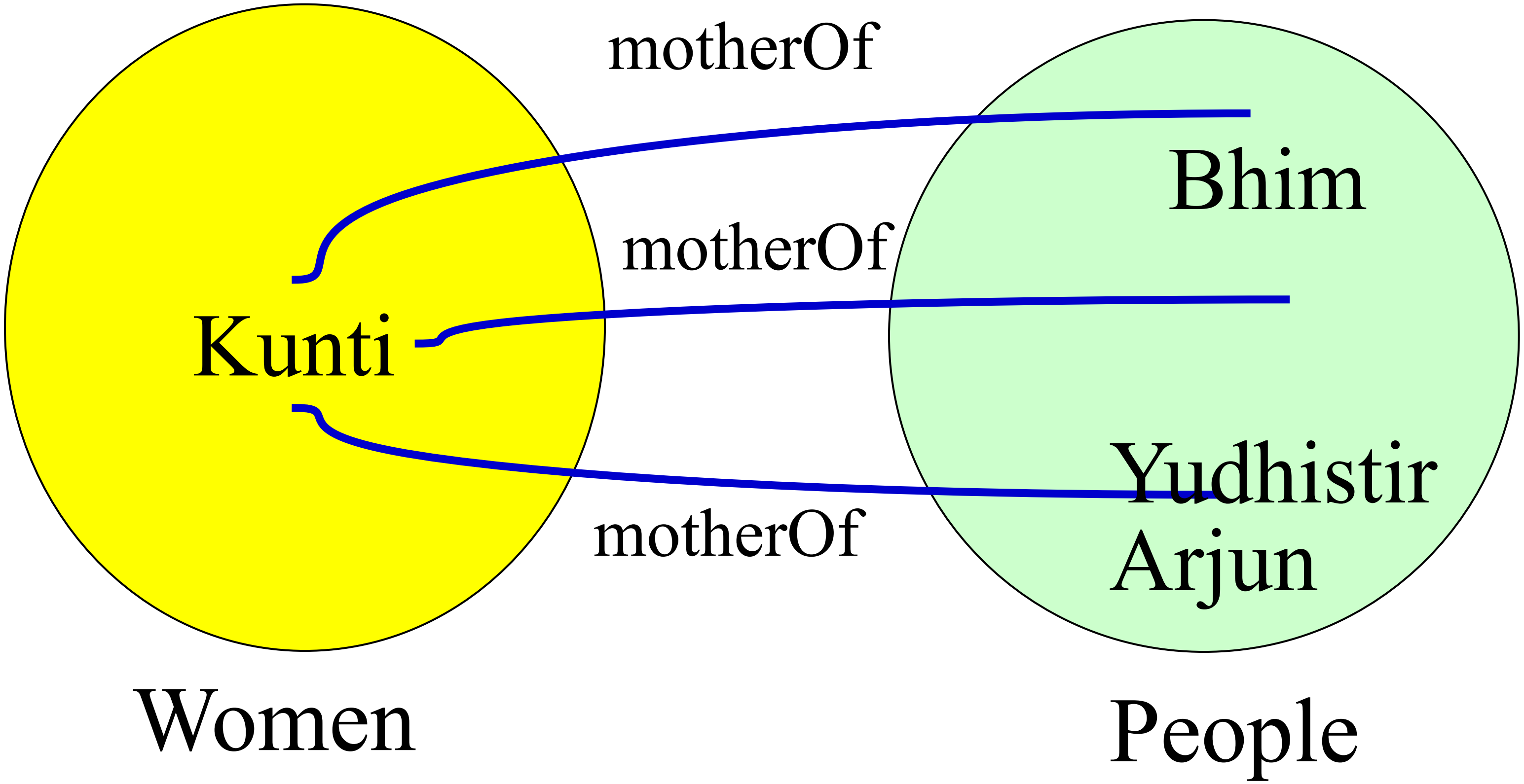
1

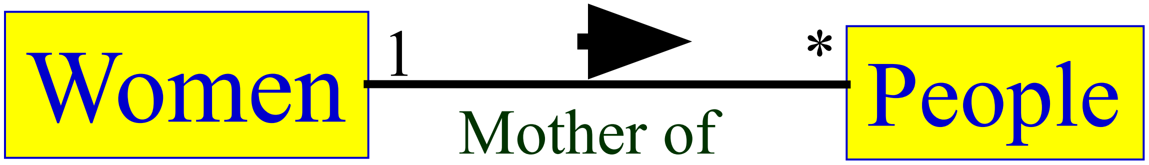
1

Associated with

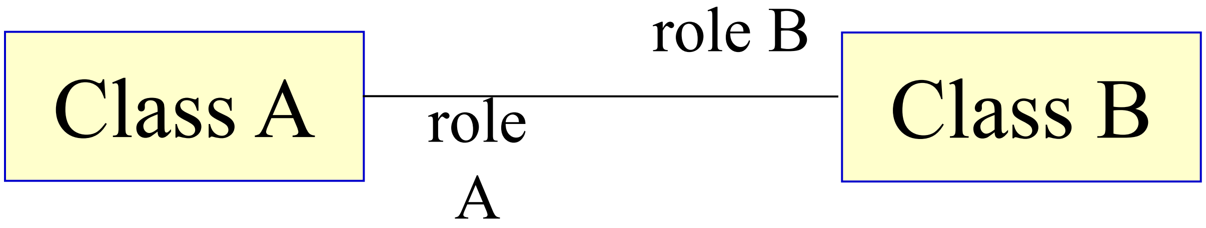
Tax_files

Multiple Association – example

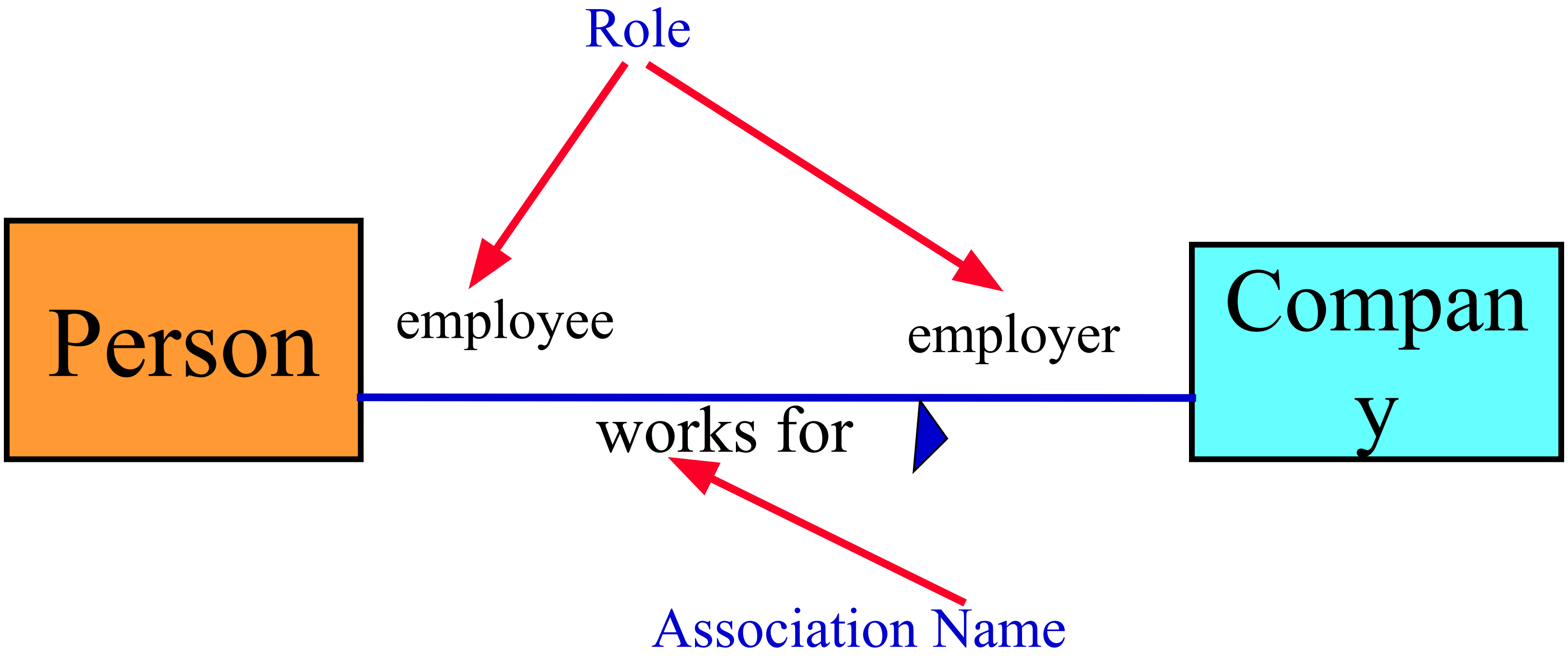




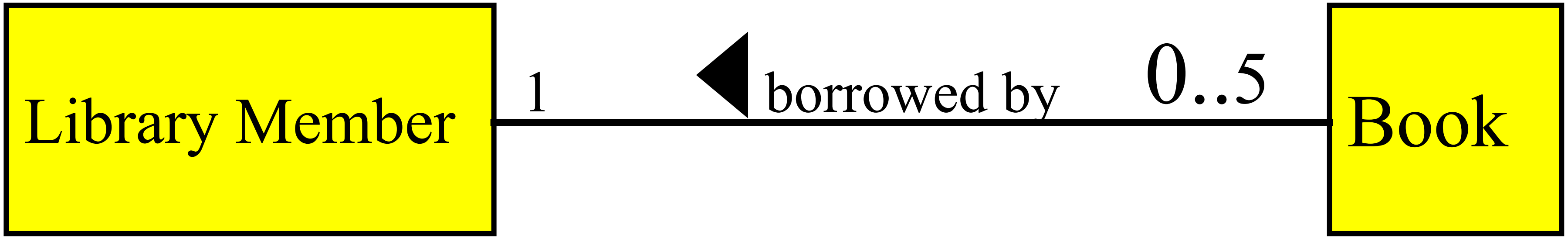
Association UML Syntax



- **A Person works for a Company.**



Association - More Examples



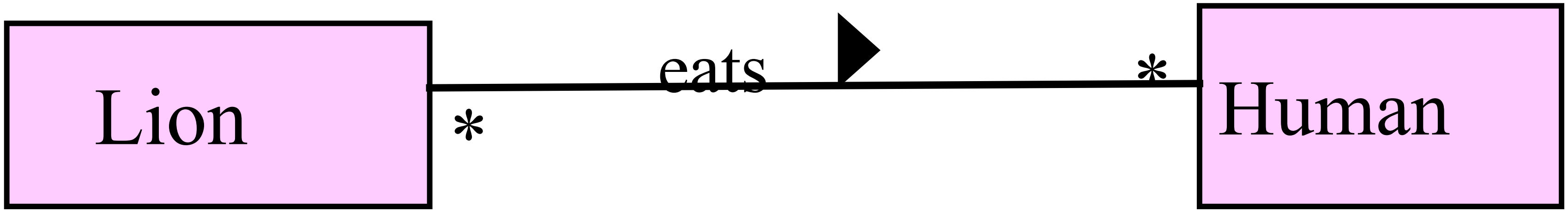
Library Member

1

borrowed by

0..5

Book



Lion

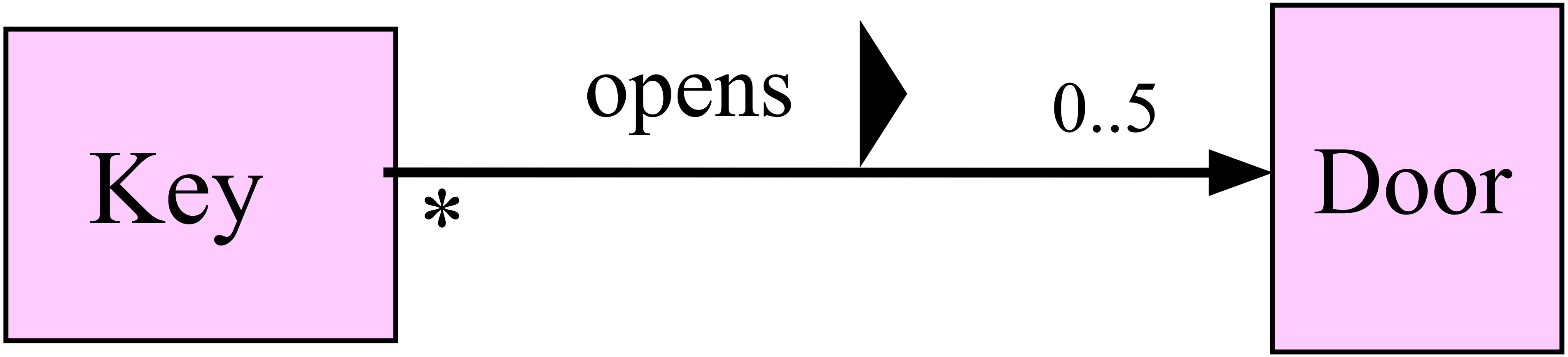
eats

*

*

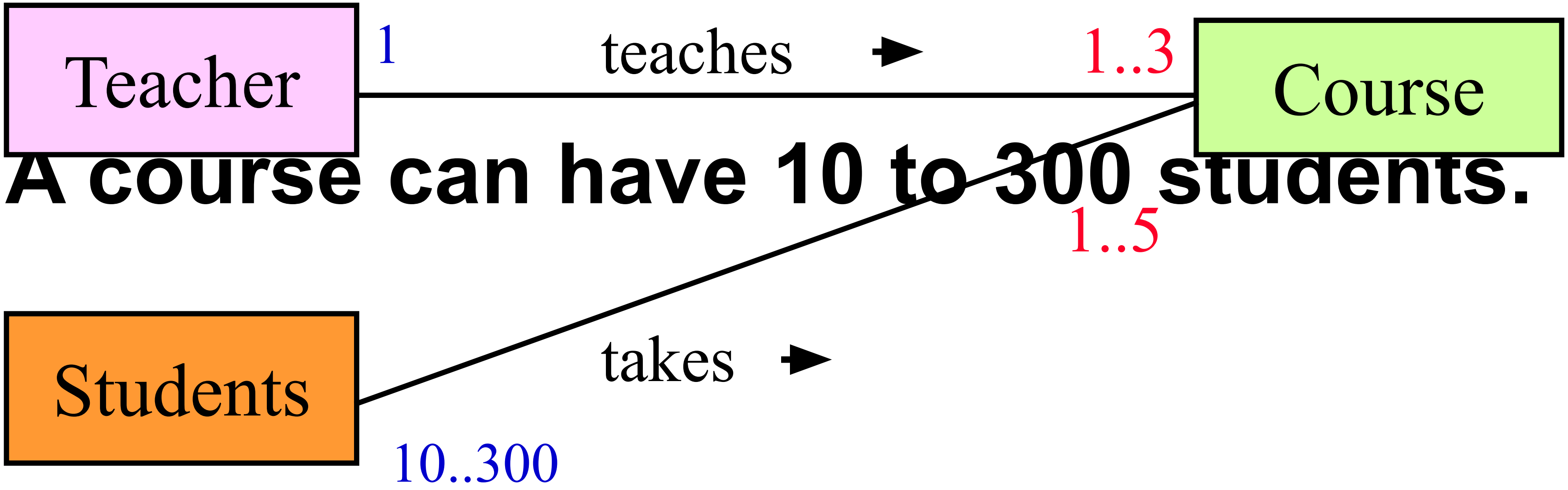
Human

Navigability



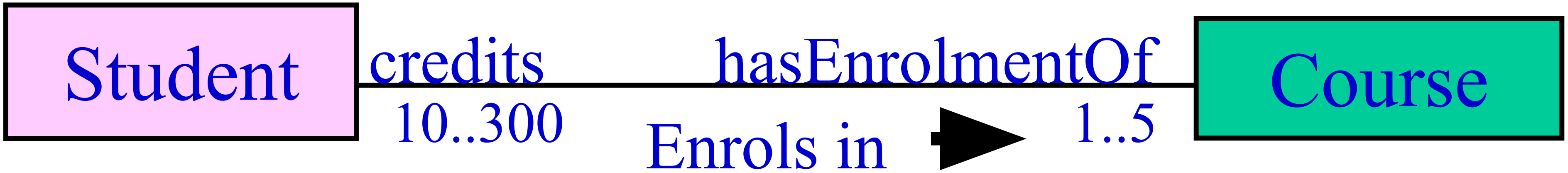
Association – Multiplicity

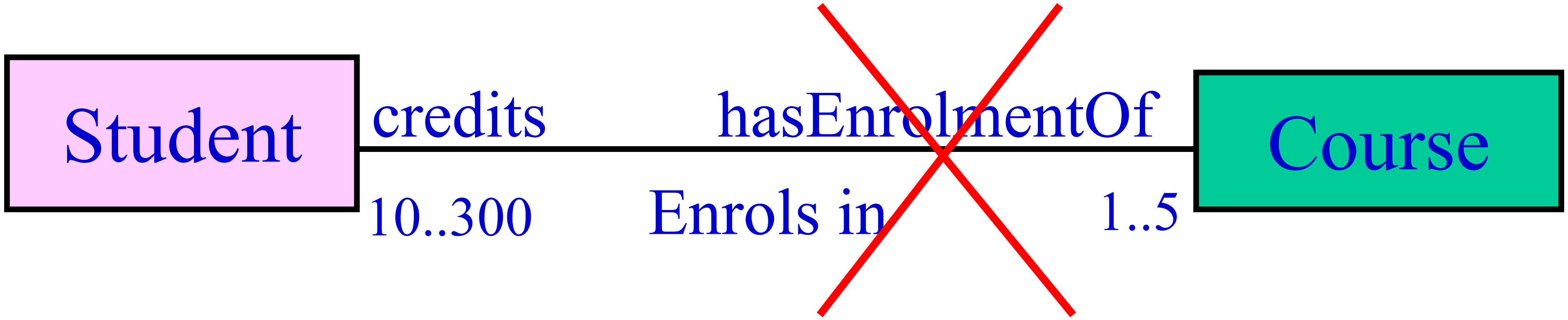
- **A teacher teaches 1 to 3 courses (subjects)**
- **Each course is taught by only one teacher.**
- **A student can take between 1 to 5**
- **A course can have 10 to 300 students.**



Quiz: Draw Class Diagram

- A Student can take up to five Courses.
- A student has to enroll in at least one course.
- Up to 300 students can enroll in a course.
- A class should have at least 10


Student
credits
10..300
hasEnrolmentOf
Enrols in
1..5
Course



Student

credits

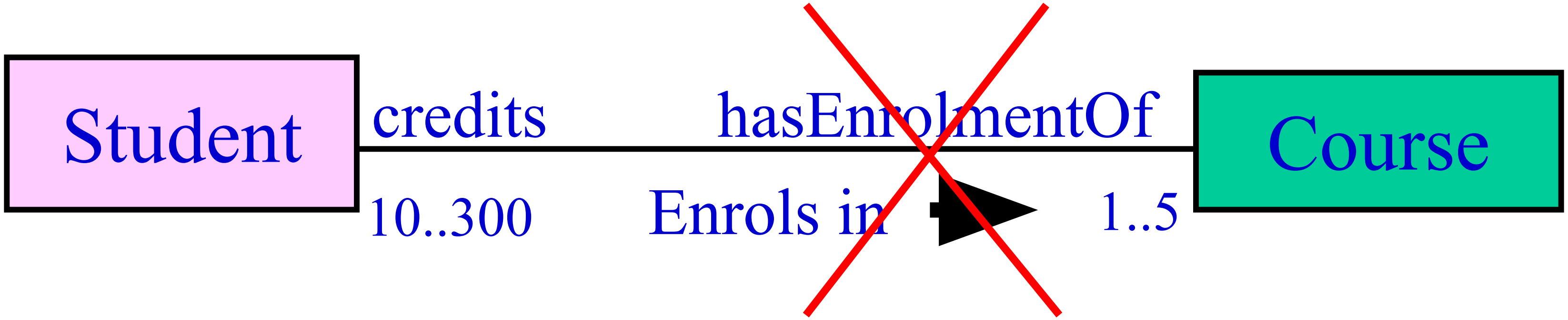
10..300

hasEnrolmentOf

Enrols in

1..5

Course



Student

credits

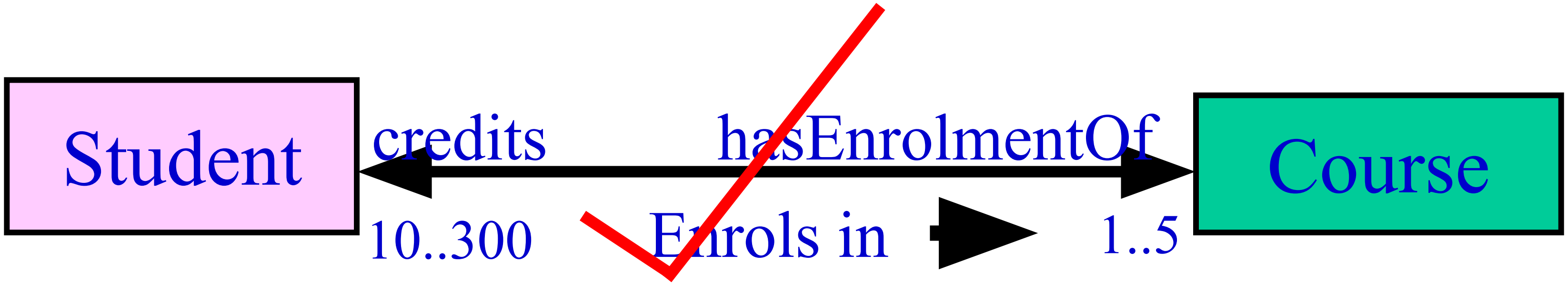
10..300

hasEnrolmentOf

Enrols in

1..5

Course



Student

credits

10..300

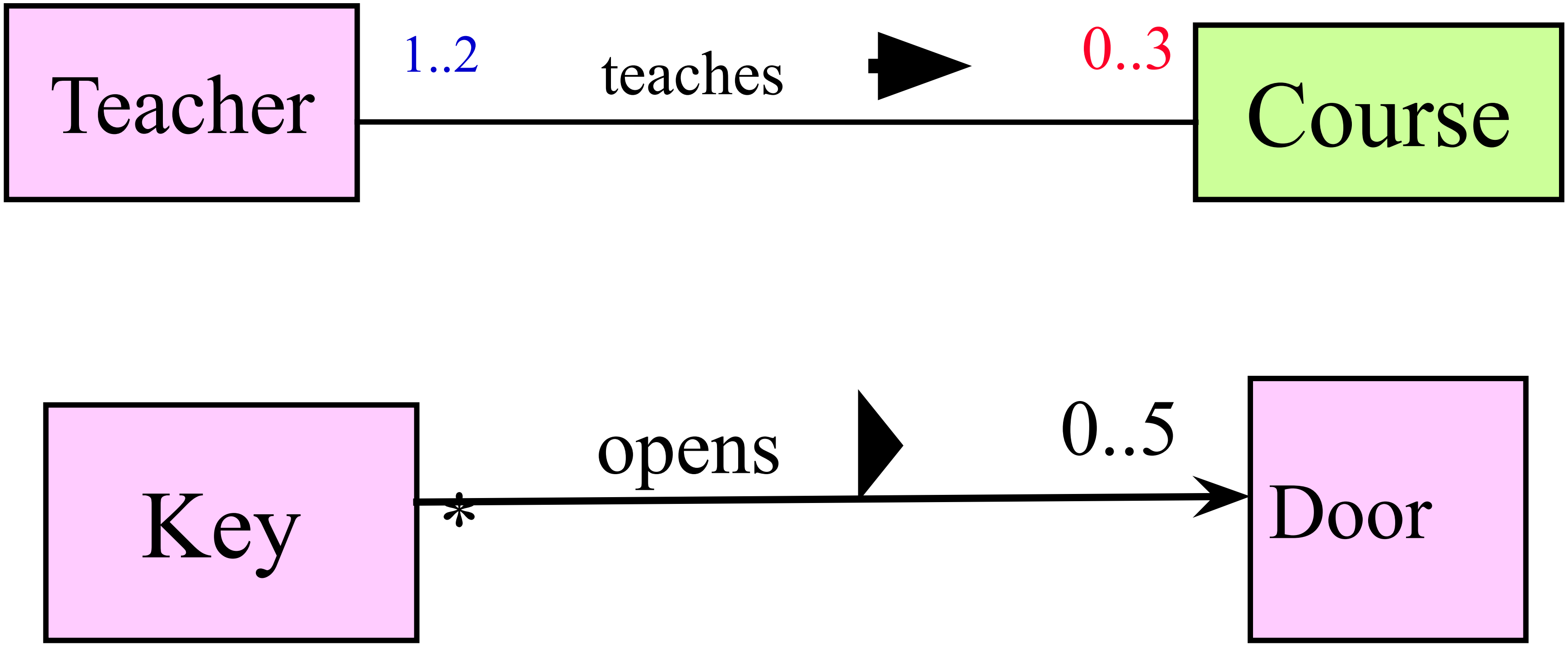
hasEnrolmentOf

Enrols in

1..5

Course

Quiz: Read the Diagram?



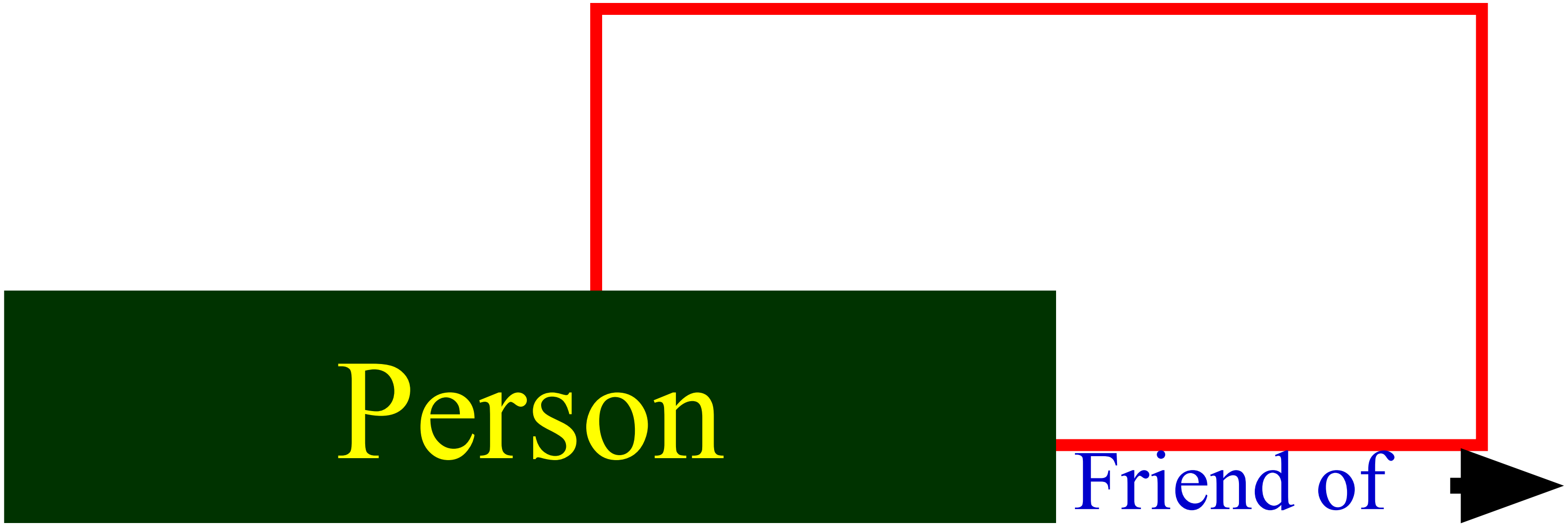
Association and Link

- A link:
  - An instance of an association
  - Exists between two or more objects
  - Dynamically created and destroyed as the run of a system proceeds
- For example:
  - An employee joins an organization.
  - Leaves that organization and joins a new organization etc.

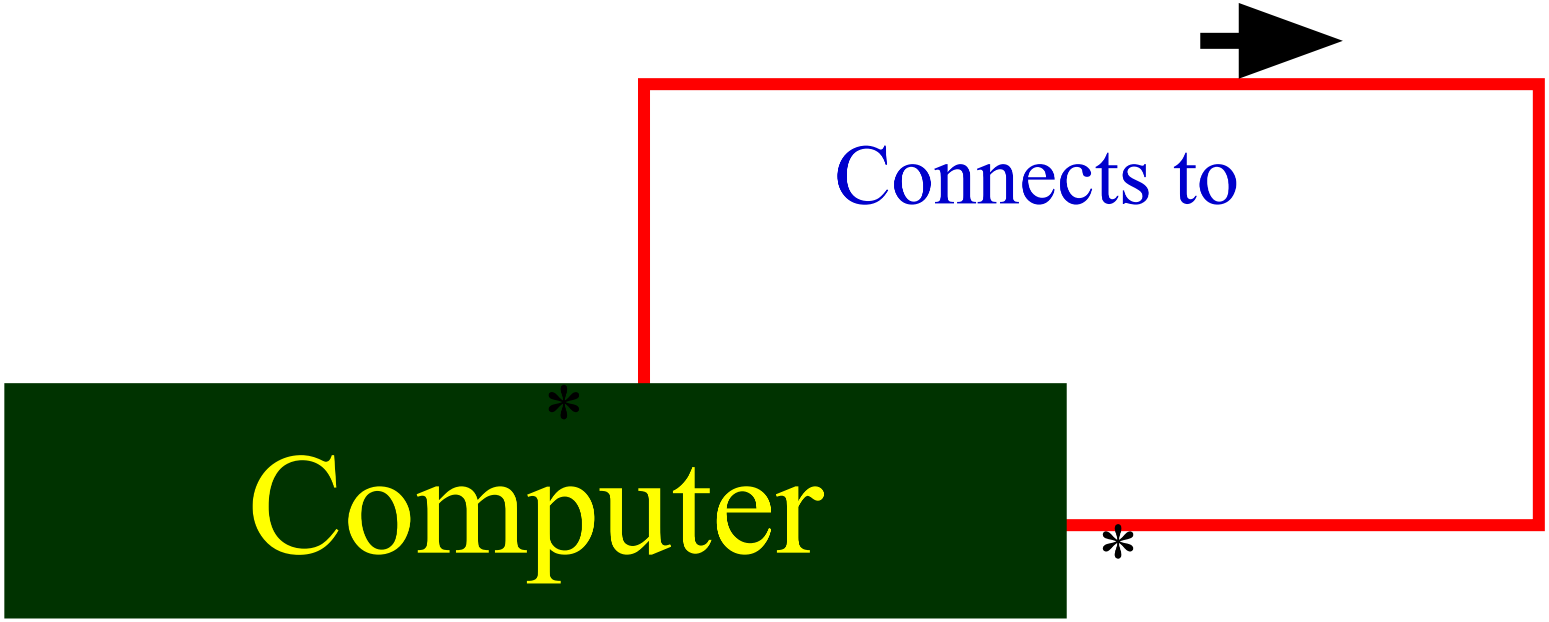
Association Relationship

- A class can be associated with itself (recursive association).
  - Give an example?
- An arrowhead used along with name:
  - Indicates direction of association.
- Multiplicity indicates # of instances taking part in the association.

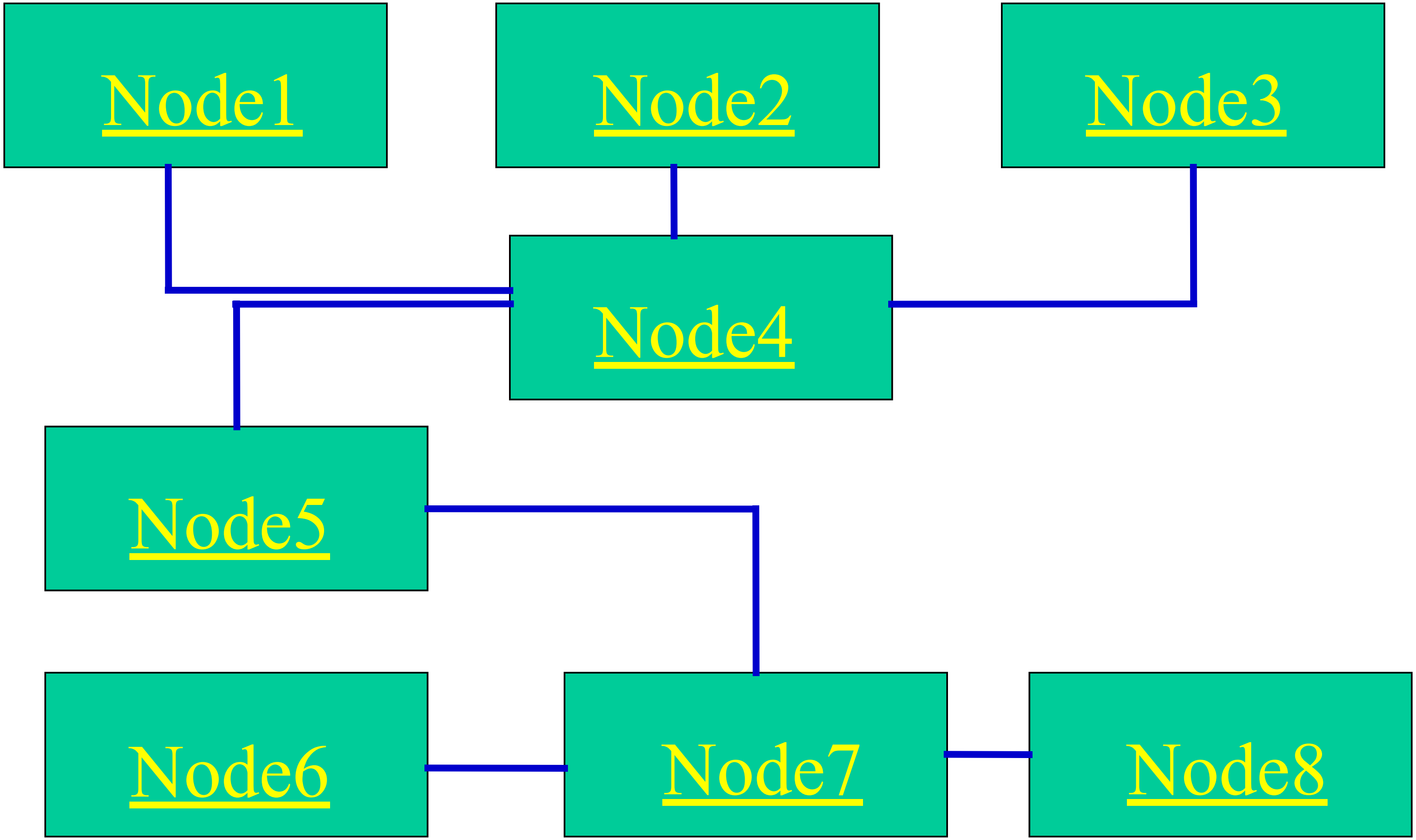
Self Association: Example 0



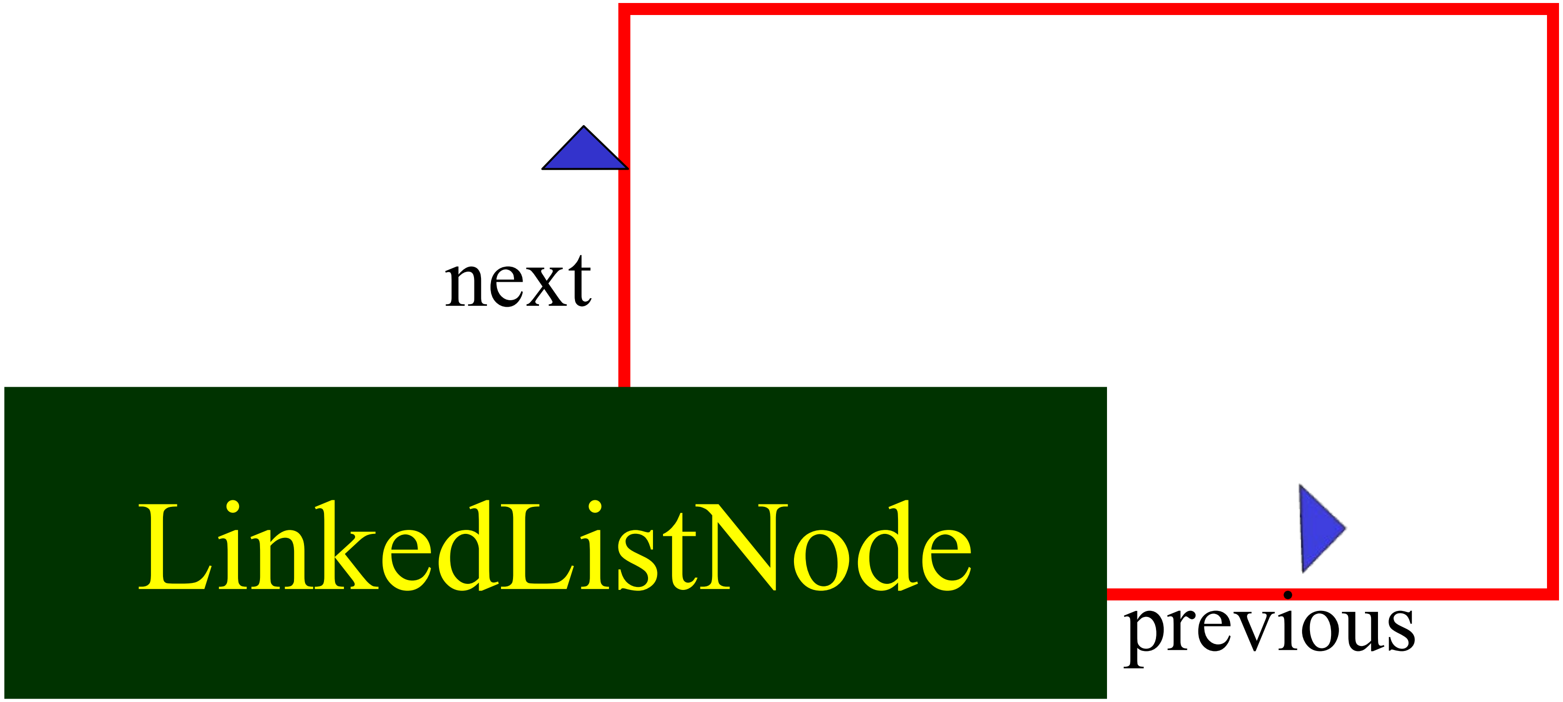
Self Association: Example 0
Computer Network



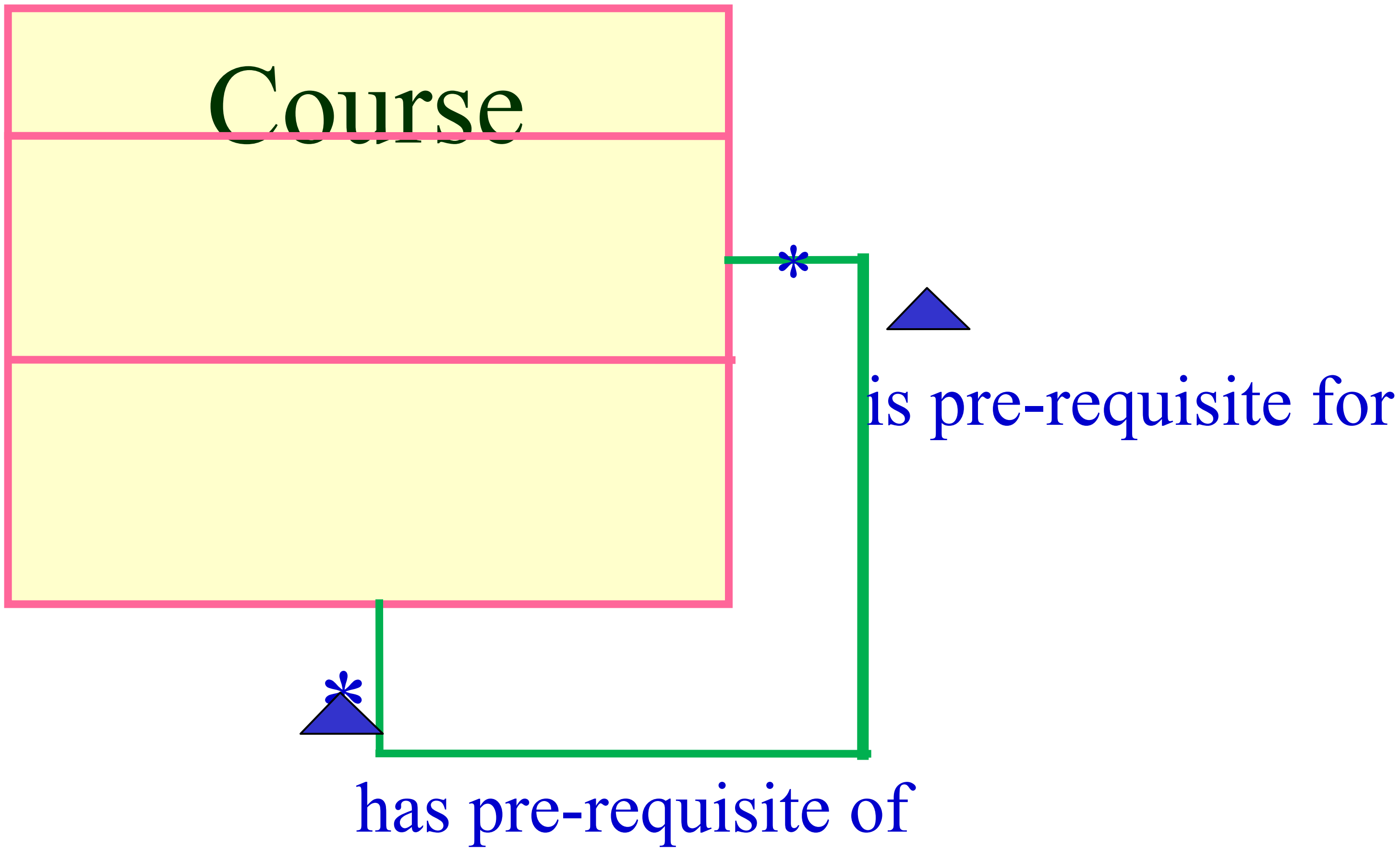
Computer Network: Object Diagram



Self Association: Example 1



Reflexive Association: Example 2

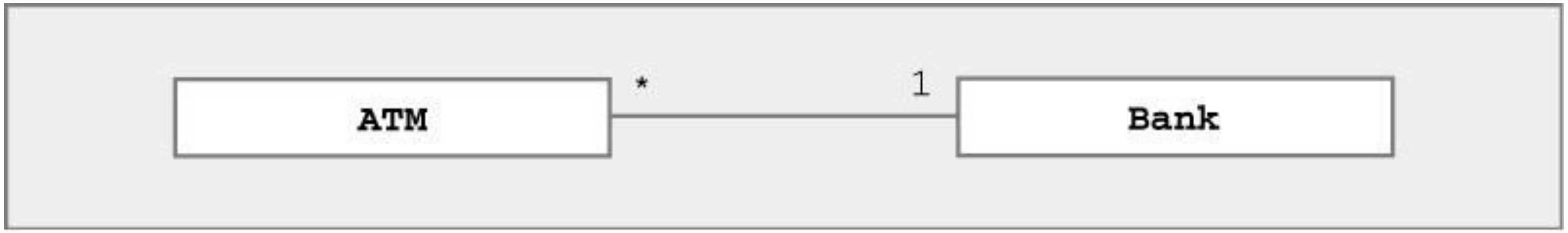


Multiplicity of Associations

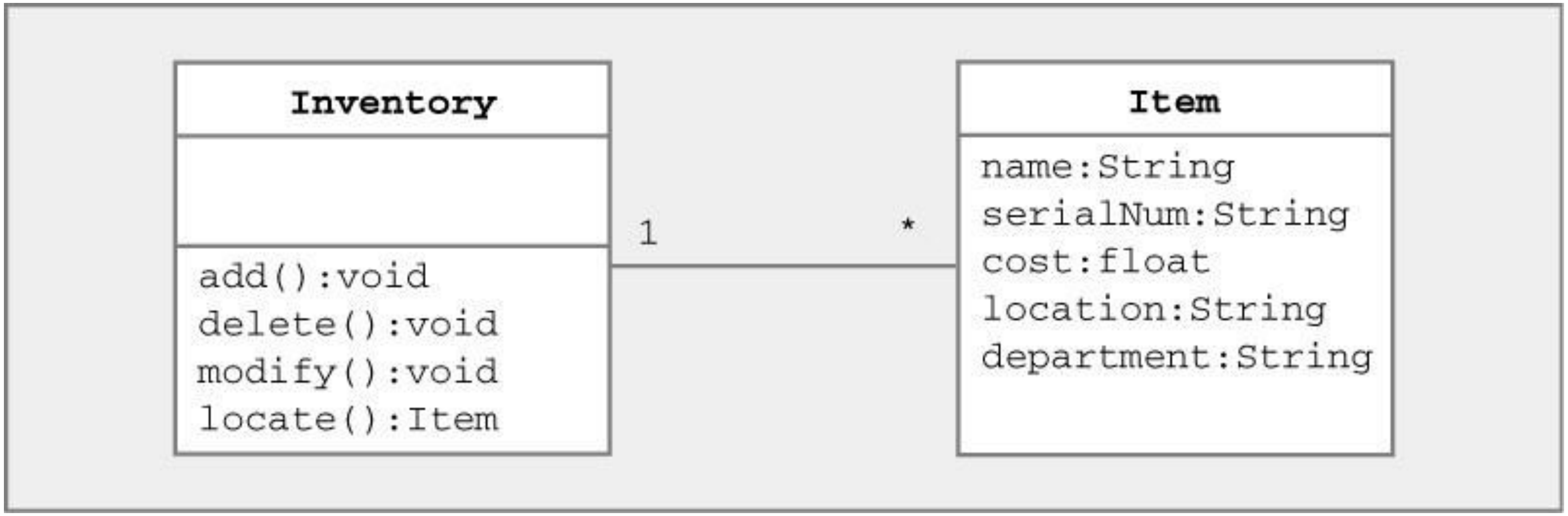
- Some relationships may be quantified
- Multiplicity denotes how many objects the source object can legitimately reference
- Notation
  - * ⇒ 0, 1, or more
  - 5 ⇒ 5 exactly
  - 5..8 ⇒ between 5 and 8, inclusive
  - 5..* ⇒ 5 or more

Multiplicity of Associations

- Many-to-one
  - Bank has many ATMs, ATM knows only 1 bank



- One-to-many
  - Inventory has many items, items know 1 inventory

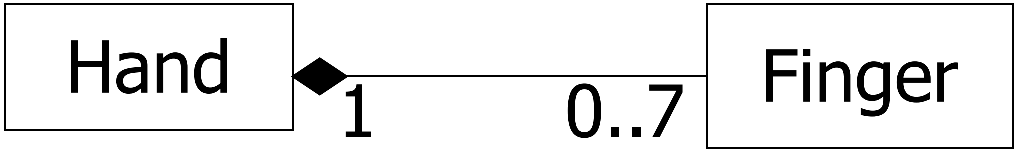


Aggregation and Composition

- **A special kind of association**
- **Models whole-part relationship between things**
- **Whole is usually referred to as *composite***

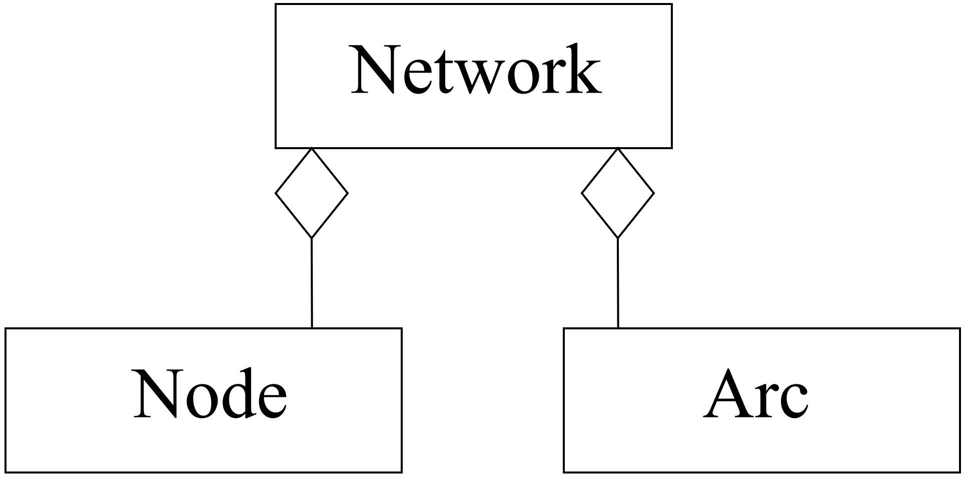
Composite aggregation

- Also referred to as composition
- Composite solely owns the part and they are in a tree structure parts hierarchy
- Most common form of aggregation
- In UML, represented by filled diamond



Shared Aggregation

- Part may be in many composite instances
- In UML, represented as hollow diamond



How to identify aggregation

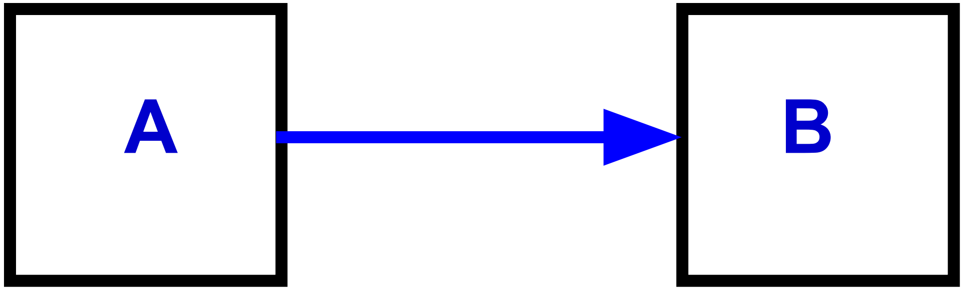
- Lifetime of part is bound within lifetime of composite
  - There is a create-delete dependency
- There is an obvious whole-part physical or logical assembly
- Some properties of composite propagate to parts (e.g., location)
- Operations applied to composite propagate to parts (e.g., destruction, movement, recording)

Why show aggregation

- Clarifies domain constraints regarding part-whole relationship
- Assists in identification of a *creator*
- Operations applied to whole should usually propagate to parts
- Identifying whole wrt a part supports encapsulation

Dependency

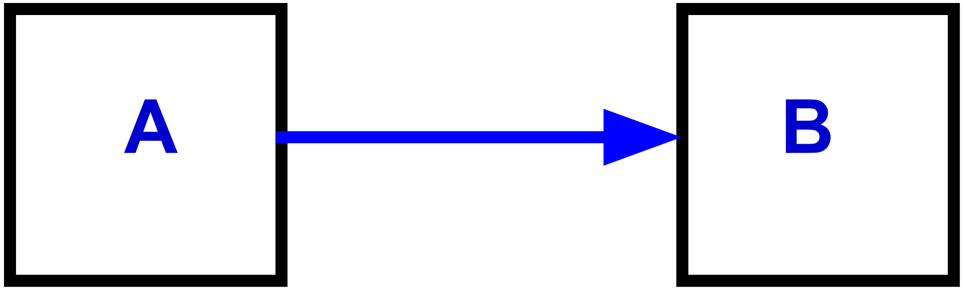
- Denotes dependence between classes
- Always directed (Class A depends on B)
- Represented by dotted line with arrowhead



A depends on B

Dependency

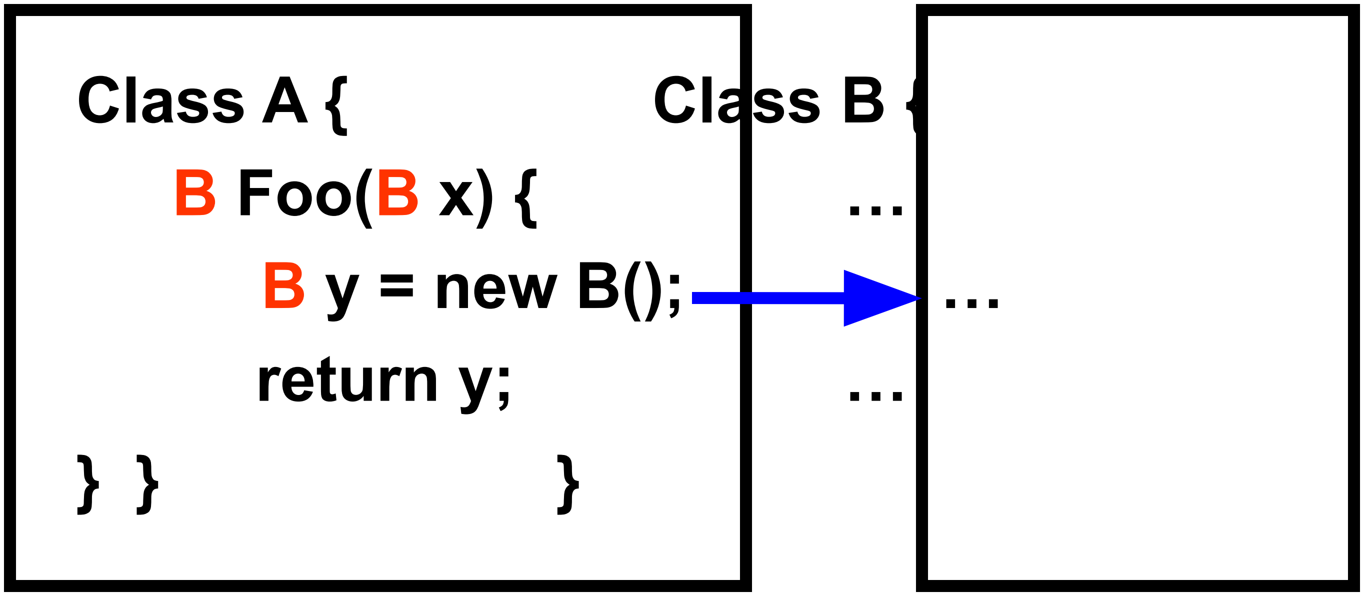
- Caused by class methods
- Method in Class A temporarily "uses a" object of type Class B
- Change in Class B may affect class A



A uses object of class B

Dependency

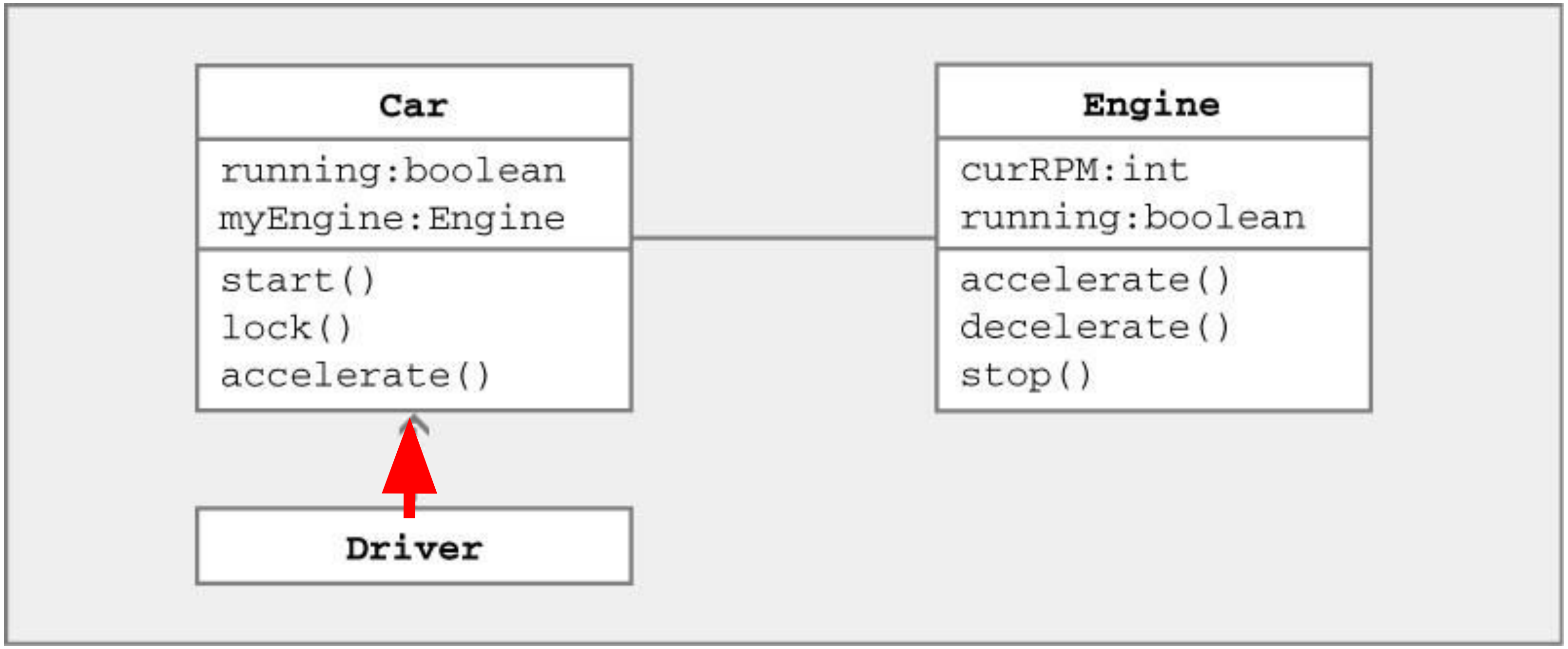
- Dependence may be caused by
  - Local variable
  - Parameter
  - Return value
- Example



```
Class A {
    B Foo(B x) {
        B y = new B();
        return y;
} }
```

```
Class B {
    …
    …
    …
}
```

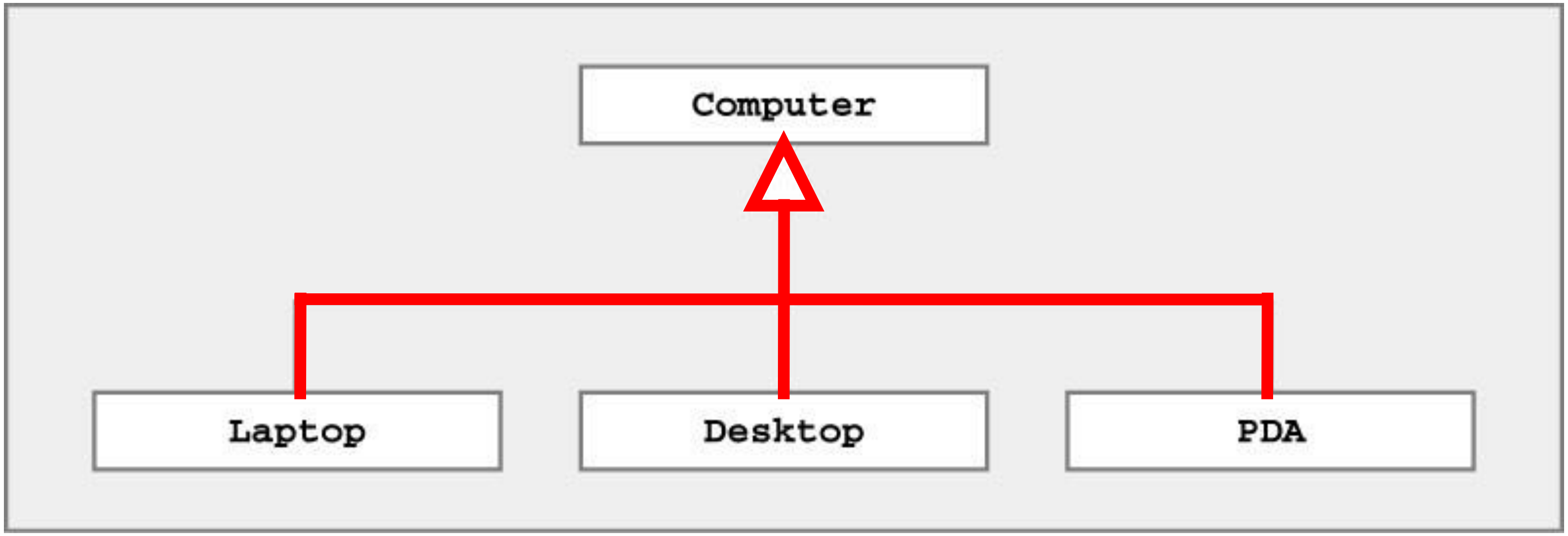
Dependency Example



Class Driver depends on Class Car

Generalization

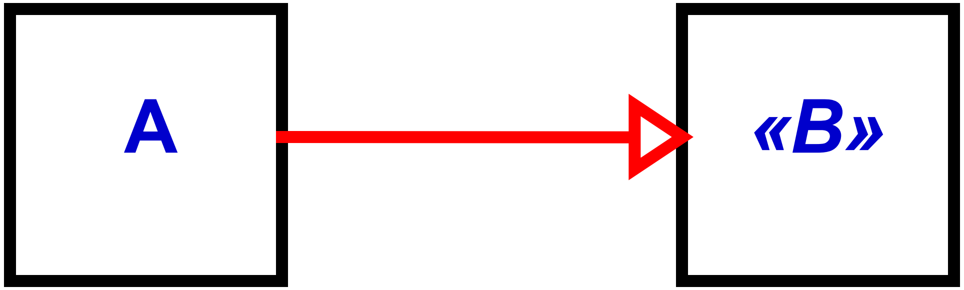
- Denotes inheritance between classes
- Can view as “is-a” relationship
- Represented by line ending in (open) triangle



Laptop, Desktop, PDA inherit
state & behavior from Computers

Implementation

- Denotes class implements Java interface
- Represented by dotted line ending in (open) triangle



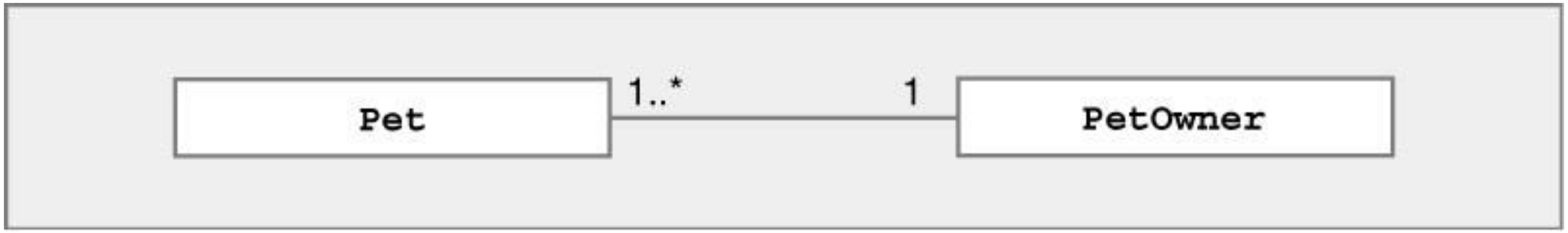
A implements interface B

UML Examples

- Read UML class diagram
- Try to understand relationships
- Examples
  - Pets & owners
  - Computer disk organization
  - Banking system
  - Home heating system
  - Printing system

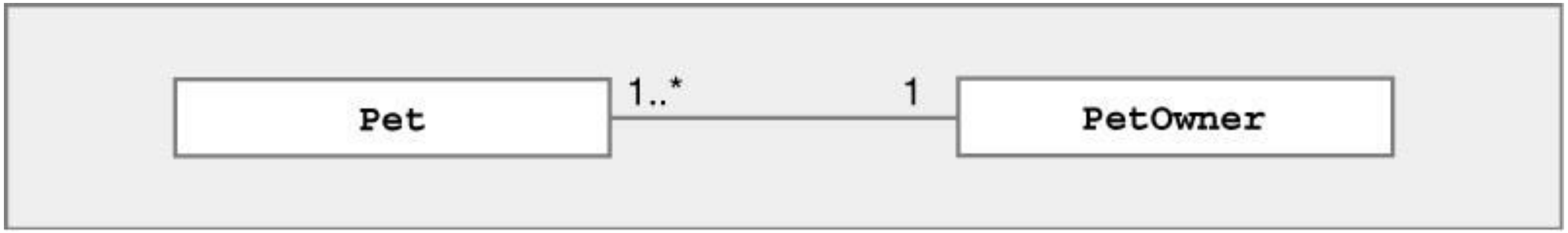
UML Example – Veterinary System

- **Try to read & understand UML diagram**



UML Example – Veterinary System

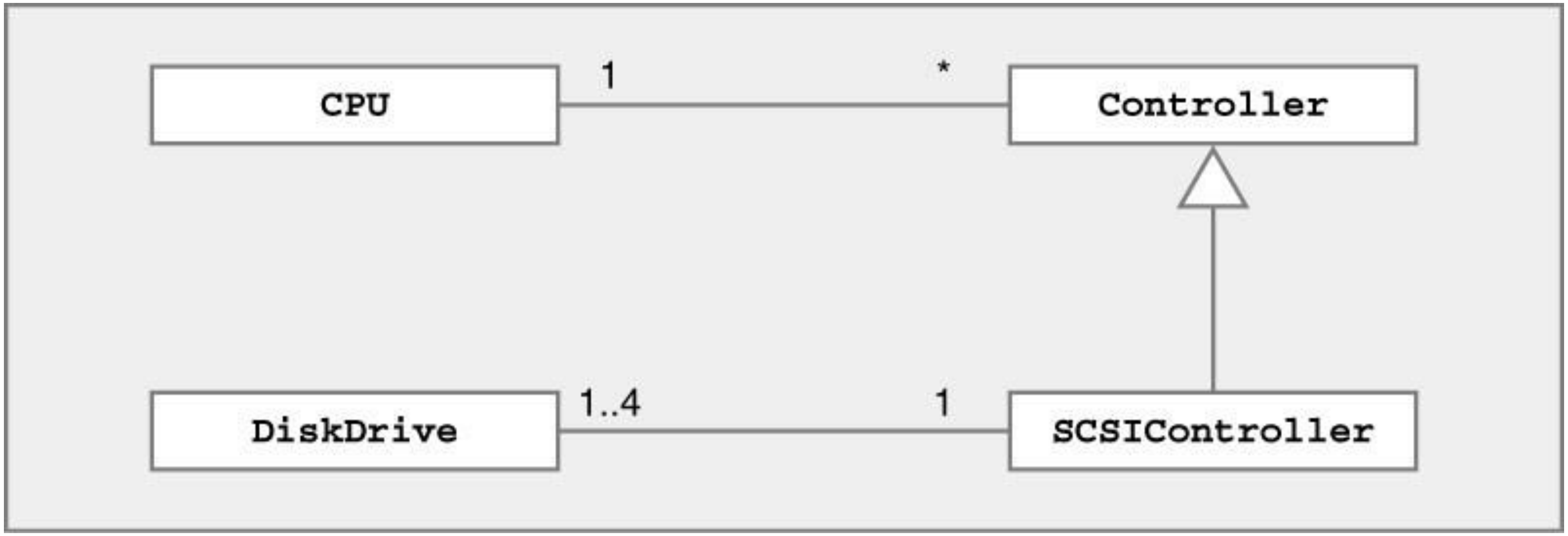
- Try to read & understand UML diagram



- 1 or more Pets associated with 1 PetOwner

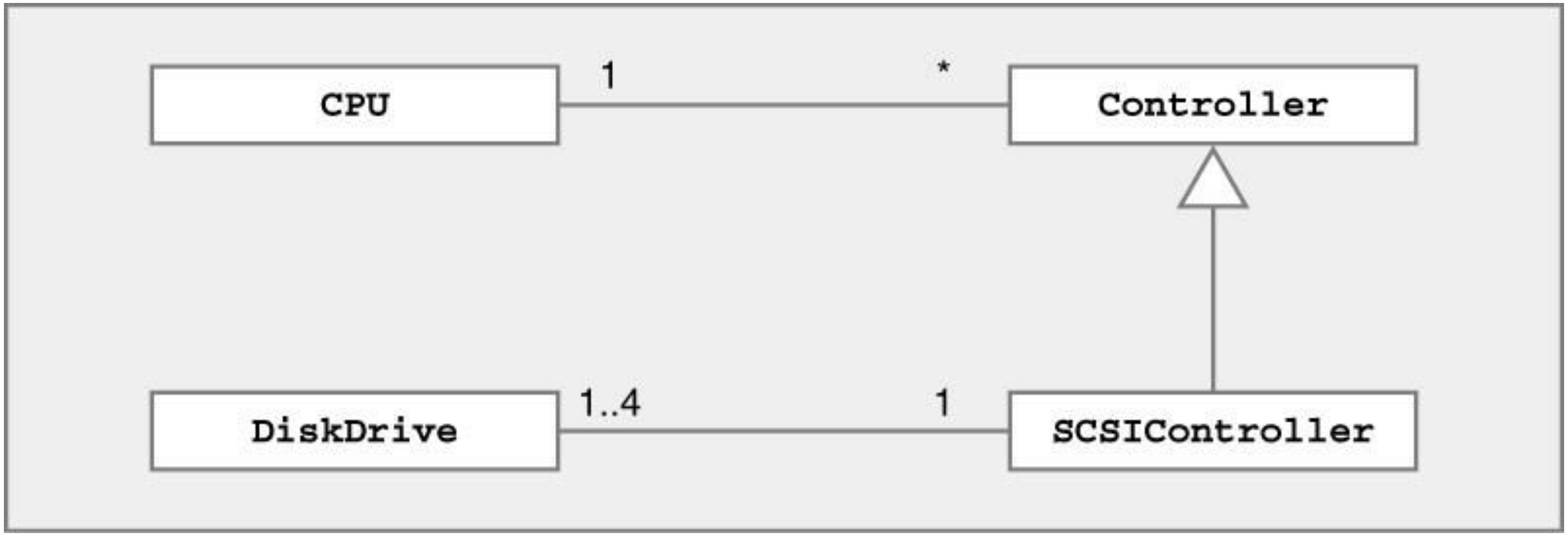
UML Example – Computer System

- **Try to read & understand UML diagram**



UML Example – Computer System

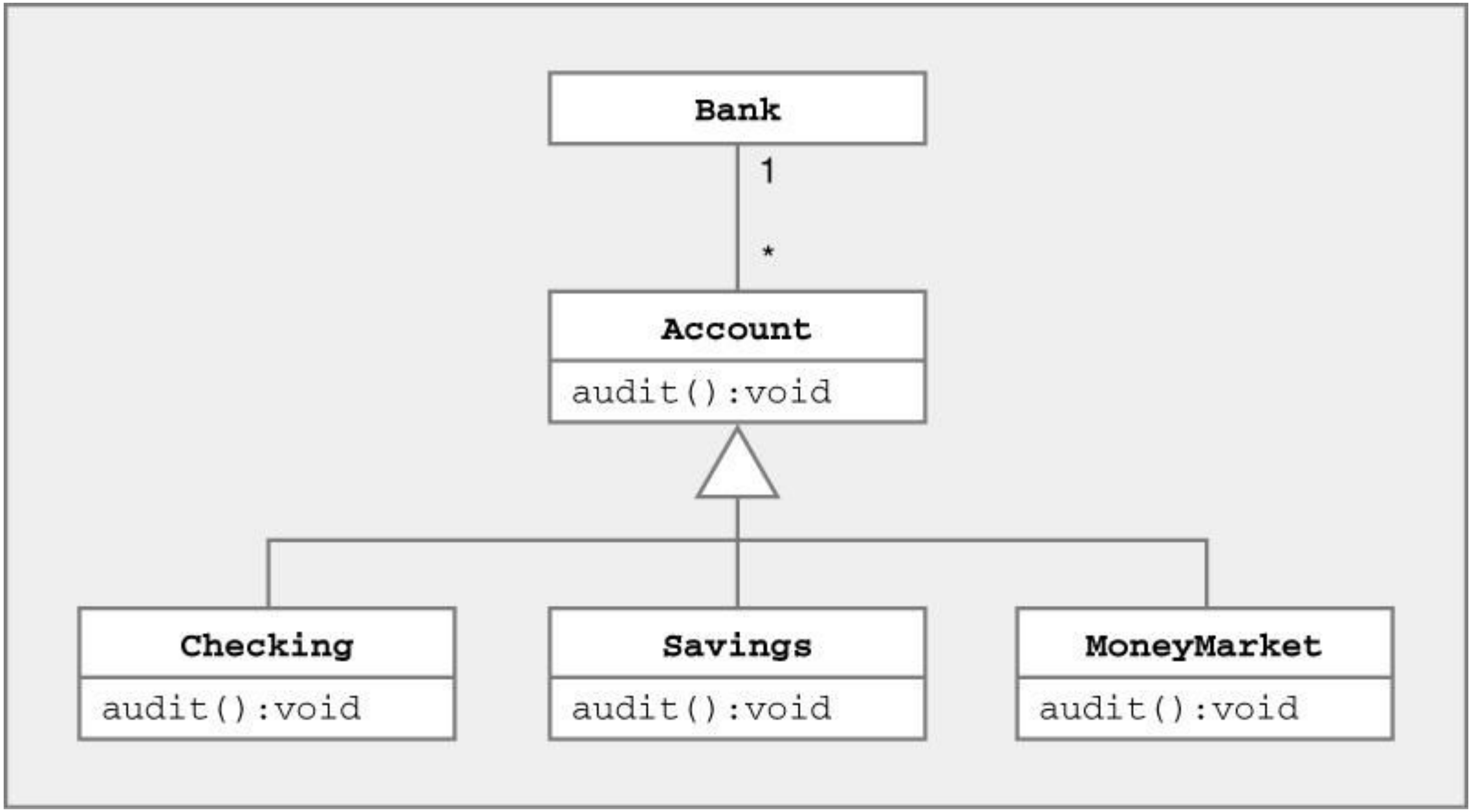
- Try to read & understand UML diagram



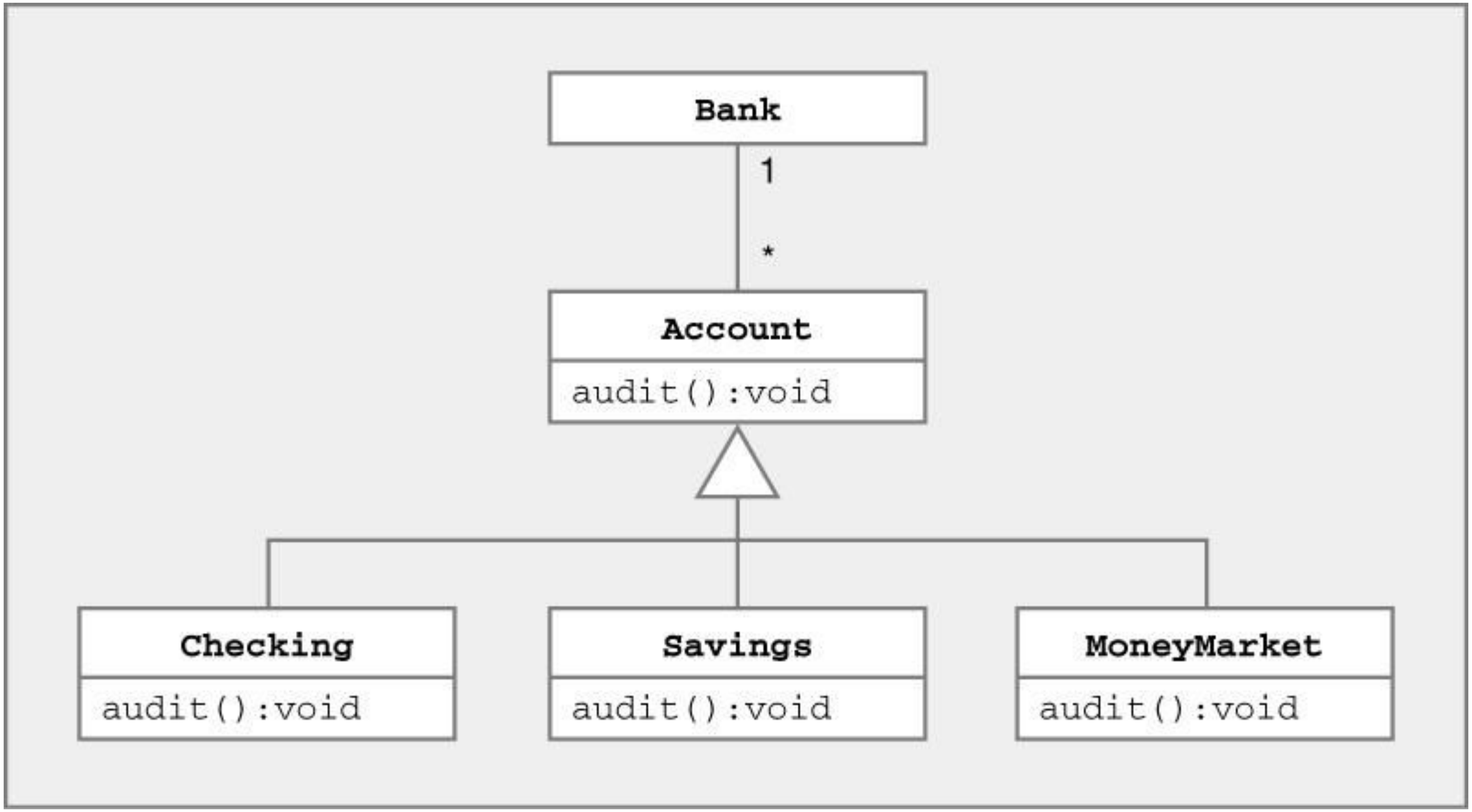
- 1 CPU associated with 0 or more Controllers
- 1-4 DiskDrives associated with 1 SCSIController
- SCSIController is a (specialized) Controller

UML Example – Banking System

- **Try to read & understand UML diagram**



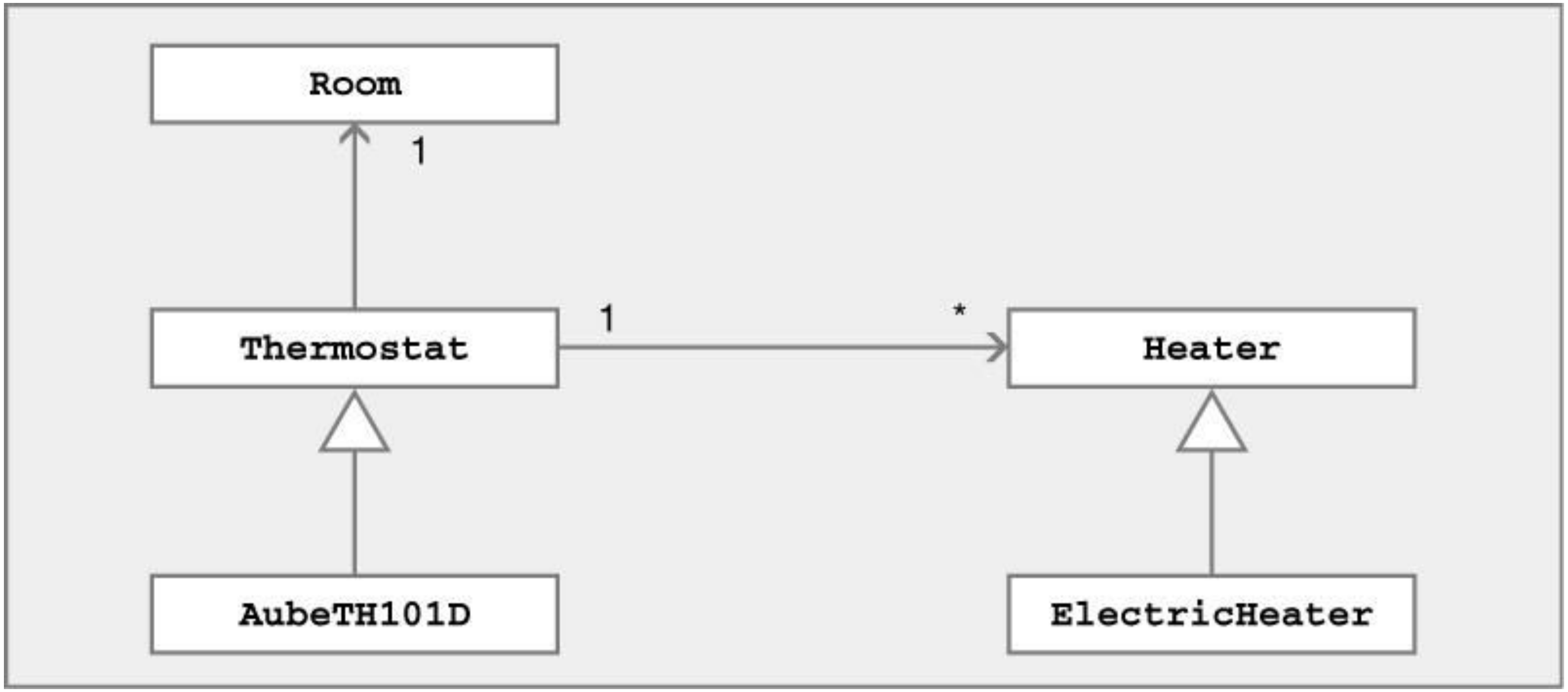
UML Example – Banking System



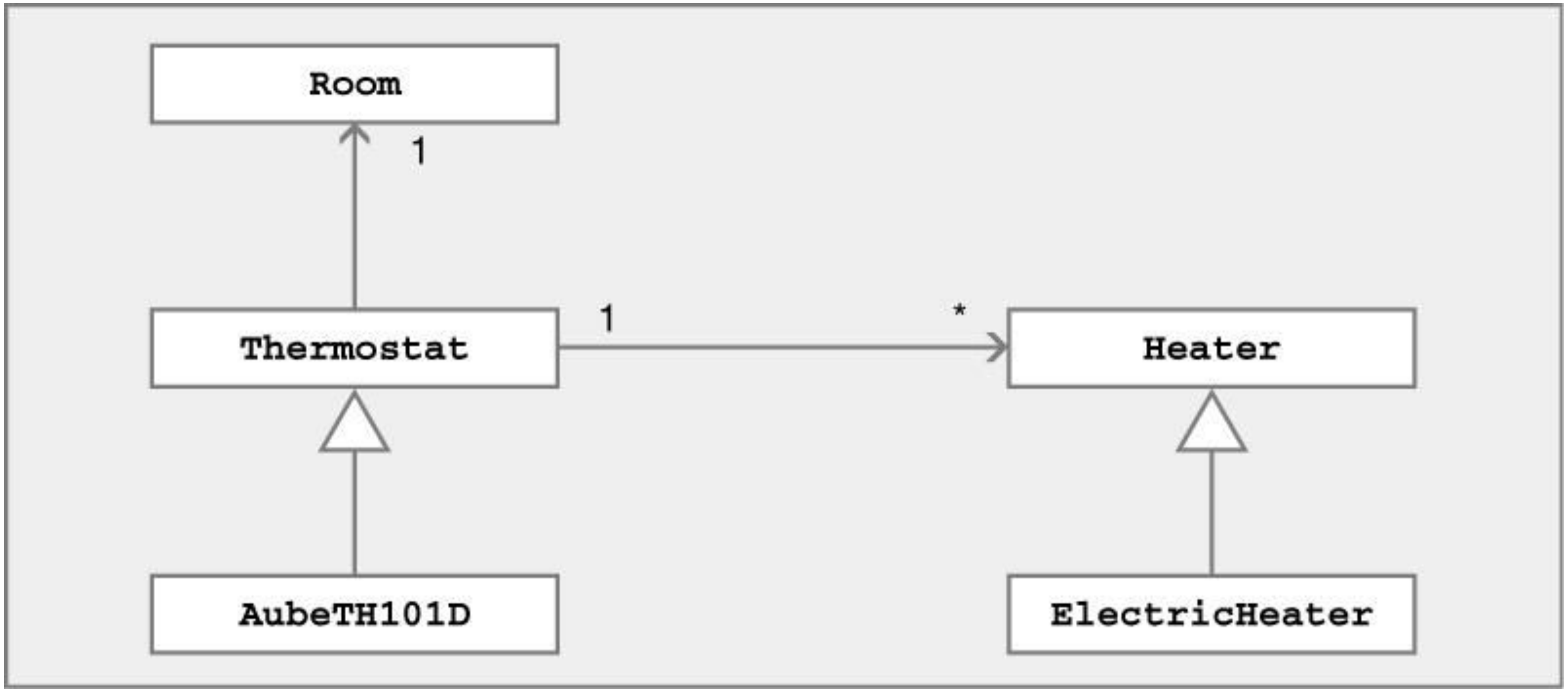
- 1 Bank associated with 0 or more Accounts
- Checking, Savings, MoneyMarket are Accounts

UML Example – Home Heating System

- **Try to read & understand UML diagram**



UML Example – Home Heating System



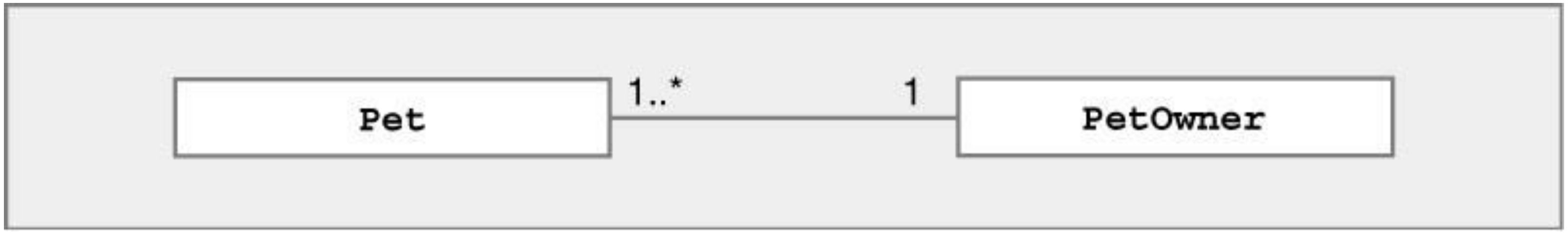
- Each Thermostat has 1 Room
- Each Thermostat associated with 0 or more Heaters
- ElectricHeater is a specialized Heater
- AubeTH101D is a specialized Thermostat

UML Class Diagrams ↔ Java

- Different representation of same information
  - Name, state, behavior of class
  - Relationship(s) between classes
- Practice deriving one from the other
  - Accurately depicting relationship between classes

UML → Java : Veterinary System

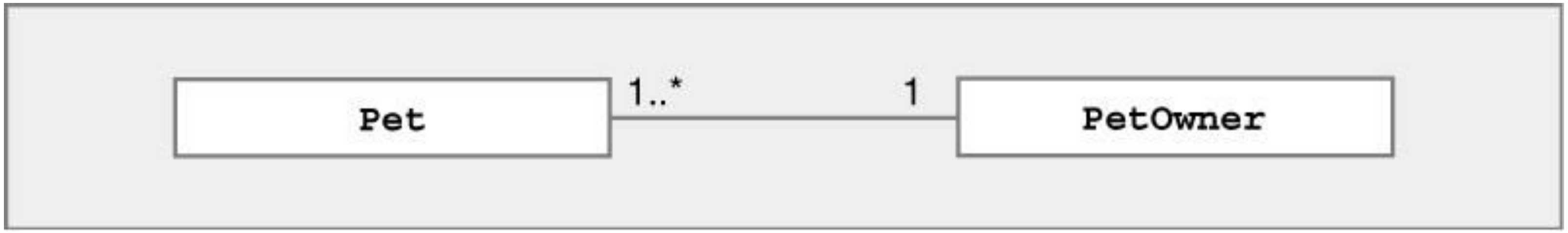
- UML



- Java

UML → Java : Veterinary System

- UML



- Java

```
class Pet {
   PetOwner myOwner;     // 1 owner for each pet
}
class PetOwner {
   Pet [ ] myPets;    // multiple pets for each owner
}
```

Java → UML : Veterinary System

- Java

```
class Pet {
   PetOwner myOwner;     // 1 owner for each pet
}
class PetOwner {
   Pet [ ] myPets;    // multiple pets for each owner
}
```

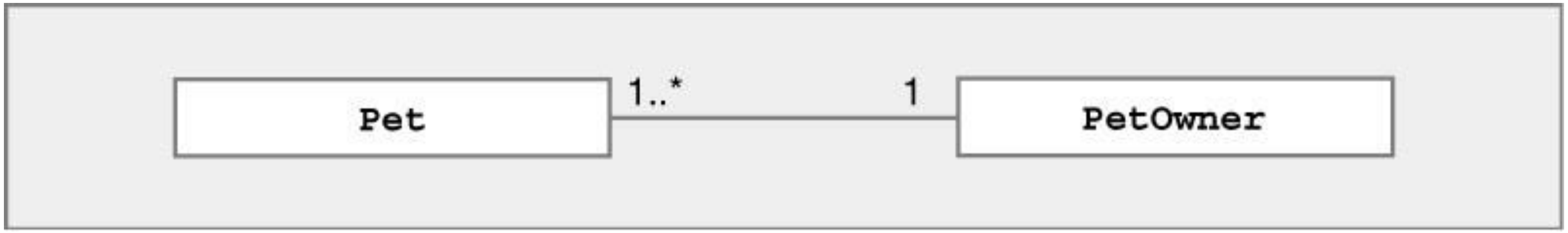
- UML

Java → UML : Veterinary System

- Java

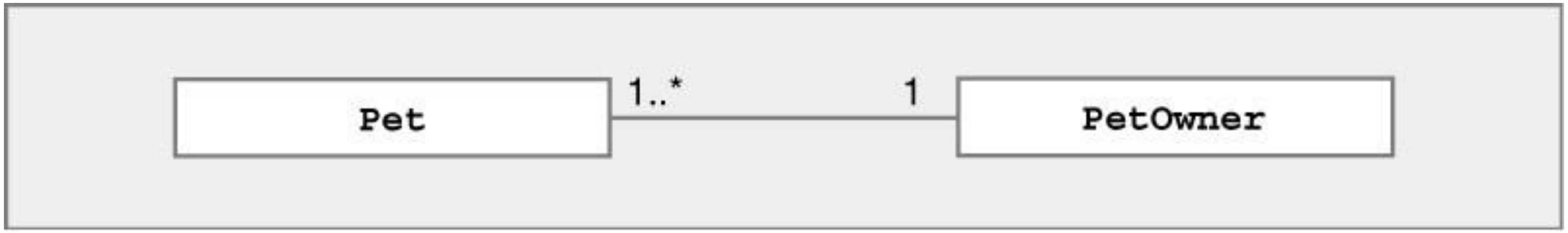
```
class Pet {
   PetOwner myOwner;     // 1 owner for each pet
}
class PetOwner {
   Pet [ ] myPets;     // multiple pets for each owner
}
```

- UML



UML Class Diagrams ↔ Java

- UML

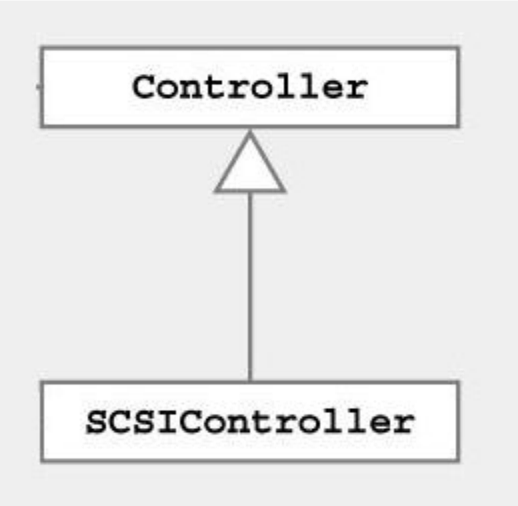


- Java

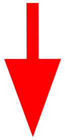
```
class Pet {
   PetOwner myOwner;     // 1 owner for each pet
}
class PetOwner {
   Pet [ ] myPets;    // multiple pets for each owner
}
```

UML → Java : Computer System

- UML

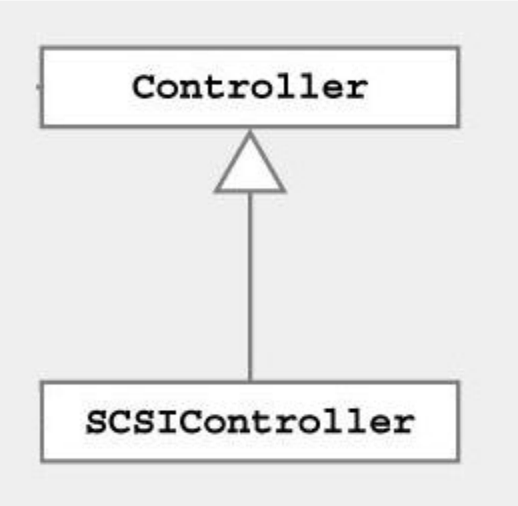


- Java



UML → Java : Computer System

- UML

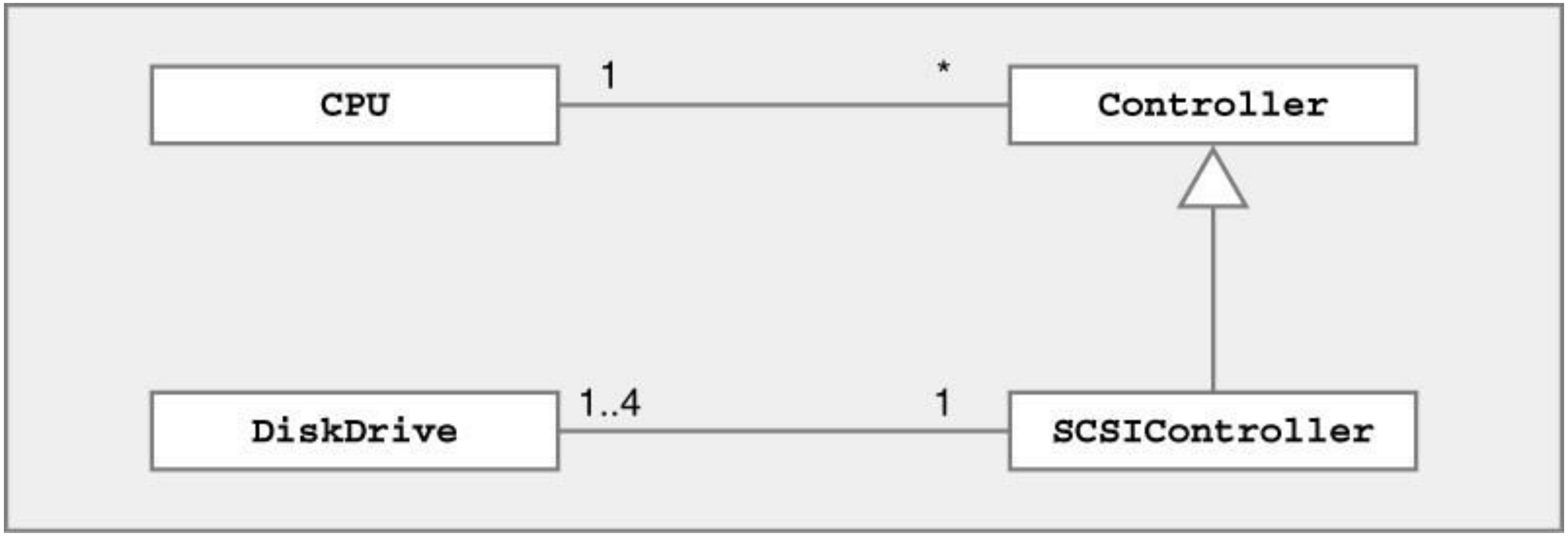


- Java

```
class Controller {
}
class SCSIController extends Controller {
}
```

UML → Java : Computer System

- UML

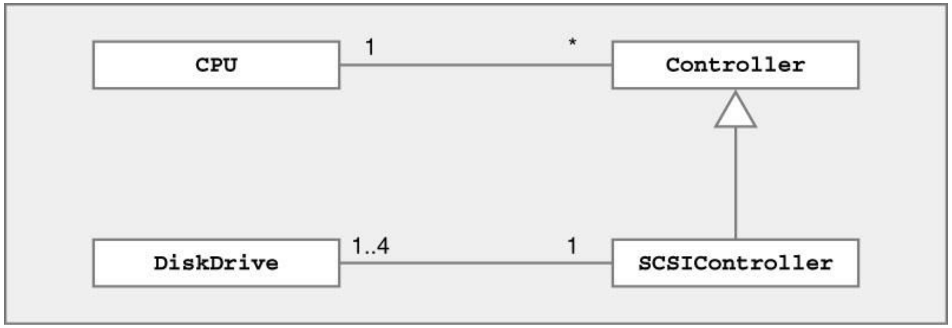


- Java
  - Design code using all available information in UML…

UML → Java : Computer System

- Java

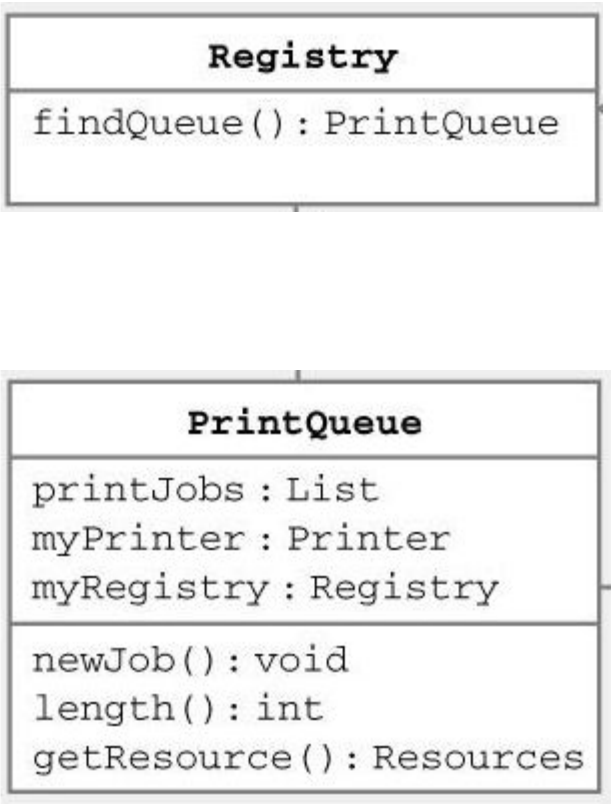
```
class CPU {
   Controller [ ] myCtlrs;
}
class Controller {
   CPU myCPU;
}
class SCSIController extends Controller {
   DiskDrive [ ] myDrives = new DiskDrive[4];
}
Class DiskDrive {
    SCSIController mySCSI;
}
```



Java → UML : Printing System

- Java

```
class Registry {
   PrintQueue findQueue();
}
class PrintQueue {
   List printJobs;
   Printer myPrinter;
   Registry myRegistry;
   void newJob();
   int length();
   Resources getResource();
}
```



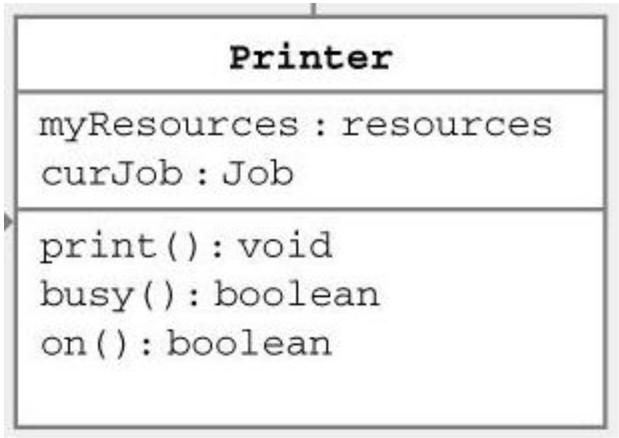
```
Registry
---------------------------
findQueue() : PrintQueue
```

```
PrintQueue
---------------------------
printJobs : List
myPrinter : Printer
myRegistry : Registry
---------------------------
newJob() : void
length() : int
getResource() : Resources
```

Java → UML : Printing System

- Java

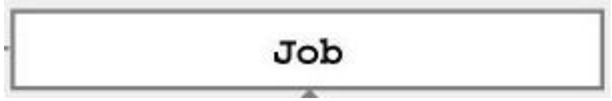
```
Class Printer {
   Resources myResources;
   Job curJob;
   void print();
   boolean busy();
   boolean on();
}
class Job {
   Job(Registry r) {
     …
   }
}
```



Printer

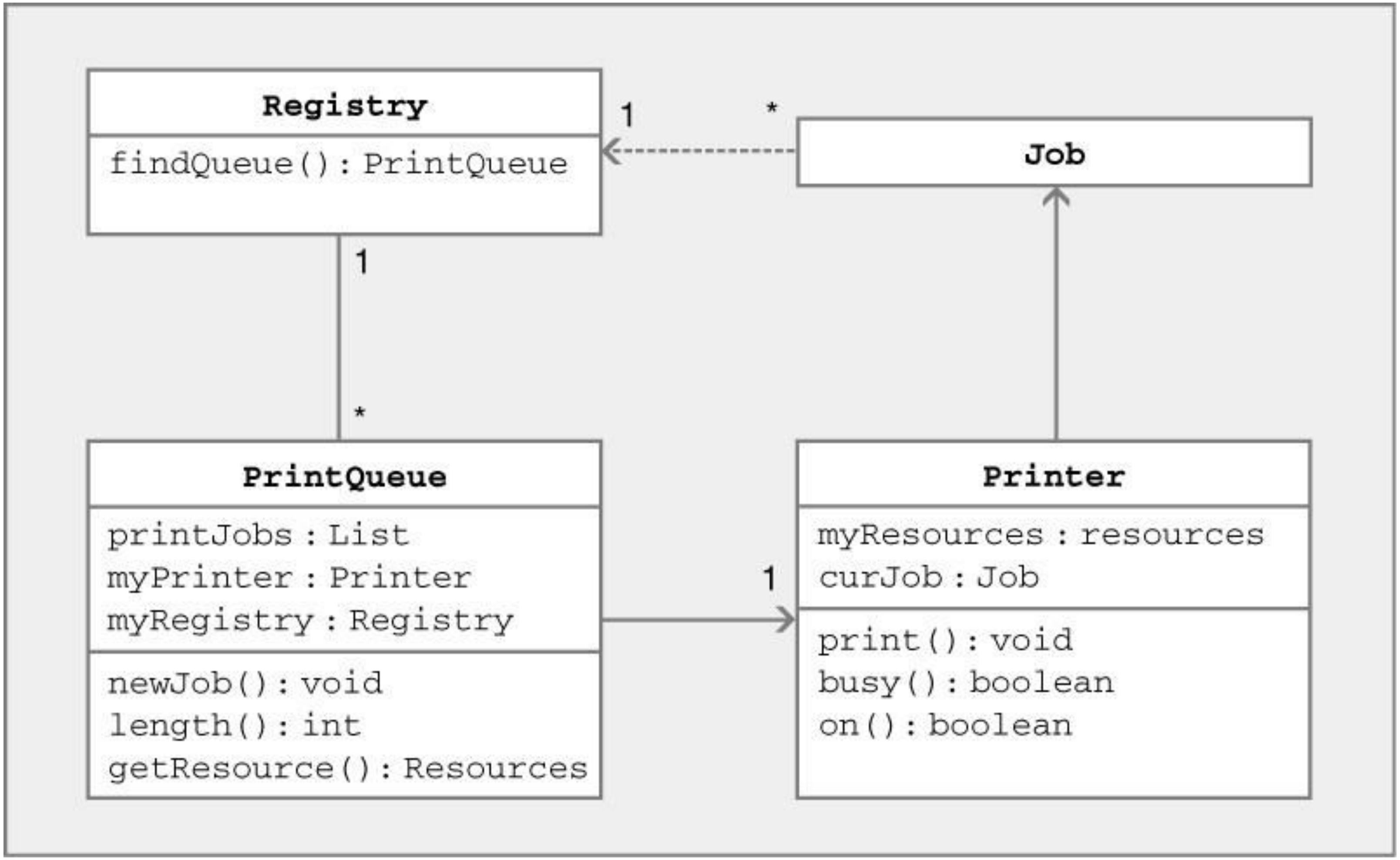
myResources : resources
curJob : Job

print() : void
busy() : boolean
on() : boolean



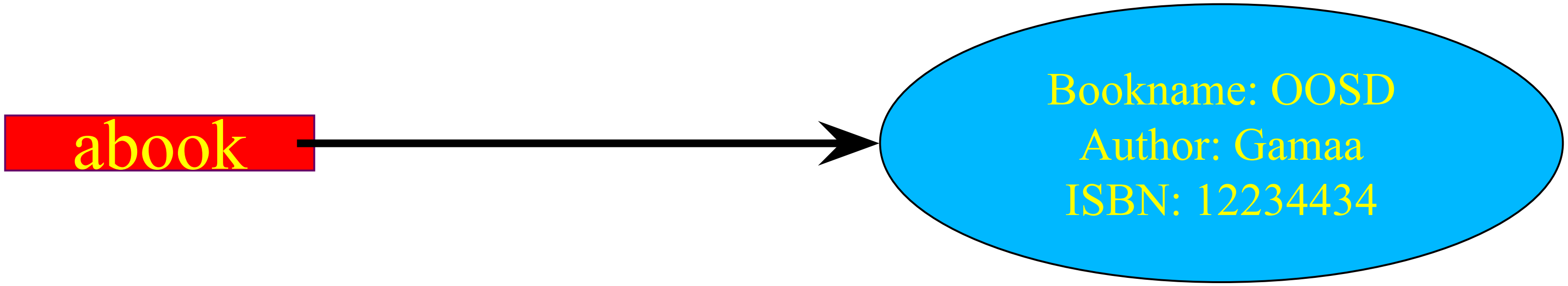
Job

Java → UML : Printing System



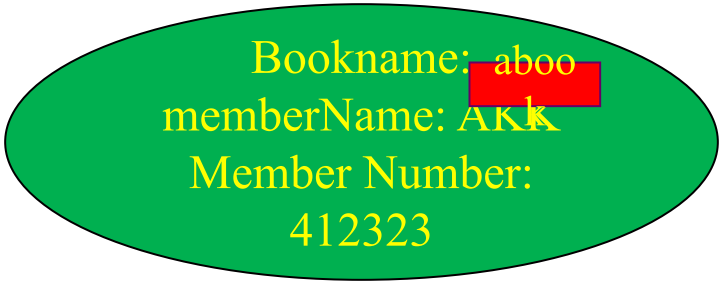
Implementing Association Relationship: Example 1

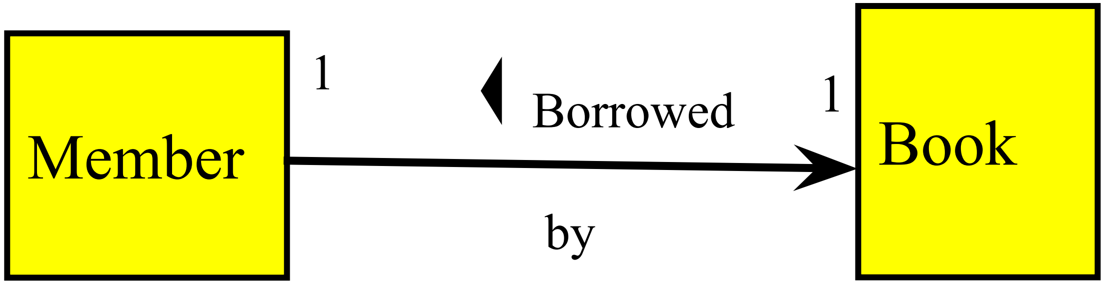
- To implement in Java:
  - Use a reference variable of one class as an attribute of another class



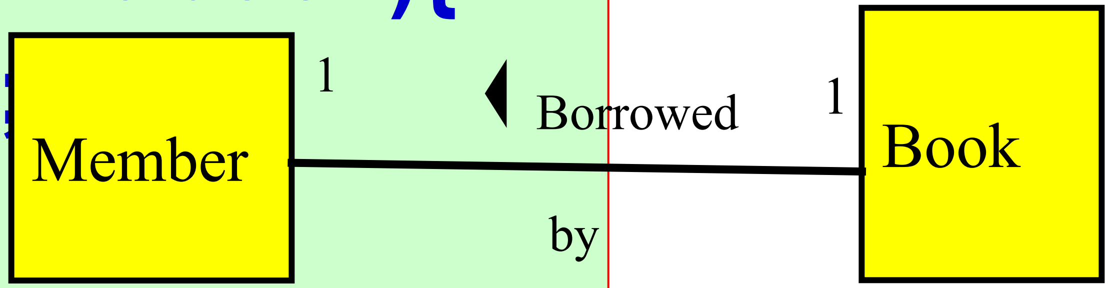
Book Reference

Book instance

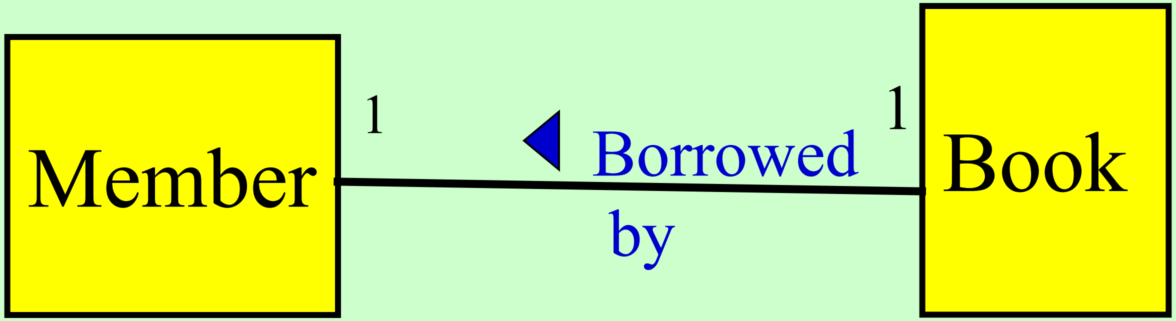




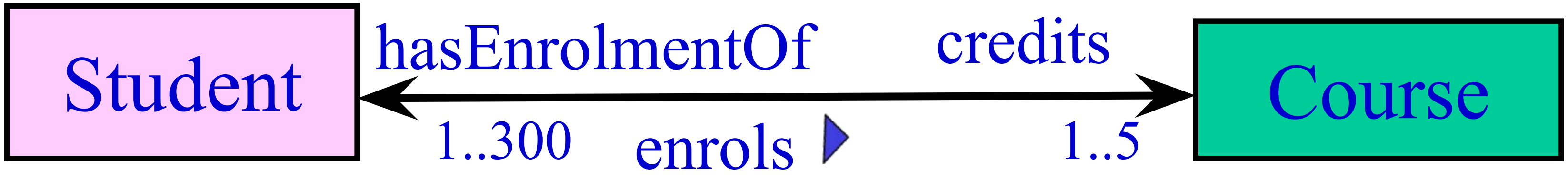
```
public class Member{
private Book book;
public issueBook(Book
  abook){
    setBook(abook);
    abook.setLender(this);
    }
    setBook(Book abook){
    book=abook;
  }
  ...
}
```



```
public class Book{
private Member member;
setLender(Member aLender){
    member=aLender;
  }
  ...
}
```



Association Implementation: Example 2

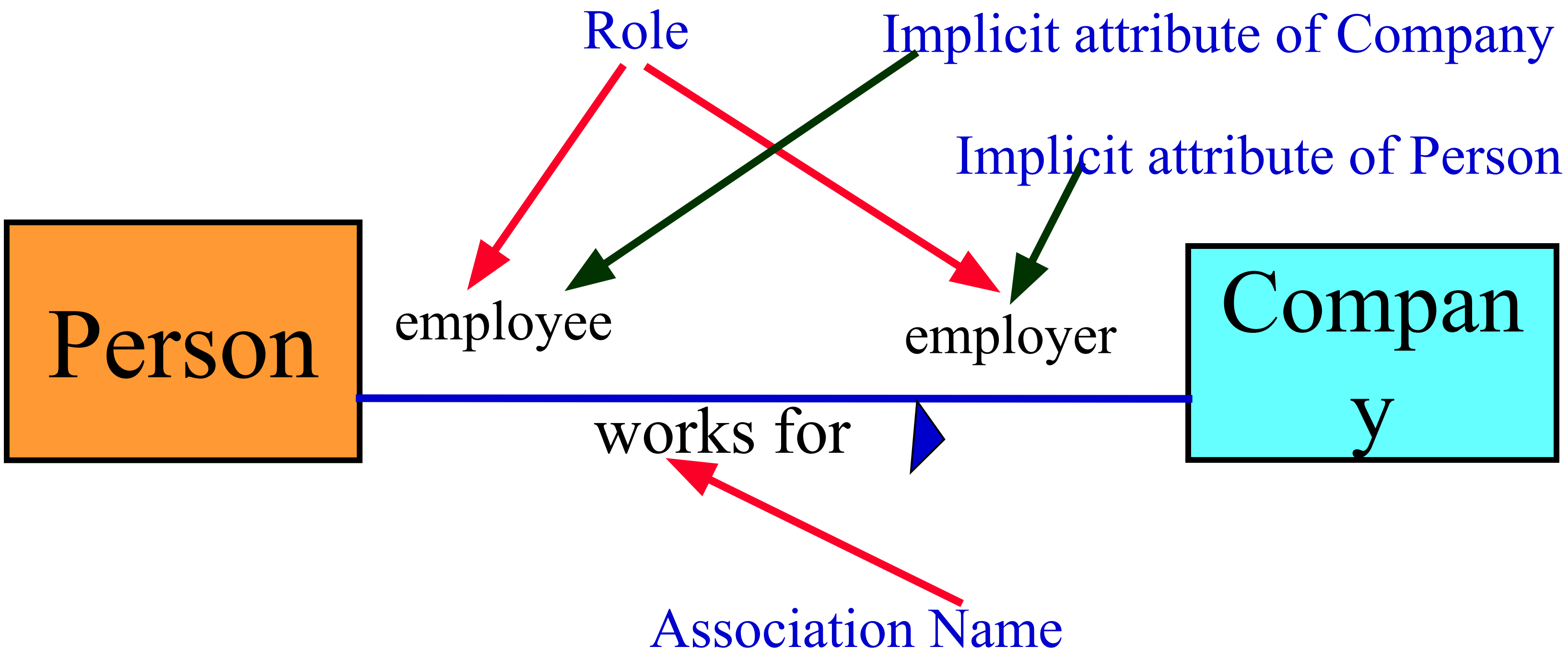


```
Class Student {
    Course credits[5];
    …
}
Class Course {
   Student hasEnrolmentOf[300];
   …
}
```

Observe the Navigation

Association Example 2

- **A Person works for a Company.**



Observe: Implicit bidirectional navigation

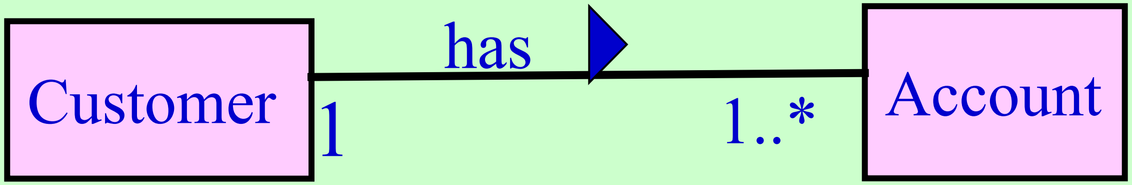
Implementation?

Example 2 Implementation

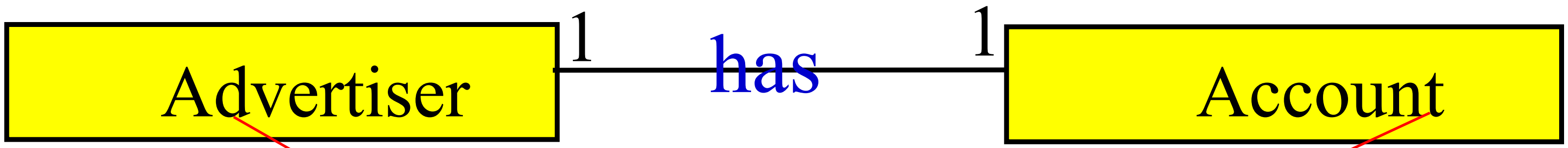
```
 public class Company {
    private Person employee;
    public void setCompany(Person p){ employee=p;}
 }
public class Person {
  private Company employer;
  public Company getWorksFor() {
    return employer;
  }
 public void setWorksFor(Company c) {
    employer=c;
  }
```

Code for Association Multiplicity

```
class Customer{

      private ArrayList <Account> accounts =
            new ArrayList<Account>();

   public Customer() {

      Account defaultAccount = new Account();

        accounts.add(defaultAccount);

    }

}
```



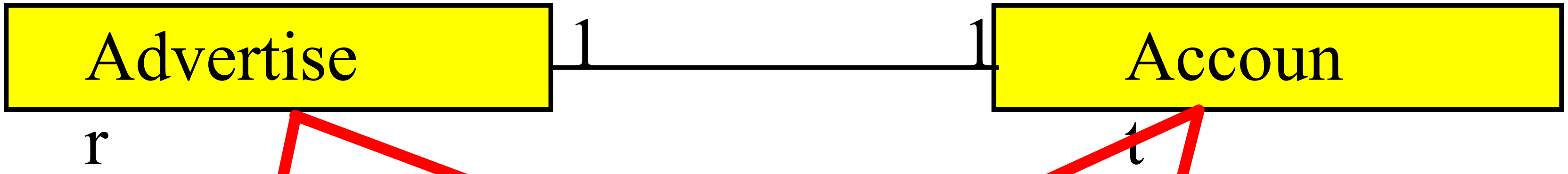
1-1 Association Example 3



```
public class Advertiser {
    private Account account;
    public Advertiser() {
        account = new Account(this);
    }
    public Account getAccount() {
        return account;
    }
}
```

Now,
Write
code for

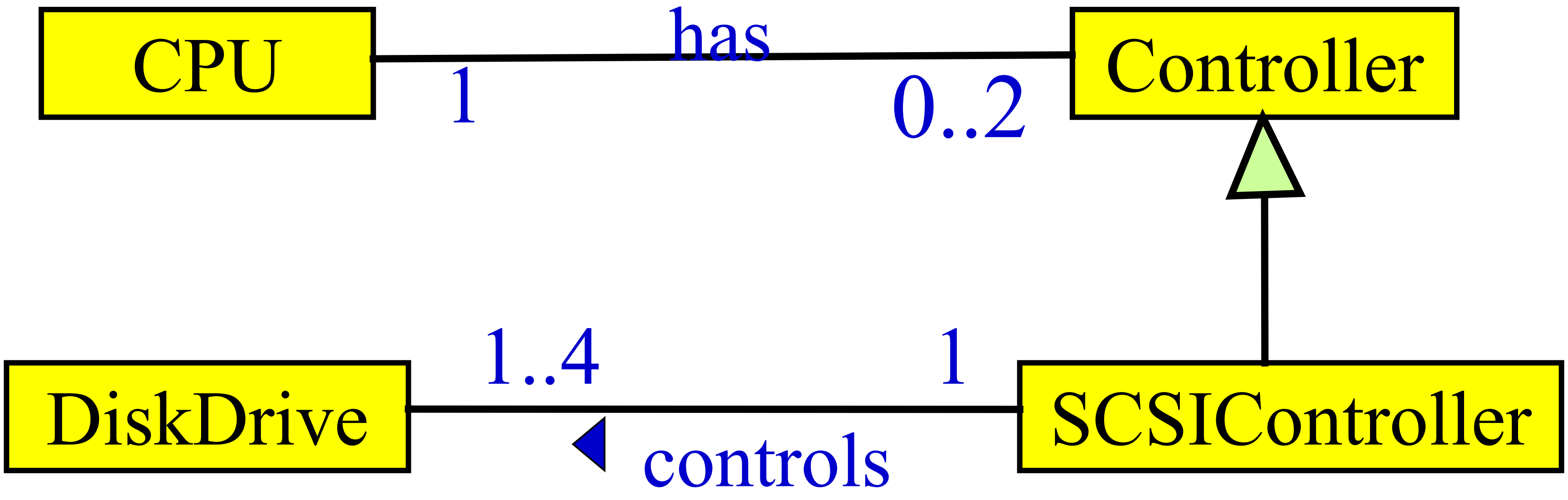
1-1 Association



```
public class Advertiser {
  private Account account;
  public Advertiser() {
    account = new
  Account(this);
  }
  public Account
  getAccount() {
    return account;
```

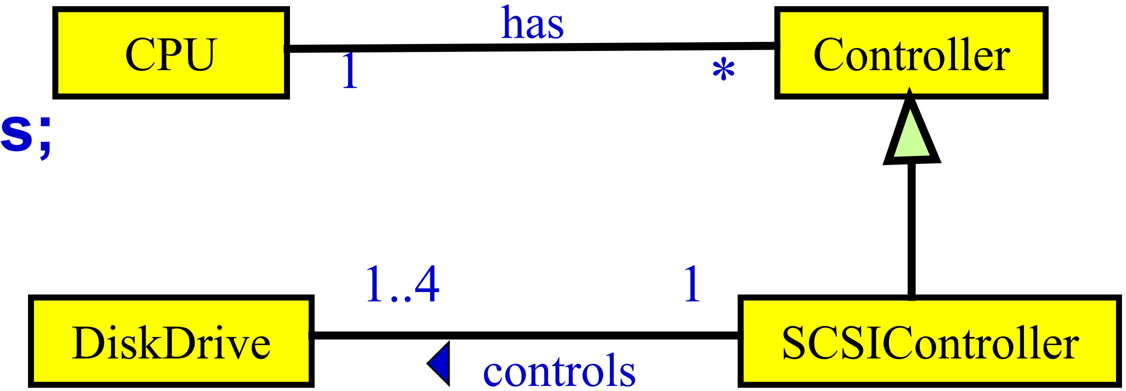
```
public class Account {
     private Advertiser owner;
   publicAccount(Advertiser
   owner) {
     this.owner = owner;
   }
   public Advertiser getOwner()
   {
     return owner;
   }
}
```

Quiz: Read and understand UML class diagram



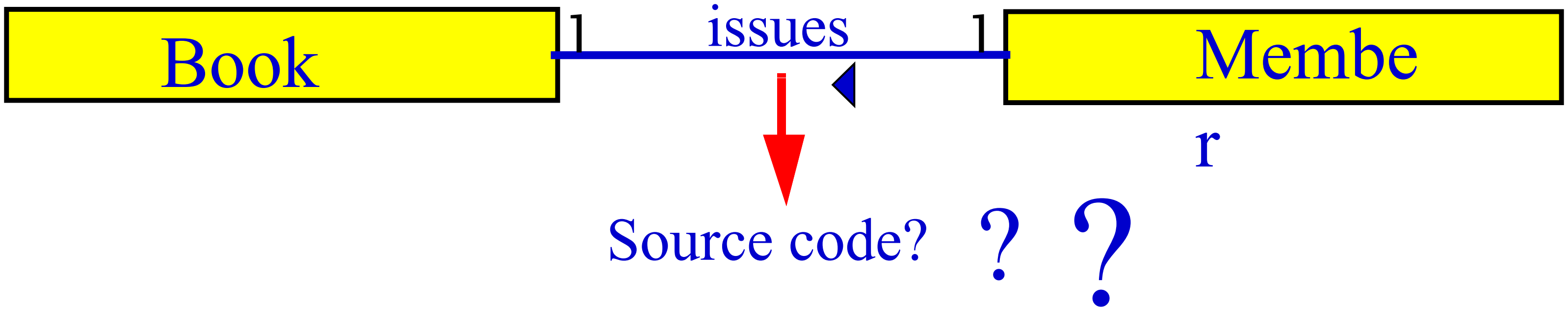
- 1 CPU has 0 to two Controllers
- 1-4 DiskDrives controlled by 1 SCSIController
- SCSIController is a (specialized) Controller

Java Code?



```
class CPU {
   Controller [ ] myCtlrs;
}
class Controller {
   CPU myCPU;
}
class SCSIController extends Controller {
   DiskDrive [ ] myDrives = new DiskDrive[4];
}
Class DiskDrive {
    SCSIController mySCSI;
}
```

Quiz 1: Write Java Code



Book

1

issues

1

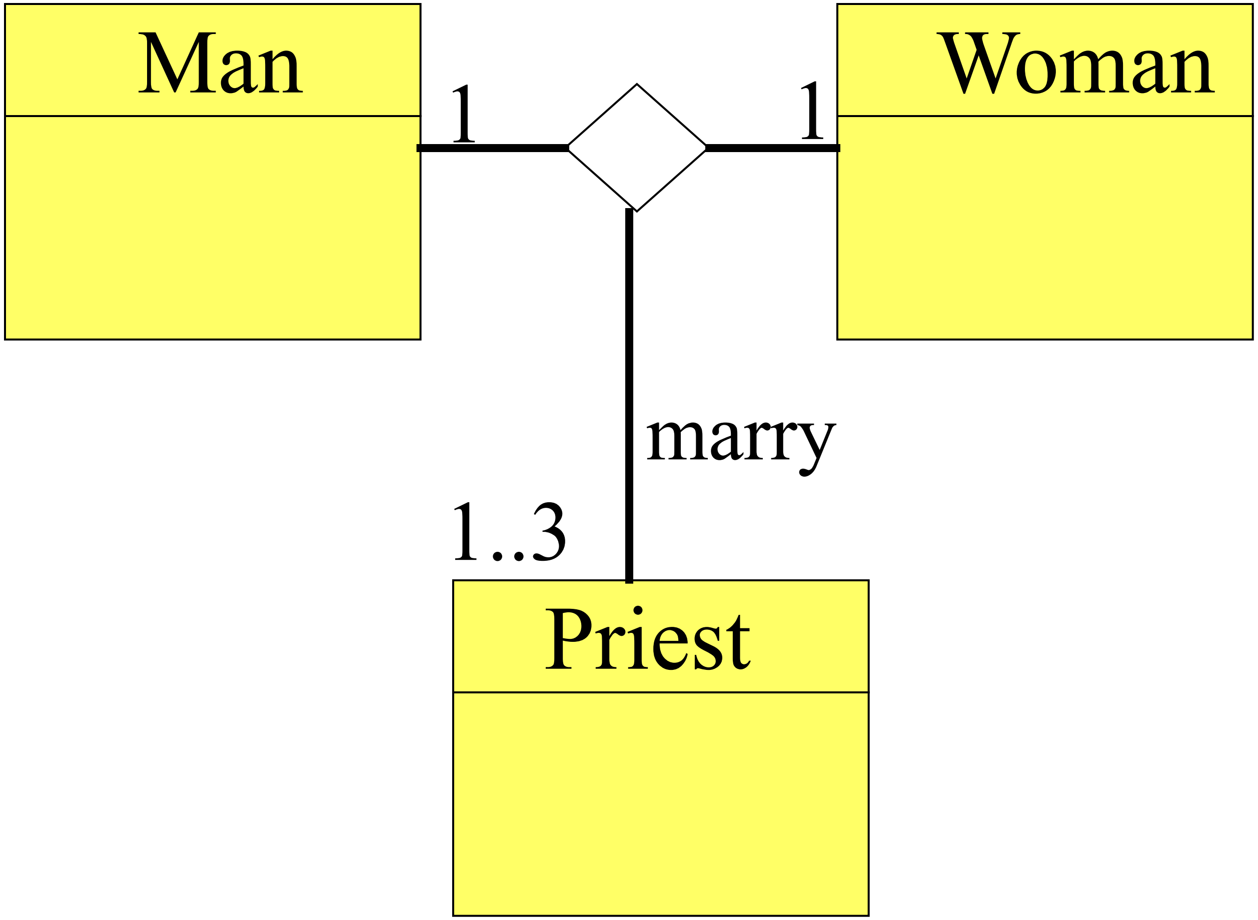
Membe
r

Source code? ? ?

Quiz 2: Draw UML Class Diagram

```
public class TreeMap {
  TreeMapNode topNode = null;
  public void add(Comparable key, Object value) {…}
  public Object get(Comparable key) {…}
}

class TreeMapNode {
  private Comparable itsKey;
  private Object itsValue;
  private TreeMapNode nodes[] = new
TreeMapNode[2];

  public TreeMapNode(Comparable key, Object
value) {…}
```

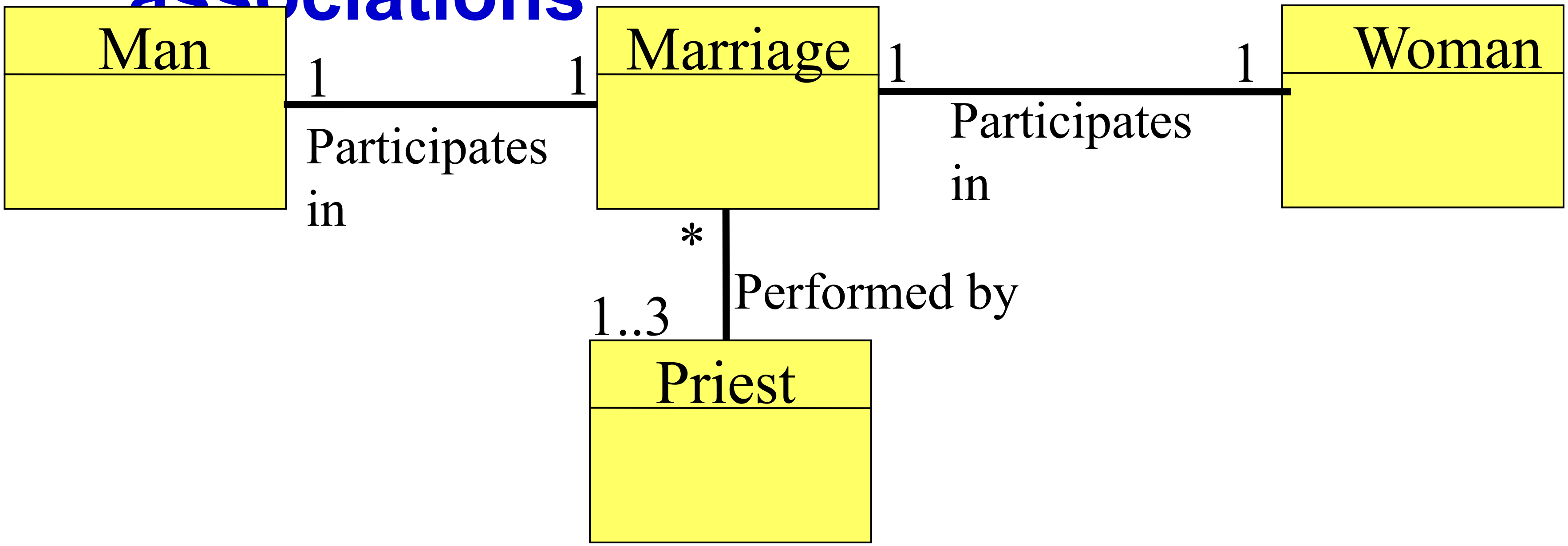
Ternary Association

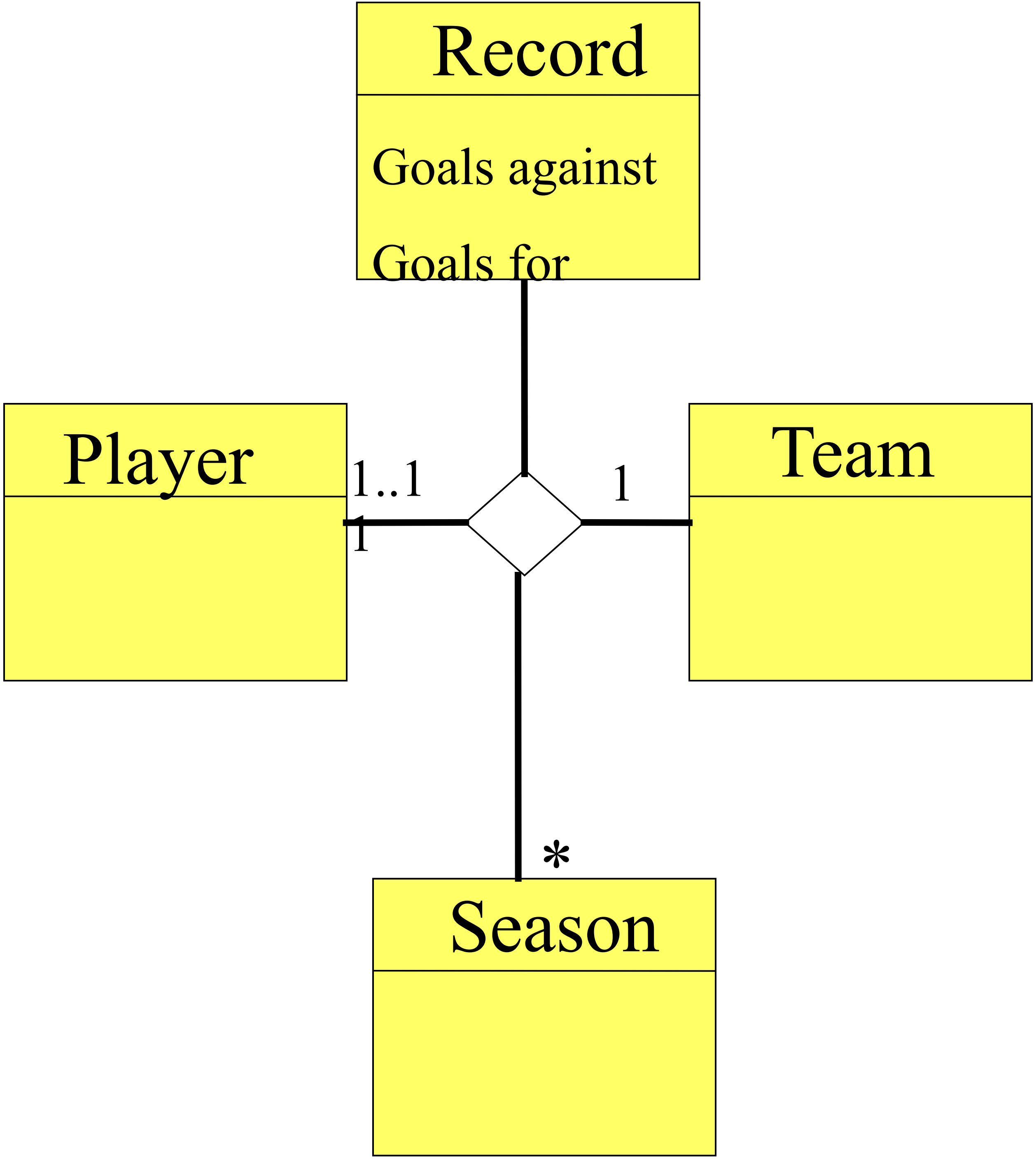


and we can add more classes to the diamond…

Implementation of Ternary Association

- There are several ways in which ternary association can be implemented.
  - One is to decompose it to a set of binary associations





Record

Goals against

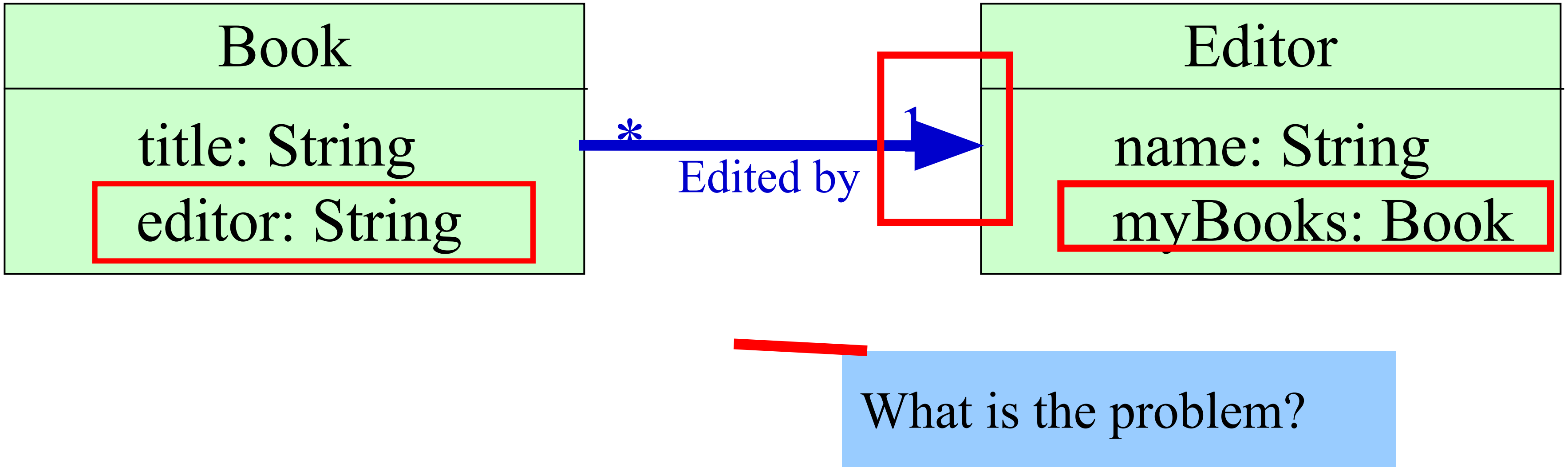
Goals for

Player

Team

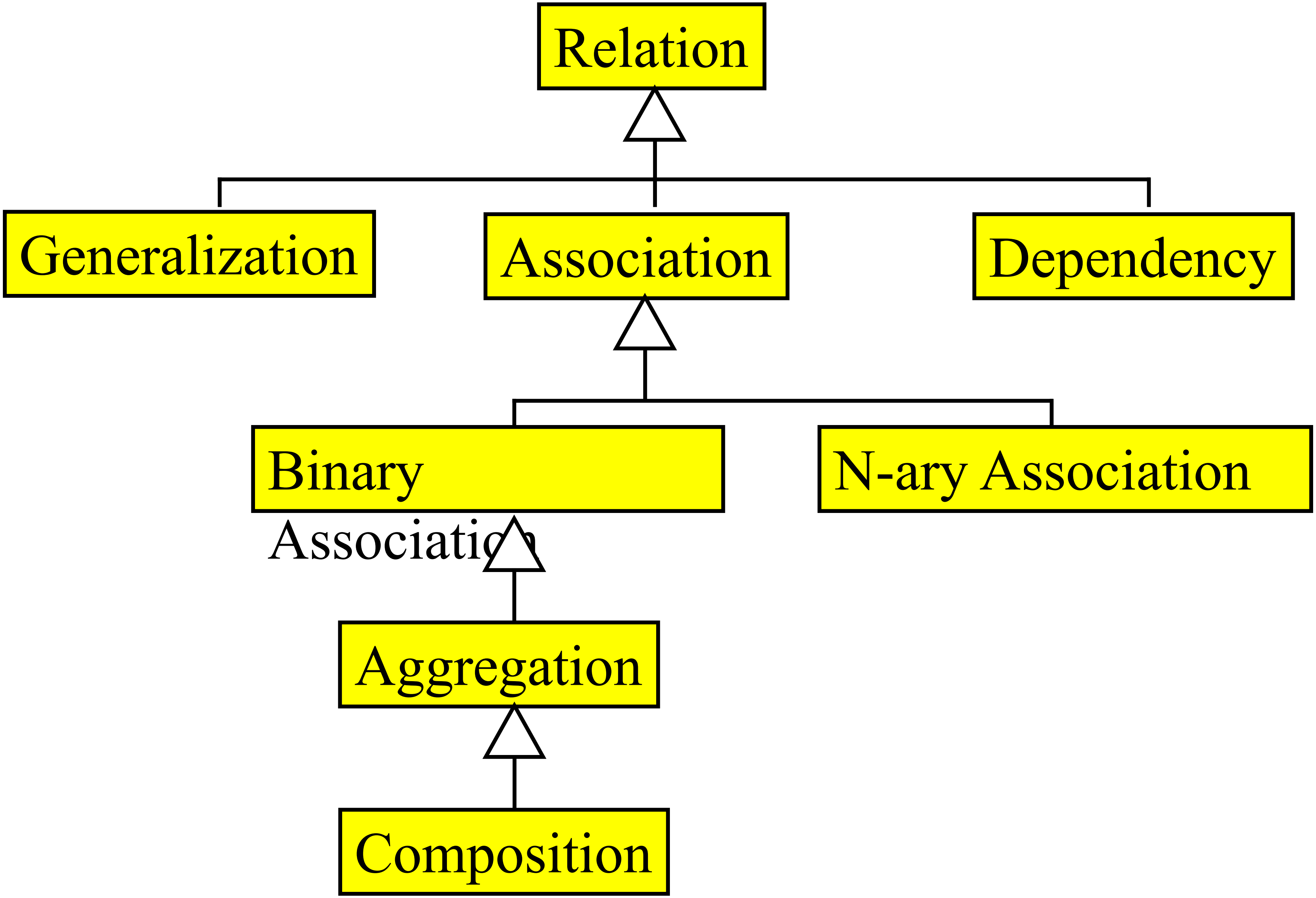
Season

Association Quiz



- **Association denoted by symbol not attributes.**
- **Implementation (pointers, arrays, vectors, ids etc) is left to the detailed design phase.**
- **Wrong arrow type**

Types of Class Relationships



Relation

Generalization

Association

Dependency

Binary Association

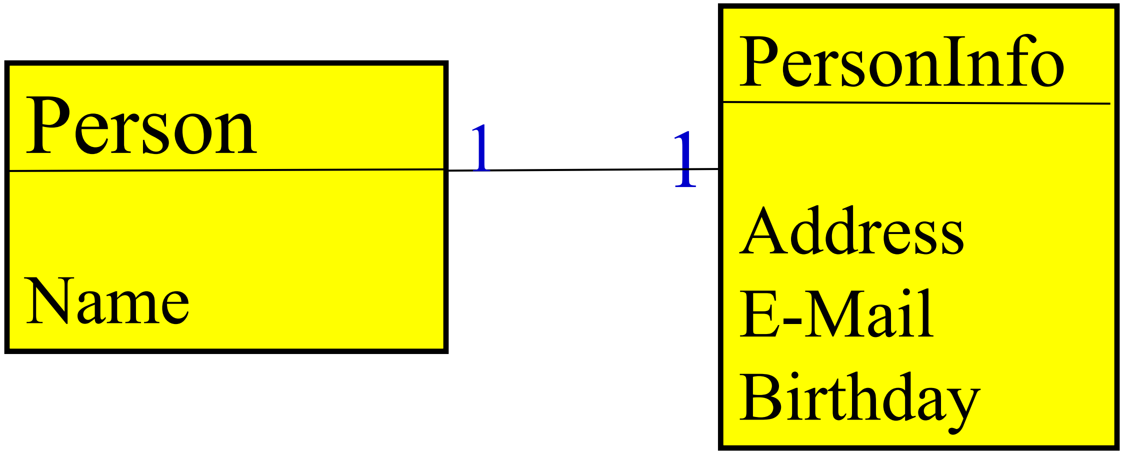
N-ary Association

Aggregation

Composition

Overdoing Associations

- Avoid unnecessary Associations



Person

Name

1

1

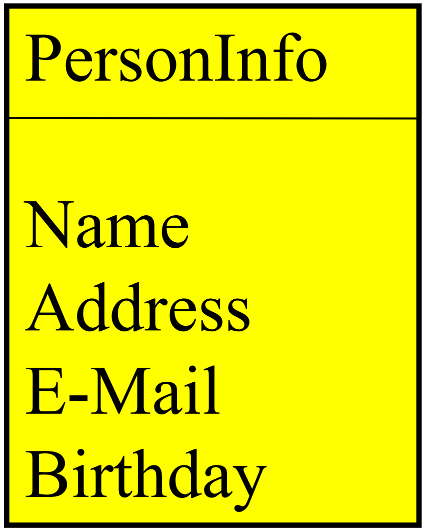
PersonInfo

Address

E-Mail

Birthday

Avoid This



PersonInfo

Name

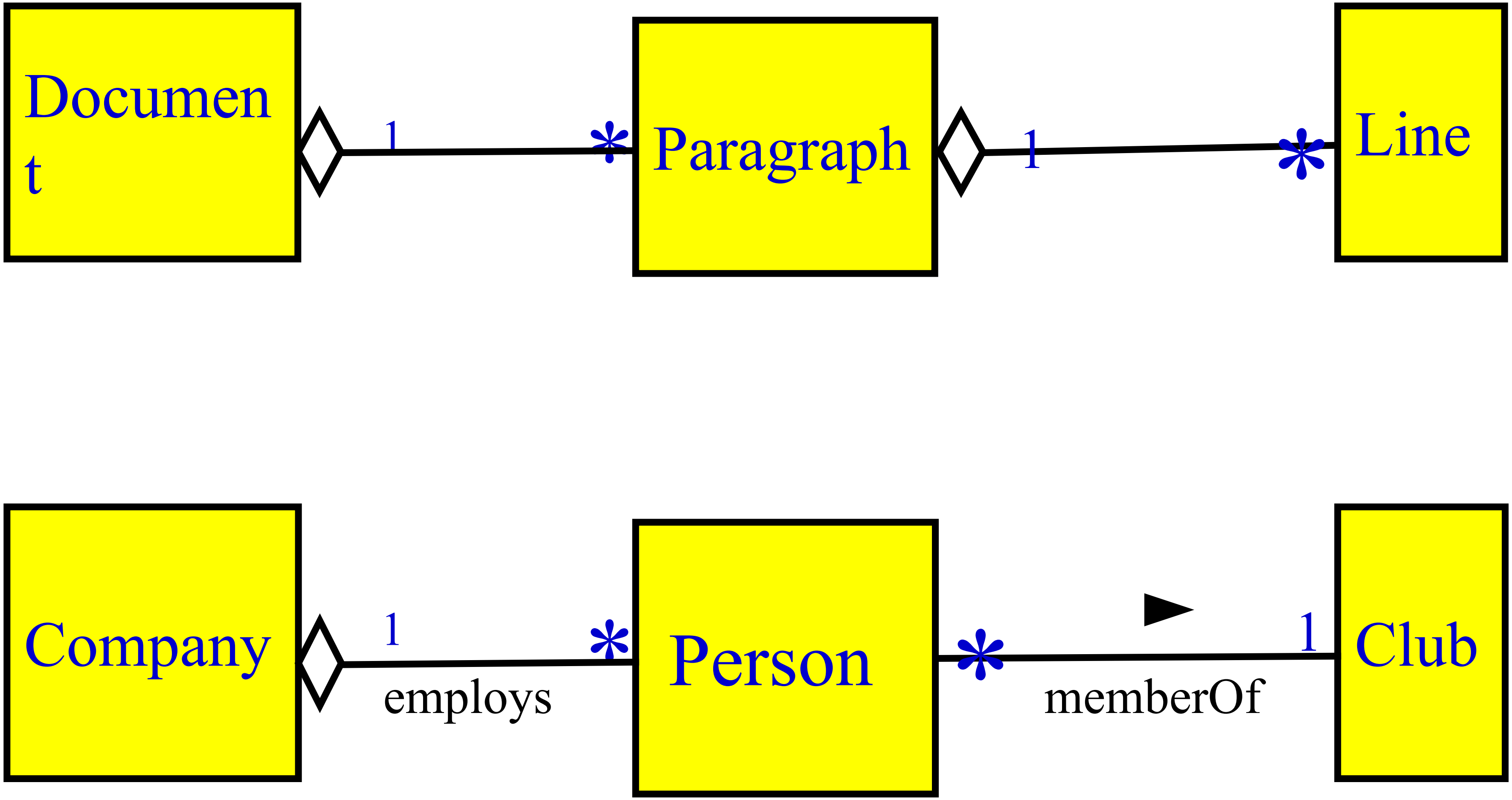
Address

E-Mail

Birthday

Do This

Aggregation Relationship



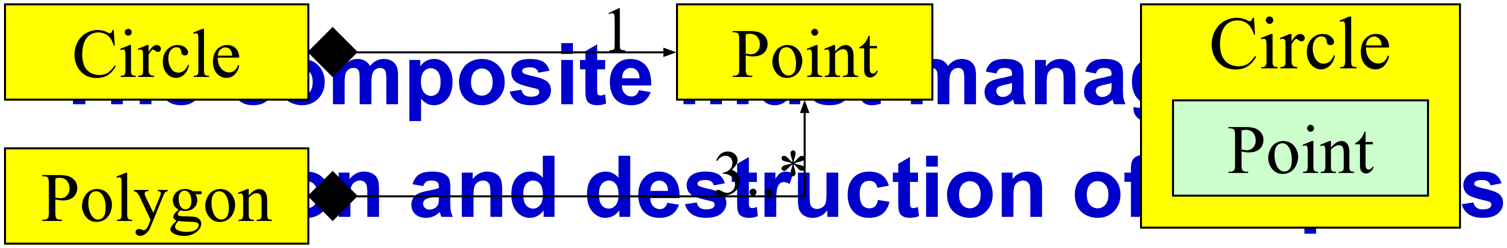
Aggregation

cont…

- An aggregate object contains other objects.
- Aggregation limited to **tree hierarchy**:
  - **No circular inclusion relation.**

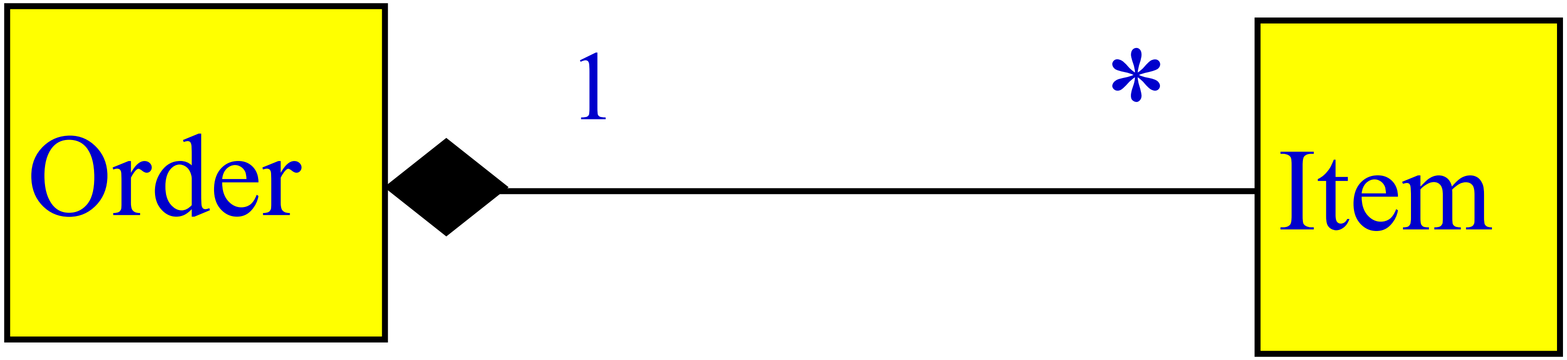
Composition

- A stronger form of aggregation
  - The whole is the sole owner of its part.
    - A component can belong to only one whole
  - The life time of the part is dependent upon the whole.
  - mposite manag
    n and destruction of s.



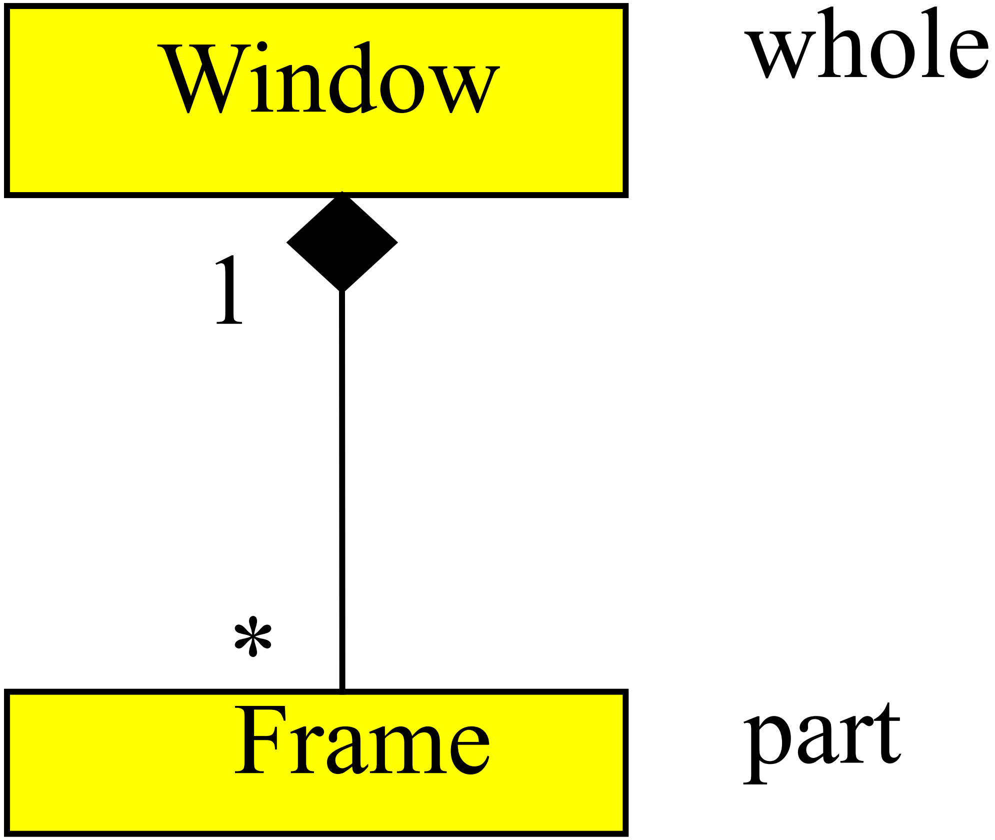
Composition Relationship

- Life of item is same as the order



Composition

- An object may be a part of ONLY one composite at a time.
  - Whole is responsible for the disposition of its parts.

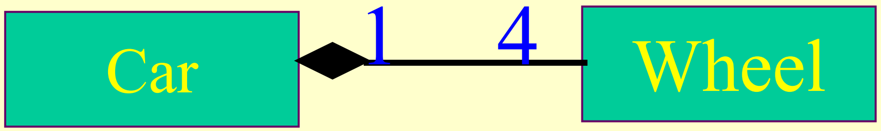


Aggregation vs. Composition

- Composition:
  - Composite and components have the same life.
- Aggregation:
  - Lifelines are different.
- Consider an order object:
  - Aggregation: If order items can be changed or deleted after placing the order.

Implementing Composition

```
public class Car{
  private Wheel wheels[4];
  public Car (){
        wheels[0] = new Wheel();
        wheels[1] = new Wheel();;
            wheels[2] = new Wheel();;
        wheels[3] = new Wheel();;
      }
}
```



Summary

- Focus: Class diagrams
  - Contents of a class
  - Relationship between classes
- You should be able to
  - Draw UML class diagram given code
  - Write code given UML class diagram