CptS 322- Software Engineering Principles I

Class Level Design
and
UML Class Diagrams

Instructor: Sakire Arslan Ay

Fall 2023

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Outline

- Designing classes
- UML class diagrams
  - Syntax and semantics
- Examples

Software Design

- Class Level Design:
  - specifying the structure of how a software system will be written and function, without actually writing the complete implementation
  - A transition from "what" the system must do, to "how" the system will do it
    - What classes will we need to implement a system that meets our requirements?
    - What fields and methods will each class have?
    - How will the classes interact with each other?

Example: Identify classes from requirements

Library Example:

- The library contains books and journals.
  - It may have several copies of a given book.
- Library members can borrow books from the library. Library staff are assumed to be library members by default.
  - Library members can normally borrow up to six items at a time,
  - But library staff may borrow up to 12 items at a time.
  - Only library staff may borrow journals.
- The system must keep track of when books and journals are borrowed and returned.

Describing designs with UML diagrams

- Class diagram (now)
  - Shows classes and relationships among them.
  - A static view of the system, displaying what classes interact but not what happens when they do interact.
- Sequence diagram
  - A dynamic view of the system, describing how objects collaborate: what messages are sent and when.

Uses for UML

- As a sketch: to communicate aspects of system
  - Forward design: doing UML before coding
    - Used to get rough selective ideas
    - Often done on whiteboard or paper
  - Backward design: doing UML after coding as documentation
- As a blueprint: a complete design to be implemented
  - Sometimes done with CASE (Computer-Aided Software Engineering) tools
- As a programming language: with the right tools, code can be auto-generated and executed from UML
  - Only good if this is faster than coding in a "real" language

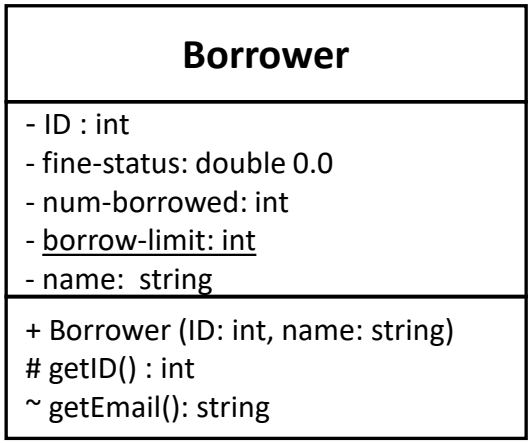
UML Class Diagrams

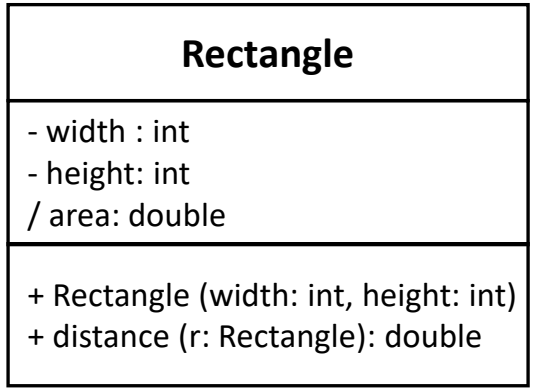
- **UML class diagram** is a visualization of:
  - the classes in an OO system,
  - their fields and methods, and
  - connections between the classes that interact or inherit from each other

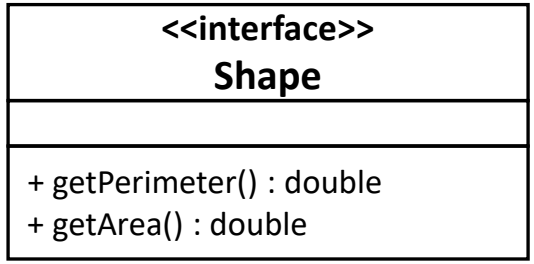
- What are some things that are not represented in a UML class diagram?
  - Details of how the classes interact with each other, and
  - algorithmic details; how a particular behavior is implemented.

UML: Class attributes

- **class name** in top of box
  - if class is defined as an interface include <<interface>> on top
  - use *italics* for an abstract class name
- attributes (optional)
  - should include all fields of the class
- operations / methods (optional)
  - may exclude trivial (get/set) methods
  - but don't omit any methods from an interface!
  - should not include inherited methods

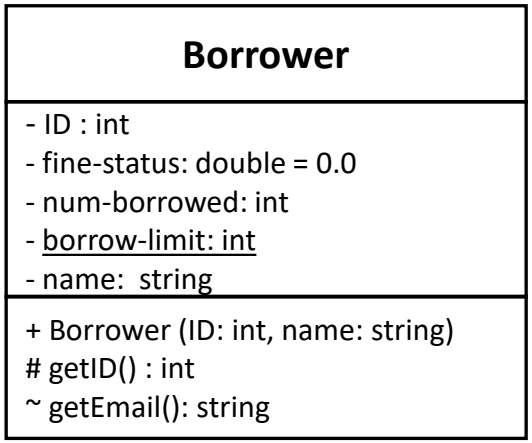


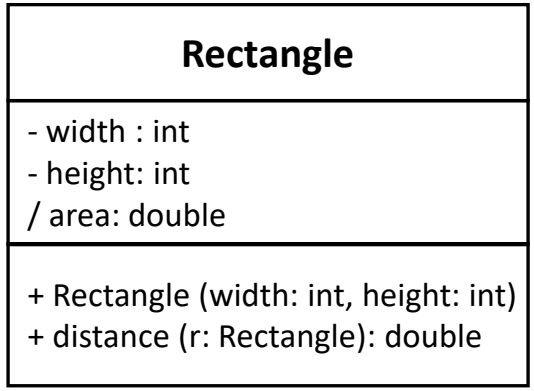


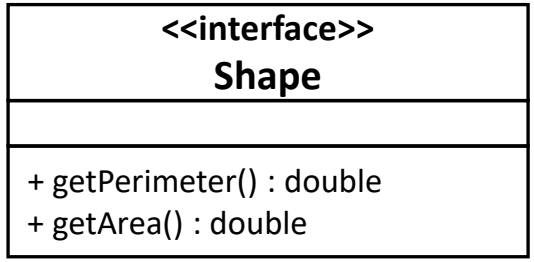


UML: Class attributes

- attributes (fields, instance variables)
  <visibility> <name> : <type> [count] = <default_value>
  - visibility: + public
    # protected
    - private
    / derived
    ~ package (visible within)
  - underline static attributes
  - derived attribute: not stored, but can be computed from other attribute values
  - attribute example:
    `- fine-status : double = 0.00`





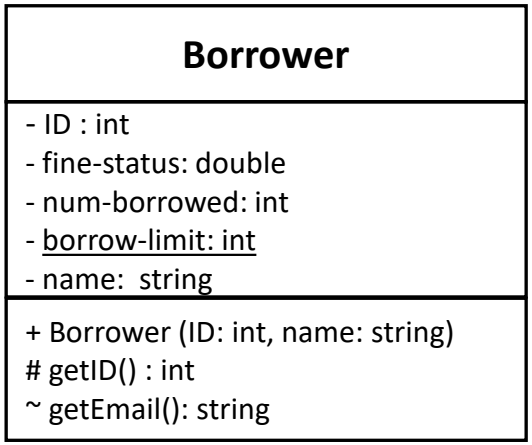


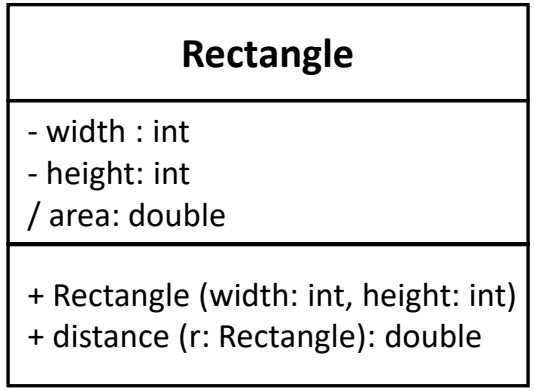
UML: Class methods

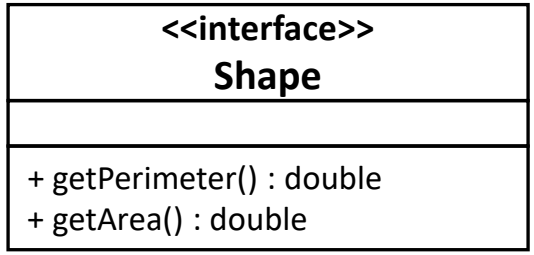
- operations / methods

  <visibility> <name> (parameters) : <return_type>

  - visibility: + public
    # protected
    - private
    ~ package (visible within)
  - underline static methods
  - parameter types listed as (name: type)
  - omit <return_type> on constructors and when return type is void
  - method example:
    `+ distance(r:Rectangle): double`

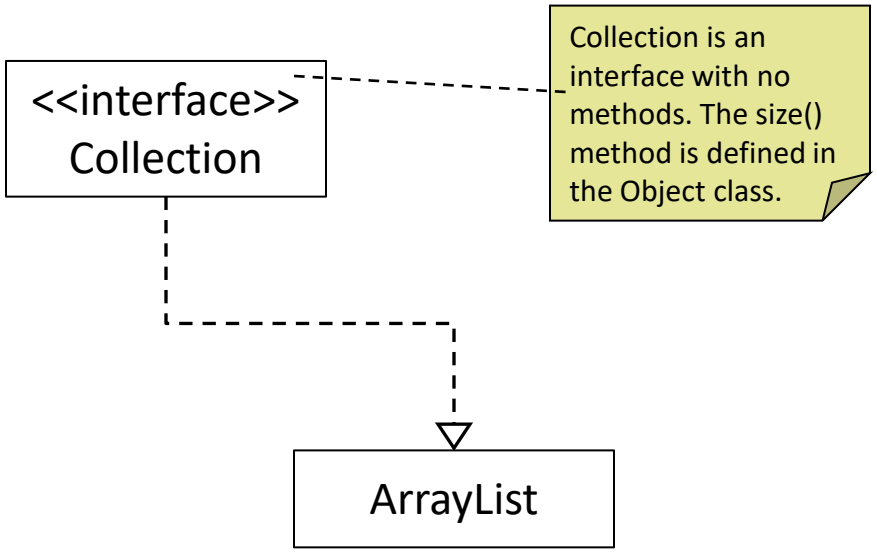






UML: Comments

- Represented as a folded note, attached to the appropriate class/method/etc. by a dashed line

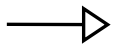
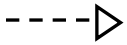


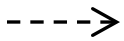
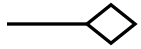
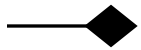
UML: Relationships between classes

1.

2.

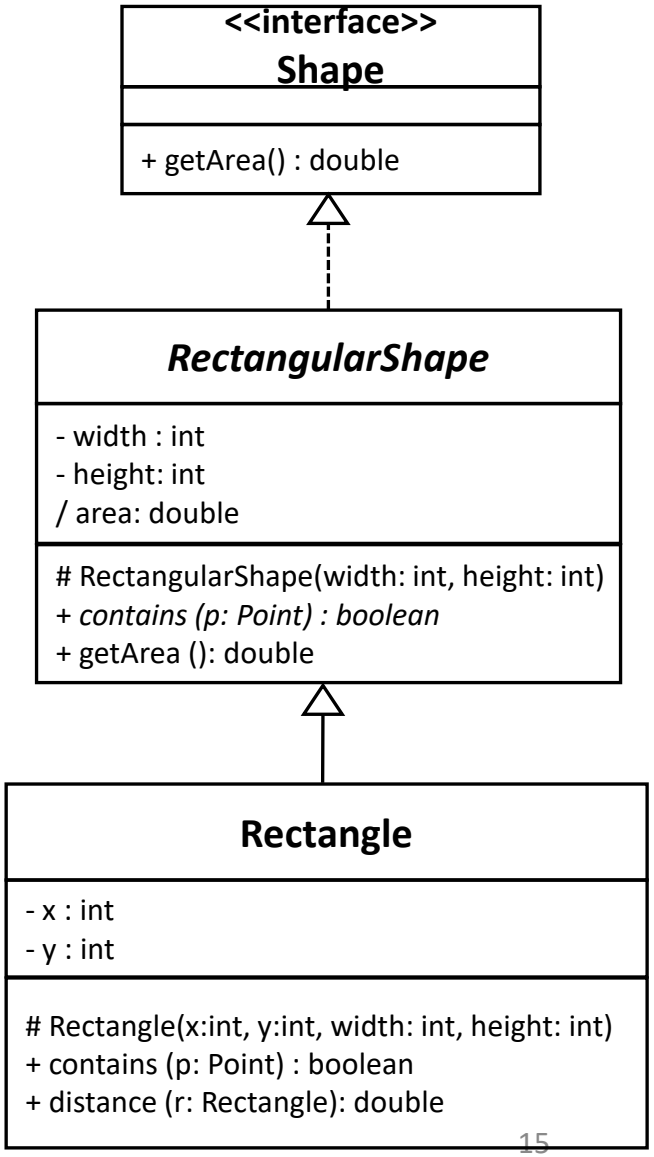
UML: Relationships between classes

1. **Generalization**: an inheritance relationship
   - inheritance between classes 
   - interface implementation 

2. **Association**: a usage relationship
   - simple association 
   - dependency 
   - aggregation 
   - composition 

UML: Generalization relationships

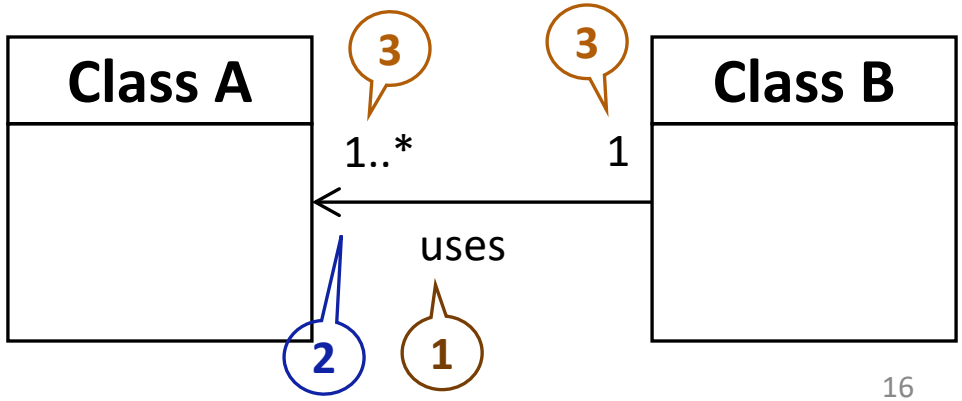
- Generalization (inheritance) relationships
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - Class, abstract class : solid line, white triangle arrow
    - interface: dashed line, white triangle arrow
  - we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent



Associational (usage) relationships

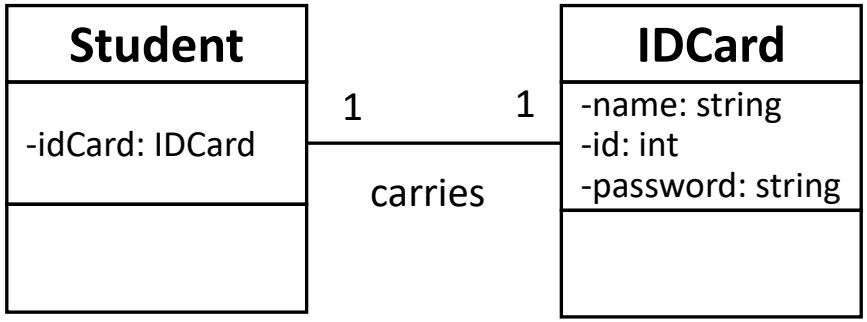
- Associational (usage) relationships

1. name (what relationship the objects have)
2. navigability (direction)
3. multiplicity (how many are used)
   - * ⇒ 0, 1, or more
   - 1 ⇒ 1 exactly
   - 2..4 ⇒ between 2 and 4, inclusive
   - 3..* ⇒ 3 or more

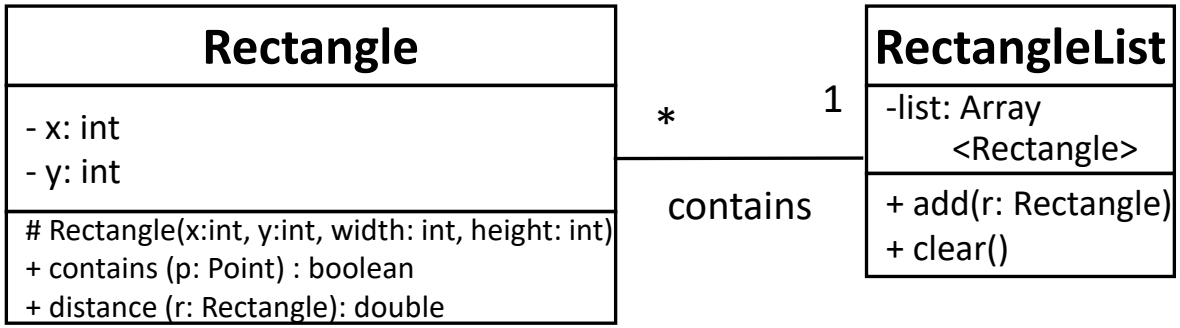


Multiplicity of associations

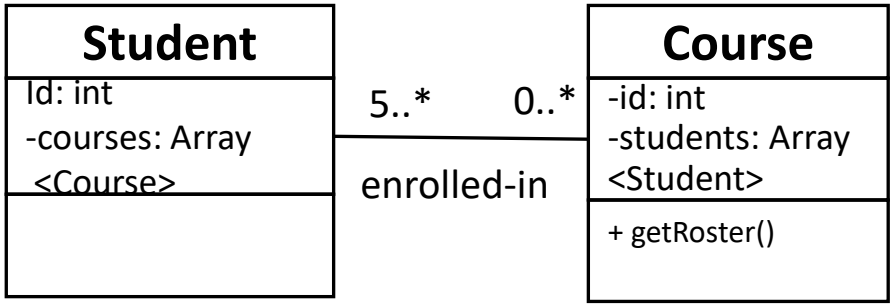
- One-to-one
  - Each student carries exactly one ID card.
  - Each ID card belongs to exactly one student.



- One-to-many
  - One rectangle list can contain many rectangles
  - One rectangle included in a single rectangle list

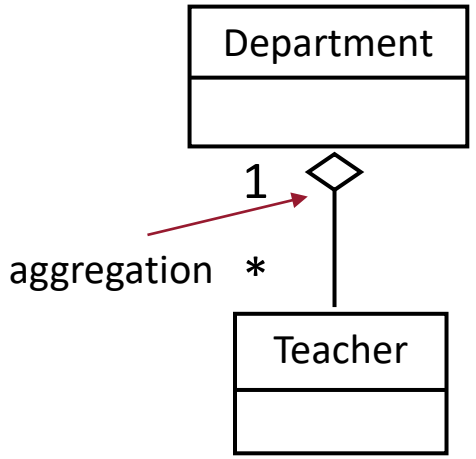


- Many-to-many
  - Each student can enroll in 0 or more courses
  - Each course has 5 or more students

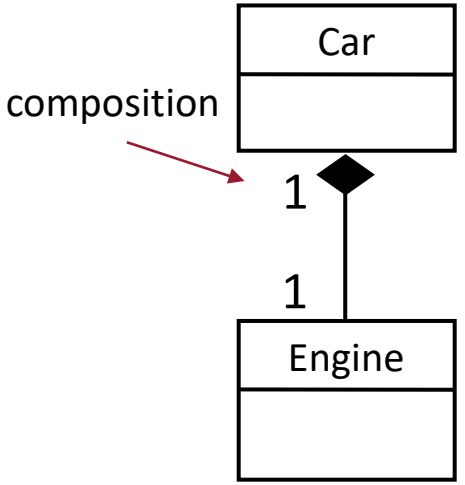


Association types

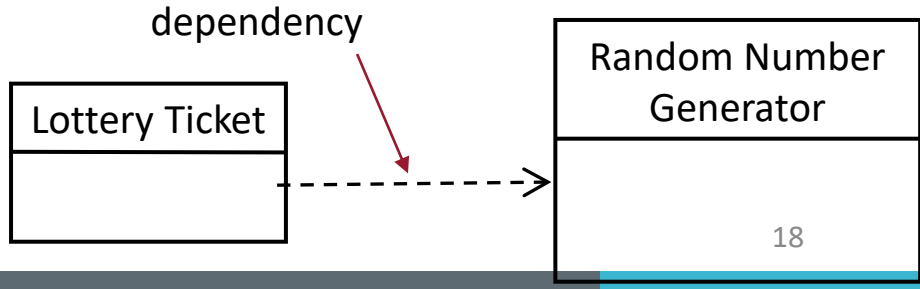
- **aggregation**: "is part of"
  - symbolized by a clear white diamond



- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

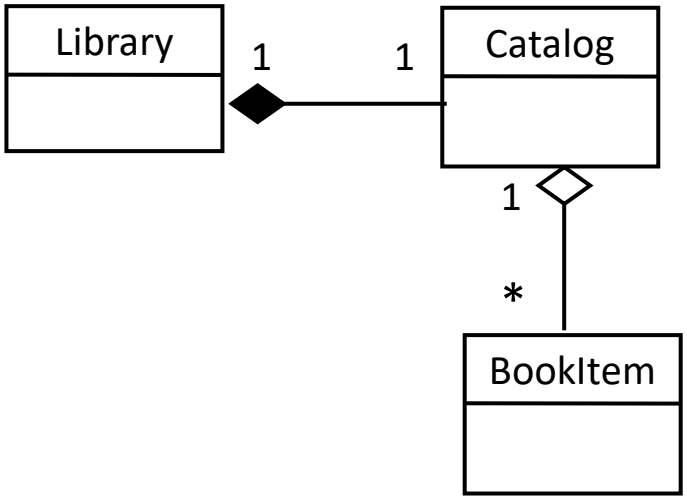


- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



Aggregation / Composition Example

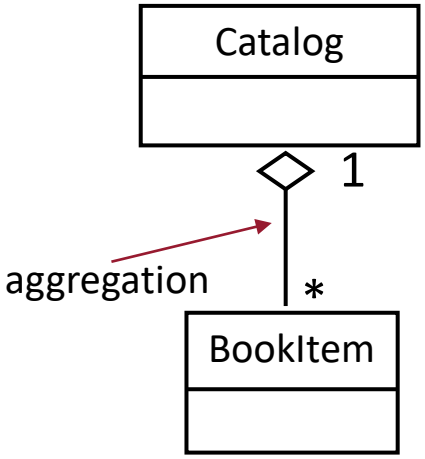
- If the library goes away
  - so does the *Catalog*: composition
  - but *BookItems* may still exist: aggregation



Association types

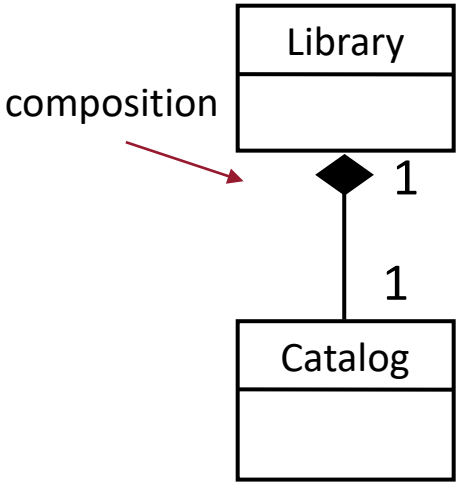
- **aggregation**: "is part of"
- Example:

```
public class Catalog {
    private ArrayList<BookItem> booklist;
    public  Catalog (){}
    public void add(BookItem p){
        this.booklist.add(p);
    }
}
public class BookItem {
    private String isbn;
}
```



- **composition**: "is entirely made of"
- Example:

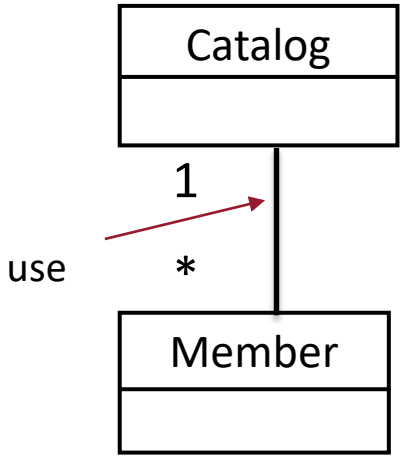
```
public class Library {
    private final Catalog catalog;
    public Library(){
        catalog =  new Catalog();
    }
}
public class Catalog {
    private String name;
}
```



Association types

- **use**:
- Example:

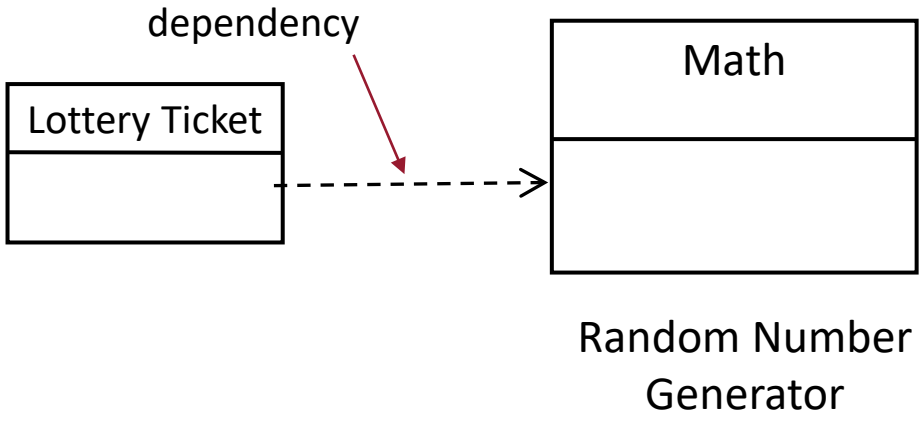
```
public class Catalog {
    private String name;
    public BookItem search(String k){
    }
}

public class Member {
    private int id;
    public void search(Catalog c, String k){
        b = c.search(k);
    }
}
```



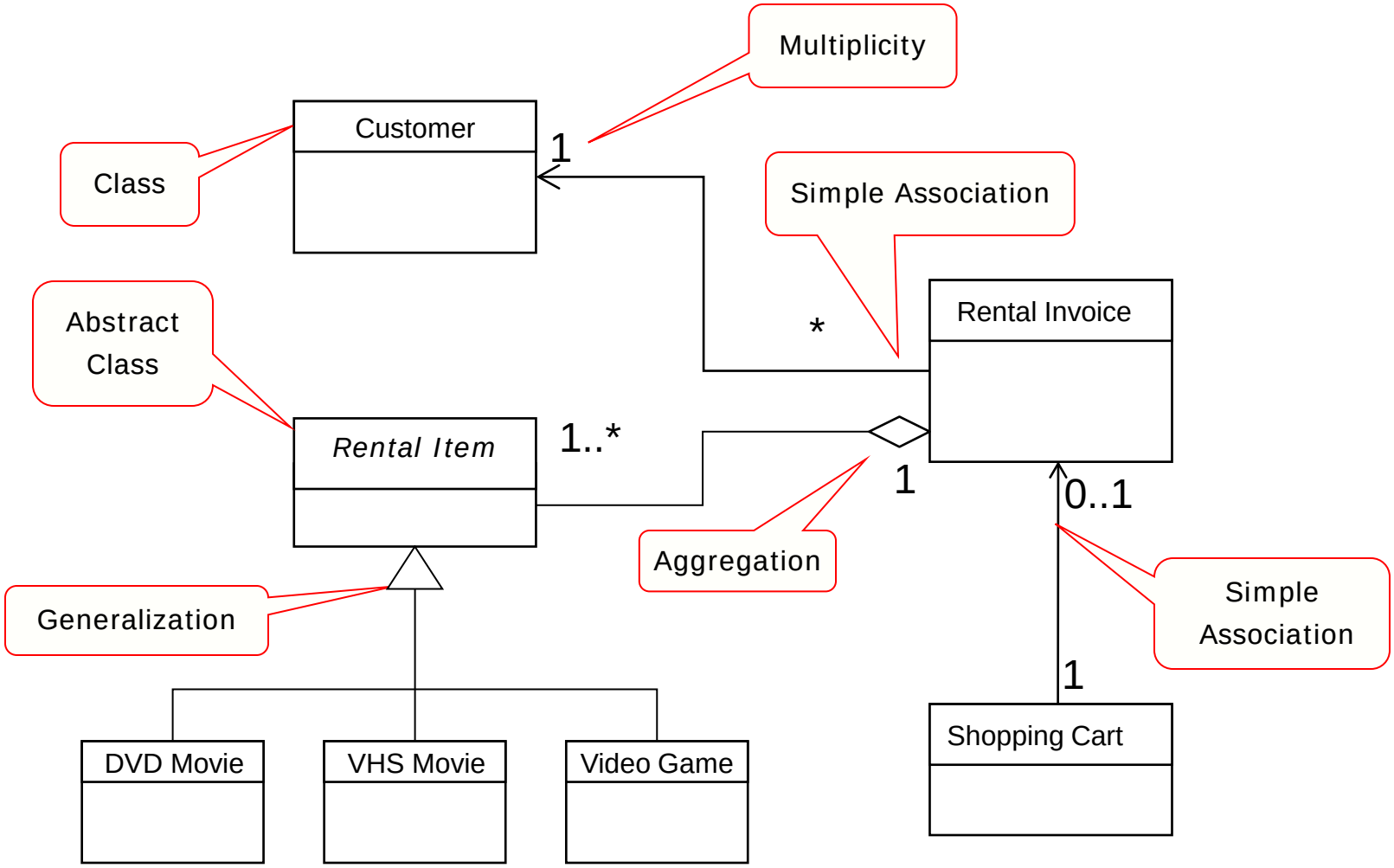
Association types

- **dependency**:

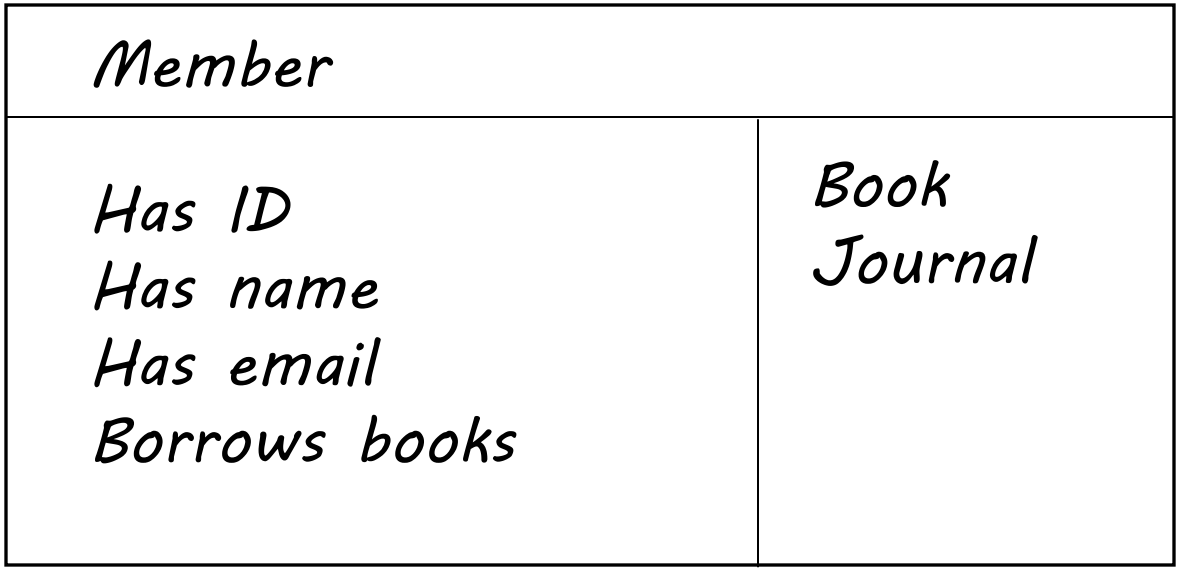
```
public class LotteryTicket {
    private int number;
    public LotteryTicket(){
        number = Math.random() * 100000;
    }
}
```



Class diagram example 2



Describing designs with CRC cards



- CRC (class-responsibility-collaborators) cards
  - on top of the card, write down the name of the class
  - below the name, list the following:
    - **responsibilities**: problems to be solved; short verb phrases
    - **collaborators**: other classes that are sent messages by this class

How to design classes?

- Identify classes and interactions from project requirements:
  - **Nouns** are potential classes, objects, and fields
  - **Verbs** are potential methods or responsibilities of a class
  - **Relationships** between nouns are potential interactions (generalization, aggregation, dependence, etc.)

- Which nouns in your project should be classes?
- Which ones are fields?
- What verbs should be methods?
- What are potential interactions between your classes?

Example: Identify classes from requirements

Library Example:

- The library contains books and journals.
  - It may have several copies of a given book.
- Library members can borrow books from the library. Library staff are assumed to be library members by default.
  - Library members can normally borrow up to six items at a time,
  - But library staff may borrow up to 12 items at a time.
  - Only library staff may borrow journals.
- The system must keep track of when books and journals are borrowed and returned.

Example: Identify classes from requirements (revisited)

Noun Identification:

- The library contains books and journals.
  - It may have several copies of a given book.
- Library members can borrow books from the library. Library staff are assumed to be library members by default.
  - Library members can normally borrow up to six items at a time,
  - But library staff may borrow up to 12 items at a time.
  - Only library staff may borrow journals.
- The system must keep track of when books and journals are borrowed and returned.

Library Example: Candidate Classes

Noun	Comments	Candidate
Library	the name of the system	
Book		
Journal		
Copy		
Library Member		
Library Staff		
Item	book or journal	
Time	abstract term	
System	general term	

Library Example: Candidate Classes

Noun	Comments	Candidate
Library	the name of the system	No
Book		Yes
Journal		Yes
Copy		Yes
Library Member		Yes
Library Staff		Yes
Item	book or journal	No
Time	abstract term	No
System	general term	No

Relations Between Classes

BookCopy	Book
LibraryMember	BookCopy, Journal
LibraryStaff	LibraryMember
LibraryStaff	BookCopy, Journal

Book
Journal
BookCopy
LibraryMember
LibraryStaff

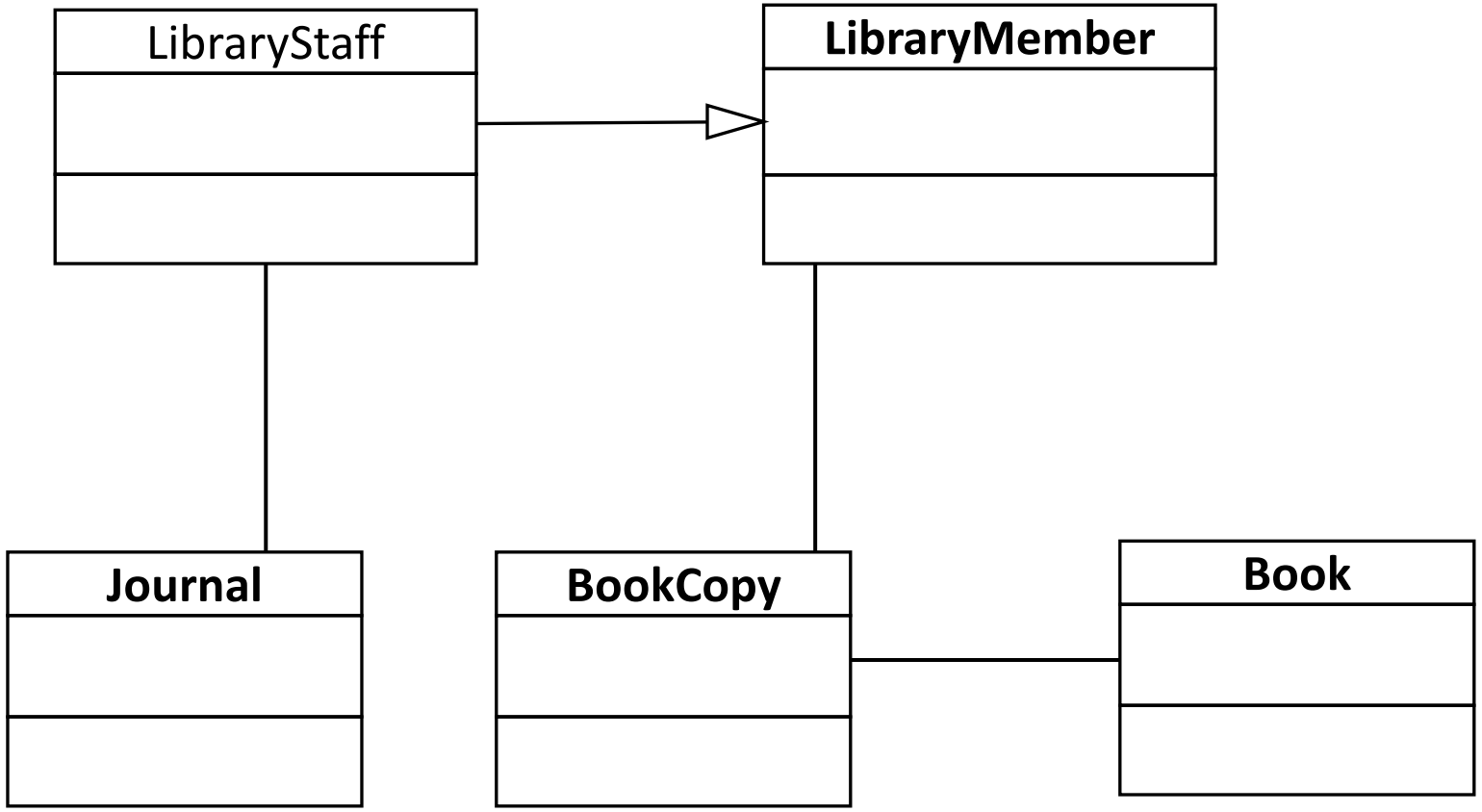
Relations Between Classes

BookCopy	is a copy of a	Book
LibraryMember	borrows/returns	BookCopy
LibraryStaff	is an	LibraryMember
LibraryStaff	borrows/returns	BookCopy, Journal

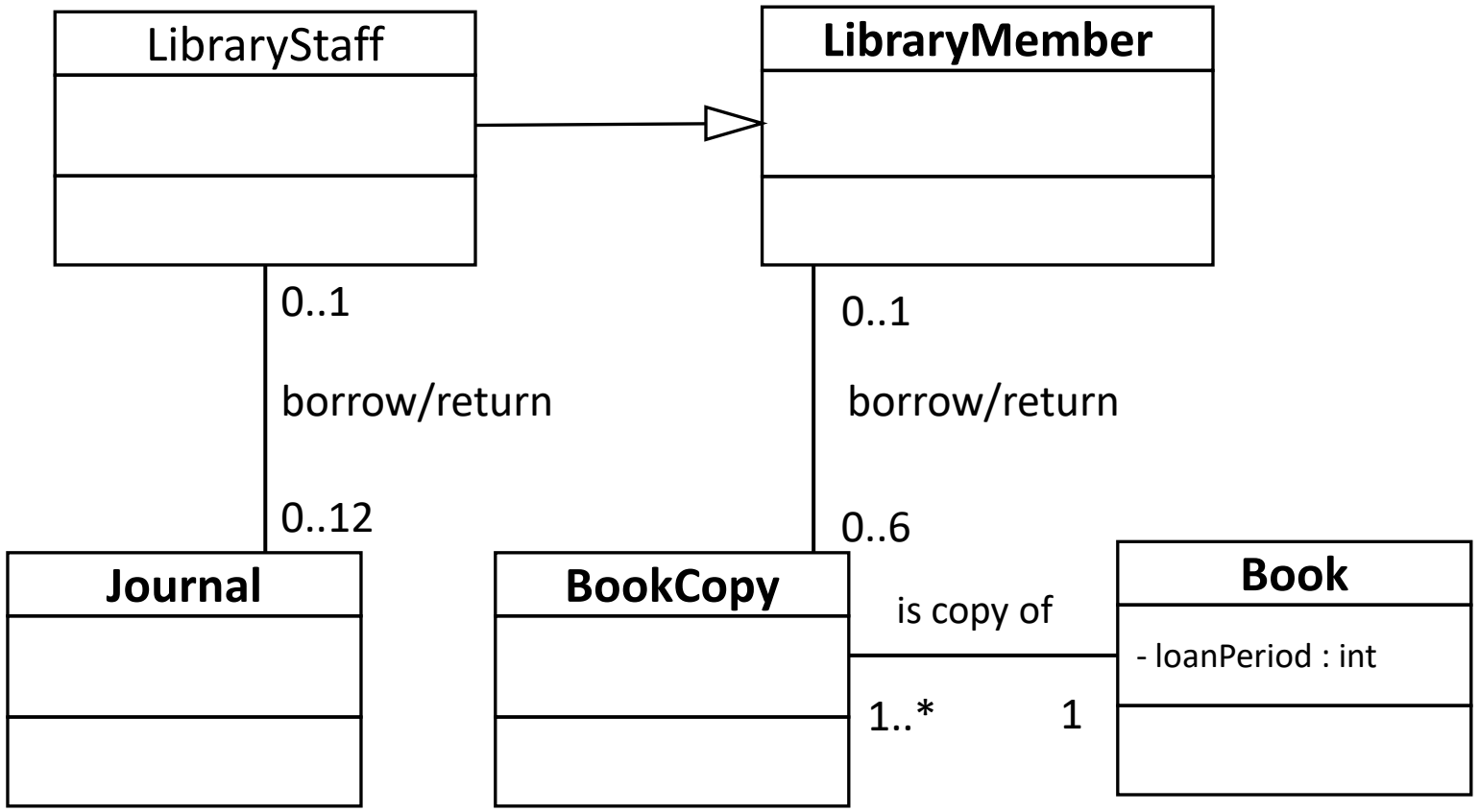
Methods of Classes

LibraryMember	borrows	BookCopy
LibraryMember	returns	BookCopy
LibraryStaff	borrows	Journal
LibraryStaff	returns	Journal

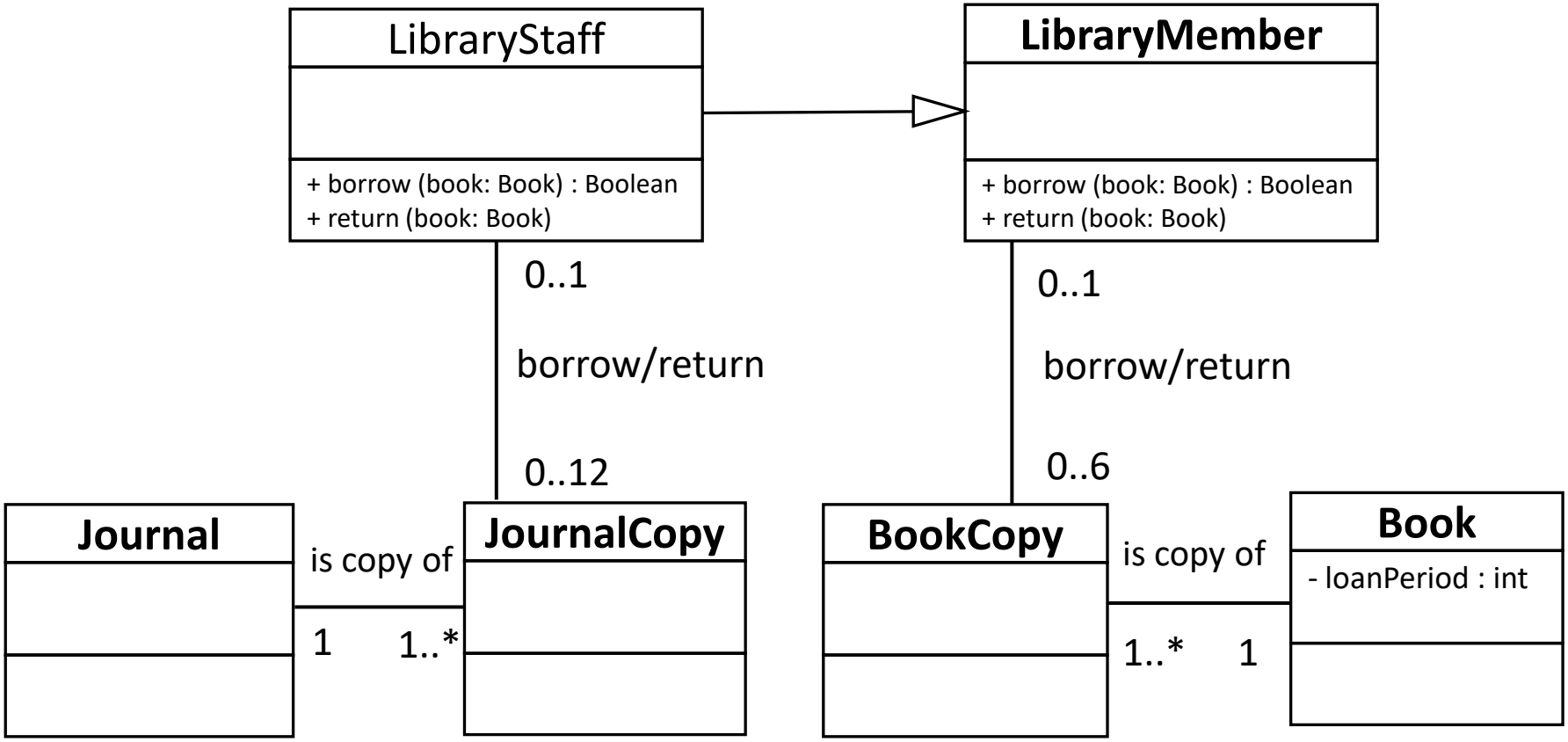
A Possible Class Diagram



A Possible Class Diagram



Revised Class Diagram



What to use class diagrams for

- Class diagrams are great for:
  - discovering related data and attributes
  - getting a quick picture of the important entities in a system
  - seeing whether you have too few/many classes
  - seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
  - spotting dependencies between one class/object and another

- Not so great for:
  - discovering algorithmic (not data-driven) behavior
  - finding the flow of steps for objects to solve a given problem
  - understanding the app's overall control flow (event-driven? web-based? sequential? etc.)

Database Design Using UML

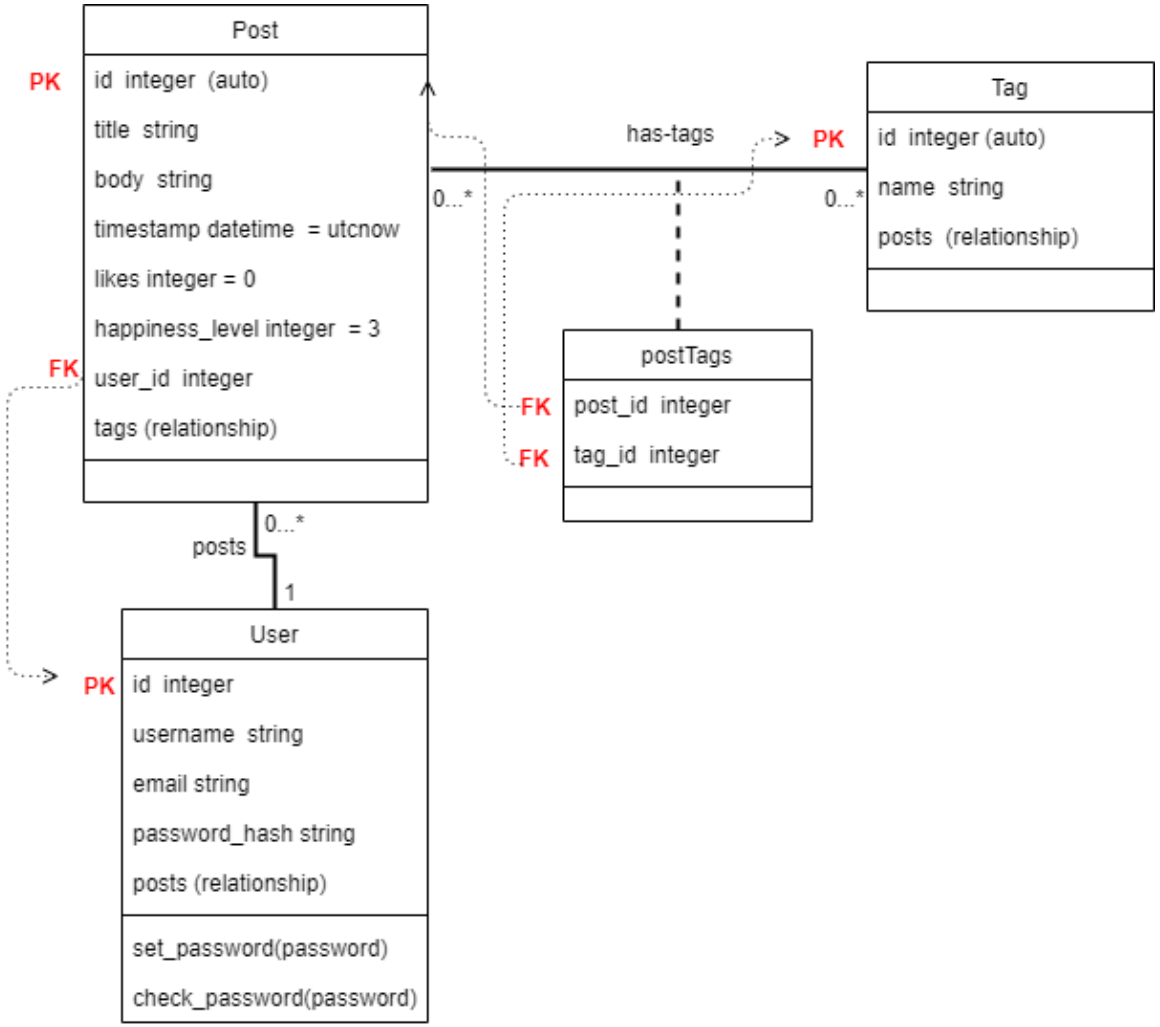
- Modeling database tables using UML
- 5 concepts
  (1) Classes
  (2) Associations
  (3) Association Classes
  (4) Subclasses
  (5) Composition & Aggregation

Database Design Using UML : Classes

- Every table is represented as a class
  - Example (Smile App):

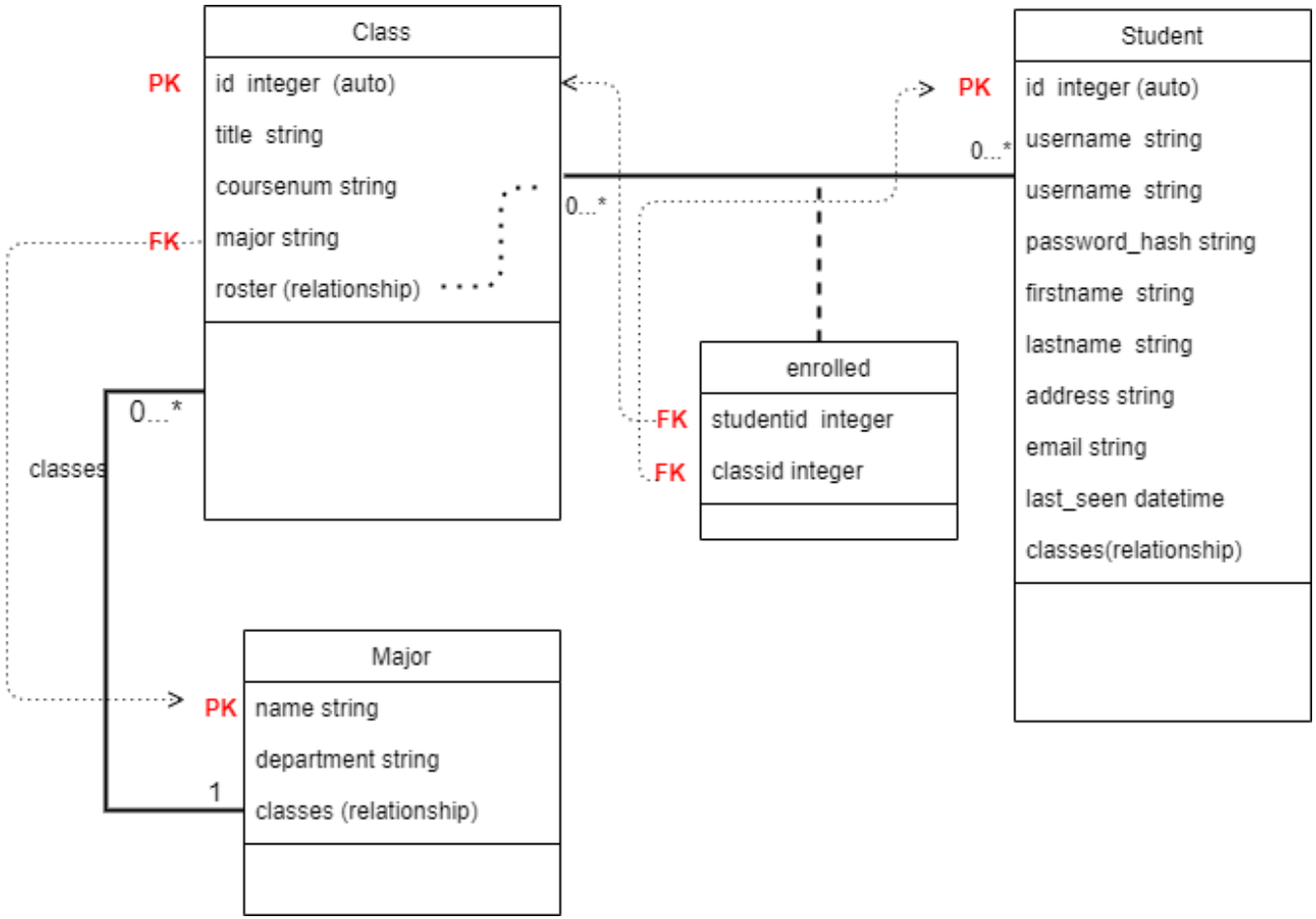
Database Design Using UML : Classes

- Every table is represented as a class
  - Example (Smile App):



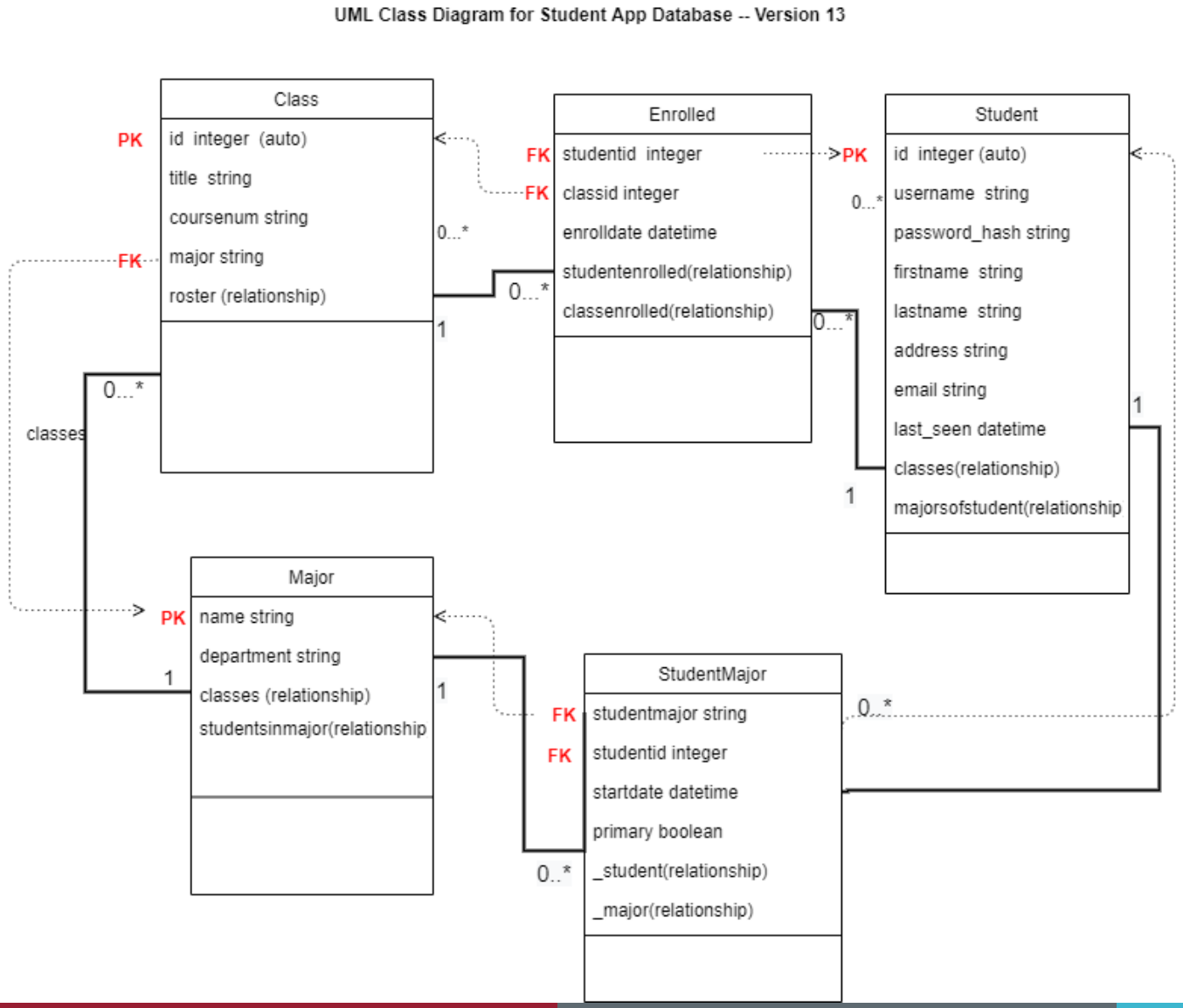
Database Design Using UML : Classes

- Example (Student App – version 11):



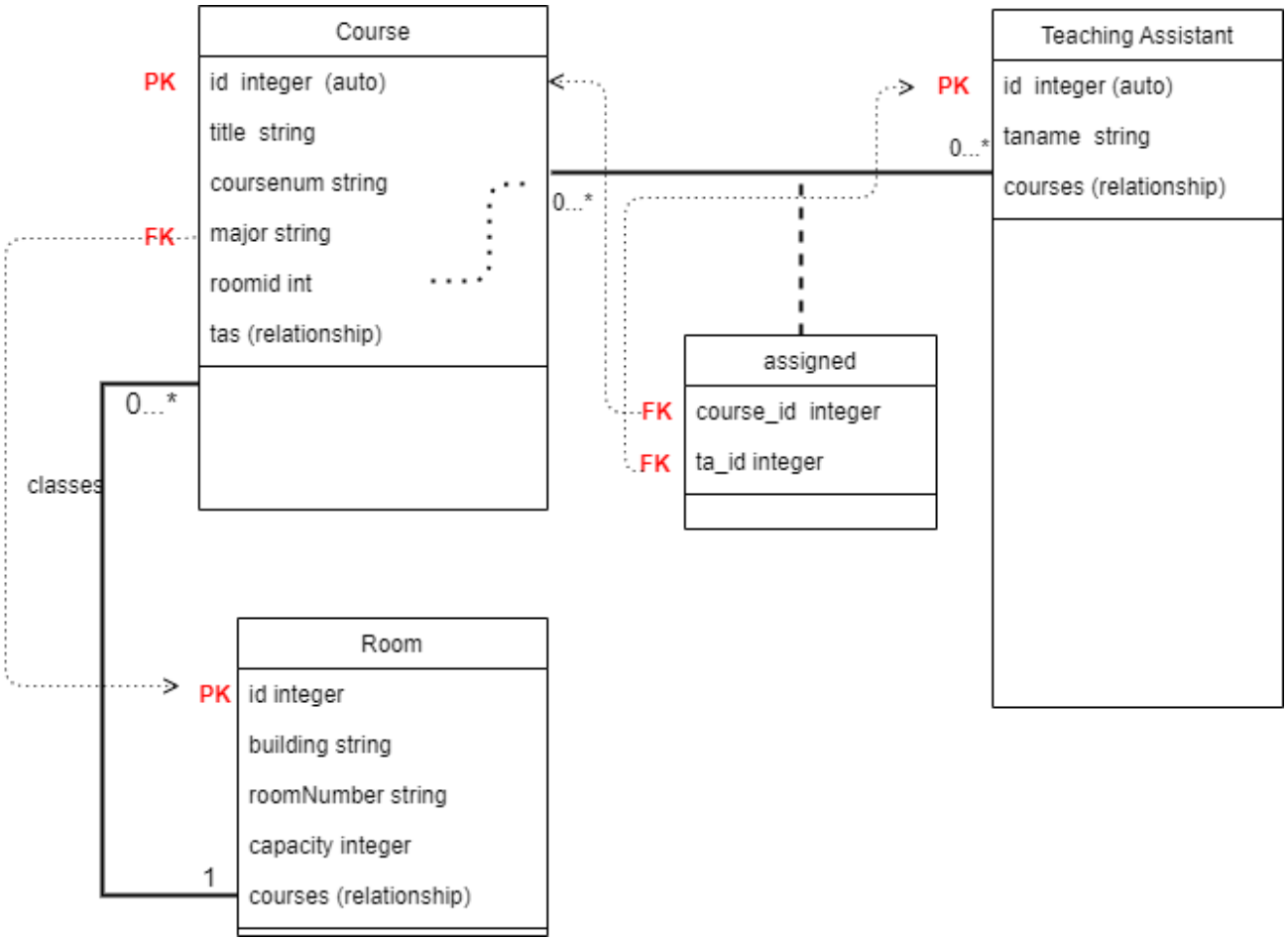
Database Design Using UML : Classes

- Example (Student App – version 13):



Database Design Using UML : Classes

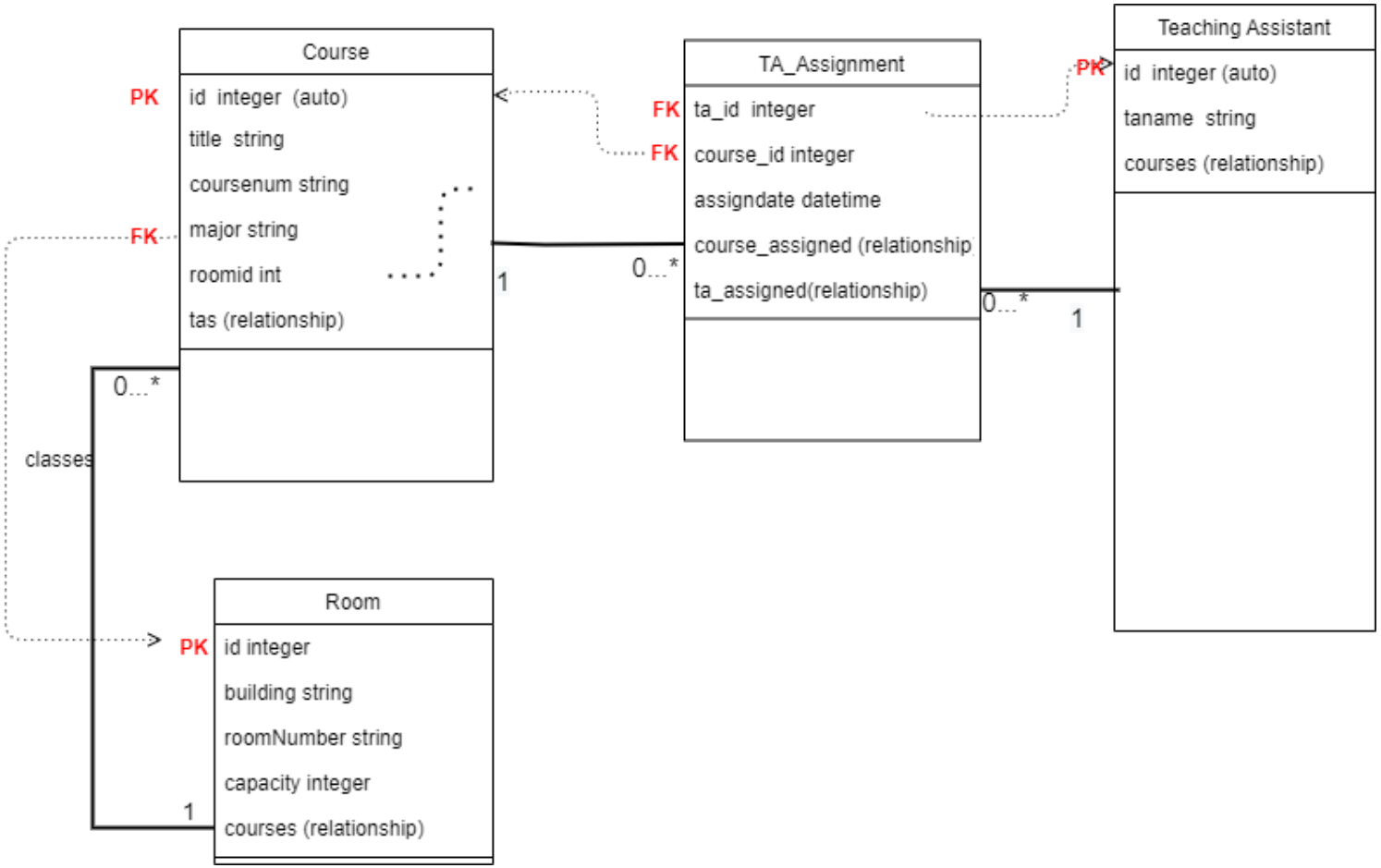
- Example (Flask Class Example – version 13):



https://github.com/WSU-CptS-arslanay/FlaskLectureExample

Database Design Using UML : Classes

- Example (Flask Class Example – version 14):



Database Design Using UML : Classes

Example (Term Project Application):