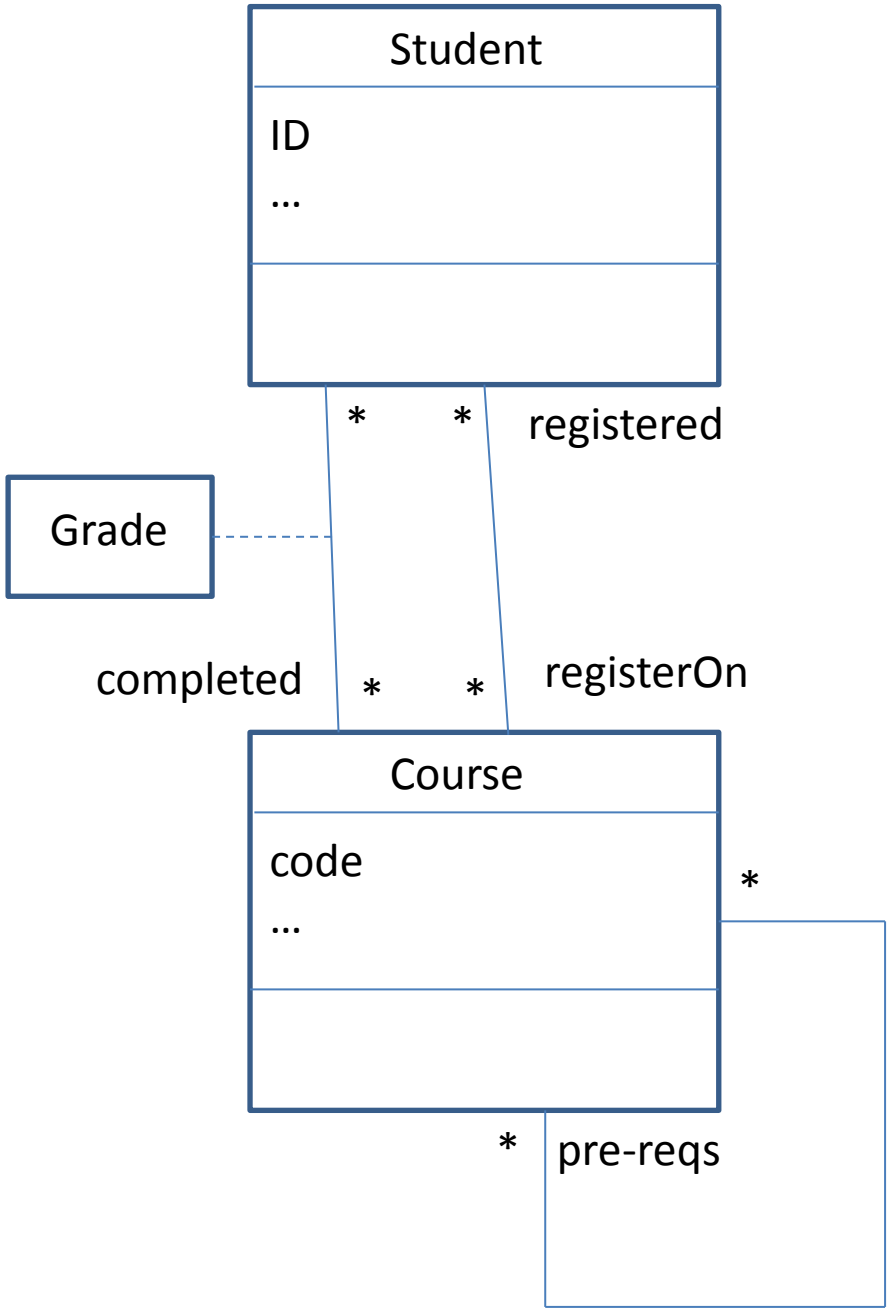
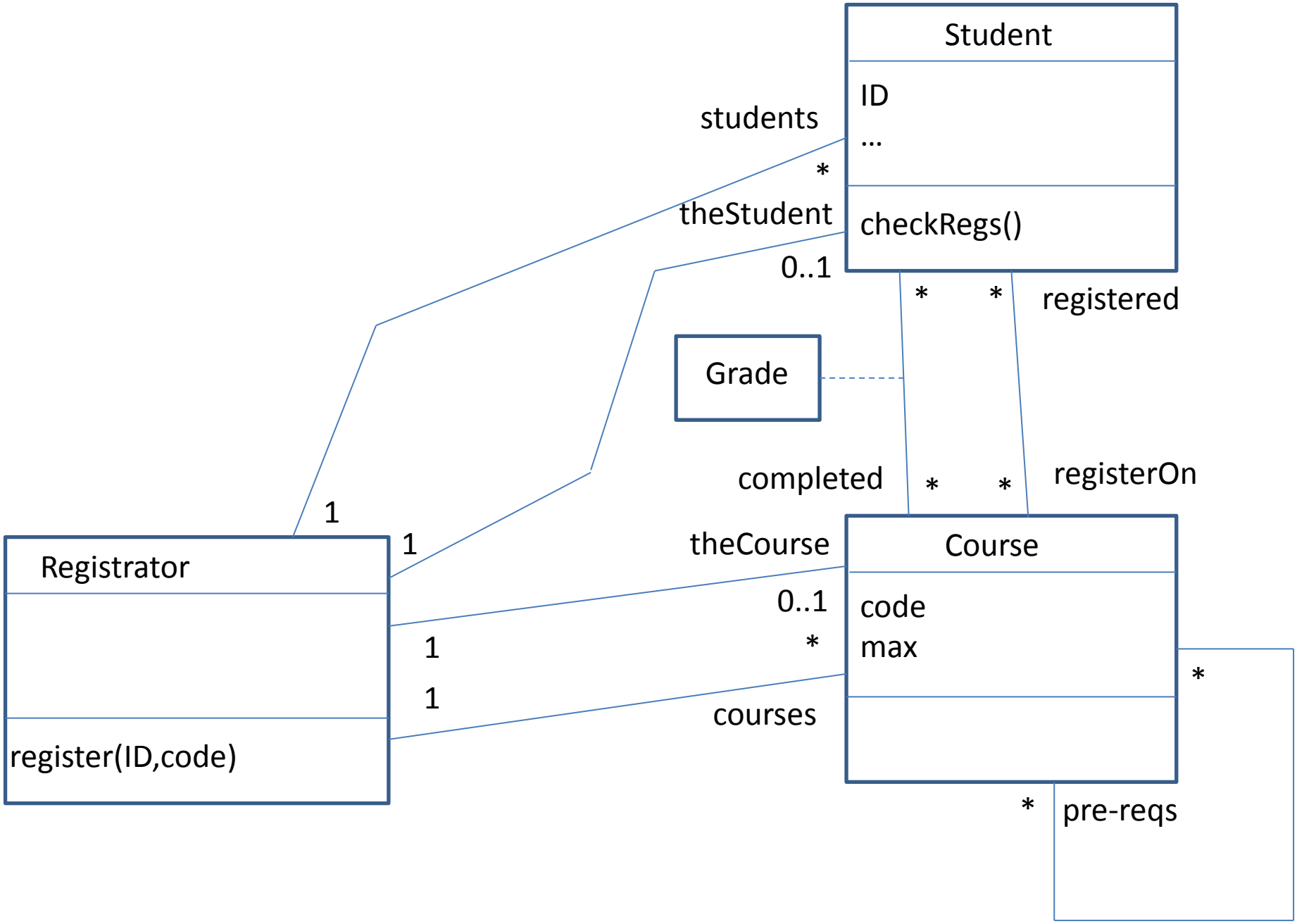
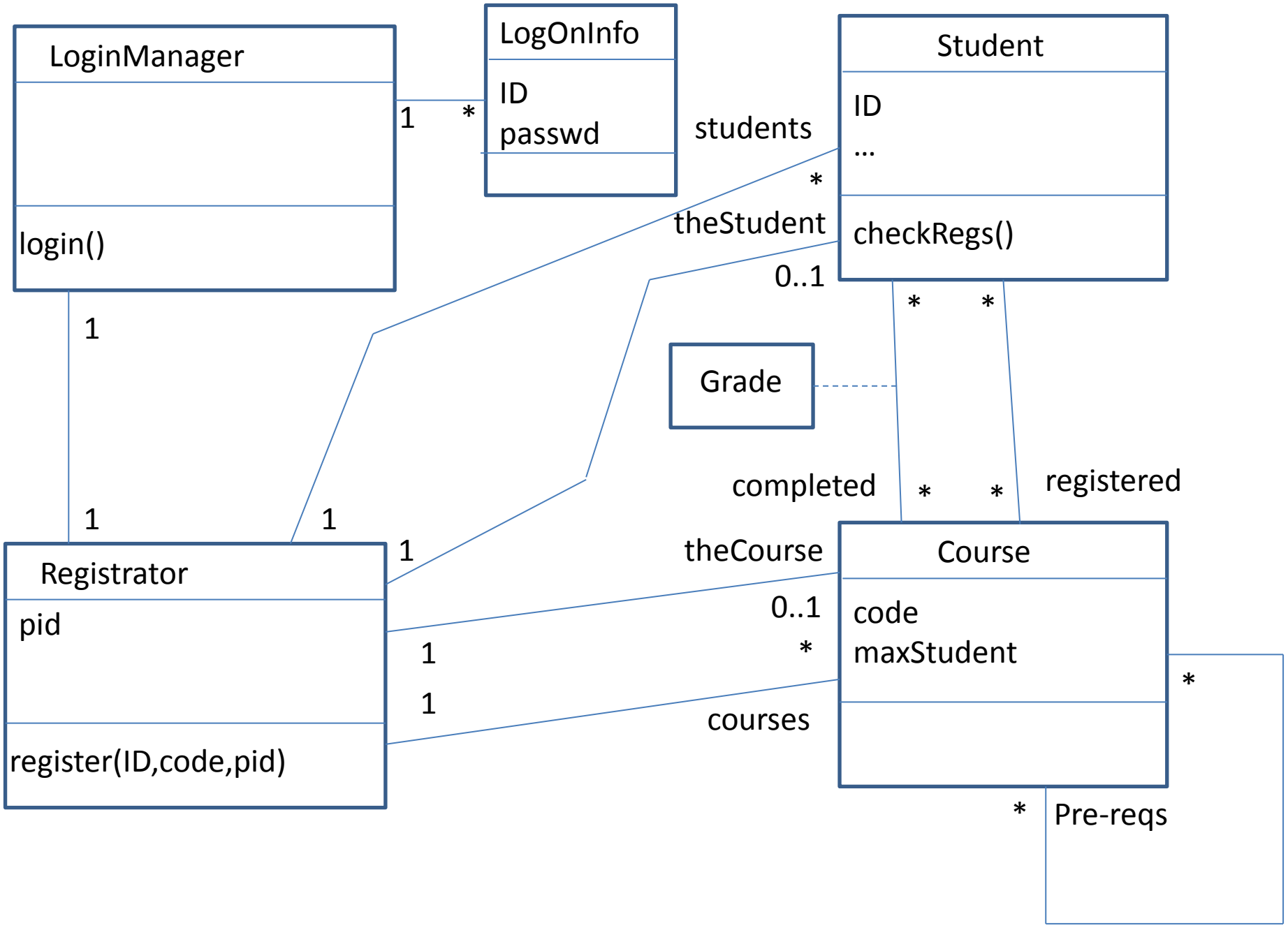
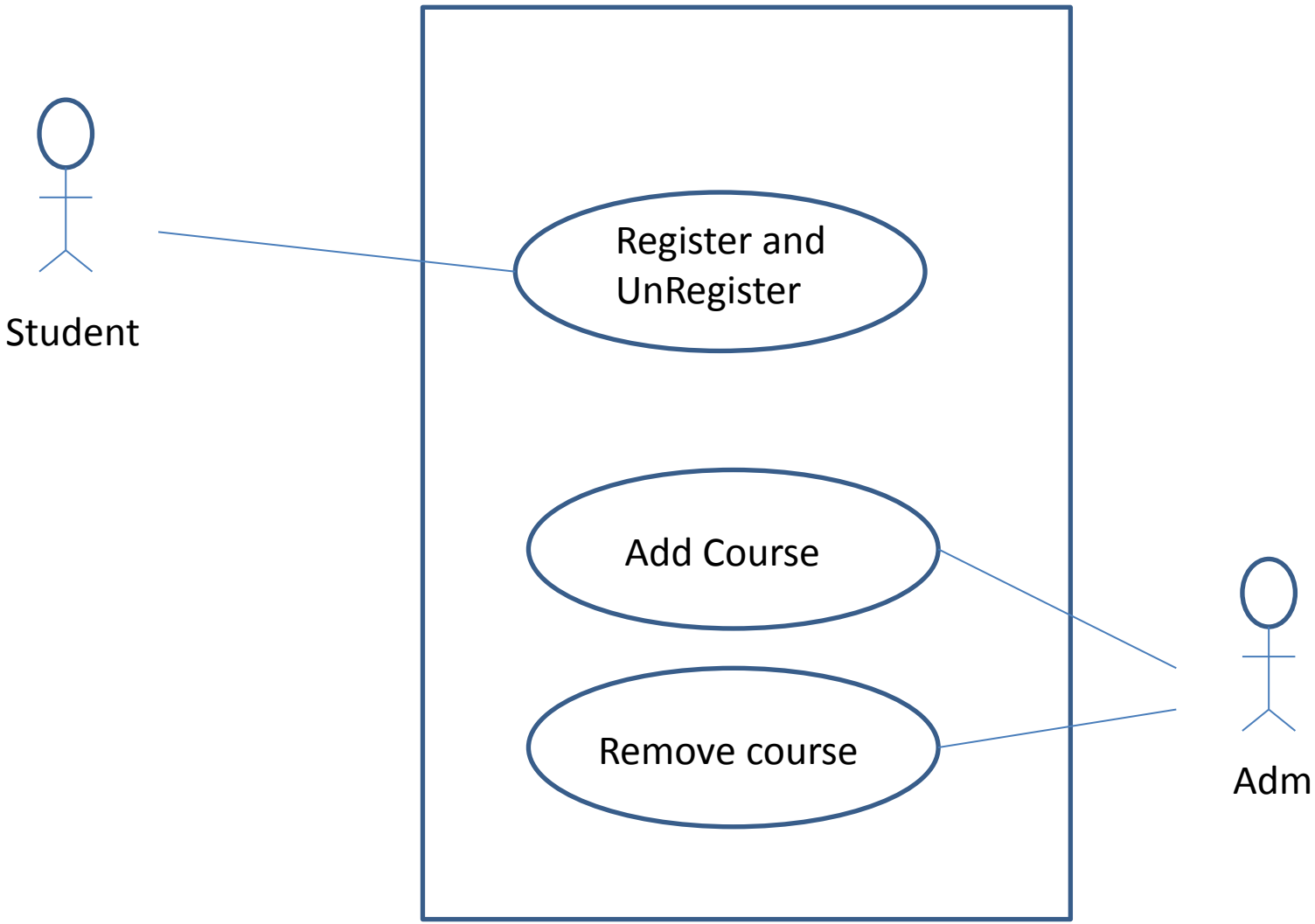
Sequence Diagrams and Class Diagrams


Student
ID
…
* *
registered
Grade
completed
* *
registerOn
Course
code
…
*
*
pre-reqs


Student
ID
...
checkRegs()
students
*
theStudent
0..1
* *
registered
Grade
completed
* *
registerOn
1
1
Registrator
register(ID,code)
theCourse
0..1
*
1
1
courses
Course
code
max
*
* pre-reqs


LoginManager
login()
LogOnInfo
ID
passwd
Student
ID
…
checkRegs()
Grade
Registrator
pid
register(ID,code,pid)
Course
code
maxStudent
Pre-reqs
1
*
students
theStudent
0..1
completed
registered
theCourse
courses


Student
Register and
UnRegister
Add Course
Remove course
Adm

Choice

…

n. System gives the option: register and unregister

(n+1). Choice: register

(n+2). The student choose to register

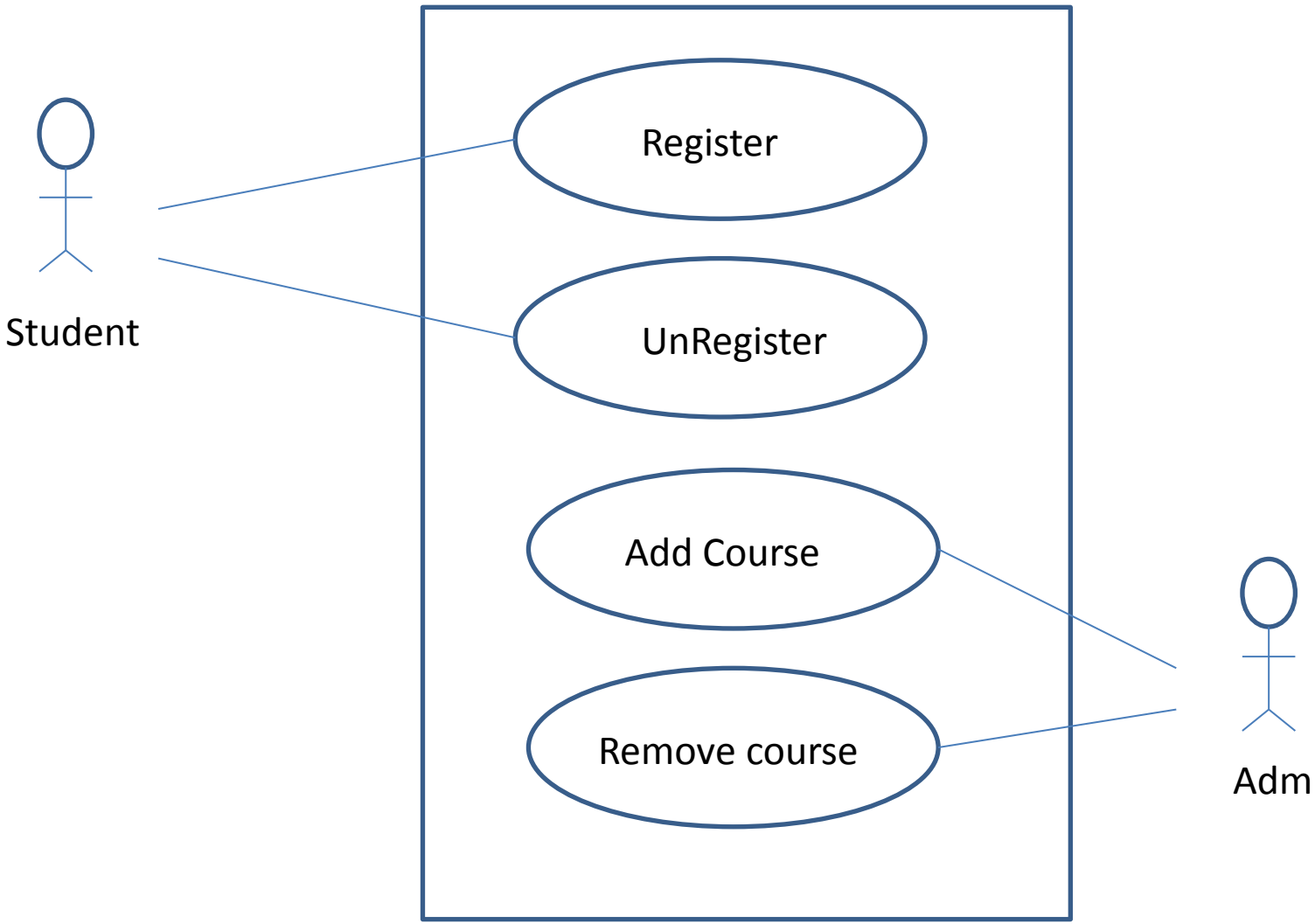
…

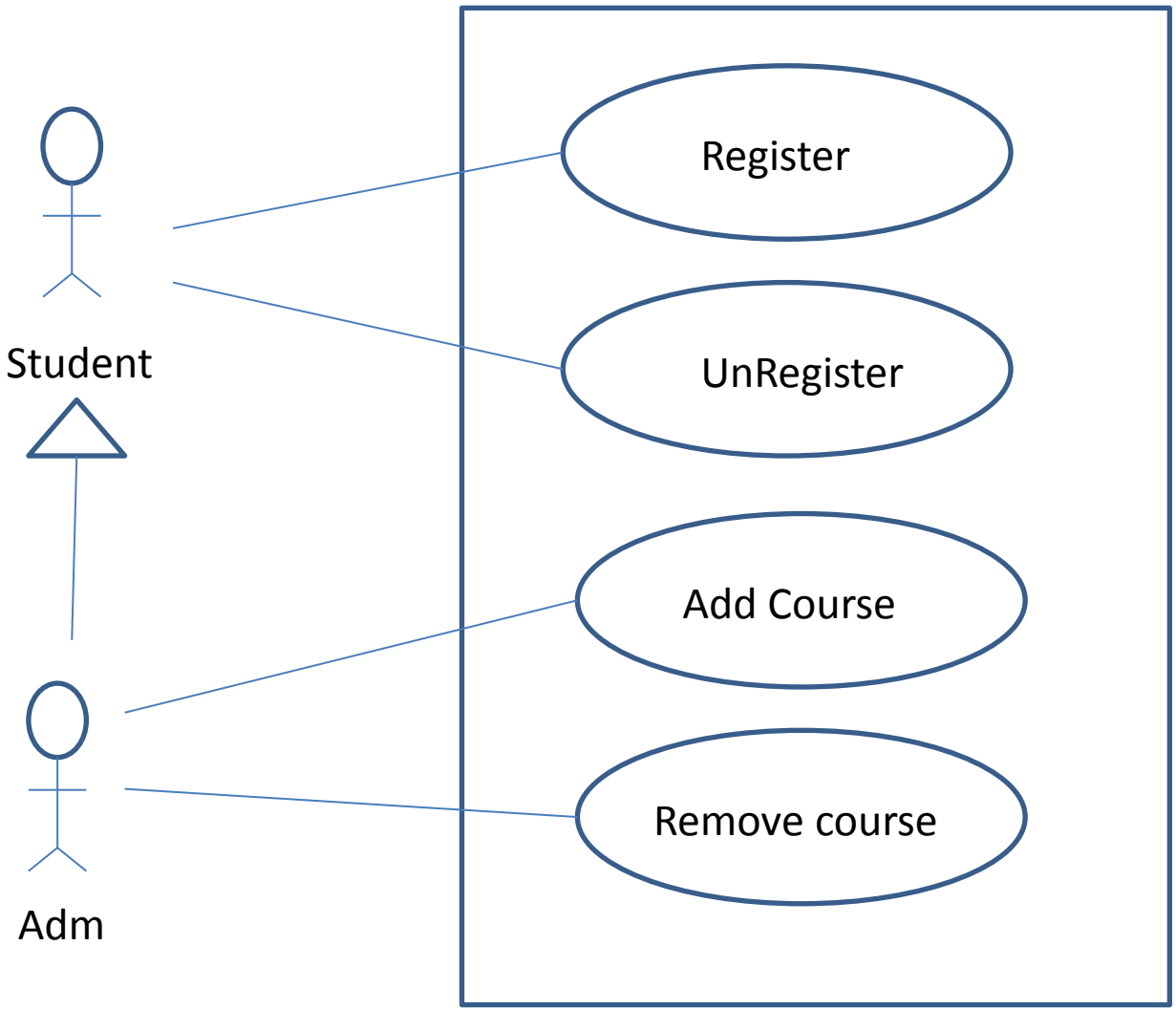
Alternative Flow

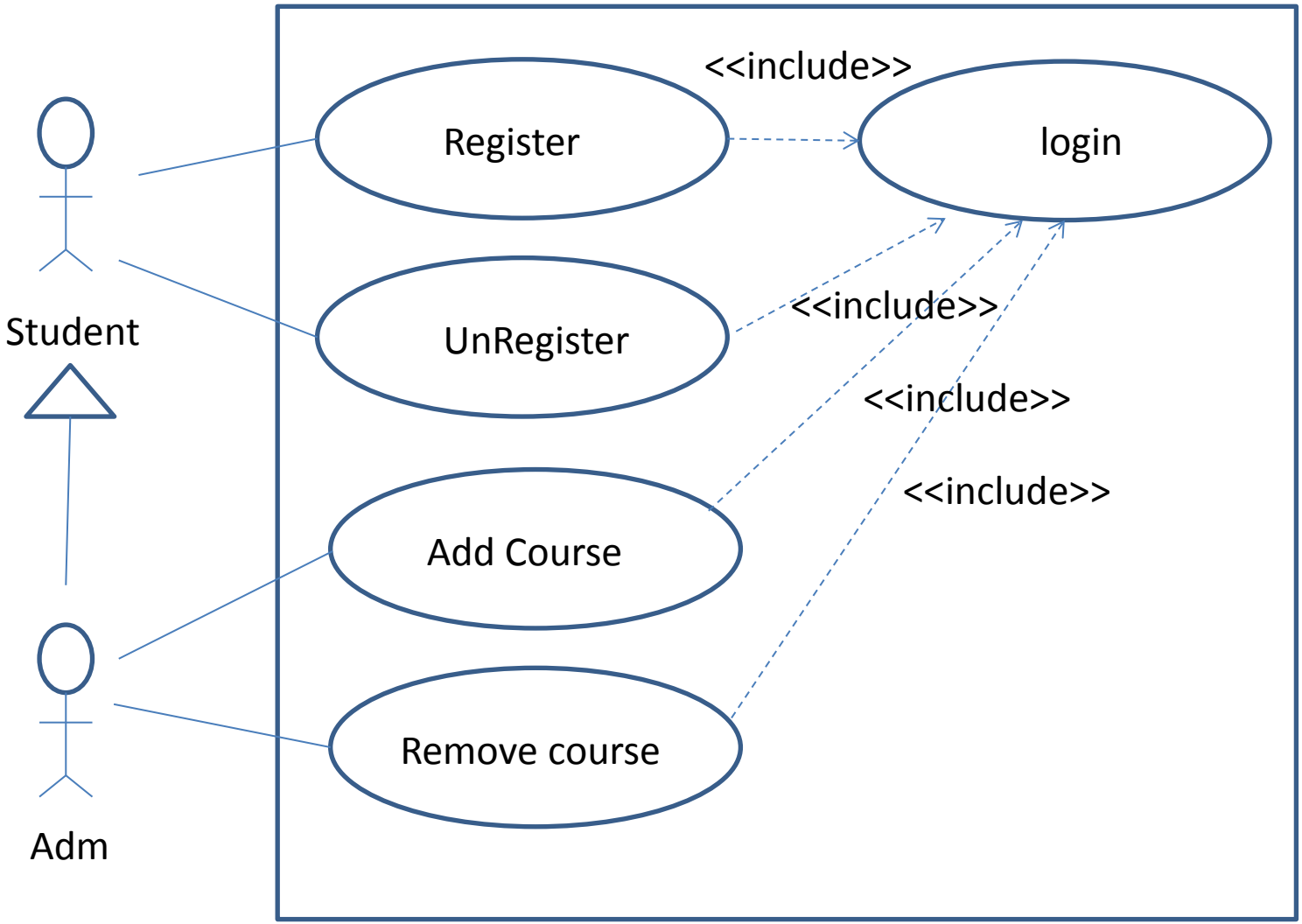
Choice: unregister

1. ….
2. ….
3. ….

Or only deal with registration


Register
UnRegister
Add Course
Remove course
Student
Adm


Register
UnRegister
Add Course
Remove course
Student
Adm



One Might Decide To Leave Out Login Since It Is Not Important For Understanding The Functionality

The same with logout

1. User register with Id and password
2. System search for a password with the given login
3. Assume: login and password is correct
4. System generates a unique process Id
5. System return the process Id

Alternative Flow

3a .Assume: Id and password was not correct less than three times

1. System inform that the id or password was wrong
2. System update the number of try
3. System inform how many more try is possible
4. Go to 1

3-5 Assume: Id and password was wrong tree times

1. Inform the student to contact the course adm

Name: Register

Actor: User

Goal: Register student to course

Description: The student is registered if there are places left and the student meets the pre-requistites

1. include login
2. Assume: login succeed
3. user register with student Id, course code and process Id
4. Assume: the process Id is correct
5. System finds student and course
6. Assume: the student exists
7. Assume: the course exists
8. The system checks if there are enough places left on the chosen course
9. Assume: there are enough places
10. The system checks if the student meets the pre-requisites
11. Assume: the student has all the required courses
12. System registers the student to the course
13. System informs student that s/he is registered

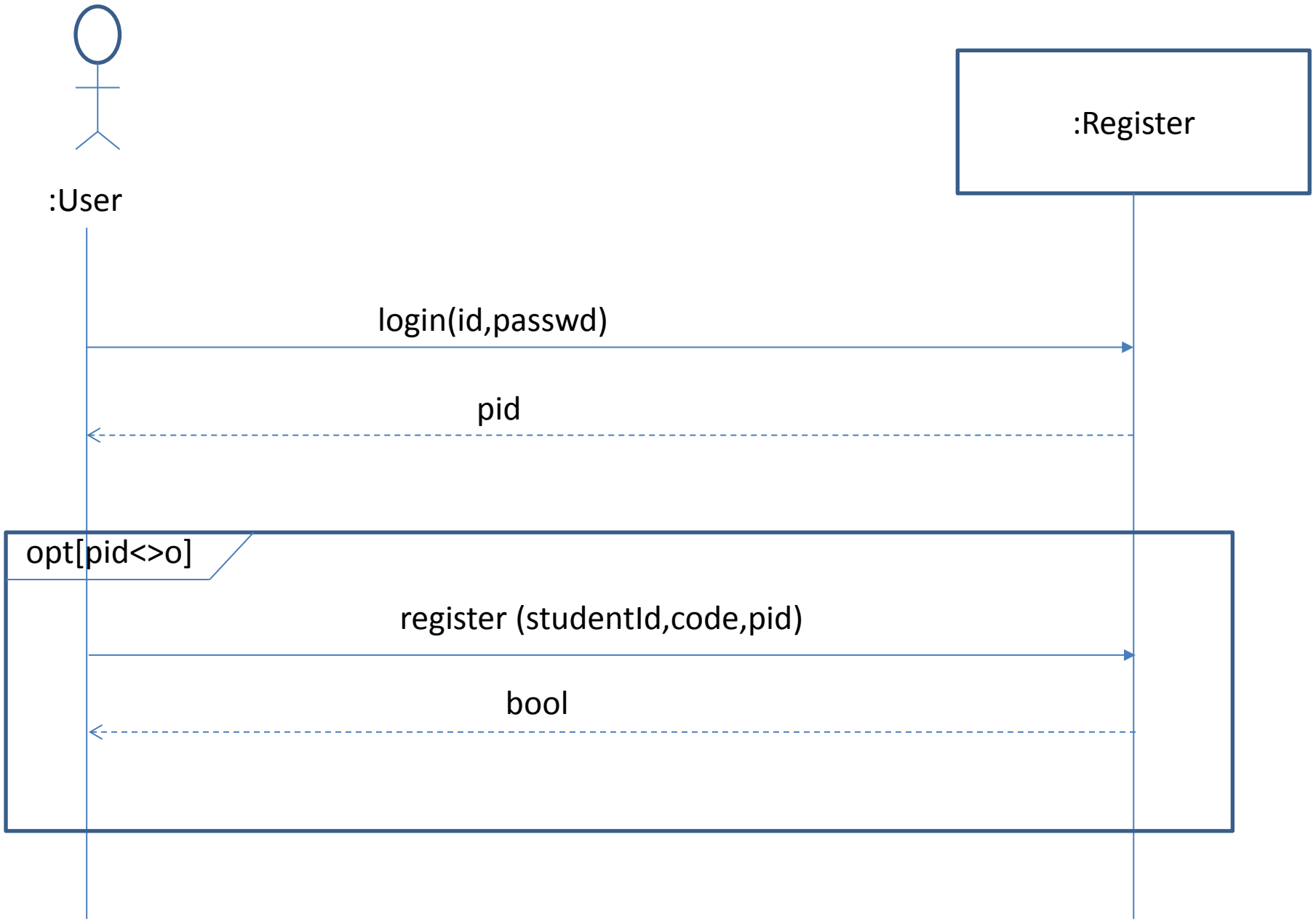
Alternative not specified

- Pre: _
- Post: if the user was logged in and the student existed and the student had the correct pre-requisites and there were places left on the course then

  The student was registered to the course

  Otherwise

  Nothing has changed in the system


:User
:Register
login(id,passwd)
pid
opt[pid<>o]
register (studentId,code,pid)
bool

Operation Register(studentId,courseId,key):Boolean

Pre: login

Post:

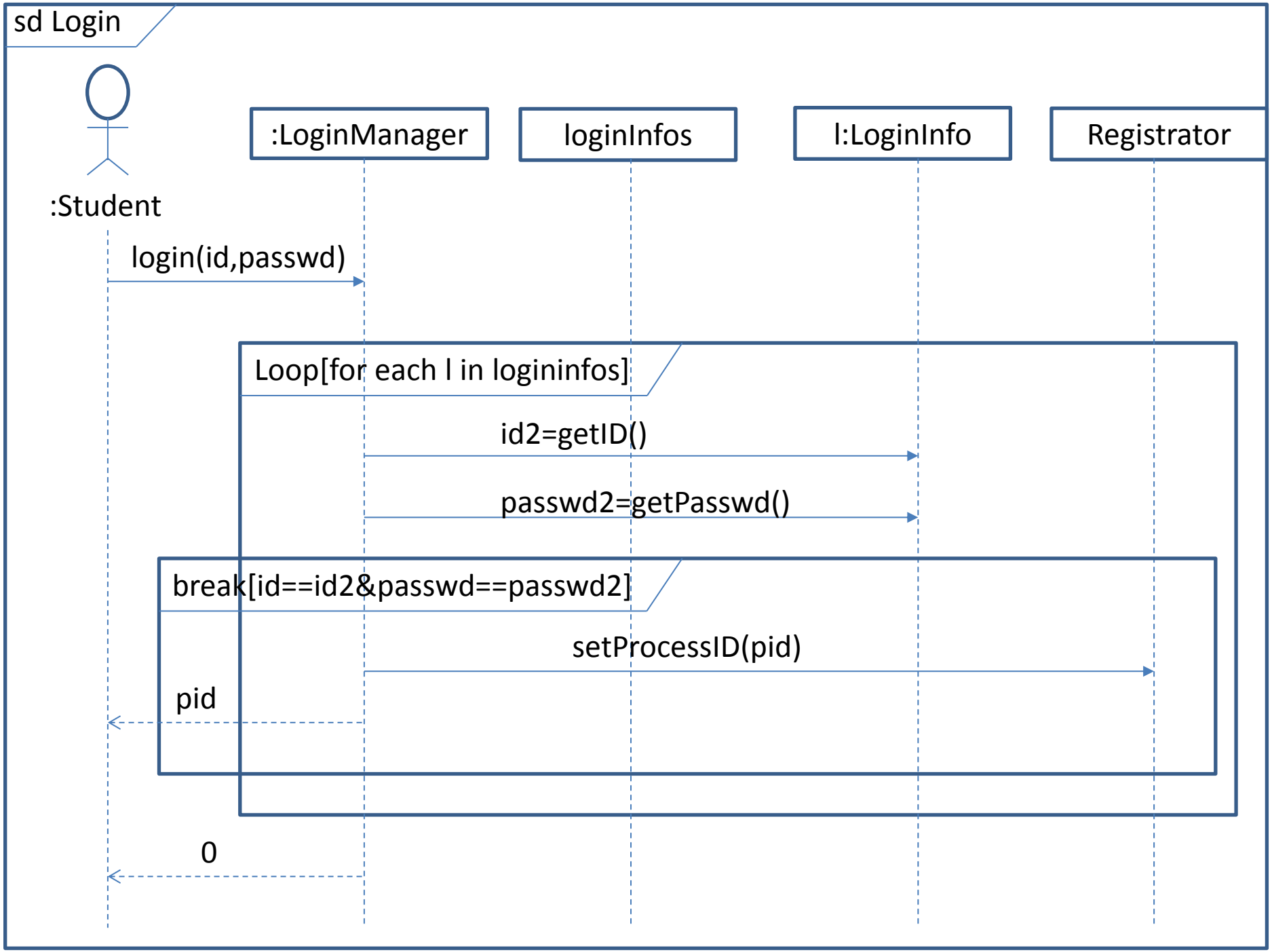
If there was places on the course and student has the correct course requirement

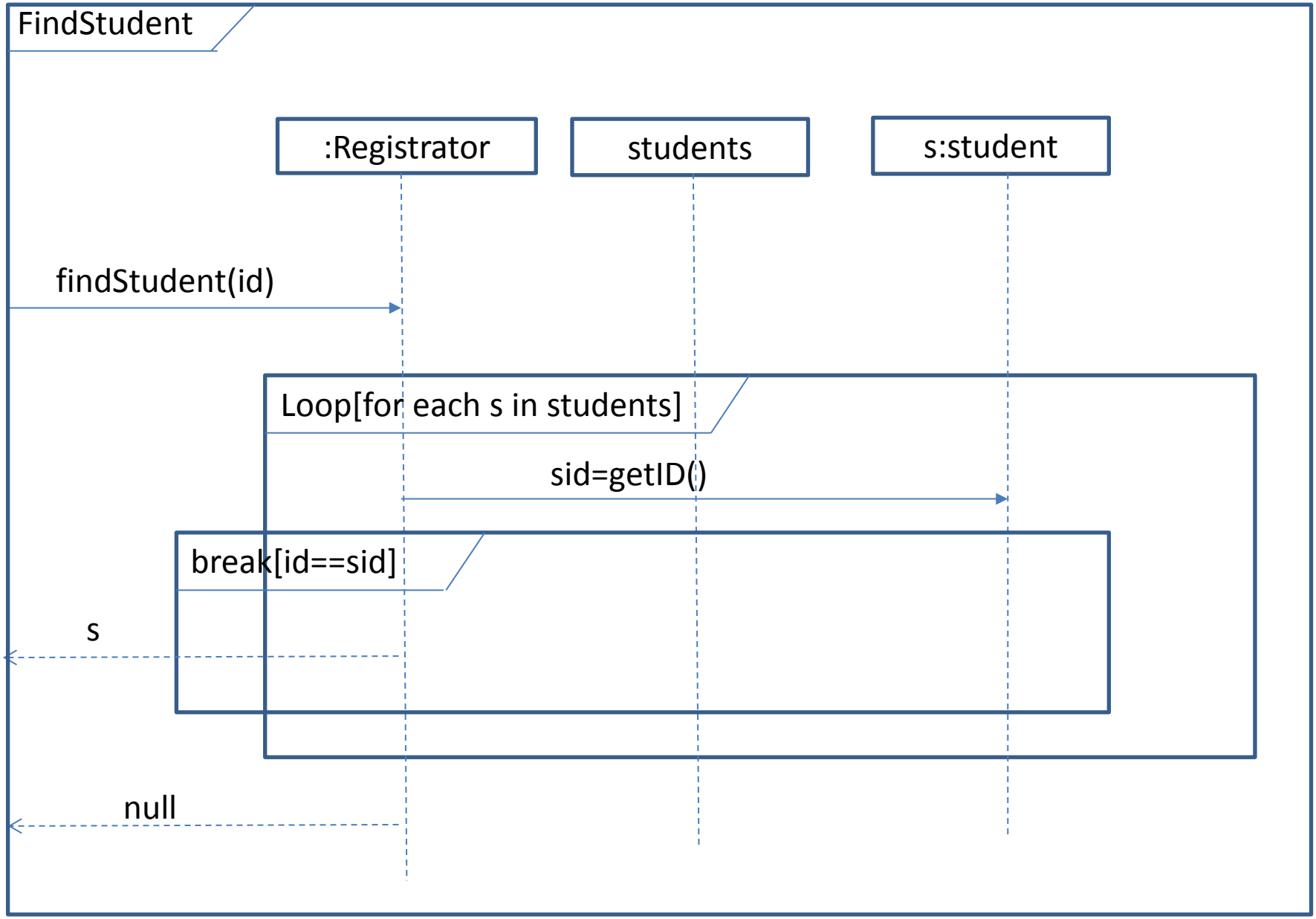
then

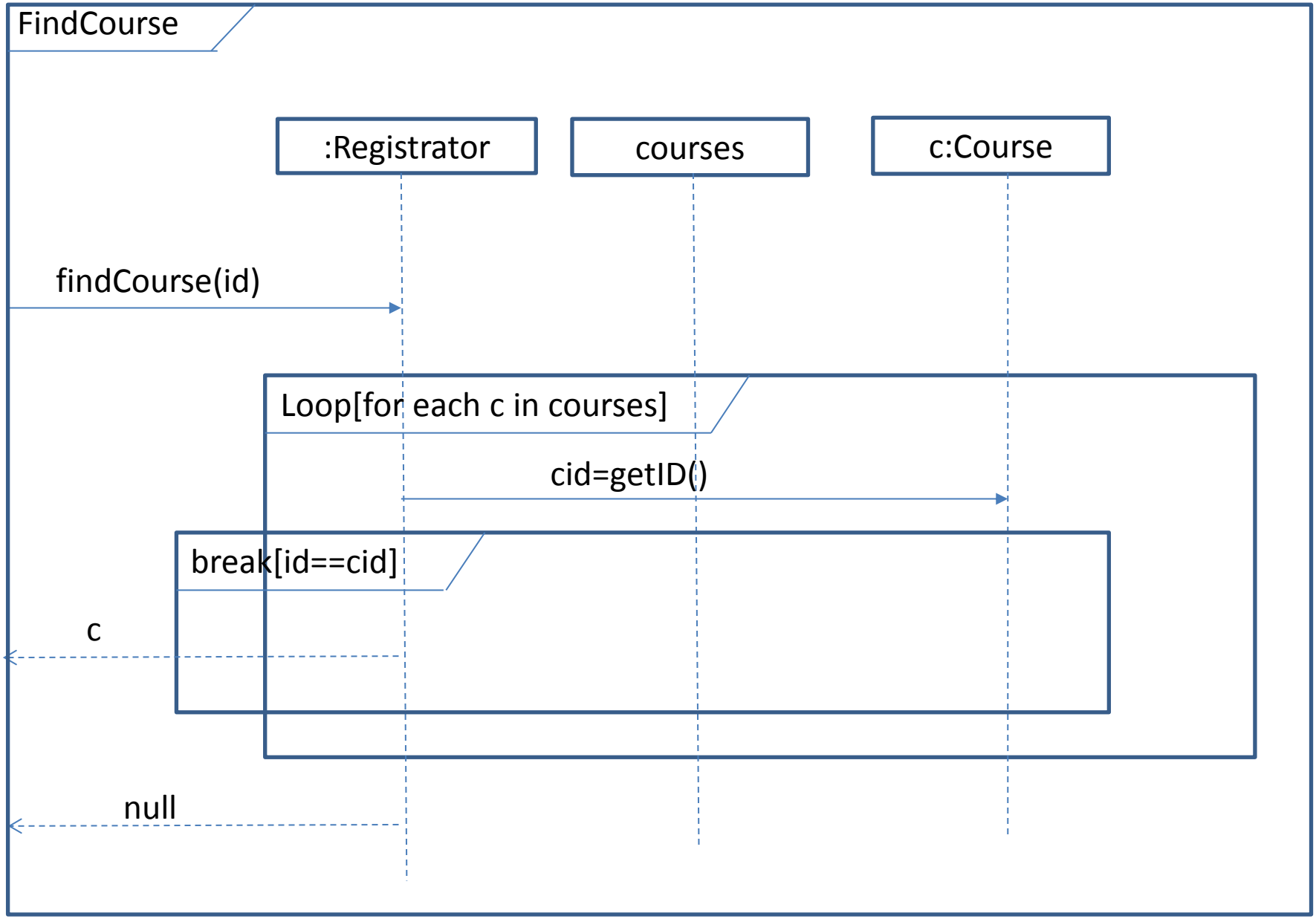
- a new link between course and student has been created to registrant that the student take the course
- Return = true

else

- No changes
- Return = false


sd Login
:Student
:LoginManager
loginInfos
l:LoginInfo
Registrator
login(id,passwd)
Loop[for each l in logininfos]
id2=getID()
passwd2=getPasswd()
break[id==id2&passwd==passwd2]
setProcessID(pid)
pid
0


FindStudent
:Registrator
students
s:student
findStudent(id)
Loop[for each s in students]
sid=getID()
break[id==sid]
s
null

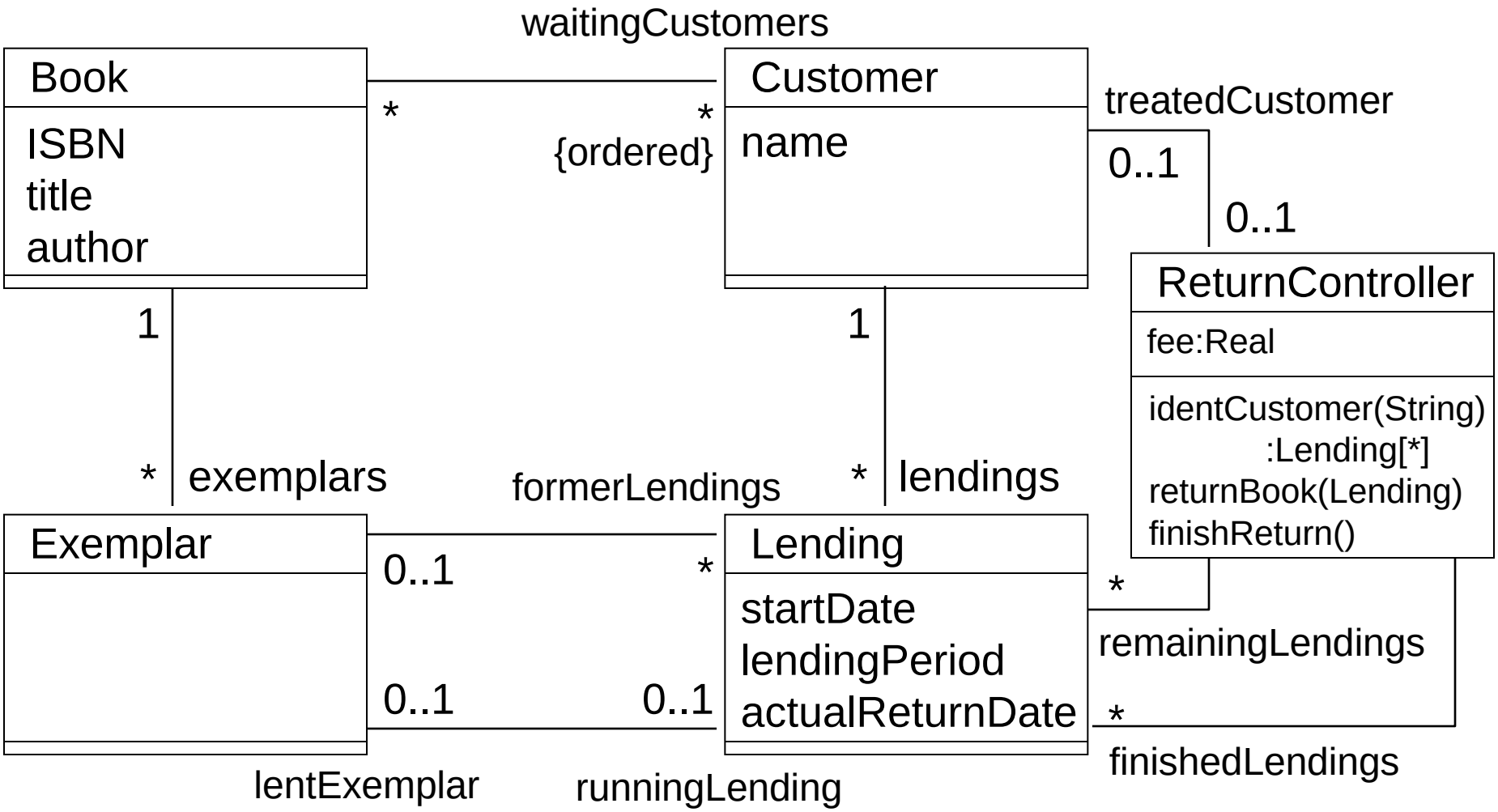

FindCourse
:Registrator
courses
c:Course
findCourse(id)
Loop[for each c in courses]
cid=getID()
break[id==cid]
c
null

Summary

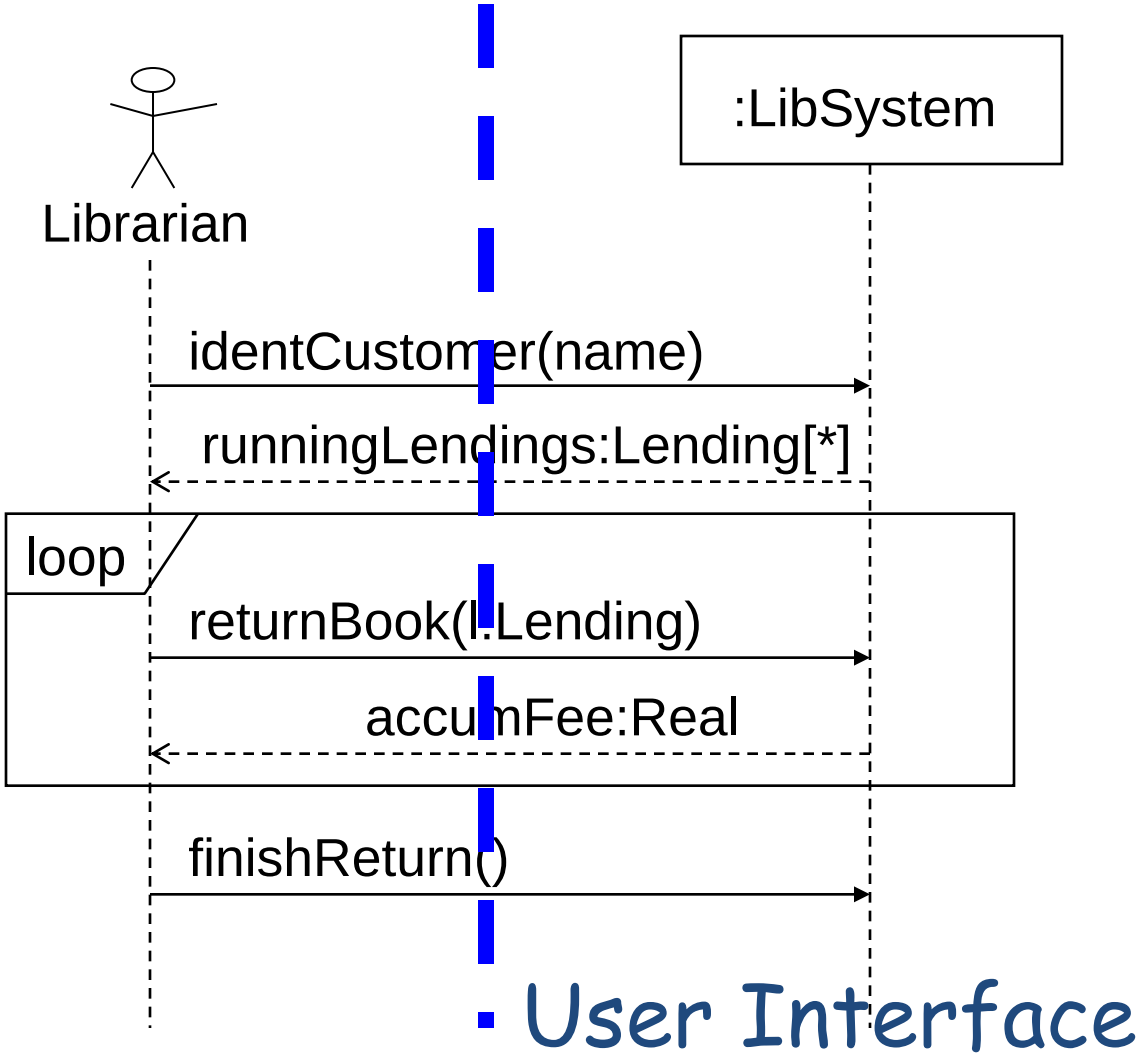
- One can produce the concept/class sequence diagram from
  - Use case
  - System sequence diagram
  - Contract of the system operation
  - Domain model (find concept)

The Same Less Abstract …

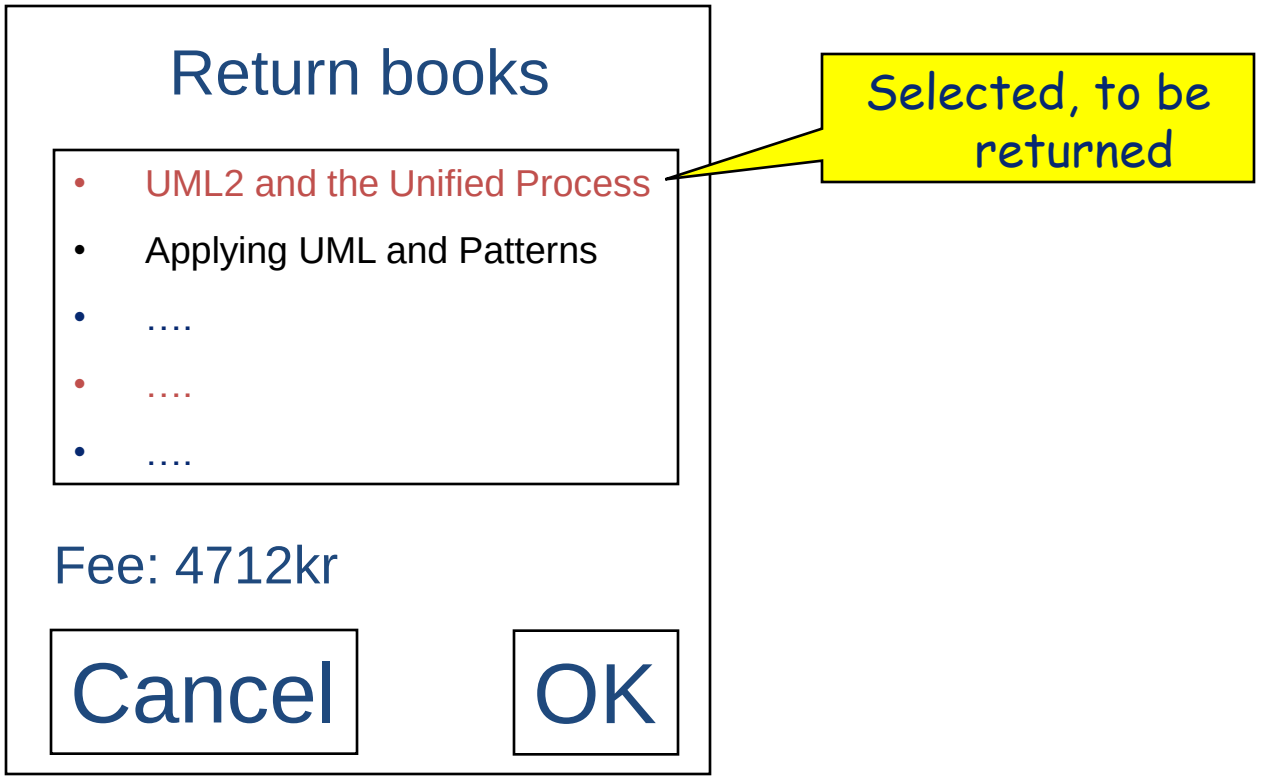
Initial Class Diagram



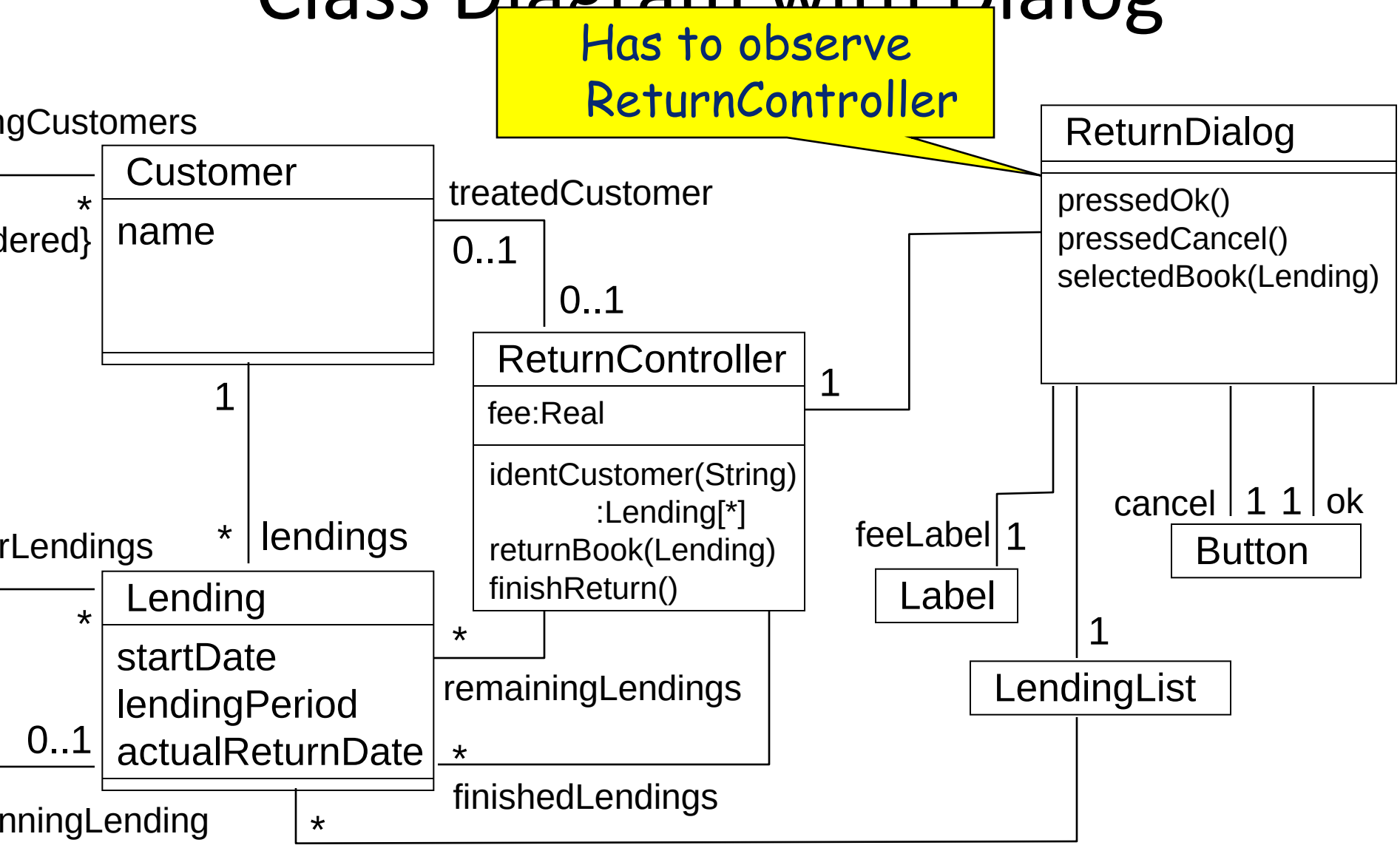
System Sequence Diagram



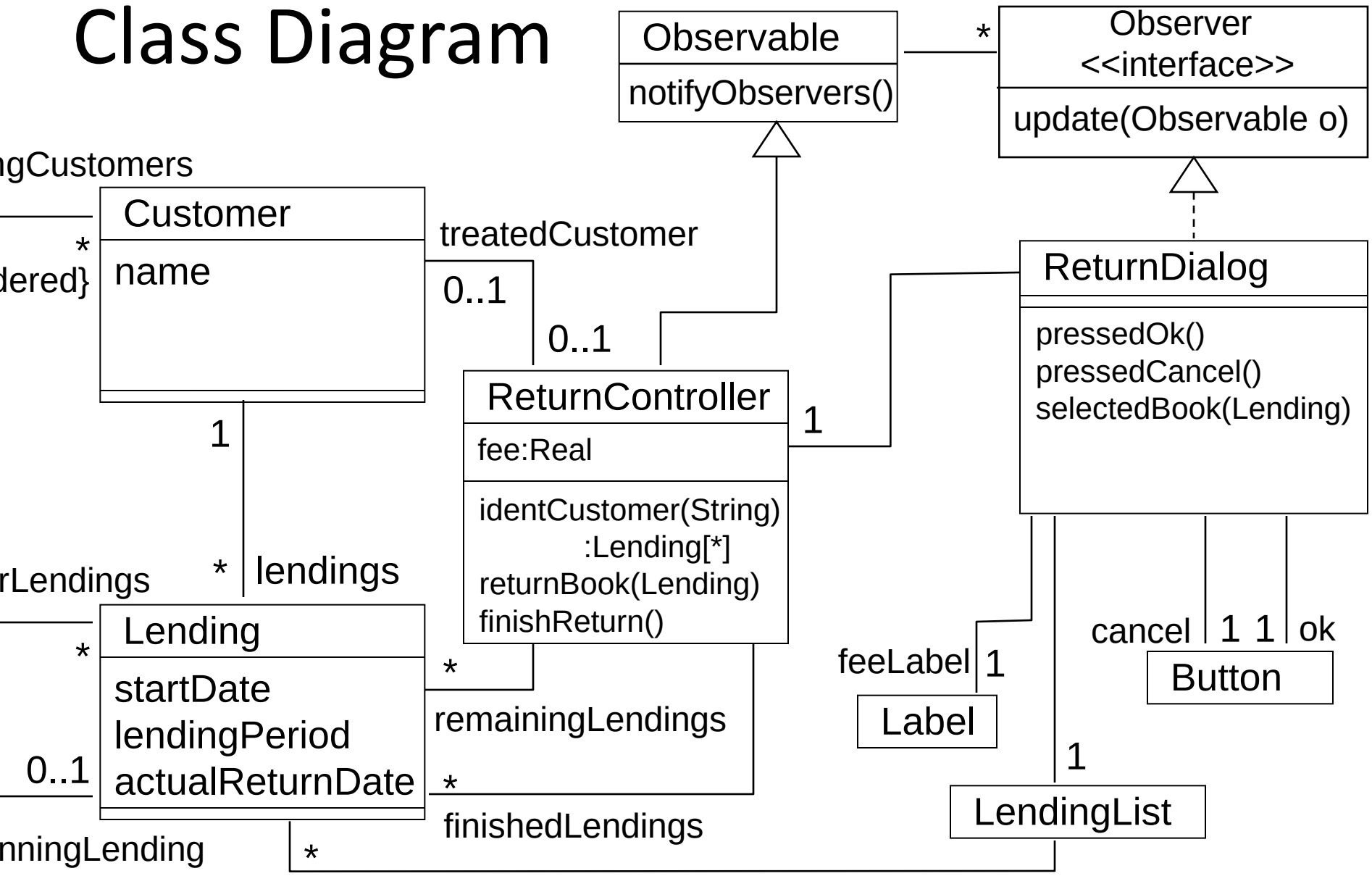
User Interface:
Dialog for Returning Books



Class Diagram with Dialog

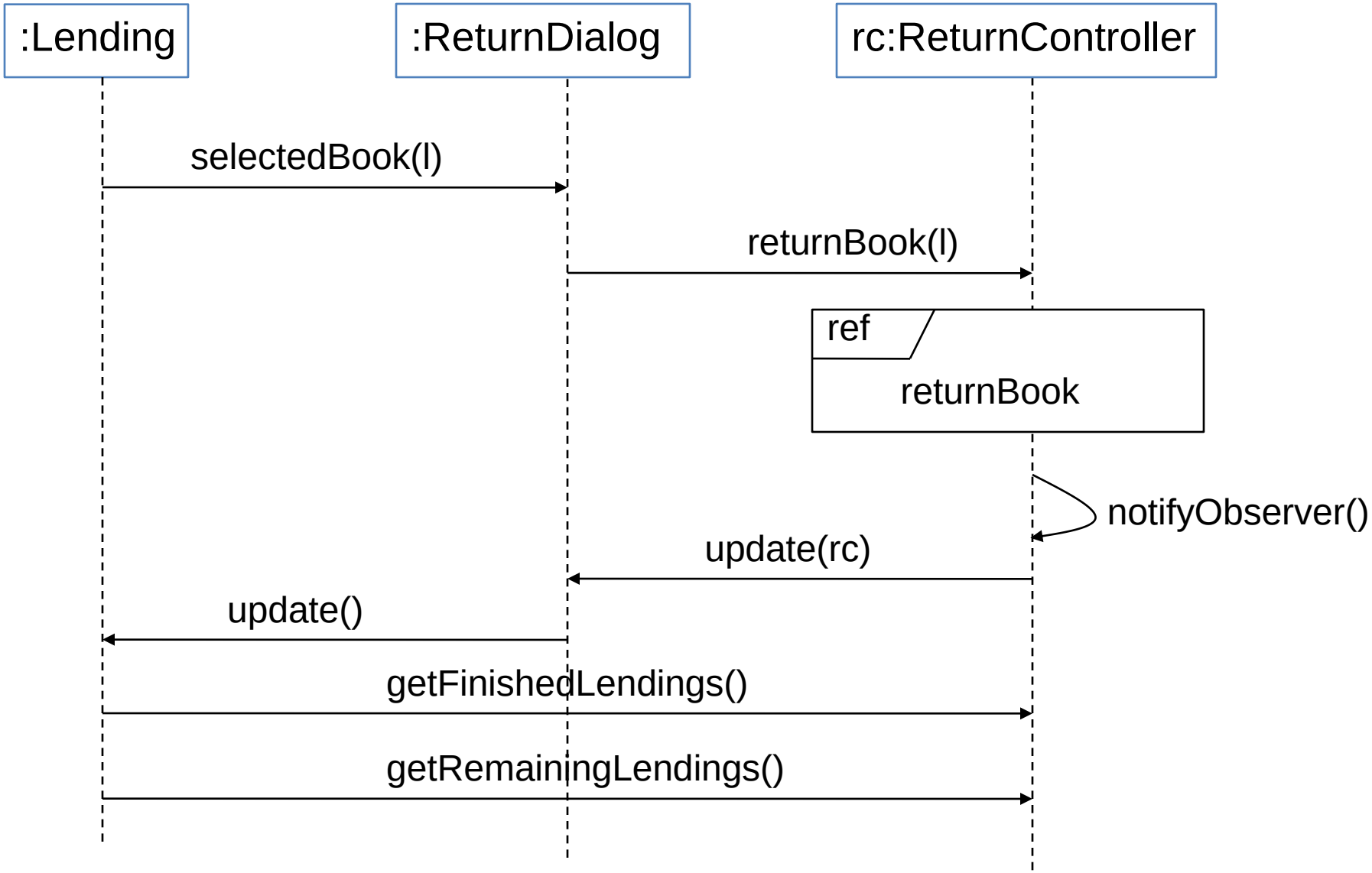


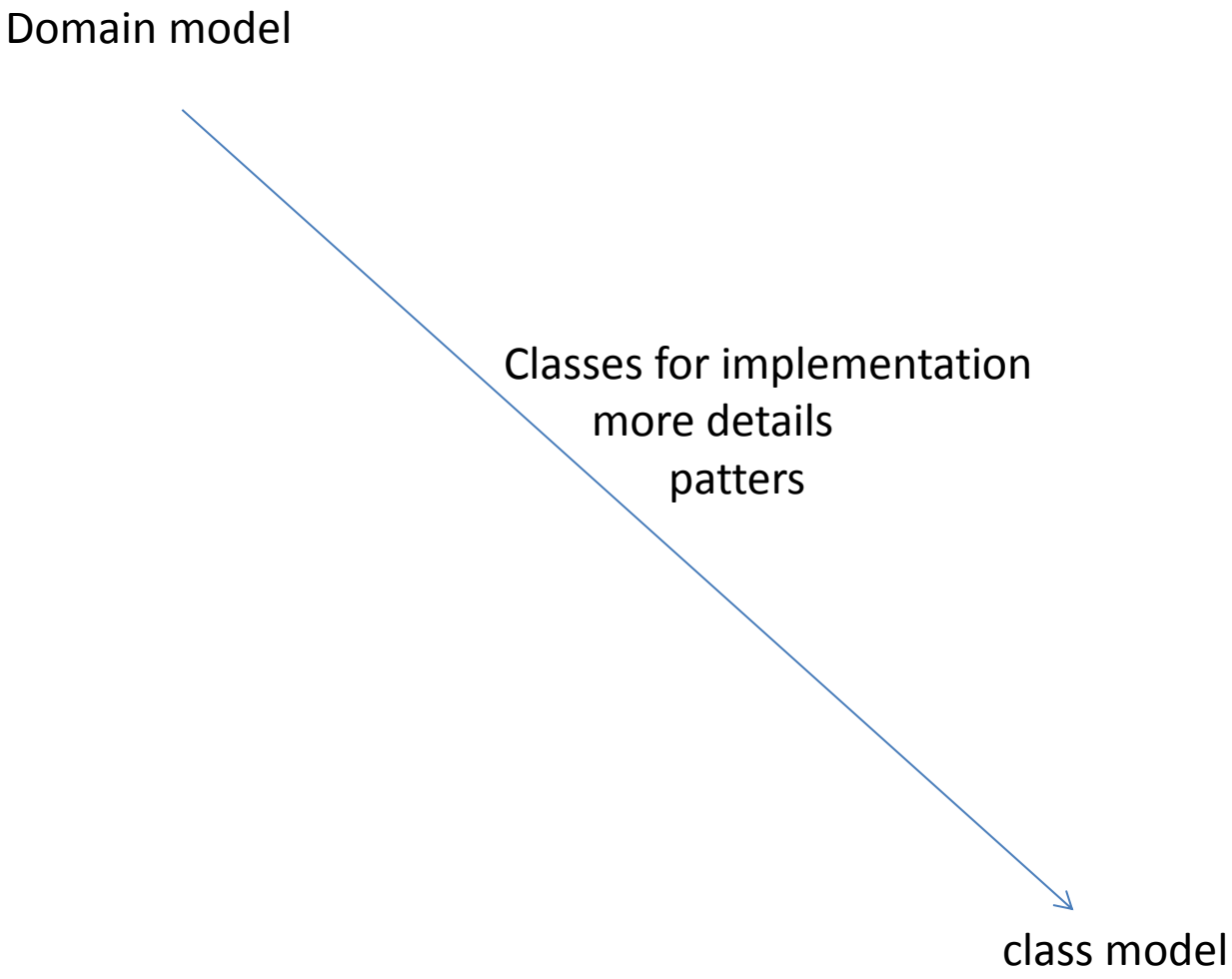
Class Diagram



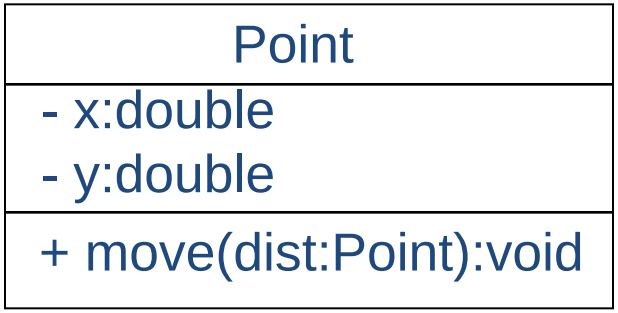
What happens when a book is selected in the dialog?

- Draw a sequence diagram …
- You can leave out those details of ReturnController:returnBook that were already shown earlier


:Lending
:ReturnDialog
rc:ReturnController
selectedBook(l)
returnBook(l)
ref
returnBook
notifyObserver()
update(rc)
update()
getFinishedLendings()
getRemainingLendings()


Domain model
Classes for implementation
more details
patters
class model

UML Classes: Visibility



Mapping visibility to java:

- **-** -> **private**
- **#** -> **protected**
- **+** -> **public**
- **~** -> **package**

(In this case the semantics of -,#,+,~ will be the one of Java.)

UML attribute

UML:

[visibility] name [multiplicity] [:type] [= initial value] [{properties}]

Properties could be:

- **changeable** (Variable may be changed.)
- **addOnly** (When **multiplicity** is bigger than one you can add more values, but not change or remove values.)
- **frozen** (Cannot be changed after it has been initialized.)

- **Example:**
  - **x : int {frozen}**

Operations/methods

UML:

[visibility] name [(parameter list)] [: return type] [{properties}]

You can have zero or more parameters. Syntax for parameters:

[direction] name : type [= default value]

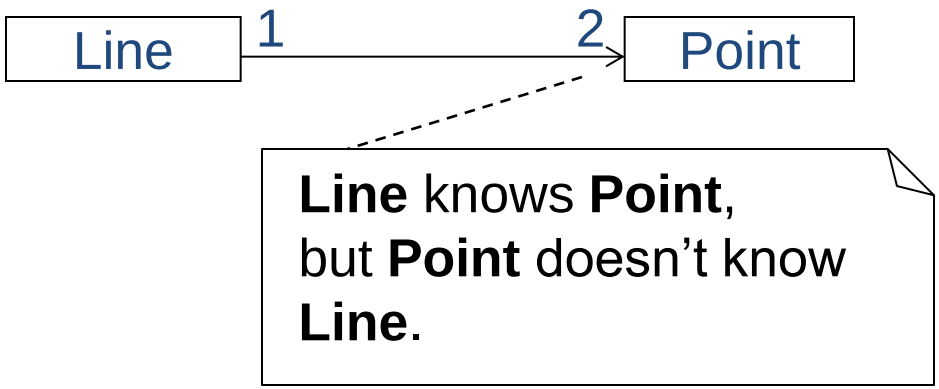
- direction: in, out, inout

- Example of a property
  - **isQuery** (no ”side effects”)

Relations

- All the associations we consider when drawing domain models can also be used in class diagrams.
- But there are some interesting issues to consider …

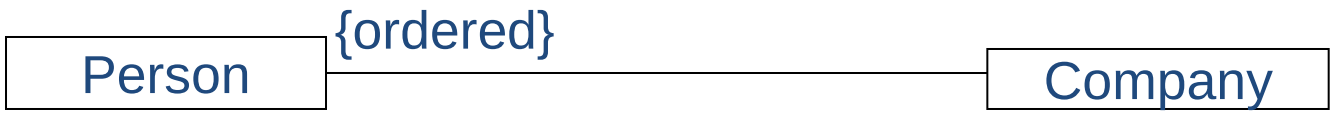
Navigability



Association constraint

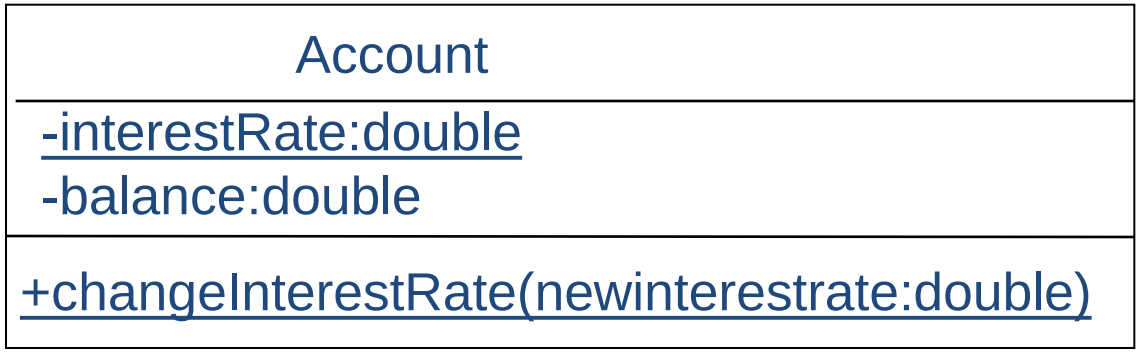
Constraint:

- **changeable** (Links may be changed.)
- **addOnly** (New links can be added by an object on the opposite side of the association.)
- **frozen** (When new links have been added from an object on the opposite side of the association, they cannot be changed.)
- **ordered** (Has a certain order)
- **bag** (multisets instead of sets)
- **…**



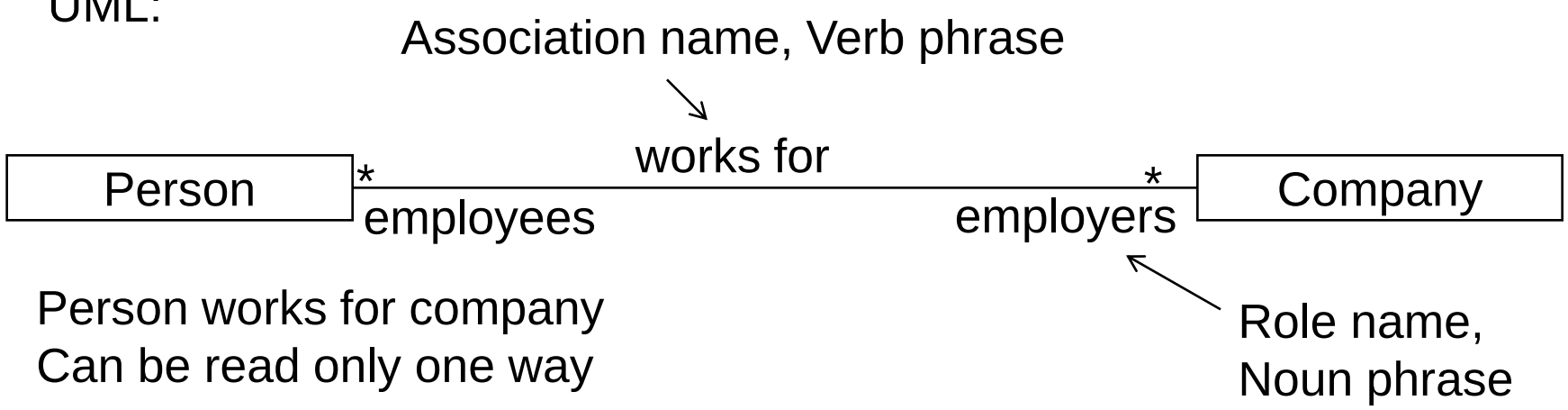
- BridgePoint do not contain properties

Class methods and class variables



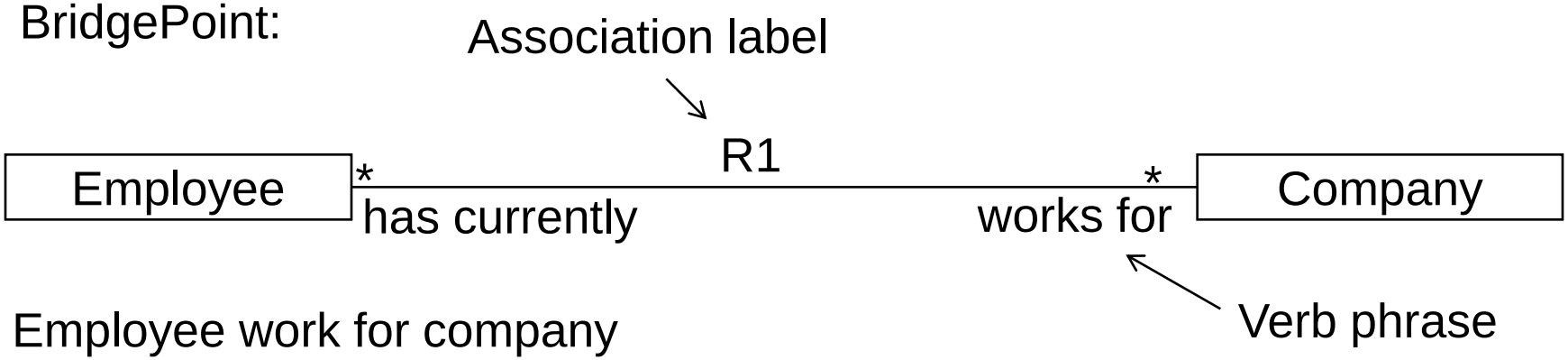
Association names UML

UML:



Association names BridgePoint

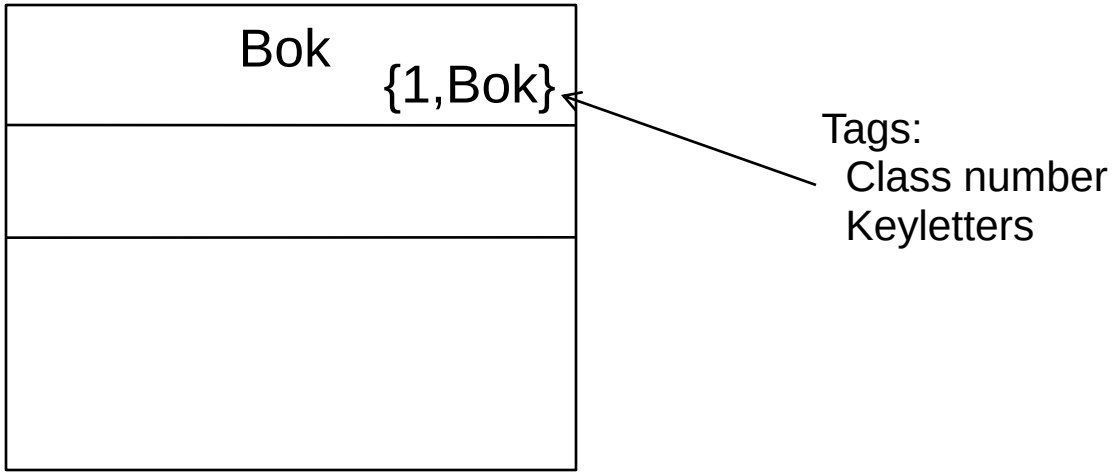
BridgePoint:



Employee work for company
Company has currently employees
Can be read in both direction

- Association label play a key role in BridgePoint.
- It is used in the OAL Object Action Language (see later slides).
  - To create link between objects (link is an instance of an association)
  - To obtain object over a link
- BridgePoint generate automatically association label.

Keyletters



Class number is produced automatically by BridgePoint. As far as I know we don't need to consider class numbers for the project.

A key letter will also be produces autmatically, but this one you should change to the same name as the class name!

- Keyletters is used in OAL

Missing elements

- At the moment BridgePoint do not handle class attribute and class operation and properties.

Class templates