CIS 771: Software Specifications

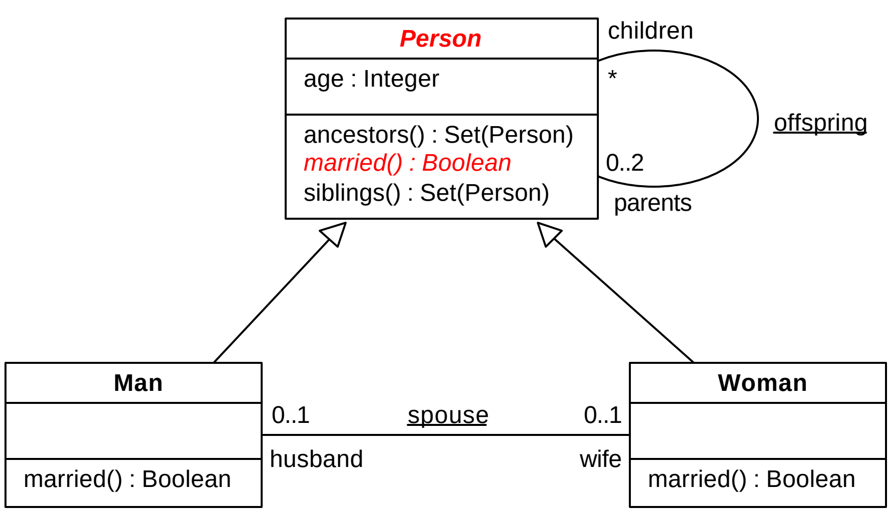
Lecture 13: Mining UML Class Diagrams for OCL Invariants

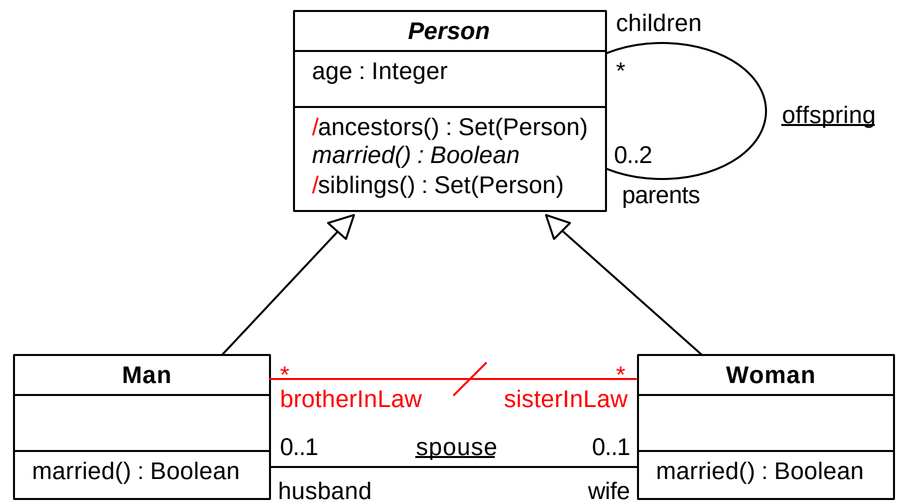
Identifying Constraints

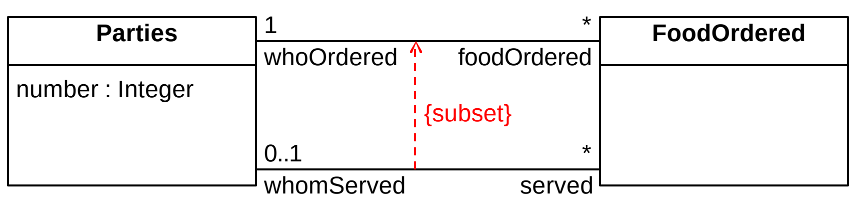
- OCL provides a language for defining invariants, but where do those invariants come from?
- Start from class diagram
  - Gives a vocabulary for your invariants
  - Gives very weak invariants, e.g., attribute types and association multiplicities
  - View constraint definition as enriching the class diagram
- What structural aspects of a class diagram suggest the need for a constraint?

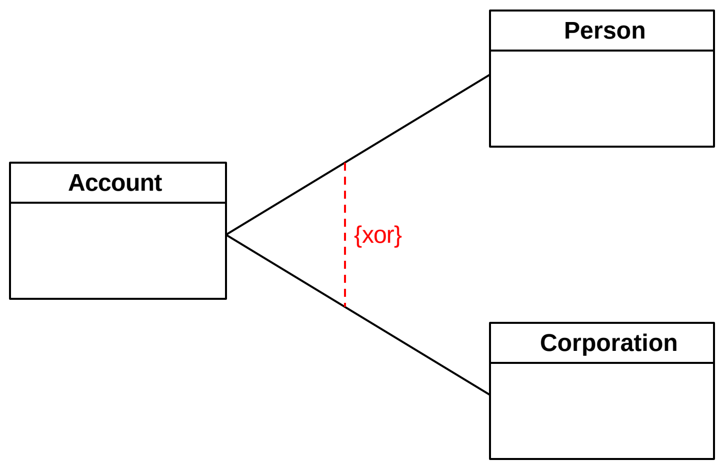
Class Diagrams Include...

- Subclass relationships
- Attributes
- Operations
- Associations with multiplicities
- Derived attributes/associations
- Certain relationships between associations




Person
age : Integer
/ancestors() : Set(Person)
married() : Boolean
/siblings() : Set(Person)
children
*
offspring
0..2
parents
Man
married() : Boolean
*
brotherInLaw
*
sisterInLaw
0..1
spouse
0..1
husband
wife
Woman
married() : Boolean


Parties
number : Integer
1
whoOrdered
*
foodOrdered
{subset}
0..1
whomServed
*
served
FoodOrdered



Advanced UML Features

- We haven't studied these UML features in detail
- Their semantics can be captured by OCL invariants

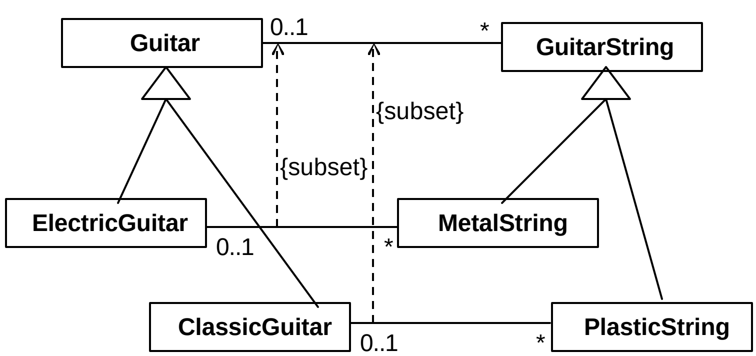
- Derived InLaw association

```
association InLaw between
   Man[*] role BrotherInLaw
   Woman[*] role SisterInLaw
end
```

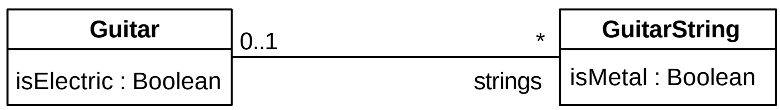
Advanced UML Features

```
context w:Woman
 inv BrotherInLaw:
  w.brotherInLaw =
  w.husband.parents.children->
    select(p | p.oclIsKindOf(Man) and
                p <> w.husband)->
   union(
    w.parents.children->
      select(s | s.oclIsKindOf(Woman) and
                  s <> w)->
      collect(s | s.oclAsType(Woman).husband))->asSet
```

Detailed Models...



…vs. Constraints




```
context Guitar
  inv ElectricHaveMetal:
    isElectric implies strings->forAll(s | s.isMetal)

  inv ClassicHavePlastic:
    not isElectric implies strings->forAll(s | not s.isMetal)
```

Types of Constraints

- Some constraints can be completely specified by the class diagram
  - E.g., abstract classes, subclasses, multiplicities
- Other constraints may be specified only partially or not at all by the class diagram
  - In what follows, we will focus on identifying these constraints

Uniqueness Constraints

- Several examples that we have looked at equate identity (in terms of some sort of ID) with equality
- Students and faculty have unique IDs

```
context Person
 inv UniqueIds:
   Person.allInstances->
     forAll(p1,p2 |
            p1.id <> p2.id implies p1 <> p2)
```

Uniqueness Constraints

- In general, we should ask the question, “Should equality imply identity?”
- In some cases, the answer is “No”
  - Two people may have the same name
  - Two people should not have the same Social Security Number
- One way to identify uniqueness is to look for “sharing”
  - If an instance can be shared (as the image of some relation), then it is not unique

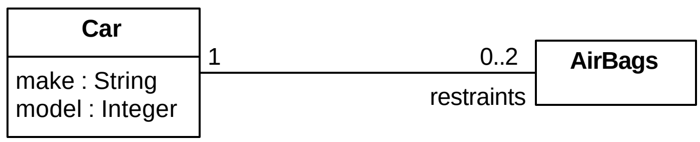
For You To Do (pause here)

- What kinds of “sharing” is present in the
  - Academia model
  - Railroad model
  - GUI model (from midterm)
- State the uniqueness constraints in these models
  - in English
  - in OCL

Optional Multiplicities

- A multiplicity by itself doesn’t always completely specify the size of an association
- Sometimes the values of other attributes or associations constrain the size of a particular association

Example

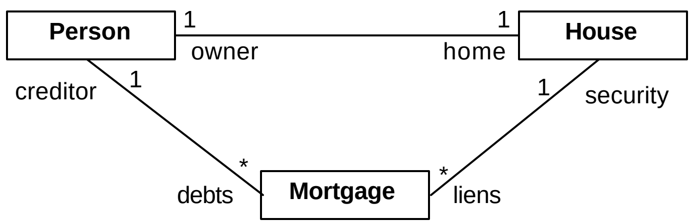



```
context Car
  inv PassiveRestraintLaw:
    model > 1996 implies restraints->notEmpty
```

Cycles in Class Diagrams

- Cycles imply that an object is related to itself
- This may be acceptable, unacceptable, or required
- We should always examine cycles to see if constraints need to be added to enforce the desired outcome
- Note that cycles need not be directed
  - Two navigation paths from same class that reach another class

Example

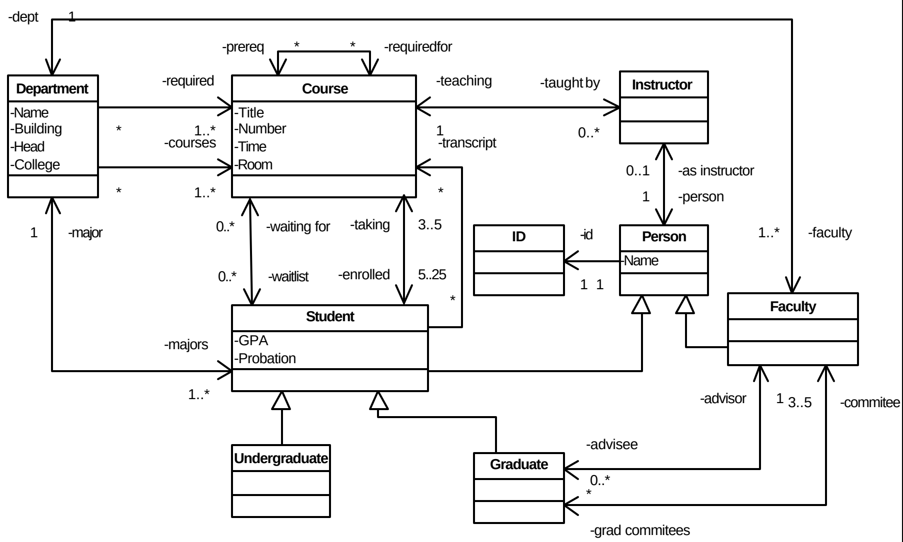



```
context Person
  inv OwnsMortgagedHouse:
    debts.security.owner->forAll(p | p = self)
```

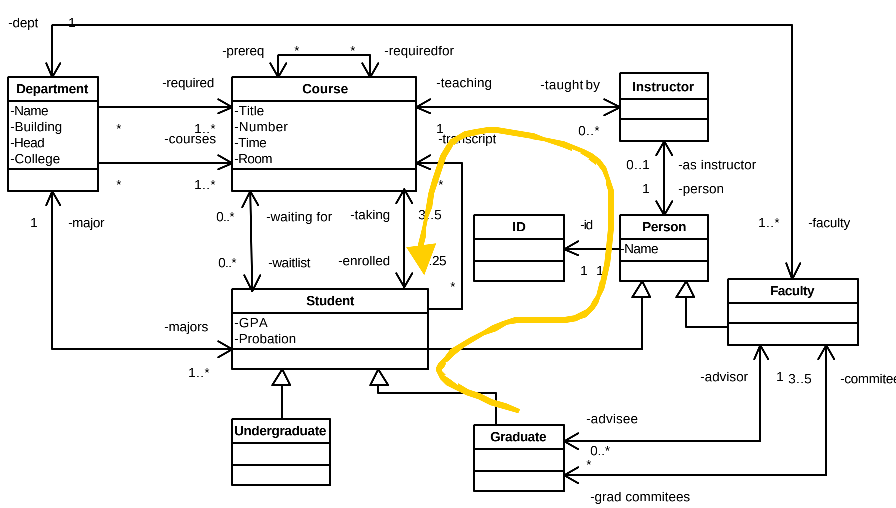
An Extended Example

- Recall the most complex version of the academia example:
  - Person, Student, Graduate, Undergrad, Faculty, Instructor
  - Department
  - Course
  - Id

Academia Class Diagram



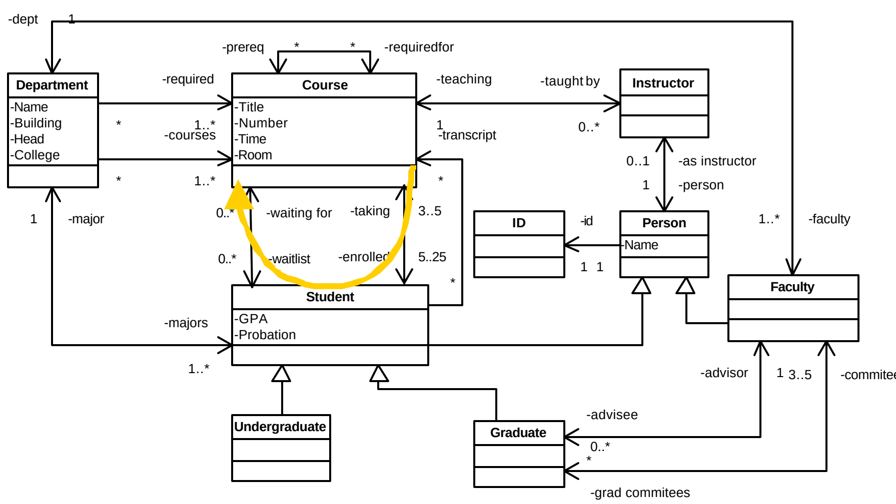
Self Teaching



No Teaching While Enrolled

```
context i:Instructor
  inv NoTeachingWhileEnrolled:
    i.instructor.oclIsKindOf(Grad) implies
    i.coursesTaught->intersection(
      i.instructor.oclAsType(Grad).taking)->
    isEmpty
```

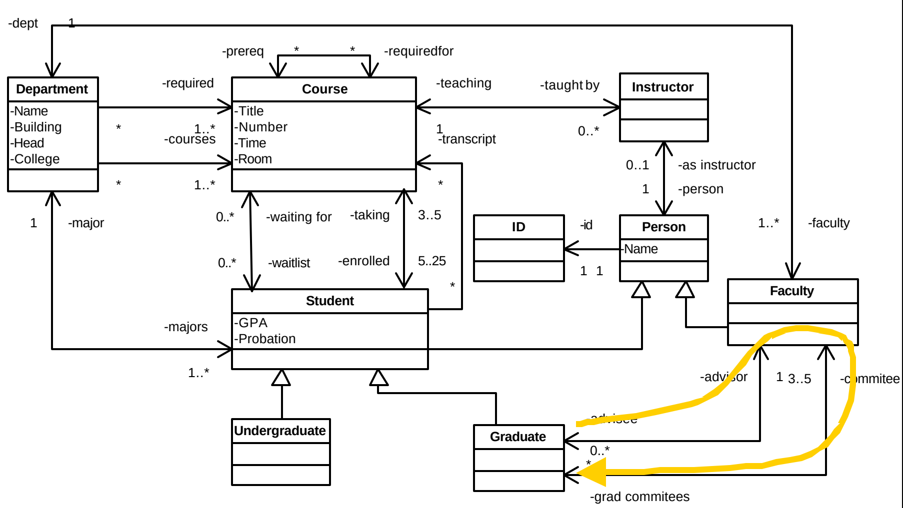
Enrolled and Waiting



No Waiting Unless Enrolled

```
context c:Course
  inv NoWaitingUnlessEnrolled:
    c.waitList->notEmpty implies
    c.enrolled->notEmpty
```

Advisor Chairs Committee

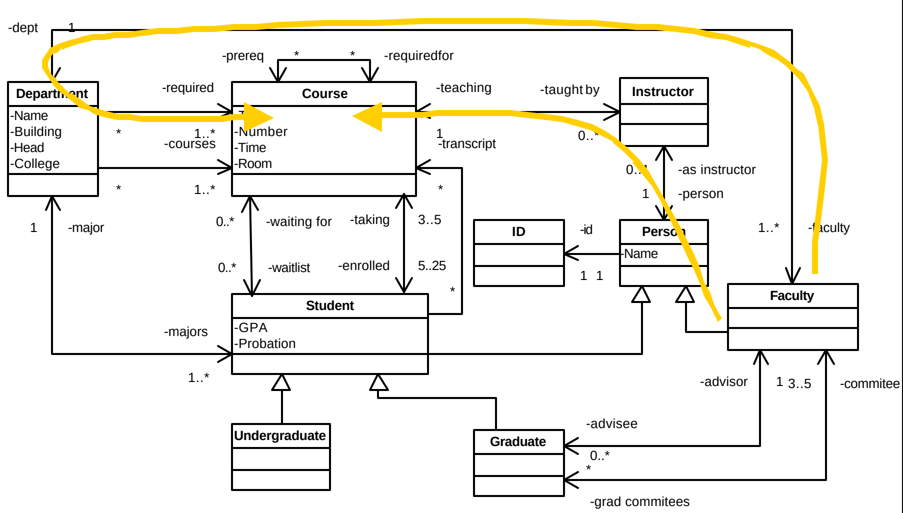


Advisor On Committee

```
context f:Faculty
  inv AdvisorOnCommittee:
    f.gradCommittees->
          includesAll(f.advisees)
```

Here we switch context for the specifying the invariant

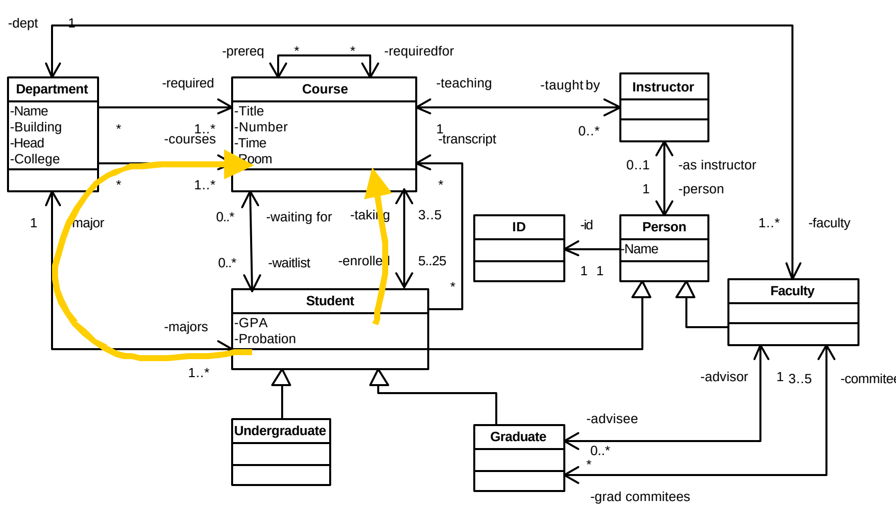
Faculty Teach Required Courses



Faculty Teach Required Courses

```
context d:Department
 inv FacultyTeachReqCourses:
  d.required.taughtBy->
  forAll(i:Instructor |
         i.instructor.oclIsKindOf(
           Faculty))
```

Students Take One Major Course



Students Take One Dept Course

```
context s:Student
 inv OneCourseFromMajor:
  s.taking->
   intersection(s.major.offers)->
     notEmpty()
```

For You To Do

- What other cycles can you find?
  - There are several more, e.g.,
    - Prerequisites required
    - No retaking courses
    - Required courses from major
- What invariants do they suggest?
  - Code those as OCL invariants and explore your sample model