

Building Models with OCL

- Introduction
- Completing UML Diagrams
- Modeling Tips and Hints
- Summary

What Is a Model?

Simply put, a **model** is a high level system description.

It consists of a consistent, coherent set of **model elements**, i.e., artifacts written in a well-defined modeling language such as UML and OCL.

Question: Why do we want to have a **model**?

View Consistency

An UML model consists of several diagrams, each of which presents one **view** of the model.

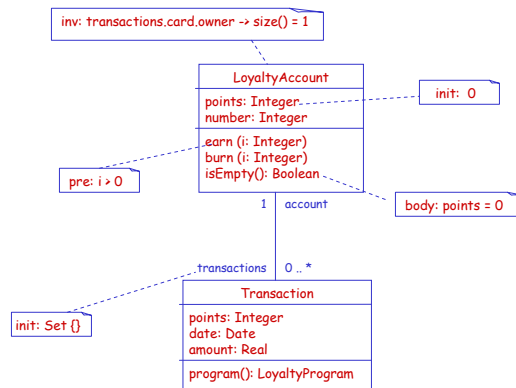
Multiple views must be **consistent**. For example, objects in the **interaction** diagram must be instances of classes present in the **class** diagram.

UML and OCL (1)

Both UML diagrams and OCL expressions are part of a model. They are connected by the **context** definition.

Note that OCL expressions can be written in a **textual** format or attached to the UML diagrams directly.

Example



Adding Extra Info

OCL expressions can add extra info. to UML diagrams in the following situations:

- Elements might be under-specified.
- Business rules need to be incorporated.
- Interfaces need to be precisely defined.
- Ambiguity must be removed.

Building Models with OCL

- Introduction
- **Completing UML Diagrams**
- Modeling Tips and Hints
- Summary

Discovering Constraints

OCL expressions can be used to specify constraints on a class diagram, but where do they come from?

- Uniqueness Constraints
- Optional & Dynamic Multiplicities
- Derived Attributes and Classes
- Inheritance
- Subset Constraints
- Association Cycles

Uniqueness Constraints (1)

Recall the following uniqueness constraint:

```
context Person
inv: Person::allInstances() -> isUnique (socSecNr)
```

Question: How to determine if the value of a attribute needs to be unique?

Uniqueness Constraints (2)

- In general, we should ask the question: "Should equality imply identity?"
 - If two people have the same name, does it mean they are the same person?
 - If two people have the same social security number, does it mean they are the same person?
- One way to identify uniqueness constraints is looking for "sharing"
 - If an instance can be shared, then it is not unique.

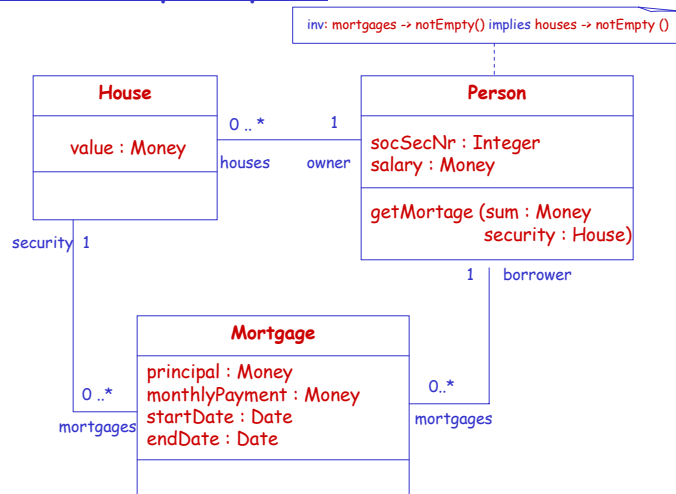
Optional Multiplicity (1)

An **optional multiplicity** indicates that an associated object may or may not exist.

The existence of an **associated** object often depends on the **state** of the objects involved.

In this case, it is necessary to use **OCL** constraints to describe precisely when the **optional** association may be empty.

Optional Multiplicity (2)

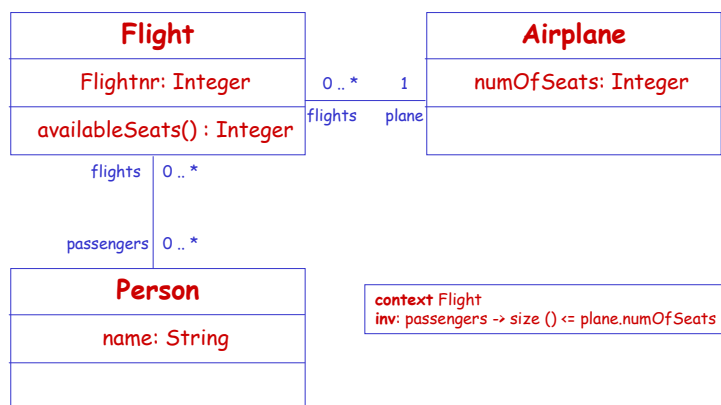


Dynamic Multiplicity (1)

A **dynamic multiplicity** is one that needs to be determined based on another value in the system.

In this case, **OCL expressions** can be used to precisely specify how to determine such a multiplicity.

Dynamic Multiplicity (2)



Derived Attributes & Classes

context TransactionReportLine::partnerName : String
derive: transaction.generatedBy.partner.name

context TransactionReportLine::serviceDesc : String
derive: transaction.generatedBy.description

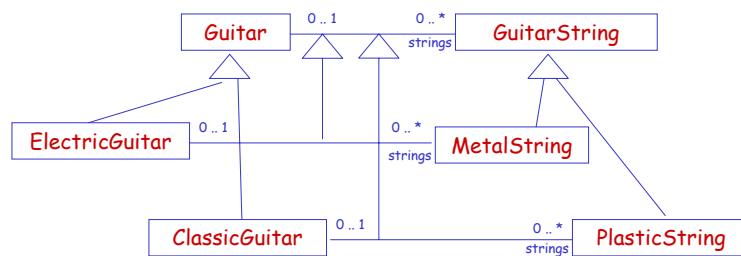
context TransactionReportLine::points : Integer
derive: transaction.points

context TransactionReportLine::amount : Real
derive: transaction.mount

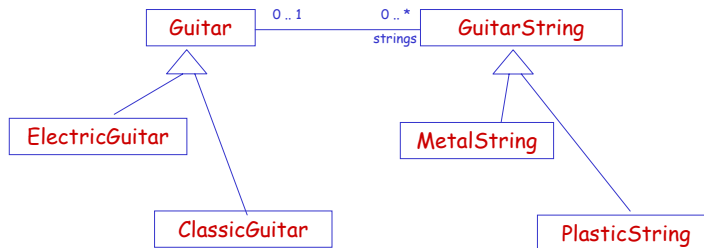
context TransactionReportLine::date : Date
derive: transaction.date

...
 ...

Inheritance (1)



Inheritance (2)



```

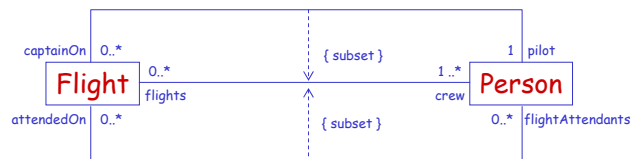
context ClassicGuitar
inv: strings -> forAll (s | s.oclIsTypeOf(PlasticString))
  
```

```

context ElectricGuitar
inv: strings -> forAll (s | s.oclIsTypeOf(MetalString))
  
```

Subset Constraints

A **subset** constraint indicates that the set of links for one association is a subset of the set of links for the other association.



```

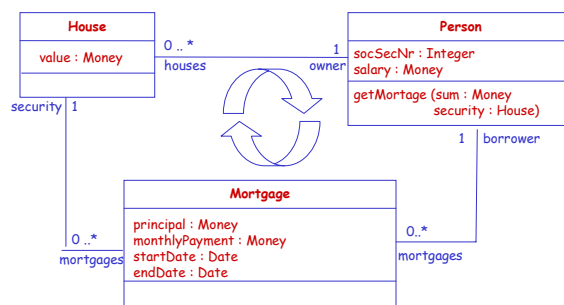
context Flight
inv: self.crew -> includes (self.pilot)
inv: self.crew -> includesAll (self.flightAttendants)
  
```

Association Cycles (1)

An **association cycle** consists of a set of **classes** and **associations** such that one can start from one class, navigate through the associations, and return to the same class.

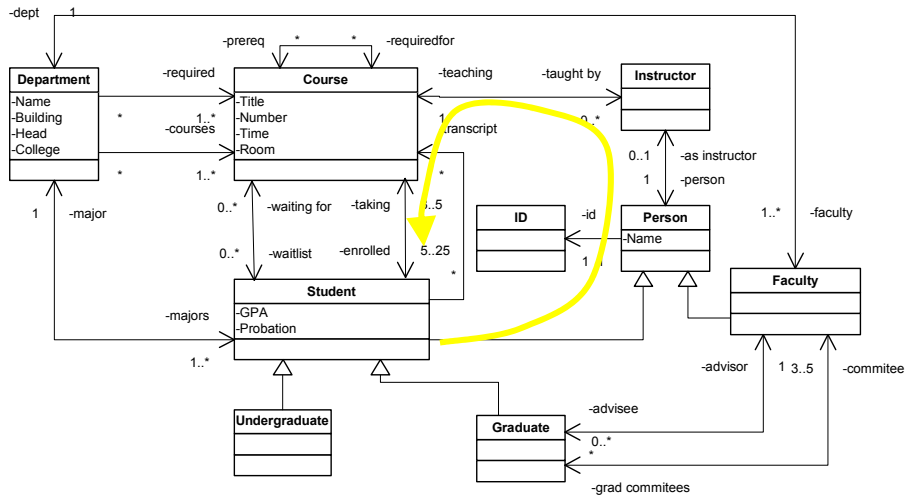
Cycles are often the source of ambiguity. The general rule is that **cycles should be checked carefully**, especially when the **multiplicities** are higher than one.

Association Cycles (2)



The model allows a person to have a mortgage that takes as security a house owned by another person!

Association Cycles (4)



From Drs. Dwyer and Hatcliff

Association Cycles (5)

The cycle allows a graduate student to teach and enroll a class at the same time. Therefore, it is desirable to impose the following invariants:

```

context Instructor
inv NoTeachingWhileEnrolled:
  self.person.oclIsKindOf(Graduate) implies
  teaching->intersection(
    self.person.taking)-> isEmpty ()
  
```

```

context Course
inv NoTeachingWhileEnrolled:
  enrolled->intersection(
    taughtBy.person -> select
    (isOclKindOf (Graduate))) -> isEmpty ()
  
```

oclIsTypeOf vs oclIsKindOf

The **oclIsTypeOf** operation returns true only if the type of the object is identical to the argument.

The **oclIsKindOf** operation returns true if the type of object is identical to the argument, or identical to any of the subtypes of the argument.

```
context Person
inv: self.oclIsKindOf(Person) = ?
inv: self.oclIsTypeOf(Person) = ?
inv: self.oclIsTypeOf(Student) = ?
inv: self.oclIsKindOf(Student) = ?
```

```
context Student
inv: self.oclIsKindOf(Person) = ?
inv: self.oclIsTypeOf(Person) = ?
inv: self.oclIsTypeOf(Student) = ?
inv: self.oclIsKindOf(Student) = ?
inv: self.oclIsTypeOf(Faculty) = ?
inv: self.oclIsKindOf(Faculty) = ?
```

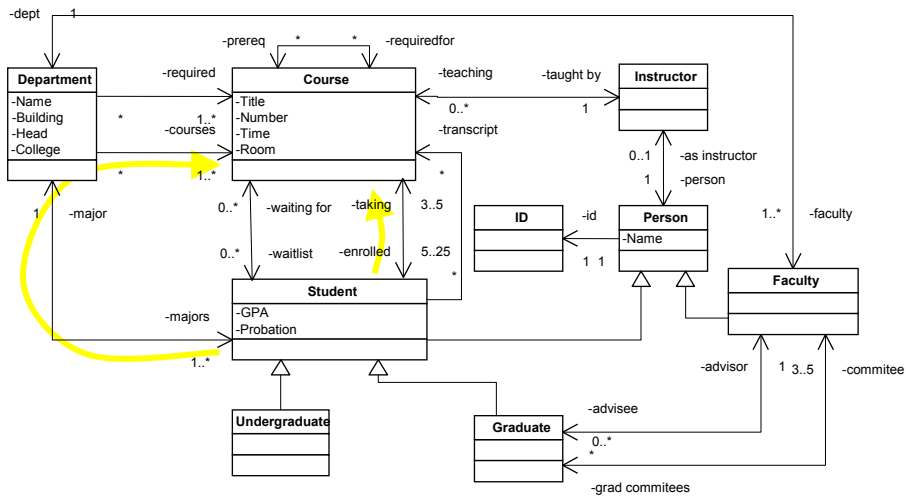
Undefined Value

Some OCL expressions will have an **undefined** value when evaluated.

OCL provides a predefined operation, **isUndefined**, that returns **true** if the argument is **undefined**, and **false** otherwise.

An expression is **undefined** if any sub-expression is **undefined**, except the following: (1) **True** or **undefined = True**; (2) **False** and **undefined = False**; (3) **False implies undefined = True**.

Association Cycles (8)



From Drs. Dwyer and Hatcliff

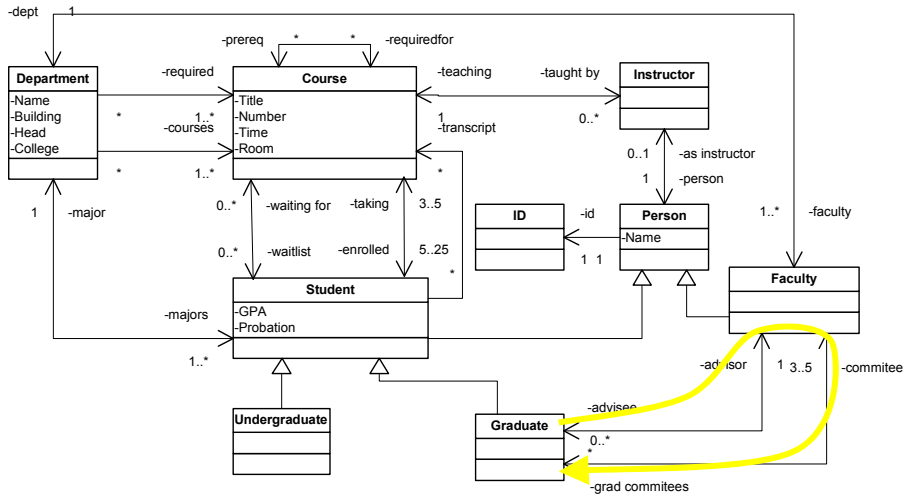
Association Cycles (9)

Each student should take at least one major course:

```

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

Association Cycles (11)



From Drs. Dwyer and Hatcliff

Association Cycles (12)

An advisor must be on the committees of his advisees:

```

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

Design by Contract

A **contract** is a **precise** specification of mutual **obligations** and **benefits** between the client and the supplier of a service.

Design by contract is a method that emphasizes the **precise** description of interface **semantics**. An object is responsible for executing a service if and only if certain conditions are fulfilled.

An Example Contract

Client (Traveler)

- **Obligation**
 - check in 10 minutes before boarding
 - < 3 small carry-ons
 - buy ticket
- **Benefit**
 - reach Boston

Supplier (Airline)

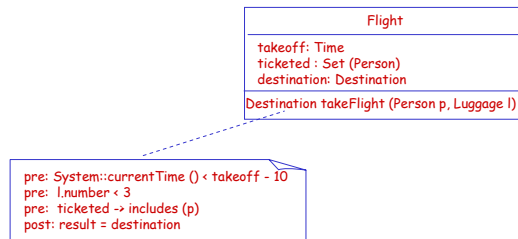
- **Obligation**
 - take traveler to Boston
- **Benefit**
 - don't need to wait for late travelers
 - don't need to store arbitrary amounts of luggage

Why Contract?

- Facilitates design
 - establishes the responsibilities of each object clearly and precisely.
- Simplifies implementation
 - allows certain assumptions to be made.
- Helps debugging
 - clearly identifies the responsible party in case of a failure
- Enables automatic reasoning
 - tools can be developed to automatic checking the conformance of a contract

Contract in OCL

In OCL, a **contract** is specified using **pre-/post-conditions**.



Building Models with OCL

- Introduction
- Completing UML Diagrams
- Modeling Tips and Hints
- Summary

Navigation

- Avoid complex navigation expressions
 - makes them easier to read and understand
 - is consistent with the encapsulation principle

```
context Membership
inv: programs.partners.deliveredServices ->
    forAll (pointsEarned = 0) implies account -> isEmpty ()
```

```
context LoyaltyProgram
def: isSaving : Boolean =
    partners.deliveredServices -> forAll (pointsEarned = 0)
```

```
context Membership
inv: programs.isSaving implies account -> isEmpty ()
```

Choose Context Wisely (1)

- ❑ If the invariant restricts the value of an attribute of one class, the class containing the attribute is a clear candidate.
- ❑ If the invariant restricts the values of attributes of more than one class, the class containing any of the attributes are candidates.
- ❑ If a class can be appointed the responsibility for maintaining the constraint, the class should be the context.
- ❑ Any invariant should navigate through the smallest possible number of associations.

Choose Context Wisely (2)



Consider how to write the constraint that two persons who are married to each other are not allowed to work at the same company.

```

context Company
inv: employees.wife -> intersection(employees) -> isEmpty()
  
```

Split and Constraints

- A large invariant can often be split into smaller ones at the **and** operations
 - easier to read and write
 - localizes the problem in case of a broken invariant
 - easier to maintain

```
context LoyaltyProgram
inv: participants -> forAll (age() > 0)
and
partners.deliveredServices -> forAll (pointsEarned = 0)
```

```
context LoyaltyProgram
inv: participants -> forAll (age() > 0)
inv: partners.deliveredServices -> forAll (pointsEarned = 0)
```

Use the collect shorthand

- Use the **collect** shorthand whenever possible
 - the shorthand is easy to read and understand

```
context Person
inv: parents -> collect (brothers) -> collect (children) -> notEmpty ()
```

||

```
context Person
inv: parents.brothers.children -> notEmpty ()
```

Always Name Association Ends

- Naming association ends is a good practice
 - indicates the purpose of the associated object
 - suggests the attribute name to maintain the association

Building Models with OCL

- Introduction
- Completing UML Diagrams
- Modeling Tips and Hints
- Summary

Summary

- ❑ A model allows one to focus on high level design decisions, rather than low level details.
- ❑ One way to identify constraints is to look for optional/dynamic multiplicities and cycles.
- ❑ OCL expressions can also be used to simplify a class diagram.
- ❑ Design by contract allows the interface of an object to be precisely specified.
- ❑ The rule of thumb for writing a good OCL spec is to keep the expressions as simple as possible.