ID2207 – Fourth Tutorial

Amirhossein Layegh

amlk@kth.se

2025-09-23

Object Constraint Language OCL

Object Constraint Language OCL

- Constraint-> is a rule that allows you to specify some limits on model elements.
- It can also be defined as a restriction on one or more values of (part of) an object-oriented model or system.
- OCL -> A language that express the constraints.

Object Constraint Language OCL

- Three types of constraints:
  - Invariant ->
  - -
  - -

Object Constraint Language OCL

- Three types of constraints:
  - Invariant -> A property that must remain true throughout the life of an object.
  - Pre-condition ->
  - -

Object Constraint Language OCL

- Three types of constraints:
  - Invariant -> A property that must remain true throughout the life of an object.
  - Pre-condition -> Something that must be true before a particular part of the system is executed. Prevent the system to enter illegal state.
  - Post-condition ->

Object Constraint Language OCL

- Three types of constraints:
  - Invariant -> A property that must remain true throughout the life of an object.
  - Pre-condition -> Something that must be true before a particular part of the system is executed.
  - Post-condition -> Something that must be true after a particular part of the system is executed.

Invariant Example

- The age of a car sharer that we analysed in Tutorial 2, must be over 21:

  Context

Invariant Example

- The age of a car sharer that we analysed in Tutorial 2, must be over 21:

```
Context CarSharer inv:
        age > 21
```

The keyword **context** introduces a model element to which the invariant applies (the class name in this example), and **inv** introduces the constraint as an invariant.

Pre-condition Example

- Making sure that enough money is in a savings account before making a withdrawal:

Pre-condition Example

- Making sure that enough money is in a savings account before making a withdrawal:

Context Account::withdraw (amount:Integer)

pre: balance >= amount

Post-condition Example

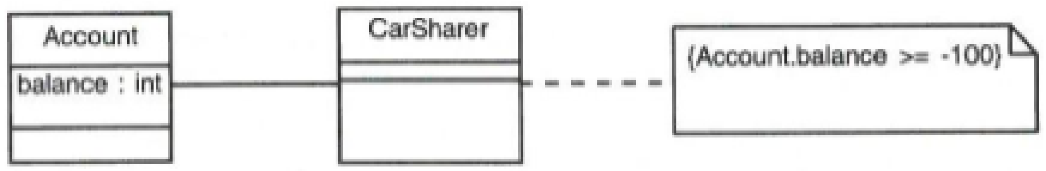
- An account's balance must be reduced by the amount withdrawn after a withdrawal

Context Account::withdraw (amount:Integer)
    post: balance = balance@pre - amount

@pre shows the initial value of a property that is changed by an operation

Navigation

- Navigate to an element of a model.



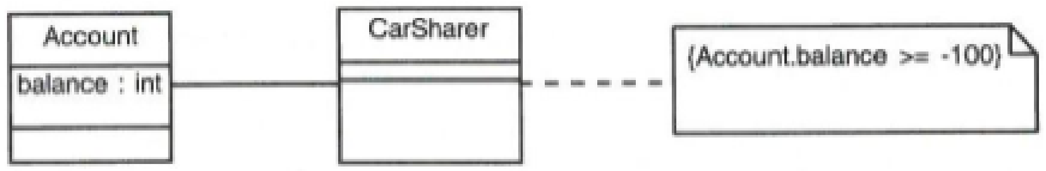
- To refer to a car sharer’s account balance:
  - the balance should be no more than 100$ overdrawn

Context Carsharer **inv:**

Account.balance >= -100

Navigation

- Navigate to an element of a model.



- To refer to a car sharer’s account balance:
  - the balance should be no more than 100$ overdrawn

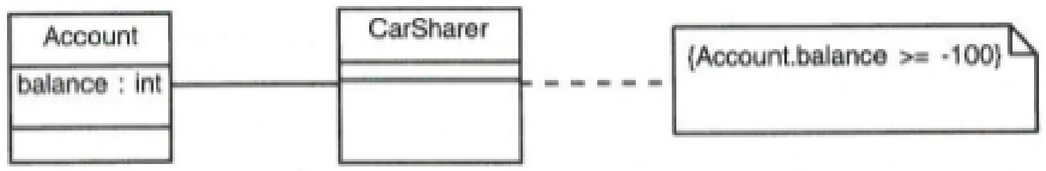
Context Carsharer **inv:**

    Account.balance >= -100

If the expression is complex, and you want to avoid ambiguity, the word **self** can be used to refer to the instance of the class that is named as the context.

Navigation

- Navigate to an element of a model.



- To refer to a car sharer's account balance:
  - the balance should be no more than 100$ overdrawn

Context Carsharer **inv:**
Account.balance >= -100

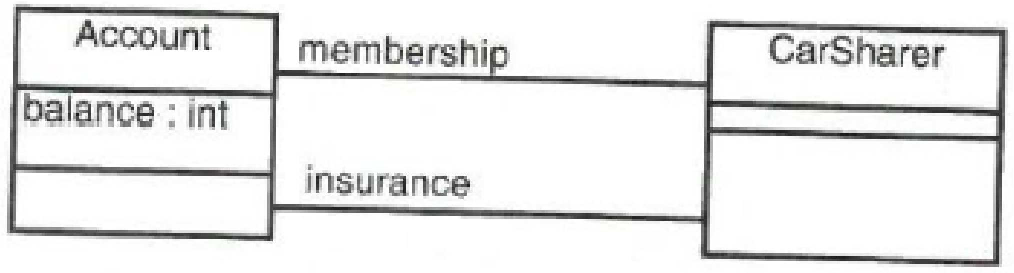
↓

Context Carsharer **inv:**
self.Account.balance >= -100

If the expression is complex, and you want to avoid ambiguity, the word **self** can be used to refer to the instance of the class that is named as the context.

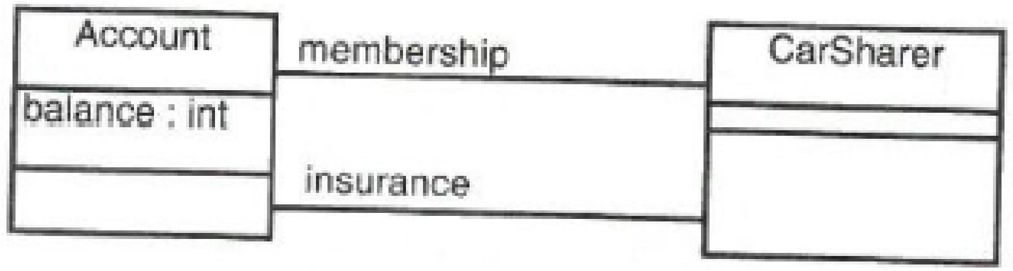
Navigation Example

- Suppose there are two types of accounts: membership and insurance and you want to indicate that the insurance account must not be overdrawn by more than 500$.



Navigation Example

- Suppose there are two types of accounts: membership and insurance and you want to indicate that the insurance account must not be overdrawn by more than 500$.

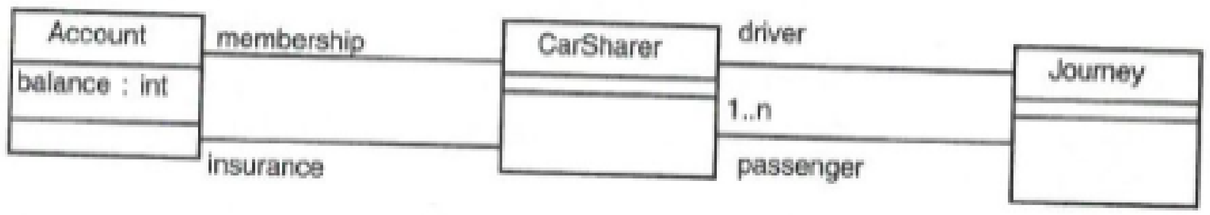


Context Carsharer **inv:**

Insurance.balance >= -500

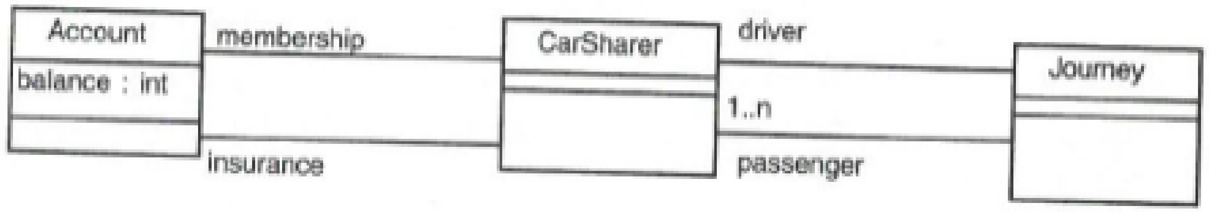
Navigation Example

- It is also possible to navigate across a number of associations.
- The driver in a journey has paid the insurance:



Navigation Example

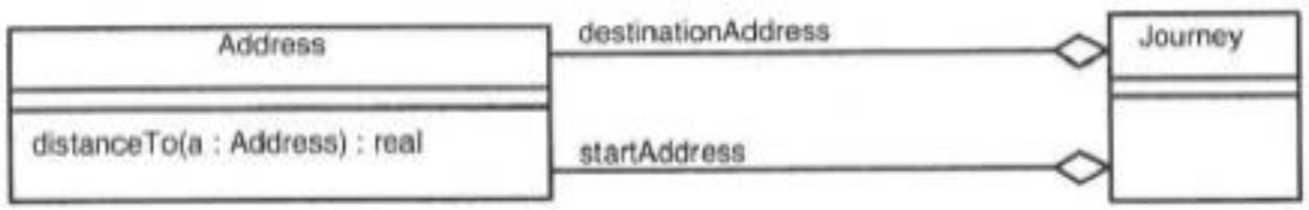
- It is also possible to navigate across a number of associations.
- The driver in a journey has paid the insurance:



Context Journey **inv:**
    driver.insurance.balance >= 0

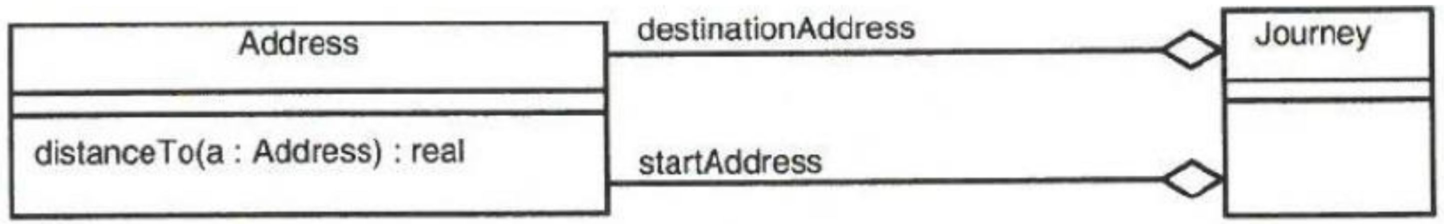
Navigation Example

- express that the distance of any journey is two miles or more:



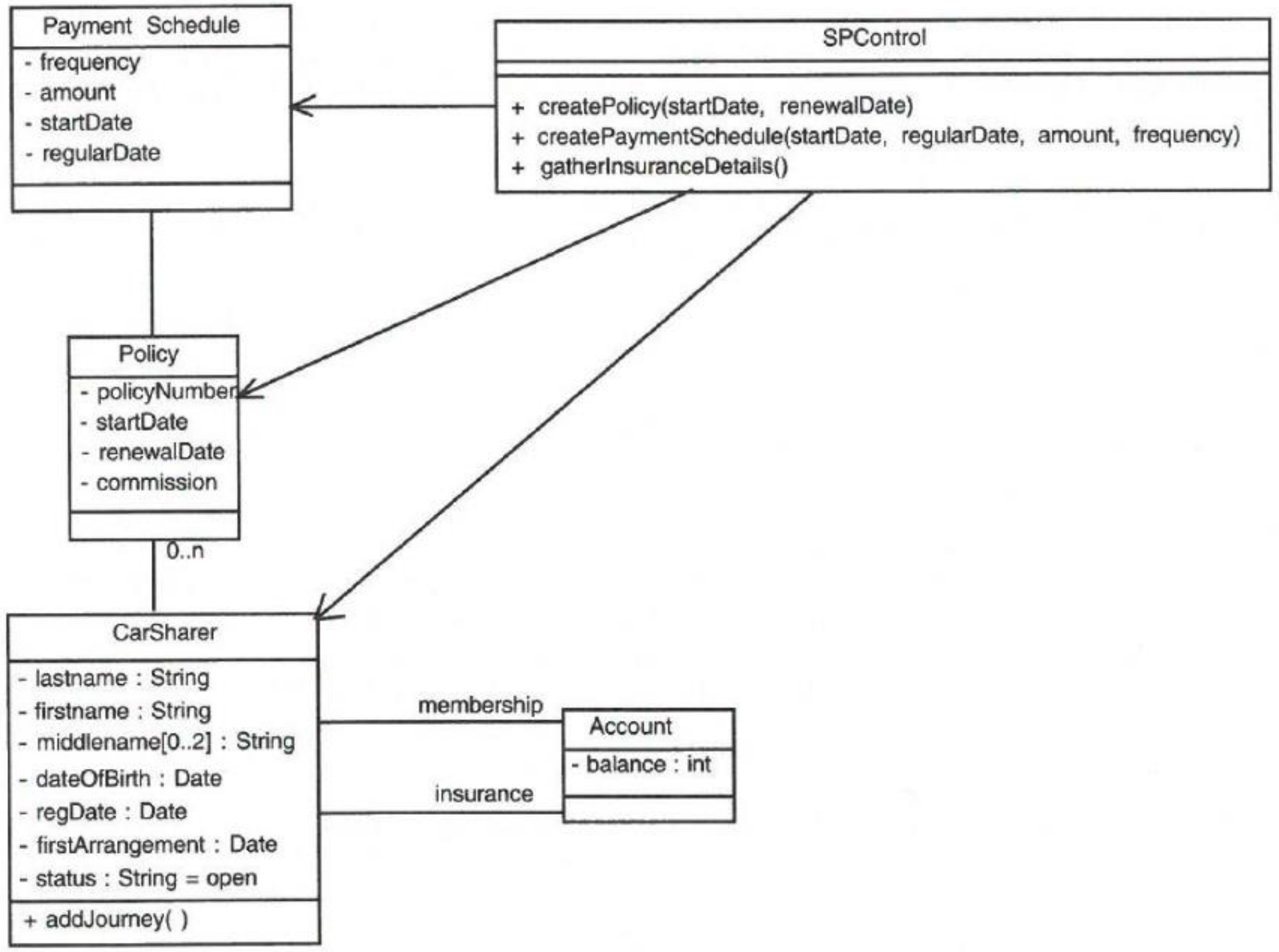
Navigation Example

- express that the distance of any journey is two miles or more:



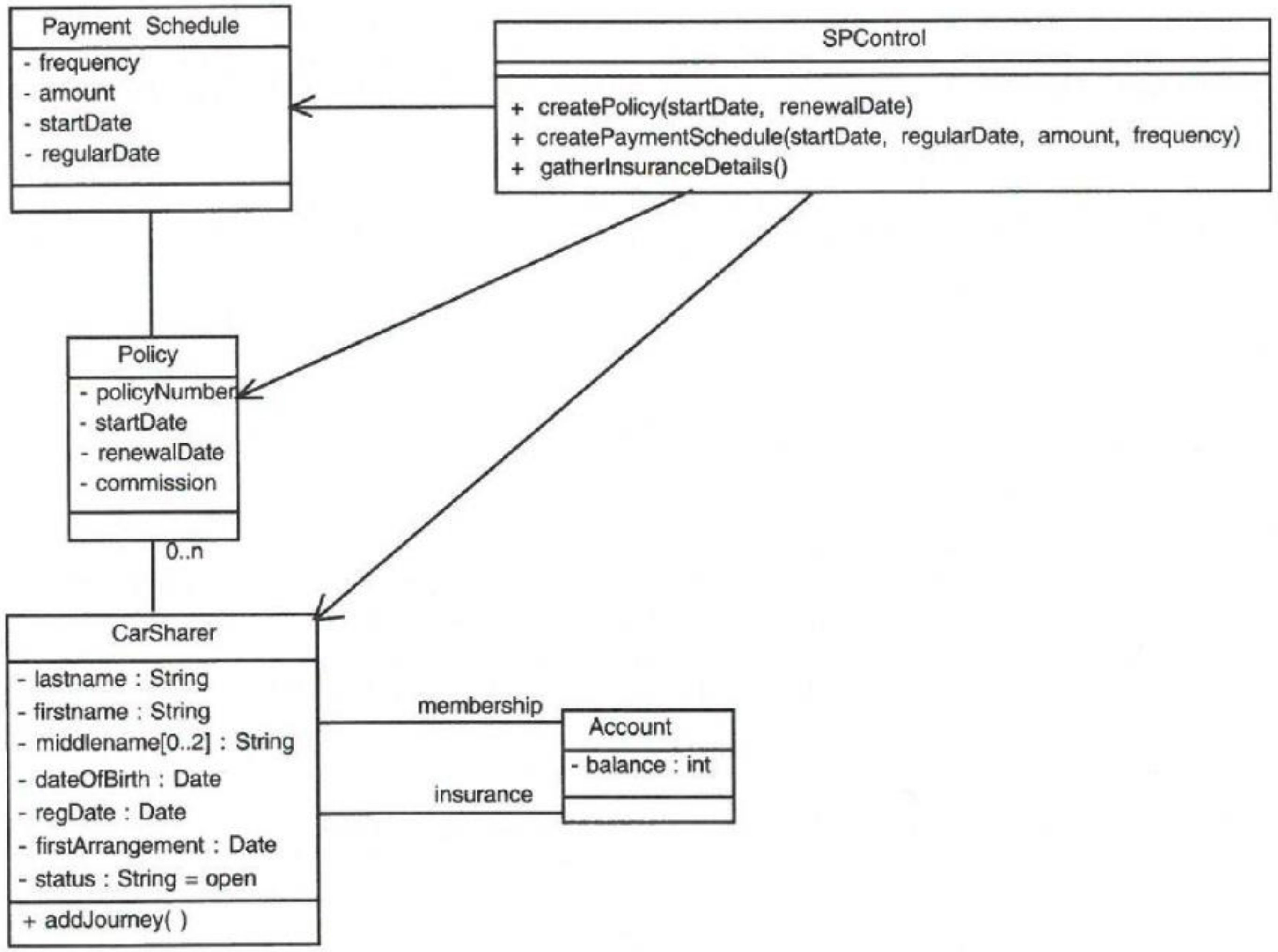
Context Journey **inv:**
    startAddress.distanceTo(self.destinationAddress) >= 2.0

Exercise One: Insurance Policies



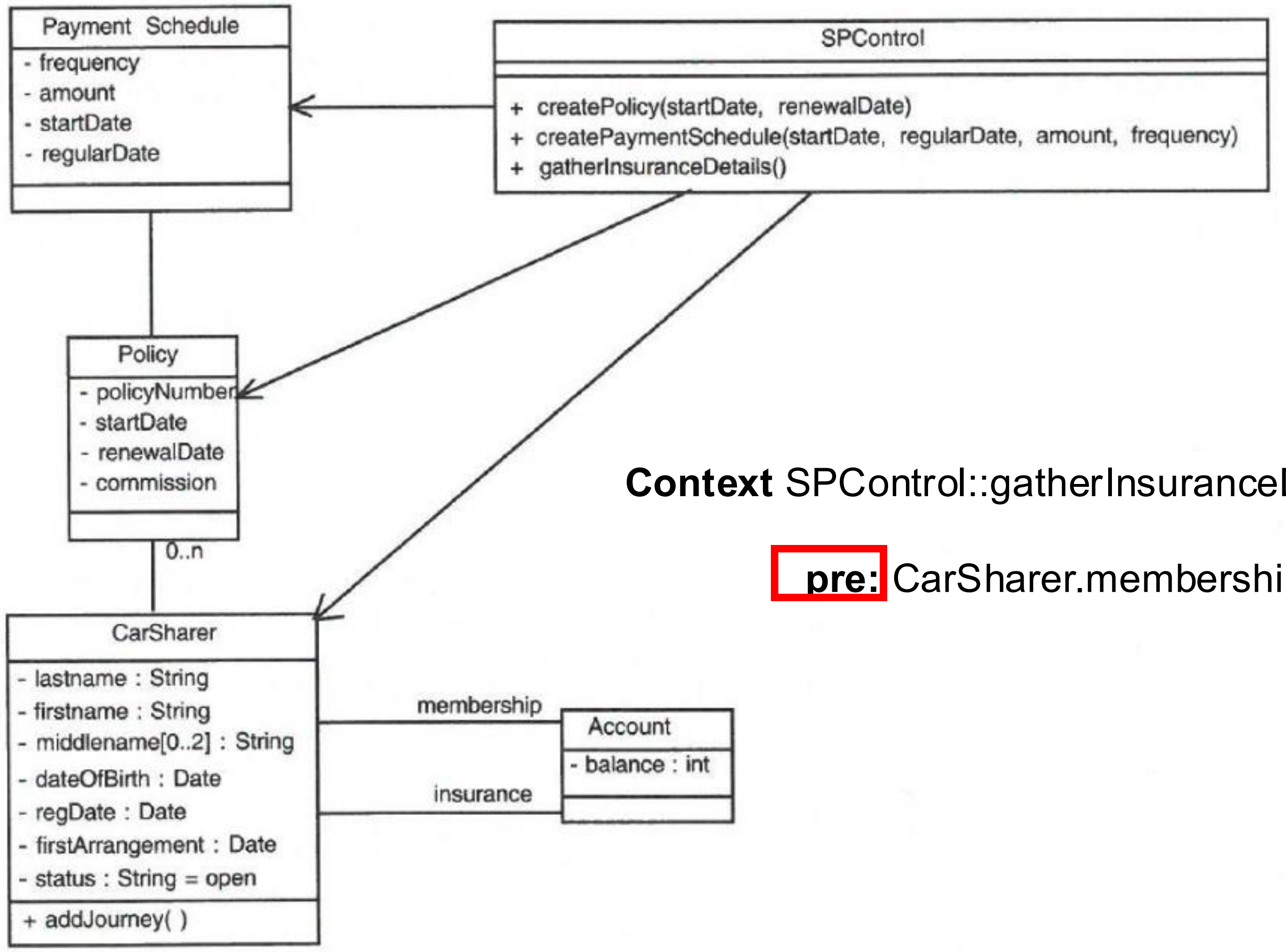
Exercise One: Insurance Policies

Membership must be fully paid before taking out insurance.



Exercise One: Insurance Policies

Membership must be fully paid before taking out insurance.

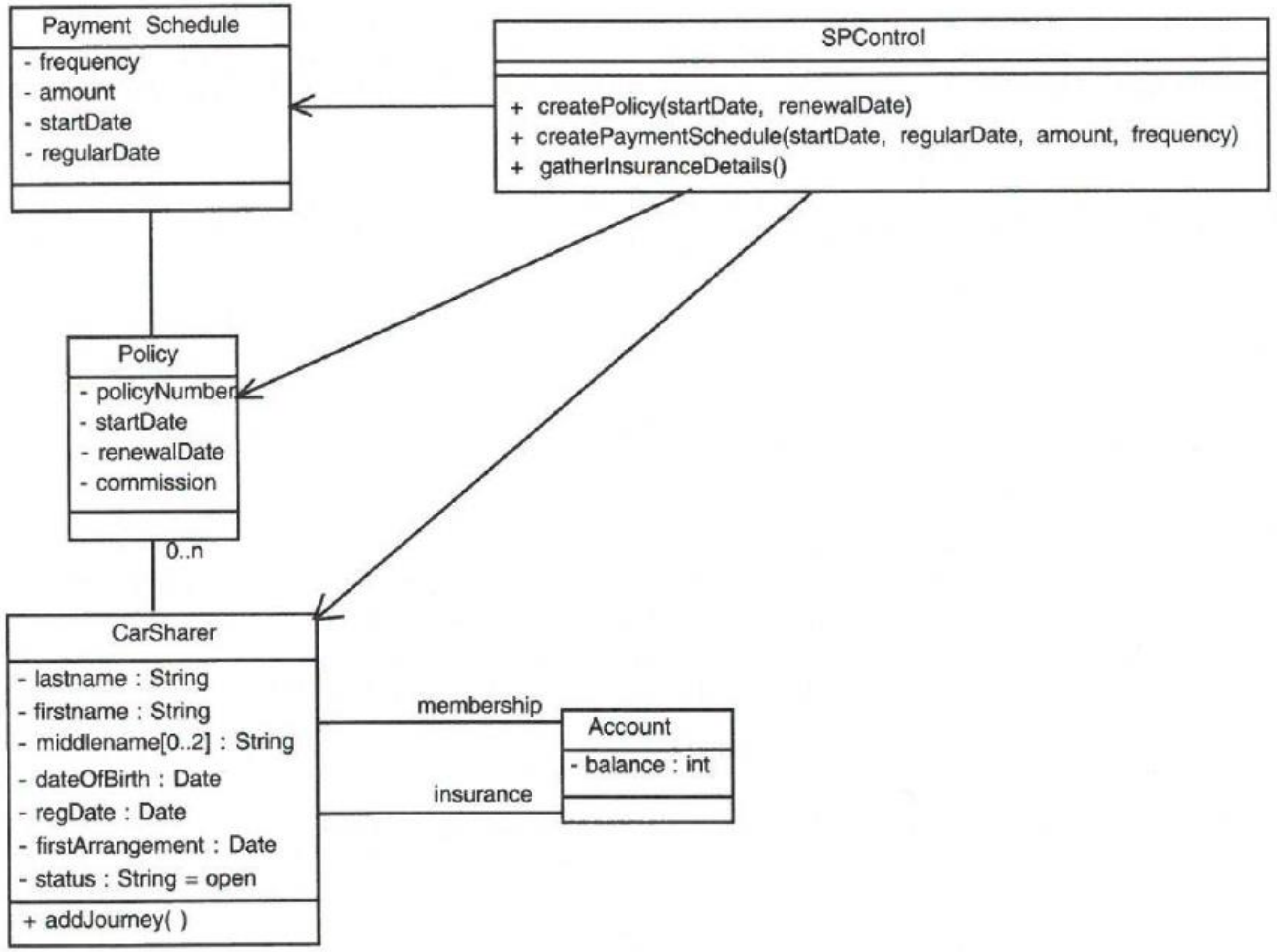


Context SPControl::gatherInsuranceDetails()

pre: CarSharer.membership.balance >=0

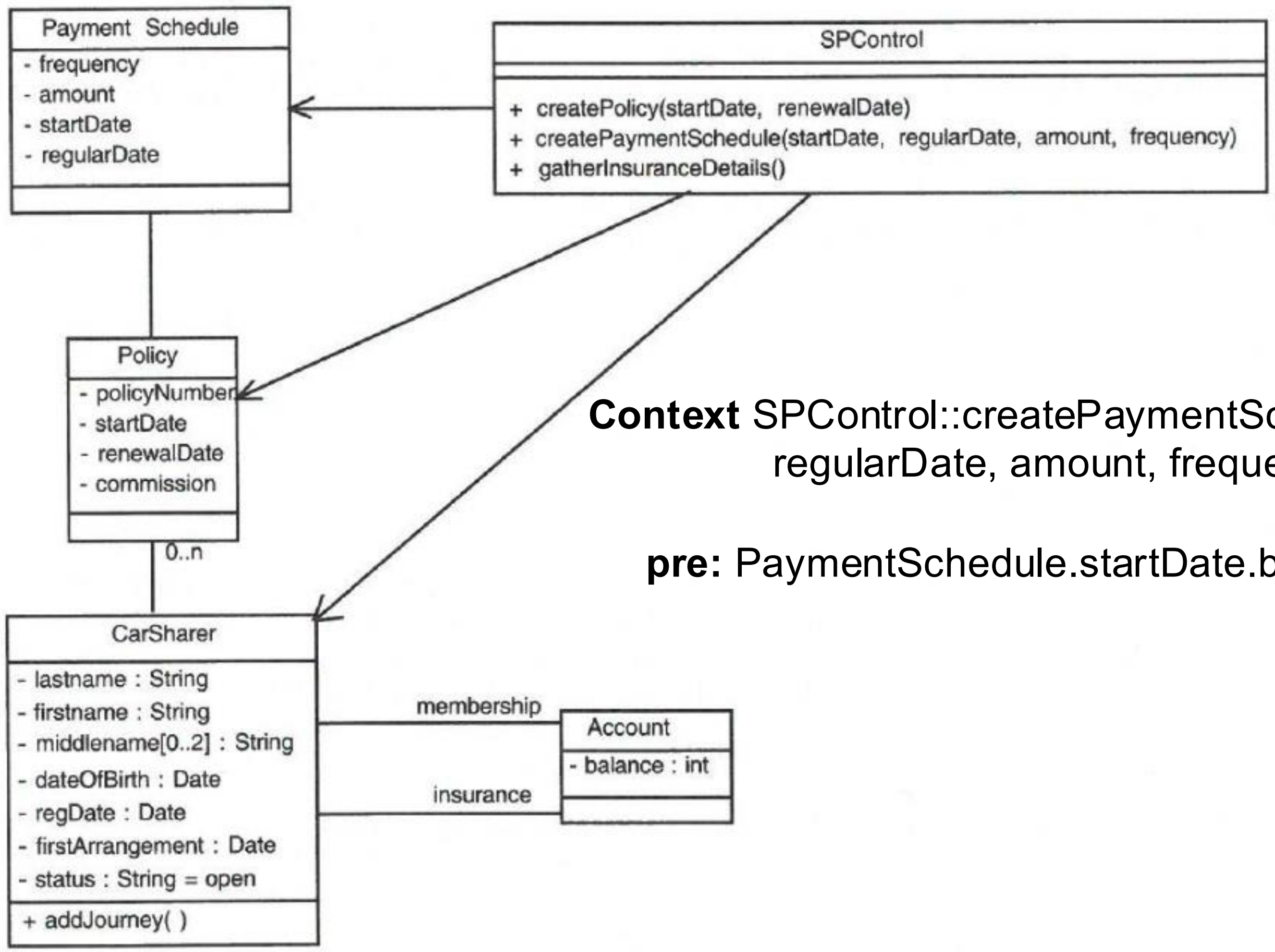
Exercise One: Insurance Policies

The first payment of the payment schedule must be made before cover is granted



Exercise One: Insurance Policies

The first payment of the payment schedule must be made before cover is granted

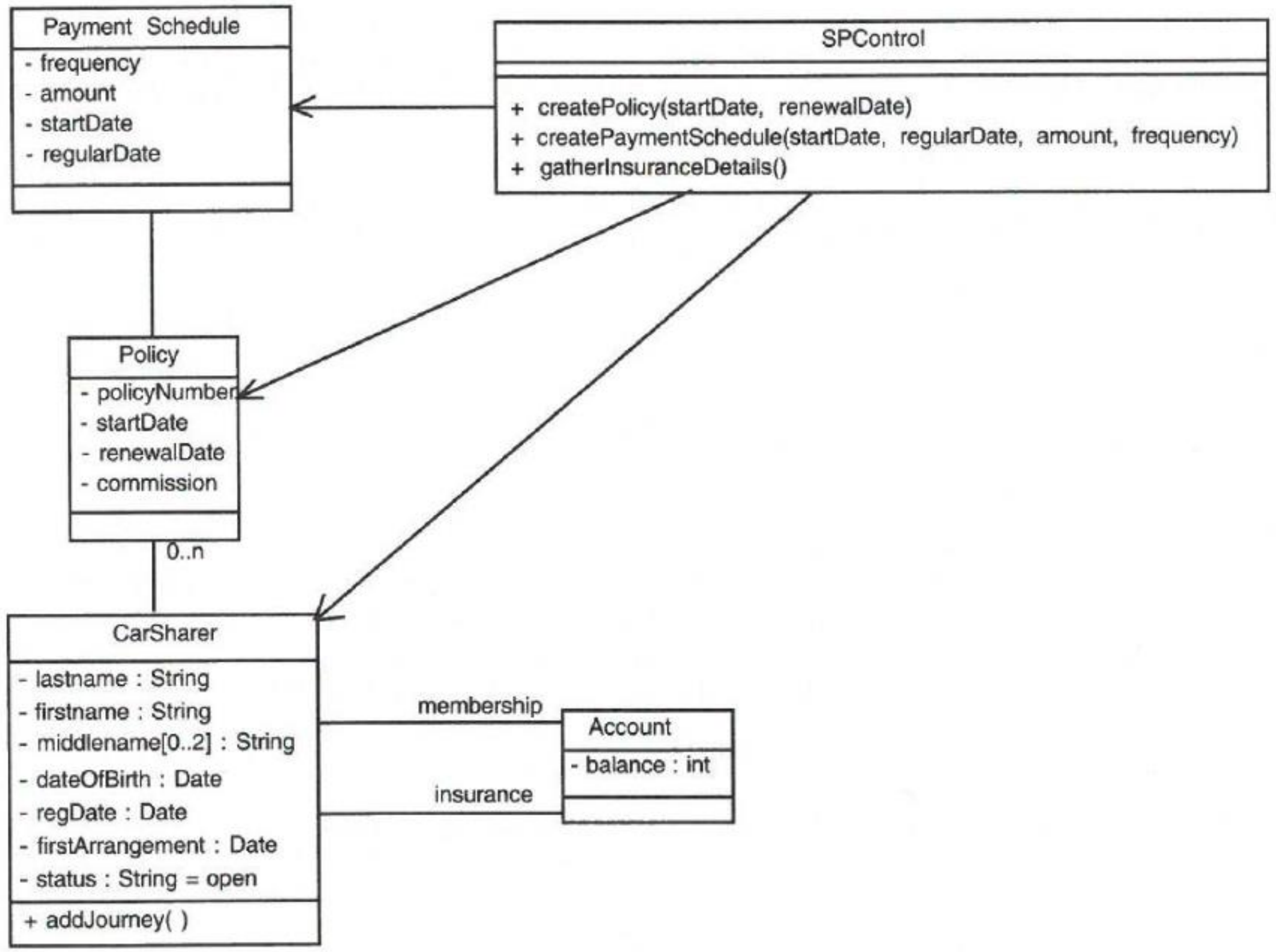


Context SPControl::createPaymentSchedule(startDate, regularDate, amount, frequency)

pre: PaymentSchedule.startDate.before(policy.startDate)

Exercise One: Insurance Policies

Two policies must not be in force at the same time for a given car sharer



Collections

- The situtation that association can refer to many objects.
  - **Sets**: consists of distinct instances and there are no repetitions in the set (NO ordering)
  - **Sequences**: an ordered collection that allows repetition
  - **Bags**: an unordered collection that allows repitition

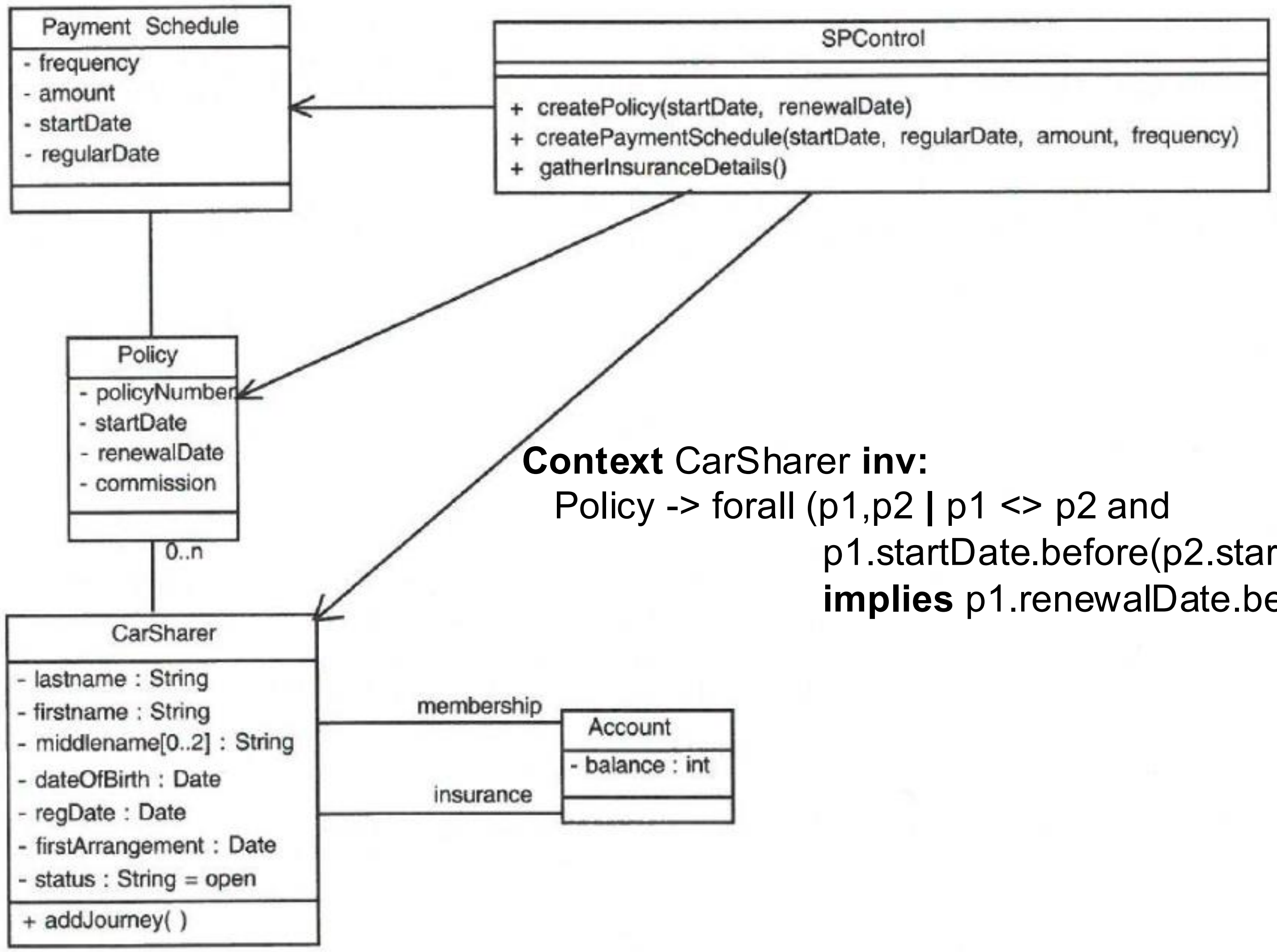
  - Example:
  - Select - picks out a subset of elements from a collection that meet a particular condition, the syntax: **collection -> select(v | boolean-expression-with-v)**

OCL provides many operations for accessing collections. The most often used are

- `size`, which returns the number of elements in the collection
- `includes(object)`, which returns `True` if `object` is in the collection
- `select(expression)`, which returns a collection that contains only the elements of the original collection for which expression is `True`
- `union(collection)`, which returns a collection containing elements from both the original collection and the collection specified as parameter
- `intersection(collection)`, which returns a collection that contains only the elements that are part of both the original collection and the collection specified as parameter
- `asSet(collection)`, which returns a set containing each element of the collection.

Exercise One: Insurance Policies

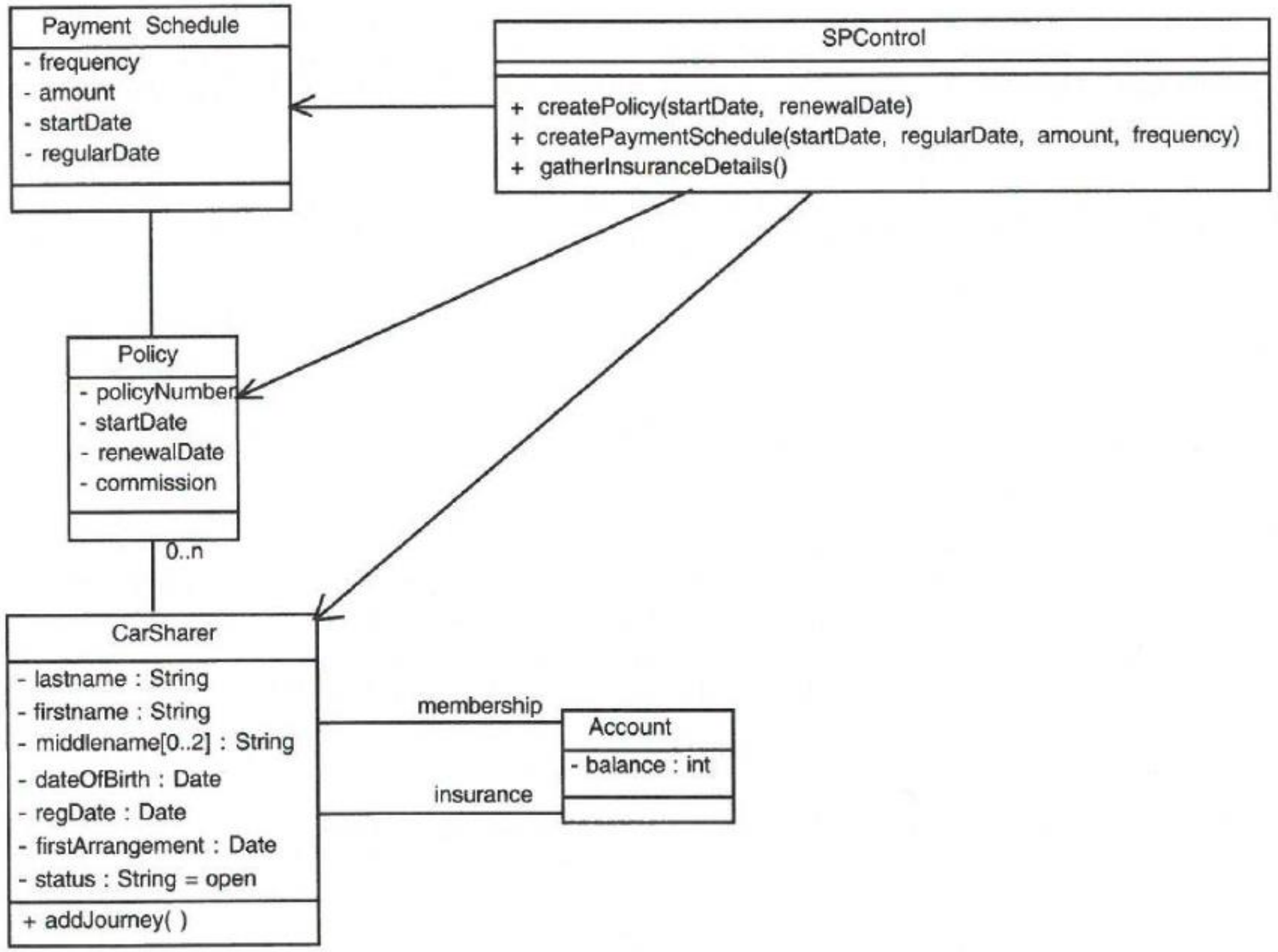
Two policies must not be in force at the same time for a given car sharer



Context CarSharer **inv:**
Policy -> forall (p1,p2 **|** p1 <> p2 and
p1.startDate.before(p2.startDate)
implies p1.renewalDate.before(p2.startDate))

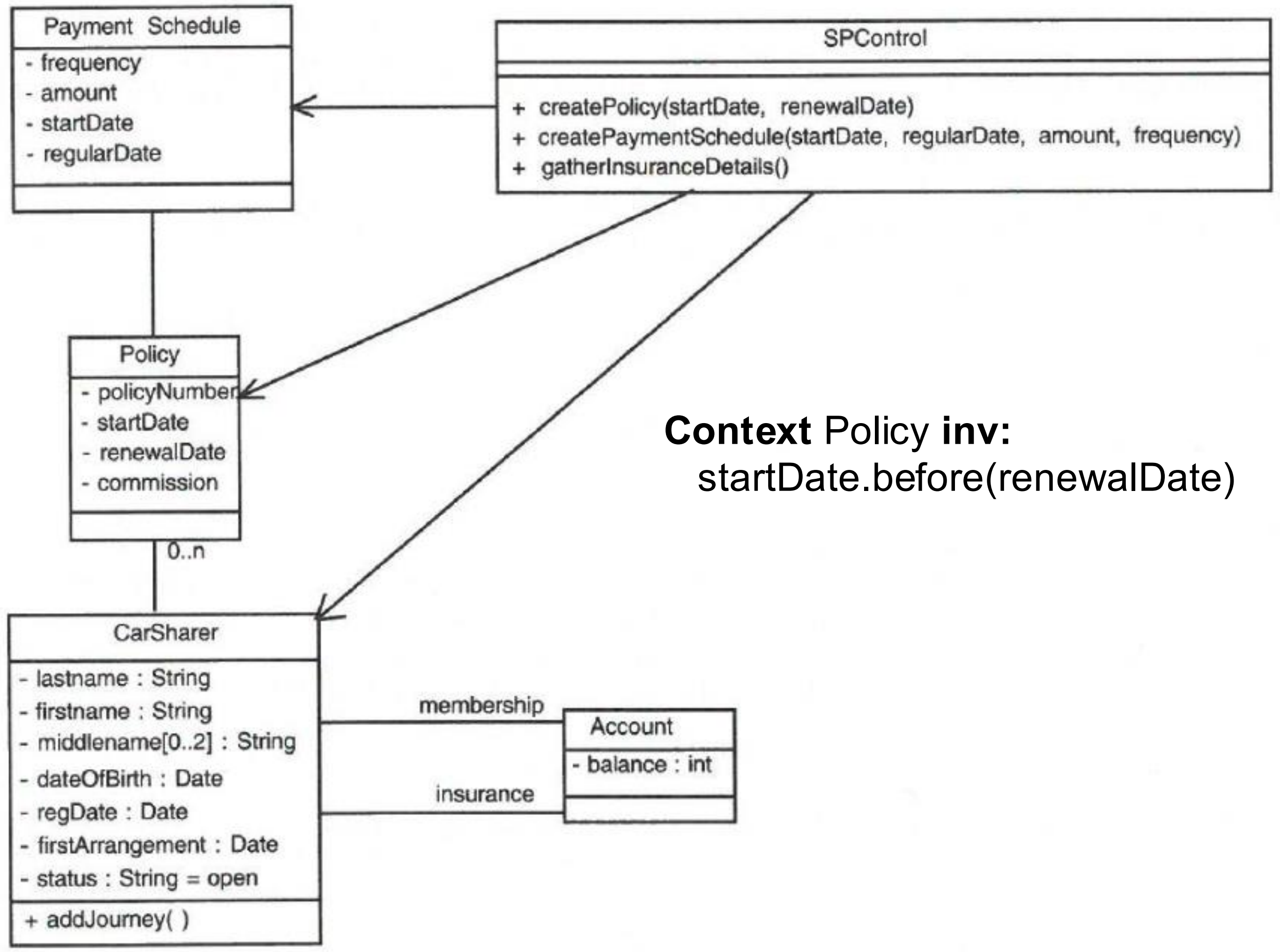
Exercise One: Insurance Policies

The renewal day of policy is always after its start day



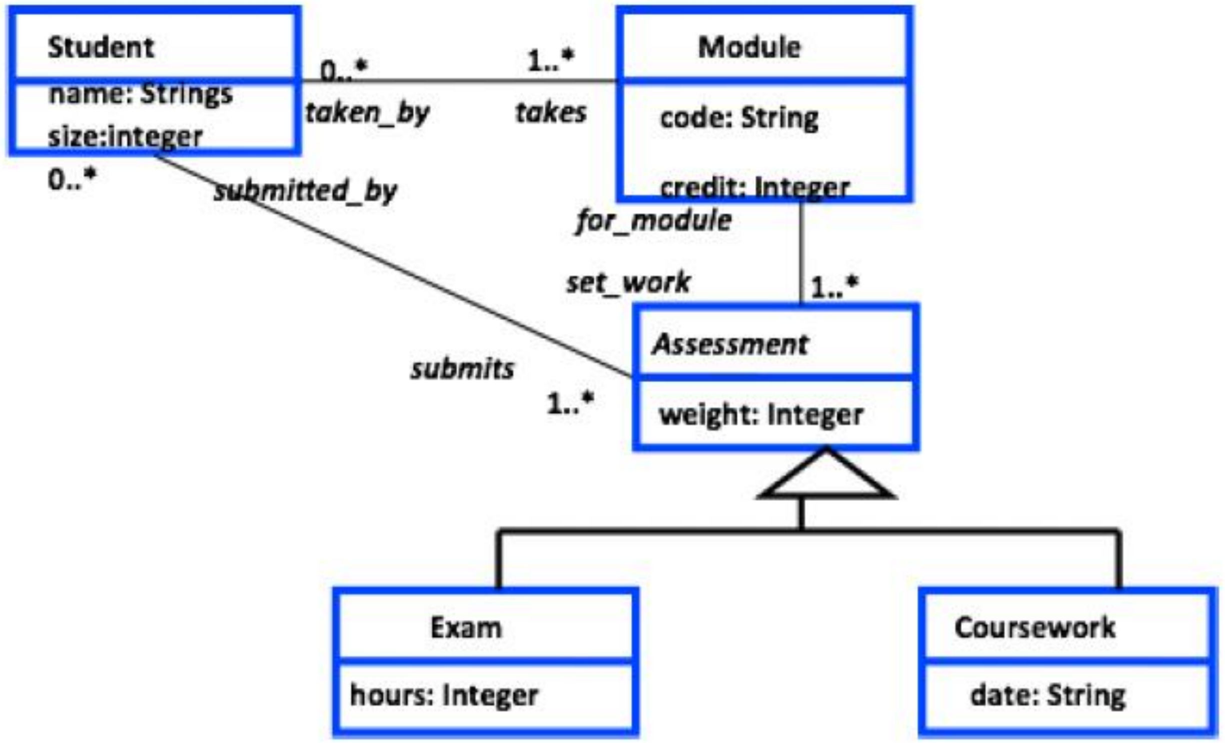
Exercise One: Insurance Policies

The renewal day of policy is always after its start day

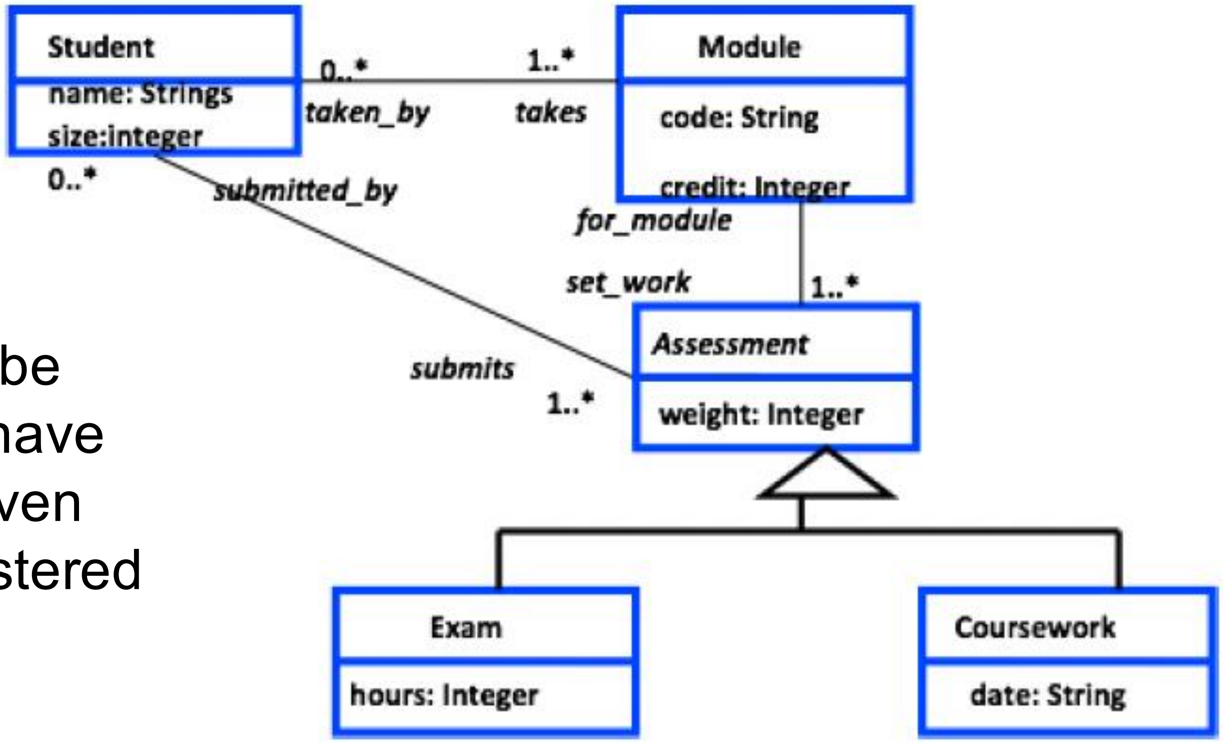


Context Policy **inv:**
startDate.before(renewalDate)

Exercise Two: Students and Exams

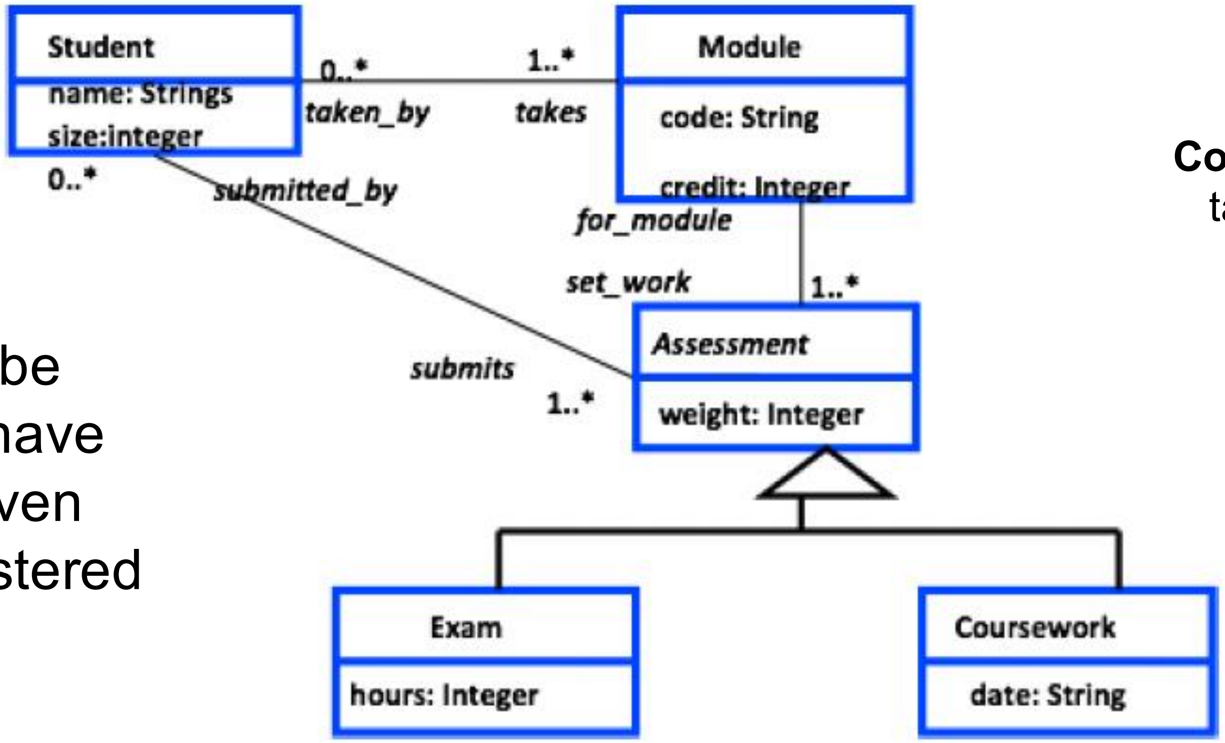


Exercise Two: Students and Exams



Modules can be taken if they have more than seven students registered

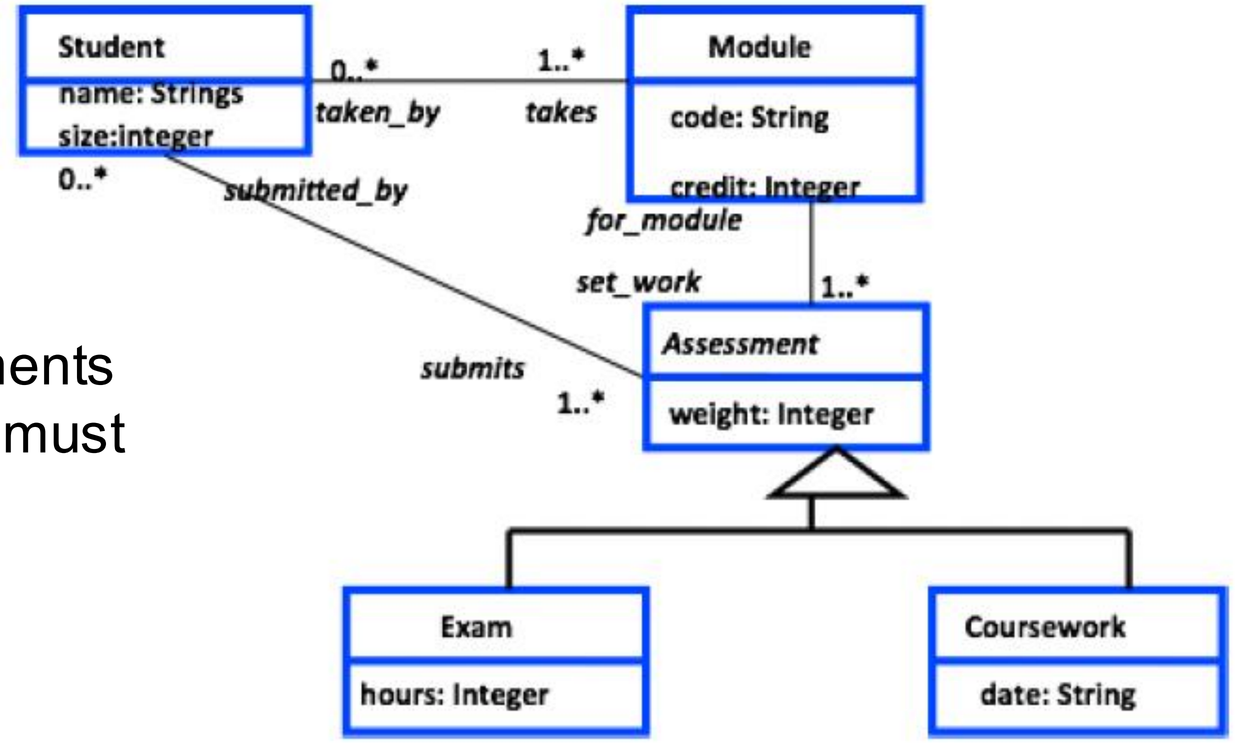
Exercise Two: Students and Exams



Context Module **inv:**
taken_by.size > 7

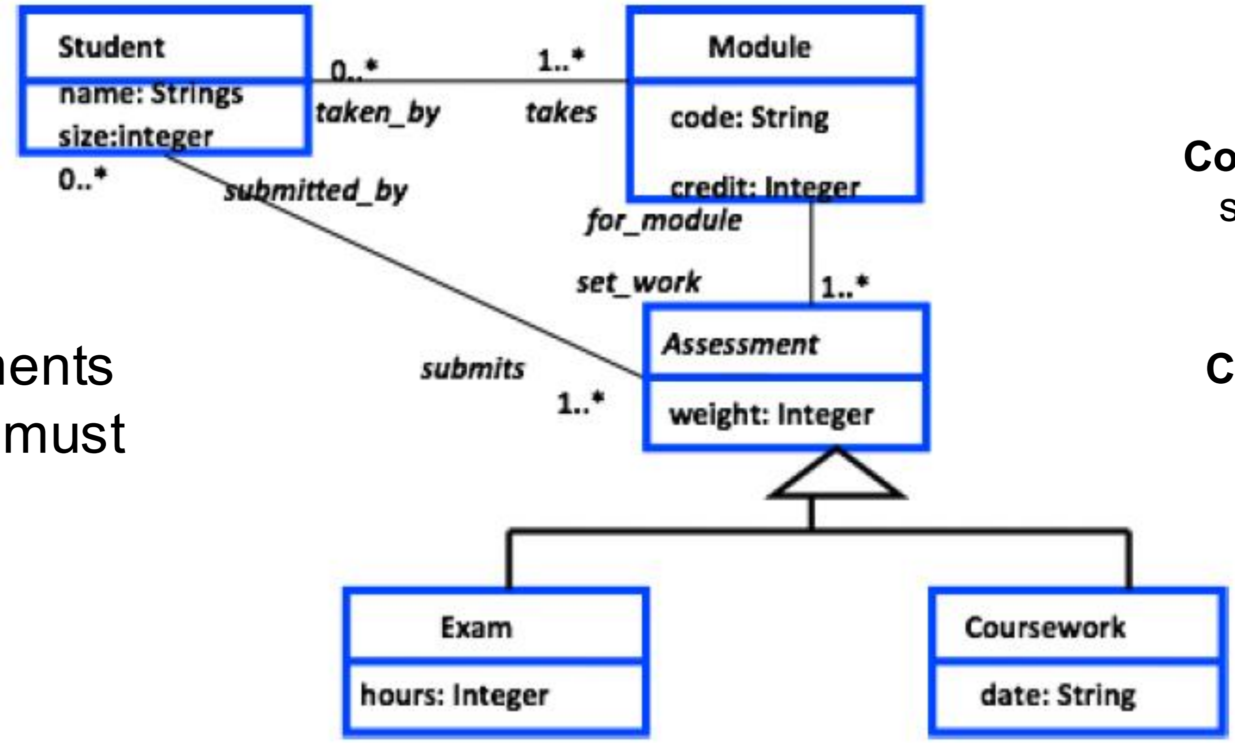
Modules can be taken if they have more than seven students registered

Exercise Two: Students and Exams



The assessments for a module must total 100%

Exercise Two: Students and Exams

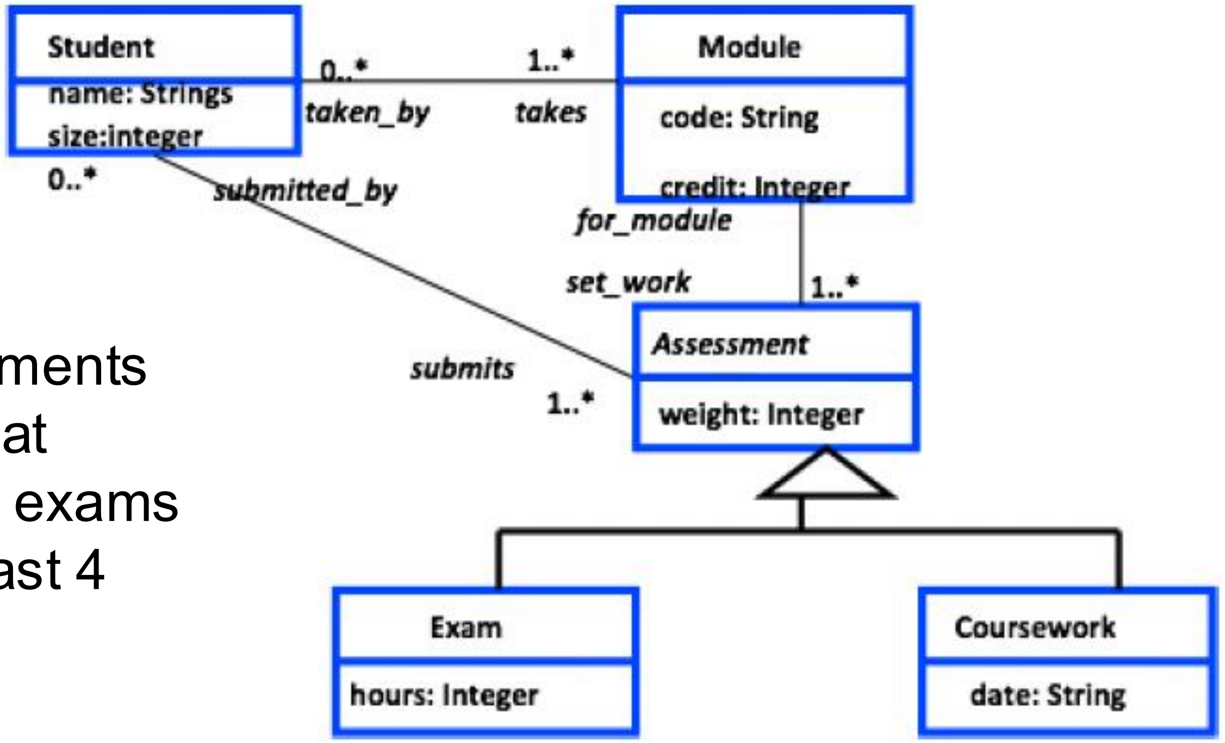


The assessments for a module must total 100%

Context Module **inv:**
set_work.weight -> sum() = 100

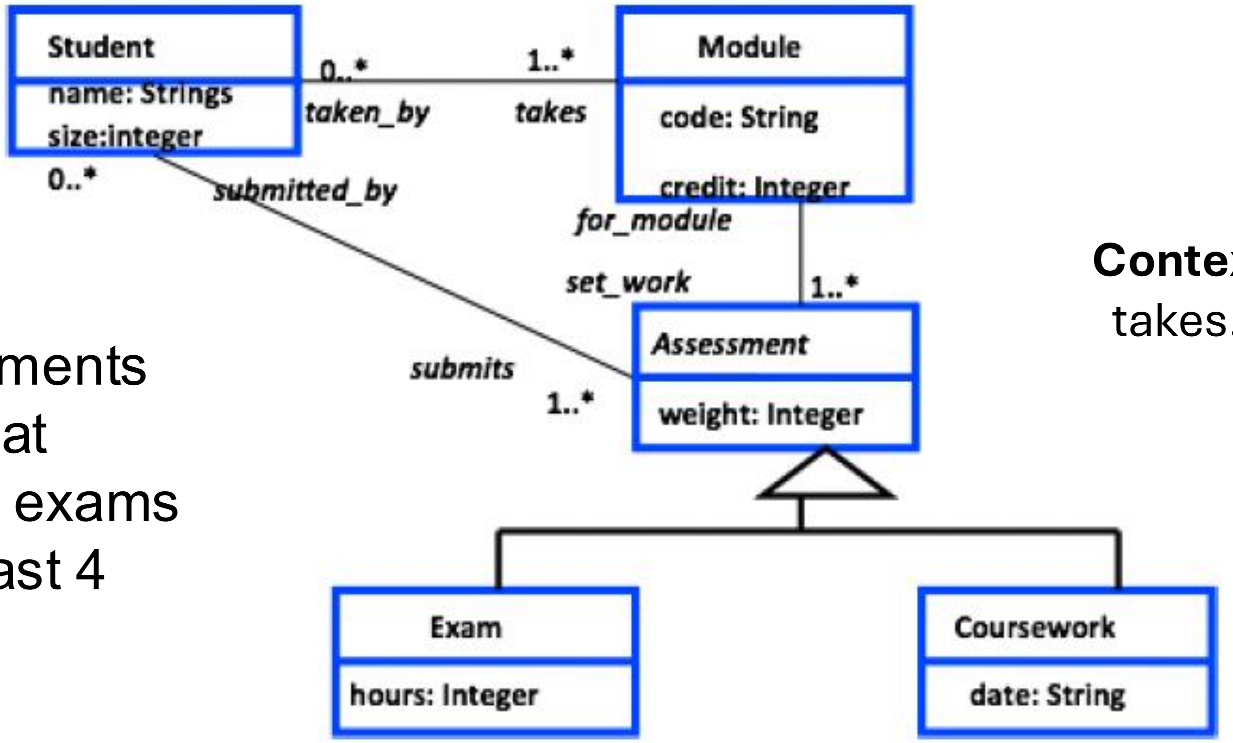
Context Module **inv:**
sum(set_work.weight) = 100

Exercise Two: Students and Exams



In the assessments of modules that students take exams must be at least 4 hours

Exercise Two: Students and Exams



In the assessments of modules that students take exams must be at least 4 hours

Context Students **inv:**
takes.set_work.hours -> forall(h| h>=4)