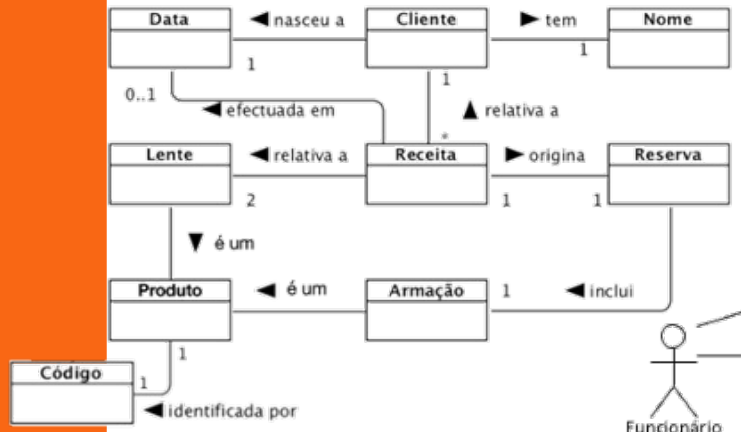
Desenvolvimento de Sistemas Software

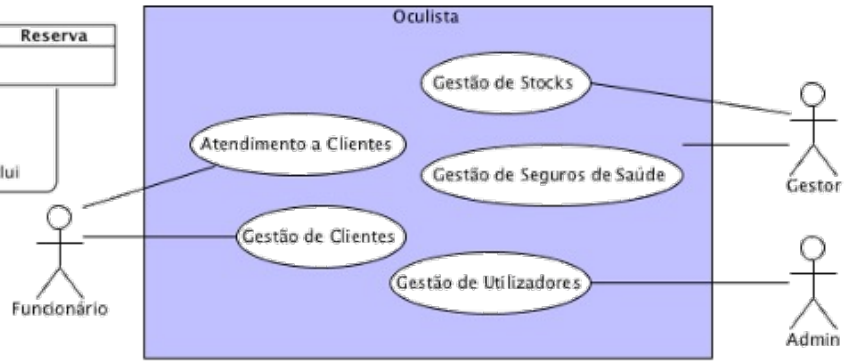
Object Constraint Language (OCL)

(actualizado: 17/11/2023)

Desenvolvimento de Sistemas Software
José Creissac Campos / António Nestor Ribeiro

Em resumo…



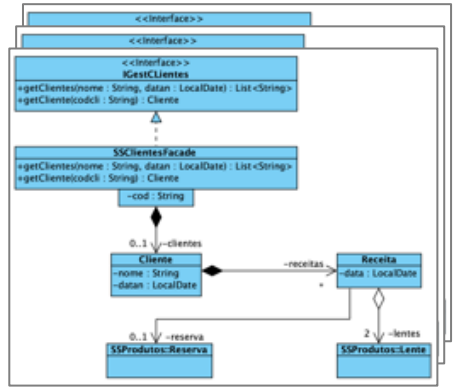


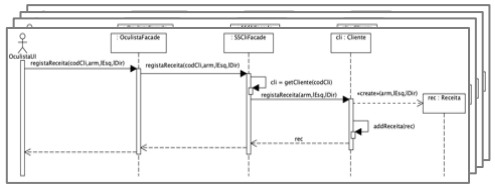
Use Case: Reservar armação e lentes

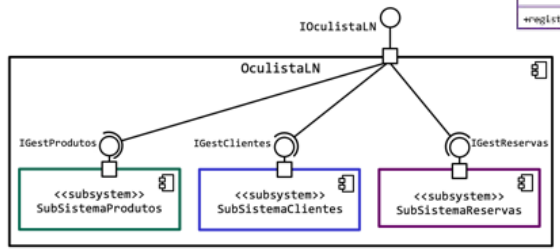
Descrição: Funcionário regista uma reserva de armação e lentes
Pós-condição: Reserva fica registada
Fluxo normal:
1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema procura clientes
3. Sistema apresenta lista de clientes
4. Funcionário selecciona cliente
5. Sistema procura cliente
6. Sistema apresenta detalhes do cliente
7. Funcionário confirma cliente
8. Sistema procura produtos e apresenta lista
9. Funcionário indica Código de armação e lentes
10. Sistema procura detalhes dos produtos
11. Sistema apresenta detalhes dos produtos
12. Funcionário confirma produtos
13. Sistema regista reserva dos produtos
14. <<include>> imprimir talão

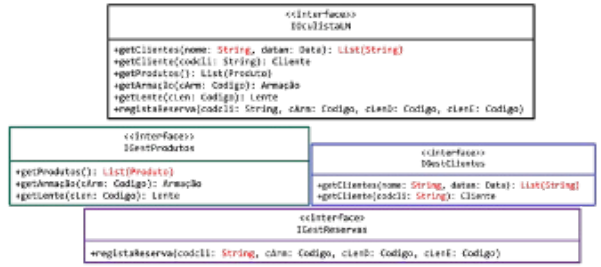
Fluxo alternativo: [lista de clientes tem tamanho 1] (passo 3)
3.1. Sistema apresenta detalhes do único cliente da lista
3.2. regressa a 7

Fluxo de excepção: [cliente não quer produto] (passo 12)
11.1. Funcionário rejeita produtos
11.2. Sistema termina processo

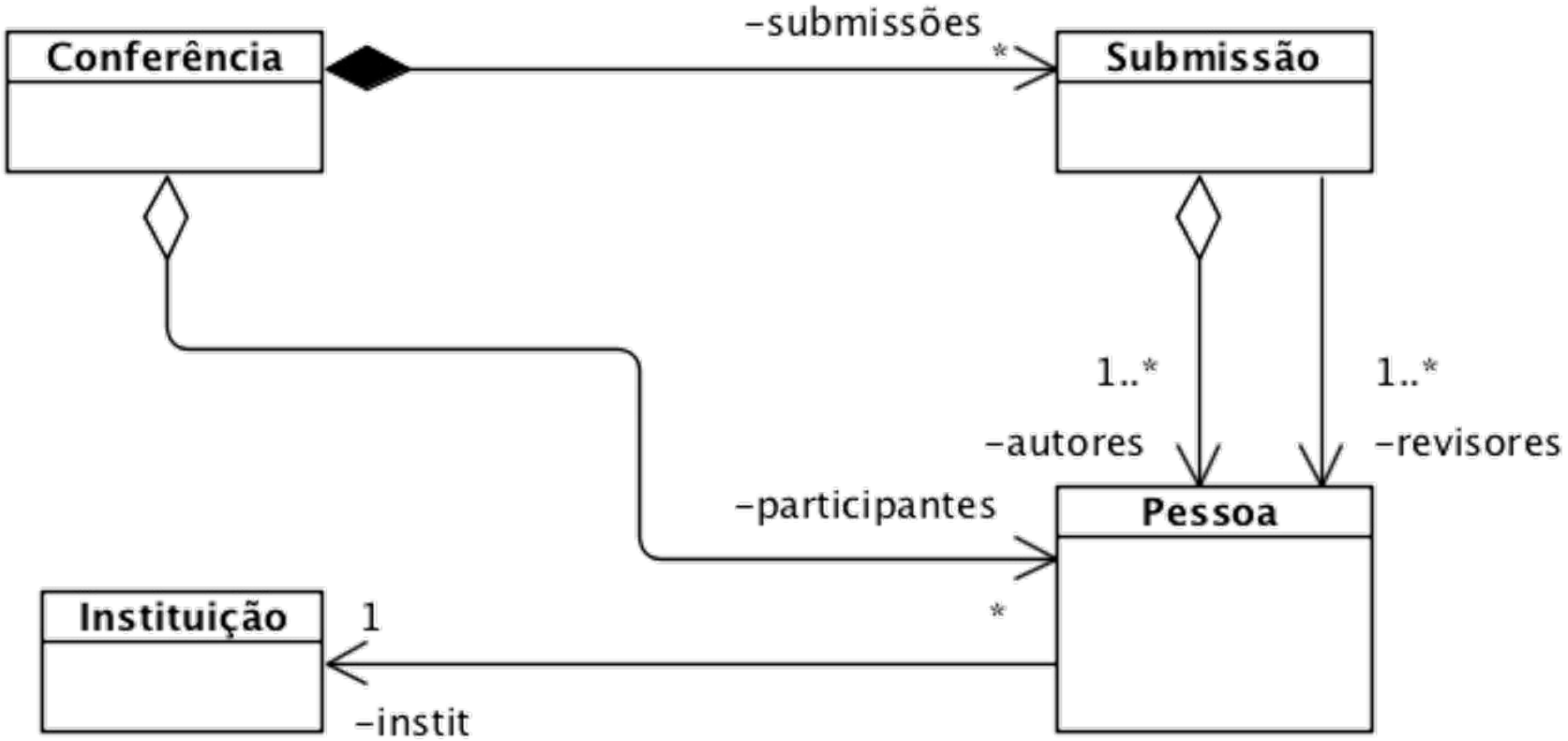






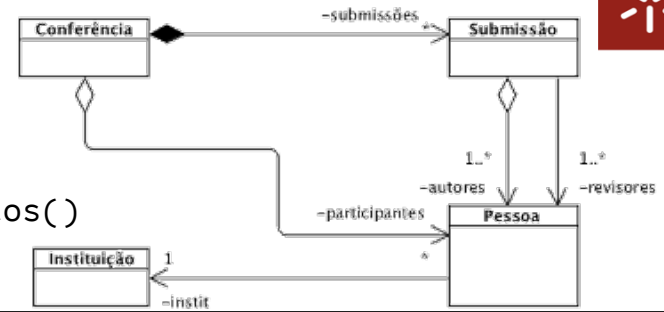


Algumas limitações…



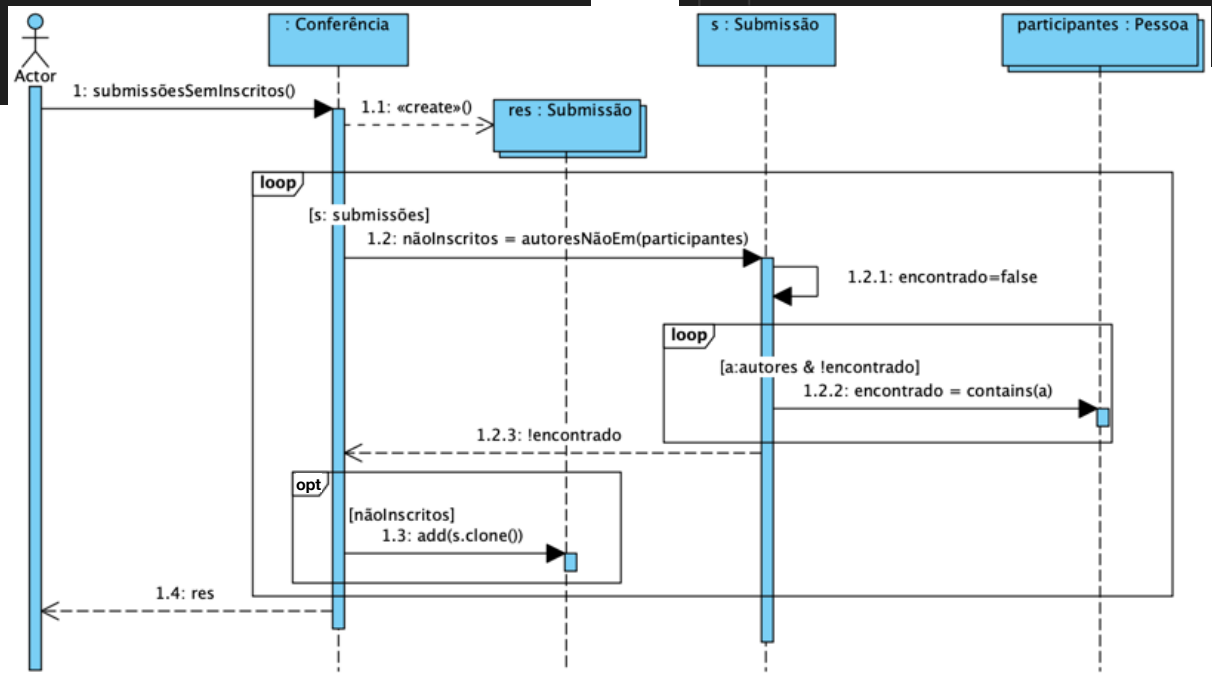
Algumas limitações…

- `public Collection<Submissão> submissõesSemInscritos()`



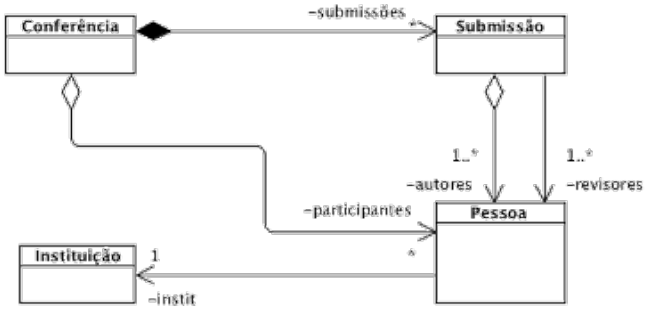

```java
public class Conferência {

    private Collection<Submissão> submissões;
    private Collection<Pessoa> participantes;

    public Collection<Submissão> submissõesSemInscritos() {
        Collection<Submissão> res = new ArrayList<>();

        for(Submissão s: submissões) {
            boolean nãoInscritos = s.autoresNãoEm(participantes);
            if (nãoInscritos) {
                res.add(s.clone());
            }
        }

        return res;
    }
```

```java
public class Submissão {

    private Collection<Pessoa> autores;
    private Collection<Pessoa> revisores;

    public boolean autoresNãoEm(Collection<Pessoa> participantes) {
        boolean encontrado = false;
        Iterator<Pessoa> itp = autores.iterator();

        while(!encontrado && itp.hasNext()) {
            encontrado = participantes.contains(itp.next());
        }
        return !encontrado;
```



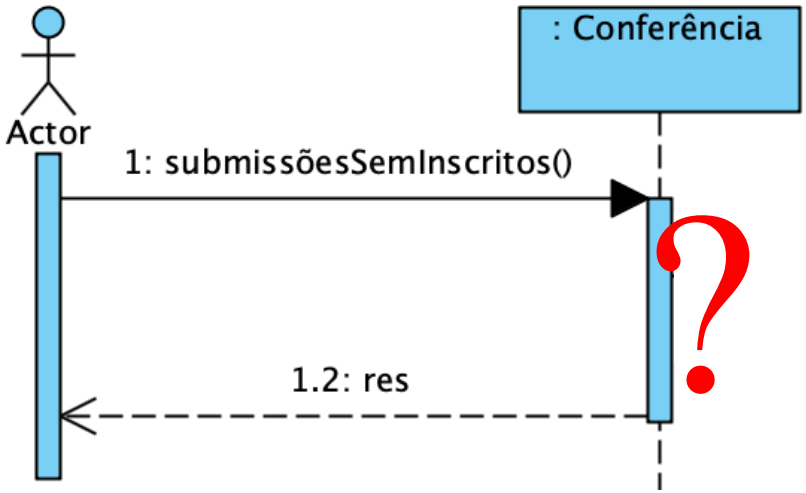
Algumas limitações…

- Paradigma funcional?



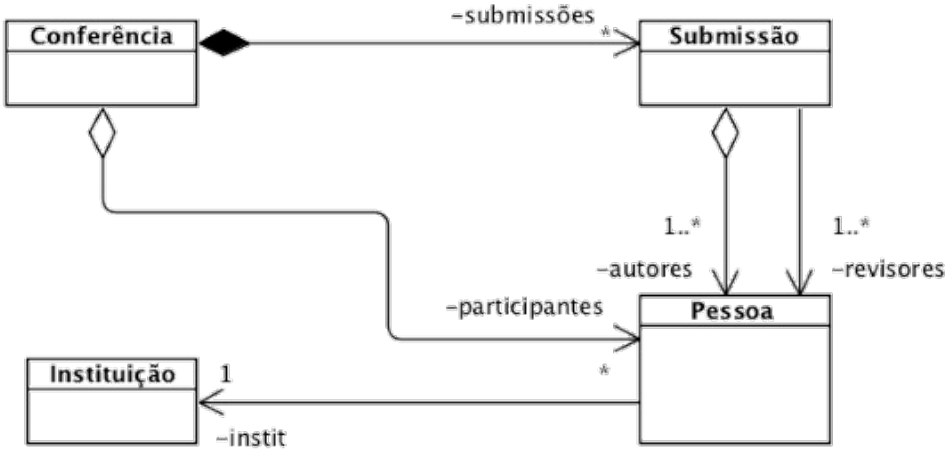

```java
public Collection<Submissão> submissõesSemInscritos() {

    return submissões.stream()
                     .filter(autoresNãoEm(participantes))
                     .map(Submissão::clone)
                     .collect(Collectors.toCollection(ArrayList::new));
}
```

Estilo funcional?



Diagramas UML nem sempre são suficientes

- Diagramas UML nem sempre são suficientes



- Quais destas afirmações são verdadeiras sobre a arquitectura?

a) É possível ter a mesma pessoa nas listas de autores e de revisores de uma submissão

b) Não é possível ter a mesma pessoa nas listas de autores e de revisores de uma submissão

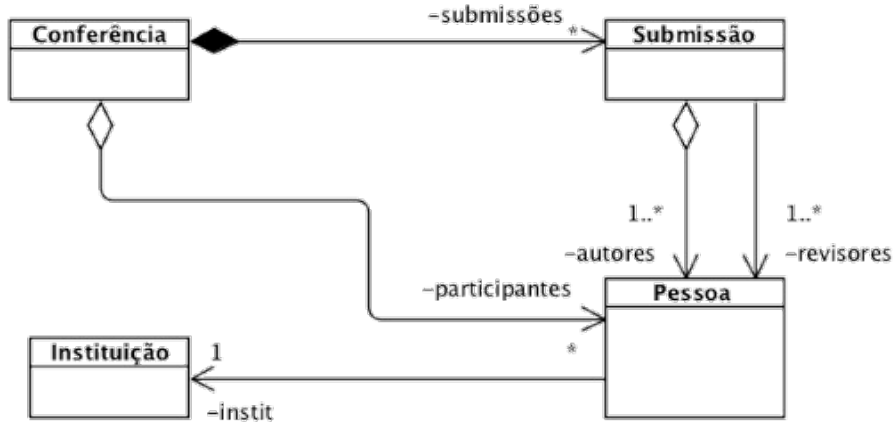
c) É possível ter a uma pessoa nas listas de autores e outra da mesma instituição na lista de revisores de uma submissão

d) Não é possível ter a uma pessoa nas listas de autores e outra da mesma instituição na lista de revisores de uma submissão

e) Nenhuma das anteriores

Diagramas UML nem sempre são suficientes

1. Os revisores de uma submissão **não** podem ser seus autores!
2. Os revisores de uma submissão **não** podem ser da mesma instituição dos autores!



- Como expressar estas restrições de forma não ambígua?

OCL: Object Constraint Language

- Linguagem declarativa
- Combina orientação a objectos com paradigma funcional

Breve História da OCL

- Em 1995 a divisão de seguros da IBM desenvolve uma linguagem para modelação de negócio
  - IBEL (Integrated Business Engineering Language)
- IBM propõe a IBEL ao OMG
  - OCL integrado na UML 1.1
- A OCL é utilizada para definir a UML 1.2

Onde utilizar OCL?

- Restrições em operações e associações:

- **Invariantes** de classe e tipos
  - Uma restrição que deve ser verdadeira num objecto durante todo o seu tempo de vida
- **Pré-condições** dos operações
  - Restrições que especificam as condições de aplicabilidade de uma operação
- **Pós-condições** dos operações
  - Restrições que especificam o resultado de uma operação

Sistema de tipos OCL

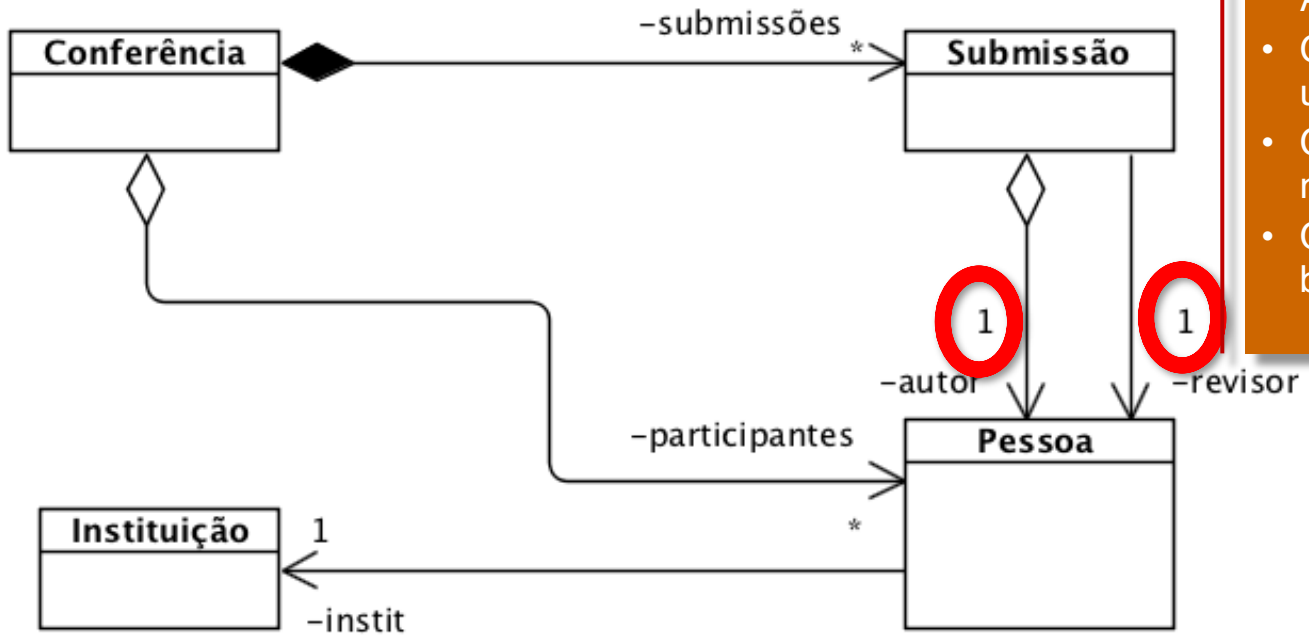
- Tipos primitivos

Type	Description	Values	Operations
Boolean		true, false	=, <>, and, or, xor, not, implies, if-then-else-endif
Integer	A whole number of any size	-1, 0, 1, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), / (real) abs(), max(b), min(b), mod(b), div(b)
Real	A real number of any size	1.5, ...	=, <>, >, <, >=, <=, *, +, - (unary), - (binary), / abs(), max(b), min(b), round(), floor()
String	A string of characters	'a', 'John'	=, <> size(), concat(s2), substring(from, to) toInteger(), toReal(),

Invariantes

1. Os revisores de uma submissão não podem ser seus autores

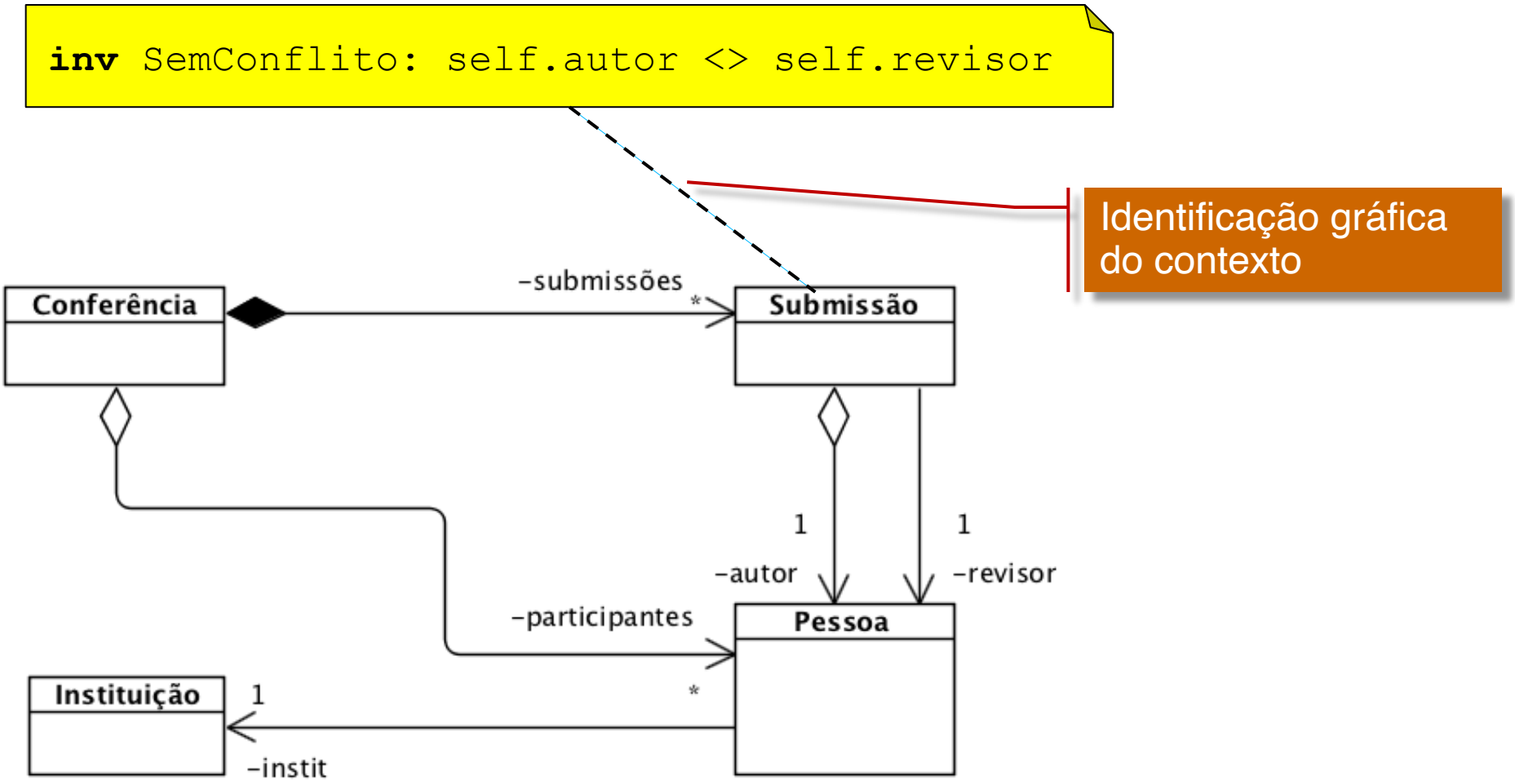
```
context Submissão
inv SemConflito: self.autor <> self.revisor
```



- Cada invariante tem um contexto (Classe, Interface ou Classe de Associação)
- O contexto pode ser referido utilizando *self* (opcional)
- Os invariantes podem ter um nome (neste caso é SemConflito)
- Os invariantes são expressões booleanas

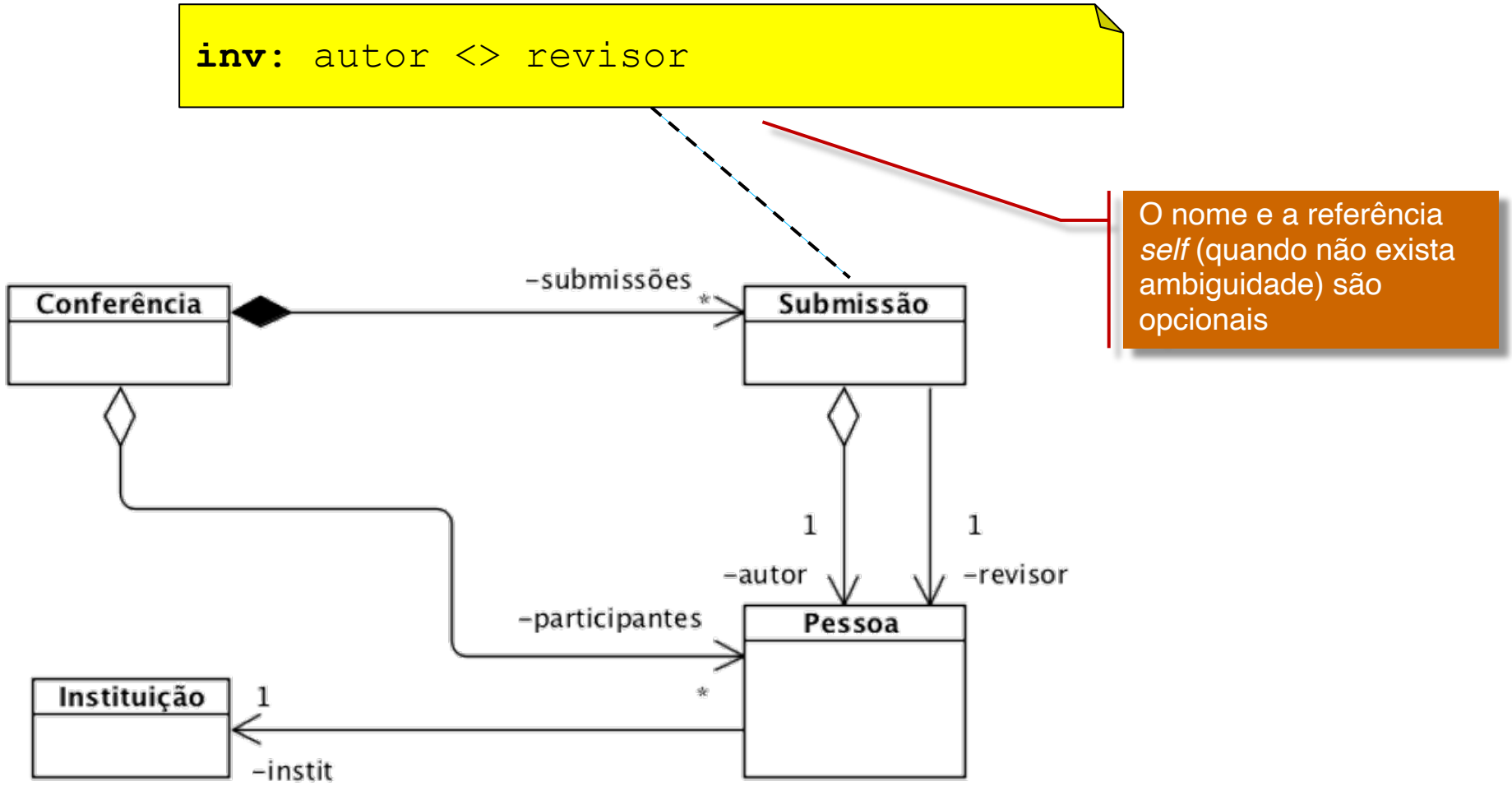
Invariantes

1. Os revisores de uma submissão não podem ser seus autores



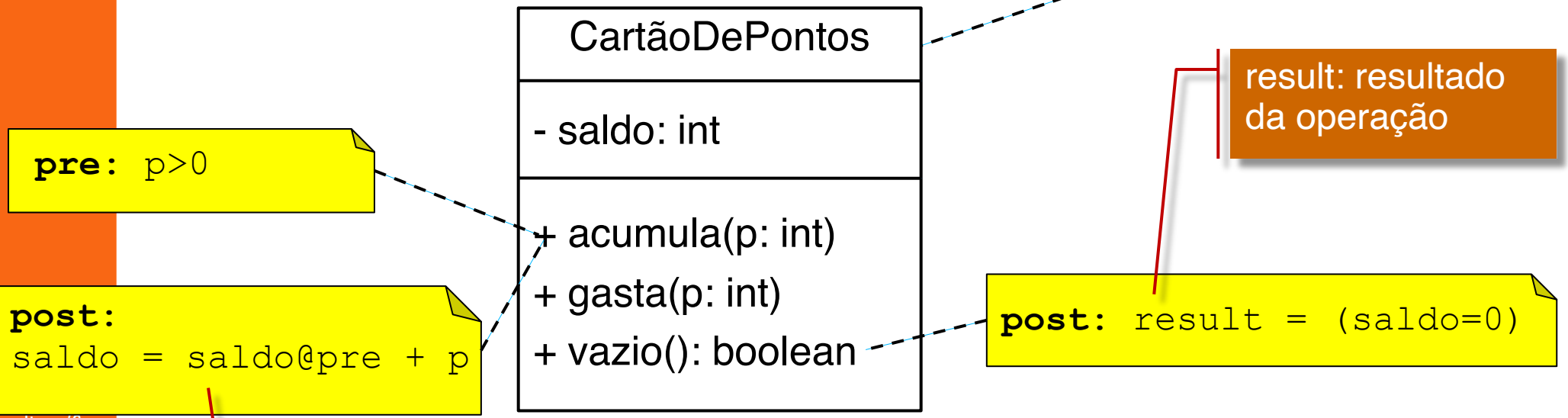
Invariantes

1. Os revisores de uma submissão não podem ser seus autores



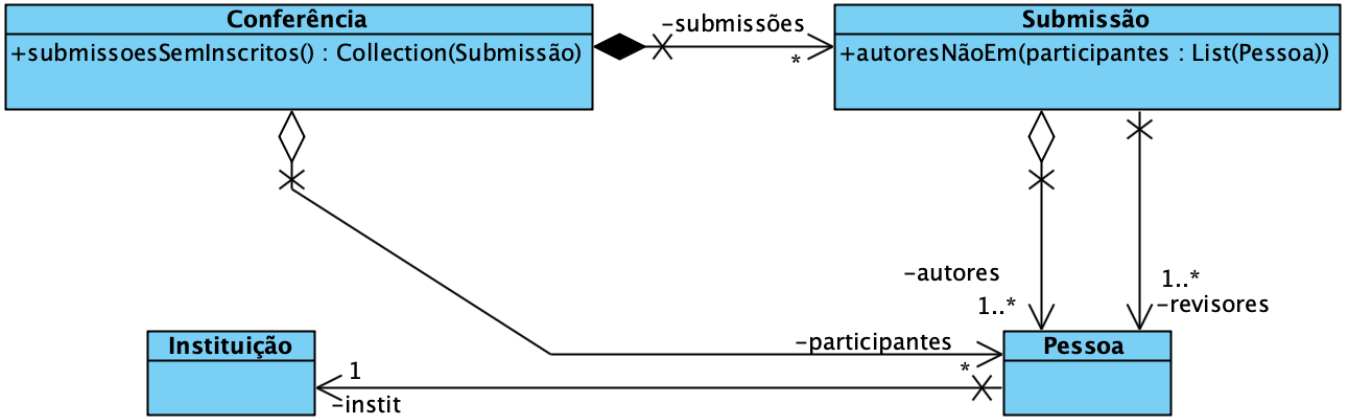
Pré- e pós-condições

Podem referir-se atributos nos invariantes

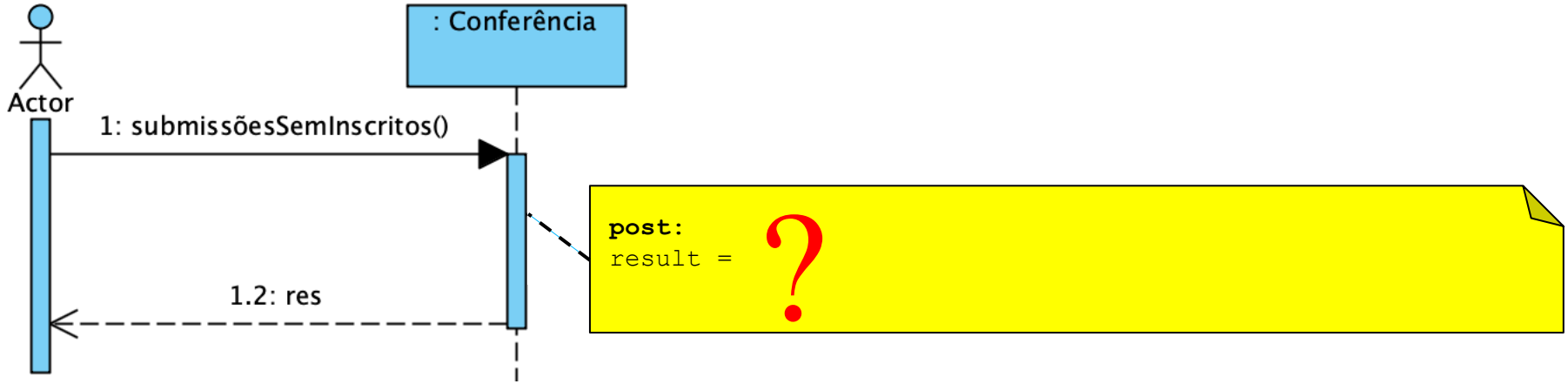



```
context CartãoDePontos::gasta(p:int)
pre: saldo>=p and p>=0

context CartãoDePontos::gasta(p:int)
post: saldo = saldo@pre - p
```

Pré- e pós-condições




```java
public Collection<Submissão> submissõesSemInscritos() {

    return submissões.stream()
                     .filter(autoresNãoEm(participantes))
                     .map(Submissão::clone)
                     .collect(Collectors.toCollection(ArrayList::new));
}
```



Sistema de tipos OCL

- Colecções e Tuplos

Description	Syntax	Examples
Abstract collection of elements of type T	**Collection(T)**	
Unordered collection, no duplicates	**Set(T)**	Set{1 , 2}
Ordered collection, duplicates allowed	**Sequence(T)**	Sequence {1, 2, 1} Sequence {1..4} (same as {1,2,3,4})
Ordered collection, no duplicates	**OrderedSet(T)**	OrderedSet {2, 1}
Unordered collection, duplicates allowed	**Bag(T)**	Bag {1, 1, 2}
Tuple (with named parts)	**Tuple(field1: T1, fieldn : Tn)**	Tuple {age: Integer = 5, name: String = 'Joe' } Tuple {name = 'Joe', age = 5}

Colecções - Operações

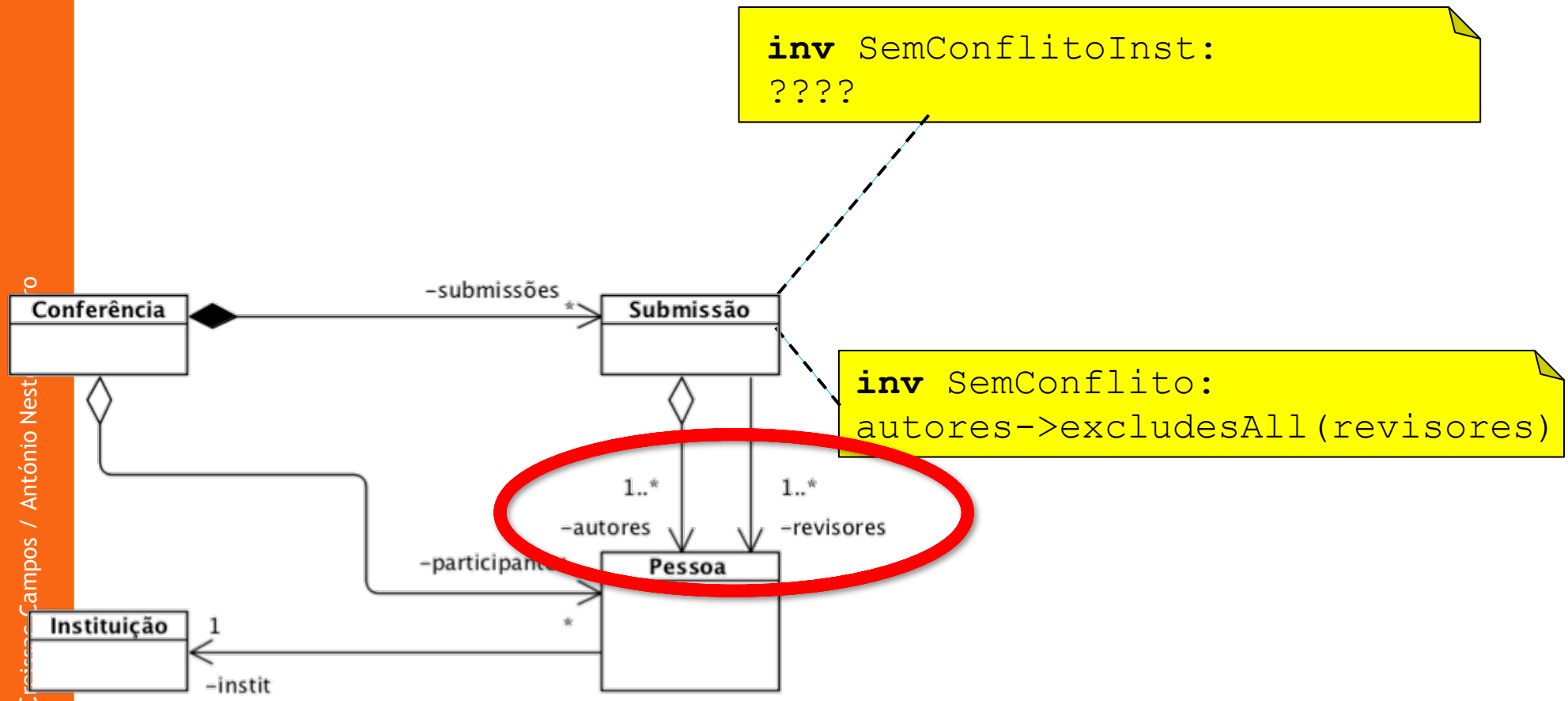
Operation	Description
size(): Integer	The number of elements in this collection (*self*)
isEmpty(): Boolean	size = 0
notEmpty(): Boolean	size > 0
includes(object: T): Boolean	True if *object* is an element of *self*
excludes(object: T): Boolean	True if *object* is not an element of *self*
count(object: T): Integer	The number of occurrences of *object* in *self*
includesAll(c2: Collection(T)): Boolean	True if *self* contains all the elements of *c2*
excludesAll(c2: Collection(T)): Boolean	True if *self* contains none of the elements of *c2*
sum(): T	The addition of all elements in *self* (T must support "+")
at(pos: Integer): T	The element at position *pos* – applicable to Sequence(T) and OrderedSet(T)

Colecções - Operações

- Set, OrderedSet, Bag e Sequence são casos particulares de Colecções (herdam as operações das colecções)
- Operações próprias
  - Set: =, union, intersection, -(difference), …
  - OrderedSet: =, union, intersection, …
  - Bag: =, union, intersection, flatten, …
  - Sequence: =, append, prepend, insertAt, subSequence, …
- As operações em colecção aplicam-se com '->' em vez de '.'
  - s1->intersection(s2)

Colecções - exemplos

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição dos autores



Colecções - iteradores (tipo *map*)

coleçãoBase -> expressão(*iterador* |*corpo*)

Expressão	Descrição
select(iterator | body): Collection(T)	A coleção de elementos da coleção base para os quais *body* é verdadeiro.
reject(iterator | body): Collection(T)	A coleção de elementos da coleção base para os quais *body* é falso.
collect(iterator | body): Collection(T2)	A coleção de elementos resultantes de aplicar *body* a cada elemento da coleção base. O resultado é *flattened*.
collectNested(iterator | body): Bag(T2) *ou* Sequence(T2)	A coleção de elementos resultantes de aplicar *body* a cada elemento da coleção base. O resultado não é *flattened* (Set passa a Bag; OrdereSet a Sequence).
sortedBy(iterator | body): Sequence(T) *ou* OrderedSet(T)	A coleção ordenada de todos os elementos da coleção base, ordenados por ordem crescente do valor resultante de aplicar *body* a cada elemento.

Exemplo: `self.autores->collect(a | a.instit)`

Colecções - iteradores (tipo *reduce*)

Expressão	Descrição
iterate(iterator: T; acc: T2 = init | body): Collection(T2)	Devolve o valor final do acumulador (*acc*) que, após inicialização, é atualizado com o valor de *body* aplicado a cada elemento da coleção base e a *acc*.
exists(iterators | body): Boolean	Verdade se *body* avalia como verdade para, pelo menos, um elemento da coleção base. Permite ter múltiplos iteradores.
forAll(iterators | body): Boolean	Verdade se *body* avalia como verdade para todos os elementos da coleção base. Permite ter múltiplos iteradores.
one(iterator | body): Boolean	Verdade se *body* avalia como verdade para exatamente um elemento da coleção base.
isUnique(iterator | body): Boolean	Verdade se *body* avalia para um valor diferente para cada um dos elementos da coleção base.
any(iterator | body): T	Devolve um valor da coleção base para o qual *body* avalia como verdade (null se não existir um).

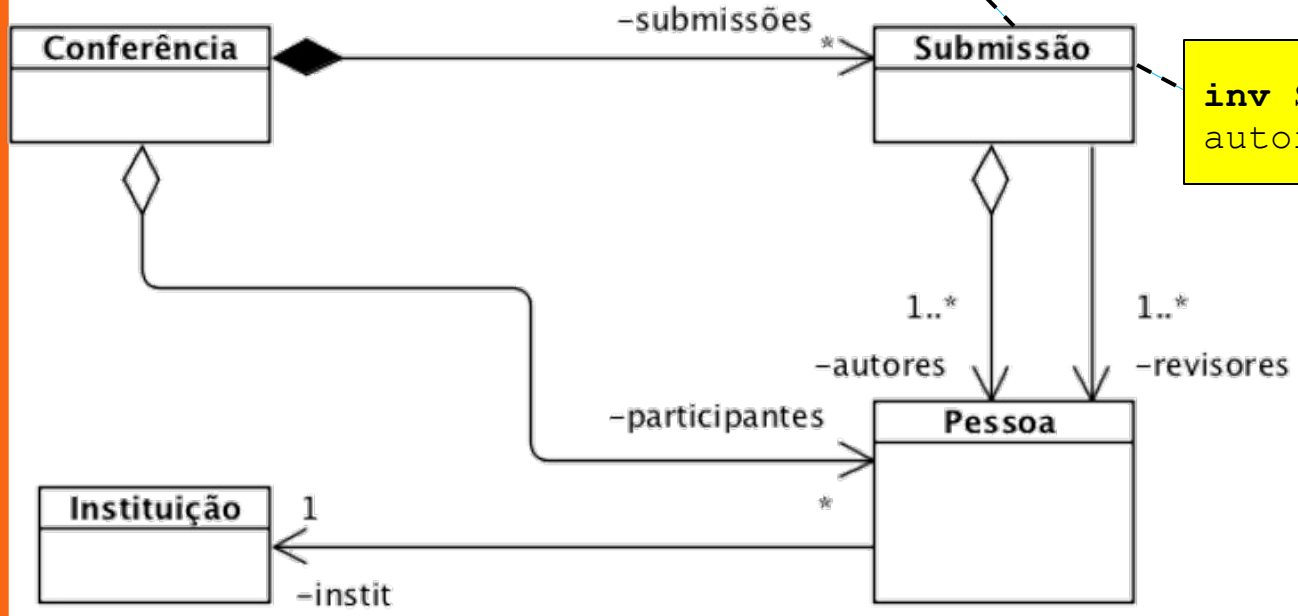
Nota: Os iteradores podem ser omitidos se não existir ambiguidade.

Colecções - exemplos

Se o iterador não é ambíguo, pode ser omitido (para autores apresenta-se a notação completa)

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição dos autores

```
inv SemConflitoInst:
(autores->collect(a | a.instit))->excludesAll(revisores->collect(instit))
```



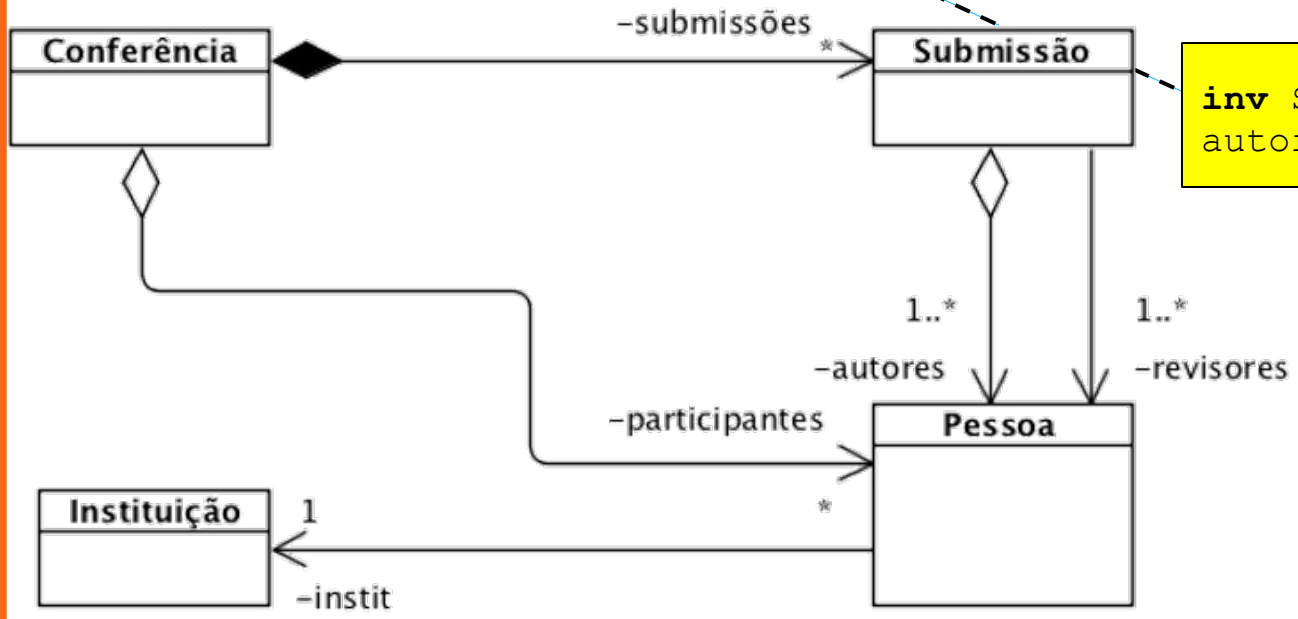

```
inv SemConflito:
autores->excludesAll(revisores)
```

Colecções - exemplos

1. Os revisores de uma submissão não podem ser seus autores
2. Os revisores de uma submissão não podem ser da mesma instituição dos autores

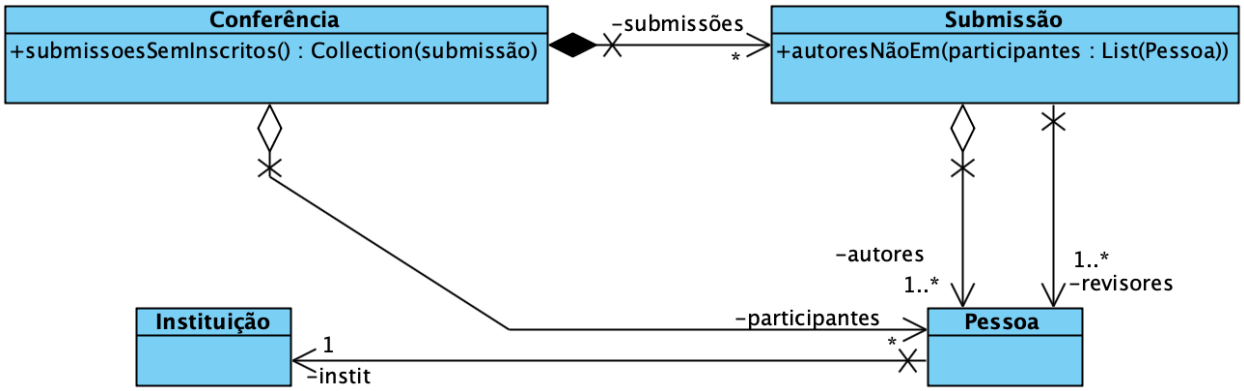
```
inv SemConflitoInst:
autores.instit->excludesAll(revisores.instit)
```

Como autores e revisores são colecções, o collect é aplicado automaticamente.

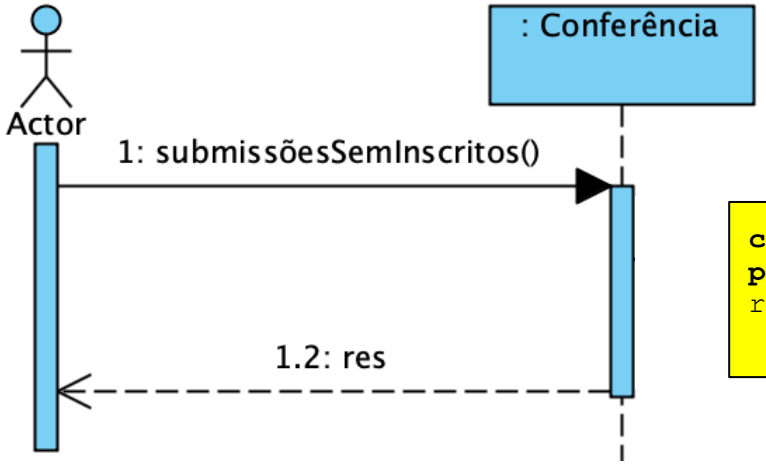



```
inv SemConflito:
autores->excludesAll(revisores)
```

Pré- e pós-condições

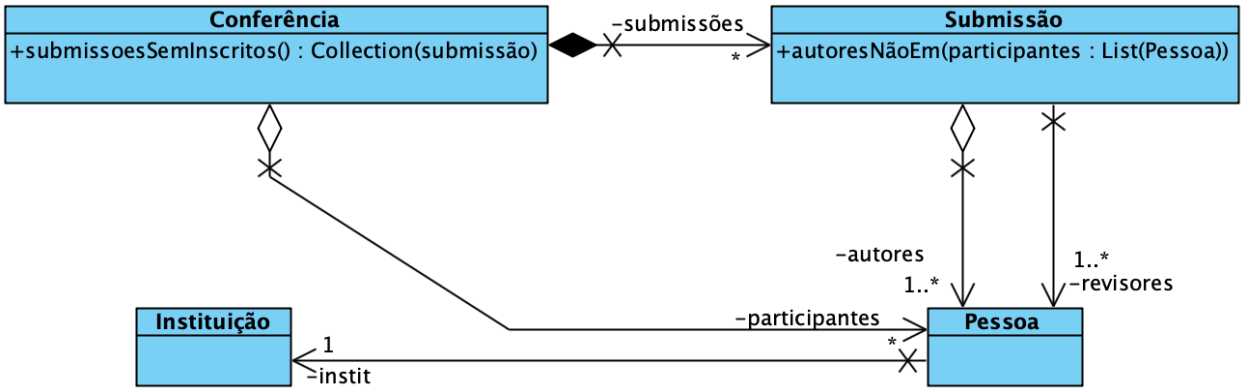



```java
public Collection<Submissão> submissõesSemInscritos() {

    return submissões.stream()
                     .filter(autoresNãoEm(participantes))
                     .map(Submissão::clone)
                     .collect(Collectors.toCollection(ArrayList::new));
}
```

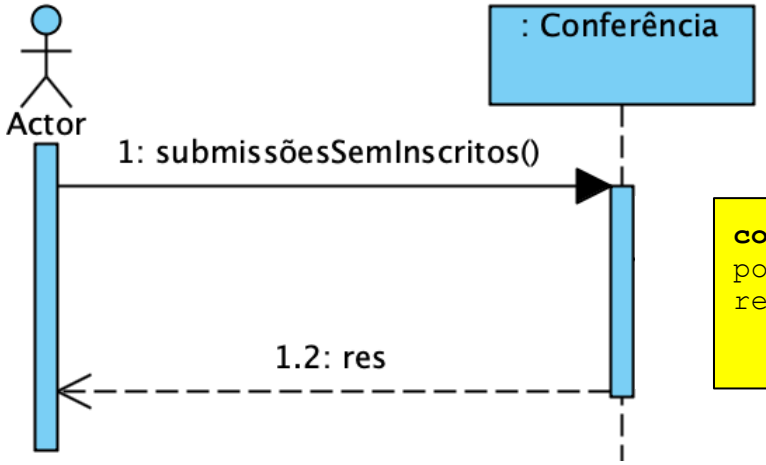


context Conferência::submissõesSe,Inscritos()
post:
result = ?

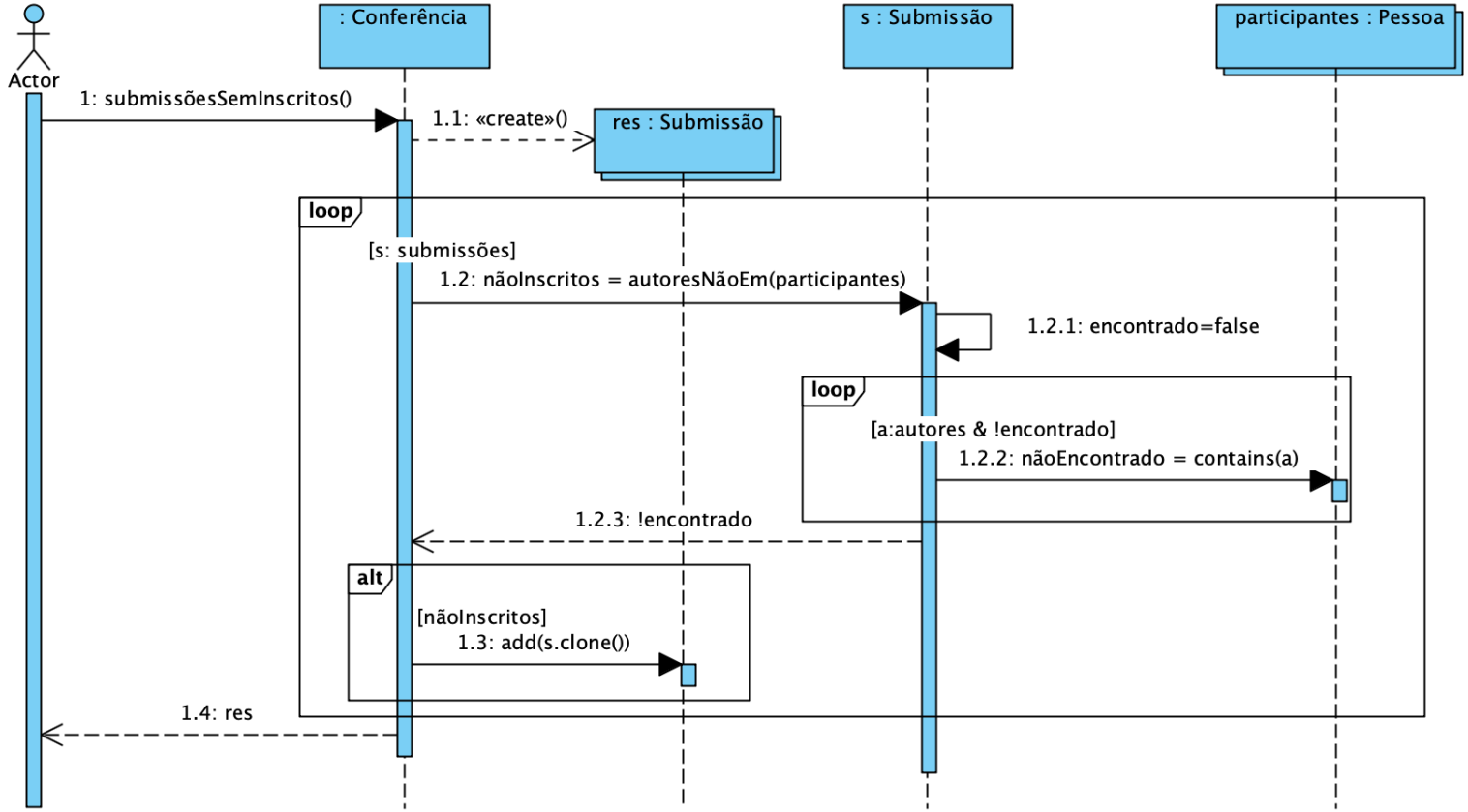
Pré- e pós-condições



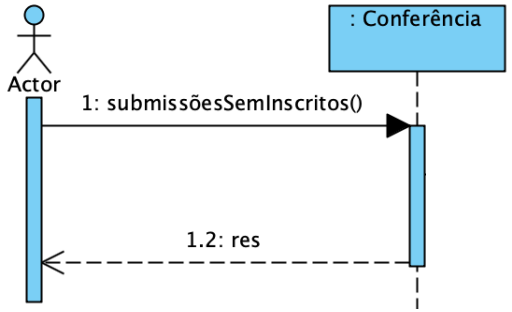

```java
public Collection<Submissão> submissõesSemInscritos() {

    return submissões.stream()
                     .filter(autoresNãoEm(participantes))
                     .map(Submissão::clone)
                     .collect(Collectors.toCollection(ArrayList::new));
}
```




```
context Conferência::submissõesSemInscritos()
post:
result = submissões->select(s|s.autoresNãoEm(participantes))
                    ->collect(s|s.clone())
```

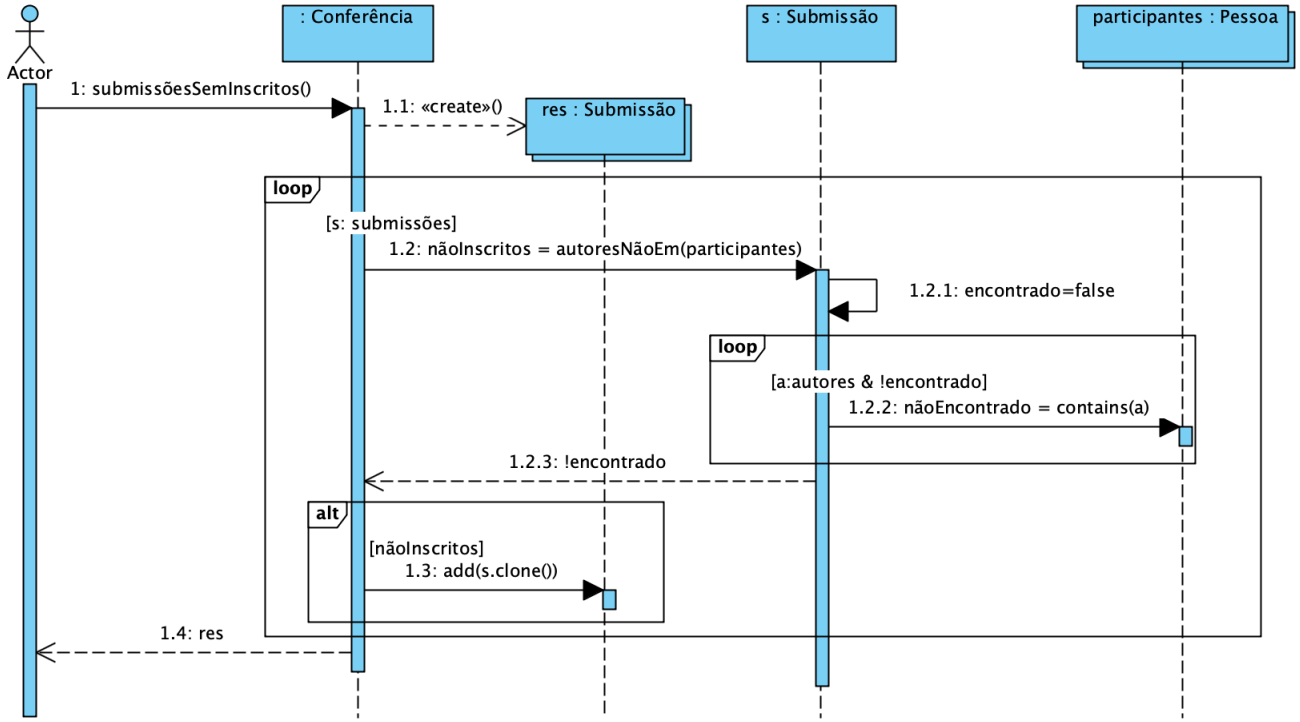


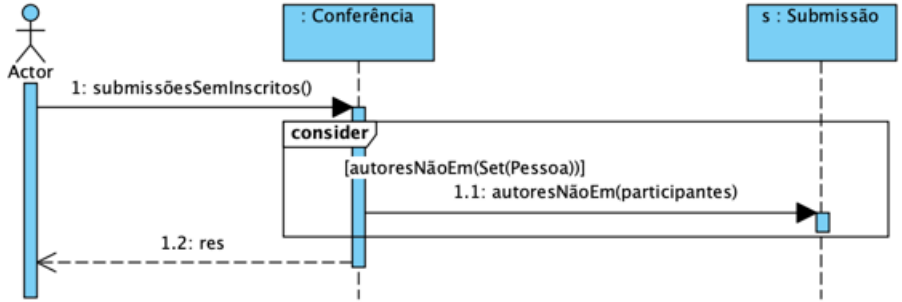
ou



- O que se ganha?
- O que se perde?

```
context Conferência::submissõesSemInscritos()
post:
result = submissões->select(s|s.autoresNãoEm(participantes))
                    ->collect(s|s.clone())
```






```
context Conferência::submissõesSemInscritos()
post:
result = submissões->select(s|s.autoresNãoEm(participantes))
                    ->collect(s|s.clone())
```

Vantagens de utilizar OCL

- Melhor documentação
  - As restrições adicionam informação acerca dos elementos e suas relações aos modelos visuais da UML
  - Permitem documentar o modelo
- Maior precisão
  - As restrições escritas em OCL têm uma semântica formal
  - Ajudam a diminuir a ambiguidade dos modelos
- Melhor Comunicação
  - Se os modelos UML são utilizados para comunicar, as expressões OCL permitem comunicar sem ambiguidade (mas perde-se alguma expressividade a nível de representação gráfica!)

Diagramas da UML 2.x