

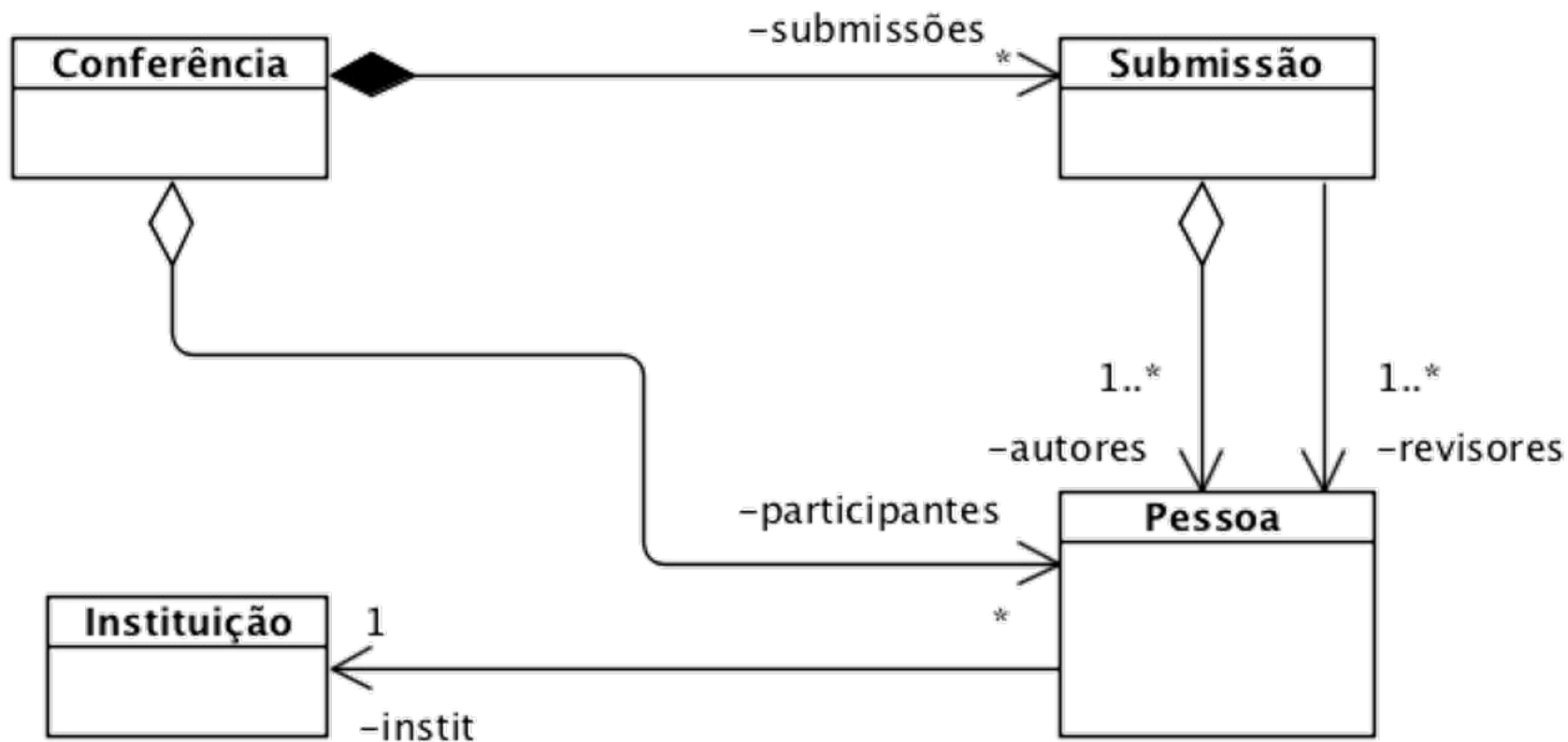


Desenvolvimento de Sistemas Software

Object Constraint Language (OCL)



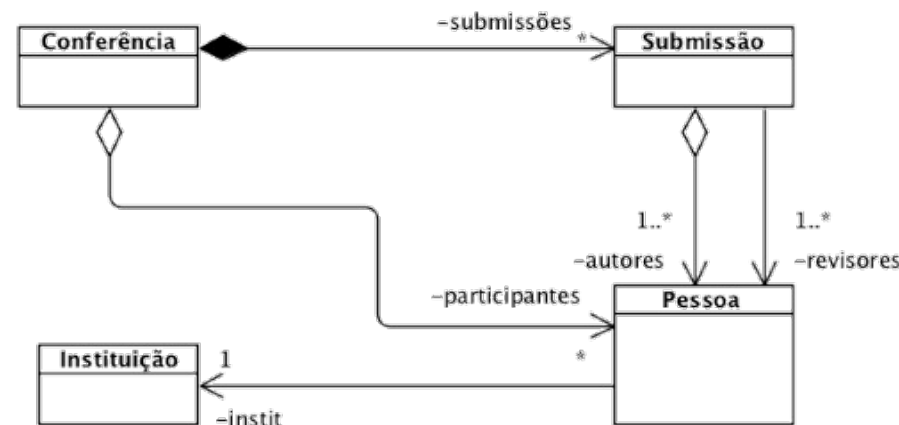
Diagramas UML nem sempre são suficientes





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 métodos
 - Restrições que especificam as condições de aplicabilidade de uma operação
- **Pós-condições** dos métodos
 - 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(),

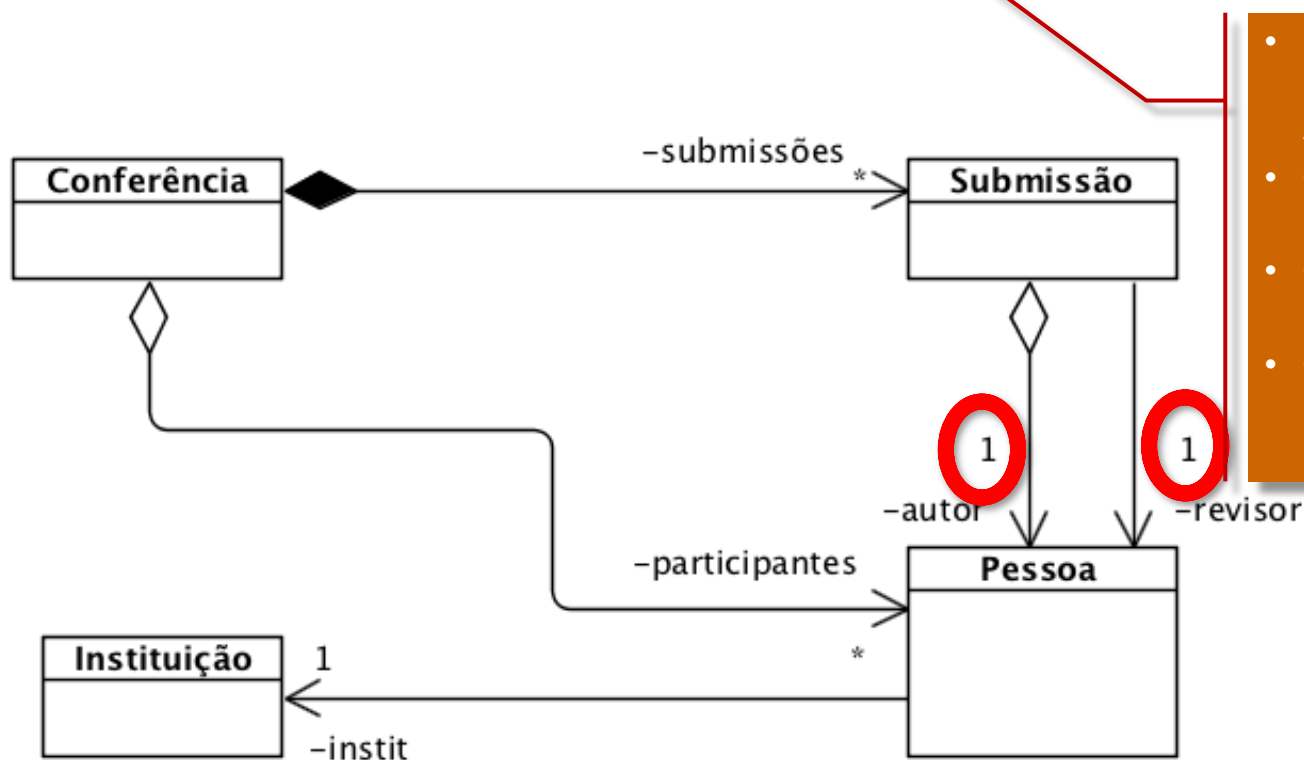


Invariantes

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

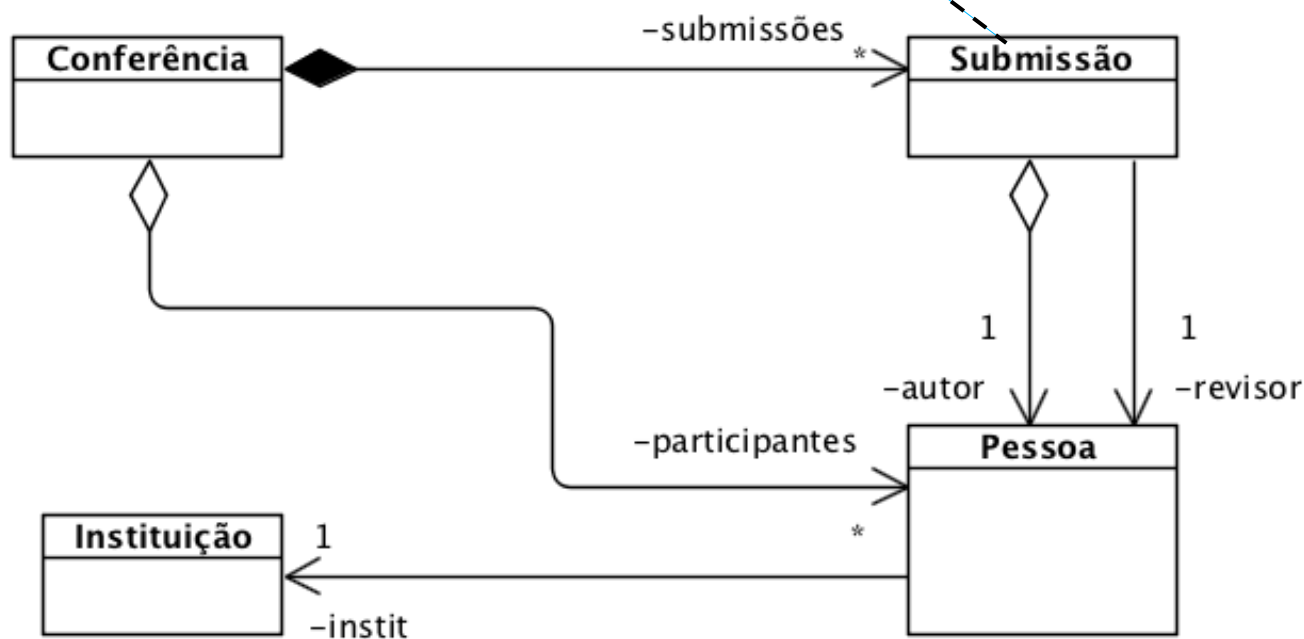


Invariantes

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

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

Identificação gráfica
do contexto



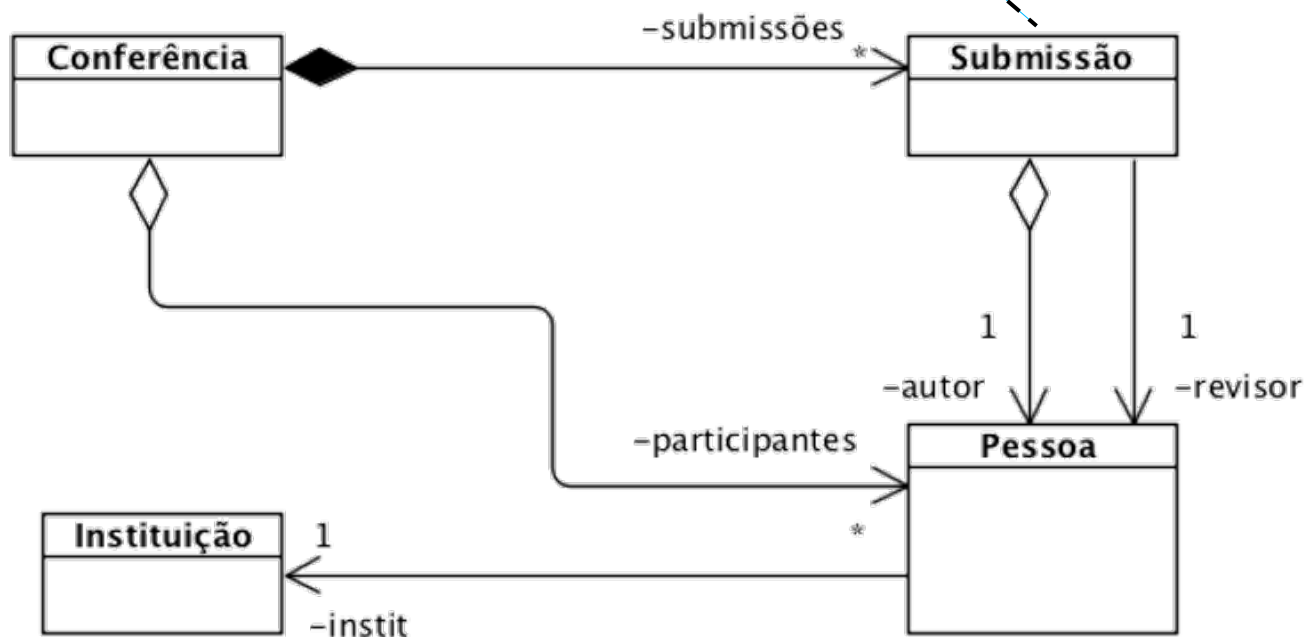


Invariantes

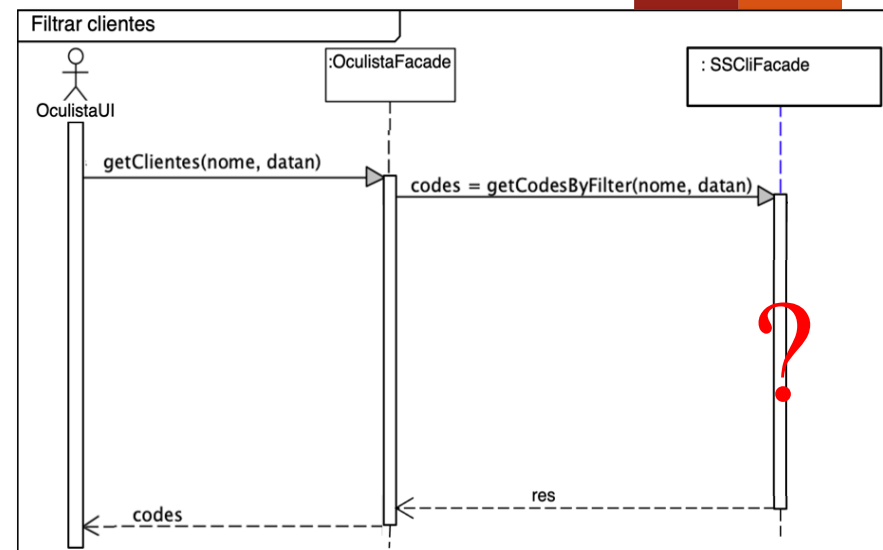
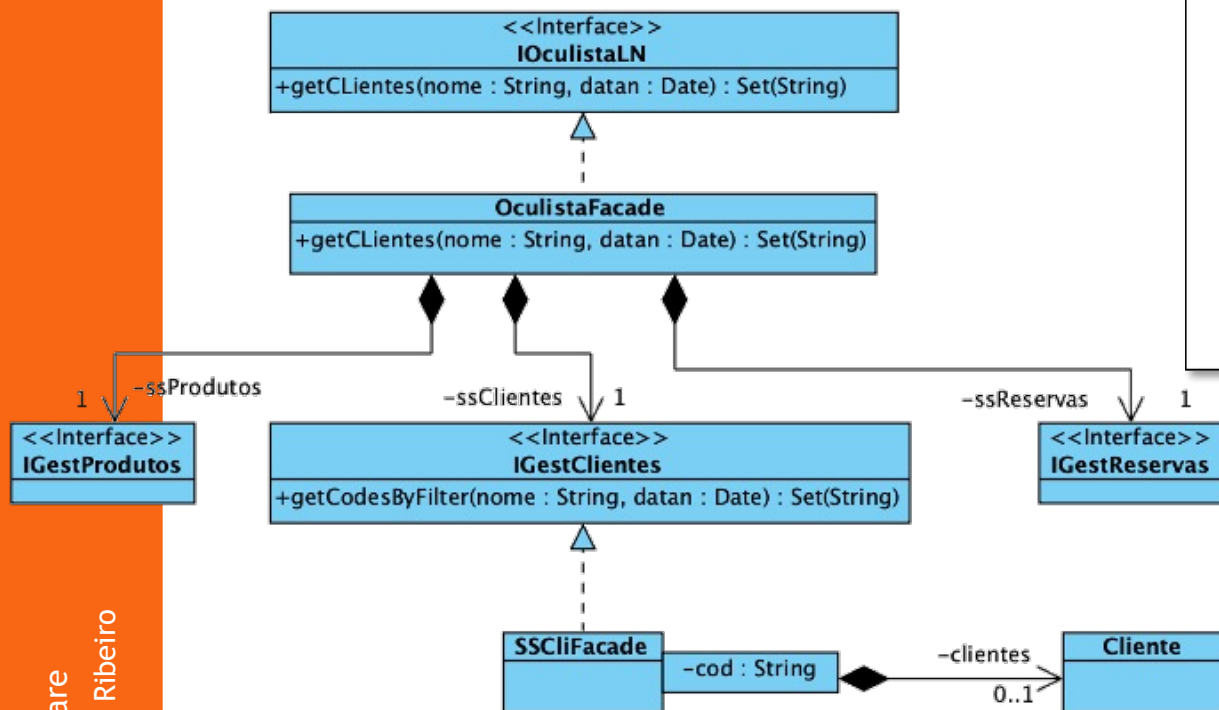
1. Os revisores de uma submissão não podem ser seus autores

inv: autor <> revisor

O nome e a referência *self* (quando não exista ambiguidade) são opcionais



Pré- e pós-condições



Programador A:

```

public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    Set<String> res = new HashSet<>();
    for (Cliente c: clientes.values()) {
        if(c.match(nome, datan)) {
            res.add(c.getCodCli());
        }
    }
    return res;
}

```

Programador B:

```

public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    return clientes.values()
        .stream()
        .filter(c->c.match(nome, datan))
        .map(Cliente::getCodCli)
        .collect(Collectors.toSet());
}

```

Programador C:

```

public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    return clientes.values().stream().map(Cliente::clone).map(c->c.getCodCli()).collect(Collectors.toSet());
}

```



Pré- e pós-condições

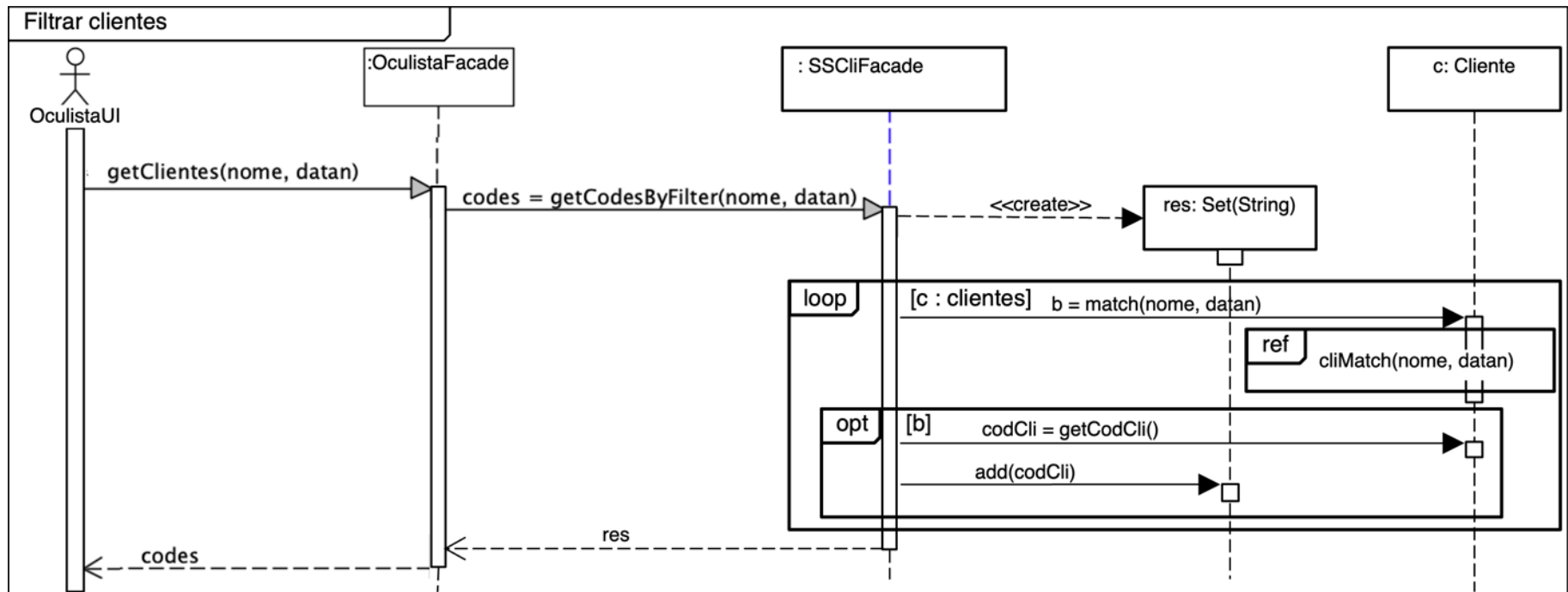
Programador A:

```
public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    Set<String> res = new HashSet<>();
    for (Cliente c: clientes.values()) {
        if(c.match(nome, datan)) {
            res.add(c.getCodCli());
        }
    }
    return res;
}
```



Programador B:

```
public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    return clientes.values()
        .stream()
        .filter(c->c.match(nome, datan))
        .map(Cliente::getCodCli)
        .collect(Collectors.toSet());
}
```





Pré- e pós-condições

Podem referir-se
atributos nos invariantes

inv: saldo >= 0

CartãoDePontos

- saldo: int

+ acumula(p: int)

+ gasta(p: int)

+ vazio(): boolean

pre: p > 0

result: resultado
da operação

post: result = (saldo=0)

post:
saldo = saldo@pre + p

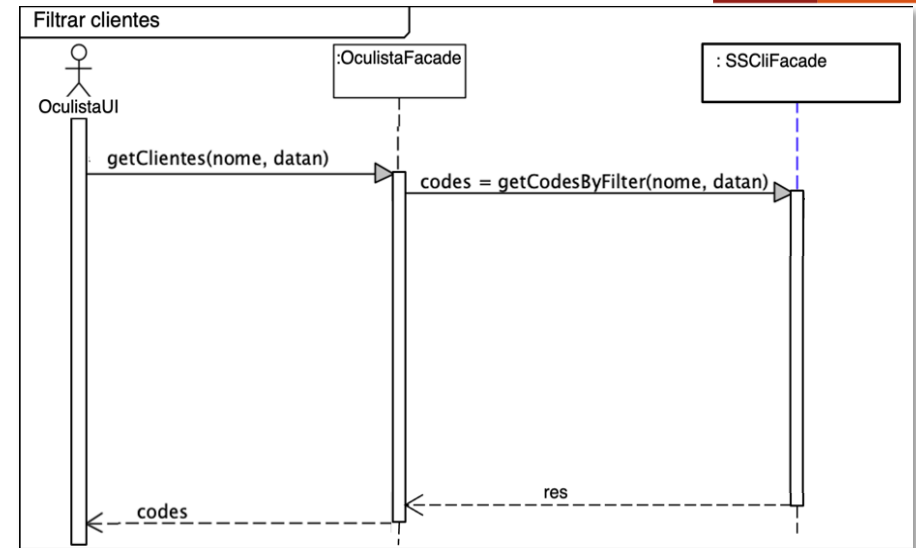
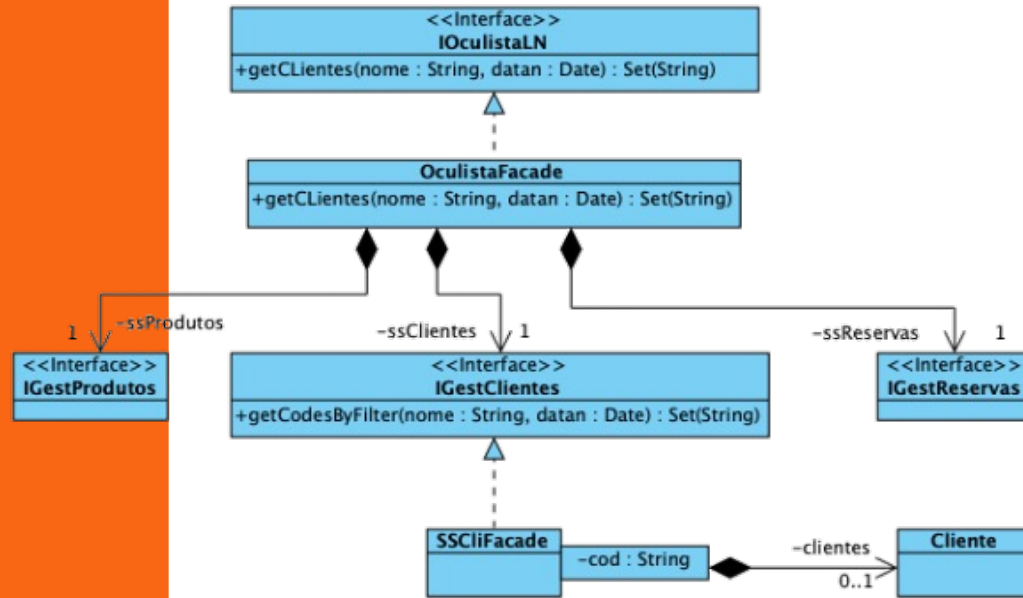
saldo@pre: valor do
saldo antes da operação

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



Programador B:

```

public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    return clientes.values()
        .stream()
        .filter(c->c. match(nome, datan))
        .map(Cliente::getCodCli)
        .collect(Collectors.toSet());
}
  
```

context `SSCLIFacade::getCodesByFilter(nome:String, datan:Date)`

post: `return = ?`



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 (<i>self</i>)
isEmpty(): Boolean	size = 0
notEmpty(): Boolean	size > 0
includes(object: T): Boolean	True if <i>object</i> is an element of <i>self</i>
excludes(object: T): Boolean	True if <i>object</i> is not an element of <i>self</i>
count(object: T): Integer	The number of occurrences of <i>object</i> in <i>self</i>
includesAll(c2: Collection(T)): Boolean	True if <i>self</i> contains all the elements of <i>c2</i>
excludesAll(c2: Collection(T)): Boolean	True if <i>self</i> contains none of the elements of <i>c2</i>
sum(): T	The addition of all elements in <i>self</i> (T must support "+")
product(c2: Collection(T2)) : Set(Tuple(first:T, second:T2))	The cartesian product operation of <i>self</i> and <i>c2</i> .



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->intercsection(s2)`

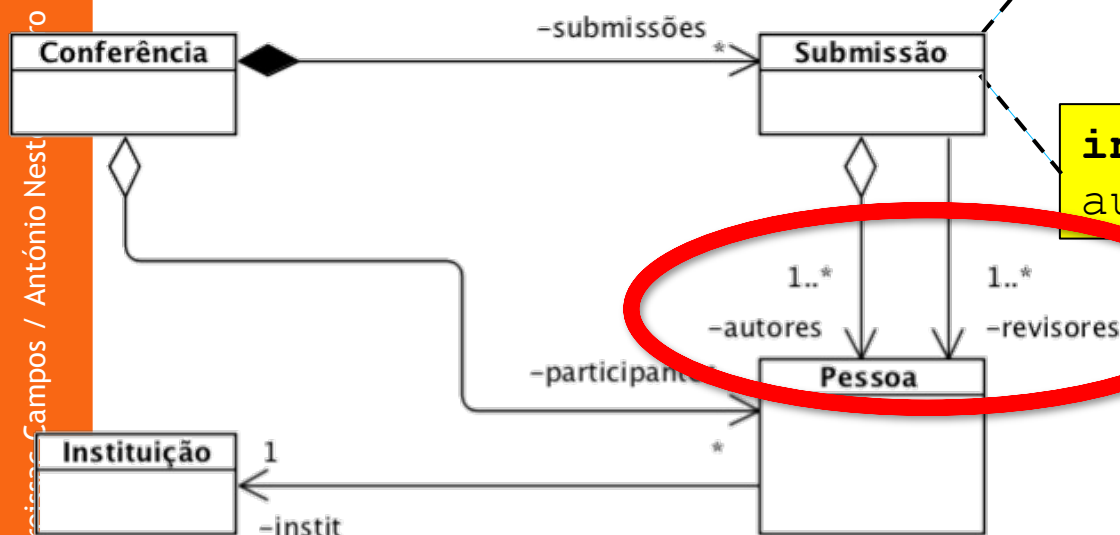


Colecções - exemplos

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

inv SemConflitoInst:
????

inv SemConflito:
autores->excludesAll(revisores)





Colecções - iteradores (tipo *map*)

Iterator expression	Description
select (iterator body): Collection(T)	The Collection of elements of the <i>source</i> collection for which <i>body</i> is true. The result collection is of the same type of the <i>source</i> collection.
reject (iterator body): Collection(T)	The Collection of elements of the <i>source</i> collection for which <i>body</i> is false. The result collection is of the same type of the <i>source</i> collection.
collect (iterator body): Collection(T2)	The Collection of elements resulting from applying <i>body</i> to every member of the <i>source</i> set. The result is flattened.
collectNested (iterator body): CollectionWithDuplicates(T2)	The Collection of elements (allowing duplicates) that results from applying <i>body</i> (of type T2) to every member of the <i>source</i> collection. The result is not flattened. Collection type conversions: Set -> Bag, OrderedSet -> Sequence.
sortedBy (iterator body): OrderedCollection(T)	Returns an ordered Collection of all the elements of the <i>source</i> collection by ascending order of the value of the <i>body</i> expression. The type T2 of the <i>body</i> expression must support "<". Collection type conversions: Set -> OrderedSet, Bag -> Sequence.



Colecções - iteradores (tipo *reduce*)

Iterator expression	Description
iterate (iterator: T; accum: T2 = init body) : T2	Returns the final value of an accumulator that, after initialization, is updated with the value of the <i>body</i> expression for every element in the <i>source</i> collection.
exists (iterators body) : Boolean	True if <i>body</i> evaluates to true for at least one element in the <i>source</i> collection. Allows multiple iterator variables.
forAll (iterators body): Boolean	True if <i>body</i> evaluates to true for each element in the source collection. Allows multiple iterator variables.
one (iterator body): Boolean	True if there is exactly one element in the <i>source</i> collection for which <i>body</i> is true
isUnique (iterator body): Boolean	Results in true if <i>body</i> evaluates to a different value for each element in the <i>source</i> collection.
any (iterator body): T	Returns any element in the source collection for which <i>body</i> evaluates to true. The result is null if there is none.

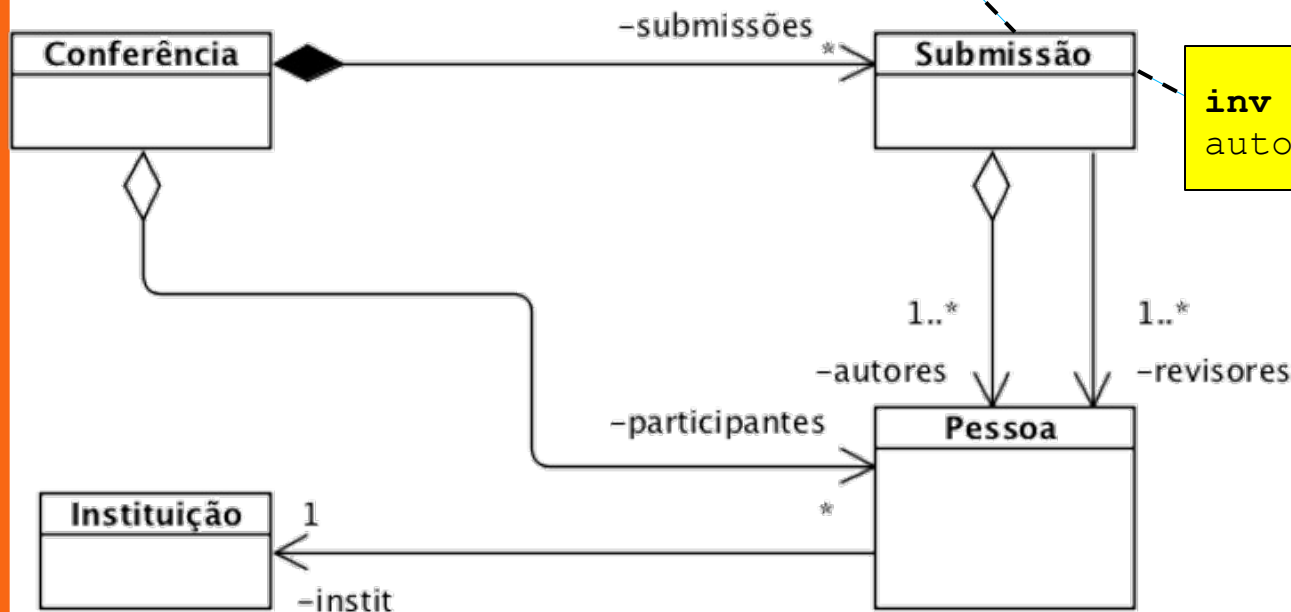
Note: The iterator variable declaration can be omitted when there is no ambiguity.

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)
```

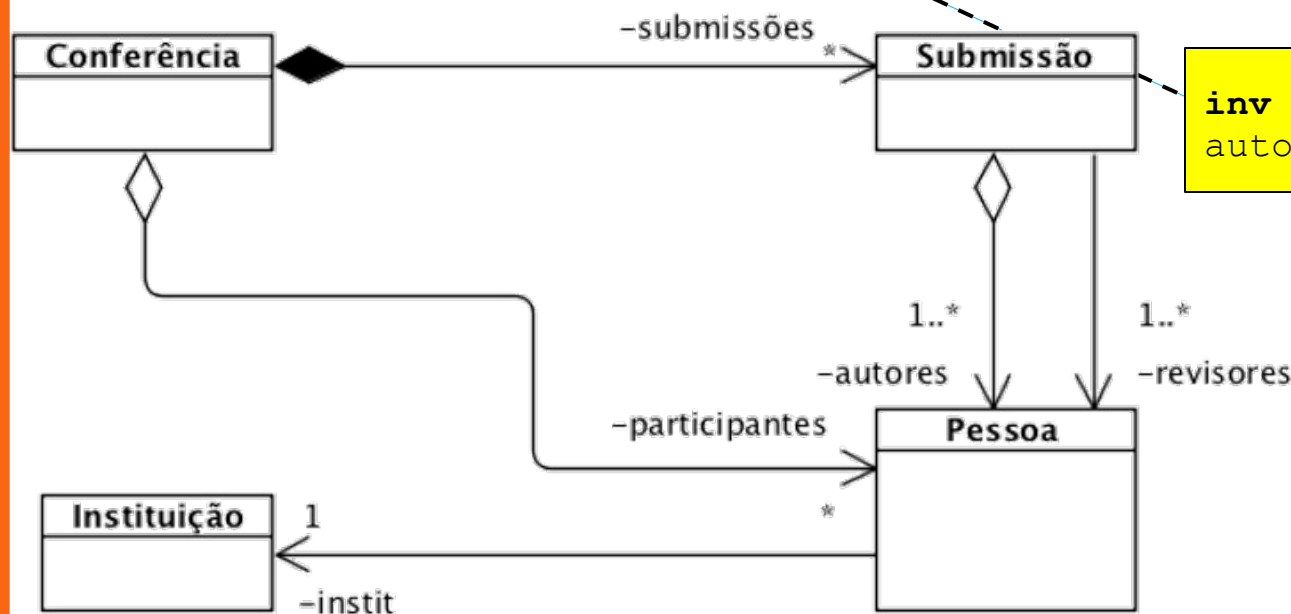


Colecções - exemplos

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)
```

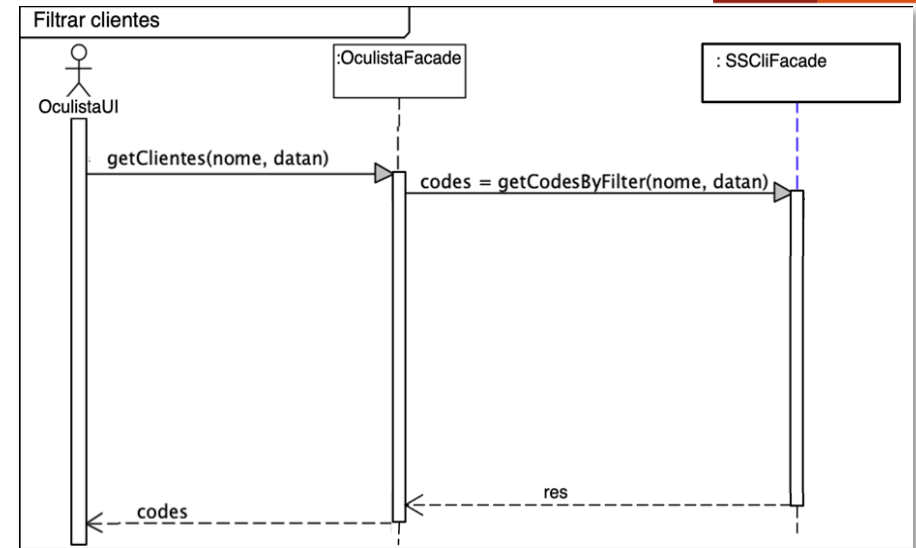
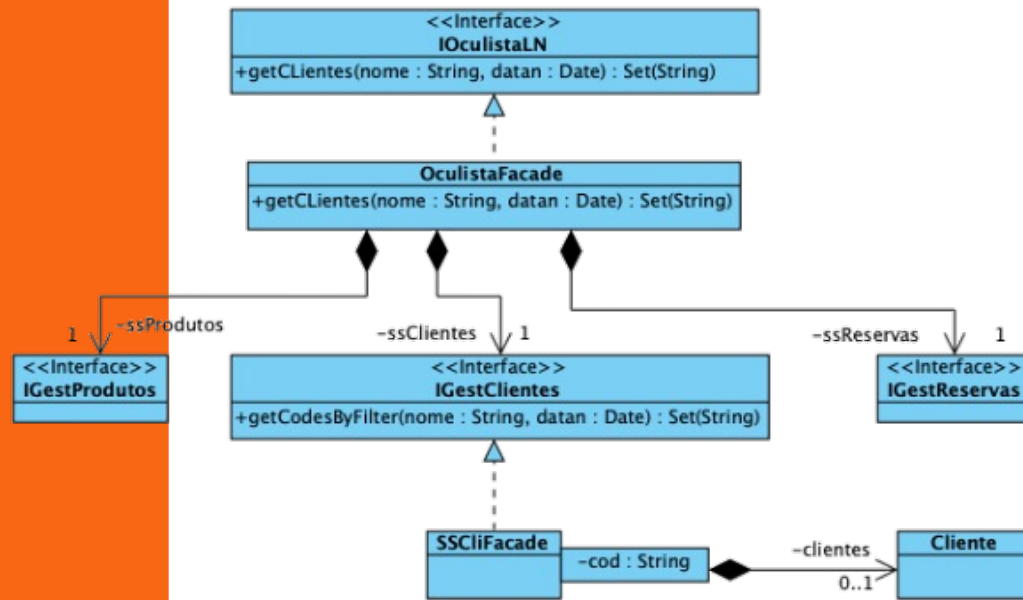
Como autores e revisores são colecções, o collect é aplicado automaticamente.



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



Pré- e pós-condições



Programador B:

```

public Set<String> getCodesByFilter(String nome, LocalDate datan) {
    return clientes.values()
        .stream()
        .filter(c->c. match(nome,datan))
        .map(Cliente::getCodCli)
        .collect(Collectors.toSet());
}
  
```

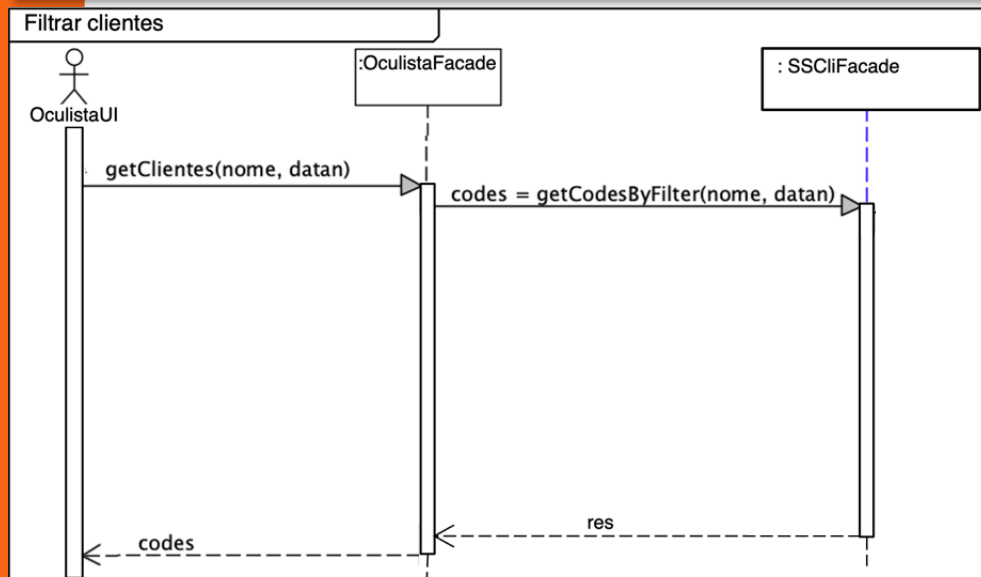
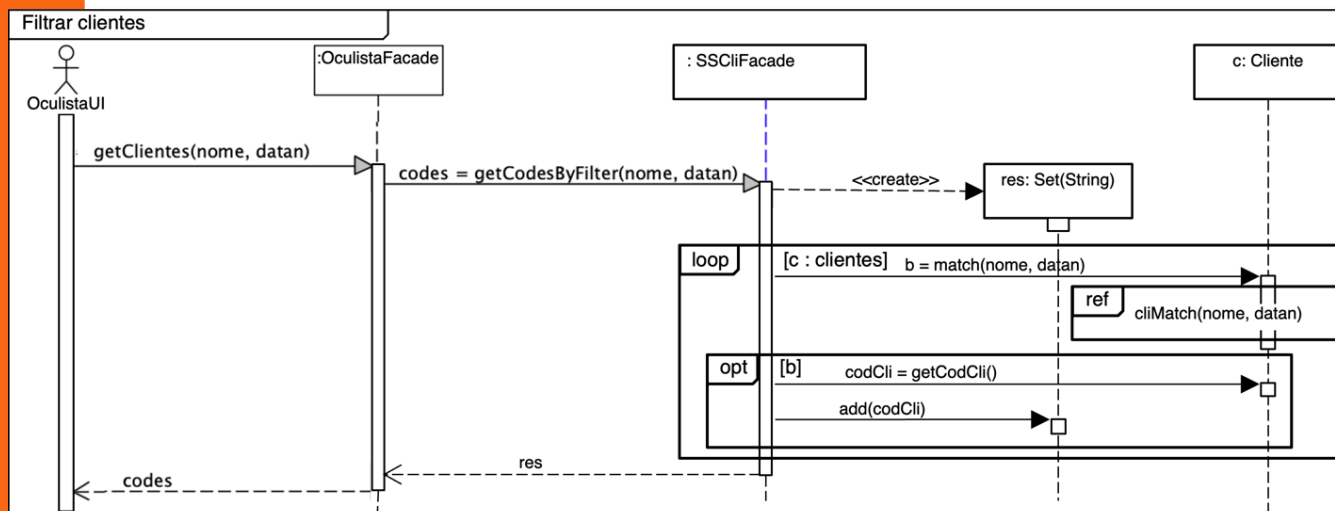
context SSCLIFacade::getCodesByFilter(nome:String, datan:Date)

post:

result = clientes->select(match(nome, datan))->collect(getCodCli())

ou:

result = clientes->select(c|c.match(nome, datan))->collect(c|c.getCodCli())



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

context SSCliFacade::getCodeByFilter(nome:String, datan:Date)
post:
 result = clientes->select(match(nome, datan))->collect(getCodCli())



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 representação gráfica!)



Diagramas da UML 2.x

