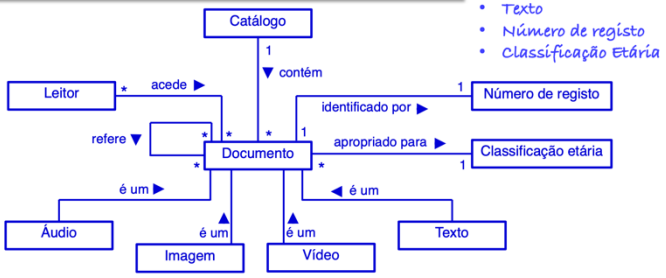
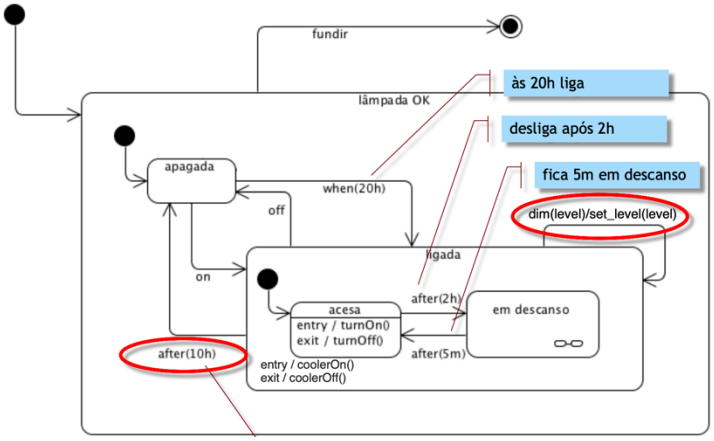
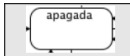
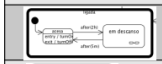
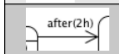
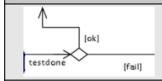
O sistema a desenvolver deverá gerir catálogos. Um catálogo contém documentos que são acedidos por leitores. Um documento pode ser do tipo áudio, imagem, vídeo ou texto e tem sempre um número de registo e uma classificação etária. Cada documento pode referir outros documentos.

Entidades:
- Catálogo
- Documento
- Leitor
- Audio
- Imagem
- Vídeo
- Texto
- Número de registo
- Classificação Etária



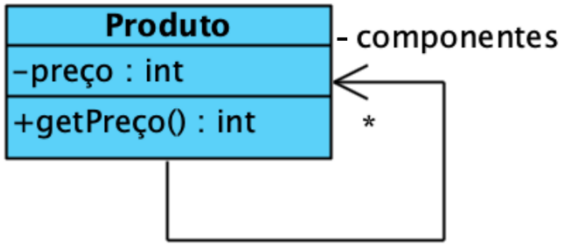
entry/*acção*	• "*acção*" é automaticamente executada quando o objecto entra no estado;
do/*acção*	• "*acção*" é continuamente executada enquanto o objecto estiver no estado;
exit/*acção*	• "*acção*" é automaticamente executada quando o objecto sai do estado;
evento/*acção*	• "*acção*" é automaticamente executada se "*evento*" ocorrer (transição interna);
evento/defer	• "*evento*" é deferido até o estado actual ser abandonado.

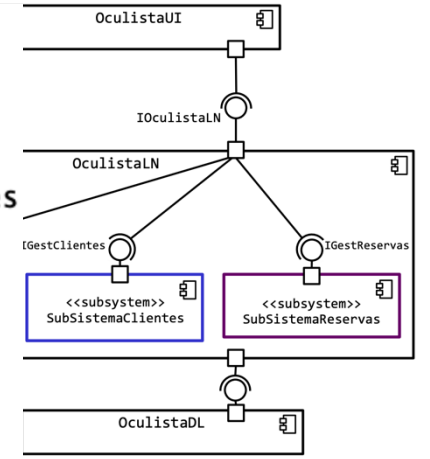


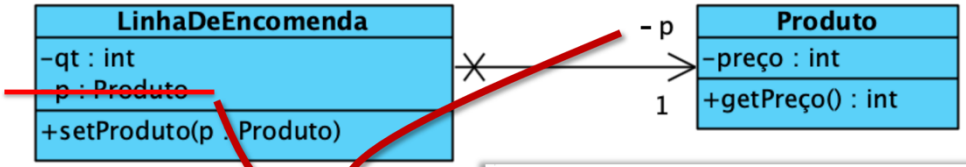
	Estado
	Estado composto
	Estado submáquina
	Pseudo-estado incial
	Estado final
	Transição (evento [condição] / acção) (entre estados vs. para o próprio estado vs. locais)
	Pseudo-estado de escolha
	Pseudo-estados de história (shallow/deep)

```
class Produto {
    private int preço;
    private Collection<Produto> componentes;

    public int getPreço() {
        return this.preço;
    }
}
```







Erro de Principiante!

Repetir as associações nos atributos.

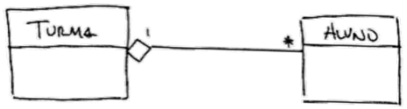
```
public class LinhaEncomenda {
    private int qt;
    private Produto p;
    private Produto p;

    public void setProduto(Produto p) {...3 lines }

}
```

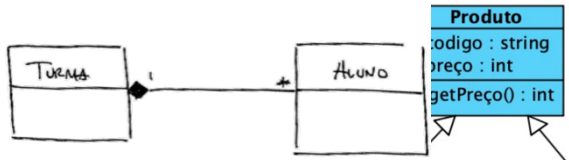
Agregação

- Os alunos fazem parte da estrutura interna da Turma
- Apesar disso, os Alunos tem existência própria

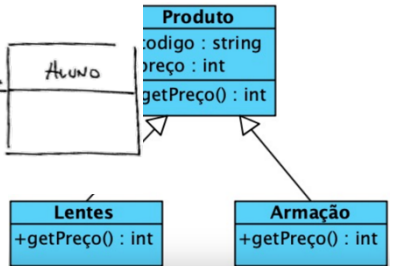


Composição

- Os alunos (da Turma) só existem no contexto da Turma
- Os alunos não têm existência para além da existência da Turma (?!)

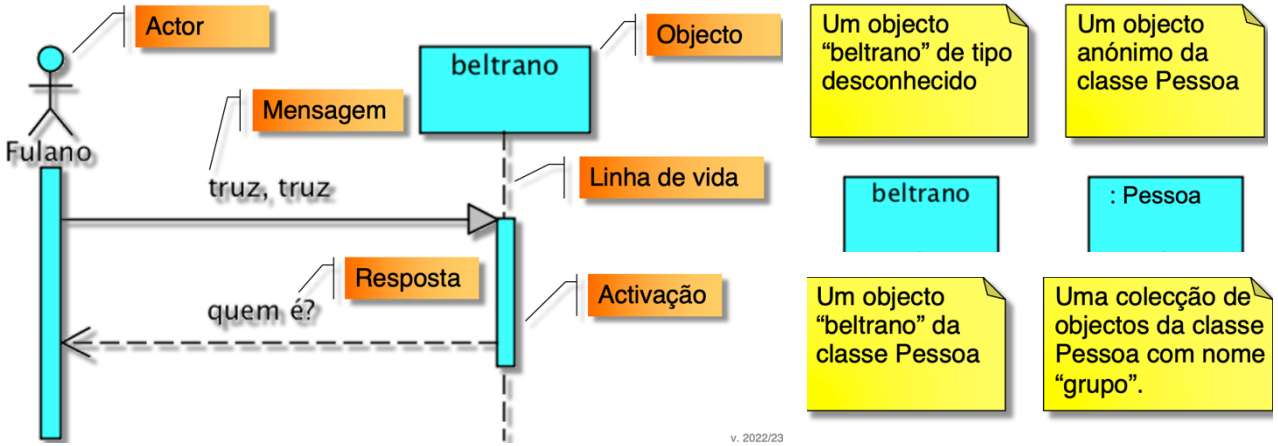


NOÇÃO DE IS-A →




```
public class Encomenda {

  private Map<Produto, LinhaEncomenda> linhas;
  ...
  public LinhaEncomenda getLinhaEnc(Produto umProd);
  public void addLinhaEncomenda(Integer qt,
                                Produto umProd);
```



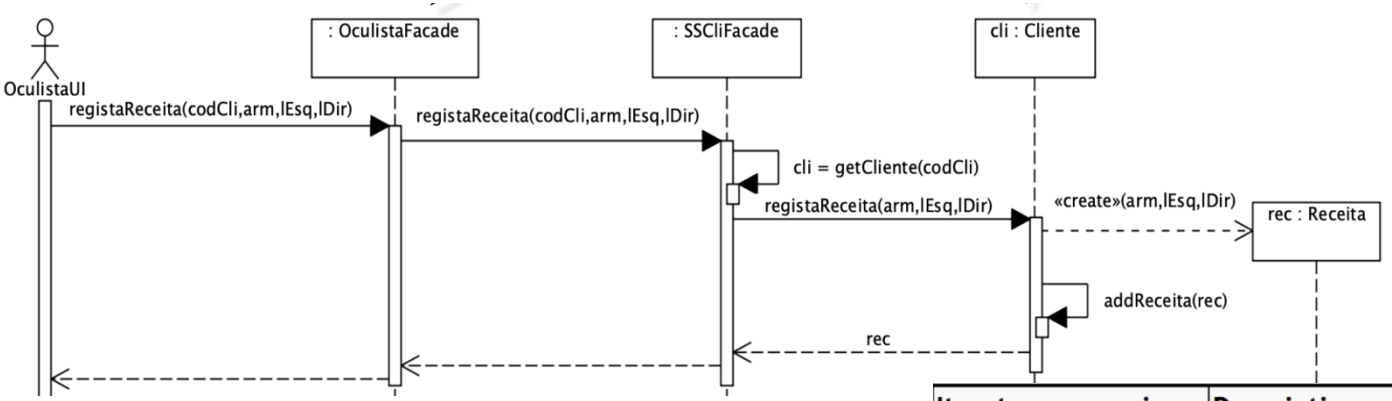
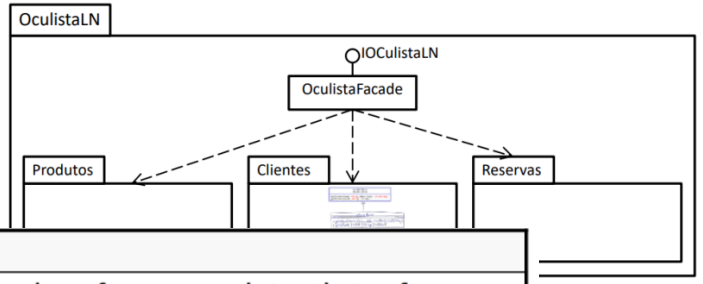


Diagrama de Package



alt - define fragmentos alternativos (mutuamente exclusivos)

loop / **loop(*n*)** - fragmento é repetido enquanto a guarda for verdadeira / ***n*** vezes

opt - fragmento opcional (ocorre se a guarda for verdadeira)

par - fragmentos ocorrem em paralelo

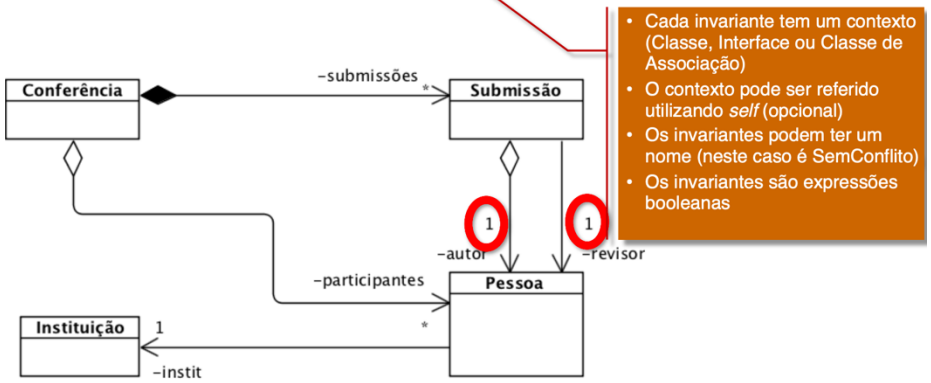
break - termina o fluxo

ref - referência a outro diagrama

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 *body* expression for every element in the *source* collection.
exists(iterators | body) : Boolean	True if *body* evaluates to true for at least one element in the *source* collection. Allows multiple iterator variables.
forAll(iterators | body): Boolean	True if *body* 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 *source* collection for which *body* is true
isUnique(iterator| body): Boolean	Results in true if *body* evaluates to a different value for each element in the *source* collection.
any(iterator | body): T	Returns any element in the source collection for which *body* evaluates to true. The result is null if there is none.

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

```
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

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}

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 "+")
product(c2: Collection(T2)) : Set(Tuple(first:T, second:T2))	The cartesian product operation of *self* and *c2*.

- 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)

Iterator expression	Description
select(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is true. The result collection is of the same type of the *source* collection.
reject(iterator | body): Collection(T)	The Collection of elements of the *source* collection for which *body* is false. The result collection is of the same type of the *source* collection.
collect(iterator| body): Collection(T2)	The Collection of elements resulting from applying *body* to every member of the *source* set. The result is flattened.
collectNested(iterator | body): CollectionWithDuplicates(T2)	The Collection of elements (allowing duplicates) that results from applying *body* (of type T2) to every member of the *source* 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 *source* collection by ascending order of the value of the *body* expression. The type T2 of the *body* expression must support "<". Collection type conversions: Set -> OrderedSet, Bag -> Sequence.

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

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