

An abstract, light blue geometric pattern consisting of interconnected lines and star-like shapes, resembling a complex network or a stylized molecular structure, is positioned in the upper left corner of the slide.

UML - Introdução

UML (Unified Modeling Language)

- Linguagem visual de modelação
 - Diagramas representam modelo do sistema
 - Ferramenta importante de comunicação
- Autores originais
 - Grady Booch
 - Ivar Jacobson
 - James Rumbaugh
- Normalização
 - OMG (Object Management Group)
 - <http://www.uml.org/>
 - Versão 2.2

Diagramas

- Estruturais
 - Estáticos (de classes, de objectos)
 - De desenho (estrutura interna, colaboração, componentes)
 - Casos de uso
- Dinâmicos
 - Máquinas de estados
 - De actividades
 - De interacção (de sequência, de comunicação)
- Físicos (de implantação)«
- Gestão de modelos (de pacotes)

Diagramas

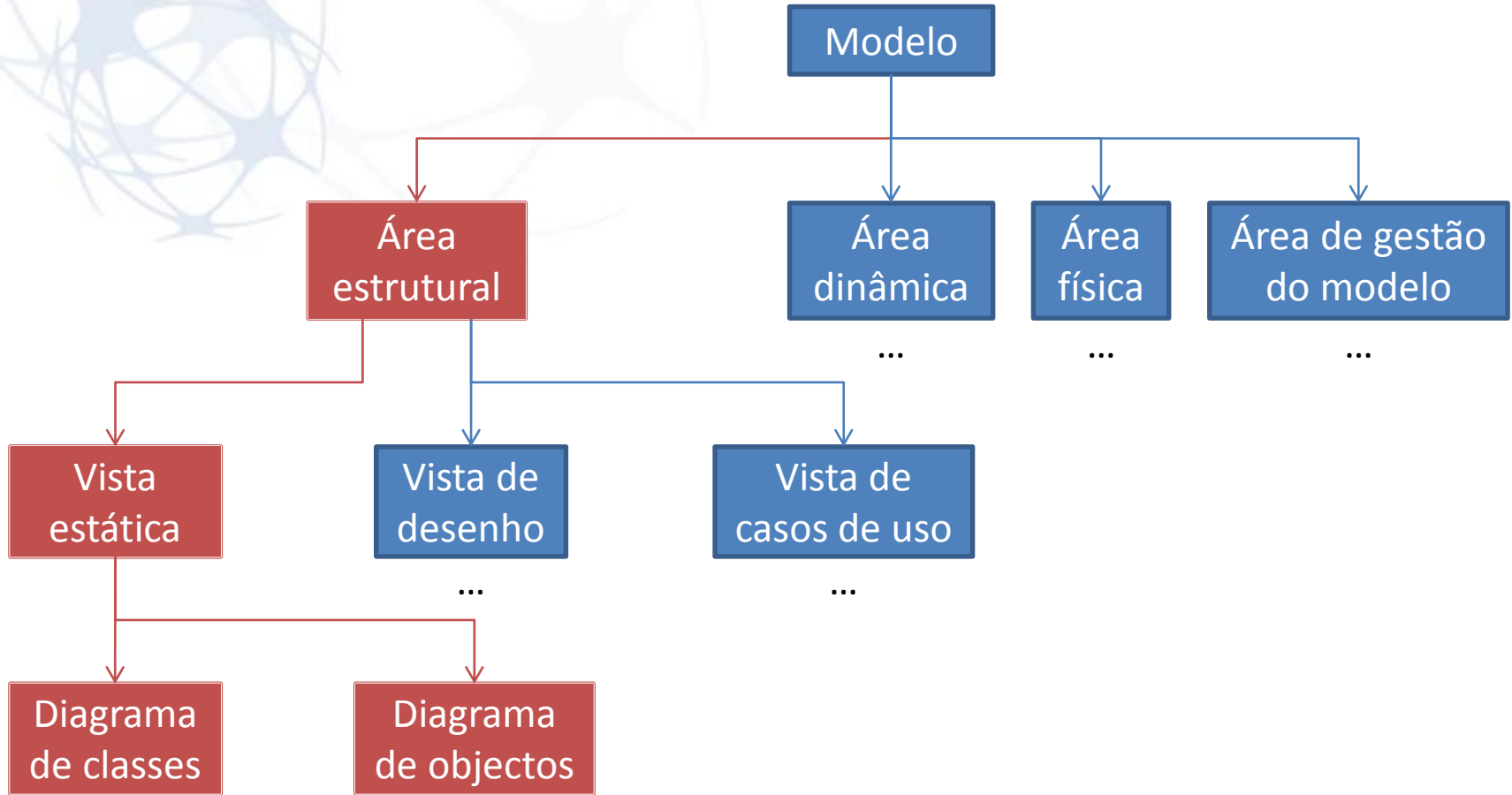


Diagrama de classes

- Representa
 - Classes
 - Relações entre classes
- Diagrama estrutural e estático
- Domínio do problema
 - Conceitos
 - Modelo de análise
- Domínio da solução
 - Classes
 - Modelo de desenho

Classes e suas relações não mudam durante execução do programa.

Modela a estrutura lógica do sistema. A perspectiva é não dinâmica: a evolução temporal do sistema em execução não é explícita.

Compreensão do problema, análise, recolha de requisitos, elaboração de glossário do domínio.

Desenho, síntese, implementação. Possível geração automática de código.

Diagrama de objetos

- Representa
 - Objectos
 - Ligações entre objetos
- Diagrama estrutural e estático

Fotografia estática do estado do sistema em execução num dado instante de tempo.

Ajuda a compreender a estrutura dos dados do sistema. A perspectiva continua a não ser dinâmica: o sistema está parado no tempo.

Classes

```
package mypackage;
public abstract
class MyAbstractClass {
    private Set<Type> set;
    private List<Type> list;
    private TreeSet<Type>
        sortedSet;
    public static final Type constant = val
    public MyAbstractClass() {...}
    private Type privateFunction(final Type
    void packagePrivateProcedure() {...}
    protected abstract Type abstractProtect
    public static void classPublicProcedure() {...}
}
```

Classe representada por caixa com compartimentos.

Nome

Atributos

Operações

mypackage::MyAbstractClass

- set: Type [*]
- list: Type [*] {ordered, nonunique}
- sortedSet: Type [*] {sorted}
+ constant: Type = value {frozen}

«constructor»+ MyAbstractClass()
- privateFunction(in parameter: Type):
Type
~ packagePrivateProcedure()
abstractProtectedFunction(): Type
+ classPublicProcedure()

Classes

```
package mypackage;

public class MyClass {
    private Set<Type> set;
    private List<Type> list;
    private TreeSet<Type> sortedSet;
    public static final Type constant
    public MyClass() {...}
    private Type privateFunction(final Type parameter) {...}
    void packagePrivateProcedure() {...}
    protected Type protectedFunction() {...}
    public static void classPublicProcedure() {...}
}
```

mypackage::MyClass
- set: Type [*] - list: Type [*] {ordered, nonunique} - sortedSet: Type [*] {sorted} <u>+ constant: Type = value {frozen}</u>
«constructor»+ MyClass() - privateFunction(in parameter: Type): Type ~ packagePrivateProcedure() # protectedFunction(): Type <u>+ classPublicProcedure()</u>

Objectos

```
import mypackage;
```

```
...
```

```
public class MyClassTester {
```

```
    public static void main(final String[] arguments) {
```

```
        MyClass localVariable = new MyClass();
```

```
        ...
```

```
    }
```

```
}
```

```
localVariable : mypackage::MyClass
```

```
set = ("João", "Maria", "Tomás")
```

```
list = (18, 20, 19)
```

```
sortedSet = ("IP", "POO")
```

```
constant = 20
```

Objectos

```
import mypackage;
```

```
localVariable : «ref»  
mypackage::MyClass
```

```
public class MyClassTester {
```

```
    public static void main(final String[] arguments) {
```

```
        MyClass localVariable = new MyClass();
```

```
        ...
```

```
    }
```

```
}
```

```
localVariable : mypackage::MyClass  
set = ("João", "Maria", "Tomás")  
list = (18, 20, 19)  
sortedSet = ("IP", "POO")  
constant = 20
```

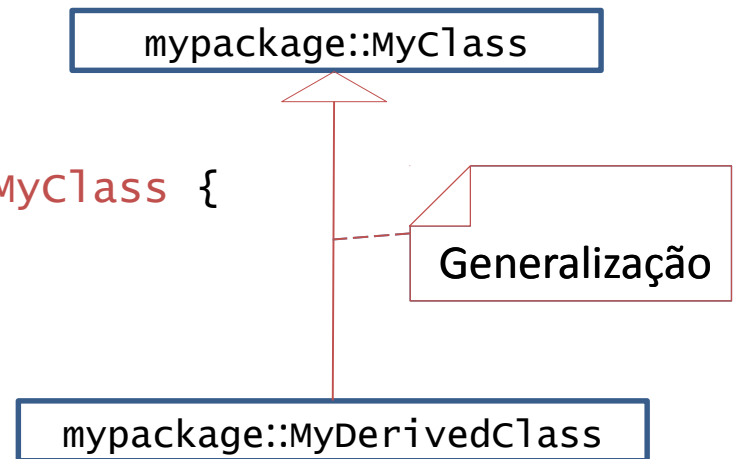
Classes: especialização

```
package mypackage;
```

```
public class MyDerivedClass extends MyClass {
```

```
...
```

```
}
```



Objectos: especialização

```
import mypackage;
```

```
localVariable: «ref»  
mypackage::MyClass
```

```
public class MyClassTester {
```

```
    public static void main(final String[] arguments) {
```

```
        MyClass localVariable = new MyDerivedClass();
```

```
        ...
```

```
    }
```

```
}
```

```
: mypackage::MyDerivedClass
```

```
set = (value3, value1, value2)
```

```
list = (value1, value2, value1)
```

```
sortedSet = (value1, value2, value3)
```

```
constant = value
```

Classes genéricas

```
package mypackage;

public class MyClass<T> {
    private Set<T> set;
    private List<T> list;
    private TreeSet<T>
        sortedSet;
    public static final T constant =
    public MyClass() {...}
    private T privateFunction(final T parameter) {...}
    void packagePrivateProcedure() {...}
    protected T protectedFunction() {...}
    public static void classPublicProcedure() {...}
}
```

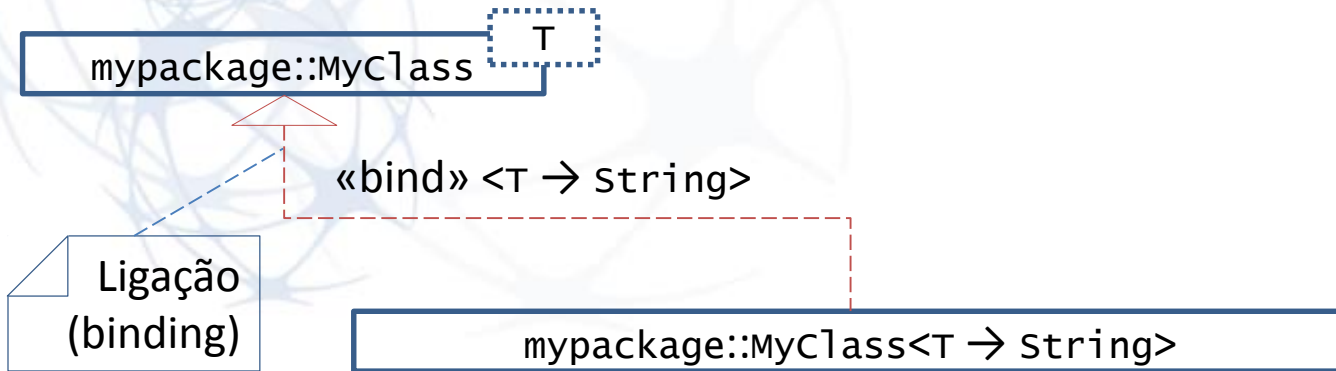
mypackage::MyClass

T

```
- set: T [*]
- list: T [*] {ordered, nonunique}
- sortedSet: T [*] {sorted}
+ constant: T = value {frozen}

«constructor»+ MyClass()
- privateFunction(in parameter: T): T
~ packagePrivateProcedure()
# protectedFunction(): T
+ classPublicProcedure()
```

Classes genéricas: ligação (*binding*)



```
public class MyClassTester {  
    public static void main(final String[] arguments) {  
        MyClass<String> localVariable = ...;  
    }  
}
```

Objetos

```
import mypackage;
```

```
...
```

```
public class MyClassTester {
```

```
    public static void main(final String[] arguments) {
```

```
        MyClass<String> localVariable = new MyClass<String>();
```

```
        ...
```

```
    }
```

```
}
```

```
: mypackage::MyClass<T → String>
```

```
set = ("string3", "string1", "string2")
```

```
list = ("string1", "string2", "string1")
```

```
sortedSet = ("string1", "string2", "string3")
```

```
constant = "string"
```

Pacotes

```
package mypackage;
```

```
...
```

```
public class MyClass {
```

```
    ...
```

```
}
```

mypackage

MyClass

Interfaces

```
package mypackage;
```

```
...
```

```
public interface MyInterface {
```

```
    Type operation();
```

```
    ...
```

```
}
```

mypackage

«interface»
MyInterface

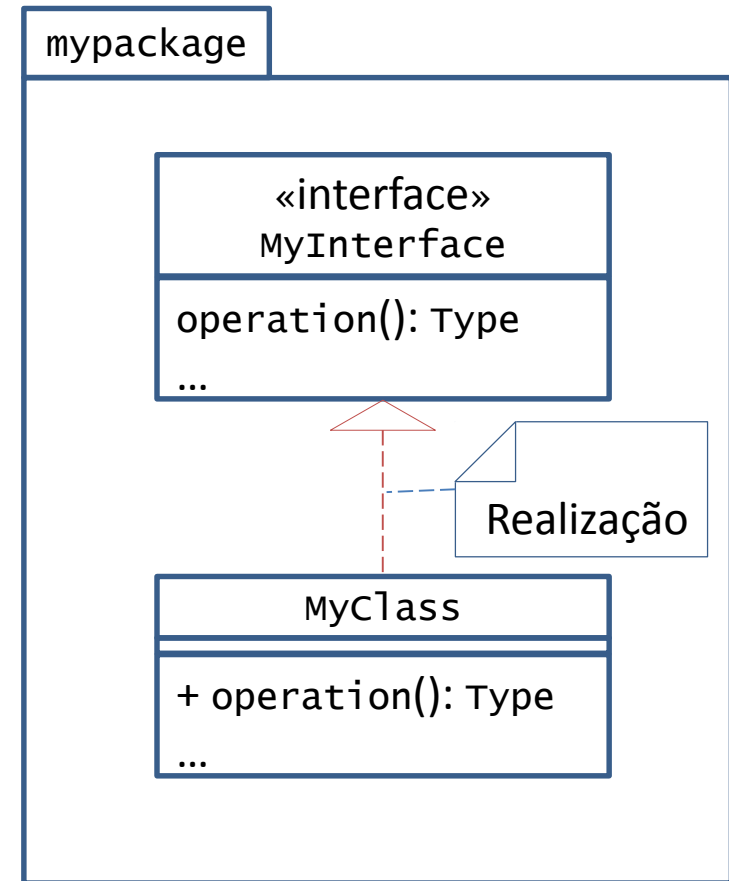
operation(): Type

...

Interfaces

```
package mypackage;
...
public interface MyInterface {
    Type operation();
    ...
}

public
class MyClass implements MyInterface {
    @Override
    public Type operation() { ... }
    ...
}
```



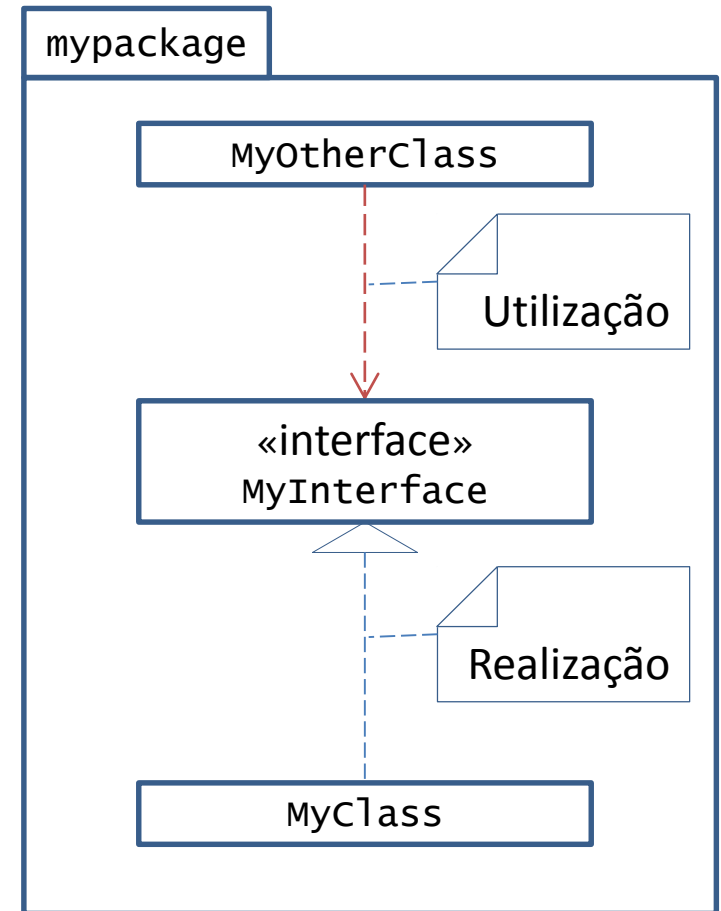
Interfaces

```
package mypackage;

public interface MyInterface { ... }

public class MyClass implements MyInterface {
    ...
}

public class MyOtherClass {
    public
        void method(final MyInterface object) {
            final Type variable =
                object.operation();
            ...
        }
    ...
}
```



Restrições

Rational
$\{\text{mdc}(\text{numerator}, \text{denominator}) = 1 \wedge 0 < \text{denominator}\}$ - numerator: int - denominator: int
...

Rational
- numerator: int - denominator: int
...

$\{\text{mdc}(\text{numerator}, \text{denominator}) = 1 \wedge 0 < \text{denominator}\}$

Principais relações entre classes

- Generalização



- Associação

- Agregação
- Composição



- Abstracção

- Realização



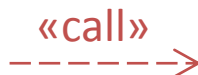
- Ligação (*binding*)



«bind» <...>

- Utilização

- Invocação
- Criação

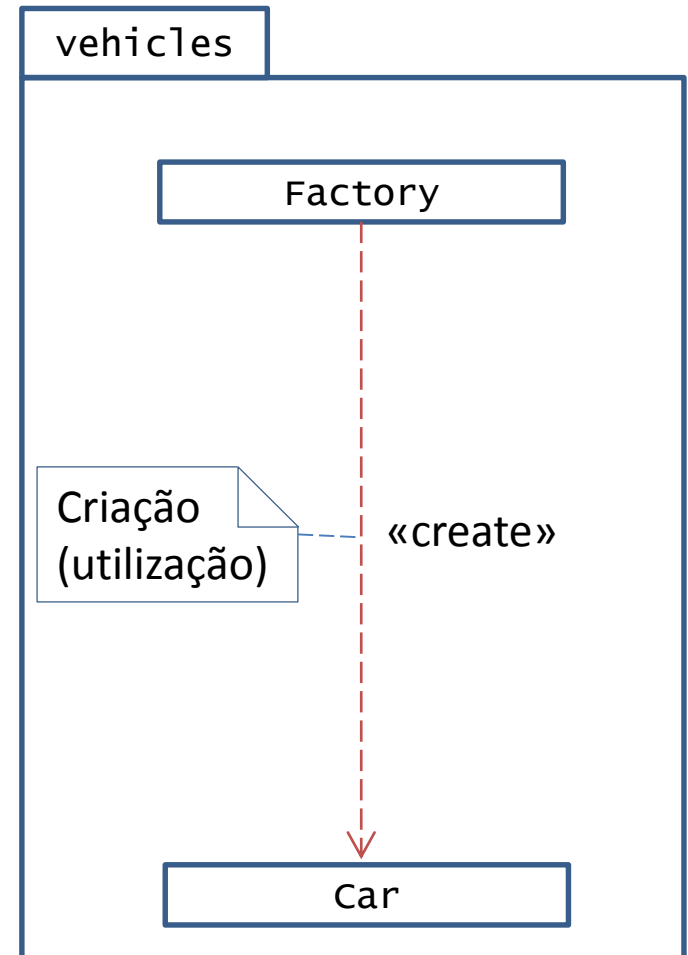


«create»

Utilização: criação

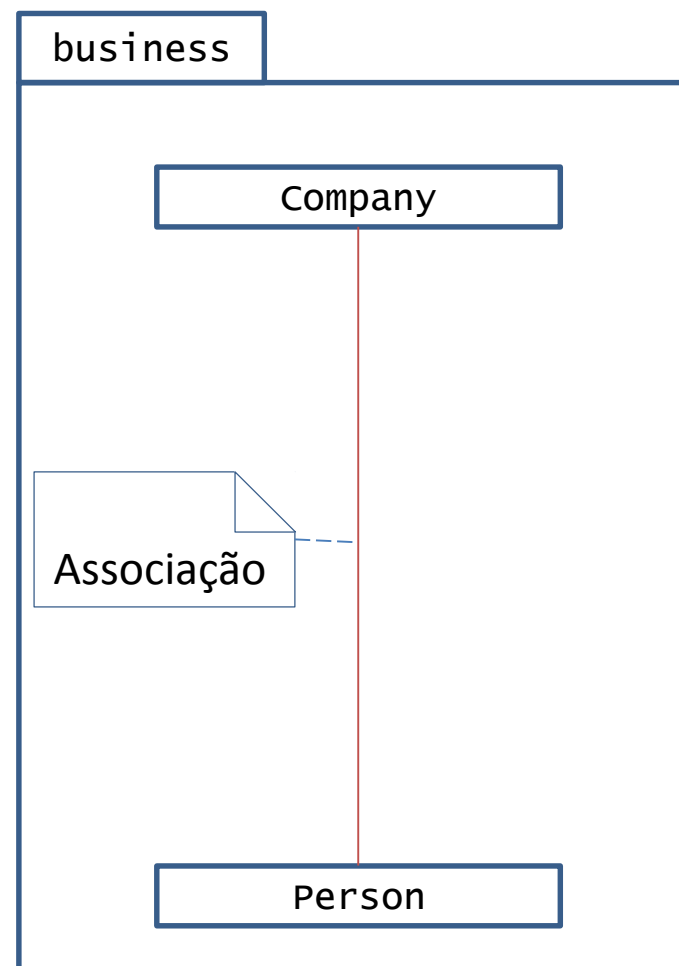
```
package vehicles;
public class Car { ... }

package vehicles;
public class Factory {
    ...
    public Car newCar(...) {
        ...
        return new Car(...);
    }
    ...
}
```



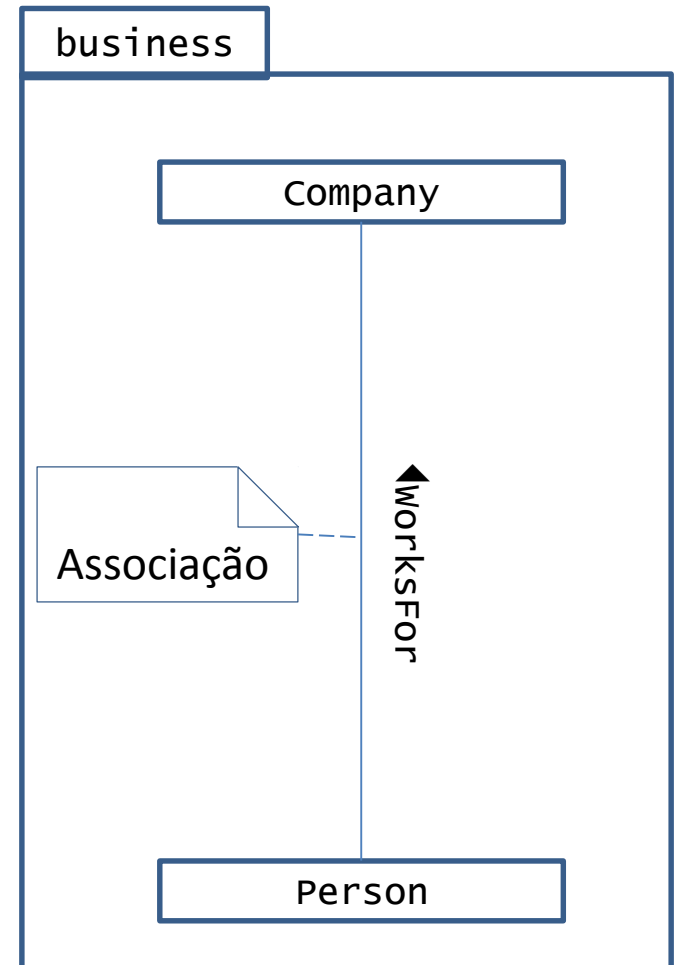
Associação

```
package business;  
public class Person { ... }  
public class Company {  
    private Person director;  
    ...  
}
```



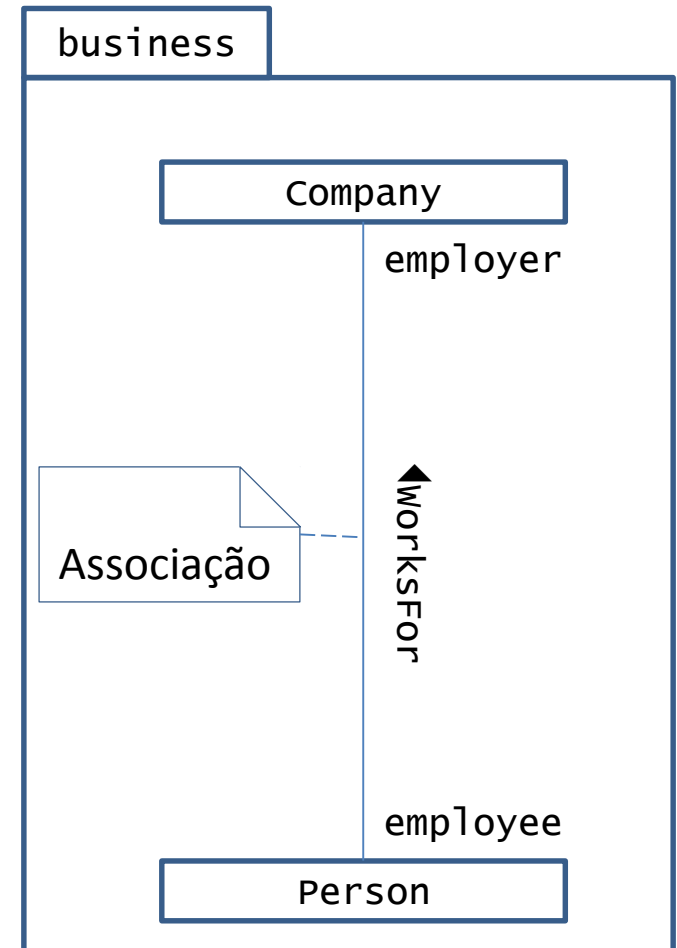
Associação: nome

```
package business;  
public class Person {  
    private Company worksFor ...  
}  
public class Company {  
}
```



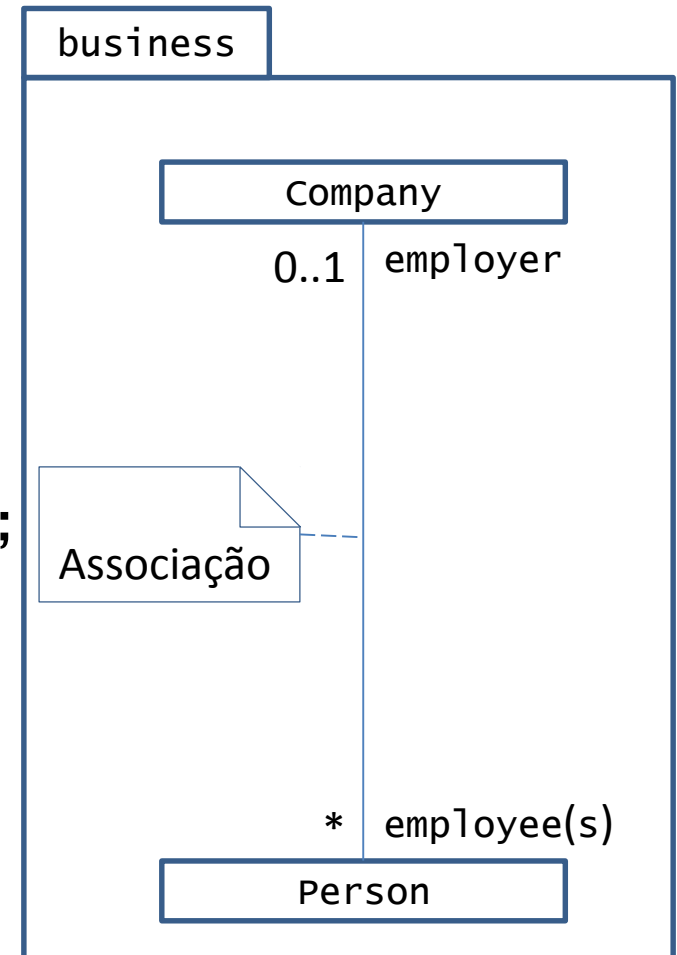
Associação: papéis

```
package business;  
public class Person {  
    private Company worksFor ...  
}  
public class Company {  
}
```



Associação: multiplicidade

```
package business;  
public class Person {  
    private Company employer;  
}  
public class Company {  
    private List<Person> employees;  
}  
  
// Normalmente só é usada  
// uma das ligações
```

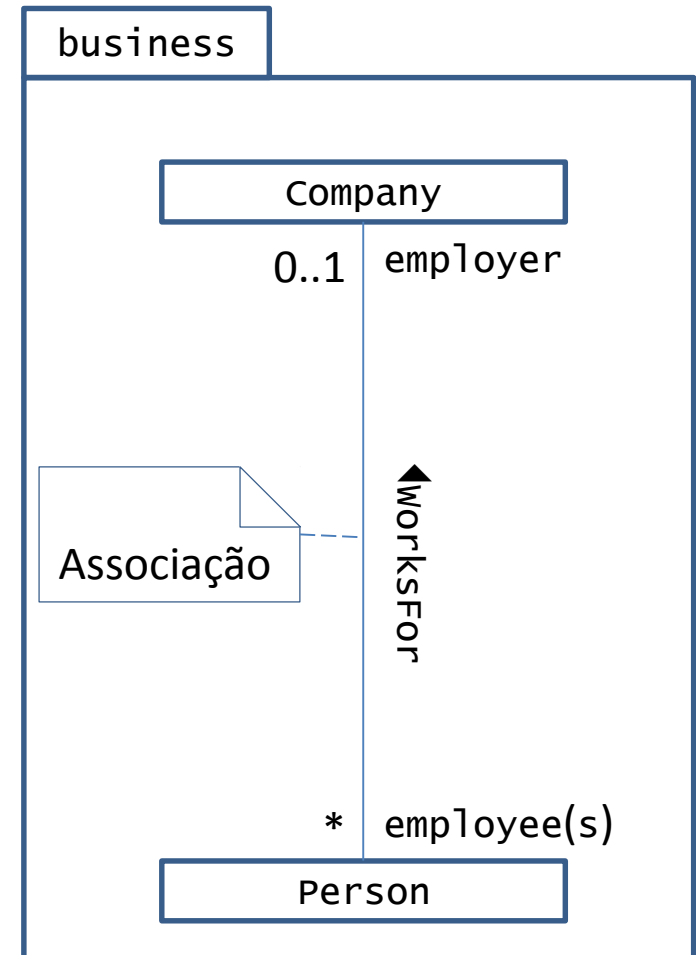


Multiplicidade

Notação	Significado
0..1	Nenhum ou um. Opcional.
1..1 1	Exactamente um. Obrigatório.
0.. <i>n</i>	De zero a <i>n</i> .
0..* *	Arbitrário. Qualquer.
<i>n</i> .. <i>n</i> <i>n</i>	Exactamente <i>n</i> .
1..*	Pelo menos 1.

Associação: representação

```
package business;  
public class Company {  
    private  
    Set<Person> employees;  
    ...  
}  
public class Person {  
    private Company employer;  
    ...  
}
```

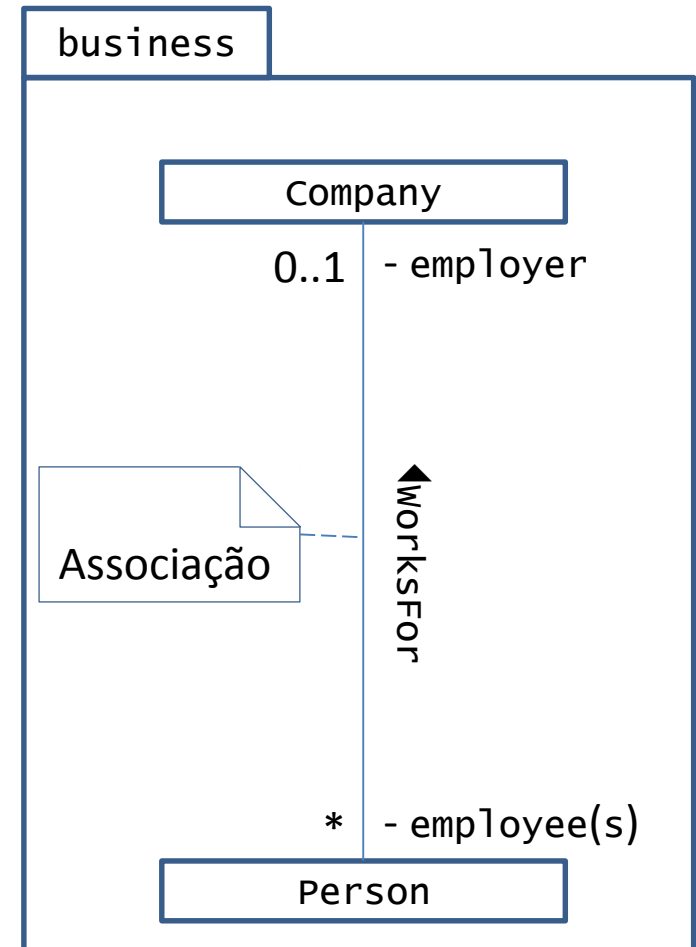


Representação e multiplicidade

Notação	Significado	Representação
0..1	Nenhum ou um. Opcional.	Atributo referência (possivelmente nulo).
1..1 1	Exactamente um. Obrigatório.	Atributo (se referência, <i>não</i> nulo). Atenção ao construtor!
0.. n	De zero a n .	Colecção (não nula) de elementos (não nulos).
0..* *	Arbitrário. Qualquer.	Colecção (não nula) de elementos (não nulos).
n .. n n	Exactamente n .	Matriz (não nula) com n elementos (não nulos). Atenção ao construtor!
1..*	Pelo menos 1.	Colecção (não nula) de elementos (não nulos). Atenção ao construtor!

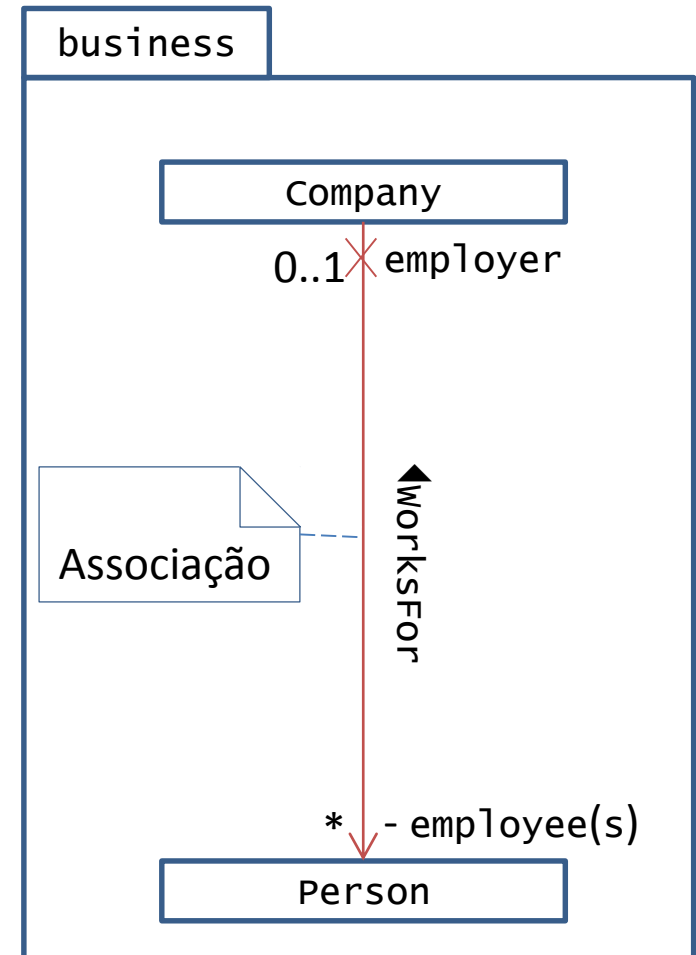
Associação: visibilidade

```
package business;  
public class Company {  
    private  
    Set<Person> employees;  
    ...  
}  
public class Person {  
    private Company employer;  
    ...  
}
```



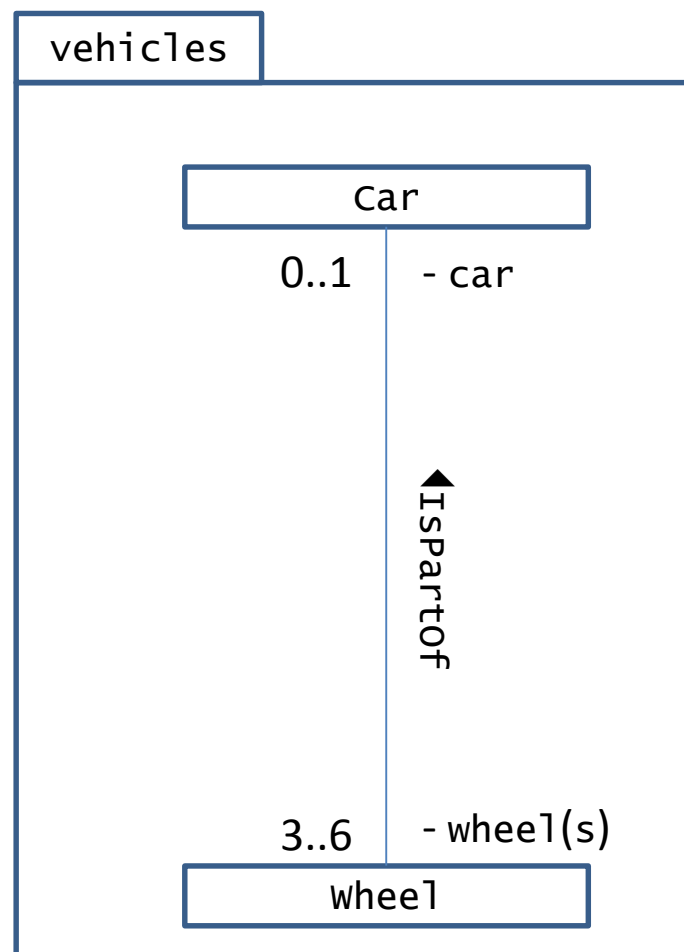
Associação: navegabilidade

```
package business;  
public class Company {  
    private  
    Set<Person> employees;  
    ...  
}  
public class Person {  
    private Company employer;  
    ...  
}
```



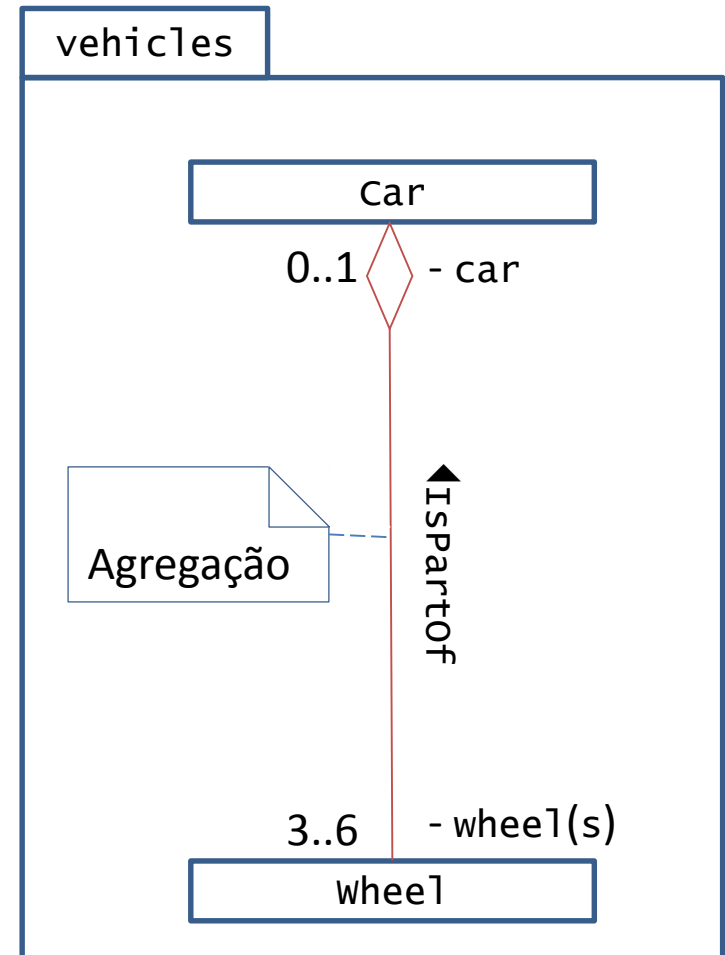
Associação

```
package vehicles;  
public class Car {  
    private Set<wheel> wheels;  
    ...  
}  
public class wheel {  
    private Car car;  
    ...  
}
```



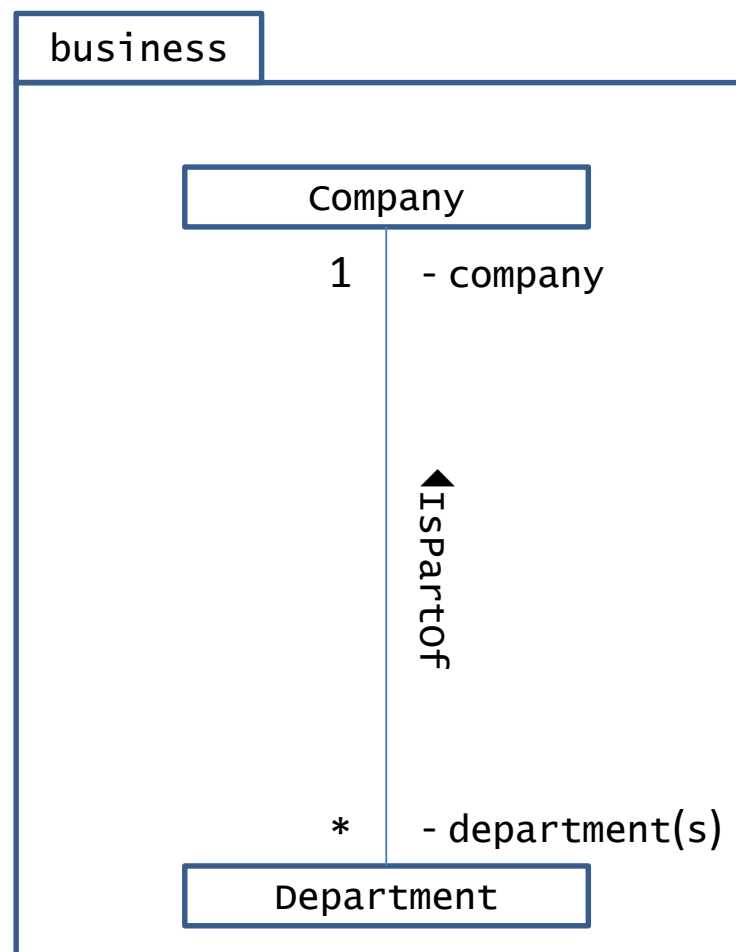
Associação: agregação

```
package vehicles;  
public class Car {  
    @Parts  
    private Set<Wheel> wheels;  
    ...  
}  
  
public class wheel {  
    @whole  
    private Car car;  
    ...  
}
```



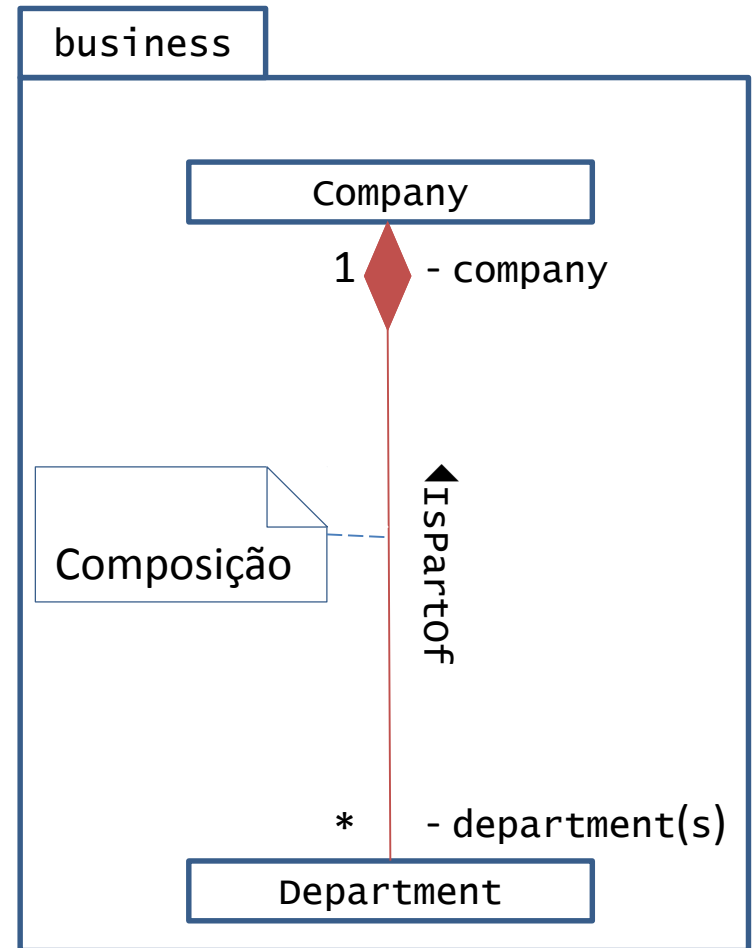
Associação

```
package business;  
  
public class Company {  
    private Set<Department>  
        departments;  
    ...  
}  
  
public class Department {  
    private Company company;  
    ...  
}
```



Associação: composição

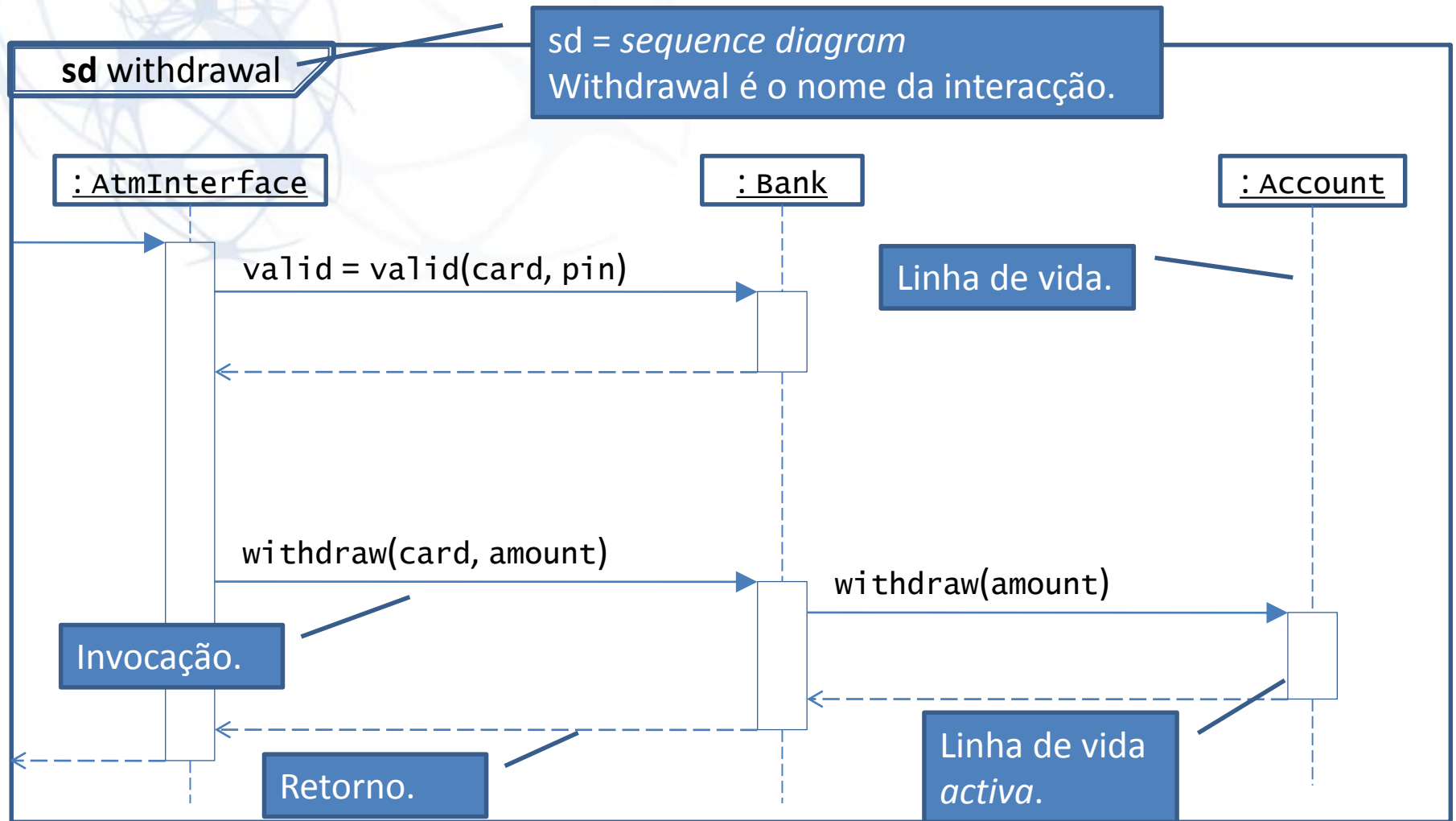
```
package business;  
public class Company {  
    @Components  
    private Set<Department>  
        departments;    ...  
}  
public class Department {  
    @Composite  
    private Company company;  
}
```



Diagramas de sequência

- Mostram **interacções** entre **entidades** numa **sequência temporal**
- Mostram
 - entidades envolvidas numa interacção
 - sequências de **mensagens** trocadas entre entidades
- Entidades podem ser
 - actores e sistema trocando mensagens (análise; domínio do problema)
 - objectos invocando operações (desenho; domínio da solução)

Diagrama de sequência



Referências

- UML® Resource Page (<http://www.uml.org/>)
- Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3.^a edição, Addison-Wesley, 2003.
ISBN: 0-321-19368-7
(1.^a e 2.^a edições na biblioteca)
- James Rumbaugh *et al.*, *The Unified Modeling Language Reference Manual*, 2.^a edição, Addison-Wesley, 2005.
ISBN: 0-321-24562-8
(1.^a edição do guia do utilizador na biblioteca)

Sumário

- Introdução ao UML
 - Noções
 - Áreas, vistas e tipos de diagrama
 - Níveis de pormenor
 - Notações para classes, objectos e relações
 - Notações para as relações mais importantes, incluindo a generalização, a associação, a agregação e a composição