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

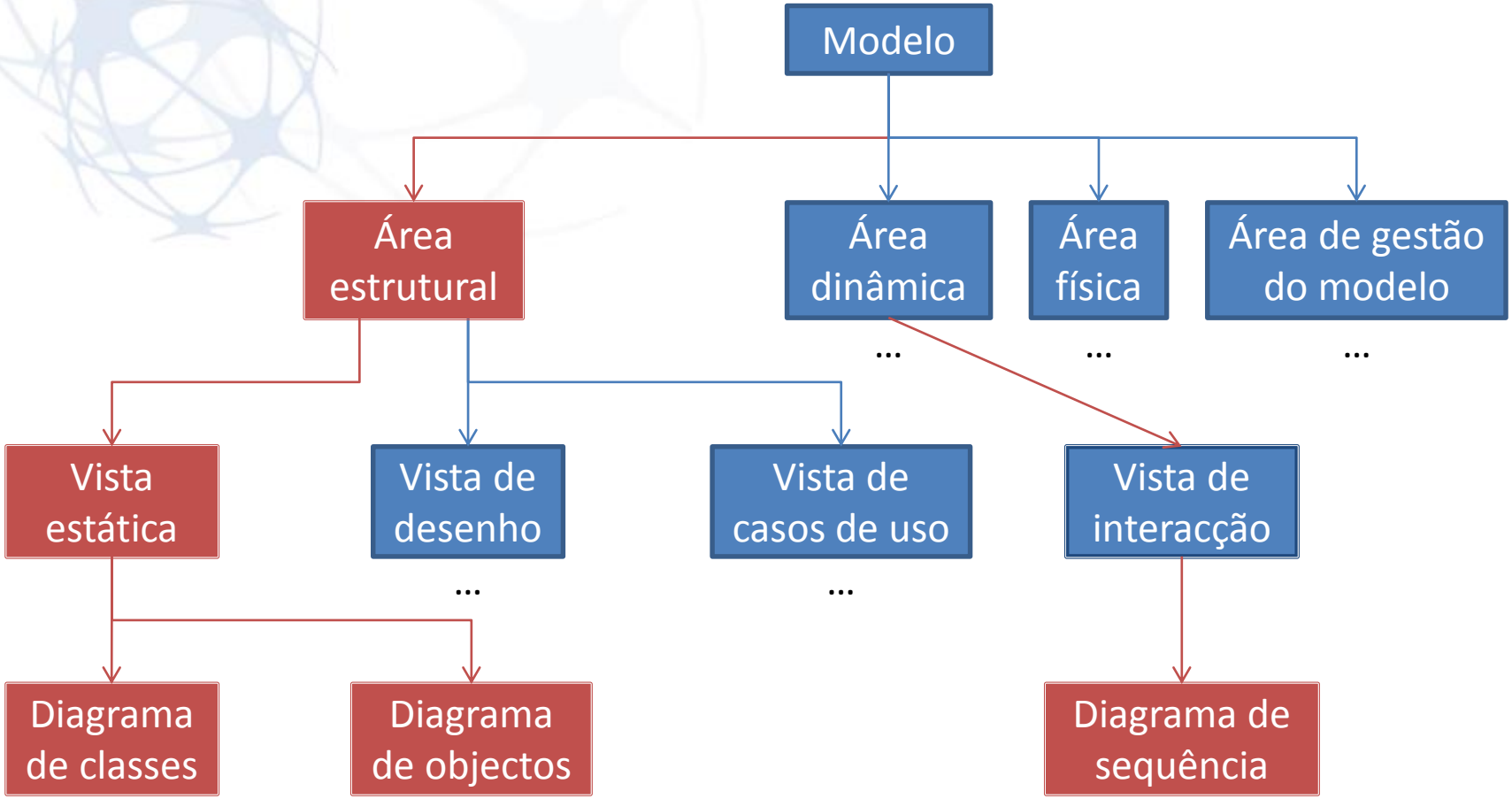

Modelo
Área estrutural
Área dinâmica
Área física
Área de gestão do modelo
...
...
...
Vista estática
Vista de desenho
Vista de casos de uso
Vista de interacção
...
...
Diagrama de classes
Diagrama de objectos
Diagrama de sequência

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

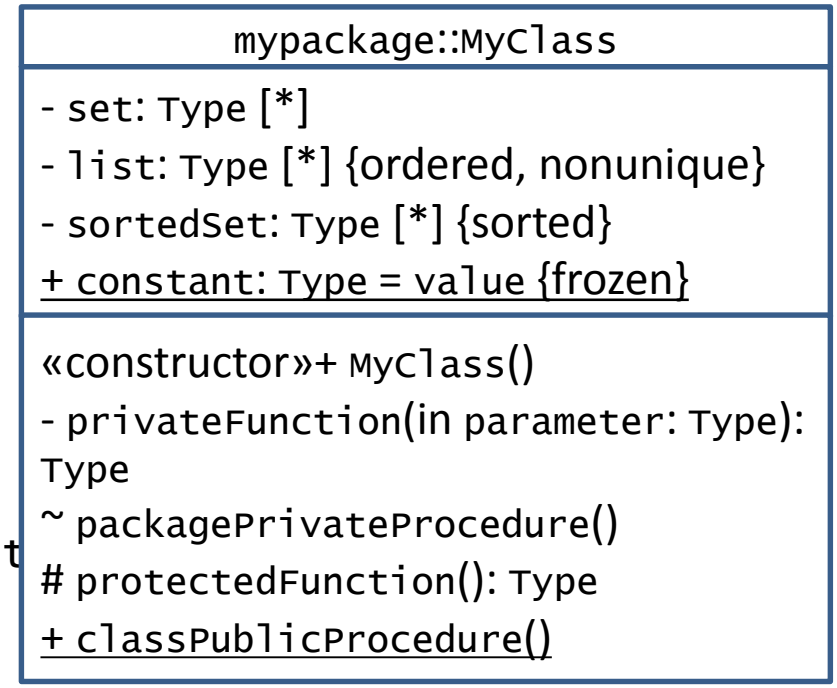
- Representa
  - Objectos
  - Ligações entre objetos
- Diagrama estrutural e estático

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

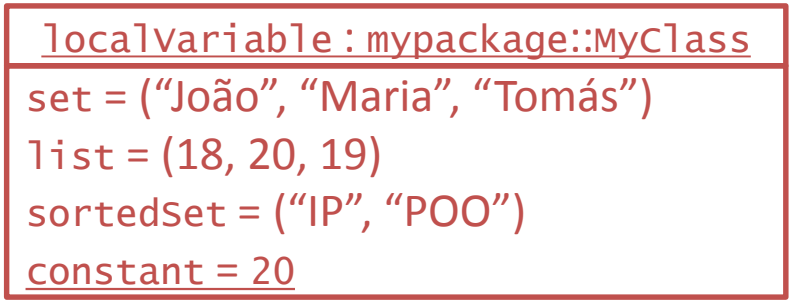
Ajuda a compreender a estrutura dos dados do sistema. A perspectiva continua a não ser dinâmica: o sistema está parado no tempo.

Classes

```
package mypackage;
public 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() {…}
}
```



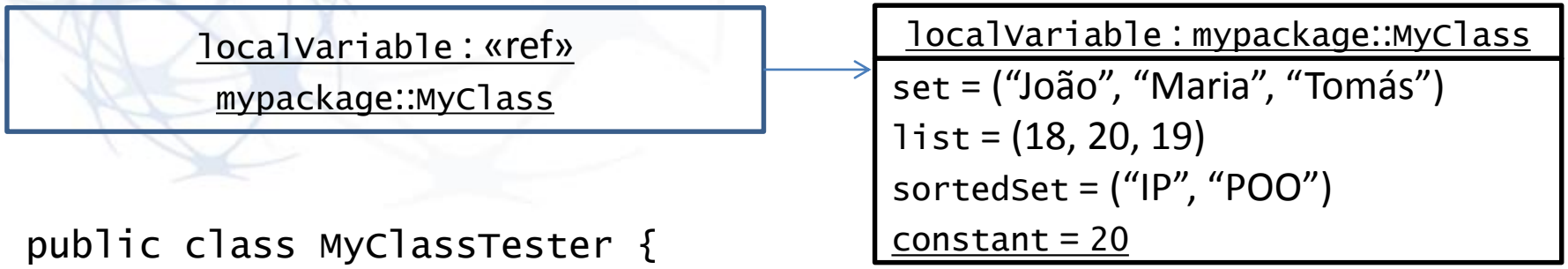
Objectos




```
import mypackage;
…
public class MyClassTester {
    public static void main(final String[] arguments) {
        MyClass localVariable = new MyClass();
        …
    }

}
```

Objectos

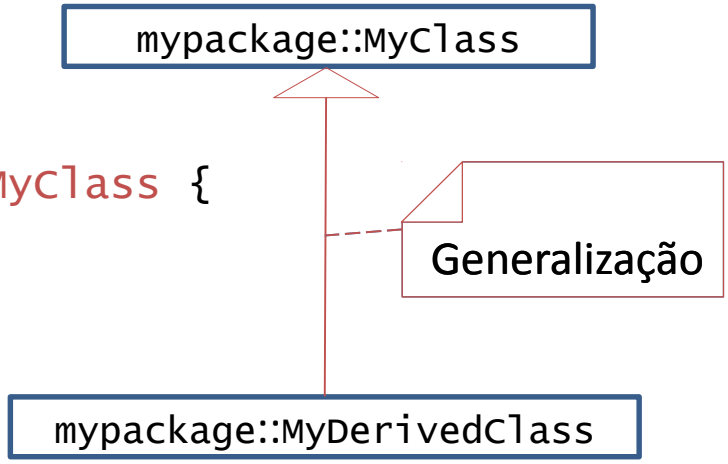
```
import mypackage;
```




```
public class MyClassTester {

    public static void main(final String[] arguments) {
        MyClass localVariable = new MyClass();
        …
    }

}
```

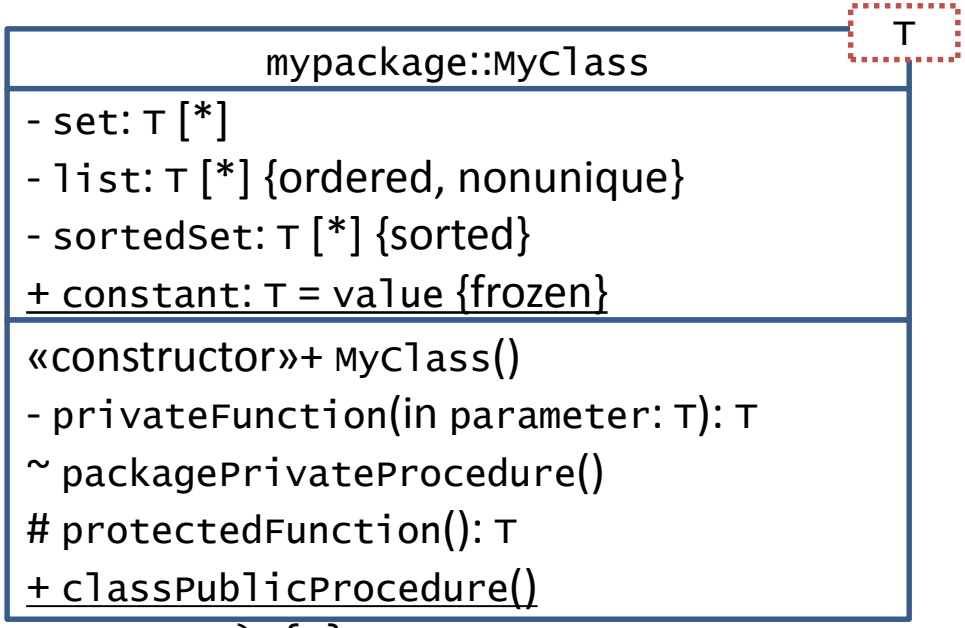
Classes: especialização

```
package mypackage;


public class MyDerivedClass extends MyClass {
     …
}
```



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() {…}
}
```



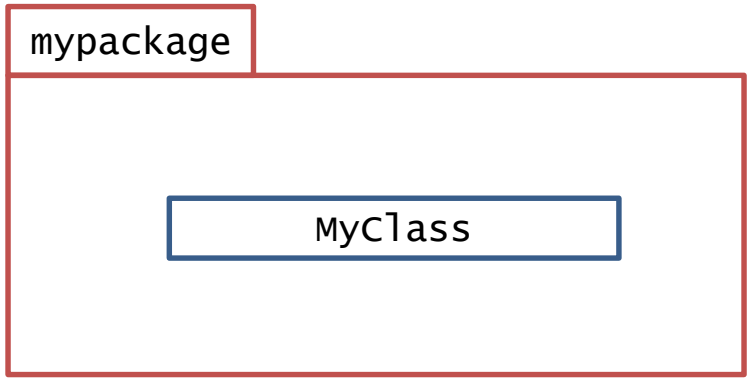
Pacotes

```
package mypackage;

…

public class MyClass {
    …
}
```

mypackage::MyClass

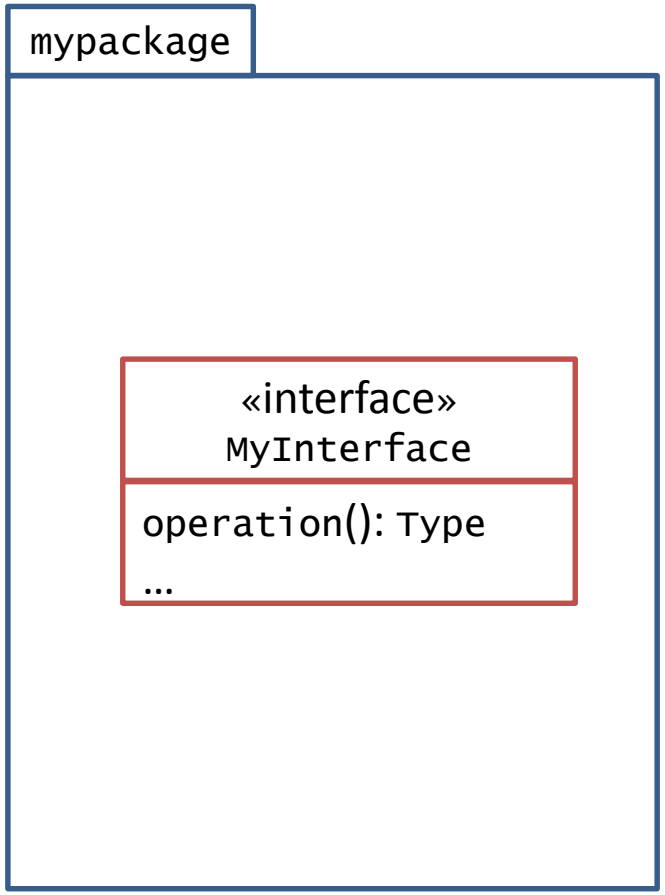
Pacotes

```
package mypackage;

…

public class MyClass {
    …
}
```



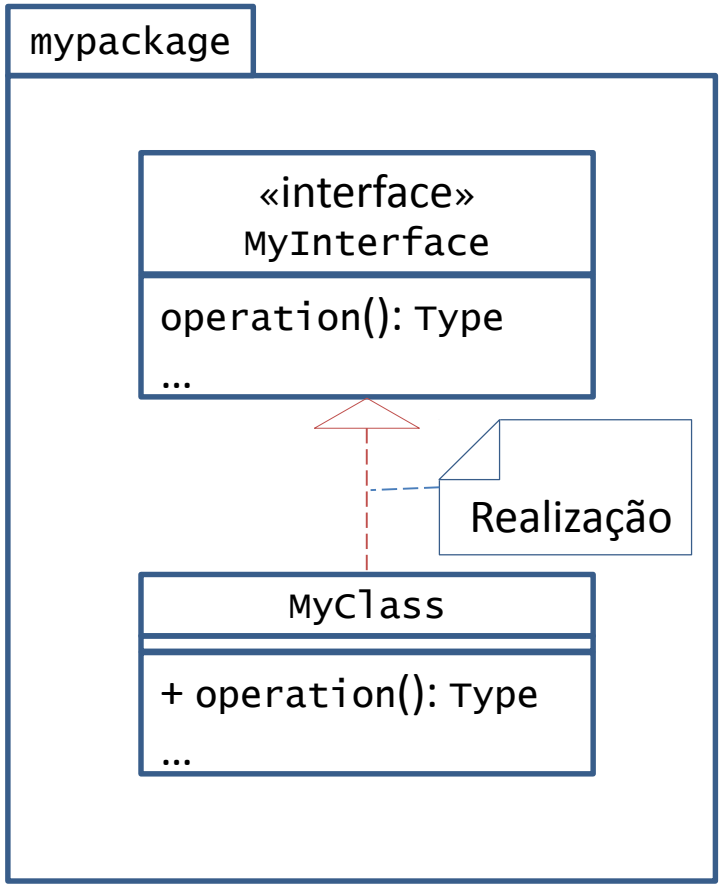
Interfaces

```
package mypackage;
…

public interface MyInterface {
    Type operation();
    …
}
```



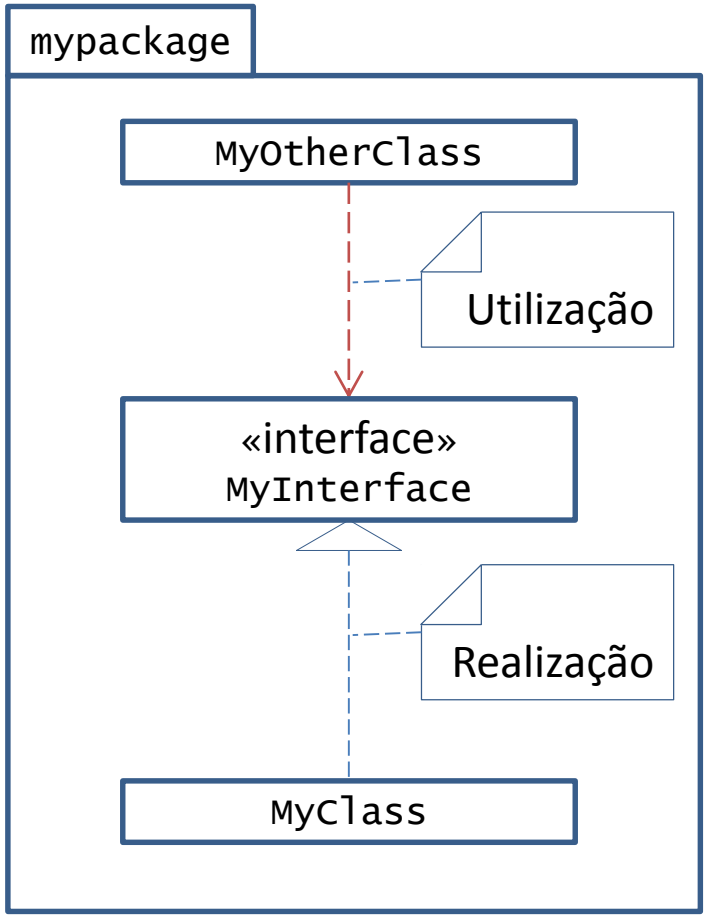
Interfaces

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

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

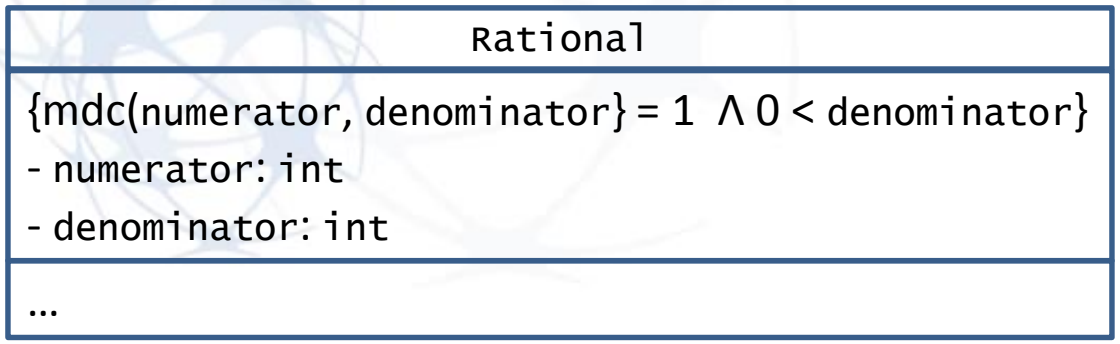


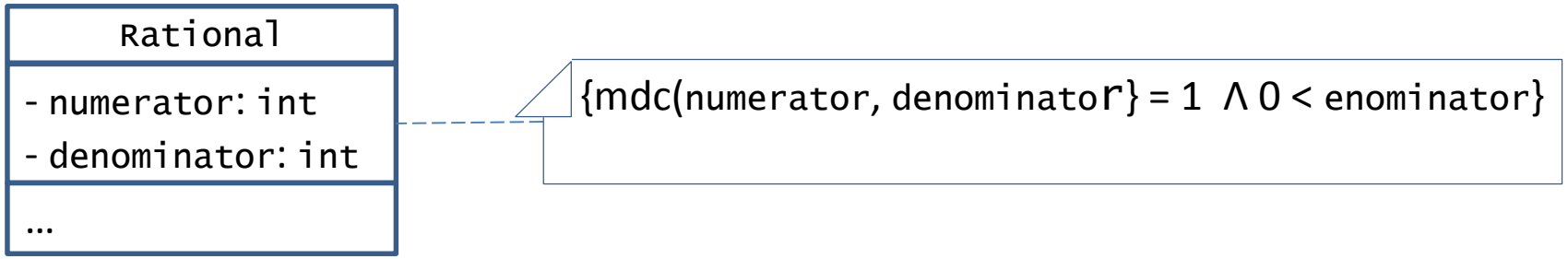
Interfaces

```
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

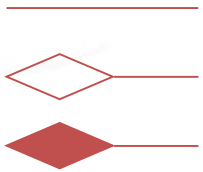

Rational
{mdc(numerator, denominator} = 1 ∧ 0 < denominator}
- numerator: int
- denominator: int
...


Rational
- numerator: int
- denominator: int
...
{mdc(numerator, denominator} = 1 ∧ 0 < enominator}

Principais relações entre classes

- Generalização

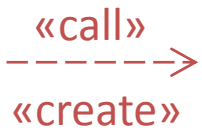
- Associação
  - Agregação
  - Composição



- Abstracção
  - Realização

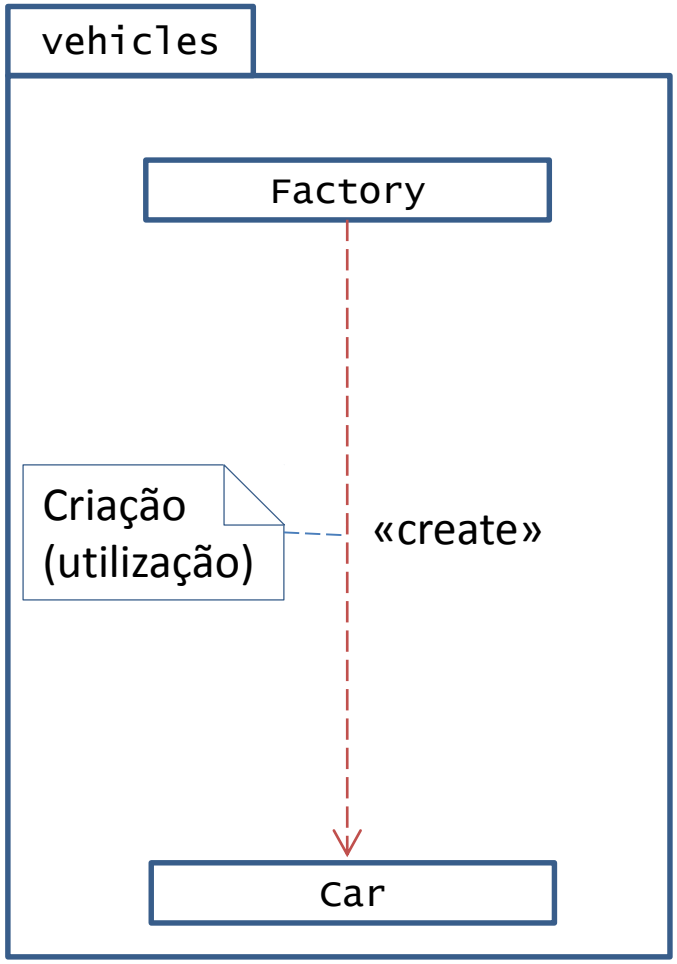
- Ligação (*binding*)

- Utilização
  - Invocação
  - Criação



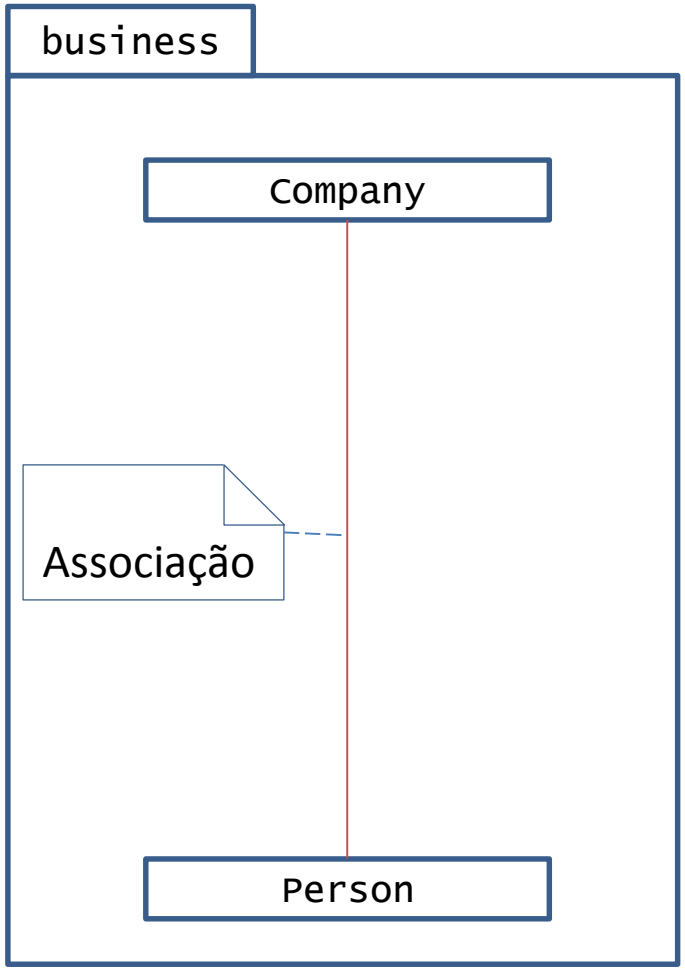
Utilização: criação

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

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



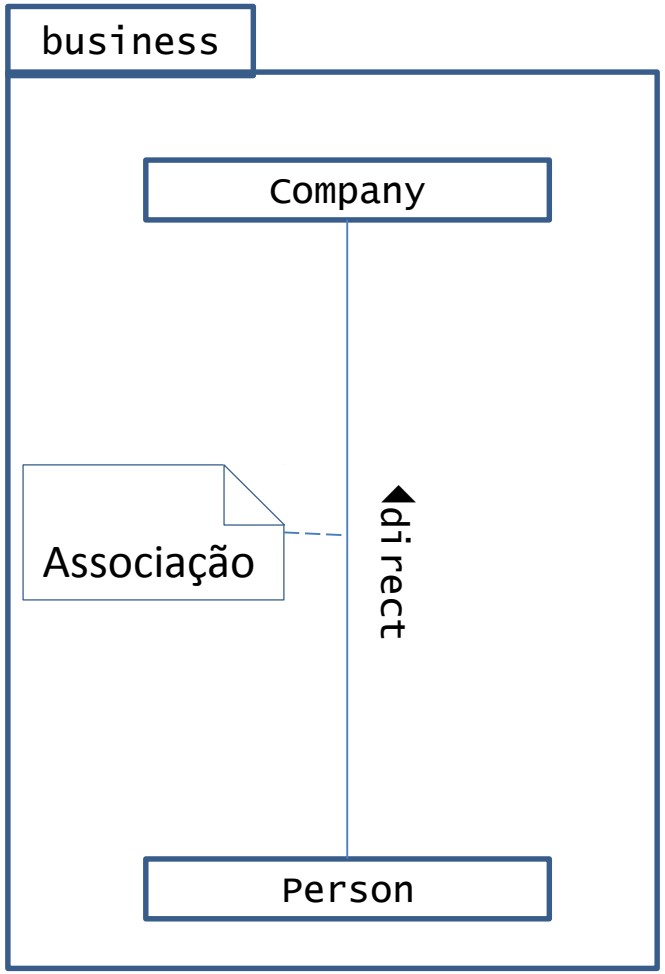
Associação

```
public class Company {
   Person director = …

}
```

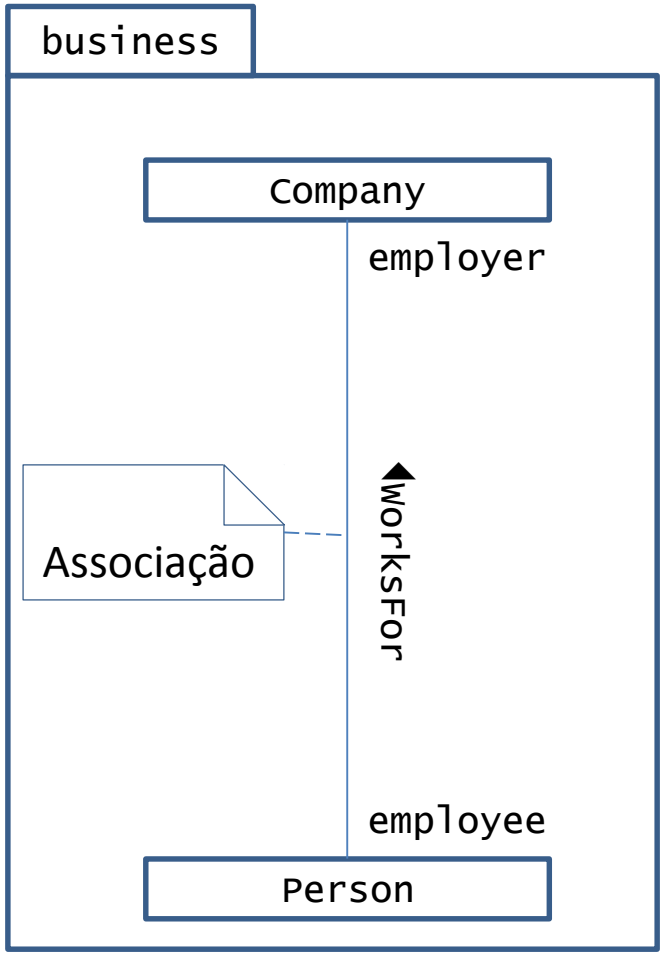


Associação: nome

```
public class Company {
   Person director = …

}
```

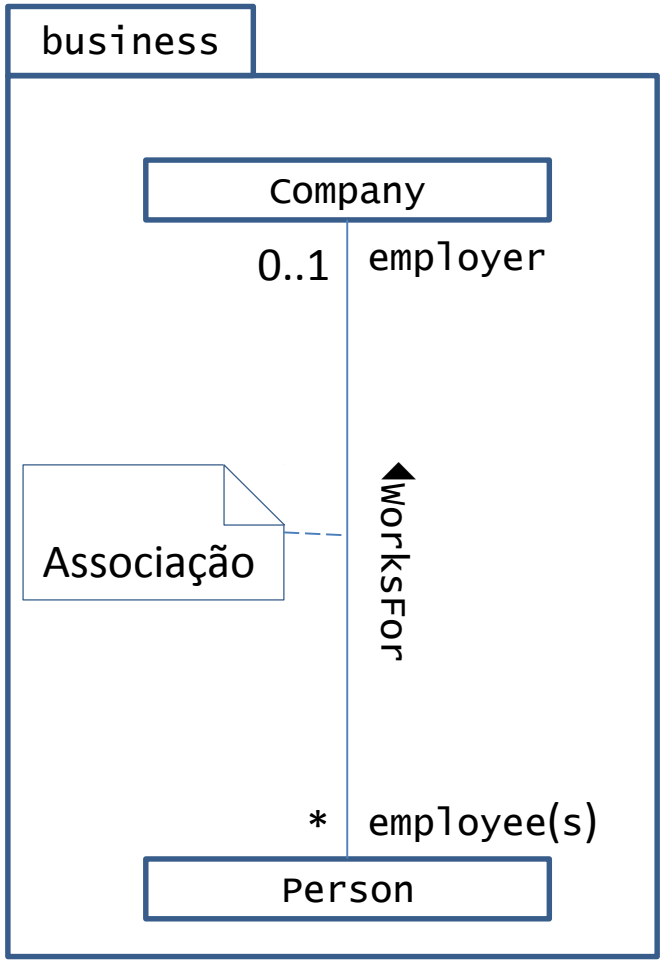


Associação: papeis



Associação: multiplicidade

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

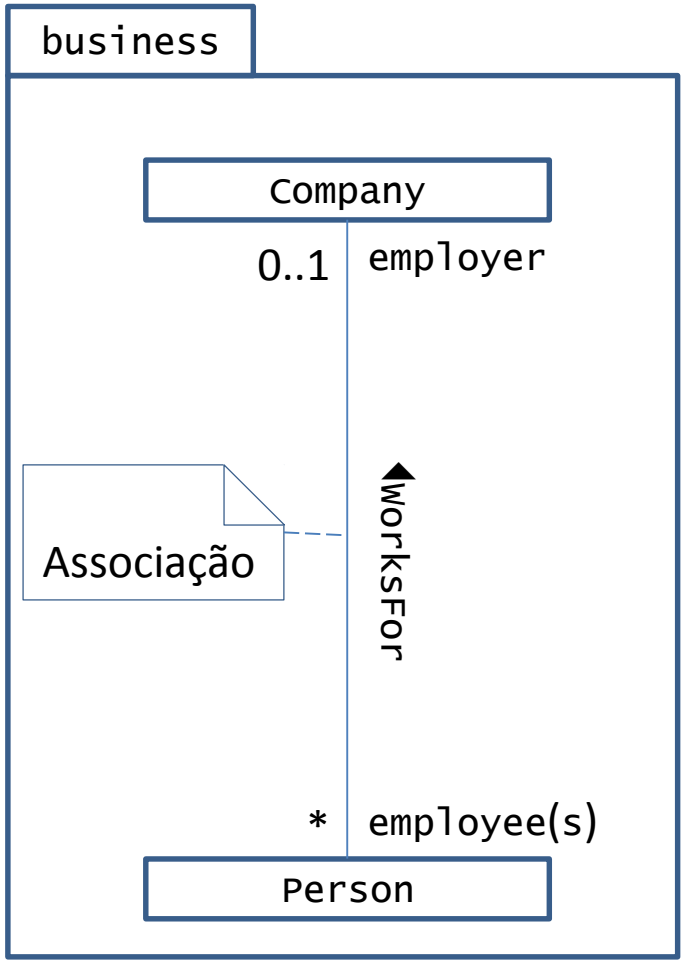


Multiplicidade

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

Associação: representação

```
package business;
public class Company {
    private
    Set<Person> workers;
    …
}
public class Person {
    private Company employer;
    …
}
```

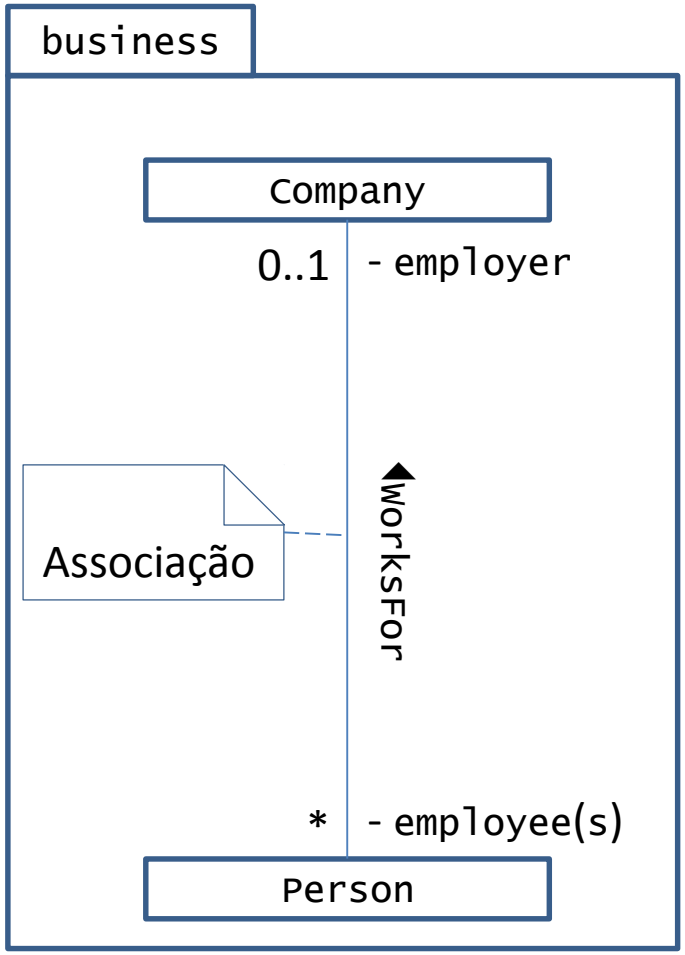


Representação e multiplicidade

Notação	Significado	Representação
0..1	Nenhum ou um. Opcional.	Atributo referência (possivelmente nulo).
1..1 1	Exactamente um. Obrigatório.	Atributo (se referência, *não* 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!

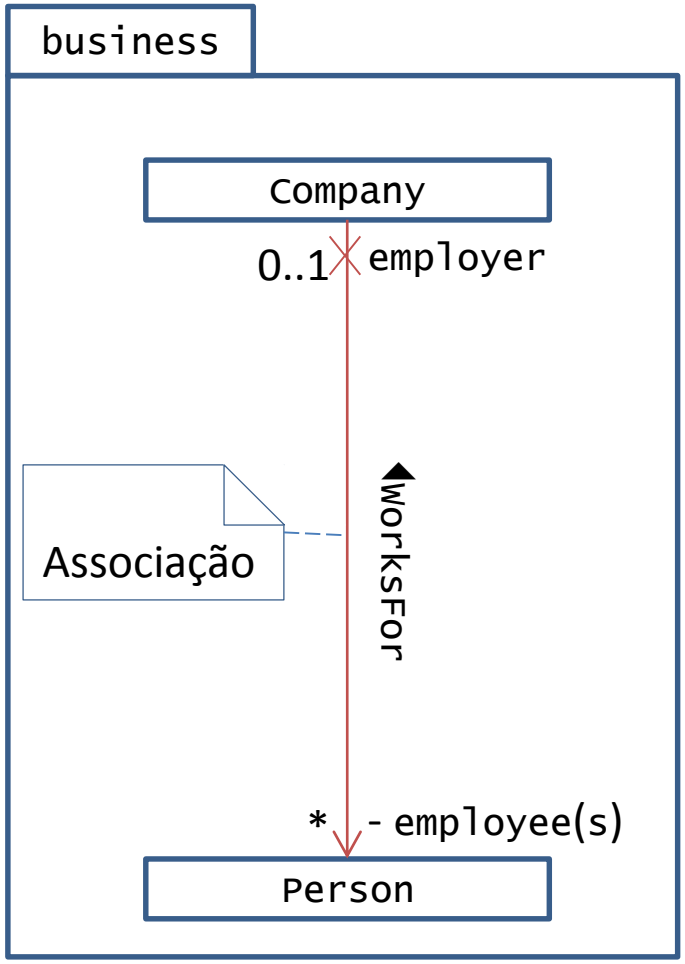
Associação: visibilidade

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



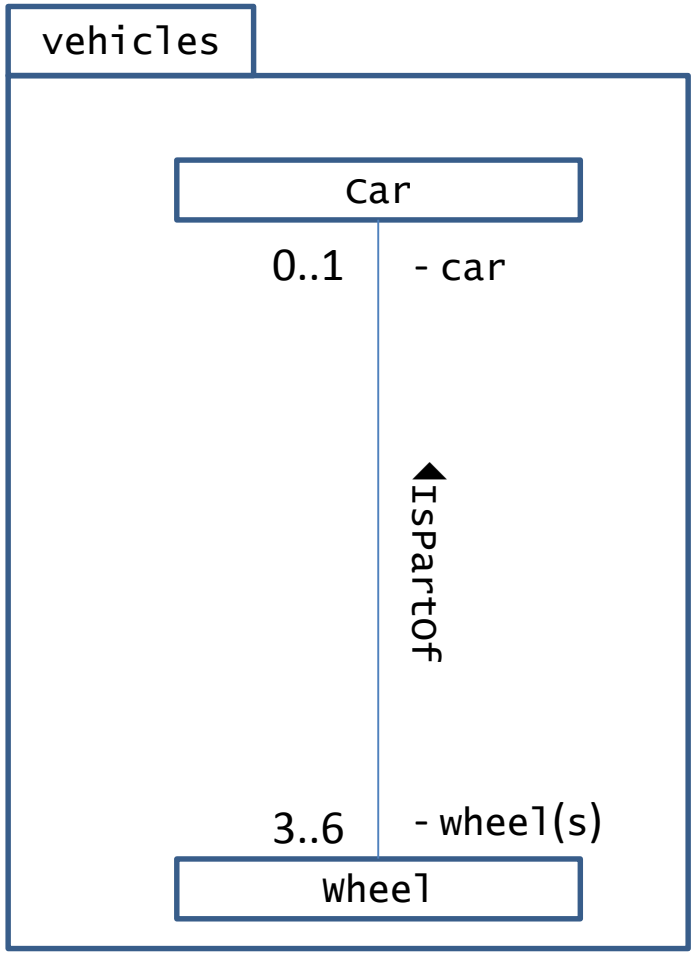
Associação: navegabilidade

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



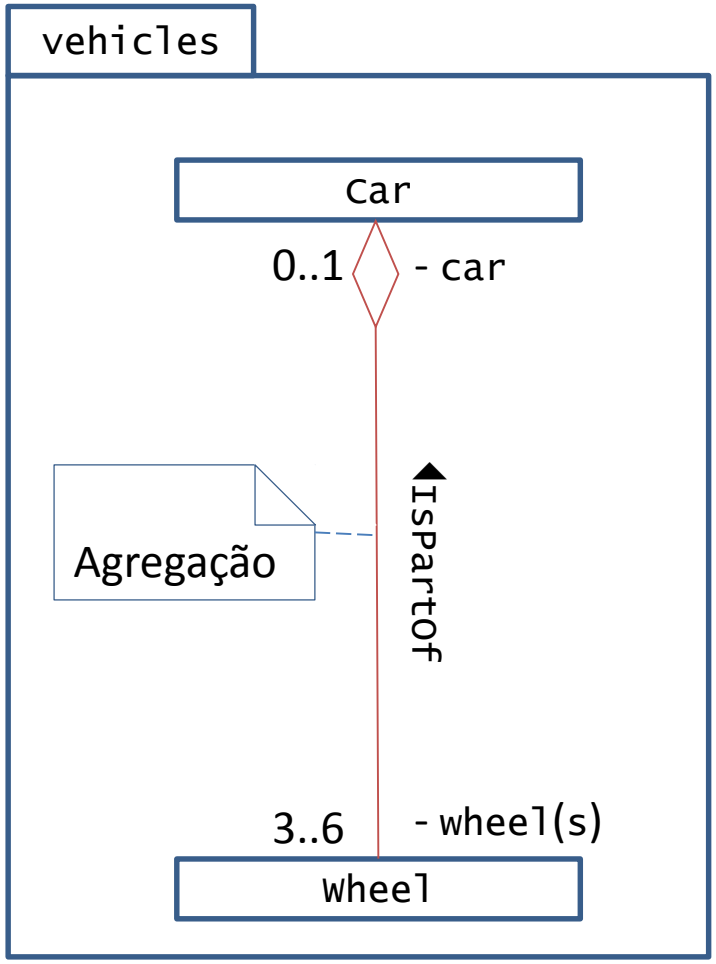
Associação

```
package vehicles;
public class Car {
    private Set<Wheel> wheels;
    …
}
public class Wheel {
    private Car car;
    …
}
```



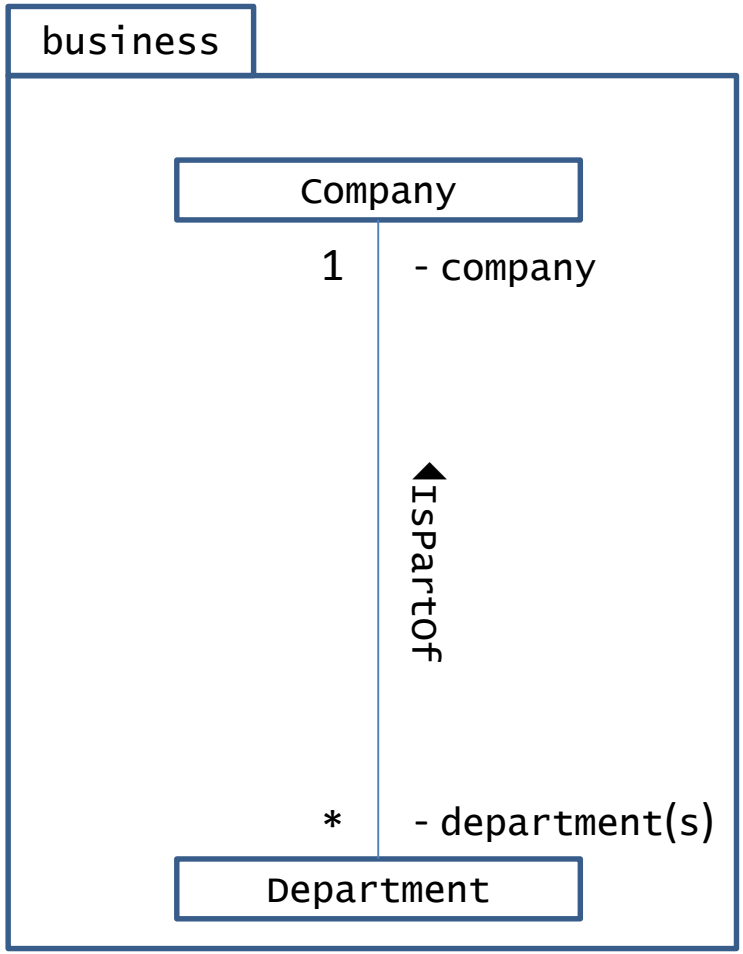
Associação: agregação

```
package vehicles;
public class Car {
    @Parts
    private Set<Wheel> wheels;
    …
}
public class Wheel {
    @Whole
    private Car car;
    …
}
```



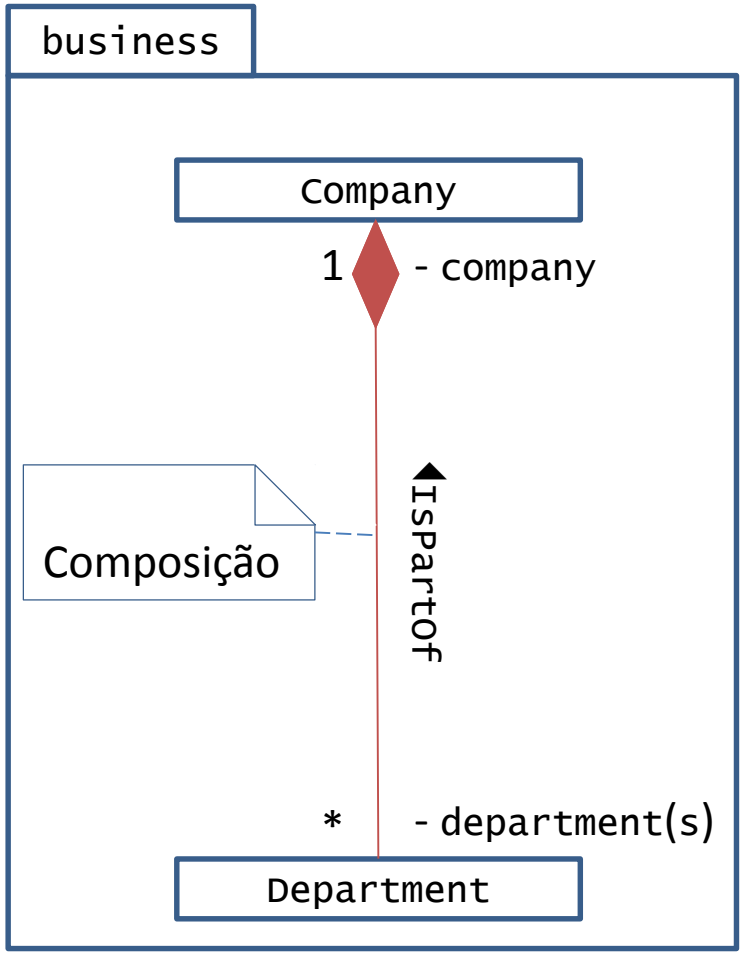
Associação

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



Associação: composição

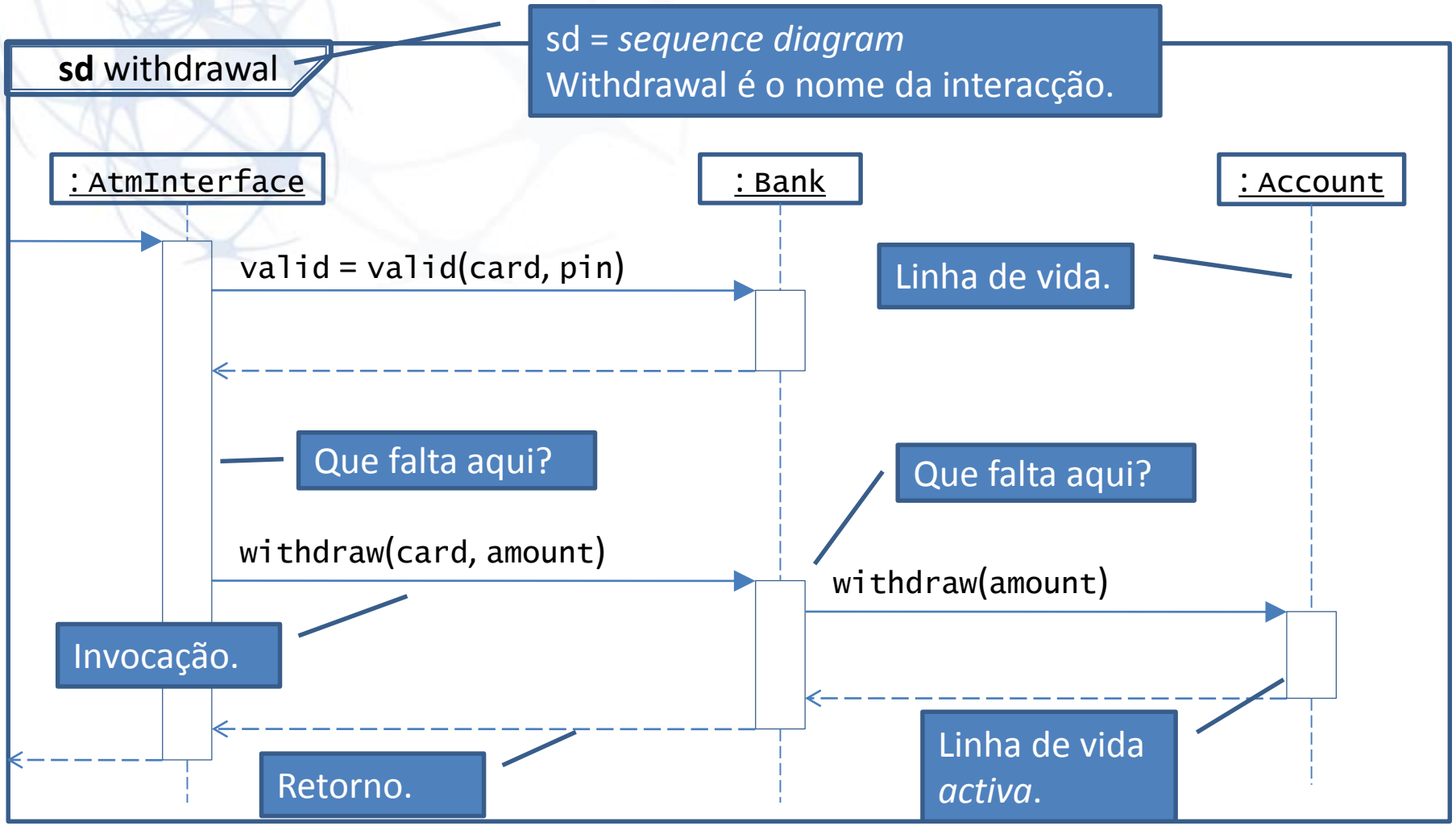
```
package business;
public class Company {
    @Components
    private Set<Department>
        departments;   …
}
public class Department {
    @Composite
    private Company company;
}
```



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)

Exemplo 1


sd withdrawal
sd = *sequence diagram*
Withdrawal é o nome da interacção.
: AtmInterface
: Bank
: Account
valid = valid(card, pin)
Linha de vida.
Que falta aqui?
Que falta aqui?
withdraw(card, amount)
withdraw(amount)
Invocação.
Retorno.
Linha de vida *activa*.

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