UML - Introduction

UML (Unified Modeling Language)

- Visual modeling language
  - Diagrams represent a model of the system
  - Important communication tool

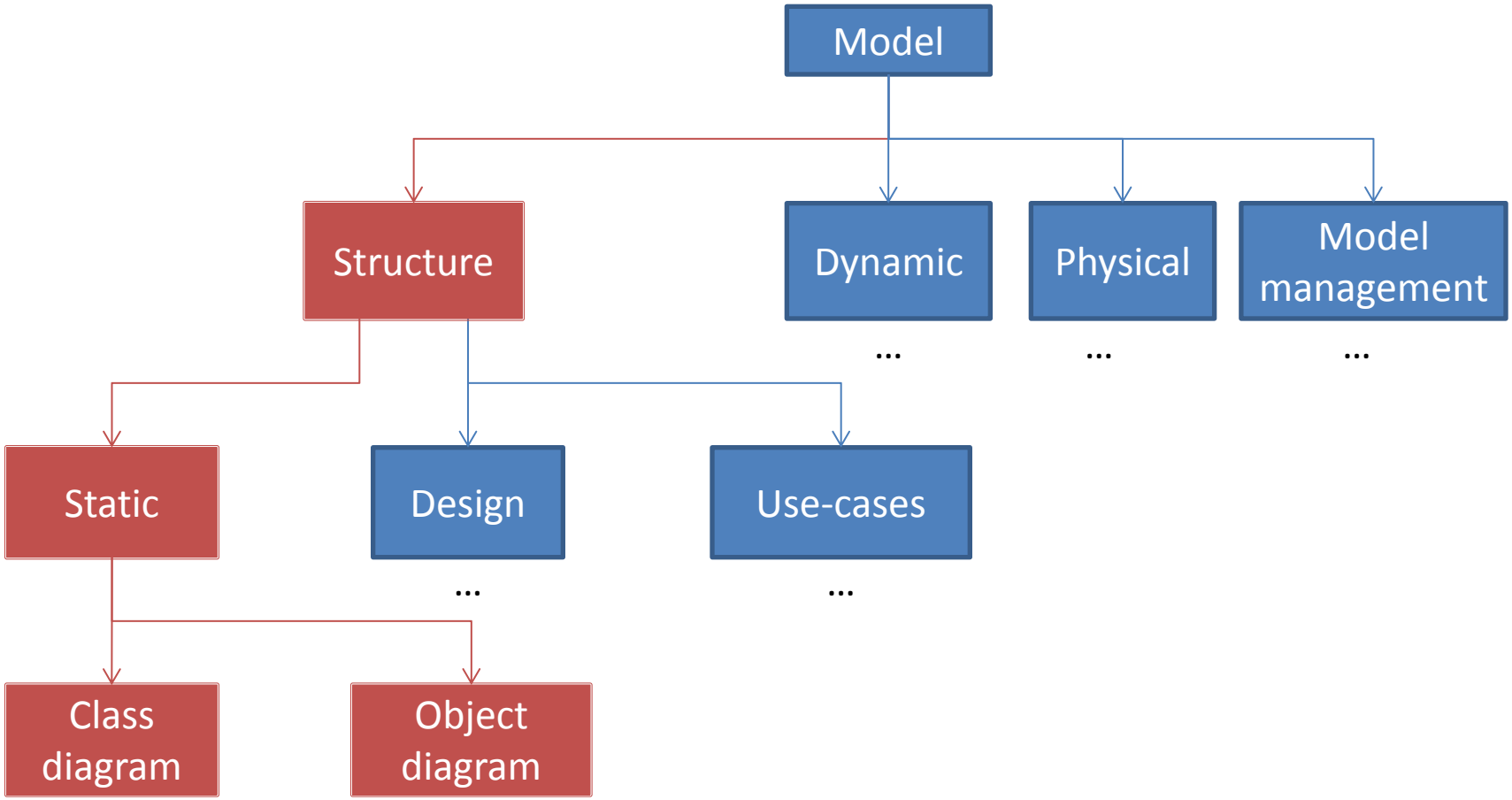
- Authors
  - Grady Booch
  - Ivar Jacobson
  - James Rumbaugh

- Normalization
  - OMG (Object Management Group)
  - http://www.uml.org/
  - Version 2.2

Diagrams

- Structure
  - Static (classes, objects)
  - Design (internal structure, colaboration, components)
  - Casos de uso
- Dynamic
  - State machines
  - Activity
  - Interaction (sequence, communication)
- Physical (implantation)
- Model management (packages)

Diagrams


Model
Structure
Dynamic
Physical
Model management
...
...
...
Static
Design
Use-cases
...
...
Class diagram
Object diagram

Class diagram

- Represents
  - Classes
  - Relations between classes
- Structural, static diagram
- Problem domain
  - Concepts
  - Analysis model
- Solution domain
  - Classes
  - Design model

Classes and their relations do not change during execution.

Model of the logical structure of the system. Static perspective: evolution of the systems during execution not explicit.

Understand the problem, analysis and requisite gathering, glossary.

Desgn, synthesis, implementation. Possible auto generation of code.

Object Diagram

- Represents
  - Objects
  - Connections between objects

Snapshot of the system execution state.

- Structural and static diagram

Helps understand data structure.
Still a static perspective.

Packages

```
package mypackage;

…

public class MyClass {
    …
}
```

```
mypackage::MyClass
```

Classes

Class represented by a box with compartments

Name

Attributes

```
package mypackage;

…

public abstract
class MyAbstractClass {

    private Set<Type> set;
    private List<Type> list;
    private TreeSet<Type>
        sortedSet;
    public static final Type constant = value;

    public MyAbstractClass() {…}
    private Type privateFunction(final Type parameter) {…}
    void packagePrivateProcedure() {…}
    protected abstract Type abstractProtectedFunction();
    public static void classPublicProcedure() {…}
}
```

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()

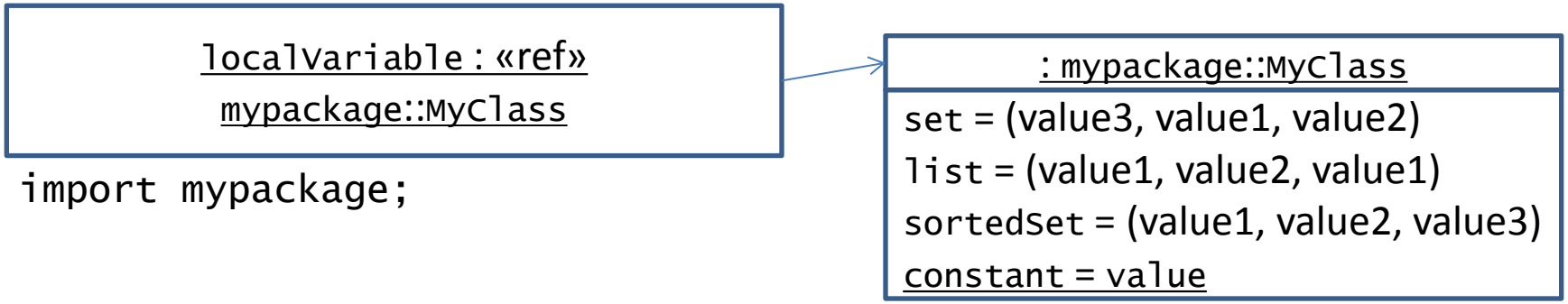
Operations

Objects

localVariable : mypackage::MyClass
set = (value3, value1, value2) list = (value1, value2, value1) sortedSet = (value1, value2, value3) constant = value

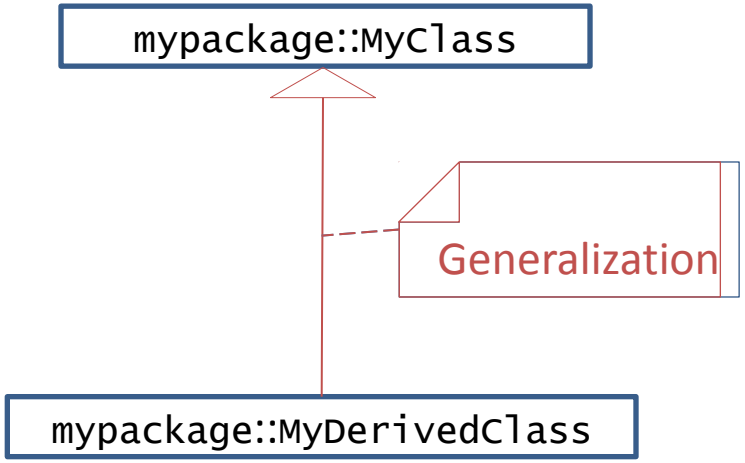
```
import mypackage;

…

public class MyClassTester {

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

}
```

Objects



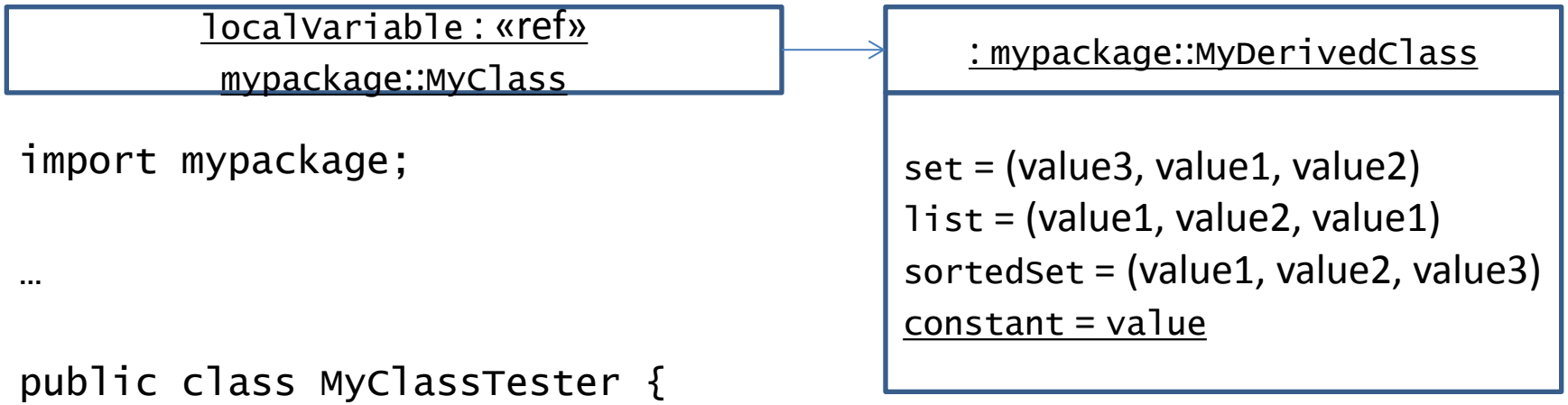

```
import mypackage;

…

public class MyClassTester {

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

}
```

Classes: specialization



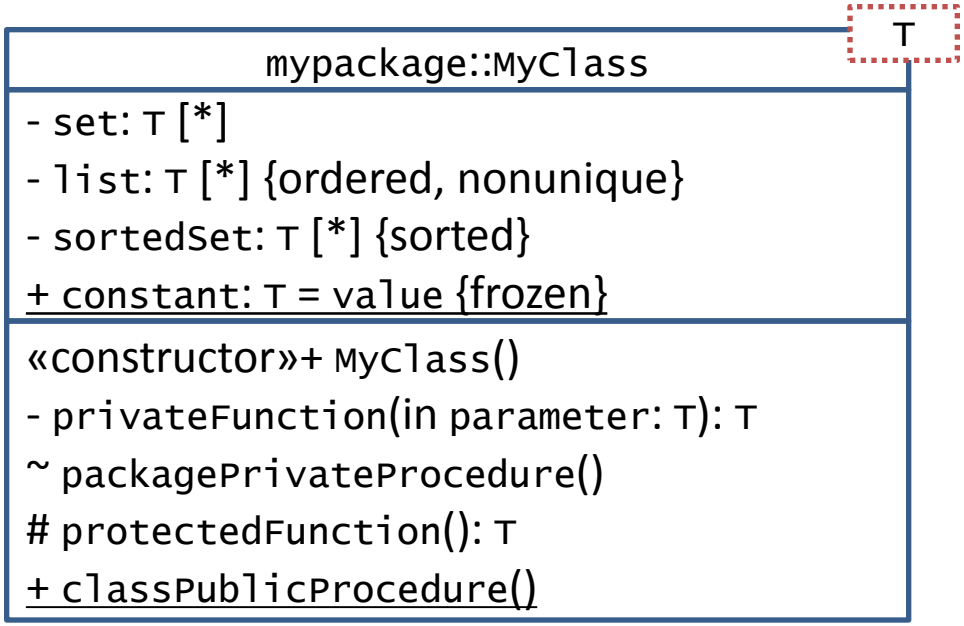

```
package mypackage;

…

public class MyDerivedClass extends MyClass {
    …
}
```

Objects: specialization



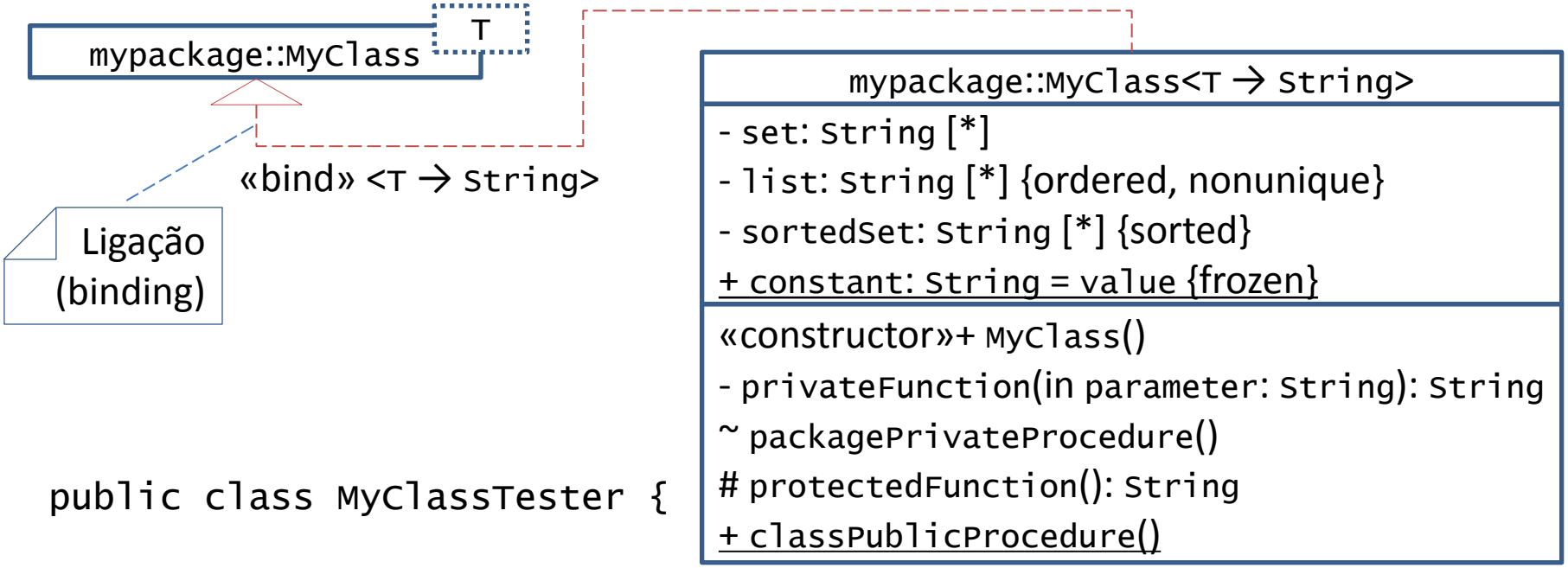

```
import mypackage;

…

public class MyClassTester {

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

}
```

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 = value;

    public MyClass() {…}
    private T privateFunction(final T parameter) {…}
    void packagePrivateProcedure() {…}
    protected T protectedFunction() {…}
    public static void classPublicProcedure() {…}
}
```

Generic classes (*binding*)




```
public class MyClassTester {

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

}
```

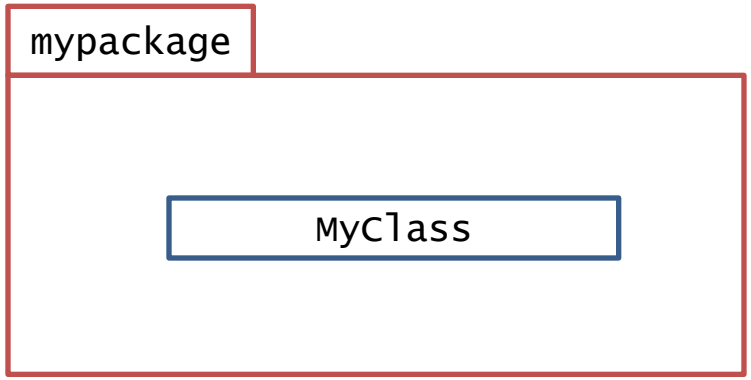
Objects

: mypackage::MyClass<T → String>
set = (“string3”, “string1”, “string2”) list = (“string1”, “string2”, “string1”) sortedSet = (“string1”, “string2”, “string3”) constant = “string”

```
import mypackage;

…

public class MyClassTester {

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

}
```

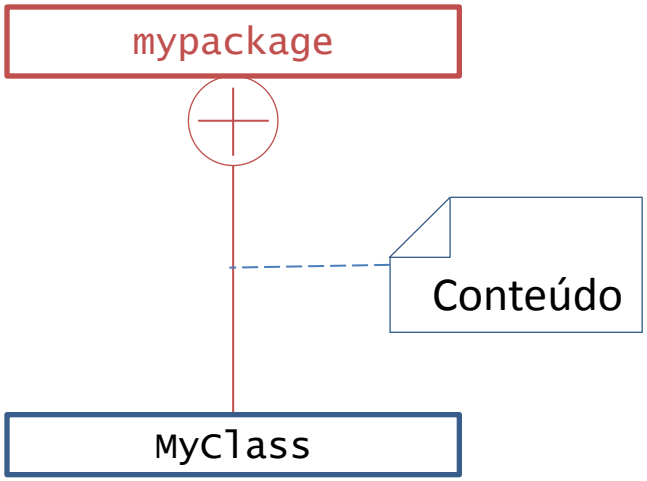
Packages

```
package mypackage;

…

public class MyClass {
    …
}
```



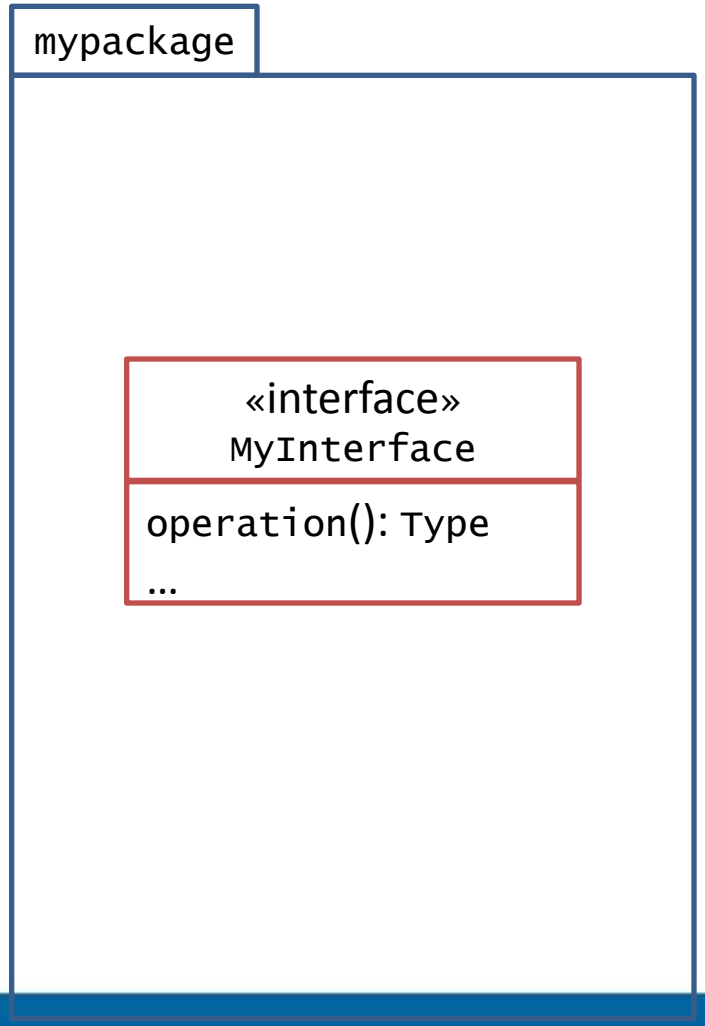
Packages

```
package mypackage;

…

public class MyClass {
    …
}
```



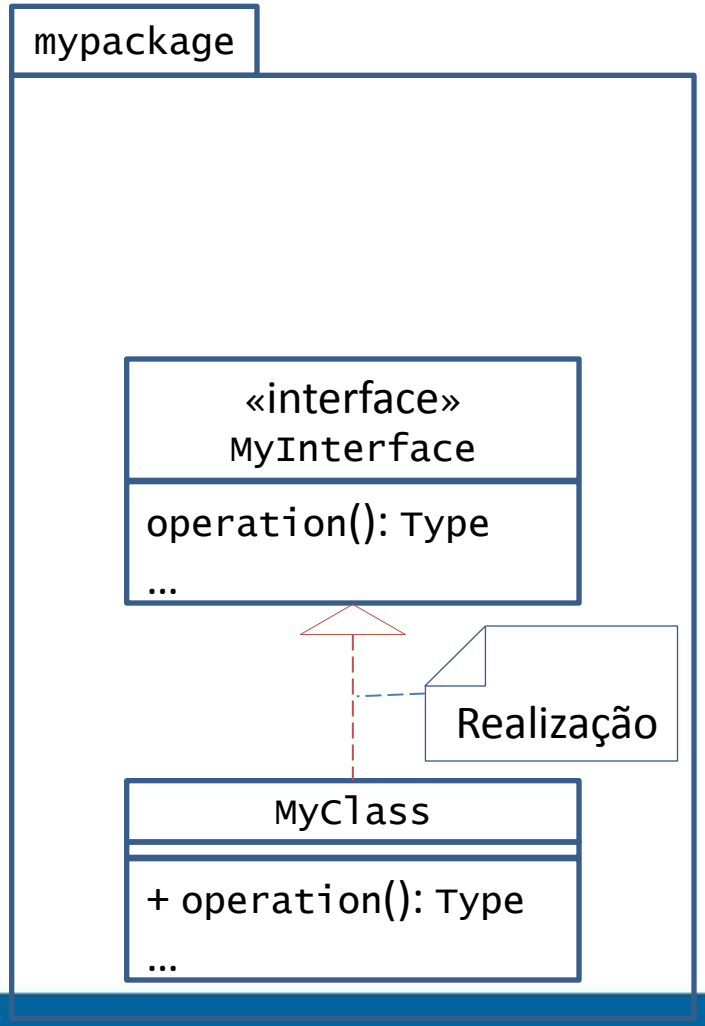
Interfaces

```
package mypackage;
…

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



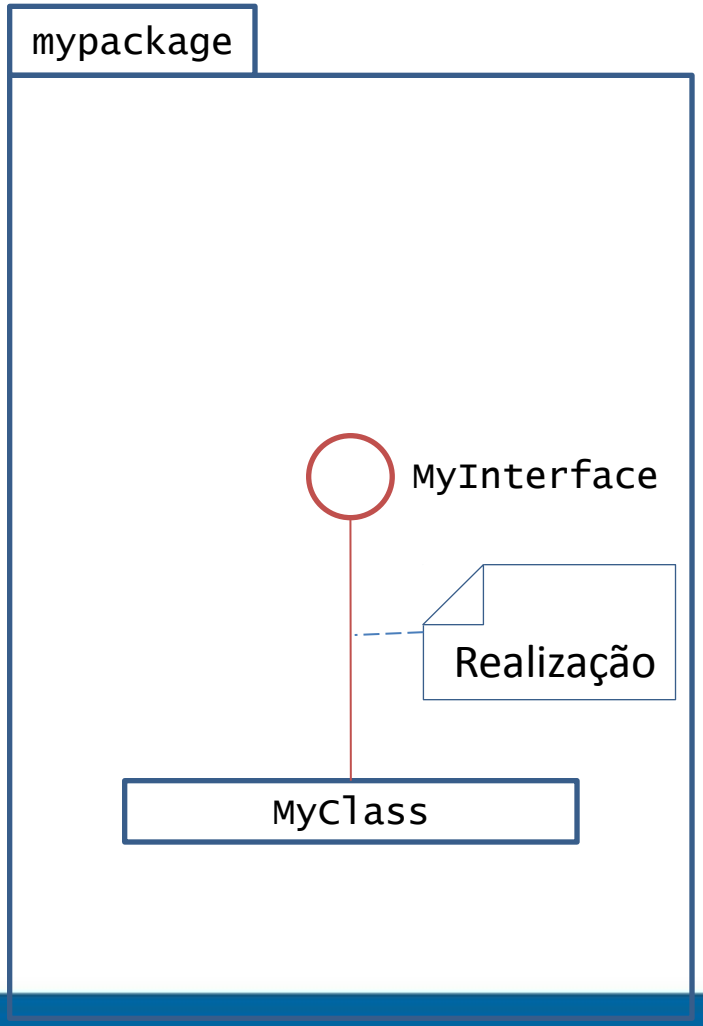
Interfaces

```
package mypackage;
…

public interface MyInterface {
    Type operation();
    …
}

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



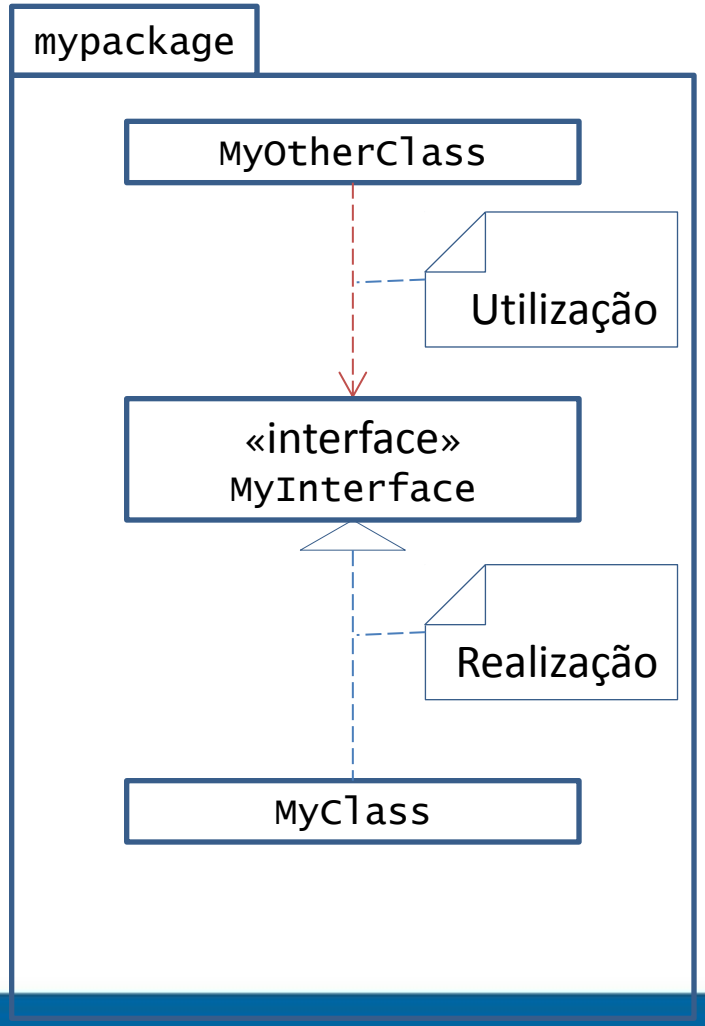
Interfaces

```
package mypackage;
…

public interface MyInterface {

    …
}

public
class MyClass implements MyInterface {

    …
}
```

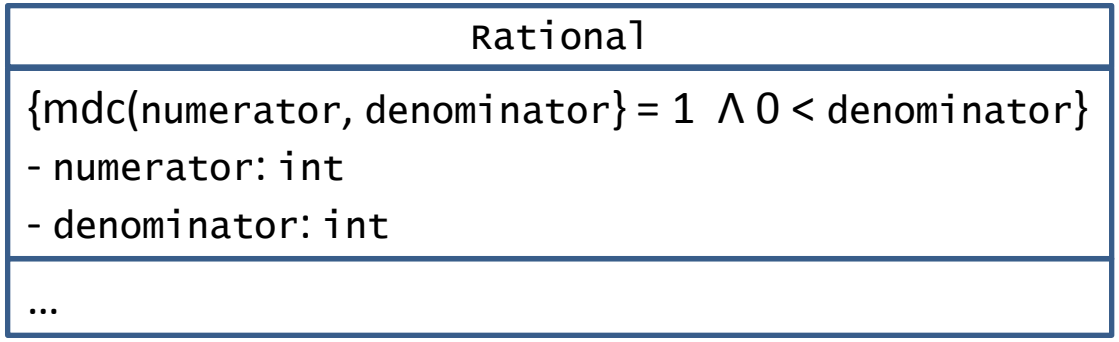


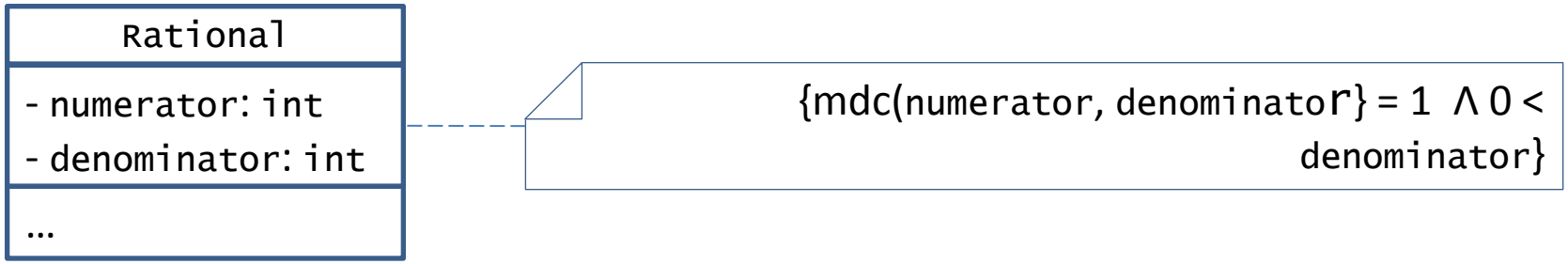
Interfaces

```
package mypackage;
…

public interface MyInterface {

    …
}

public
class MyClass implements MyInterface {

    …
}

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



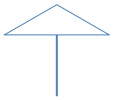
Restrictions



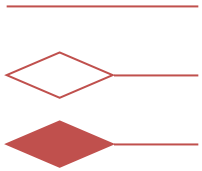


Main relations between classes

- Generalization

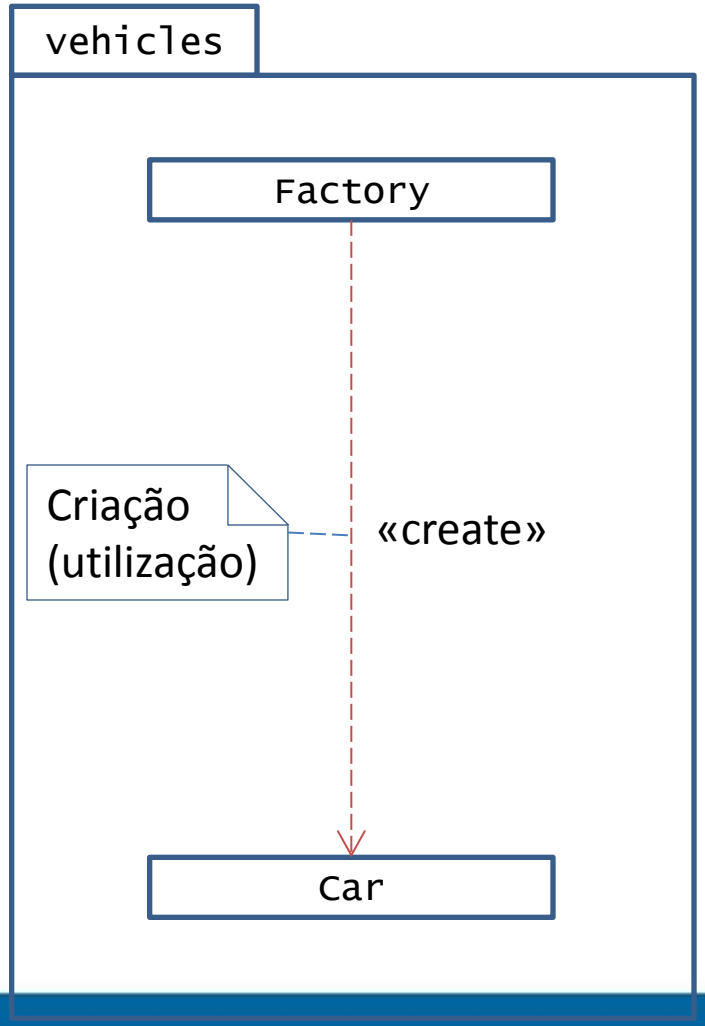


- Association
  - Agregation
  - Composition

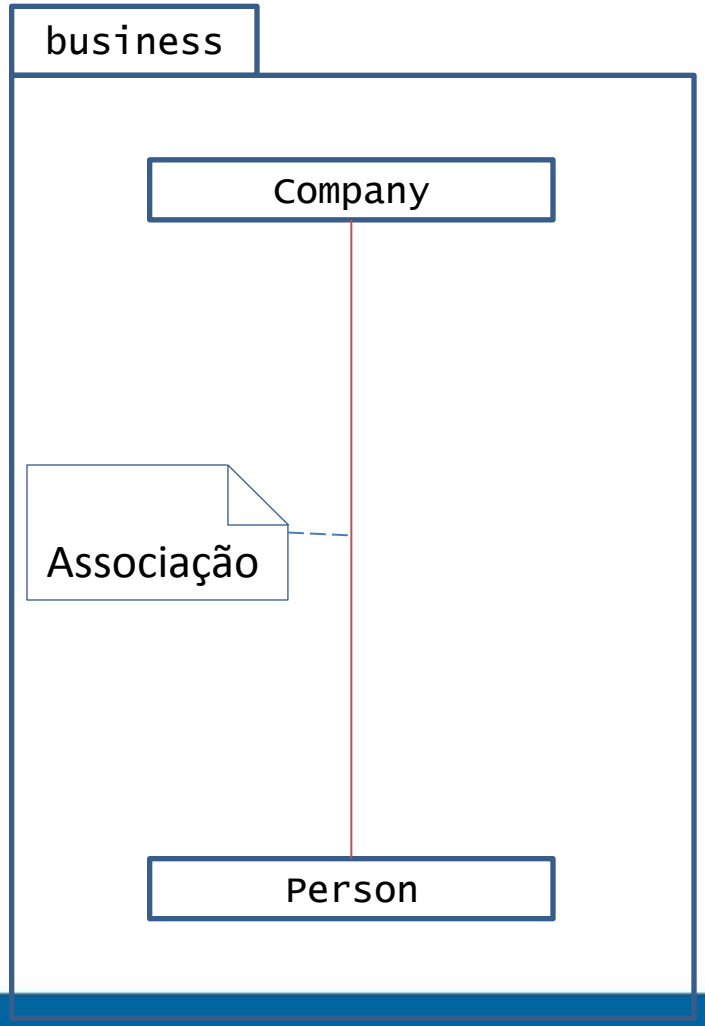


Utilização: criação

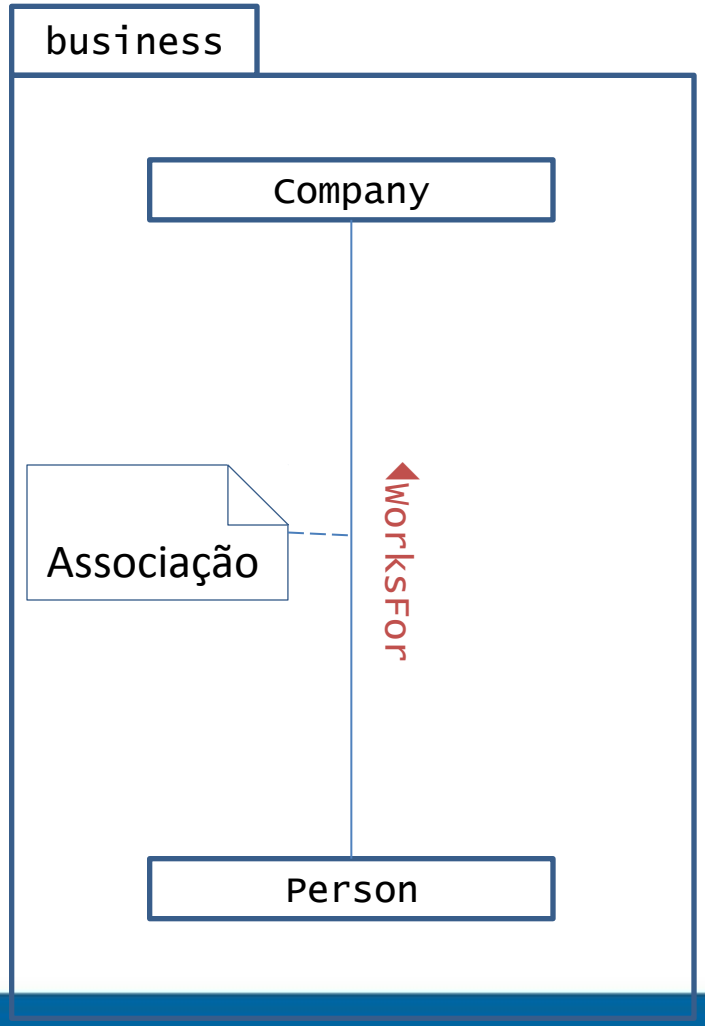
```
package vehicles;

public class Car {
    …
}

package vehicles;

public class Factory {
    …

    public Car newCar(…) {
        …
        return new Car(…);
    }

    …
}
```



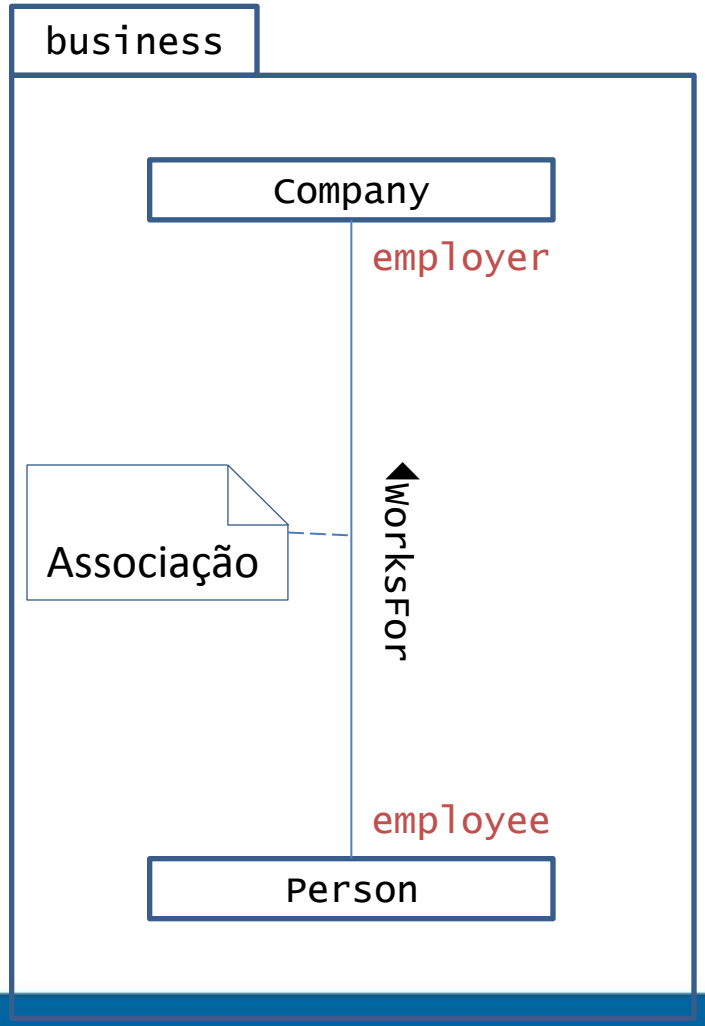
Association



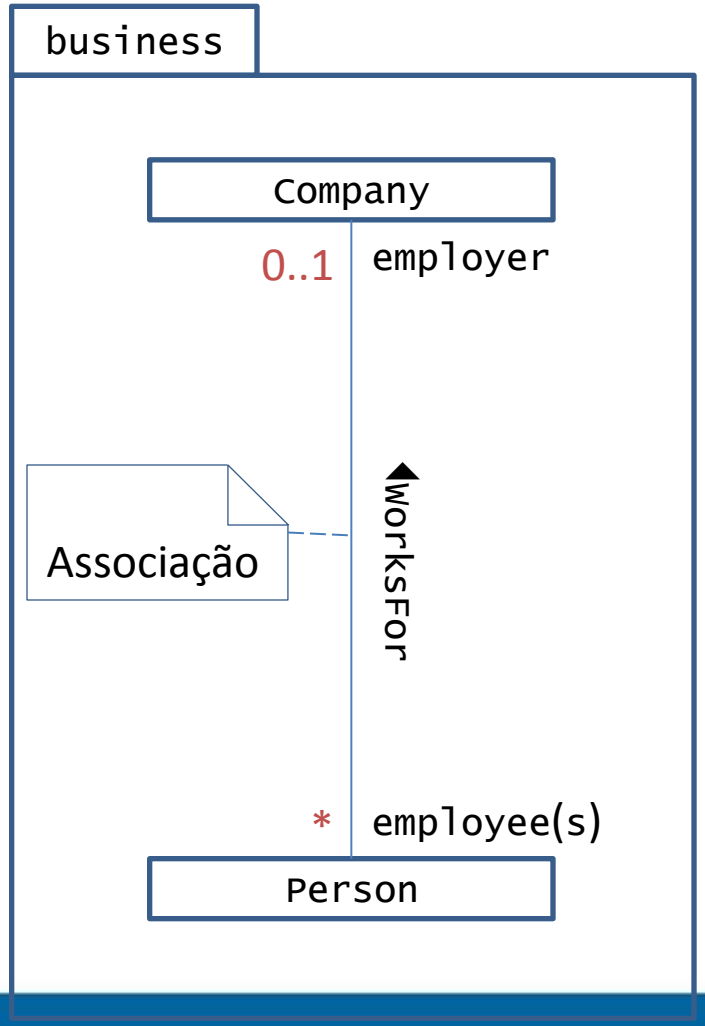
Association: name



Association: roles



Association: multiplicity

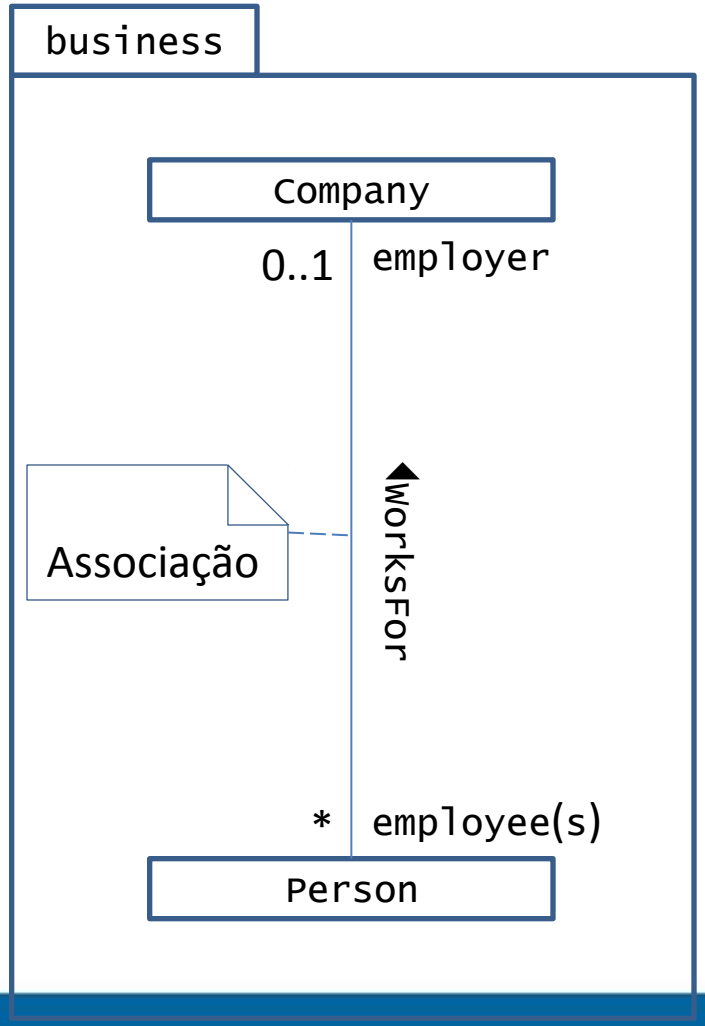


Multiplicity

Notation	Meaning
0..1	None or one. Optional.
1..1 1	Exactly one. Mandatory.
0..*n*	Zero to *n*.
0..* *	Arbitrary. Any.
n..*n* *n*	Exactly *n*.
1..*	At least 1.

Association: representation

```
package business;

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

public class Person {
    private Company employer;
    …
}
```

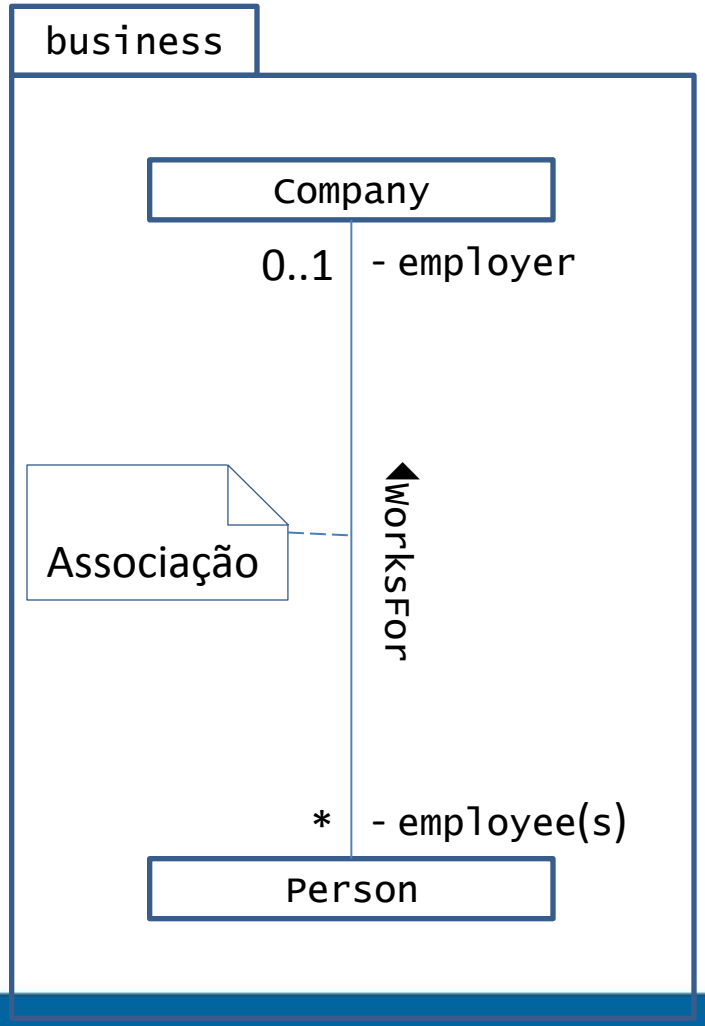


Representation and multiplicity

Notation	Meaning	Representation
0..1	None or one. Optional.	Reference attribute (possibly null).
1..1 1	Exactly one. Mandatory.	Attribute (if reference , not null).
0..*n*	Zero to *n*.	Collection (not null) of elements (not null).
0..* *	Arbitrary. Any.	Collection (not null) of elements (not null).
n..*n* *n*	Exactly *n*.	Matrix (not null) with *n* elements (not null).
1..*	At least 1.	Collection (not null) of elements (not null).

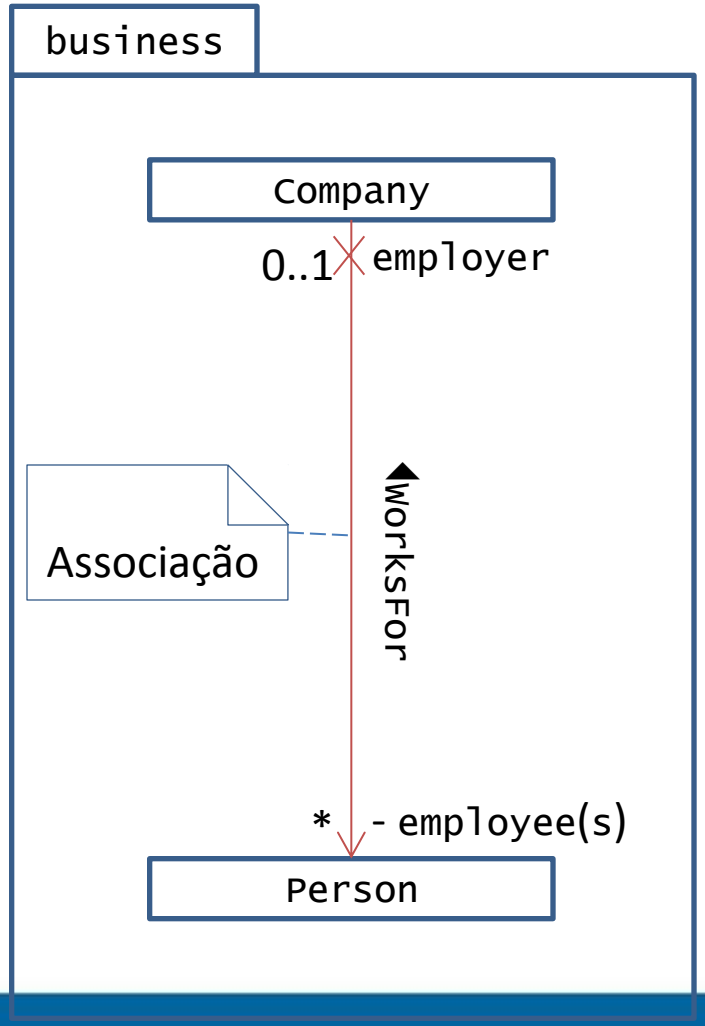
Association: visibility

```
package business;

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

public class Person {
    private Company employer;
    …
}
```



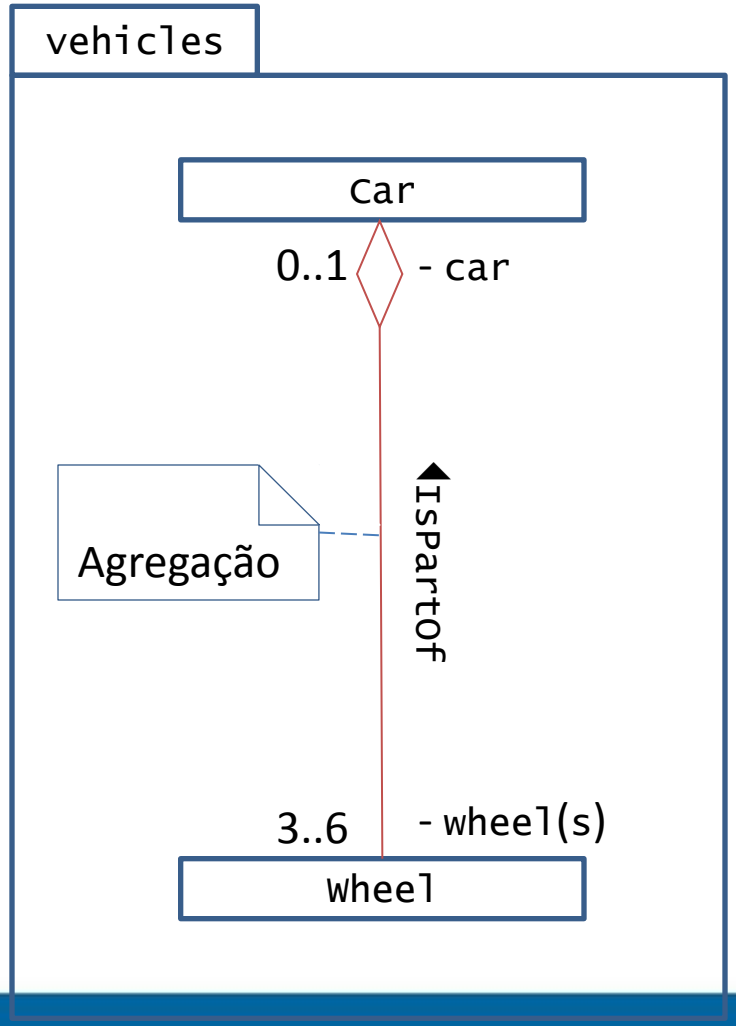
Association: navegation

```
package business;

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

public class Person {
    private Company employer;
    …
}
```



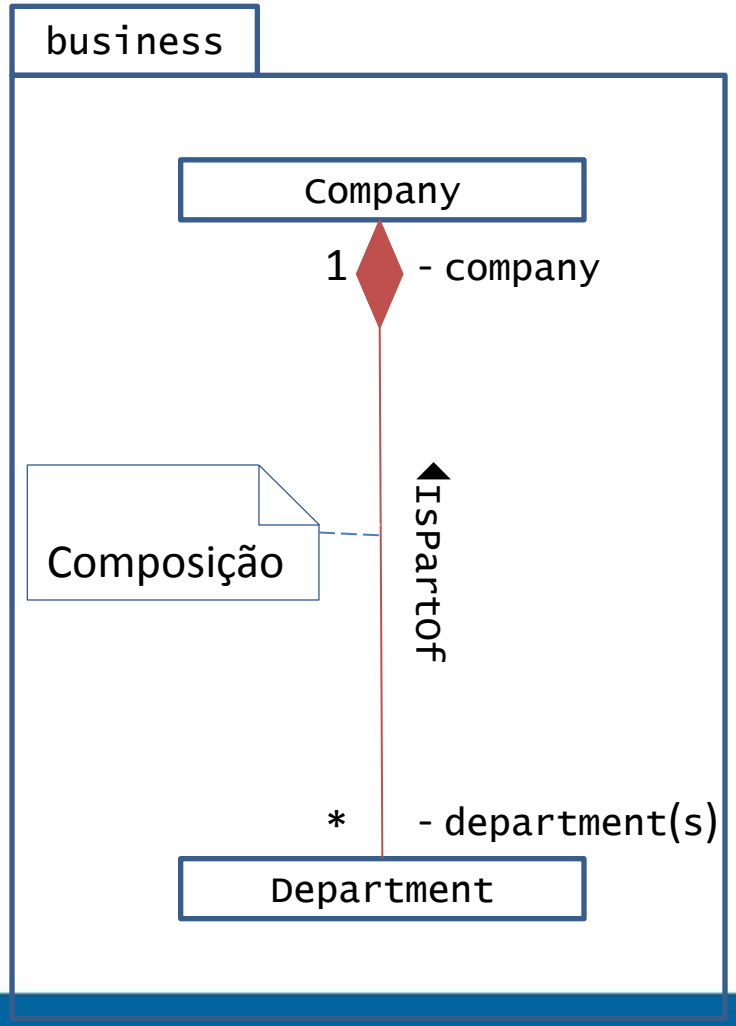
Association: agregation

```
package vehicles;

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



Association: composition

```
package business;

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



References

- 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)

Summary

- Introdction to UML