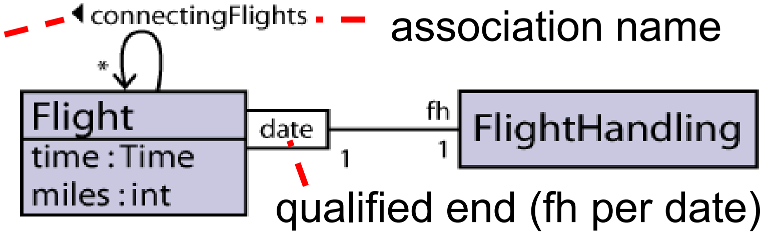
Associations

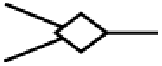
- **Associations** describe sets of tuples whose values refer to typed instances.
  - In particular, structural relationship between classes
  - Instances of associations are called links.



reading direction

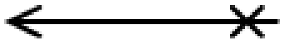
association name

qualified end (fh per date)



ternary association

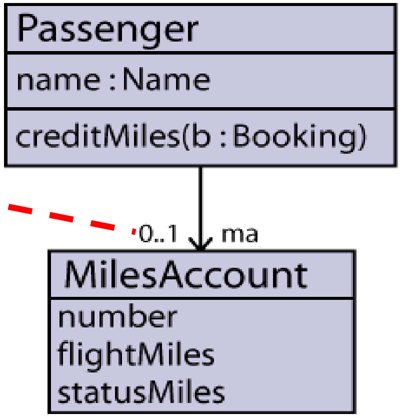
- **Association ends** are properties.
  - correspond to properties of the opposite class (default multiplicity is 0..1)
- Association ends may be navigable.
  - in contrast to general properties



navigable

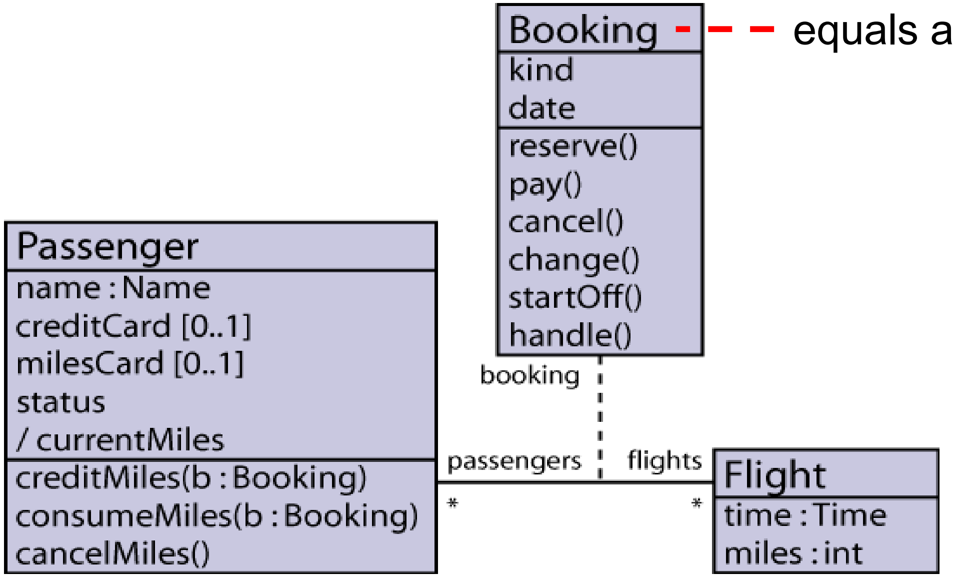
not navigable

association end



Association classes

- **Association classes** combine classes with associations.
  - not only connect a set of classifiers but also define a set of features that belong to the relationship itself and not to any of the classifiers



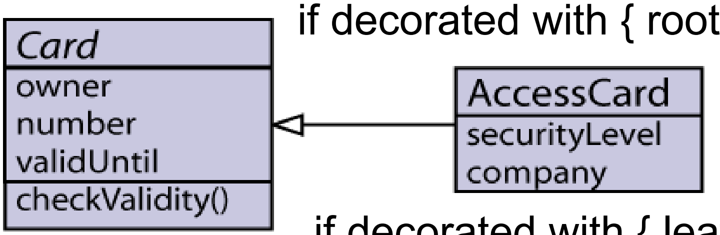
equals association name

- each instance of Booking has one passenger and one flight
- each link of Booking is one instance of Booking

Inheritance (1)

- **Generalizations** relate specific classes to more general classes.
  - instances of specific class also instances of the general class
  - features of general class also implicitly specified for specific class

{ abstract } class (no direct instances, only specializations may have instances)

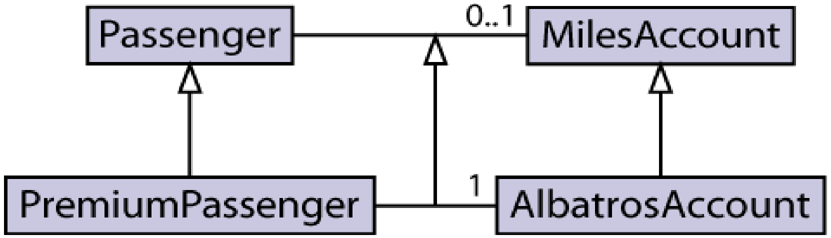


if decorated with { root }: no superclass

if decorated with { leaf }: no subclass

  - implies substitutability (in the sense of Liskov & Wing)
    - must be specified on specific class separately by { substitutable }

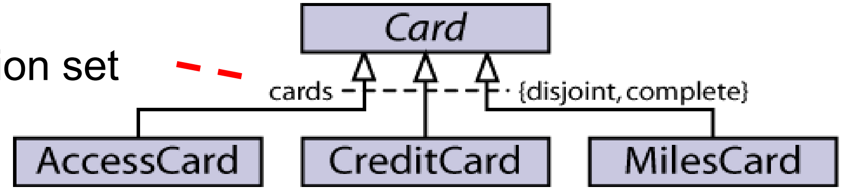
- Generalizations also apply to associations.
  - as both are Classifiers



Inheritance (2)

- **Generalization sets** detail the relation between a general and more specific classifiers.
  - { complete } (opposite: { incomplete })
    - all instances of general classifier are instances of one of the specific classifiers in the generalization set
  - { disjoint } (opposite: { overlapping })
    - no instance of general classifier belongs to more than one specific classifier in the generalization set
  - default: { disjoint, incomplete }

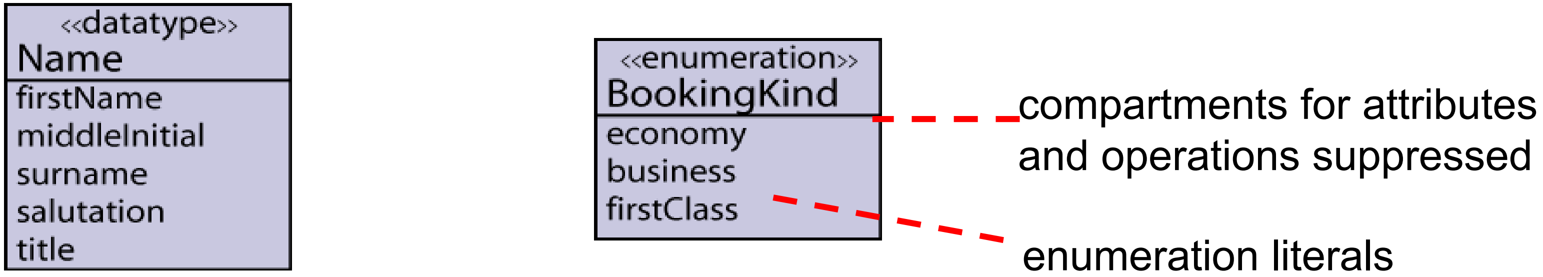
name of generalization set



  - several generalization sets may be applied to a classifier
    - useful for taxonomies

Data types and enumerations

- **Data types** are types whose instances are identified by their value.
  - Instances of classes have an identity.
  - may show structural and behavioural features



«datatype»
Name
firstName
middleInitial
surname
salutation
title

«enumeration»
BookingKind
economy
business
firstClass

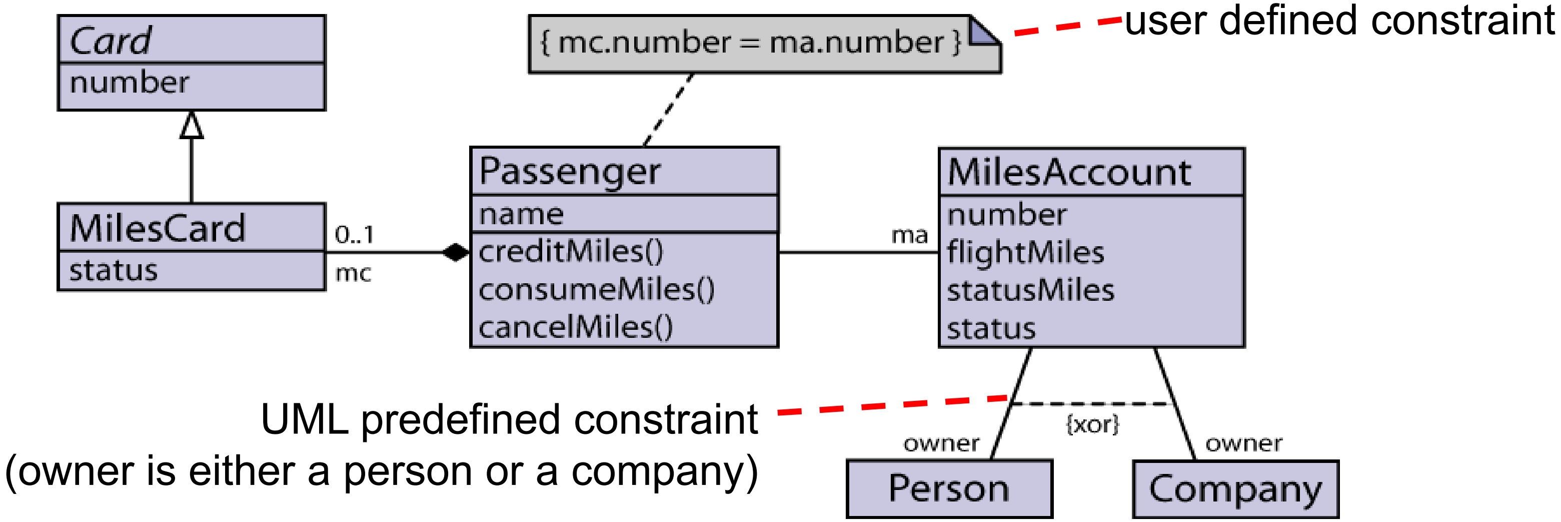
compartments for attributes and operations suppressed

enumeration literals

- **Enumerations** are special data types.
  - instances defined by enumeration literals
    - denoted by *Enumeration*::*EnumerationLiteral* or *#EnumerationLiteral*
  - may show structural and behavioural features

Constraints

- **Constraints** restrict the semantics of model elements.
    - constraints may apply to one or more elements
    - no prescribed language
        - OCL is used in the UML 2 specification
        - also natural language may be used

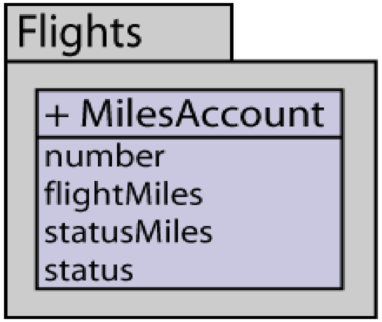


user defined constraint

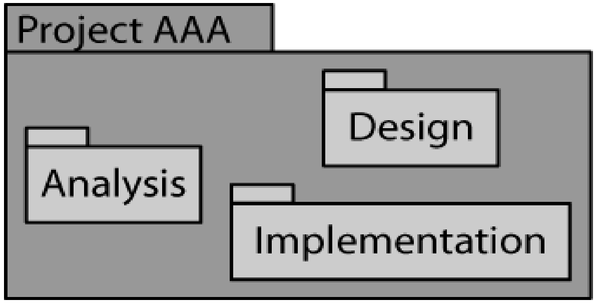
UML predefined constraint
(owner is either a person or a company)

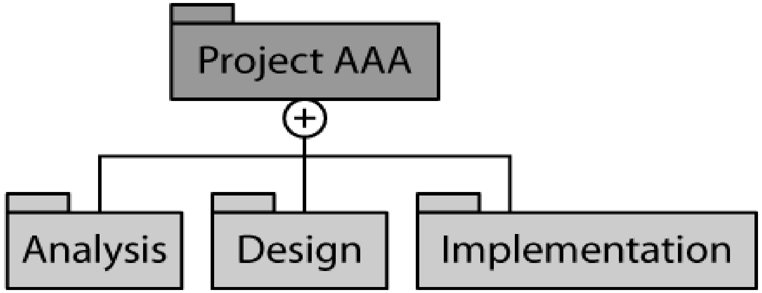
Packages (1)

- **Packages** group elements.
  - Packages provide a **namespace** for its grouped elements.
  - Elements in a package may be
    - public (+, visible from outside; default)
    - private (-, not visible from outside)
  - Access to public elements by qualified names
    - e.g., Flights::MilesAccount



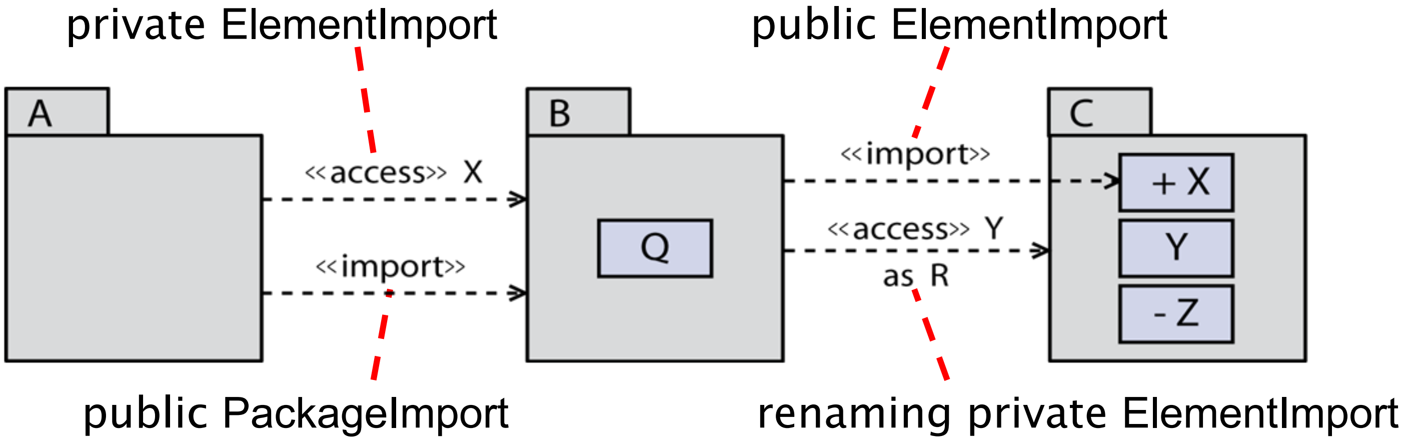
Notational variants





Packages (2)

- **Package imports** simplify qualified names.

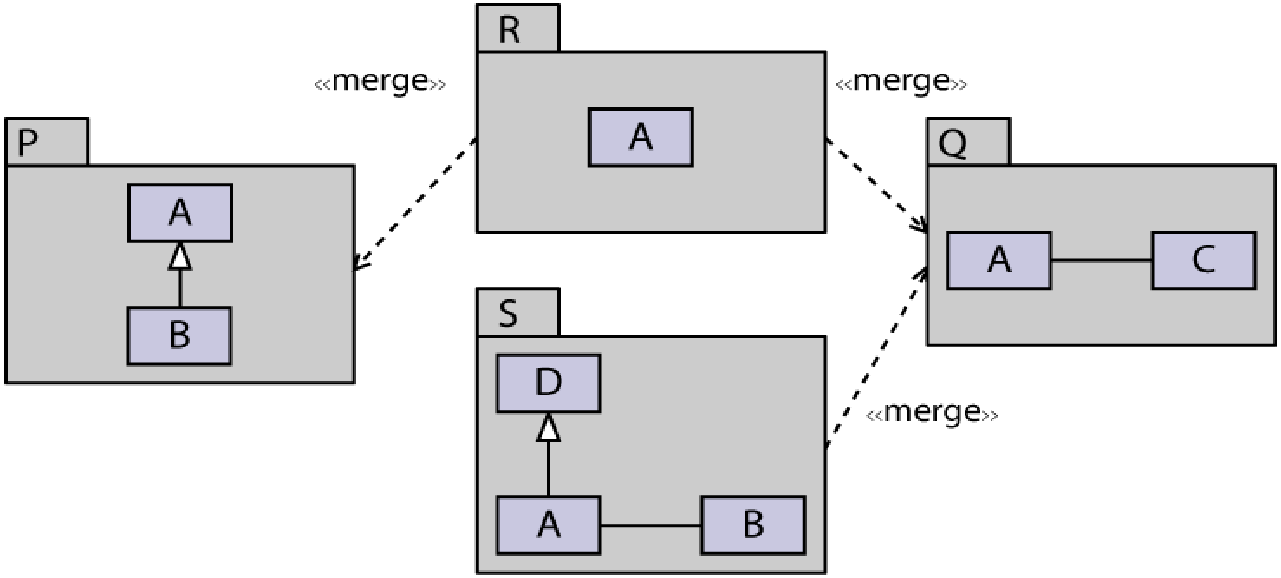


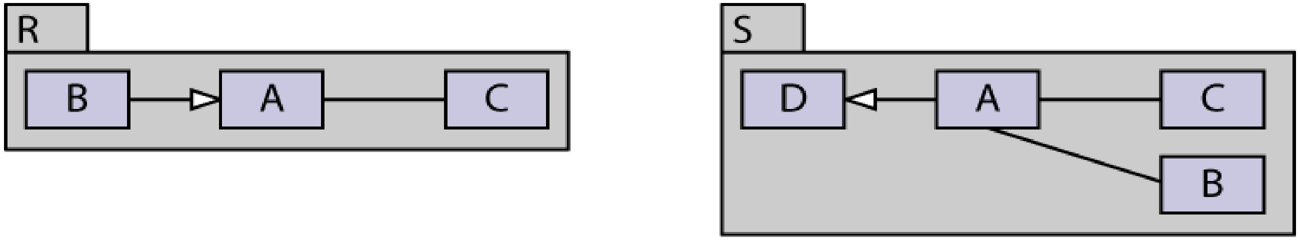
Package	Element	Visibility	
A	X	private	separate private element import (otherwise public overrides private)
A	Q	public	all remaining visible elements of B
B	X	public	public import
B	Q	public	default visibility
B	R	private	private import, renaming

Packages (3)

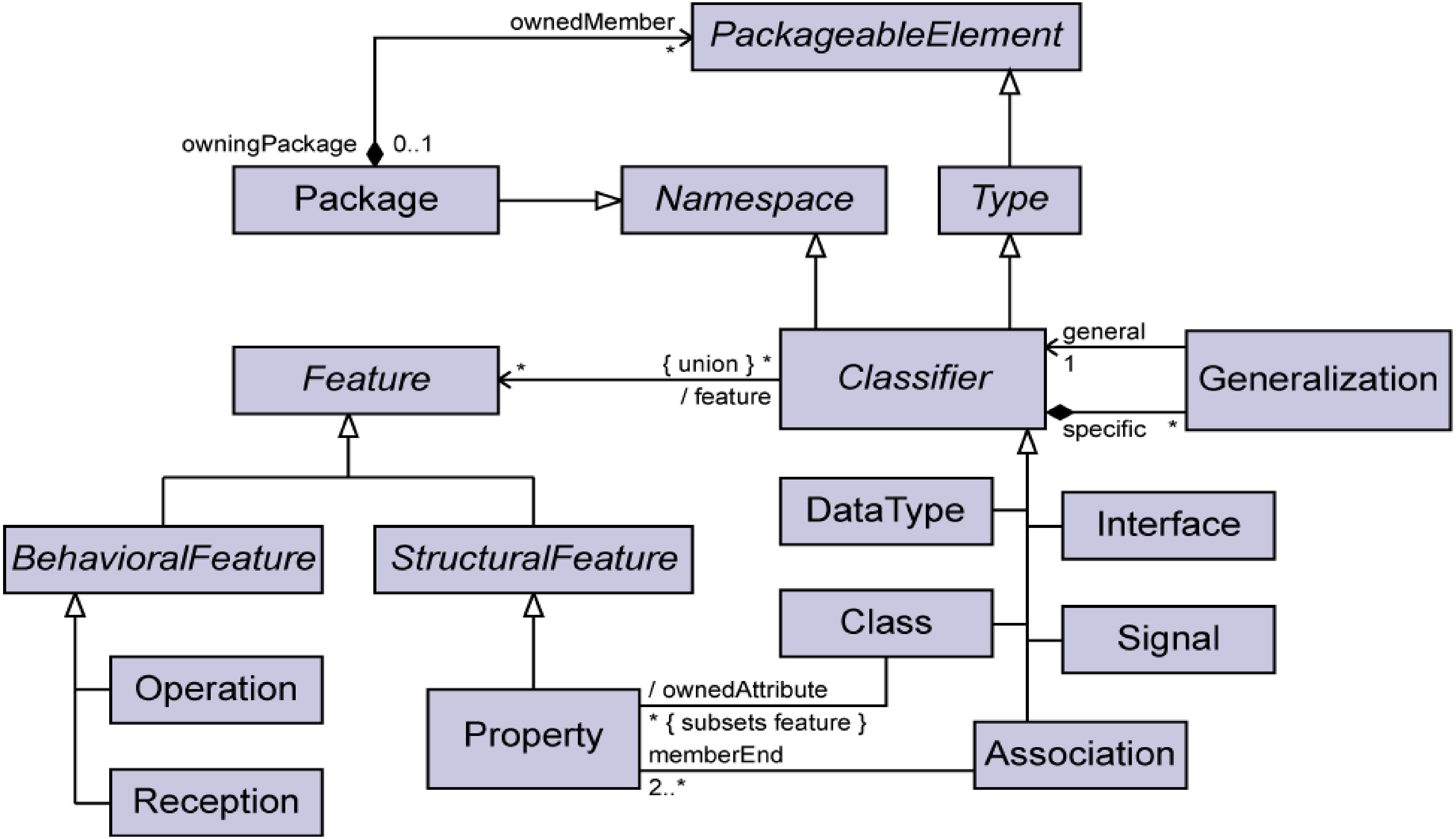
- **Package mergings** combine concepts incrementally.
  - … but use with care

- The receiving package defines the increment.
- The receiving package is simultaneously the resulting package.
- Merging is achieved by (lengthy) transformation rules (not defined for behaviour).
- Package merging is used extensively in the UML 2 specification.





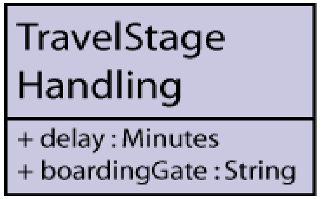
Metamodel



ownedMember
*
PackageableElement
owningPackage
0..1
Package
Namespace
Type
general
1
Generalization
{ union } *
/ feature
*
Feature
Classifier
specific
*
DataType
Interface
BehavioralFeature
StructuralFeature
Class
Signal
Operation
Property
/ ownedAttribute
* { subsets feature }
memberEnd
2..*
Association
Reception

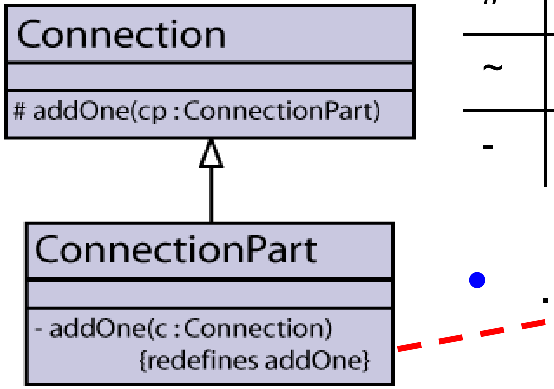
Features

- … belong to a namespace (e.g., class or package)



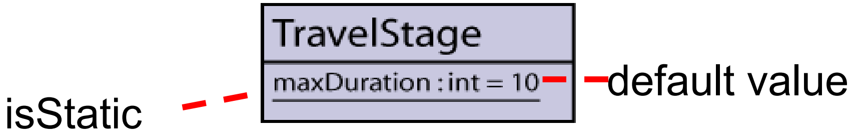
Visibility kinds (no default)

		visible to elements …
+	public	that can access owning namespace (by membership, import, or access)
#	protected	with generalization to owning namespace
~	package	in the same package as the owning namespace
-	private	in owning namespace only



- … are redefinable (unless decorated by { leaf })
  - in classes that specialize the context class

- … can be defined on instance or class level

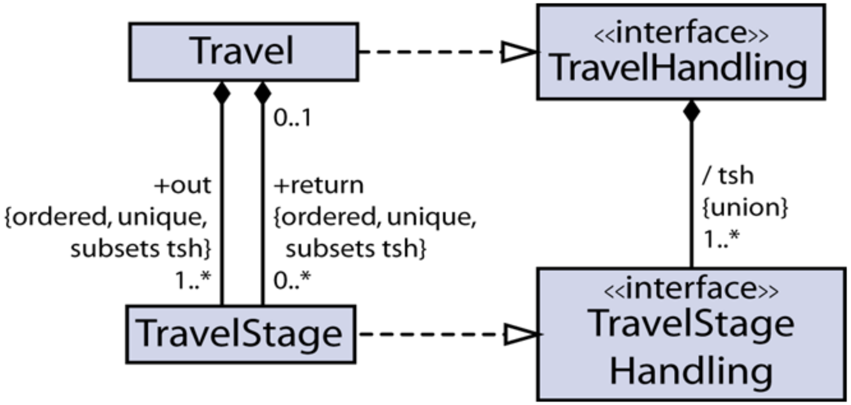


Properties

Aggregation kinds (default: *none*)

none	———	reference
shared	◇———	undefined (!)
composite	◆———	value

{ ordered }	{ unique }	Collection type
√	√	`OrderedSet`
√	×	`Sequence`
×	√	`Set` (default)
×	×	`Bag`



/ ({ derived }) can be computed from other information (default: false)

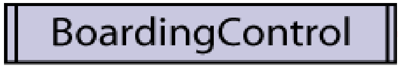
{ readOnly } can only be read, not written (default: false = unrestricted)

{ union } union of subset properties (implies derived)

{ subsets … } which property this property is a subset of

Behavioral features

- … are realized by behaviors (e.g., code, state machine).
  - { abstract } (virtual) behavioral features declare no behavior
    - behavior must be provided by specializations
  - Exceptions that may be thrown can be declared
  - Limited concurrency control
    - { active } classes define their own concurrency control



active class (with own behavior which starts on instance creation)

- in passive classes:

Call concurrency kinds (no default)

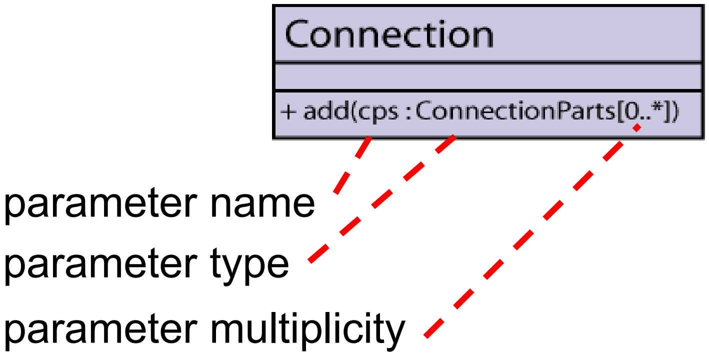
{ sequential }	no concurrency management
{ guarded }	only one execution, other invocations are blocked
{ concurrent }	all invocations may proceed concurrently

Operations (1)

- An **operation** specifies the name, return type, formal parameters, and constraints for invoking an associated behaviour.
  - «pre» / «post»
    - precondition constrains system state on operation invocation
    - postcondition constrains system state after operation is completed
  - { query }: invocation has no side effects
    - «body»: body condition describes return values
  - { ordered, unique } as for properties, but for return values
  - exceptions that may be thrown can be declared

Parameter direction kinds (default: in)

in	one way from caller
out	one way from callee
inout	both ways
return	return from callee (at most 1)



Operations (2)

- Several *semantic variation points* for operations
  - What happens, if a precondition is not satisfied on invocation?
  - When inherited or redefined
    - invariant, covariant, or contravariant specialization?
    - How are preconditions combined?

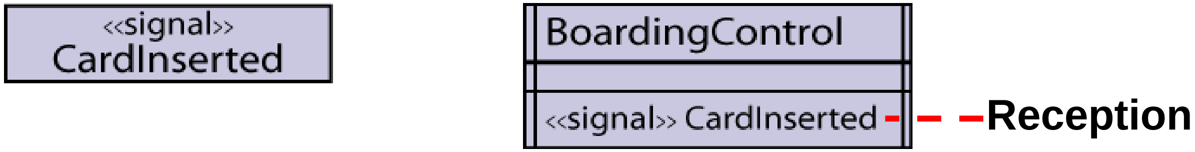
- **No predefined resolution principle** for inherited or redefined operations
  - “The mechanism by which the behavior to be invoked is determined from an operation and the transmitted argument data is a semantic variation point.”
  - a single-dispatch, object-oriented resolution principle is mentioned explicitly in the UML 2 specification

- Operation invocations may be **synchronous** or **asynchronous**.

Signals and receptions

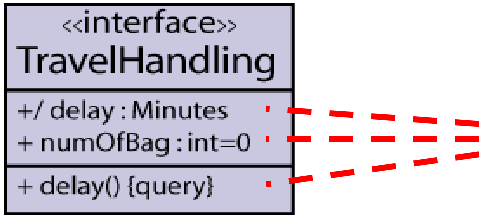
- A **signal** is a specification of type of send request instances communicated between objects.
  - Signals are classifiers, and thus may carry arbitrary data.
  - A signal triggers a reaction in the receiver in an asynchronous way and without a reply (no blocking on sender).

- A **reception** is a declaration stating that a classifier is prepared to react to the receipt of a signal.
  - Receptions are behavioral features and thus are realized by behavior (e.g., a state machine).



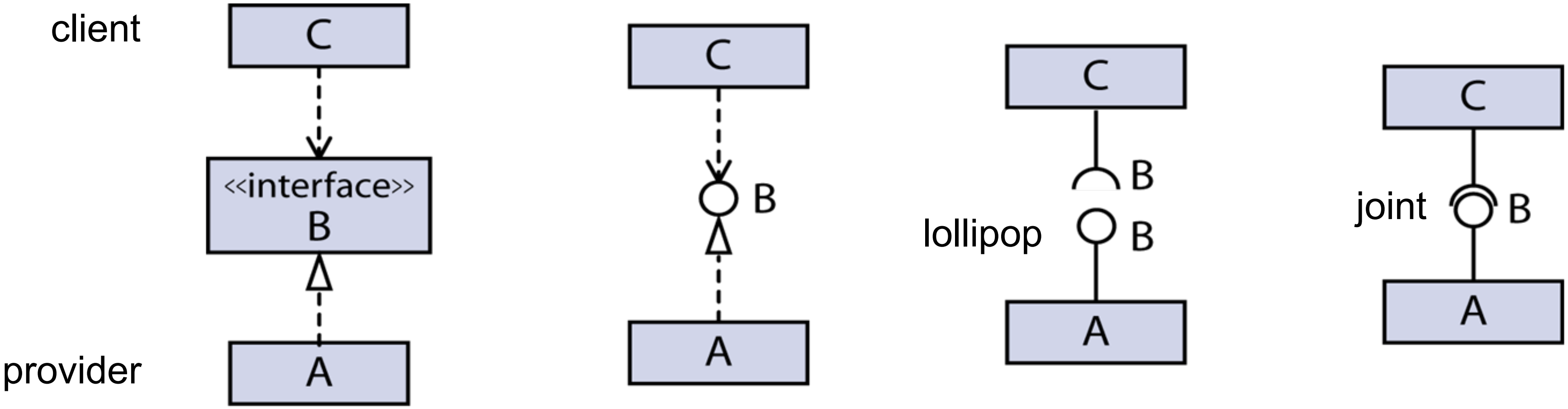
Interfaces

- **Interfaces** declare a set of coherent public features and obligations.
  - i.e., specify a contract for implementers (realizers)



features to be offered

Several notations for client/provider relationship



client
provider
C
<<interface>>
B
A
C
B
A
C
B
lollipop
B
A
C
joint
B
A