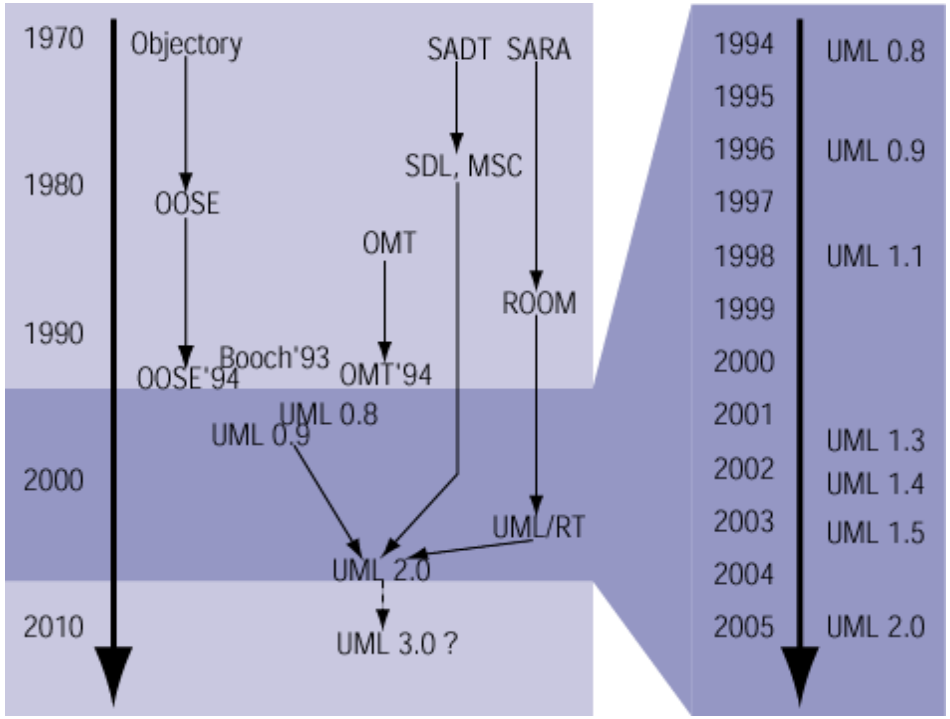
Unified Modeling Language 2

History and Predecessors

- The UML is the “lingua franca” of software engineering.
- It subsumes, integrates and consolidates most predecessors.
- Through the network effect, UML has a much broader spread and much better support (tools, books, trainings etc.) than other notations.
- The transition from UML 1.x to UML 2.0 has
  - resolved a great number of issues;
  - introduced many new concepts and notations (often feebly defined);
  - overhauled and improved the internal structure completely.
- While UML 2 still has many problems, it is much better than what we ever had before.

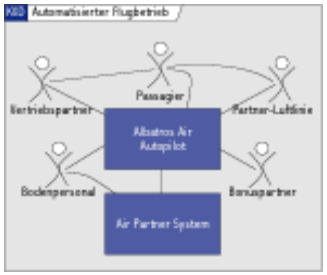


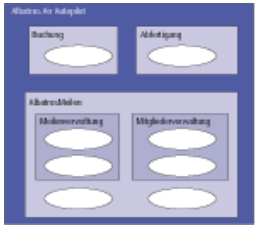
current version (“the standard”) UML 2.4.1
formal/2011-08-06 of August ’11

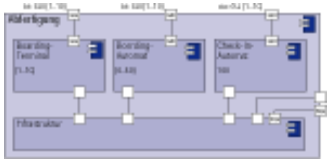
Usage Scenarios

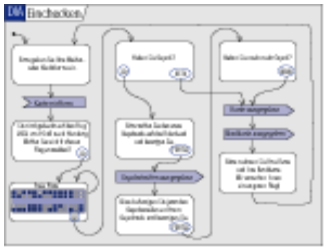
- UML has not been designed for specific, limited usages.
- There is currently no consensus on the rôle of the UML:
  - Some see UML only as tool for sketching class diagrams representing Java programs.
  - Some believe that UML is "*the prototype of the next generation of programming languages*".
- UML is a really a system of languages ("notations", "diagram types") each of which may be used in a number of different situations.
- UML is applicable for a multitude of purposes and during all phases of the software lifecycle – to varying degrees.

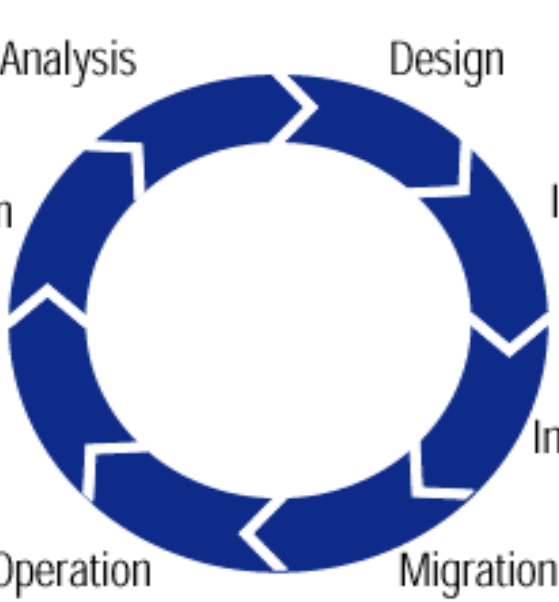
Usage Scenarios

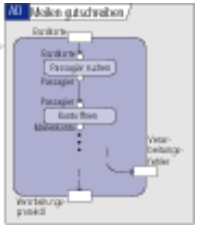


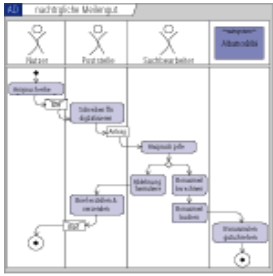


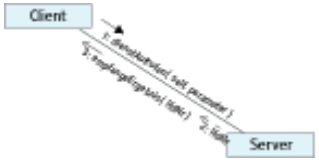


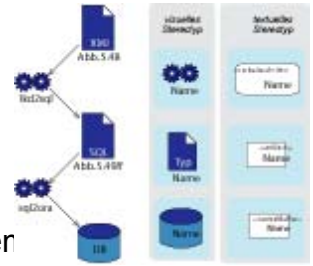


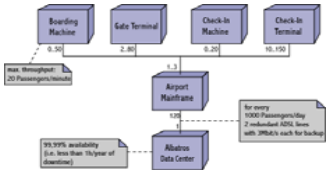


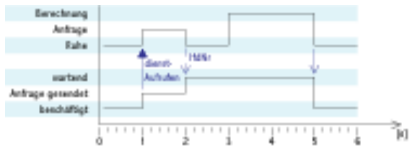


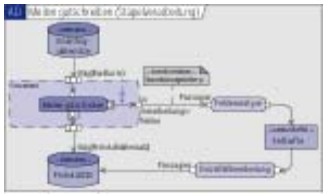


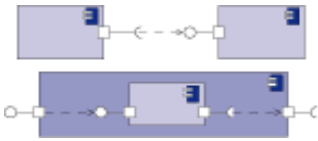












ML, with ser

Diagram types in UML 2

UML is a coherent system of languages rather than a single language.
Each language has its particular focus.

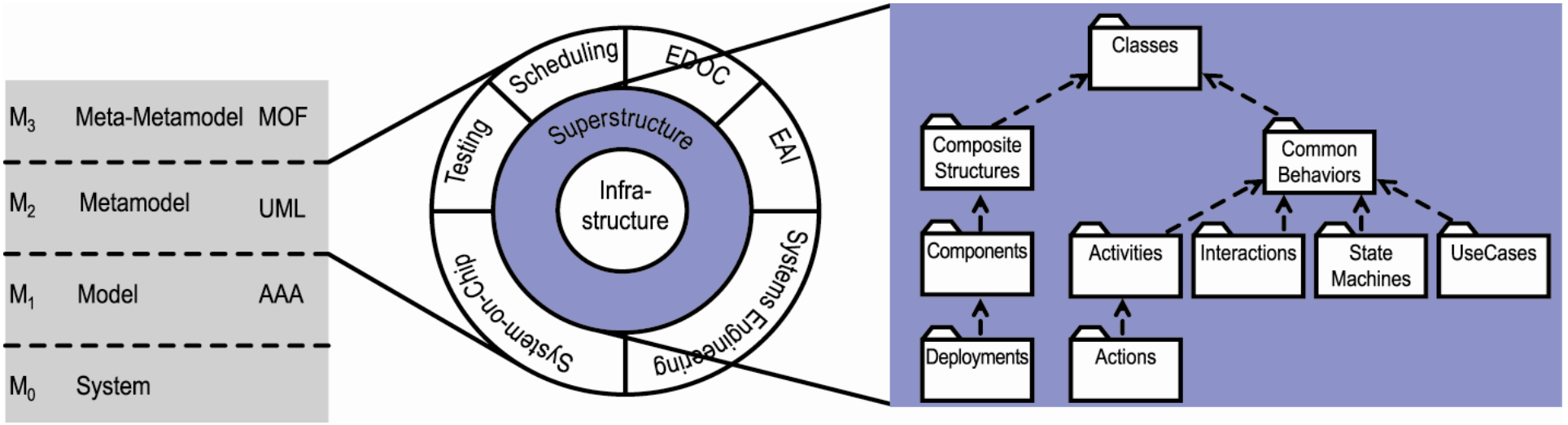
Structure	Class Diagram		static structure (generic/snapshot)
	Composite Structure Diagram		logical system structure
	Component Diagram		physical system structure
	Deployment Diagram		computing infrastructure / deployment
	Package Diagram		containment hierarchy
Behavior	Use Case Diagram		abstract functionality
	Activity Diagram		controlflow and dataflow
	Interaction	Sequence Diagram	*interactions by message exchange* message exchange over time
		Communication Diagram	structure of interacting elements
		Timing Diagram	coordinated state change over time
		Interaction Overview Diagram	flows of interactions
	State Machine Diagram		event-triggered state change

Internal Structure: Overview

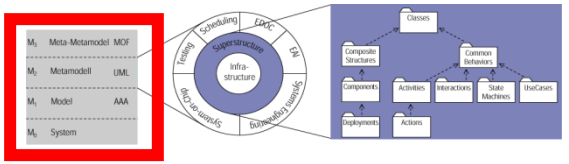
- The UML is structured using a metamodeling approach with four *layers*.
- The M_2-layer is called metamodel.

- The metamodel is again structured into *rings*, one of which is called superstructure, this is the place where concepts are defined ("the metamodel" proper).

- The Superstructure is structured into a tree of *packages* in turn.

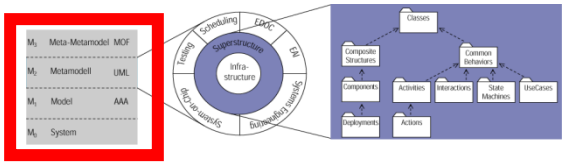


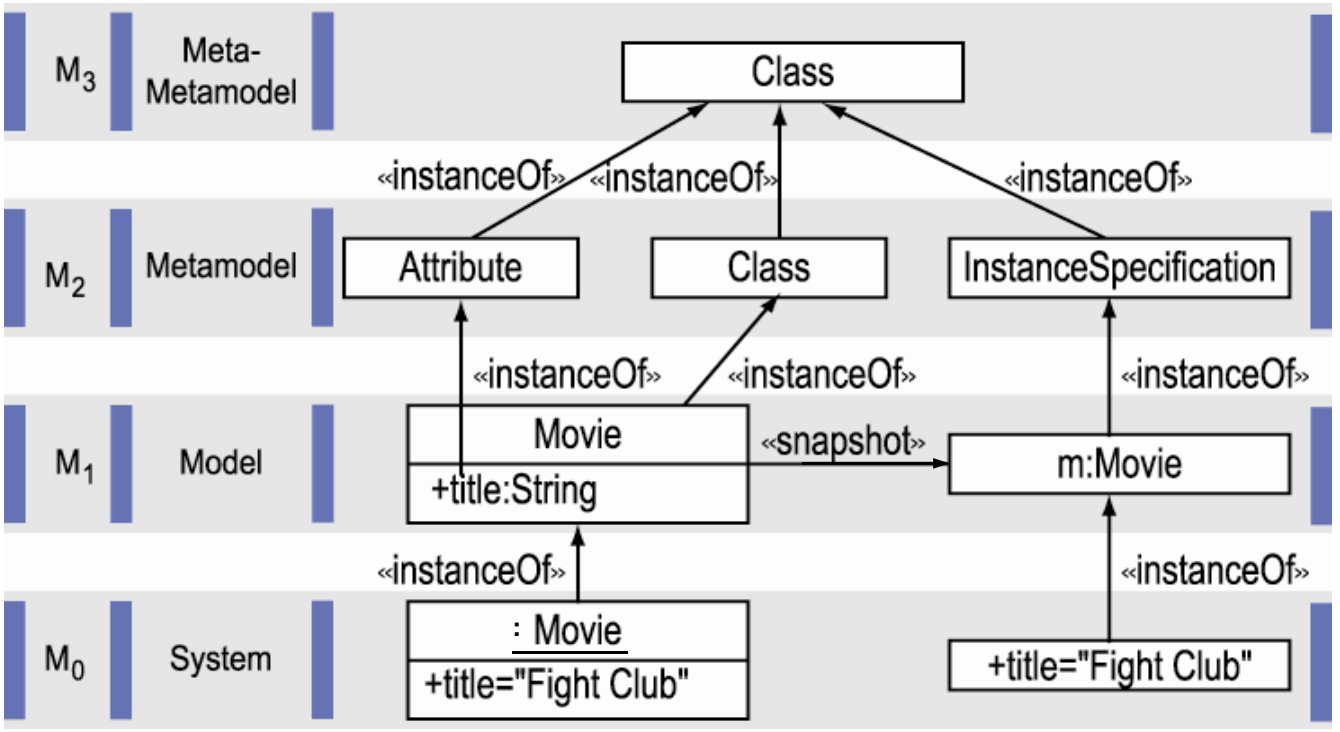
Internal Structure: Layers



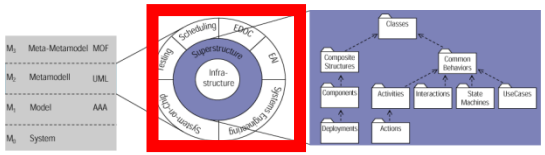
M_3	Meta-Metamodel	EBNF	Meta Object Facility (MOF)
M_2	Metamodel	Java grammar	Unified Modeling Language (UML) Common Warehouse Metamodel (CWM)
M_1	Model	a Java program	Albatros Air Autopilot
M_0	System	an execution of a Java program	a runtime state in a deployment of Albatros Air Autopilot

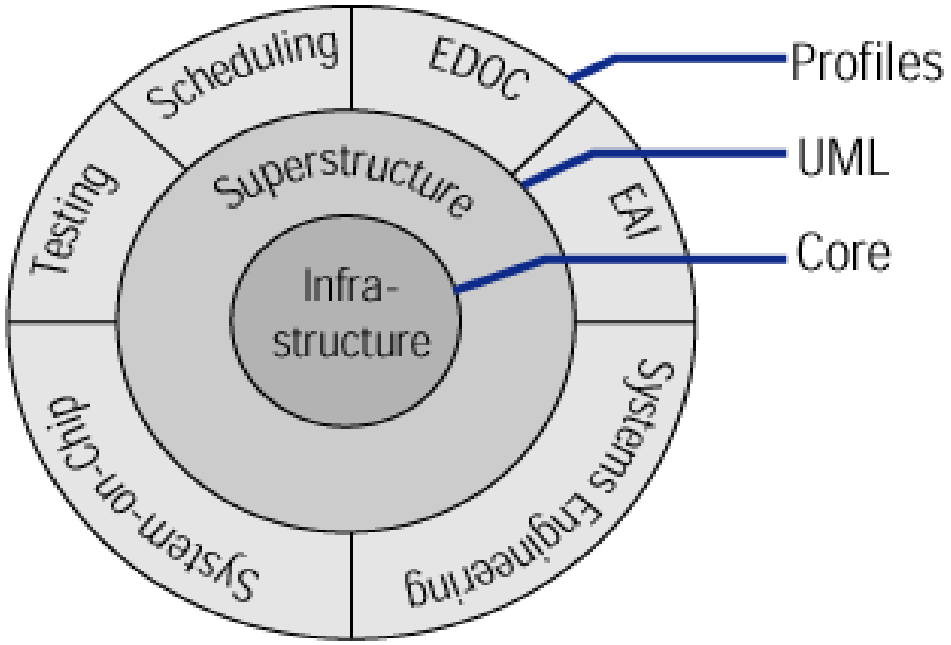
Internal Structure: Layers



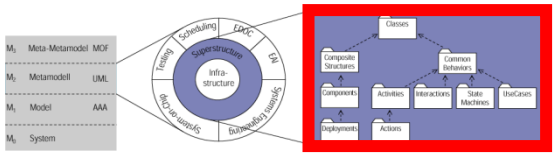


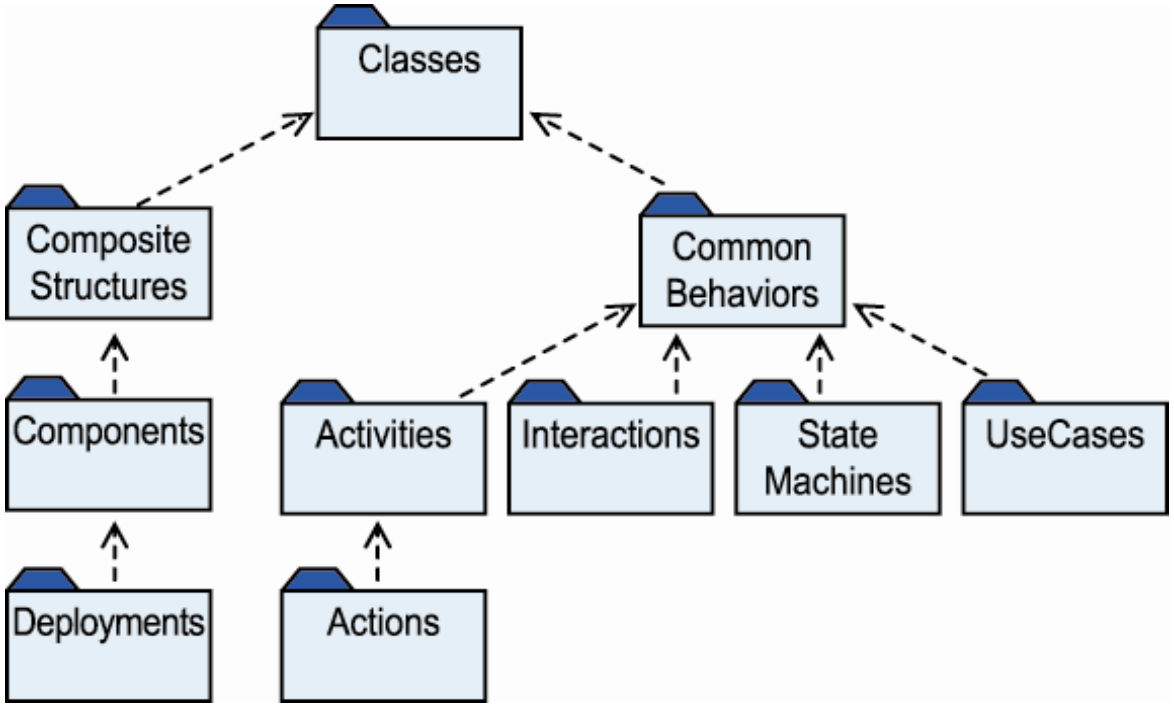
Internal Structure: Rings





Internal Structure: Packages





UML is not (only) object oriented

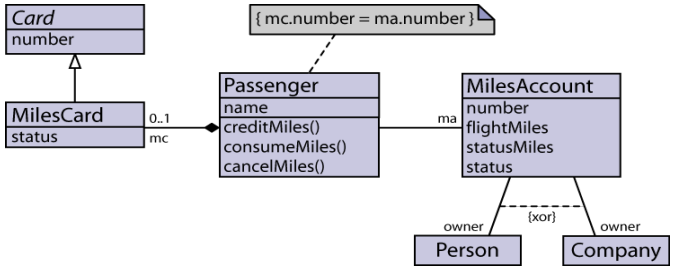
- A popular misconception about UML is that it is “object oriented” by heart – whatever that means.

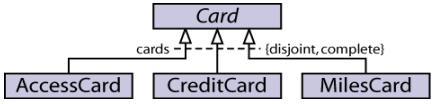
- It is true that
  - UML defines concepts like class and generalization;
  - UML is defined using (mainly) a set of class models;
  - UML 2 rediscovers the idea of behaviour embodied in objects.

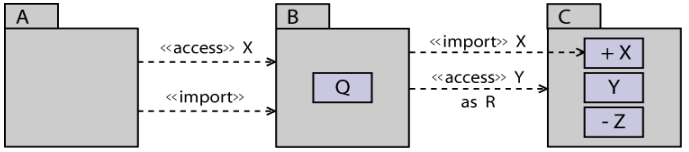
- However, UML 2
  - also encompasses many other concepts of non- or pre-OO origin (Activities, StateMachines, Interactions, CompositeStructure, …);
  - may be used in development projects completely independent of their implementation languages (Java, Cobol, Assembler, …);
  - is not tied to any language or language paradigm, neither by accident nor purpose.

Unified Modeling Language 2

Classes and packages







History and predecessors

- **Structured analysis and design**
  - Entity-Relationship (ER) diagrams (Chen 1976)
- **Semantic nets**
  - Conceptual structures in AI (Sowa 1984)
- **Object-oriented analysis and design**
  - Shlaer/Mellor (1988)
  - Coad/Yourdon (1990)
  - Wirfs-Brock/Wilkerson/Wiener (1990)
  - OMT (Rumbaugh 1991)
  - Booch (1991)
  - OOSE (Jacobson 1992)

Usage scenarios

- Classes and their relationships describe the vocabulary of a system.
  - **Analysis**: Ontology, taxonomy, data dictionary, …
  - **Design**: Static structure, patterns, …
  - **Implementation**: Code containers, database tables, …

- Classes may be used with different meaning in different software development phases.
  - meaning of generalizations varies with meaning of classes

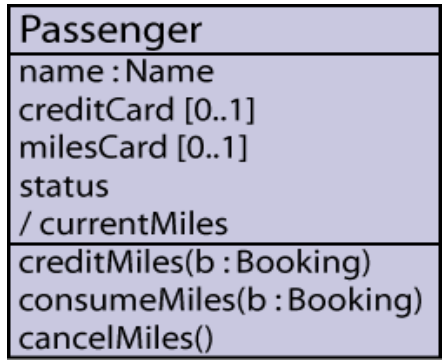
	Analysis	Design	Implementation
Concept	√		×
Type		√	√
Set of objects		√	√
Code	×		√

Classes

- Classes describe a set of instances with common features (and semantics).
  - Classes induce types (representing a set of values).
  - Classes are namespaces (containing named elements).

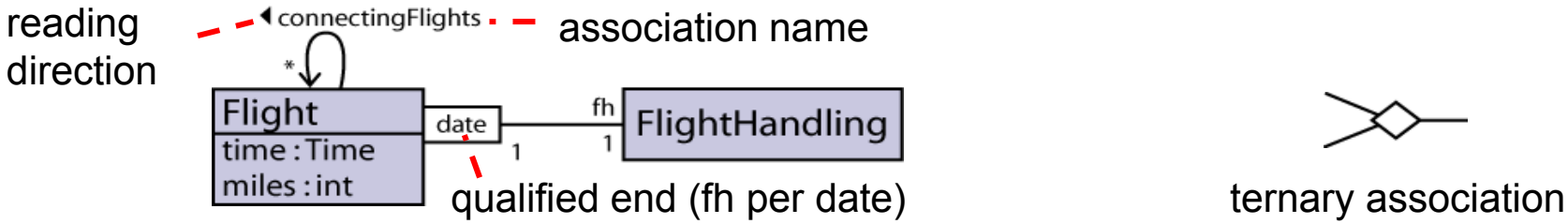
- **Structural features** (are typed elements)
  - properties
    - commonly known as attributes
    - describe the structure or state of class instances
    - may have multiplicities (e.g. **1, 0..1, 0..*, *, 2..5**)

- **Behavioral features** (have formal parameters)
  - operations
    - services which may be called
    - need not be backed by a method, but may be implemented otherwise

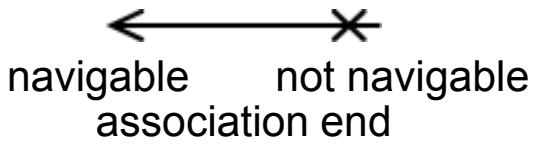


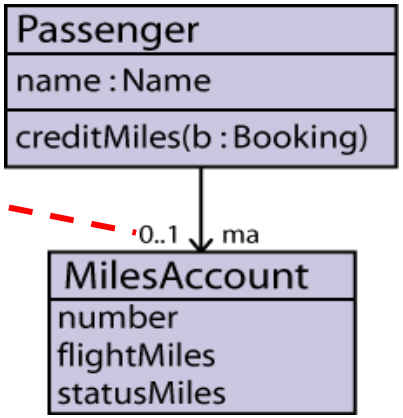
Associations

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



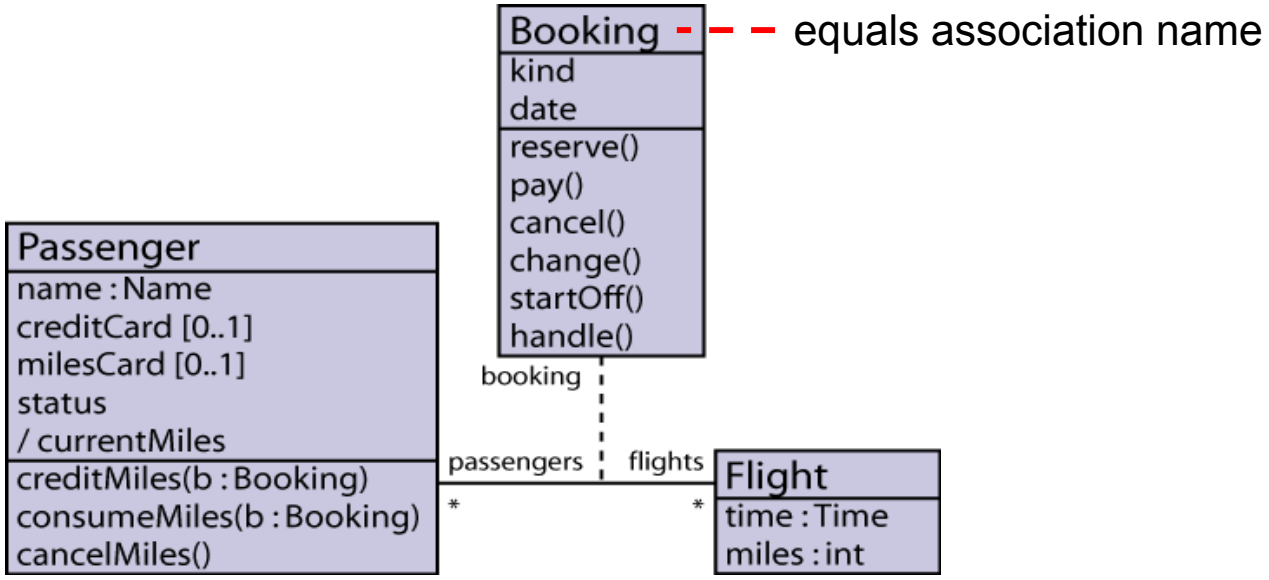
- **Association ends** are properties.
  - correspond to properties of the opposite class (default multiplicity is 0..1)
- Association ends may be navigable.
  - in contrast to general properties





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



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