Chapter 2:

Unified Modeling Language

Object-**O**riented
Software**C**onstruction

Prof. Dr. Armin B. Cremers,
Tobias Rho, Daniel Speicher, Holger Mügge

(based on Bruegge & Dutoit)

Overview: Modeling with UML

- What is modeling?
- What is UML?
- Some important diagrams
  - Use case diagrams
  - Class diagram
  - Object diagram
  - Sequence Diagram
  - State Chart Diagram

What is modeling?

- Modeling is on building an abstraction of a system to improve the understanding of it
- Abstractions are simplifications
  - They ignore irrelevant details
  - They only represent the relevant details
- What is *relevant* or *irrelevant* depends on the purpose of the model

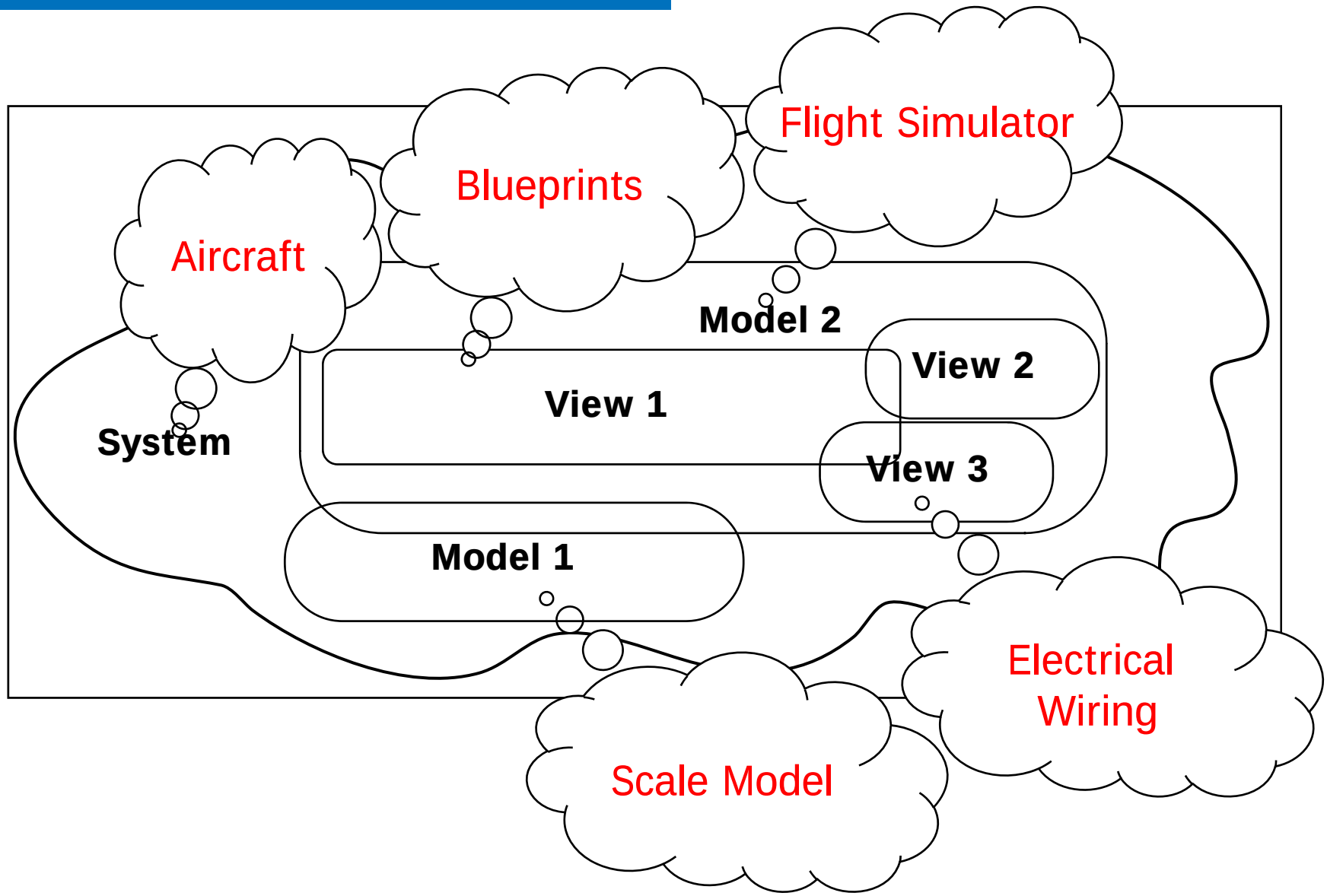
Systems, Models, and Views

- A *model* is an abstraction describing a subset of a system
- A *view* depicts selected aspects of a model
- Views and models of a single system may overlap each other

Examples

- System: Aircraft
- Models: Flight simulator, scale model
- Views: All blueprints, electrical wiring, fuel system

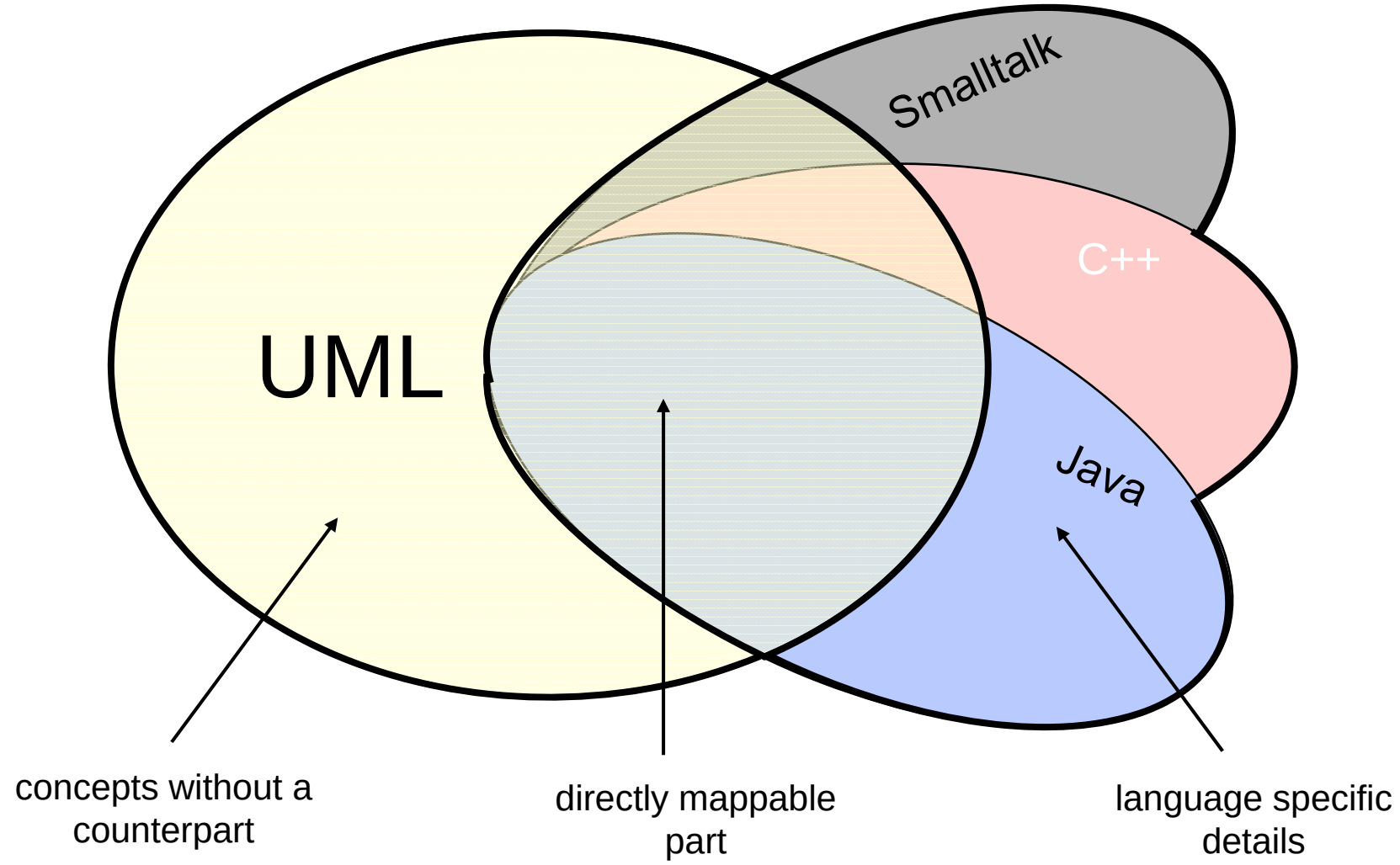
Systems, Models, and Views



What is UML?
Historical Overview

- UML (Unified Modeling Language)
  - A default standard for modeling object-oriented software
  - language for
    - visualizing, specifying, constructing, and documenting the artifacts of software-intensive systems
- Reference: “The Unified Modeling Language User Guide”, Addison Wesley, 2005.
- Supported by a range of CASE tools
  - Rational
  - Together
  - MagicDraw, ….
- You can model 80% of most problems by using about 20% UML - We teach you those 20%

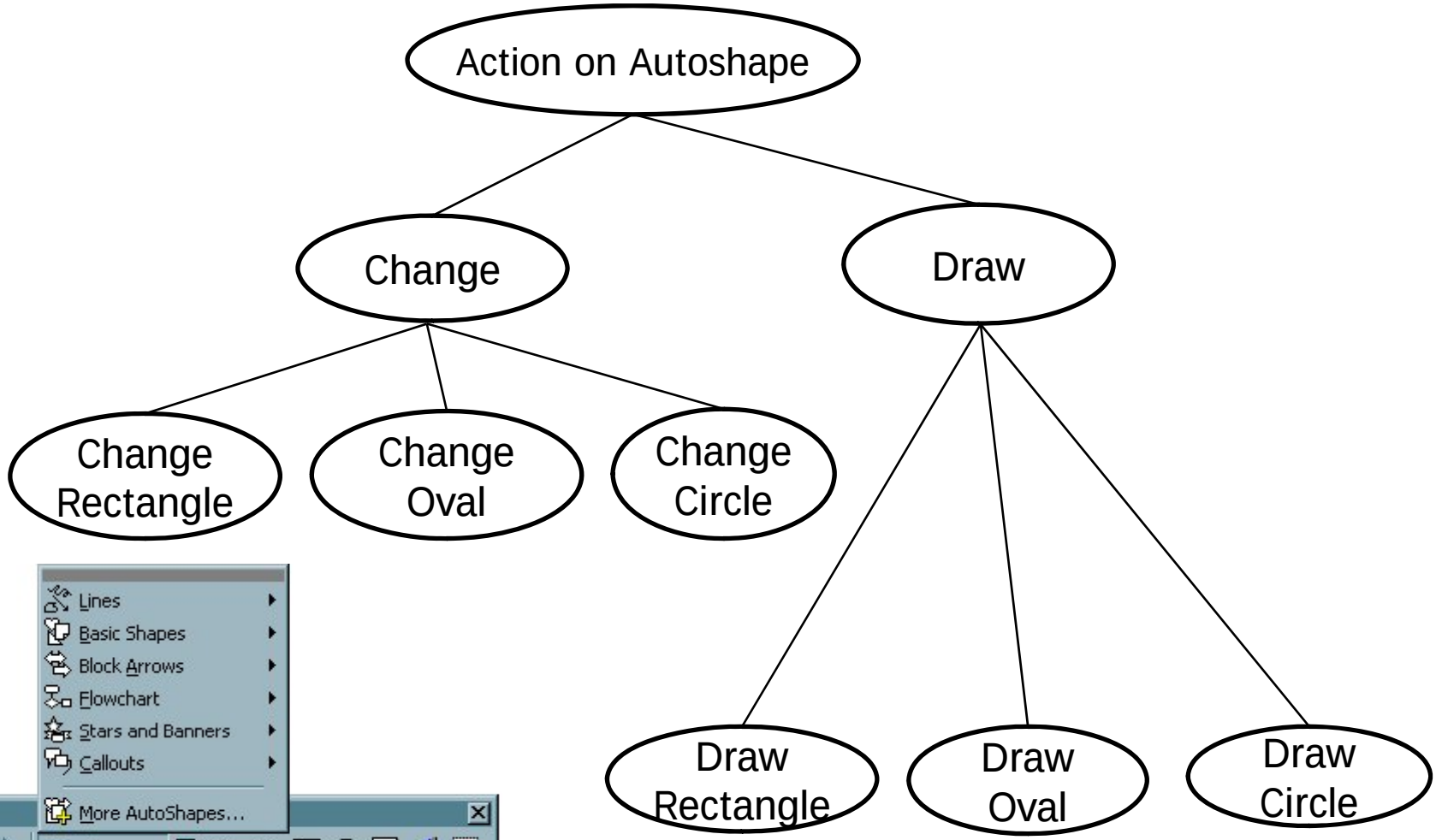
Relationship between UML and current OO languages



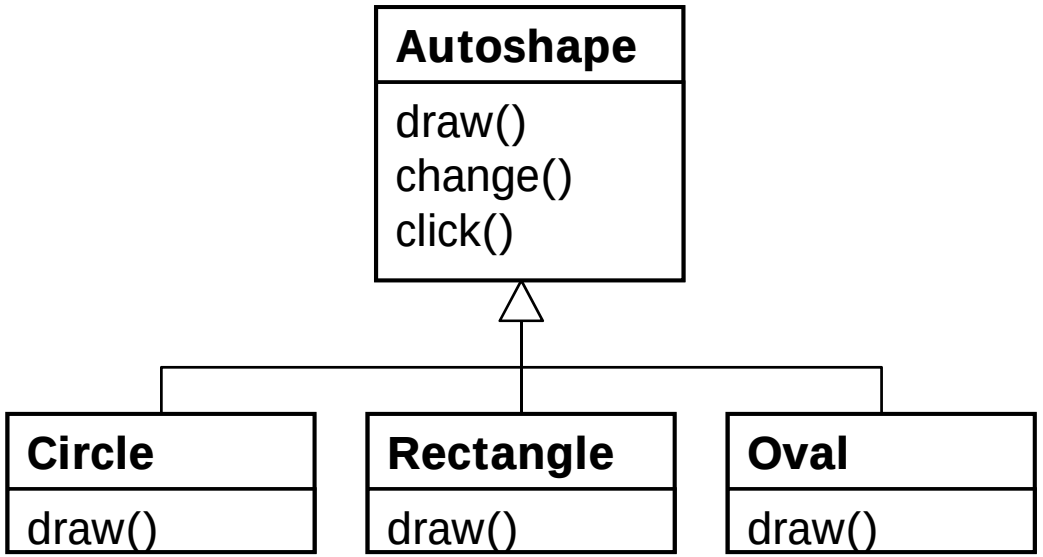
Different Views

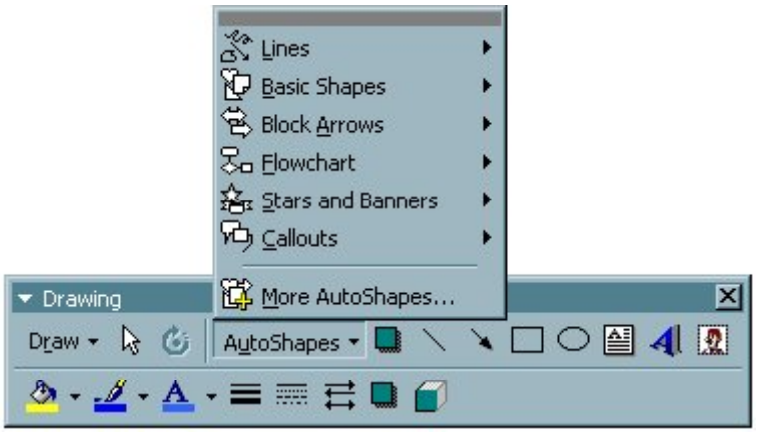
- Functional requirements view
  - Models the user's point of view
- Static structural view
  - Models static structure of the system using classes, attributes, operations, and relationships
- Dynamic behavior view
  - Models collaborations among objects and changes to the internal states of objects

Functional requirements view
Functional decomposition: Use case diagram

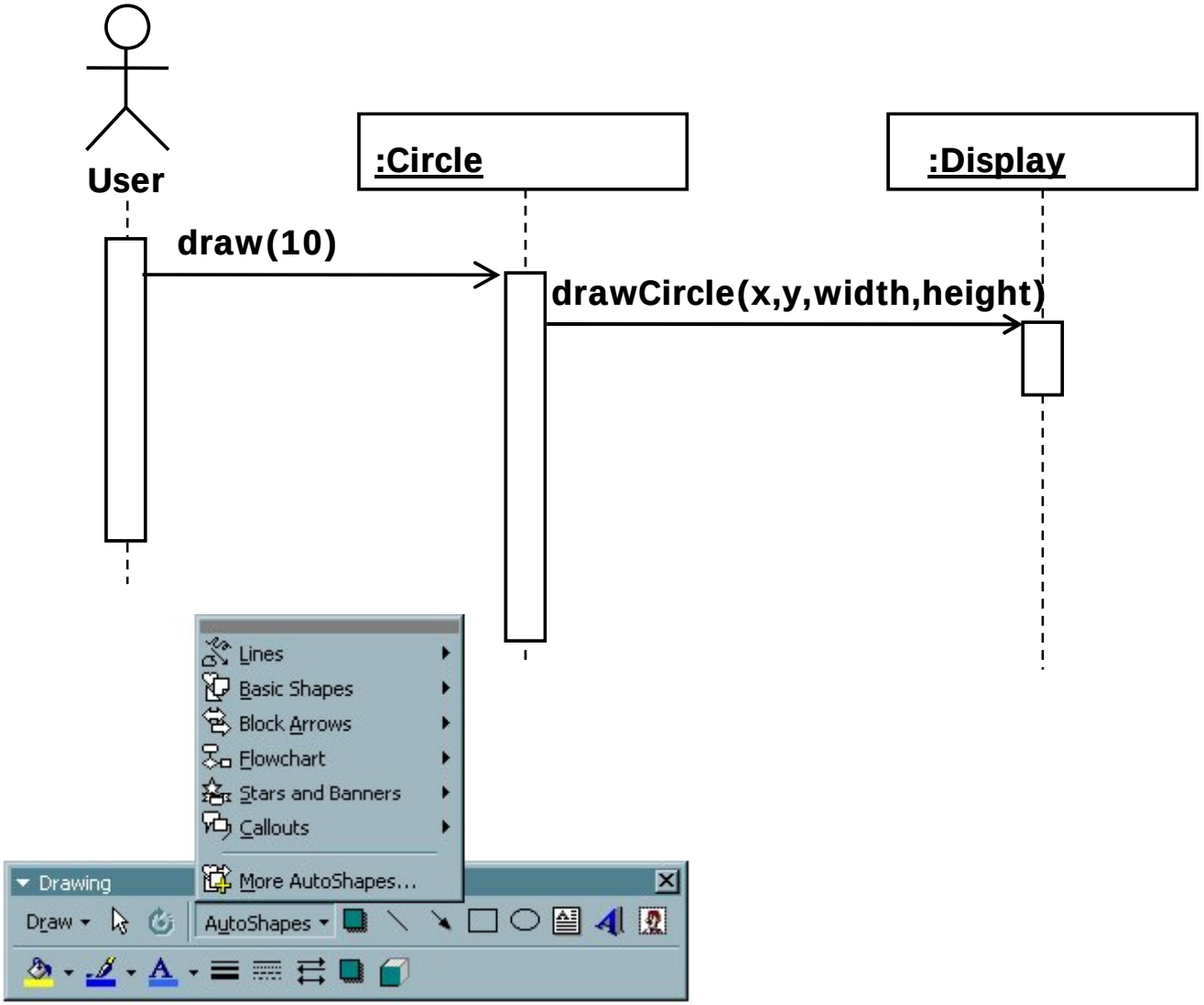


Structural View
Object-oriented decomposition: Class diagram





Dynamic behavior view
Dynamic Model: Sequence Diagram



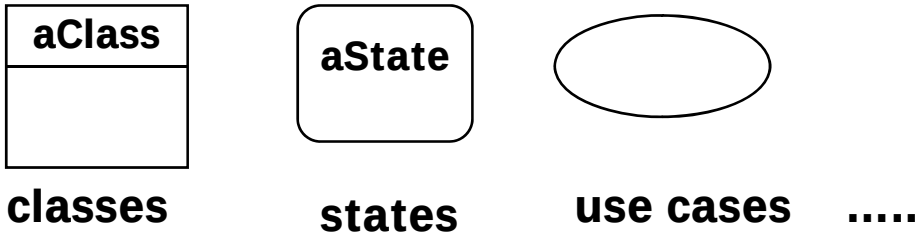
What is UML?
Engineering aspects

- Forward Engineering
  - Generation of code from a UML model into a programming language
- Reverse Engineering
  - Reconstruct a model from an implementation back into UML models

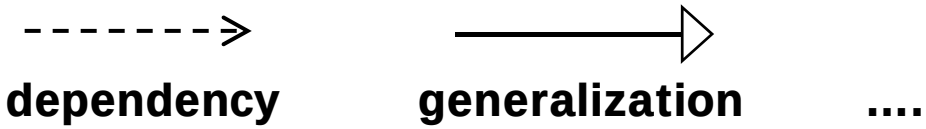
- UML provides means for modeling and documenting all kind of artifacts available for a software system:
  - Architecture
  - Design
  - Source Code
  - Requirements
  - Tests
  - Project Plans

What is UML?
Building Blocks of UML

- The vocabulary of the UML encompasses three kinds of building blocks:
  - Things (first class citizens in a model)

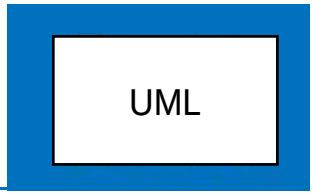


  - Relationships (to tie things together)



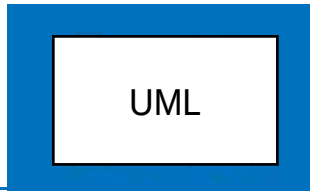
  - Diagrams
    - Group collections of things
    - Graphical presentation of a set of elements, most often rendered as graphs (vertices = things; paths = relationships)

What is UML?
Building Blocks of UML – Overview on diagrams



1. Class diagram
2. Object diagram
3. Component diagram
4. Composite structure diagram
5. Use case diagram
6. Sequence diagram
7. Communication diagram
8. State diagram
9. Activity diagram
10. Deployment diagram
11. Package diagram
12. Timing diagram
13. Interaction overview diagram

What is UML?
Building Blocks of UML – Overview on diagrams



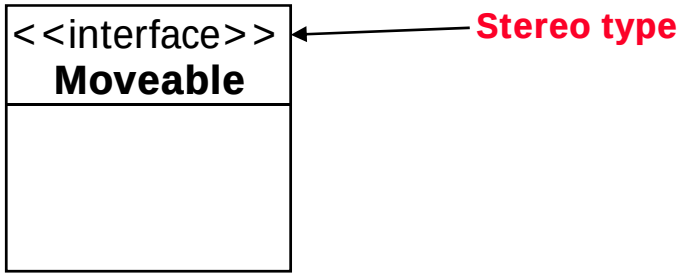
1. **Class diagram**
2. **Object diagram**
3. Component diagram
4. Composite structure diagram
5. **Use case diagram**
6. **Sequence diagram**
7. Communication diagram
8. **State diagram**
9. **Activity diagram**
10. Deployment diagram
11. Package diagram
12. Timing diagram
13. Interaction overview diagram

What is UML?
Meta Model (in short)

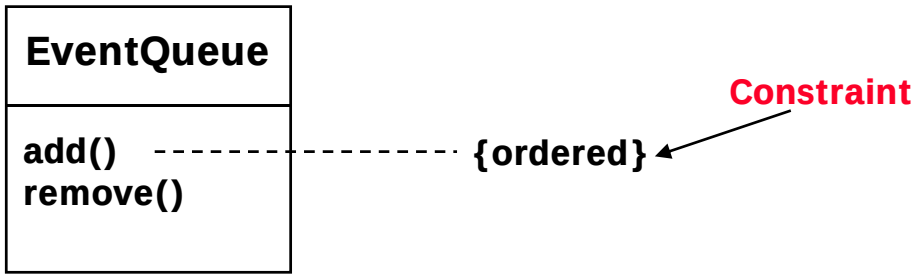
- Formal definition of all elements
- Meta-Models for all diagram types based on
- Meta-Metamodel, the Meta-Object-Facility (MOF)
  - Profiles (stereo types, constraints, tagged values)
- Foundation for
  - Extending diagram types
  - Defining new types
  - exporting/importing (XMI) and transforming models
  - generate application code
  - → Model-driven software development
    - → more this in the ATSC class

What is UML?
Extensibility Mechanisms - Example

- Stereotypes
  - Extend the vocabulary of the UML for creating new building blocks
  - Mostly used for language-dependent features



- Constraint
  - Modify semantics of a building block



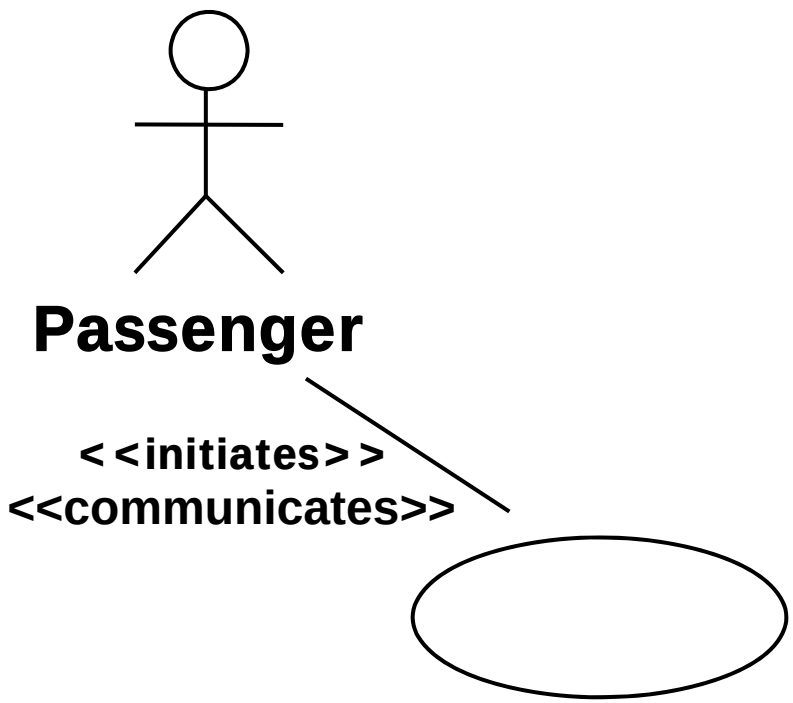
UML Diagrams

- Use case Diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Classes, Attributes, Associations
- Object diagrams
  - Show objects and their relations at a point in time.
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object (essentially a finite state automaton)

Use Case Diagrams
The idea

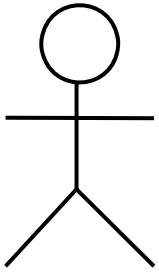
- Use case diagrams represent the functionality of the system from user's point of view
- Does also model the functional behavior of a system
- Used during requirements elicitation to represent external behavior
- Can be used to discuss current and future functionality with the users
- Most often the starting point for all further activities

Use Case Diagrams
The idea 2 (Basic ingredients)



- ***Actors*** models an external entity which communicates with the system
- ***Use cases*** represent a sequence of interaction with the system and thus a certain functionality of it
- The **use case model** is the set of all use cases. It is a complete description of the functionality of the system and its environment
- Description of **dependencies** between system functionalities and external actors

Use Case Diagrams
Actors



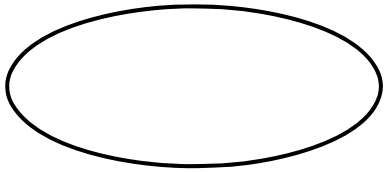
Passenger

- An actor models an *external* entity which communicates with the system
  - User
  - External system (DB, software, devices)
- An actor has a unique name and an optional description
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates

Class Diagrams
Actor vs. class

- What is the difference between an *actor*, a *class* and an *object*?
- Actor:
  - An entity outside the system to be modeled, interacting with the system (“Passenger”, “Satellite”)
- Class:
  - An abstraction modeling an entity in the problem domain, must be modeled inside the system (“Terminal”, “Order”)
- Object:
  - A specific instance of a class (“Joe, the passenger who is purchasing a ticket from the ticket distributor”).

Use Case Diagrams
Use Case



PurchaseTicket

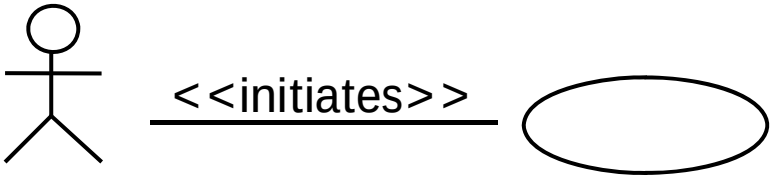
- An use case represents a class of functionality provided by the system as a sequence of interaction (event flow)

A use case consists of

- Unique name
- Participating actors
- Entry conditions
- Flow of events
- Exit conditions
- Special requirements

→ Hard to infer from the graphical representation

Use Case Diagrams
Textual Representation

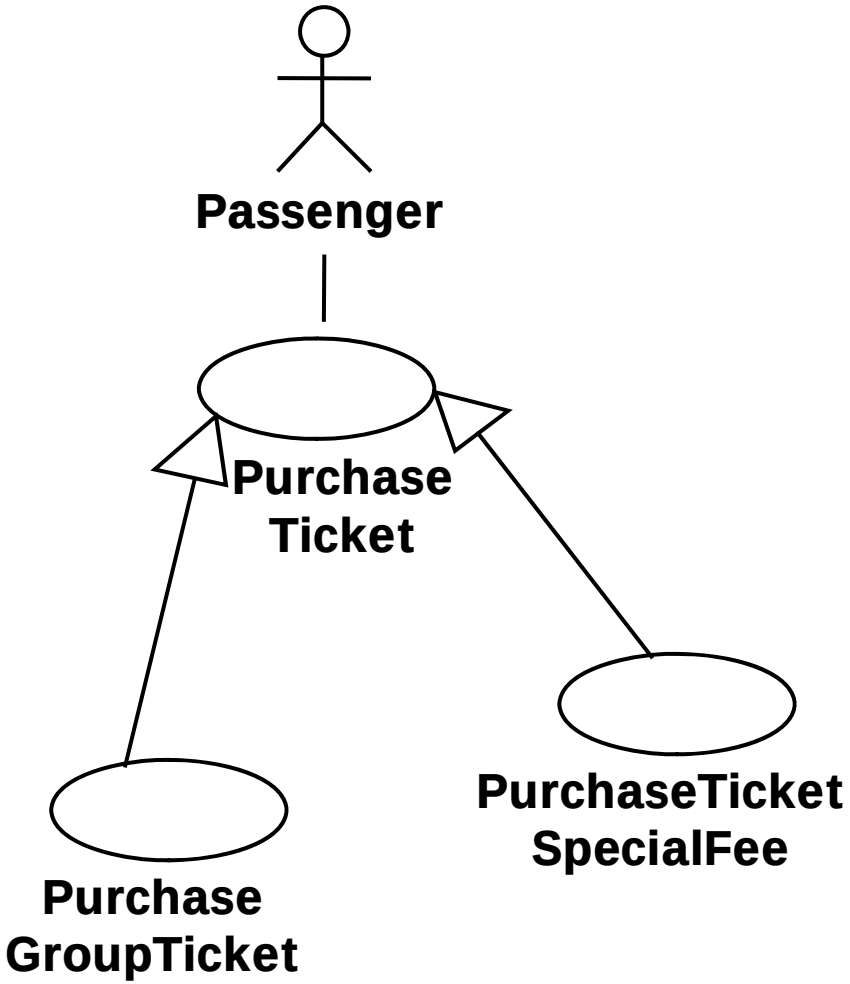


Use case name	Purchase ticket
Participating Actors	Initiated by **Passenger**
Flow of Events	1. Passenger selects the number of zones to be traveled. 2. SYSTEM displays the amount due. 3. Passenger inserts money, of at least the amount due (Extension Point: HandleException) 4. SYSTEM returns change. 5. SYSTEM issues ticket.
Entry Condition	Passenger standing in front of ticket distributor.
Exit Condition	Passenger has a correct ticket.
Quality Requirements	• The **Passenger's** ticket is issued within 15 seconds.

Initialized by an actor

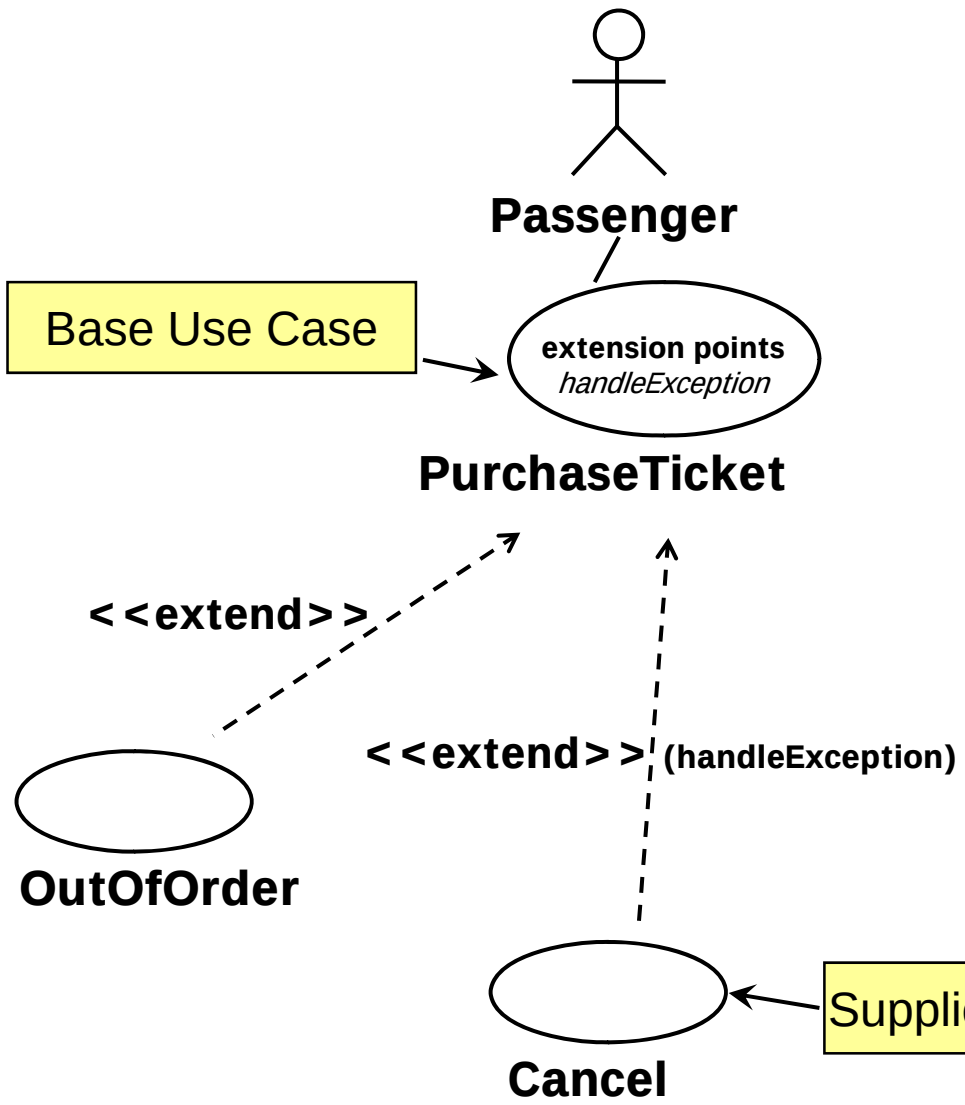
Initialized by the system

Use Case Diagrams
The *Generalization / Inheritance* Relationship



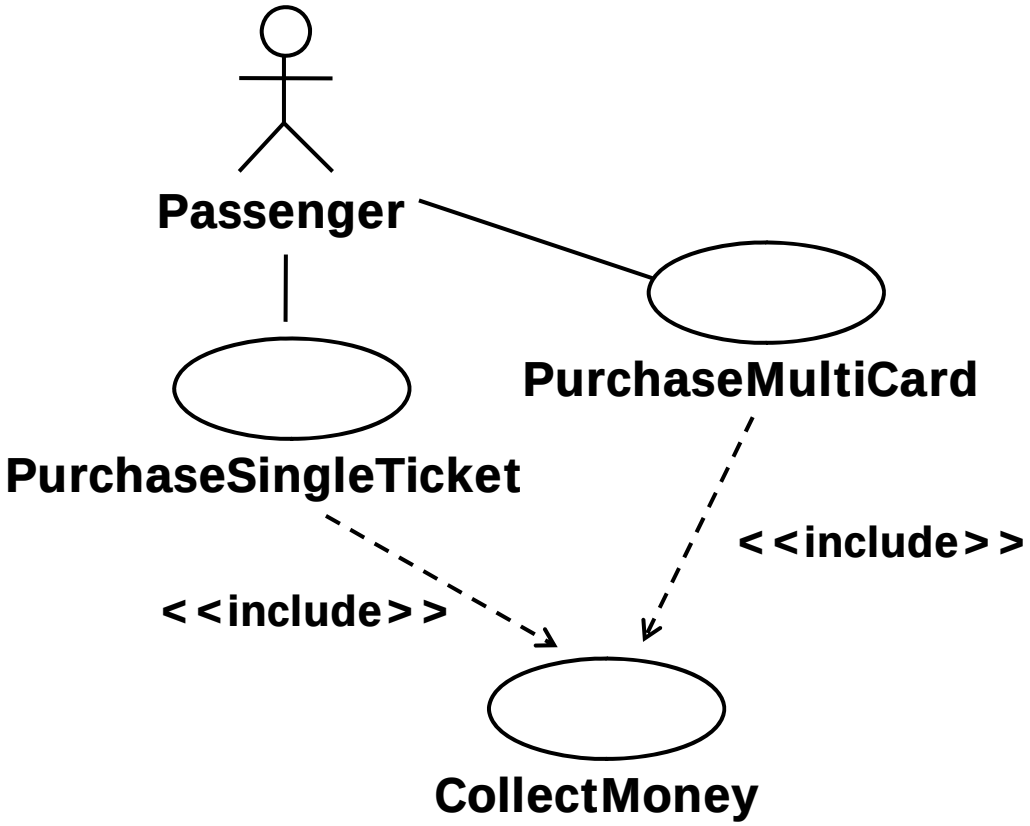
- **Generalization** is used like between classes
- Means specialization (inverted direction)
- Mostly the same use case but slightly different (enhanced) functionality
- Functionality can be used or overridden

Use Case Diagrams
The *<<extend>>* Relationship



- ***<<extend>>*** relationship is used to extend a use case by exceptional, optional or seldom invoked cases.
- Extension points may be defined in the base use case
- Condition must be specified when the extending use case will be inserted
- The direction of a ***<<extend>>*** relationship is from the extending (supplier) use case to the extended (base) use case
- Base use case can run independently

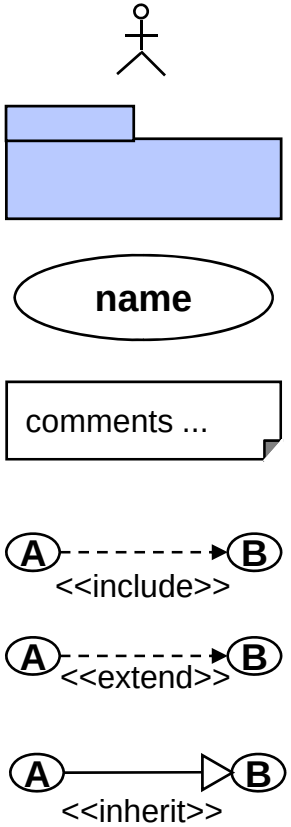
Use Case Diagrams
The *<<include>>* Relationship

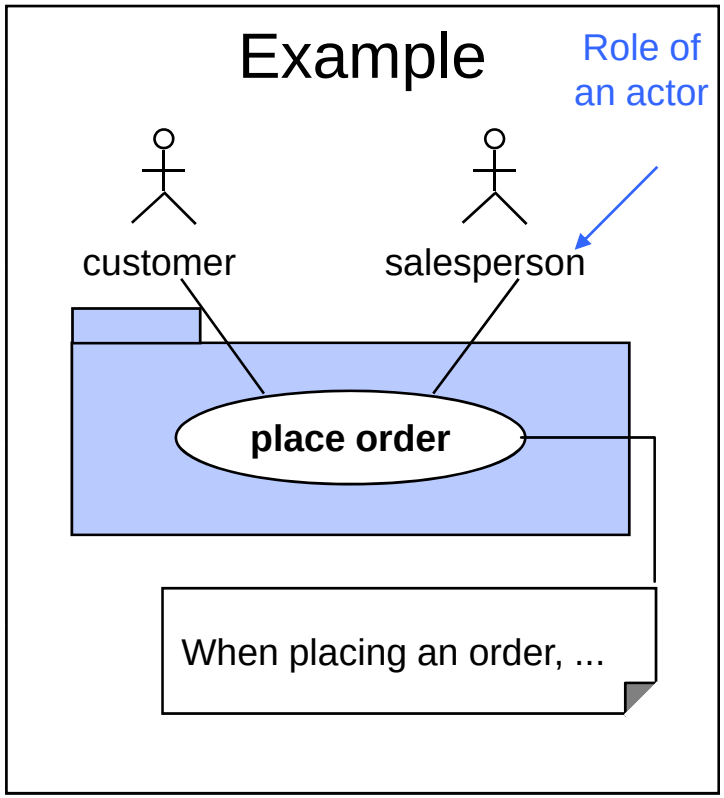


- ***<<include>>*** relationship is used to include behavior that is factored out of the use case
- ***<<include>>*** behavior used to share helpful behavior among use cases
- The direction of a ***<<include>>*** relationship is from the base use case to the using (supplier) use case (unlike ***<<extend>>*** relationships).
- Base use case *cannot* run independently

Use Case Diagrams
Overview

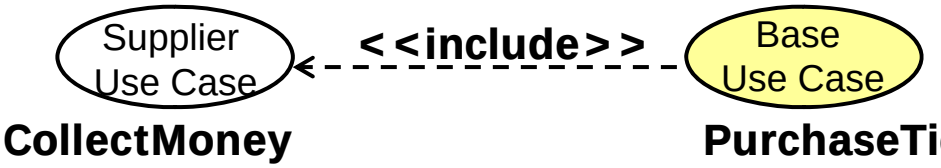
- Elements
  - Actor
  - Package
  - Use cases
  - Annotations
  - Relationships





Use Cases in a textual way
Describing <<include>> and <<extend>>

<<include>> relationship



Flow of Events

...

3. **Passenger** confirms the purchase of a ticket

4. SYSTEM receives the confirmation and asks for inserting coins (by using Use Case *CollectMany)*

<<extend>> relationship

Flow of Events

The *Cancel* use case extends any use case in which EP handleException was defined:

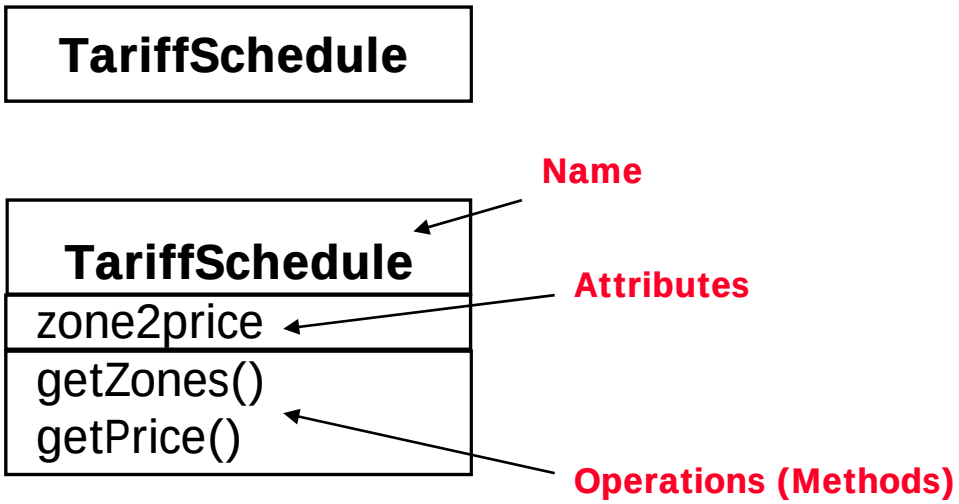
1. User presses Button Cancel

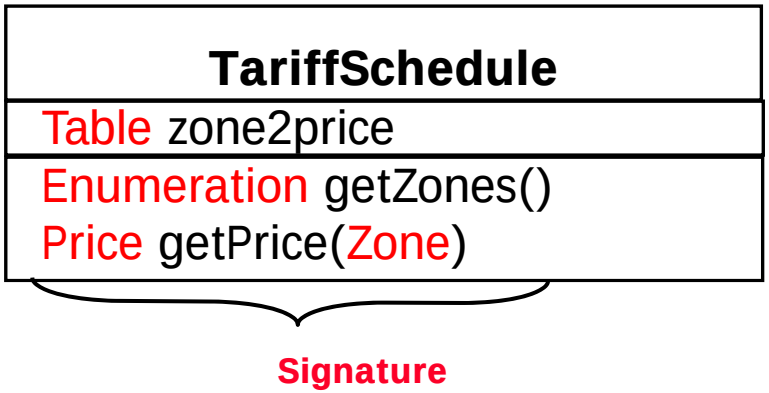
2. SYSTEM rolls back the transaction

UML Diagrams

- Use case Diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Classes, Attributes, Associations
- Object diagrams
  - Show objects and their relations at a point in time.
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object (essentially a finite state automaton)

Class Diagrams
Classes

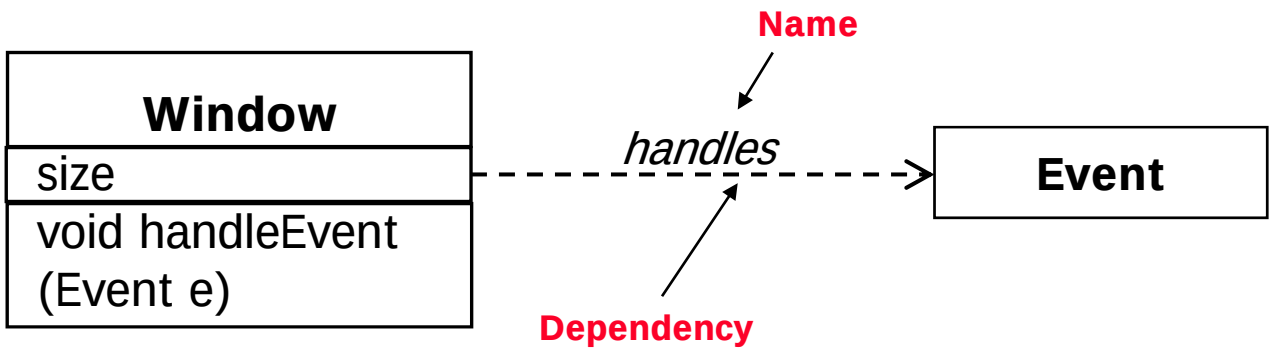




- A class represents a concept
- A class encapsulates state (attributes) and behavior (operations)
- Each attribute has a type
- Each operation has a signature
- The class name is the only mandatory information

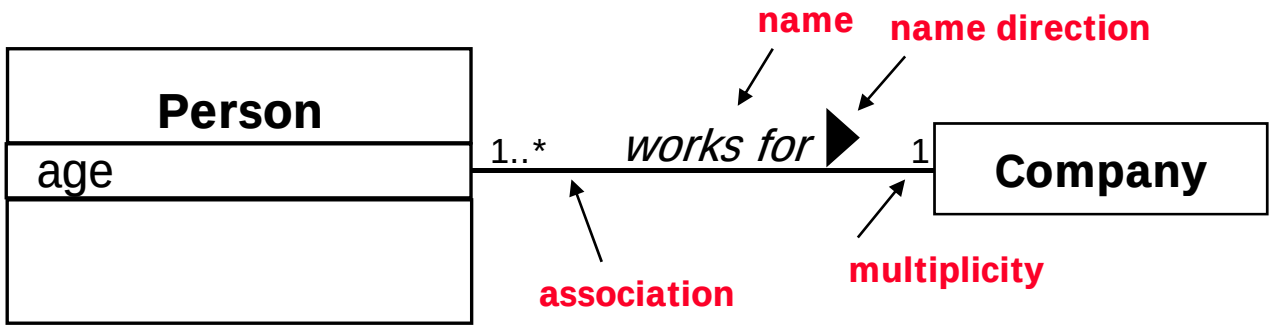
Class Diagrams
Relationships (Dependency)

- A dependency is a relationship that states that one thing (class) *uses* some information (methods, variables) of another things (class).



Class Diagrams
Relationships (Association)

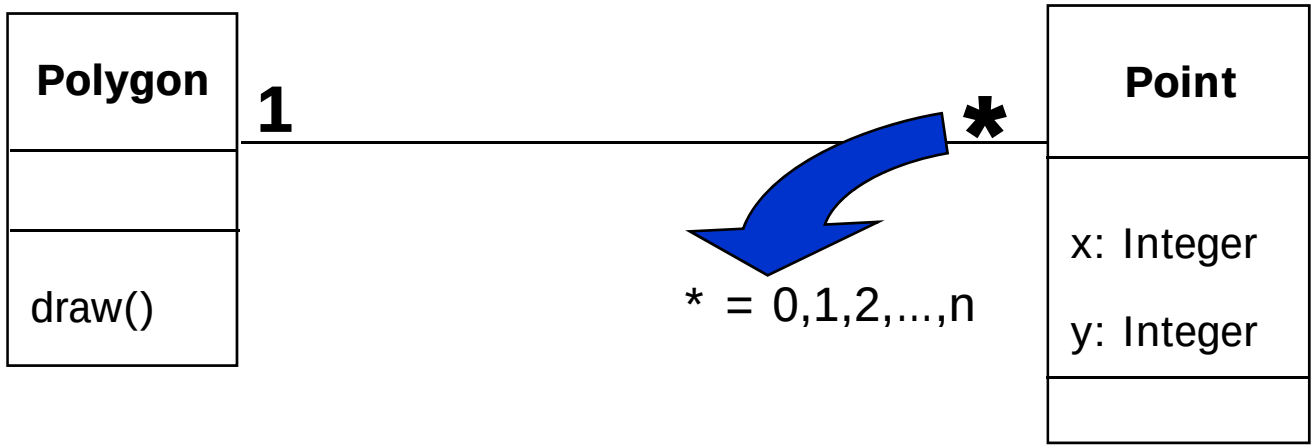
- An association is a structured relationship that objects of one class are connected to objects of another one



- The *multiplicity* of an association end denotes how many objects the source object (object of the other end) can reference
- The multiplicity is a range of integers often including a minimum and maximum value:
  - 1 (exactly one)
  - 0..1 (zero or one)
  - 0..* (or just *) (many)
  - 1..* (one or more)
  - 3 (or 3..3) (exact number)
  - 4..5 (exact range)

Class Diagrams
1-to-1 and 1-to-many Associations

One-to-one association



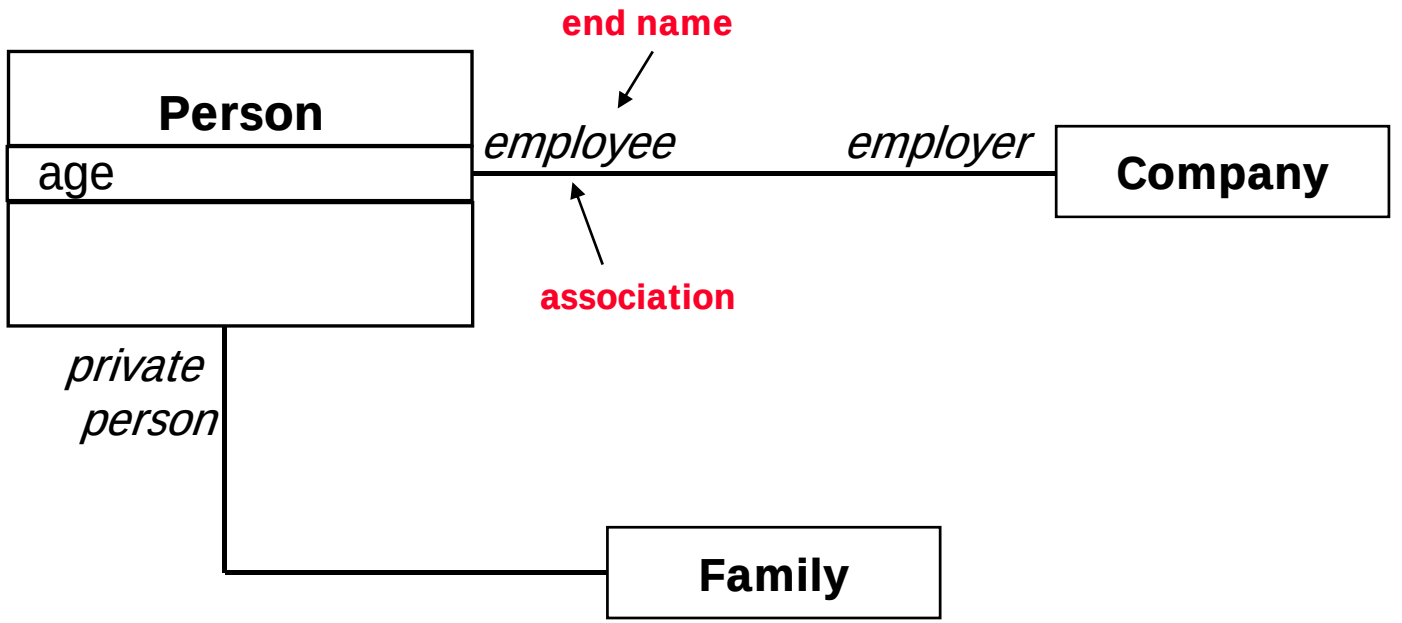
One-to-many association

Class Diagrams
Many-to-Many Associations

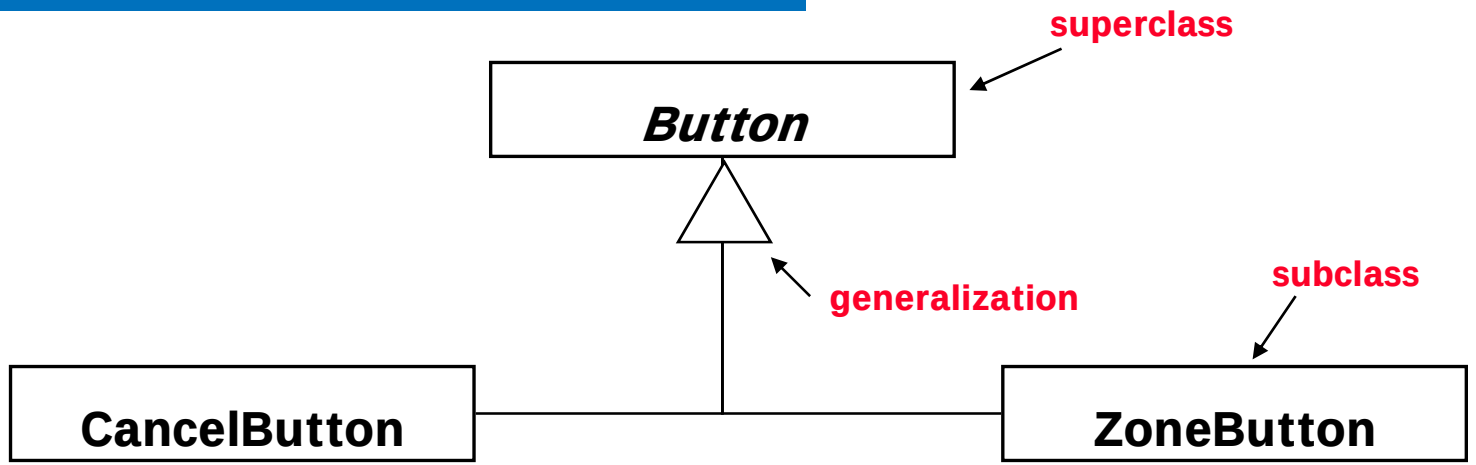
Many-to-many association

Class Diagrams
Roles

- The purpose a class plays in an association can be described by role.
- Useful when a class is associated with many classes

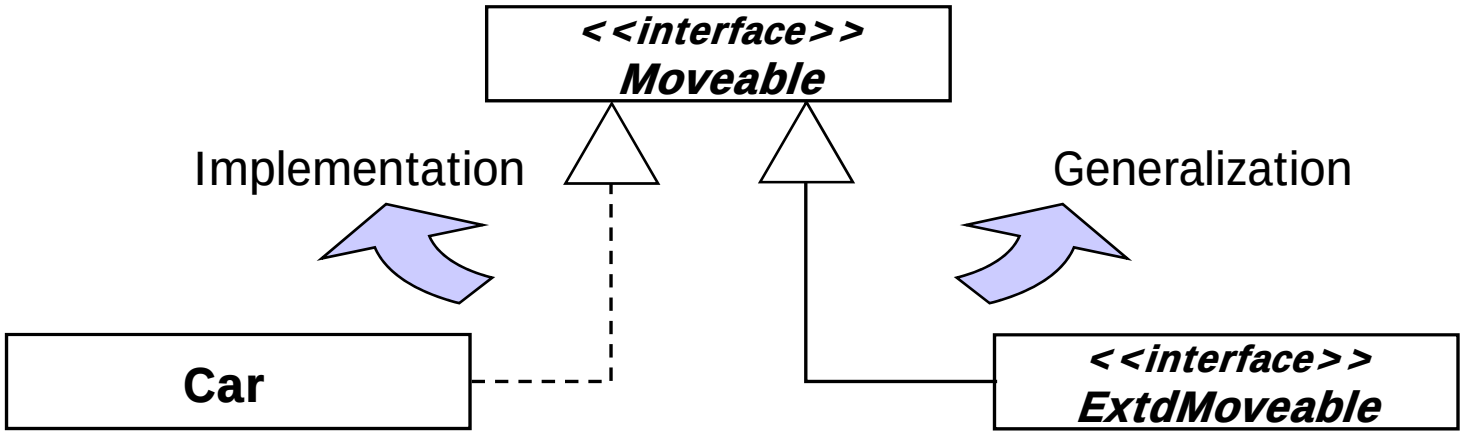


Class Diagrams
Generalization (is-kind-of)



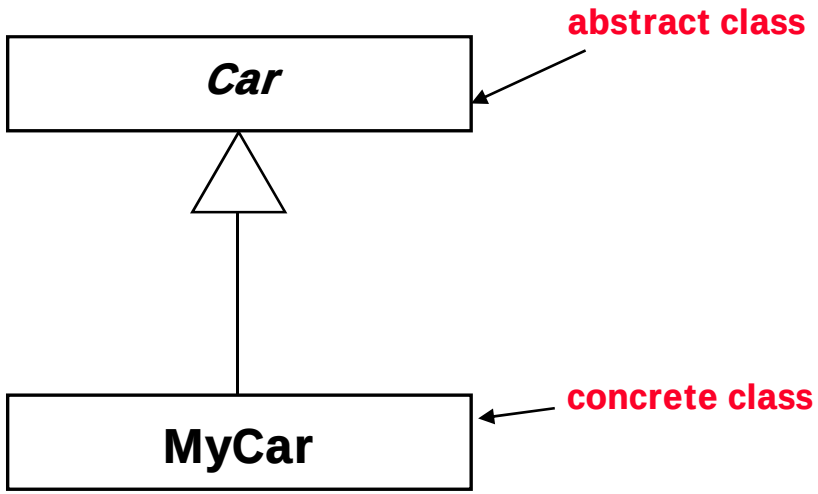
- An generalization is relationship between a general class (superclass, parent) and a more specific kind of things (subclass, child)
- Subclasses inherit the attributes and operations of the parent superclass
- Subclass may implement further attributes and methods
- If a new method A of a subclass has the same signature as a method of the superclass, then the implementation of the new method overrides the implementation of the superclass (*polymorphism*)

Class Diagrams
Inheritance of Interface



- Interfaces are classes that define a set of externally accessible operations without implementations → contract for classes
- Implementation classes must provide concrete implementation
  - Otherwise: Abstract classes
- Stereotype: Used for extending UML elements

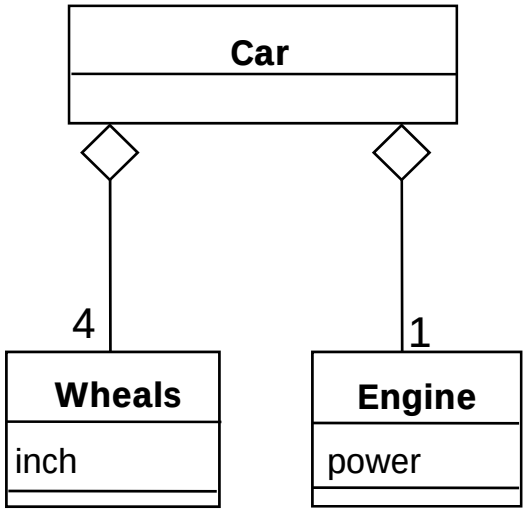
Class Diagrams
Abstract classes



- Abstract class contains both implemented methods and interface information of methods (no implementation given)
- Can’t be instantiated directly
- Subclass must implemented interface methods and may override implemented methods
- If some interface methods are not implemented, then the subclass is again an abstract class.

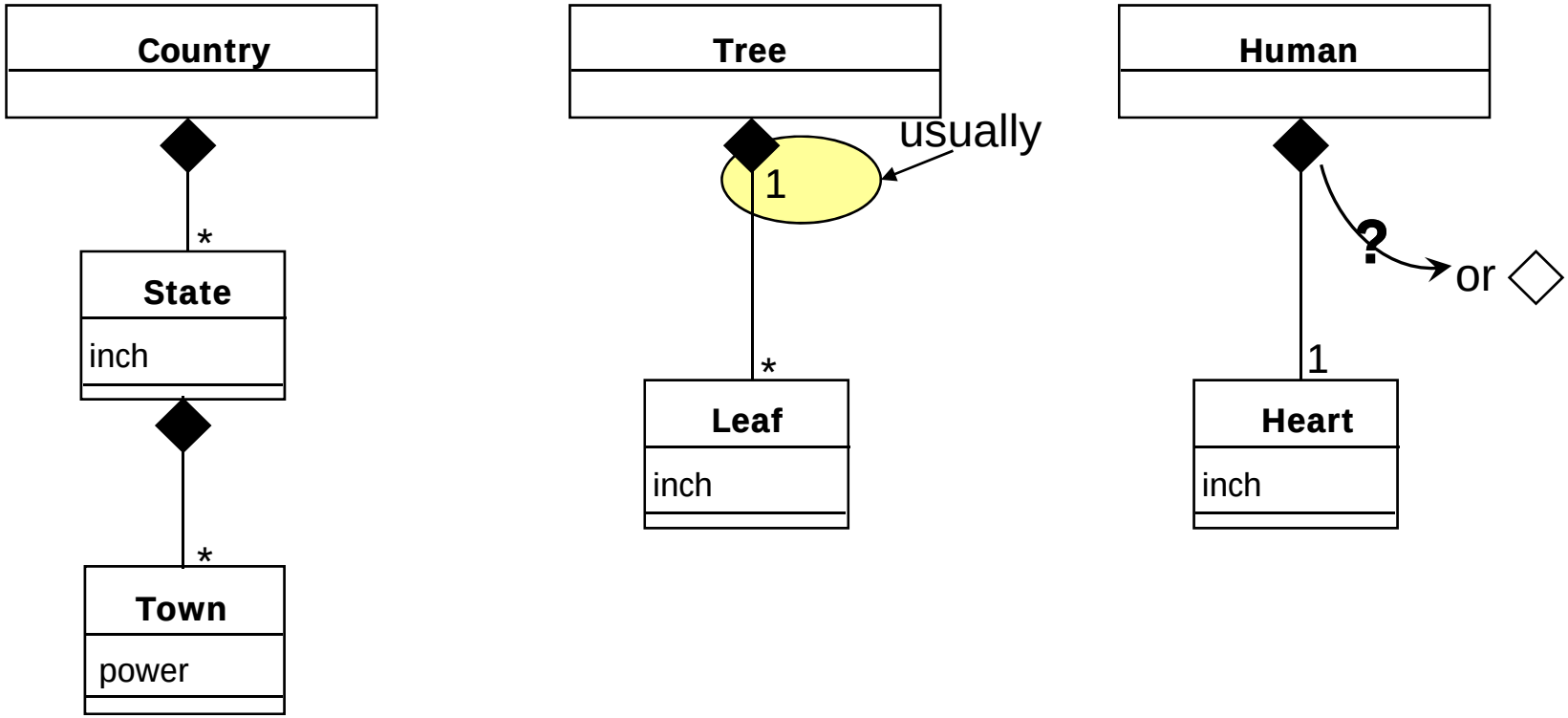
Class Diagrams
Aggregation

- An aggregation is a special case of association denoting a “is-part of” hierarchy
- The aggregate (whole) is the parent class, the parts are the children class
- Aggregate and its part can exist independently
- Parts can be rebound to other aggregates



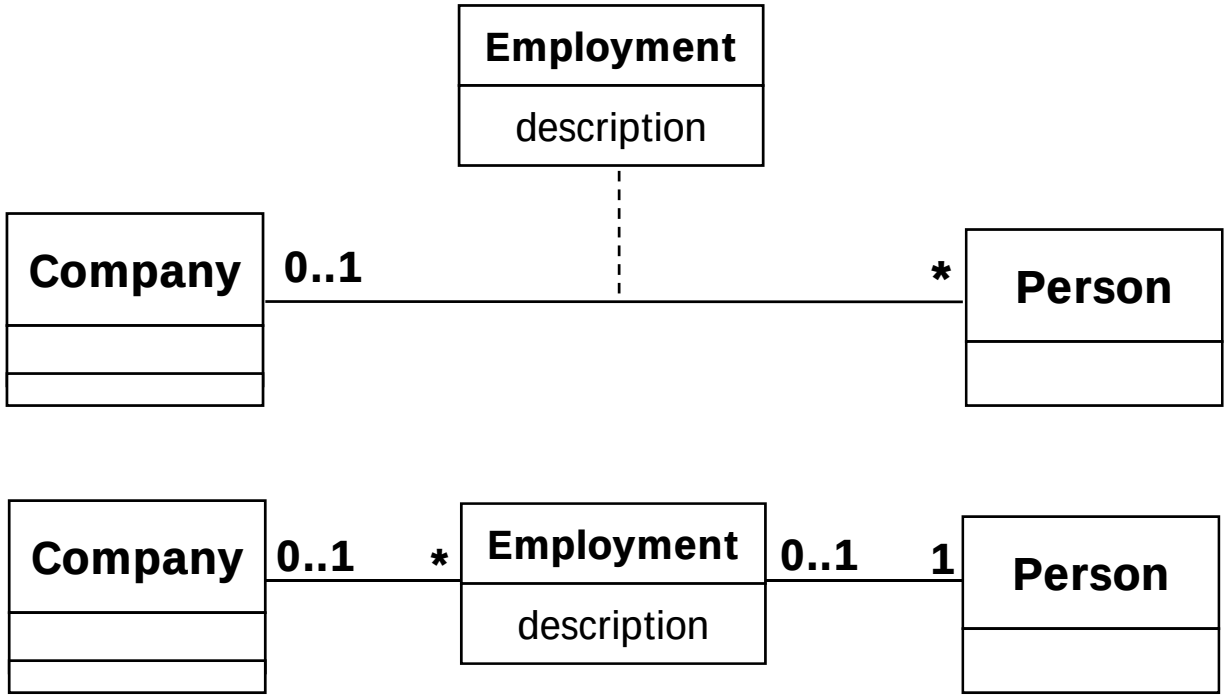
Class Diagrams
Composition

- A solid diamond denotes composition, a strong form of aggregation where parts cannot exist without the aggregate
- Deletion of the aggregate causes the deletion of its parts
- Parts cannot be rebound to other aggregates
- Usage is not always obvious → Usage should be argued



Class Diagrams
Association Classes

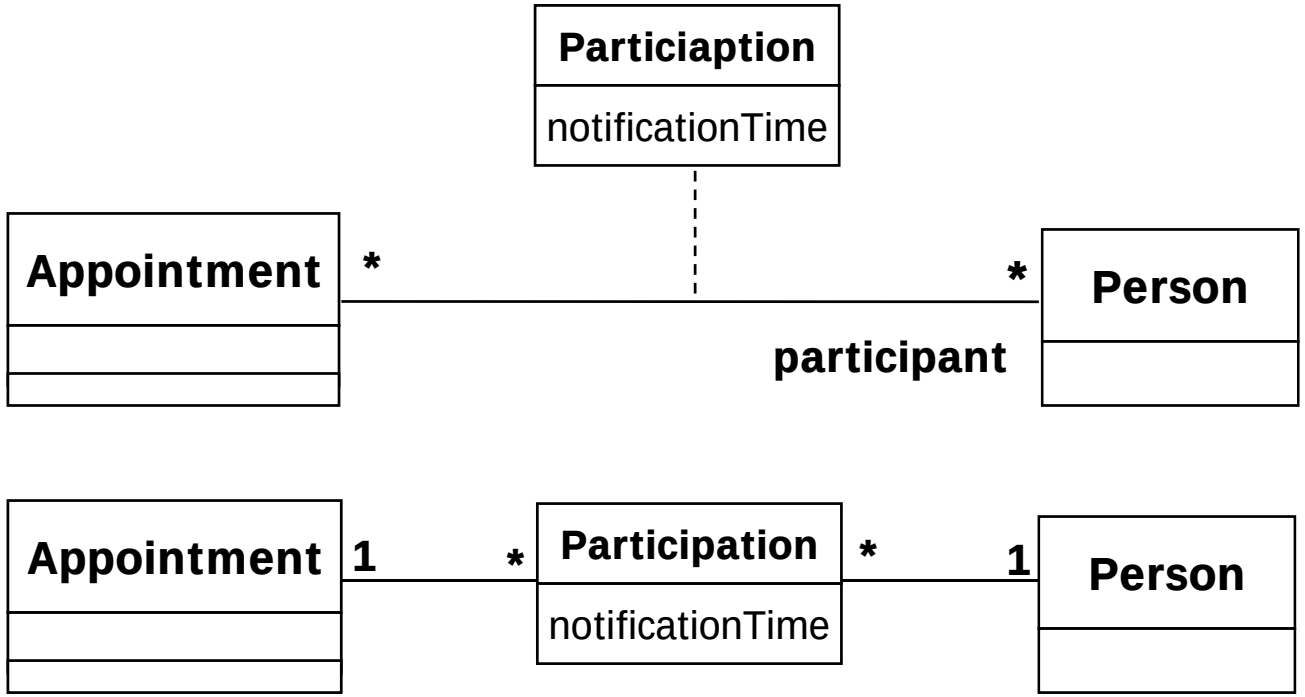
- Attributes and operations can be attached to an association (class)
- Association class cannot exist without the classes it links



→ **Alternative modeling: differences?**

Class Diagrams
Association Classes

- And this case?



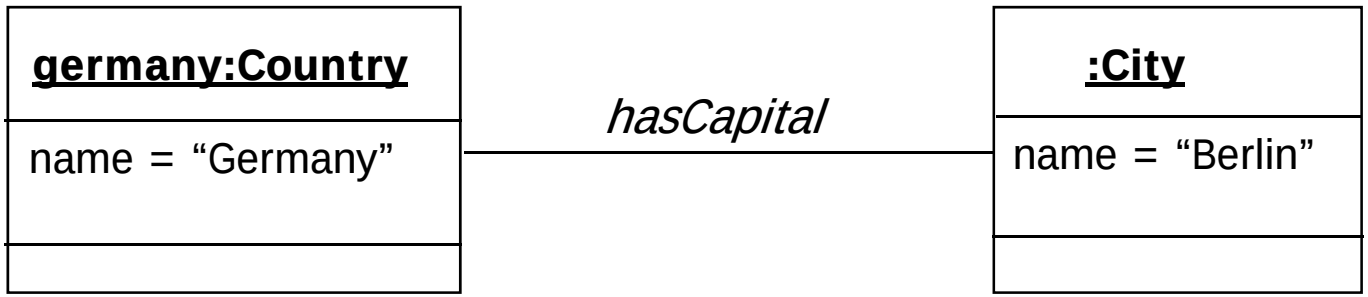
Different Semantics:
in the second variant a Person may be linked twice with the same Appointment

UML Diagrams

- Use case Diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Classes, Attributes, Associations
- Object diagrams
  - Show objects and their relations at a point in time.
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object (essentially a finite state automaton)

Object Diagram

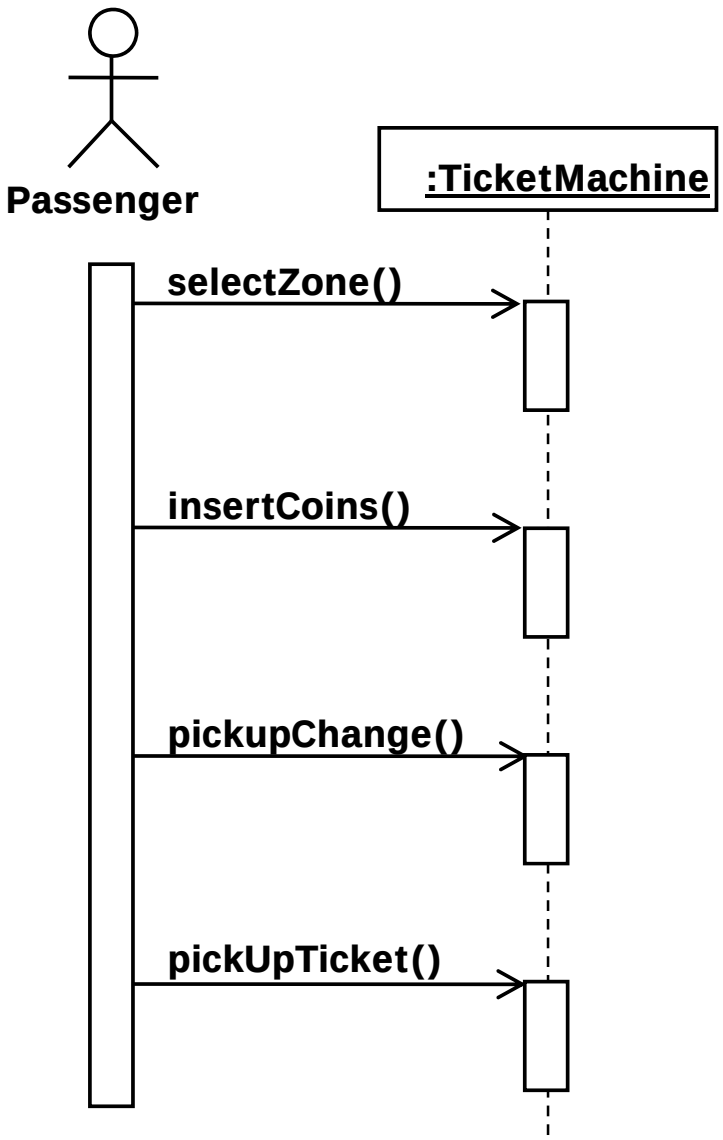
- Describe a snapshot of a system a certain point in time.
- Object graph with concrete attributes
- Useful to illustrate complicated data structures
- Object diagrams are only examples, not the definition of a system
- Name: <optional instance name>:<Class Name>
- Name is underlined



UML Diagrams

- Use case Diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Classes, Attributes, Associations
- Object diagrams
  - Show objects and their relations at a point in time.
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object (essentially a finite state automaton)

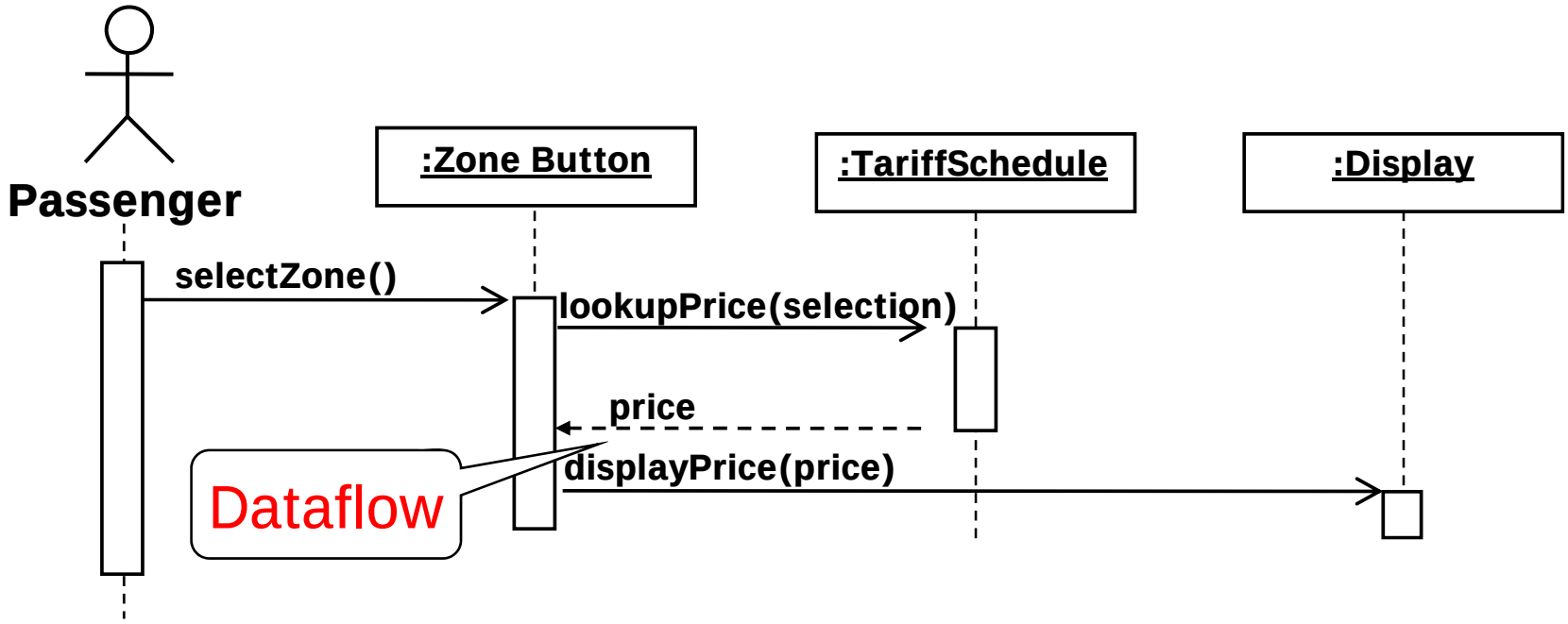
Sequence diagrams



- Used during requirements analysis
  - To refine use case descriptions
  - to find additional objects ("participating objects")

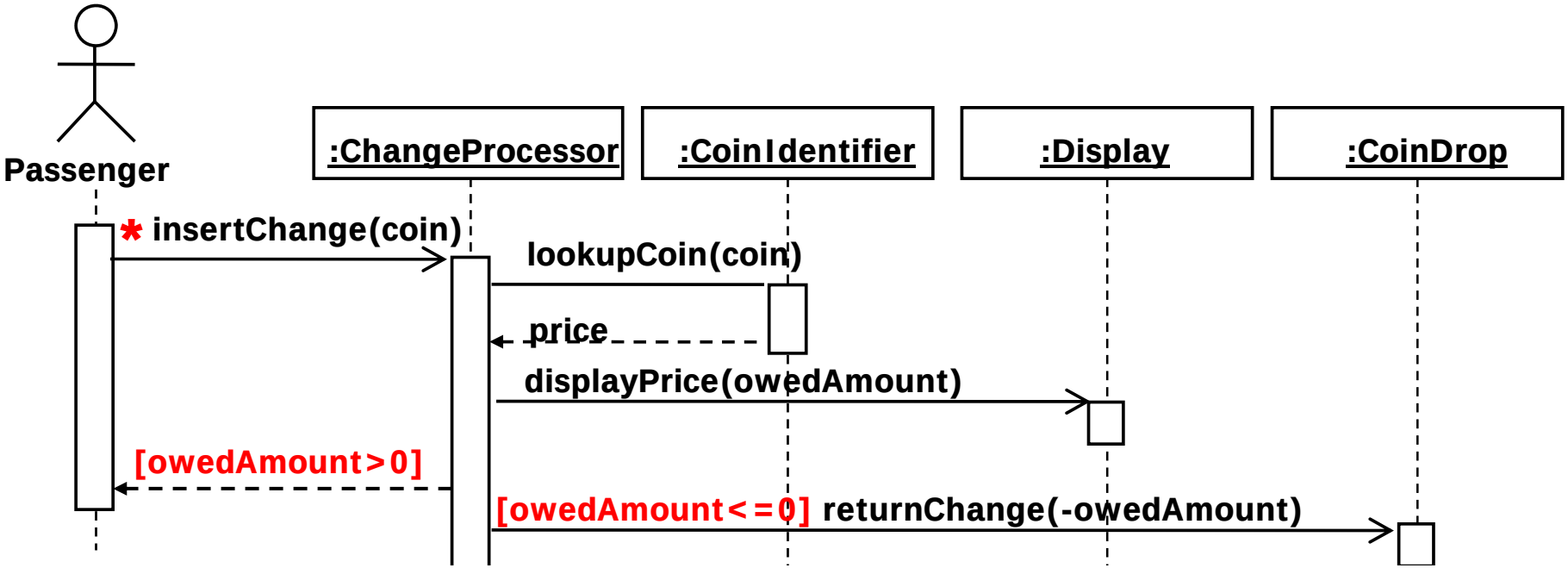
- ***Class roles*** represent a specific part involved in the collaboration (*objects, actors*)
- ***Messages*** represent information that is exchanged during interactions (*labeled solid arrows*)
- ***Activations*** to represent the time during which as class is performing an action or when it is active and has focus on control (*vertical bars*)
- ***Lifelines*** represent the existence of a role over a period of time (*dashed lines*)

Sequence diagrams:
Nested messages



- Horizontal dashed arrows indicate data flow
- Stick arrowhead: flat flow of control:
  - Control is passed to receiver
  - No further details about communication are provided
- Diagram uses the **Instance Form** of a Sequence Diagram
  - *Concrete* Sequences visualize *one possible* interaction
  - Describes one actual scenario
  - The source of an arrow indicates the activation which sent the message

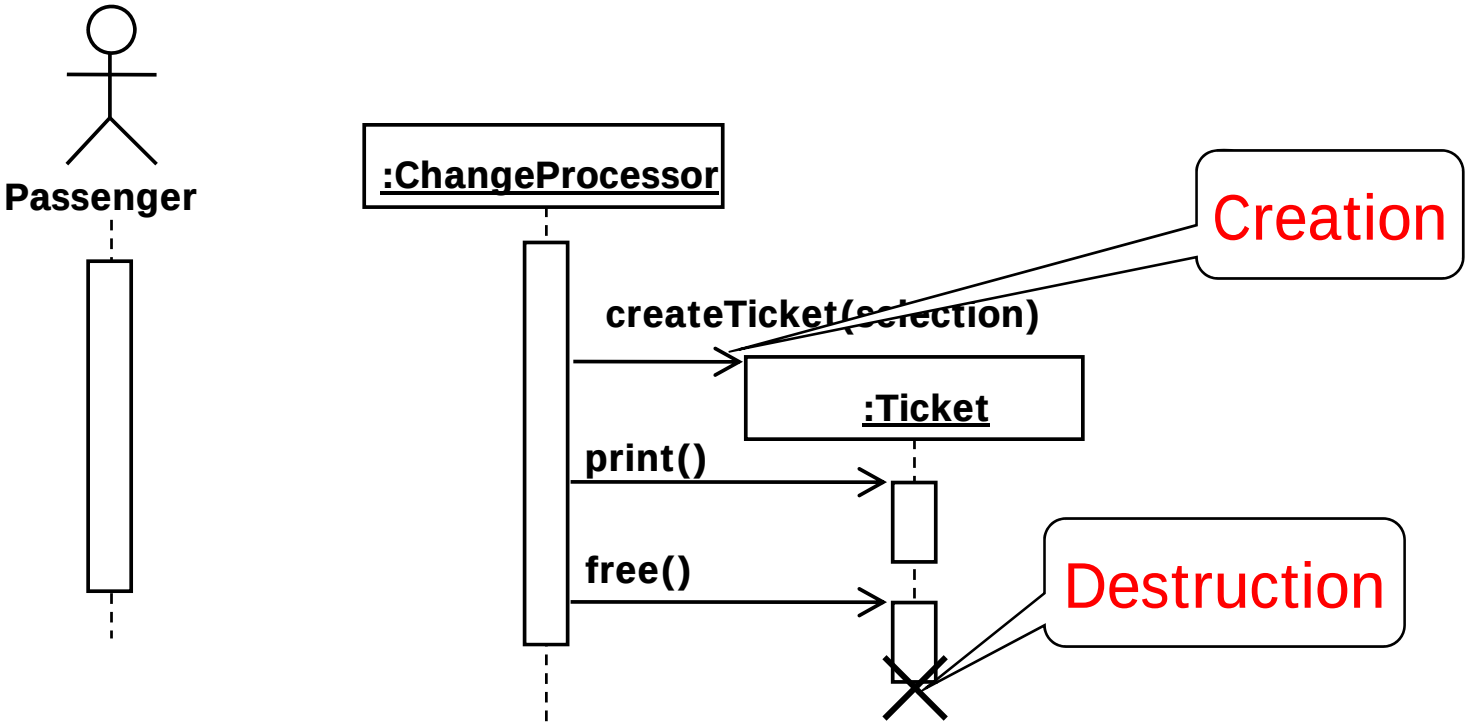
Sequence diagrams:
Iteration & Condition



...to be continued...

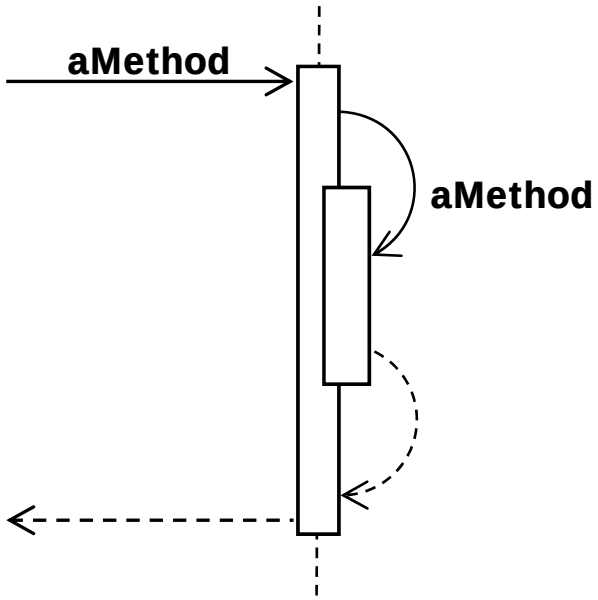
- This example uses the **Descriptor Form**
  - *Abstract* Sequences visualize *all possible* interactions
  - Describes all possible scenarios
  - Iteration is denoted by a * preceding the message name
  - Condition is denoted by boolean expression in [ ] before the message name

Sequence diagrams:
Creation and destruction



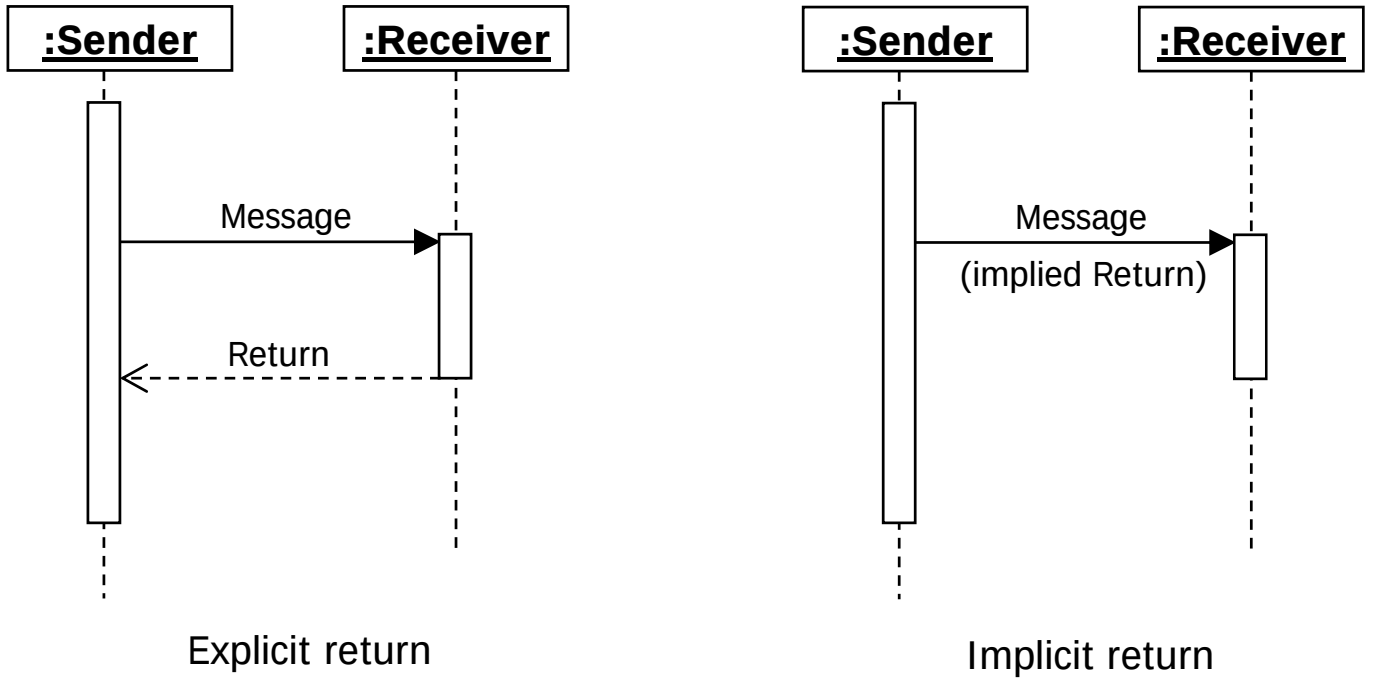
- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
- In garbage collection environments, destruction can be used to denote the end of the useful life of an object

Sequence diagrams:
Self-Delegation



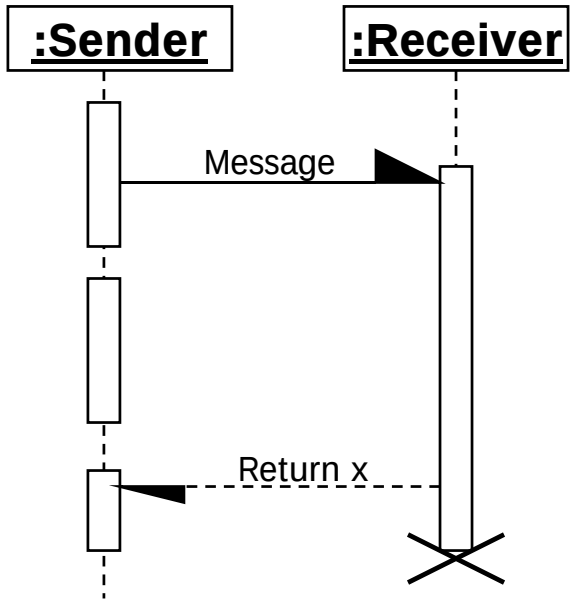
- An operation may invoke itself recursively (self-delegation)
- Nesting may occur to an arbitrary depth

Sequence diagrams:
Synchronous interaction



- Represent a nested control flow via an operation call. The operation call invokes an operations synchronously (filled arrowhead)
- The sender passes control to the receiver via the message and pauses to wait for the receiver to return control (implicitly or explicitly)
- Nested sequences complete before their outer sequence resume

Sequence diagrams:
Concurrency (Asynchronous interaction)



- Represent a non-nested flow of control via a message. The message invokes an operation asynchronously (half-filled arrowhead).
- Both Objects stay active, can communicate with each other
- If the receiver returns information to the sender, the information must be explicit

UML Diagrams

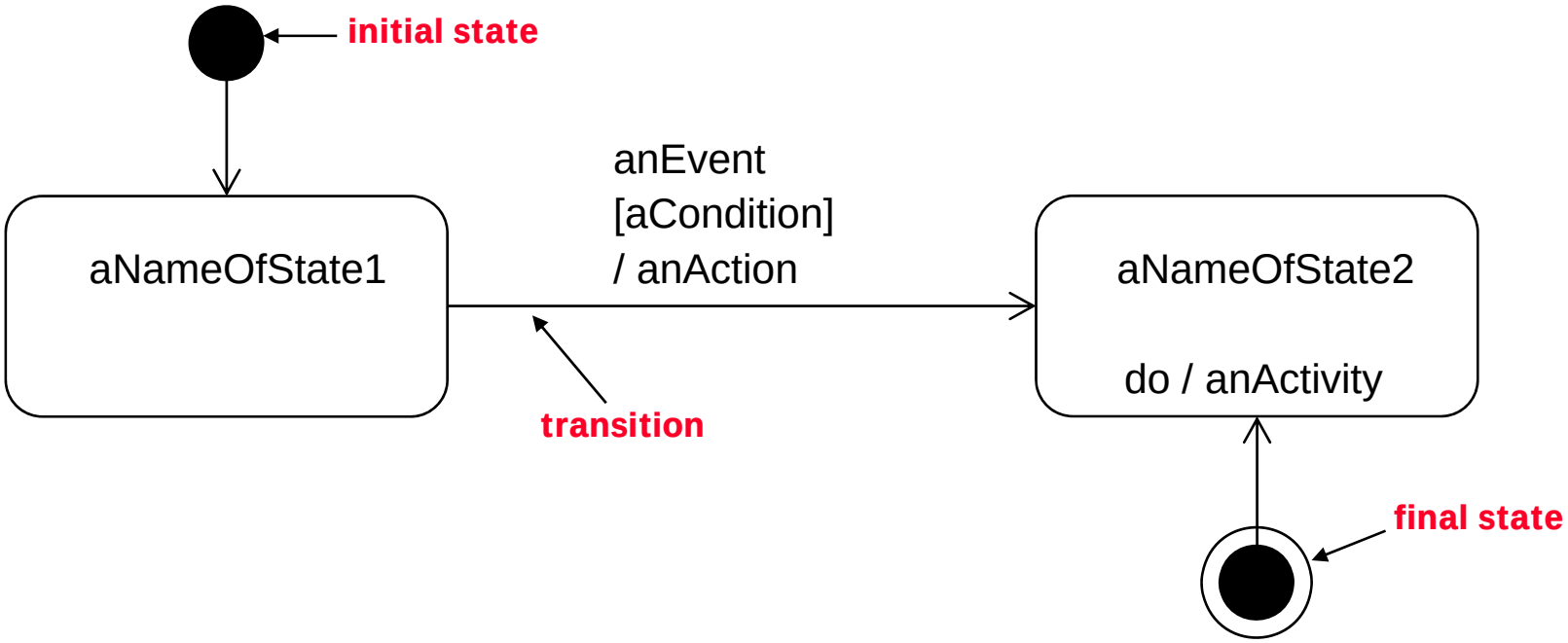
- Use case Diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Classes, Attributes, Associations
- Object diagrams
  - Show objects and their relations at a point in time.
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object (essentially a finite state automaton)

State Diagrams
Overview

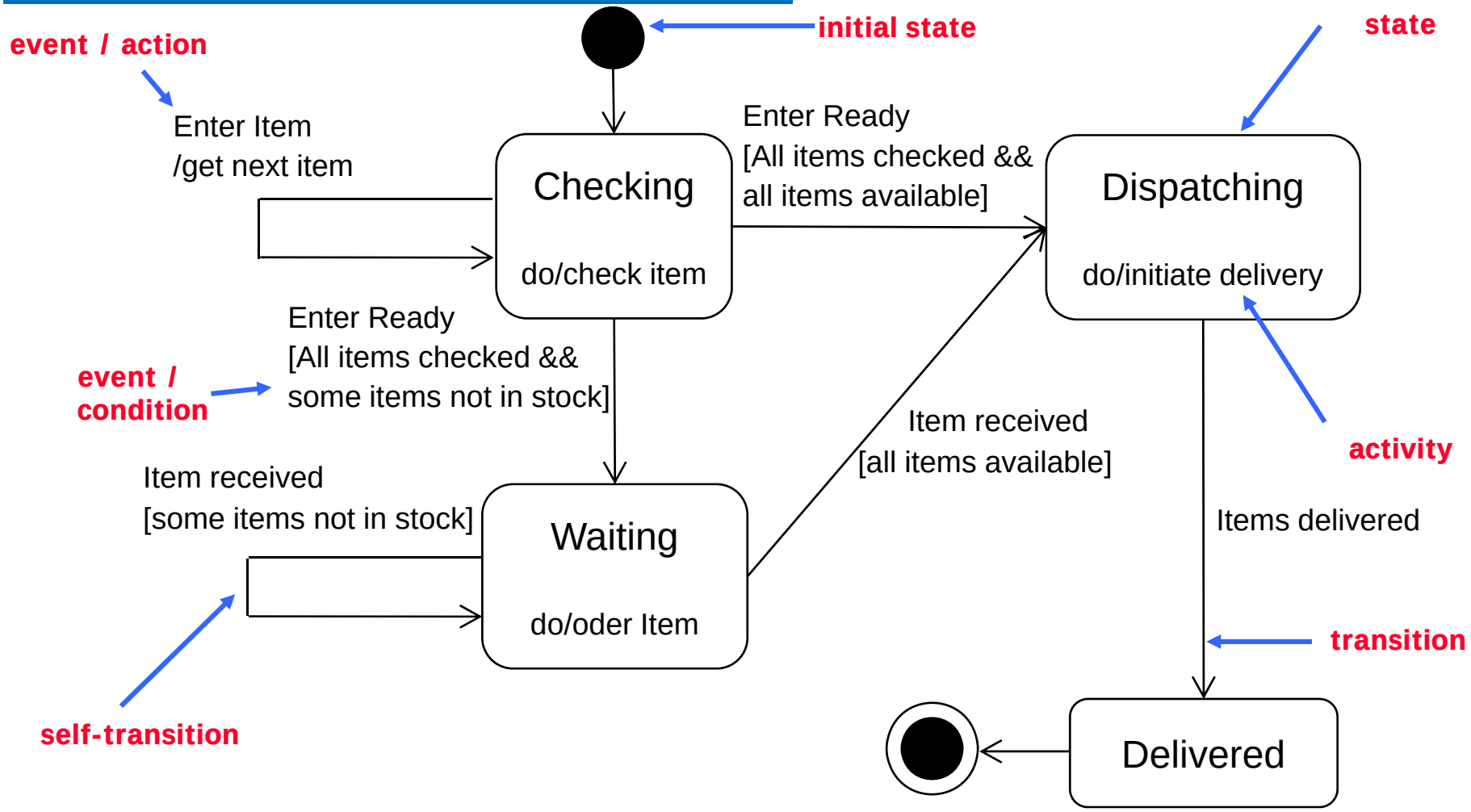
- Modeling the behavior of one single object
- State machine
  - Behavior that specifies the sequences of states an object goes through during its lifetime in response to events
- States
  - Condition or situation during the lifetime of an object during which it satisfies some condition, performs some activity, or waits for events
  - Can have concurrent and sequential sub states
- Event
  - A significant occurrence that has a location in time and space
- Transition
  - A relationship between two states
  - Adorned by conditions, event, actions (some executable computation)

State Diagrams
Basic Model

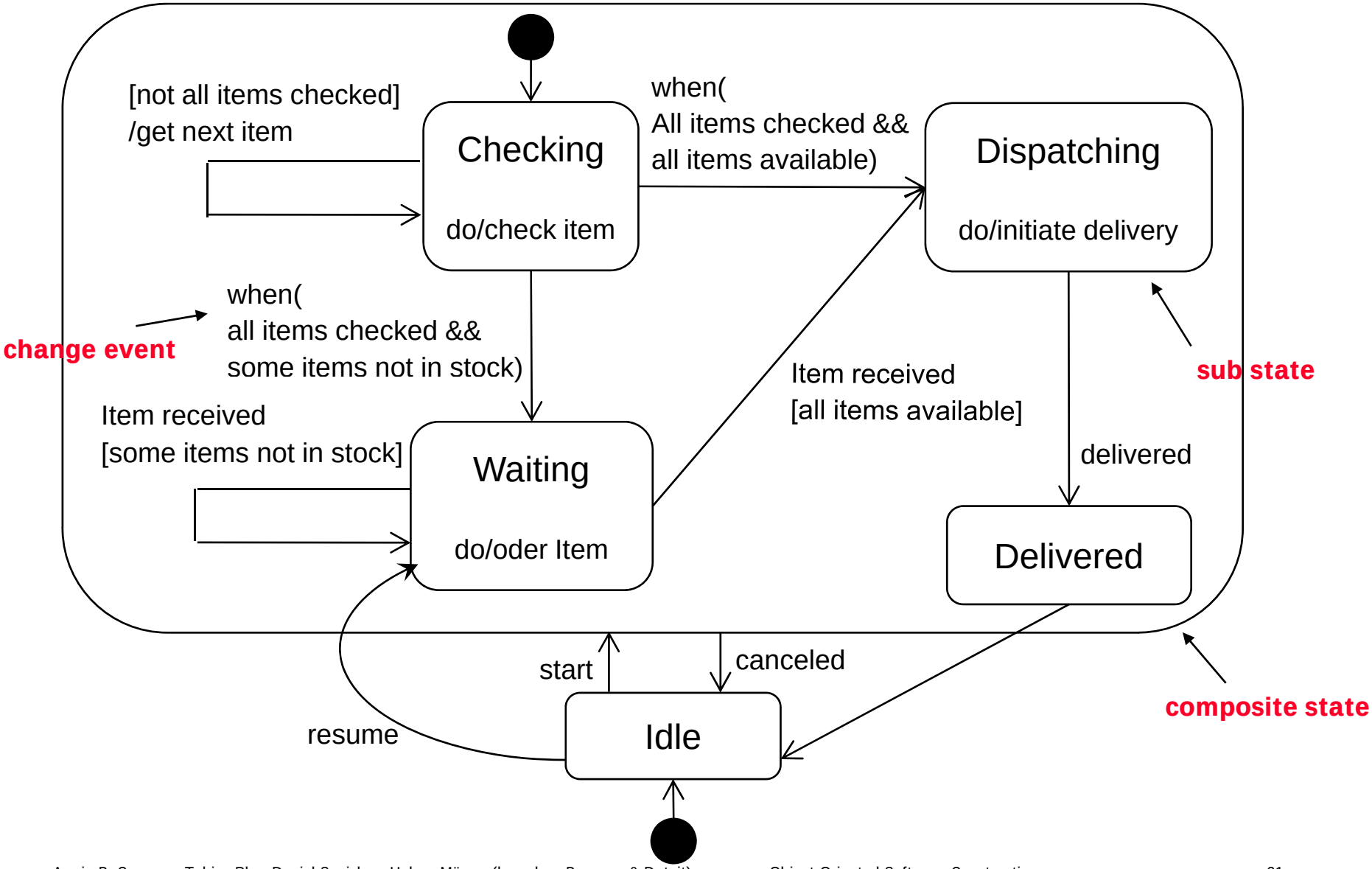
- An object in the first state will perform certain action and will enter the second state
  - When a specific event occurs
  - And specified additonal conditions are satisfied



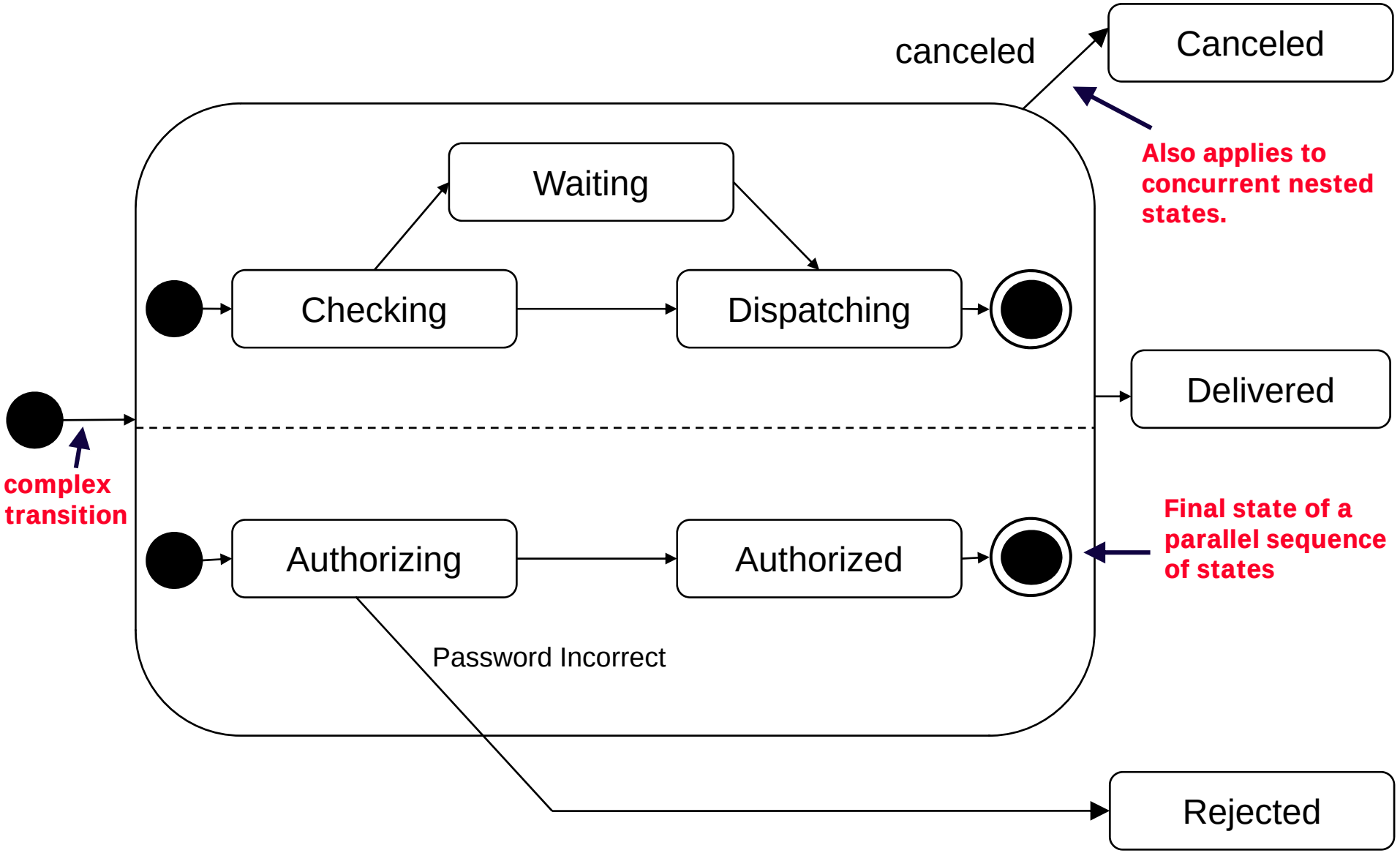
State Diagram
Simple Example



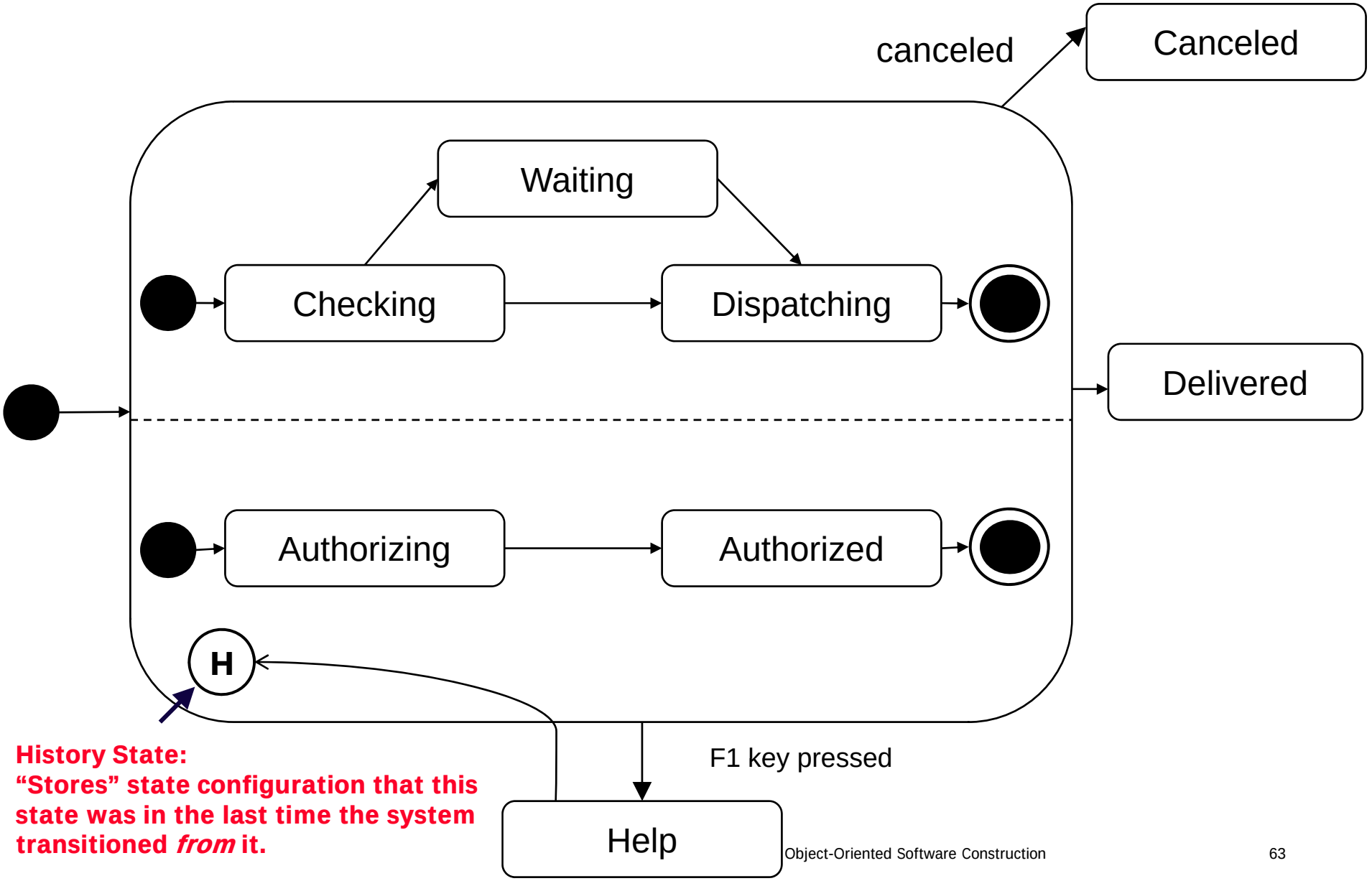
State Diagram
Nested States and Change Events



State Diagram
Concurrent and States Complex Transitions



State Diagram
History States



History State:
"Stores" state configuration that this state was in the last time the system transitioned *from* it.

Summary

- Idea of models
- Introduction into UML (Overview)
  - Functional model
  - Object model
  - Dynamic model