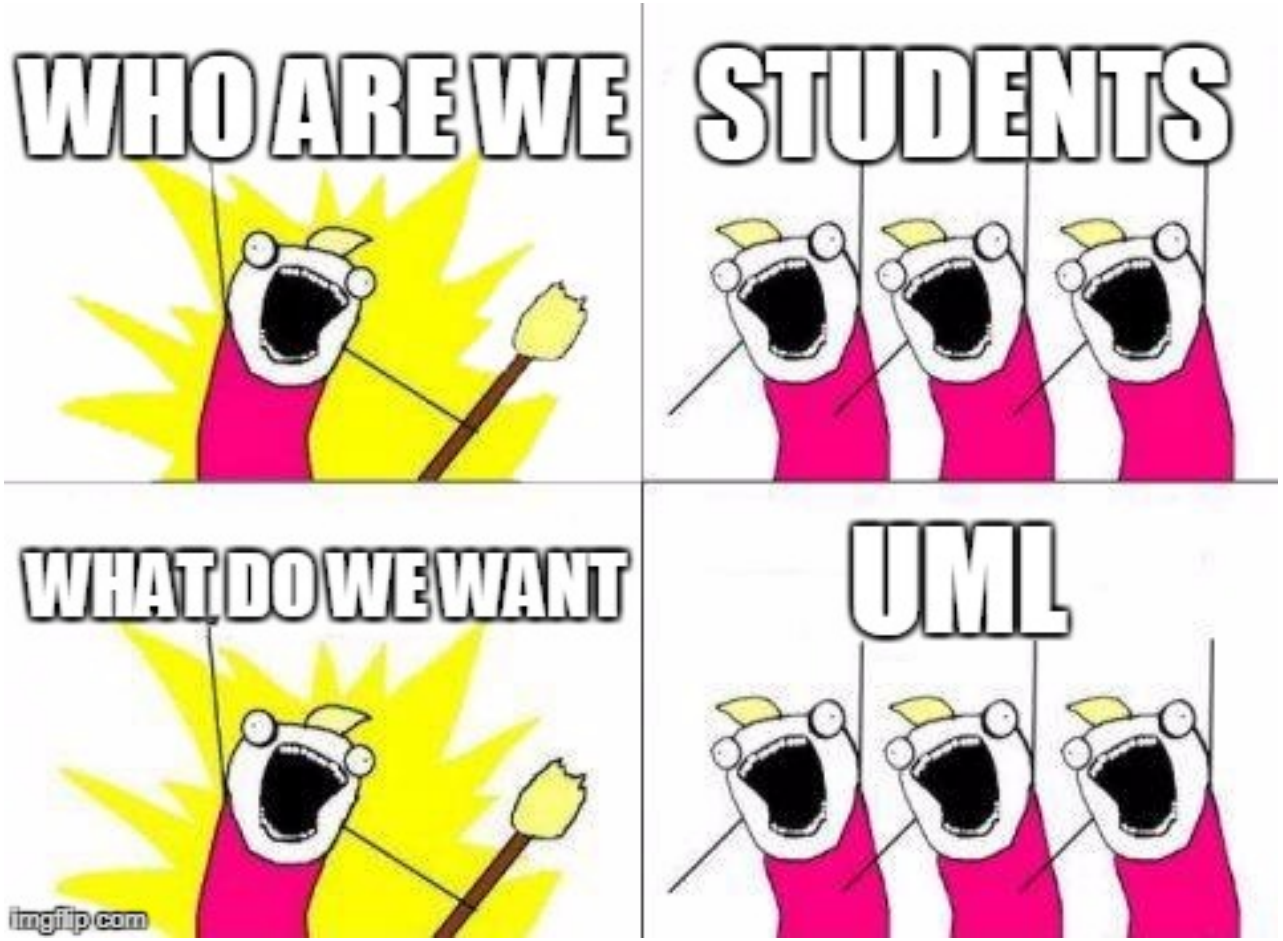
Unified Modeling Language:
Introduction and Overview

Autumn 2017

Design of Software Systems

Assoc. Prof. Dr. Marco Kuhrmann, Elena Markoska


WHO ARE WE
STUDENTS
WHAT DO WE WANT
UML
imgflip.com

UML – History/Overview

- UML → Unified Modeling Language
- Published and maintained by the Open Management Group: http://www.omg.org

- Original purpose: unification of the plethora of different modeling languages in Software Engineering (inflation mid 1990's)

- UML is:
  A standardized visual modeling language aiming at the specification, construction, and documentation of (object-oriented) software systems

- UML is **not**:
  A method/an approach to describe the software development process.

UML- Overview

UML Diagrams

Structure Diagrams

- Class diagram
- Package diagram
- Object diagram
- Component diagram
- Composition (structure) diagram
- Deployment diagram

Behavior Diagrams

- Use Case diagram
- Activity diagram
- State machine

Interaction Diagrams

- Sequence diagram
- Timing diagram
- Communication diagram
- Interaction overview diagram

UNIFIED MODELING LANGUAGE:

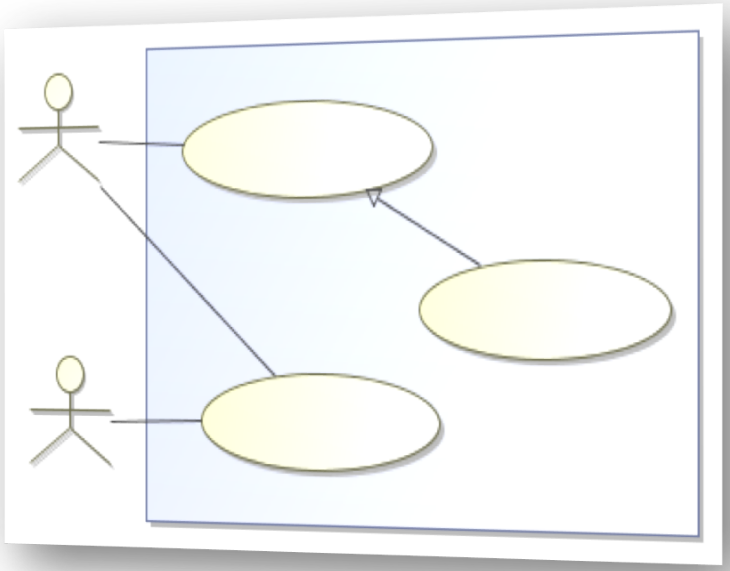
USE CASE DIAGRAMS
CLASS DIAGRAMS
COMPONENT DIAGRAMS
SYSTEM DIAGRAMS
STATE DIAGRAMS

UNIFIED MODELING LANGUAGE:
USE CASE DIAGRAMS

Design of Software Systems

Use Case Diagram

- A use case diagram describes the external behavior of a system from the perspective of a use
- A use case diagram illustrates use cases and relationships between them
- Often used in the analysis phase of a project together with text templates for process refinements
- Relevant notation elements
  - Actor
  - Use Case
  - Relationship
  - System Border



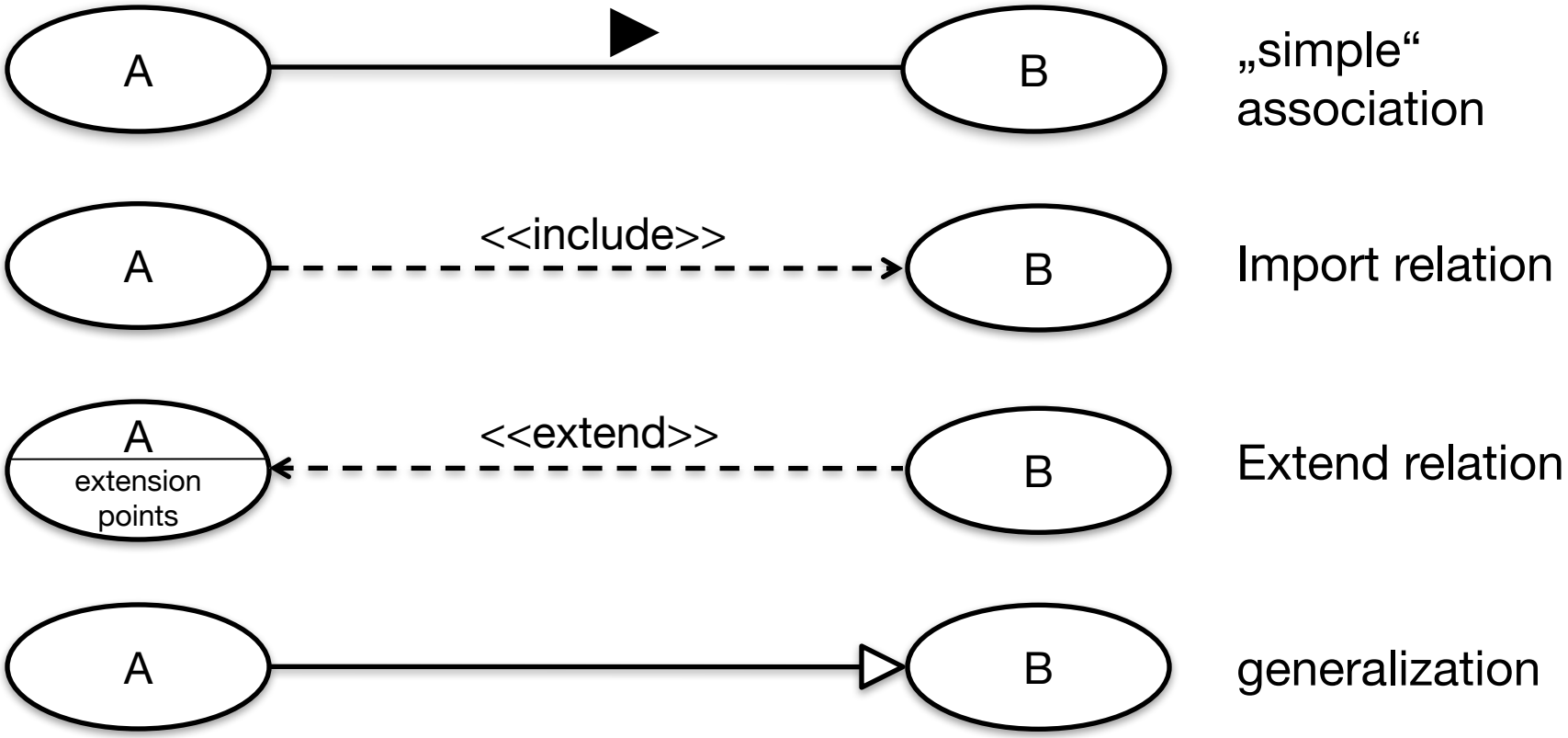
Use Case Diagram – Terminology/Definitions

- **Actors** interact with a system, trigger use cases, or participate. Actors are roles that are recognized by persons/systems outside the “visible” system borders.

- A **Use Case** describes a set of interactions between a system and its environment, which are – stepwise executed – form a specific behavior.

- Use cases as well as use cases and actors are embedded into a dependency network (**relationships**), e.g., actors are assigned to use cases, and use cases can depend on each other.

Use Case Diagram – Notation Elements



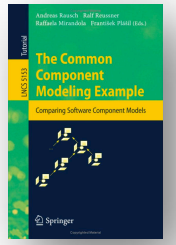
Use Case Diagram – Terminology/Definitions

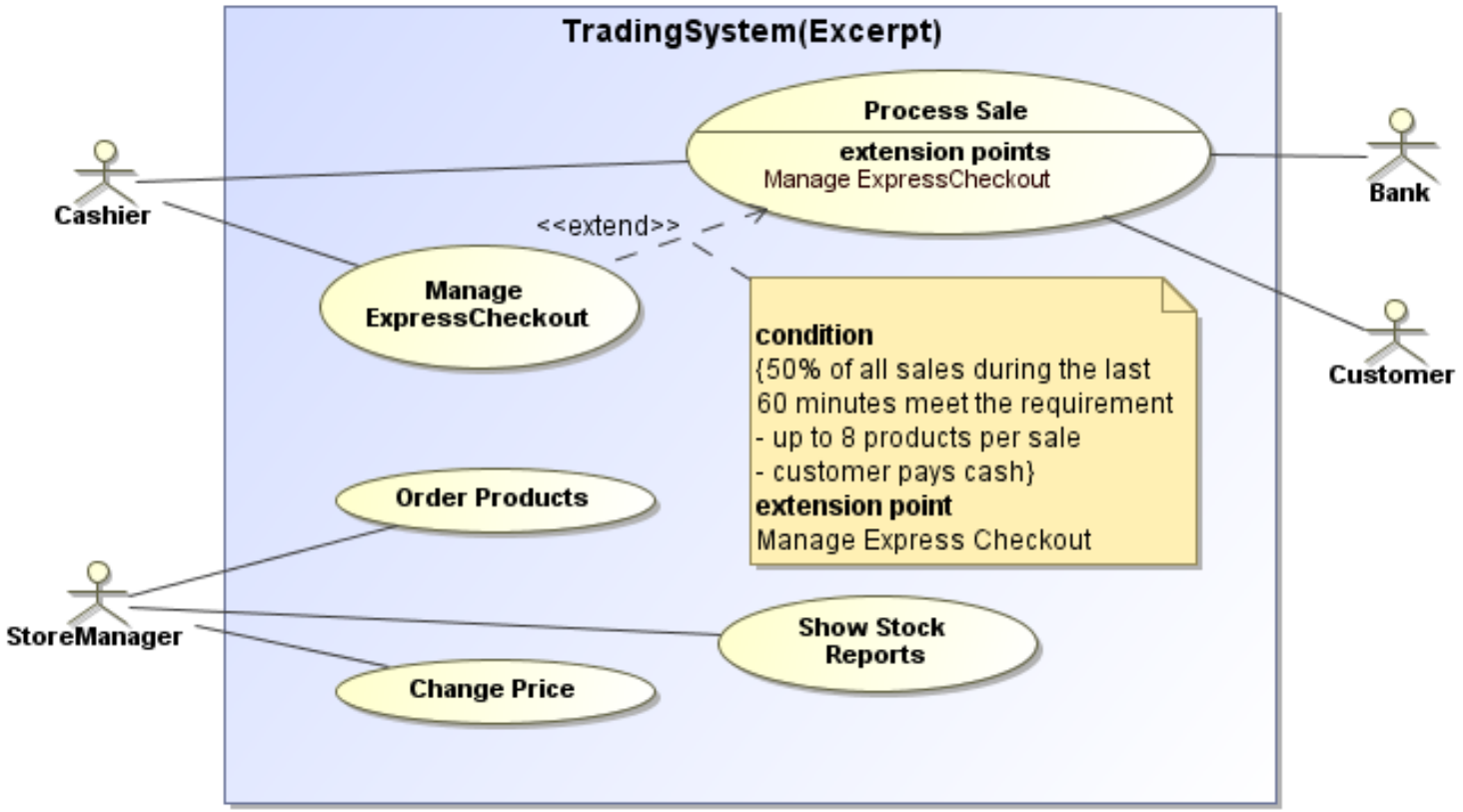
- Relation type: **Include**
  - Use case A “imports” the behavior of use case B
  - In every run of A, the execution of B is included (A is superset of B)
  - Used when: B occurs in multiple use cases (encapsulation)

- Relation type: **Extend**
  - Use case B “extends” the behavior of use case A
  - At least on procedure from A includes the behavior of B
  - Used when: to model “special” cases

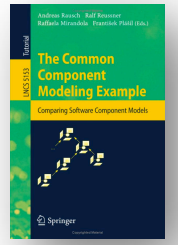
- Relation type: **Inherit**
  - Use case A refines the behavior of use case B

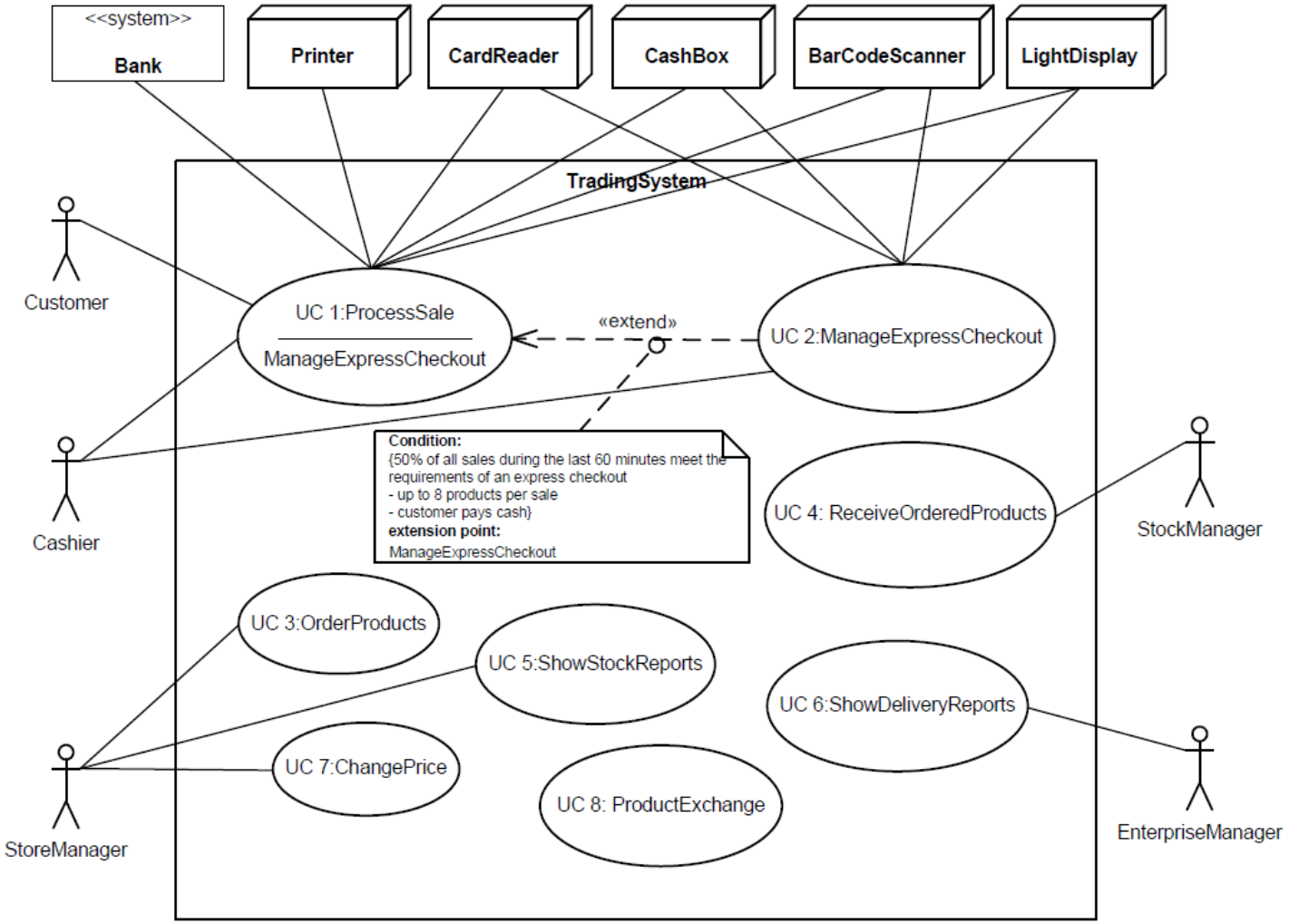
Example


The Common Component Modeling Example
Springer


TradingSystem(Excerpt)
Process Sale
extension points
Manage ExpressCheckout
<<extend>>
Manage ExpressCheckout
condition
{50% of all sales during the last 60 minutes meet the requirement
- up to 8 products per sale
- customer pays cash}
extension point
Manage Express Checkout
Order Products
Show Stock Reports
Change Price
Cashier
StoreManager
Bank
Customer

Example


The Common Component Modeling Example
Springer

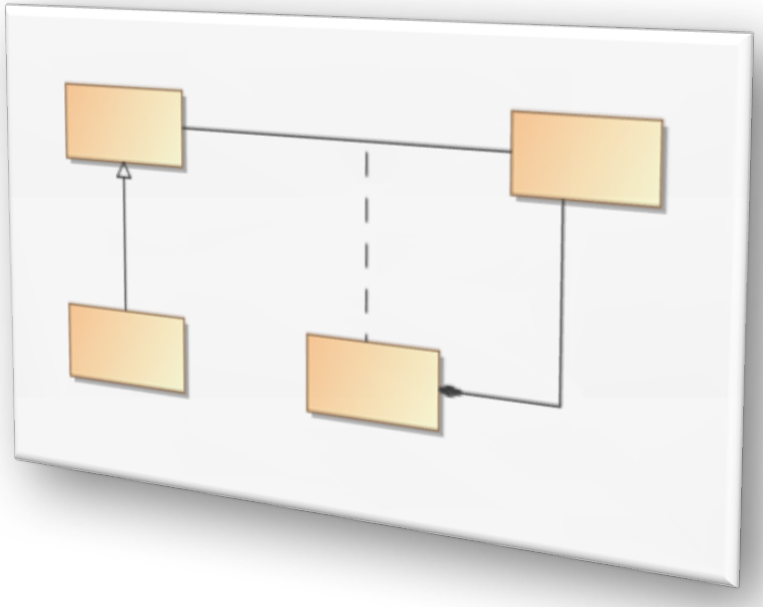

<<system>>
Bank
Printer
CardReader
CashBox
BarCodeScanner
LightDisplay
TradingSystem
Customer
Cashier
StoreManager
StockManager
EnterpriseManager
UC 1:ProcessSale
ManageExpressCheckout
«extend»
UC 2:ManageExpressCheckout
Condition:
{50% of all sales during the last 60 minutes meet the requirements of an express checkout
- up to 8 products per sale
- customer pays cash}
extension point:
ManageExpressCheckout
UC 4: ReceiveOrderedProducts
UC 3:OrderProducts
UC 5:ShowStockReports
UC 6:ShowDeliveryReports
UC 7:ChangePrice
UC 8: ProductExchange

UNIFIED MODELING LANGUAGE:
CLASS DIAGRAMS

Design of Software Systems

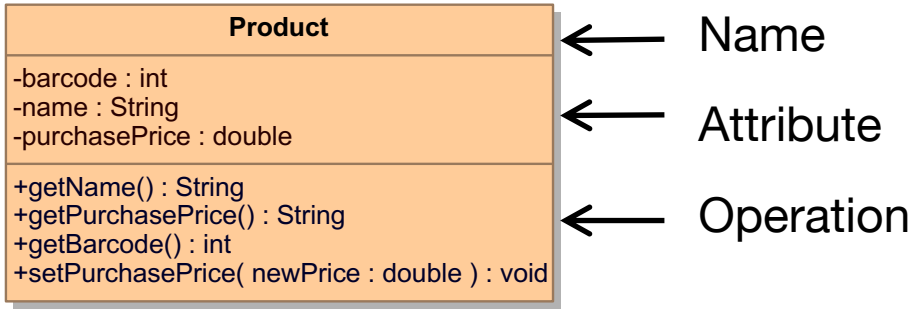
Class Diagrams

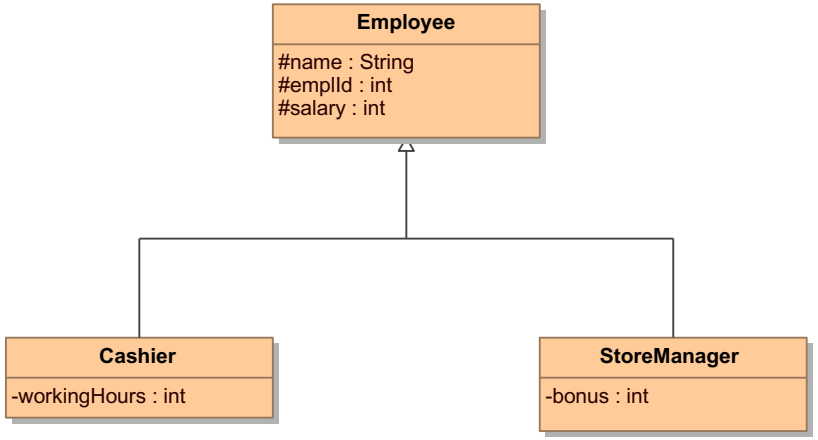
- System/software components are often modeled using component or class diagrams
- A **class** is the key element in OOA/D
- UML Class Diagrams focus on modeling the static structure
  - Attributes and operations
  - Relationships between classes
- Relevant notation elements
  - Class
  - Association
  - Generalization
  - (Package)



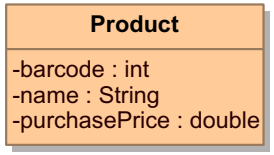
Class Diagram – Notation

- Classes have a name, a set of attributes, and a set of operations

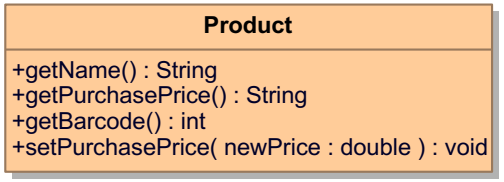




Operations can be hidden



…as well as attribues…

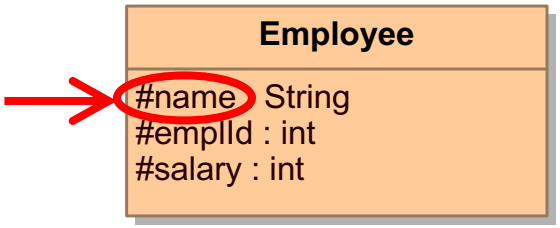


…or both!

Class Diagram – Attributes and Visibility

- Attributes have a visibility → important for, e.g., generalization
- Private attributes: scope is the defining class
  - Invisible (non-accessible) in other classes
  - Not even in children of the defining class
- Public attributes are visible everywhere…
  - Violates the data encapsulation principle
  - Avoid!
- Something in the "middle": protected:

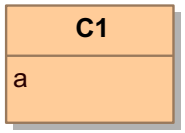
Visibility: „#" = „protected"
That is, the attribute is only visible in the same class or children thereof…

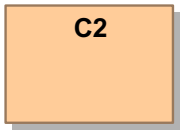


Note: modern object-oriented languages implement visibilities…

Class Diagrams – Visibility Overview

- Given the classes C1 and C2
- And an attribute a in C1





- Visibility/accessibility is given when:

…and for C2:

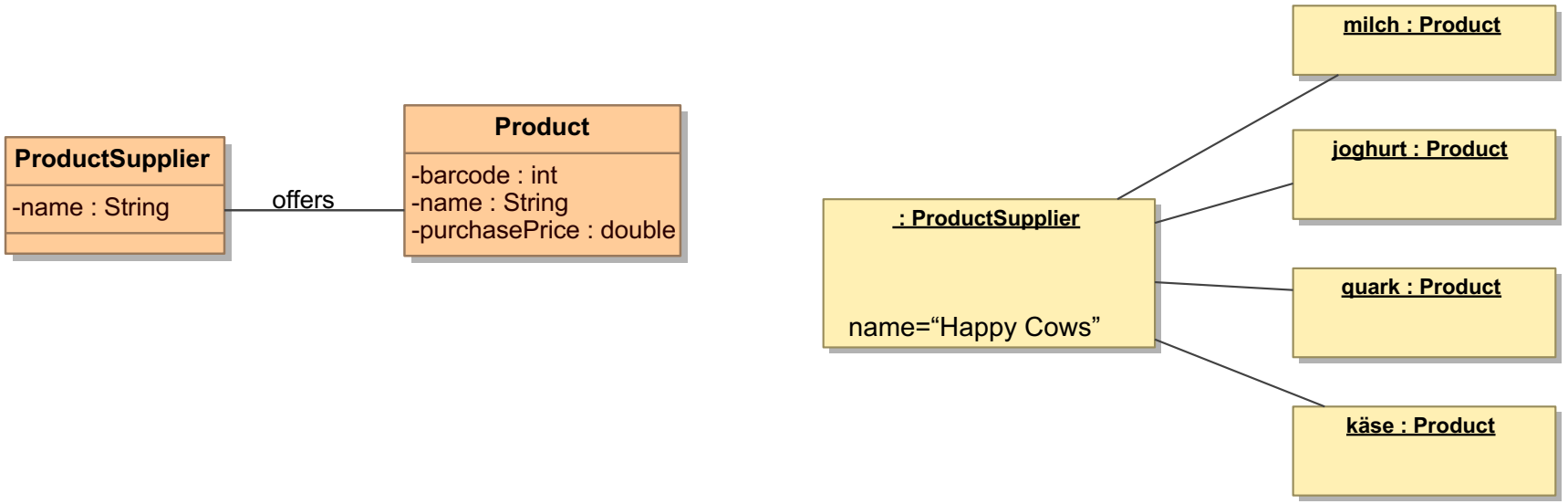
a is declared…

Visibility	Symbol	C2 = C1	C2 is child of	C2 is defined in the same package	C2 is somewhere
private	-	✓			
protected	#	✓	✓		
package	~	✓	✓	✓	
public	+	✓	✓	✓	✓

Class Diagrams – Associations

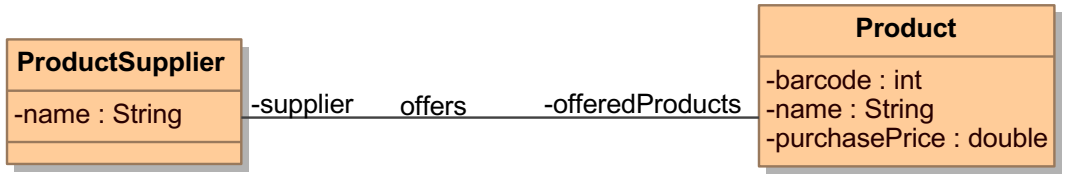
- Classes are usually embedded into a complex network of dependencies → expressed by UML associations

- Example:
  - every “Product” is provided by a “Supplier” from which a supermarket can order the products
  - There is an explicit association between both classes

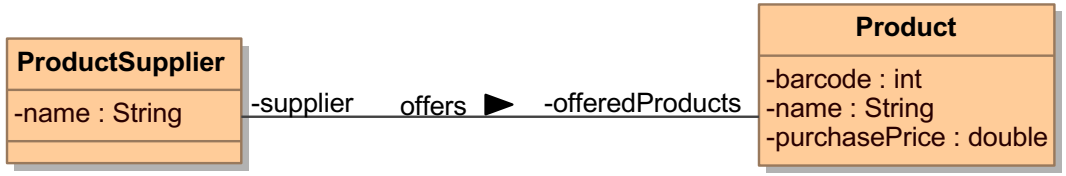


Class Diagrams – Associations: Notation

- Associations can be **Named**
- Association ends can have **Role** names (including visibility)
  - Note: Roles usually become attributes

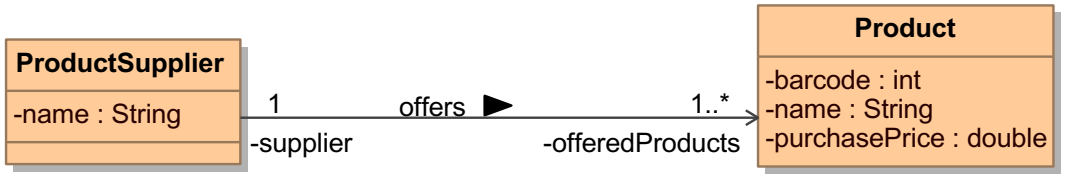


- Sometimes the direction of reading is made explicit



Class Diagrams – Associations: Notation

- Associations have **Multiplicities/Cardinalities**



- Enhancement of the expressiveness; used when
  - One instance of the class refers to a set of instances of another class, e.g., a supplier offers more than one product
  - Structural constraints to be expressed, e.g., at least one → "1..*"

- If there are no explicit cardinalities: assumption = "1"

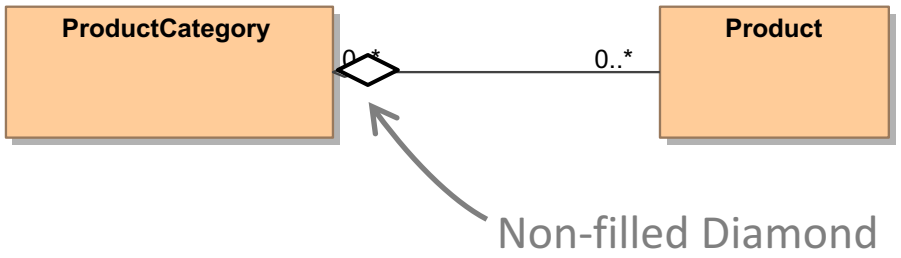
Class Diagrams – Associations: Notation

- Special association: the composition
- Used when: modeling a “whole-part” relationship, e.g.:
  - Group – person
  - Car – wheels, seats, etc.

- UML → **Composition** association
  - The “whole” owns its parts, and every part belongs to exactly one composite
  - If the whole is deleted, all parts are deleted, too → “existence”

Class Diagrams – Associations: Notation

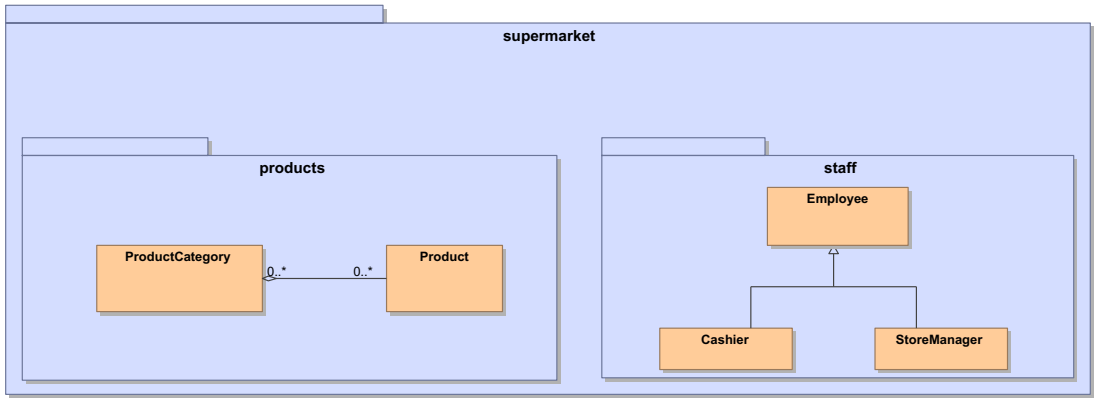
- Another form of the “whole-part” – the **Aggregation**
- Difference to the composition
  - A part may belong to multiple wholes, e.g., a person may be part of different groups
  - If the whole is deleted, the parts may still exist



Class Diagrams – Advanced Structuring

- Class diagrams can easily become very complex, e.g., inheritance hierarchies, sheer number of classes, etc.
- Potential problems: complexity, understandability, conflicts,…
- UML-approach: **Packages**
  - Logical grouping of classes
  - Definition of namespaces
  - Hierarchical order

- Access: fully-qualified name:
  p1::p2:: … ::pN::class



Class Diagrams – What remains…

This was just a brief introduction…

- Further elements relevant for classes and class diagrams:
  - Inheritance details, including
  - Interfaces, interface inheritance
  - Interface implementation
  - Nested types
  - Ports

- Special UML design elements beyond classes:
  - Data types
  - Enumerations

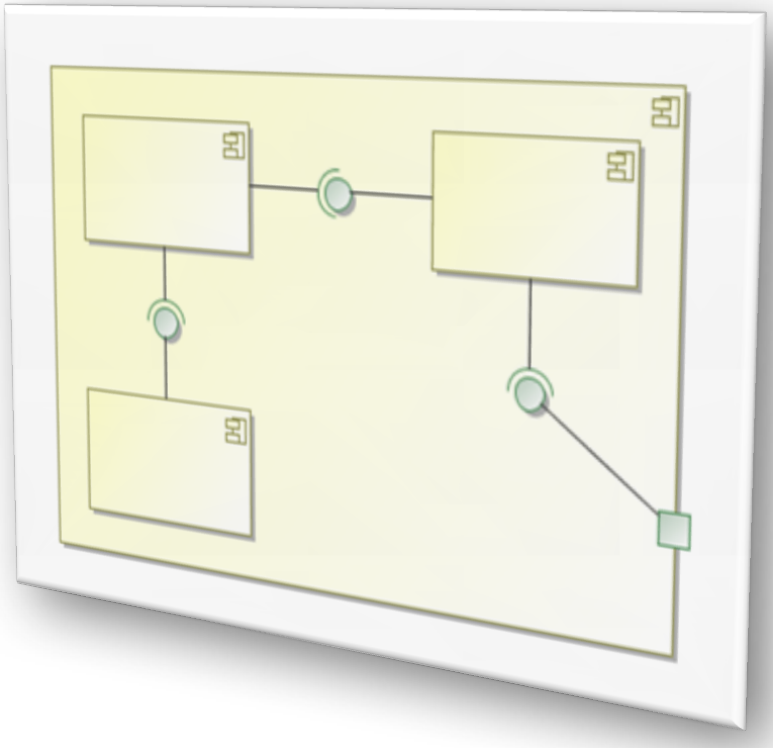
- The UML is (today) quite complex, and there are several UML-profiles, e.g., BPMN, SPEM, etc.

UNIFIED MODELING LANGUAGE:
COMPONENT DIAGRAMS

Design of Software Systems

Component Diagrams

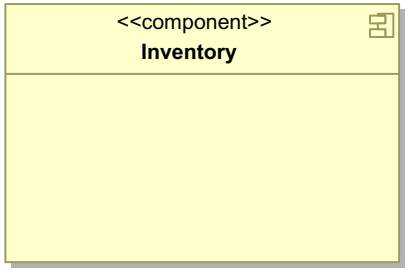
- While class diagrams describe the system elements from a fine-grained perspective…
- Component diagrams describe the structure using (coarse-grained) components, interfaces, and connections
- Focus either on the logical or on the physical (technical) view



- **Recall:** What is a component?

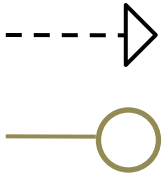
- Relevant notation elements
  - Component (type)
  - Interface (type, realization, and use)
  - Hierarchy

Component Diagrams – Terminology/Definitions

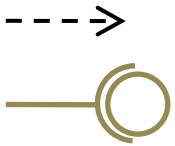


Component: encapsulates modular parts of a system, e.g., components have a behavior, which is offered or consumed via interfaces. Components define a type.

Interface: defines a set of operations (but do **not** implement them – can be compared to abstract classes).



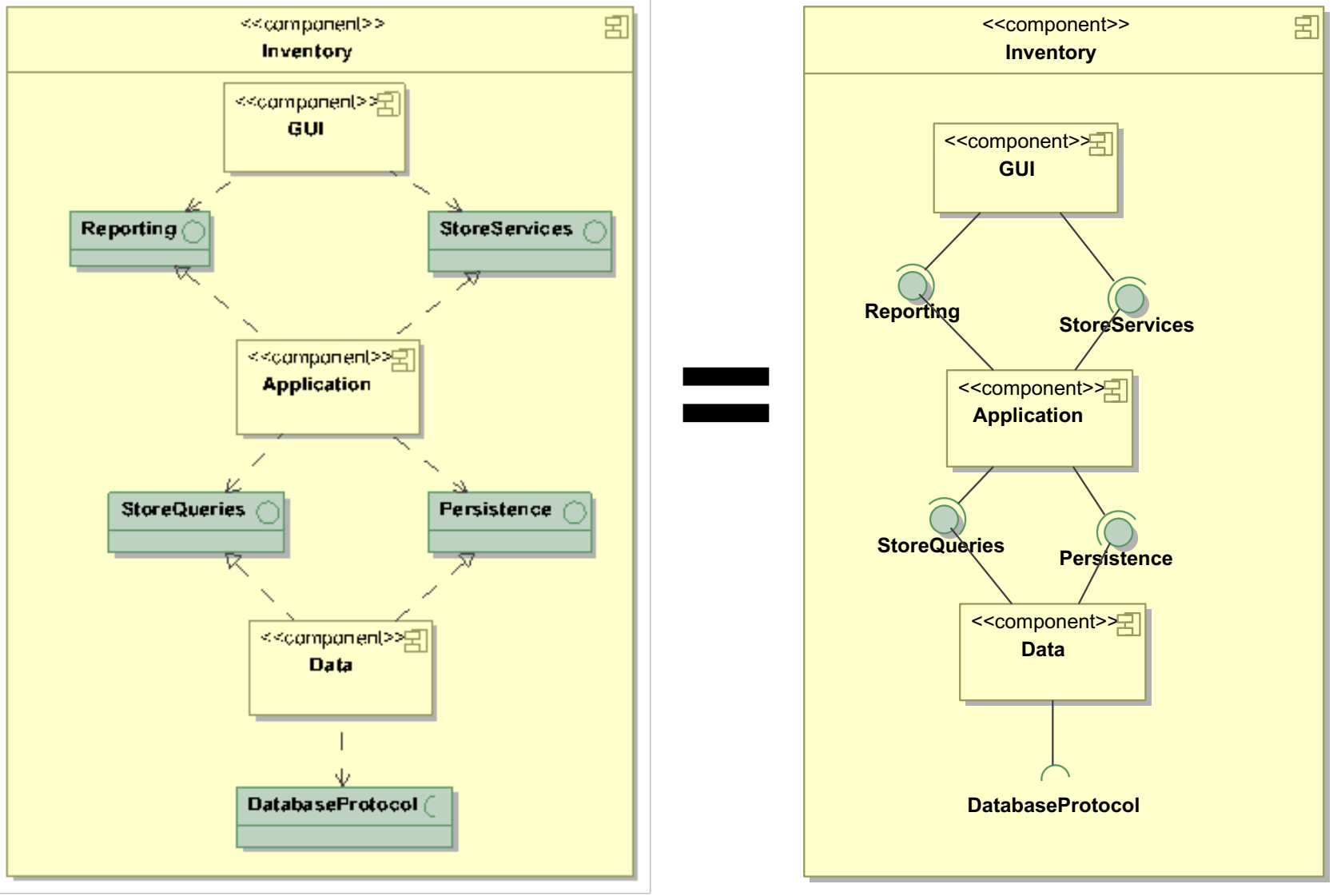
Interface Realization: shows that a component provides a realization of an interface, i.e. "hidden" and inside the component, an implementation is provided.



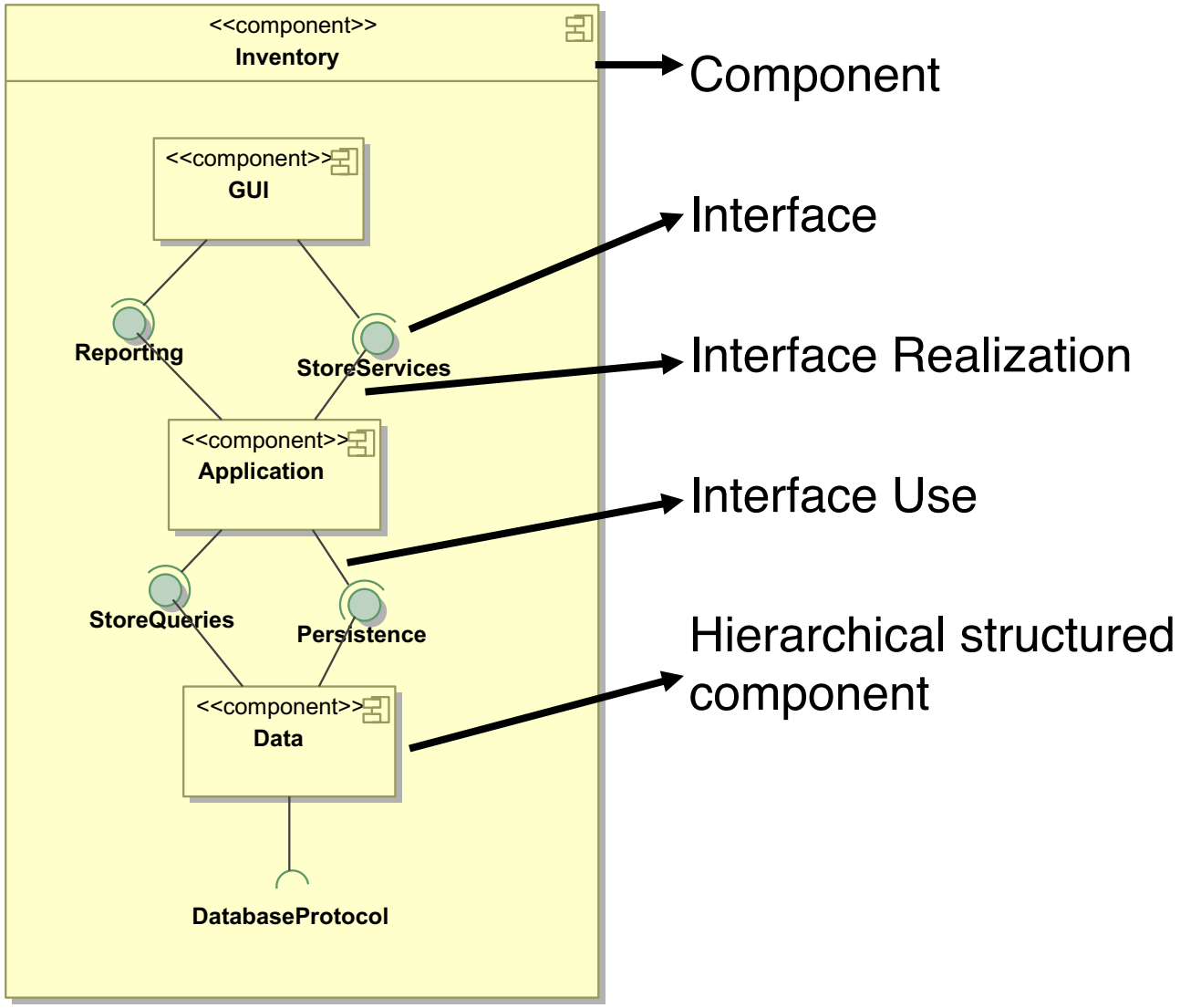
"Lollipop Notation"

Interface Use: shows that a component requires a particular interface in order to realize its own behavior (e.g., calling operations, consuming services…).

Component Diagrams – Notation



Component Diagrams – Notation



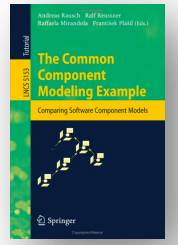
Component Diagrams – What remains…

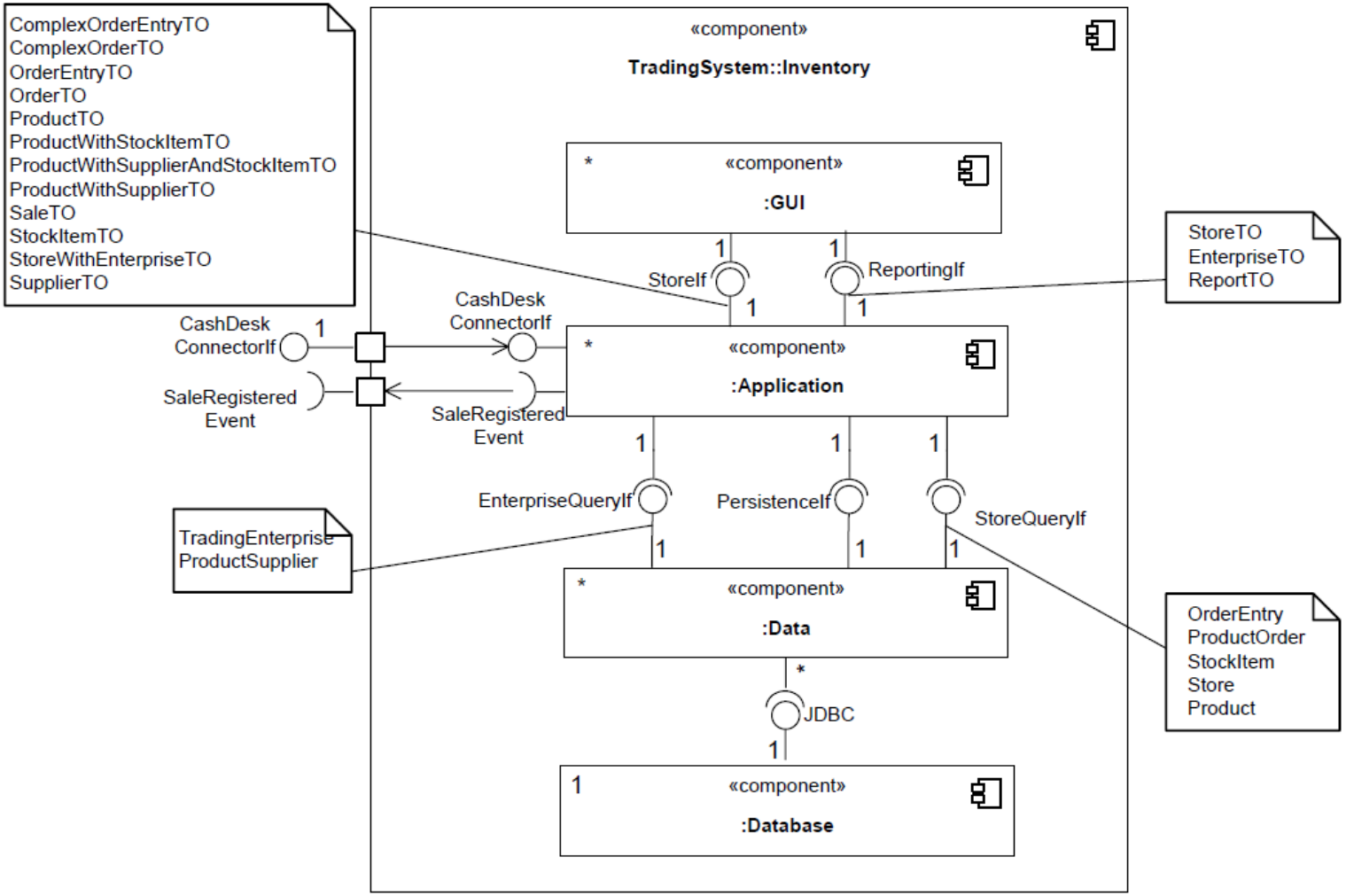
This was just a brief introduction…

- Further elements relevant for components and component diagrams:
  - Ports
  - Connectors
  - Delegation

- Further problems
  - Component diagrams give information about **Types**
  - Composition of components → Component Structure Diagram
    - What is the structure of a component?
    - Major problem: components usually address the black-box view
  - What is exactly needed to realize a component, e.g., number of required interface instances

Example


The Common Component Modeling Example
Springer


ComplexOrderEntryTO
ComplexOrderTO
OrderEntryTO
OrderTO
ProductTO
ProductWithStockItemTO
ProductWithSupplierAndStockItemTO
ProductWithSupplierTO
SaleTO
StockItemTO
StoreWithEnterpriseTO
SupplierTO
«component»
TradingSystem::Inventory
«component»
:GUI
StoreIf
ReportingIf
StoreTO
EnterpriseTO
ReportTO
CashDesk
ConnectorIf
CashDesk
ConnectorIf
SaleRegistered
Event
SaleRegistered
Event
«component»
:Application
EnterpriseQueryIf
PersistenceIf
StoreQueryIf
TradingEnterprise
ProductSupplier
«component»
:Data
JDBC
«component»
:Database
OrderEntry
ProductOrder
StockItem
Store
Product

UNIFIED MODELING LANGUAGE:
SYSTEM DIAGRAMS

Design of Software Systems

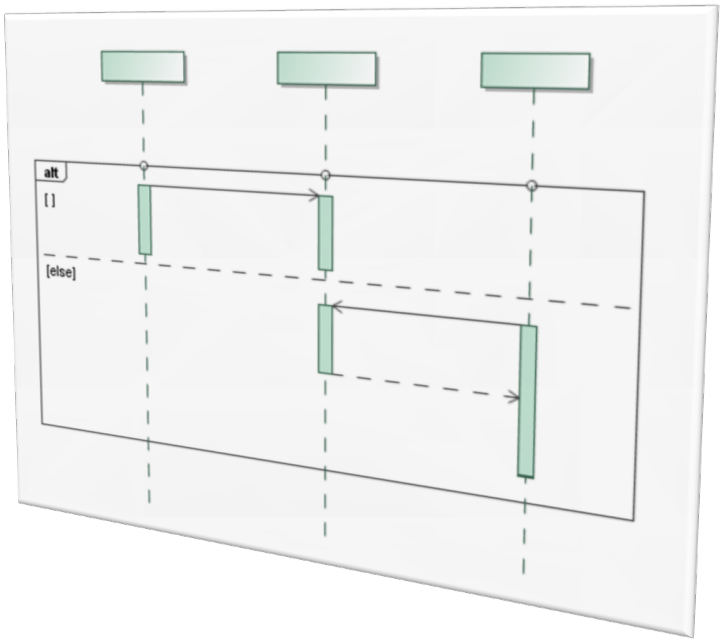
System Behavior

- We now talk about **Instances** (not types)
- We are interested into
  - Element interaction at runtime →
  - Behavior, interaction, life cycle
- Purpose (depending on the context)
  - Data flow, control flow
  - State, processes, etc.

- UML provides the following diagram types:
  - Interaction diagrams, i.e. Sequence Diagram
  - Activity Diagram
  - State Machine

Sequence Diagrams

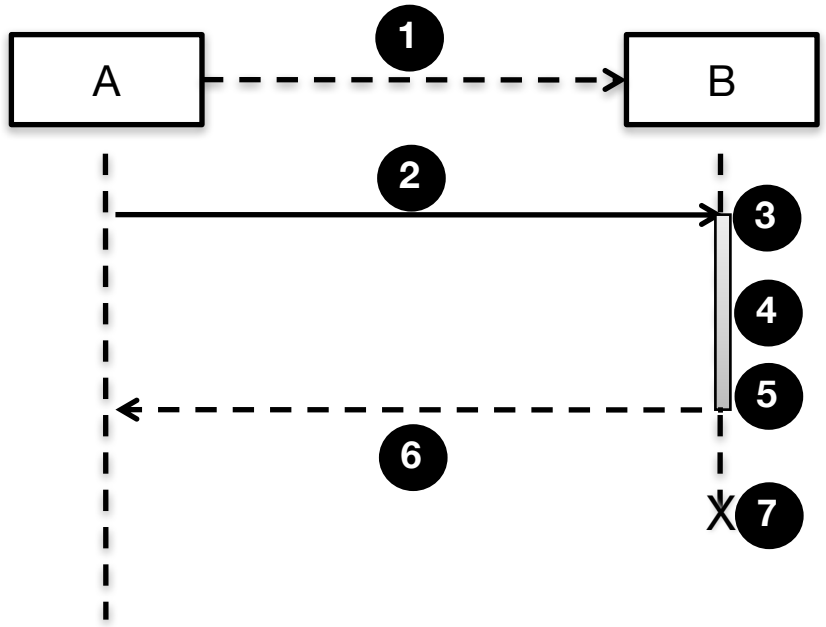
- A sequence diagram describes:
  - Interactions
  - Communication pattern in scenario
  - Information exchange between communication partners
    - Within a system
    - Between systems



- Relevant notation elements
  - Communication partners, e.g., objects
  - Life lines
  - Messages
  - Action sequences
  - Further elements to control a interaction/process

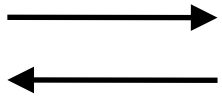
Sequence Diagrams – Notation, Details

1. Construction call
2. Message, e.g., method call
3. Start execution
4. Message processing
   → communication partner performs some operations; does “something”
5. End of execution
6. Response message, e.g., return the operation result, jump back
7. Destructor
   - Instance is deleted
   - Life line ends

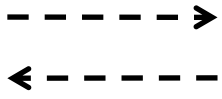


Sequence Diagrams – Notation Elements

- Synchronous communication (call, wait for response, continue):
  - Call
  - Response

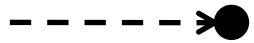


- Asynchronous communication (call, continue):
  - Call
  - Response



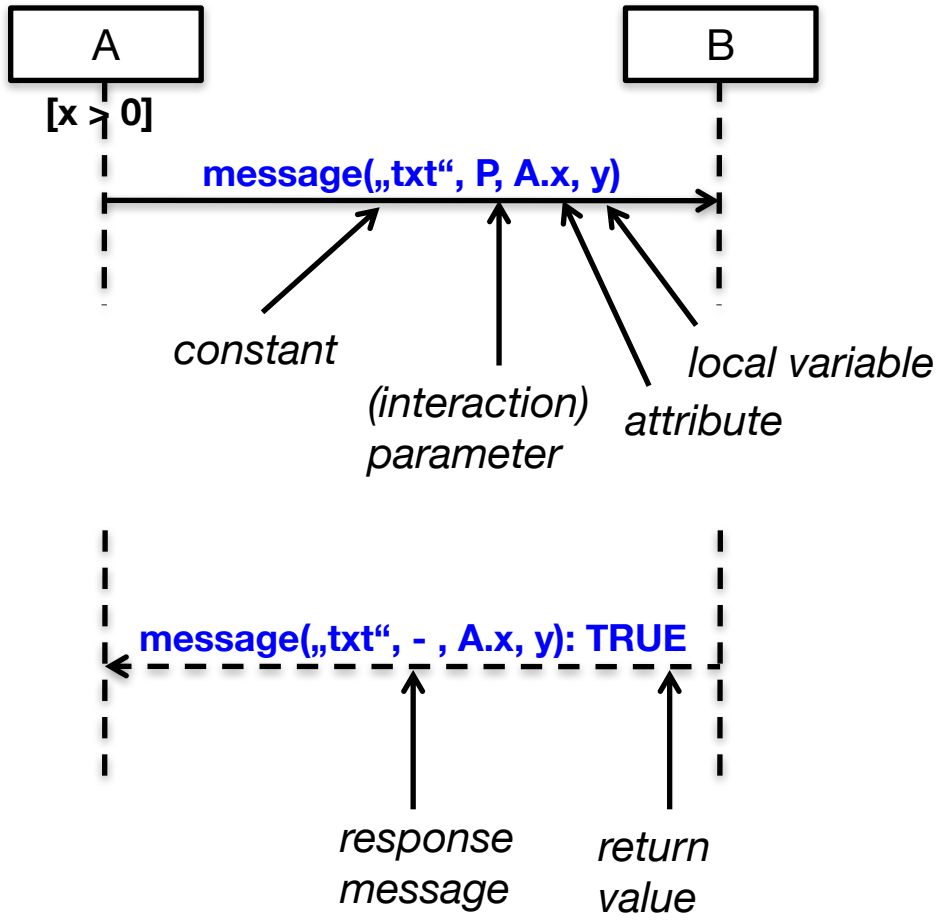
- Special notation elements:
  - “found message”
    A message without known sender

  - “lost message”
    A message of which the receiver is not known



Sequence Diagrams – Notation Elements

- Message arguments:
  - Attributes of sending life lines
  - Constants
  - In-/out-parameter
  - Attributes of classifiers
  - Generic values (wild cards)
- Number of arguments → number of parameters of the called operation
- Response message syntax:
  - Name and parameters are repeated
  - Return value → …**:value**

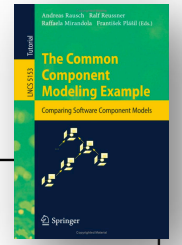


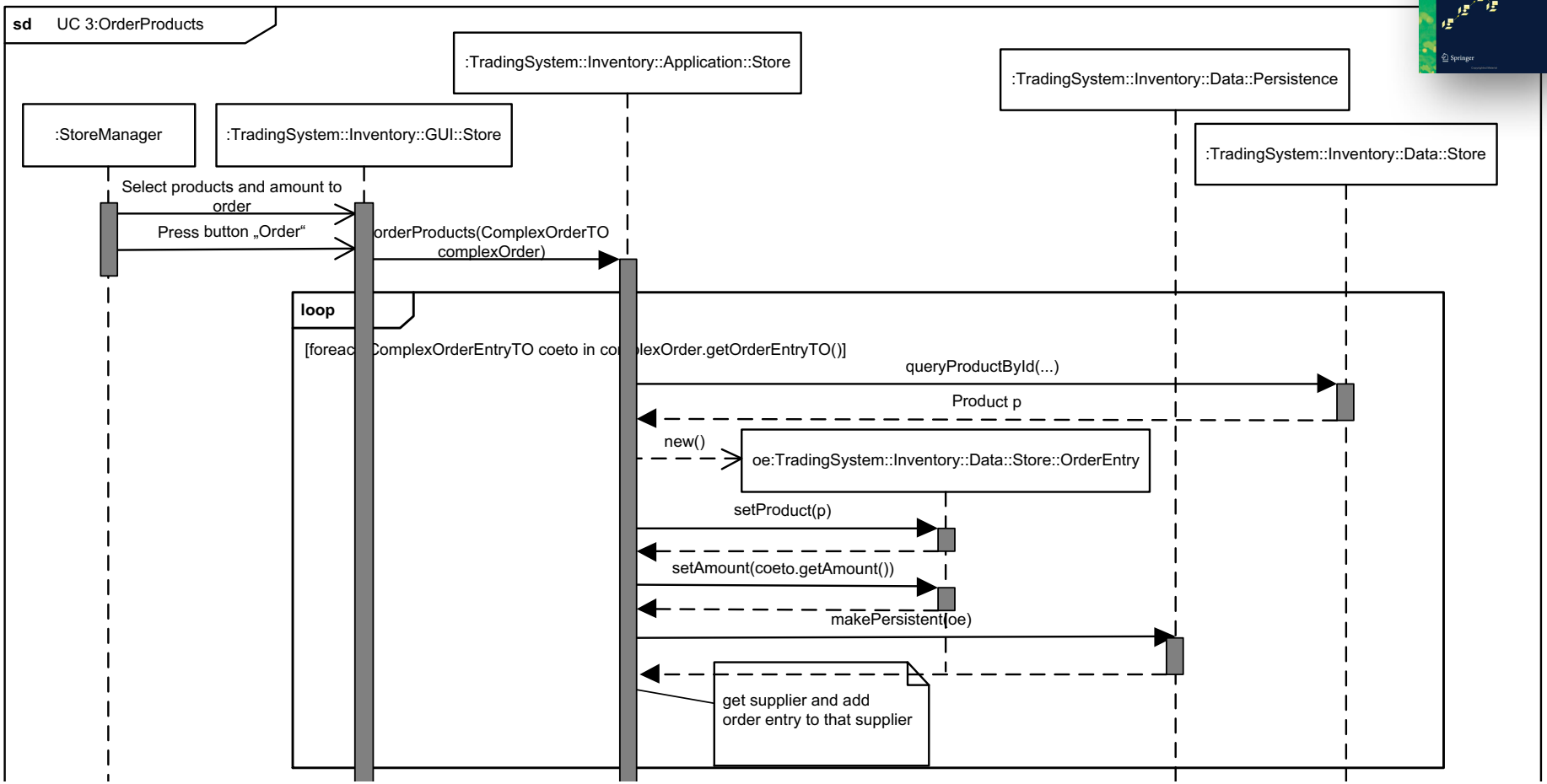
Sequence Diagrams – What remains…

This was just a brief introduction…

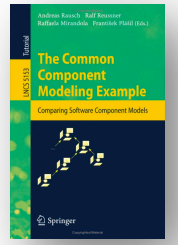
- Further elements relevant for sequence diagrams:
  - Interaction references, e.g., “nested” sequence charts
  - Interaction operators, e.g.:
    - Loops
    - Decisions/alternatives

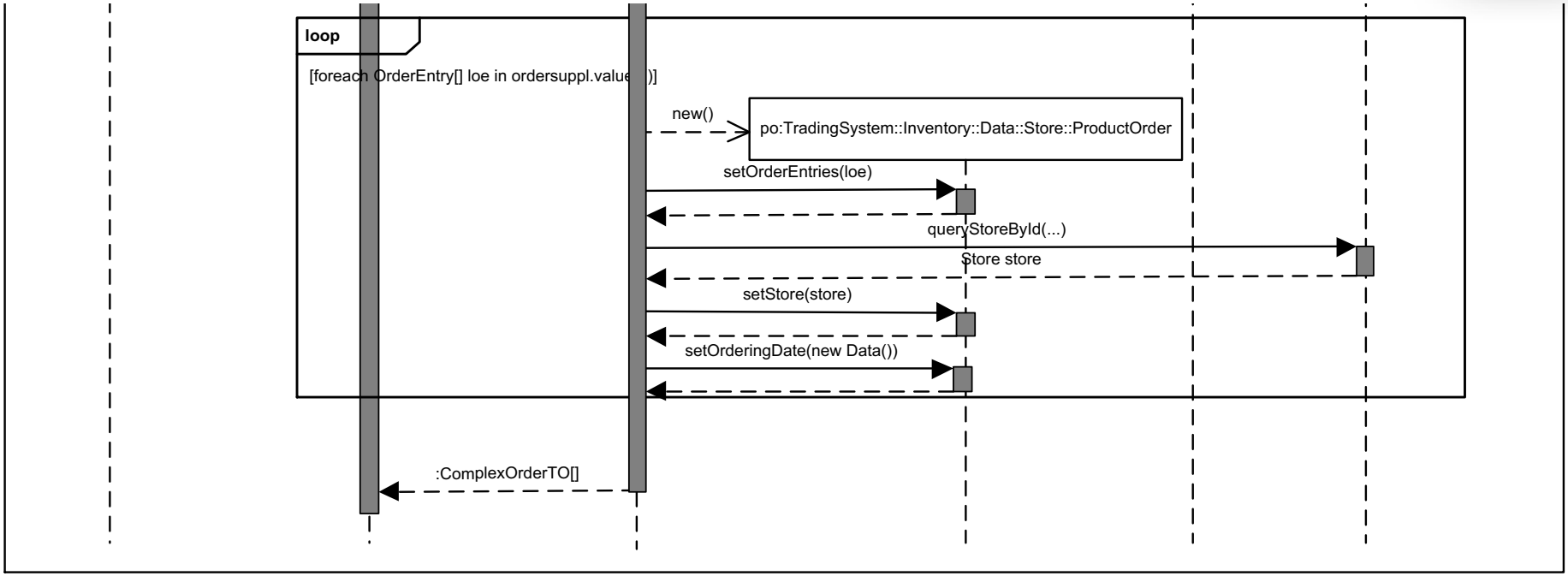
Example


The Common Component Modeling Example
Comparing Software Component Models
Springer


sd UC 3:OrderProducts
:TradingSystem::Inventory::Application::Store
:TradingSystem::Inventory::Data::Persistence
:StoreManager
:TradingSystem::Inventory::GUI::Store
:TradingSystem::Inventory::Data::Store
Select products and amount to order
Press button „Order“
orderProducts(ComplexOrderTO complexOrder)
loop
[foreac ComplexOrderEntryTO coeto in co lexOrder.getOrderEntryTO()]
queryProductById(...)
Product p
new()
oe:TradingSystem::Inventory::Data::Store::OrderEntry
setProduct(p)
setAmount(coeto.getAmount())
makePersistent(oe)
get supplier and add order entry to that supplier

Example


The Common Component Modeling Example
Springer


loop
[foreach OrderEntry[] loe in ordersuppl.value())]
new()
po:TradingSystem::Inventory::Data::Store::ProductOrder
setOrderEntries(loe)
queryStoreById(...)
Store store
setStore(store)
setOrderingDate(new Data())
:ComplexOrderTO[]

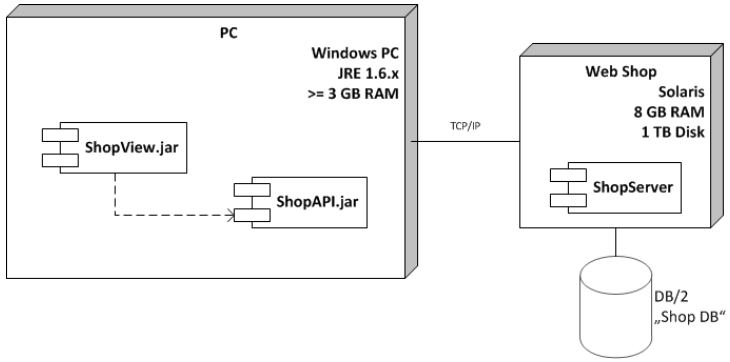
Deployment Diagram

Purpose: description of the system environment and component distribution

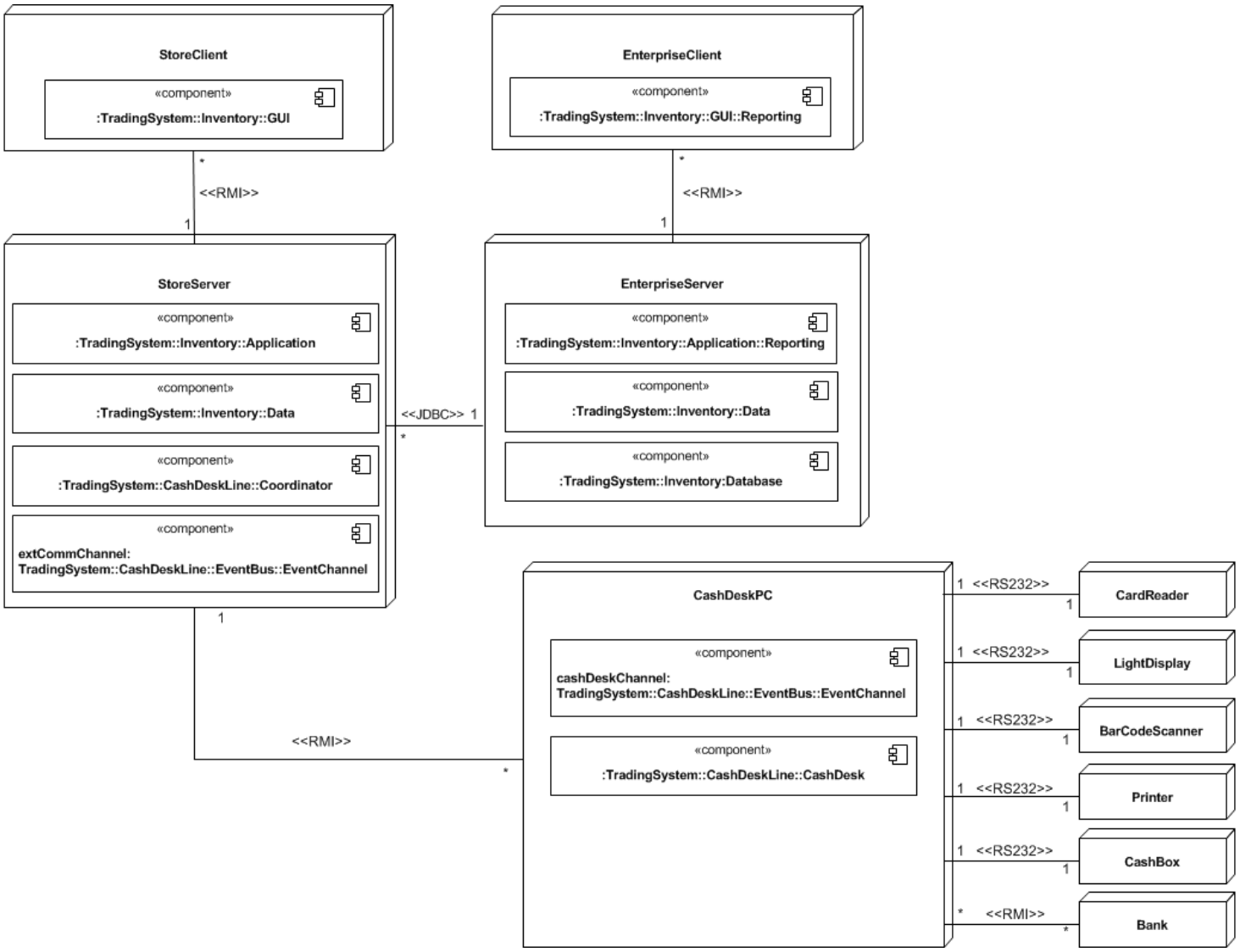
- Components of the technical infrastructure
  - Operating systems, 3rd-party systems
  - Hardware
  - Performance specification
- Runtime components
  - Executable software components to deploy
  - Communication between components

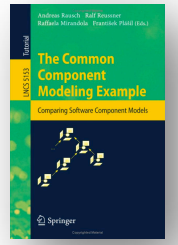
- Notation elements: UML Deployment Diagram
  - Nodes = technical elements
  - Component- and package icons = software systems
  - Associations = physical connection

- **Note:** representation and level of detail should be adjusted in response to the actual context/requirements



Example


StoreClient
«component»
:TradingSystem::Inventory::GUI
EnterpriseClient
«component»
:TradingSystem::Inventory::GUI::Reporting
*
<<RMI>>
1
*
<<RMI>>
1
StoreServer
«component»
:TradingSystem::Inventory::Application
«component»
:TradingSystem::Inventory::Data
«component»
:TradingSystem::CashDeskLine::Coordinator
«component»
extCommChannel:
TradingSystem::CashDeskLine::EventBus::EventChannel
<<JDBC>> 1
*
EnterpriseServer
«component»
:TradingSystem::Inventory::Application::Reporting
«component»
:TradingSystem::Inventory::Data
«component»
:TradingSystem::Inventory:Database
1
<<RMI>>
*
CashDeskPC
«component»
cashDeskChannel:
TradingSystem::CashDeskLine::EventBus::EventChannel
«component»
:TradingSystem::CashDeskLine::CashDesk
1 <<RS232>> 1
CardReader
1 <<RS232>> 1
LightDisplay
1 <<RS232>> 1
BarCodeScanner
1 <<RS232>> 1
Printer
1 <<RS232>> 1
CashBox
* <<RMI>> *
Bank


The Common Component Modeling Example
Comparing Software Component Models
Springer

State machine Diagram

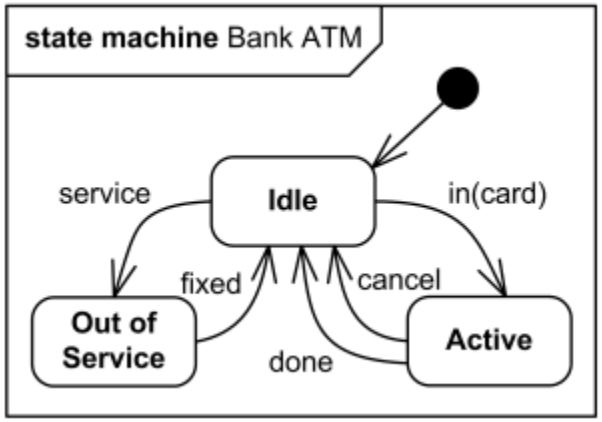
State machine diagram shows the discrete behaviour of a part of a system through **finite** state transitions.

The UML includes notation to illustrate events and states of things (transactions, Use Cases, people,. . . etc.).

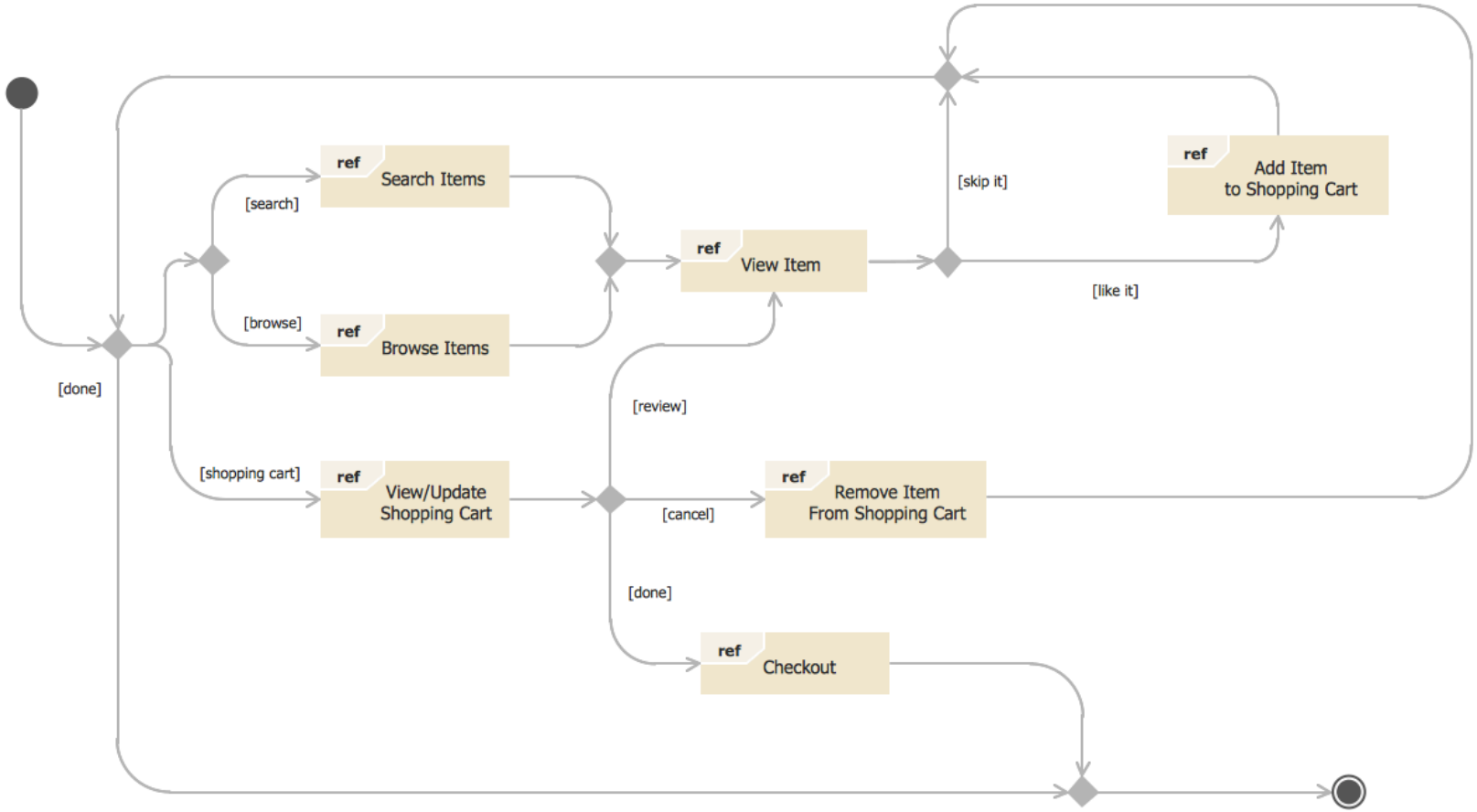
An event is a significant or noteworthy occurence e.g. a telephone receiver is taken off the hook

A state is the condition of an object at a moment in time

A transition is a relationship between two states that indicates when an event occurs e.g. when the event “off hook” occurs, transition the telephone from “idle” to “active” state



State machine diagram example



UML- What did we cover so far?

UML Diagrams

Structure Diagrams

- **Class diagram**
- Package diagram
- Object diagram
- **Component diagram**
- Composition (structure) diagram
- **Deployment diagram**

Behavior Diagrams

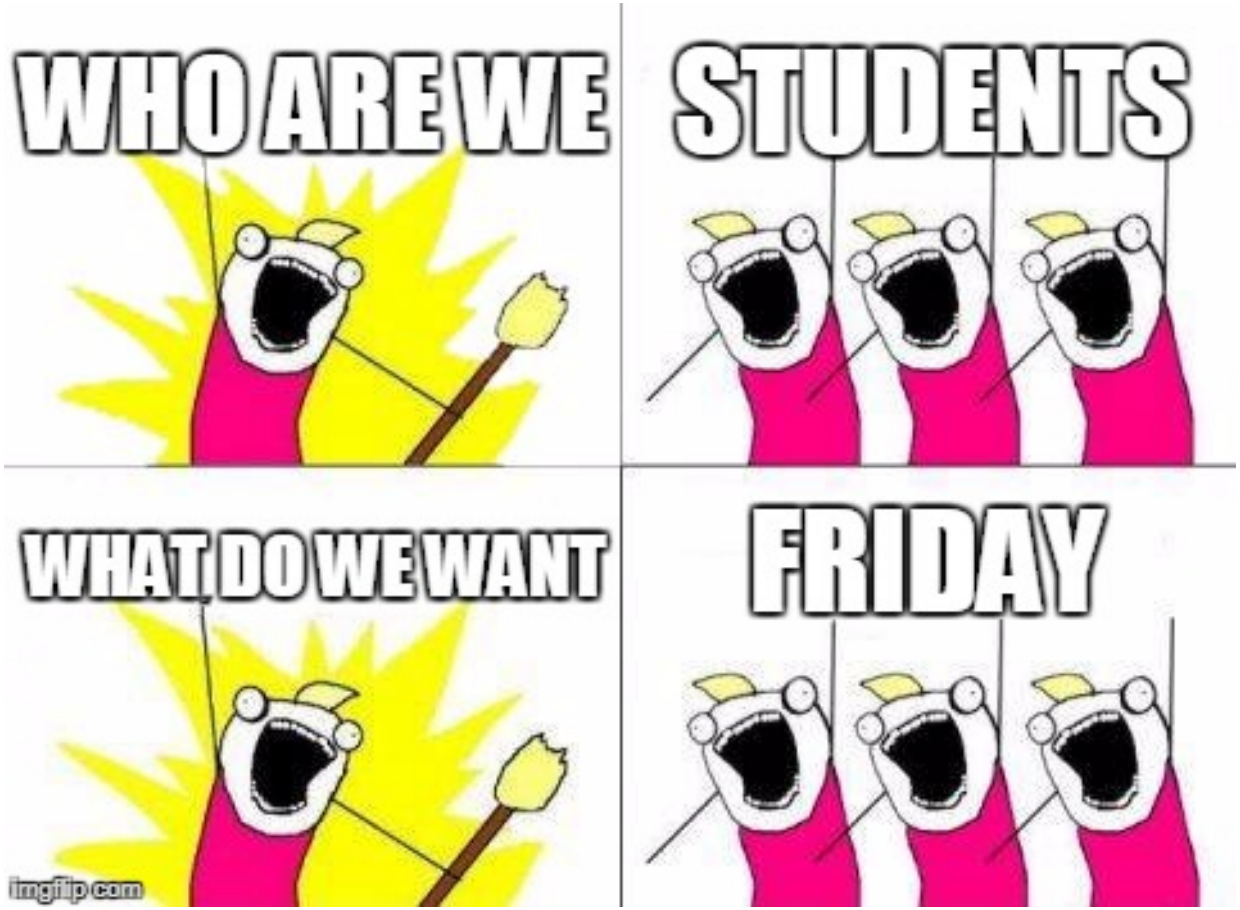
- **Use Case diagram**
- Activity diagram
- **State machine**

Interaction Diagrams

- **Sequence diagram**
- Timing diagram
- Communication diagram
- Interaction overview diagram

UML – Summary and what remains…

- Today we covered
  - Use case diagrams
  - Class diagrams
  - Component diagrams
  - System diagrams
  - State machine diagrams
- There are **many** more diagrams, and
- There are more options to extend and customize UML
  → topics that we did not cover:
  - The UML metamodel, cf. MOF (Meta Object Facility)
  - Special language elements
    - Stereotypes
    - Tagged values
    - Constraints, cf. OCL (Object Constraint Language)
    - Notes
  - UML Profiling → based on the MOF, you can create your own UML-based modeling languages; examples
    - BPMN (Business Process Modeling Notation)
    - SPEM (Software & Systems Process Engineering Metamodel)
    - Other so-called *domain-specific languages*


WHO ARE WE
STUDENTS
WHAT DO WE WANT
FRIDAY
imgflip.com