

MACIASZEK, L.A. (2005):
Requirements Analysis and System Design, 2nd ed.
Addison Wesley, Harlow England, 504p.
ISBN 0 321 20464 6

Chapter 3.2
Objects and Object Modeling
Fundamentals of object modeling

© Pearson Education Limited 2005

Topics

- *Use case modeling*
- *Activity modeling*
- *Class modeling*
- *Interaction modeling*
- *Statechart modeling*
- *Implementation models*

Kinds of UML models

- *State model*
 - *static view of the system*
 - *models data requirements and operations on data*
 - *operations obtain from behavioral model*
 - *class diagram*
- *Behavior model*
 - *operational view of the system*
 - *models function requirements*
 - *diagrams*
 - *use case*
 - *sequence*
 - *collaboration*
 - *activity*
- *State change model*
 - *dynamic view of the system*
 - *models object evolution over time*
 - *statechart diagram*

Use case modeling

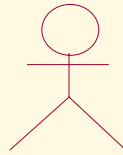
- **Use case** - outwardly visible and testable system behavior
- **Actor** - whoever or whatever (person, machine, etc.) that interacts with a use case
 - a role that somebody or something plays, not a particular person or machine
 - actor receives a **useful result**
- **Use case** represents a complete unit of functionality of value to an actor
 - There may be some use cases that do not directly interact with actors
 - In many instances, a function requirement maps directly to a use case
- **Use Case Diagram** is a visual representation of actors and use cases together with any additional definitions and specifications
- Unless stated otherwise, **annotated UML diagram** is synonymous with **UML model**

Assignment of reqs to actors and use cases

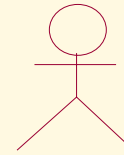
Req#	Requirement	Actor	Use case
1	Before a video can be rented out, the system confirms customer's identity and standing by swiping over scanner his/her Video Store membership card.	Customer, Employee	Scan Membership Card
2	A video tape or disk can be swiped over scanner to obtain its description and price (fee) as part of customer's enquiry or rental request.	Customer, Employee	Scan Video Medium
3	Customer pays the nominal fee before the video can be rented out. The payment may be with cash or debit/credit card.	Customer, Employee	Accept Payment Charge Payment to Card
4	The system verifies all conditions for renting out the video, acknowledges that the transaction can go ahead, and can print the receipt for the customer.	Employee, Customer	Print Receipt

Actors

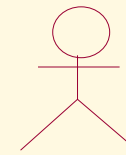
Stick-person
icons



Customer

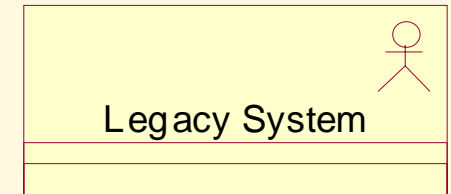
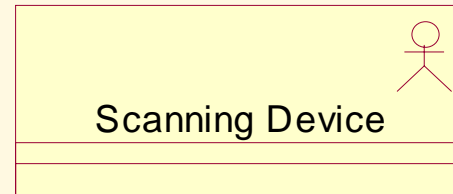
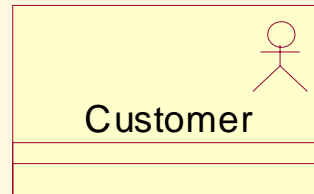


Scanning Device

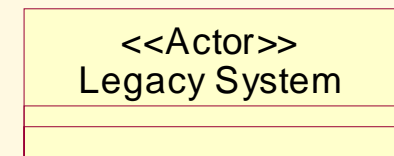
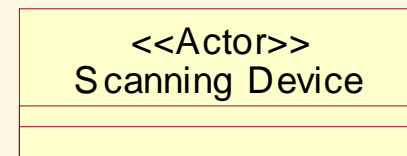
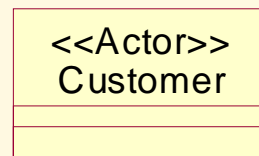


Legacy System

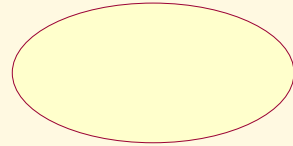
Decorated
classes



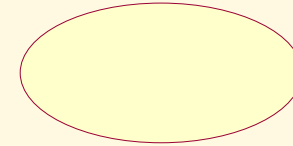
Stereotyped
classes



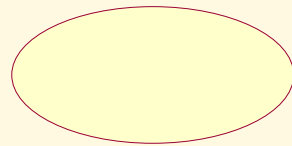
Use cases



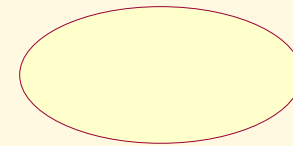
Scan Membership Card



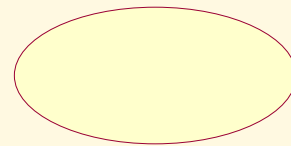
Scan Video Medium



Accept Payment

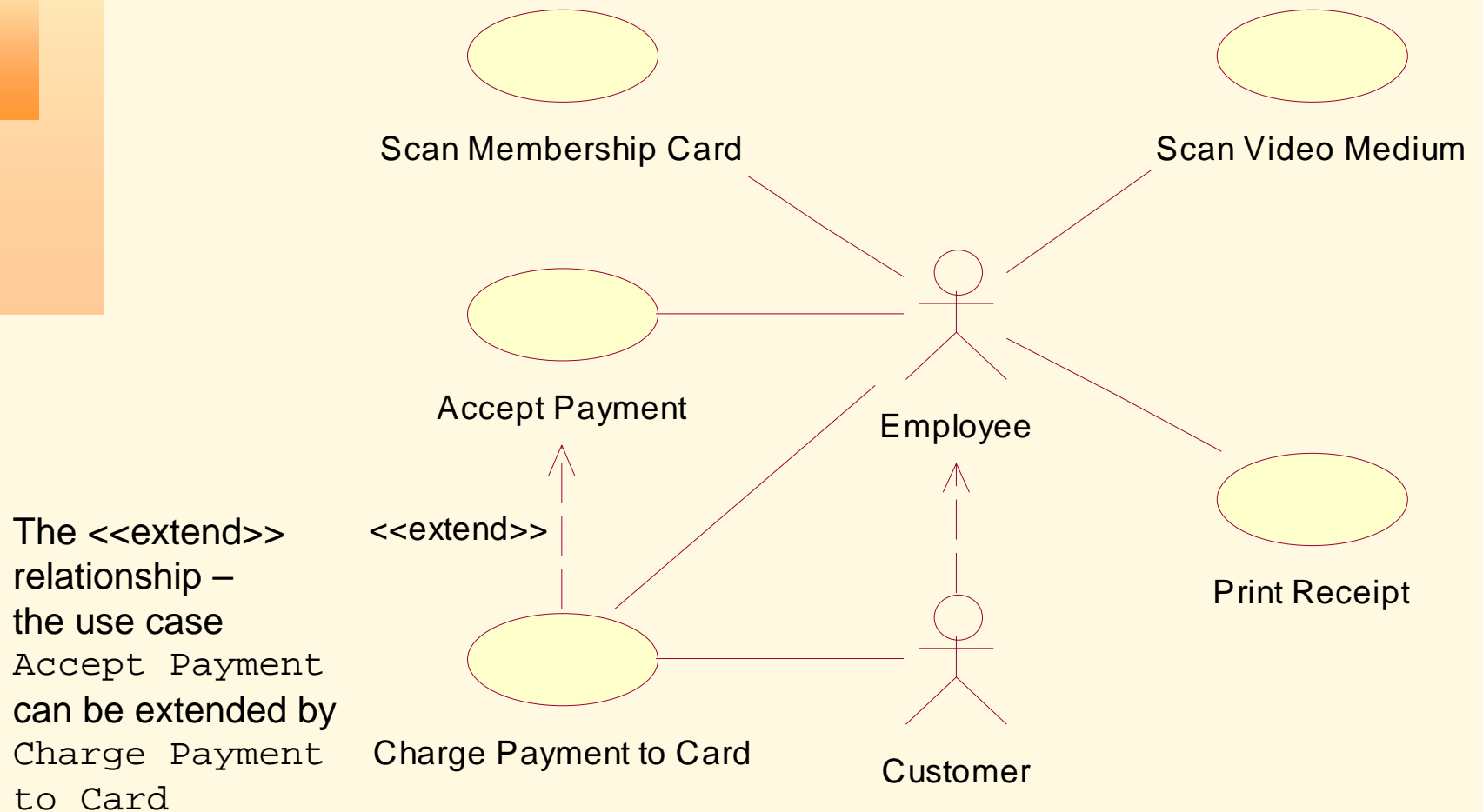


Charge Payment to Card



Print Receipt

Use case diagram



The <<extend>> relationship – the use case Accept Payment can be extended by Charge Payment to Card

Documenting use cases

- **Brief Description**
- **Actors** involved
- **Preconditions** necessary for the use case to start
- **Detailed Description** of flow of events that includes:
 - **Main Flow** of events, that can be broken down to show:
 - **Subflows** of events (subflows can be further divided into smaller subflows to improve document readability)
 - **Alternative Flows** to define exceptional situations
- **Postconditions** that define the state of the system after the use case ends

Narrative use case specification

Use case	Accept Payment
Brief description	This use case allows an Employee to accept the payment from <code>Customer</code> for a video rental.
Actors	Employee (primary), Customer(secondary).
Preconditions	Customer expresses readiness to rent the video and he/she possesses valid membership card and the video is available for rental.
Main flow	The use case begins when the Customer decides to pay for the video rental and offers cash or debit/credit card payment. The Employee requests the system to display the rental charge...
Alternative flows	The Customer does not have sufficient cash and does not offer the card payment. The <code>Employee</code> asks the system to verify ...
Postconditions	If the use case was successful, the payment is recorded in the system's database. Otherwise, ...

Activity modeling

■ **Activity model**

- *Can graphically represent the flow of events of a use case*
- *Can also be used to understand a business process at a high-level of abstraction before use cases are produced*

■ **Shows the steps of a computation**

- *Each step is a **state** of doing something*
- *Execution steps are called **action states***
- *Depicts which steps are executed in sequence and which can be executed concurrently*
- **Transition** – *the flow of control from one action state to the next*

■ **Use case descriptions** are (usually) written from an outside actor's perspective

■ **Activity models** can take an inside system's viewpoint

Finding actions in use case flows

No.	Use case statement	Action state
1	The Employee requests the system to display the rental charge together with basic customer and video details.	Display transaction details
2	If the Customer offers cash payment, the Employee handles the cash, confirms to the system that the payment has been received, and asks the system to record the payment as made.	Key in cash amount; Confirm transaction
3	If the Customer offers debit/credit card payment, the Employee swipes the card, and requests then the Customer to type the card's PIN number, select debit or credit account, and transmit the payment. Once the payment is electronically confirmed by the card provider, the system records the payment as made.	Swipe the card; Accept card number; Select card account; Confirm transaction

Actions

Display transaction details

Swipe the card

Accept card number

Key in cash amount

Enter card number manually

Select card account

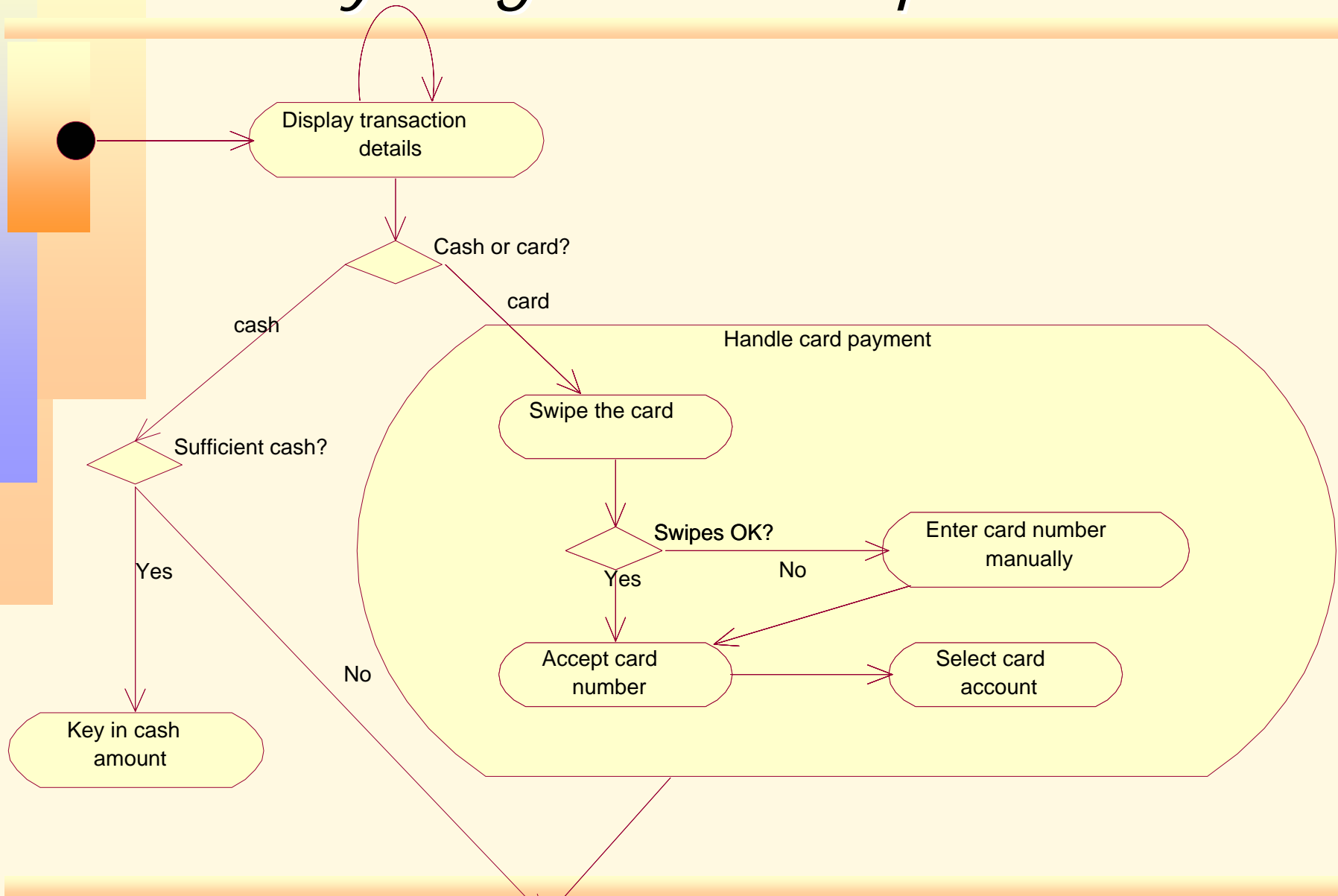


- *The Customer card does not swipe properly through the scanner. After three unsuccessful attempts, the Employee enters the card number manually .*

Activity diagram

- **Activity Diagram** shows transitions between actions
- A solid filled circle represents the **initial state**
- The **final state** is shown using so called bull's eye symbol
- Transitions can **branch and merge** (diamond) – **alternative** computation **threads**
- Transitions can **fork and re-join** (bar line) – **concurrent** (parallel) computation **threads**
- Activity diagram without concurrent processes resembles a conventional **flowchart**

Activity diagram - excerpt



Class modeling

- Captures **system state** – *the function of the system's information content at a point in time*
- *Class modeling elements*
 - *classes themselves*
 - *attributes and operations of classes*
 - *relationships – associations, aggregation and composition, generalization*
- **Class diagram** – *combined visual representation for class modeling elements*
- *Class modeling and use case modeling are typically conducted in parallel*

Persistent entity classes

- *So far, we have used classes to define "business objects"*
 - *Called **entity classes** (model classes)*
 - *Represent **persistent** database objects*
- *"Persistence saves the state and class of an object across time or space"*
- *A need for **mapping** between entity classes in program's memory and the corresponding tables in a relational database*
 - *the mapping enables **loading** table records to memory as entity objects and **unloading** from memory to database*

Transient program classes

- *Transient program classes*
 - **entity classes** – loaded to memory from persistent database
 - classes that define GUI objects (such as screen forms) – **presentation (boundary) classes** (view classes)
 - classes that control the program's logic and process use events – **control classes**
 - classes responsible for communication with external data sources – **foundation (database) classes**
 - classes responsible for managing entity objects in memory cache and for conducting business transactions – **mediator classes**
- *Classes other than entity classes may or may not be addressed in requirements analysis – may be delayed until the system design*

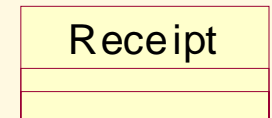
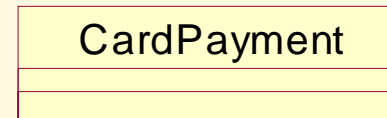
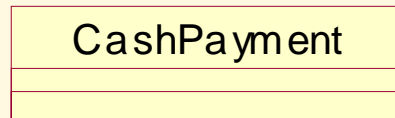
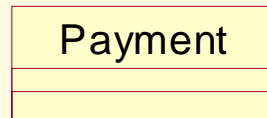
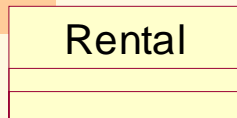
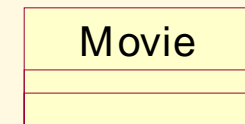
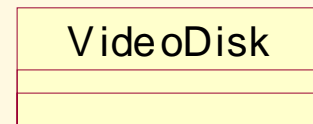
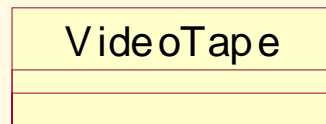
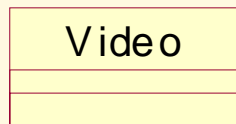
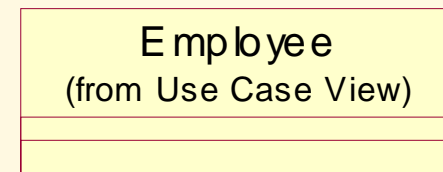
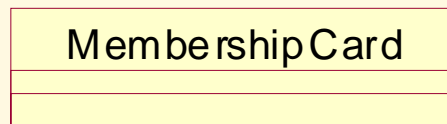
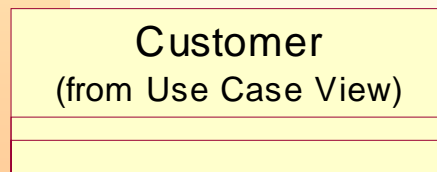
Assignment of requirements to entity classes

R#	Requirement	Entity class
1	Before a video can be rented out, the system confirms customer's identity and standing by swiping over scanner his/her Video Store membership card.	Video, Customer, MembershipCard
2	A video tape or disk can be swiped over scanner to obtain its description and price (fee) as part of customer's enquiry or rental request.	VideoTape, VideoDisk, Customer, Rental
3	Customer must pay the nominal fee before the video can be rented out. The payment may be with cash or debit/credit card.	Customer, Video, Rental, Payment
4	The system verifies all conditions for renting out the video, acknowledges that the transaction can go ahead, and prints the receipt for the customer.	Rental, Receipt

Classes







- *Is this a class?*
 - *Is the concept a container for data?*
 - *Does it have separate attributes that will take on different values?*
 - *Would it have many instance objects?*
 - *Is it in the scope of the application domain?*

Classes






Attributes

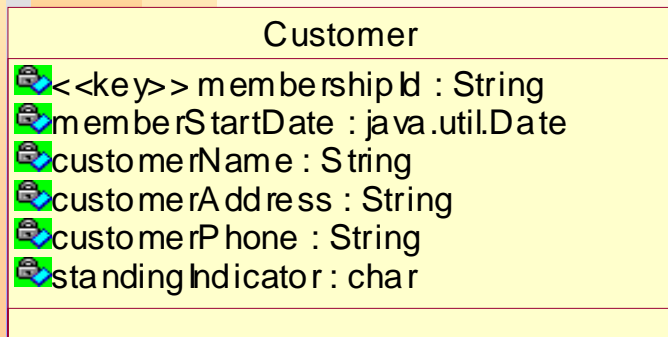
Customer

 <<key>> membershipId : String
 memberStartDate : java.util.Date
 customerName : String
 customerAddress : String
 customerPhone : String
 standingIndicator : char

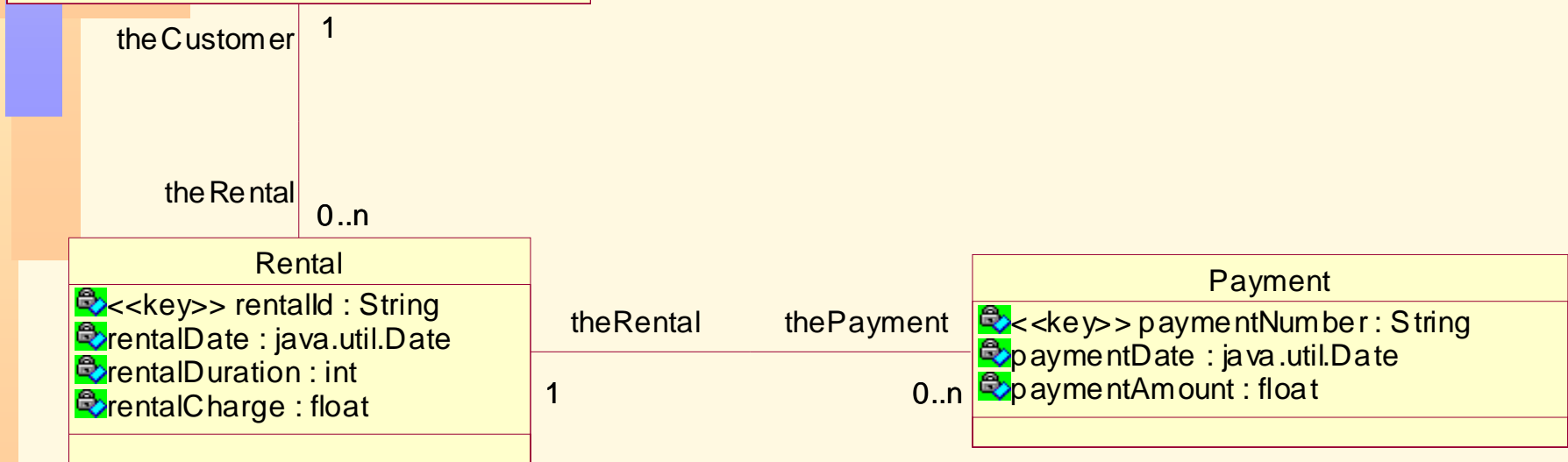
MembershipCard

 <<key>> membershipId : String
 cardIssueDate : java.util.Date
 cardExpiryDate : java.util.Date

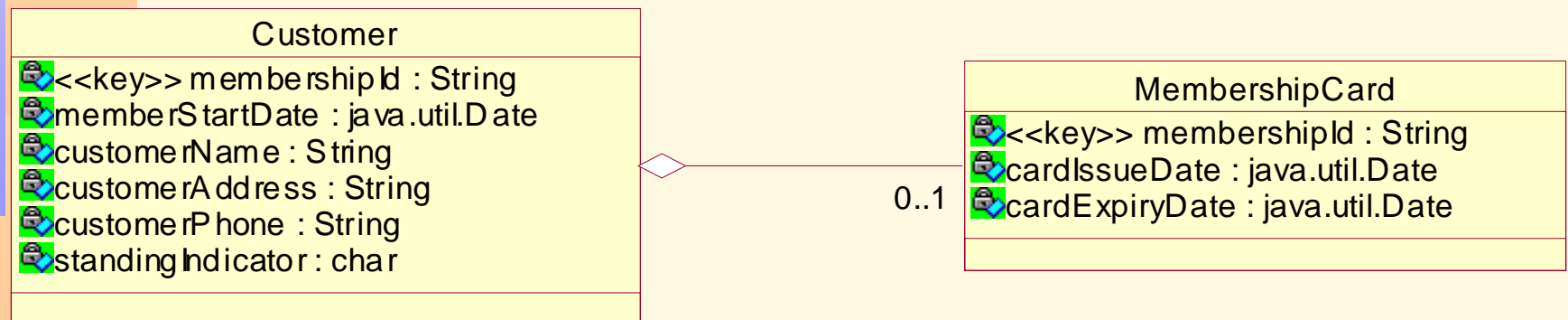
Associations



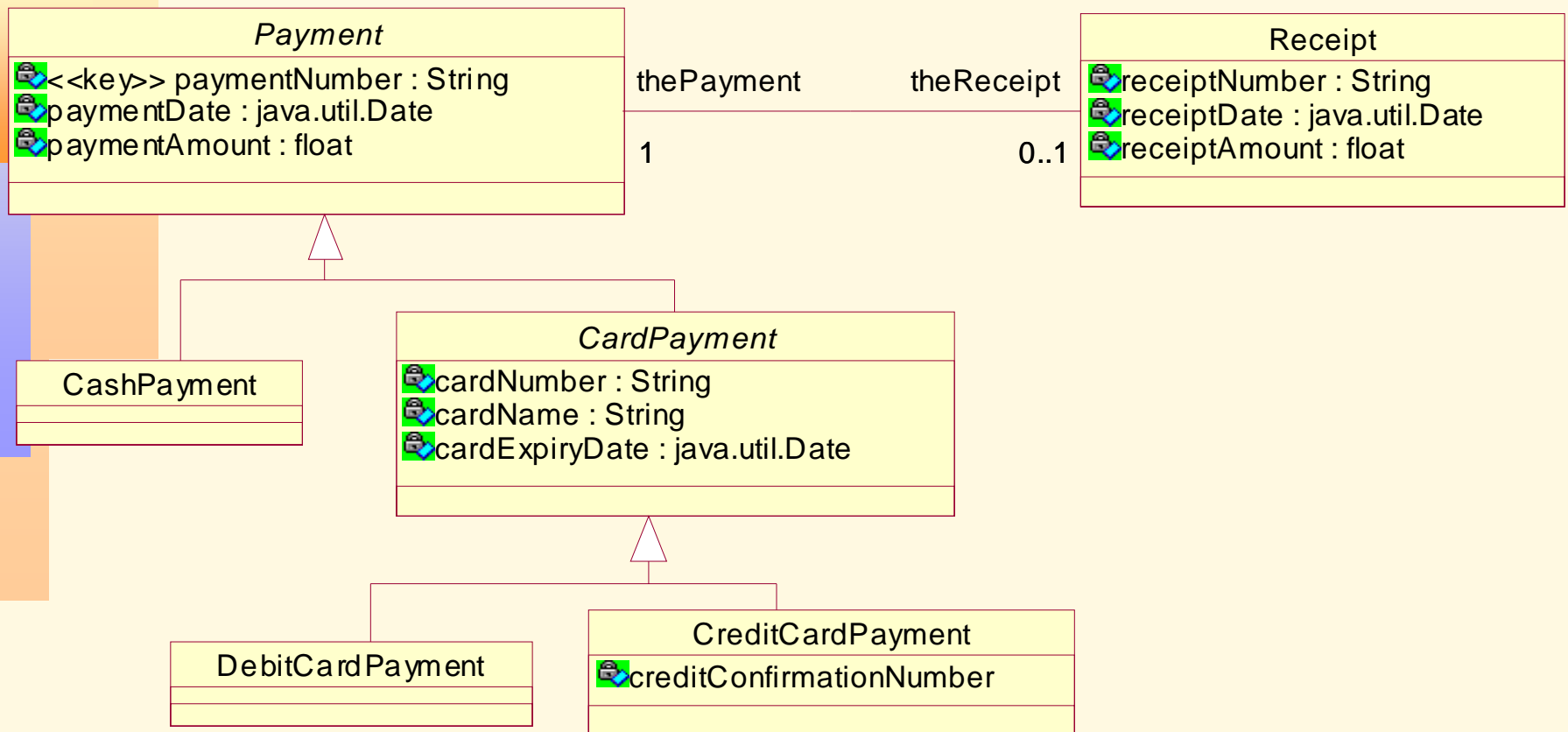
Is there a need for an association between Customer and Payment?



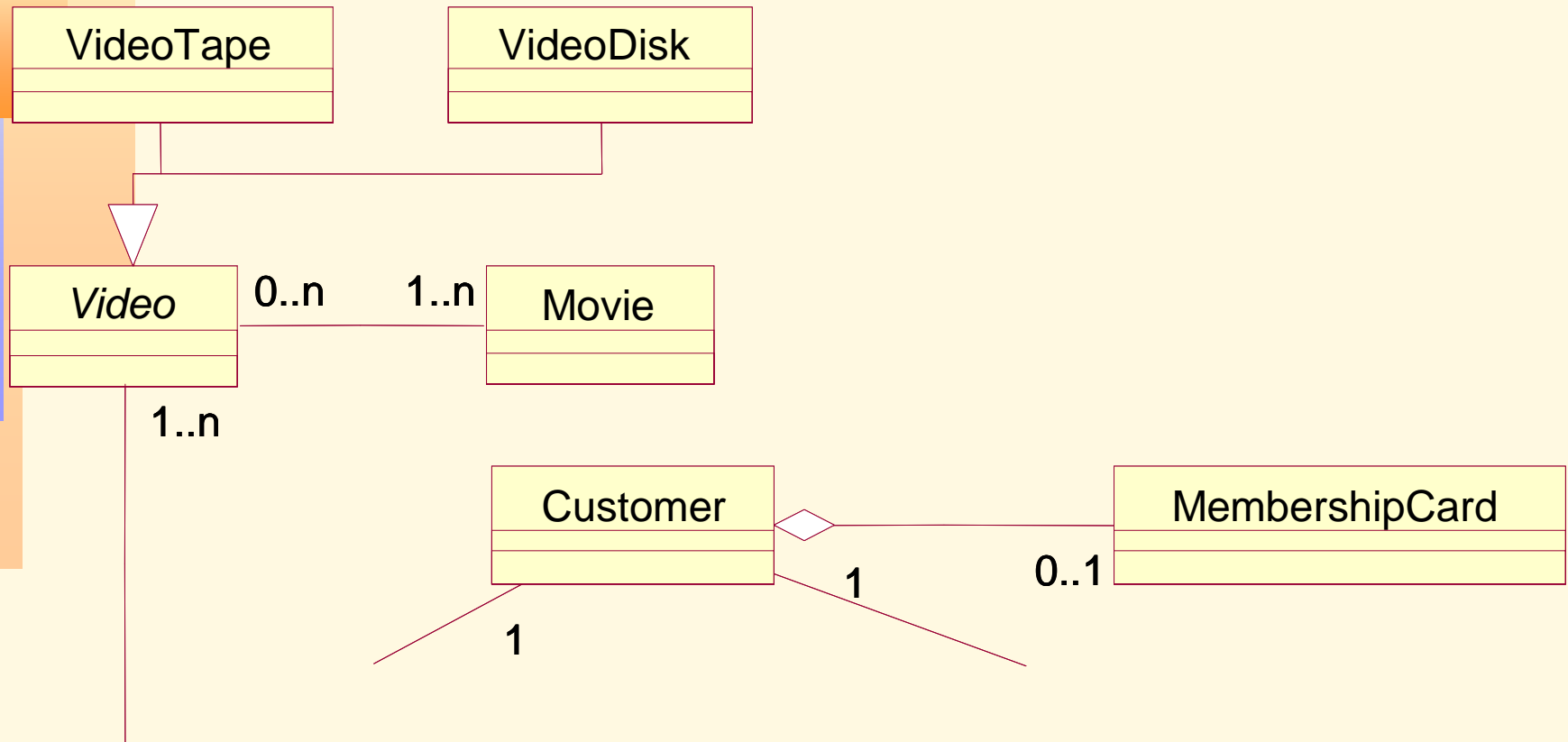
Aggregations



Generalizations



Class diagram - excerpt



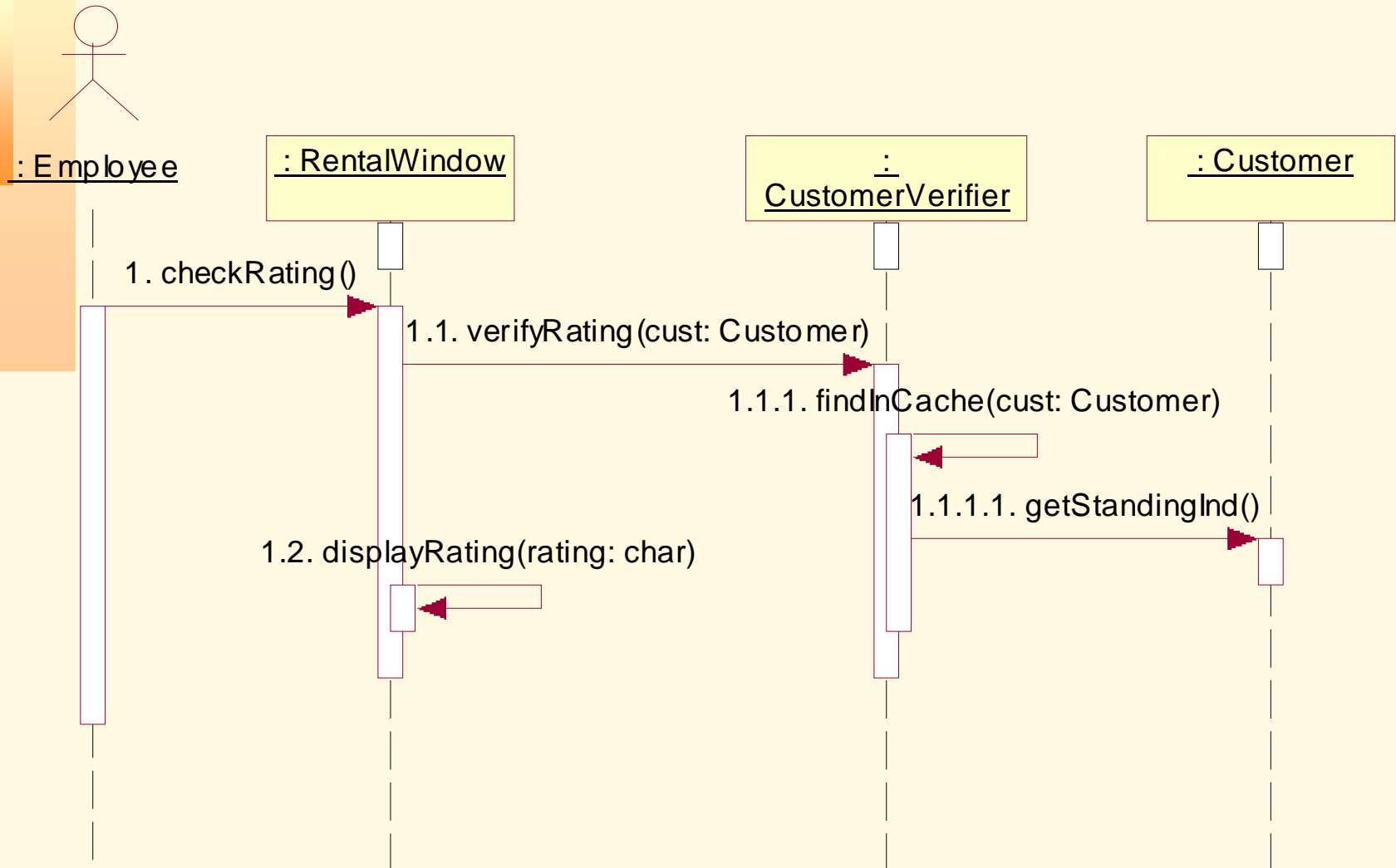
Interaction modeling

- *Captures interactions between objects needed to execute a use case or part of it*
- *Shows the sequencing of events (messages) between collaborating objects*
- *Used in more advanced stages of requirements analysis, when a basic class model is known, so that the references to objects are backed by the class model*
- *Two kinds of interaction diagrams*
 - **Sequence diagram** – concentrate on time sequences
 - **Collaboration diagram** – emphasize object relationships

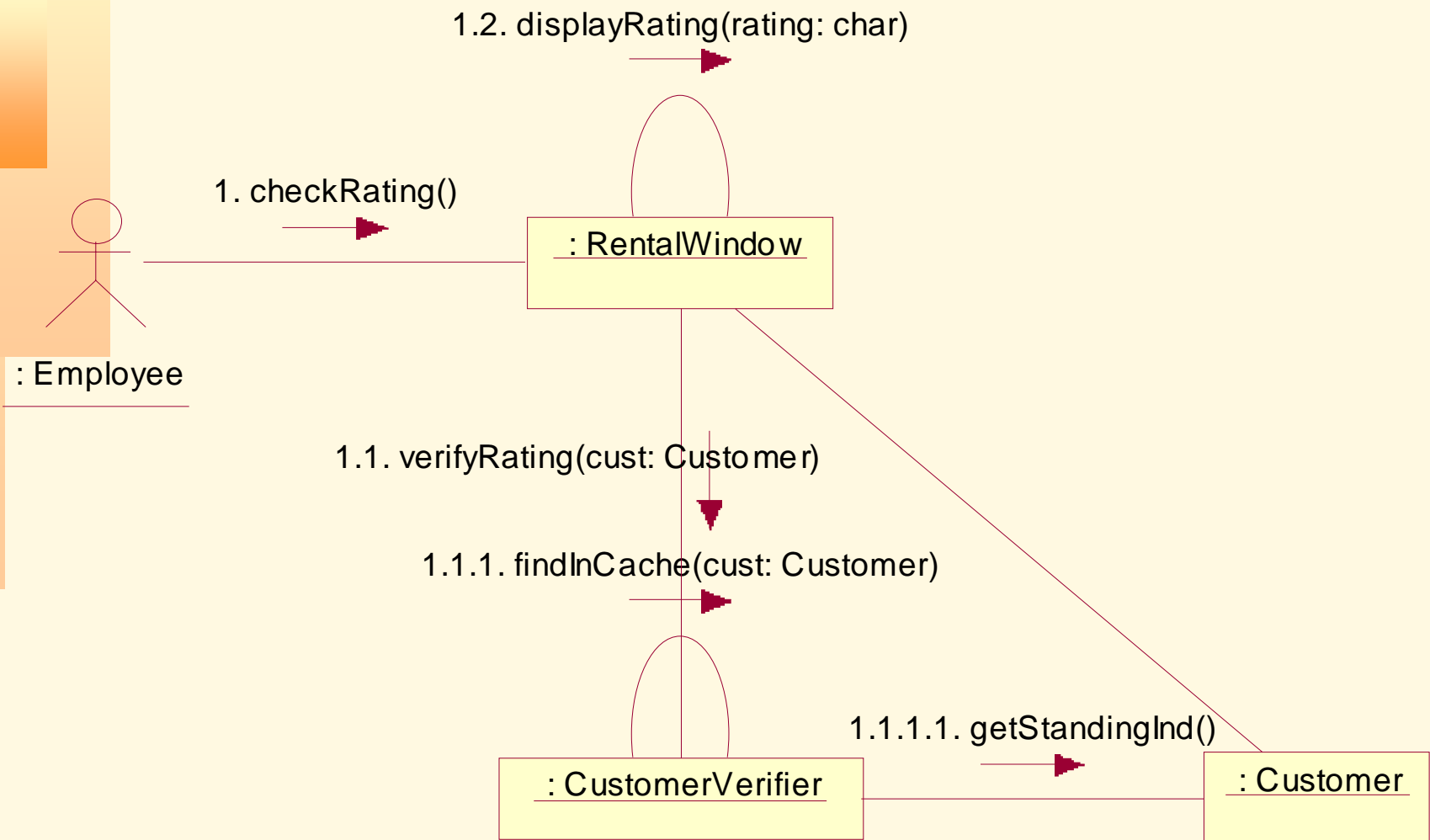
Interactions

- **Interaction** – set of messages in some behavior that are exchanged between objects across links
- **Sequence Diagram**
 - Objects - horizontal dimension
 - Message sequence - top to bottom on vertical dimension
 - Each vertical line - the object's **lifeline**
 - **Activation** – a method activated on a lifeline
 - Arrow - **message** from a calling object (sender) to an operation (method) in the called object (target)
 - Showing the **return** of control from the target to the sender is not necessary
 - **Iteration marker** – an asterisk in front of the message label – indicates iterating over a collection

Sequence diagram for "verify customer"



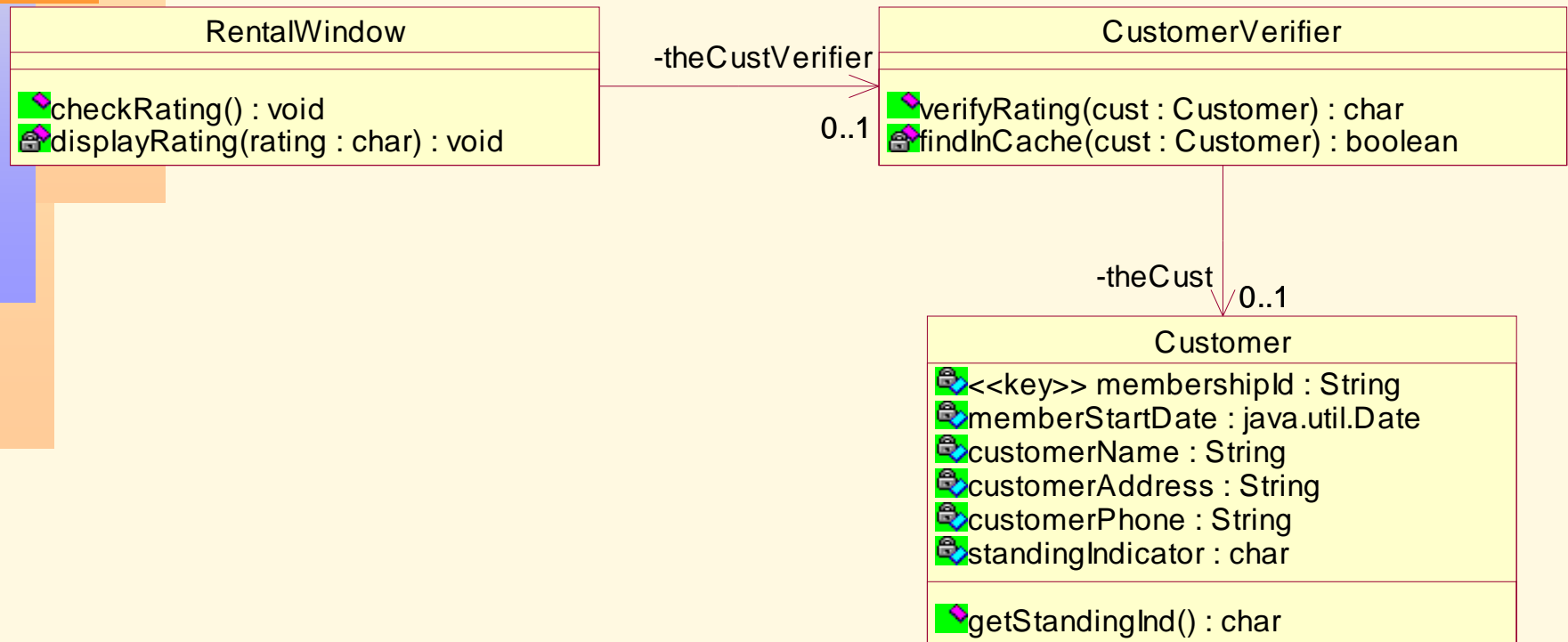
Collaboration diagram for "verify customer"



Operations

- *Examining the interactions can lead to the discovery of operations*
 - *Each **message** invokes an operation in the called object*
 - *The operation has the same name as the message*
- *Similarly, the presence of a message in a sequence diagram stipulates the need for an association in the class diagram*

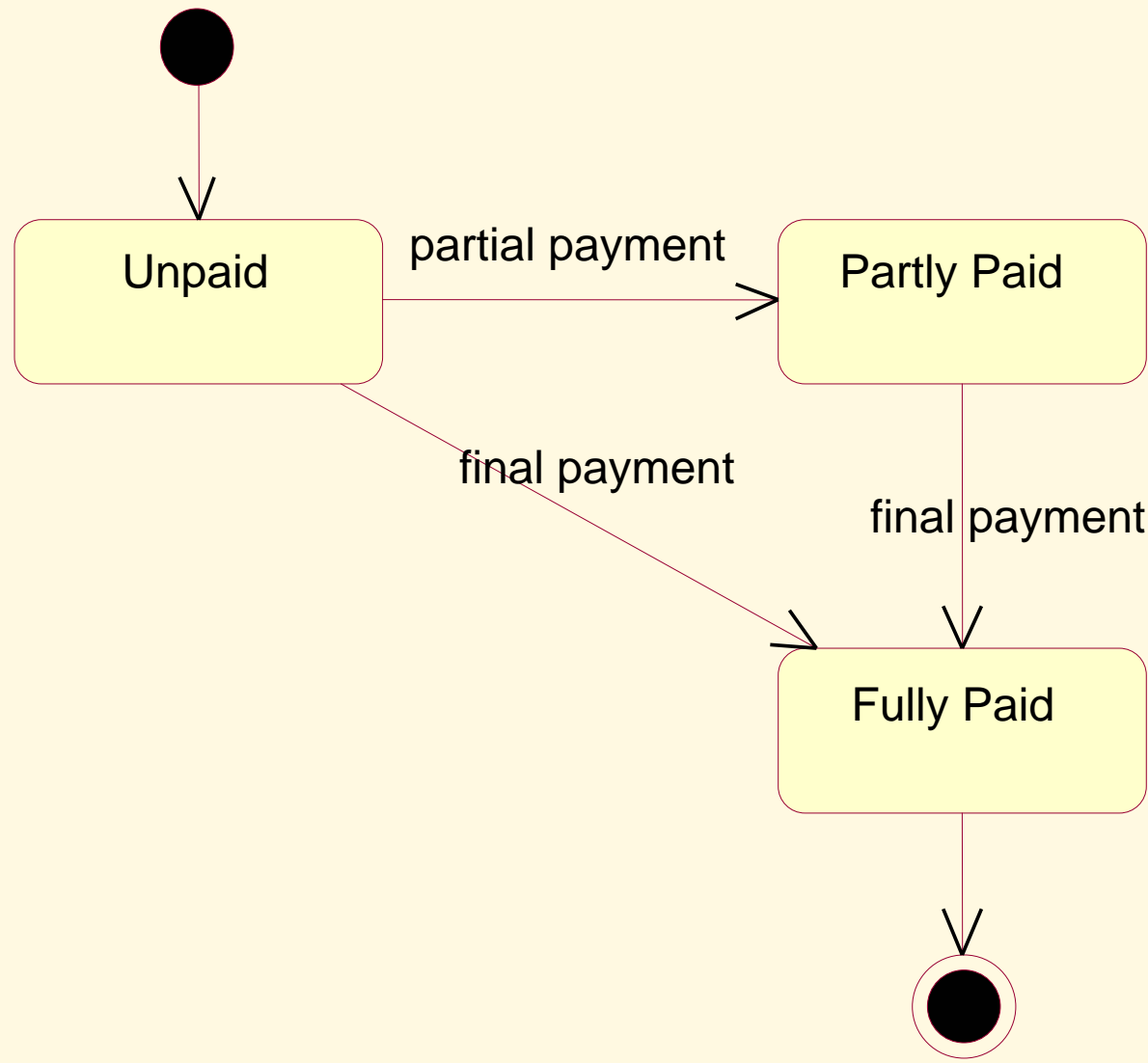
Adding operations to classes



Statechart modeling

- *Captures dynamic changes of class states – the life history of the class*
- *These dynamic changes describe typically the behavior of an object across several use cases*
- **State** of an object – designated by the current values of the object's attributes
- **Statechart diagram** – a bipartite graph of
 - **states** (rounded rectangles) and
 - **transitions** (arrows) caused by **events**
- *Statechart diagram is a model of **business rules***

States and transitions

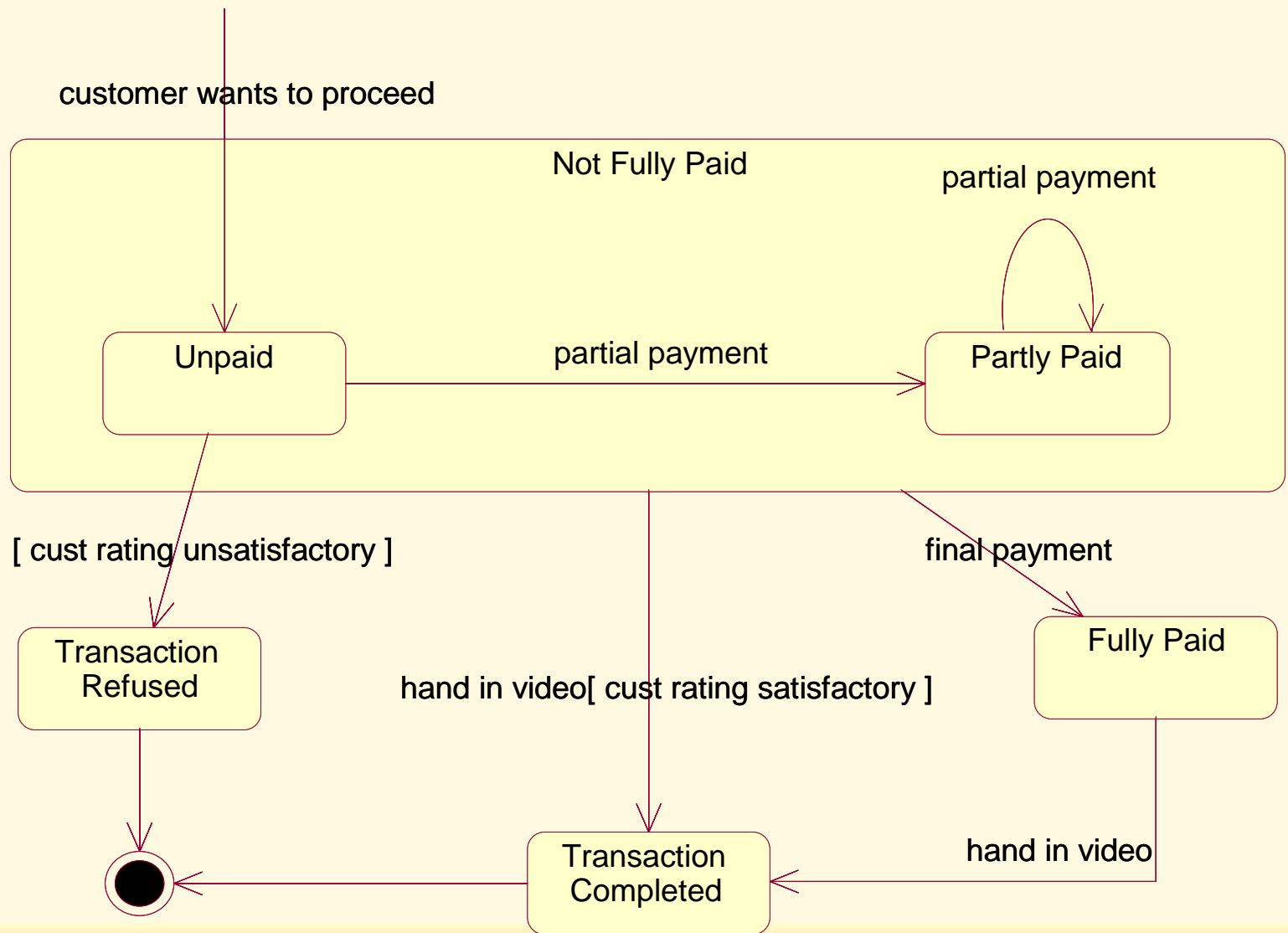


Statechart diagram

- *Normally attached to a class, but can be attached to other modeling concepts, e.g. a use case*
- *When attached to a class, the diagram determines how objects of that class react to events*
 - *Determines – for each object state – what **action** the object will perform when it receives an event*
 - *The same object may perform a different action for the same event depending on the object's state*
 - *The action's execution will typically cause a state change*
- *The complete description of a **transition** consists of three parts*

`event (parameters) [guard] / action`

Statechart diagram - excerpt



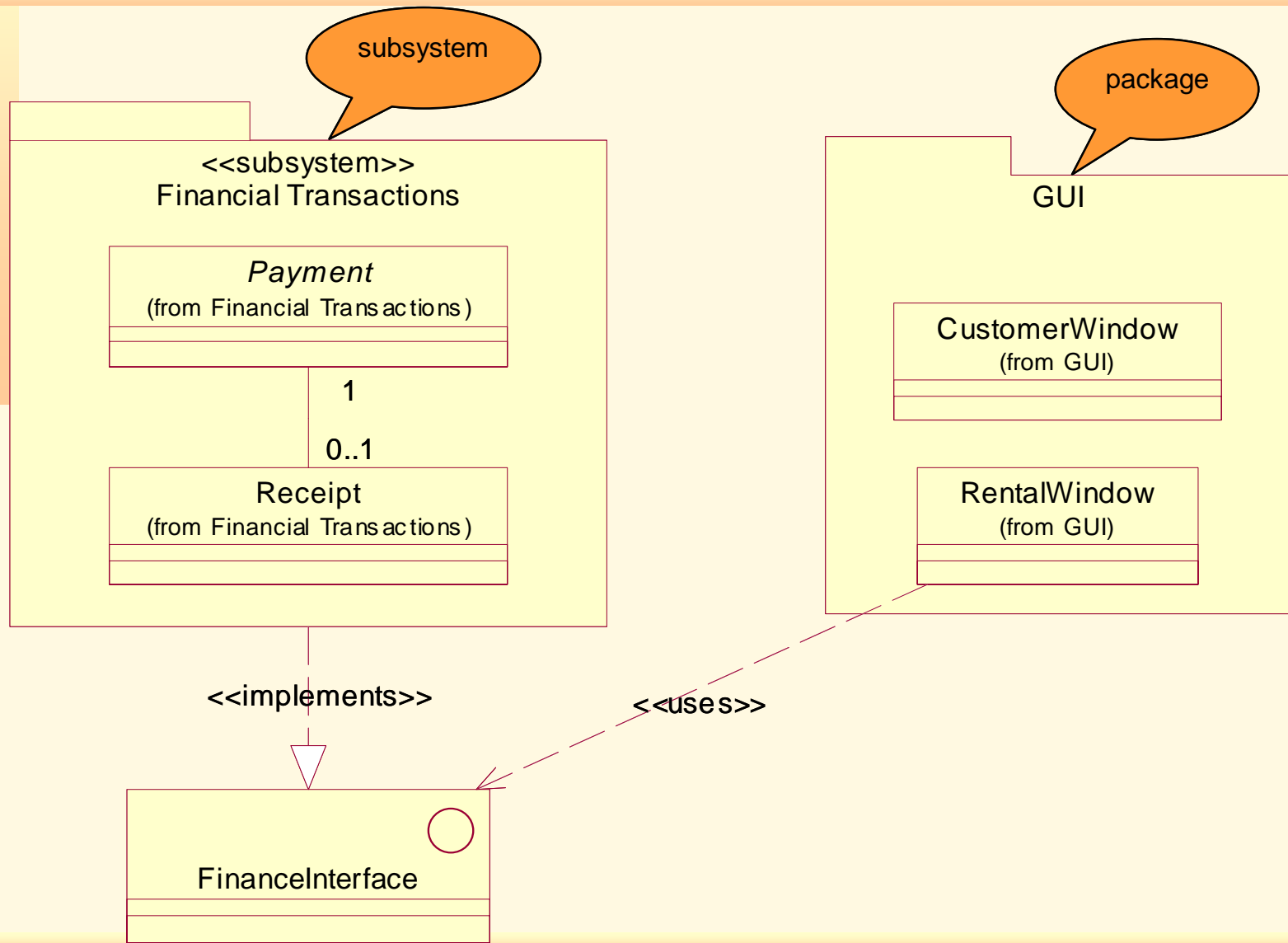
Implementation models

- **Component** - *modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces*
 - → **component diagram**
- **Node** - *physical object that represents a processing resource, generally, having at least a memory and often processing capability as well*
 - *include computing devices but also human resources or mechanical processing resources*
 - → **deployment diagram**

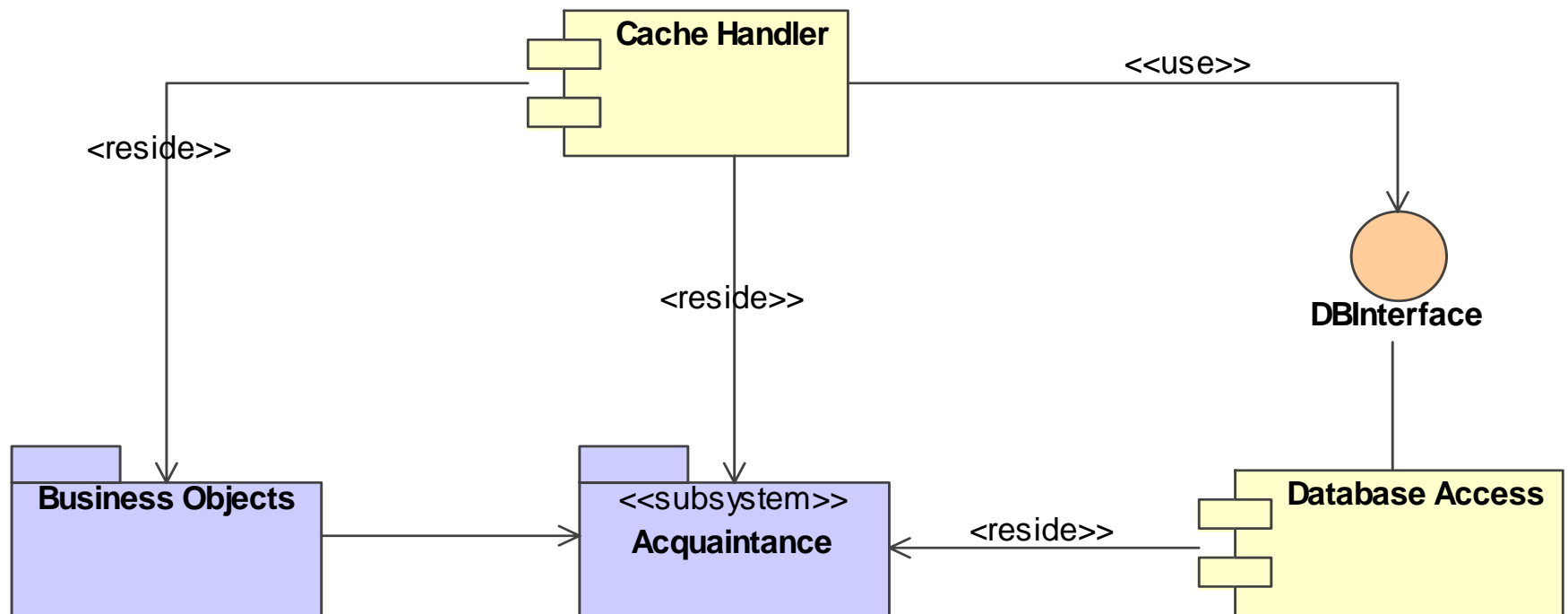
Subsystems and packages

- *“divida et impera” (divide-and-conquer)*
- **Subsystem**
 - *encapsulates some part of system behavior*
 - *its services are the result of the services provided by its classes*
 - *its services are defined using interfaces*
- **Package**
 - *grouping of modeling elements*
 - *like subsystem, its services are the result of the services provided by its classes*
 - *unlike subsystem, its services are not exposed using interfaces*
- *“The difference between a subsystem and a package is that, for a package, a client asks some element inside the package to fulfill a behavior; for a subsystem, a client asks the subsystem itself to fulfill the behavior.” from Ferm (2003, p.2)*

Subsystems and packages

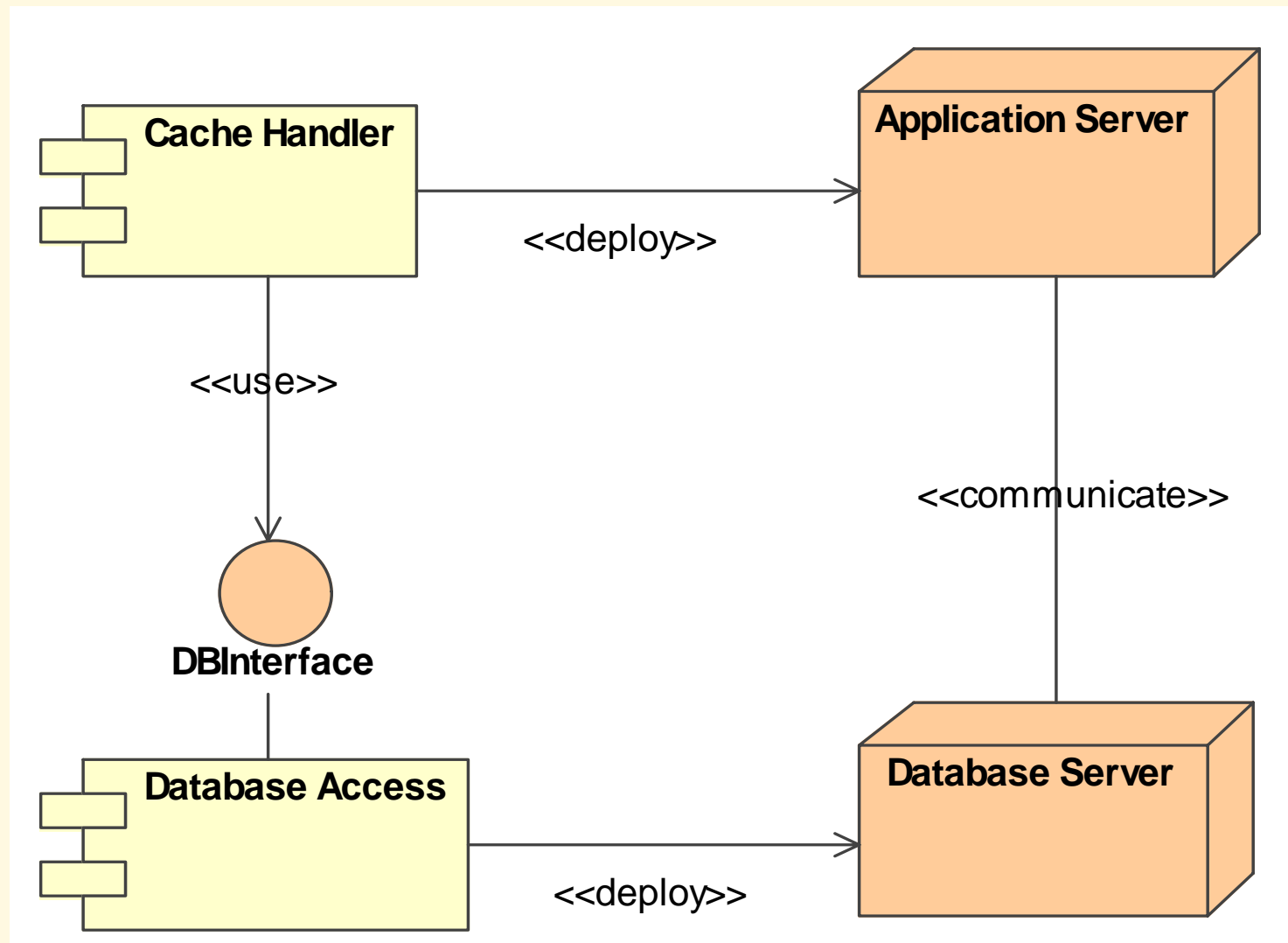


Component diagram



- **«reside» dependency** to other implementation-time modeling element
 - the client component depends on the supplier element
 - the supplier element resides in the component (admittedly, the direction of the arrow is somewhat counterintuitive as far as “what resides in what” is concerned).
 - the same modeling element may reside in more than one component

Deployment diagram



Summary

- **Three kinds of models** – state, behavior and state change models.
- The **use case model** is the main UML representative and the focal point of behavior modeling.
- The **activity model** can graphically represent the flow of events of a use case.
- **Class modeling** integrates and embodies all other modeling activities.
- **Interaction modeling** captures interactions between objects needed to execute a use case or part of it.
- A **statechart model** specifies dynamic changes in a class.
- UML provides **component diagrams** and **deployment diagrams** as two tools for architectural/structural modeling of physical implementation of the system.