MACIASZEK, L.A. (2007):
Requirements Analysis and System Design, 3rd ed.
Addison Wesley, Harlow England
ISBN 978-0-321-44036-5

Chapter 3
Fundamentals of Visual Modeling

Topics

- Use case view
- Activity view
- Structure view
- Interaction view
- State machine view
- Implementation view

Kinds of UML (logical) 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
    - activity
    - communication
- State change model
  - dynamic view of the system
    - models object evolution over time
  - state machine diagram

1. The use case view

- the focal point of behavior modeling
- presents the dynamic view of the system
- it models function requirements
- represents business transactions, operations, and algorithms on data

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

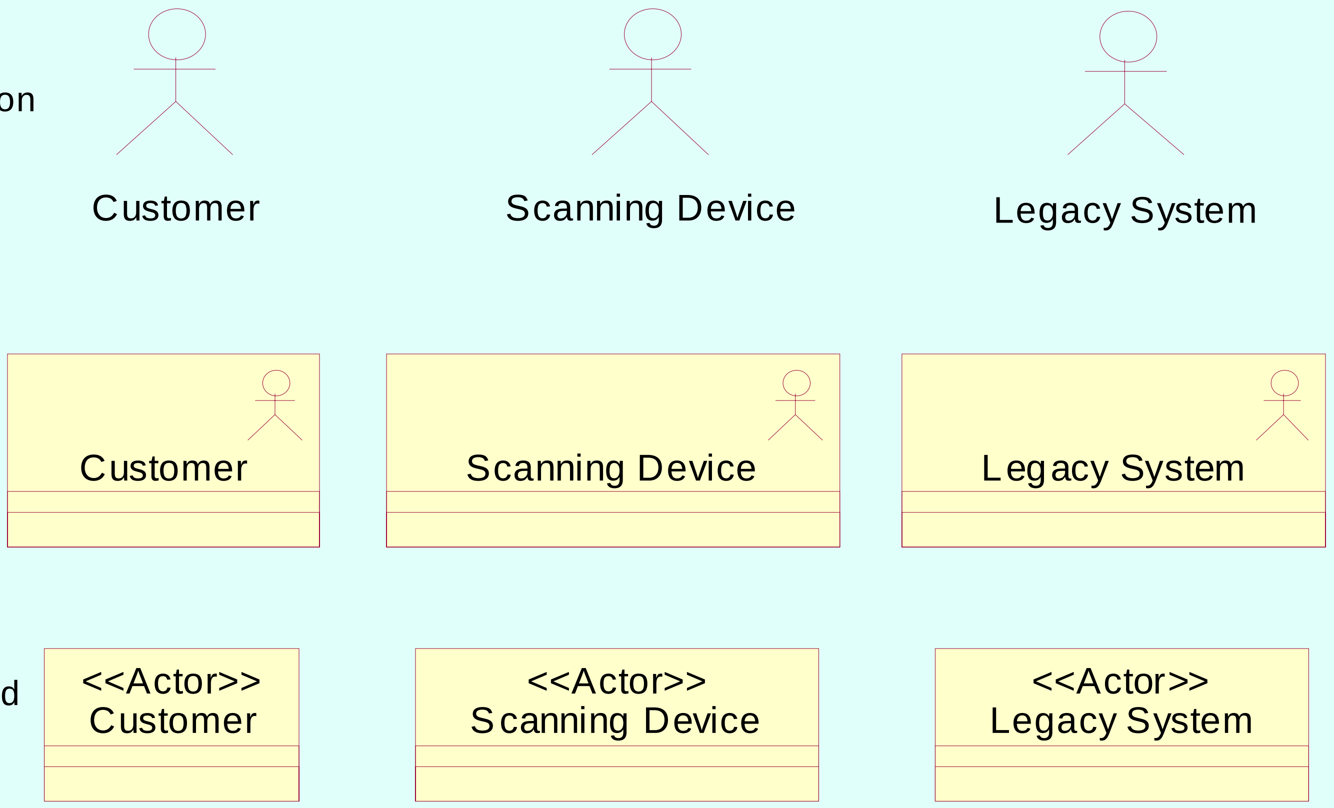
Req#	*Requirement*	*Actor*	*Use case*
1	*Before a video can be rented out, the system confirms customer's identity and standing by swiping over scanner his/her Video Store membership card.*	*Customer, Employee*	*Scan Membership Card*
2	*A video tape or disk can be swiped over scanner to obtain its description 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

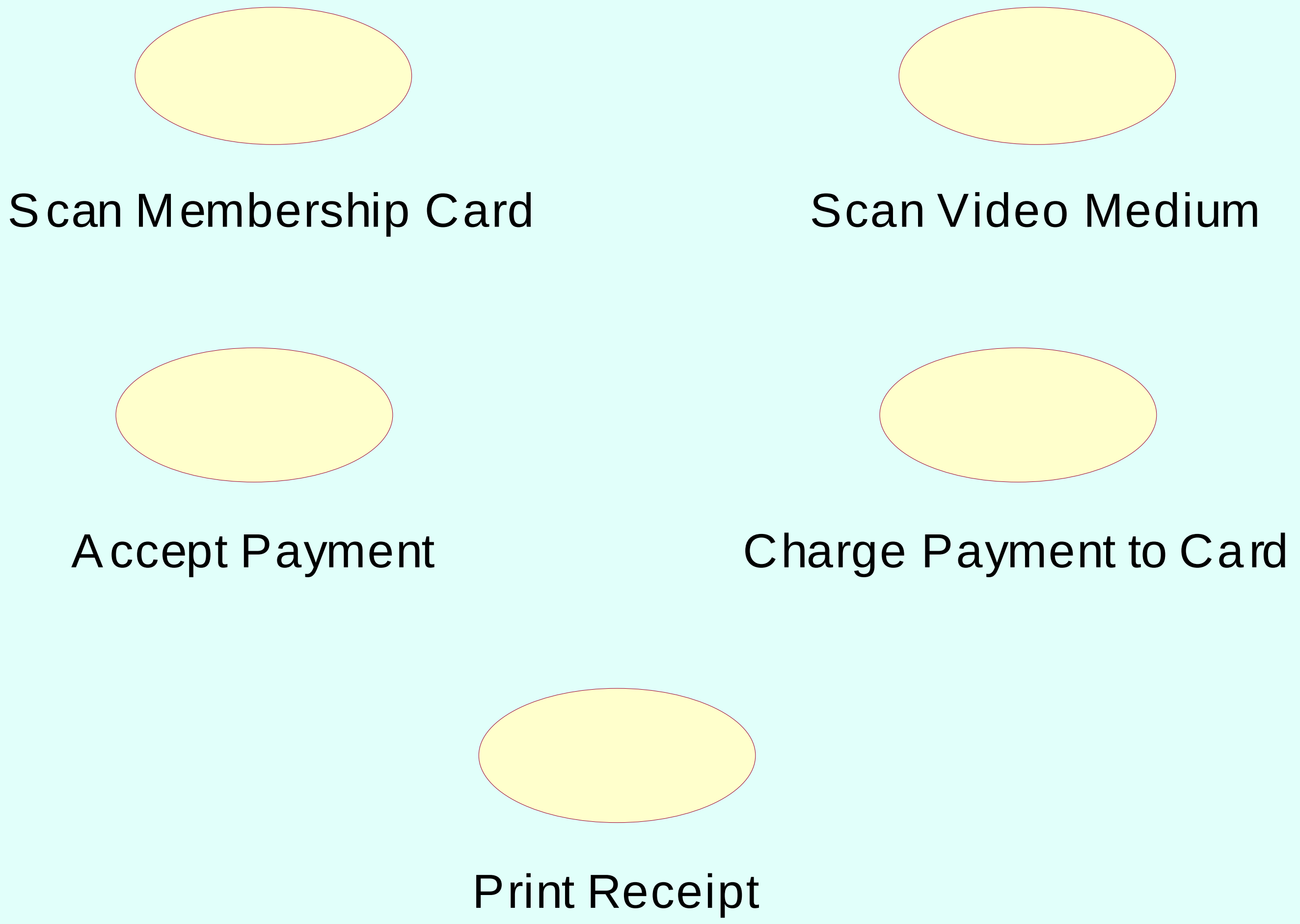
Stick-person icons

Decorated classes

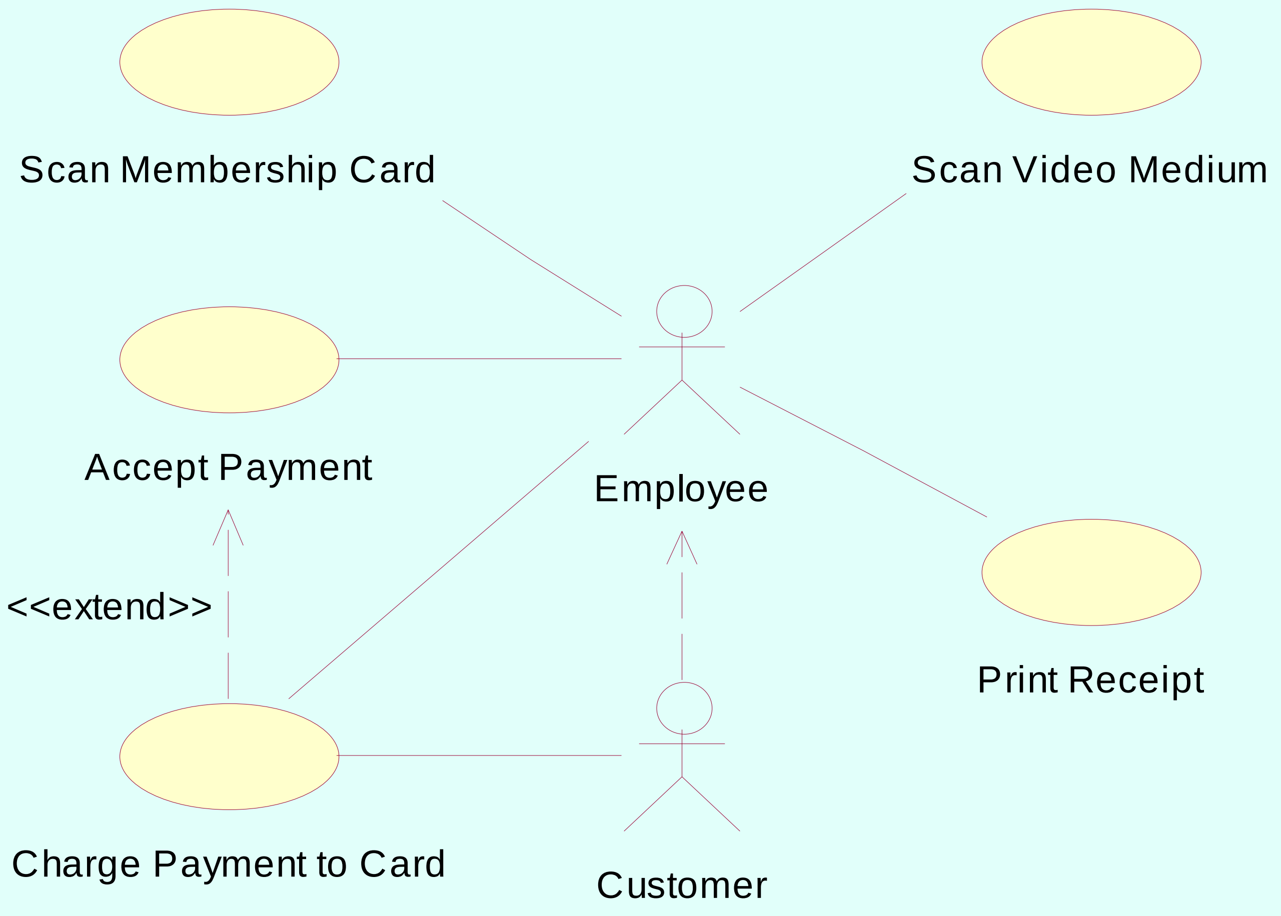
Stereotyped classes



Use cases

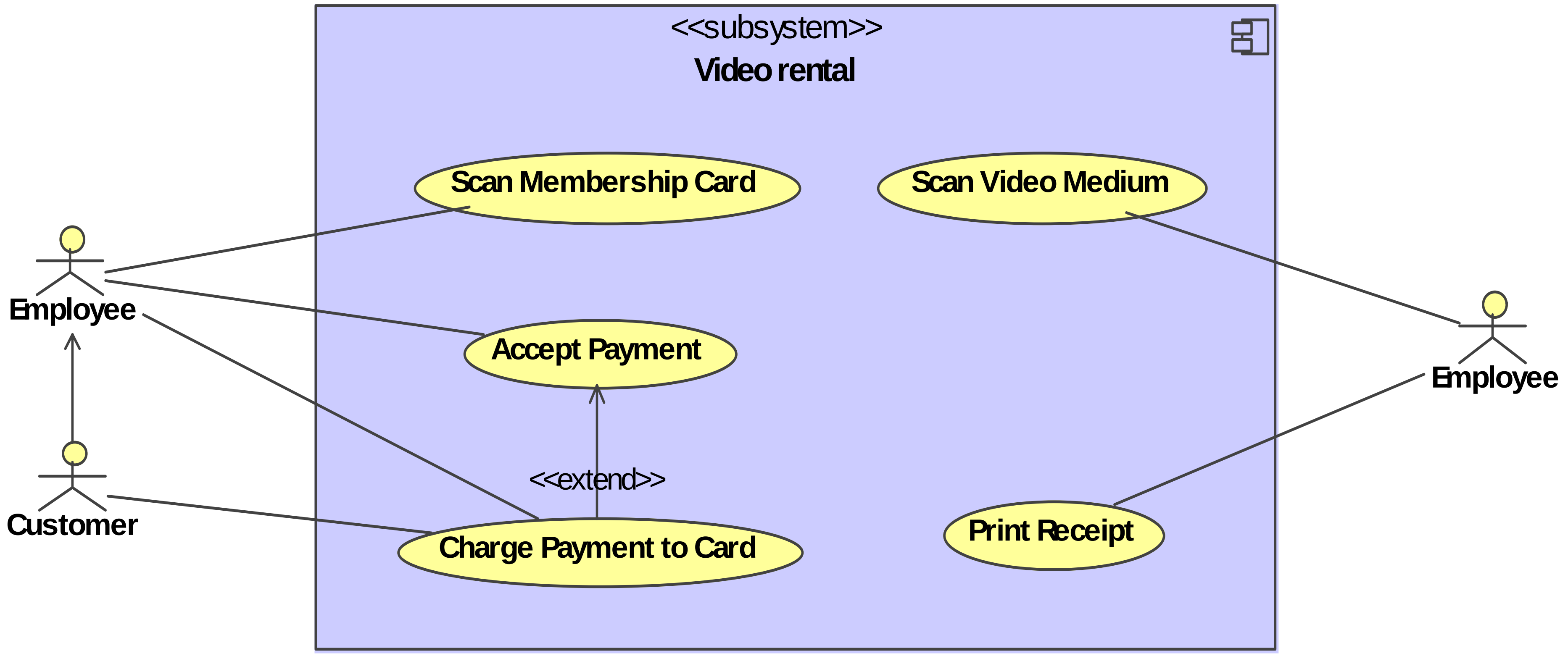

Scan Membership Card
Scan Video Medium
Accept Payment
Charge Payment to Card
Print Receipt

Use case diagram



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

Use case diagram featuring a subject



A *subject* is any group of use cases for which a use case model is drawn (e.g. a subsystem, component, class)

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 `Customer` for a video rental.
Actors	Employee, Customer.
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 `Employee` asks the system to verify ...
Postconditions	If the use case was successful, the payment is recorded in the system’s database. Otherwise, ...

Review Quiz 3.1

1. What are the most important behavioral modeling techniques?
2. Is the use case diagram the same as the use case specification?

2. The activity view

- represents a behavior that is composed of individual elements
- the behavior may be a specification of a use case
- it may also be a piece of functionality that can be reused in many places

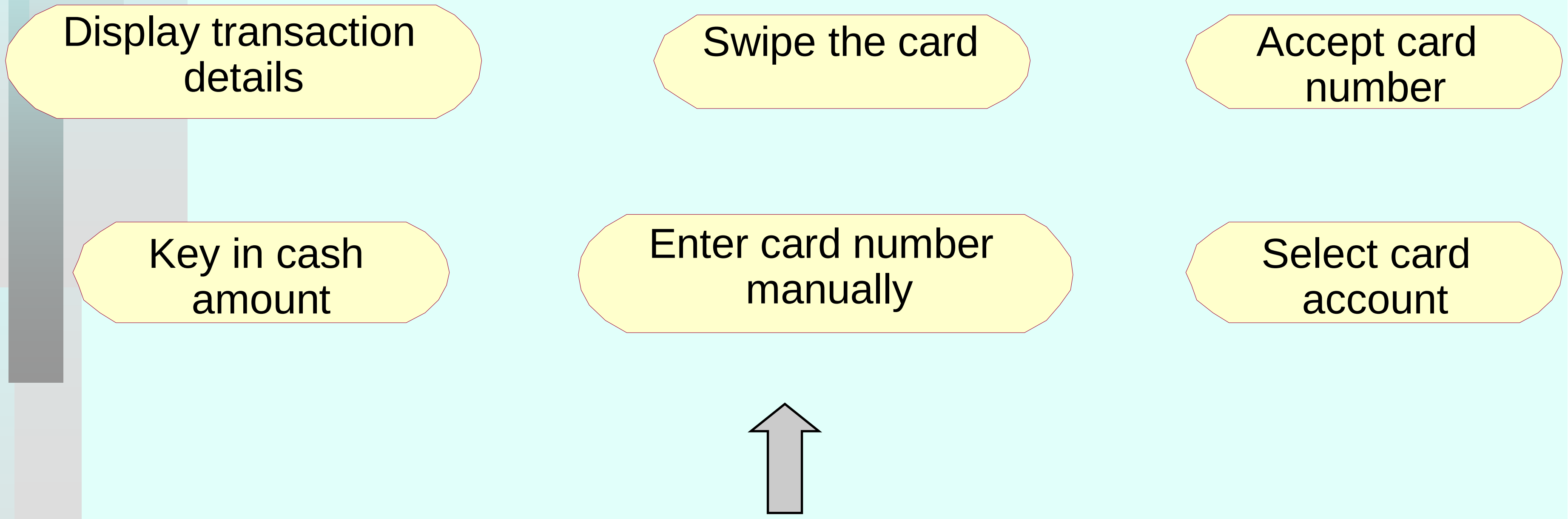
Activity modeling

- **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
    - – at a much lower level of abstraction to design complex sequential algorithms or to design concurrency in multi-threaded applications
- Shows the steps of a computation
  - Each step is a **state** of doing something
  - Execution steps are called **actions**
  - Depicts which steps are executed in sequence and which can be executed concurrently
  - **Control flow** – the flow of control from one action to the next

Finding actions in use case flows

No.	Use case statement	Action
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 then requests the `Customer` to type the card's PIN number, select debit or credit account, and transmit the payment. Once the payment has been confirmed electronically by the card provider, the system records the payment as made.	Swipe the card; Accept card number; Select card account; Confirm transaction

Actions

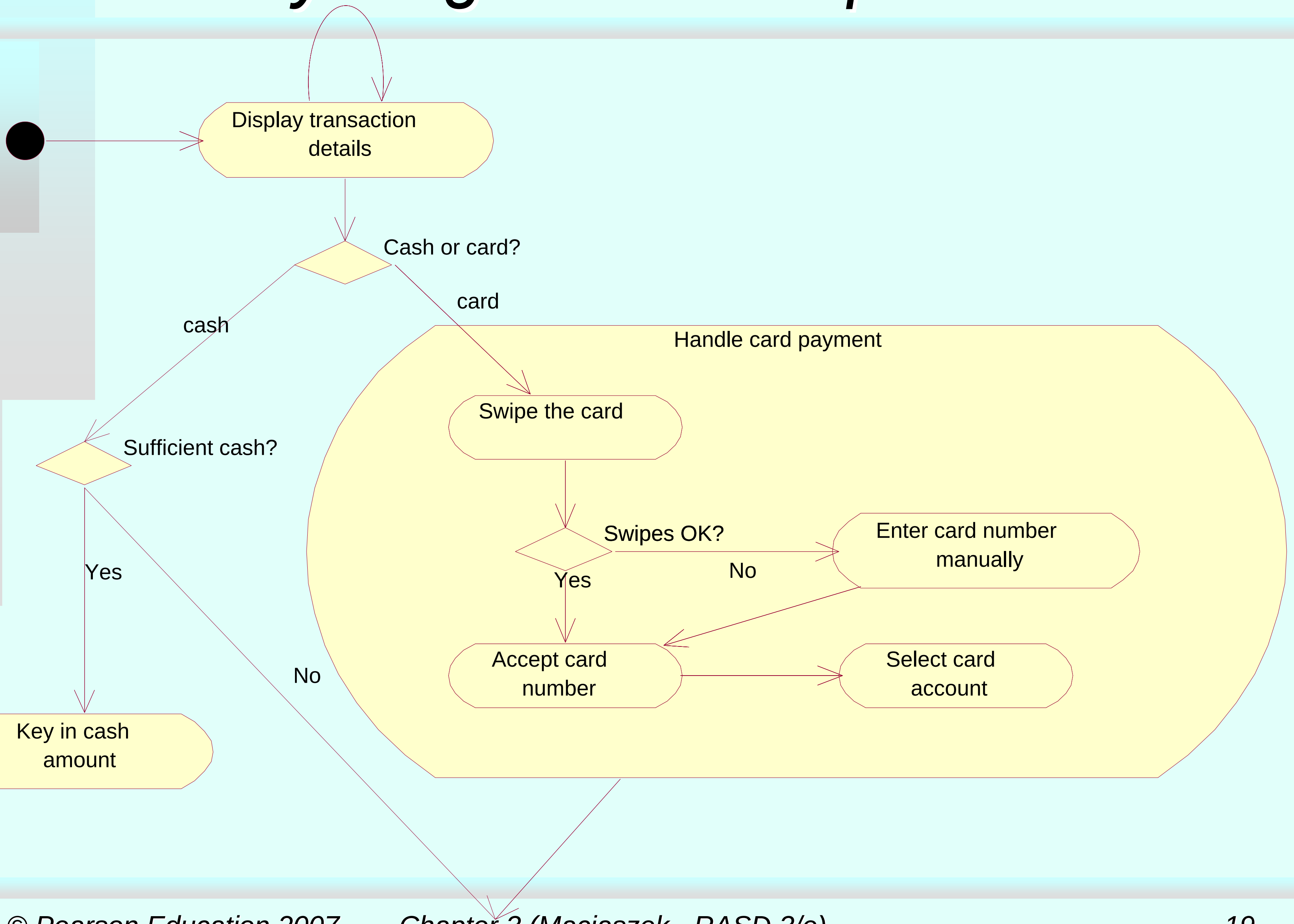


- The `Customer`’s card does not swipe properly through the scanner. After three unsuccessful attempts, the Employee enters the card number manually .

Activity diagram

- **Activity Diagram** shows transitions between actions
- A solid filled circle represents the start of an activity.
- The end of an activity is shown using a 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



Review Quiz 3.2

1. Can an activity model be used as a specification of a use case?
2. Flows in an activity diagram connect actions and other diagram nodes. What are these other nodes?

3. The structure view

- it represents data structures and their relationships
- it identifies operations that act on these data

Class modeling

- Captures the **static view** of the system– although it also identifies operations that act on data
- 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 forms or webpages) – **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 – **resource 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

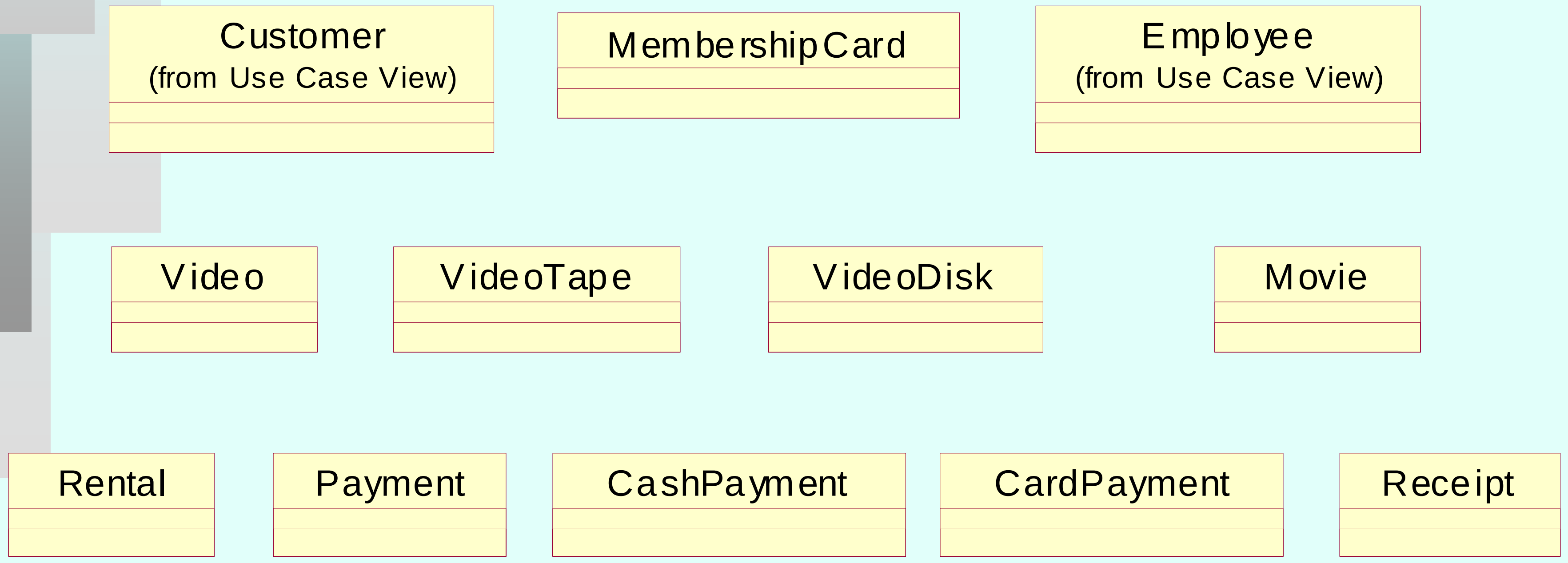
Assignment of requirements to entity classes

R#	Requirement	Entity class
1	Before a video can be rented out, the system confirms customer’s identity and standing by swiping his/her Video Store membership card over a scanner .	`Video, Customer, MembershipCard`
2	A video tape or disk can be swiped over the scanner to obtain its description and price (fee) as part of a 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 in cash or by 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`

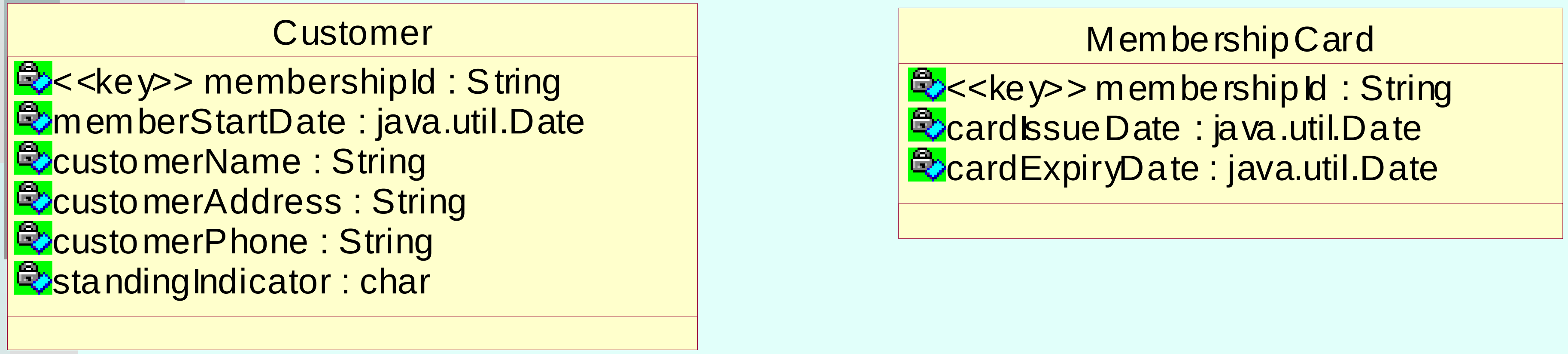
Classes

- Is this a class?
  - Is it 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?

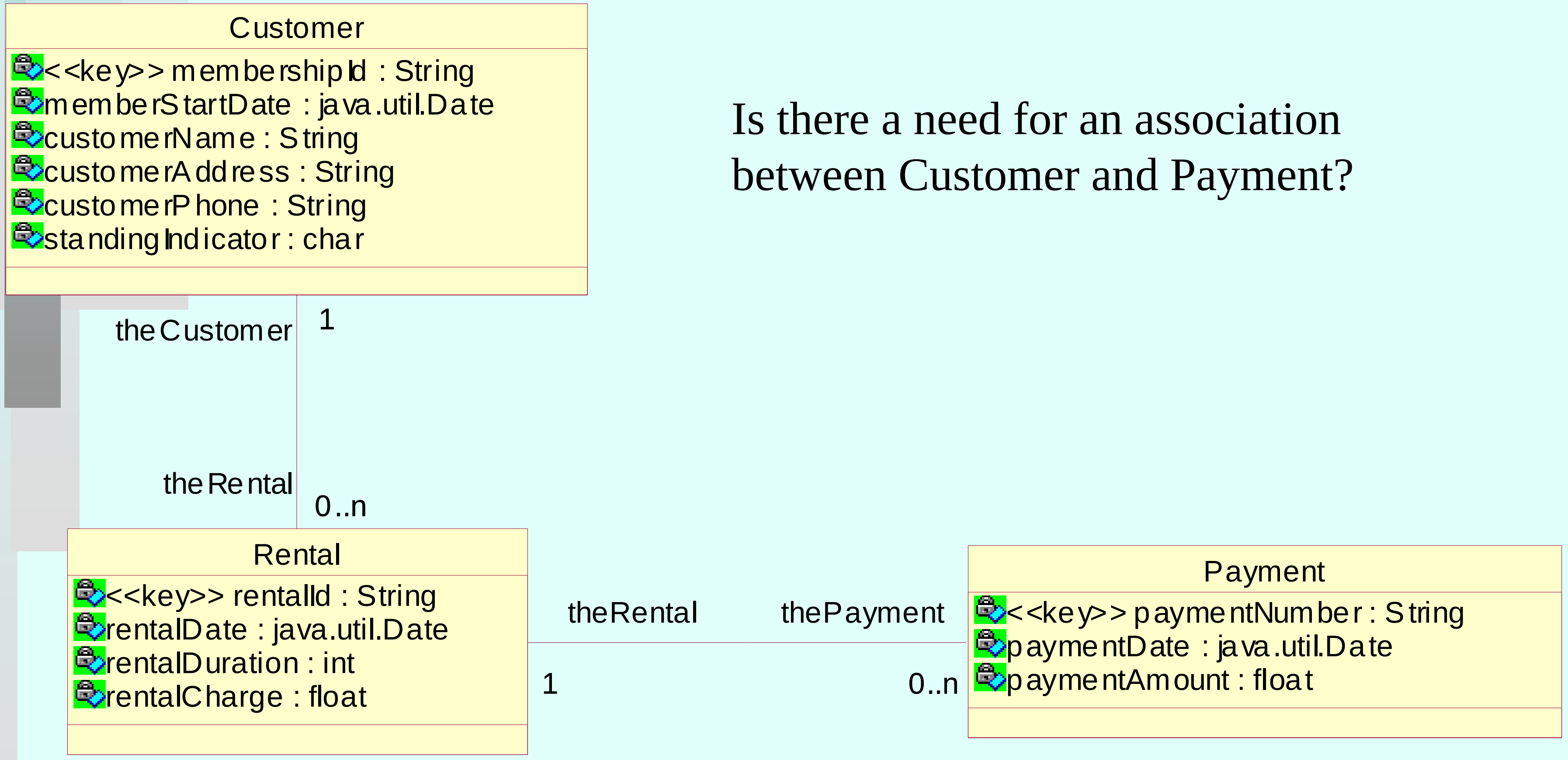
Classes


Customer
(from Use Case View)
MembershipCard
Employee
(from Use Case View)
Video
VideoTape
VideoDisk
Movie
Rental
Payment
CashPayment
CardPayment
Receipt

Attributes

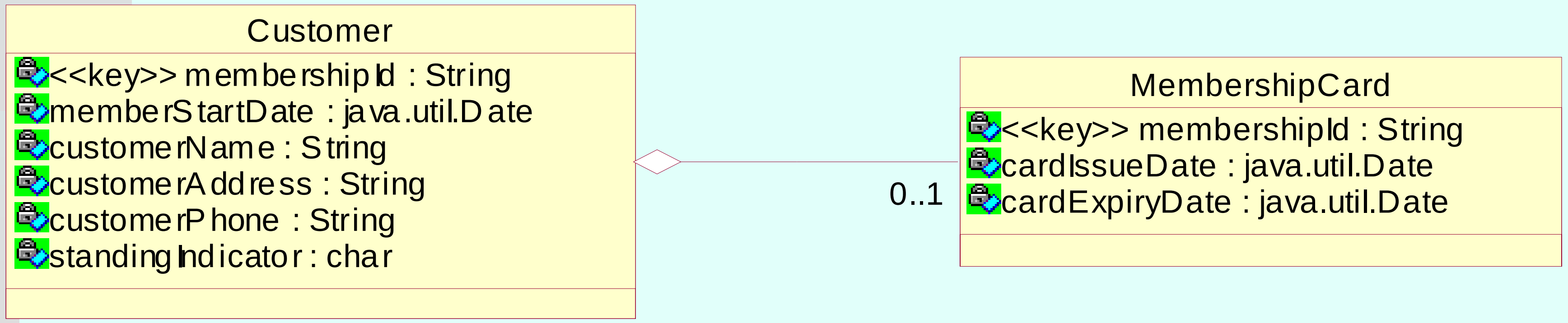

Customer
<<key>> membershipId : String
memberStartDate : java.util.Date
customerName : String
customerAddress : String
customerPhone : String
standingIndicator : char
MembershipCard
<<key>> membershipId : String
cardIssueDate : java.util.Date
cardExpiryDate : java.util.Date

Associations



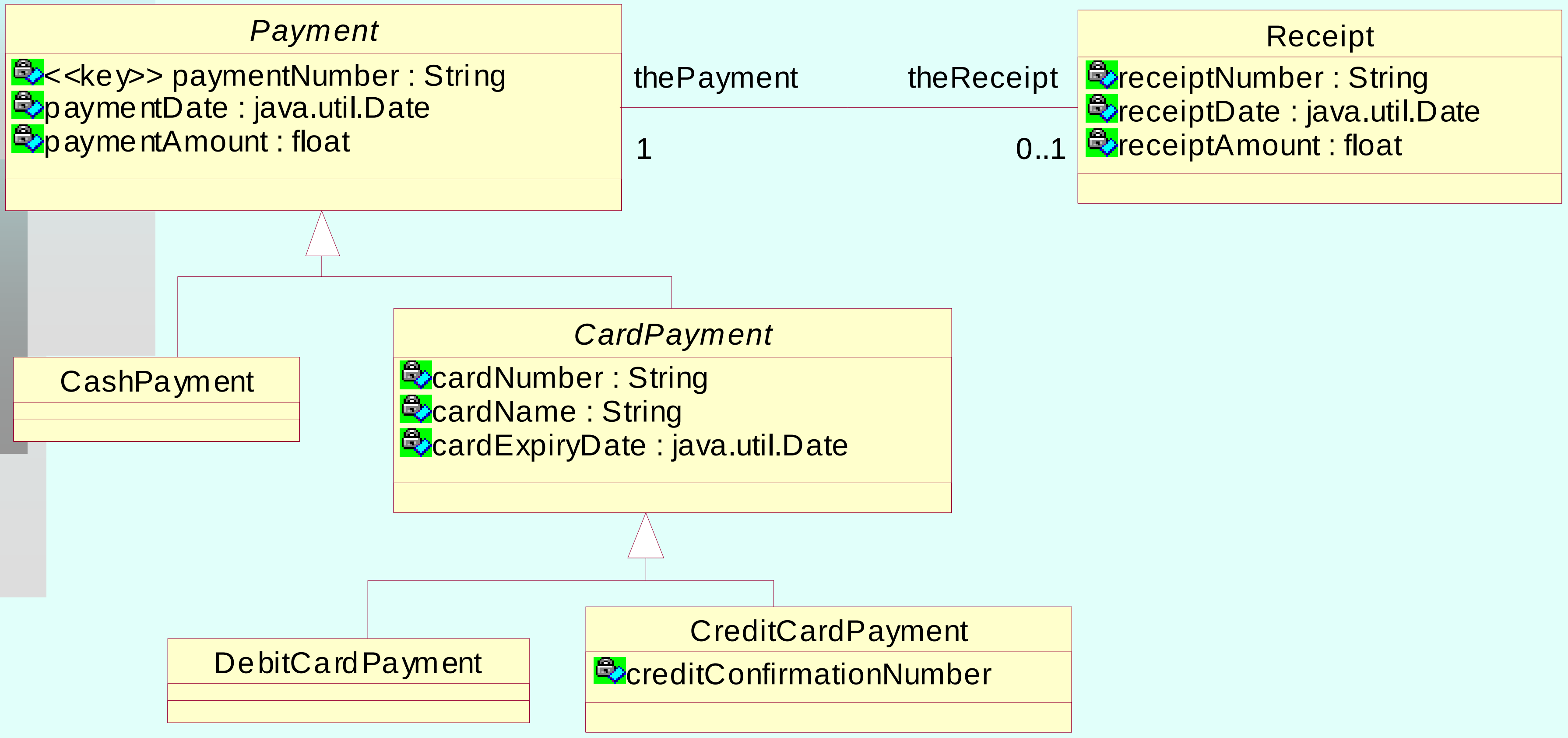
Is there a need for an association between Customer and Payment?

Aggregations

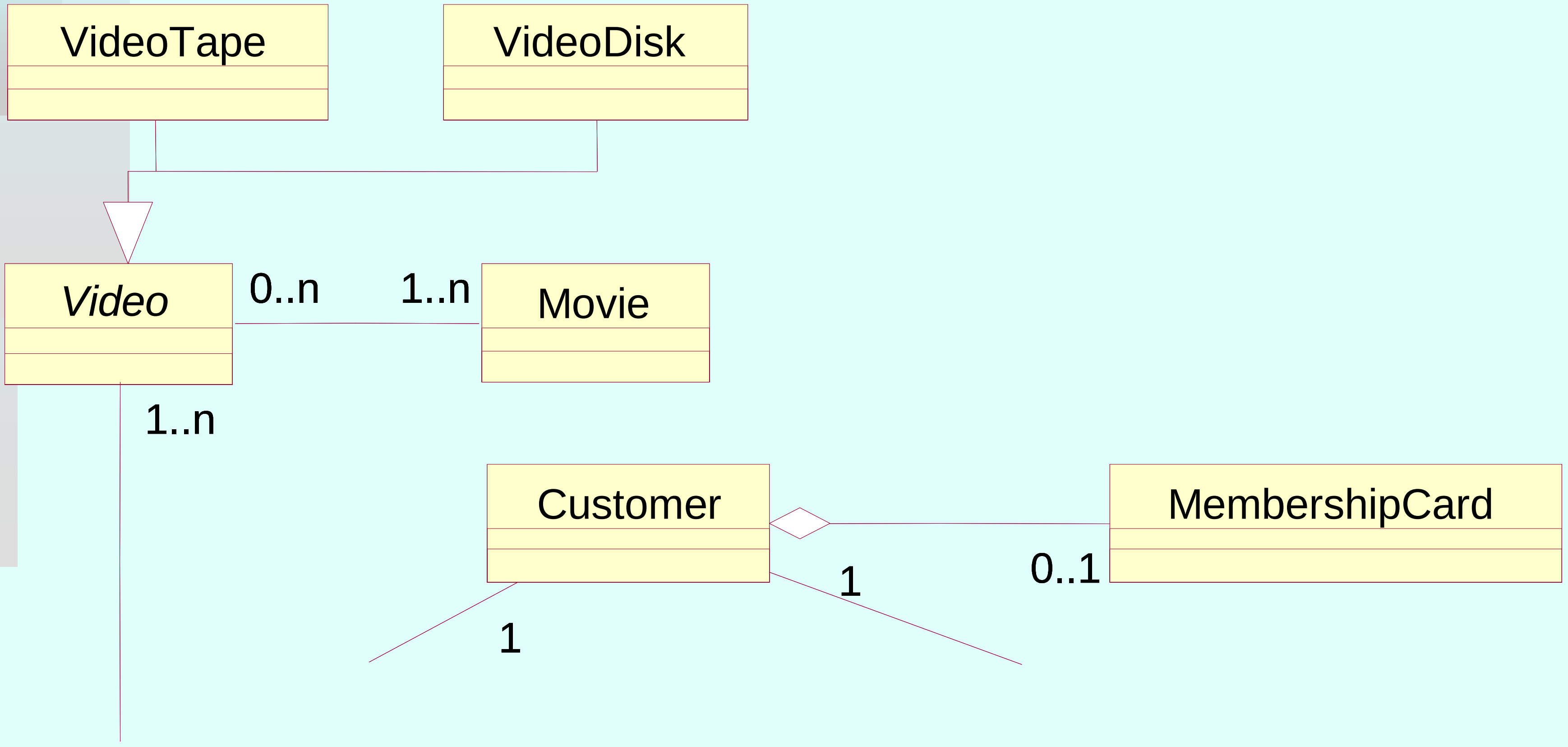


Customer
<<key>> membershipId : String
memberStartDate : java.util.Date
customerName : String
customerAddress : String
customerPhone : String
standingIndicator : char
0..1
MembershipCard
<<key>> membershipId : String
cardIssueDate : java.util.Date
cardExpiryDate : java.util.Date

Generalizations



Class diagram - excerpt



Review Quiz 3.3

1. Are the notions of an entity class and a business object synonymous?
2. Does the concept of multiplicity apply to aggregation?

4. The interaction view

- captures the interactions between objects, which need to communicate in order to execute a use case or part of it
- used in more advanced stages of requirements analysis, when a basic class model is known

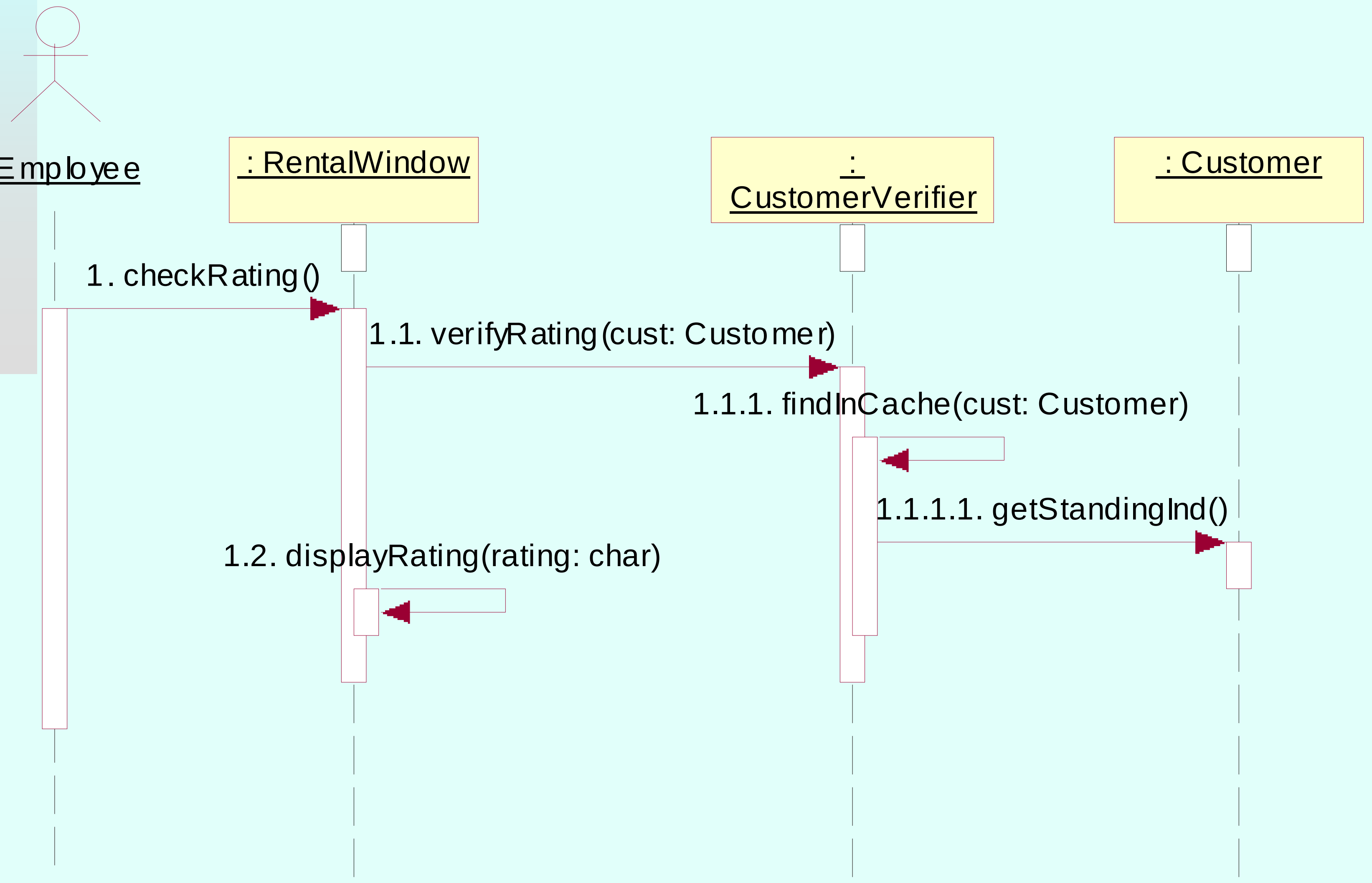
Interaction modeling

- Captures interactions between objects needed to execute a use case or part of it
- Shows the sequencing of events (messages) between collaborating objects
- Used typically 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
  - **Communication diagram** (called collaboration diagram prior to UML 2.0) – emphasize object relationships

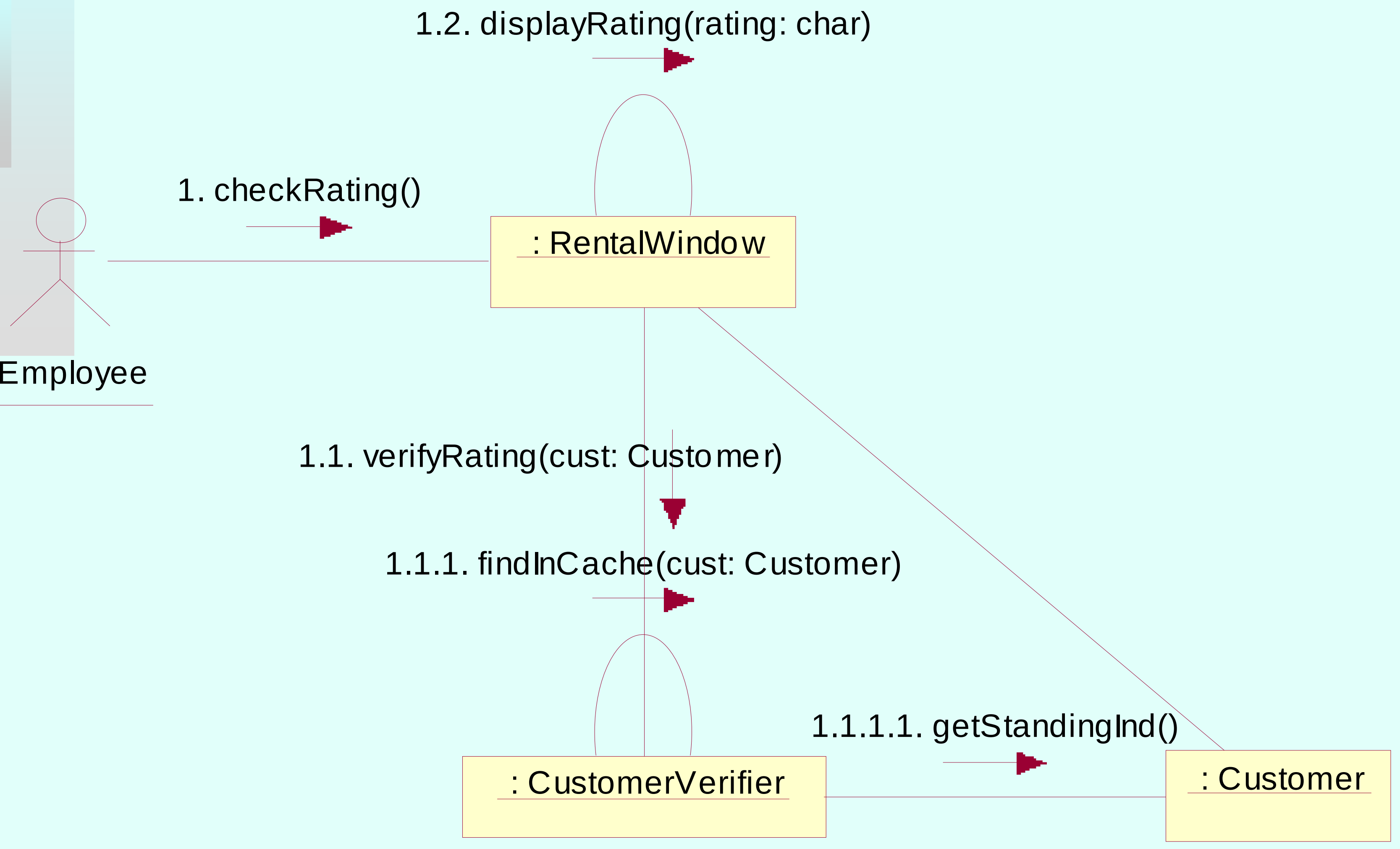
Interactions

- **Interaction** – set of events (messages) in some behavior that are exchanged between roles (objects) across links
- Sequence Diagram
  - Roles - horizontal dimension
  - Message sequence - top to bottom on vertical dimension
  - Each vertical line - the object's **lifeline**
  - **Activation** (*execution specification*) – 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"


: Employee
: RentalWindow
: CustomerVerifier
: Customer
1. checkRating()
1.1. verifyRating(cust: Customer)
1.1.1. findInCache(cust: Customer)
1.1.1.1. getStandingInd()
1.2. displayRating(rating: char)

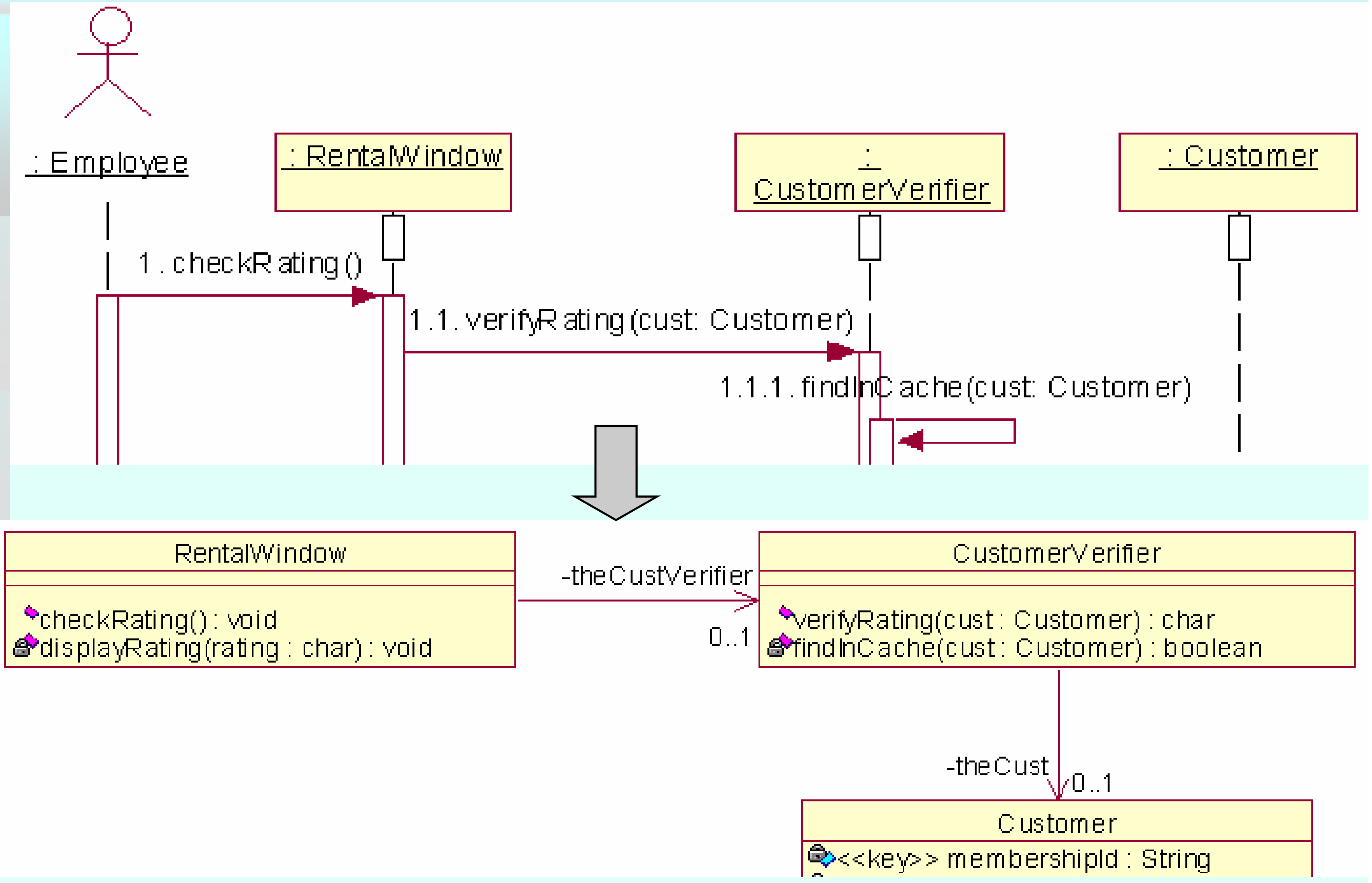
Collaboration diagram for “verify customer”


1.2. displayRating(rating: char)
1. checkRating()
: RentalWindow
: Employee
1.1. verifyRating(cust: Customer)
1.1.1. findInCache(cust: Customer)
1.1.1.1. getStandingInd()
: CustomerVerifier
: Customer

Methods (operations)

- Examining the interactions can lead to the discovery of methods
  - Each **message** invokes a method on the called object
  - The method has the same name as the message
- The presence of a message in a sequence diagram stipulates the need for an association in the class diagram (in particular with regard to entity classes)

Adding methods to classes



: Employee
: RentalWindow
: CustomerVerifier
: Customer
1 . checkRating ()
1 .1 . verifyRating (cust: Customer)
1 .1 .1 . findInCache (cust: Customer)
RentalWindow
checkRating() : void
displayRating(rating : char) : void
-theCustVerifier
0..1
CustomerVerifier
verifyRating(cust : Customer) : char
findInCache(cust : Customer) : boolean
-theCust
0..1
Customer
<<key>> membershipId : String

Review Quiz 3.4

1. Is the modeling element of lifeline present in sequence diagrams or in communication diagrams?
2. Are the concepts of message and method the same?

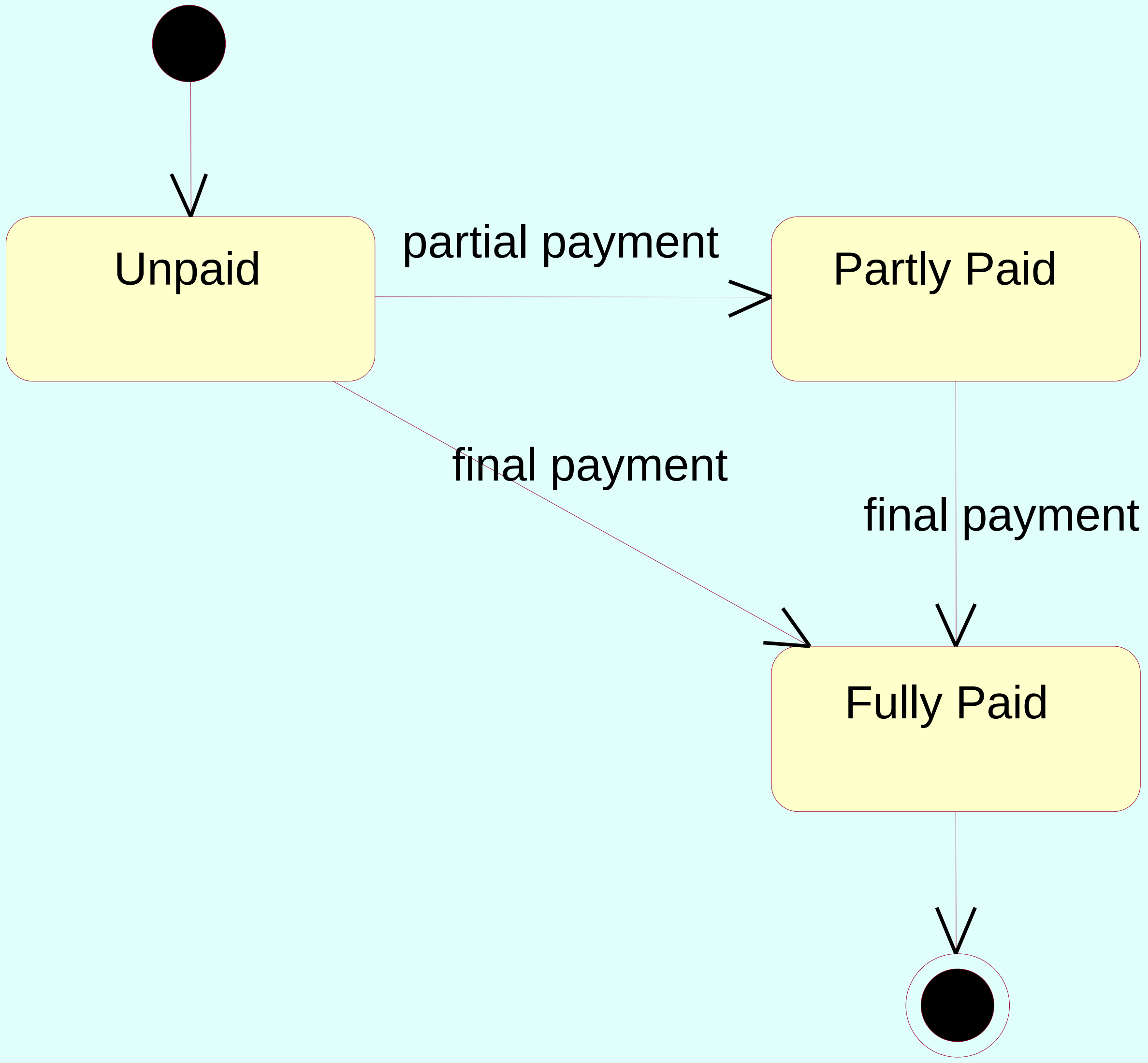
5. The state machine view

- specifies dynamic changes in a class
- describes various states in which objects of the class can be

State machine 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
- **State machine diagram** – a bipartite graph of
  - **states** (rounded rectangles) and
  - **transitions** (arrows) caused by **events**
- State machine diagram is a model of **business rules**

States and events

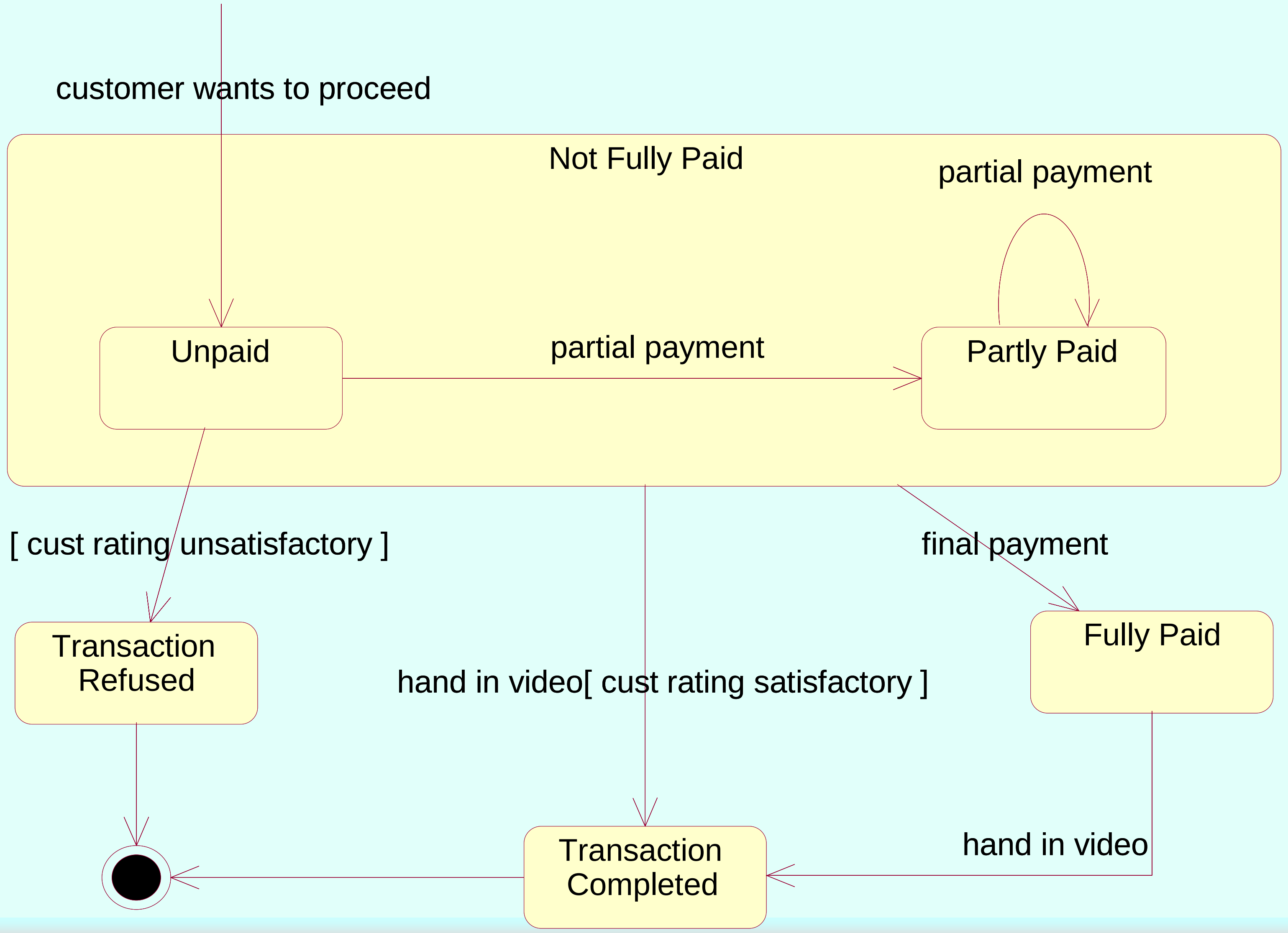


State machine diagram

- 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



Review Quiz 3.5

1. Can the state of an object depend on that object’s association links?
2. How is a guard different from an event?

The implementation view

- architectural/structural modeling of the physical implementation of a system
- provides a connection between the logical and physical structure of the system

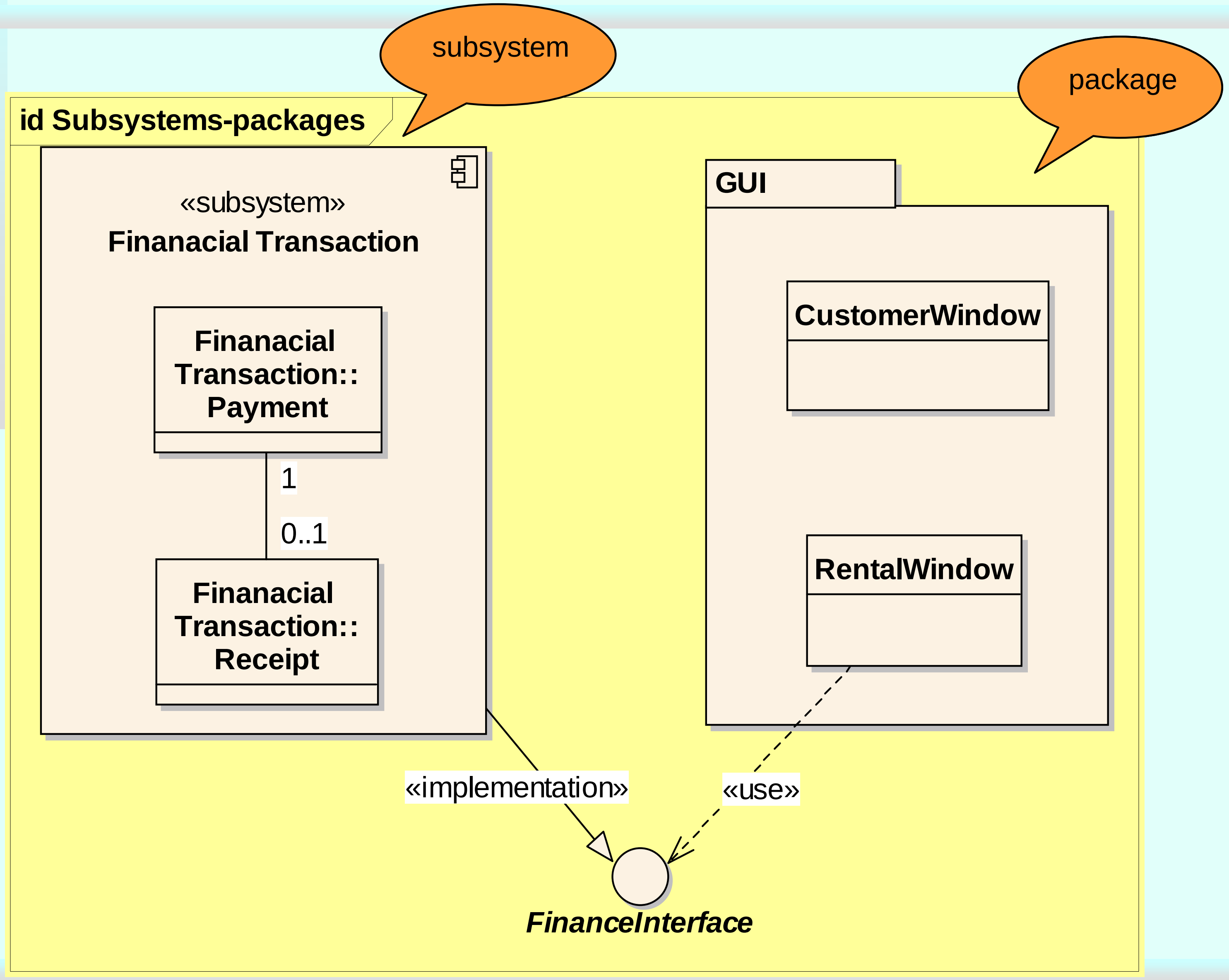
Implementation models

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

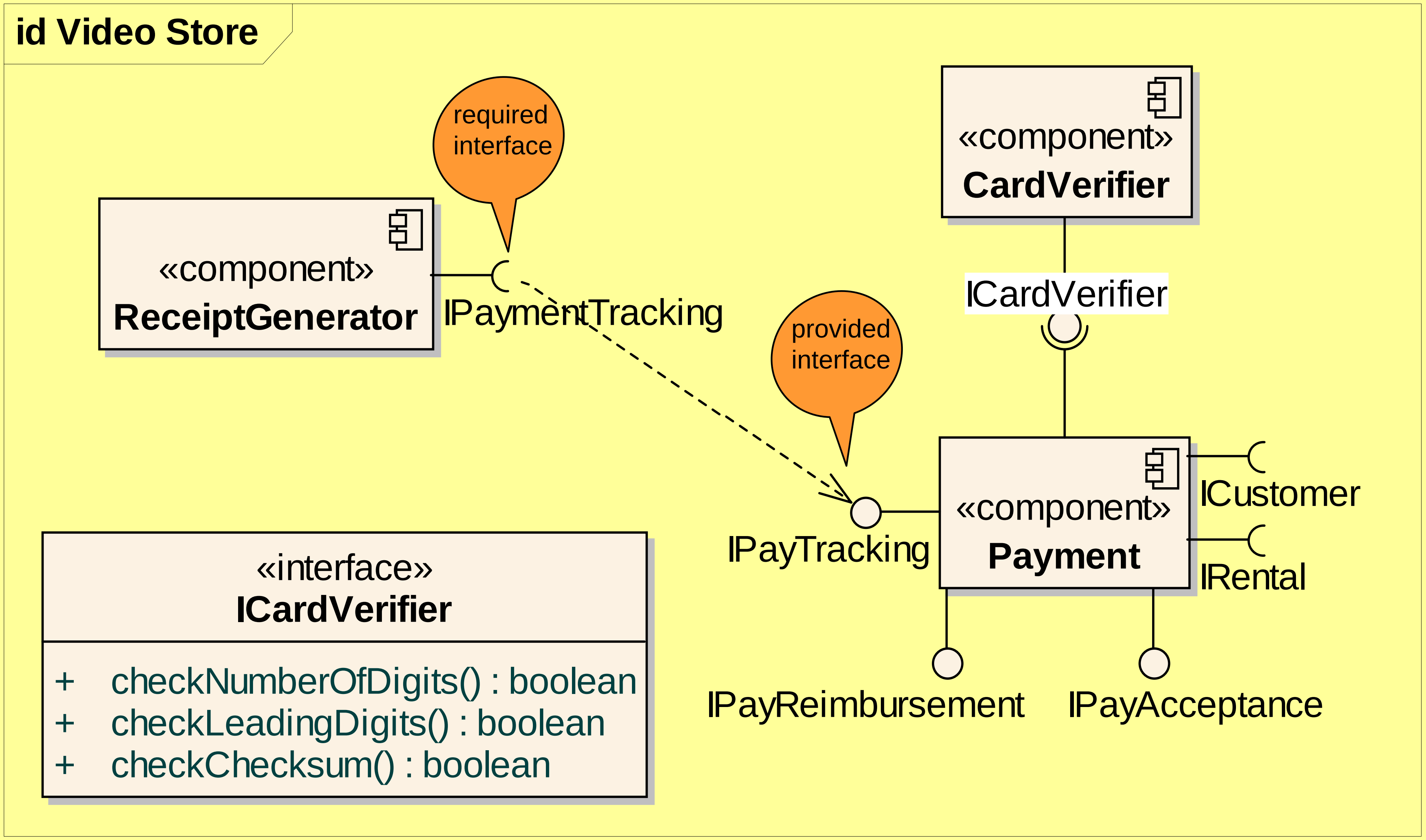
Subsystems and packages

- “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)

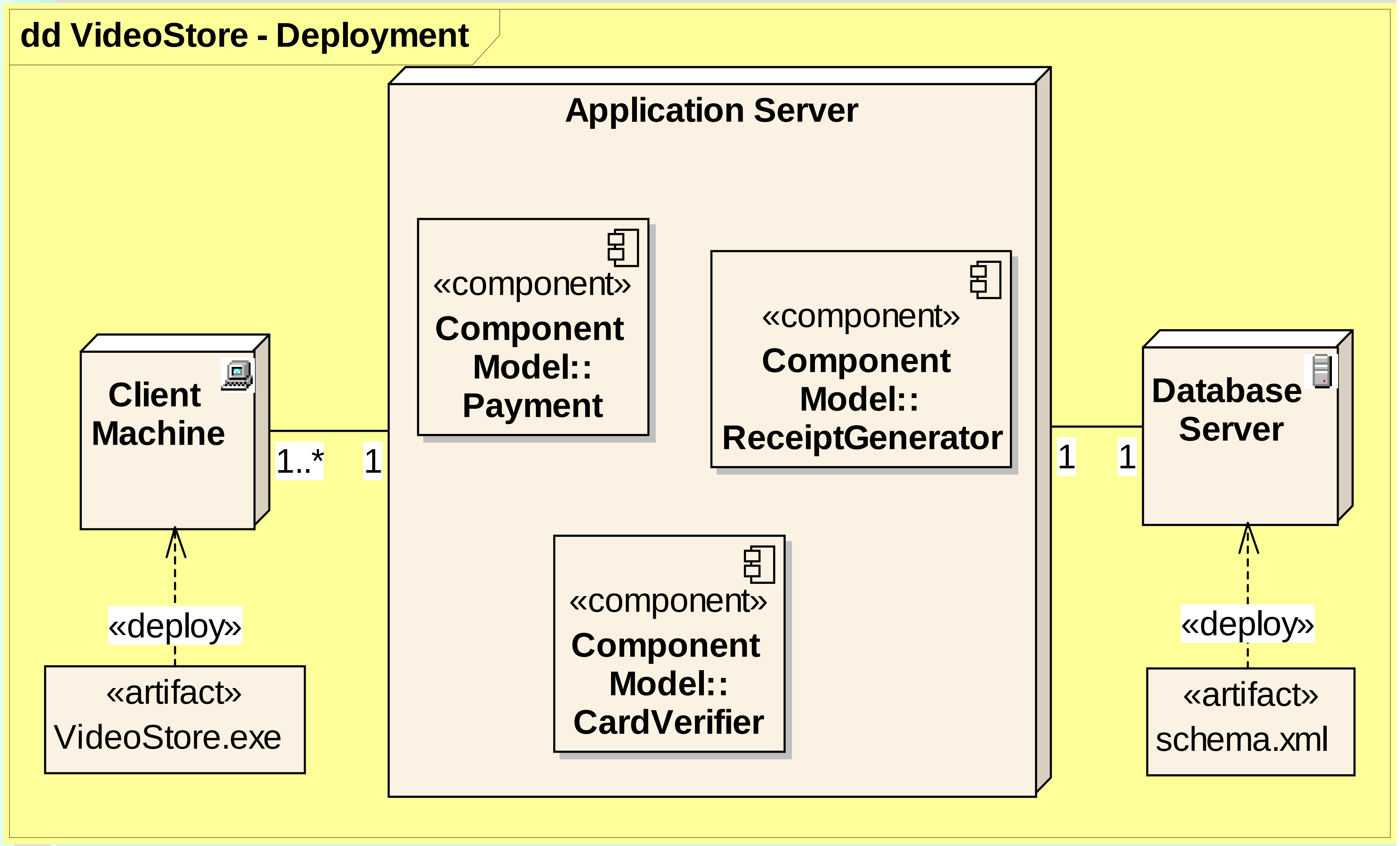
Subsystems and packages



Component diagram



Deployment diagram


dd VideoStore - Deployment
Application Server
«component» Component Model:: Payment
«component» Component Model:: ReceiptGenerator
«component» Component Model:: CardVerifier
Client Machine
1..*
1
1
1
Database Server
«deploy»
«artifact» VideoStore.exe
«deploy»
«artifact» schema.xml

Review Quiz 3.6

1. Is subsystem modeled as a stereotype of package or stereotype of component?
2. Give some examples of artifacts that can be deployed on a node.

Summary

- 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 the interactions between objects needed to execute a use case or part of it.
- A **state machine 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.