

Operations

- Syntax:
visibility name(parameter: type, ...): returnType {properties}
- Visibility as with attributes
 - not really important in conceptual models
- Properties
 - any string
 - predefined: “query”
 - constraints (preconditions, postconditions)
- Don't list obvious operations
 - unless you need a detailed design diagram
- Don't list private operations in conceptual models

497-5-23

Constraints

- Constraints that cannot be expressed in the diagram notation can be included as text
- Example:
customerNumber: int { >=0 }
- Always in braces
- All diagram elements can be annotated with constraints
- Can be:
 - natural language text
 - Object Constraint Language
 - predefined properties
 - Example: {query} for operations
 - any other text

497-5-24

Hints for Class Diagrams

- Remember: models are for communication
- Remember: include only important stuff
- How do I find classes, attributes and so on?
 - look through your use cases
 - classes often correspond to nouns
 - associations often correspond to verbs
- A class should
 - represent a coherent concept
 - Principle: Low Coupling, High Cohesion
 - have a small, well-defined set of responsibilities
 - have a number of collaborations
 - be named with a singular noun that says what each each instance of the class is
 - have no more than 10-20 operations

497-5-25

Hints for Class Diagrams

- Class diagrams should
 - have a single purpose
 - have a title that expresses the purpose
 - show only things that are relevant for this purpose
- Avoid
 - cyclical dependencies, if possible
 - generalization hierarchies with more than 5 levels
- Use colors to highlight and group things
 - unless you have to print it in black-and-white!
- Lay out classes in a meaningful way
 - similar classes close to each other
 - top: closer to the user, bottom: closer to the data structures
- Avoid crossing lines if possible

497-5-26

Analysis Example: Video Store

Check out a video

1. Clerk scans customer card
2. Clerk scans videos
3. Monitor displays price and rental period

Check in a video

1. Clerk scans video
2. System calculates late fees

Register new customer

1. Clerk enters customer data
2. Clerk swipes credit card
3. System prints customer card

497-5-27

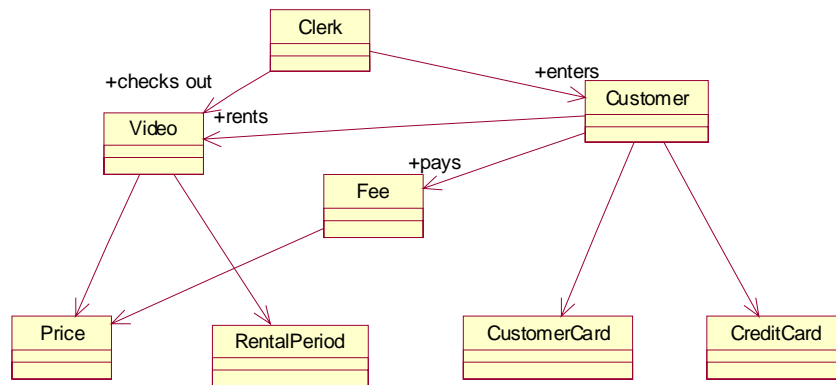
What conceptual classes do we have?

- Video
- Customer
- CustomerCard
- Clerk
- Price
- RentalPeriod
- Fee
- CreditCard

- Others?

497-5-28

First Draft of Class Diagram



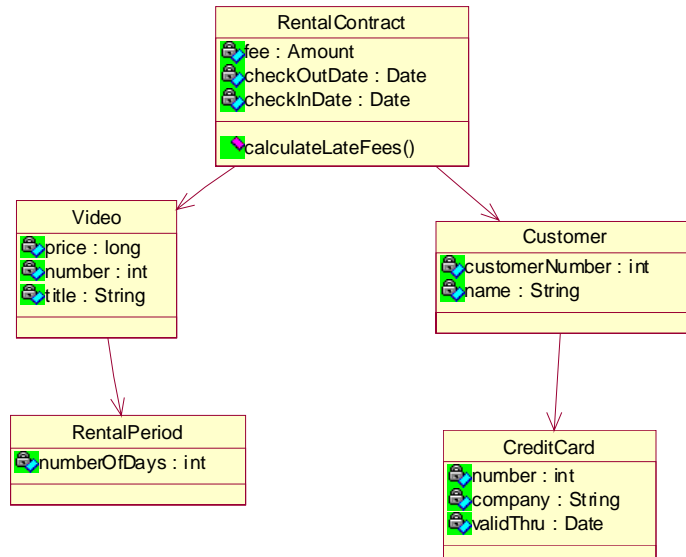
497-5-29

Thoughts about Our Class Diagram

- Do we need CustomerCard?
 - Is Customer enough?
- Do we need Clerk?
 - Only if the clerk logs in or something — but not specified in use case!
- What is a Price? Just a dollar amount? How about Fee?
- Where to represent the actual transaction?
 - When which customer rents which video for what price

497-5-30

Revision of Analysis Class Diagram



497-5-31

An Additional Use Case

Exchanging the Pricing System

1. Manager enters new pricing system

What is a pricing system?

- It says what category of videos can be rented how long for what price
- For example:
 - new movies: 3 days, \$2.50
 - kids' movies: 5 days, \$2.00

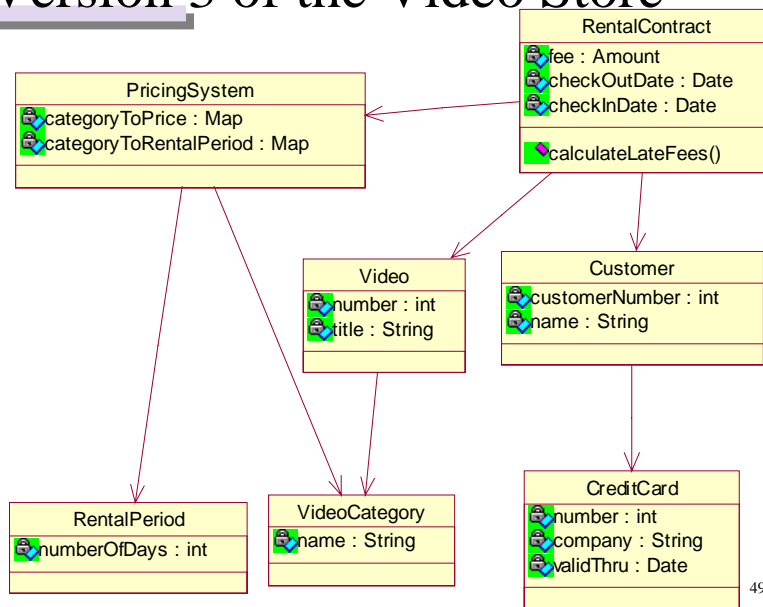
497-5-32

What does this change?

- We need something to represent a pricing system
- We need categories of videos
- Price can't be a property of the Video any longer
- RentalContract needs to be aware of the pricing system that was valid at time of rental

497-5-33

Version 3 of the Video Store



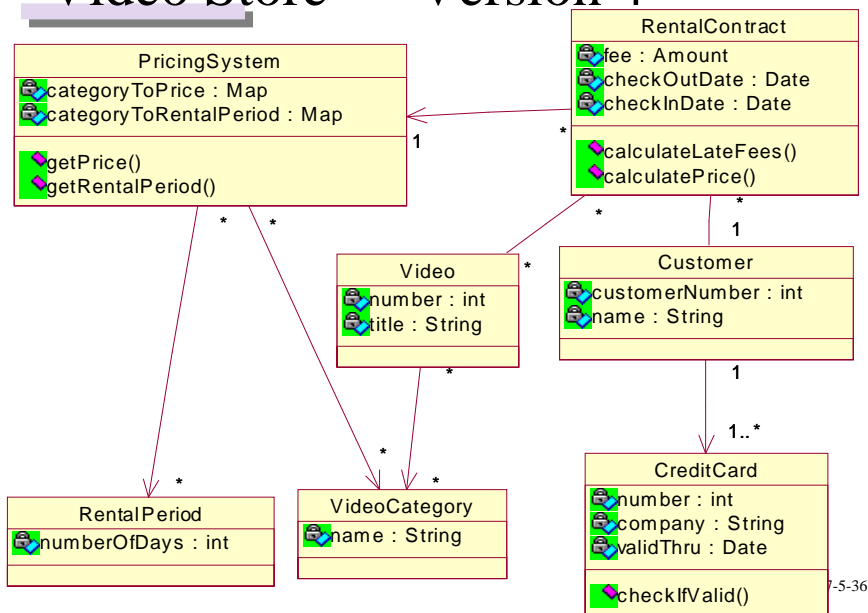
497-5-34

Video Store: Finishing Up

- Let's add in multiplicities:
 - How many PricingSystems per RentalContract?
 - How many Videos per RentalContract?
 - How many VideoCategories per Video?
 - How many CreditCards per Customer?
- Some role names to make it clearer?
 - Can't think of any
- Any operations that make it clearer?
 - calculate price of a RentalContract
 - check validity of credit card
 - get rental period of a video

497-5-35

Video Store — Version 4



7-5-36