

## 5. Analysis

1. Introduction
2. Requirements Capture
3. Process Models
4. Project Management
5. Analysis
  - 5.1 Object and Class Diagram Basics
  - 5.2 Analysis Patterns
  - 5.3 Interaction Diagrams
  - 5.4 State Machines and State Diagrams
  - 5.5 Activity and Dataflow Diagrams
  - 5.6 Object Constraint Language
  - 5.7 Assertions
6. Design
7. Testing and Implementation

© 2004. 497-5-1

## Analysis

- Analysis is the technical part of identifying requirements
- Turns informal requirements into something more formal
- Prepares design
- Models the real world
  - the way it is now
  - the way it should be after this program is done
- Usual parts of an analysis model:
  - domain model (or: business model)
  - workflow model
  - domain objects
  - relationships between domain objects
  - typical interactions between domain objects

© 2004. 497-5-2

## Notations for Analysis

- Object Diagrams
  - show objects and their links
- Class Diagrams
  - abstract domain object to domain classes
- Activity (or: Control-Flow) Diagrams
  - show workflows
- Data-Flow Diagrams
  - show how data get exchanged
- Interaction Diagrams
  - show typical interactions between domain objects
- State Diagrams
  - show the different states of a domain object
- Object Constraint Language
  - for expressing constraints that are too complex for diagrams

© 2004. 497-5-3

## What is UML?

- Unified Modeling Language
- Standard object-oriented notation for requirements and design
- formal syntax, informal semantics
- meta-model
- goal: communication
- several diagram types
- used widely in industry
- standardized by the Object Management Group (OMG)
- replaces several older notations: OMT, Booch, OOSE...
- three Amigos: Booch, Rumbaugh, Jacobson
- tools: Rational Rose, Argo/UML

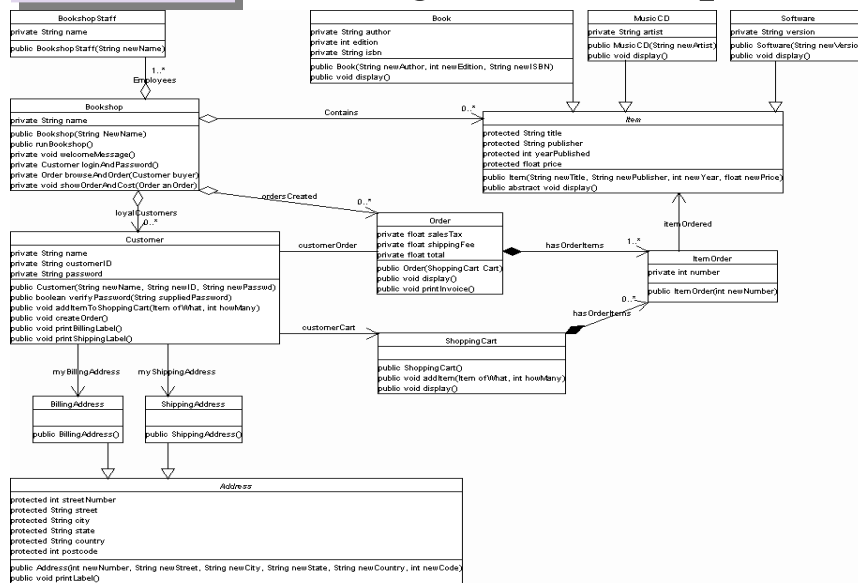
© 2004. 497-5-4

# UML Diagrams

Requirements Capture	Use-Case Diagram
Analysis	Class Diagram Sequence Diagram Collaboration Diagram State Diagram Package Diagram Activity Diagram
Design	Class Diagram Sequence Diagram Collaboration Diagram State Diagram Package Diagram Deployment Diagram

© 2004. 497-5-5

# UML Class Diagram (Example)



## 5.1 Object and Class Diagram

### Basics

- Learn what object and class diagrams are
- Relations
- Two simple examples: library and video store

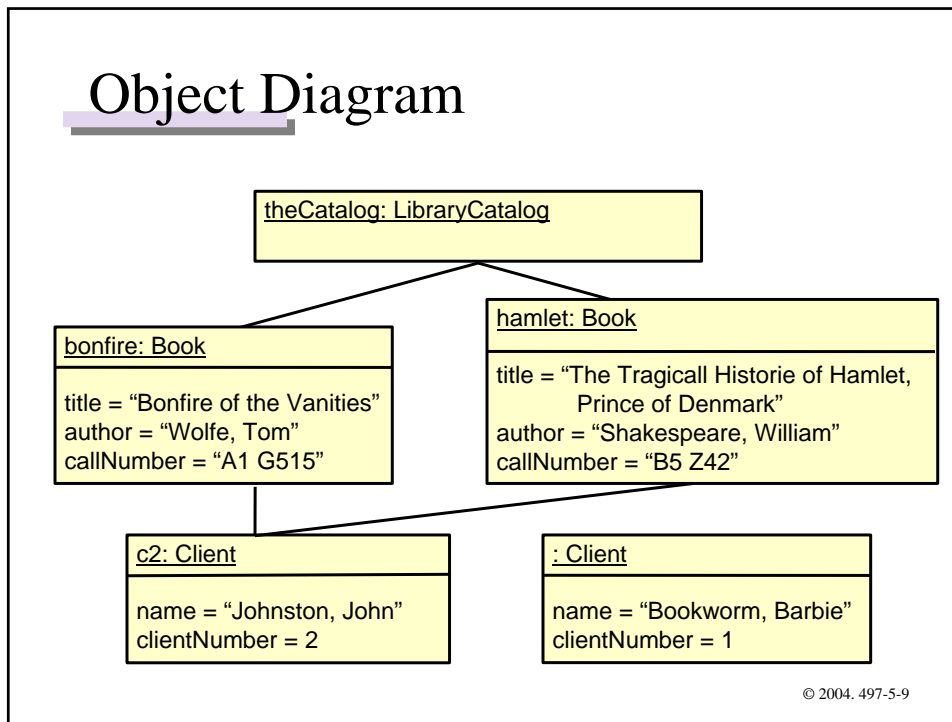
© 2004. 497-5-7

### Analysis Example: Library

- Library
  - has books
  - a book has an author and a title
  - has users
  - has a catalog
  - users are stored in a file
  - book has call number
- Also...
  - videos
  - audio CDs
  - users check in books, check out books
  - user can put hold on books

© 2004. 497-5-8

# Object Diagram



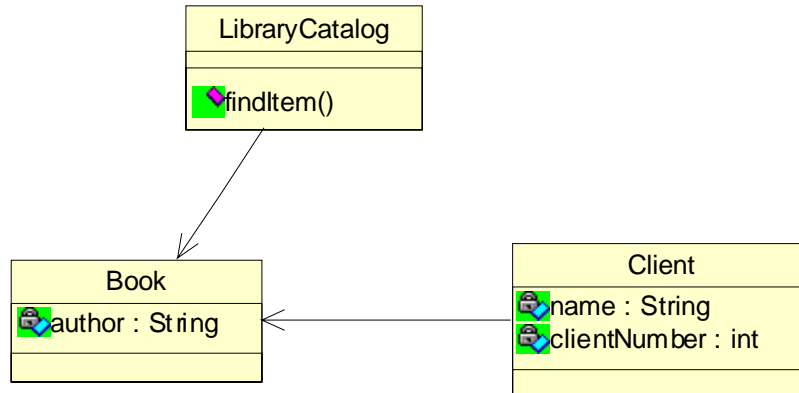
# Elements of UML Object

## Diagrams

- Objects
  - name: optional  
anObject
  - class name: don't forget the colon before it  
: Object
  - attributes (in Java: fields)  
name = value
- Links
  - may be named
  - in Java: references
- Object diagram shows a set of objects at a given time
  - completeness impossible
- Later: interaction diagrams
  - more complex object diagrams

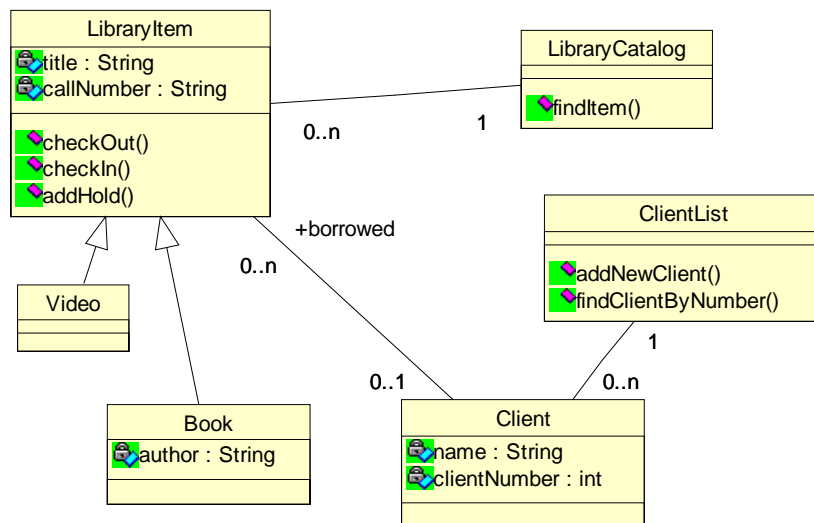
© 2004. 497-5-10

# Class Diagram



© 2004. 497-5-11

# Library Class Diagram — Refined



© 2004. 497-5-12

## Elements of UML Class

### Diagrams

- Classes
  - name
  - attributes (in Java: fields)
    - visibility, name, type
  - operations (in Java: methods)
    - visibility, name, return and parameter types
- Associations
  - in Java: references
  - in C++: pointers or references
  - at each end:
    - role name
    - multiplicity (0..1, 1, \*, 1..\*, and others)
- Generalizations
  - in Java and C++: inheritances

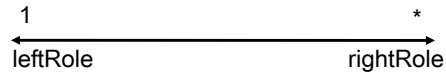
© 2004. 497-5-13

## Purpose of Class Diagrams

- Most important UML diagram
- Class diagrams can completely specify a system
- Used both in analysis and design
- Two different purposes:
  - conceptual model
    - used in analysis
    - shows little detail
    - real world
  - implementation model
    - used in design
    - shows detail
    - adapted to programming language, platform
- Can easily be mapped to code (and back)

© 2004. 497-5-14

# Associations



- Relations between classes
- Roles
  - analogous to names of instance variables
- Multiplicities
  - 0, 1, \*, 0..1, 1..\*, 5..6, and so on
  - says how many objects each object knows
  - would be realized through arrays, Sets, Lists, and so on
- Navigability
  - bidirectional: each class references the other
  - unidirectional: A knows B, but B doesn't know A
  - no arrow heads: means either "bidirectional" or "not specified"

© 2004. 497-5-15

# Naming Conventions for Associations

- If you have an association with a role called "client":
  - there may be an instance variable "client"
  - there may be operations "getClient()", "setClient()"
  - encapsulation principle
- If there is only one association to class "Something"
  - role name redundant
  - role often assumed to be called "something"

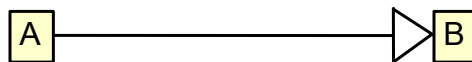
© 2004. 497-5-16

## Associations: Conceptually

- Responsibility
  - A knows B
  - A talks to B
  - A is responsible for updating B
  - A can get data from B
  - A is owner of B
  - A can do things with B
- A link between objects of the two classes

© 2004. 497-5-17

## Generalizations



- Another relation among classes: “is a kind of”
- B generalizes A
- A inherits from B
- Substitutability
  - A can be substituted for B
  - A does everything that B does, and possibly more
- Polymorphism possible
  - A has the same specification as B
  - A implements unspecified features differently
  - Different replies to the same request possible
- Multiple inheritance possible

© 2004. 497-5-18

# Background on Relations

## (Review)

- A relation between A and B is a subset of  $A \times B$
- Example: Divides relation on  $\{2..6\}$   
divides =  $\{ (2,2), (2,4), (2,6), (3,3), (3,6), (4,4), (5,5), (6,6) \}$
- Well-known relations from mathematics
  - divides
  - =
  - <, >
  - functions are special relations
    - A function  $A \rightarrow B$  is a relation between A and B so that each element of A occurs in exactly one pair

© 2004. 497-5-19

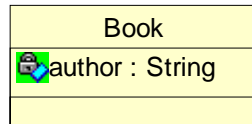
# Background on Relations

- Properties of relations
  - reflexive:  $a=a$
  - symmetric:  $a=b \Rightarrow b=a$
  - transitive:  $a>b \text{ and } b>c \Rightarrow a>c$
- UML relations
  - associations
    - may be symmetric: if bidirectional
  - generalizations
    - are transitive:  
X inherits from Y  
Y inherits from Z  
*implies*: X inherits from Z
    - are never symmetric:  
X inherits from Y *implies* Y does not inherit from X

© 2004. 497-5-20

## Associations and Attributes

- ...are exchangeable!



© 2004. 497-5-21

## Attributes

- Are used instead of associations:
  - for frequently used types (int, String, ...)
  - to denote value semantics (copied not referenced)
    - in Java: primitive types
    - in C++: members that are not pointers or references
  - to save space
- Syntax:
  - visibility name: type
- Everything's optional
- Visibility
  - + public - private ~ package # protected
- List attributes only if they help in understanding class
  - should become private in implementation model

© 2004. 497-5-22

## 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

© 2004. 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

© 2004. 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

© 2004. 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

© 2004. 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

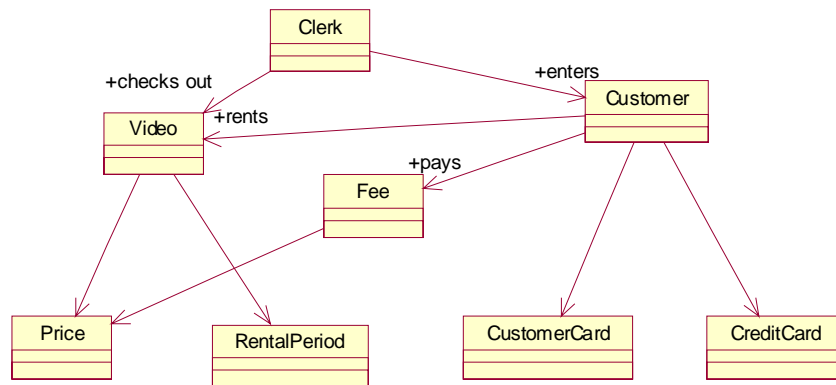
© 2004. 497-5-27

## What conceptual classes do we have?

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

© 2004. 497-5-28

## First Draft of Class Diagram



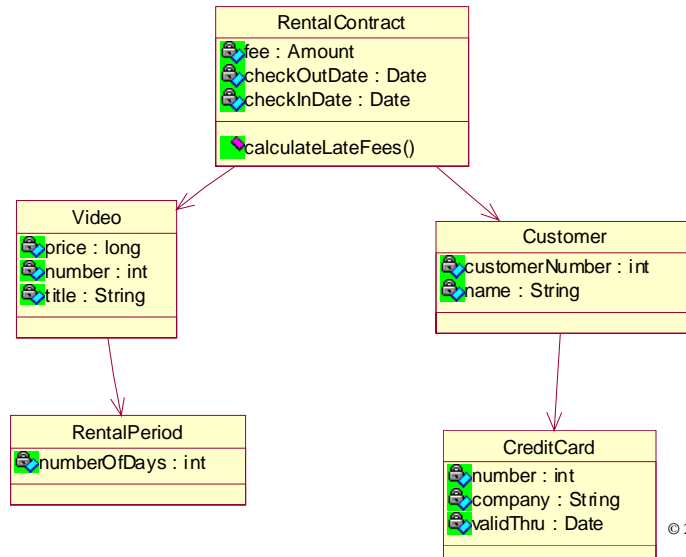
© 2004. 497-5-29

## 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

© 2004. 497-5-30

# Revision of Analysis Class Diagram



## 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

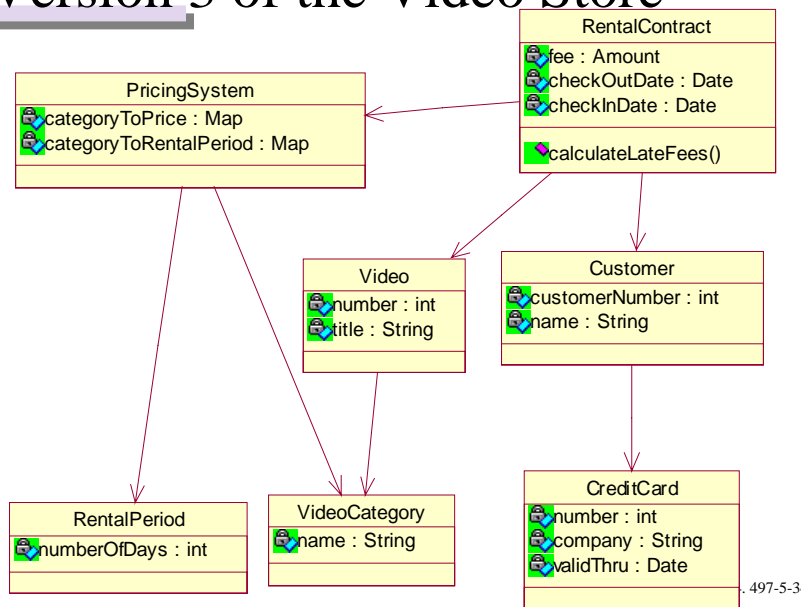
© 2004. 497-5-32

## 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

© 2004. 497-5-33

## Version 3 of the Video Store



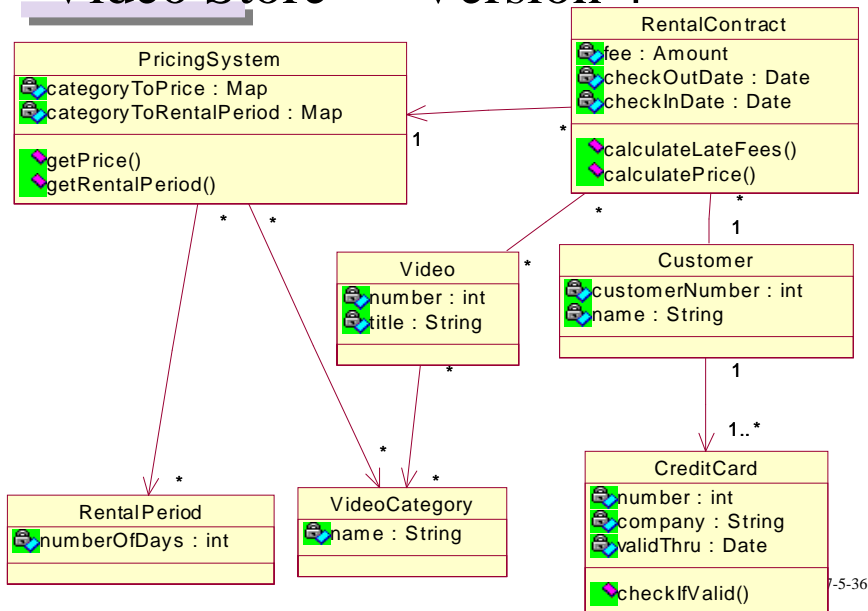
.497-5-34

## 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

© 2004. 497-5-35

## Video Store — Version 4



7-5-36