

7. Design

7. Design

- 7.1 Design Principles
- 7.2 Object Concepts
- 7.3 UML Odds and Ends
- 7.4 Mapping UML to Code
- 7.5 Code Reuse
- 7.6 Design Patterns
- 7.7 Class Design
- 7.8 Refactoring
- 7.9 User Interfaces
- 7.10 Persistence

What is Design?

- Take the analysis model and make it more detailed
- Take the analysis model and add technical considerations:
 - programming language
 - operating system
 - class library
 - database
 - architecture
 - distribution
 - interoperability
 - other nonfunctional requirements
- Prepare implementation
 - the implementer should not have to worry about design issues
- The “How” of the program

What is Design?

- Design versus Analysis
 - analysis models the real world
 - analysis deals with domain concerns
 - analysis is imprecise
- Design versus Implementation
 - design decides on the structure of the code
 - decisions that concern more than one method or class
 - implementation focuses on one method at a time
 - implementation decides on algorithms, private variables, details

Design Activities

- Define subsystems
 - a subsystem is a set of related classes
- Refine classes
 - define operations
 - adapt to technical constraints
- Identify reusable code
 - class libraries:
 - Standard Template Library (STL)
 - Java standard library (Java Development Kit – JDK)
 - open source projects
 - commercial software
- Define additional classes
 - to solve technical issues
 - GUI
 - Database connectivity
 - Network connectivity

Design Models

- Specification View
 - focuses on classes that are used by many other classes
 - uses interfaces a lot
 - does not show attributes
 - does not show private operations
 - serves to understand the purpose of classes
- Implementation View
 - shows each individual class
 - shows private elements, if needed
 - serves to prepare implementation of the classes shown

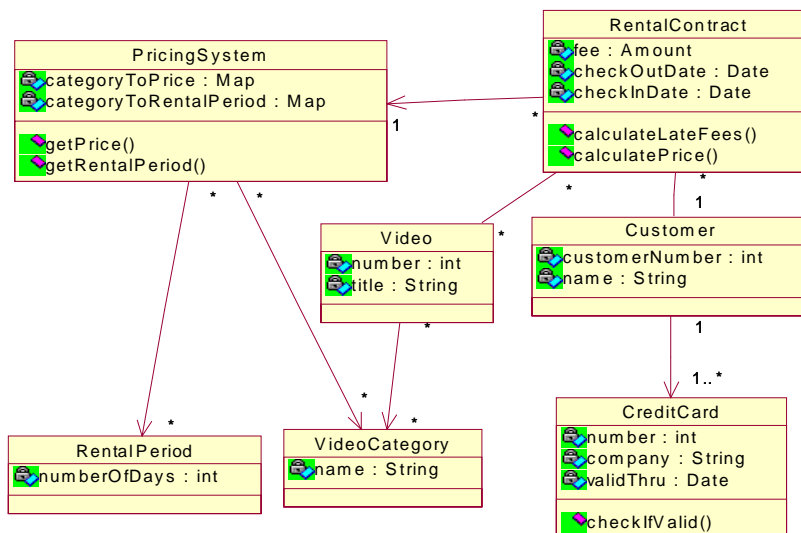
Things Used in Design

- Notations
 - Class diagrams
 - Interaction diagrams
 - State diagrams
 - Deployment diagrams
- Tools
 - UML diagram tools
 - Reverse engineering and code generation tools
- Methods
 - Design Patterns
 - Code Reuse
 - Refactoring

Things that can Go Wrong in Design

- Insufficient separation between analysis and design
 - easily happens because same notations are used
 - problem: one of them often done incompletely
- Insufficient understanding of reused code
 - didn't learn enough about the class library
 - will make implementation very hard
- Not enough support classes
 - to communicate with Gui, database
- Badly documented design
 - if the implementer doesn't understand it, he/she won't implement it correctly
- Architecture doesn't work
 - the large-scale design doesn't work (too slow, too difficult...)
 - hard to make sure it works without implementing it

Example: Video Store (from Analysis)



Design Example: Video Store

- What do we need to change about our analysis model to turn it into a design model?
 - decide on a programming language — Java?
 - add a Gui
 - which library? Java Swing?
 - define Gui classes
 - for windows, dialogs
 - figure out a way to communicate with Gui
 - MVC?
 - add a way to store data
 - our own file format? Java serialization?
 - requires serialization methods
 - an SQL database?
 - requires SQL transformation
 - an object database?

Design Example: Video Store

- What do we need to change? (continued)
 - interoperation with other programs?
 - standard data exchange format: XML?
 - data exchange through Internet
 - central server of the store chain
 - design architecture
 - three-tier?
 - reusable code?
 - don't know of any
 - restrictions because of non-functional requirements?
 - performance
 - usability
 - security
 - add in all kinds of details
 - all the missing operations

Three-Tier Architecture

- A very common architecture for application software
- Simple
- Top tier: user interface
 - windows, dialogs, keyboard shortcuts
 - determines overall control flow
- Middle tier: domain logic
 - all the classes from analysis
 - here the actual task of the program is solved
- Bottom tier: data storage
 - database and database connectivity
- Constraint:
 - each tier knows only about the tier directly below

Top-Down versus Bottom-Up

- Two design strategies
- Top-Down
 - start with the requirements
 - divide them into subsystems
 - repeat for each subsystems
 - stop when you are detailed enough to implement
- Bottom-Up
 - start with likely methods or classes
 - compose those into likely subsystems
 - repeat with the subsystems
 - stop when you reach your requirements
- Which one to use?
 - Use both!
 - With bottom-up alone, it's hard to fulfill the requirements
 - With top-down alone, it's hard to create a good design