

6.3 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

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

6.3 UML Odds and Ends

- Just a few things we didn't have time to cover before
- Additional elements of class diagrams
- Extension mechanisms
- Packages
- Notes
- Deployment diagrams

Aggregation



- Aggregation is a kind of association
- "is part of" relation
 - a book is part of a library catalog
 - a library catalog has any number of books
- No really clear meaning
 - may make a diagram more readable
 - maps to the same code as any other association

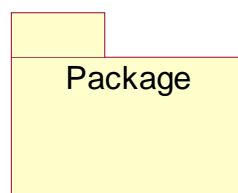
Composition



- Composition is a kind of aggregation
- But with a well-defined meaning!
- Example

```
class Rectangle {
    private Point[] p = new Point[ 4 ];
    //the elements of p are never given out
    Rectangle( int x, int y, int height, int width ) {
        p[ 0 ] = new Point( x, y );
        ... }
}
```
- Meaning: each part belongs to exactly one composite
 - When the composite is destroyed, the part is destroyed
 - Parts cannot be shared among composites

Packages



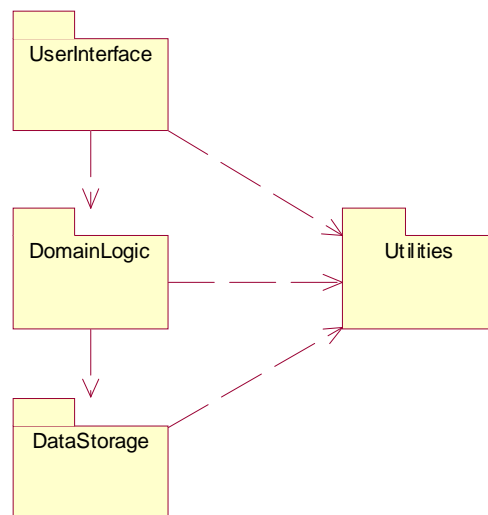
- A grouping mechanism
- A package can contain any kind of UML elements
- Two uses:
 - to hide part of a diagram
 - to show the relationship between large numbers of elements

Dependency

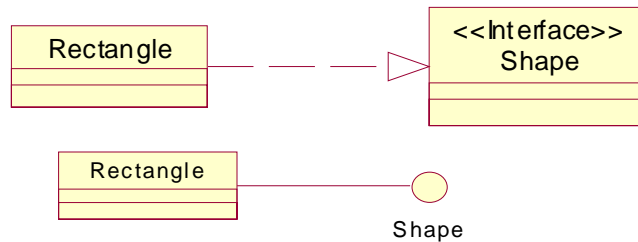


- A depends on B
- Any kind of “uses” relation between classes that is not an association or a generalization
- Examples:
 - A instantiates B (but does not keep the reference)
 - A has operations that have objects of type B as parameters
- Generally:
 - Class A cannot be used without class B
- Can also be used with packages

Example: Three-Tier Architecture

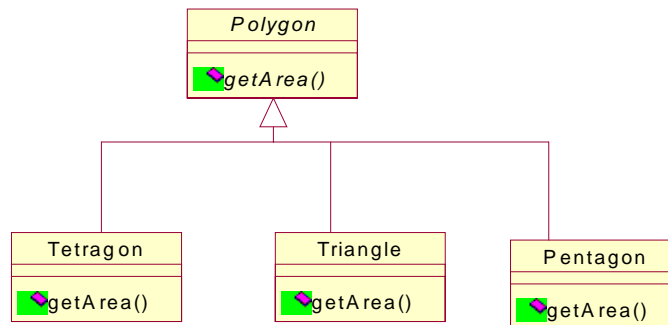


Interfaces



- Class Rectangle implements interface Shape
- Top: full representation
- Bottom: elided representation
 - no operations of the interface can be shown
- The relation between class and interface is a realization
- Interfaces are a kind of class in UML

Abstract Classes

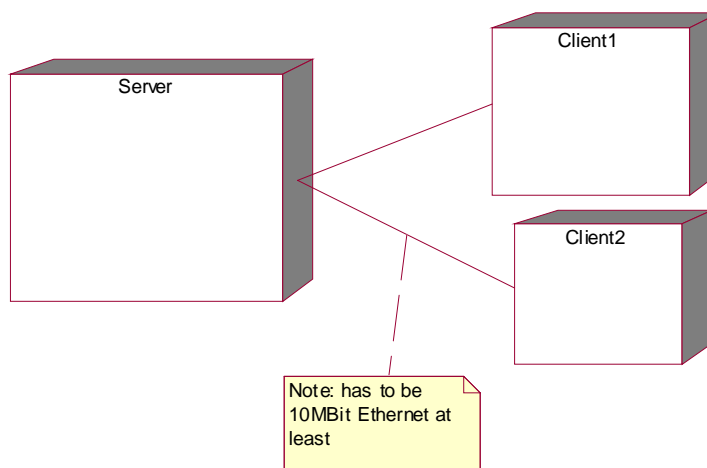


- Abstract classes and operations
 - have names in italics
- By the way...
 - this is an other way to show generalizations
 - several arrows made into one

UML Extension Mechanisms

- Stereotypes
 - allow user-defined UML elements
 - for example:
 - «interface»
 - «user interface class»
 - those thingies are called guillemets
 - can also be represented through their own icons
 - such as a circle for interfaces
- Profiles
 - set of stereotypes for a purpose
 - for example: USDP profile
- Constraints
 - we already discussed those
 - {abstract}

Deployment Diagram



Deployment Diagrams and Notes

- Deployment diagrams
 - show structure of distributed systems
- Elements of deployment diagrams
 - hosts
 - network connections
 - inside of the hosts, files can be shown
 - files installed on the host
 - executables running on the host
- Notes
 - rectangle with a folded-over corner
 - can be used in all UML diagrams
 - can be attached to all diagram elements
 - must be attached to something with a broken line