Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 10: Analysis Packages

Analysis Workflow: *Packages*

- The *analysis workflow* consists of the following activities:
  - **Architectural analysis**
  - Analyze a use case
  - Analyze a class
  - **Analyze a package**

Packages

- The *package* is the UML mechanism for grouping things.
- Packages serve many purposes:
  - they group semantically related elements;
  - they provide units of configuration management and parallel work;
  - they provide an encapsulated namespace in which all names must be unique - to access an element within the namespace you must specify both the element name and the namespace name.
- Every model element is owned by one package:
  - the packages form a hierarchy;
  - the root package may be stereotyped «topLevel»;
  - by default, model elements are placed in the «topLevel» package.

Package Visibility and Stereotypes

- Package elements may have visibility:
  - visibility is used to control the coupling between packages;
  - there are two levels of visibility:
    - public (+) - elements are visible to other packages;
    - private (-) - elements are completely hidden.
- Package stereotypes:
  - «Framework»- a package that contains model elements that specify a reusable architecture;
  - «modelLibrary»- a package that contains elements that are intended to be reused by other packages.
- A package defines an encapsulated namespace:
  - use qualified names to refer to elements in other packages, for example:

    Library::Users::Librarian

Packages: Example

Membership

Membership

+ClubMembership
+Benefits
+MembershipRules
+MemberDetails
+MemberDetails::Member
-JoiningRules

public (exported) elements

private element

qualified name

Membership

+MemberDetails

«import»

+Member

-JoiningRules

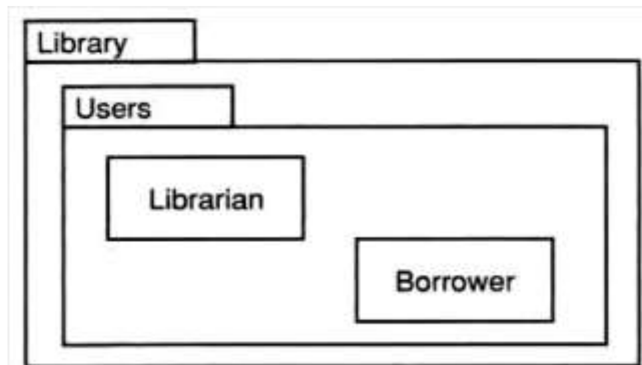
+ClubMembership

+Benefits

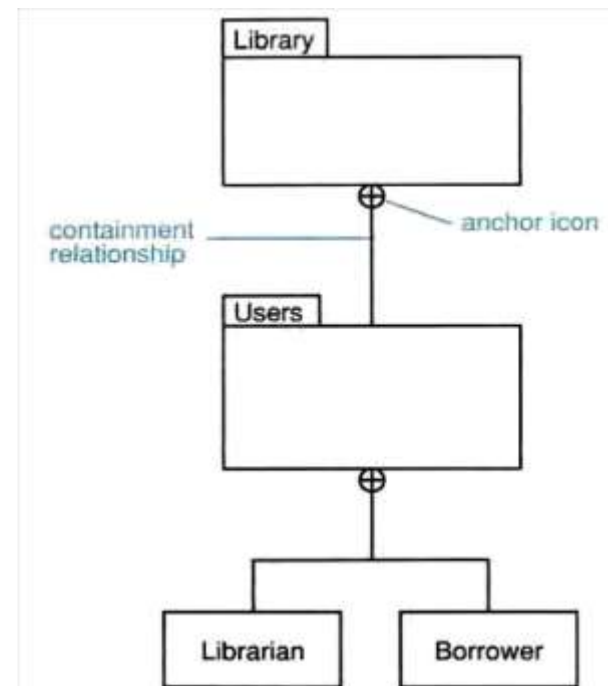
+MembershipRules

Nested Packages

- The inner package can see all of the public members of its outer packages;
- The outer package can't see any of the members of its inner packages unless it has an explicit dependency on them (usually «access» or «import»)
  - this allows you to hide implementation details in nested packages.



Library::Users::Librarian



Package Dependencies

- A dependency relationship between packages indicates that the client package depends in some way on the supplier package.
  - **«use»** (default) **-** an element in the client package uses a public element in the supplier package.
  - **«import» -** public elements of the supplier namespace are added as *public* elements to the client namespace. Elements in the client can access all public elements in the supplier by using unqualified names.
  - **«access» -** public elements of the supplier namespace are added as *private* elements to the client namespace. Elements in the client can access all public elements in the supplier by using unqualified names.
  - **«trace» -** the client is a historical development of the supplier. This usually applies to models rather than elements.
  - **«merge» -** public elements of the supplier package are merged with elements of the client package. Only used in metamodeling.

Package Dependencies

Membership

«import»

package dependency

MemberDetails

public visibility

+Member

-JoiningRules

+ClubMembership

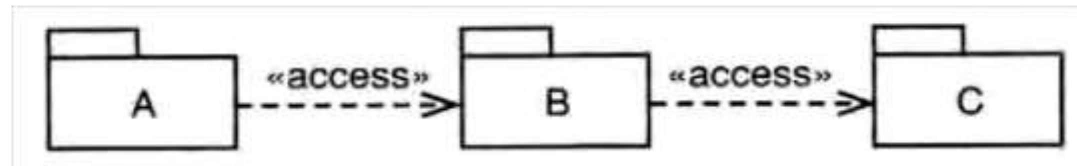
+Benefits

private visibility

+MembershipRules

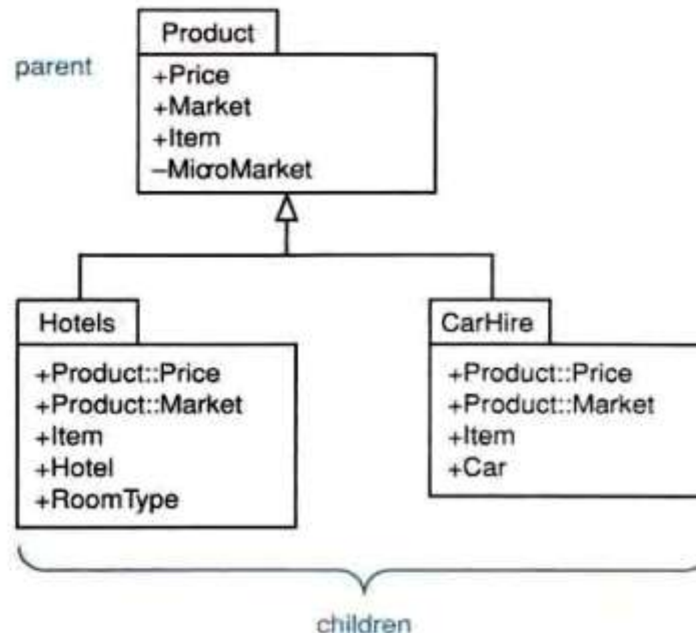
Relationship Transitivity

- Transitivity: If A has a relationship to B and B has a relationship to C, then A has a relationship to C.
  - «import» is transitive.
  - «access» is not transitive.



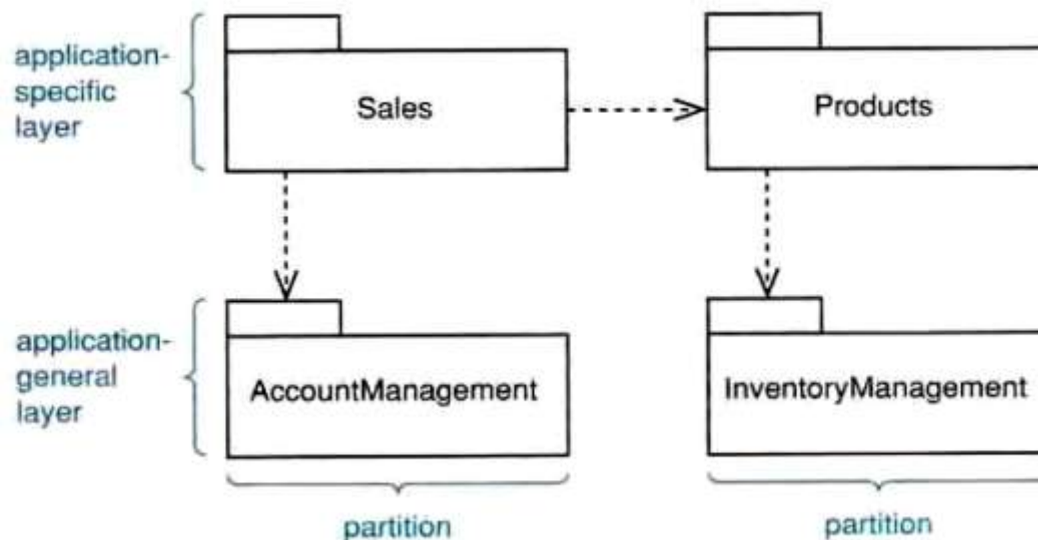
Package Generalization

- Very similar to class generalization;
- The child packages:
  - inherit elements from their parent package;
  - can add new elements;
  - can override parent elements.



Architectural Analysis and Package Analysis

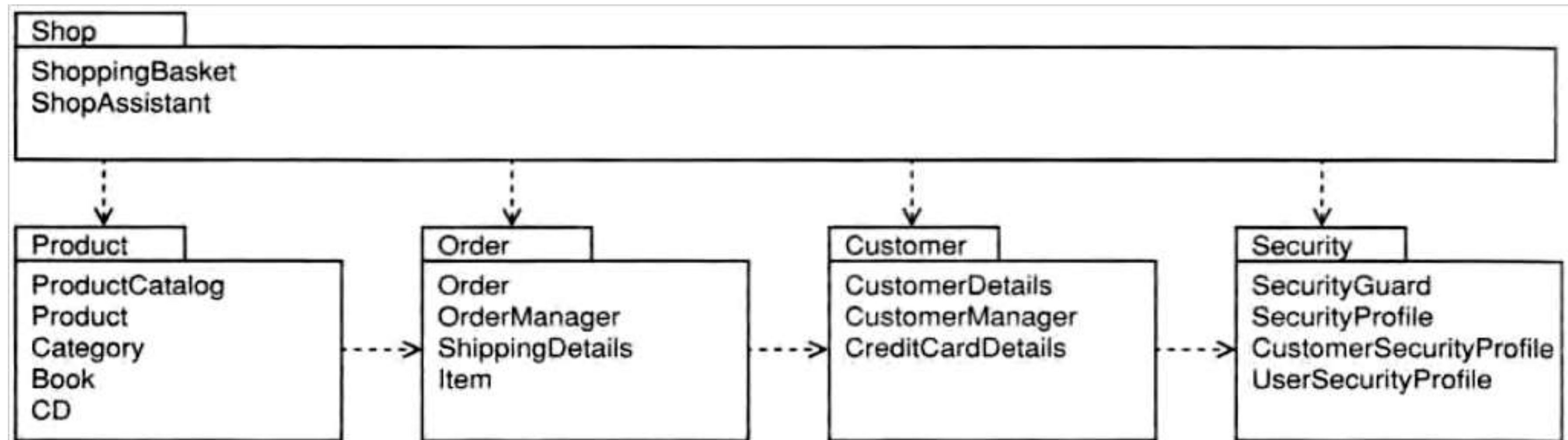
- identify subsystem architecture;
- partition cohesive sets of analysis classes into analysis packages;
- layer analysis packages according to their semantics;
- attempt to minimize coupling by:
  - minimizing package dependencies;
  - minimizing the number of public elements in all packages;
  - maximizing the number of private elements in all packages.



Finding Analysis Packages

- Examine subsystems.
- Examine analysis classes - look for:
  - cohesive clusters of closely related classes;
  - inheritance hierarchies;
  - classes are most closely related by (in order) inheritance, composition, aggregation, dependency.
- Examine use cases:
  - clusters of use cases that support a particular business process or actor *may* have analysis classes that should be packaged together;
  - related use cases *may* have analysis classes that should be packaged together;
  - be careful - analysis packages often cut across use cases!
- Refine the package model to maximize cohesion within packages and minimize dependencies between packages by:
  - moving classes between packages;
  - adding packages;
  - removing packages;
  - remove cyclic dependencies by merging packages or by splitting them.

Finding Analysis Packages



Eliminating Cyclic Package Dependencies

A B

There can be more complex cycles involving three or more packages!

merge

split

A

C

A B

Reference

- Arlow, J., Neustadt, I., *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, 2nd Ed. Addison-Wesley, 2005.