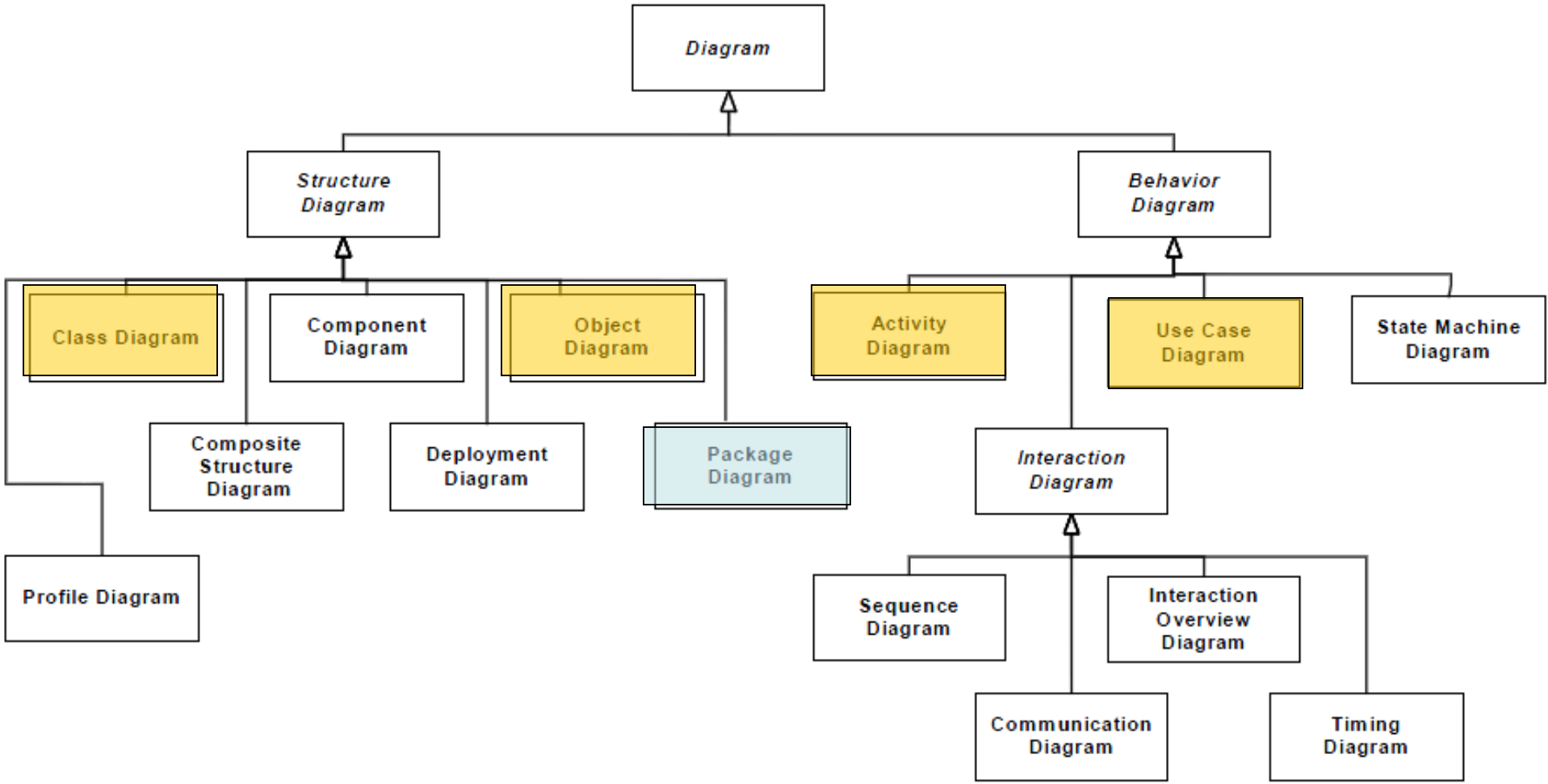
Object-Oriented Technology and UML

Package Diagram

Package Diagram



Topics

- Basic concepts
- Constituent elements
- Representation
- Reading method
- Modeling

Package

- basic building block
  - rule
  - common mechanism
  - relationship
  - thing
    - structural thing
    - behavioral thing
    - grouping thing
    - annotational thing
  - diagram

Package

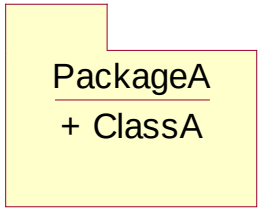
- In the UML, the package is a general-purpose mechanism for organizing modeling elements into groups

The role of the package

- Modeling Groups of Elements
  - Most of the time you'll use packages to group the same basic kind of elements
- Modeling Architectural Views
  - As you consider the different views of a software system's architecture, you need even larger chunks. You can use packages to model the views of an architecture

Graphical representation of package

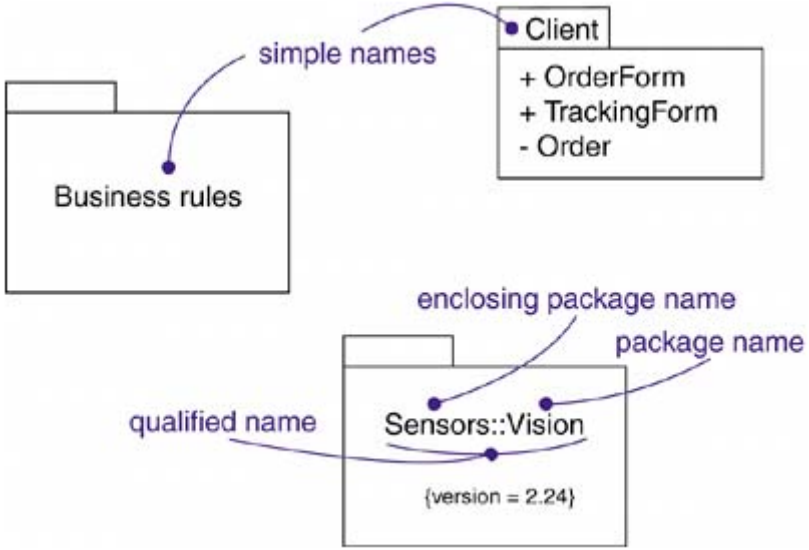
- Graphically, a package is rendered as a tabbed folder. The name of the package goes in the folder (if its contents are not shown) or in the tab (if the contents of the folder are shown)




```
//Source file: C:\\Users\\NI\\Desktop\\temp\\Code\\PackageA\\ClassA.java

package PackageA;

public class ClassA
{

   /**
    * @roseuid 4AC7657E0055
    */
   public ClassA()
   {
      }
}
```

Names of Package

- Every package must have a name that distinguishes it from other packages
- A name is a textual string
  - That name alone is known as a simple name
  - A qualified name is the package name prefixed by the name of the package in which that package lives



Owned Elements

- A package may own other elements, including
  - Classes
  - Interfaces
  - Components
  - Nodes
  - Collaborations
  - Use cases
  - Diagrams
  - and even other packages

Visibility

- You can control the visibility of the elements owned by a package just as you can control the visibility of the attributes and operations owned by a class
  - +: public
  - #: protected
  - -: private

Stereotype of package

- To specialize the use of a package in a development process, the UML stereotype extension mechanism can be used
  - <<system>>
  - <<subsystem>>
  - <<facade>>
  - <<stub>>
  - <<framework>>
  - More…

Relationships between Packages

- Dependency
  - << import>>
  - << access>>
  - << trace>>
- Generalization

<< import>>

- Specifies that the public contents of the target package enter the public namespace of the source, as if they had been declared in the source
- Code
  - using
  - import

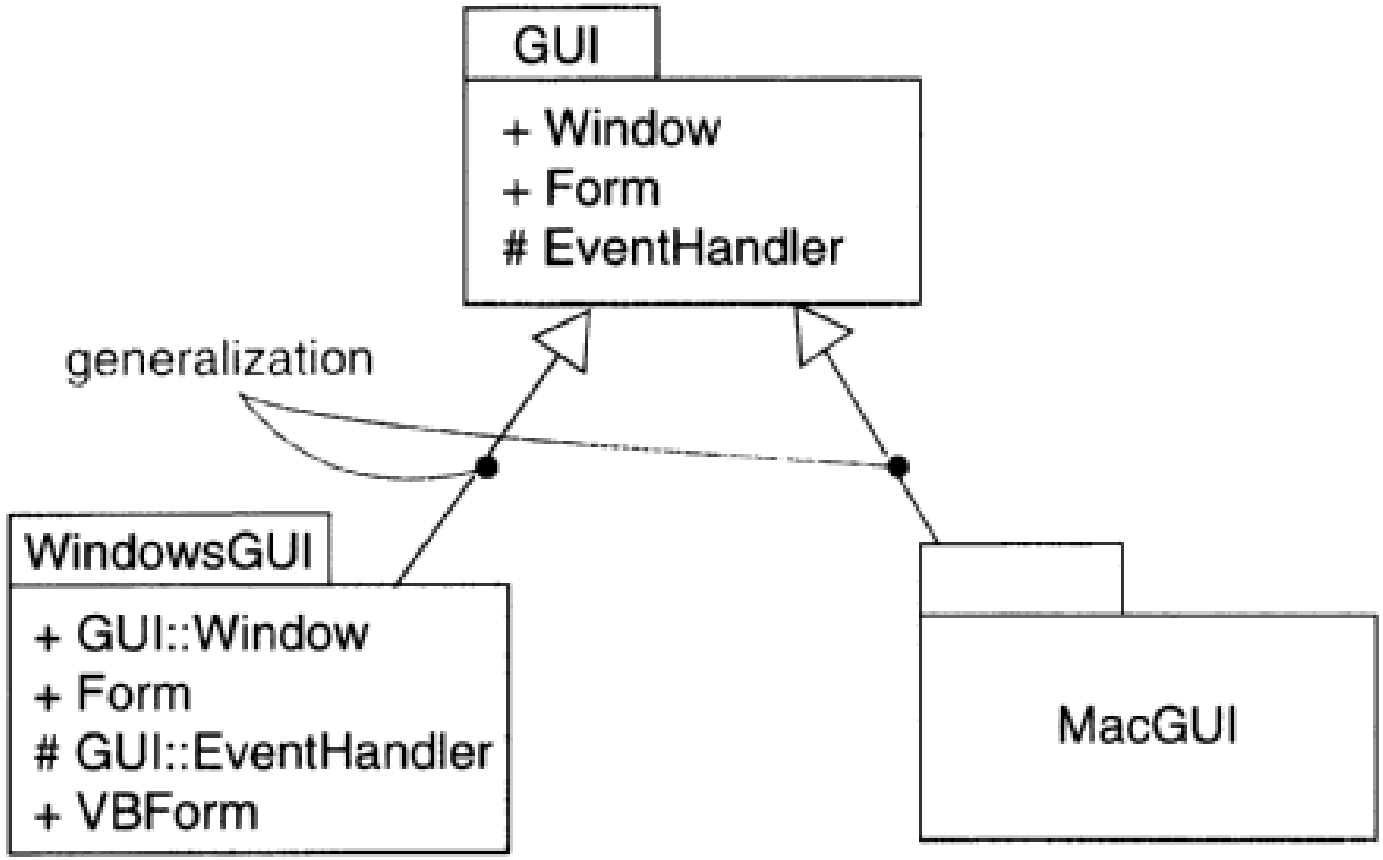
<< access>>

- Specifies that the public contents of the target package enter the private namespace of the source. The unqualified names may be used within the source, but they may not be re-exported

<< trace>>

- Specifies that the target is a historical predecessor of the source from an earlier stage of development
- You'll use trace when you want to model the relationships among elements in different models

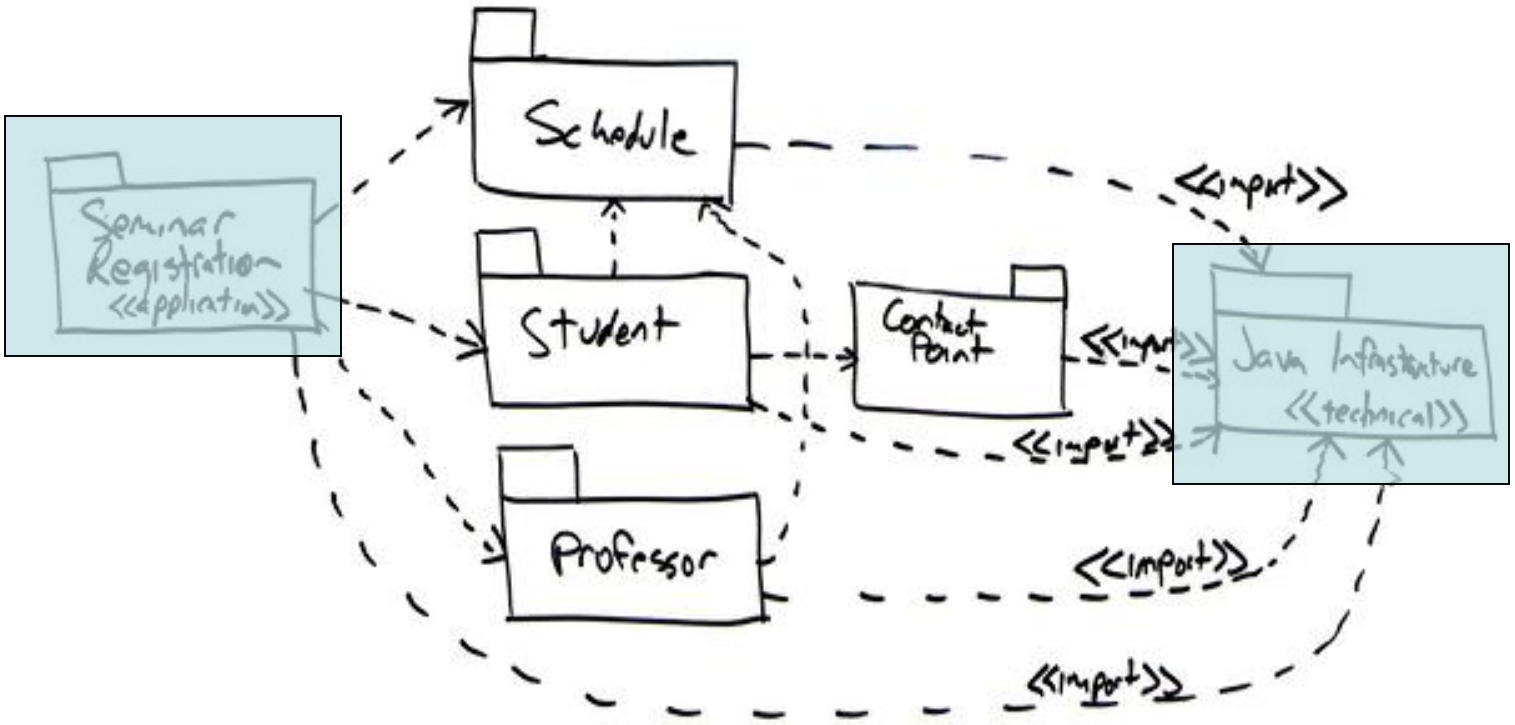
Generalization

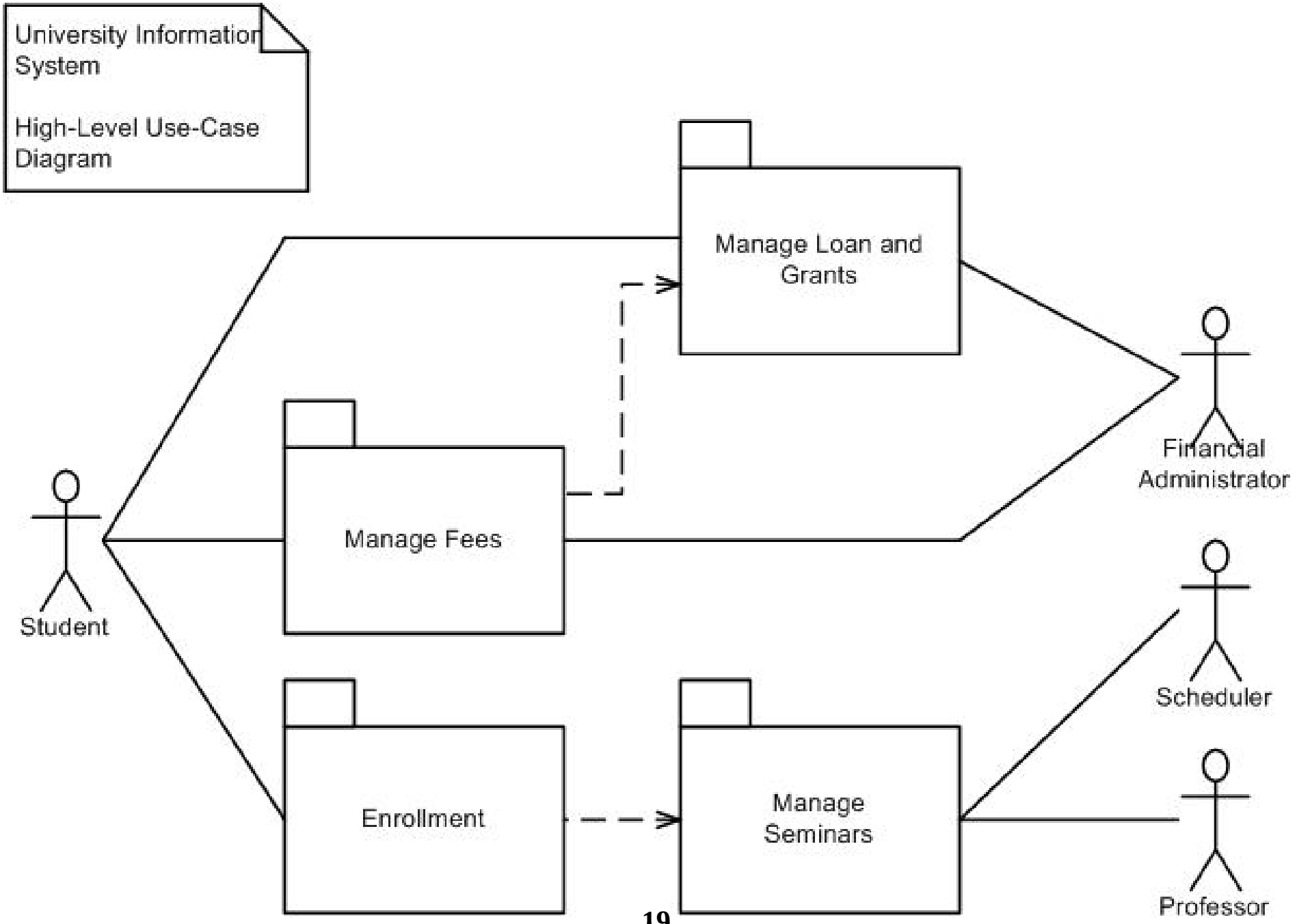


Package Diagram

- A package diagram shows the decomposition of the model itself into organization units and their dependencies
- When to use
  - People find package diagrams extremely useful on larger-scale systems to get a picture of the dependencies between major elements of a system. These diagrams correspond well to common programming structures
  - Plotting diagrams of packages and dependencies helps you keep an application's dependencies under control

Example of Package Diagram




University Information System
High-Level Use-Case Diagram
Manage Loan and Grants
Financial Administrator
Manage Fees
Student
Scheduler
Enrollment
Manage Seminars
Professor

How to read a Package Diagram

- Understand the semantics of each package and its containing elements
- Further understand the relationship between packages
- Find the package with most complex dependencies, and begin to read this package, and then read more simple packages successively

Steps of Package Diagram Modeling

- Identify the packages
- Determining the relationships between packages
- Indicate the relationships between packages

Principles of Package Design

- The Release Reuse Equivalency Principle (REP)
- The Common Reuse Principle (CRP)
- The Common Closure Principle (CCP)
- The Acyclic Dependencies Principle (ADP)

The Release Reuse Equivalency Principle (REP)

- The granule of reuse is the granule of release
- One criterion for grouping classes into packages is reuse

The Common Reuse Principle (CRP)

- Classes that aren’t reused together should not be grouped together
- A dependency upon a package is a dependency upon everything within the package

The Common Closure Principle (CCP)

- Classes that change together, belong together
- We would like to minimize the number of packages that are changed in any given release cycle of the product

The Acyclic Dependencies Principle (ADP)

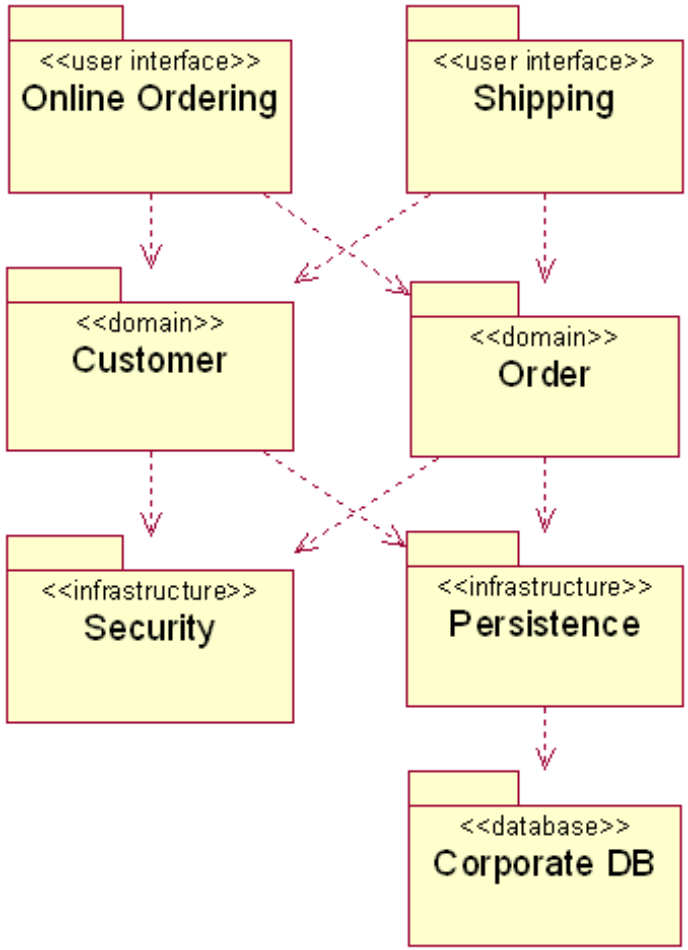
- The dependencies between packages must not form cycles

Style of UML Package Diagram

- Class Package Diagram Guidelines (Create Class Package Diagrams to Logically Organize Your Design)
  - Classes of a framework belong in the same package
  - Classes in the same inheritance hierarchy typically belong in the same package
  - Classes related to one another via aggregation or composition often belong in the same package
  - Classes that collaborate with each other a lot often belong in the same package

Style of UML Package Diagram

- Indicate Architectural Layers with Stereotypes on Packages



Style of UML Package Diagram

- Include Actors on Use-Case Package Diagrams
- Arrange Use-Case Package Diagrams Horizontally