3.6 Package Diagrams

Subject/Topic/Focus:

- Packaging, Decomposition

Summary:

- Dependency
- Package, Sub-Package
- Package Generalization

Literature:

- Fowler

Motivation

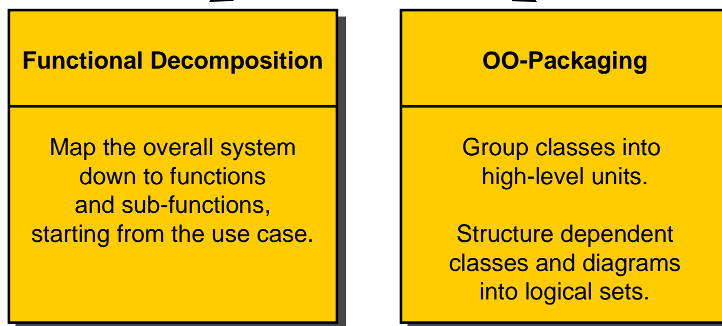
Typical systems grow complex, i.e., hundreds of classes.

- How do I control the complexity?
  - A class diagram should fit on a sheet of paper (A4).
  - A developer may grasp 7(+/-2) classes at a glance.
- How do I restrict scope (name space) and control change propagation?
- How do I build nested hierarchies of classes?
- How do I build layered architectures of classes?
- How do I show dependencies between classes at a higher abstraction level?
- How do I slice development work within a team?
- How do I specify interfaces between groups of classes, i.e., distinguish public from implementation-dependent classes?

Packaging

One of the oldest question arising in software development is:

How do you break down a **large** system into **smaller** systems?



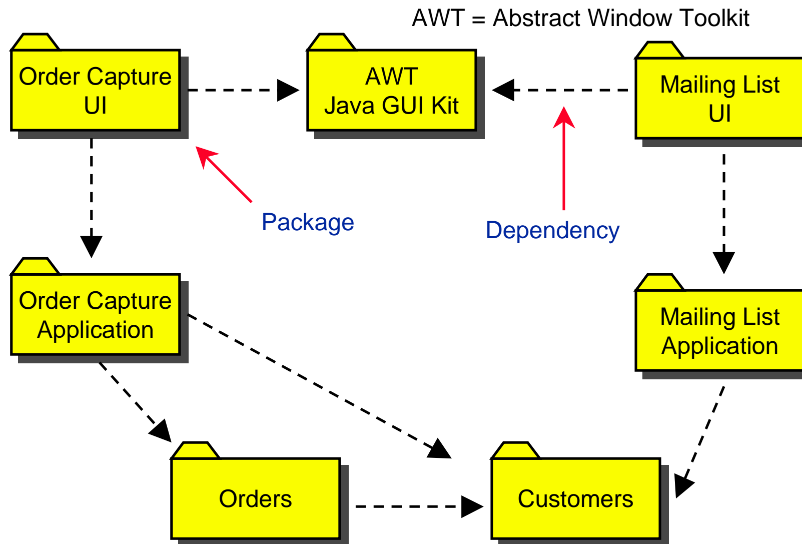
Package Diagrams

- Package diagrams and package dependencies result from class diagrams.
- Class A depends on class B, A → B, if changes to the definition of class B may cause an effect on class A.

  Examples:
  - class A sends a message to class B;
  - class A has class B as part of its data;
  - class A mentions class B as parameter to an operation.
- In an ideal oo-world, modifications which don't change a class interface should not affect any other classes.

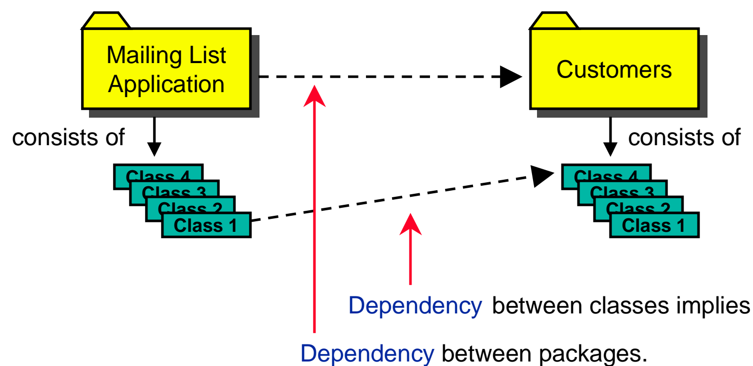
The art of large scale design involves **minimizing dependencies**.

Example: Package Diagram



Dependencies (1)

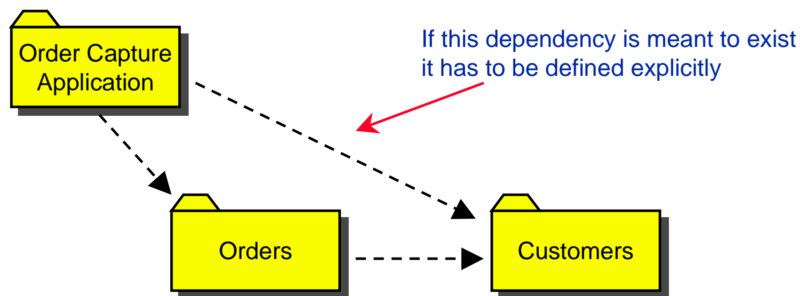
If any dependency exists between any two classes in two packages this implies a dependency between the two packages.



Dependencies (2)

There is a vital difference between package dependencies and compilation dependencies:

Dependencies between packages are **not** transitive !

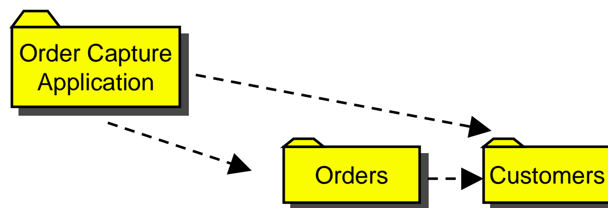


Dependencies in Programming Languages

Java	C++
❍ *Imports* are non-transitive. ❍ *Optional package visibility* to reduce the level of nested details. ❍ Possibility of generating *Facades* which are explicit interface classes between packages.	❍ *Includes* in C++ are transitive. ❍ No compiler supported "hide" option for nested classes. ❍ Complex *makefiles* which include tests on already defined objects.

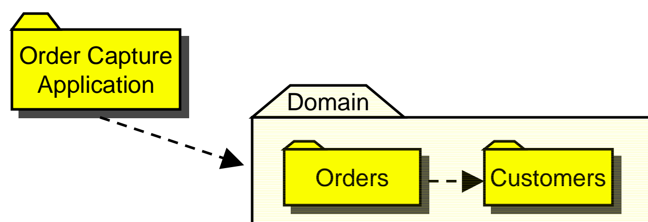
Nested Packages (1)

Instead of drawing many separate dependencies, the technique of sub-packages reduces redundant dependency information.



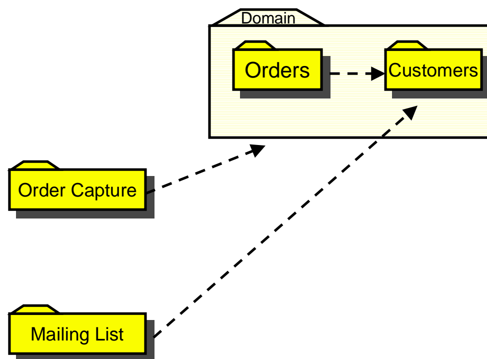
Nested Packages (2)

- Instead of drawing many separate dependencies, the technique of sub-packages reduces redundant dependency information.
- Draw dependencies to and from the overall package, instead of many separate dependencies.



Nested Packages (3)

- An overall package can contain:
  - classes
  - class diagrams
  - package diagrams

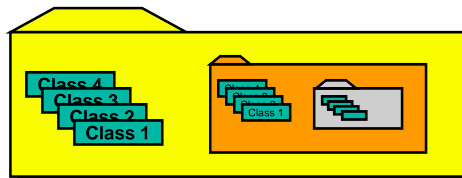


- Dependencies on an overall package represent dependencies on all members of the package.
- Separate dependencies on single members of the overall package can still occur.

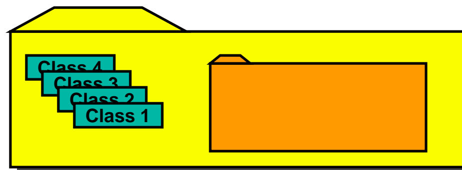
Dependencies: Visibility

What does it mean to draw a dependency to a package that contains subpackages?

- Convention **"transparent"**: gives visibility to the classes in the package <u>and</u> in the subpackage.



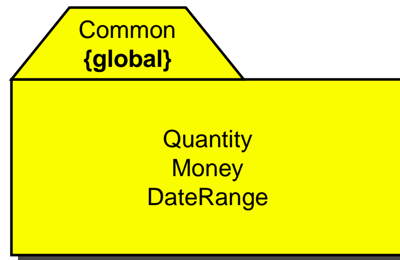
- Convention **"opaque":**gives visibility to the top-level classes only, <u>not</u> to the nested classes.



- Make clear, which convention you use in your project (by use of <<transparent>> or <<opaque>> stereotypes).

Global Dependency

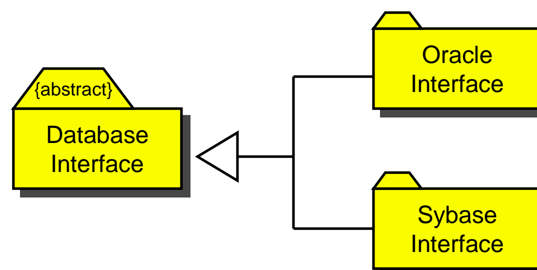
The **{global}** flag in a package tab signals that all packages in the system are dependent on this package.



Use this notational option **sparingly**!
A **change** in this package **effects all** packages!

Package Generalization

- Generalization between packages means that the specific package must conform to the interface of the general package.
- To emphasize the role of a general interface, the package can be marked as `{abstract}`.
- Example: An abstract database interface consisting of several classes is implemented either for Oracle or for Sybase.



When to Use Package Diagrams

- Use package diagrams for distributing and balancing work between development groups.
- Package diagrams are helpful to explore the possibilities of partitioning tasks in the development process.
- Package diagrams are extremely useful for testing purposes: rather apply tests on packages (i.e., several interdependent classes) than on single routines.