Package Diagram

Meet 14

PACKAGE DIAGRAM

AGENDA

- What is A Package ?
- Package Notation
- Package Stereotype
- Package Visibility
- Relation Among Packages
- Dependency stereotypes (Import & Access)
- Merging Package
- Use Case Package
- Package Architecture Concepts
- Exam Sample Questions (OMG-OCUP2-FOUND100)

PACKAGE DIAGRAM

WHAT IS A PACKAGE ?

Package

Package is

A general-purpose mechanism for organizing elements into groups
[Package provide Namespace]

Package Container For

any of the logical model elements such as
Use Case diagrams,
Sequence diagrams,
Class diagrams,
and even other packages.

PACKAGE NOTATION

Graphically

Package is rendered as a tabbed folder

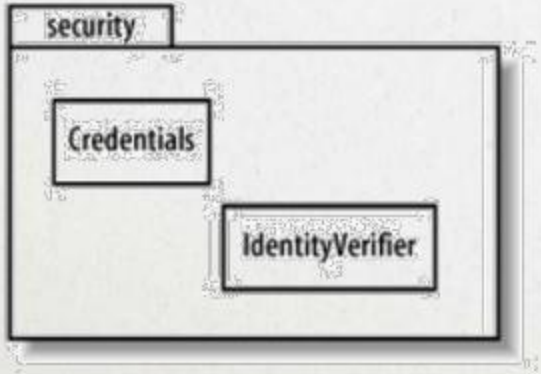
Package

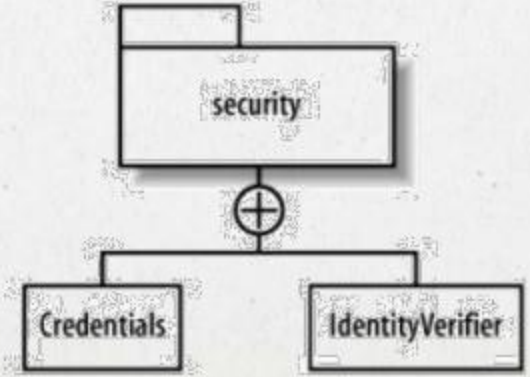
Package Content

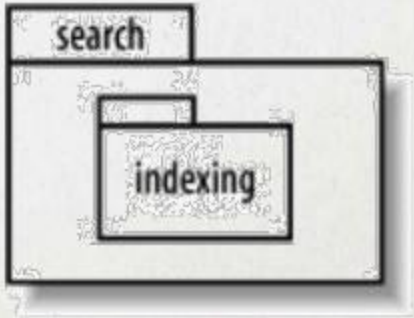
can be drawn

inside the package

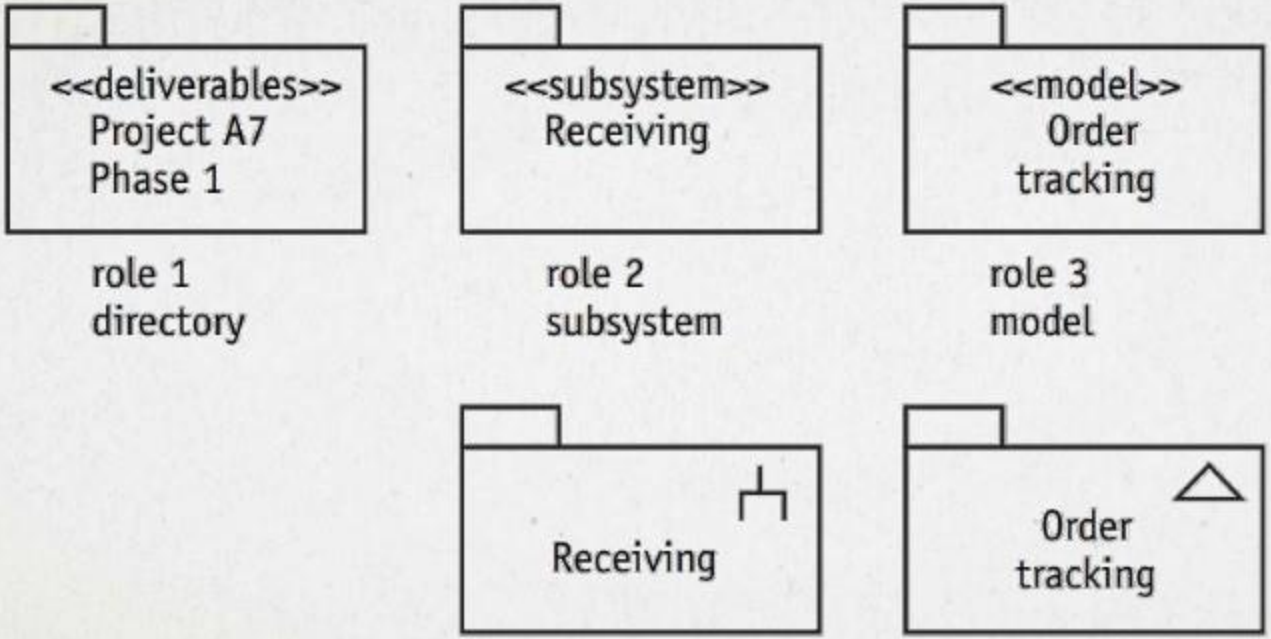
OR outside the package attached by a line







PACKAGE STEREOTYPE



Package

PACKAGE VISIBILITY

Elements in a package may have public or private visibility

Visibility	Visibility description
Public	Elements within package are accessible outside the package
Private	Elements available only to other elements inside the package

Package

+ public
- Private

RELATION AMONG PACKAGES

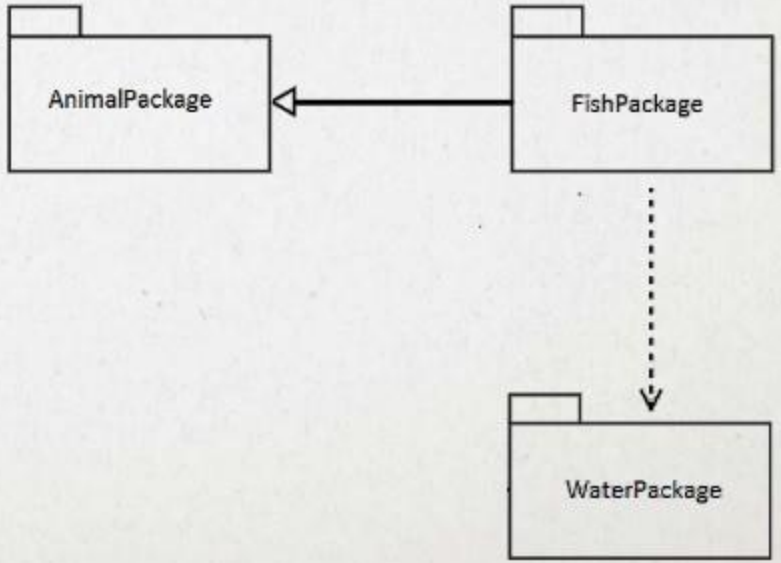
dependency relationship

- Two packages one depend on correct occurrence of the other
- means that at least one class in a package has to **communicate** with at least one class in the other package
- Eg : relation between FishPackage , WaterPackage

Package

Generalization relationship

- Parent-child relationship
- means that at least one class in a package has to **inherit** with at least one class in the other package
- Eg : relation between FishPackage , AnimalPackage



DEPENDENCY STEREOTYPES (IMPORT & ACCESS)

Package Import

- import can be a public import Or private import with public as the default
- Public import use <<import>> stereotype
- Private import use <<access>> stereotype

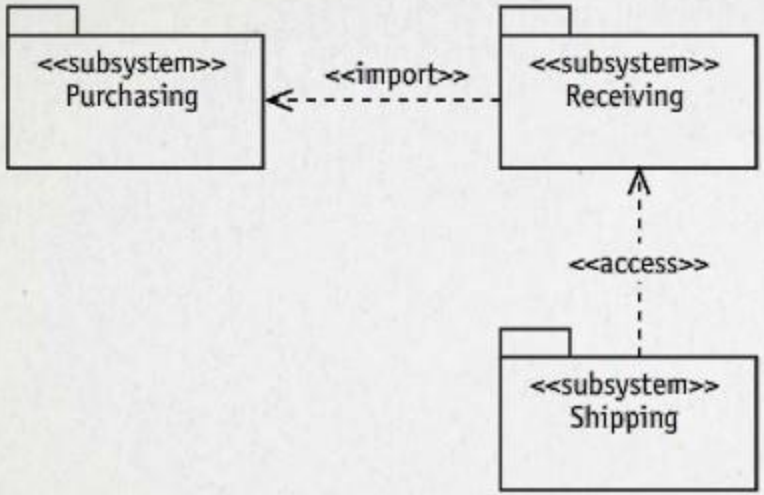
A public import <<import>>

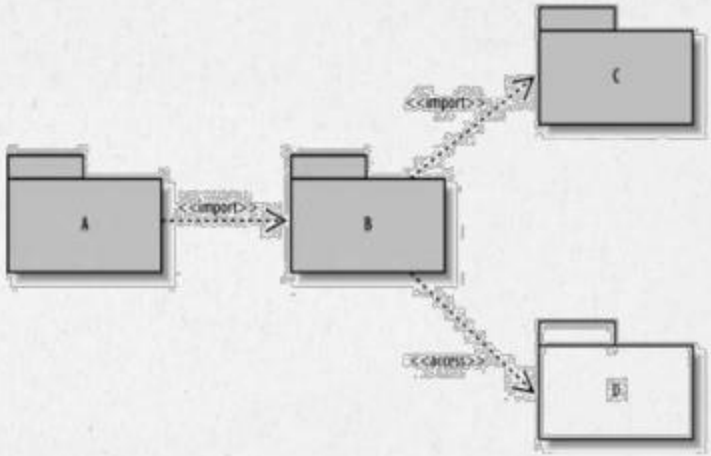
means imported elements have public visibility inside the importing namespace;

A private import <<access>>

means imported elements have private visibility inside the importing namespace.

DEPENDENCY STEREOTYPES (IMPORT & ACCESS)





Package

If Package X import **receiving** „
it will **see purchasing** (public import) – include in its runtime

If Package Y import **shipping** „
it **won't see receiving** (private import - access) – access only means not include in its runtime

MERGING PACKAGE

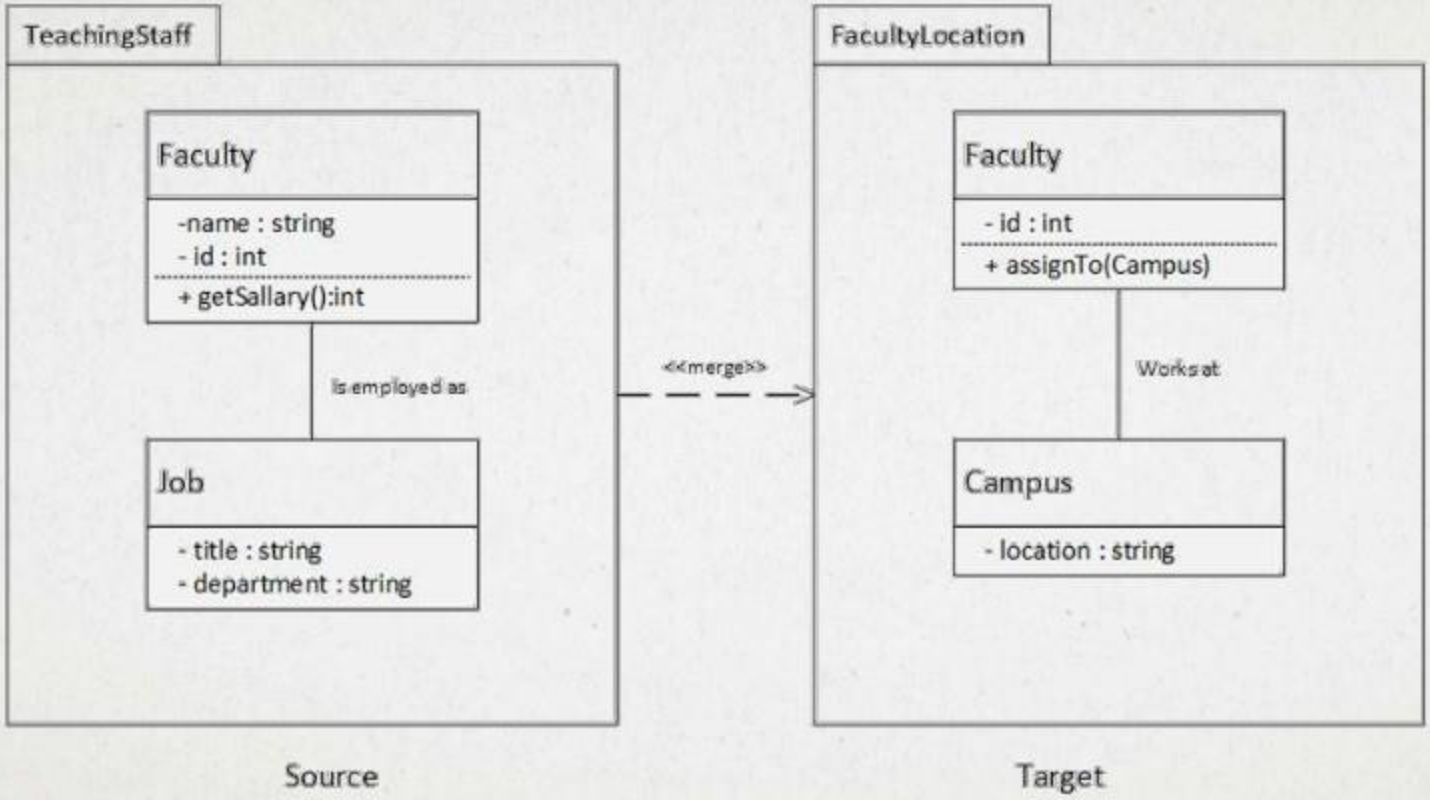
Mechanism to merge the content of package

When package merges another package , any class of the same type and name automatically extends (OR has a generalization relationship) to the original one

Notes :

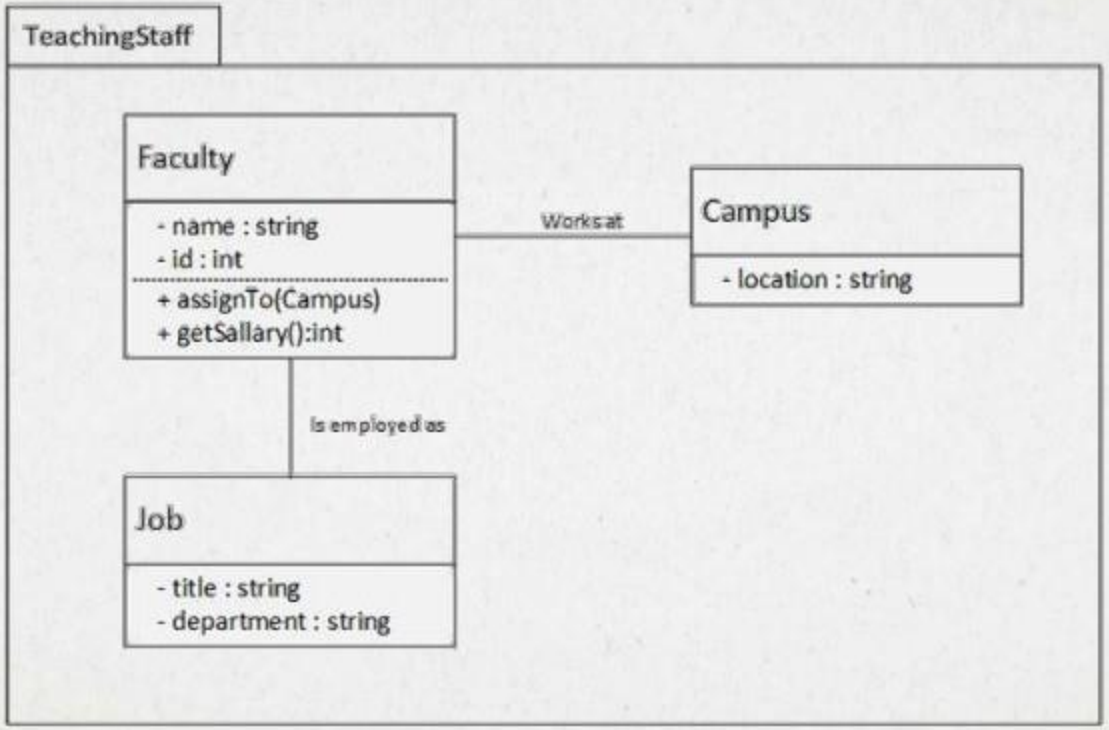
- Private members of a package aren't merged with anything.
- Subpackages within the merged package are added to the merging package if they don't already exist.
- If a subpackage with the same name already exists in the merging package, another merge is started between the two subpackages.

MERGING PACKAGE - SAMPLE



Package

MERGING PACKAGE – SAMPLE RESULT



Result After Merging

Package

USE CASE PACKAGE

Use case packages

- organize the functional behavior of a system during analysis
- provide understandable terms for team members outside analysis team

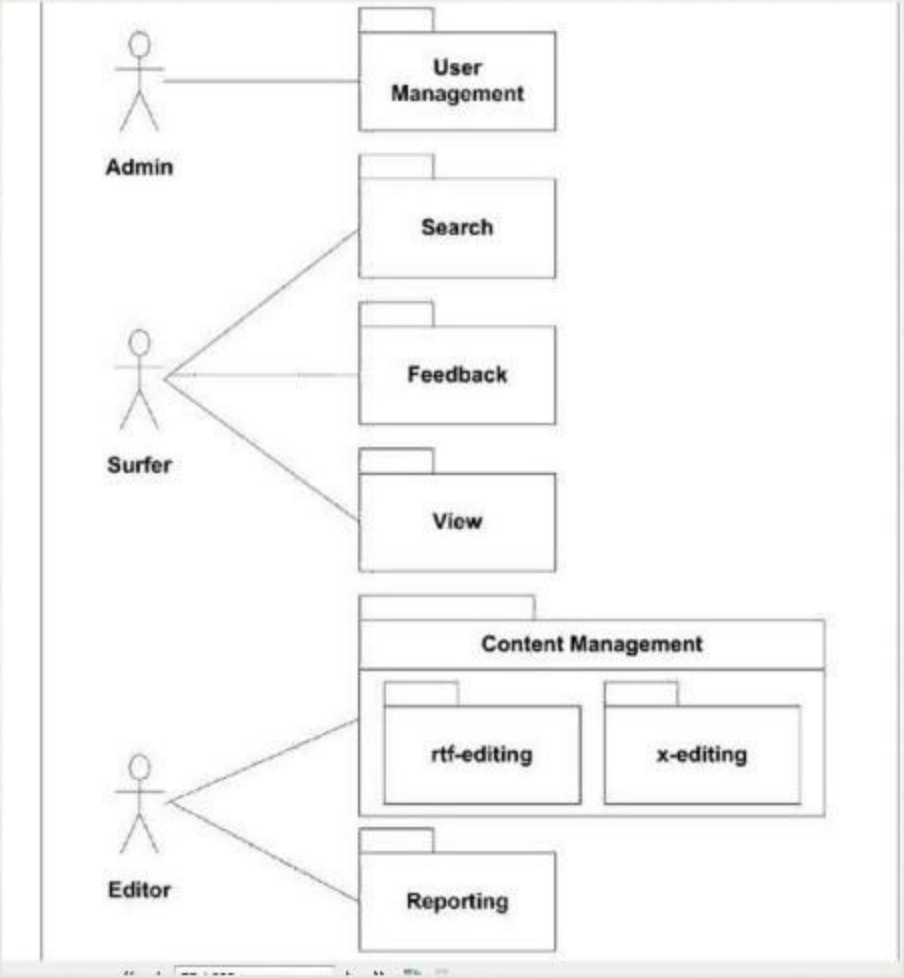
Package

Managers

can depend on use case package
to discuss project at appropriate level of details without getting down into details

Tracking use case packages means tracking customer values

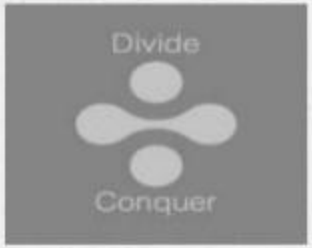
USE CASE PACKAGE - SAMPLE



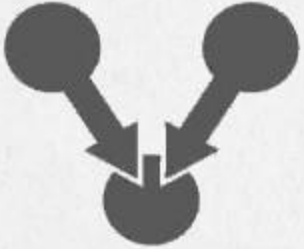
Package

PACKAGE ARCHITECTURE CONCEPTS

Package



Decomposition

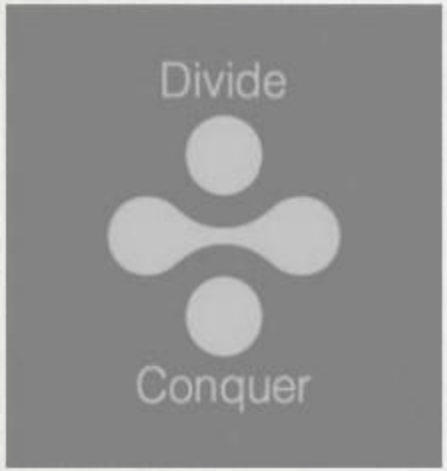


Uses

Generalization

Layered

PACKAGE – DECOMPOSITION STYLE



- Architect tend to attach a problem by use of divide and conquer technique (divide complex problem to smaller ones)
- Show the structure of modules and submodules
- divide responsibilities into manageable pieces (implementation units)
- Code organization into modules and show how system responsibilities Are partitioned across them
- Decomposition defined modules that may appear in other styles like uses, generalization, layered, other module based views
- Usually decomposition is first step architect start with to model their system (First step towards details architecture)

PACKAGE – DECOMPOSITION STYLE DESIGN CRITERIA

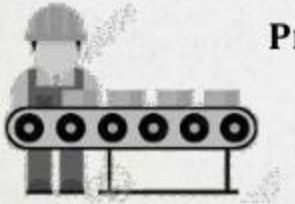
Build versus buy decisions

Some modules may be bought from market place, or reuse of old projects or obtained as open source.

Achievement of certain quality attributes

For example, to support Modifiability:

- Information hiding design principle will reduce side effects.
- Limit global impact of local design changes.
- Eg. Remote control with TV or Air Conditioner



Product line implementation

- make products of product family, make some sort of separation
- Separate common modules from variable modules that differ across products

Team Allocation

- make responsibilities done in parallel, separate modules that can be allocated to different team should be defined
- Skills of development team may change decomposition,

Module

Decomposition

PACKAGE – DECOMPOSITION STYLE USAGE

Support the learning process about a system for newcomers to the project

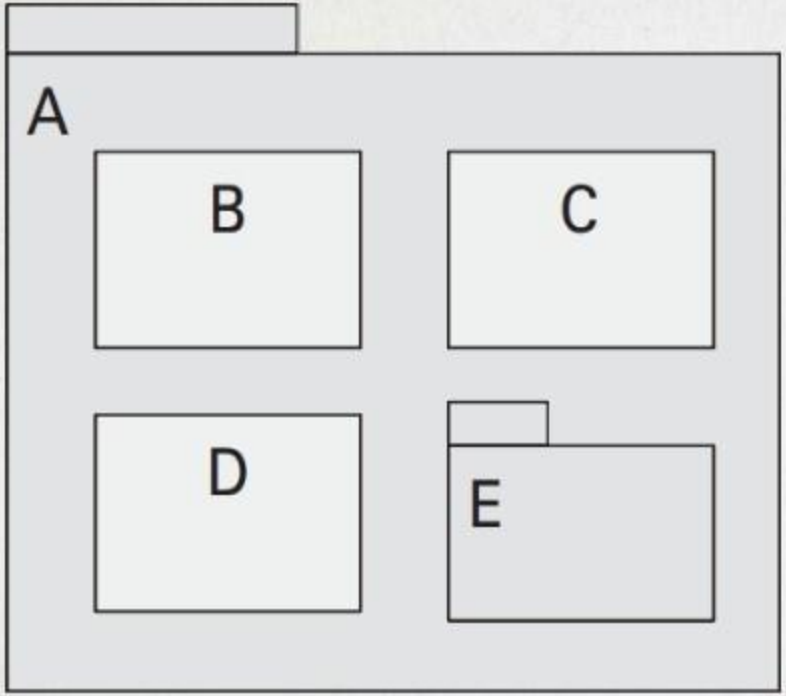
input for the work assignment view of a system

Show effects of change in addition to uses style

Module

Decomposition

PACKAGE – DECOMPOSITION STYLE IN PRACTICE



UML Diagram

Behavior Hiding Module
- Function Driver Module
  - Air Data Computer Module
  - Audible Signal Module
  - Computer Fail Signal Module
  - Doppler Radar Module
  - Flight Information Display Module
  - Forward Looking Radar Module
  - Head-Up Display Module
  - Inertial Measurement Set Module
  - Panel Module
  - Projected Map Display Set Module
  - Shipboard Inertial Nav System Module
  - Visual Indicator Module
  - Weapon Release Module
  - Ground Test Module
  - Shared Services Module
- Mode Determination Module
  - Panel I/O Support Module
  - Shared Subroutine Module
  - Stage Director Module
  - System Value Module

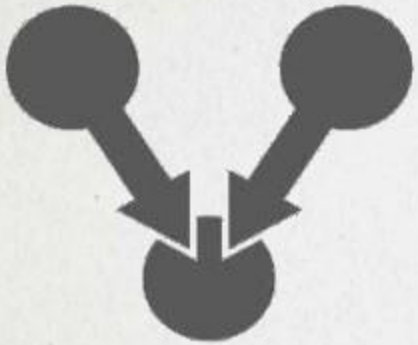
Software Decision Hiding Module
- Application Data Type Module
  - Numeric Data Type Module
  - State Transition Event Module
- Data Banker Module
  - Singular Values Module
  - Complex Event Module
  - Filter Behavior Module

List Catalog

Module

Decomposition

PACKAGE – USES STYLE



Uses

- The uses style shows how modules depend on each other
- helpful for planning because it helps define subsets and increments of the system being developed
- module uses another module if its correctness depends on the correctness of the other
- Goes one step further to reveal which modules use which other modules. This style tells developers what other modules must exist for their portion of the system to work correctly.

Module

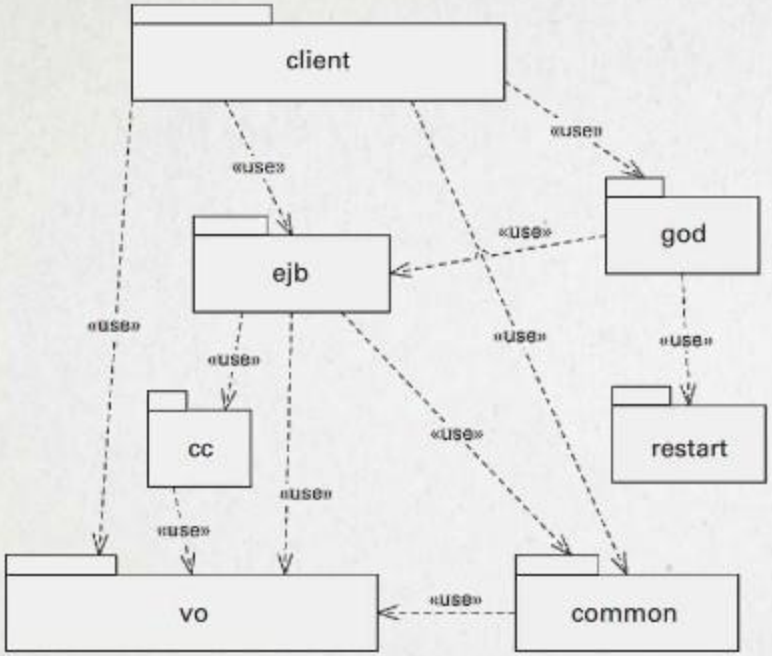
PACKAGE – USES STYLE USAGE

Planning incremental development and subsets

Debugging & testing

Message the effect of change

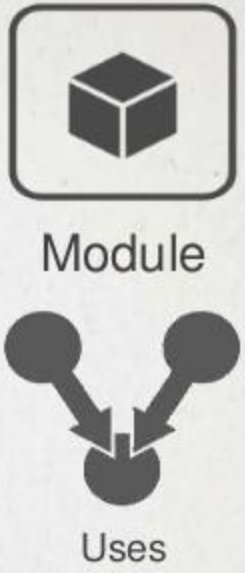
PACKAGE – USES STYLE - IN PRACTICE



UML Diagram

used module \ using module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	1	0	0	1	0	0	0
cc	0	1	0	0	0	0	0
god	1	0	0	0	0	0	0
restart	0	0	0	1	0	0	0
common	1	1	0	0	0	0	0
vo	1	1	1	0	0	1	0

DSM
(dependency structure matrix)



PACKAGE – GENERALIZATION STYLE

- The generalization style results when the is-a relation is employed
- The parent module is a more general version of the child modules (The parent module owns the commonalities, and the variations are manifested in the children)
- Extensions can be made by adding, removing, or changing children
- A change to the parent will automatically change all the children that inherit from it
- Generalization may represent inheritance of either interface, implementation, or both

PACKAGE – GENERALIZATION STYLE USAGE

Reusable Modules

enable incremental Extension

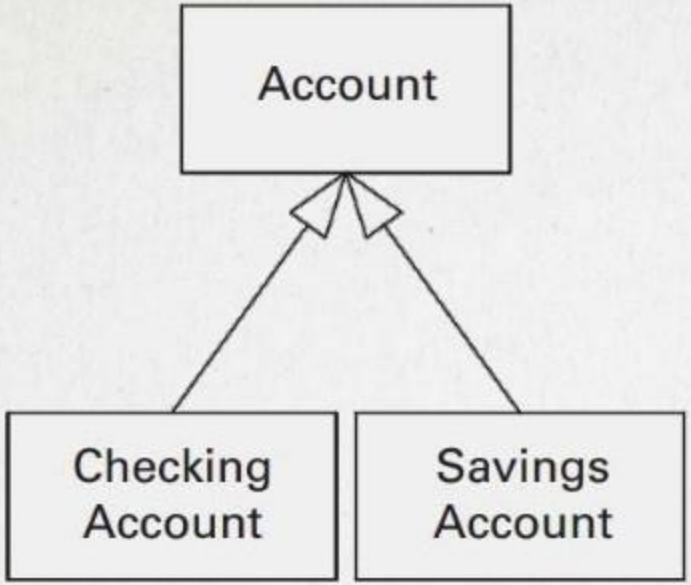
Capturing commonalities with variations as children

Expressing inheritance In Object oriented

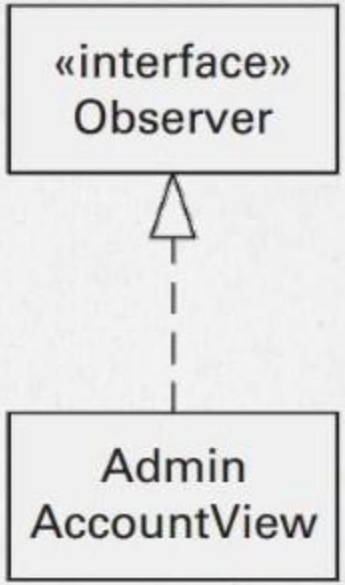
Module

Generalization

PACKAGE – GENERALIZATION STYLE – IN PRACTICE



Inheritance



Realization(implementation)

PACKAGE – LAYERED STYLE

Module

Layered

- The layered style, like all module styles, reflects a division of the software into units
- Each layer represents a grouping of modules that offers a cohesive set of services
- The layered view of architecture, shown with a layer diagram, is one of the most commonly used views in software architecture
- Layering has one more fundamental property: The layers are created to interact according to a strict ordering relation
- unidirectional allowed-to-use relation with each other.
- n-tier architecture is a physical structuring mechanism, while a n-layer architecture is a logical structuring mechanism.

PACKAGE – LAYERED STYLE USAGE

Promoting reuse

Achieving separation of concerns

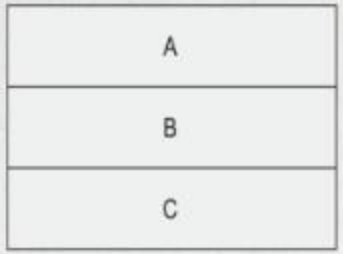
modifiability & portability

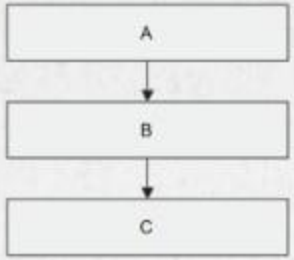
Managing complexity & facilitating code structure communication to developers

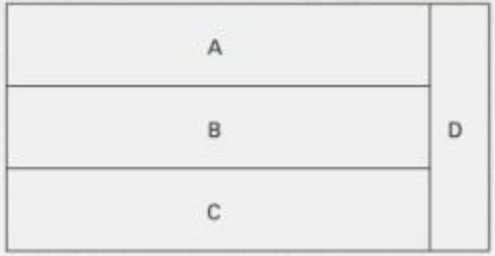
Module

Layered

PACKAGE – LAYERED STYLE – IN PRACTICE







Module

Layered