# CSCCORE1I
# Object Oriented Software Development
*with C++*

# Lecture 14: Unified Modeling Language (UML)

Dr Kapil Dev
devk@hope.ac.uk

## Learning Objectives

Understand the following UML Diagrams

- Package Diagrams
- Component Diagrams
- Code generation from UML Diagrams

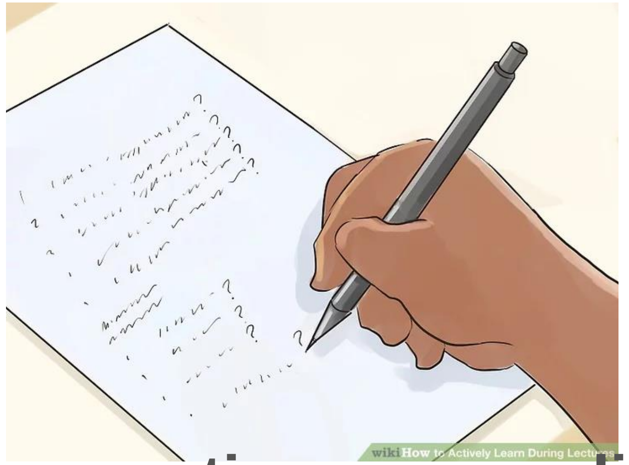
# Learning Actively in Lectures

Sit in the front

Stretch and stay hydrated

Find a study buddy

Jot down main ideas



Write questions as you listen

Stay focused

# UML Diagrams

## Structural Diagrams

- Class;
- Object
- Component
- Deployment
- Composite Structure
- *Package*

## Behavioral Diagrams

*Represent the dynamic aspects.*

- Use case
- Sequence;
- Collaboration
- Statechart
- Activity

## Package Diagrams

# Package Diagrams

- A package is a grouping of model elements
- A package can contain model elements of different kinds, including other packages to create hierarchies.

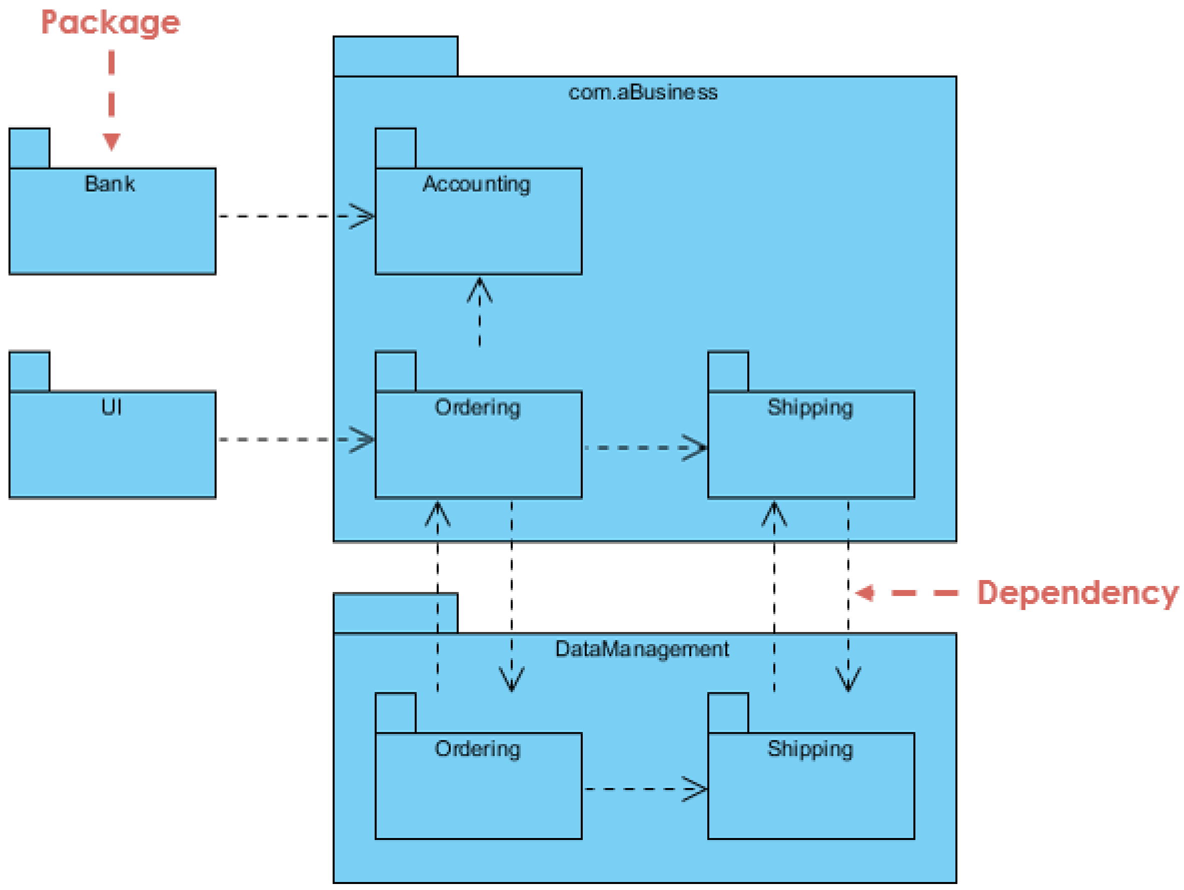
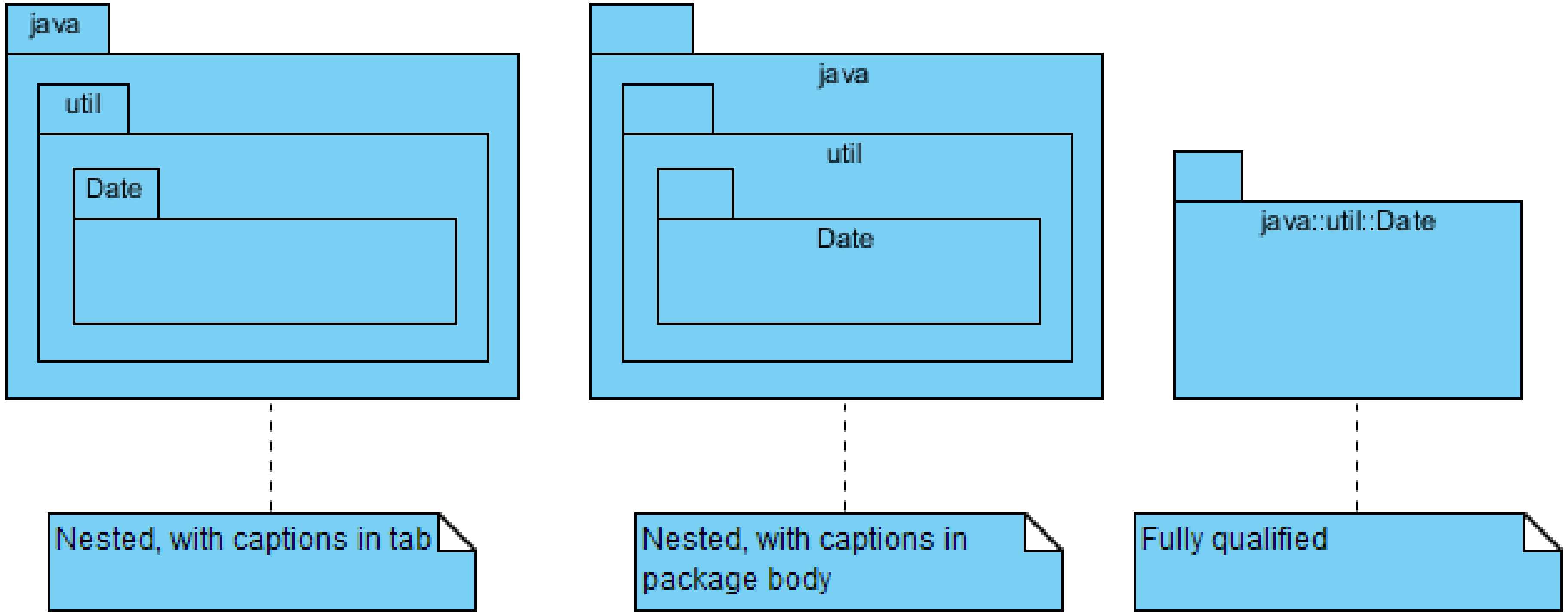


Fig: classes grouped into packages

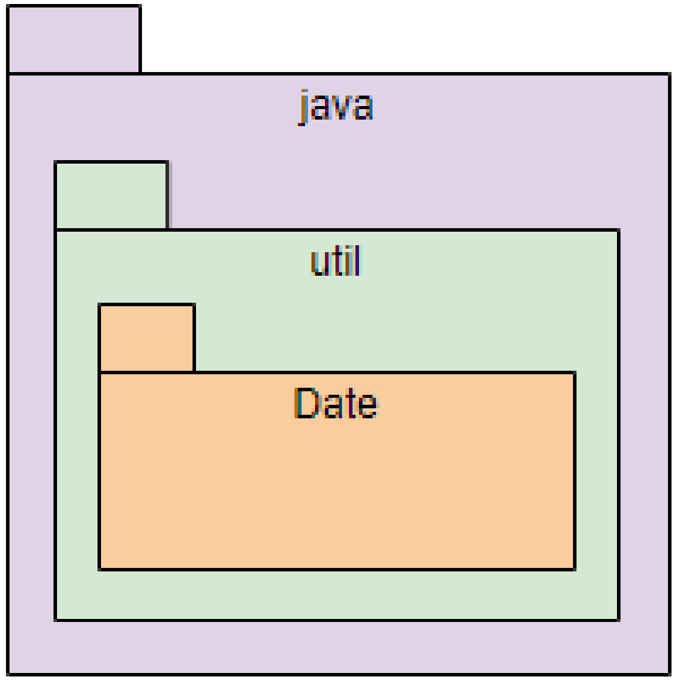
# Package Diagrams

- Package diagram follows hierarchal structure of nested packages.
- Atomic module for nested package are usually class diagrams.
- Classes inside different packages could have the same name


java
util
Date
java
util
Date
java::util::Date
Nested, with captions in tab
Nested, with captions in package body
Fully qualified

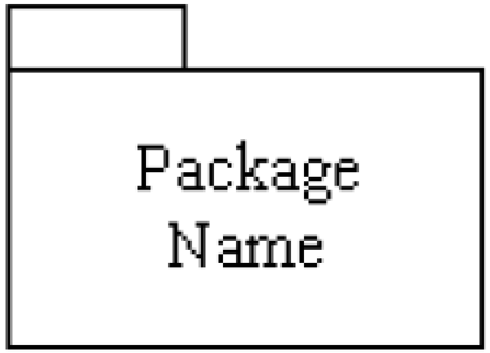
# Why use Package Diagrams?

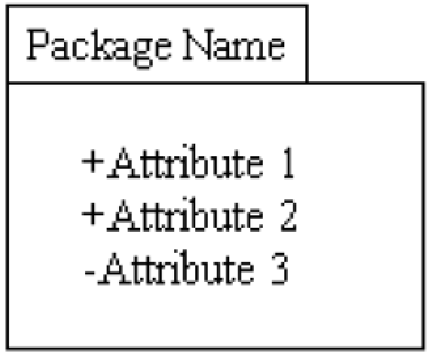
- Increases the level of abstraction for complex diagrams
- Depict a high-level overview of your requirements or architecture/design
- To logically modularize a complex diagram
  - For example, to organize Java source code

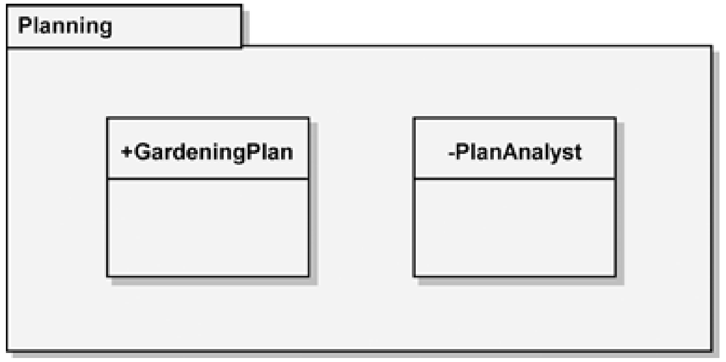


## Package Diagrams: Notation

- Represented as tabbed folders



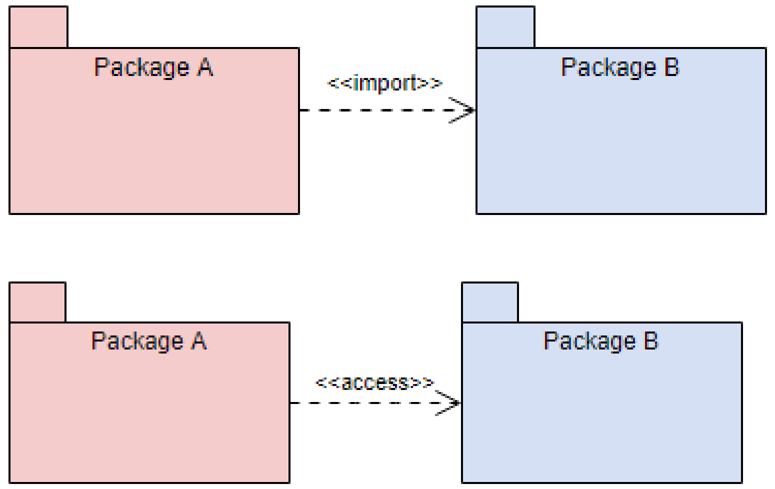


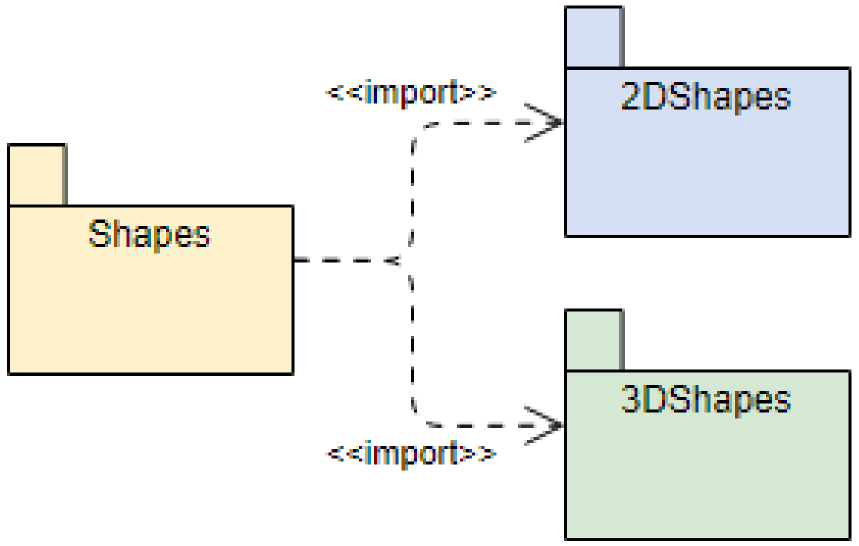


- Can use visibility markers
  - + Public
  - - Private
  - # Protected

# Package Diagram Dependency

- There are two sub-types involved in dependency. They are <<access>> & <<import>>



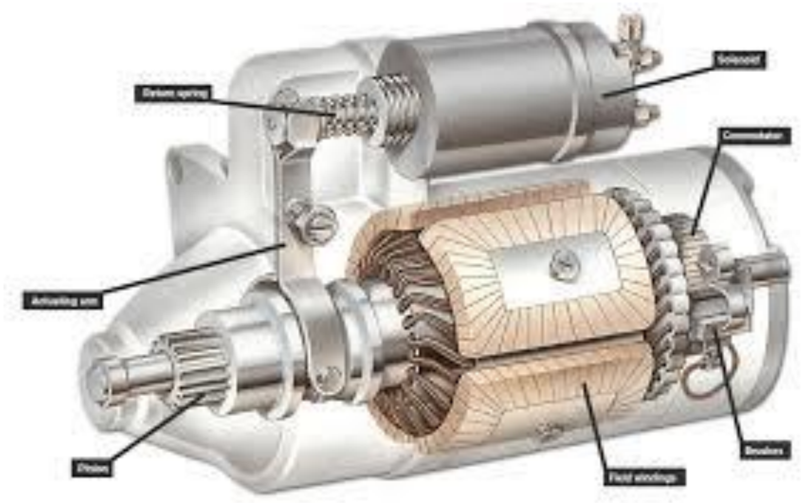


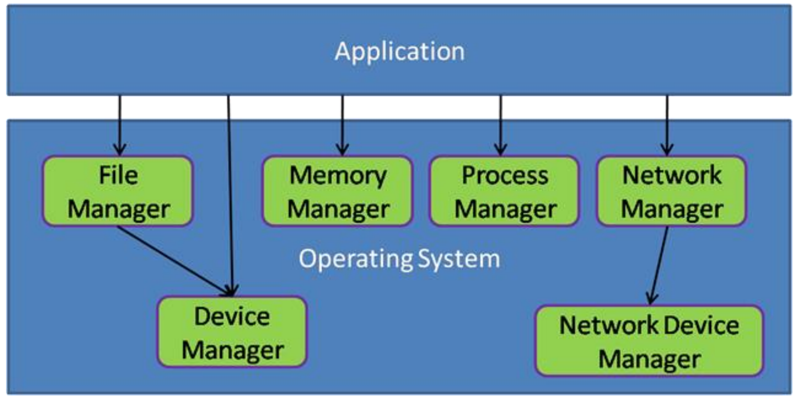
Import and access are really two sides of the same coin: <<import>> is a public package import, whereas <<access>> is a private package import.

## Component Diagrams

# Components in UML

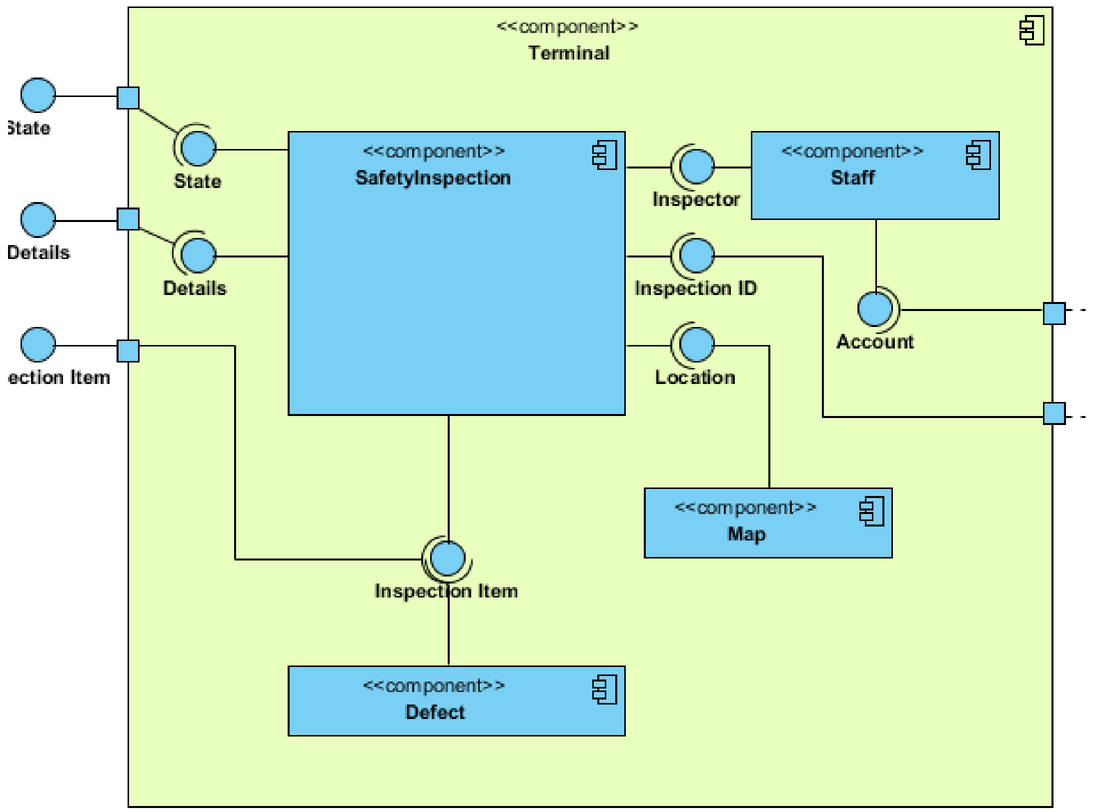
- A component represents a reusable piece of software that provides some meaningful aggregate of functionality.
- At the lowest level, a component is a cluster of classes that are themselves cohesive.
- Each class in the system must live in a single component.





# What is a Component Diagram?

- Helps to model the physical aspect of an Object-Oriented software system.
- Illustrates the architectures of the software components and the dependencies between them.
- Those software components including run-time components, executable components also the source code components.
- Often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

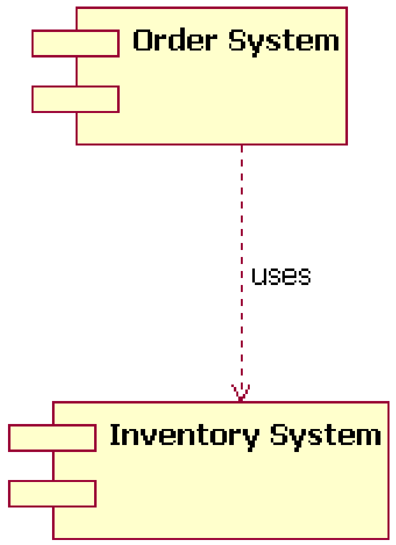


## Why use a component diagram?

- Allow an architect to verify that a system’s required functionality is being implemented by components, thus ensuring that the eventual system will be acceptable
- Useful communication tools for various groups. The diagrams can be presented to key project stakeholders and implementation staff.
- Developers find the component diagram useful because it provides them with a high-level, architectural view of the system that they will be building

# Notations

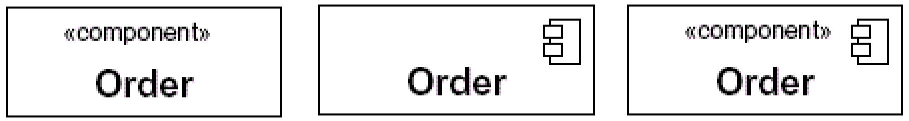
- Figure shows a simple component diagram
  - Example shows a relationship between two components: an Order System component that uses the Inventory System component.
  - Drawing a component is now very similar to drawing a class on a class diagram
  - Which means that the notation rules that apply to the class classifier also apply to the component classifier.



Order System's general dependency

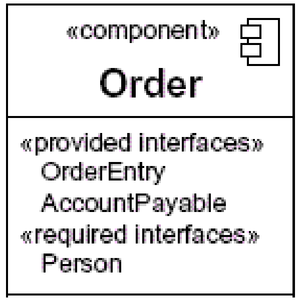
# Notations

- Component:
  - drawn as a rectangle



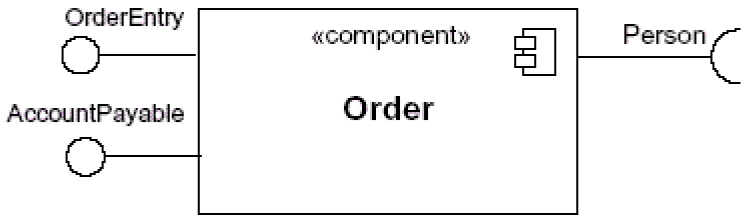
The different ways to draw a component

- Interface
  - interfaces provided represent the formal contract of services



- Additional compartment here shows the interfaces that the Order component provides and requires
- the component provides the interfaces of OrderEntry and AccountPayable. Additionally, the component also requires another component that provides the Person interface.

  - another way to show a component's provided and required interfaces



# Component Relationships Modeling

- When showing a component's relationship with other components, the full circle (lollipop) and half-circle (socket) notation must also include a dependency arrow (as used in the class diagram).

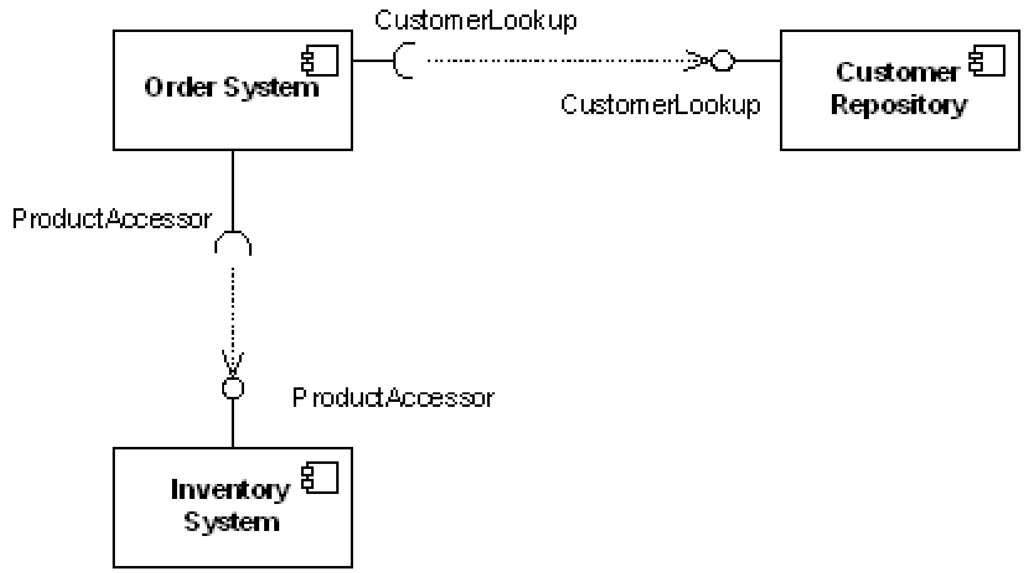


Figure: Component diagram that shows how the Order System component depends on other components

## Showing a component's internal structure

- Draw the component larger than normal and place the inner parts inside the name compartment

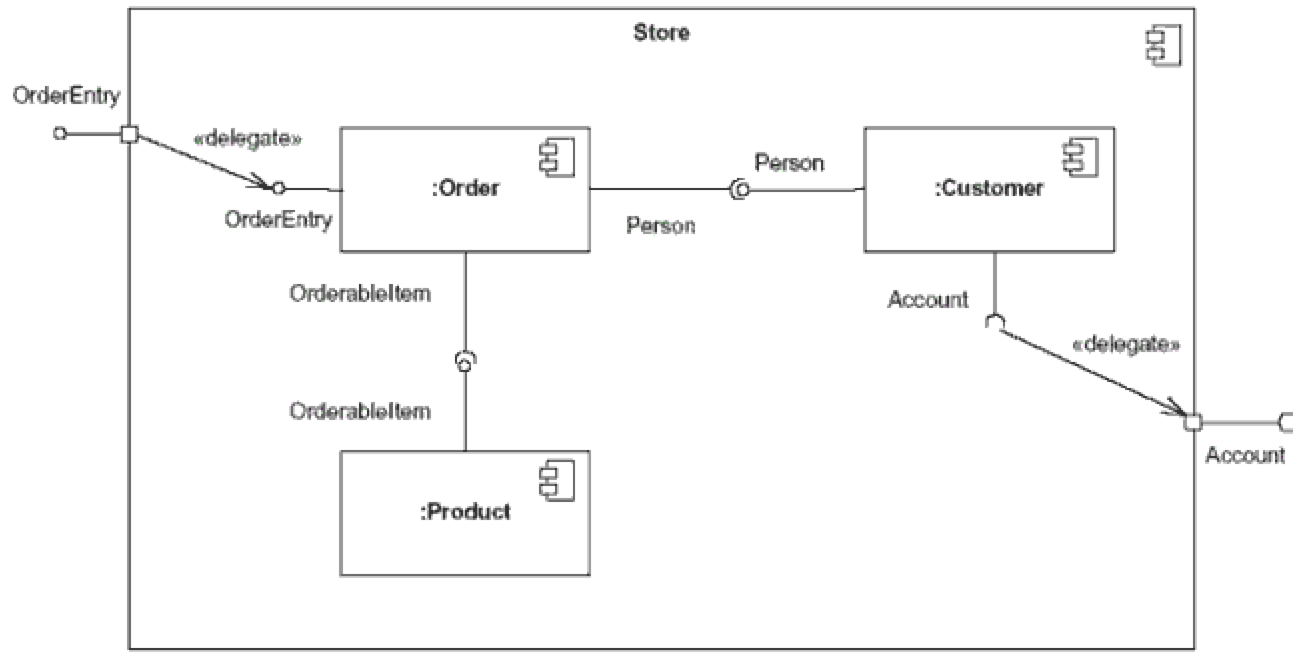
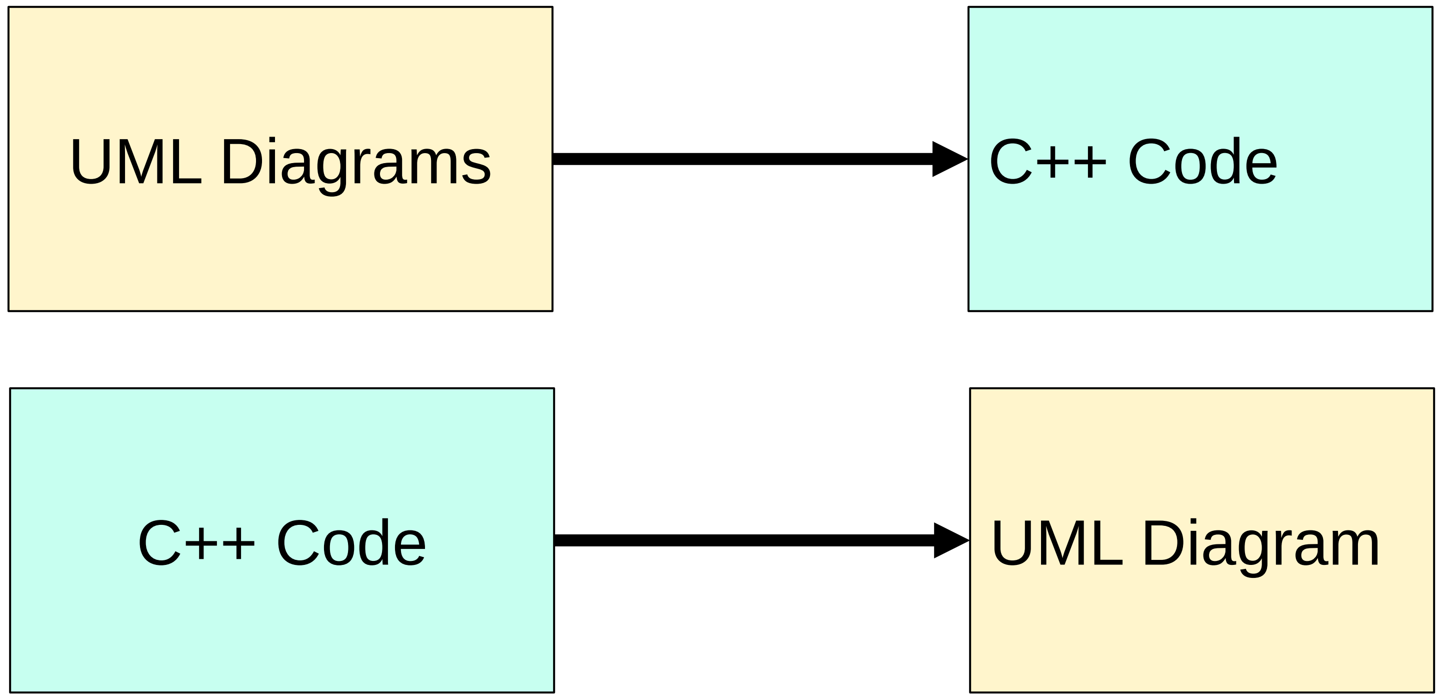


Figure: Component's inner structure is composed of other components

## C++ Code Generation

## Round-trip engineering

- Ability of a UML tool to perform code generation from models, and model generation from code (a.k.a., reverse engineering)



## Code Generation Tools

- Means that user creates UML diagrams, which have some connected model data, and the UML tool derives from the diagrams part or all of the source code for the software system
- There is some debate among software developers about how useful code generation as such is
- An often cited criticism is that the UML diagrams lack the detail that is needed to contain the same information as is covered with the program source

## Reverse Engineering

- Means, that the UML tool reads program source code as input and *derives* model data and corresponding graphical UML diagrams from it
- Challenging to do:
  - source code often has much more detailed information
  - language features, like class- or function templates of the C++ programming language, are notoriously hard to convert automatically to UML diagrams

## UML to Code Tools

- Rational Rhapsody, a modeling environment based on UML
  - generate software applications in various languages including C, C++, Ada, Java and C#
  - Production quality code means the generated code can be deployed with your product, you don’t need to re-write it
  - Code is generated from at least the following important UML diagrams: Class Diagrams, Sequence Diagrams and Activity Diagrams
- Sparx Systems Enterprise Architect is a visual modeling and design tool based on the OMG UML
- BOUML: free UML 2 tool box including a modeler allowing you to specify and generate code in C++, Java, Idl, Php, Python and MySQL.

# Thank you!