Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 18:

Interfaces and Components

Design Workflow: *Architecture and Subsystems*

- Place in the *Design Workflow:*
  - **Architectural Design**
  - Design a Use Case
  - Design a Class
  - **Design a Subsystem**
- concerned with breaking a system up into subsystems that are as independent as possible.
  - Interactions between subsystems are mediated by interfaces.

Interfaces

- Interfaces allow software to be designed to a contract rather than to a specific implementation.
- An interface specifies a named set of public features.
  - Interfaces separate specification of functionality from implementation.
  - Interfaces may be attached to classes, subsystems, components, and any other classifier and define the services offered by these.
  - If a classifier inside a subsystem realizes a public interface, the subsystem or component also realizes the public interface.
  - Anything that realizes an interface agrees to abide by the contract defined by the set of operations specified in the interface.

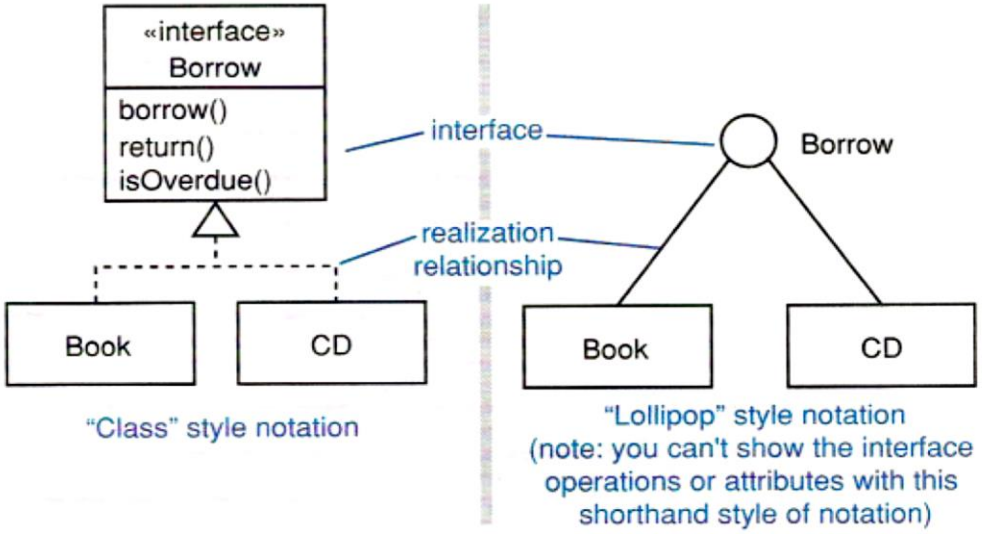
Interface Semantics

- A classifier realizing an interface has the following responsibilities for each feature:

Interface specifies	Realizing classifier
Operation	Must have an operation with the same signature and semantics
Attribute	Must have public operations to set and get the value of the attribute – the realizing classifier is *not* required to actually have the attribute specified by the interface, but it must behave as though it has
Association	Must have an association to the target classifier – if an interface specifies an association to another interface, the implementing classifiers of these interfaces must have an association between them
Constraint	Must support the constraint
Stereotype	Has the stereotype
Tagged value	Has the tagged value
Protocol (e.g., as defined by a protocol state machine – see Section 21.2.1)	Must realize the protocol

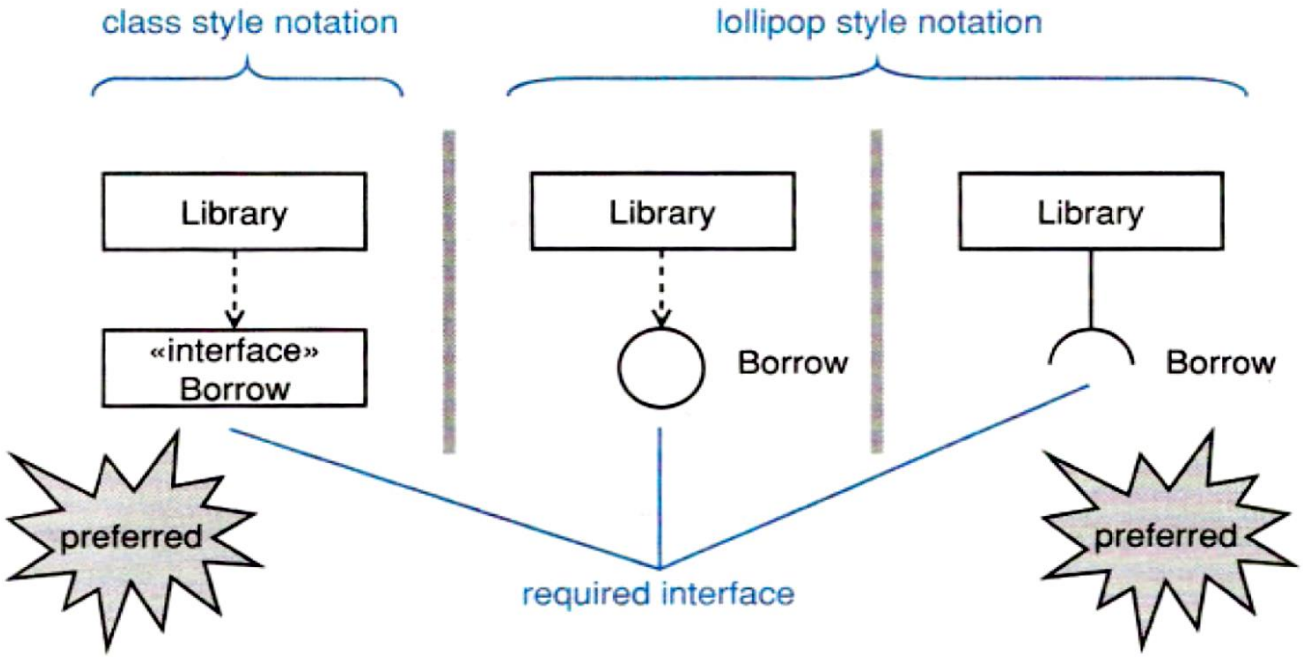
Provided Interface

- An interface provided by a classifier:
  - the classifier realizes the interface;
  - use the" class" style notation when you need to show the operations on the model;
  - use the shorthand "lollipop" style notation when you just want to show the interface without operations.



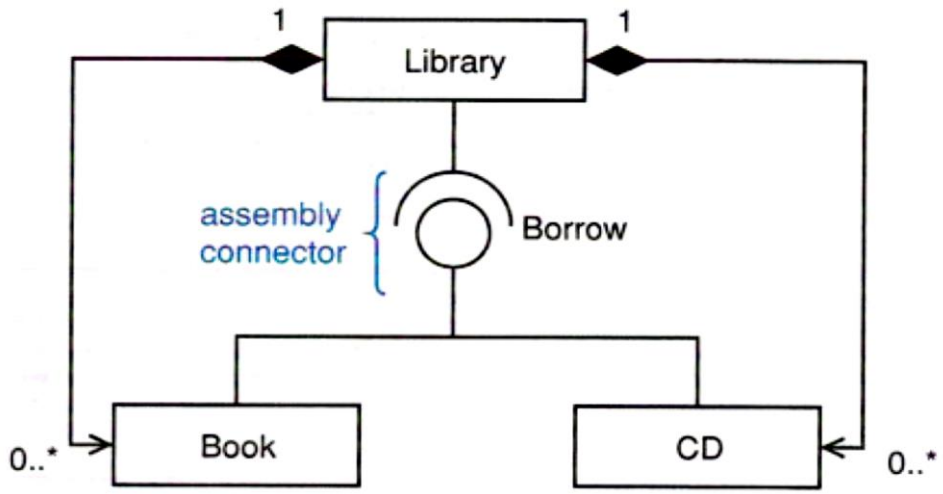
Required Interface

- An interface required by a classifier:
  - the classifier requires another classifier that realizes the interface;
  - show a dependency to a class style interface, a lollipop style interface, or use an assembly connector.



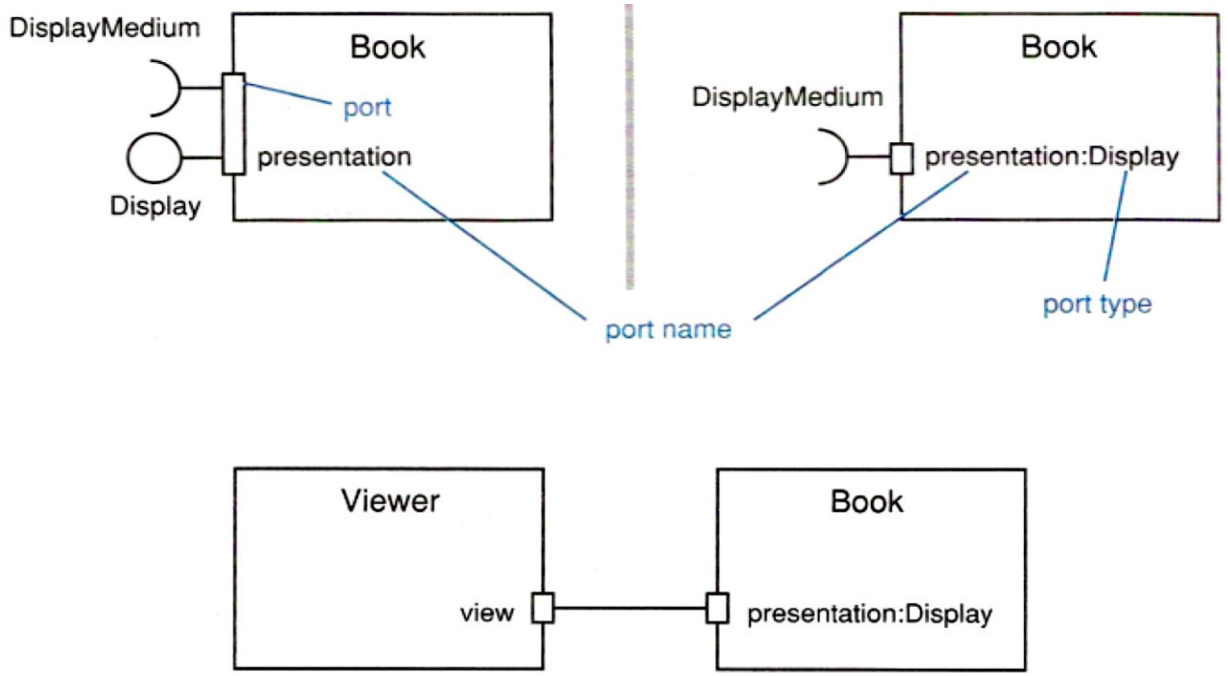
Assembly Connector

- Joins provided and required interfaces.



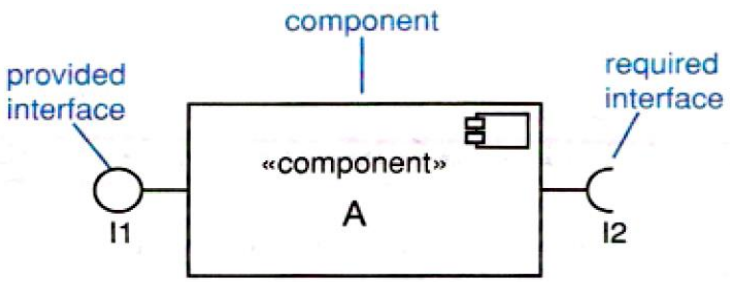
Ports

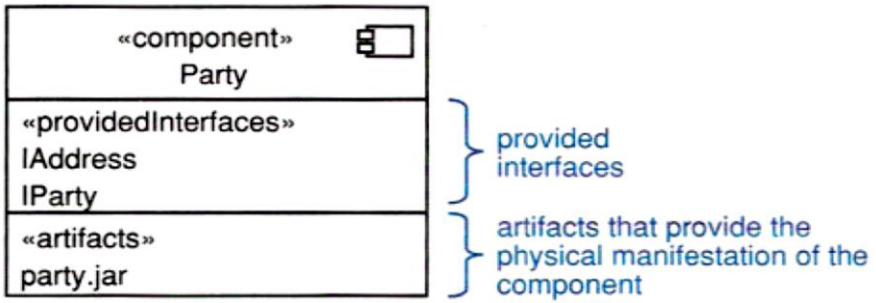
- Port - groups a semantically cohesive set of provided and required interfaces:
  - may have a name, type, and visibility.



Components

- Component - a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment:
  - may have attributes and operations;
  - may participate in relationships;
  - may have internal structure;
  - its external behavior is completely defined by its provided and required interfaces;
  - components are mapped to one or more artifacts (runtime manifestations).





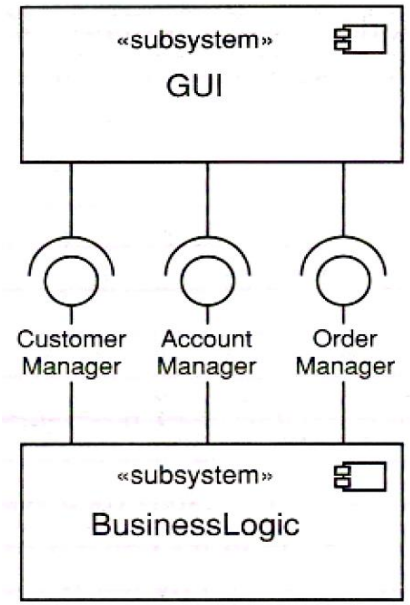
CBD and Subsystems

- Component-Based Development (CBD) is about constructing software from plug-in parts:
  - you use interfaces to make components "pluggable";
  - by designing to an interface, you allow the possibility of many different realizations by many different components.

- Subsystem - a component that acts as a unit of decomposition for a larger system:
  - a component stereotyped «subsystem»;
  - is used to decompose a large system into manageable chunks;
  - breaking a system down into subsystems is a key to successful CBD using UP.

Subsystems: Applications

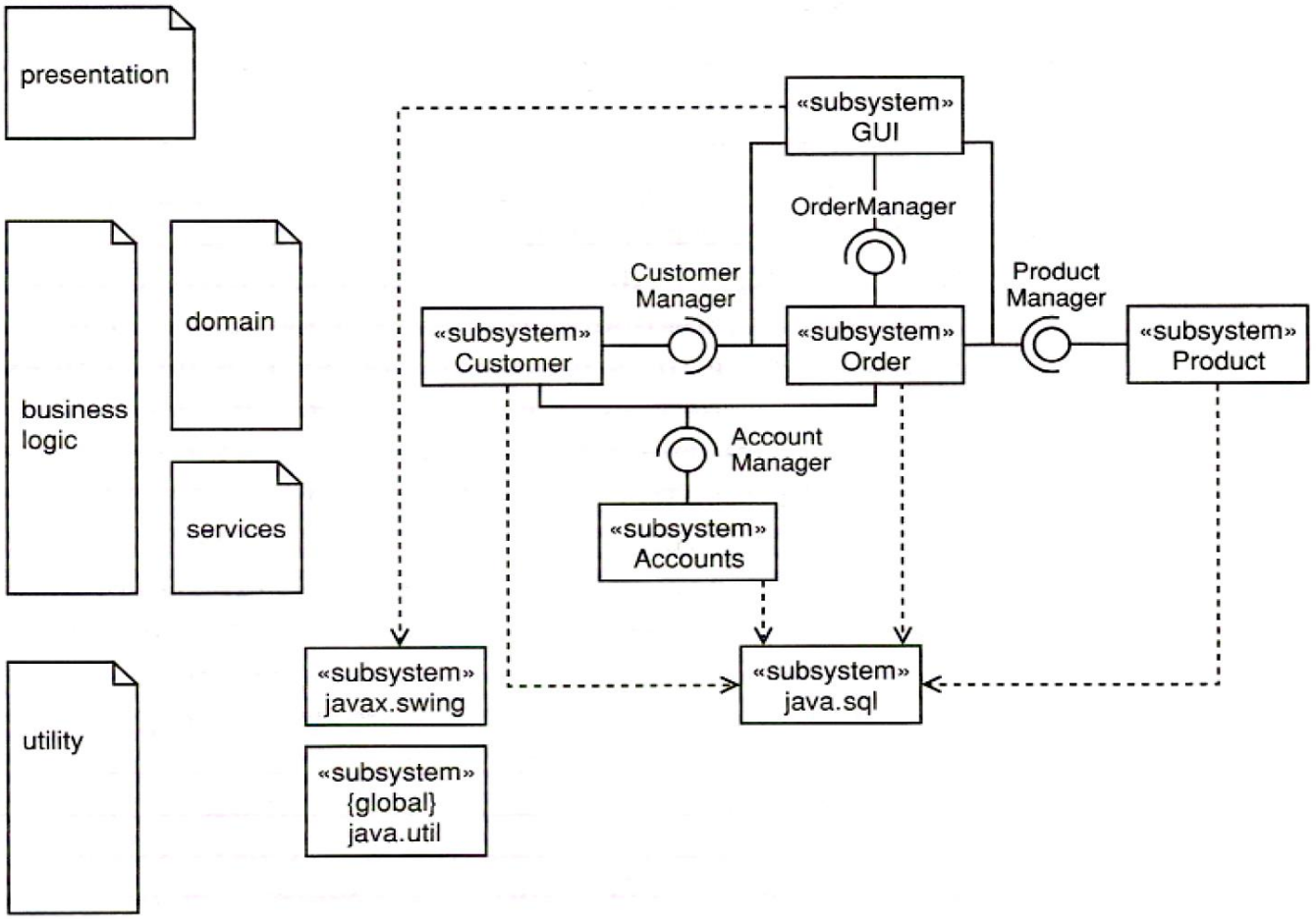
- Subsystems are used to:
  - separate design concerns;
  - represent large-grained components;
  - wrap legacy systems.



Designing with Interfaces

- Use interfaces to hide the implementation details of subsystems:
  - the *Facade* pattern hides a complex subsystem behind a simple interface provided by a wrapper object;
  - the *layering* pattern organizes subsystems into semantically cohesive layers:
    - a layer can only interact with its adjacent layers;
    - all dependencies between layers should be mediated by an interface;
    - example layers include presentation, business logic, and utility layers.

Designing with Interfaces: *Layering* Pattern



Reference

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