CSCI 5828: Foundations of Software Engineering

Lecture 22: OO Design

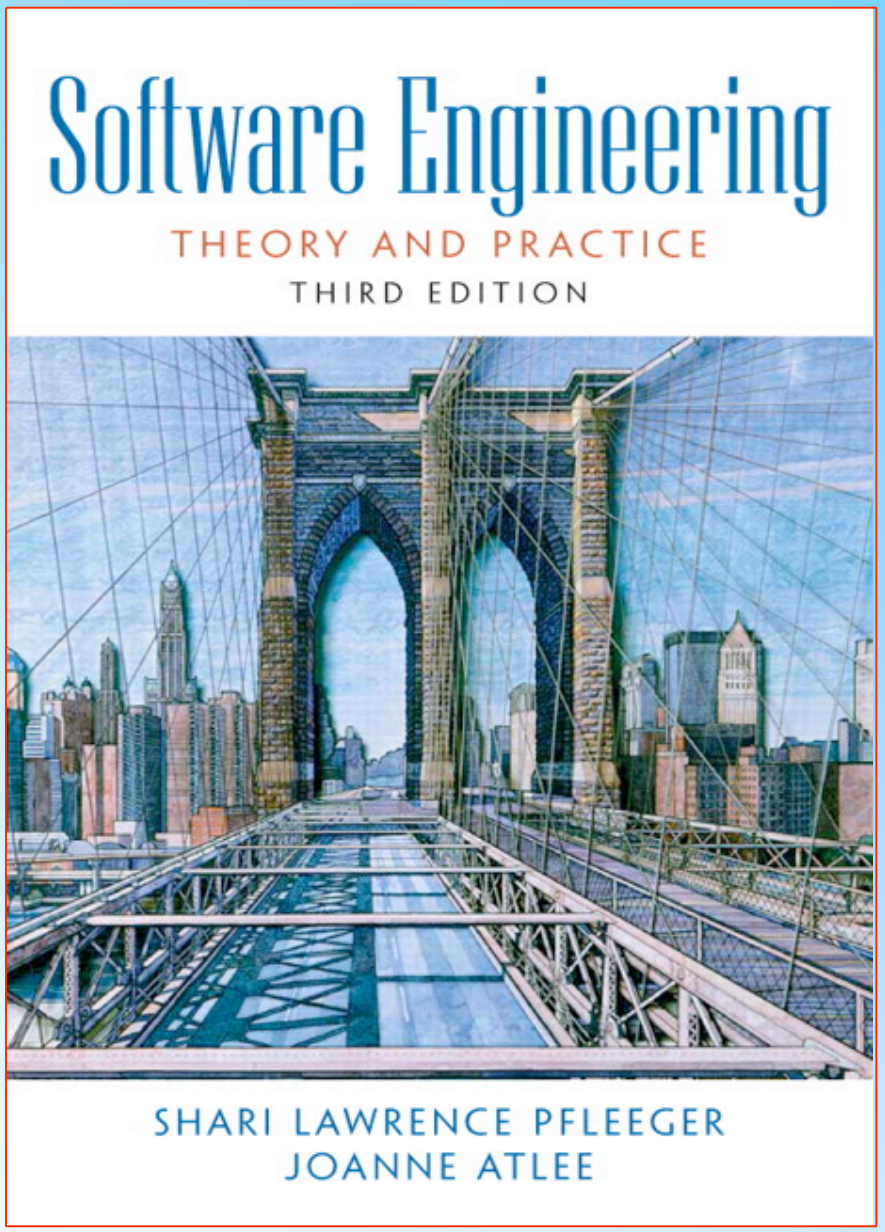
Slides created by Pfleeger and Atlee for the SE textbook

Some modifications to the original slides have been made by Ken Anderson for clarity of presentation

04/03/2008

Chapter 6

Considering Objects



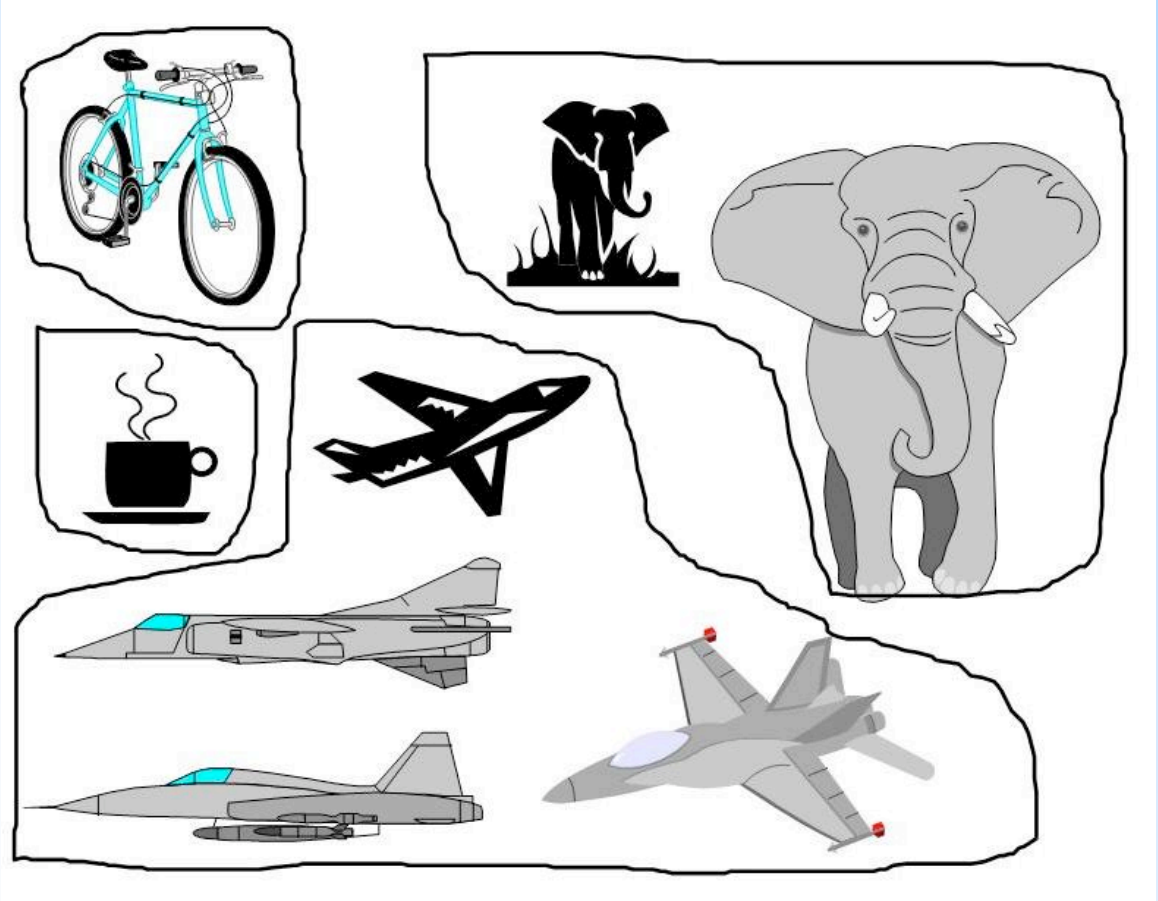
ISBN 0-13-146913-4
Prentice-Hall, 2006

6.1 What is OO?

- **Object-orientation** is an approach to software development that organizes both the problem and its solution as a collection of discrete objects
  - Each object has data (attributes) and behavior (methods) given to it by its class
  - Classes can be related to one another in various ways
    - use-relationships known as associations
    - whole-part relationships (aka aggregation/composition)
    - is-a relationships (aka inheritance)

6.1 What is OO?
Objects and Classes (continued)

- Examples of objects grouped into classes



6.1 What is OO?
Object-Orientation Characteristics

- Identity
  - Each object has a unique identity; often tied to the application domain
- Abstraction
  - Each class provides a public API that defines the services and data it provides
- Classification
  - Classes allow objects to be grouped into categories
- Encapsulation
  - Classes have private data and methods that can't be accessed externally
- Inheritance
  - Classes can have is-a relationships (more later)
- Polymorphism
  - We can deal with objects as instances of a base class but they will provide behavior that matches their true (sub) class
- Persistence: we can save/load objects just as we can save/load data

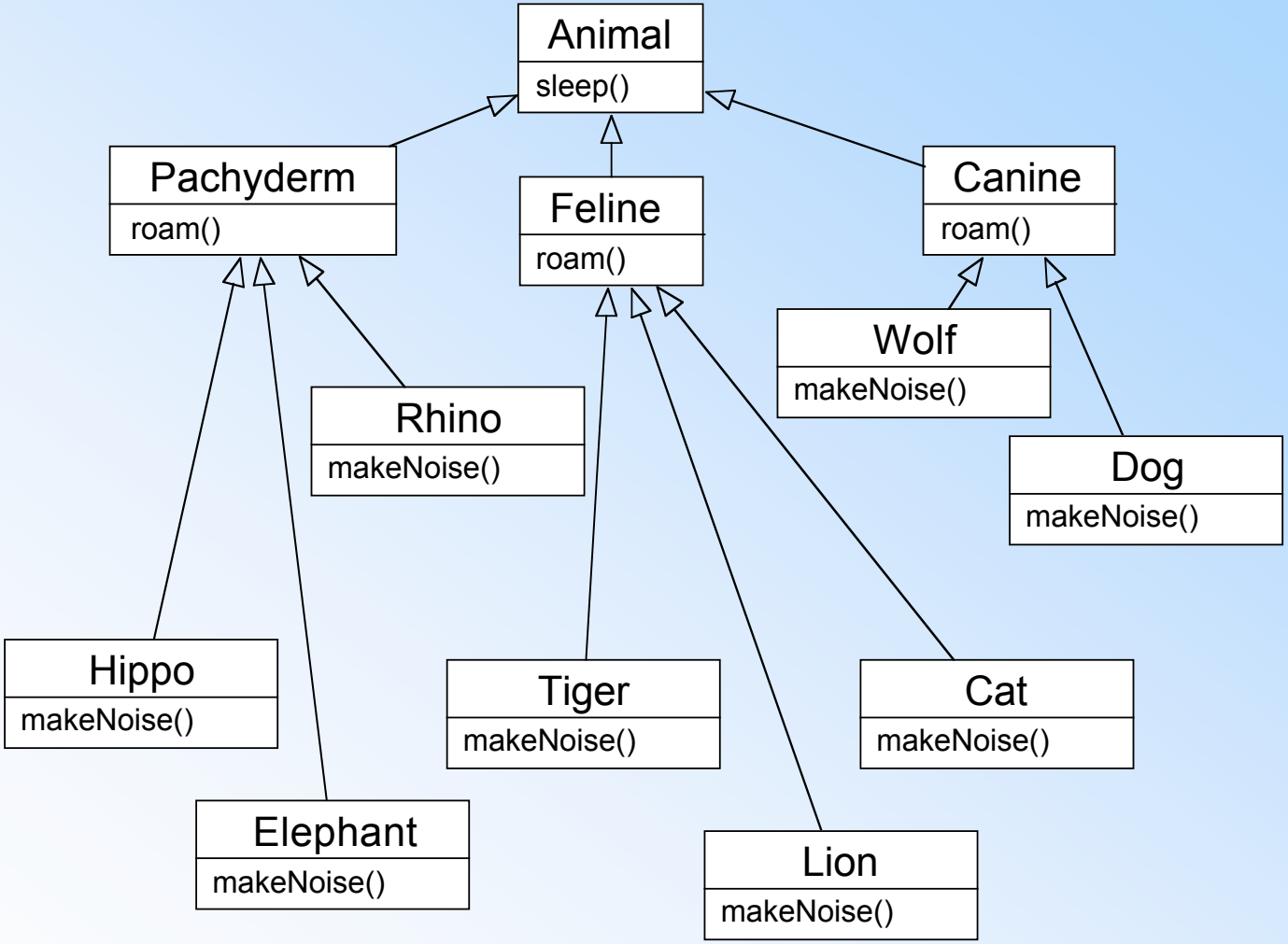
6.1 What is OO?
Objects and Classes (continued)

- We can represent a class using a box
- Box represents
  - object's name
  - attributes
  - behaviors

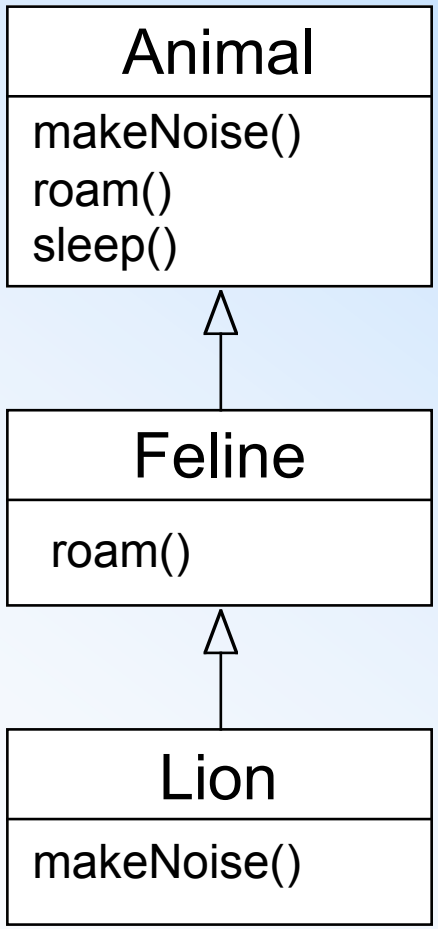
6.1 What is OO?
Classes Hierarchy

- A class hierarchy is organized according to the sameness or differences among classes
  - Exhibits OO classes' inheritance structure
- A class is refined into subclasses
- Subclasses inherit the structure (atts) as well as the behavior (methods) of its superclass

Inheritance Example



Polymorphism Example




```
Animal a = new Lion()
a.makeNoise();
a.roam();
a.sleep();
```

6.2 The OO Development Process

- One advantage of OO development is its consistency of terminology/concepts across various stages of the software life cycle
- Describing classes using OO representation requires three perspectives
  - *Static views*: descriptions of the object, attributes, behaviors, and relationships
  - *Dynamic views*: describe communication, control/timing, and the state and changes in state
  - *Restrictions*: describe constraints on the structure

6.2 The OO Development Process
OO Requirements

- Done in user's language and discusses the concepts and scenarios likely in the application domain
- Concepts include services and responsibilities
- Domain knowledge enables the developers
  - to understand the context of use for the system
  - to describe requirements in a way that users understand

6.2 The OO Development Process
OO Design

- Starts with OO requirements representation
  - Use cases, class diagrams that document domain knowledge
- System design identifies overall system architecture
  - Architectural components can be decomposed into subsystems
  - Classes can be assigned to subsystems if needed
- Program design adds more detail to classes, including relationships, details of 3rd party class libraries to be used, and non-functional concerns

6.2 The OO Development Process
OO Coding and Testing

- Coding proceeds by translating the models to an OO programming language
- It is necessary to refine the hierarchical structures and make adjustments as the requirements grow and mature
- Testing involves the same activities that are performed with any software system
  - Unit testing
  - Integration testing
  - System testing

6.3 Use Cases

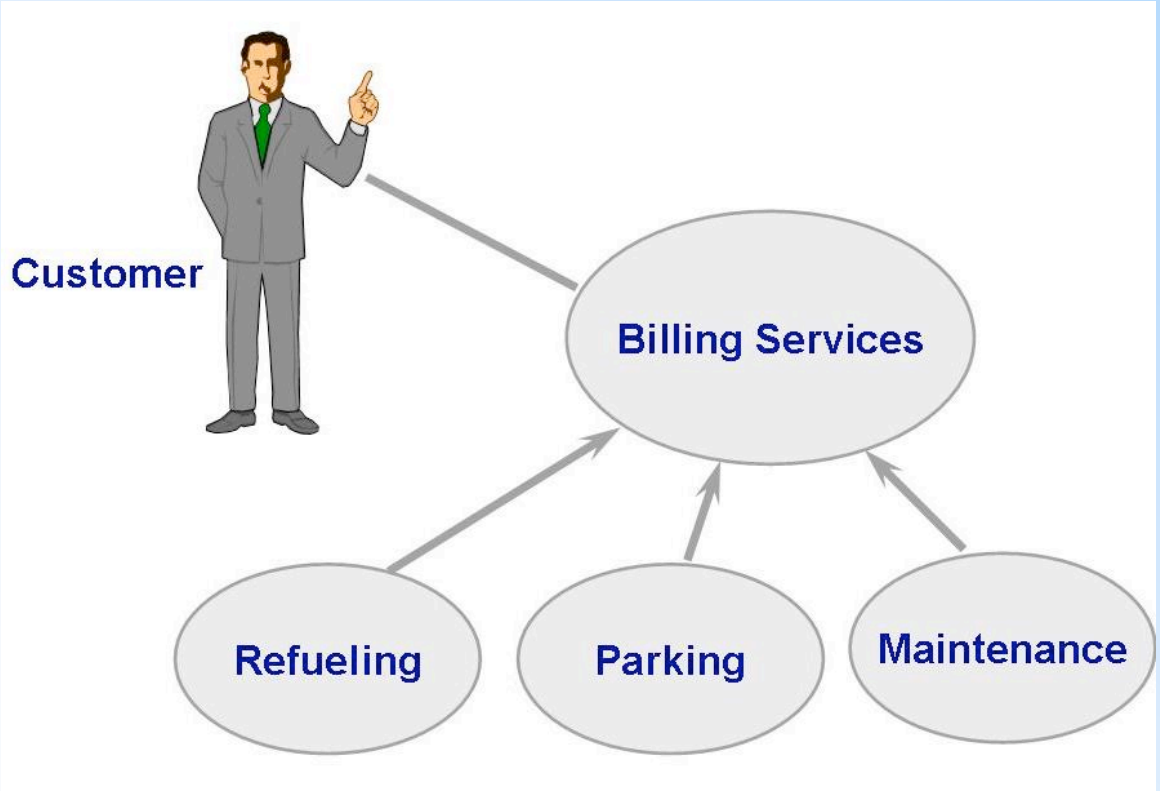
- Describes particular functionality that a system is supposed to perform or exhibit by modeling the dialog that a user, external system, or other entity will have with the system to be developed
- Diagrams have four elements
  - actors
  - cases
  - extensions
  - uses
- Note: use case diagrams are good for giving an overview but horrible at providing useful details

6.1 What is OO?
Sidebar 6.1 Royal Service Station Requirements

- Royal Service station provides three types of services
- The system must track bills, the product, and services
- System to control inventory
- The system to track credit history, and payments overdue
- The system applies only to regular repeat customers
- The system must handle the data requirements for interfacing with other systems
- The system must record tax and related information
- The station must be able to review tax record upon demand
- The system will send periodic message to customers
- Customers can rent parking space in the station parking lot
- The system maintains a repository of account information
- The station manager must be able to review accounting information upon demand
- The system can report an analysis of prices and discounts
- The system can not be unavailable for more than 24 hours
- The system must protect customer information from unauthorized access
- The system will automatically notify the owners of dormant accounts

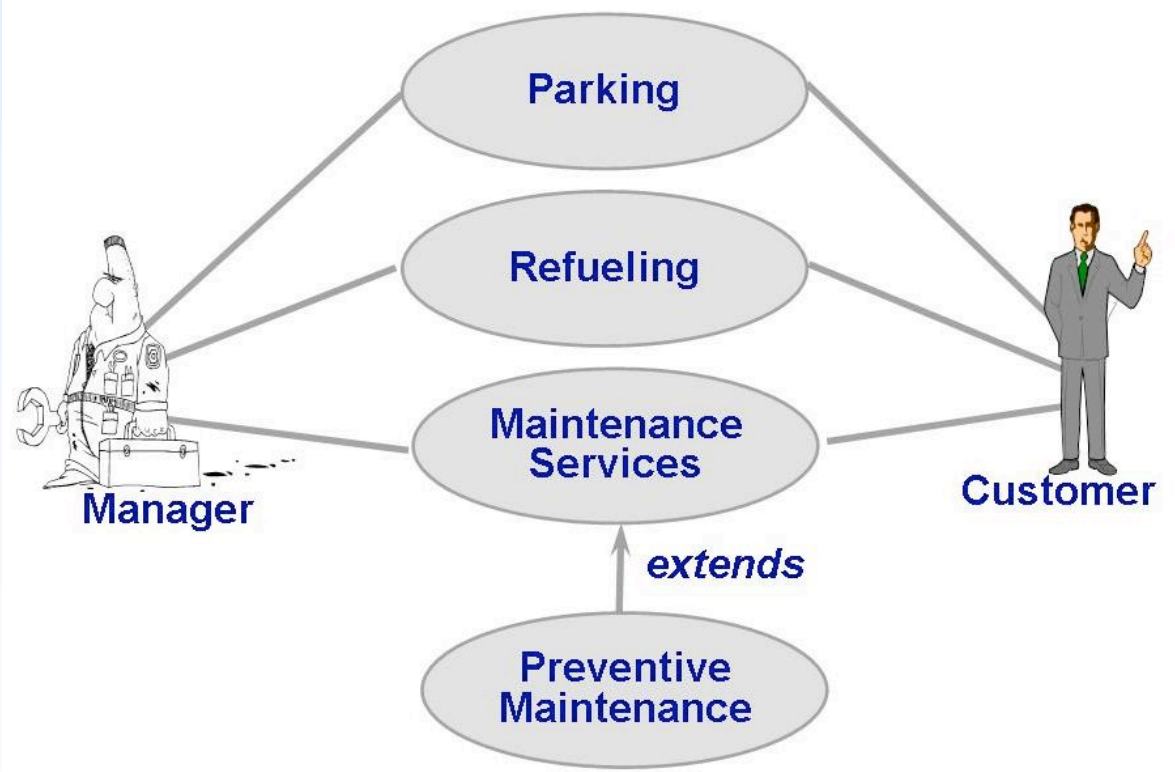
6.3 Use Cases
Example of Use Cases

- High-level view of Royal Service Station requirements



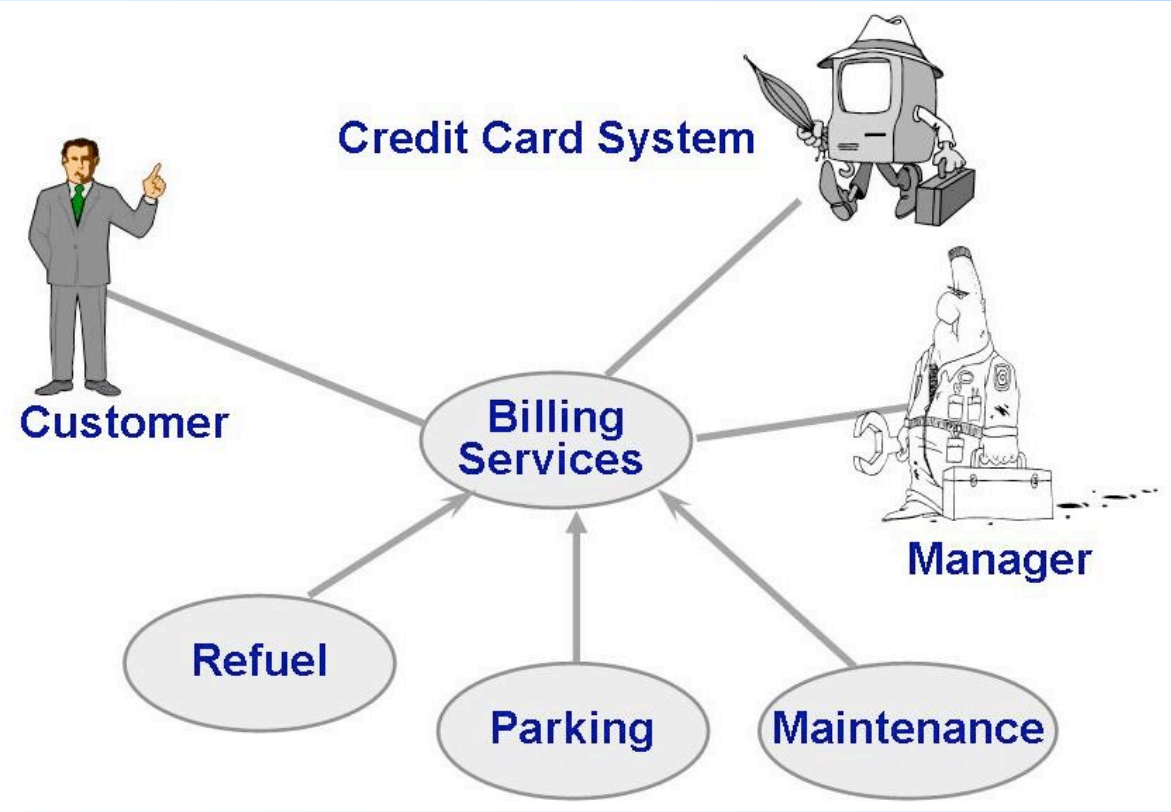
6.3 Use Cases
Example of Use Cases (continued)

- First extension part of the use case diagram of Royal Service Station requirements to include preventive maintenance



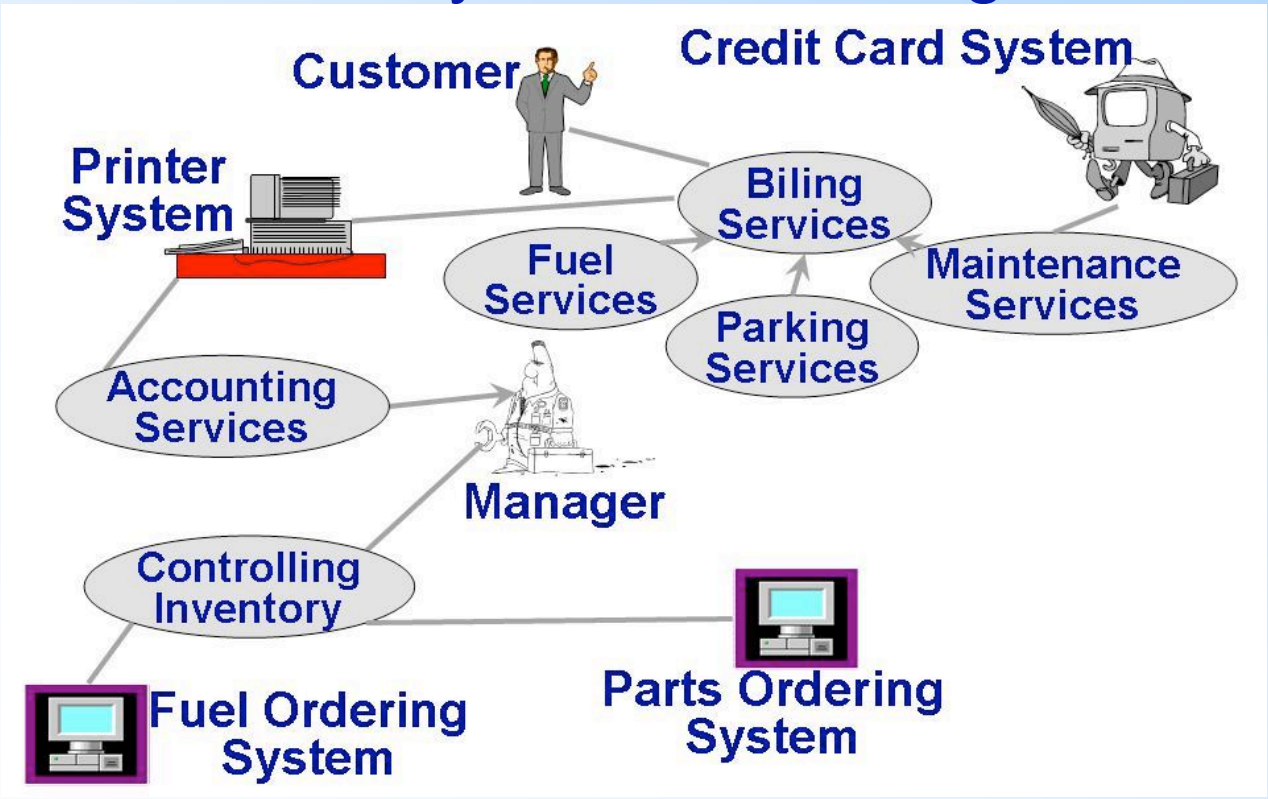
6.3 Use Cases
Example of Use Cases (continued)

- Second extension of Royal Service Station diagram to include credit card system



6.3 Use Cases
Example of Use Cases (continued)

- Third extension of Royal Service Station diagram to include inventory and accounting



6.3 Use Cases
Identifying Participants

- What stakeholders interact with the system
  - A stakeholder can be
    - a user
    - a system
  - Interactions can include
    - using the system to get information
    - using the system to complete a task
    - supplying the system with information
    - supplying a service to the system that it needs to perform its own tasks

Example Use Case

- Use cases have
  - actors
  - success path
  - failure cases
- Use cases may
  - vary in formality
  - invoke other use cases

Use Case 1 **Buy Stocks over the Web**

Primary Actor: Purchaser
Scope: Personal Advisors / Finance package (PAF)
Level: User goal
Stakeholders and Interests:
Purchaser—wants to buy stocks and get them added to the PAF portfolio automatically.
Stock agency—wants full purchase information.
Precondition: User already has PAF open.
Minimal Guarantee: Sufficient logging information will exist so that PAF can detect that something went wrong and ask the user to provide details.
Success Guarantee: Remote web site has acknowledged the purchase; the logs and the user's portfolio are updated.
Main Success Scenario:
1. Purchaser selects to buy stocks over the web.
2. PAF gets name of web site to use (E*Trade, Schwab, etc.) from user.
3. PAF opens web connection to the site, retaining control.
4. Purchaser browses and buys stock from the web site.
5. PAF intercepts responses from the web site and updates the purchaser's portfolio.
6. PAF shows the user the new portfolio standing.

Extensions:
2a. Purchaser wants a web site PAF does not support:
2a1. System gets new suggestion from purchaser, with option to cancel use case.
3a. Web failure of any sort during setup:
3a1. System reports failure to purchaser with advice, backs up to previous step.
3a2. Purchaser either backs out of this use case or tries again.
4a. Computer crashes or is switched off during purchase transaction:
4a1. (What do we do here?)
4b. Web site does not acknowledge purchase, but puts it on delay:
4b1. PAF logs the delay, sets a timer to ask the purchaser about the outcome.
5a. Web site does not return the needed information from the purchase:
5a1. PAF logs the lack of information, has the purchaser update questioned purchase.

Example Use Case (Low Formality)

Use Case 4 **Buy Something (Casual Version)**

The Requestor initiates a request and sends it to her or his Approver. The Approver checks that there is money in the budget, checks the price of the goods, completes the request for submission, and sends it to the Buyer. The Buyer checks the contents of storage, finding the best vendor for goods. The Authorizer validates Approver's signature. The Buyer completes request for ordering, initiates PO with Vendor. The Vendor delivers goods to Receiving, gets receipt for delivery (out of scope of system under design). The Receiver registers delivery, sends goods to Requestor. The Requestor marks request delivered.

At any time prior to receiving goods, the Requestor can change or cancel the request. Canceling it removes it from any active processing (deletes it from system?). Reducing the price leaves it intact in processing. Raising the price sends it back to the Approver.

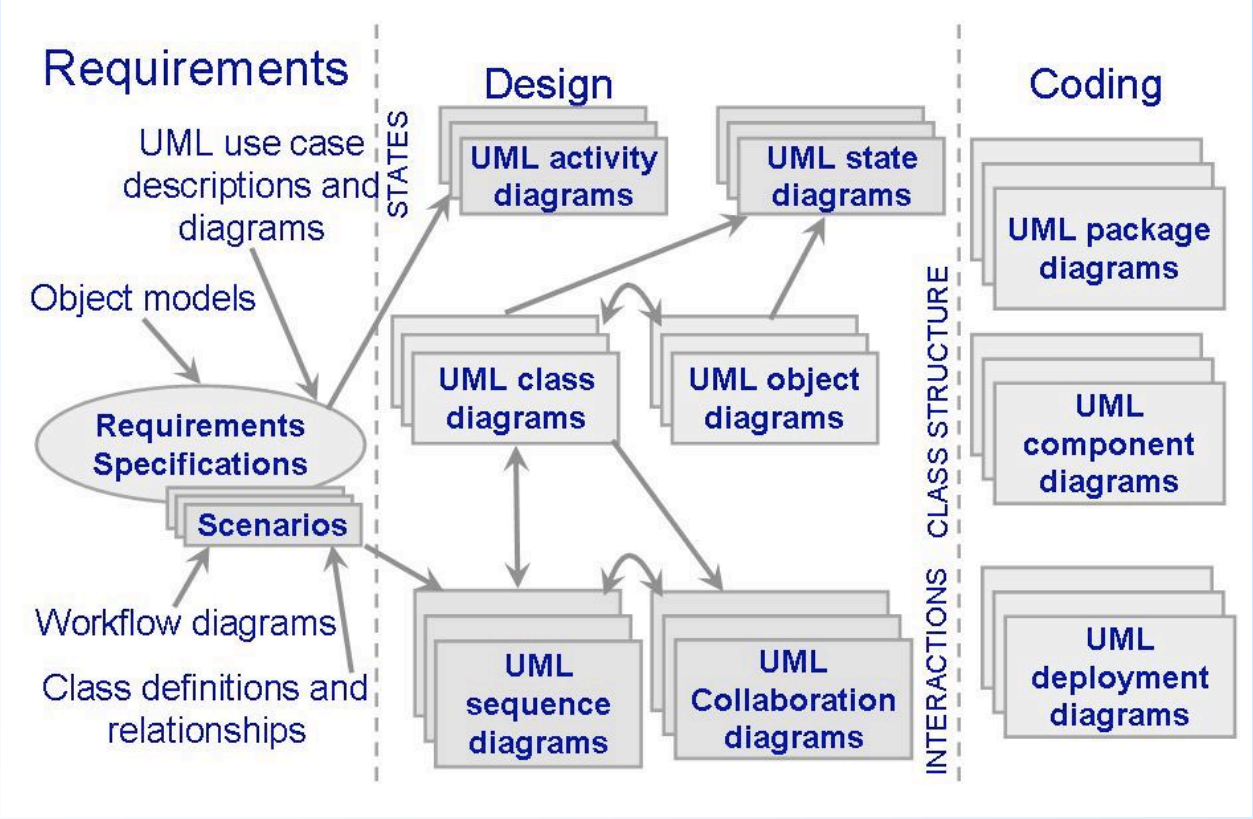
- Note: Examples come from Alistair Cockburn's excellent book: "Writing Effective Use Cases" Copyright 2001. Addison-Wesley

6.4 Representing OO: An Example Using UML
UML in the OO Process

- The Unified Modeling Language (UML) provides multiple notations for representing information in a software life cycle
  - Activity diagrams
  - Class diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Package diagrams
  - Component diagrams
  - Deployment diagrams
- We will cover only a few

6.4 Representing OO: An Example Using UML
UML in the OO Process (continued)

- How UML can be used in life cycle (note: component/deployment diagrams in wrong place!)

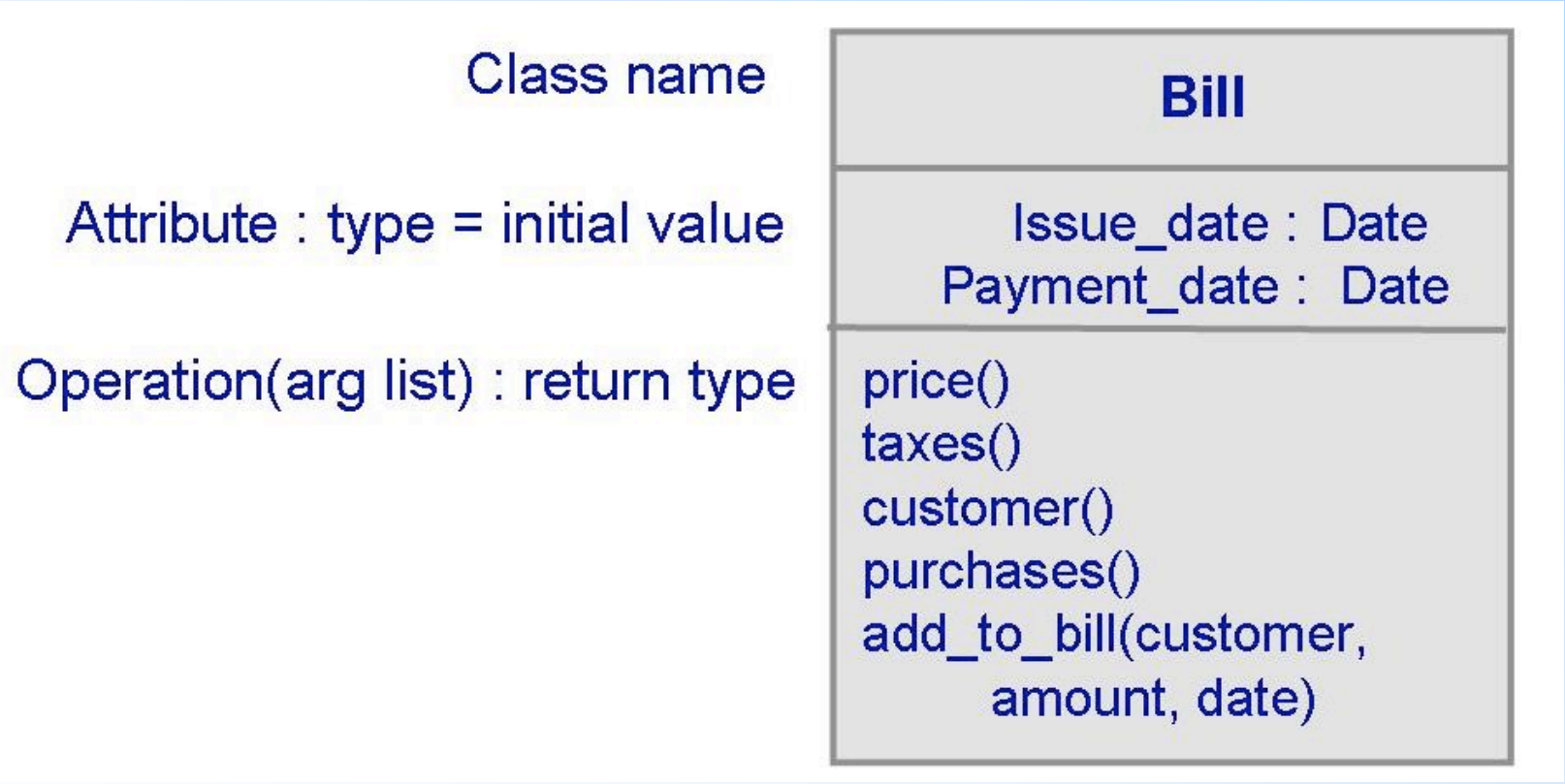


6.5 OO System Design
How do you find classes

- They will often be obvious as you learn about the application domain
  - What are the important concepts that need to be tracked by the system?
    - What are their attribues?
    - What services do they provide?
- One (slightly useful) trick
  - Look for nouns (objects) and verbs (methods) in the documents provided by your users
  - This gives you a candidate set of objects that can be evolved into classes over time

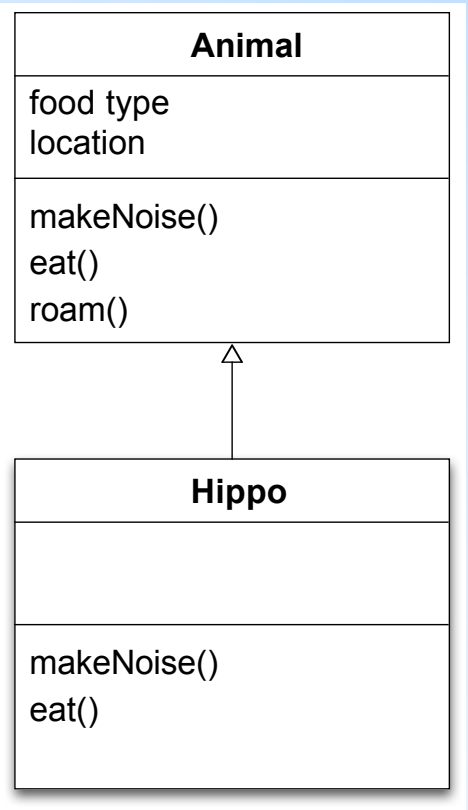
6.5 OO System Design
UML Diagram

- A UML box used to illustrate the component of a class



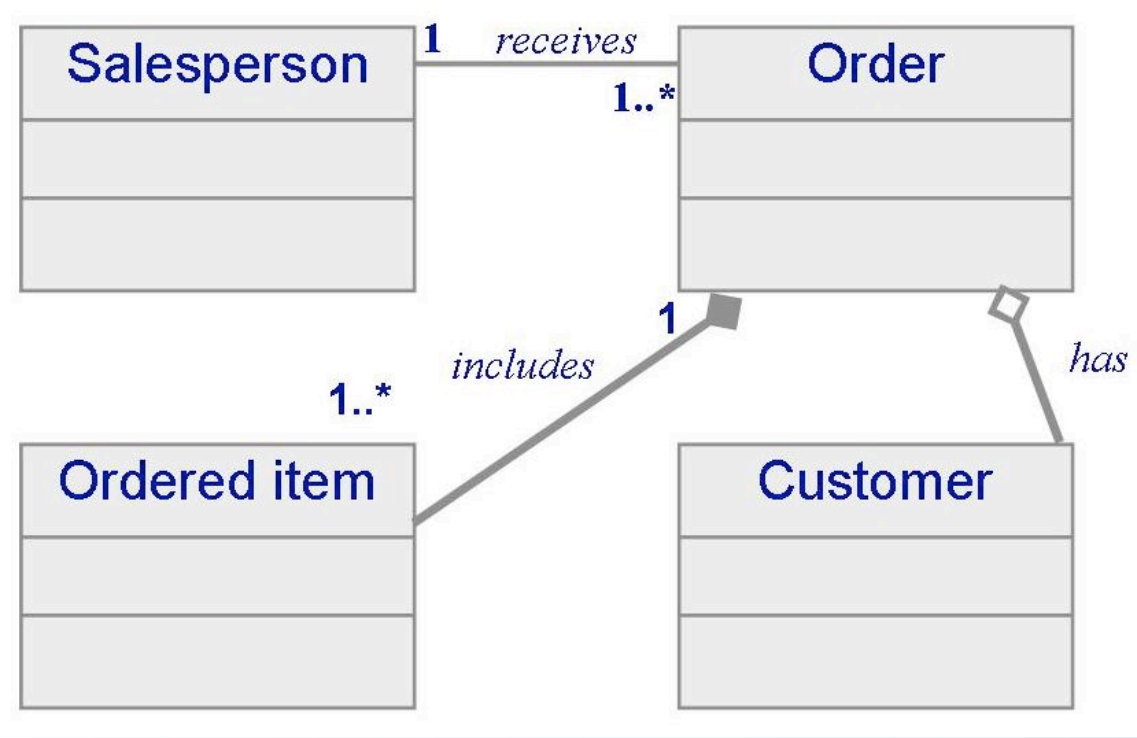
6.5 OO System Design
UML Diagram to Describe Relationship

- Inheritance relationship (*is-a* relationship): lower box inherits the attributes and behaviors of the upper box



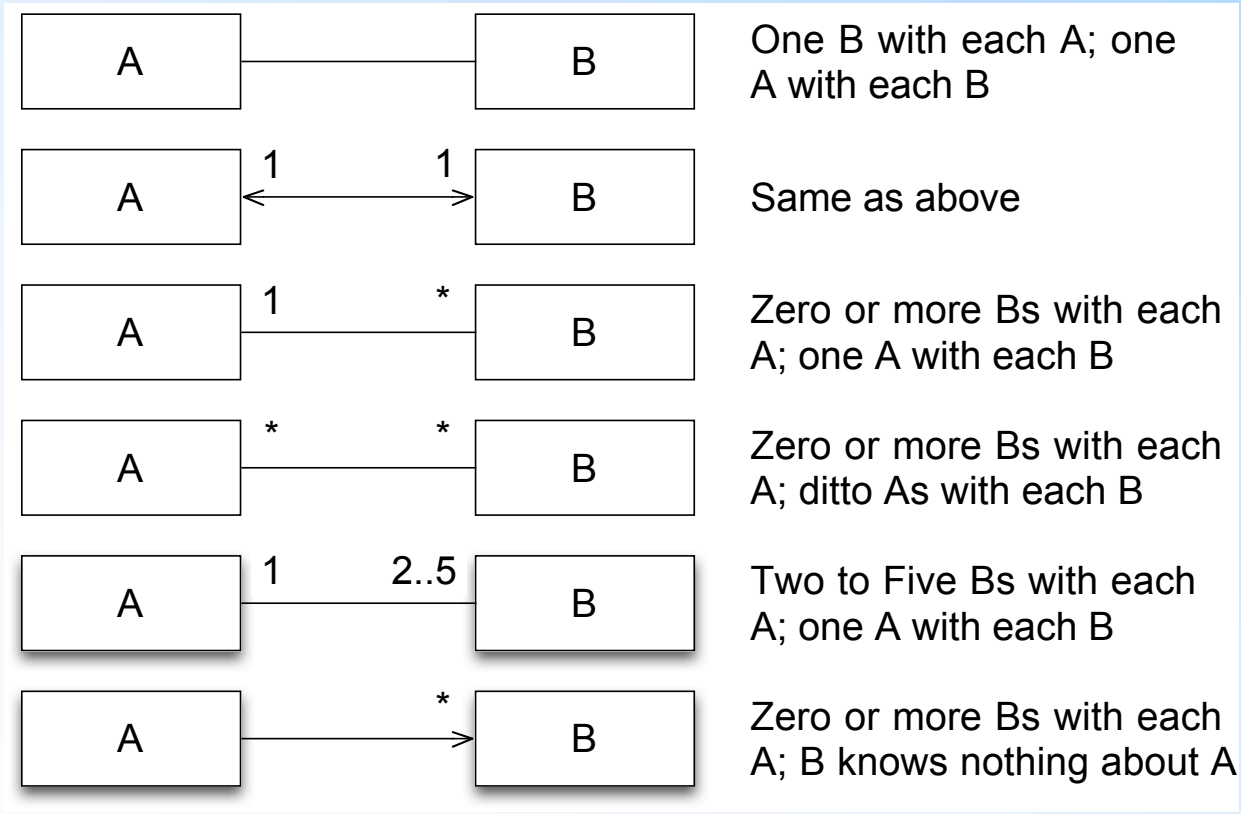
6.5 OO System Design
UML Diagram to Describe Relationship (continued)

- Each order is associated with salesperson (**associated**)
- The order item is part of the order (**composition**); the order also has a customer (aggregation)

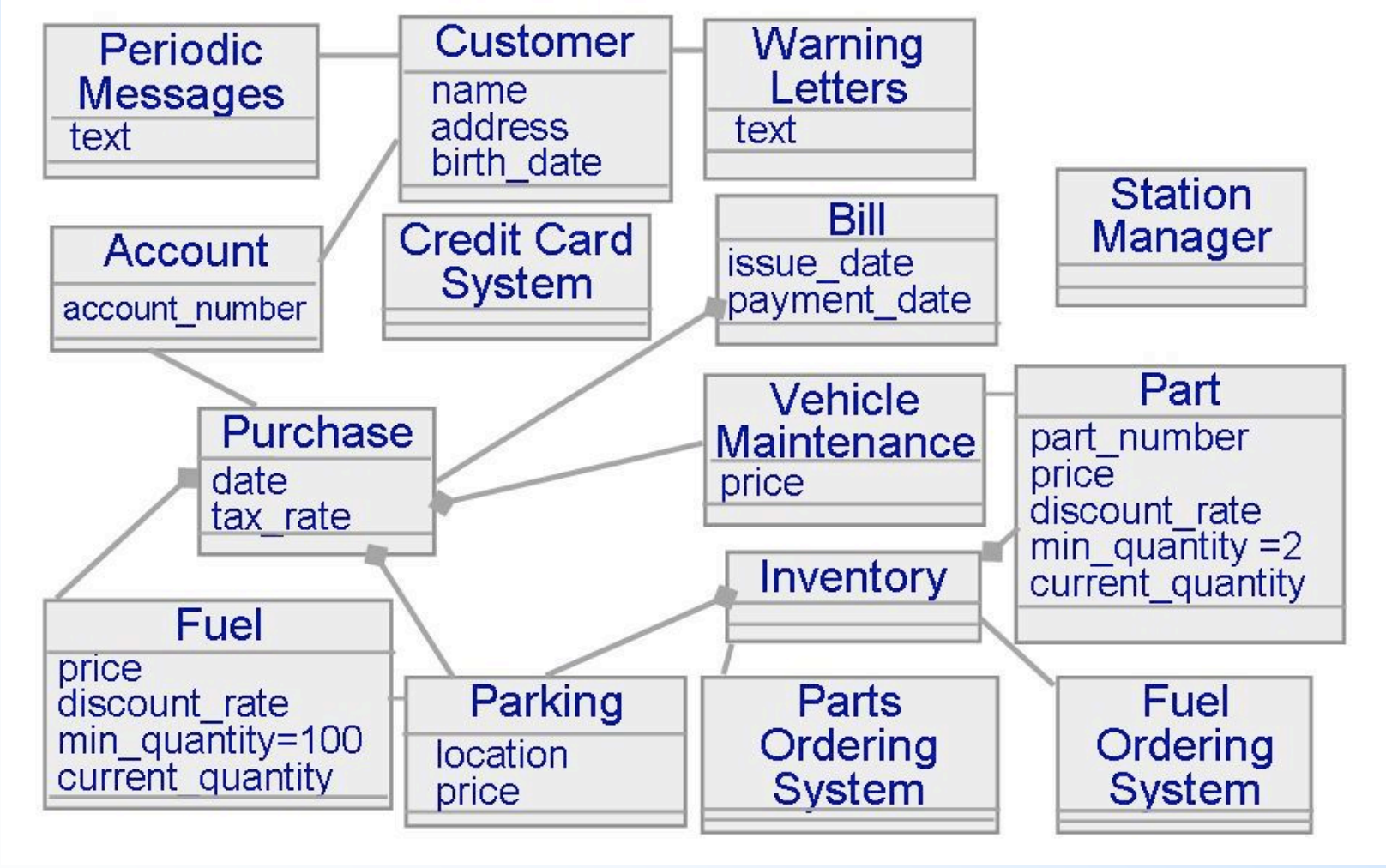


6.5 OO System Design
UML Diagram to Describe Relationship (continued)

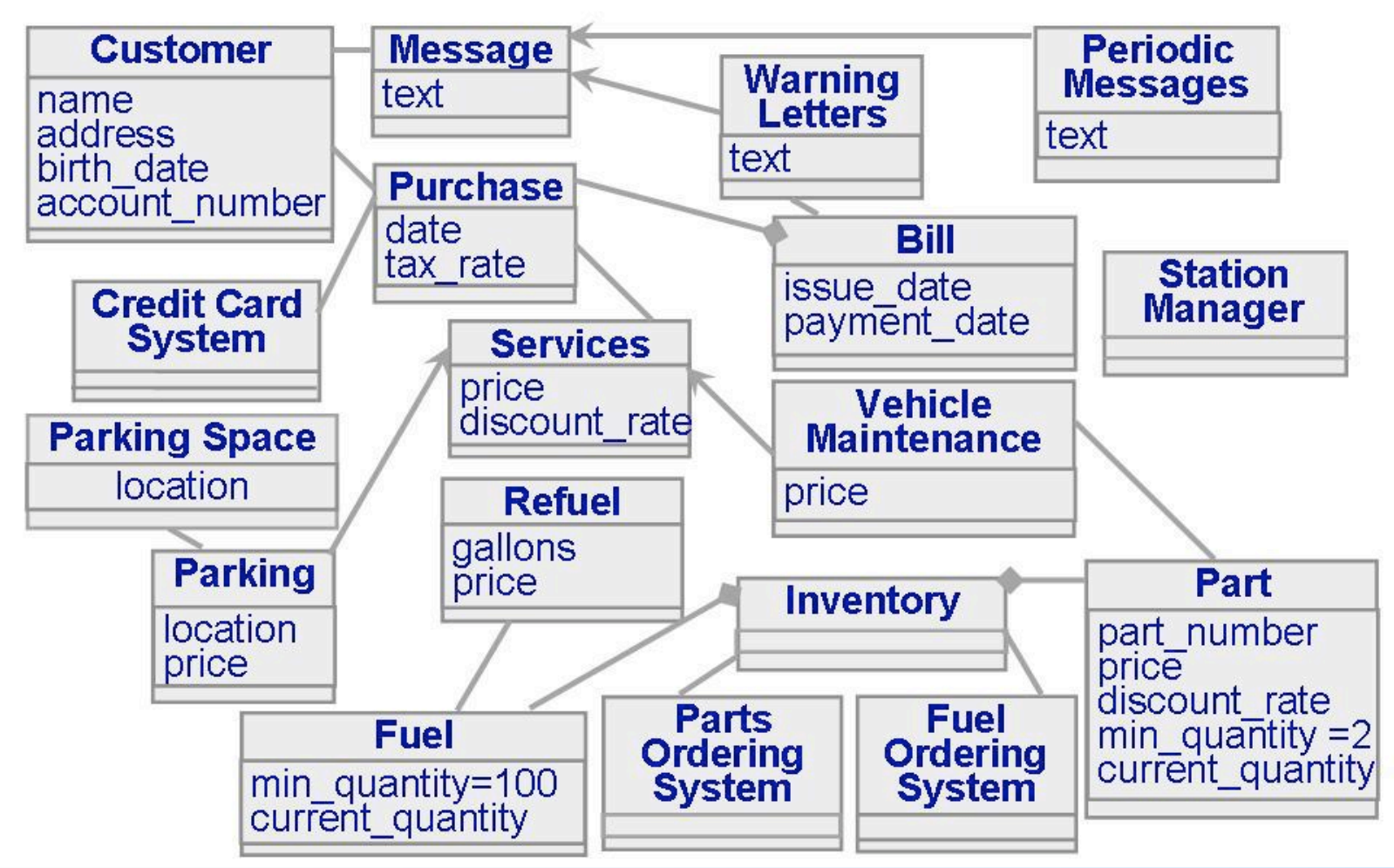
- Graphical representation of several other ways of denoting relationships between classes



6.5 OO System Design
First Cut at Royal Service Station Design

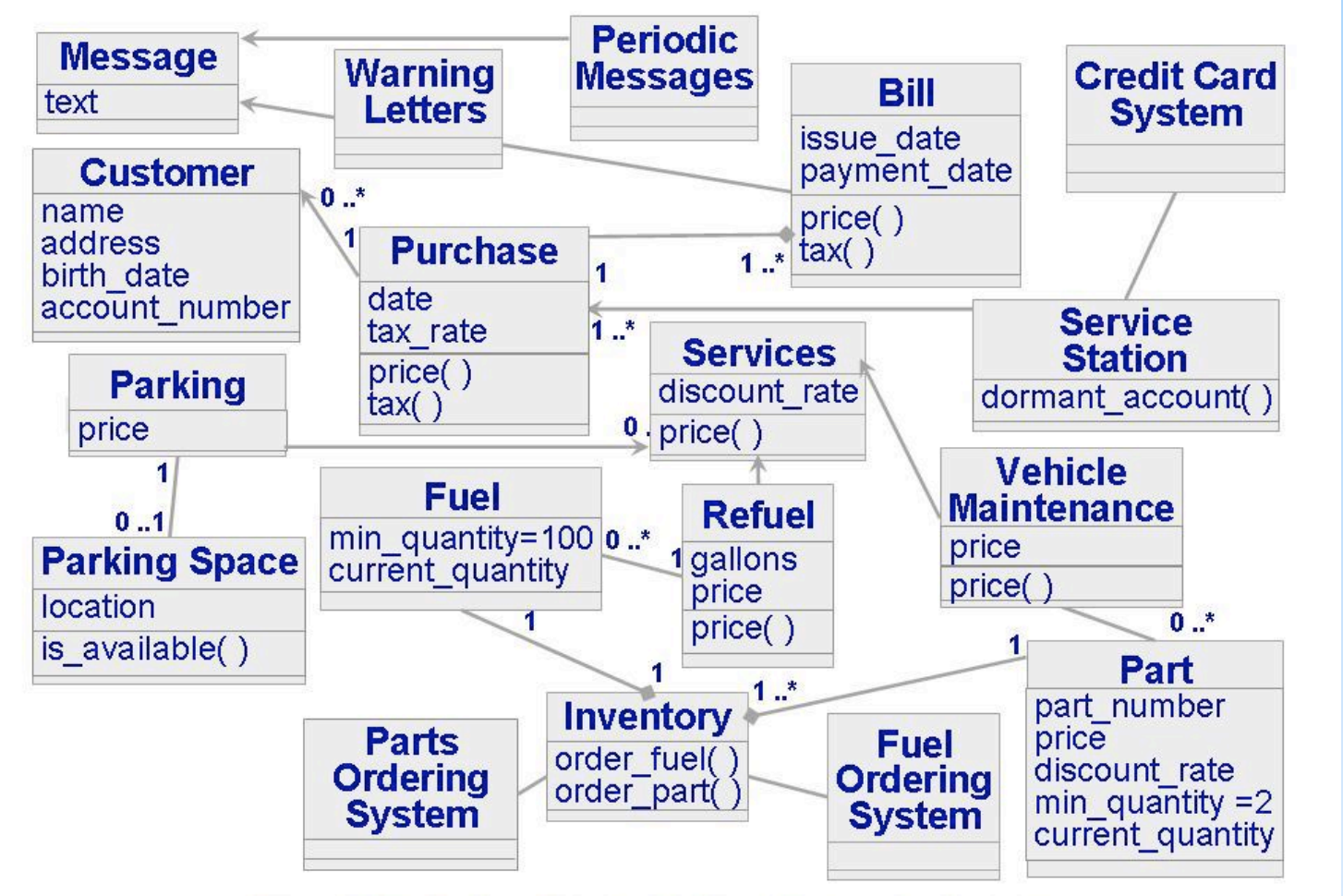


6.5 OO System Design
Second Cut at Royal Service Station Design



6.5 OO System Design

Third and Final Cut at Royal Service Station Design

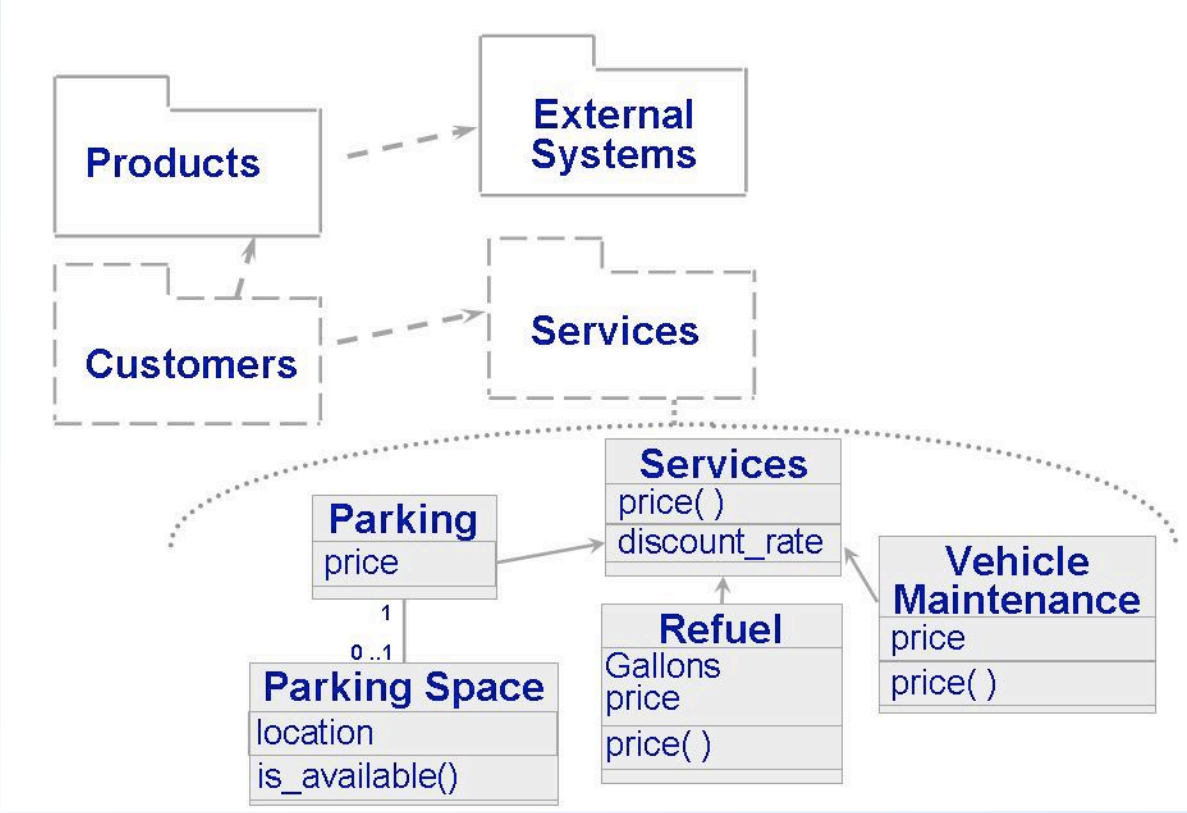


6.5 OO System Design
Package Diagram

- View the system as a small collection of packages, which can be expanded to a larger set of classes
  - Show dependencies among classes that belong to different packages
    - Two items are dependent if changes to the definition of one may cause changes to the other

6.5 OO System Design
Package Diagram for the Royal Service Station

- There are four major packages
- The service package consists of five key classes

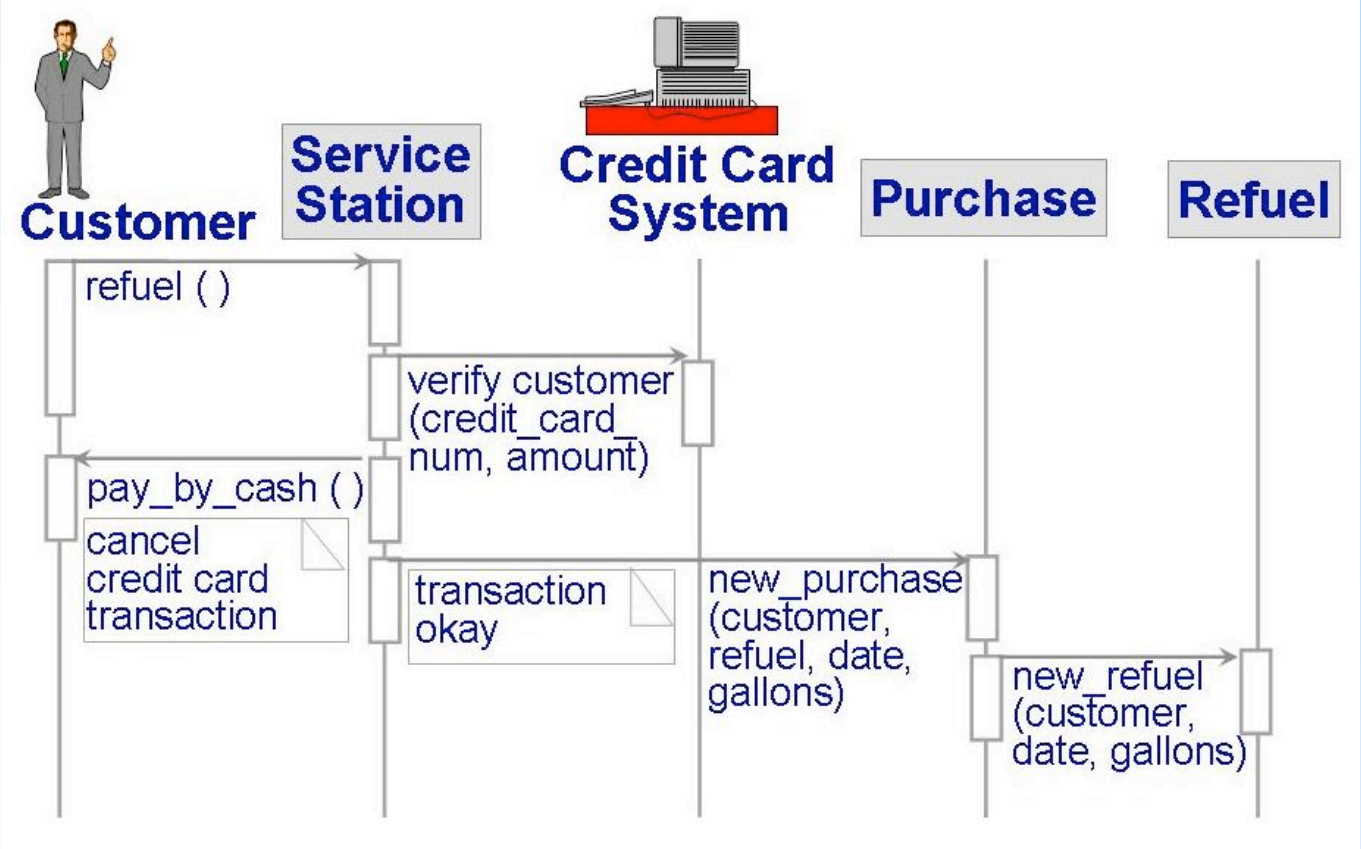


6.5 OO System Design
Interaction Diagram

- Describe how operations and behaviors are handled by the objects in the design
- Two kinds of interaction diagrams
  - *Sequence diagram*: shows the sequence in which activities or behaviors occur
  - *Collaboration diagram*: shows how the objects are connected statically

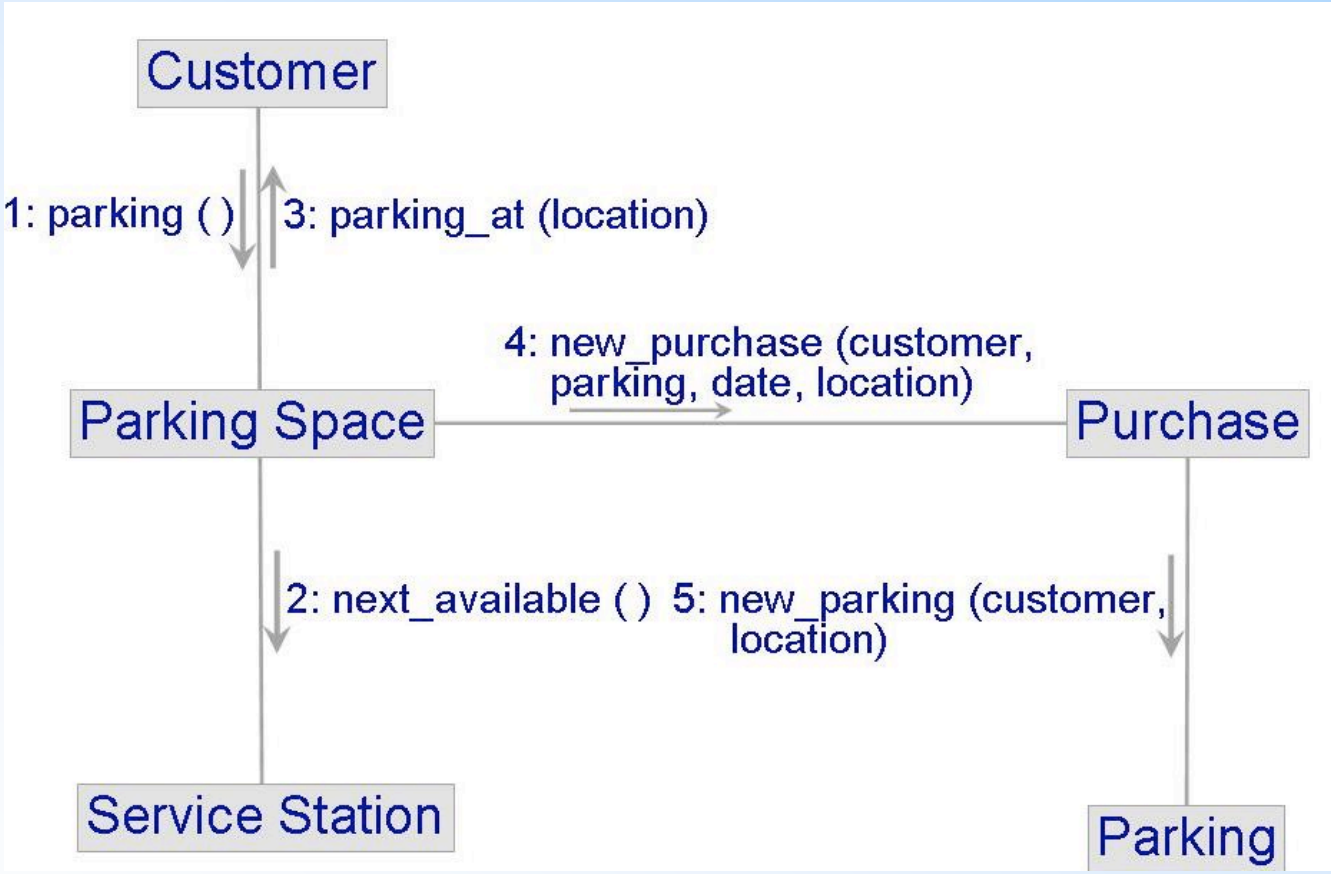
6.5 OO System Design
Sequence Diagram for the Royal Service Station

- Showing the use case of the *refuel* class



6.5 OO System Design
Collaboration Diagram for the Royal Service Station

- Shows the *parking* use case

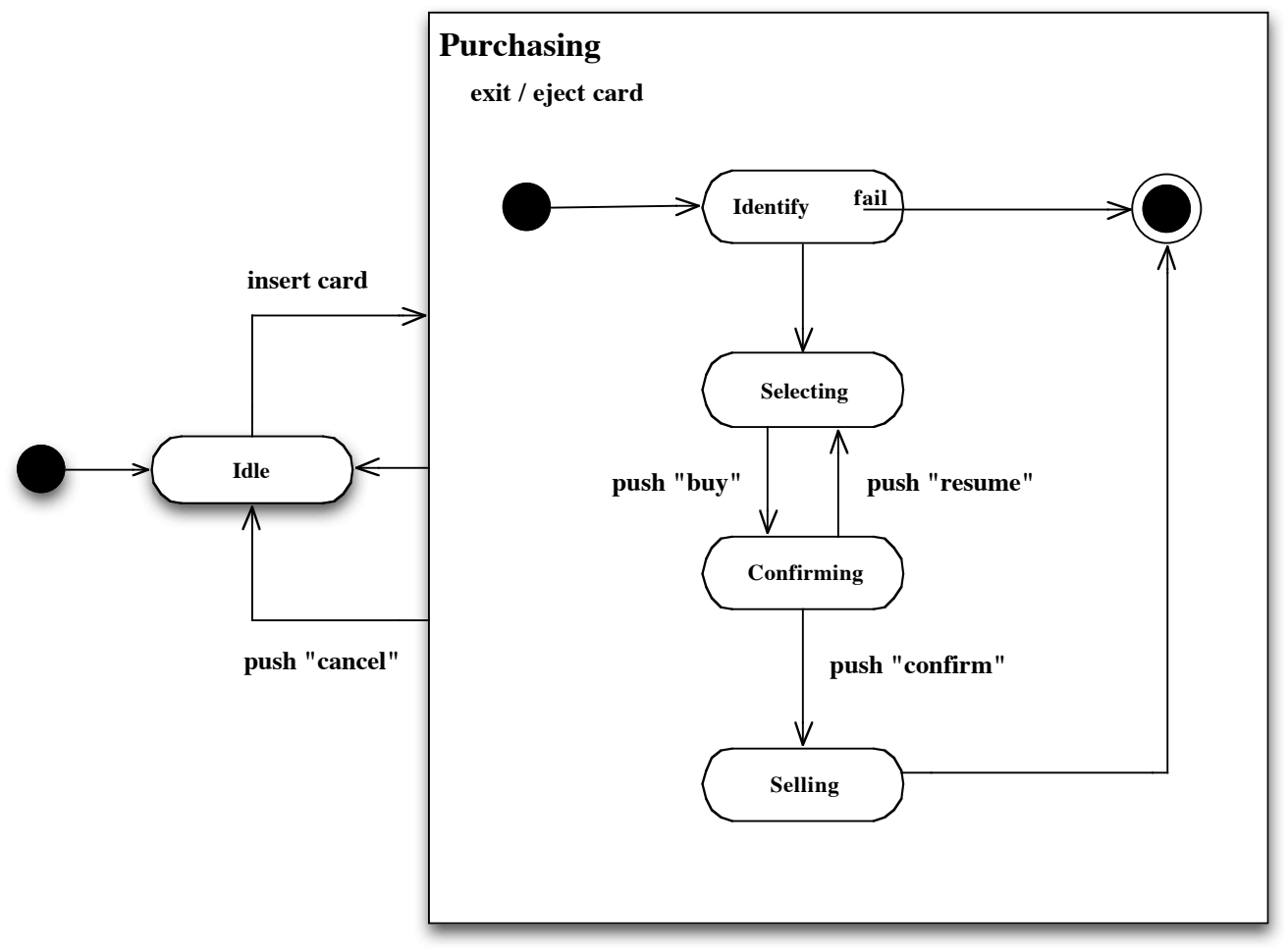


6.5 OO System Design
State Diagram

- To document a dynamic model of the system
- Shows
  - the possible states an object can take
  - the events that trigger the transition from one state to the next
  - the actions that result from each state change
- Needed only for classes where the objects exhibit dynamic behavior, with many attribute values and messages
- Note: state diagrams in textbook are horrible! (They don't show labels, use incorrect notation, and don't make sense!) Fortunately, we've seen examples of good state diagrams in the concurrency textbook

State Diagram Example: Whole System Granularity

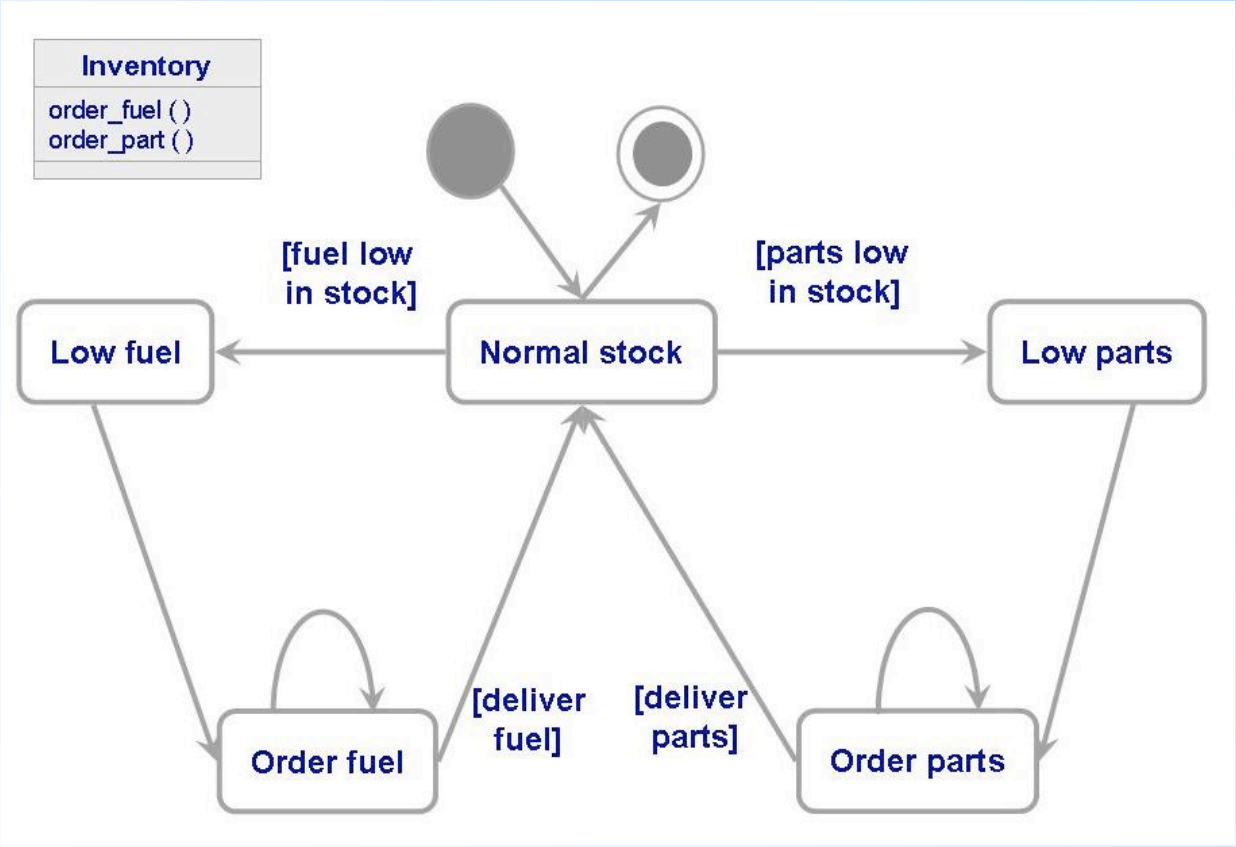
- Example from UML Reference Manual by "Three Amigos". © 1999 Addison Wesley



6.5 OO System Design

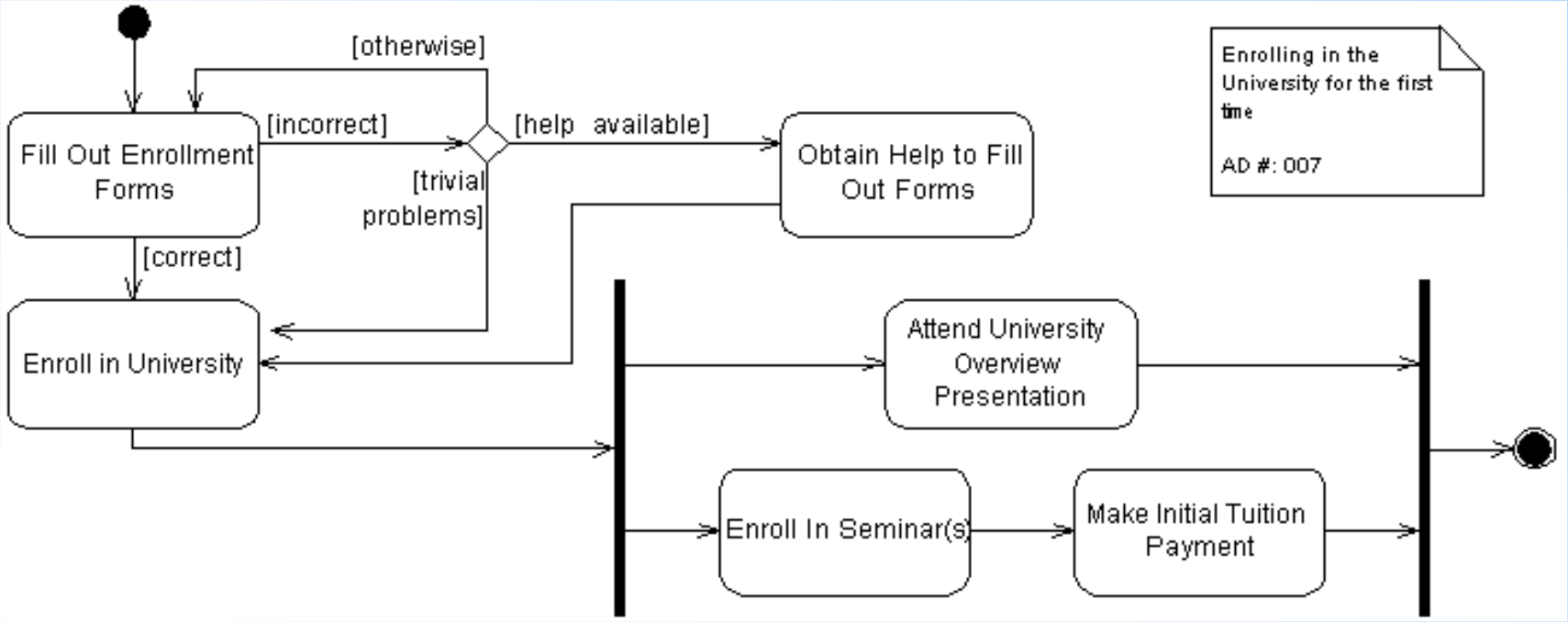
State Diagram for the Royal Service Station System

- State diagram for the *inventory* class, orders parts or fuel: a different condition triggers each state



6.5 OO System Design
Activity Diagram

- To model the flow of procedures or activities in a class
- A decision node is used to represent a choice of which activity to invoke



OO Design: How To

- Basic steps (note: different from text book)
  - Step 1: Analyze/Create use cases
  - Step 2: Create activity diagrams for each use case
  - Step 3: Create class diagram based on 1
  - Step 4: Create interaction diagrams for activities contained in diagrams from step 2
  - Step 5: Create state diagrams for classes created in step 3
  - Step 6: Iterate; each step above will reveal information about the other models that will need to be updated
    - for instance, methods specified on objects in a sequence diagram, have to be added to the class diagram
    - classes in the class diagram, should appear in at least one sequence diagram

OO Design: How To (as a picture)