

软件工程概论

Software Engineering

刘伟

liuwei@xidian.edu.cn

88204608

CH6. Concerning Objects

Content

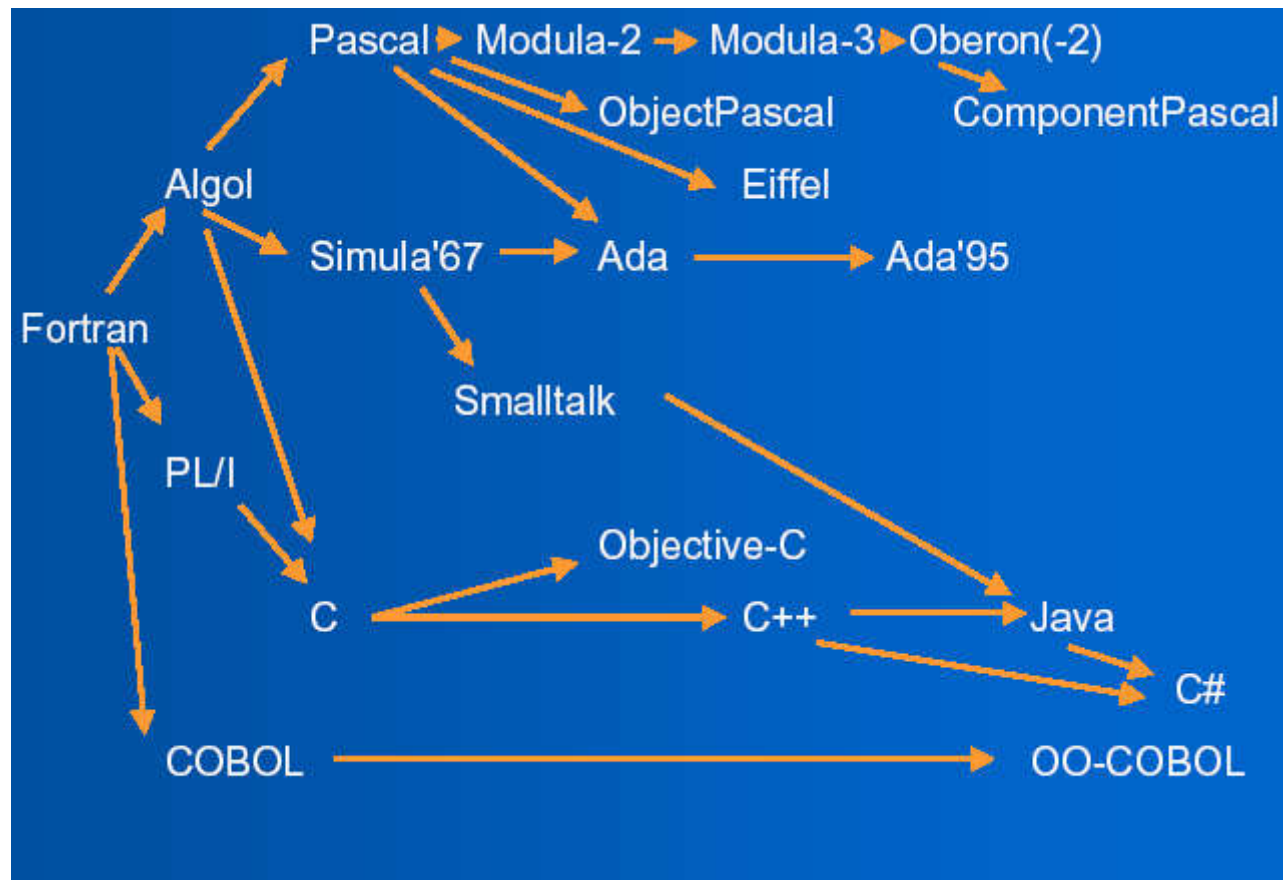
- ❑ What is OO?
- ❑ The OO Development Process
- ❑ UML
- ❑ OOA
- ❑ OO System Design
- ❑ OO Program Design
- ❑ OO Measurement

6.1 What is OO?

	结构化方法	OO方法
概念	功能的集合，通过模块以及模块和模块之间的分层调用关系实现	事物的集合，通过对象以及对象和对象之间的通讯联系实现；
构成	结构化软件=过程+数据，以过程为中心；	面向对象软件=（数据+相应操作）的封装，以数据为中心；
运行控制	顺序处理方式，由过程驱动控制；	交互式、并行处理方式，由消息驱动控制；
开发	工作重点是设计	工作重点是分析
应用	数据类型比较简单的数值计算软件	大型复杂的人机交互式软件；

6.1 What is OO?

The History of OT

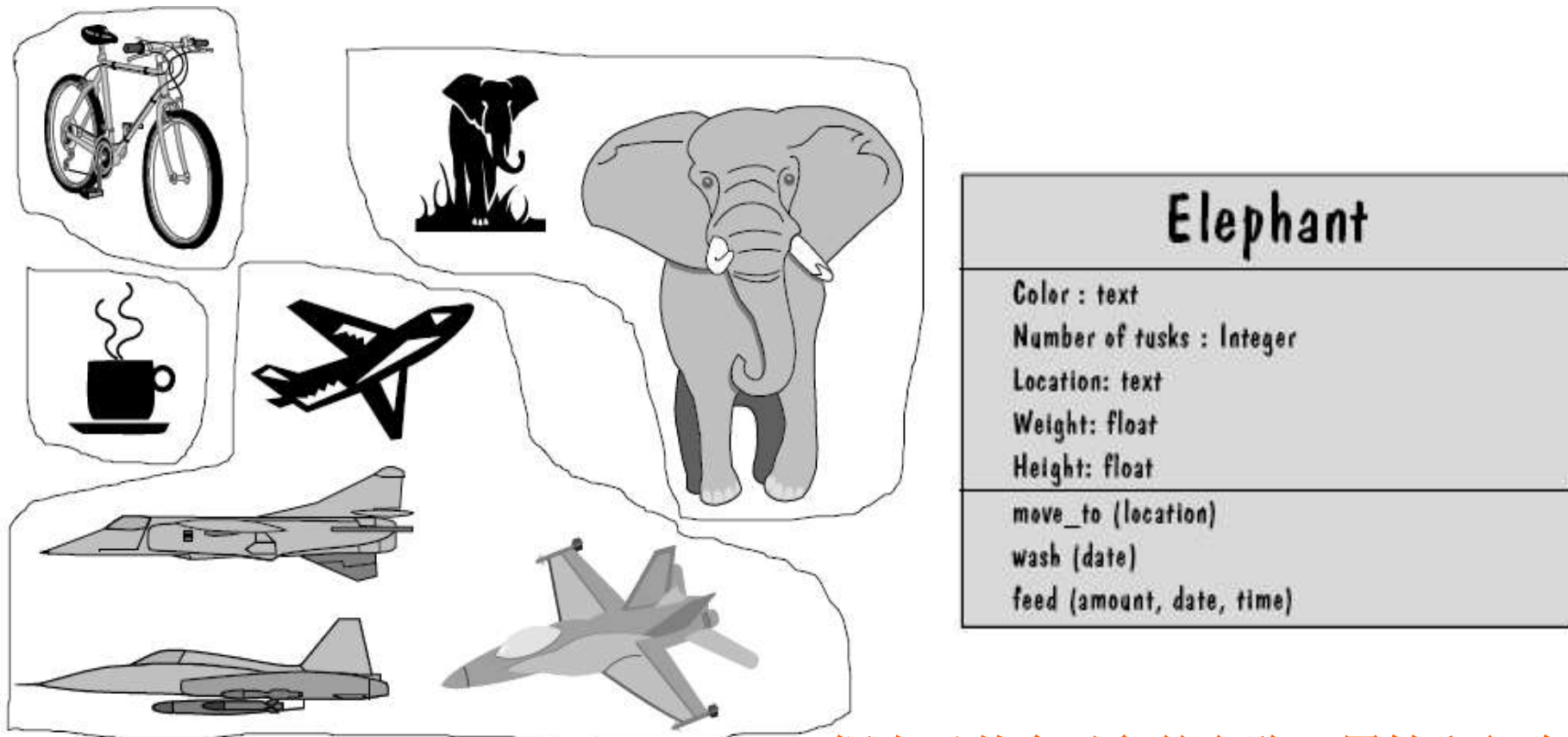


6.1 What is OO?

- Identity 一致性
- Abstraction 抽象
- Classification 分类
- Encapsulation 封装
- Inheritance 继承
- Polymorphism 多态性
- Persistence 连续性

6.1 What is OO?

Examples of objects grouped into classes



框表示某个对象的名称、属性和行为

6.1 What is OO?

Object的特点

- ❑ 以数据为中心;
- ❑ 对象是主动的;
- ❑ 属性和操作封装，信息是隐蔽的。
- ❑ 对象独立处理自身的数据，并通过消息传递进行通信，具有并行的性质
- ❑ 高模块独立性，对象是OO软件的基本模块，低耦合，高内聚。
- ❑ 对象具有唯一识别的功能，行为比较丰富
- ❑ 对象必须参与一个或多个对象类

6.1 What is OO?

Class类

□ 定义

- 具有相同结构、操作，并遵守相同约束规则的对象聚合成一组，这组对象集合称为对象类，简称类。

□ 类层次（Hierarchy）

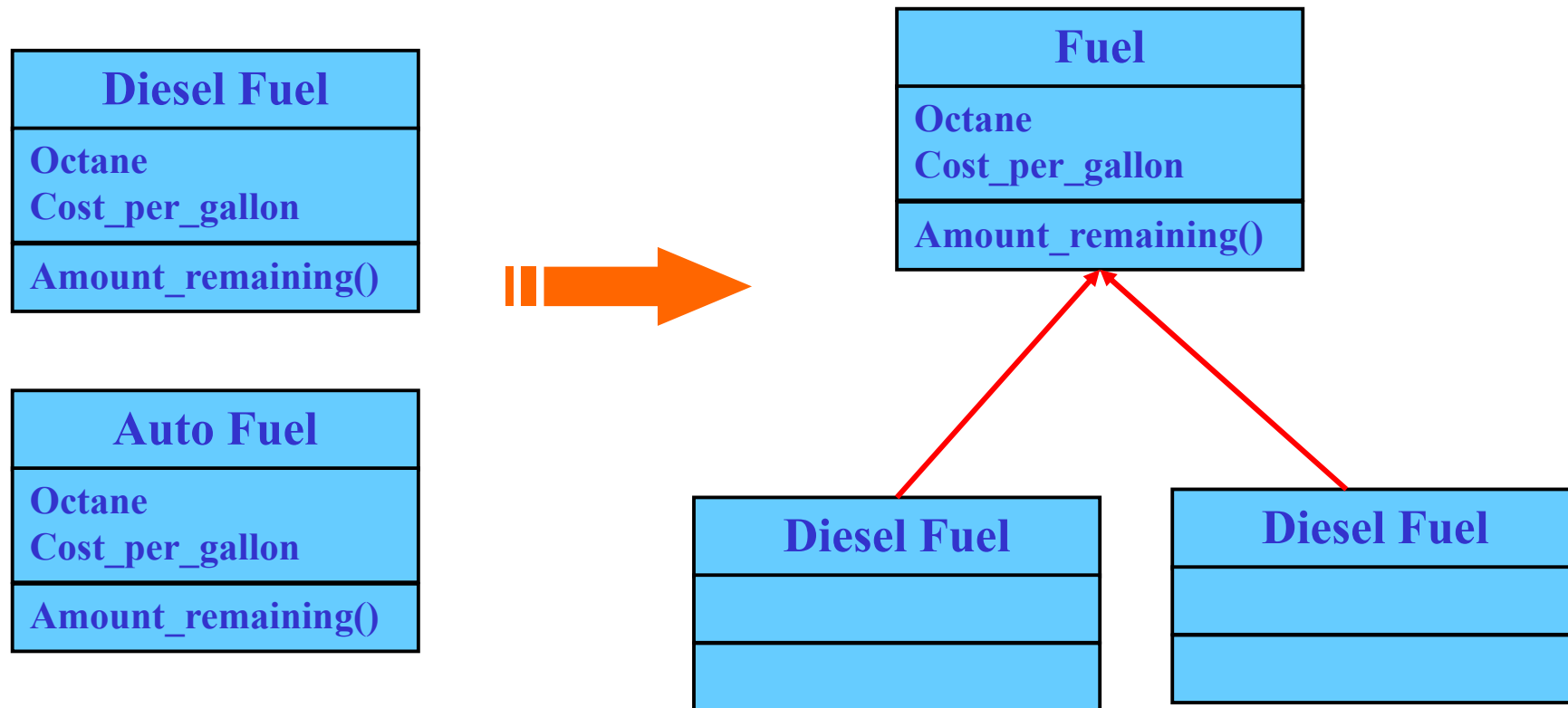
- 子类、派生类
- 父类、基类、超类

□ 类实例（Instance）

- 由某个特定类所描述的一个具体对象。
- 如类抽象“中国人”的一个实例“王志东”

6.1 What is OO?

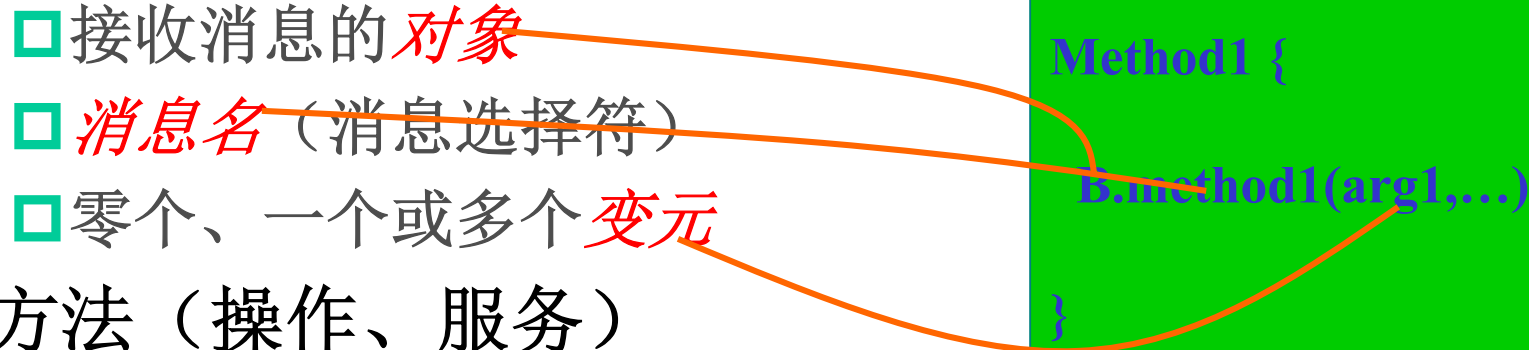
Forming a hierarchy组成层次



6.1 What is OO?

消息、方法与属性

- 消息就是某个操作的规格说明，
 - 接收消息的 **对象**
 - **消息名**（消息选择符）
 - 零个、一个或多个 **变元**
- 方法（操作、服务）
- 属性是类中所定义的数据，是实体性质的抽象



```
A
Method1 {
    B.method1(arg1,...)
}
```

6.1 What is OO?

Inheritance 继承性

□ 继承性的含义

- 对象共享所在类的结构、操作和约束等语义特性
- 多层类层次的继承传递性
- 类格的继承（多重继承）

□ 继承性的作用

- 继承性使所建立的软件系统具有开放性
- 继承是信息组织和分类的有效方法
- 提高代码可重用率和可靠性，降低开发工作量

6.1 What is OO?

多态性与重载

□ 多态性含义

- 把相同的操作施加于不同类型的对象，获得不同的结果
- 虚函数 (Virtual)
- 动态联编 (Dynamic binding)

□ 重载 Overloading

- 运算符重载：同一运算符作用于多种数据类型上。
- 函数名重载：相同的方法作用于不同的对象类型产生不同的行为效果。
- 静态联编

6.2 The OO Development Process

- ❑ 面向对象的方法从问题模型开始，然后就是识别对象、不断细化的过程。它从本质上就是迭代的和渐增的。
- ❑ **快速原型**和**反馈环路**是必需的标准结构。开发过程就是一次次的迭代反复过程。
- ❑ 传统开发模式的分析、设计和编码等各个阶段之间的明显界限变得模糊起来。其原因是因为**对象的概念弥漫了整个开发过程**。对象和它们之间的关系成为分析、设计和编码等各个阶段的共同表达媒介。
- ❑ 开发的重心从编码向分析偏移，从功能为中心向数据为中心偏移。面向对象开发的迭代和无缝性使得重用变得更加自然。

6.2 The OO Development Process

- ❑ 三个层次的概念：OOP、OOA和OOD
- ❑ OOA建立于以前的信息建模技术的基础之上，可以定义为是一种以从问题域词汇中发现的类和对象的概念来考察需求的分析方法。OOA的结果是一系列从问题域导出的“黑箱”对象。
- ❑ OOD阶段，注意的焦点从问题空间转移到了解空间。OOD是一种包含对所设计系统的逻辑的和物理的过程描述，以及系统的静态和动态模型的设计方法。

6.2 The OO Development Process

- ❑ 1988, Meyer 用语言作为表达设计的工具
- ❑ 1991, Booch的OOD技术扩展了他以前在Ada方面的工作。Booch是最先使用类图, 类分类图, 类模板和对象图来描述OOD的人
- ❑ 1990, Wrifs-Brock's的OOD技术是由职责代理来驱动的
- ❑ 1991, Rumbaugh使用对象模型、动态模型和功能模型来描述一个系统
- ❑ Coad和Yourdon确定了一个多层OO模型
- ❑ 1992, Ivar Jacobson 提出了Objectory方法(或Jacobson法), 特别强调了 “Use Case”的使用
- ❑ 1995年, Booch,Rumbaugh和Jacobson联手合作, 提出了第一版的UML(Unified Modelling Language),一体化建模语言。(目前已经成为OO建模语言的事实标准)

6.3 UML

- ❑ UML融合了Booch、OMT和OOSE方法中的基本概念，而且这些基本概念与其他面向对象技术中的基本概念大多相同；
- ❑ UML不仅仅是上述方法的简单汇合，而是扩展了现有方法的应用范围；
- ❑ UML是标准的建模语言,而不是标准的开发过程。尽管UML的应用必然以系统的开发过程为背景,但由于不同的组织和不同的应用领域,需要采取不同的开发过程。

6.3 UML - RUP软件开发过程

- ❑ Rational公司1998年发布了名为Rational Unified Process的面向对象软件开发过程框架。
- ❑ 该框架将软件开发过程分为四各阶段：
 - ❑ 初始阶段
 - ❑ 细化阶段
 - ❑ 构造阶段
 - ❑ 移交阶段
- ❑ 该过程框架强调的原则：
 - ❑ 用例驱动(Use Case Driven)
 - ❑ 以架构为中心(Architecture-Centric)
 - ❑ 迭代增量(Iterative and Incremental)

6.3 UML

□ UML的语义

- 描述基于UML的精确元模型定义。
- 元模型为UML的所有元素在语法和语义上提供了简单、一致、通用的定义性说明,使开发者能在语义上取得一致,消除了因人而异的最佳表达方法所造成的影响。此外UML还支持对元模型的扩展定义。

□ UML表示法

- 定义UML符号的表示法,为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。这些图形符号和文字所表达的是应用级的模型,在语义上它是UML元模型的实例。

6.3 UML

□ UML的用于描述模型的基本词汇（“构造块”）：

- 事物（Things）
- 关系（Relationships）
- 图（Diagrammes）

6.3 UML - 模型内的组织和UML表述

□ 事物(Things)

□ 结构事物(Structure Thing)

□ UML中的静态元素：类、接口、协作等

□ 行为事物(Behavioral Thing)

□ UML中的静态元素：交互(Interaction)、状态机(State Machine)

□ 组织事物(Grouping Thing)

□ UML的分组元素：包(Package)

□ 注释事物(Annotation Thing)

□ UML的分组元素：注释(Note)

6.3 UML - 模型内的组织和UML表述

□ 关系(Relationships)

- 关联关系(Association)
- 依赖关系(Dependency)
- 泛化关系(Generalization)
- 聚合关系(Aggregation)
- 实现关系(Realization)

□ 图(Diagrams)

- 9种图

6.3 UML - UML的9个模型

序号	模型名称	模型定义和解释
1	业务模型	建立业务流程的抽象
2	领域模型	建立系统的语境(业务操作规则)
3	用例模型	建立系统的功能需求
4	分析模型	建立概念设计（逻辑设计）
5	设计模型	建立问题的解决方案
6	过程模型	建立系统的并发和同步机制
7	部署模型	建立系统的硬件拓扑网络结构
8	实现模型	建立的软硬件配置设计
9	测试模型	建立系统的测试计划设计

6.3 UML - UML的9种图

图名称	图定义	图性质
类图	一组类、接口、协作及它们的关系	静态图
对象图	一组对象及它们的关系	静态图
用例图	一组用例、参与者及它们的关系	静态图
顺序图	一个交互，强调消息的时间顺序	动态图
协作图	一个交互，强调消息发送和接受的对象的结构组织	动态图
状态图	一个状态机，强调对象按事件排序的行为	动态图
活动图	一个状态机，强调从活动到活动的流动	动态图
构件图	一组构件及关系	静态图
配置图 (实施图)	一组接点及它们的关系	静态图

包图：包中的类以及包与包之间的关系 (静态图)

6.3 UML - UML的5种视图

□ 在UML 中，系统的表示使用5种不同的“视图”（UML定义的五类图，共 9 种图形），它们可以从软件开发的不同阶段、不同视角 和不同层次对所开发的系统进行描述。每个视图由一组图定义。

- 用户模型视图：使用use-case建模
- 结构模型视图：对静态结构（类、对象和关系）建模
- 行为模型视图：使用表示系统的动态或行为
- 实现模型视图：表示系统的结构和行为
- 环境模型视图：表示系统将实现的环境的结构和行为

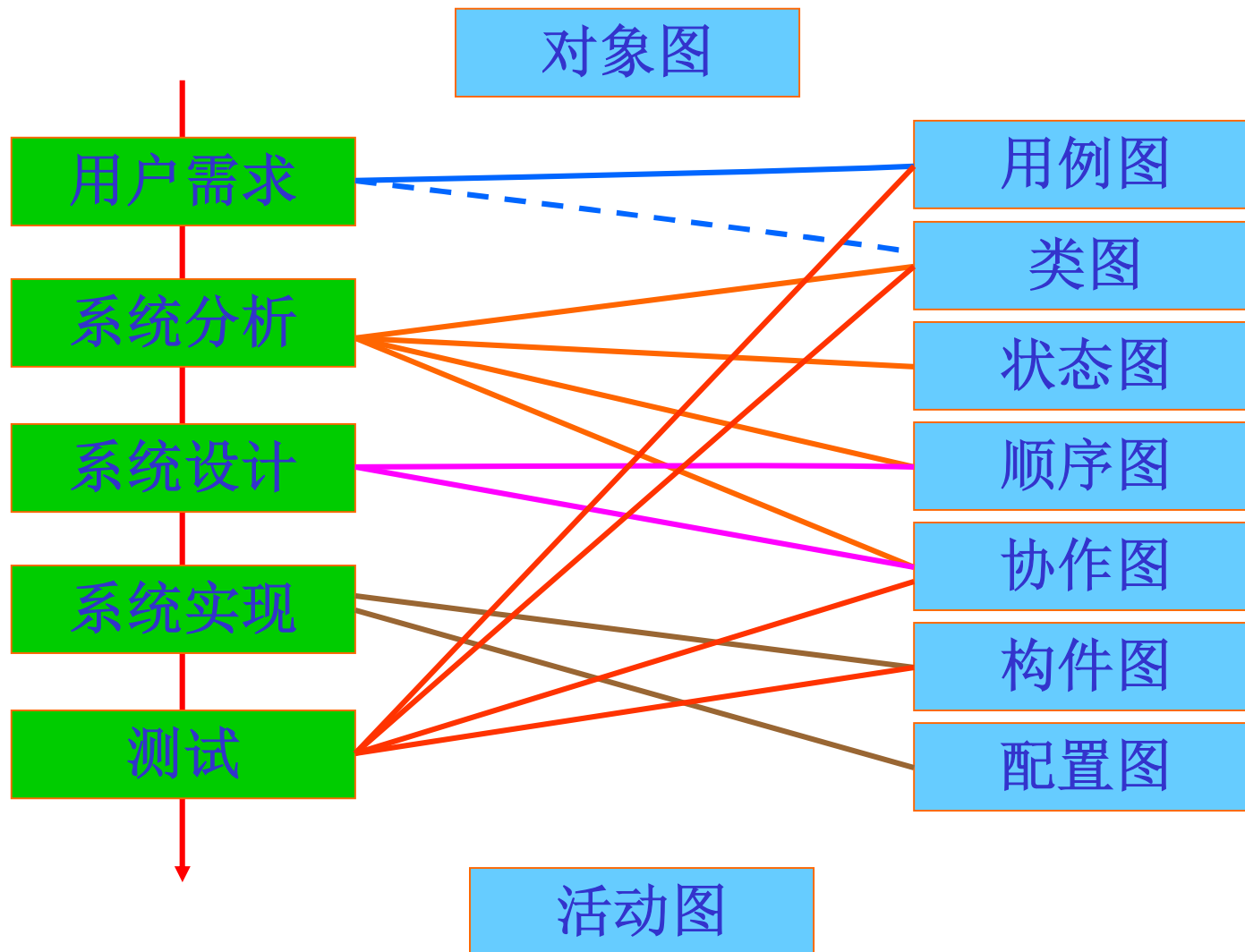
6.3 UML - UML的5种视图

视图名称	视图内容	静态表现	动态表现	观察角度
用户模型视图	系统行为, 动力	用例图	交互图、状态图、活动图	用户、分析员、测试员
结构模型视图	问题及解决方案	类图、对象图	交互图、状态图、活动图	类、接口、协作
行为模型视图	性能、可伸缩性, 吞吐量	类图、对象图	交互图、状态图、活动图	线程、进程
实现模型视图	构件、文件	构件图	交互图、状态图、活动图	配置、发布
环境模型视图	部件的发布、交付、安装	配置图 (实施图)	交互图、状态图、活动图	拓扑结构的节点

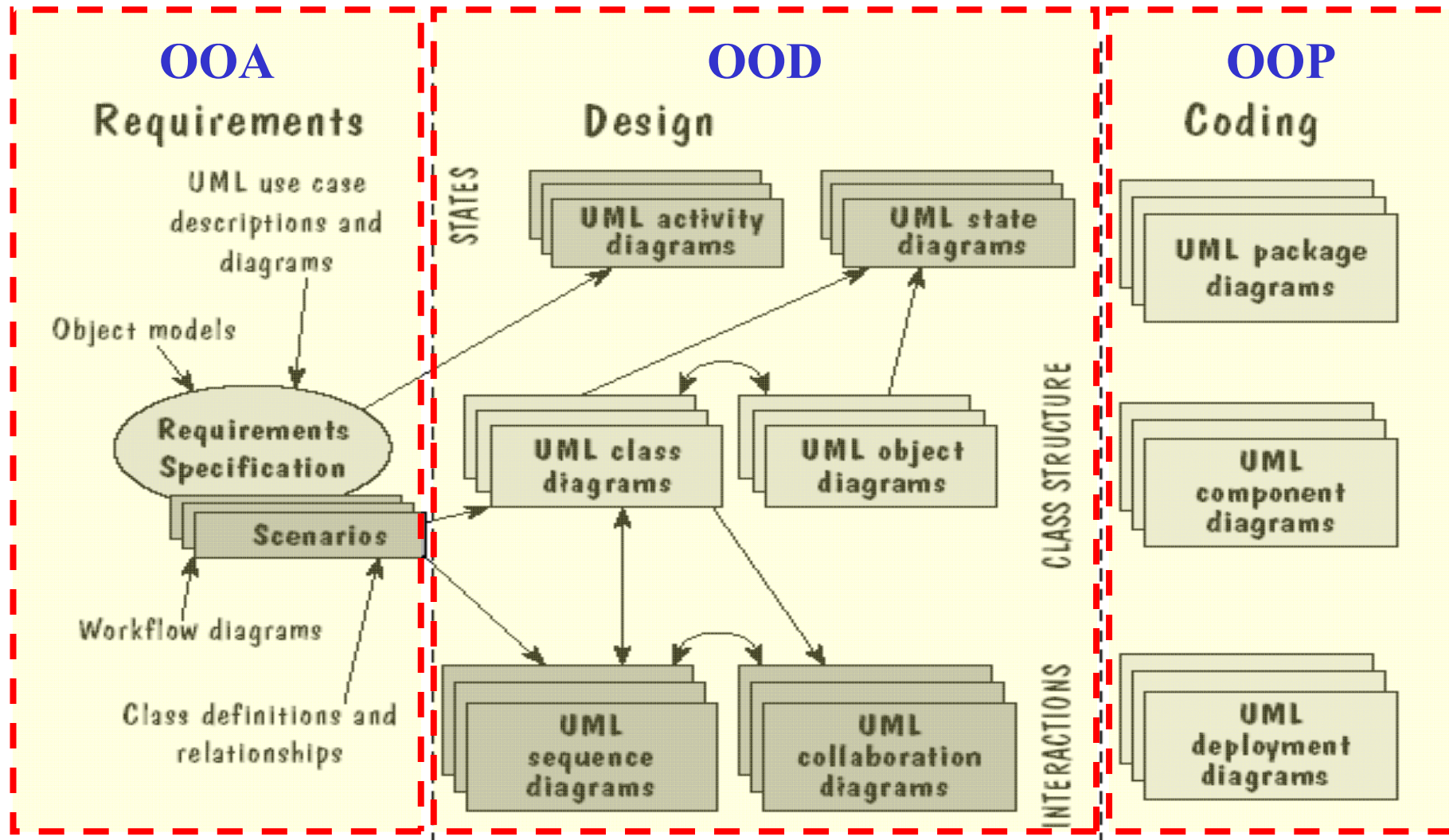
6.3 UML - UML用于软件的开发

- UML是一种建模语言，常用于建立软件系统的模型，适应于系统开发的不同阶段。
- 标准建模语言UML的主要内容可以归纳为两大类：
 - 静态建模机制
 - 动态建模机制
- UML是一种建模语言，不是一种方法，它独立于过程。利用它建模时，可遵循任何类型的建模过程。

6.3 UML - UML用于软件系统开发的不同阶段



How UML supports the development process



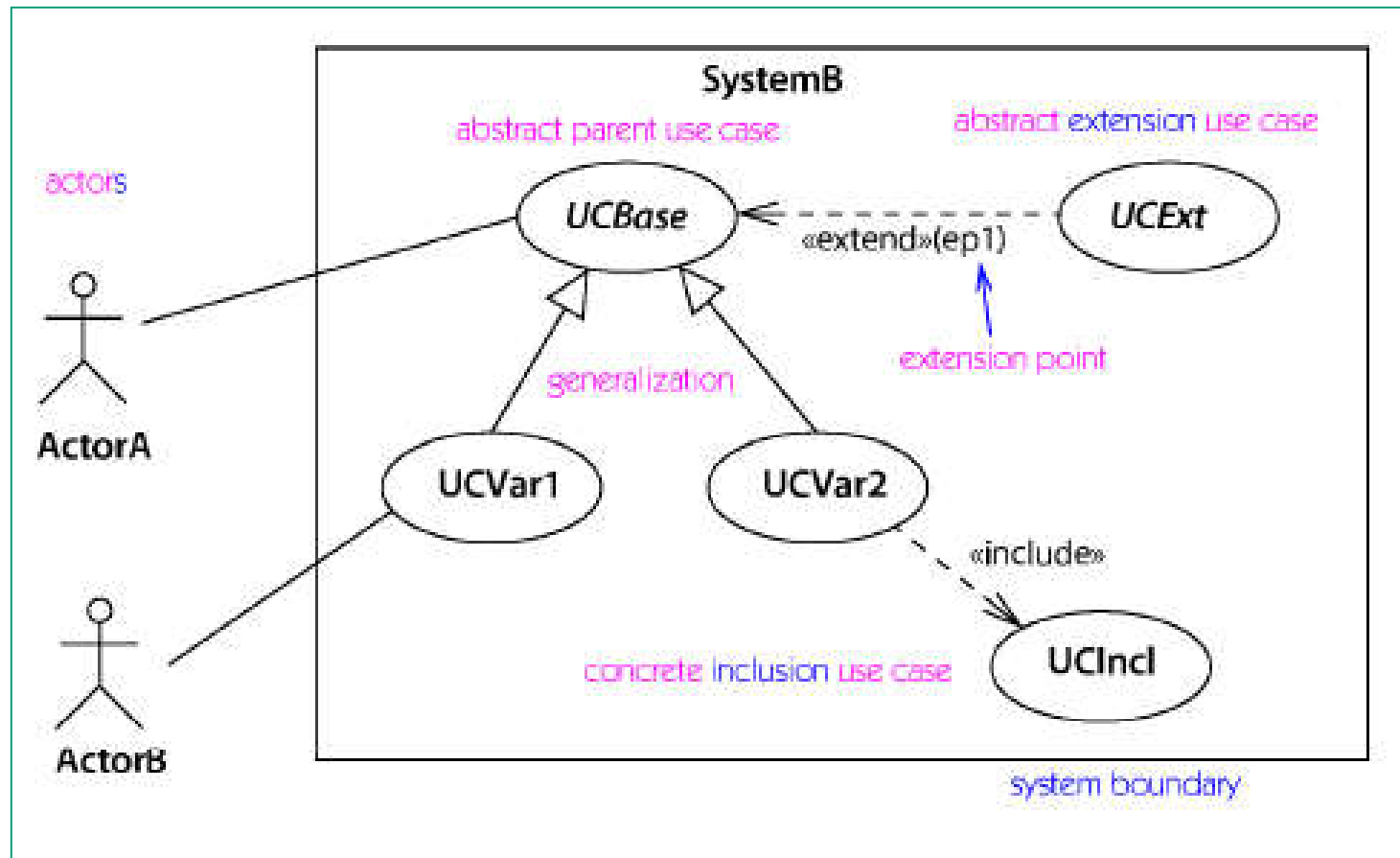
6.4 OOA

本节课中实例的需求说明 P158~P259

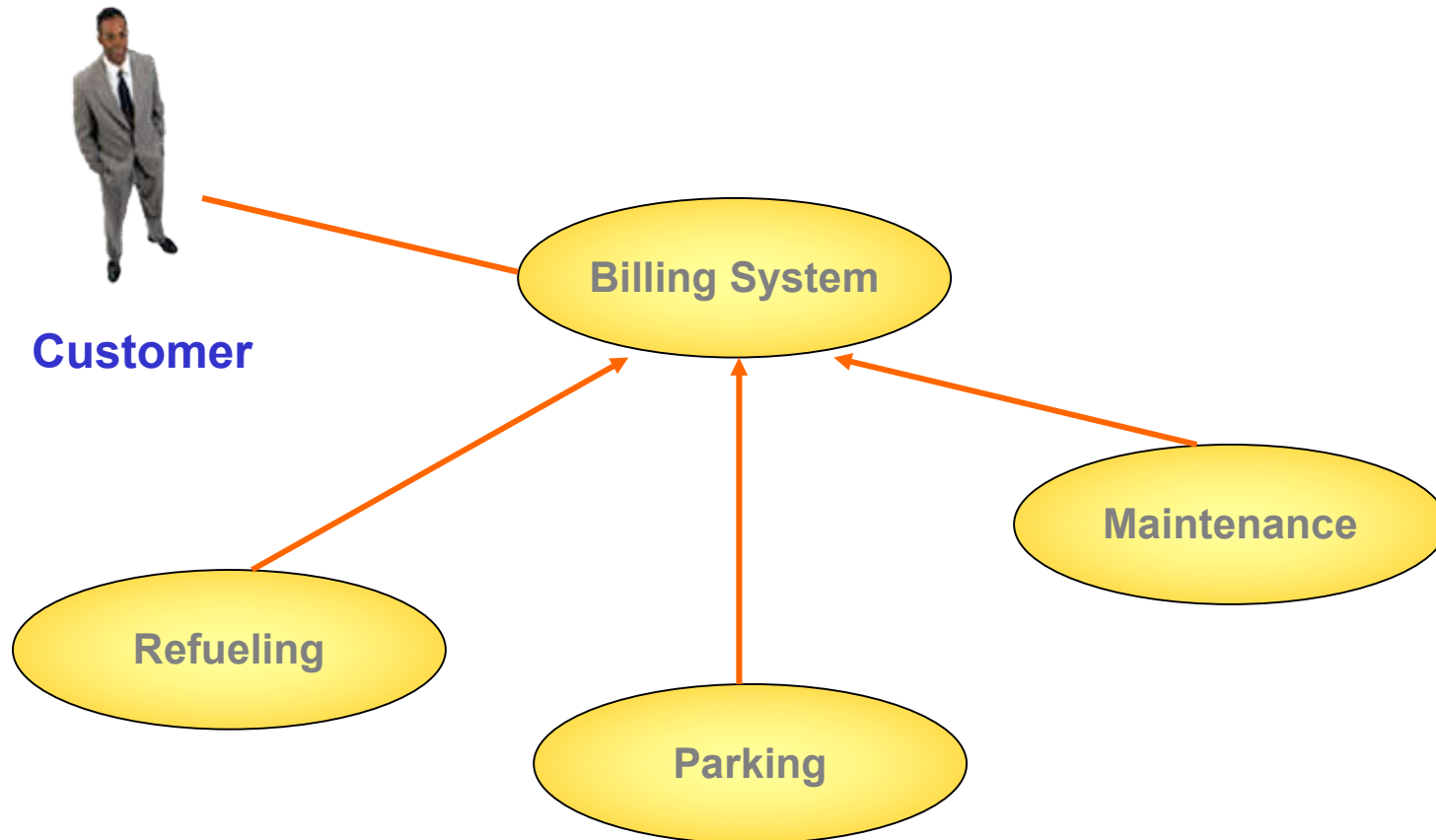
□ Royal Service Station Requirements

- Services: refueling, vehicle maintenance, parking
- Be automatically billed or be sent a monthly paper bill
- Pay using cash, credit card or personal check
- Parking is sold according to daily, weekly, and monthly rates
- Only the station manager can enter or change prices
- A discount, this discount may vary from one customer to another
- A 5% local tax applies to all purchases
-

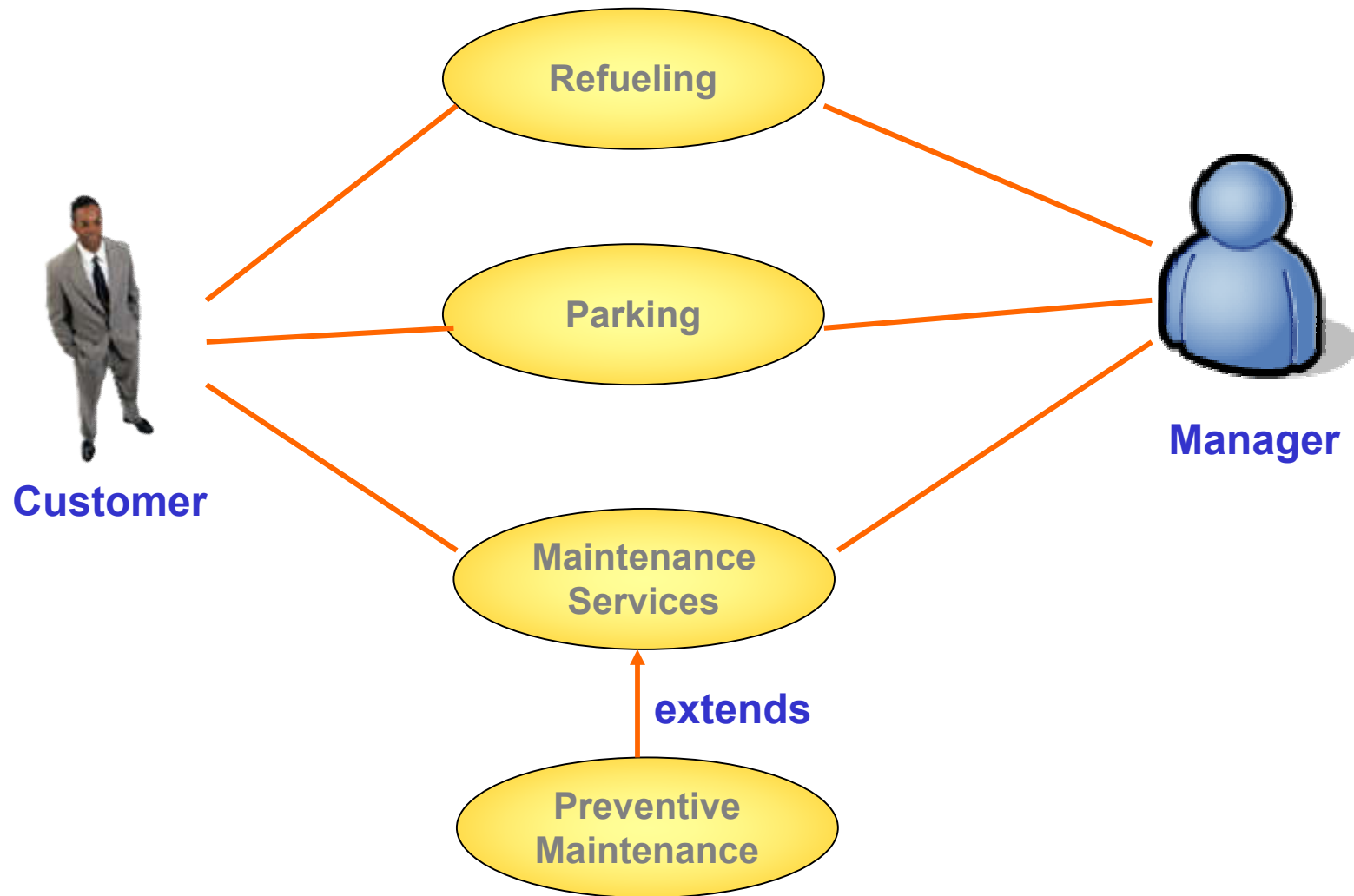
6.4 OOA - Use cases用例



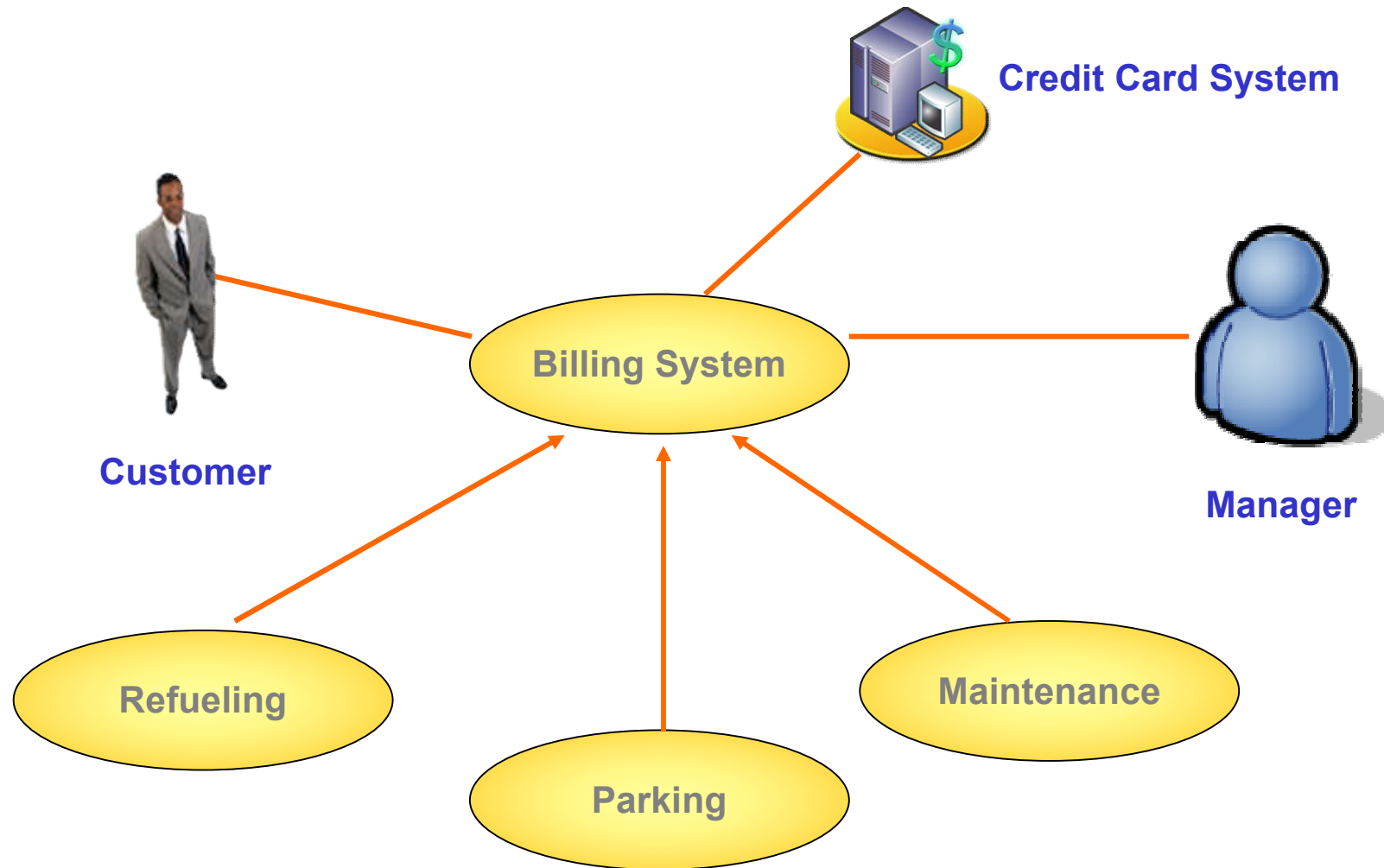
6.4 OOA - Overview of royal service station



6.4 OOA – First extension



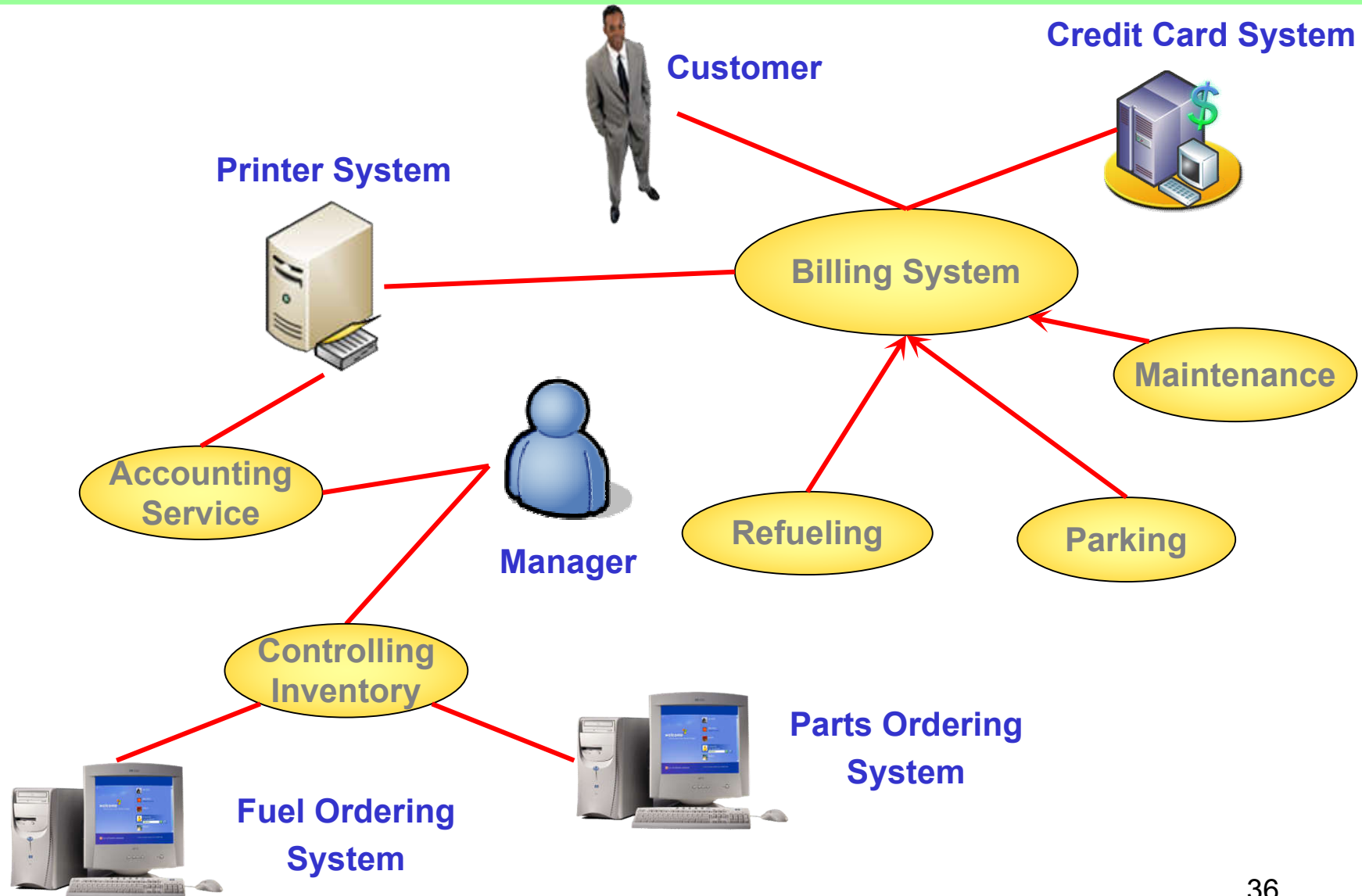
6.4 OOA – Second extension



6.4 OOA – Identifying participants

- ❑ What users or groups use the system to perform a task?
- ❑ What users or groups are needed so that the system can perform its functions?
- ❑ What external systems use the system to perform a task?
- ❑ What external systems, users or groups send information to the system?
- ❑ What external systems, users or groups receive information from the system?

6.4 OOA – The last extension



6.4 OOA – Document Use Cases

- ❑ We must document every Use Cases:
 - ❑ What that it does ?
 - ❑ When it does it?
 - ❑ Which data and behaviors are involved?
- ❑ The result is a comprehensive description of how the system will work, including its interactions with systems beyond its boundary.

6.5 OO System Design

First cut at object classes

- ❑ *Extract **classes**, **attributes** and **behaviors** from the requirements or Use Cases statements.*
 - ❑ Structures
 - ❑ External systems
 - ❑ Devices
 - ❑ Roles
 - ❑ Operating procedures
 - ❑ Places
 - ❑ Organizations
 - ❑ Things that are manipulated by the system to be built

6.5 OO System Design

First cut at object classes - examples

- | | |
|--|--------------------------------------|
| <input type="checkbox"/> personal check | <input type="checkbox"/> Discounts |
| <input type="checkbox"/> Paper bill | <input type="checkbox"/> tax |
| <input type="checkbox"/> Credit card | <input type="checkbox"/> Parking |
| <input type="checkbox"/> Station manager | <input type="checkbox"/> Maintenance |
| <input type="checkbox"/> Purchase | <input type="checkbox"/> Cash |
| <input type="checkbox"/> Fuel | <input type="checkbox"/> Prices |
| <input type="checkbox"/> Services | <input type="checkbox"/> |

Which is class or attribute?

6.5 OO System Design

Guidelines for building classes

- ❑ *Extract **nouns** from specification to get candidate classes*
 - ❑ What needs to be “processed” in some way?
 - ❑ What items have multiple attributes?
 - ❑ When do you have more than one object in a class?
 - ❑ What is based on the requirements themselves, not derived from your understanding of the requirements?
 - ❑ What attributes and operations are always applicable to a class or object?

6.5 OO System Design

First Grouping of Attributes and Classes

Attributes	Classes
Personal check	Customer
Tax	Maintenance
Price	Services
Cash	Parking
Credit card	Fuel
Discount	Bill
	Purchase
	Station manager

6.5 OO System Design

Second Grouping of Attributes and Classes

Attributes	Classes
Personal check	Customer
Tax	Maintenance
Price	Services
Cash	Parking
Credit card	Fuel
Discount	Bill
Birth date	Purchase
Name	Periodic message
Address	Station manager

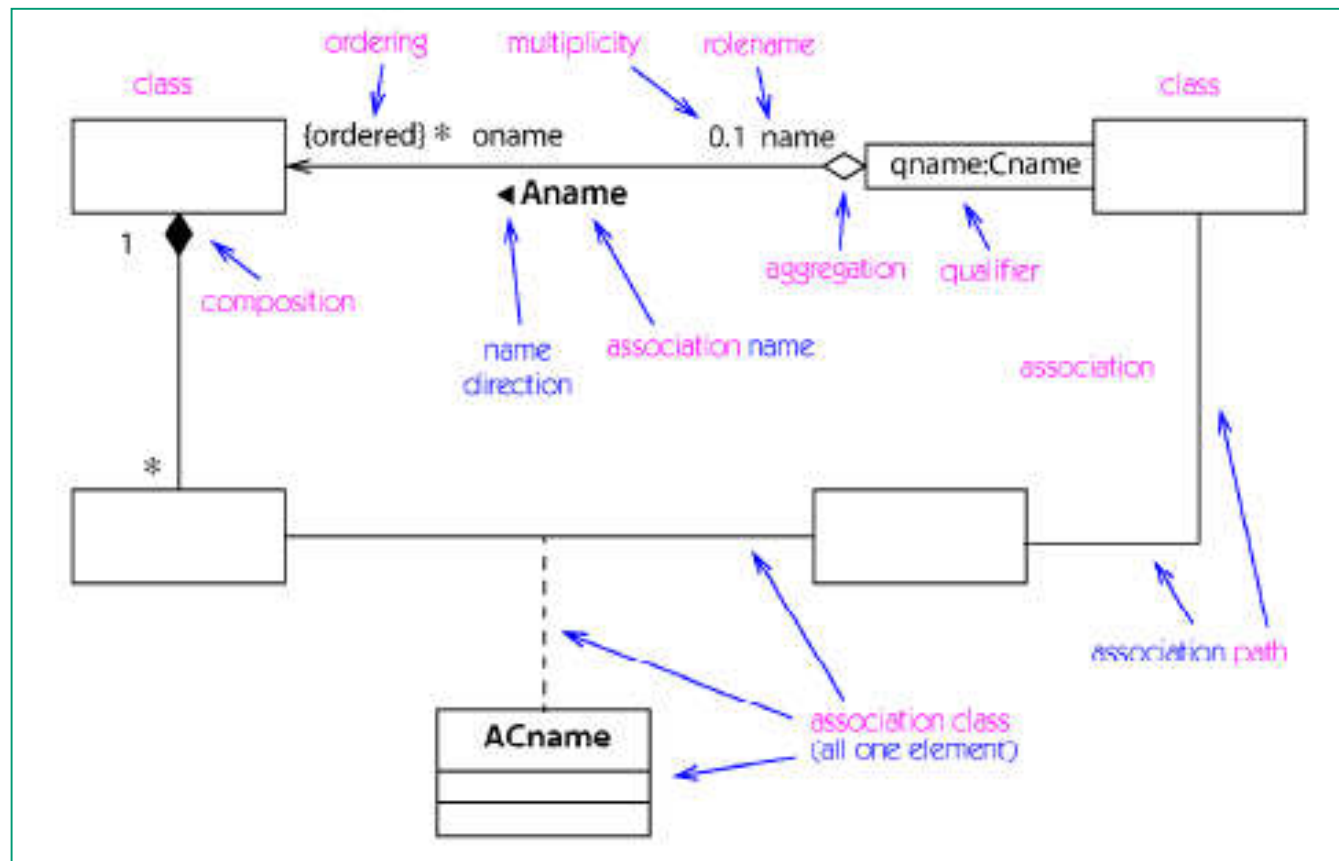
6.5 OO System Design

Guidelines for identifying behaviors

- ❑ Extract verbs from specification to identify behaviors
 - ❑ Imperative verbs 祈使动词
 - ❑ Passive verbs 被动词
 - ❑ Actions 动作
 - ❑ Things or reminded events 事情或想起来的事件
 - ❑ Roles 任务
 - ❑ Operating procedures 运行程序
 - ❑ Services provided by an organization

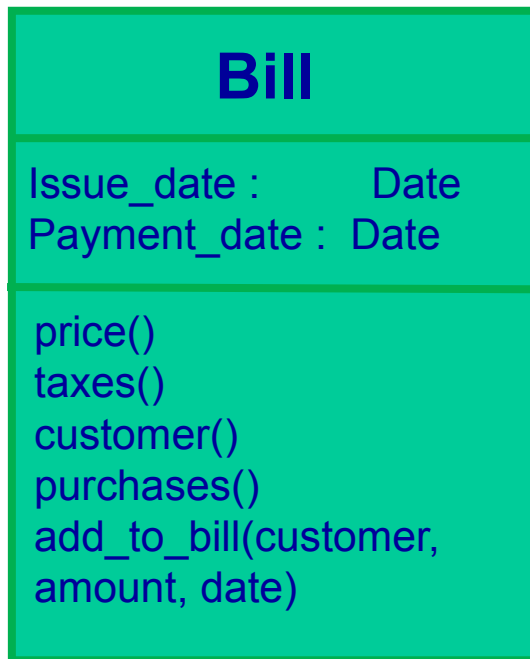
6.5 OO System Design

UML class diagrams



6.5 OO System Design

UML class diagrams



Class Name

Attribute : type = initial value

Operation(arg list) : return type

6.5 OO System Design

UML class diagrams

Superclass

Supertype

Inheritance (*is-a*)

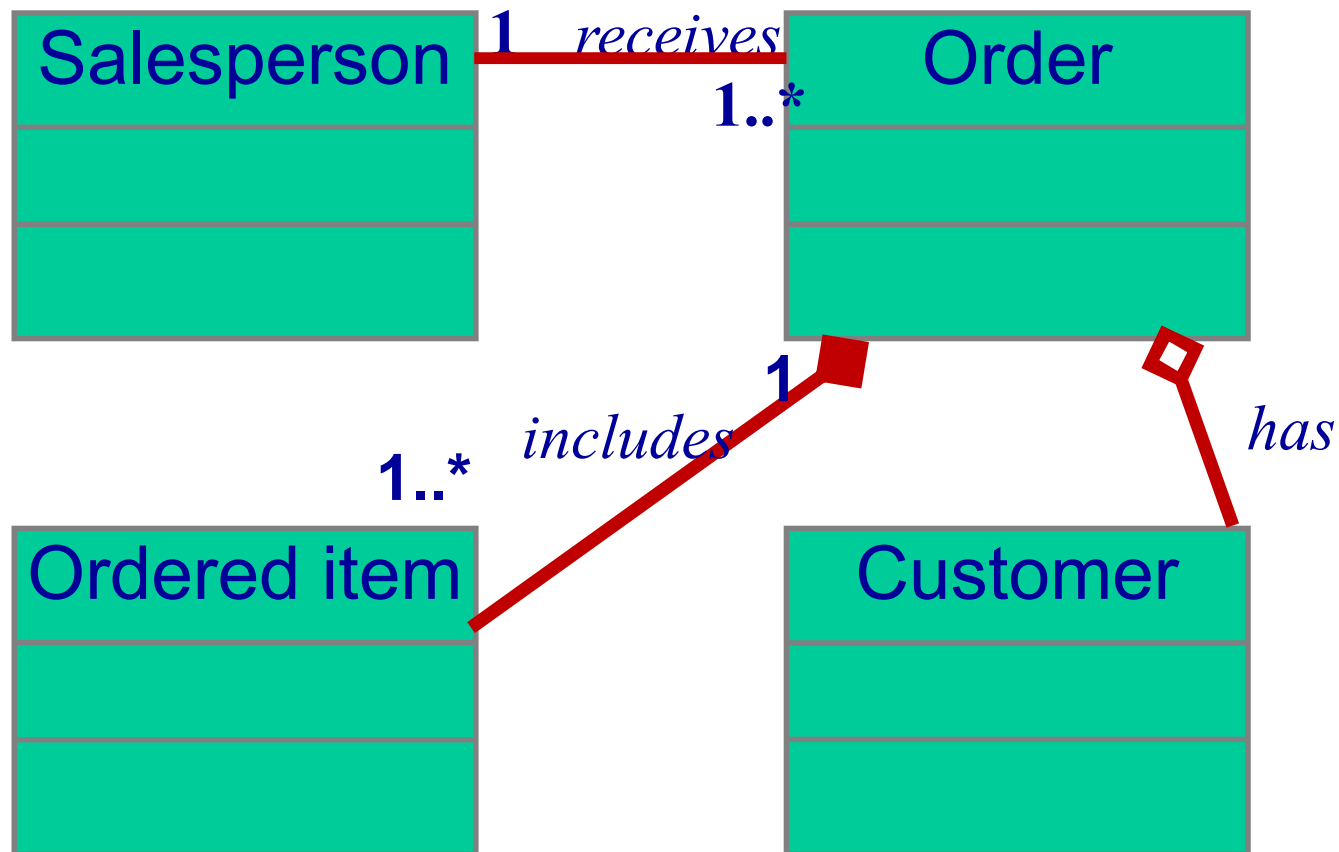
Subclass

Subtype



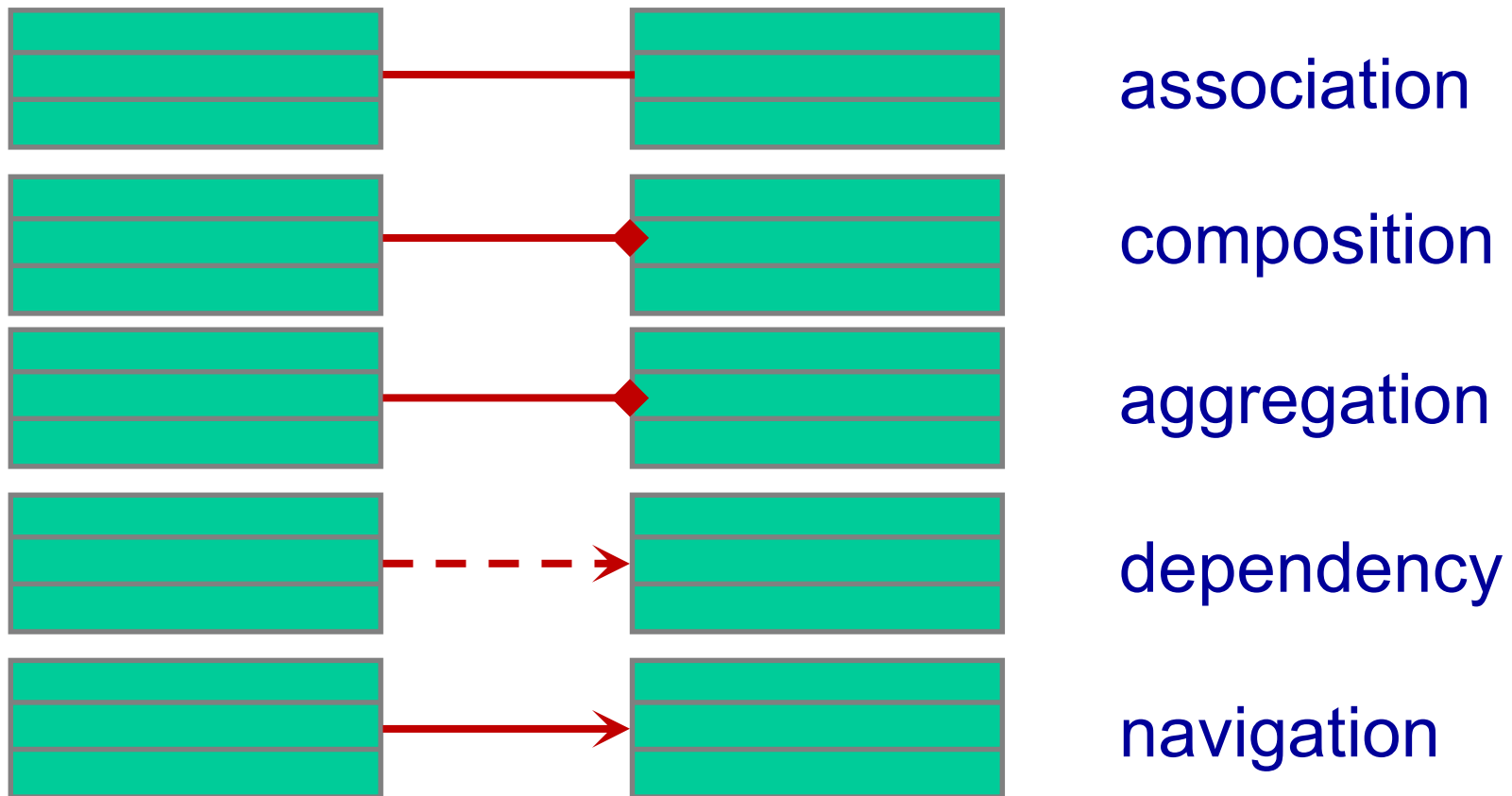
6.5 OO System Design

UML class diagrams



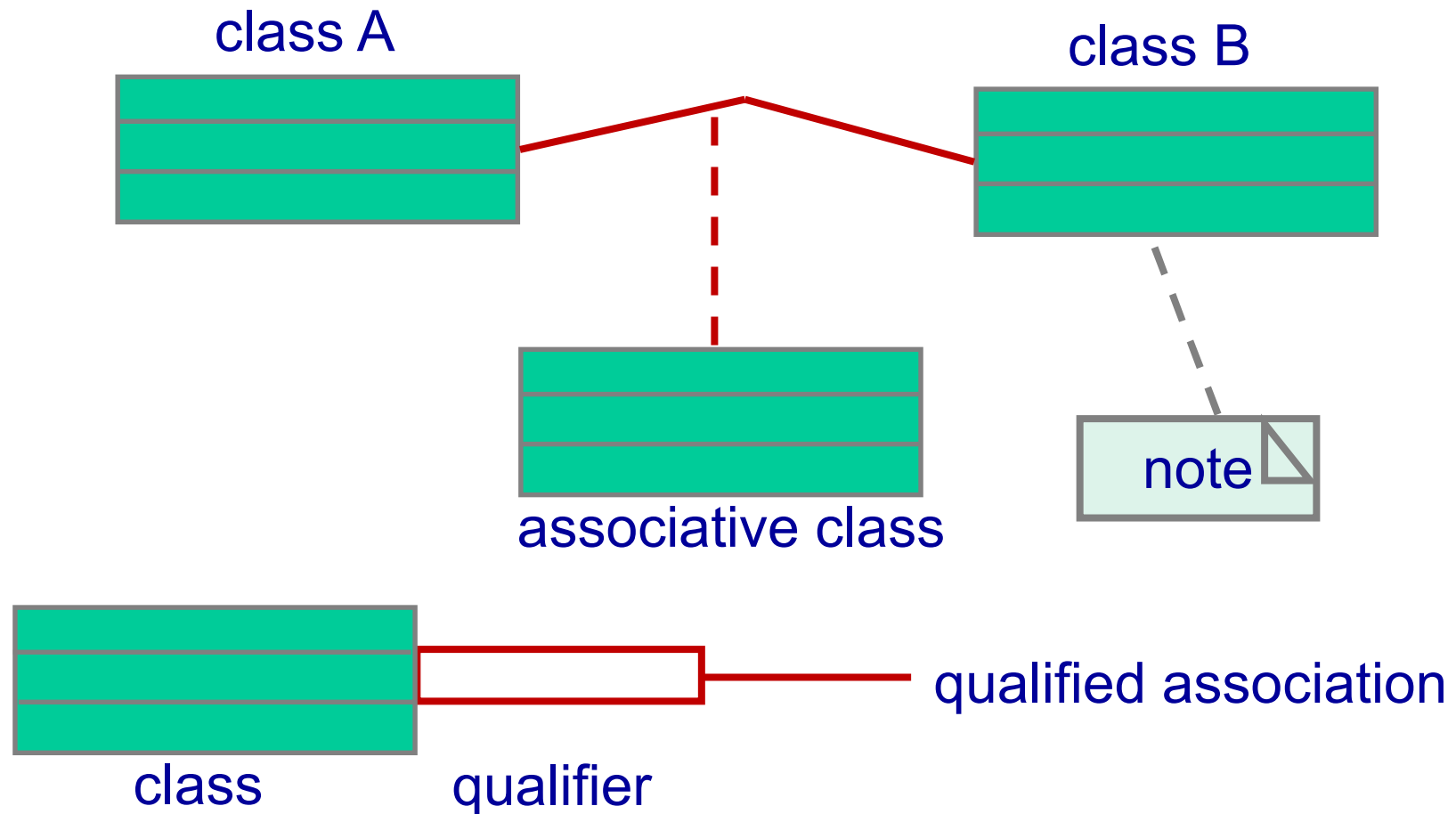
6.5 OO System Design

UML class diagrams

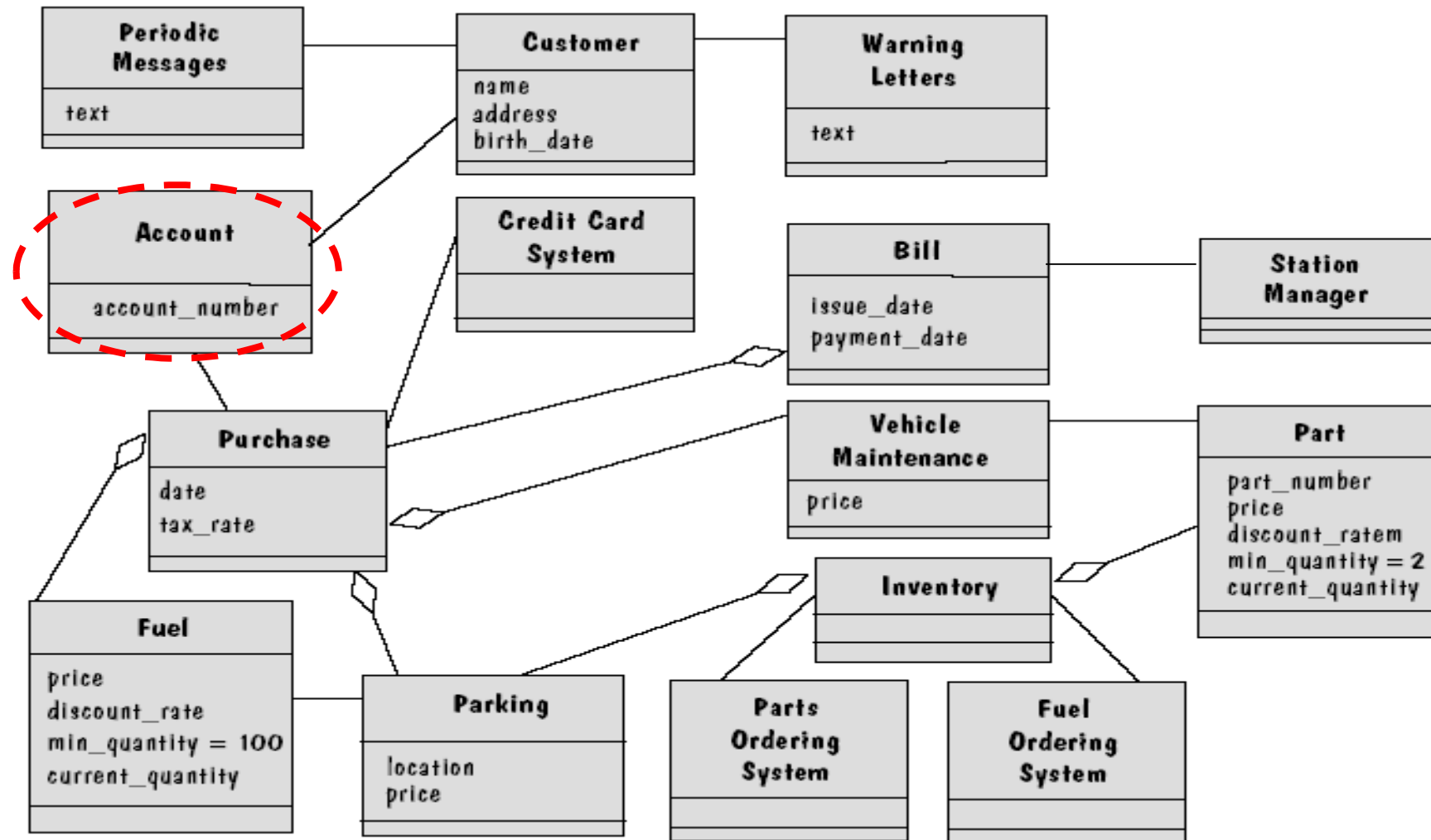


6.5 OO System Design

UML class diagrams



6.5 OO System Design – First Design

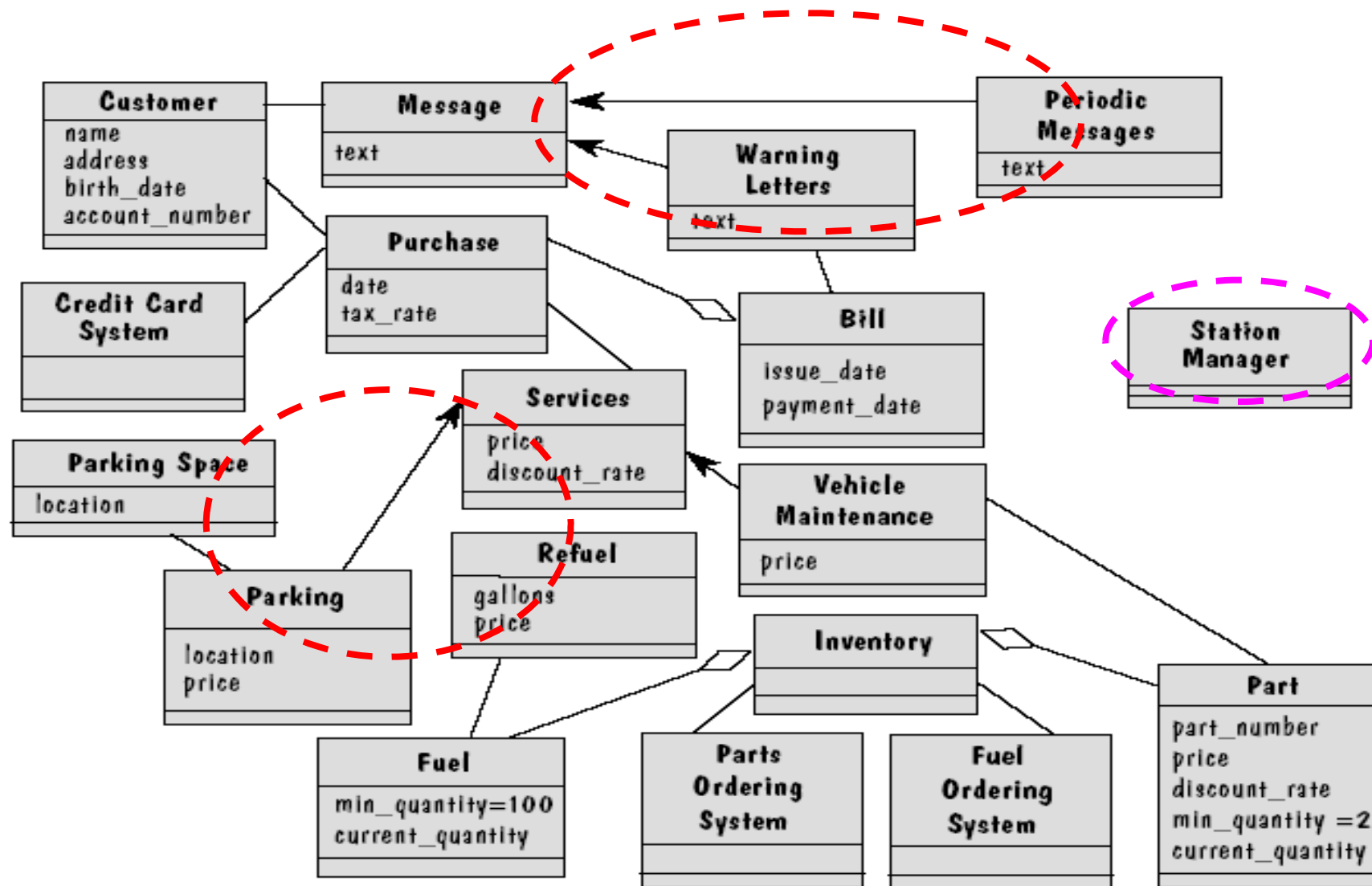


6.5 OO System Design

Sway Back and Forth

- ❑ Abstract a *Message* Class from *warning letter* and *periodic message*.
- ❑ The *fuel* class should be connected to *inventory class*
- ❑ Delete the *account* class(has only one attribute) and add the account number to the *customer class*.
- ❑ Add three new class, *refuel, parking space, service*.
 - ❑ For taking advantage of the polymorphism (price, discount)
 - ❑ Remove the discount rate from the *fuel class* and put it in the *part class*.
 - ❑ Remove the price from the *fuel class* and add it to the *refuel class*.
 - ❑ Put the price of a parking space, location in the *parking class*.

6.5 OO System Design – Second Design

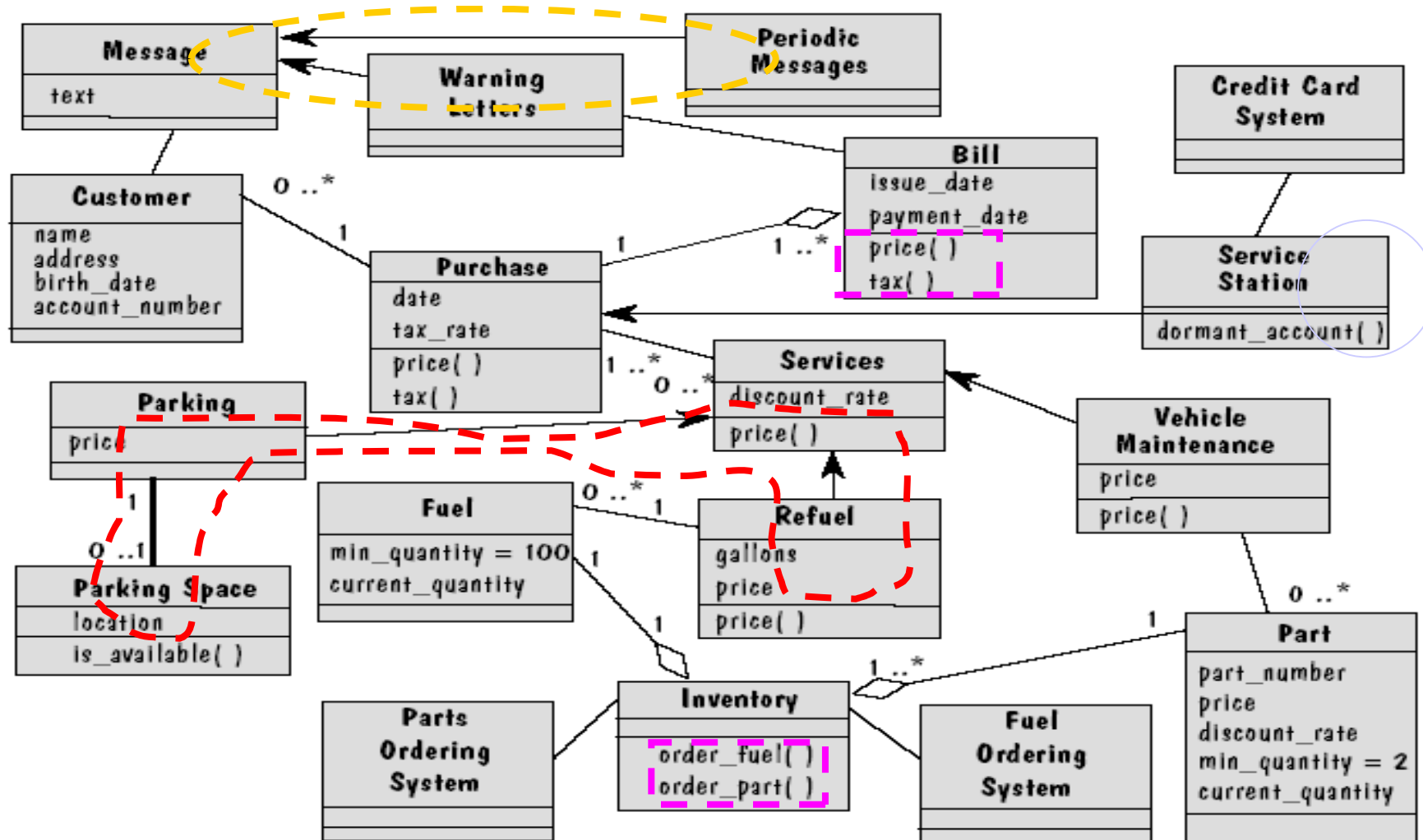


6.5 OO System Design

Sway Back and Forth

- Delete *Station Manager class*
- Add service station to handle the tracking of dormant accounts

6.5 OO System Design – Final Design



6.5 OO System Design

Other UML Diagrams 其他UML图

- ❑ Class description template
- ❑ Package diagrams
- ❑ Sequence diagrams
- ❑ Collaboration diagrams
- ❑ State diagrams
- ❑ Activity diagrams

Class name: *refuel*

Category: service

External documents:

Export control: Public

Cardinality: n

Hierarchy:

Superclasses: services

Associations:

fuel in association <name>

Operation name: price

Public member of: refuel

Documentation:

// Calculates fuel final price

Preconditions:

gallons > 0

Object diagram:

(unspecified)

Semantics:

final_price = gallons * price

Object diagram:

(unspecified)

Public interface:

Operations:

price

Private interface:

Attributes:

gallons

price

Implementation:

Attributes:

gallons

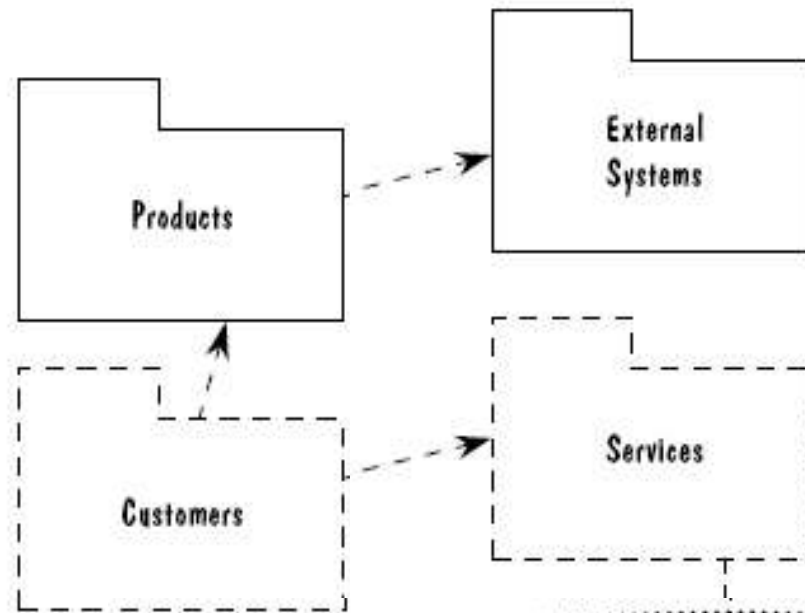
price

State machine: no

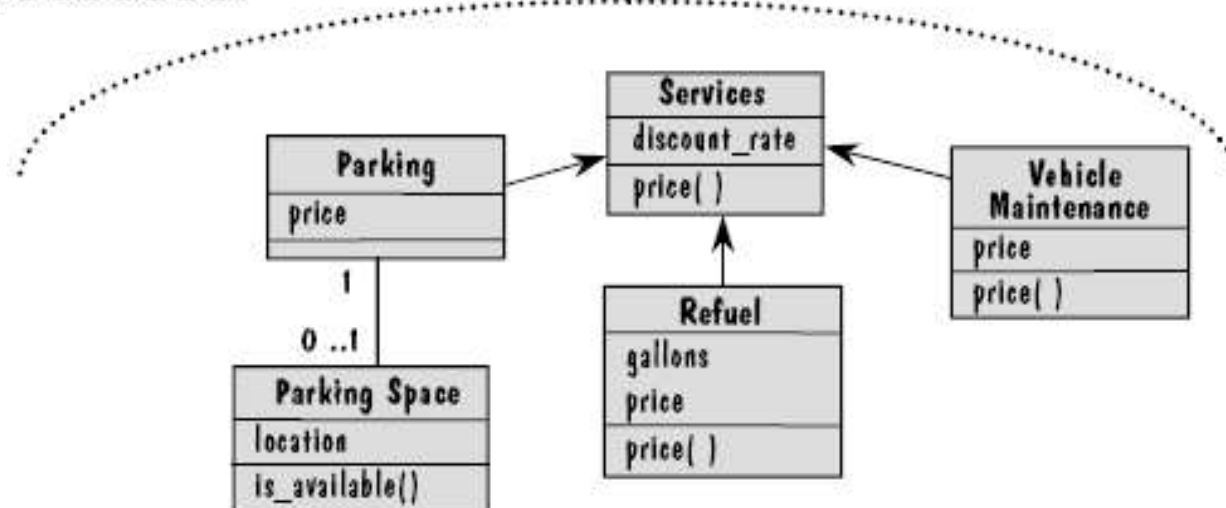
Concurrency: sequential

Persistence: transient

6.5 OO System Design – Package diagrams

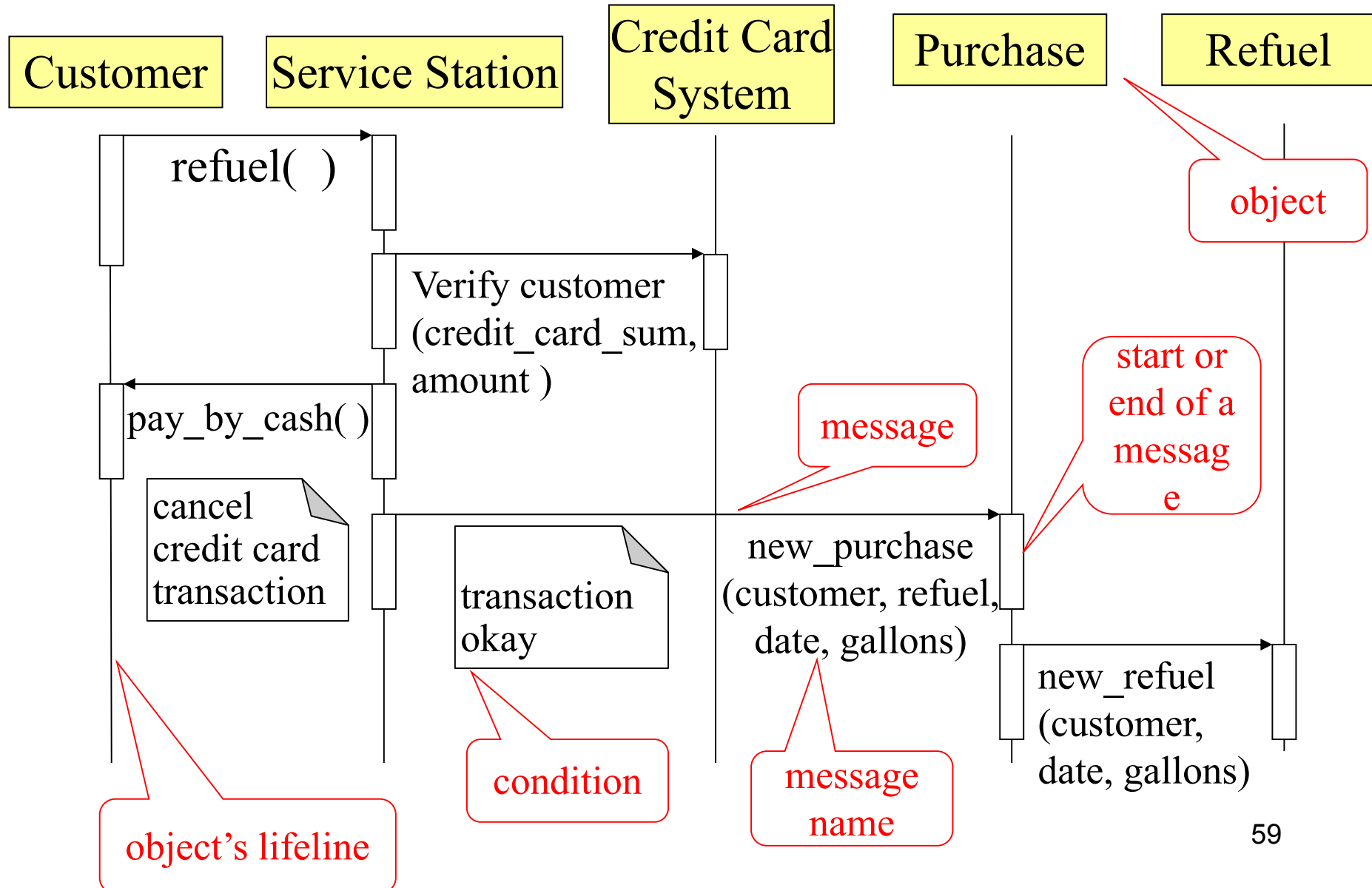


- View the system as a small collection of the packages 将系统看作是包的集合

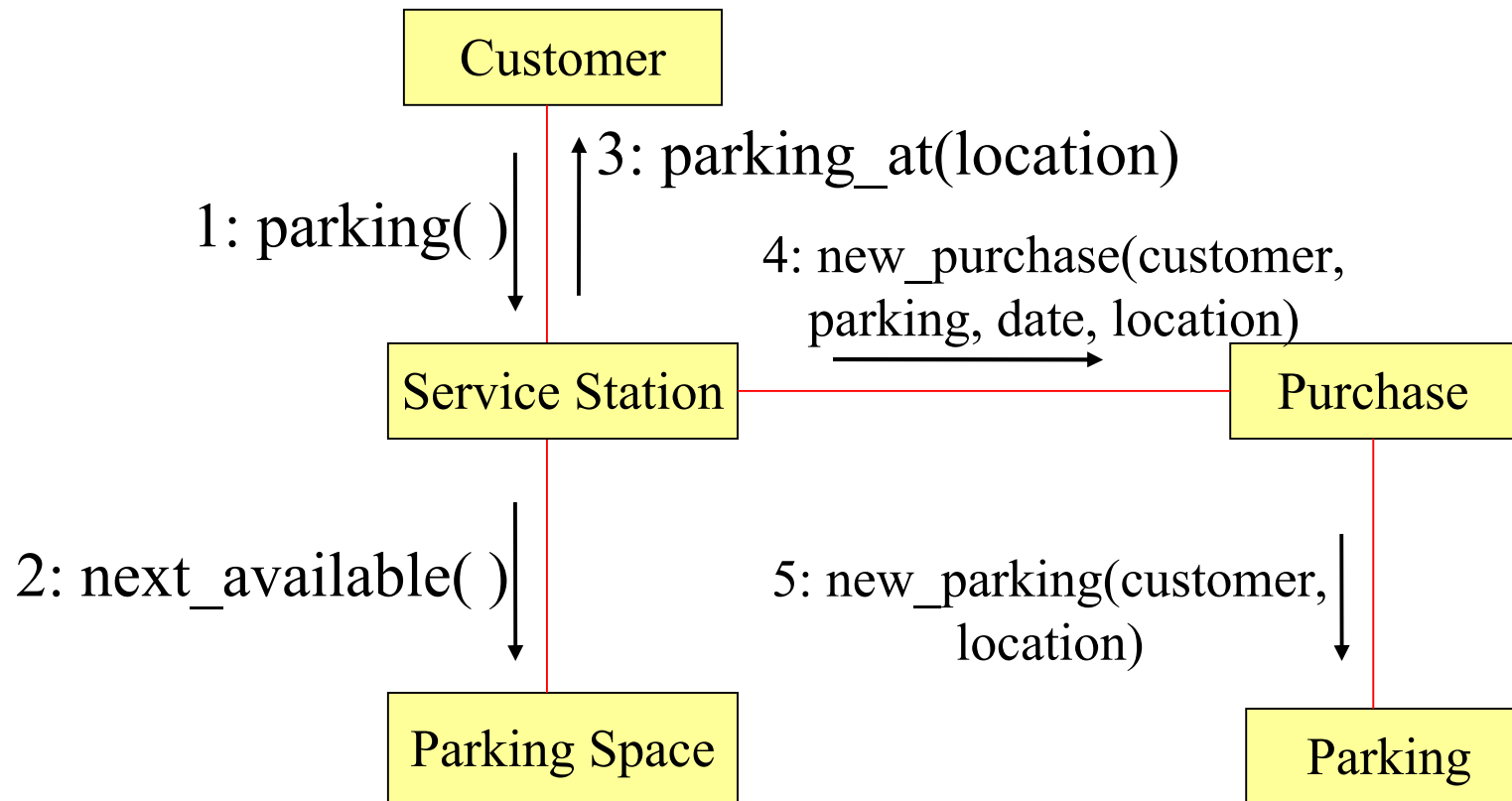


6.5 OO System Design - Interaction Diagrams

- ❑ One interaction diagram for one use case
 - ❑ Describe how operations and behaviors are handled by the objects in the design
- ❑ Sequence diagram
 - ❑ Sequence in which activities or behaviors occur
- ❑ Collaboration diagram
 - ❑ How the objects are connected statically

Sequence diagram for the *refuel* use case

Collaboration diagram for the *parking* use case



State diagram and activity diagram

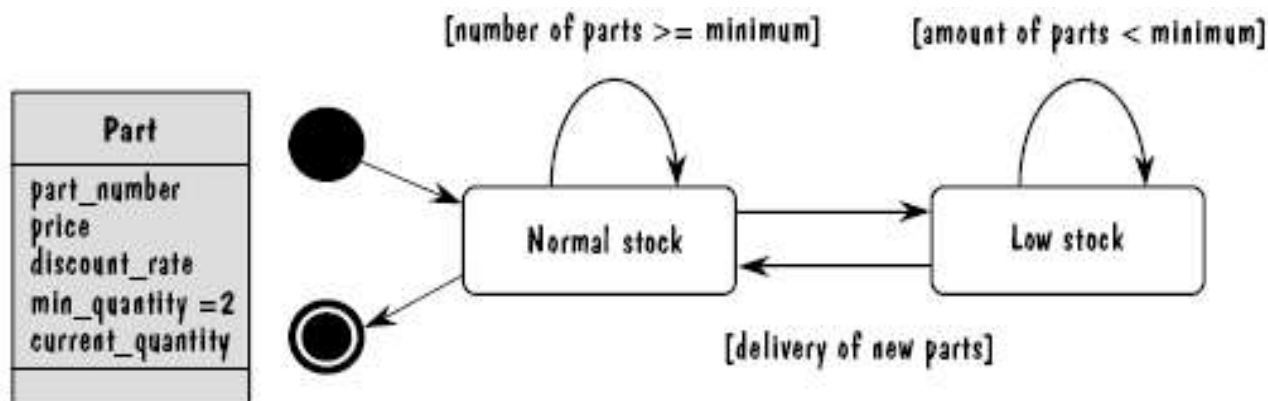
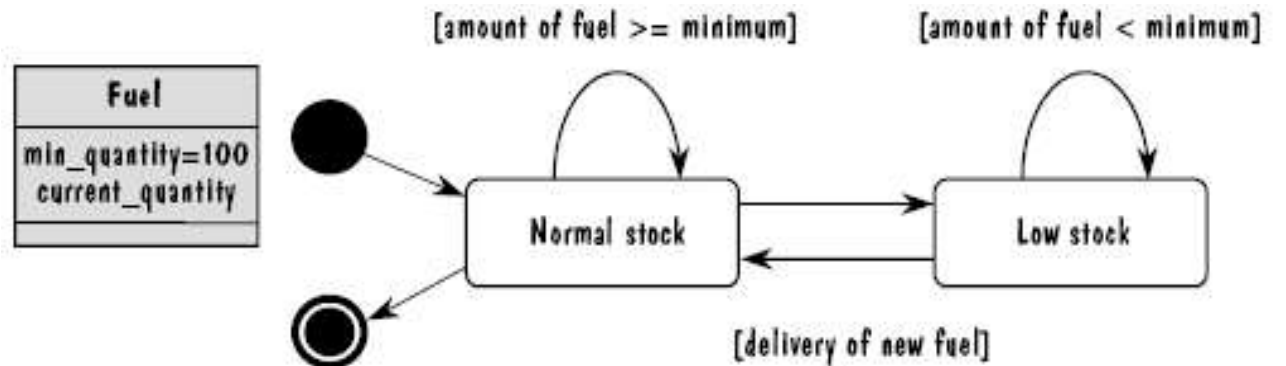
□ State diagram shows

- the possible states an object can take 某个对象存在的所有可能的状态
- the events that trigger the transition from one state to the next 从一个状态到下个状态所需触发的事件
- the actions that result from each state change 每一个状态变化得到的活动

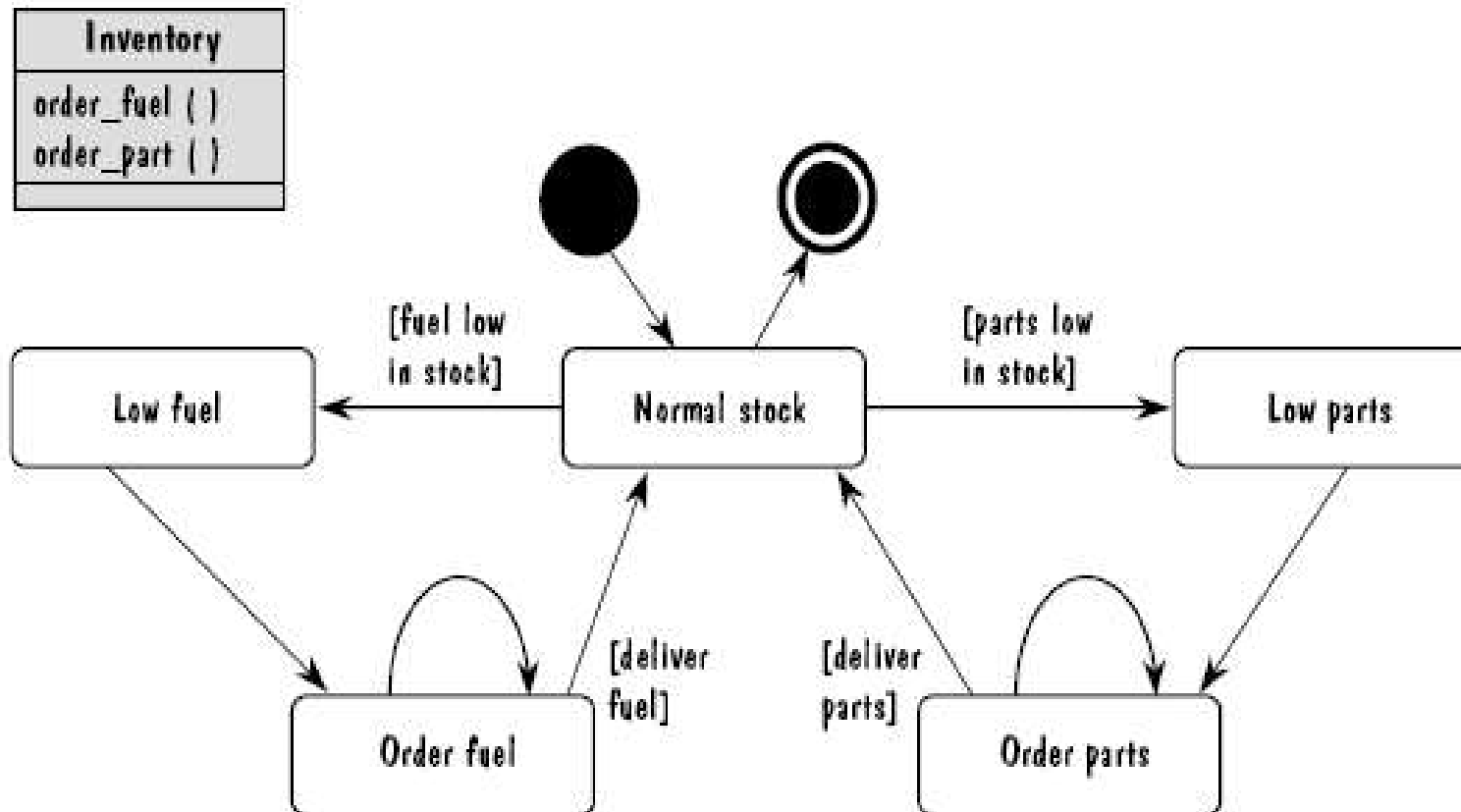
□ Activity diagram

- To model the flow of procedures or activities in a class 为在某个类中的过程或活动建模

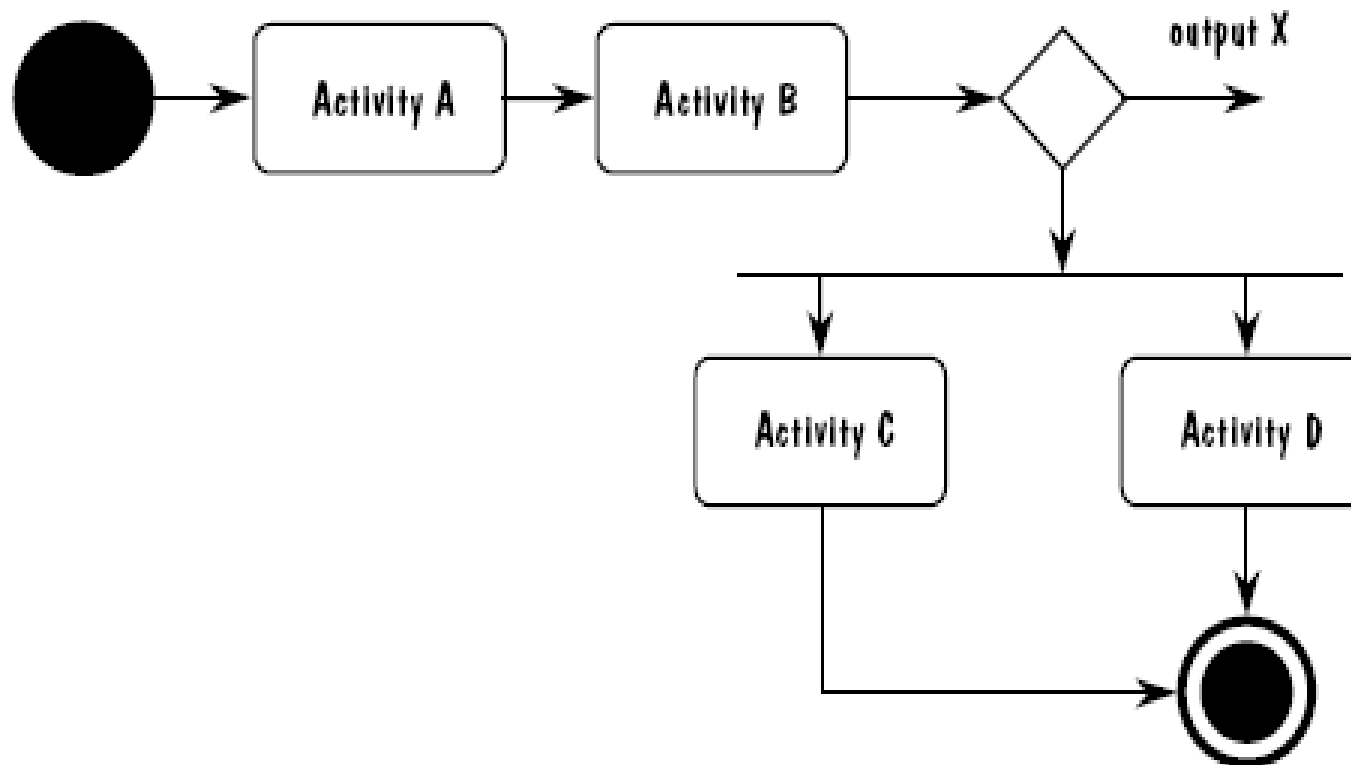
State Diagrams Examples

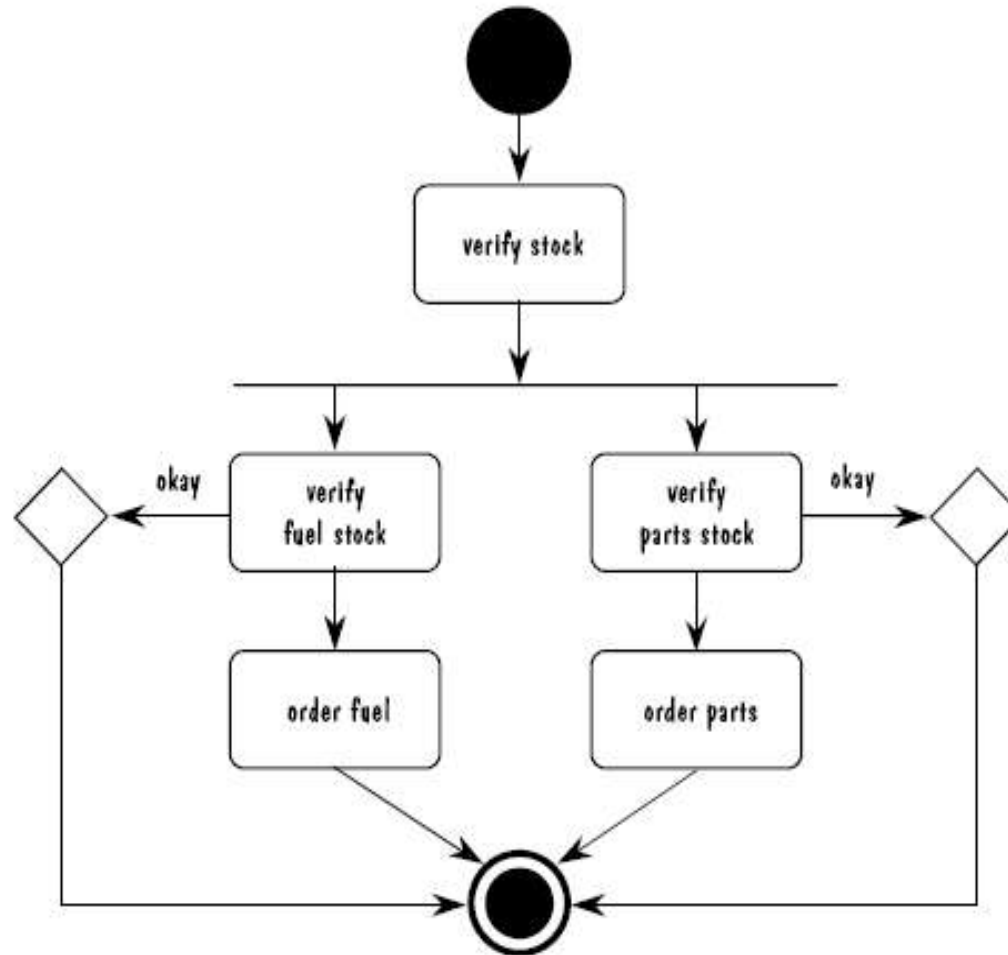


We can put the two similar objects together

State diagrams for *inventory* class

Basic Activity Diagram



Activity diagrams for *inventory* class 存货类活动图

6.6 OO Program Design

Program design considerations

- ❑ Nonfunctional requirements
- ❑ Reused components
- ❑ Reusable components
- ❑ User interface requirements
- ❑ Data structure and management details

6.6 OO Program Design – User interface design

- ❑ Defining the humans who will interact with system
- ❑ Developing scenarios for each way that the system can perform a task.
- ❑ Designing a hierarchy of user commands.
- ❑ Refining the sequence of users interaction with the system
- ❑ Designing relevant classes in the hierarchy to implement the user interface design decisions.
- ❑ Integrating the user interface classes in the overall system classes hierarchy.

6.6 OO Program Design – User interface design

Before

Royal Service Station
65 Ninth Avenue
New York City, NY
BILL

Customer: _____
Date: _____

<u>Date</u>	<u>Type</u>	<u>Amount</u>

Total: _____

After

BILL

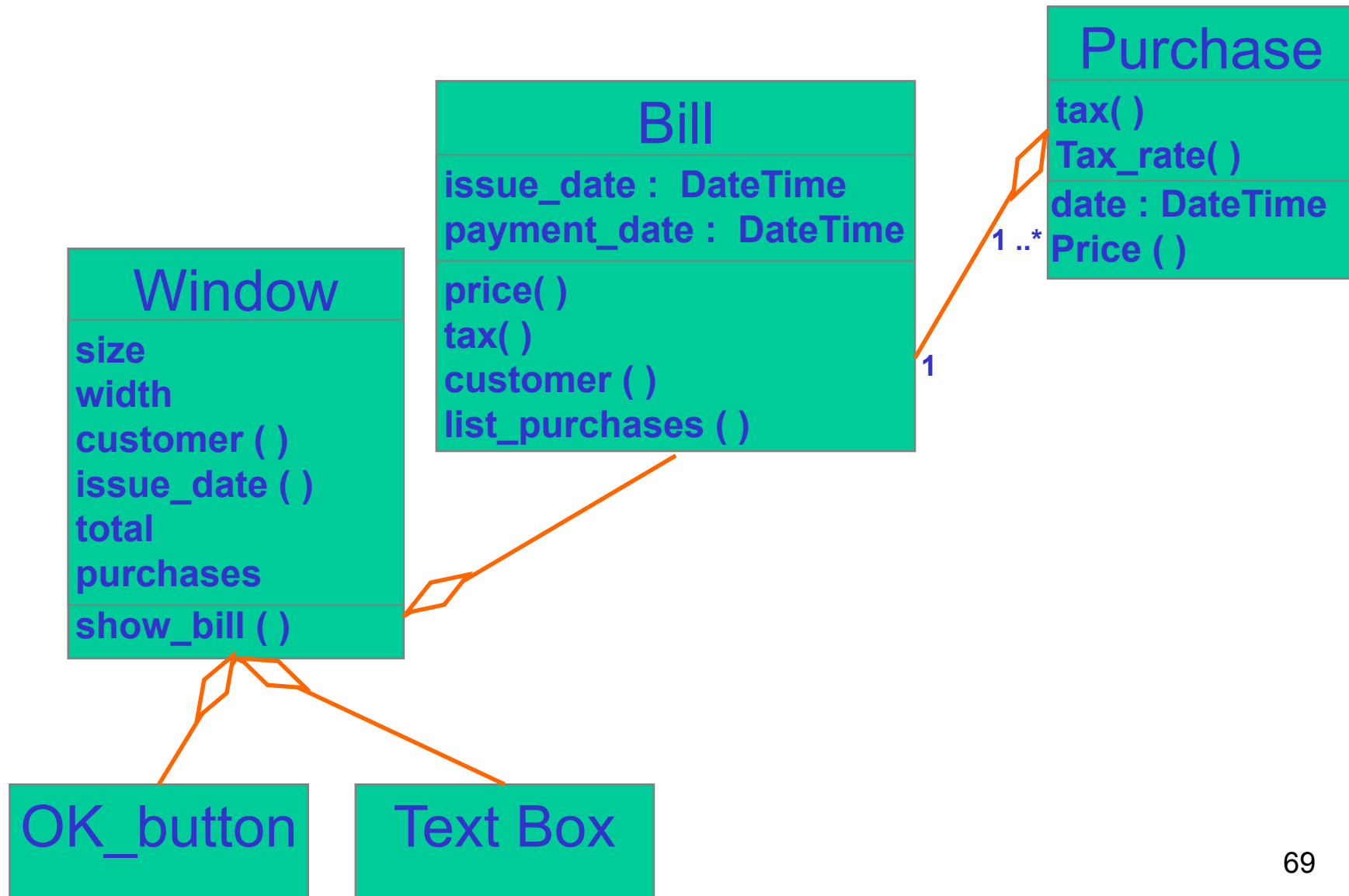
Customer name:

Issue date:

Date	Purchases Type	Amount
<div>OK? <input type="text"/></div>		

Total:

6.6 OO Program Design – User interface design

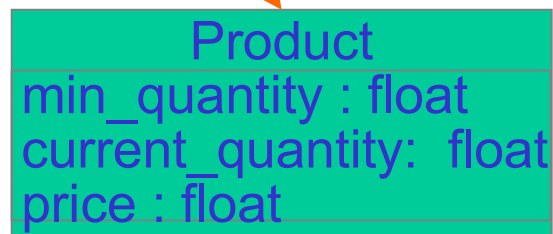
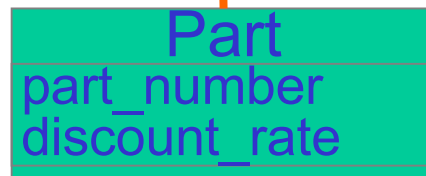
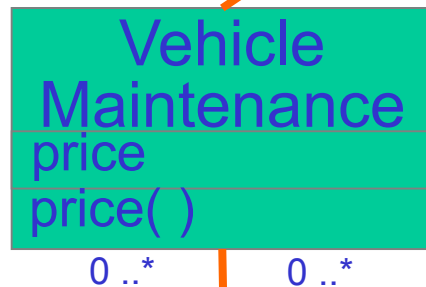
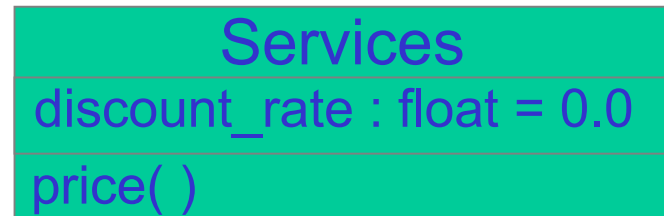


6.6 OO Program Design – Data management design

□ Four steps:

- Identify the data, data structures, and relationship among them.
- Design services to manage the data structure and relationships.
- Find tools, such as database management systems, to implement some of the data management tasks.
- Design classes and class hierarchy to oversee the data management functions.

6.6 OO Program Design – Data management design



vehicle maintenance	ID	discount_rate	price
	1	10	20
	2	10	35
	3	15	5

part X vehicle	ID	part_number
	1	1
	2	12
	3	6

part	part_number	min_quantity	current_quantity	price	discount
	1	10	20	134.00	0.0
	2	10	14	6.50	0.0
	3	15	25	21.75	5.0

下节课内容

第七章 编码；第八章 程序测试

□ 下节课之前请大家阅读第七章、第八章