Unit1:Object-oriented foundation and class diagram

- 1.1 Object-oriented foundation
  - 1.1.1 Object-oriented and procedure-oriented programming
  - 1.1.2 class and object
  - 1.1.3 class attribute and operation
  - 1.1.4 Message
- 1.2 Designing class diagram
  - 1.2.1 Introduction
  - 1.2.2 UML Class Diagrams
  - 1.2.3 Relationships Between Classes
  - 1.2.4 Common Class Structures
  - 1.2.5 Modeling the Bright Fresh Milk System

1.2 Designing class diagram

目标：

1. 理解UML类图的含义
2. 理解类图中常用类结构
3. 掌握基于需求设计类图的方法

1.2.1 Introduction

- 根据某系统的需求，采用Java（或其它编程语言）编程语言开发一个软件系统，首先需要进行需求分析和设计。
  - 建筑物搭建时，需将需求、分析、设计、实现、布署等各项工作流程的构想与结果予以呈现。例如，建筑师会将其设计的建筑物画成图纸（即设计模型）；
  - 软件开发如同建筑物搭建，也需要设计模型，以呈现面向对象的设计思想。
    - 设计模型应遵循标准（**职业品格和行为习惯的养成**）：限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。

1.2.1 Introduction

- UML（统一建模语言，Unified Modeling Language）是为面向对象开发系统的产品进行说明、可视化和编制文档的方法，计算机行业标准协会OMG组织1997年采纳UML作为面向对象的需求分析、设计的标准建模语言，要在团队中开展面向对象软件的开发，必须掌握该建模语言。
- SysML(Systems Modeling Language)，系统建模语言，是UML在系统工程应用领域的延续和扩展，是近年提出的系统体系结构设计的多用途建模语言，为系统的结构模型、行为模型、需求模型和参数模型定义语法和语义。

1.2 Designing classes diagram

- ☐ 1.2.1 Introduction
- ✓ 1.2.2 UML Class Diagrams
- ☐ 1.2.3 Relationships Between Classes
- ☐ 1.2.4 Common Class Structures
- ☐ 1.2.5 Modeling the Bright Fresh Milk System

1.2.2 UML Class Diagrams(cont.introduction)

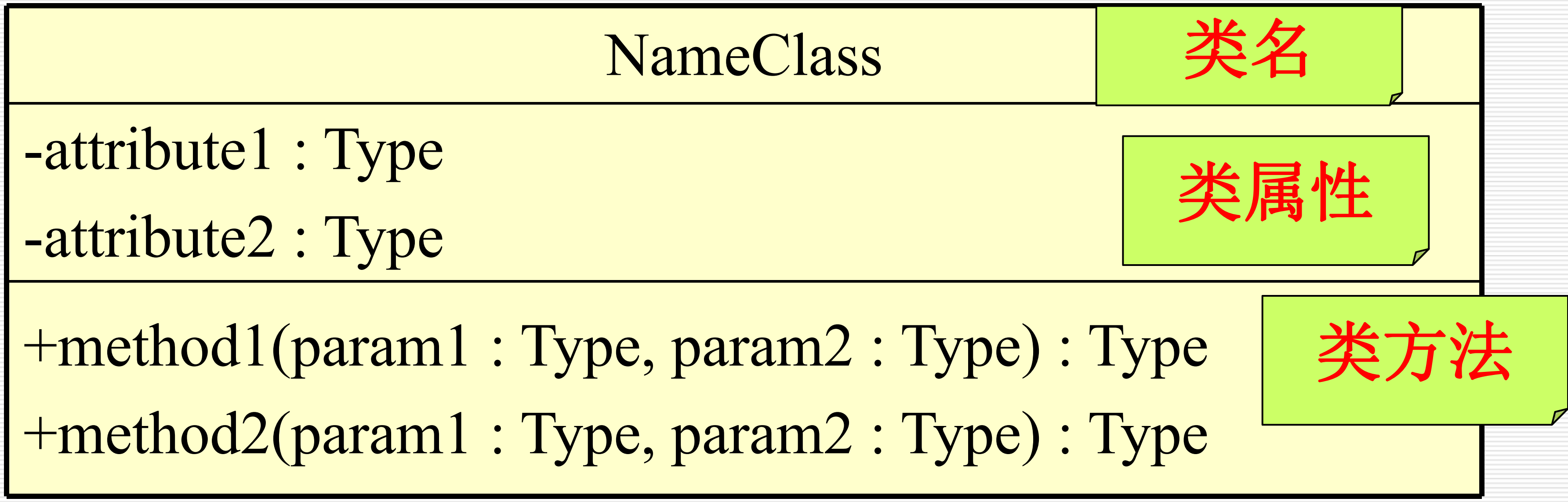
- Throughout this course, we will use class diagrams to discuss the diverse elements of object-oriented design.
  - UML introduction3-类图
  - UML for Java Programmers.pdf
- UML绘图工具：
  - eclipseUML
  - Violet.jar（Java开源建模软件）
  - Microsoft Visio
  - IBM RSA(Rational Rose)
  - ……

1.2.2 UML Class Diagrams(cont.introduction)

- Class diagrams:
  - 类图的设计也是面向对象分析和设计阶段的第一个最关键的步骤，系统动态行为的建模都以此为基础，可以在设计阶段对设计方案进行评估，尽早发现问题，尽早解决问题。
  - 通过对用户需求的分析，在软件开发的分析和设计阶段，通过建立类图描述软件系统的对象类型以及它们之间的关系，为进一步软件的编码实现提供足够的信息。
  - 开发人员、测试人员和维护人员通过类图，可以查看编码的详细信息，即软件系统的实现由哪些类构成，每个类有哪些属性和方法，以及类之间的源码依赖关系。

1.2.2 UML Class Diagrams(cont. Class Notation)

- A class is represented by a rectangle with three compartments.


NameClass
-attribute1 : Type
-attribute2 : Type
+method1(param1 : Type, param2 : Type) : Type
+method2(param1 : Type, param2 : Type) : Type
类名
类属性
类方法

- 找出customerDescription.doc中的某个类，并采用UML类的语法进行绘制

1.2.2 UML Class Diagrams(cont. Class Notation)

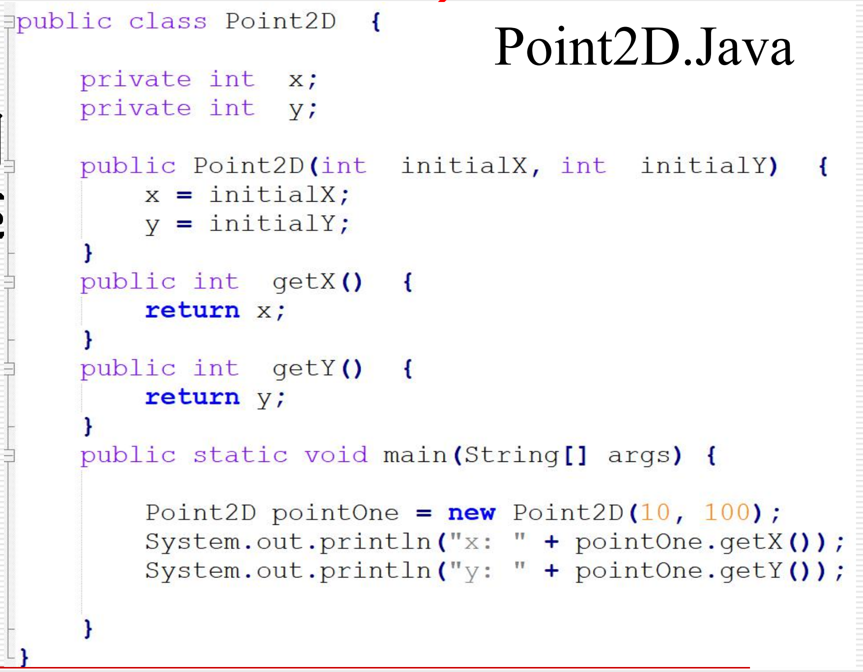
- 类的属性列表的每个属性信息占据一行，每行属性信息的格式如下：
  - **name：attribute type**
- 类的操作列表的每个操作的信息也单独占据一行，每行操作信息的格式如下：
  - **name(parameter list): type of value returned**
    - 参数列表中每个参数的格式如下：
      - **parameter name：parameter type**

1.2.2 UML Class Diagrams(cont. Class Notation)

- 类中成员访问权限表示
  - - indicates private visibility
  - + indicates public visibility
  - # indicates protected visibility
  - ~ indicates friendly visibility
- 在具体程序实现时，类可以用面向对象语言中的类结构描述

{C++} Java

Point2D.Java



```
public class Point2D  {

    private int  x;
    private int  y;

    public Point2D(int  initialX, int  initialY)  {
        x = initialX;
        y = initialY;
    }
    public int  getX()  {
        return x;
    }
    public int  getY()  {
        return y;
    }
    public static void main(String[] args) {

        Point2D pointOne = new Point2D(10, 100);
        System.out.println("x:  " + pointOne.getX());
        System.out.println("y:  " + pointOne.getY());

    }
}
```

1.2 Designing classes diagram

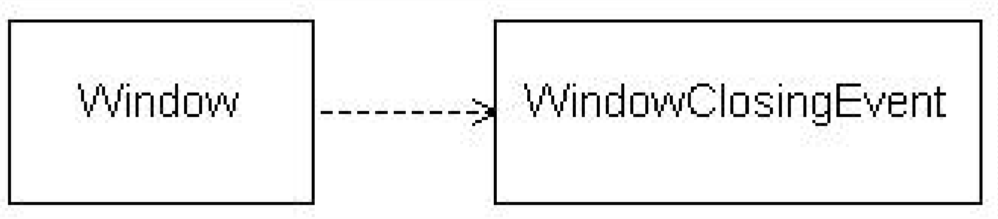
- ☐ 1.2.1 Introduction
- ☐ 1.2.2 UML Class Diagrams
- ✓ 1.2.3 Relationships Between Classes
- ☐ 1.2.4 Common Class Structures
- ☐ 1.2.5 Modeling the Bright Fresh Milk System

1.2.3 Relationships Between Classes

- 类之间关系是系统设计的关键，面向对象中定义的类之间关系包括：
  - 依赖关系、关联关系、聚合关系（基本聚合和组合聚合）和继承关系。
  - **关联关系**和**继承关系**是最重要和最常用的，一般设计的类图都要体现类之间的**关联关系**和**继承关系**。

1.2.3 Relationships Between Classes (cont.)

- **依赖关系**是类与类之间最弱的关系，是指一个类（依赖类）“使用”或“知道”另一外一个类（目标类）。它是一个典型的瞬时关系，依赖类和目标类进行简单的交互，但是，依赖类并不维护目标类的对象，仅仅是临时使用而已。
  - 例如对于窗体类Window，当它关闭时会发送一个类WindowClosingEvent对象，窗体类Window使用类WindowClosingEvent，它们之间的依赖的表示如下图所示：



of

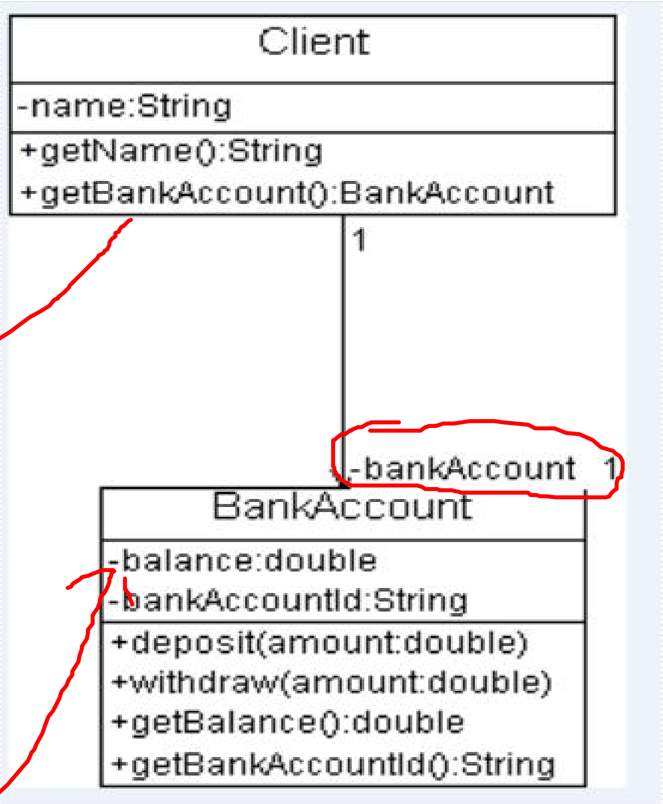
1.2.3 Relationships Between Classes (cont.)

for

- **Associations(关联关系)**
  - *Associations* are stronger than dependencies and typically indicate that one class retains a relationship to another class over an extended period of time. **Associations are typically read as "...has a...".**
    - **For example a Client has a BankAccount**

1.2.3 Relationships Between Classes (cont.)

- **One-way Associations**
  - The first class has a reference to an object of the second class, but the second class does not have a reference to an object of the first class.
    - UML indicates a one-way association with an arrow at the end of the association line. The attribute in the first class that contains a reference to an object of the second class is also written at the end of the line.

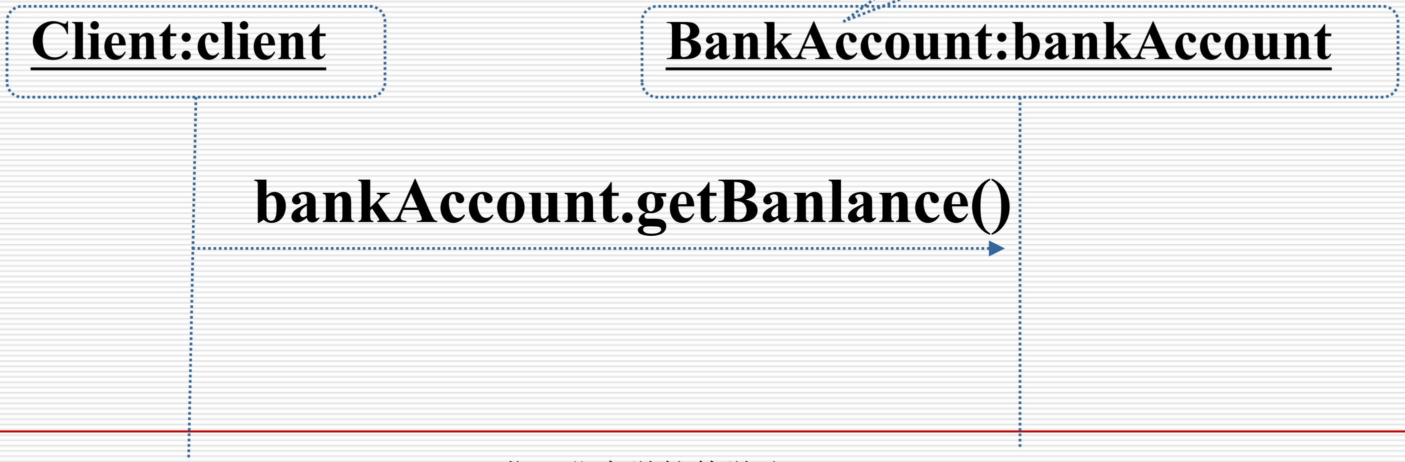


1.2.3 Relationships Between Classes(cont.)

```
public  class Client {
        private String name;
       private BankAccount bankAccount;
       …
}
```

Client能否实现 “返回姓名为张三的账户余额”的功能？ 请给出实现过程。

给对象发消息：通过调用某个对象的方法实现



1.2.3 Relationships Between Classes(cont.)

- 常用的关联数量的表示及其含义有以下几种：

具体数字n	比如1，比如6
* 或 0..*	0到任意多个
0..1	0个或1个
1..*	1到任意多个

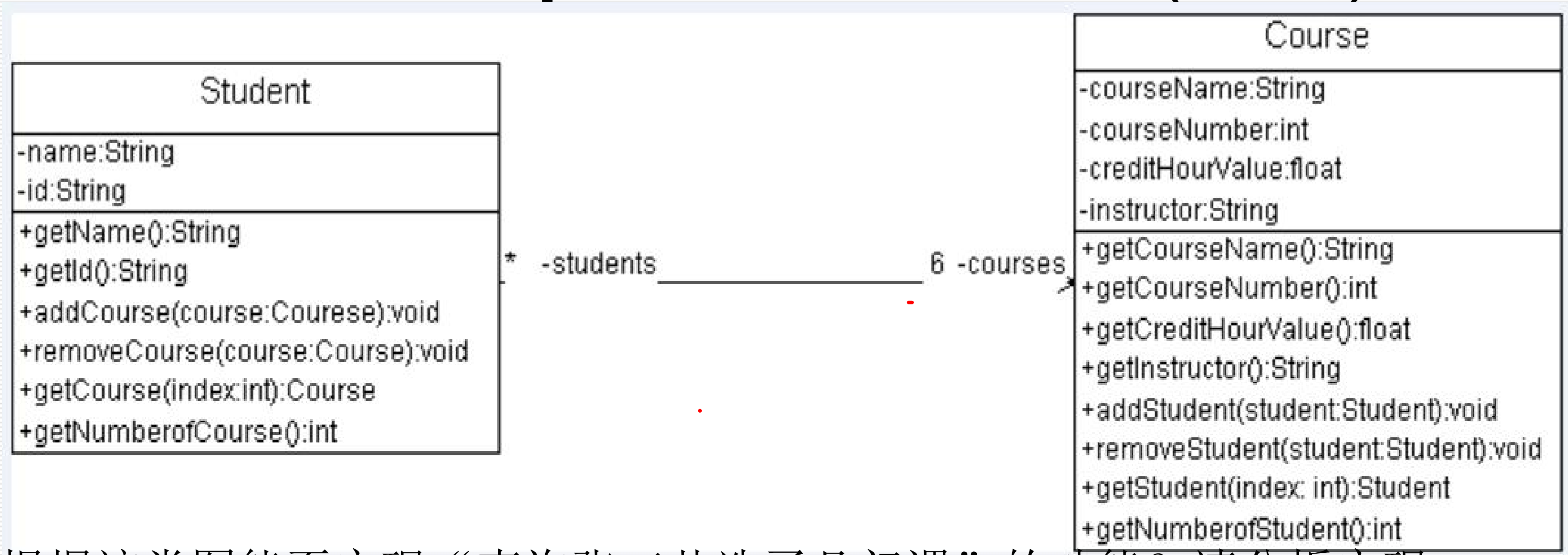
- 找出customerDescription.doc中的一对一的关联关系
  - customer-System-Core.png

1.2.3 Relationships Between Classes(cont.)

Two-way Associations

- In a two-way association, each class contains a reference to an object of the other class.
  - 在类图中用一条直线连接两个类表示它们之间的双向关联关系。
  - 例如，在一个需求描述中，一个学生可以选修6门课程，一门课程可以由若干学生选修。
    - 和类**Course**的一个对象关联的类**Student**对象的数量为**0**到任意多个，用“*”来表示
    - 和类**Student**的一个对象关联的类**Course**对象的数量为**6**

1.2.3 Relationships Between Classes(cont.)



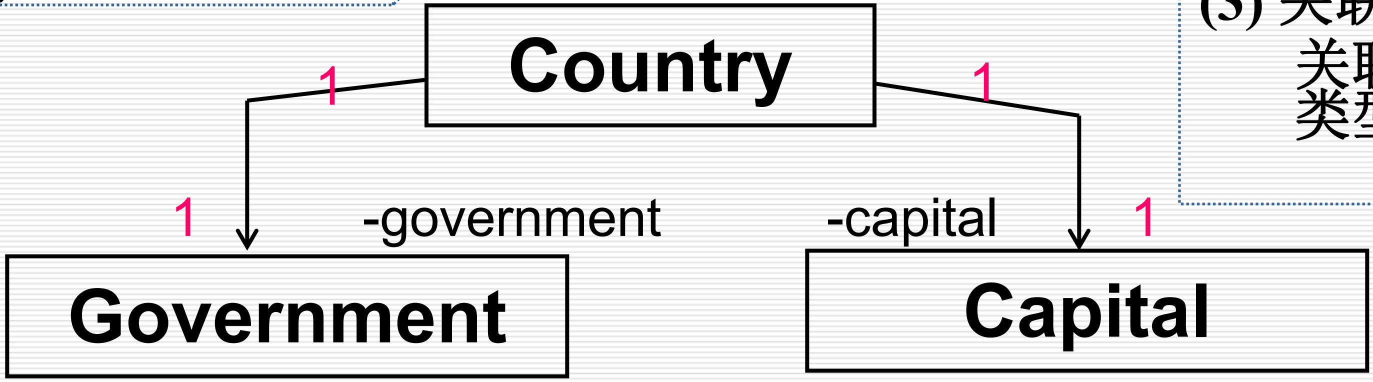
根据该类图能否实现“查询张三共选了几门课”的功能？请分析实现过程。

1. 假设有很多Student对象，例如，student1,student2,…
2. 给每个Student对象发消息getName，例如，studentX.getName()，找到其返回值是张三的对象studentX
3. 给student1对象发消息getNumberofCourse()，即student1. getNumberofCourse(), 返回所选课程的门数

1.2.3 Relationships Between Classes(cont.)

关联关系的三要素是？

(1) 关联的方向
(2) 关联的数量
(3) 关联的属性
关联属性的数据类型是？

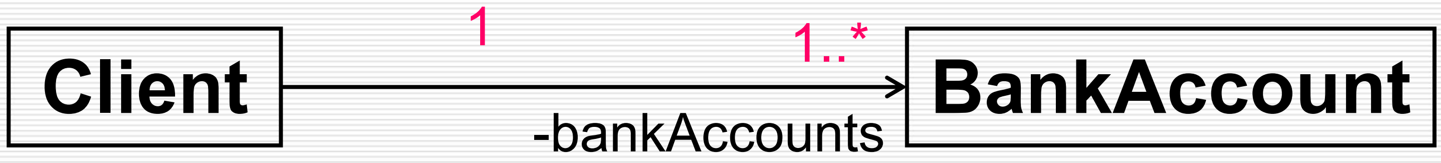


- ***One-to-one Association***
  - **In a one-to-one association, exactly one instance of each class is related with exactly one instance of the other class.**
  - 注：关联的属性是单数；

1.2.3 Relationships Between Classes(cont.)

```
// In  Java
public  class  Country{
        …
        private Government   government;
        private Capital  capital;
        …
}
```

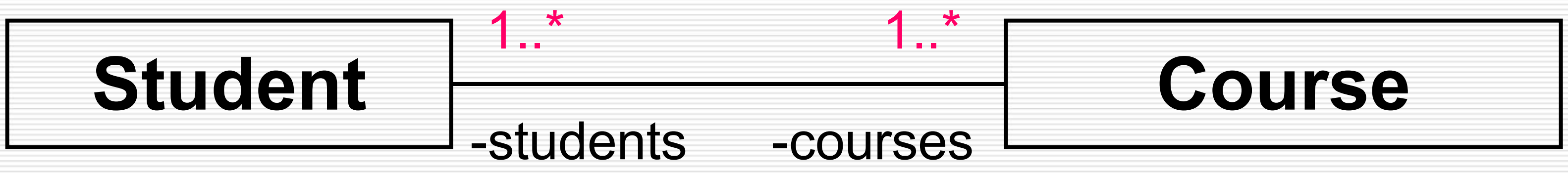
1.2.3 Relationships Between Classes(cont.)



(1) 关联的方向
(2) 关联的数量
(3) 关联的属性
关联属性的数据类型是？

- ***One-to-many Association***
  - **In a one-to-many association between classes A and B, one instance of class A may be related with many instances of class B, but one instance of class B is related with exactly one instance of class A.**
  - 注：关联的属性是复数

1.2.3 Relationships Between Classes(cont.)



- ***Many-to-many Association***
  - **In a many-to-many association between classes A and B, one instance of class A may be related with many instances of class B, and one instance of class B may be related with many instances of class A.**
  - 总结：类与类之间的关联关系

1.2.3 Relationships Between Classes(cont.)

- 找出customerDescription.doc中的一对多的关联关系
  - customer-System-Core.png

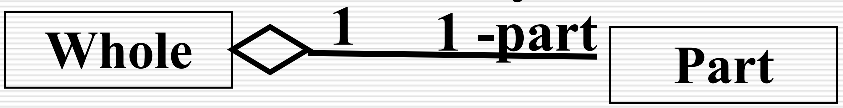
In Java or C++, one-to-many association is most commonly implemented with a Vector, a List, or some other container type.

For example

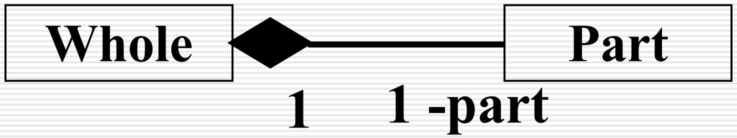
- Client.h Client.cpp
- Client.java

1.2.3 Relationships Between Classes(cont.)

- Aggregation is a special type of relationship used to model a "whole to its parts" relations.
  - Basic aggregation relationships
    - The lifecycle of a *part* class is independent from the *whole* class's lifecycle.



  - Composition aggregation relationships
    - The part class's instance lifecycle is dependent on the Whole class's instance lifecycle.

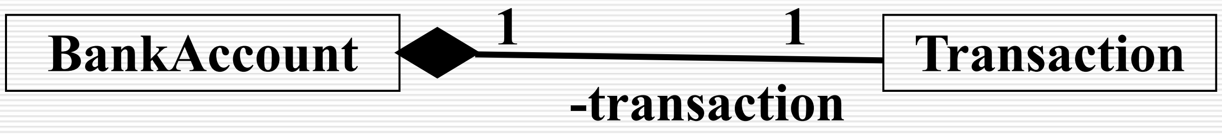


1.2.3 Relationships Between Classes(cont.)

- **Basic aggregation relationships**
  - **For example, we can think of *Car* as a whole entity and *Car Wheel* as part of the overall Car.**
    - **The wheel can be created weeks ahead of time,and it can sit in a warehouse before being placed on a car during assembly.**
    - **Car.png**

1.2.3 Relationships Between Classes(cont.)

- Composition aggregation relationships
  - For example, Both BankAccount and Transactions are modeled as classes, and a deposit or withdrawal transaction cannot occur before a bank account exists.
    - Transactions class's instance is dependent upon the Bank Account class's instance.when the BankAccount is emoved/destroyed, the Transaction instance is automatically removed/destroyed as well.



1.2.3 Relationships Between Classes(cont.)

- 聚合是关联的一种特殊形式，代表两个类之间的整体/局部关系，在概念上，整体处于比局部更高的一个级别，而关联暗示两个类在概念上位于相同的级别。
- **用Java编程实现时，整体类Whole包含一个局部类Part类型的属性变量：**

```
public  class Whole{
                private Part itsPart;
                ……
}
```

- 聚合还暗示着类图中不存在回路(即不存在**Self-Containing Classes**)，只能是一种单向关系。

1.2.3 Relationships Between Classes(cont.)

- **Specialization/Generalization(继承关系) represents the is-a relationship.**
- 由于现实世界中很多实体都有继承的含义，所以在软件建模中，将有继承含义的两个实体建模为由继承关系的两个类。举例：
  - **a client is-a person.**

1.2.3 Relationships Between Classes(cont.)

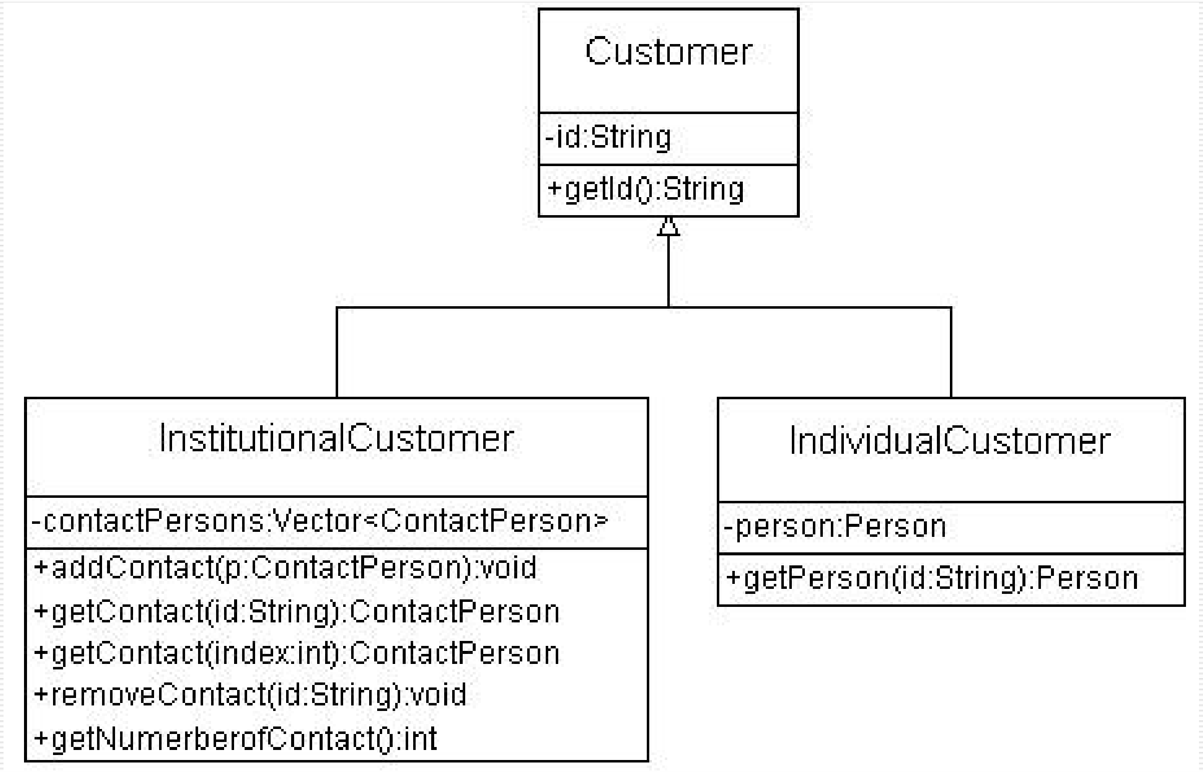
- 如果两个类有继承关系，被继承的类称为父类或超类，继承了父类或超类的所有数据和操作的类称为子类。子类自动拥有父类的所有数据和操作。
  - 子类可以在继承父类的基础上进行扩展，添加自己新的操作，子类也可以覆写父类中的操作，使得其操作的行为有别于父类。
  - 子类对象可以当作父类对象使用（即子类对象可以为父类型的对象变量赋值）。

1.2.3 Relationships Between Classes(cont.)

- 根据“is-a”和“继承的特征”找出customerDescription.doc中的继承关系
  - 如果既可以用关联也可以用继承，建议采用关联建模，除非用到多态（子类对象都作为父类对象进行统一访问的某些应用场景，**增进共同性，包容差异性，共同性和差异性的辩证统一**）的优势，才考虑用继承。

1.2.3 Relationships Between Classes(cont.)

- 在类图中，为了建模继承关系，从子类画一条实线引向父类，在线的末端画一个带空心的三角形指向父类。


Customer
-id:String
+getId():String
InstitutionalCustomer
-contactPersons:Vector<ContactPerson>
+addContact(p:ContactPerson):void
+getContact(id:String):ContactPerson
+getContact(index:int):ContactPerson
+removeContact(id:String):void
+getNumerberofContact():int
IndividualCustomer
-person:Person
+getPerson(id:String):Person

1.2 Designing classes diagram

- ☐ 1.2.1 Introduction
- ☐ 1.2.2 UML Class Diagrams
- ☐ 1.2.3 Relationships Between Classes
- ✓ 1.2.4 Common Class Structures
- ☐ 1.2.5 Modeling the Bright Fresh Milk System

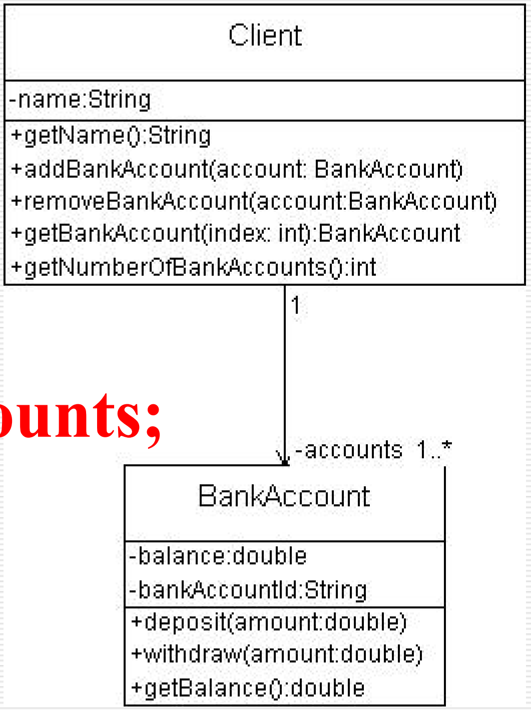
1.2.4 Common Class Structures

- 类和类之间的关系依据具体的应用而定；但是，有一些类结构在许多设计中被用到，这些常用的类结构被认为是基本的构建块，用以构建复杂的应用系统：
  - Collections
  - Self-Containing Classes
  - Relationship Loops

I. Collections

- A *collection* models a one-to-many relationship.
  - For example
  - public class Client {

```
        …
private Vector<BankAccount>  accounts;
        …
}
```

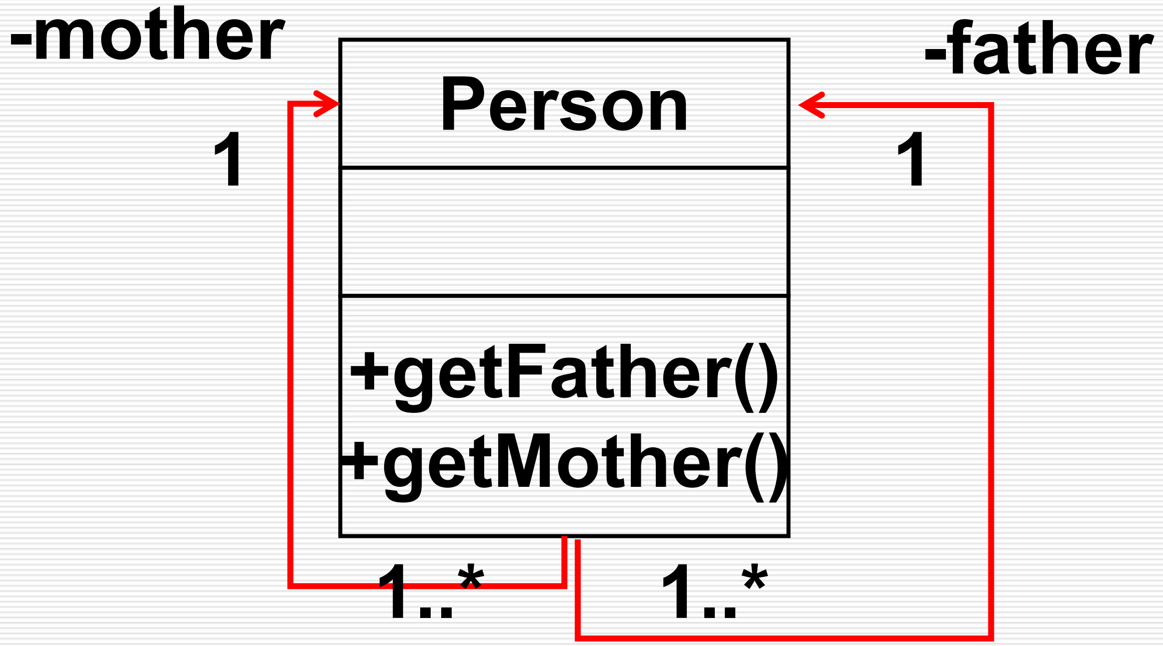


1. Collections(cont.)

- 包含容器的类提供的典型方法包括：
  - 将对象插入容器；
    - addBankAccount(account:BankAccount)
  - 将对象从容器中删除；
    - removeBankAccount(account:BankAccount)
  - 取得容器中的对象；
    - getBankAccount(index:int)
    - getBankAccount(id:String)
  - 返回容器中的对象个数；
    - getNumberOfBankAccounts():int
- 光大旅游服务接口示例

2. Self-Containing Classes

- A class can have an association with itself.
- In such cases, the class contains attributes with references to objects of the same class.



2. Self-Containing Classes (cont.)

```
public class Person {
        …
        private Person father;
        private Person mother;
       …
        public Person  getFather() {
               return father;
        }
        public Person  getMother() {
               return mother;
        }
      …
}
```

3. Relationship Loops (cont.)

- **For example 1:** A component system has composite components, and composite components contain atomic components or more composite components, or both. Each atomic component or composite component has name, type and creation date. In addition, each atomic component has functional description.
- 功能要求：
  - 1.可以打印指定原子构件的名字，类型，创建日期及功能描述
  - 2.可以打印指定复合构件内各原子构件或复合构件的相关信息
- **Class Diagram**
  - **Component-basic-v1.png**; **Component-basic-v2.png**

3. Relationship Loops (cont.)

- 在一个构件系统的应用中，根据需求规格说明需要创建两个类CompositeComponent和AtomicComponent;
  - 由于这两个类存在相同的属性name、date、type和操作getName( )、getDate( )、getType( )，为了便于代码复用，以及表述这两个类之间的共性，可以设计一个存放通用代码的类Component作为基类。
- 改进方案:
  - Component-v1.png, Component-v2.png
- **Self-containment can be found in relationships that include two or more classes.**

3. Relationship Loops (cont.)

- 类CompositeComponent和类AtomicComponent在继承类Component的基础上分别扩展了新操作。
- 总结：
  - 在面向对象的设计中，如果发现类B和类C存在同样的代码，可以设计一个类A，用于存放通用的代码，使得类B和类C继承类A，通过继承，类B和类C可以复用类A的代码。
  - 如果类A与类B所有子类关联，则类A仅需与基类关联即可，同时将类A中的属性变量/方法参数的类型设为基类，属性变量/方法形式参数可以接受从该基类导出的任何子类对象。

3. Relationship Loops (cont.)

- For example 2: Specification:
  - employee-salary.doc
- 两种设计方案的类图(Class diagram):
  - ✓ employee-salary.png
    - employee-salary-part0.png
    - employee-salary-part1.png
  - ✓ employee-salary1.png

1.2 Designing classes diagram

- ☐ 1.2.1 Introduction
- ☐ 1.2.2 UML Class Diagrams
- ☐ 1.2.3 Relationships Between Classes
- ☐ 1.2.4 Common Class Structures
- ✓ 1.2.5 Modeling the Bright Fresh Milk System

1.2.5 Model the Bright Fresh Milk System

- Object-oriented design is the process of building class diagram for a system to be developed.
  - Class diagram specifies the classes, their attributes and methods, and their relationships with other classes.

- **The steps to create class diagram from a system specification are the following:**
  - ① **Identify classes of objects from the system specification.**
  - ② **Identify relationships between classes.**
  - ③ **Identify attributes of each class.**
  - ④ **Identify methods of each class.**
  - ⑤ **Model system using UML.**

1. Specification of the the Bright Fresh Milk Systen

- For example milkSystemDescription.docx

a) First, we list the nouns/noun phrases in the system specification:
   - ✓ milkSystemSolution.xlsx

2. Identifying Classes(cont.)

b) Then, we prune the list

① **Convert plural nouns to their singular form. In an object model, class names are singular.**

② **Eliminate nouns that represent objects. Replace them with generic nouns.**

③ **Eliminate vague nouns.**

④ **Eliminate irrelevant nouns to the system.**

⑤ **Eliminate nouns that are a generic term for the attributes. (for example, information)**

⑥ **Eliminate nouns that are class attributes. If you cannot identify the attributes of the noun, then it is possible that the noun refers to a class attribute.**

⑦ **Eliminate references to the system itself**

2. Identifying Classes(cont.)

c) Next, we group the synonyms and then choose the best name for the class from each group;
   i. Names should be natural,conventional(using accepted terms from problem domain),and easy to understand and remember.
   ii. They should not be misleading.Using nouns for class names is recommended.

d) Finally, we select the classes that are relevant to the system. Look for more relevant classes. Search the application domain.

3. Identifying Relationships(cont.)

- **The steps for identifying the association and specialization/generalization**
- **Relationships are the following:**
  - ✓ **Create an *n x n* table where *n* is the number of classes. Label the rows and columns with the class names.**
  - ✓ **milkSystemSolution.xlsx**

3. Identifying Relationships(cont.)

b) **To identify the specialization/generalization relationships, for each cell in the row A and column B, ask the following questions:**

   i. **Is an instance of class *A* an instance of class *B*?**

   Mark the cell in the row *A* and column *B* with an *S*.

   ii. **Is an instance of class *B* an instance of class *A*?**

   Mark the cell in the row A and column B with a G.

3. Identifying Relationships(cont.)

- For example *a client is a person that has one or more accounts.*

	Person	Client	Account
Person	*X*	*G*	*X*
Client	*S*	*X*	accounts
Account	*X*	*X*	*X*

- milkSystemSolution.xlsx
- milkSystemDescription.docx

3. Identifying Relationships(cont.)

c) **To identify the association relationships, evaluate each cell in the row A and column B:**
   i. **If there is no association between class *A* and class *B*, mark the cell with an X.**
   ii. **If there are one or more associations between class *A* and class *B*, then insert the association attributes.**

3. Identifying Relationships(cont.)

- **For example** ***a client is a person that has one or more accounts.***

	Person	Client	Account
Person	*X*	*G*	*X*
Client	*S*	*X*	**accounts**
Account	*X*	*X*	*X*

- **milkSystemSolution.xlsx**
- **milkSystemDescription.docx**

4. Identifying attributes

- Attributes are the data that an object is responsible for knowing and maintaining.
- The steps for identifying the class attributes are the following:
  - Look for adjectives and possessive phrases such as *"the X of Y"* and *"Y's X"* in the system specification. For example, "number of the account" and "client's name."
  - Associated attribute between classes
  - Use your knowledge of the application domain to define the set of attributes needed for the system being developed.
- milkSystemSolution.xlsx

5. Identifying Methods

- **Methods describe the actions that an object is responsible for providing.**
  a) **The steps for identifying the class methods are the following:**
     - **To identify behaviors, look for verbs.**
       - **For example, the statement "*the client deposits money into the account*" indicates that class Account should define a method deposit.**

6. Identifying Methods(cont.)

b) Use your knowledge of the application domain to define the set of methods needed for the system being developed. Look in the application domain for actions that:
   i. Set and get values of attributes.
   ii. Output or display a result.
   iii. Perform calculations using an object's values.
   iv. …

c) If there are any collections held by the object, include the methods needed to add, remove, and access elements of these collections.

- milkSystemSolution.xlsx

7. Core class diagram of the the Milk system

- **Core class (of the the bright fresh milk system): ProductCatalog, Jelly, MilkDrink, Order, Product, PureMilk, SaleItem, StoreSales, Yogurt**

- **milkSystemSolution-core.png**

8. Driver class of the bright fresh milk system

- 要求给用户提供的功能：
  - displaySales()
  - addOrder()
  - removeProduct()
  - addProduct()
  - displayOrder()
  - displayProduct()
  - readProduct()
  - displayCatalog()

9.   Driver class of the bright fresh milk system(cont.)

- 要求软件给用户提供具体的功能；
- 需要一种方法来测试这些核心类，看是否满足用户需求？
  - 软件需要提供相应的驱动类(或测试类)
  - **Driver class** **(of the bright fresh milk system):**

    BrightFreshMilkSystem

10. Driver class of the bright fresh milk system(cont.)

- 解决方案： (…)
  - 与具体应用环境相关的操作的实现代码放入 Drive class
    - 便于代码的复用
    - 程序结构清晰，降低编码和维护的工作量。
  - milkSystemSolution.png
  - 能否满足需求？(绘制分析的示意图)

第一单元结束

1.2 Designing class diagram

您是否具备如下能力？

目标：

1. 采用UML类图对需求进行建模的能力
2. 分析UML类图设计方案是否可以实现用户所要求的功能？