一、OOA、OOD、OOP 介绍

OO

面向对象，全称是 `Object Oriented` 。是一套集编程思想,方法,原则,模式,解决方案等为一体的编程模式。

OOA

面向对象分析，全称是 Object Oriented Analysis。

分析阶段主要解决以下问题：

- 建立针对业务问题域的清晰视图
- 列出系统必须要完成的核心任务
- 针对问题域建立公共词汇表
- 列出针对此问题域的最佳解决方案

此阶段要解决的核心问题是"`what to do`?"

OOD

面向对象设计，全称是 Object Oriented Design。

设计阶段主要解决以下问题：

- 如何解决具体的业务问题
- 引入系统工作所需的各方面的支持元素
- 定义系统的实现策略

此阶段要解决的核心问题是"`How to do`?"

OOP

面向对象编程，全称是 Object Oriented Programming。

面向对象(Object-Orientation,简称 OO)是一种系统建模技术/编程思想。OOP 是按照 OO 的方法学来开发程序的编程方式。

总结

`OOAD`（Object Oriented Analysis Design，面向对象的分析和设计，面向对象分析与设计）是现代软件企业广为采用的一项有效技术。OOAD 方法要求在设计中要映射现实世界中指定问题域中的对象和实体，

例如：顾客、汽车和销售人员等。这就需要设计要尽可能地接近现实世界，即以最自然的方式表述实体。所以面向对象技术的优点即为能够构建与现实世界相对应的问题模型，并保持他们的结构、关系和行为为模式。

二、面向对象的特点

2.1、抽象

抽象就是将一些事物的共性和相似点抽离出来，并将这些属性归为一个类（`抽象类`），这个类只考虑这些事物的共性和相似之处，并且会忽略与当前业务和目标无关的那些方面，只将注意力集中在与当前目标有关的方面。

忽略掉一个对象或实体的细节而只关注其本质特征的过程。

2.2、封装

封装是为了隐藏内部实现细节，是保证软件部件具有优良的模块性的基础。封装的目标就是要实现软件部件“`高内聚，低耦合`”，防止程序之间的相互依赖性带来的变动影响。

目的:实现信息的隐藏

1）属性信息

2）行为的操作的信息

隐藏方法实现的细节。

代码重用。

2.3、继承

在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并可以加入若干新的内容，或修改原来的方法（Override，重写方法）使之更适合特殊的需要，这就是继承。

继承是子类自动共享父类数据和方法的机制，这是类之间的一种关系，提高了软件的可重用性和可扩展性。

is a 关系

1）子类可以继承父类的属性、方法

2）子类可以有自己的特性存在。

2.4、多态

多态是运行时刻接口匹配的对象相互替换的能力。指程序定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编译期并不确定，而是在程序运行期间才确定（称之为动态绑定），

即一个引用变量指向的是哪个类的实例对象，在编译期间并不确定，在运行阶段才能决定，因此，这样就可以使得引用变量绑定到各种不同的类实现上，从而实现不同的行为。多态性增强了软件的灵活性和扩展性。

多态：相同类域的不同对象调用相同方法时的不同表现形式

```Java
Personp1=newStudent();

Personp2=newTeacher();

p1.work();//听课

p2.work();//讲课
```

1）子类继承父类,类实现接口

2）子类重写父类的同名方法,类实现接口中的抽象方法

3）父类的引用指向子类的对象,接口类型的引用指向具体实现类的对象

重写/重载

编译时多态：重载

运行时多态：重写

三、UML 类图及类图的关系

统一建模语言简介

统一建模语言（`Unified Modeling Language，UML`）是用来设计软件蓝图的可视化建模语言，1997 年被国际对象管理组织（OMG）采纳为面向对象的建模语言的国际标准。它的特点是简单、统一、图形化、能表达软件设计中的动态与静态信息。

统一建模语言能为软件开发的所有阶段提供模型化和可视化支持。而且融入了软件工程领域的新思想、新方法和新技术，使软件设计人员沟通更简明，进一步缩短了设计时间，减少开发成本。它的应用领域很宽，不仅适合于一般系统的开发，而且适合于并行与分布式系统的建模。

UML 从目标系统的不同角度出发，定义了用例图、`类图`、对象图、状态图、活动图、时序图、协作图、构件图、部署图等 9 种图。

本部分主要介绍软件设计模式中经常用到的**类图以及类之间的关系**。

类、接口和类图

1. 类

> 类（Class）是指具有相同属性、方法和关系的对象的抽象，它封装了数据和行为，是面向对象程序设计（OOP）的基础，具有`封装性、继承性和多态性`等三大特性。

在 UML 中，**类使用**包含类名、属性和操作且带有分隔线的**矩形来表示**。

1. 类名（Name）是一个字符串，例如，Student。
2. 属性（Attribute）是指类的特性，即类的成员变量。UML 按以下格式表示：

```
[可见性]属性名:类型[=默认值]
```

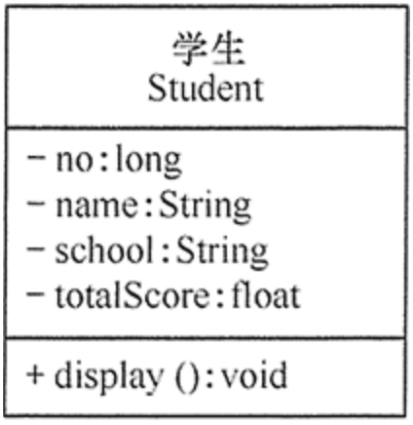
> 例如：–name:String

注意：“可见性”表示该属性对类外的元素是否可见，包括公有（Public）、私有（Private）、受保护（Protected）和朋友（Friendly）4 种，在类图中分别用符号+、–、#、~ 表示。

1. 操作（Operations）是类的任意一个实例对象都可以使用的行为，是类的成员方法。UML 按以下格式表示：

```
[可见性]名称(参数列表)[:返回类型]
```

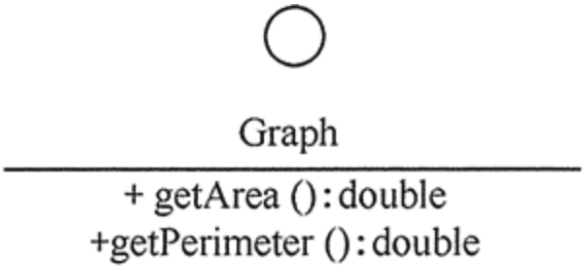
> 例如：+display():void。



2. 接口

> 接口（Interface）是一种特殊的类，它具有类的结构但不可被实例化，只可以被子类实现。它包含抽象操作，但不包含属性。它描述了类或组件对外可见的动作。

在 UML 中，**接口使用**一个带有名称的**小圆圈来进行表示**。

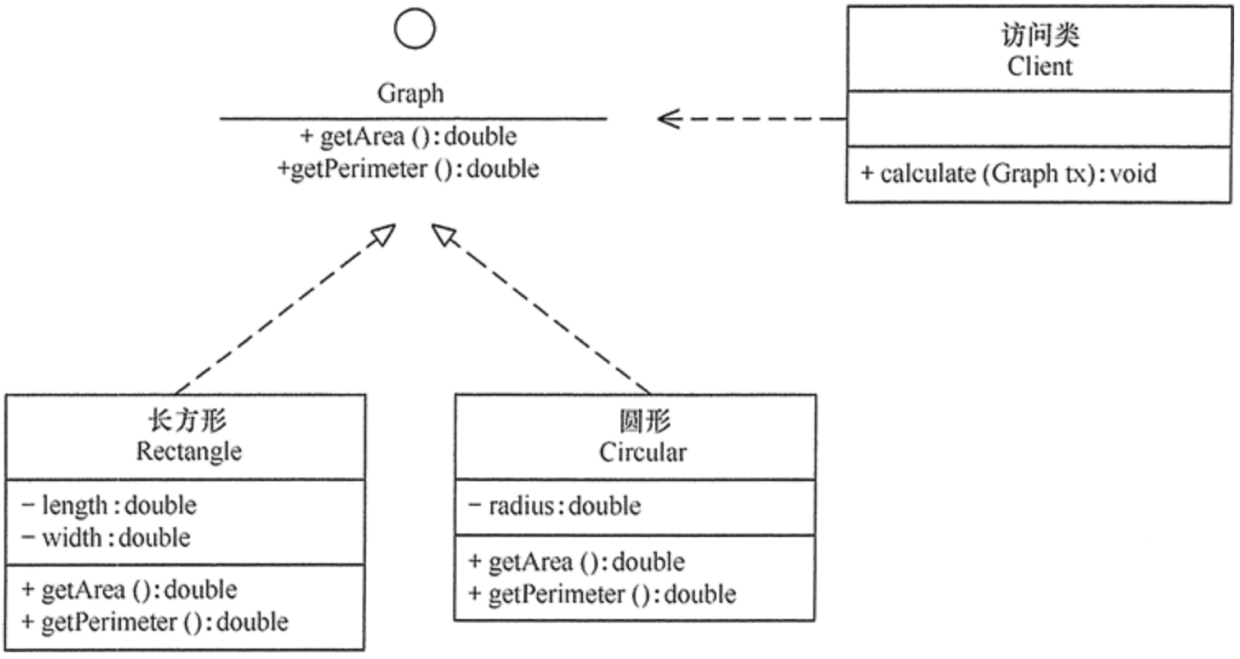


3. 类图

类图（ClassDiagram）是用来显示系统中的类、接口、协作以及它们之间的静态结构和关系的一种静态模型。它主要用于描述软件系统的结构化设计，帮助人们简化对软件系统的理解，它是系统分析与设计阶段的重要产物，也是系统编码与测试的重要模型依据。

类图中的类可以通过某种编程语言直接实现。类图在软件系统开发的整个生命周期都是有效的，它是面向对象系统的建模中最常见的图。

> 下图所示是“计算长方形和圆形的周长与面积”的类图，图形接口有计算面积和周长的抽象方法，长方形和圆形实现这两个方法供访问类调用。



类之间的关系

在软件系统中，类不是孤立存在的，类与类之间存在各种关系。

根据类与类之间的耦合度从弱到强排列，UML 中的类图有以下几种关系：依赖关系、关联关系、聚合关系、组合关系、泛化关系和实现关系。其中泛化和实现的耦合度相等，它们是最强的。

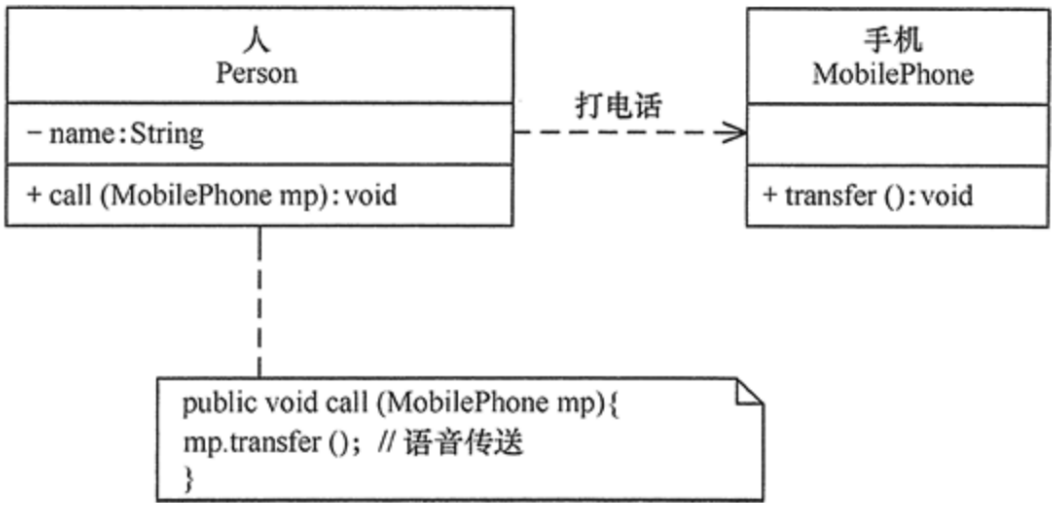
1. 依赖关系

依赖（Dependency）关系是一种使用关系，它是对象之间耦合度最弱的一种关联方式，是临时性的关联。

在代码中，某个类的方法通过**【局部变量、方法的参数或者对静态方法】**的调用来访问另一个类（被依赖类）中的某些方法来完成一些职责。

在 UML 类图中，依赖关系使用**带箭头的虚线**来表示，箭头从使用类指向被依赖的类。

> 下图所示是人与手机的关系图，人通过手机的语音传送方法打电话。



2. 关联关系

关联（Association）关系是对象之间的一种引用关系，用于表示一类对象与另一类对象之间的联系，如老师和学生、师傅和徒弟、丈夫和妻子等。

关联关系是类与类之间最常用的一种关系，分为一般关联关系、聚合关系和组合关系。我们先介绍一般关联。

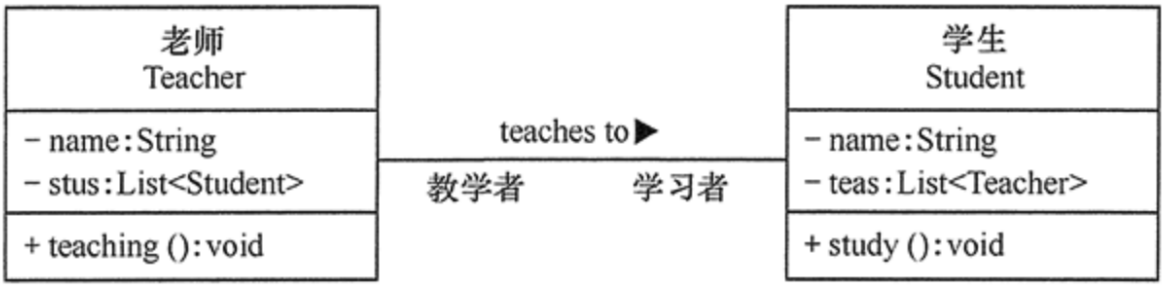
一般关联

关联可以是双向的，也可以是单向的。

在 UML 类图中，双向的关联可以用**`带两个箭头或者没有箭头的实线`**来表示，**单向的关联用**带一个箭头的实线**来表示，箭头从使用类指向被关联的类。也可以在关联线的两端标注角色名，代表两种不同的角色。

在代码中通常将一个类的对象作为另一个类的`【成员变量】`****来实现关联关系。**

> 下图所示是老师和学生的关系图，每个老师可以教多个学生，每个学生也可向多个老师学，他们是双向关联。



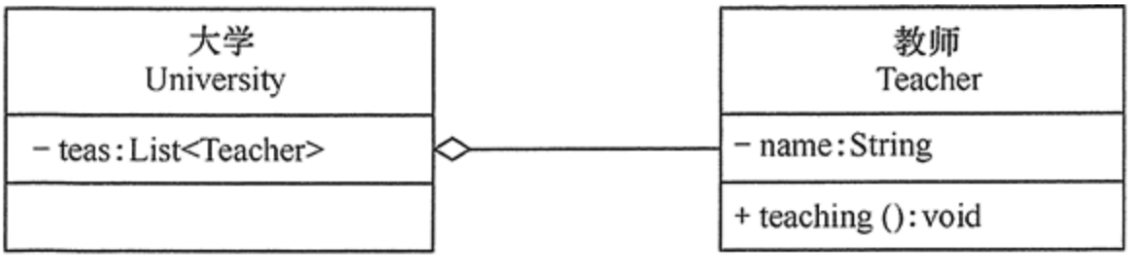
聚合关系（has a）

聚合（Aggregation）关系是关联关系的一种，是强关联关系，是整体和部分之间的关系，是 `has-a` 的关系。

聚合关系也是通过成员对象来实现的，其中成员对象是整体对象的一部分，但是成员对象可以脱离整体对象而独立存在。例如，学校与老师的关系，学校包含老师，但如果学校停办了，老师依然存在。

在 UML 类图中，聚合关系可以用**`带空心菱形的实线`**来表示，菱形指向整体。

> 下图所示是大学和教师的关系图。



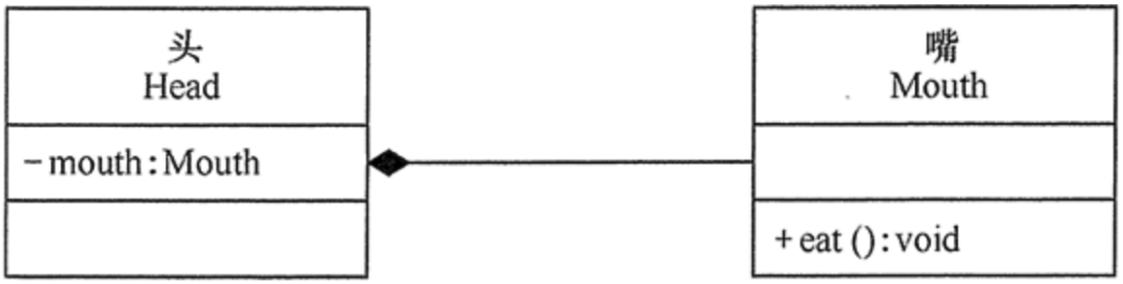
组合关系

组合（Composition）关系也是关联关系的一种，也表示类之间的整体与部分的关系，但它是一种更强烈的聚合关系，是 `contains-a` 关系。

在组合关系中，整体对象可以控制部分对象的生命周期，一旦整体对象不存在，部分对象也将不存在，部分对象不能脱离整体对象而存在。例如，头和嘴的关系，没有了头，嘴也就不存在了。

在 UML 类图中，组合关系用** `带实心菱形的实线` **来表示，菱形指向整体。

> 下图所示是头和嘴的关系图。

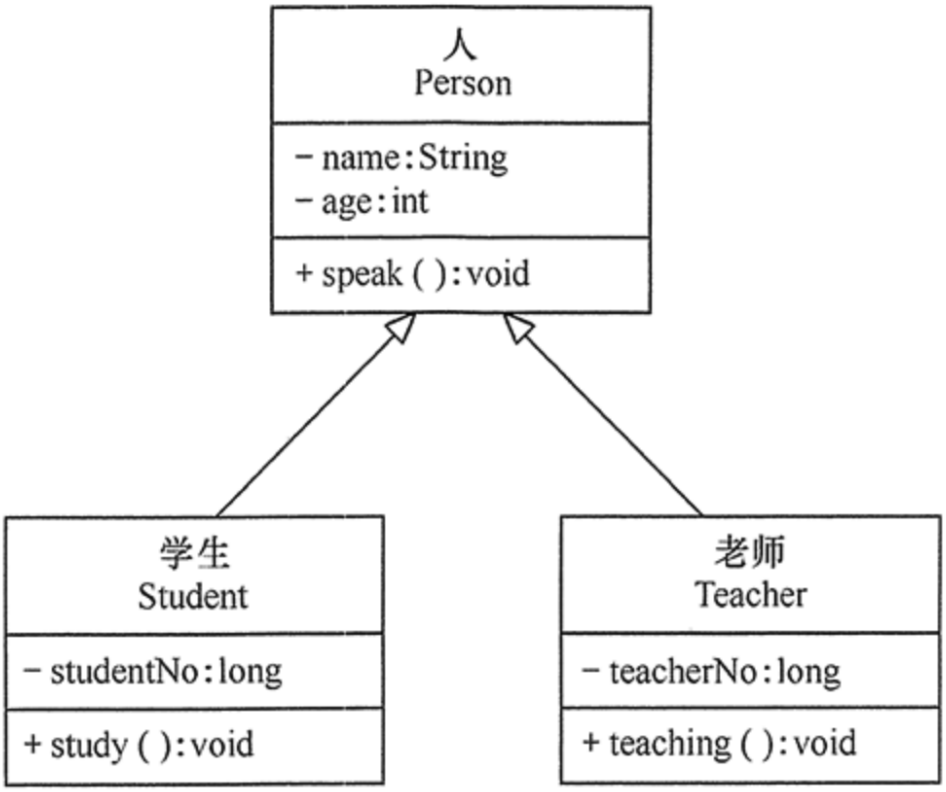


3.泛化关系（继承 is a）

泛化（Generalization）关系是对象之间耦合度最大的一种关系，表示一般与特殊的关系，是父类与子类之间的关系，是一种继承关系，是 `is-a` 的关系。

在 UML 类图中，泛化关系用** `带空心三角箭头的实线` **来表示，箭头从子类指向父类。在代码实现时，使用面向对象的继承机制来实现泛化关系。

> 例如，Student 类和 Teacher 类都是 Person 类的子类，其类图如下图所示。

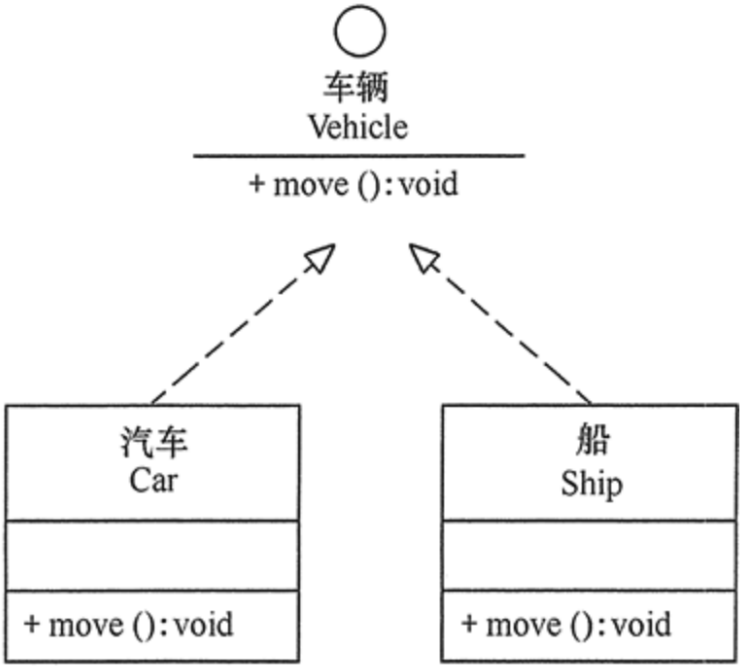


4.实现关系（接口）

实现（Realization）关系是接口与实现类之间的关系。在这种关系中，类实现了接口，类中的操作实现了接口中所声明的所有的抽象操作。

在 UML 类图中，实现关系使用**带空心三角箭头的虚线**来表示，箭头从实现类指向接口。

> 例如，汽车和船实现了交通工具，其类图如下图所示。



总结

类与类之间的关系

1. 依赖关系（局部变量、形参、静态方法的调用）
2. 关联关系（成员变量）
    a. 一般关联关系
    b. 聚合关系（has – a）
    c. 组合关系（contains–a）
3. 继承关系（父类和子类的关系）is – a `extends`
4. 实现关系（接口和实现类） `implements`