

课程目标

- 1、通过重新认识 UML，掌握 UML 的定义、特点和分类。
- 2、重点介绍类图和时序图各图例的含义。
- 3、掌握各种图例和标线的记忆技巧。

内容定位

适合所有的 Java 开发人员。

什么是 UML？

UML 的定义

统一建模语言（Unified Modeling Language，UML）是一种为面向对象系统的产品进行说明、可视化和编制文档的一种标准语言，是非专利的第三代建模和规约语言。

UML 使用面向对象设计的建模工具，但独立于任何具体程序设计语言。

UML 应用场景

UML 是在开发阶段，说明、可视化、构建和书写一个面向对象软件密集系统的制品的开放方法。最佳的应用是工程实践，对大规模，复杂系统进行建模方面，特别是在软件架构层次，已经被验证有效。统一建模语言（UML）是一种模型化语言。模型大多以图表的方式表现出来。一份典型的建模图表通常包含几个块或框，连接线和作为模型附加信息之用的文本。这些虽简单却非常重要，在 UML 规则中相互联系和扩展。

UML 的目标是以面向对象图的方式来描述任何类型的系统，具有很宽的应用领域。其中最常用的是建立软件系统的模型，但它同样可以用于描述非软件领域的系统，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。总之，UML 是一个通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行建模，而且适用于系统开发的不同阶段，从需求规格描述直至系统完成后的测试和维护。

UML2.0 一共有 13 种图形（UML1.5 定义了 9 种，2.0 增加了 4 种）。分别是：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 9 种，包图、时间图、组合结构图、交互概览图 4 种。

用例图（Usecase Diagrams）：用来描述用户的需求，从用户的角度描述系统的功能，并指出各功能的执行者，强调谁在使用系统，系统为执行者完成哪些功能。

类图（Class Diagrams）：用于定义系统中的类。

对象图（Object Diagrams）：是类图的一个实例，描述了系统在具体时间点上所包含的对象以及各个对象之间的关系。

构件图（Component Diagrams）：一种特殊的 UML 图来描述系统的静态实现视图。

部署图（Deployment Diagrams）：定义系统中软硬件的物理体系结构。

状态图（State Chart Diagrams）：用来描述类的对象所有可能的状态以及时间发生时状态的转移条件。

协作图（Collaboration Diagrams）：描述对象之间的合作关系，更侧重和用户对象说明哪些对象有消息的传递。

活动图（Activity Diagrams）：用来描述满足用例要求所要进行的活动以及活动间的约束关系。

时序图 (Sequence Diagrams)：描述对象之间的交互顺序，着重体现对象间消息传递的时间顺序，强调对象之间消息的发送顺序，同时显示对象之间的交互过程。

包图 (Package Diagrams)：对构成系统的模型元素进行分组整理的图。

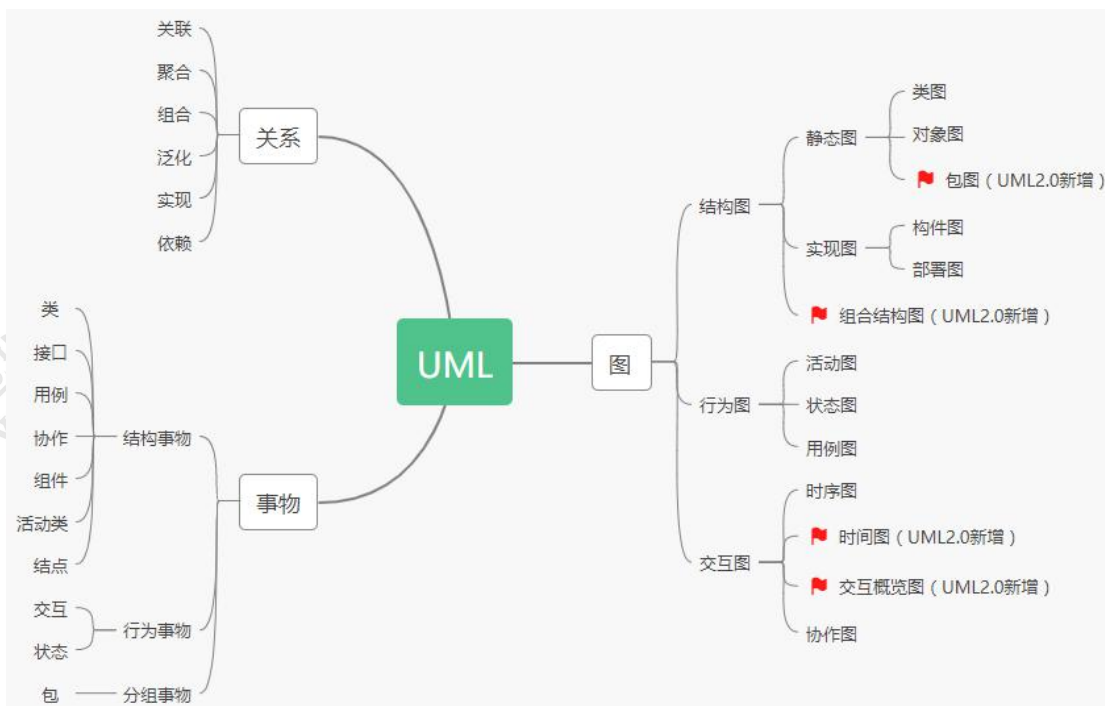
组合结构图 (Composite Structure Diagrams)：表示类或者构建内部结构的图。

时间图 (Timing Diagrams)：用来显示随时间变化，一个或多个元素的值或状态的更改，也显示时控事件之间的交互和管理它们的时间和期限约束。

交互概览图 (Interaction Overview Diagrams)：用活动图来表示多个交互之间的控制关系的图。

UML 绘图规范

建模的核心是模型，模型是现实的简化，模型是真实系统的左营，它提供了系统的设计蓝图。给软件系统建模是，需要采用通用的符号语言，这种描述模型所使用的语言称为建模语言。下图完整地描述了 UML 所能描述的所有事物和关系。



事物描述

类 (Class)：对一组具有相同属性，方法，关系和语义的对象的描述。



接口 (Interface)：描述了一个类或构件的一个服务的操作集。它仅仅是定义了一组操作的规范，并没有给出这组操作的具体实现（代码）。

用例 (Usecase)：是对一组动作序列的描述，系统执行这些动作将产生一个对特定的参与者 (Actor) 有价值且可观察的结果。

包 (Package)：是分组事物中最主要的，是 UML 中唯一的组织机制。

事物关系描述

关联 (Association)：是一种拥有的关系，具有方向性，如果一个类单方向的访问另一个类，则称为单向关联（用一个箭头的实线表示）；如果两个类对象可以互相访问，则称为双向关联（用两个箭头或不用箭头的实线表示）；一个对象能访问关联对象的数目叫做“多重性”。

用带普通箭头的实线表示 ，箭头指向被拥有者，或不用箭头的实线表示 。


聚合 (Aggregate)：是整体与部分的关系。当某个实体聚合成另一个实体时，该实体还可以是另一个实体的部分。

用带空心菱形的实线表示 ，菱形指向整体，箭头指向个体。

组合 (Combination)：整体与部分的关系，组合比聚合更加严格，当某个实体组合成另一个实体时，二者具有相同的生命周期，例如手臂和人之间存在的是组合关系。

用带实心菱形的实线表示 ，菱形指向整体，箭头指向个体。


泛化 (Generalization)：表示一个更泛化的元素和一个更具体的元素之间的关系，与继承是同一个概念。

用带三角箭头的实线表示 ，箭头指向父类。

实现 (Realization)：类与接口的关系，类实现接口。

用带三角箭头的虚线表示 ，箭头指向父接口。

依赖 (Dependency)：如果一个类的改动会影响到另一个类，则两个类之间存在依赖关系，一般而言，依赖是单向的。

用带普通箭头的虚线表示 ，箭头指向被依赖者。

类图 (Class Diagrams)

在 UML 2.0 的 13 种图形中，类图是使用频率最高的 UML 图之一。类图是描述系统中的类，以及各个类之间的关系的静态视图。能够让我们在正确编写代码以前对系统有一个全面的认识。类图是一种模型类型，确切的说，是一种静态模型类型。类图表示类、接口和它们之间的协作关系，用于系统设计阶段。

类图用三个矩形表示，最上面的部分标识类的名称；中间的部分标识类的属性；最下面的部分标识类的方法，如下图所示：

+ 表示 public

- 表示 private

表示 protected

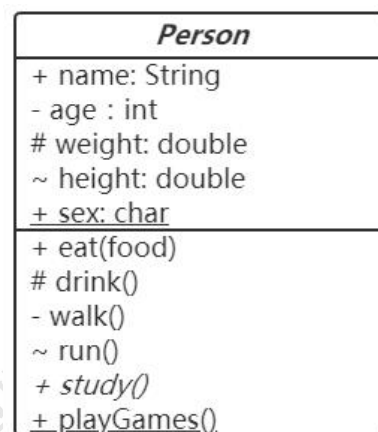
~ 表示 default，可省略不写。

字段和方法返回值的数据类型非必须。

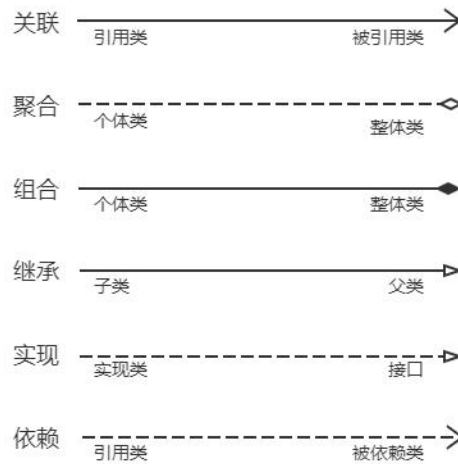
抽象类或抽象方法用斜体表示。

静态类或静态方法加下划线。

如果是接口在类名上方加 <<Interface>>。



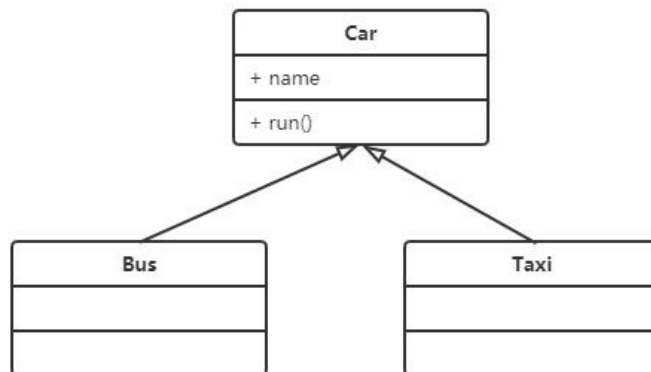
类与类之间的关系主要有六种：关联、聚合、组合、继承、实现和依赖，这六种关系的箭头表示如下：



接着我们来了解类关系的具体内容。

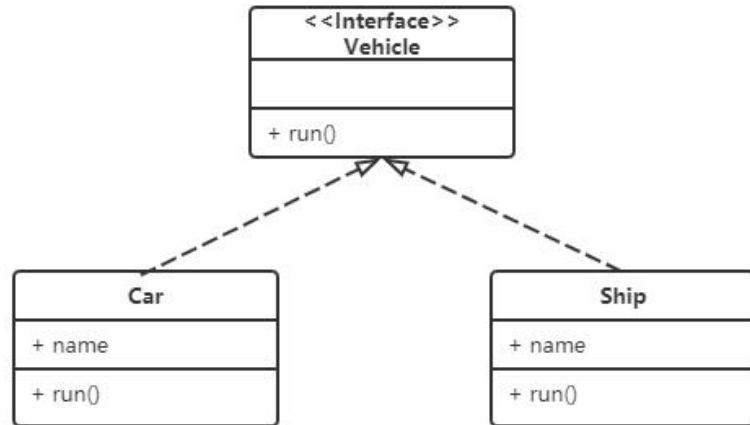
继承关系

继承关系中，子类继承父类的所有功能，父类所具有的属性、方法，子类应该都有。子类中除了与父类一致的信息以外，还包括额外的信息。例如：公交车、出租车和小轿车都是汽车，他们都有名称，并且都能在路上行驶。其类图如下：



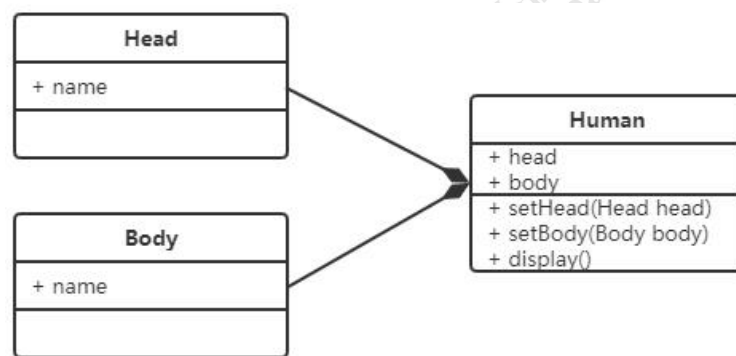
实现关系

接口（包括抽象类）是方法的集合，在实现关系中，类实现了接口，类中的方法实现了接口声明的所有方法。例如：汽车和轮船都是交通工具，而交通工具只是一个可移动工具的抽象概念，船和车实现了具体移动的功能。



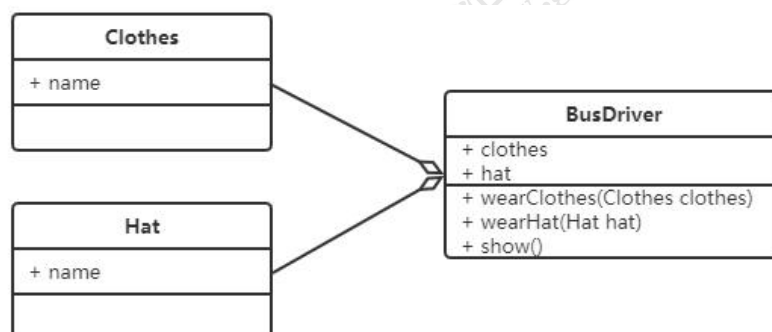
组合关系

组合关系表示类之间整体与部分的关系，整体和部分有一致的生存期。一旦整体对象不存在，部分对象也将不存在，是同生共死的关系。例如：人由头部和身体组成，两者不可分割，共同存在。



聚合关系

聚合关系也表示类之间整体与部分的关系，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独立存在。例如：公交车司机和工衣、工帽是整体与部分的关系，但是可以分开，工衣、工帽可以穿在别的司机身上，公交司机也可以穿别的工衣、工帽。



关联关系

关联关系是类与类之间最常用的一种关系，表示一类对象与另一类对象之间有联系。组合、聚合也属于关联关系，只是关联关系的类间关系比其他两种要弱。

关联关系有四种：双向关联、单向关联、自关联、多重数关联。例如：汽车和司机，一辆汽车对应特定的司机，一个司机也可以开多辆车。



在多重性关系中，可以直接在关联直线上增加一个数字，表示与之对应的另一个类的对象的个数。

1..1：仅一个

0..*：零个或多个

1..*：一个或多个

0..1：没有或只有一个

m..n：最少 m、最多 n 个 ($m \leq n$)

依赖关系

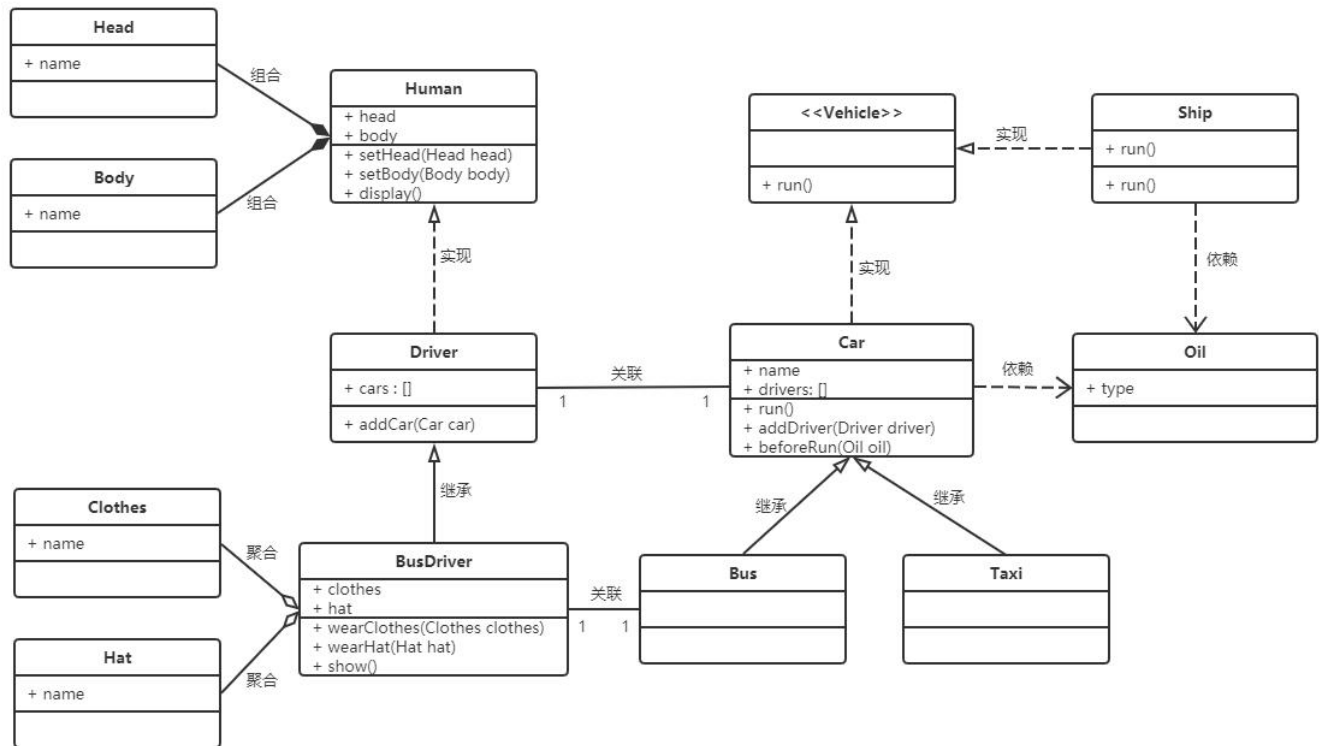
大多数情况下，依赖关系体现在某个类的方法使用另一个类的对象作为参数。

依赖关系是一种“使用”关系，特定事物的改变有可能会影响到使用该事物的其他事物，在需要表示一个事物使用另一个事物时使用依赖关系。

例如：汽车依赖汽油，如果没有汽油，汽车将无法行驶。



这六种类关系中，组合、聚合和关联的代码结构一样，可以从关系的强弱来理解，各类关系从强到弱依次是：继承→实现→组合→聚合→关联→依赖。



UML 类图是面向对象设计的辅助工具，但并非必须工具，所以我们把它作为架构师软技能来讲解。

类关系记忆技巧

箭头方向：从子类指向父类。

记忆技巧：1、定义子类是需要通过 extends 关键字指定父类；

2、子类一定是知道父类定义的，但父类并不知道子类的定义；

3、只有知道对方信息时才能指向对方；

4、所以箭头的方向是从子类指向父类。

继承实现：用线条连接两个类。

记忆技巧：1、空心三角箭头表示继承或实现。

2、实线表示继承，是 is-a 的关系，表示扩展，不虚，很结实；

3、虚线表示实现，虚线代表“虚”无实体。

关联依赖：用线条连接两个类。

记忆技巧：1、虚线表示依赖关系：临时用一下，若即若离，虚无缥缈，若有若无；

表示一种使用关系，一个类需要借助另一类来实现功能；

一般是一个类将另一个类作为参数使用，或作为返回值。

2、实线表示关联关系：关系稳定，实打实的关系，铁哥们；

表示一个类对象和另一个类对象有关联；

通常是一个类中有另一个类对象作为属性。

组合聚合：用菱形表示。

记忆技巧：1、菱形就是像是一个盛东西的器皿（比如盘子）；

2、聚合：空心菱形，代表空器皿里可以放很多相同的东西，

聚集在一起（箭头方向所指的类）；

整体和局部的关系，两者有独立的生命周期，是 has-a 的关系；

弱关系，消极的词：弱-空。

3、组合：实心菱形，代表器皿里已经有实体结构的存在，生死与共；

整体与局部的关系，和聚合关系对比，关系更加强烈；

两者具有相同的生命周期，contains-a 的关系；

强关系，积极的词：强-满。

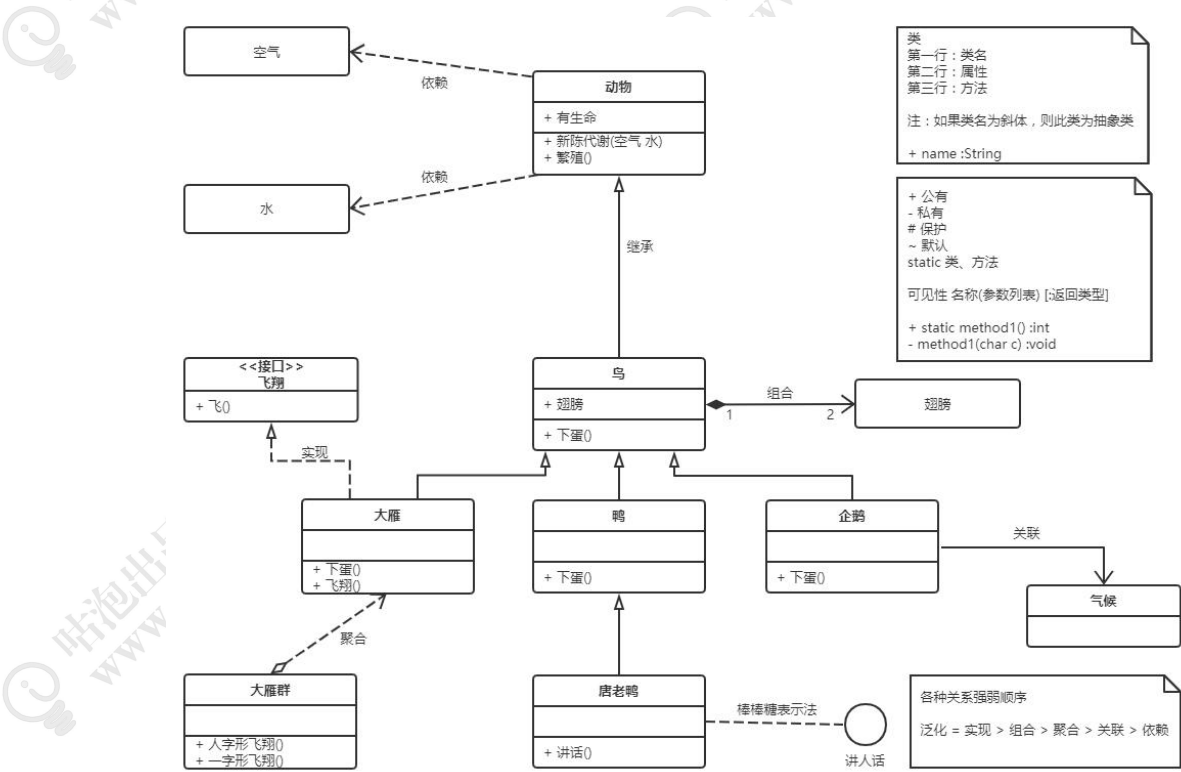
注意：UML 类关系图中，没有实心箭头。

案例分析

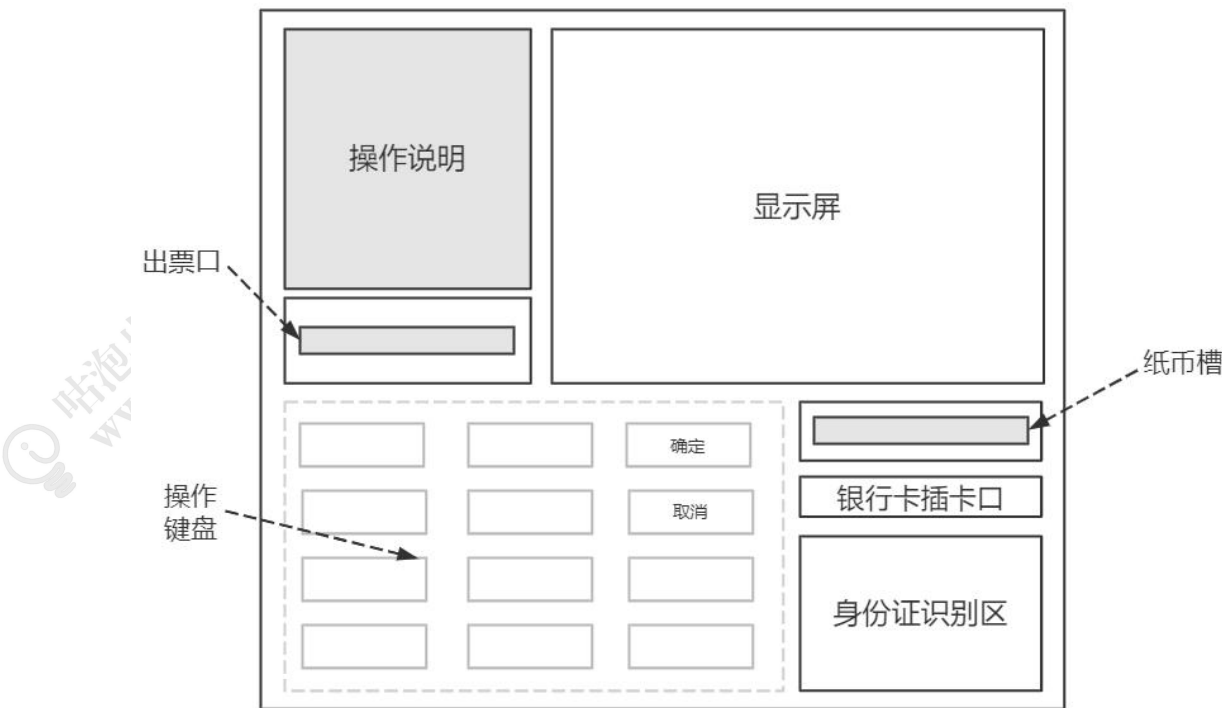
下面我们来通过两个经典案例来加深和巩固一下对类图的理解。

生活案例之动物衍生关系图

下面这张图是来自《大话设计模式》这本书中，对动物衍生关系描述类图。这个图非常有技术含量也非常经典，大家可以和我一起来好好理解一下。



实战案例之 12306 自动售票机类关系图



售票机面板相关部件的作用如下所述：

- (1) 在身份证识别区放入二代身份证。
- (2) 显示屏显示所有的车次、车票种类（单程票、多次往返票和座席种类）。
- (3) 乘客选择车次和座位类型确定下单。
- (4) 继续/取消键盘上的取消按钮用于取消购票过程，继续按钮允许乘客连续购买多张票。
- (5) 插卡口接受 PayCard（银行卡）和纸币槽接受现金。
- (6) 打印机用于输出车票。
- (7) 所有部件均可实现自检并恢复到初始状态。

类说明：

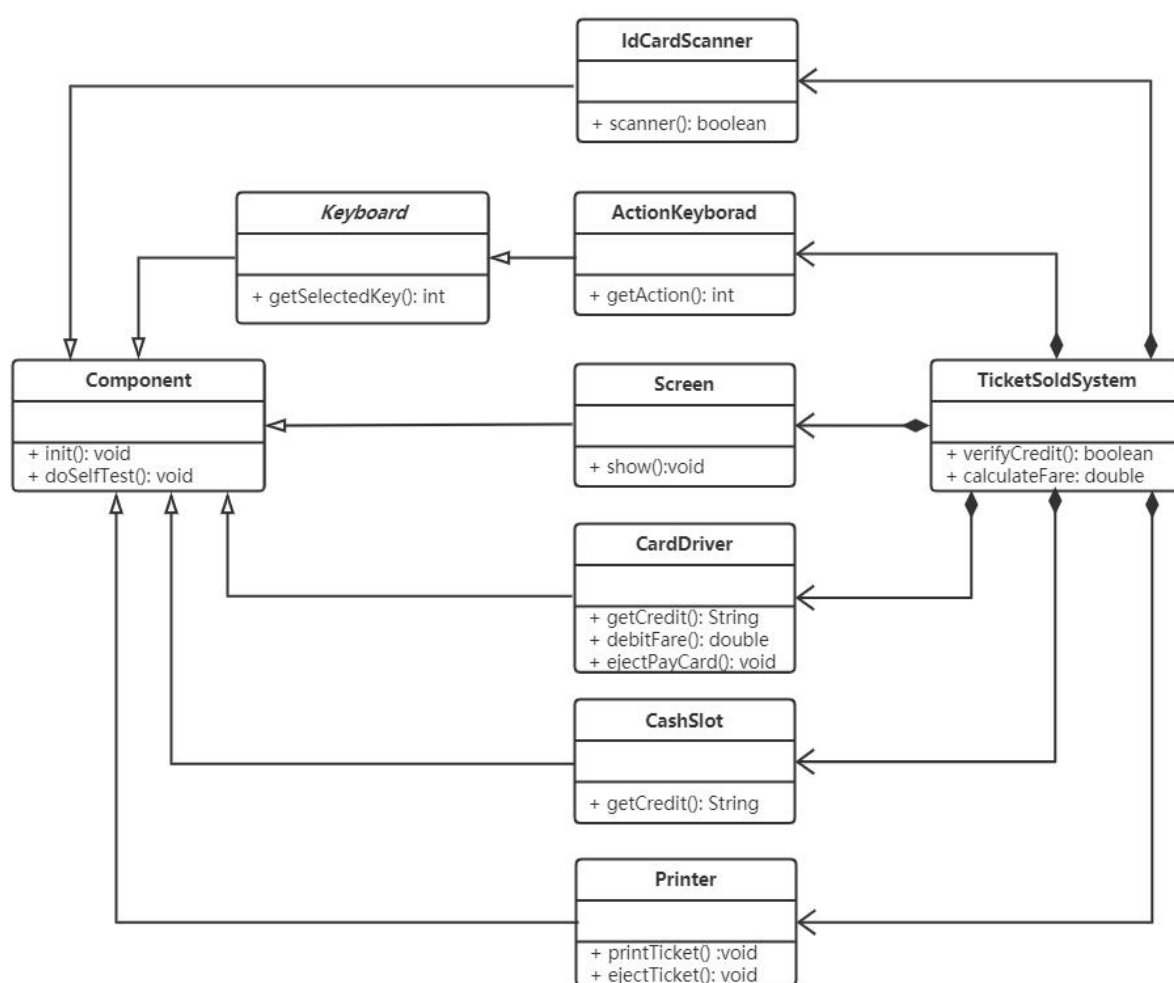
类名	说明
Component	抽象部件类，所有部件类的父类
Keyboard	抽象键盘类
ActionKeyboard	继续/取消键盘类
Screen	显示屏类
CardDriver	卡驱动器类
Printer	打印车票

方法说明：

类名	方法名	说明
Component	init()	初始化部件
	doSelfTest()	自检

Keyboard	getSelectedKey()	获取按键值
Screen	show()	显示信息
	touch()	触摸选择
Printer	printTicket()	打印车票

采用面向对象方法开发该系统，使用 UML 进行建模，绘制该系统的初始类图。



时序图 (Sequence Diagrams)

时序图描述对象之间消息的发送顺序，强调时间顺序。时序图是一个二维图，横轴表示对象，纵轴表示时间，消息在各对象之间横向传递，依照时间顺序纵向排列。用箭头表示消息、用竖虚线表示对象生命线。

时序图的作用

- 1、展示对象之间交互的顺序。将交互行为建模为消息传递，通过描述消息是如何在对象间发送和接收的来动态展示对象之间的交互；
- 2、相对于其他 UML 图，时序图更强调交互的时间顺序；
- 3、可以直观的描述并发进程。

时序图组成元素

角色 (Actor)

系统角色，可以是人、机器、其他系统、子系统；在时序图中用表示。

对象 (Object)

1、对象的三种命名方式

第一种方式包括对象名和类名，例如：直播课时:课时，在时序图中，用“对象：类”表示；

第二种方式只显示类名，即表示它是一个匿名对象，例如：:课程；在时序图中，用“：类”表示；

第三种方式只显示对象名不显示类名，例如：讲师；在时序图中，用“对象”表示。

2、命名方式的选择

三种命名方式均可，哪种最容易让阅读该时序图的人理解，就选择哪种。

3、对象的排列顺序

对象的左右顺序并不重要，但是为了作图清晰整洁，通常应遵循以下两个原则：把交互频繁的对象尽可能的靠拢；把初始化整个交互活动的对象放置在最左端。

生命线 (Lifeline)

在时序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间。

控制焦点 (Focus of Control)

又称为激活期，表示时间段的符号，在这个时间段内对象将执行相应的操作。可以理解为 Java 语言中一对大括号{ }中的内容；用小矩形表示。

消息 (Message)

消息一般分为同步消息 (Synchronous Message)，异步消息 (Asynchronous Message) 和返回消息 (Return Message)。

- 1、消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息的接收者放弃或者返回控制。用来表示同步的意义；
- 2、消息发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接受者返回消息或者控制。异步消息的接收者和发送者是并发工作的。
- 3、返回消息表示从过程调用返回。

自关联消息

表示方法的自身调用或者一个对象内的一个方法调用另外一个方法。

组合片段

组合片段用来解决交互执行的条件和方式，它允许在序列图中直接表示逻辑组件，用于通过指定条件或子进程的应用区域，为任何生命线的任何部分定义特殊条件和子进程。组合片段共有 13 种，名称及含义如下：

片段类型	名称	说明
Opt	选项	包含一个可能发生 或可能不发生的序列。可以在临界中指定序列发生的条件。
Alt	抉择	<p>包含一个片段列表,这些片段包含备选消息序列。在任何场合 下只发生一个序列。</p> <p>可以在每个片段中设置一个临界来指示该片段可以运行的条件。 else 的临界指示其他任何临界都不为 True 时应运行的片段。如果所有临界都为 False 并</p>

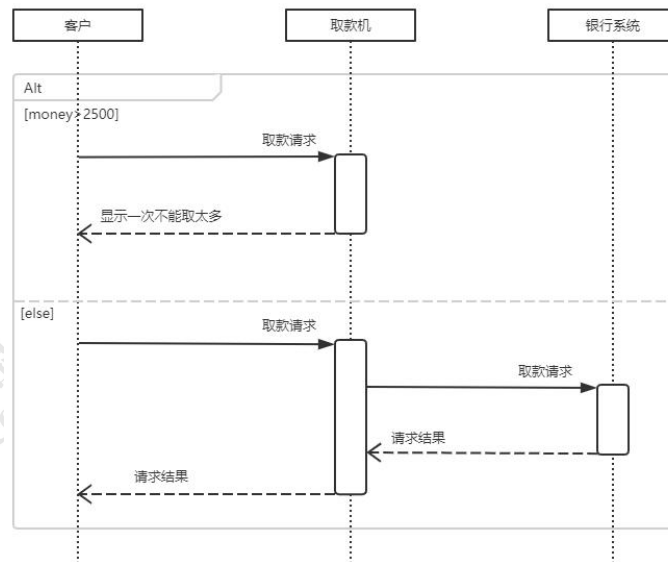
		且没有 else ,则不执行任何片段。
Loop	循环	片段重复一定次数。可以在临界中指示片段重复的条件。 Loop 组合片段具有 "Min" 和 "Max" 属性,它们指示片段可以重复的最小和最大次数。默认值是无限限制。
Break	中断	如果执行此片段,则放弃序列的其余部分。可以使用临界来指示发生中断的条件。
Par	并行	并行处理。片段中的事件可以交错。
Critical	关键	用在 Par 或 Seq 片段中。指示此片段中的消息不得与其他消息交错。
Seq	弱顺序	有两个或更多操作数片段。涉及同一生命线的消息必须以片段的顺序发生。如果消息涉及的生命线不同,来自不同片段的消息可能会并行交错。
Strict	强顺序	有两个或更多操作数片段。这些片段必须按给定顺序发生。
Consider	考虑	指定此片段描述的消息列表。其他消息 可发生在运行的系统中,但对此描述来说意义不大。 在 "Messages" 属性中键入该列表。
Ignore	忽略	此片段未描述的消息列表。这些消息可发生在运行的系统中,但对此描述来说意义不大。 在 "Messages" 属性中键入该列表。
Assert	断言	操作数片段指定唯一有效的序列。通常用在 Consider 或 Ignore 片段中。
Neg	否定	此片段中显示的序列不得发生。通常用在 Consider 或 Ignore 片段中。

常用组合片段举例：

用来指明在两个或更多的消息序列之间的互斥的选择，相当于经典的 if..else..

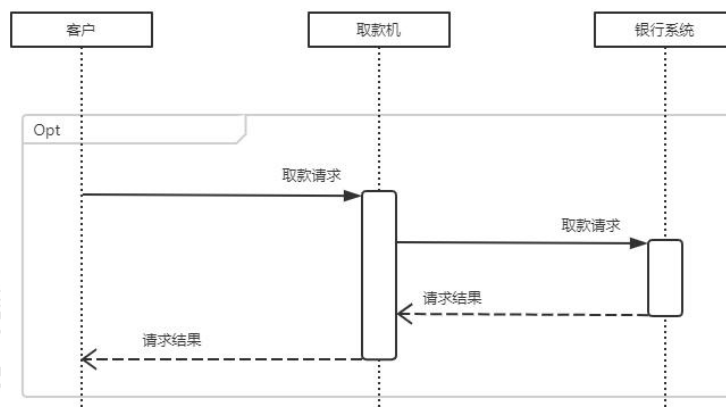
1、抉择 (Alt)

抉择在任何场合下只发生一个序列。可以在每个片段中设置一个临界来指示该片段可以运行的条件。else 的临界指示其他任何临界都不为 True 时应运行的片段。如果所有临界都为 False 并且没有 else，则不执行任何片段。



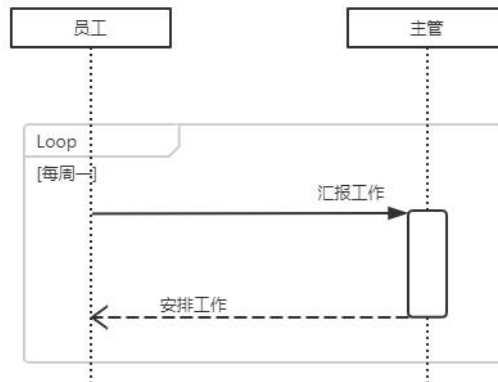
2、选项 (Opt)

包含一个可能发生或不发生的序列；

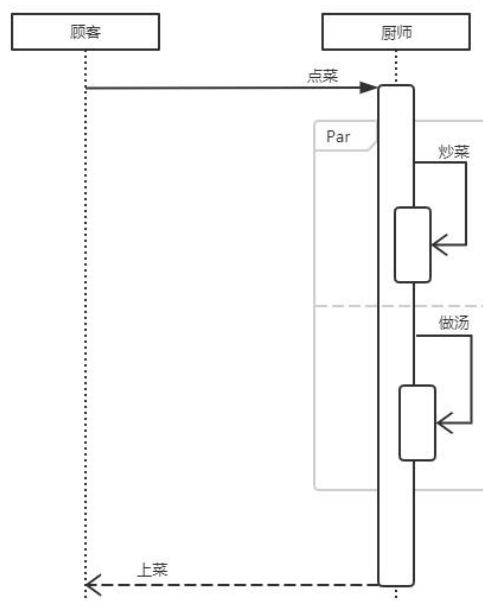


3、循环 (Loop)

片段重复一定次数，可以在临界中指示片段重复的条件。



4、并行 (Par)



时序图画法及实践

时序图的绘制步骤可简单总结如下：

- 1、划清边界，识别交互的语境；
- 2、将所要绘制的交互场景中的角色以及对象梳理出来；
- 3、从触发整个交互的某个消息开始，在生命线之间从上到下依次画出所有消息，并注明每个消息的特性（如参数等）。

```
public class Server {
    Device device;
    void open(){
        //...
    }
    void print(String s){
        device.write(s);
        //...
    }
    void close(){
        //...
    }
}
```

```
public class Device {
    void write(String s){
        //...
    }
}
```

```
public class Client {
    Server server;
    void work(){
        server.open();
        server.print("hello");
        server.close();
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Client client = new Client();
        client.work();
    }
}
```

```
1: 同步调用
2: 异步调用
3: 返回
```

