在 Java 或其它面向对象设计模式中，类与类之间主要有 6 种关系，他们分别为：依赖，关联，聚合，组合，继承，实现。他们的耦合度依次增强。

UML图关系详解

1. 依赖 (Dependency)

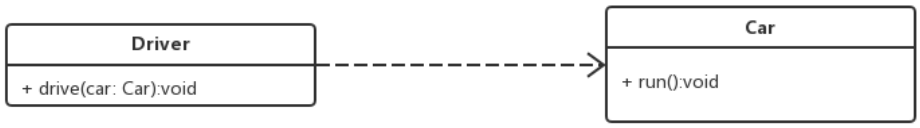
依赖关系的定义为：对于两个相对独立的对象，当一个对象负责构造另一个对象的实例，或者依赖另一个对象的服务时，这两个对象之间主要体现为依赖关系。

可以简单的理解为：类 A 使用到了类 B，而这种使用关系具有偶然性，临时性，非常弱的，但是 B 类中的变化会影响到类 A，比如某个学生要用笔写字，学生与笔的关系就是一种依赖关系，如果笔没水了，那学生就不能写字了(B 类的变化会影响类 A) 或者换另一只笔继续写字(临时性体现)。

思考下面这样的场景：

你是一名出租车司机，每天开着公司给你分配的车去载客，而每天出租车可能不同，我只是个司机，公司给我什么车我就开什么车，我使用这个车。

具体 UML 类图表现为：



Java 代码：

```
class Driver {
    public void drive(Car car) {
    car.run();
    }
}
class Car {
    public void run(){}
}
```

依赖关系不一定表现为形参，一共可以有三种表现形式:

```
class Driver {
    //通过形参方式发生依赖关系
    public void drive1(Car car) {
    car.run();
    }
     //通过局部变量发生依赖关系
    public void drive2() {
    Car car = new Car();
    car.run();
    }
    //通过静态变量发生依赖关系
    public void drive3() {
    Car.run();
    }
```

```
}
```

2. 关联 (Association)

关联关系的定义为：对于两个相对独立的对象，当一个对象的实例与另一个对象的一些特定实例存在固定的对应关系时，这两个对象之间为关联关系。

它体现的两个类中一种强依赖关系，比如我和我的朋友，这种关系比依赖更强，不存在依赖关系中的偶然性，关系也不是临时的，一般是长期性的。

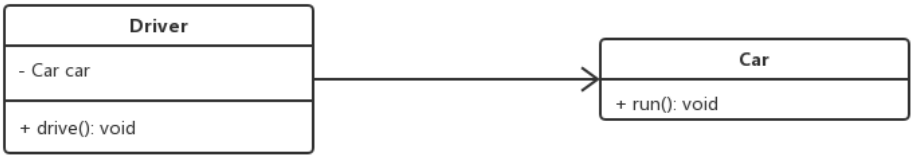
关联关系分为单向关联和双向关联：

1. 在 Java 中，单向关联表现为：类 A 当中使用了 类 B，其中类 B 是作为类 A 的成员变量。
2. 双向关联表现为: 类 A 当中使用类 B 作为成员变量，同时类 B 中也使用了类 A 作为成员变量。

根据可以上面的例子可以修改为以下的场景：

我是一名老司机，车是我自己的，我拥有这辆车，平时也会用着辆车去载客人。

用 UML 类图表示为:



双向关联的话箭头可以省略。

用 Java 代码表示为：

```
class Driver {
    private Car car = new Car();
    public void drive() {
    car.run();
    }
}
class Car {
    public void run(){}
}
```

依赖和关联区别：

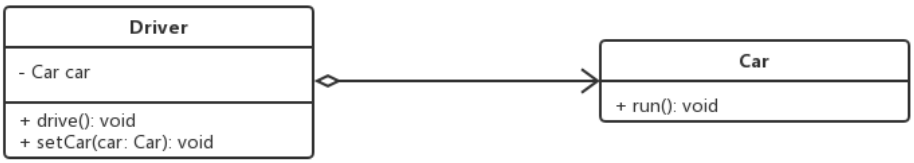
- 用锤子修了一下桌子，我和锤子之间就是一种依赖，我和我的同事就是一种关联。
- 依赖是一种弱关联，只要一个类用到另一个类，但是和另一个类的关系不是太明显的时候（可以说是“uses”了那个类），就可以把这种关系看成是依赖，依赖也可说是一种偶然的关系，而不是必然的关系。
- 关联是类之间的一种关系，例如老师教学生，老公和老婆这种关系是非常明显的。依赖是比较陌生，关联是我们已经认识熟悉了。

3.聚合 (Aggregation)

聚合关系是关联关系的一种，耦合度强于关联，他们的代码表现是相同的，仅仅是在语义上有所区别：关联关系的对象间是相互独立的，而聚合关系的对象之间存在着包容关系，他们之间是“整体-个体”的相互关系。

聚合关系中作为成员变量的类一般使用 set 方法赋值。

用 UML 类图表示为：



Java 代码:

```
class Driver {
    private Car car = null;
    public void drive() {
    car.run();
    }
    public void setCar(Car c){
    car = c;
    }
}
class Car {
    public void run(){}
}
```

4. 组合 (Composition)

相比于聚合，组合是一种耦合度更强的关联关系。存在组合关系的类表示“整体-部分”的关联关系，“整体”负责“部分”的生命周期，他们之间是共生共死的；并且“部分”单独存在时没有任何意义。

对比与聚合关系，我们可以将前面的例子变为下面的场景：

1. 车是一辆私家车，是司机财产的一部分，**强调的是人财产的部分性**，则相同的代码即可表示聚合关系。
2. 车是司机必须有的财产，要想成为一个司机必须要现有财产，车要是没了，司机也不想活了。而且司机要是不干司机了，这车也就没了。

所以，关联、聚合、组合只能配合语义，结合上下文才能够判断出来，而只给出一段代码让我们判断是关联，聚合，还是组合关系，则是无法判断的。

用 UML 类图表示为:

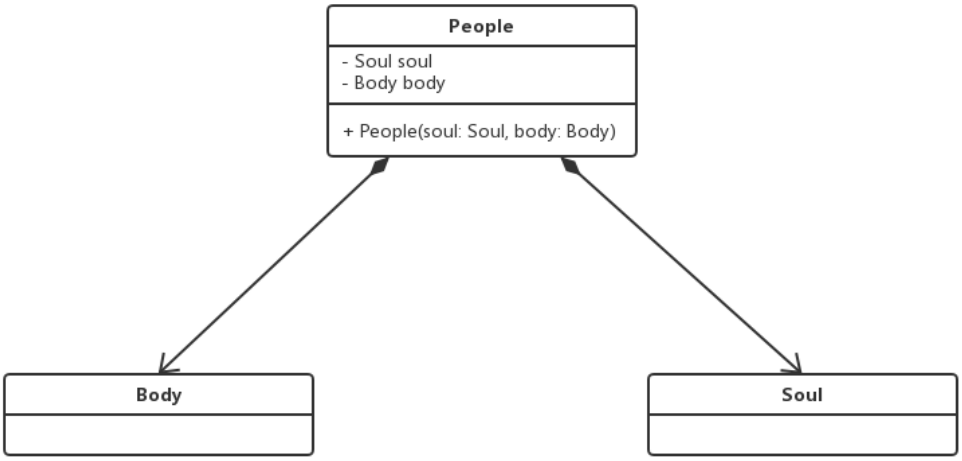
Java 代码:

```
class Driver {
    private Car car = null;
    public Driver(Car car) {
    this.car = car;
    }
    public void drive() {
    car.run();
    }
}
class Car {
    public void run(){}
}
```

再举一个恰当点的例子：

人和灵魂，身体之间是组合关系，当人的生命周期开始时，必须同时拥有灵魂和肉体，当人的生命周期结束时，灵魂肉体随之消亡；无论是灵魂还是肉体，都不能单独存在，他们必须作为人的组成部分存在。

用 UML 类图表示为



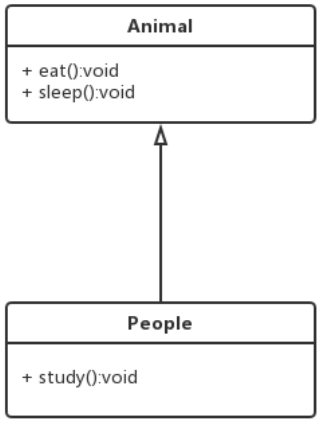
5. 继承 (Generalization)

继承表示类与类 (或者接口与接口) 之间的父子关系。在 Java 中，用关键字 extends 表示继承关系。

思考下面的场景：

人是一种高级动物，不仅可以像动物可以吃和睡觉，而且还可以学习。

用 UML 类图表示为:



Java 代码表示为:

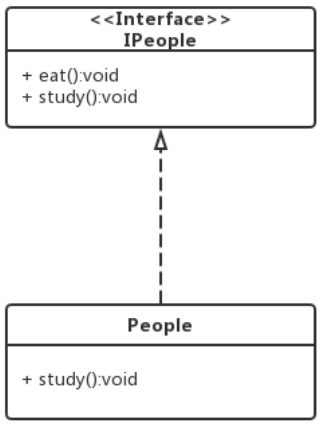
```
class Animal {
    public void eat(){}
    public void sleep(){}
}
class People extends Animal {
    public void study(){}
}
```

6. 实现 (Implementation)

表示一个类实现一个或多个接口的方法。接口定义好操作的集合，由实现类去完成接口的具体操作, 在 Java 中使用 implements 表示。在 Java 中，如果实现了某个接口，那么就必须实现接口中所有的方法。

比如一个人可以吃饭和学习，那么就可以定义一个人的接口。让具体的人去实现它。

用 UML 类图表示为:



Java 代码:

```
interface IPeople {
    public void eat();
    public void study();
}
class People implements IPeople {
    public void eat(){
    }
    public void study(){
    }
}
```