

单例模式仅仅涉及到一个类 是最简单的设计模式

工厂模式很常用

工厂模式 是用来创建实例化对象的

通过工厂模式实现类创建者和调用者的分离

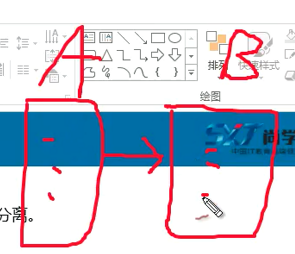
 调用者也要去new

面向对象的原则

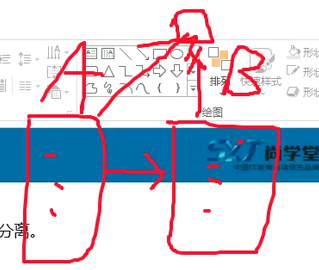
OCP ---- 开放封闭原则 ---- 对修改封闭 对扩展开放 -----

比如 写了两个类 现在要扩展新功能 不要修改已有的两个类 要扩展出第三个类来增加新功能

DIP ---- 依赖倒置原则 ---- 以前类之间是直接调用的

 A直接调用B

B有一个接口 以后A直接调用B的接口 ----- 这样依赖从具体变成了依赖抽象



LOD ---- 迪米特原则 ----- 仅仅与直接朋友通信，避免和陌生人

---- 说白了就是我的这个类尽量少和其他类发生关系

这些原则和模式让我们的项目就是容易扩展

依赖的手段只有一个就是：分工

 ---- 分工就是设计模式的灵魂

过去的人 工作 做饭 洗衣都会 ---- 没有分工

但是 现在我只要会java 可以不会做饭 --- 有分工 ---- 这样的社会发展才好

只有分工 项目才容易扩展

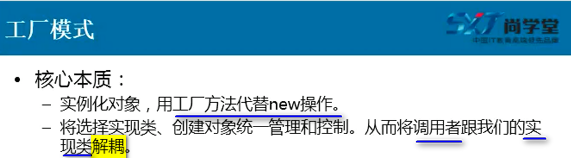
设计模式都是这对面向对象来说的

工厂模式是大类 分成了简单工厂 工厂方法和抽象方法

三个对比先不着急

这是一个个发展的关系

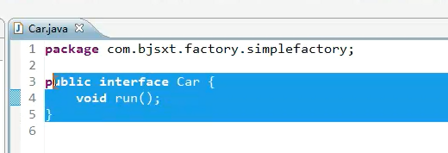
先看简单工厂

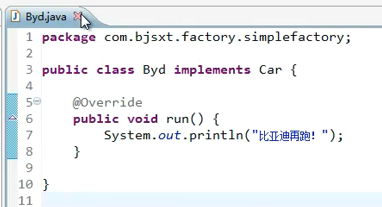


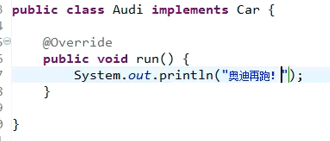
核心本质 ---- 工厂方法代替new操作

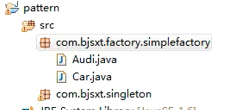
----- 实现类创建对象统一管理和控制 ---- 从而将调用者 跟 实现类 解耦【实际上类图表现为 client和实现类之间没有uml的线的关系】

示例代码



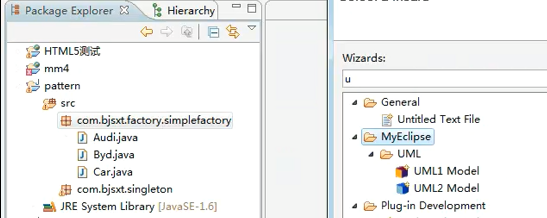




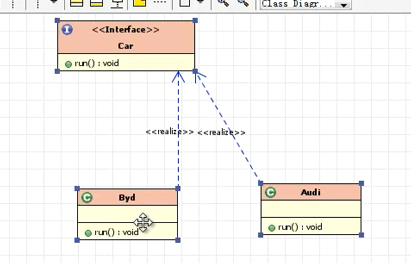


这样 Car这个接口有两个实现类 Audi和Byd

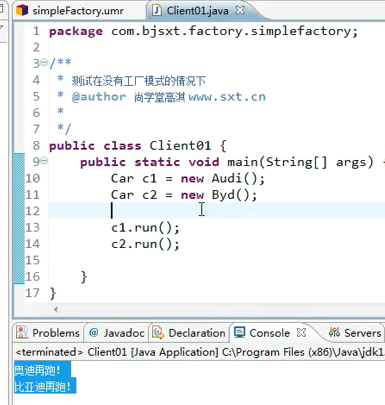
画出来类图 –eclipse画



这三个类拖过来之后 自动出现



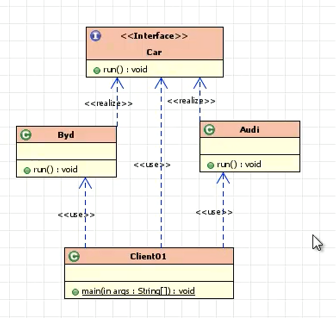
没有工厂的时候 我要创建几个车 做法是



问题是 Client01既要知道接口 也要知道具体的实现类 要知道的 或者依赖的太多了

Client

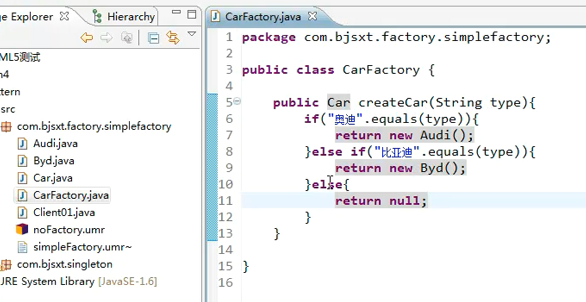
类图如下：



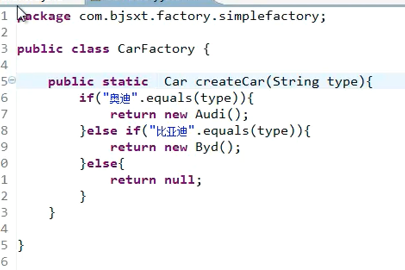
【可以看出来 Client01和接口Car 两个类Byd Audi都由线的关联 所以 出现了耦合】

也就是这个人既要会种麦子 还要会把麦子打成粉

实现一个工厂类



为了方便调用 这个createCar方法修改成静态的



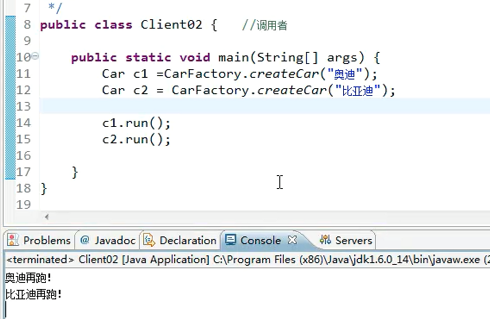
有一个开闭原则 这个如果要添加一个类 我这个CarFactory就要修改代码 违反了开闭原则

简单工厂是有小问题的 --- 后面使用工厂方法模式

---因为不是写jdk 写的是小项目 这样的问题是可以容忍的

【考虑一个模式是否满足开闭原则 就是 按照定义 考虑 如果扩展了新的功能 类的代码是否要有变化？】

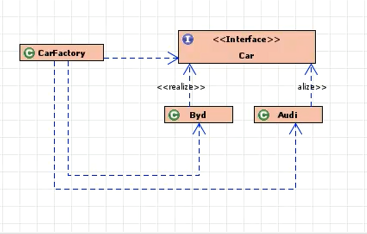
这样在Client中 不用和具体的实现类 Audi Byd打交道了 直接调用的是接口Car 这样 就说明了 客户端和具体类实现了解耦合 没有线关联了



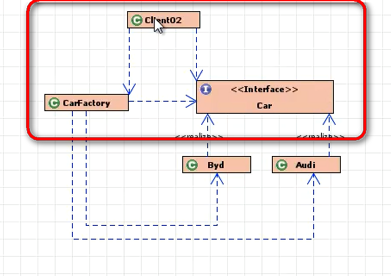
**我这样多增加了一个类实现了分工** ----- CarFactory的工作就是用来创建对象的

如果我有很多产品 Car有很多实现类 我的Client仅仅需要依赖CarFactory和Car 不会和Car的众多实现类进行调用 解耦合

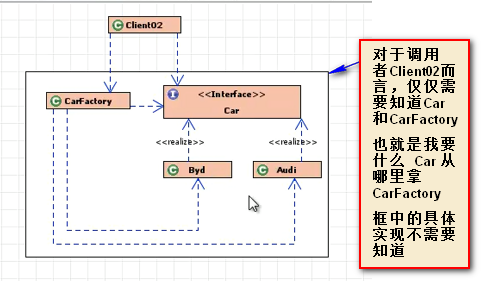
现在的类图：



对于外部调用类来讲

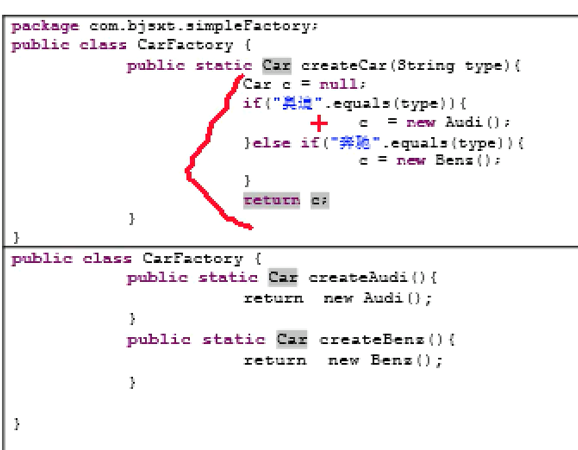


【设计模式的核心就是分工 增加了新的类就是实现具体的分工】



这是使用了工厂类之间的调用关系

第一个类图很简单 这个复杂了 ---- 整体关系是越来复杂了 但是 对于调用者是简单地！并且更加容易扩展

 ----- 有两种方式 if-else 另一种就是 针对每一种车 实现一个方法 这个都放在CarFactory中，还是简单工厂



简单工厂也叫静态工厂 --- 因为为了方便调用 工厂中的方法都是静态的 所以叫静态工厂

问题就是 为了扩展就要修改工厂类的方法 ---- 修改了类或者方法

违反了开闭原则 对修改关闭 对扩展开放