河北师范大学软件学院
Software College of Hebei Normal University

第一章 软件体系结构概述

任课教师： 武永亮
wuyongliang@edu2act.org

授课教师

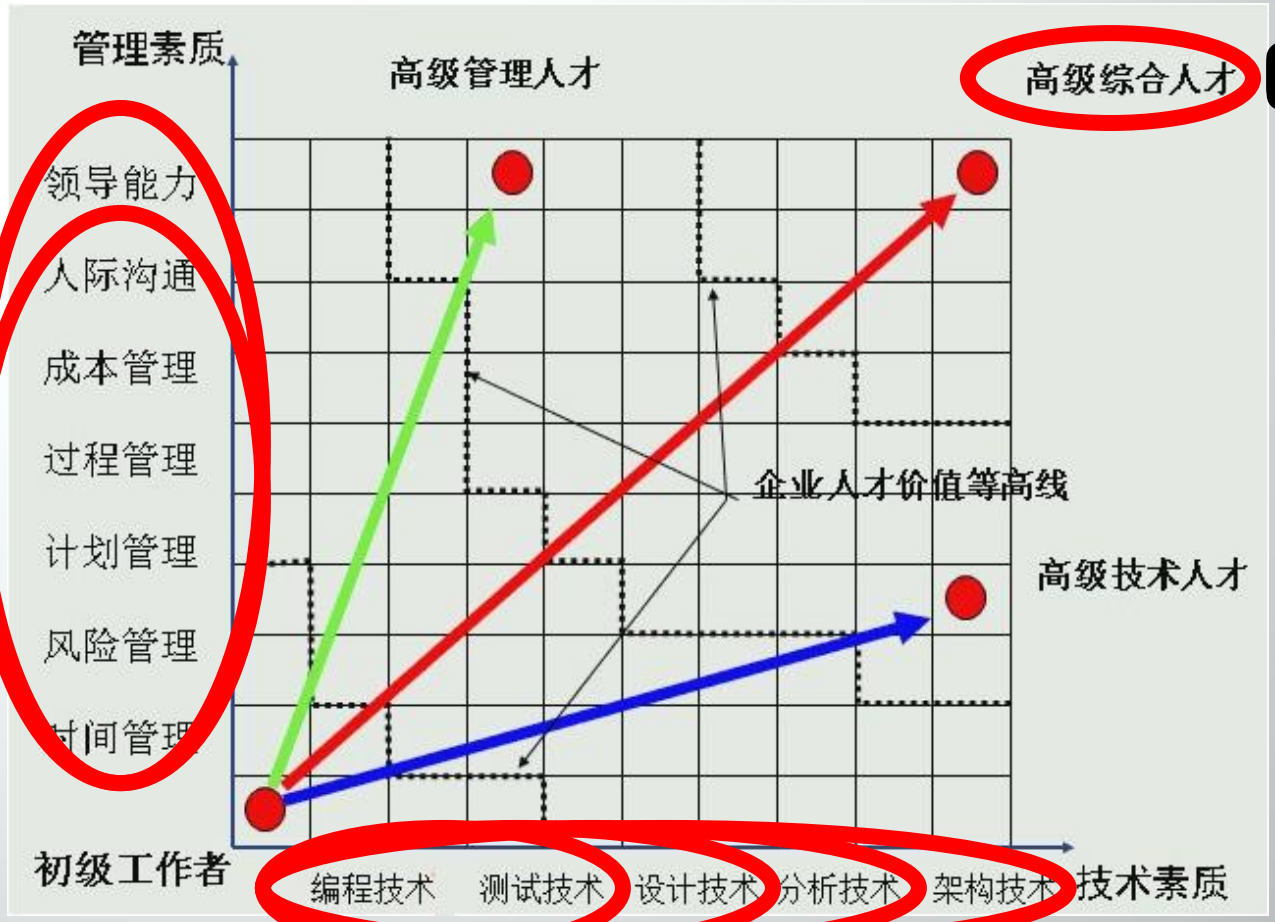
- 姓名：武永亮
- 邮箱：wuyongliang@edu2act.org
- QQ：395928533
- 钉钉直播群号：30105998

课程内容

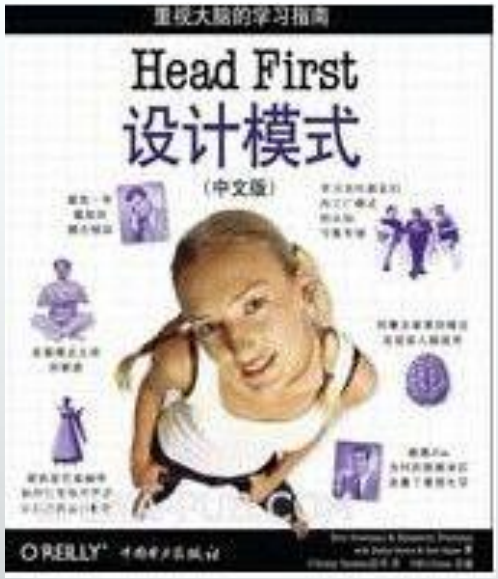
- 课程简介
- 课程内容及学习方式
- 第一个设计模式——单例模式
- 好设计的原则

课程内容

- 课程简介
- 课程内容及学习方式
- 第一个设计模式——单例模式
- 好设计的原则


管理素质
高级管理人才
高级综合人才
C'X'O
领导能力
人际沟通
成本管理
过程管理
计划管理
风险管理
时间管理
企业人才价值等高线
高级技术人才
初级工作者
编程技术
测试技术
设计技术
分析技术
架构技术
技术素质
技术经理
项目经理
工程师
设计师
分析师
架构师

教材及参考书


重现大脑的学习指南
Head First
设计模式
（中文版）
O'REILLY


大话
设计
模式
小菜
大鸟
Baidu 百科


软件工程技术丛书
设计系列
面向模式的
软件体系结构
卷1：模式系统
Pattern-Oriented Software Architecture
Volume 1: A System of Patterns


高等学校软件工程系列教材
软件设计模式与体系结构
蔚蓝网
www.wl.cn

考核方式

- 选修课
  - 32学时、2学分
- 平时成绩：70%
  - 平时表现：15%（课堂提问、课堂纪律、课堂出勤）
  - 雪梨平时作业：55%
- 期末成绩：30%
  - 考试形式：闭卷、笔试
- 课程资料：https://github.com/edu2act/course-Software-architecture
- 雪梨作业平台：http://www.edu2act.cn/team/20192020-di-er-xue-qi-ruan-jian-ti-xi-jie-gou-ke-c/

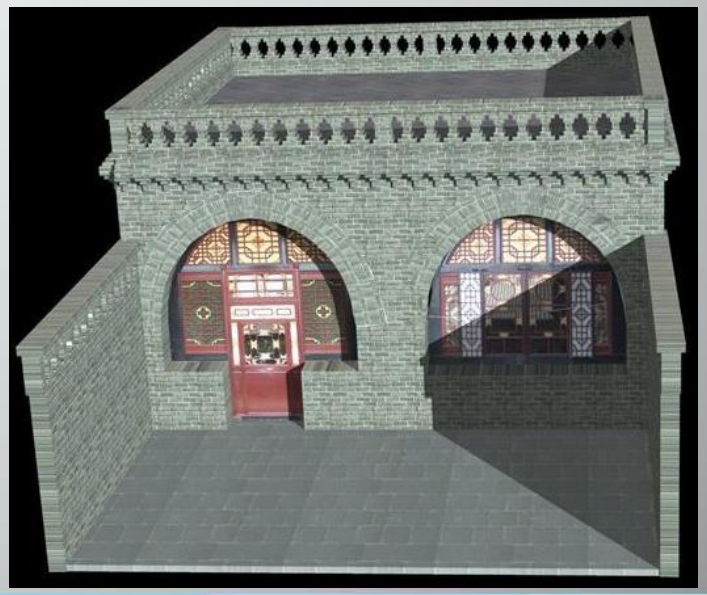
课程内容

- 课程简介
- 课程内容及学习方式
- 第一个设计模式——单例模式
- 好设计的原则

- **我们讲什么？**
  - 体系结构模式
  - 设计模式

什么是模式？

Alexander：每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，你可以无数次地使用那些已有的解决方案，无需再重复相同的工作。



什么是模式？

- 模式是一条由三部分组成的规则。
- 一个特定环境、一个问题、一个解决方案。
- 模式的核心思想：进行设计的复用。

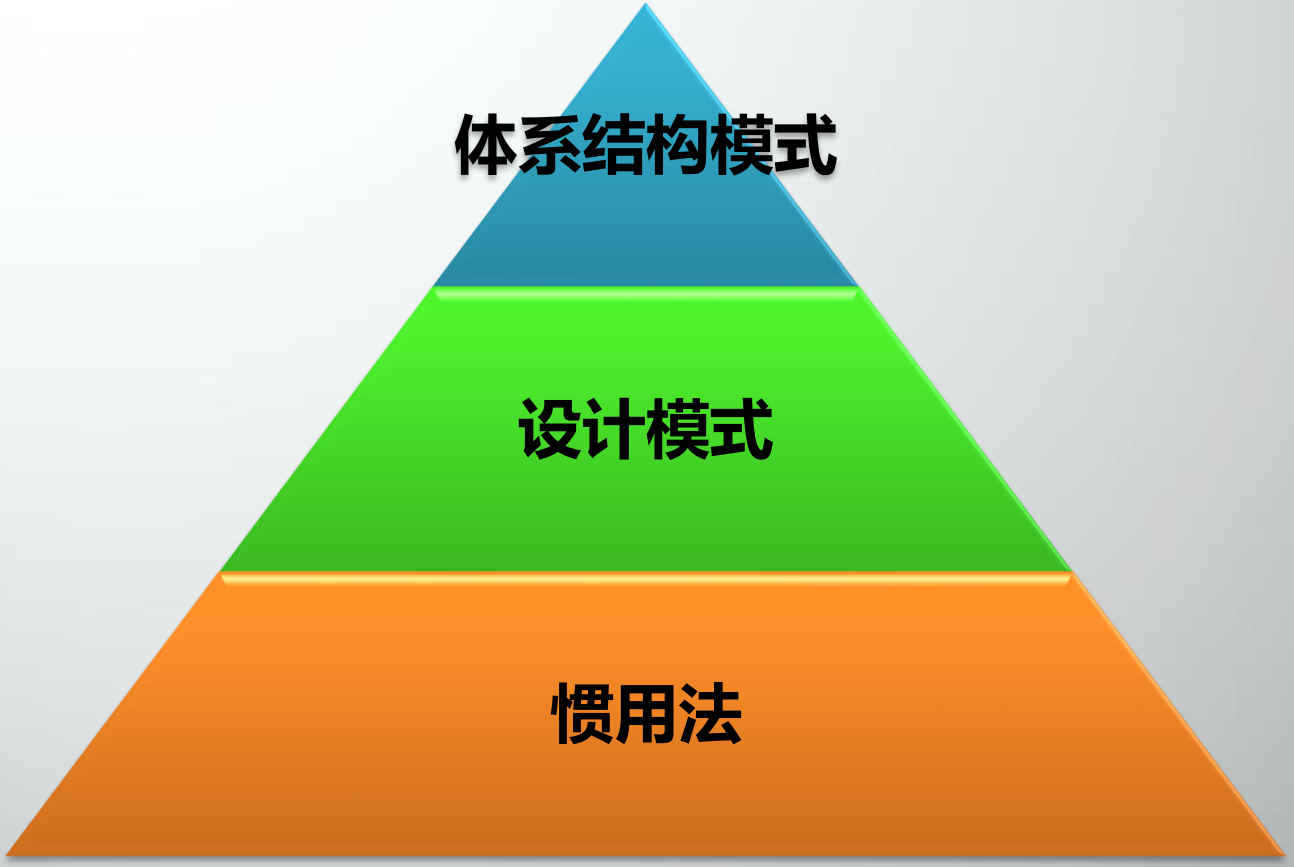
环境+问题+解决方案

设计模式与体系结构模式

设计模式：描述了定制化的相互通信的对象与类，以解决特定环境中的通用设计问题。

体系结构模式：是对系统的高层设计，是从一个较高的层次来考虑组成系统的构件、构件之间的连接关系，以及系统需满足的约束等，用以实现体系结构级的设计复用。通常又被成为架构模式、体系结构风格。

他们之间的关系



学习的方式

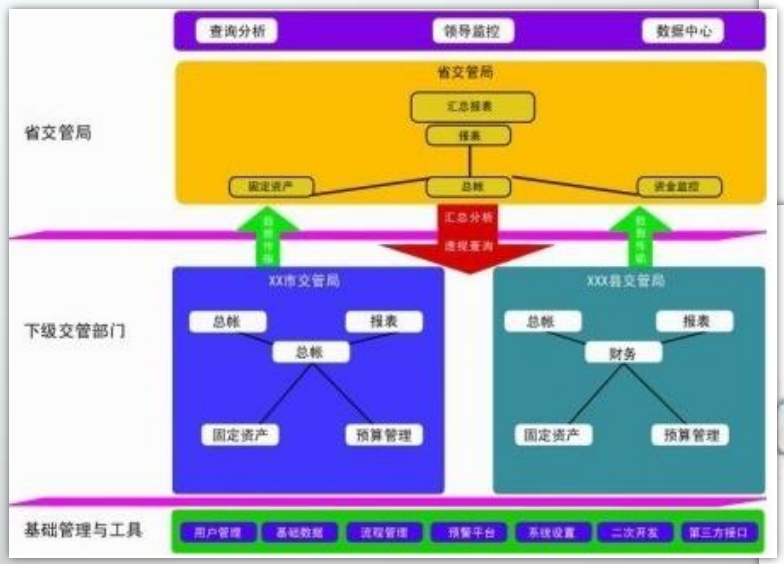
环境+问题+解决方案

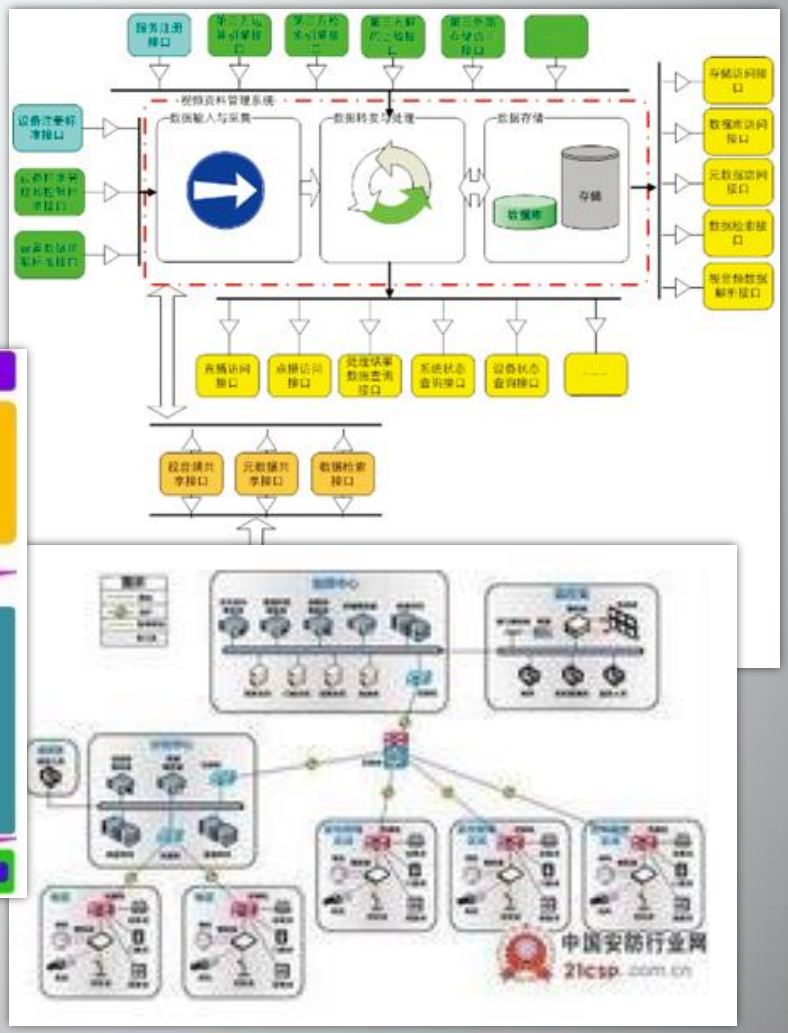

举个栗子！

全国交通违法数据联网——环境

全国交通违法数据联网——问题

- 平台异构
- 语言异构
- 系统架构异构





全国交通违法数据联网——解决方案

面向服务的体系结构——SOA！

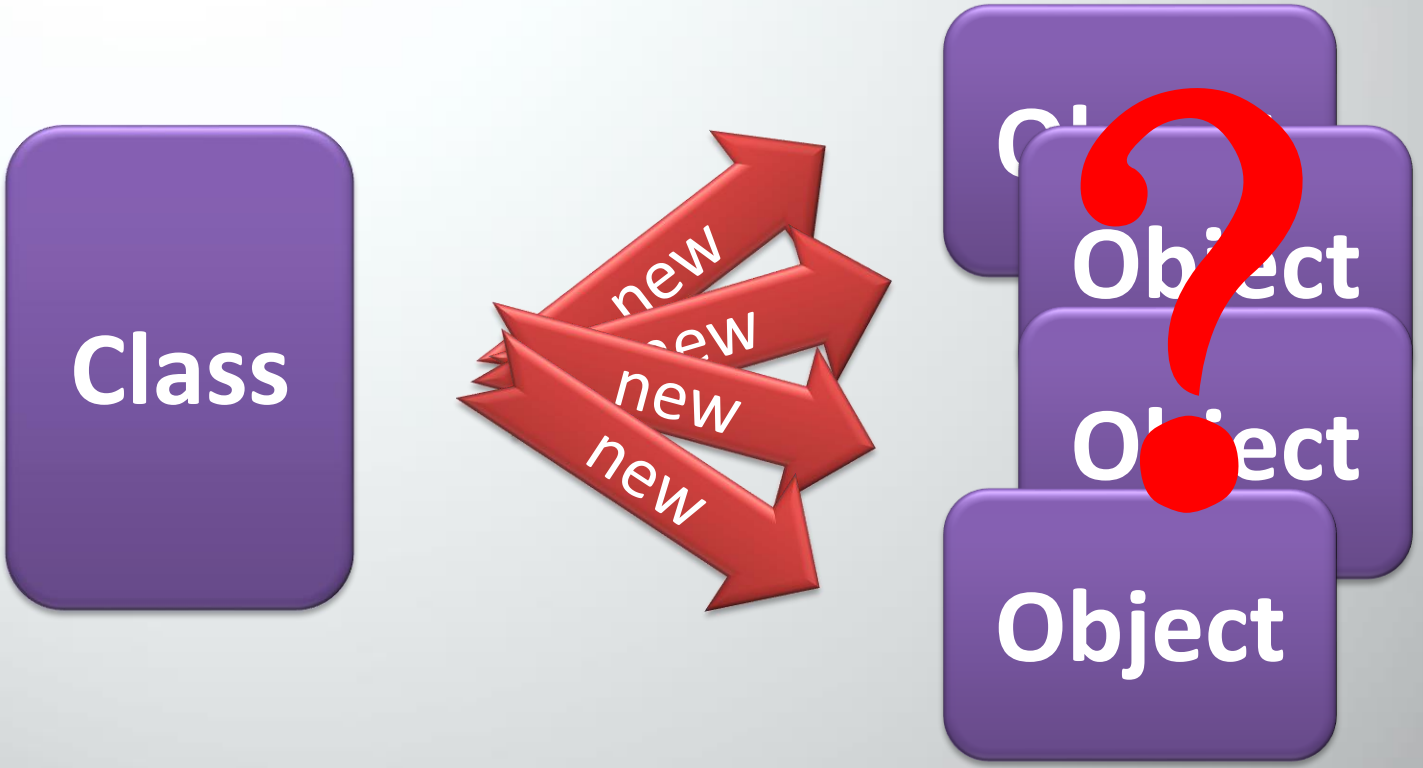
课程内容

- 课程简介
- 课程内容及学习方式
- 第一个设计模式——单例模式
- 好设计的原则

什么是单例模式

单例模式：
确保一个类仅有一个唯一的实例，并且提供一个全局的访问点。

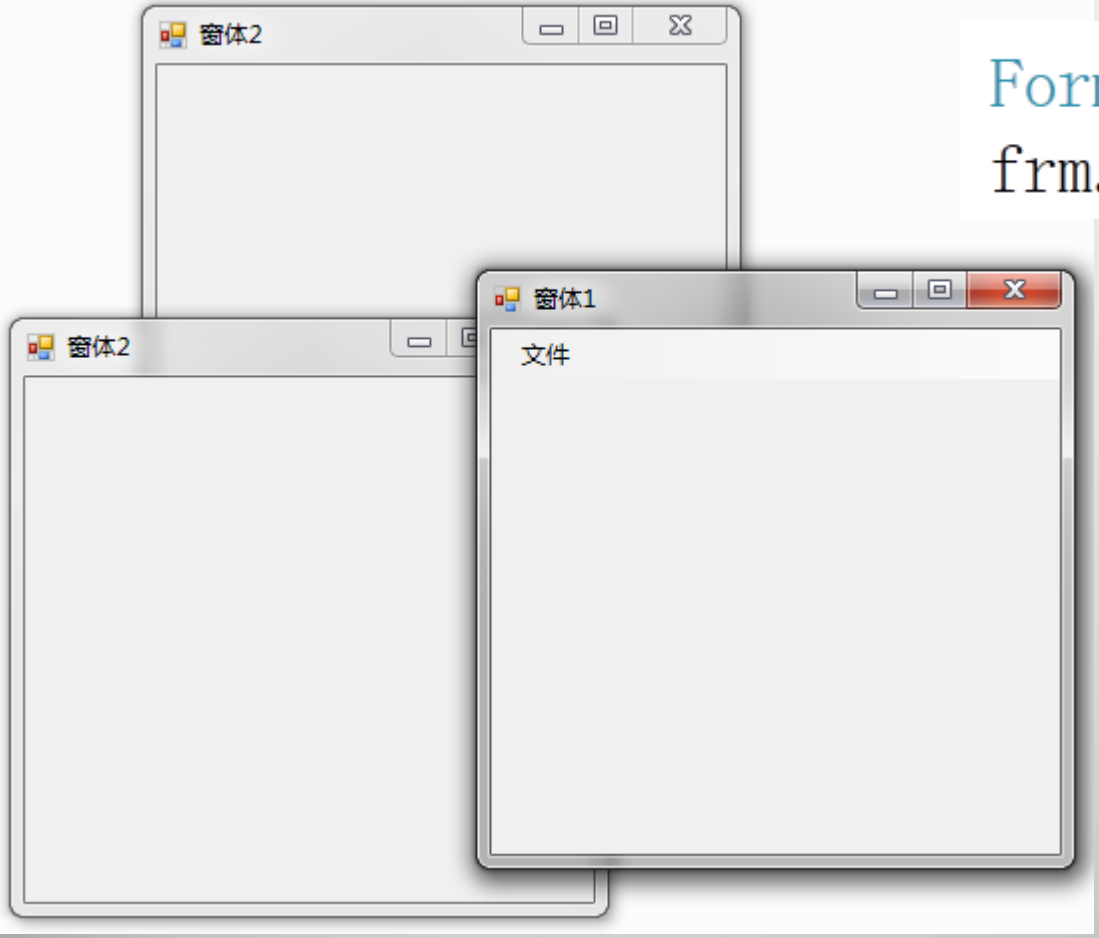
环境及问题



环境及问题

- 单例模式要解决的问题——独生子女

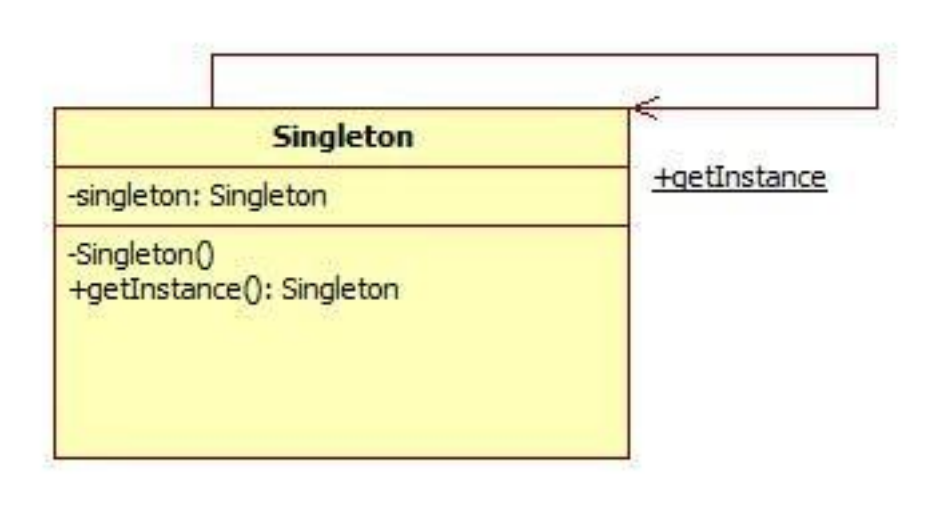
第一个设计模式——单例模式



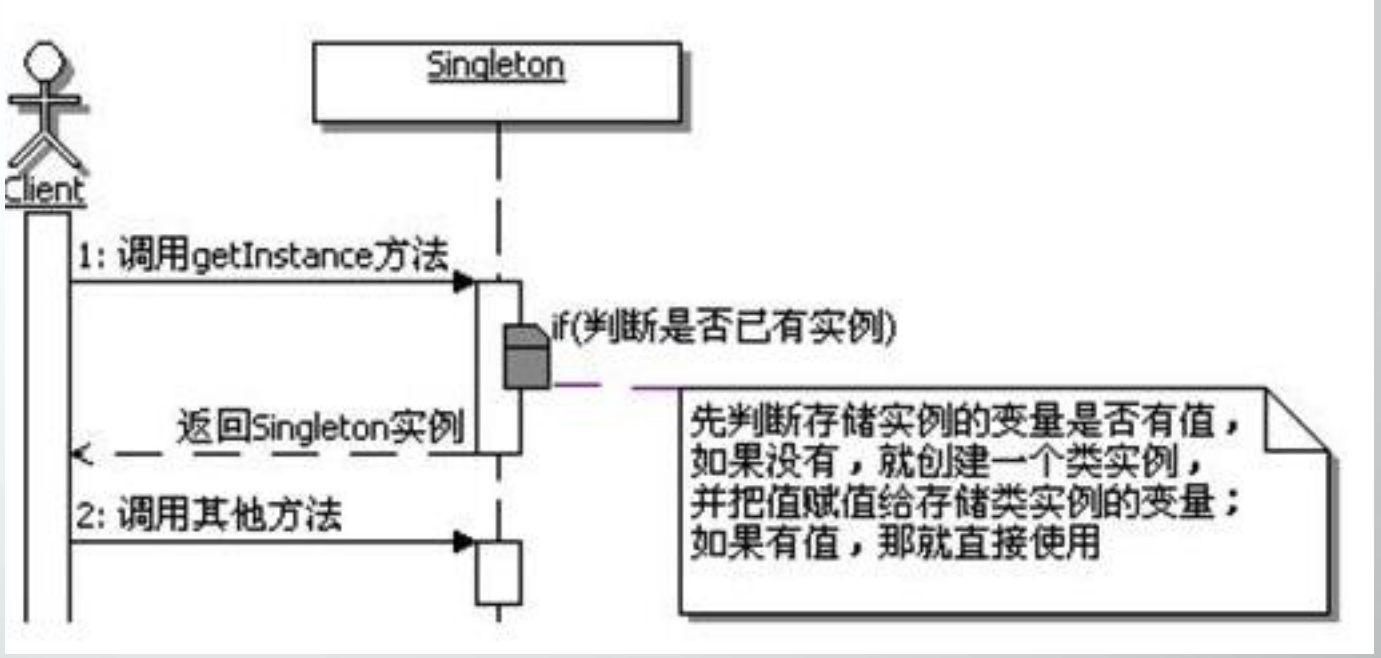

```
Form frm = new Form2();
frm.Show();
```

解决方案

- 首先将构造函数声明成私有类型，屏蔽通过直接实例化的形式来访问。
- 其次控制全局只有一个实例的类—Static
- 第三，提供一个可以获得实例的方法，用于返回类的实例，并保证得到的是同一个对象。



解决方案


Singleton
Client
1: 调用getInstance方法
if(判断是否已有实例)
返回Singleton实例
2: 调用其他方法
先判断存储实例的变量是否有值，
如果没有，就创建一个类实例，
并把值赋值给存储类实例的变量；
如果有值，那就直接使用

解决方

```
class Singleton
{
  //使用静态私有全局变量保存唯一的实例
    private static Singleton singleton;
    private  Singleton()
    {

    }
    /**
     * 这里保证只会实例化一次，也就是第一次进行实例化，接下来继续调用就
不会去实例化
     * @return singleton
     */
    public static Singleton getInstance()
    {
      if(null==singleton)
      {
        singleton=new Singleton();

      }
      return singleton;
    }

}
```

解决方案

```
class Singleton
{
  private static Singleton singleton=new Singleton();
  private Singleton()
  {

  }
  public static Singleton getInstance()
  {
    return singleton;
  }
}
```

解决方案

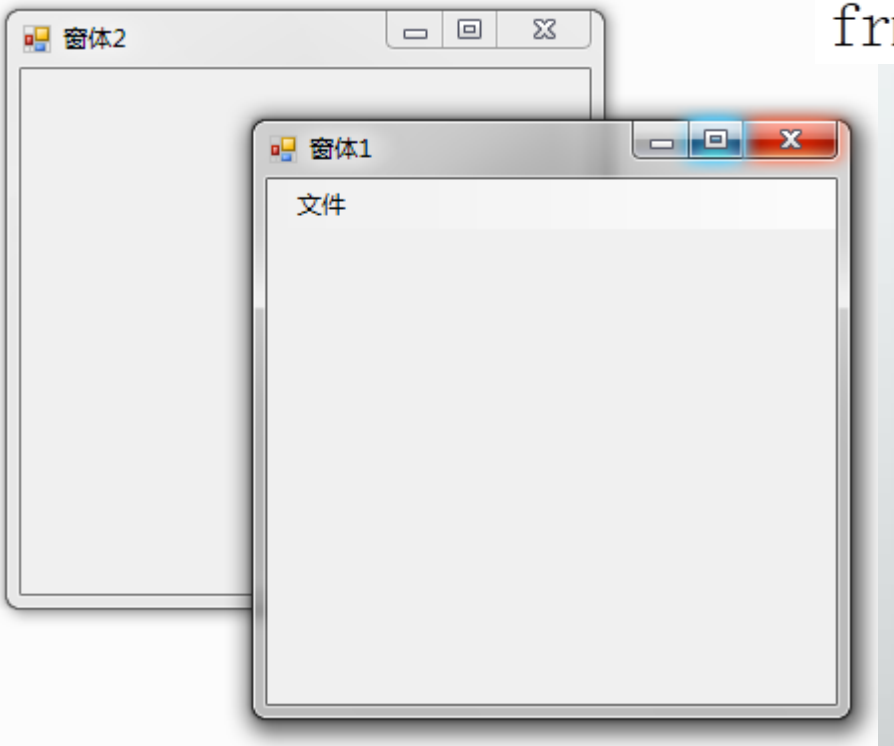
```
public partial class Form2 : Form
  {
        private Form2 Form2();

        private static Form2 instance;

        public static Form2 GetInstance()
         {
            if (instance==null)
            {
                instance = new Form2();
            }
            return instance;
         }
  }
```

第一个设计模式——单例模式

```
Form frm = Form2.GetInstance();
frm.Show();
```



单例模式的应用举例

- Windows的Task Manager（任务管理器）
- 网站的计数器、 应用程序的日志应用
- 数据库连接池的设计
- Web应用的配置对象的读取
- 操作系统的文件系统

单例模式的应用场景

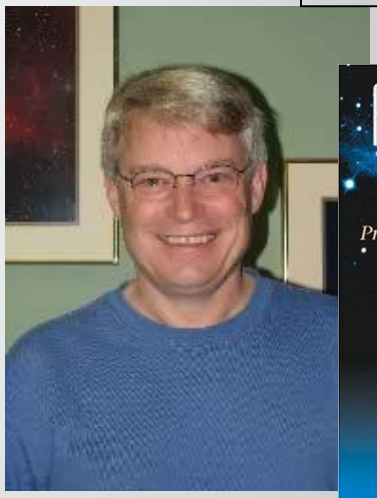
- 资源共享的情况下，避免由于资源操作时导致的性能损耗。如上述中的日志文件，应用配置
- 控制资源的情况下，方便资源之间的互相通信。如数据库连接池等

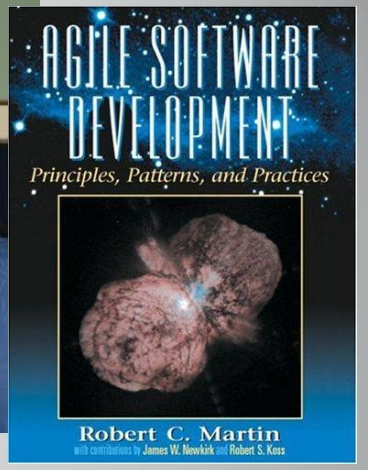
课程内容

- 课程简介
- 课程内容及学习方式
- 第一个设计模式——单例模式
- 好设计的原则

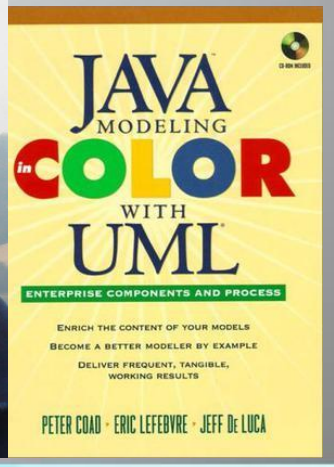
- **设计正在“腐烂”的征兆**
  - 过于僵硬Rigidity
  - 过于脆弱Fragility
  - 不可重用性immobility
  - 粘滞性过高viscosity
- **好的系统设计应该具备如下三个性质**
  - 可扩展性(Extensibility)
  - 灵活性(Flexibility)
  - 可插入性(Pluggability)

Robert C.Martin





Peter Coad



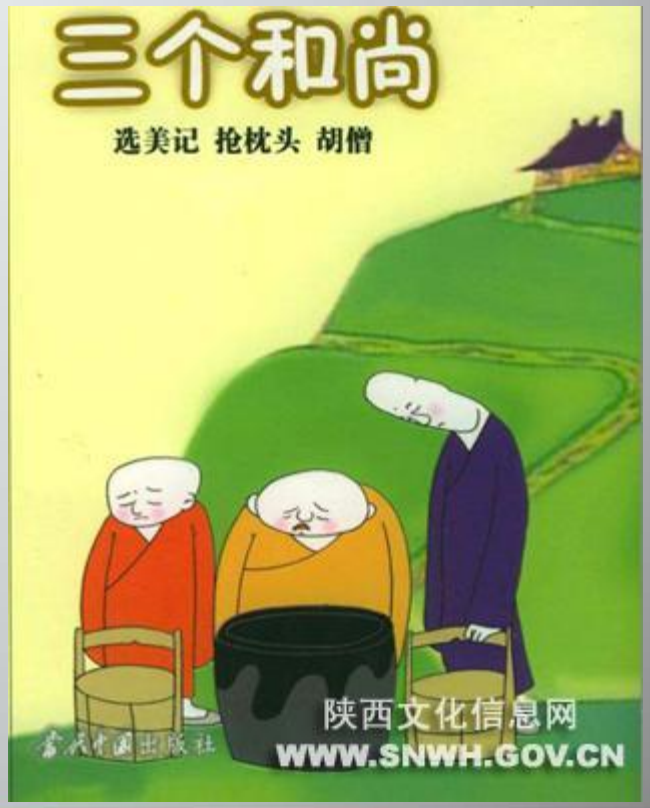
面向对象设计原则

设计原则名称	设计原则简介	重要性
单一职责原则 (Single Responsibility Principle, SRP)	类的职责要单一，不能将太多的职责放在一个类中。主要为了解耦和增强内聚性（高内聚、低耦合）。	★★★★☆
开闭原则 (Open-Closed Principle, OCP)	软件实体对扩展是开放的，但对修改是关闭的，即在不修改一个软件实体的基础上去扩展其功能。	★★★★★
里氏代换原则 (Liskov Substitution Principle, LSP)	在软件系统中，一个可以接受基类对象的地方必然可以接受一个子类对象。	★★★★☆
依赖倒转原则 (Dependency Inversion Principle, DIP)	要针对抽象层编程，而不要针对具体类编程。	★★★★★
接口隔离原则 (Interface Segregation Principle, ISP)	使用多个专门的接口来取代一个统一的接口。	★★☆☆☆
合成复用原则 (Composite Reuse Principle, CRP)	在系统中应该尽量多使用组合和聚合关联关系，尽量少使用甚至不使用继承关系。	★★★★☆
迪米特法则 (Law of Demeter, LoD)	一个软件实体对其他实体的引用越少越好，或者说如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用，而是通过引入一个第三者发生间接交互。	★★★☆☆

单一职责原则

- 高内聚性原则
- 避免相同的职责（也称为功能）分散到不同的类中实现。
- 避免一个类承担过多的职责。

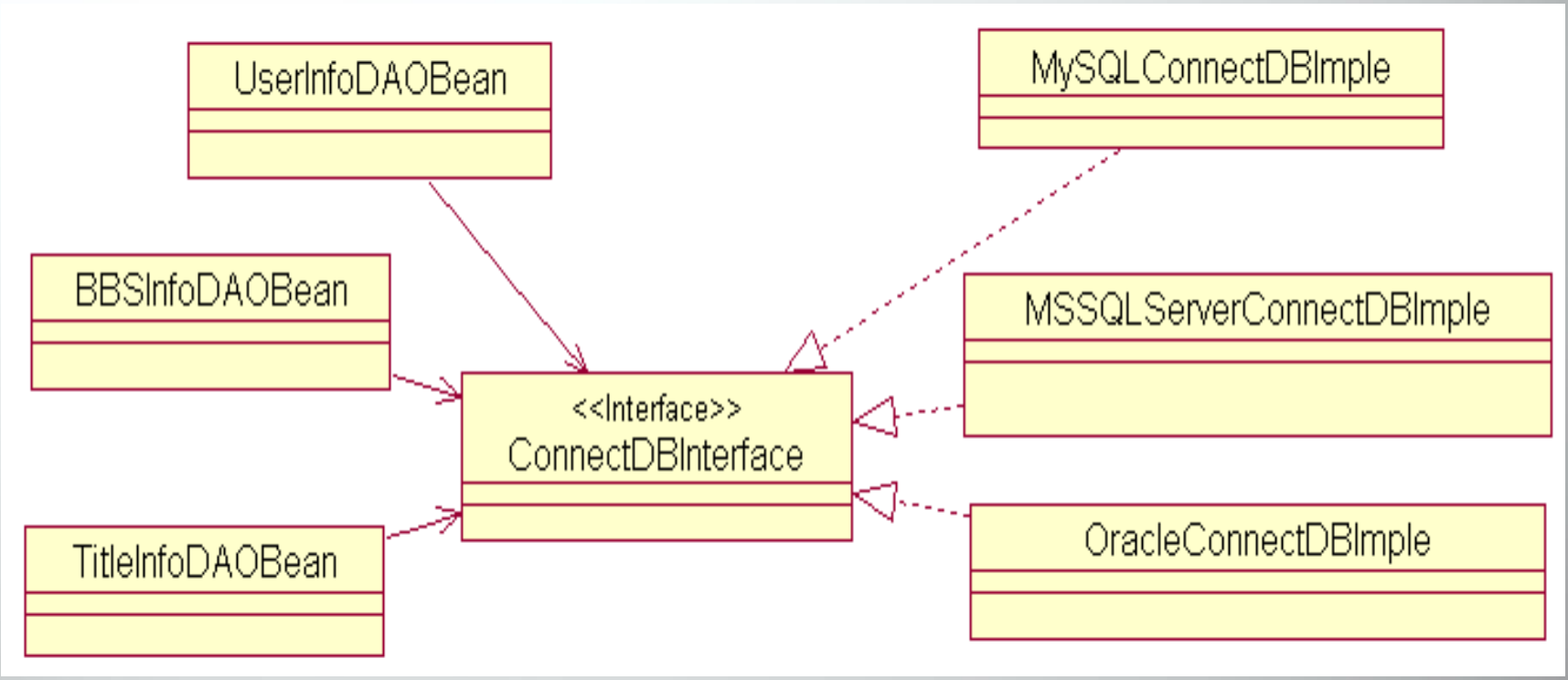
可以减少类之间的耦合



- **单一职责原则生活中的示例**
  - 组织机构的设置
  - 公司人员的分工

单一职责原则示例

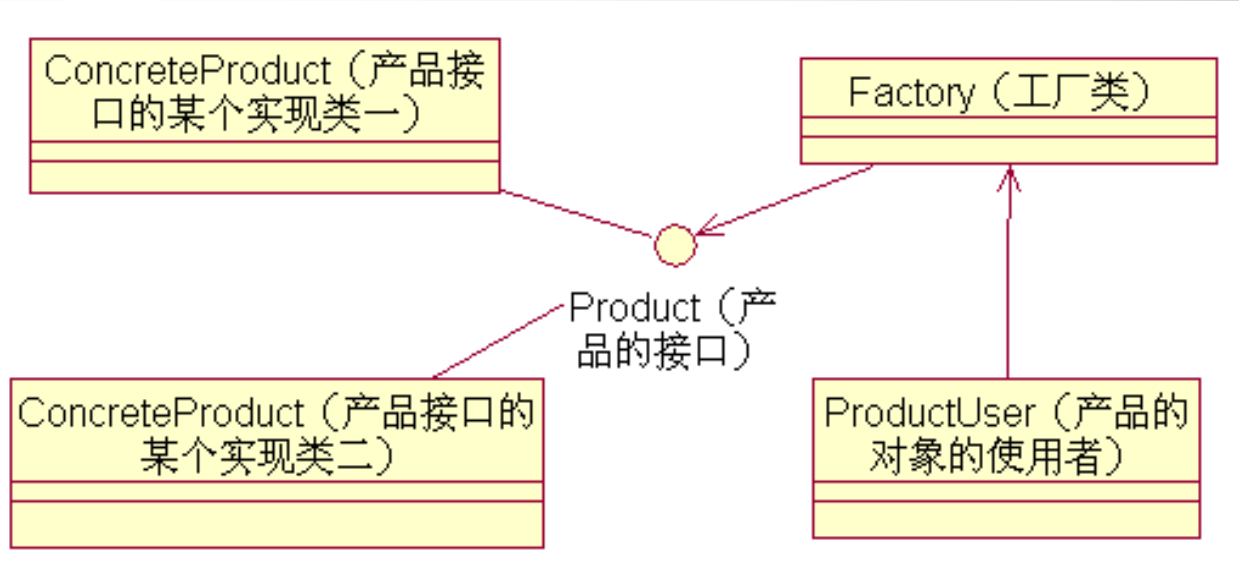
- 类的设计主要工作是“发现职责”并“分离职责”



数据库连接和数据库访问操作相互分离

遵守单一职责原则的设计模式

- 工厂模式
  - 分离对象的“创建”和对象的“使用”



开闭原则

- Open(Open for extension)
  - 模块的行为必须是开放的、支持扩展的，而不是僵化的
- Closed(Closed for modification)
  - 在对模块的功能进行扩展时，不应该影响或大规模地修改已有的程序模块
- 绝大部分的设计模式都符合开闭原则
- 抽象化是开闭原则的关键

要求开发人员可以在不修改系统中现有的代码的前提下，而实现对应用系统的软件功能的扩展。

里氏代换原则

- 主要是针对继承的设计原则
- 子类型必须能够替换掉它们的父类型、并出现在父类能够出现的任何地方。
- 子类可以扩展父类的功能，但不能改变父类原有的功能。

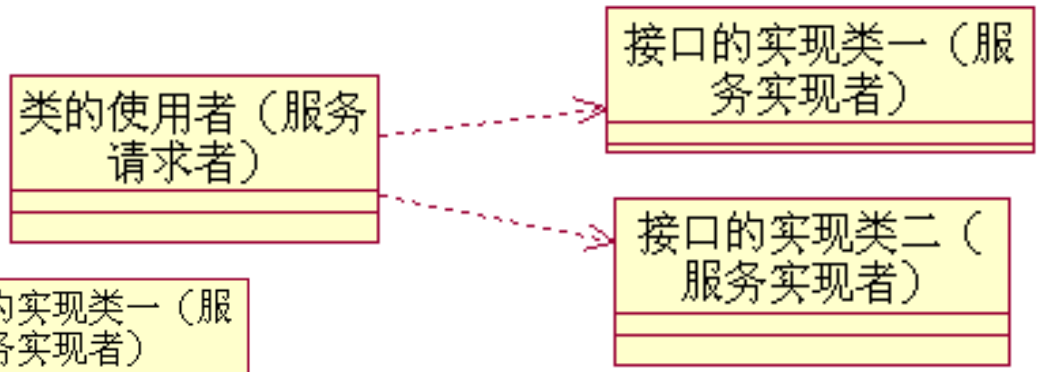
例子：生物学的分类体系中把企鹅归属为鸟类。模仿这个体系，设计出这样的类和关系。

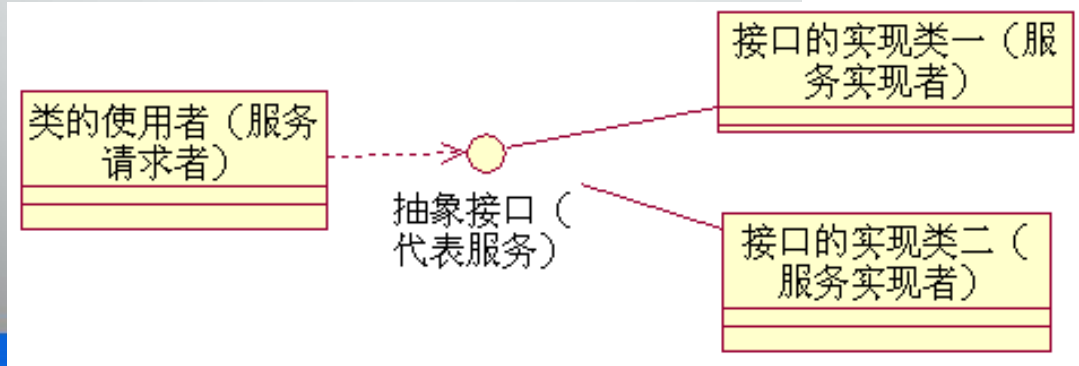
里氏替换原则

- 子类可以实现父类的抽象方法，但不能覆盖父类的非抽象方法。
- 子类中可以增加自己特有的方法。
- 当子类的方法重载父类的方法时，方法的前置条件（即方法的形参）要比父类方法的输入参数更宽松。
- 当子类的方法实现父类的抽象方法时，方法的后置条件（即方法的返回值）要比父类更严格。

依赖倒置原则（面向接口编程）

- 将依赖关系倒置为依赖接口
  - 上层模块不应该依赖于下层模块，它们共同依赖于一个抽象
  - 父类不能依赖子类，它们都要依赖抽象类
  - 抽象不能依赖于具体，具体应该要依赖于抽象





接口隔离

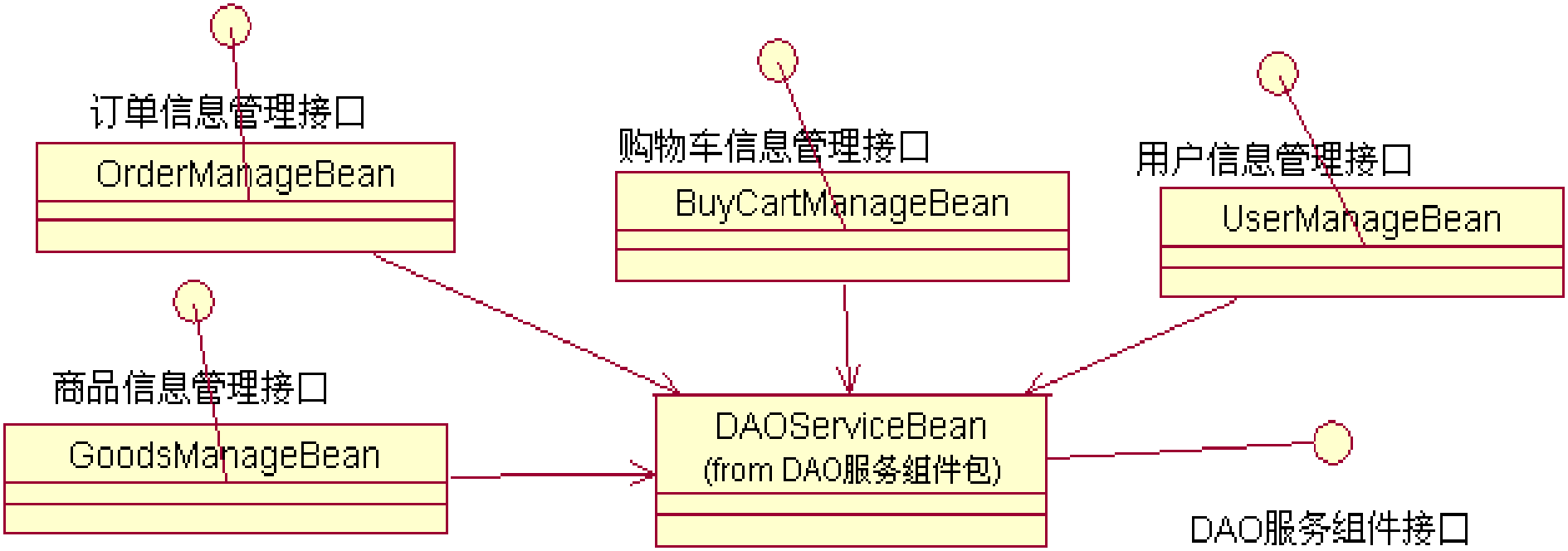
- 一个类对另外一个类的依赖性应当是建立在最小的接口上
- 客户端不应该依赖那些它不需要的接口（方法）

```
package com.px1987.webbbs.dao;
import java.util.*;
public interface UserManageDAOInterface {
    public Connection getConnection();
    public void closeDBCon();
    public ArrayList selectAllUserInfo();
    public UserInfoPO selectOneUserInfo(String registerUserID);
    public boolean insertOneUserInfo(UserInfoPO oneRegisterUserInfo);
    public boolean updateOneUserInfo(UserInfoPO oneUpdatedRegisterUserInfo);
    public boolean deleteOneUserInfo(String registerUserID);
}
```

接口的污染！！！

如何避免不良的接口设计

- 用多个专门的接口，而不使用单一的总接口。
- 一个接口就只代表一个角色
- 使用接口隔离原则拆分接口时，首先必须满足单一职责原则



合成复用原则

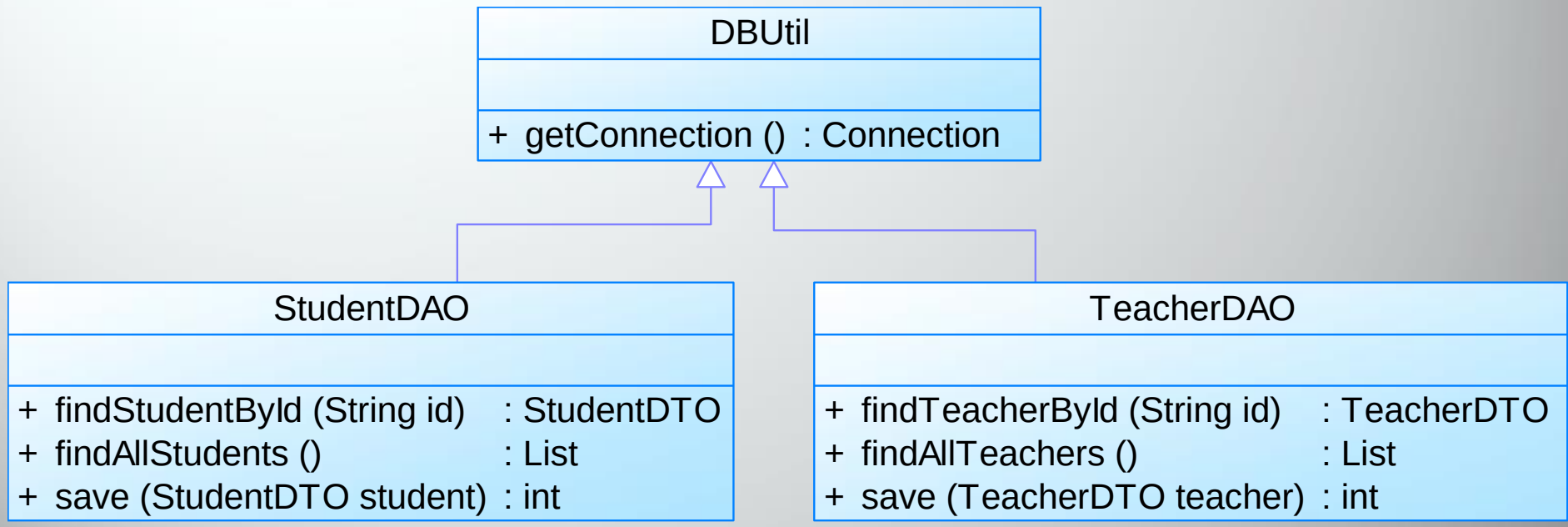
- 又称为组合/聚合复用原则
- 尽量使用对象组合，而不是继承来达到复用目的
- 一个新的对象里通过关联关系（包括组合关系和聚合关系）来使用一些已有的对象
- 新对象通过委派调用已有对象的方法达到复用其已有功能的目的

简言之：要尽量使用组合/聚合关系，少用继承！

？？？

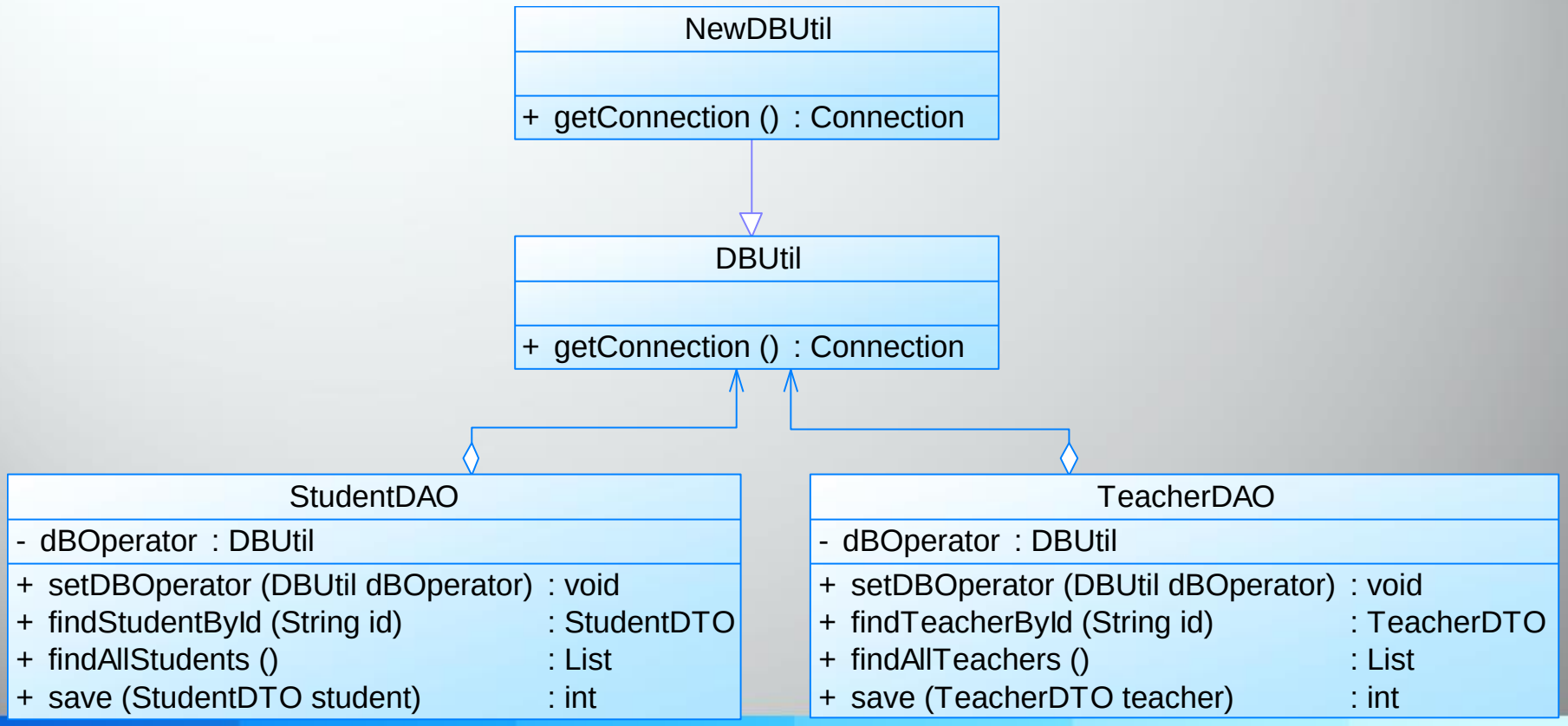
合成复用原则

- 继承复用：实现简单，易于扩展，没有足够的灵活性（“白箱”复用）



合成复用原则

- 组合/聚合复用：耦合度相对较低，选择性地调用成员对象的操作；可以在运行时动态进行。（“黑箱”复用）



迪米特法则

- 要求一个软件实体应当尽可能少的与其他实体发生相互作用
- 又称为最少知识原则
  - 不要和“陌生人”说话
  - 只与你的直接朋友通信
  - 每一个软件单位对其他的单位都只有最少的知识，而且局限于那些与本单位密切相关的软件单位

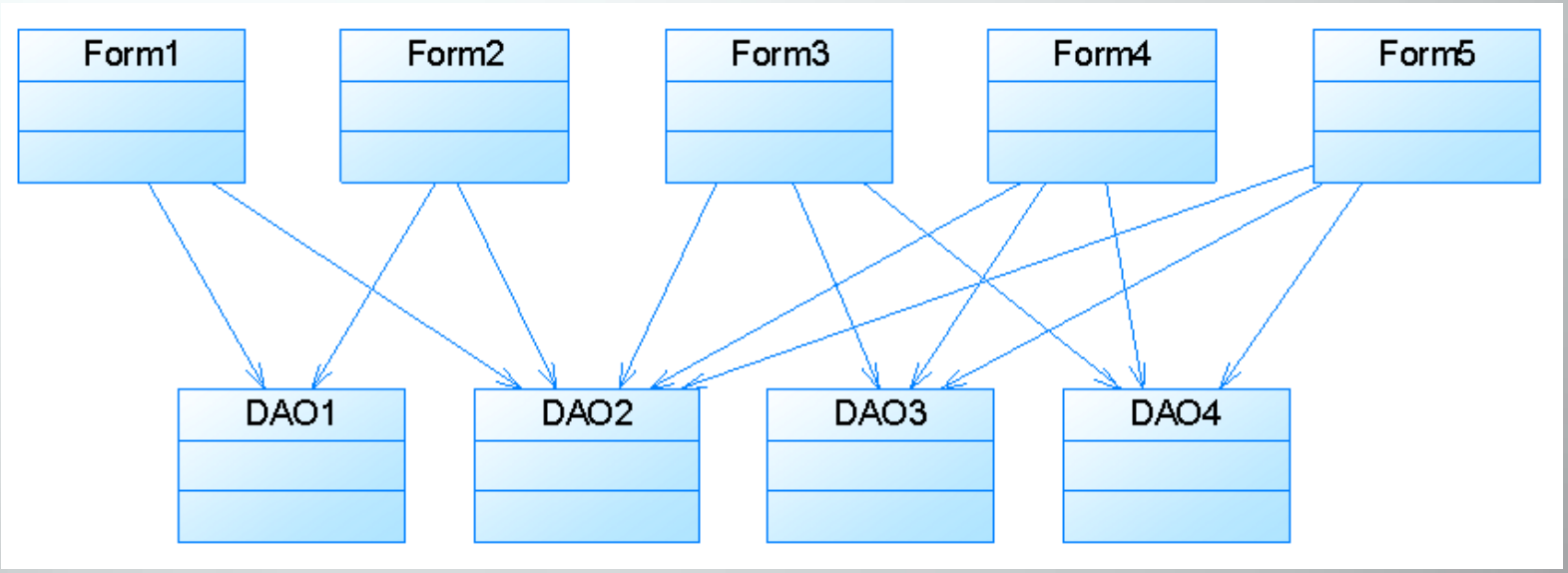
创建松耦合的类

迪米特法则应用时注意

- 在类的划分上，应该创建有弱耦合的类；
- 在类的结构设计上，每一个类都应当尽量降低成员的访问权限；
- 在类的设计上，只要有可能，一个类应当设计成不变类；
- 在对其他类的引用上，一个对象对其它对象的引用应当降到最低；
- 尽量降低类的访问权限；
- 谨慎使用序列化功能；
- 不要暴露类成员，而应该提供相应的访问器(属性)。

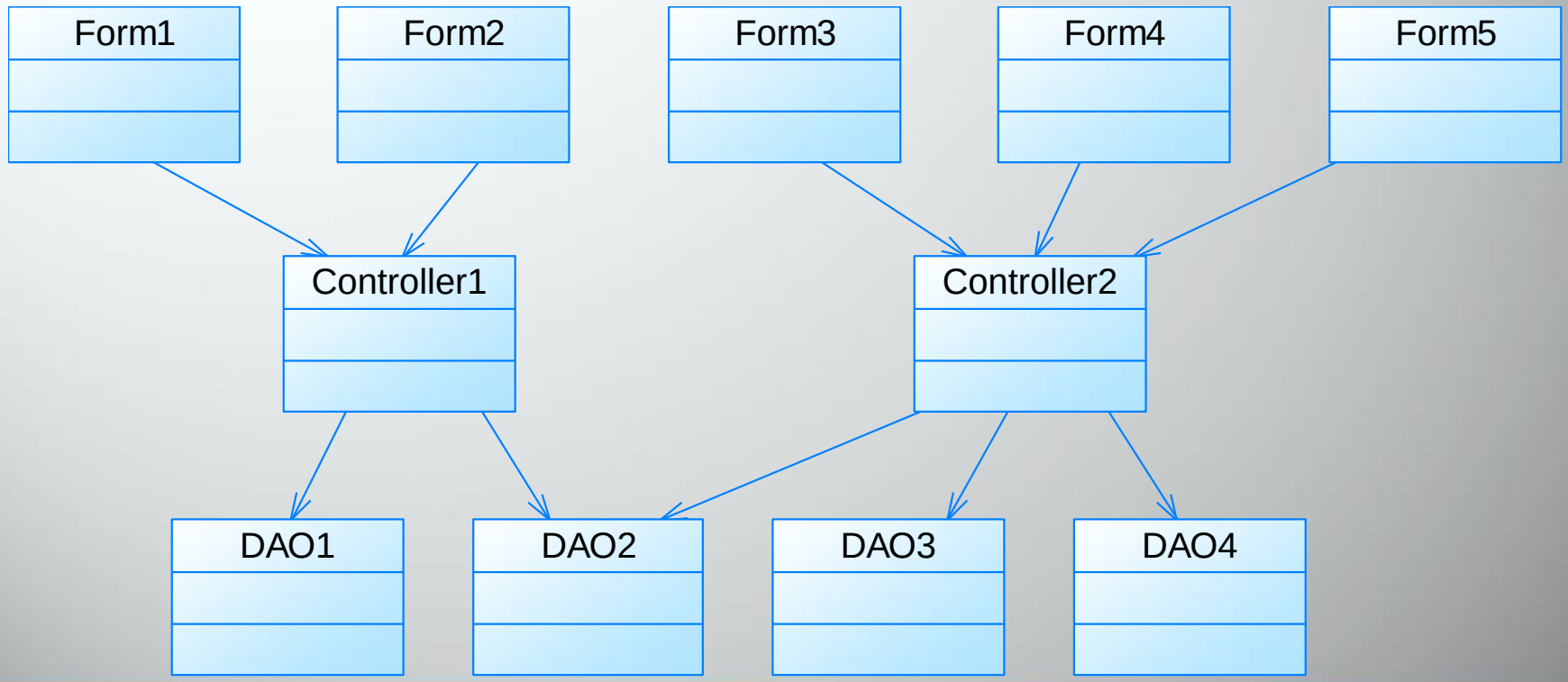
迪米特法则

- 某系统界面类(如Form1、Form2等类)与数据访问类(如DAO1、DAO2等类)之间的调用关系较为复杂，如图所示：



迪米特法则

- 如果其中的一个类需要调用另一个类的某中一个方法，可以通过第三者转发这个调用



好设计的原则

- 单一职责原则要求在软件系统中，一个类只负责一个功能领域中的相应职责。
- 开闭原则要求一个软件实体应当对扩展开放，对修改关闭，即在不修改源代码的基础上扩展一个系统的行为。
- 里氏代换原则可以通俗表述为在软件中如果能够使用基类对象，那么一定能够使用其子类对象。

好设计的原则

- 依赖倒转原则要求抽象不应该依赖于细节，细节应该依赖于抽象；要针对接口编程，不要针对实现编程。
- 接口隔离原则要求客户端不应该依赖那些它不需要的接口，即将一些大的接口细化成一些小的接口供客户端使用。
- 合成复用原则要求复用时尽量使用对象组合，而不使用继承。
- 迪米特法则要求一个软件实体应当尽可能少的与其他实体发生相互作用。

小结

- 课程主要介绍设计模式和体系结构模式
- 学习方式为环境、问题、解决方案
- 单例模式解决的问题是“独生子女”
- 单例模式的解决方案是利用Static
- 好设计的原则

Thank You，谢谢！