Chapter 10

- 构件设计

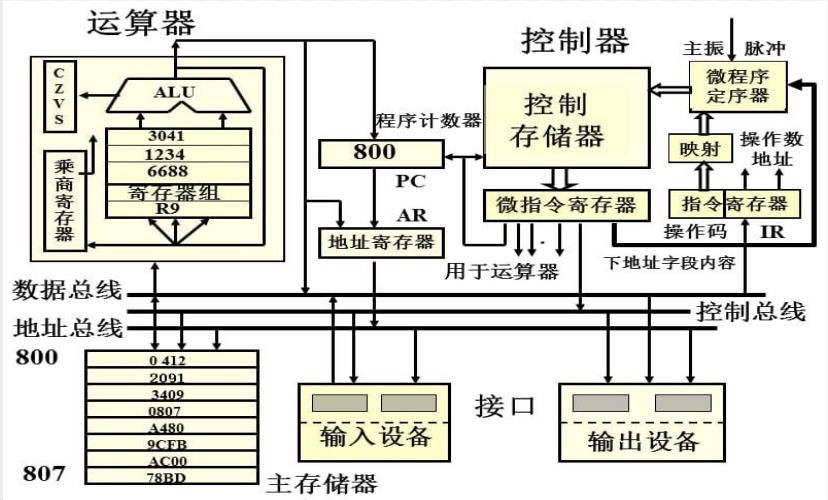
Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

10.1 构件设计基本概念

- 构件设计是在体系结构设计的基础上，对其中每个构件内部构成的进一步设计。
- 体系结构设计相当于确定了计算机的部件组成，连接关系等。
- 构件设计则着重于设计每个部件的内部细节，如运算器的内部电路和元件等。



10.1.1 什么是构件

- 构件是计算机软件中的一个模块化的构造块。它部件封装了实现并暴露一系列接口。

- 面向对象视角

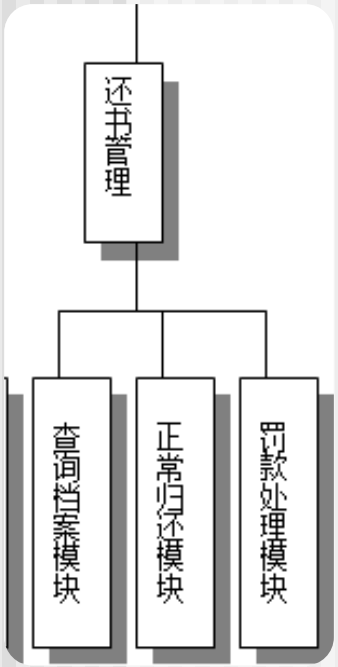
  - 构件封装了一组相互协作的类（也可以是一个类）。

- 结构化视角

  - 构件也称功能模块（如系统中图像读入模块、图像压缩模块）

  - 封装了数据结构、算法和接口

10.1.2 结构化视角下构件设计



接口： 接收读者号、图书号，归还日期， 返回罚款金额

处理功能和算法描述：

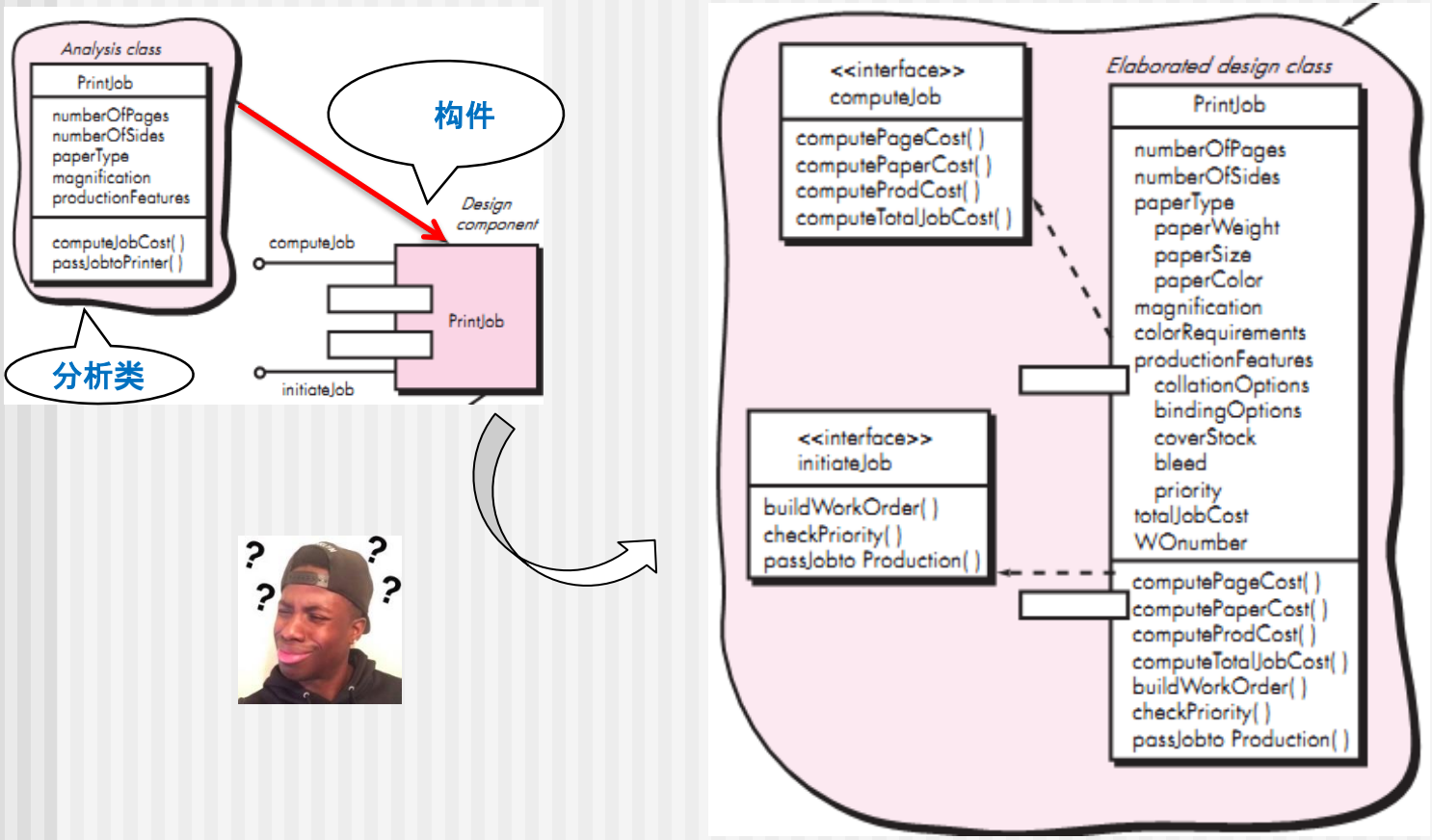
输入	罚款处理模块
读者号	通过读者号，图书号获取图书借阅日期
图书号	超期天数=归还日期-借阅日期
归还日期	罚款金额=超期天数*0.10 元
	INSERT INTO 罚款记录(读者号,读者名,图书号, 借阅日期，归还日期,罚款金额,操作员)
	Return 罚款金额

构件设计：细化模块内部的数据结构、算法和接口的过程。

10.1.3 面向对象视角下构件设计

设计元素包括子系统、构件、设计类

10.1.3 面向对象视角下构件设计



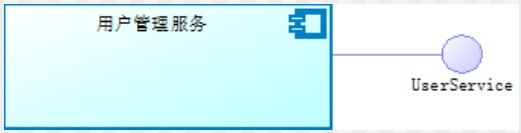
构件设计 ：确定设计类并对其细化的过程。

10.1.4 面向对象构件构成

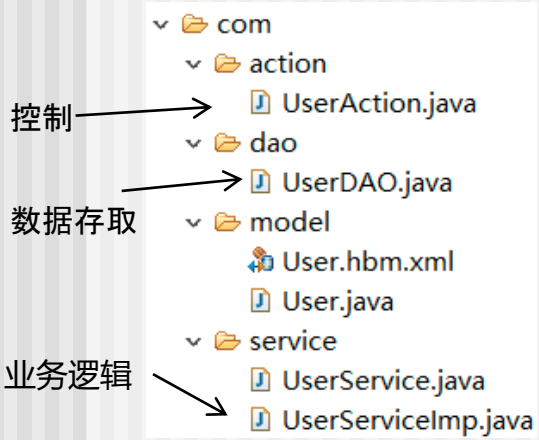
- 面向对象构件可能由多个设计类构成

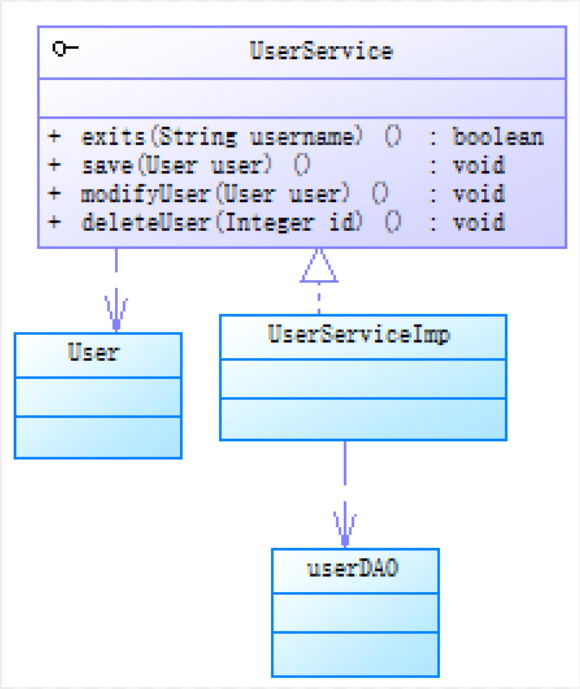
(1) 构件接口与实现需要严格分离。

(2) 基于设计框架，一个构件需要分解为多个设计类。



SSH框架中的Service构件





10.1.4 面向对象构件构成

- Service具体实现

```
public interface UserService {
    public boolean exits(String username);
    public void save(User user);
    public void modifyUser(User user);
    public void deleteUser(Integer id);
}

public class UserServiceImp implements UserService {
    @Override
    public boolean exits(String username){
        List<User> userList = userDAO.findByUsername(username);
        if(userList.size()>0)
            return true;
        else
            return false;
    }
```

10.1.5 面向对象构件设计关键问题

- 构件由哪些设计类组成？
- 设计类之间是什么关系，如何优化？
- 构件提供的接口是什么？
- 每个设计类的详细构成（属性和方法）？

前3个问题都与设计优化原则相关

通过设计优化，可以提高构件复用性，灵活适应系统变更，并减少副作用的传播。

10.2 构件设计指导原则

(1)开闭原则（OCP）

对扩展开放：构件的行为是可扩展或改变的

对修改关闭：当构件改变行为时，不要修改其源码

实现途径: 用抽象构建框架，用继承/实现扩展细节。

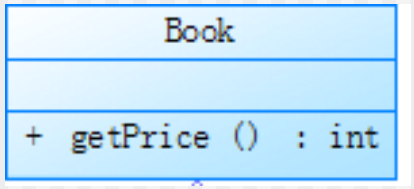
- 需求变化时，不是通过修改类本身来完成，而是通过扩展一个子类来改变构件的行为。

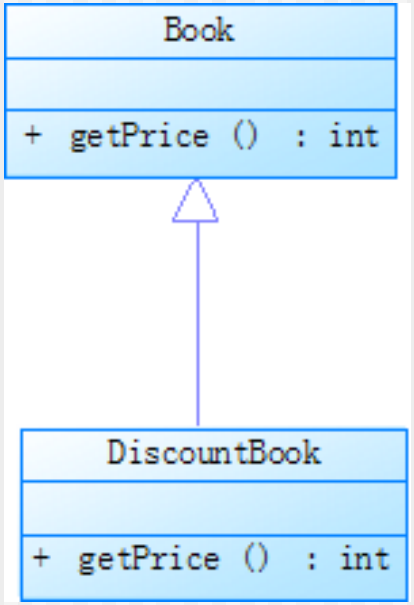
优点：尽量不修改原有的代码，实现一个热插拔的效果。

10.2 构件设计指导原则

(1)开闭原则（OCP）

- 如果某书店新增“打折书籍”业务，但打折书籍的计价方式与正常书籍不同；
- 是否应该修改Book类的计价方法呢？





10.2 构件设计指导原则

(2)LSP:Liskov Substitution Principle

LSP 原则用来指导继承关系中子类该如何设计，子类的设计要保证在替换父类的时候，不改变原有程序的逻辑以及不破坏原有程序的正确性。

简单理解：子类可以扩展父类的功能，但不能改变父类原有的功能。

LSP原则能减少代码冗余，避免运行期的类型判别（实现动态加载）。

10.2 构件设计指导原则

(2) LSP: Liskov Substitution Principle

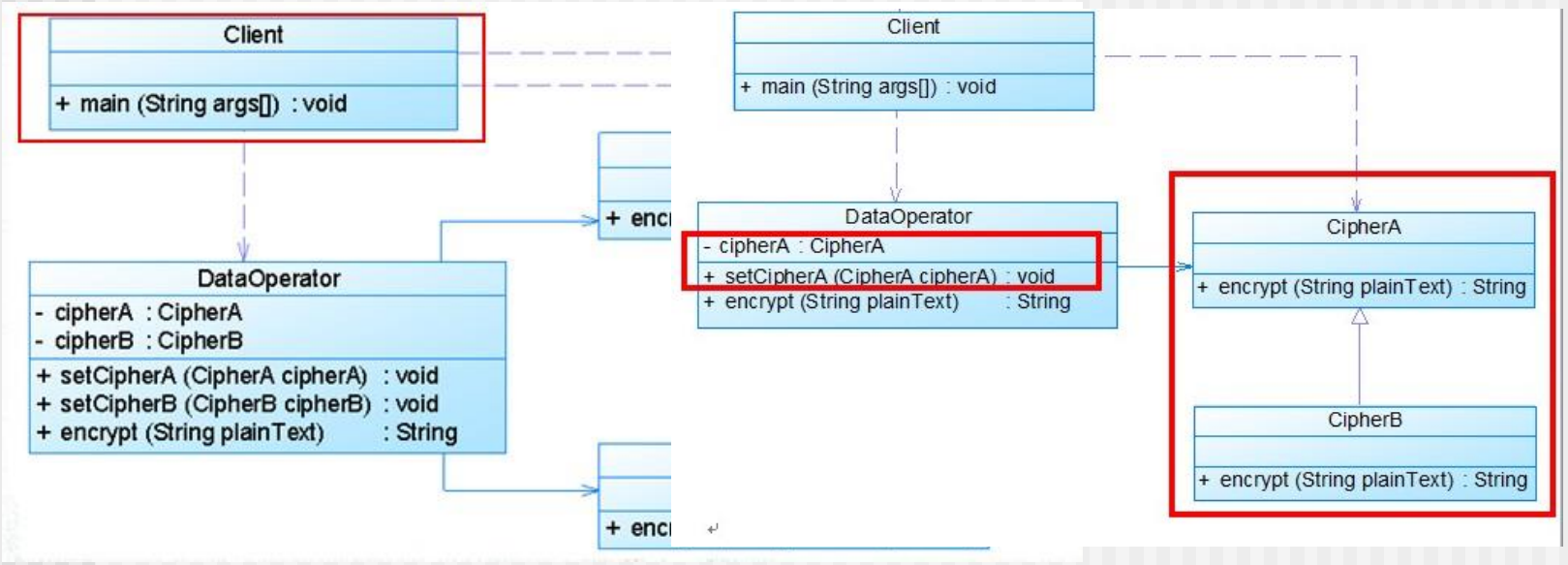
```
abstract class Bird {
     public abstract void fly() ;
}
public class Sparrow extends Bird {  //麻雀
       public void fly() {  System.out.println("In Sparrow flying ");}
}
public class Parrot extends Bird {//鹦鹉
       public void fly() {  System.out.println("In Parrot flying ");   }
}
public class Test {
    public static void showFly(Bird bird) {   //子类对象能替换父类对象
      bird.fly();
     }
    public static void main(String[] args) {
        showFly(new Sparrow());
        showFly(new Parrot());

    }
}
```

如果新增鸵鸟类继承自Bird，可以吗？

10.2 构件设计指导原则

某系统需要实现对重要数据的加密处理，在类DataOperator中需要调用加密类的加密算法，系统提供了两个不同的加密类，CipherA和CipherB，它们实现不同的加密方法，在DataOperator中可以选择其中的一个实现加密操作。如图所示：



如果增加了一种新的加密类，该怎么处理？如何优化上述类图结构

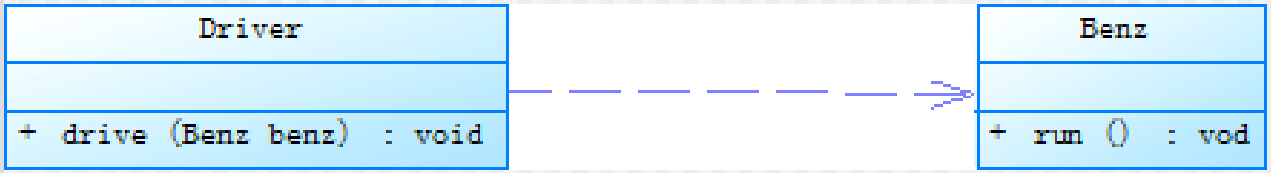
10.2 构件设计指导原则

(3) 依赖倒置原则： Dependency Inversion Principle

依赖于抽象，而非具体实现。 具体方法是：

- 首先，分离接口和实现。
- 将类之间的依赖关系建立在高层抽象（如抽象类、接口）基础上。

```
public class Driver {
    public void drive(Benz benz){
        benz.run();          //Driver直接依赖于Benz
    }
}
```



10.2 构件设计指导原则

- 如果司机不仅要开奔驰，还要开宝马车，又该怎么实现呢？
- 一种解决方案：修改Driver类增加新的drive方法

```
public class Driver {
    public void drive(Benz benz){
            benz.run();   //Driver依赖于Benz
    }
    public void drive(BMW bmw){
            bmw.run();   //Driver依赖于BMW
    }
}
```

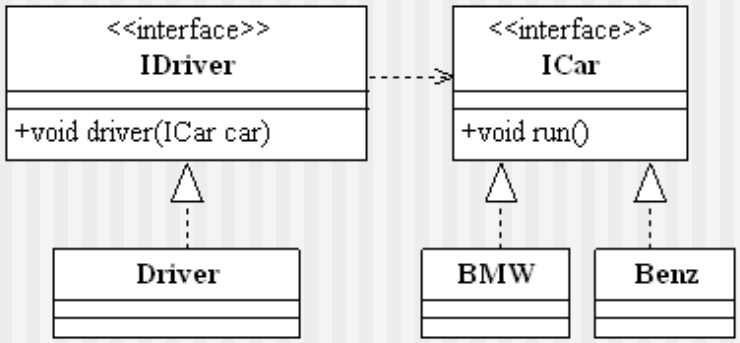
如果又增加一种新车怎么办？

- 缺点：系统缺乏稳定性

10.2 构件设计指导原则

- 依赖倒置原则实现

```
public class Driver implements IDriver{
   public void drive(ICar car){     //依赖于接口Icar
         car.run();    }
}
public class Client {
   public static void main(String[] args) {
      IDriver zhangSan = new Driver();
      ICar benz = new Benz();
      zhangSan.drive(benz);
      ICar bmw = new BMW();
      zhangSan.drive(bmw);
  }
}
```

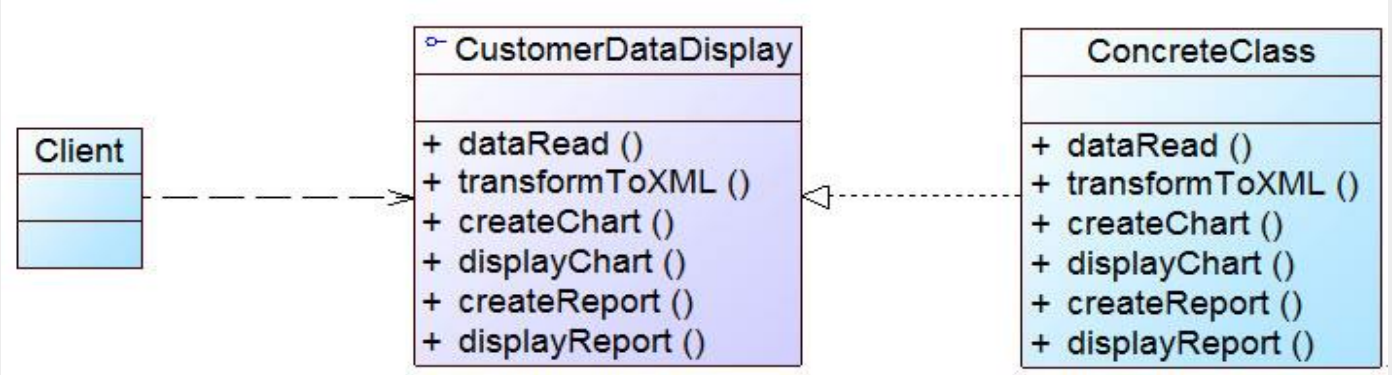


优点：减少类间的耦合性，提高系统的稳定性

10.2 构件设计指导原则

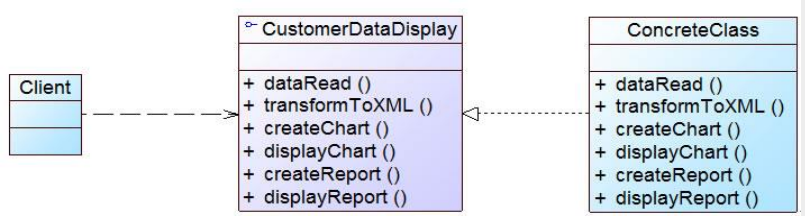
(4) 接口分离原则 Interface Segregation Principle

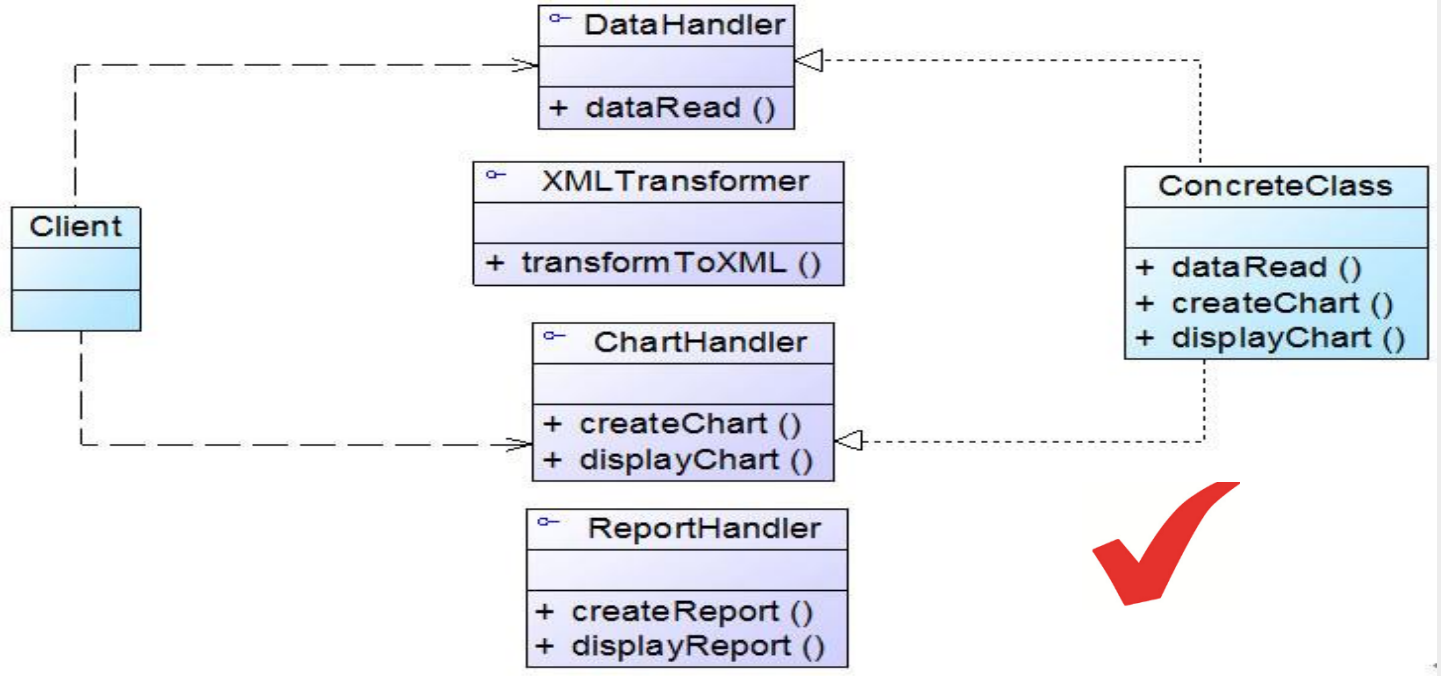
- 建立单一接口，不要建立臃肿庞大的接口。
- 即：接口尽量细化，同时接口中的方法尽量少



10.2 构件设计指导原则

(4) 接口分离原则 Interface Segregation Principle





10.2 构件设计指导原则

(5) 单一职责原则：一个类或接口只承担一类职责。

```
class Course {
 public void study(String courseName){
  if(“直播课”.equals(courseName)){                    //直播课学习职责
     System.out.println(courseName + "不能快进");
  }else{                                                    //录播课学习职责
     System.out.println(courseName + "可以反复回看");
  }
 }
}
```

改进：分解为两个设计类LiveCourse和ReplayCourse

10.2 构件设计指导原则

(6) 迪米特法则：尽量减少对象之间的交互，从而减小类之间的耦合。

- 内聚性和耦合性

内聚性意味着构件只封装那些密切相关的类，或者类只封装自身密切相关的属性和操作。

根据程度可细分为：功能内聚、分层内聚、通信内聚等

耦合性是类之间彼此联系程度的一种定性度量。

根据程度可细分为：内容耦合、共用耦合、控制耦合、标记耦合……

10.3 面向对象构件设计过程

- Step 1. 识别构件中与问题域相对应的设计类。包括界面类、控制类、实体类。
  - 一般可为每个用例设计一个控制类。如果多个用例中任务较为相似，也可以公用一个控制类。
  - 如果实体类的职责比较单一，直接将其映射为设计类；
  - 如果某类的职责过于复杂，应该将其分解为多个设计类（单一职责）。

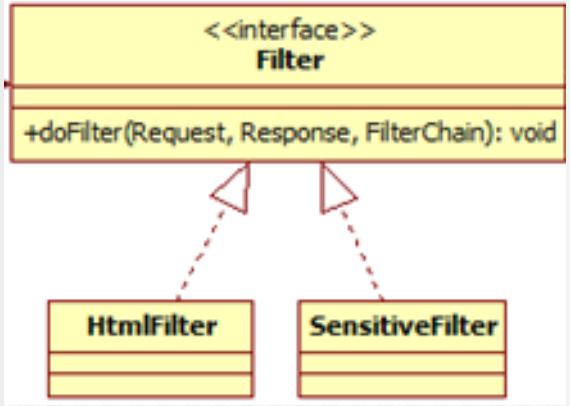
10.3 面向对象构件设计过程

- Step 2. 识别构件中与基础设施相对应的设计类。
  - 这些设计类往往在分析模型中并没有定义。
  - 如：
    - 通讯构件 (JMS)
    - 事务管理类(TransactionManager)
    - 安全服务类(SecurityManager)

10.3 面向对象构件设计过程

例如：为提高安全性，需要增加权限控制基础设施类

```
public class HtmlFilter implements Filter {
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain
chain) {
        HttpServletRequest request = (HttpServletRequest) req;
        HttpServletResponse response = (HttpServletResponse) resp;
        User user = (User) request.getSession().getAttribute("user"); // 检查用户
是否登录
        if (user == null) {   // 没登录，登录去
            ……
        }
```

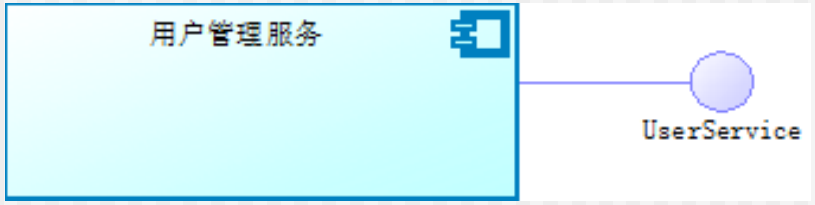


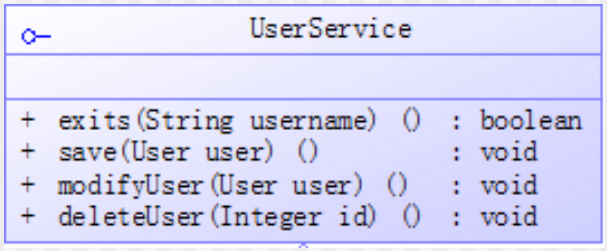
面向对象构件设计过程

- Step 3. 细化所有的设计类，定义该类中所有的接口、属性和操作。
  - Step 3a. 通过设计时序图或协作图说明协作细节。
    - ✓ 将时序图中职责分解为具体操作，并分配给相关类；
    - ✓ 定义操作对应的方法名，方法参数，返回值、可见性等。

10.3 面向对象构件设计过程

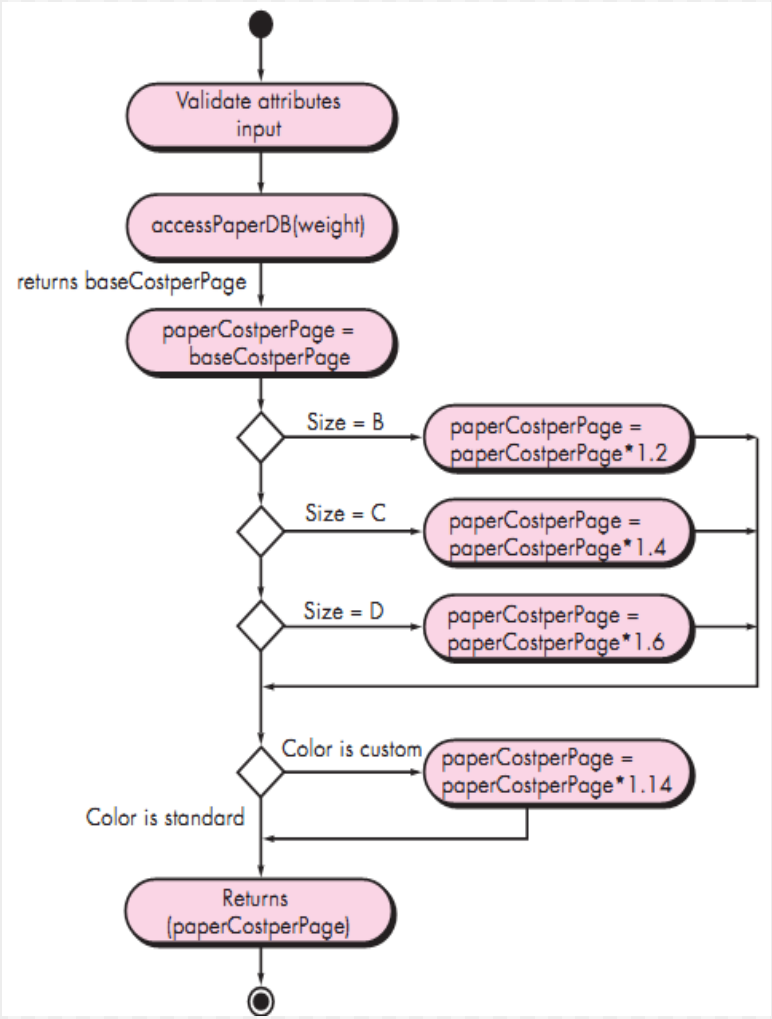
- Step 3b. 为每个构件确定合适的接口。
  - ✓ 简单构件可直接定义public方法作为接口；
  - ✓ 复杂构件单独定义接口类，构件负责实现接口；
  - ✓ 确定接口的参数；





10.3 面向对象构件设计过程

- Step 3c. 细化属性，并定义相应的数据类型和数据结构。
- Step 3d. 描述类成员方法中的处理逻辑。
  - ✓绘制程序流程图或活动图
  - ✓如定义类PrintJob中的方法computePaperCost()



10.3 面向对象构件设计过程

- Step 4. 说明持久数据源 (数据库或文件)，并确定管理数据源所需要的类.
  - ✓ 对象可暂存于内存中，但要持久保存，需要通过数据存储实现。
  - ✓ 可以通过对象关系映射（ORM）将类映射为表，以简化数据库使用。

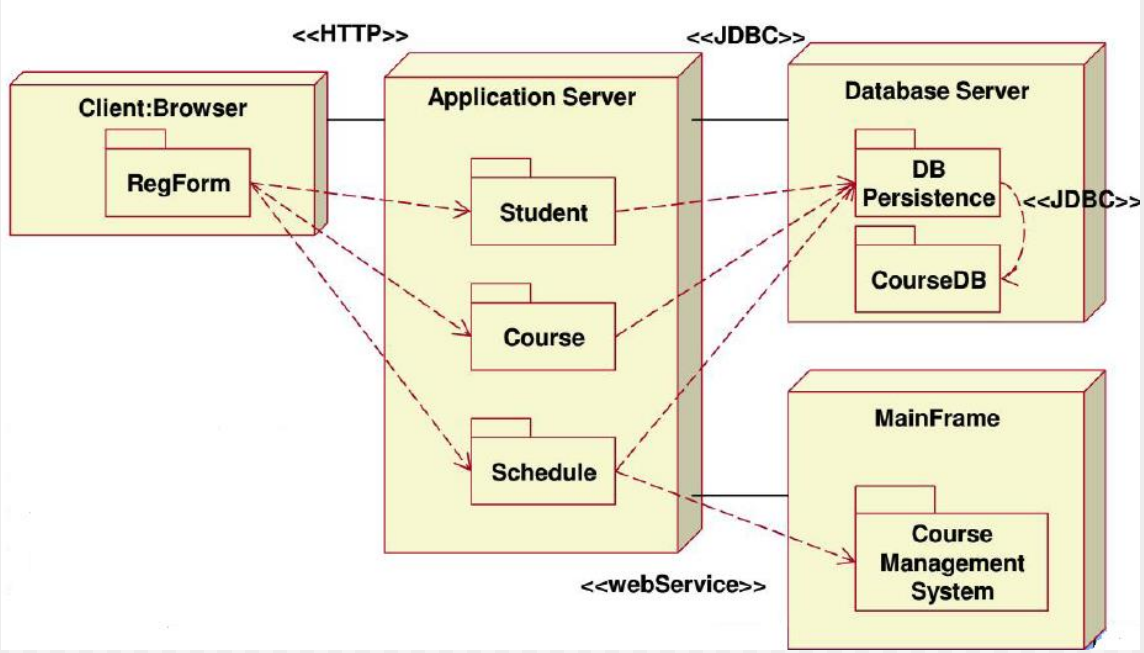
目前流行的**ORM**产品：

- Apache OJB
- Hibernate
- JPA(Java Persistence API)

10.3 面向对象构件设计过程

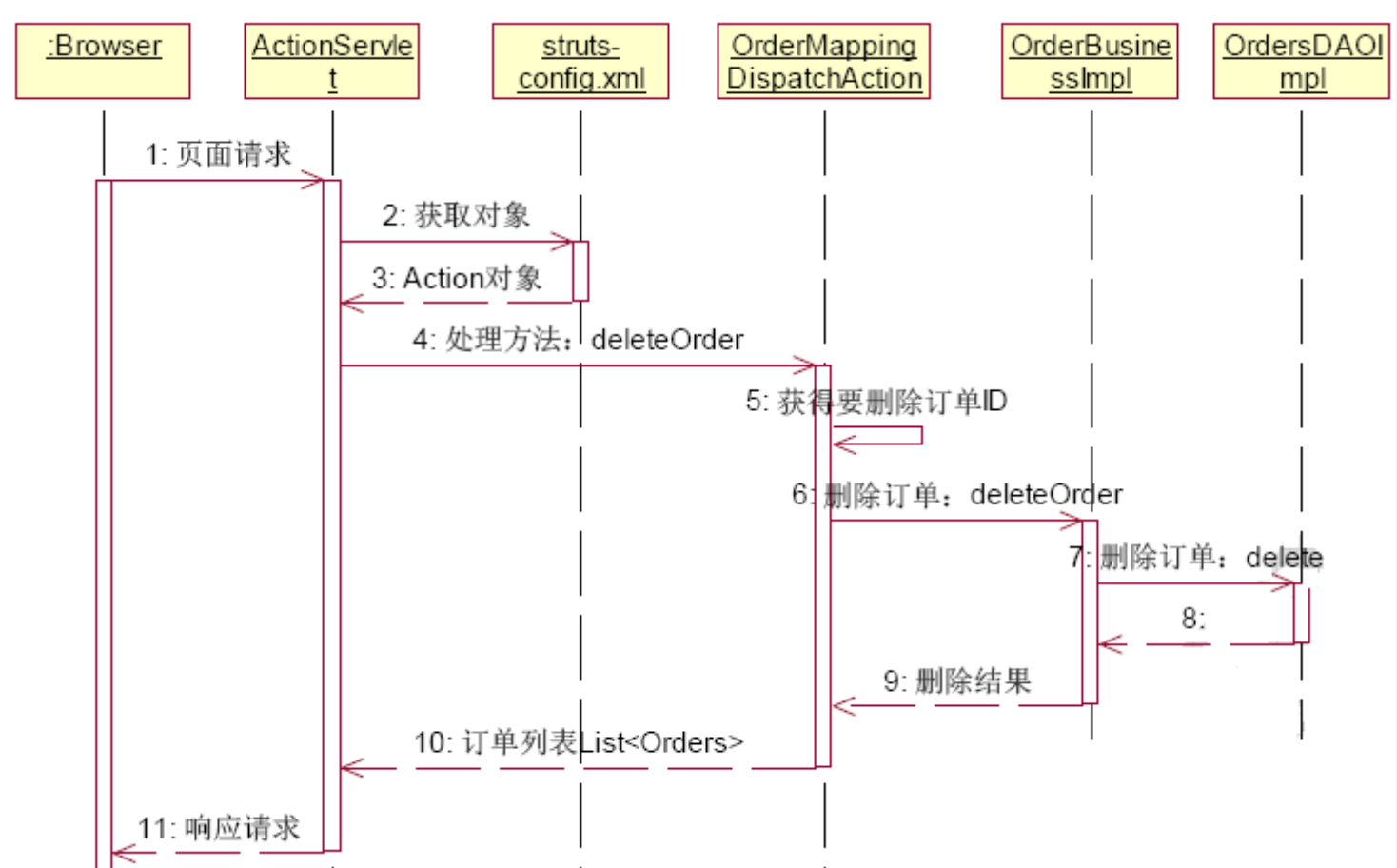
Step 5. 细化部署图以提供额外的实现细节。

- ✓ 为系统选择硬件配置和运行平台；
- ✓ 将类、包、子系统分配到各个硬件结点上



10.4构件设计实例

以某电商系统中“订单管理”子系统为例，该子系统包括订单删除，订单查询，查看订单详情等。



10.4 构件设计实例

分为：

界面构件，包括myorders.jsp

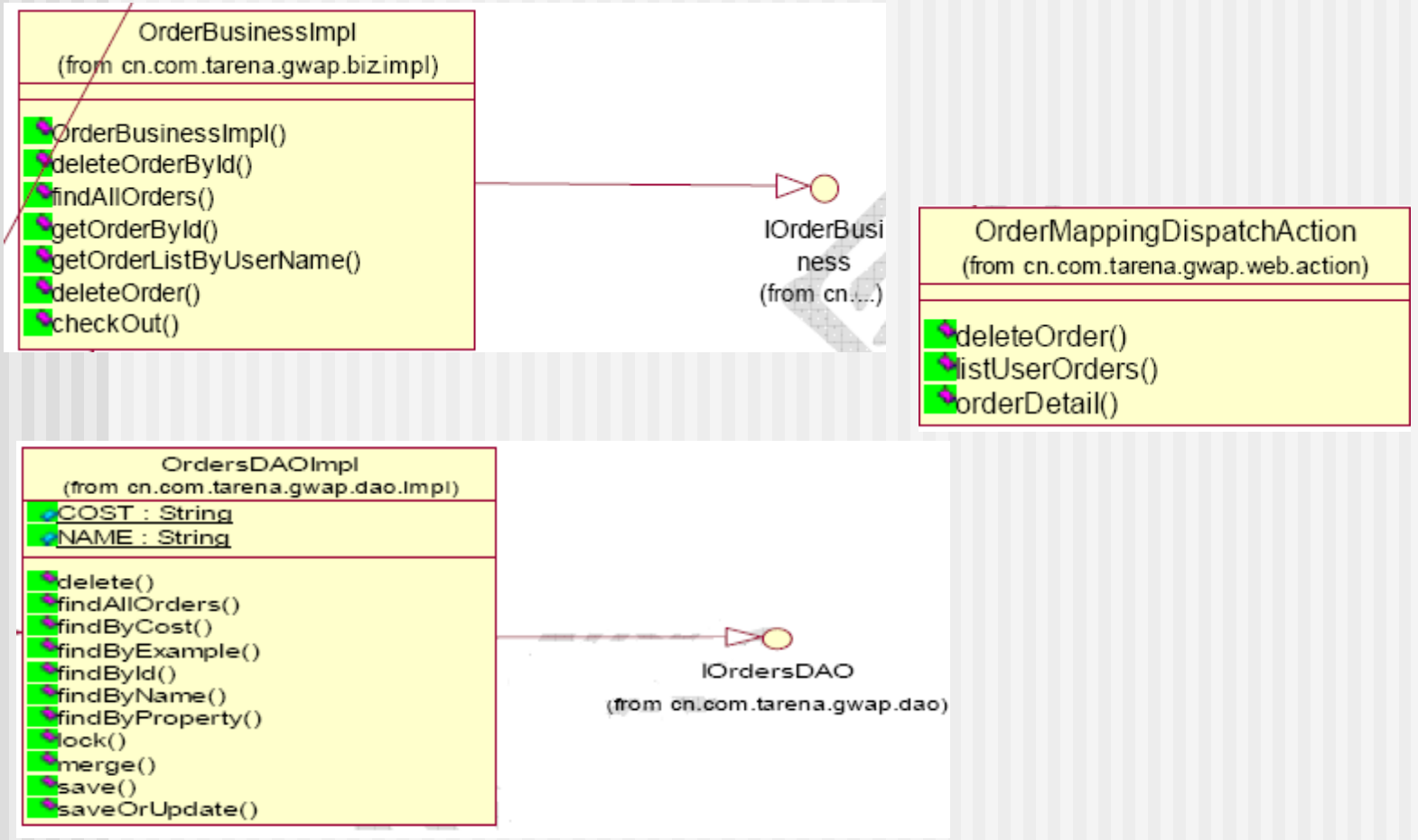
控制构件，包括orderMappingDispatchAction类，可根据参数执行删除，新增，查询等控制。

模型构件，包括OrderBusiness和OrderDAOImpl类

➢ 组件定义

View	/order/myorders.jsp	
Action	OrderMappingDispatchAction	deleteOrder 方法
Service	OrderBusiness	订单管理业务类
Dao	OrdersDAOImpl	订单管理数据持久层操作

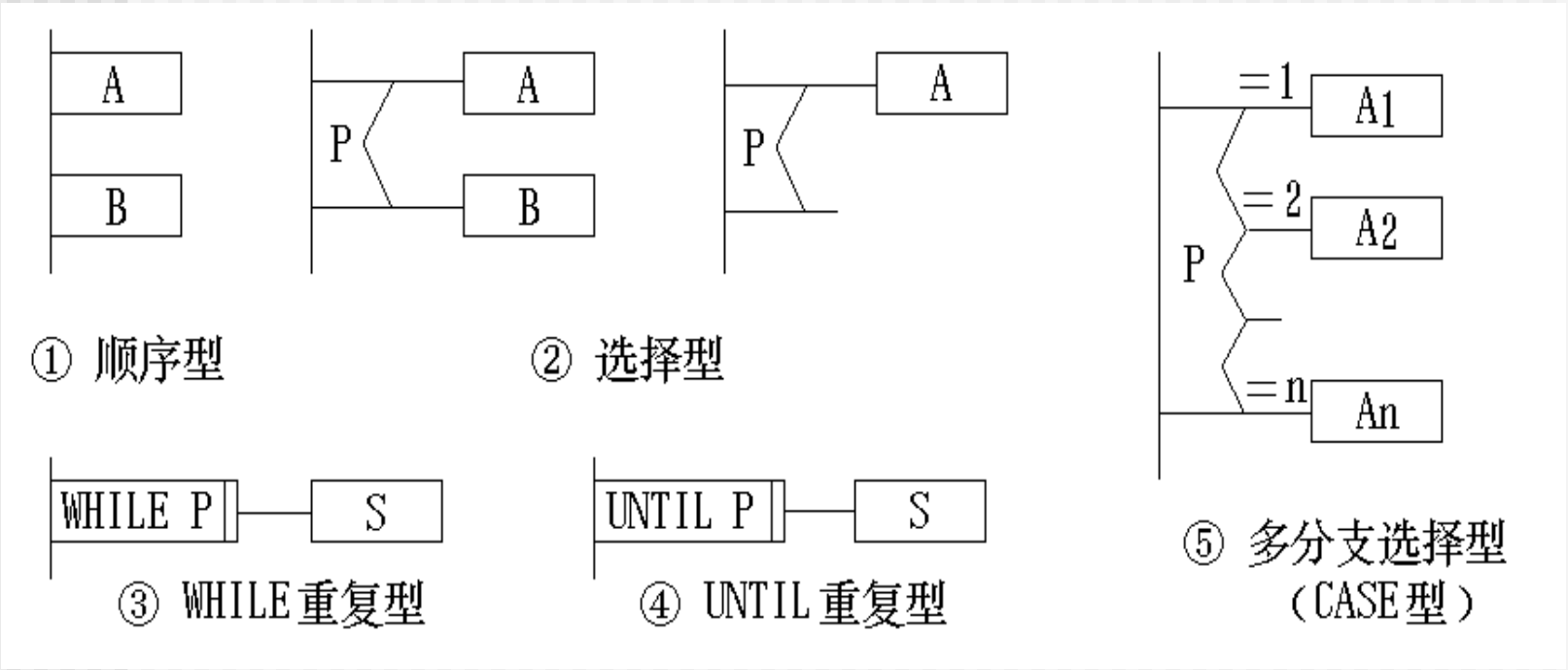
10.3 构件设计实例


OrderBusinessImpl
(from cn.com.tarena.gwap.bizimpl)
OrderBusinessImpl()
deleteOrderById()
findAllOrders()
getOrderById()
getOrderListByUserName()
deleteOrder()
checkOut()
IOrderBusi
ness
(from cn....)
OrderMappingDispatchAction
(from cn.com.tarena.gwap.web.action)
deleteOrder()
listUserOrders()
orderDetail()
OrdersDAOImpl
(from cn.com.tarena.gwap.dao.impl)
COST : String
NAME : String
delete()
findAllOrders()
findByCost()
findByExample()
findById()
findByName()
findByProperty()
lock()
merge()
save()
saveOrUpdate()
IOrdersDAO
(from cn.com.tarena.gwap.dao)

10.5 传统构件设计过程

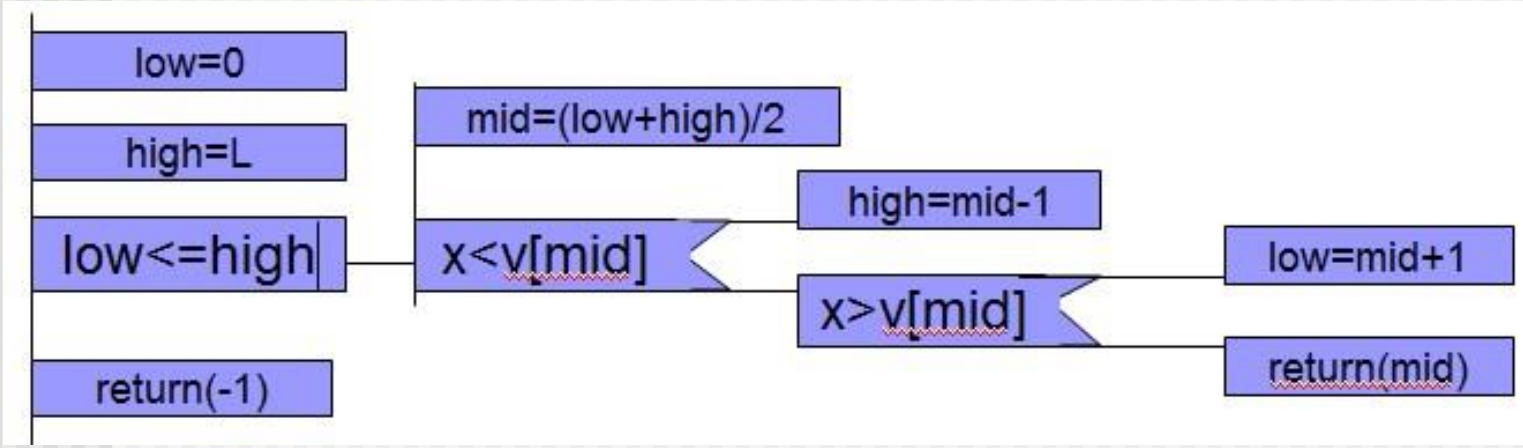
- 确定每个模块的算法，用工具描述算法逻辑。
- 确定每一模块的数据结构。
- 确定模块接口细节。
- 常用算法描述工具包括
  - ✓ 程序流程图
  - ✓ 问题分析图(PAD)
  - ✓ 盒图(N-S图)
  - ✓ 伪代码(PDL)

Program Analysis Diagram (PAD)

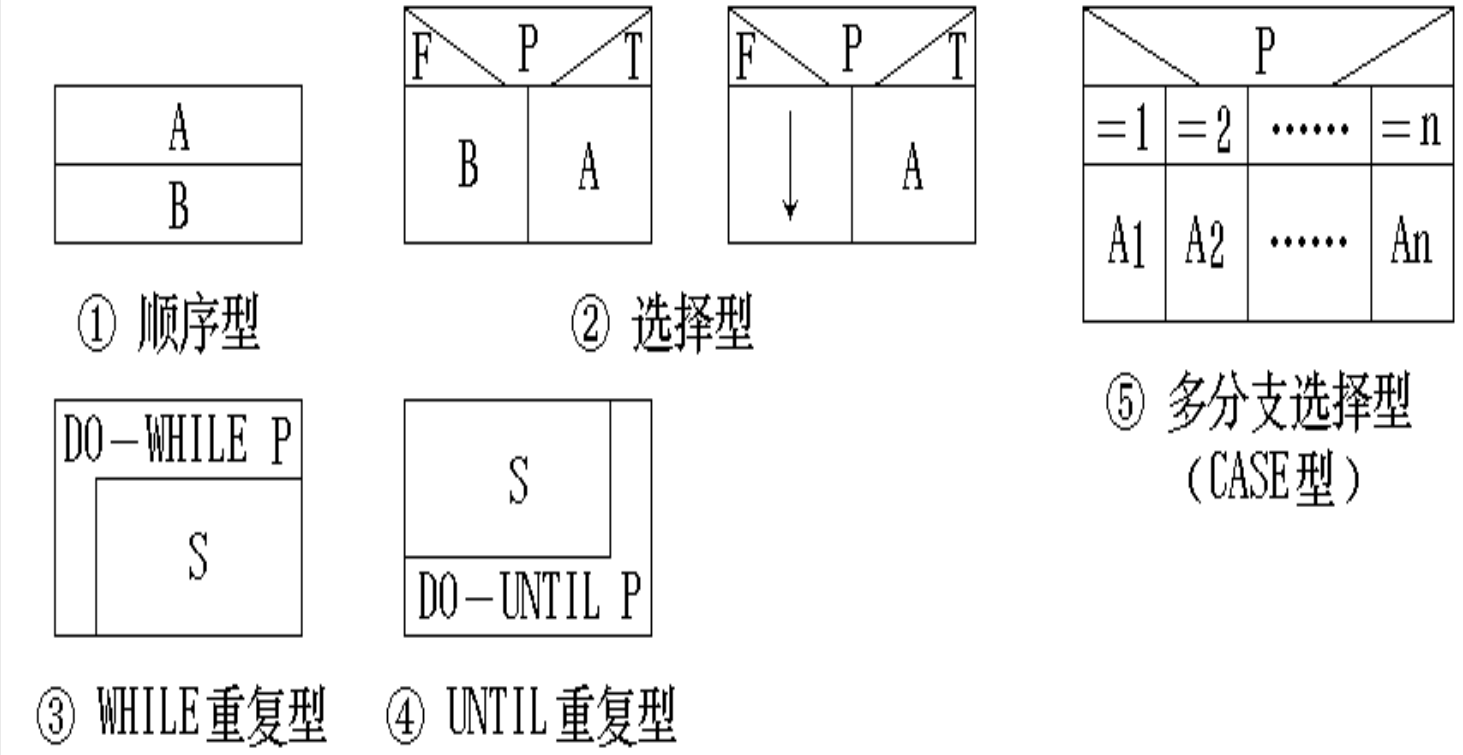


Program Analysis Diagram (PAD)

例：模块binary(x,v,n,p)完成判断一个特定值x是否出现在已经按递增顺序排好序的数组v中。如果存在则返回相应下标mid ，否则返回值-1。



NS Diagram


A
B
① 顺序型
F P T
B A
F P T
A
② 选择型
P
=1 =2 ····· =n
A1 A2 ····· An
⑤ 多分支选择型
(CASE型)
DO-WHILE P
S
③ WHILE重复型
S
DO-UNTIL P
④ UNTIL重复型