软件设计师案例重点

案例一 数据流图

1、数据流图的注意事项

名称	注意事项
数据流	数据流的流向： ①从一个加工流向另一个加工 ②从加工流向数据存储（写） ③从数据存储流向加工（读） ④从外部实体流向加工（输入） ⑤从加工流向外部实体（输出）
加工	加工要有输入输出： ①有输入没有输出--黑洞 ②有输出没有输入--奇迹 ③输入不足以产生输出--灰洞
外部实体	外部实体可以是人、物品、其他系统。

2、结构化语言对于加工的描述示例

```
学士学位授予的结构化语言描述
IF 学分达到要求        THEN
  IF 未受留校察看处分  THEN
     授予学士学位
  ELSE (受到留校察看处分)
     不授予学士学位，毕业一年以后可再次申请
  ENDIF
ELSE (学分未达到要求)
  不授予学士学位
ENDIF
```

案例二 ER 图

1、关系型数据库数据的约束条件：

①实体完整性。实体完整性是指实体的主属性不能取空值。

②参照完整性。在关系数据库中主要是值得外键参照的完整性。若 A 关系中的某个或者某些属性参照 B 或其他几个关系中的属性，那么在关系 A 中该属性要么为空，要么必须出现 B 或者其他的关系的对应属性中。

③用户定义完整性。用户定义完整性反映的某一个具体应用所对应的数据必须满足一定的约束条件。

2、E-R 图向关系模式的转换，具体转换规则如下：

（1）一个实体转换为一个关系模式，实体的属性就是关系的属性，实体的主键就是关系的主键。

（2）一个 1:1 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。

（3）一个 1:n 联系可以转换为一个独立的关系模式，也可以与任意 n 端对应的关系模式合并。

（4）一个 m:n 联系转换为一个独立的关系模式，与该联系相连的各实体的主键以及联系本身的属性均转换为关系的属性，而关系的主键为各实体主键的组合。

（5）三个以上实体间的一个多元联系可以转换为一个独立的关系模式，与该联系相连的各实体的主键和联系本身的属性均转换为关系的属性，而关系的主键为各实体主键的组合。

（6）多值属性的处理。如果 E-R 图中某实体具有一个多值属性，则应该进行优化，把该属性提升为一个实体，通常称为弱实体；或者在转化为关系模式时，将实体的主键与多值属性单独构成一个关系模式。

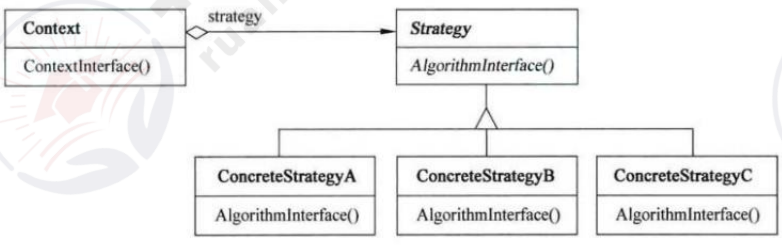
（7）BLOB 型属性的处理。应采用的设计方案是将 BLOB 属性与关系的主键独立为一个关系模式。
（8）派生属性的处理。因为派生属性可由其他属性计算得到，因此，在转化成关系模式时，通常不转换派生属性。
（9）在面向对象的模型中，关系模式就对应类，关系模式的属性就对应类的属性。

案例三 uml 图
1、用例之间的关系有三种：包含 include、扩展 extend、泛化 generalize。在包含关系中基用例必须和子用例一起使用才够完整。扩展关系是对基用例的扩展，基用例是一个完整的用例，即使没有子用例的参与，也可以完成一个完整的功能。
2、类之间的关系有：依赖、泛化、关联（组合、聚合)、实现。
3、类的分类：
①实体类。实体类对应系统需求中的每个实体，它们通常需要保存在永久存储体中，一般使用数据库表或文件来记录，实体类既包括存储和传递数据的类，还包括操作数据的类。实体类来源于需求说明中的名词，如学生、商品等。
②控制类。控制类用于体现应用程序的执行逻辑，提供相应的业务操作，将控制类抽象出来可以降低界面和数据库之间的耦合度。
③边界类。边界类用于对外部用户与系统之间的交互对象进行抽象，主要包括界面类，如对话框、窗口、菜单等。

案例四 C 语言算法
1、算法的时间复杂度

排序方法	时间复杂度（最好）	时间复杂度（最坏）	时间复杂度（平均）	稳定性
直接插入	O(n)	$O(n^2)$	$O(n^2)$	稳定
简单选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	不稳定
冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	稳定
希尔排序	O(n)	$O(n^{1.3})$	$O(n^{1.3})$	不稳定
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	稳定

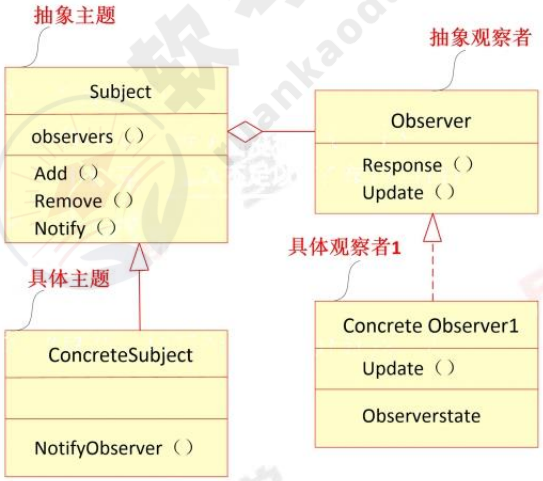
案例五、六 面向对象程序语言
1、策略模式
定义一系列的算法，把它们一个个封装起来，并且使它们可以相互替换。此模式使得算法可以独立于使用它们的客户而变化。



- Strategy（策略）定义所有支持的算法的公共接口。Context 使用这个接口来调用某 ConcreteStrategy 定义的算法。
- ConcreteStrategy（具体策略）以 Strategy 接口实现某具体算法。
- Context（上下文）用一个 ConcreteStrategy 对象来配置;维护一个对 Strategy 对象的引用;可定义一个接口来让 Strategy 访问它的数据。

2、观察者模式

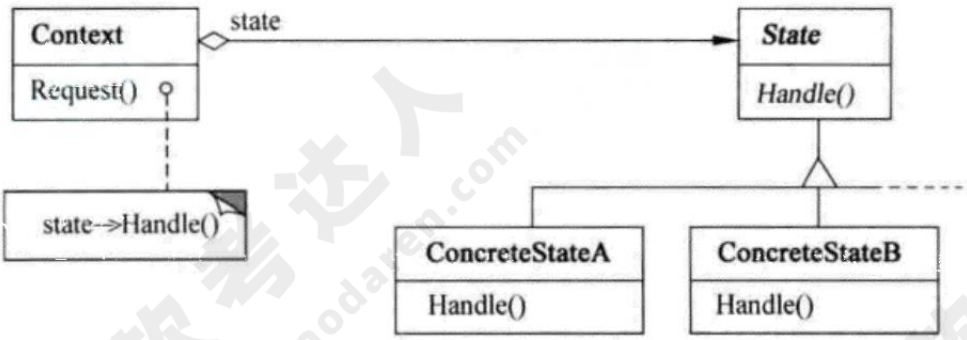
观察者模式定义了对象间的一种一对多依赖关系，使得每当一个对象改变状态，则所有依赖于它的对象都会得到通知并被自动更新。



- 抽象主题（Subject）：将全体观察者对象的引用存入至某一个列表中，提供某一接口用于添加或删除观察者对象。
- 具体主题（ConcreteSubject）：实现抽象主题中的通知方法，当具体主题的内部状态发生改变时，通知所有注册过的观察者对象。
- 抽象观察者（Observer）：定义某一类接口帮助全体具体观察者，获取更新通知。
- 具体观察者（Concrete Observer）：抽象观察者的实现，以便在得到目标的更改通知时更新自身的状态。

3、状态模式

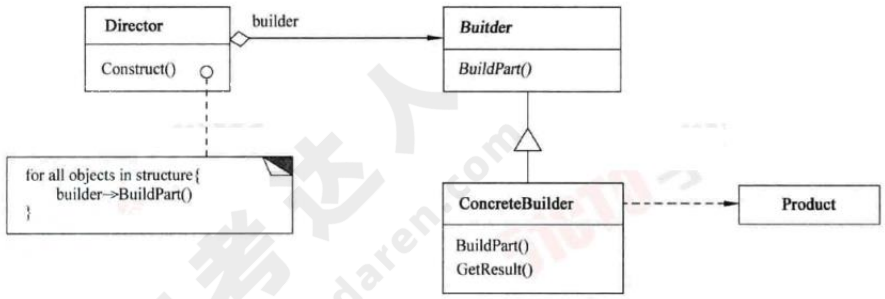
对于对象内部的状态，允许其在不同的状态下，拥有不同的行为，对状态单独封装成类。



- Context（上下文）定义客户感兴趣的接口，维护一个 ConcreteState 子类的实例，这个实例定义当前状态。
- State（状态）定义一个接口以封装与 Context 的一个特定状态相关的行为。
- ConcreteState（具体状态子类）指每个子类实现与 Context 的一个状态相关的行为。

4、建造者模式

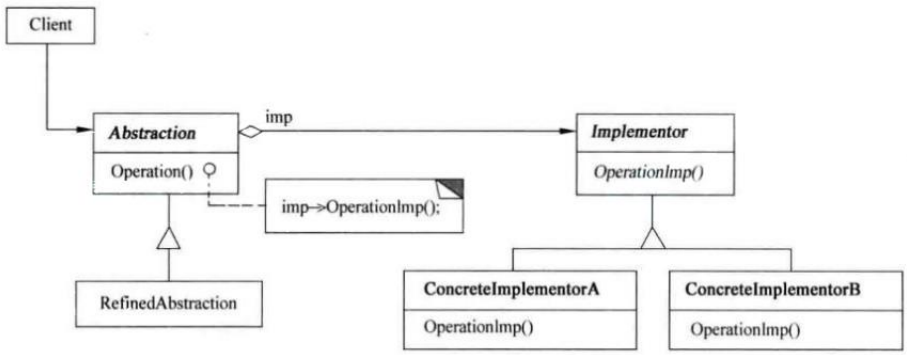
将一个复杂的对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。



- Builder：抽象建造者。它声明为创建一个 Product 对象的各个部件指定的抽象接口。
- ConcreteBuilder：具体建造者。实现抽象接口，构建和装配各个部件。
- Director：指挥者。构建一个使用 Builder 接口的对象。它主要是用于创建一个复杂的对象，它主要有两个作用，一是：隔离了客户与对象的生产过程，二是：负责控制产品对象的生产过程。
- Product：产品角色。一个具体的产品对象。

5、桥接模式

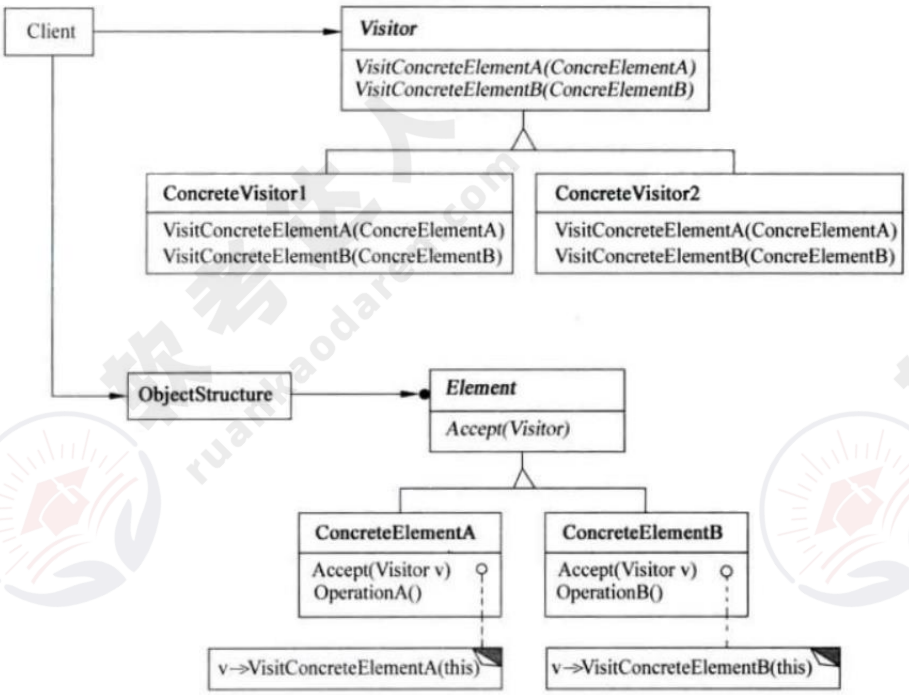
将抽象部分与它的实现部分分离，使它们都可以独立地变化。



- Abstraction 定义抽象类的接口，维护一个指向 Implementor 类型对象的指针。 RefinedAbstraction 扩充由 Abstraction 定义的接口。
- Implementor 定义实现类的接口，该接口不一定要与 Abstraction 的接口完全一致;事实上这两个接口可以完全不同。一般来说，Implementor 接口仅提供基本操作，而 Abstraction 定义了基于这些基本操作的较高层次的操作。
- ConcreteImplementor 实现 Implementor 接口并定义它的具体实现。

6、访问者模式

表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用与这些元素的新操作。



- Visitor（访问者）为该对象结构中 ConcreteElement 的每一个类声明一个 Visit 操作。该操作的名字和特征标识了发送 Visit 请求给该访问者的那个类，这使得访问者可以确定正被访问元素的具体的类。这样访问者就可以通过该元素的特定接口直接访问它。
- ConcreteVisitor（具体访问者）实现每个有 Visitor 声明的操作，每个操作实现本算法的一部分，而该算法片段乃是对应于结构中对象的类。ConcreteVisitor 为该算法提供了上下文并存储它的局部状态。这一状态常常在遍历该结构的过程中累积结果。
- Element（元素）定义以一个访问者为参数的 Accept 操作。
- ConcreteElement（具体元素）实现以一个访问者为参数的 Accept 操作。
- ObjectStructure（对象结构）能枚举它的元素;可以提供一个高层的接口以允许该访问者访问它的元素;可以是一个组合或者一个集合，如一个列表或一个无序集合。