软件工程：第十一章 面向对象实现

导学目标

- 了解面向对象实现的任务、目标以及注意事项
- 了解面向对象语言的技术和特点
- 了解程序设计风格的准则
- 了解面向对象设计的测试策略
- 了解如何去设计面向对象测试用例

第一节 面向对象实现概述

1、面向对象实现的任务

- 将面向对象设计结果翻译成面向对象程序
- 测试并调试面向对象程序

2、面向对象测试的目标

- 用尽可能低的测试成本发现尽可能多的软件错误

3、注意事项

- 面向对象独有的封装、继承和多态机制增加了测试的难度

第二节 程序设计语言

- 面向对象设计的结果既可以支持面向对象语言，也可以用非面向对象语言实现
- 使用面向对象的语言时，由于语言本身的特点，编译程序可以自动地将面向对象的概念映射到目标程序中
- 使用非面向对象语言编写面向对象程序时，须由程序员自身把面向对象的概念映射到目标程序中

1、选择编程语言的关键因素

- 一致的表示方法
- 可重用性
- 可维护性

2、面向对象语言的技术特点

- 支持类与对象概念的机制
- 实现整体-部分结构的机制
- 实现一般-特殊结构的机制
- 实现属性和服务机制
- 类型检查
- 类库
- 效率
- 持久保存对象
- 参数化类

- 开发环境

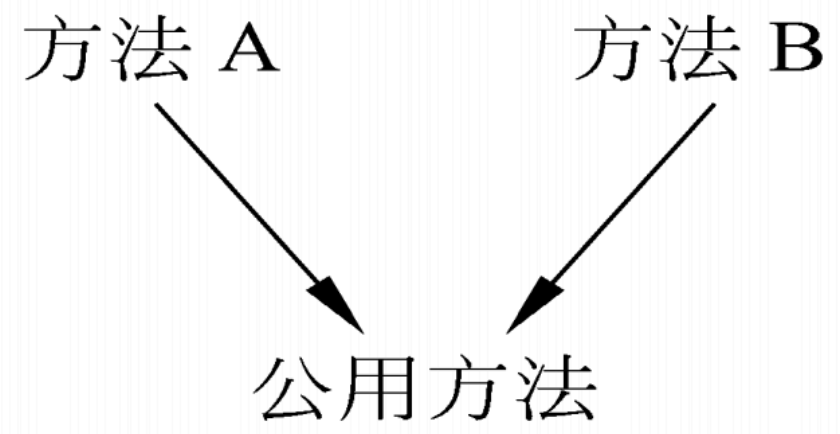
3、面向对象语言的优点

- 将来能够占主导地位
- 可重用性
- 类库和开发环境
- 其他因素

第三节 程序设计风格

1、程序设计准则

1. **提高重用性**
    - 提高方法的内聚
        - 方法只完成单个功能。涉及多个不相关功能，分解
    - 减小方法的规模
        - 方法规模过大，分解
    - 保持方法的一致性
        - 功能相似方法有一致名字、参数特征(包括参数个数、类型和次序)、返回值类型、使用条件及出错条件等
    - 把策略和实现分开
        - 负责做出决策，提供变元，管理全局资源，称策略方法
        - 负责完成具体操作，称实现方法
        - 编程时不要把策略和实现放在同一个方法中
    - 全面覆盖
        - 应针对所有组合写方法
    - 尽量不使用全局信息
        - 可以降低方法和外界的耦合程度
    - 利用继承机制
        - 实现共享和提高重用程度的主要途径
        - 调用子过程：把公共代码分离出来，构成一个公用方法



    - 分解因子：

        - 从不同类相似方法分解出不同的代码，余下作为公用方法中公共代码。把分解出的因子作为名字相同算法不同的方法，在不同类中定义
    - 使用委托
    - 把代码封装在类中：把被重用的代码封装在类中
2. **提高可扩充性**
    - 封装实现策略
        - 应把类的实现策略(包括数据结构、算法等)封装起来，对外提供公有接口
    - 不要用一个方法关联多条关联链
        - 一个方法应只包含对象模型中有限内容。否则导致方法过分复杂，不易理解和修改扩充
    - 避免使用多分支语句
        - 不要根据对象类型选择应有的行为，否则增添新类时不得不修改原有的代码，要合理的利用多态机制
          精心确定公有方法
    - 公有方法是向公众公布的接口
3. **提高健壮性**
    - 预防用户的操作错误
        - 任何输入(错误)，必须接受检查，给出提示信息，再次接收用户输入
    - 检查参数的合法性
    - 不要预先确定限制条件
    - 先测试后优化

第四节 测试策略

1、测试策略

单元测试——> 集成测试——> 验收测试

2、面向对象的单元测试

- 最小的可测试单元是对象与类
- 测试面向对象软件时，不能孤立地测试单个操作
- 传统的测试方法都可使用，等价类划分、边值分析、逻辑覆盖法、基本路径法

3、面向对象的集成测试

- 在面向对象的软件中不存在层次的控制结构，传统的自顶向下或自底向上的集成策略就没有意义了
- 此外，由于构成类的各个成分彼此间存在直接或间接的交互，一次集成一个操作到类中(传统的渐增式集成方法)通常是不现实的
- 面向对象软件的集成测试主要有下述两种不同的策略
    1. 基于线程的集成测试
        - 把响应系统的一个输入或一个事件所需类集成起来
    2. 基于使用的集成测试
        - 先测独立类，测完后测独立类下一层类(依赖类)，到测完

4、面向对象确认测试

- 不再考虑类之间相互连接的细节，集中检查用户可见的动作和用户可识别的输出

- 传统黑盒测试方法也可以使用，但是主要还是根据动态模型和描述系统行为的脚本设计确认测试用例

第五节 测试用例的设计

- 与传统软件测试(测试用例的设计由软件的输入-> 处理-> 输出视图或单个模块的算法细节驱动)不同，面向对象测试关注于设计适当的**操作序列**以检查**类的状态**

1、测试类的方法

1. **随机测试**
    - 在类的多个操作排列中，随机选择
    - 示例：
        - 在银行应用系统的account(账户)类操作：
            - open(打开)、setup、deposit(存款)、withdraw(取款)、balance(余额)、summarize(清单)、creditLimit(透支限额)和close(关闭)
        - 一个ccount类实例的最小行为历史包括下列操作：

          $$\text{open} \cdot \text{setup} \cdot \text{deposit} \cdot \text{withdraw} \cdot \text{close}$$

            - 这就是对account类的最小测试序列
        - 在下面的序列中可能发生许多其他行为：

          $$\text{open} \cdot \text{setup} \cdot \text{deposit} \cdot [\text{deposit} \mid \text{withdraw} \mid \text{balance} \mid \text{summarize} \mid \text{creditLimit}]^n \cdot \text{withdraw} \cdot \text{close}$$

        - 随机产生不同的操作序列

          测试用例#r1：open • setup • deposit • deposit • balance • summarize • withdraw • close

          测试用例#r2：open • setup • deposit • withdraw • deposit • balance • creditLimit • withdraw • close
    - 执行上述这些及另外一些随机产生的测试用例，可以测试类实例的不同生存历史
2. **划分测试**
    1. 基于状态的划分
        - 根据改变类状态能力划分：改变类状态、不改变类状态
        - 以account类为例：
            - 改变类状态：deposit、withdraw
            - 不改变类状态：balance、summarize、creditLimit
        - 设计测试用例，以分别测试改变状态的操作和不改变状态的操作
        - 测试用例(最小测试序列除外)

          测试用例#p1：open • setup • deposit • deposit • withdraw • withdraw • close

          测试用例#p2：open • setup • deposit • summarize • creditLimit • withdraw • close
    2. 基于属性的划分
        - 根据类操作属性：使用该属性、修改属性、不操作该属性
        - 示例：account类可根据balance属性把操作定义划分三个类别
            - 使用balance的操作
            - 修改balance的操作
            - 不使用也不修改balance的操作

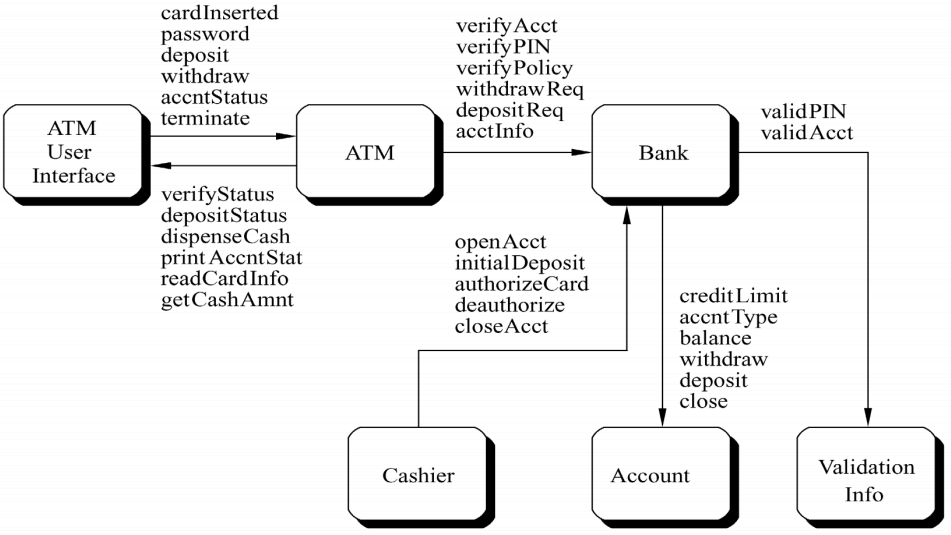
- 为每个类别设计测试序列

3. 基于功能的划分
   - 根据类操作完成的功能进行划分
   - 示例：acount类
     - 初始化操作(open、setup)
     - 计算操作(deposit、withdraw)
     - 查询操作(balance、summarize、creditLimit)
     - 终止操作(close)
   - 为每一个类别设计测试用例

3. **基于故障的测试**
   - 与传统的错误推断法类似，首先推断出软件可能有的错误，然后设计出最可能发现这些错误的测试用例
   - 很大程度上依赖程序员的直觉和经验，如果推断准确，效果较好；如果不准确，效果也会不好

2、集成测试的方法

1. 多类测试
   - 测类间协作，同样可采用随机测试和划分测试
   - **随机测试**
     - 生成多类随机测试的步骤
       - 对于每个客户类，使用类操作符列表来生成一系列随机测试用例，这些操作符向服务器类实例发送消息
       - 对所生成的每个消息，确定协作类在服务器对象中的对应操作符
       - 对服务器对象中的每个操作符，确定传递的消息
       - 对每个消息，确定下一层被调用的操作符，并把这些操作符结合进测试序列中
     - 示例：



     - Bank类对ATM类的操作序列

$$\text{verifyAcct} \cdot \text{verifyPIN} \cdot [(\text{verifyPolicy} \cdot \text{withdrawReq}) | \text{depositReq} | \text{acctInfoReq}]^n$$

     - 对Bank类的随机测试用例可能是

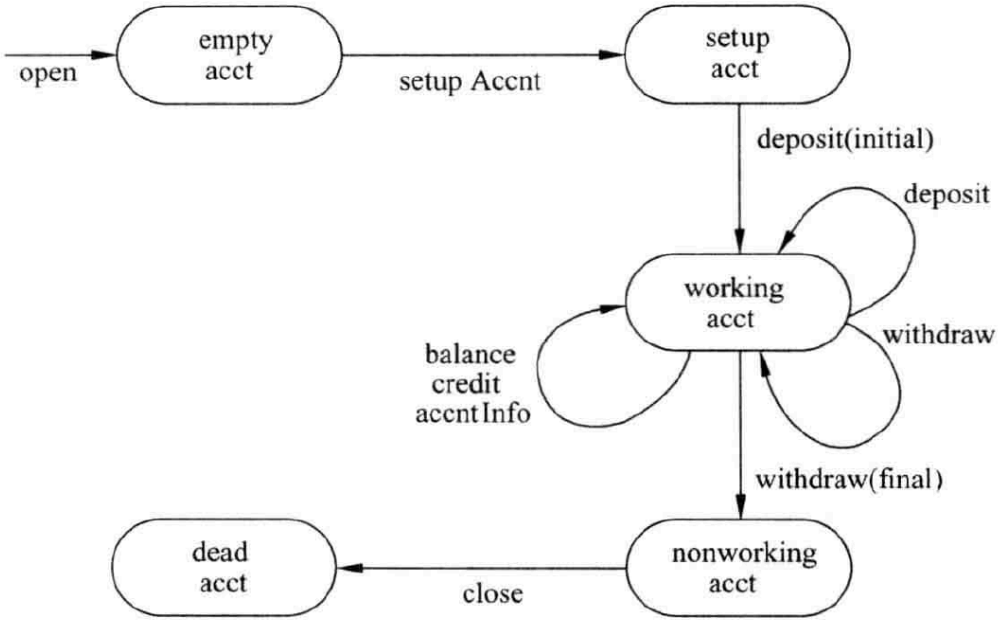
$$\text{测试用例} \# \text{r3}: \text{verifyAcct} \cdot \text{verifyPIN} \cdot \text{depositReq}$$

- 为了考虑测试涉及协作者，考虑与测试用例1每个操作相关联消息，Bank必须和ValidationInfo协作以执行verifyAcct和verifyPIN，Bank还必须和Account协作以执行deposit测试这些协作的新的测试用例

$$测试用例\#r4:verifyAcct_{Bank}\cdot[validAcct_{ValidationInfo}]\cdot verifyPIN_{Bank}\cdot[validPIN_{validationInfo}]\cdot depositReq\cdot[deposit_{account}]$$

- **划分测试**
  - 根据特定类的接口来划分，如bank类的方法分服务于ATM或服务于cashier

2. 从动态模型导出测试用例
   - 测试用例涵盖所有状态。如下图，操作系列使account类实例遍历所有允许的状态转换
   - Account类的状态转换图



   - 测试用例1：
     - `open · setupAccount · deposit(initial) · withdraw(final) · close`
   - 测试用例2：
     - `open · setupAccount · deposit(initial) · deposit · balance · credit · withdraw(final) · close`
   - 测试用例3：
     - `open · setupAccount · deposit(initial) · deposit · withdraw · accntInfo · withdraw(final) · close`
   - 导出更多的测试用例以保证该类的所有行为都被适当地测试
   - 在类的行为导致与一个或多个类协作的情况下，应该使用多个状态图去跟踪系统的行为流