面向对象的软件测试

费飞辉（fff）
中科大软件学院

面向对象的软件测试

- 1、面向对象测试的概述
- 2、开发前期的面向对象测试
  - 2.1 面向对象分析的测试
  - 2.2 面向对象设计的测试
  - 2.3 面向对象编程的测试
- 3、开发后期的面向对象测试
  - 3.1 面向对象的单元测试
  - 3.2 面向对象的集成测试
  - 3.3 面向对象的确认测试
  - 3.4 面向对象的系统测试

面向对象的软件测试

- 1、面向对象软件测试 概述
  - 面向对象软件测试的目标与传统测试一样

    即用尽可能低的测试成本和尽可能少的测试用例，发现尽可能多的软件缺陷。

    面向对象的测试策略也遵循从“小型测试”到“大型测试”，即从单元测试到最终的功能性测试和系统性测试。
  - 变化

    面向对象=对象+类+继承+通信(Coad和Yourdon定义)

    新特点：封装、继承、多态

面向对象的软件测试

- 这些变化带来了产生新错误的可能，带来了测试的变化：

1）基本功能模块

系统的基本构造单元不再是传统的功能模块，而是类和对象。在测试过程中，不能仅检查输入数据产生的输出结果是否与预期结果相吻合，还要考虑对象的状态变化，方法间的相互影响等。

2）系统的功能实现

系统的功能体现在对象间的协作上，而不再是简单的过程调用。原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。

3）封装对测试的影响

封装使对象的内部状态隐蔽，如果类中未提供足够的存取函数来表明对象的实现方式和内部状态，则类的信息隐蔽机制将给测试带来困难。

（续）

面向对象的软件测试

- 这些变化带来了产生新错误的可能性:

4）继承对测试的影响

继承削弱了封装性，产生了类似于非面向对象语言中全局数据的错误风险。由于继承的作用，一个函数可能被封装在具有继承关系的多个类中，子类中还可以对继承的特征进行覆盖或重定义。

若一个类得到了充分的测试，当其被子类继承后，继承的方法在子类的环境中的行为特征需要重新测试。

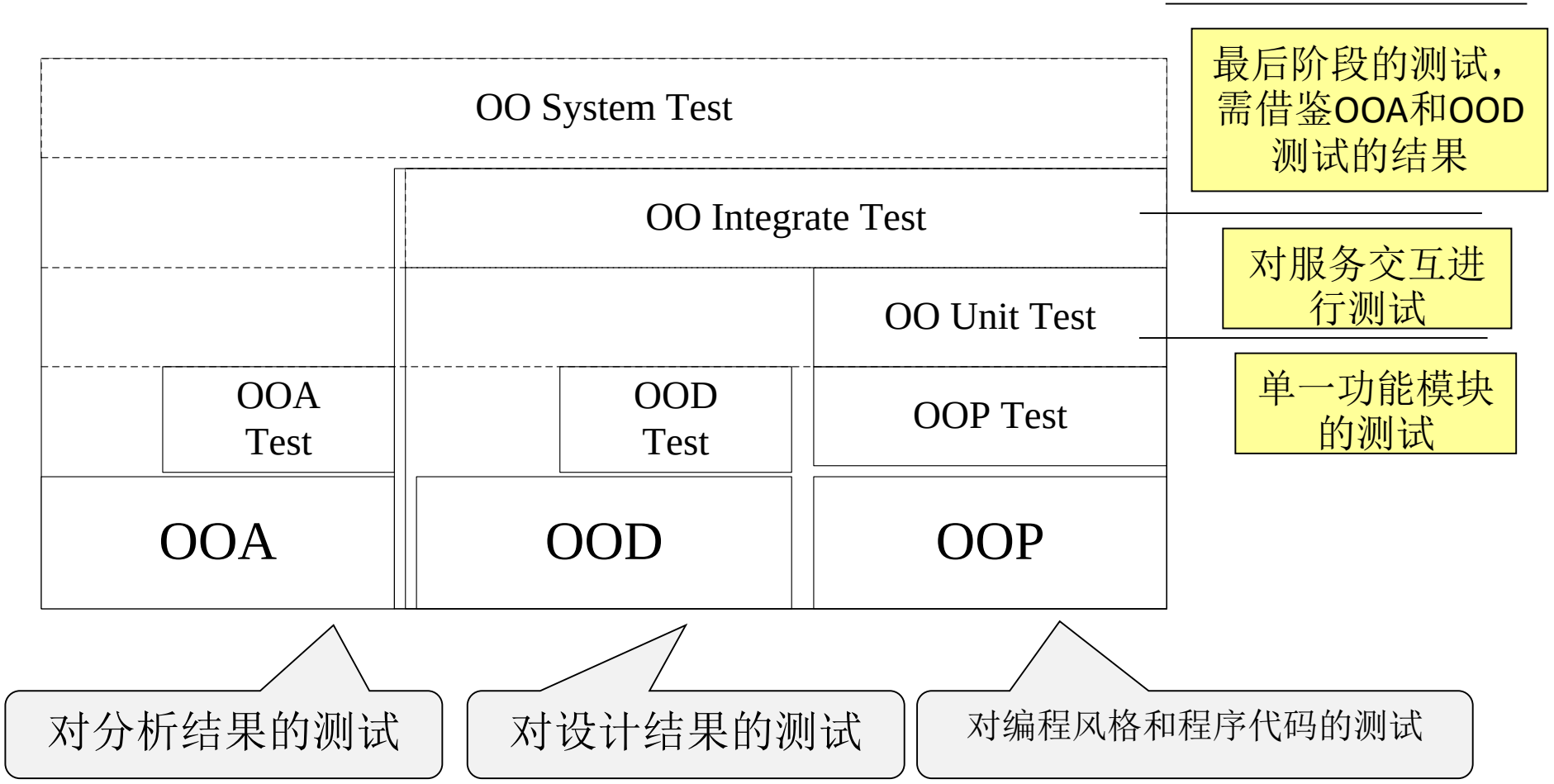
5）多态对测试的影响

多态依赖于不规则的类层次的动态绑定，可能产生非预期的结果。某些绑定能正确的工作但并不能保证所有的绑定都能正确地运行。以后绑定的对象可能很容易将消息发送给错误的类，执行错误的功能，还可能导致一些与消息序列和状态相关的错误。

面向对象的软件测试

- 面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析（OOA），面向对象设计（OOD），和面向对象编程（OOP）三个阶段
- 分析阶段产生整个问题领域的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。
- 针对这种开发模型，结合传统的测试步骤的划分，本着在整个开发过程中不断测试的原则，应将开发阶段的测试与编码完成后的单元测试、集成测试、系统测试用一个测试模型描述。

面向对象的软件测试


OO System Test
OO Integrate Test
OO Unit Test
OOA Test
OOD Test
OOP Test
OOA
OOD
OOP
最后阶段的测试，需借鉴OOA和OOD测试的结果
对服务交互进行测试
单一功能模块的测试
对分析结果的测试
对设计结果的测试
对编程风格和程序代码的测试

面向对象的软件测试

- ，由上图可见，开发阶段的面向对象的软件测试有：

  面向对象分析的测试(OOA Test)

  面向对象设计的测试(OOD Test)

  面向对象编程的测试(OOP Test)

  面向对象单元测试(OO Unit Test)

  面向对象集成测试(OO Integrate Test)

  面向对象系统测试(OO System Test)

- OOA Test和OOD Test：是对分析结果和设计结果的测试，主要是对分析设计产生的文本进行，是软件开发前期的关键性测试。

面向对象的软件测试

- OOP Test：主要针对编程风格和代码实现进行测试，其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。
- 面向对象单元测试：是对程序内部具体单一的功能模块的测试，主要就是对类和类成员函数的测试。
- 面向对象集成测试：主要对系统内部的相互服务进行测试，如成员函数间的相互作用，类间的消息传递等。

  面向对象集成测试不但要基于面向对象单元测试，更要参见OOD或OOD Test结果。
- 面向对象系统测试：主要以用户需求为测试标准，也要借鉴OOA或OOA Test结果。

面向对象的软件测试

2、开发前期的面向对象测试

面向对象的系统开发经历：面向对象分析（OOA）、面向对象设计（OOD）、面向对象编程（OOP）等三个阶段。

在这个时期的测试工作主要是静态测试。通过各种评审和质量分析活动，完成必须的测试工作，及时检测和克服各种缺陷。

面向对象的软件测试

2.1 面向对象分析的测试 （OOA Test）

- 面向过程分析

  传统的面向过程分析是一个功能分解的过程，是把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。

- 面向对象分析(OOA)

  是“把E—R图和语义网络模型，即信息模型中的概念，与面向对象程序设计语言中的重要概念结合在一起而形成的分析方法”，最后得到问题领域的可视的形式描述。

面向对象的软件测试

2.1面向对象分析的测试 （OOA Test）（续）

- OOA直接映射问题空间，全面的将问题空间中实现功能的现实抽象化，将问题空间中的实例抽象为对象，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和操作表示实例的特性和行为
- 对一个系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性
- OOA的结果是为后续阶段中类的选定和实现，类层层次结构的组织和实现提供平台。
- OOA测试的重点在其完整性和冗余性。

  OOA对问题领域分析抽象的不完整，最终会影响软件的功能实现，导致软件开发后期大量可避免的修补工作；而一些冗余的对象或结构会影响类的选定、程序的整体结构或增加程序员不必要的工作量。因此，OOA测试的重点在其完整性和冗余性。

面向对象的软件测试 - OOA Test

2.1面向对象分析的测试 （OOA Test）（续）

根据Coad和Yourdon方法所提出的OOA实现步骤，对OOA阶段的测试划分为以下五个方面：

1）对认定的类的测试

2）对认定的结构的测试

3）对认定的主题的测试

4）对定义的属性和实例关联的测试

5）对定义的服务和消息关联的测试

面向对象的软件测试 - OOA Test

1）对认定的类的测试

OOA中认定的类是对问题空间中的结构，其他系统，设备，被记忆的事件，系统涉及的人员等实际实例的抽象。对它的测试可以从如下方面考虑：

- 认定的类是否全面，是否问题领域中所有涉及到的对象都反映在认定的类中。
- 认定的类是否具有多个属性。只有一个属性的类通常应看成其他类的属性，而不是抽象为独立的类。
- 认定为同一个类的对象是否有共同的，区别于其他类对象的共同属性。
- 对认定为同一类的对象是否提供或需要相同的服务，如果服务随着不同的对象而变化，认定的对象就需要分解或利用继承性来分类表示。
- 如果系统不需要始终保持类所代表的对象的信息，认定的类也无必要存在。
- 认定的类的名称应该尽量准确，适用。

面向对象的软件测试 - OOA Test

2）对认定的结构的测试

- 认定的结构指的是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。
- 认定的结构分为两种：分类结构和组装结构。

分类结构，又称泛化结构，体现了问题空间中，对象的一般与特殊的关系

组装（复合）结构体现了问题空间中，对象的整体与局部的关系。

面向对象的软件测试 - OOA Test

- 对认定的分类结构（泛化结构）测试可从如下方面着手：
  - 对于结构中的一个类，尤其是处于高层的类，看是否能在问题领域中派生出其下一层的类。
  - 对于结构中的一个类，尤其是处于同一低层的类，看是否能抽象出在现实世界中有意义的更一般的上层的类。
  - 高层的对象的特性和服务是否完全体现下层的共性
  - 低层的对象是否基于其上层类的属性和服务并具有自己的特殊性。
- 对认定的组装结构的测试从如下方面入手：
  - 整体类和局部类的聚合（组装）关系是否符合现实的关系。
  - 整体类的局部类是否在问题空间中有实际应用。
  - 整体类中是否遗漏了在问题空间中有用的局部类。
  - 局部类是否能够在问题空间中组装新的有现实意义的整体类。

面向对象的软件测试 - OOA Test

3）对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象，是为了提供OOA分析结果的可见性，如同文章对各部分内容的概要。对主题层的测试应该考虑以下方面：

- 贯彻George Miller 的“7 ± 2”原则，如果主题个数超过7个，就要求对有较密切属性和服务的主题进行归并。
- 主题所反映的一组类和结构是否具有相同和相近的属性和服务。
- 认定的主题是否是类和结构更高层的抽象，是否便于理解OOA结果的概貌（尤其是对非技术人员的OOA 结果读者）。
- 主题间的消息连接（抽象）是否代表了主题所反映的类和结构之间的所有关联。

面向对象的软件测试 - OOA Test

4 ）对定义的属性和实例关联的测试

属性描述类或结构中实例（对象）的特性。而实例连接则反映实例集合之间的映射关系。对属性和实例关联的测试从如下方面考虑：

- 定义的属性是否对相应的类和泛化结构的每个实例都适用。
- 定义的属性在现实世界中是否与这种实例关系密切。
- 定义的属性在问题领域中是否与这种实例关系密切。
- 定义的属性是否能够不依赖于其他属性被独立理解。
- 定义的属性在泛化结构中的位置是否恰当，低层类的共有属性是否在其上层类的属性中有定义。
- 问题领域中每个类的属性是否定义完整。
- 定义的实例连接是否符合实际。
- 在问题领域中实例连接的定义是否完整，特别需要注意一对多和多对多的实例连接。

面向对象的软件测试 - OOA Test

5）对定义的服务和消息关联的测试

定义的服务，就是定义的每一种对象和结构在问题空间所要求的行为。由于问题域中实例间必要的通信，在OOA 中相应需要定义消息关联。

对定义的服务和消息关联的测试从如下方面进行：

- 类和结构在问题领域中的实例具有不同的状态，是否为状态转换定义了相应的服务。
- 类或结构所需要的服务是否都定义了相应的消息连接。
- 定义的消息连接所调用的服务是否正确。
- 沿着消息连接所执行的线索（消息的调用序列）是否合理，是否符合实际。
- 定义的服务是否有重复，是否定义了能够得到的服务。

面向对象的软件测试

2.2 面向对象设计的测试 （OOD Test）

- 面向对象设计（OOD）从“建模的观点”出发，基于OOA模型归纳出类，并建立类的层次结构或进一步构造成类库，实现分析结果对问题领域的抽象。
- OOD归纳出的类，可以是OOA类的简单延续，也可以是基于设计要求新建立或从已有类演化的类。因此，OOD是OOA的进一步细化和更高层的抽象。
- OOD确定类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便实现功能的重用和扩增，以不断适应用户的要求。

面向对象的软件测试

2.2 面向对象设计的测试 OOD Test（continue）

- 对OOD的测试，应从如下三方面考虑：
  - 对认定的类的测试
  - 对构造的类层次结构的测试
  - 对类库的支持的测试（复用性测试）
- 对认定的类的测试：
  - 认定的类原则上应该尽量是基础类
  - 是否涵盖了OOA中所有认定的对象
  - 是否能体现OOA中定义的属性
  - 是否能实现OOA中定义的服务
  - 是否对应着一个含义明确的数据抽象
  - 是否尽可能少的依赖其它类
  - 类中的方法是否单用途

面向对象的软件测试

2.2 面向对象设计的测试 OOD Test（continue）

- 对构造的类层次结构的测试：

为能充分发挥面向对象的继承共享特性，OOD的类层次结构，通常基于OOA中产生的泛化结构的原则来组织，着重体现父类和子类之间一般性和特殊性关系。

在当前的问题领域，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此应做如下几个方面的检查：

- 类层次结构是否涵盖了所有定义的类
- 是否能体现OOA中所定义的实例关联
- 是否能实现OOA中所定义的消息关联
- 子类是否具有父类没有的新特性
- 子类间的共同特性是否完全在父类中得以体现

面向对象的软件测试

2.2 面向对象设计的测试 OOD Test（continue）

- 对类库支持的测试：

  对类库的支持虽然也属于类层次结构的组织问题，但其强调的重点是软件的复用。

  由于它并不直接影响当前软件的开发和功能实现，可以将其单独提出来测试。

  有关类库支持的测试可从以下几个方面入手：

  - 一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口（包括名字和参数表）
  - 类中方法的功能是否较单纯，相应的代码行是否较少（建议不超过100行）
  - 类的层次结构是否是深度大，宽度小

面向对象的软件测试

2.3 面向对象编程的测试 OOP Test

- 面向对象程序的特点

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变

- 封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试
- 继承使得代码的重用率提高，同时也使错误传播的概率提高
- 多态使得程序内同一函数的行为复杂化，测试时必须考虑不同类型具体执行的代码和产生的行为

面向对象的软件测试

2.3 面向对象编程的测试 OOP Test（continue）

- 面向对象程序是把功能的实现分布在类中

  与某种设计功能相关的一组对象，通过对象提供的服务和对象之间的消息传递，共同协作来实现这个功能。这种面向对象程序风格，可将出现的错误精确定位在某一个具体的对象。

因此，在面向对象编程(OOP)阶段，忽略类功能实现的细则，将测试的目光集中在类功能的实现和相应的面向对象程序风格，主要体现为以下两个方面

- 数据成员是否满足数据封装的要求
- 类是否实现了要求的功能

面向对象的软件测试

1、数据成员是否满足数据封装的要求

就是检查其数据成员是否能被外界直接调用。更直观的说，当改变数据成员的结构时，看其是否影响了类的对外接口，是否会导致相应外界必须改动。

如：VS中 C++

Label1.Text=“hi”;

2、类是否实现了要求的功能

类的功能都是通过类的成员函数实现的。在分析类功能实现时，应分析类成员函数的正确性。

- 需注意的是，测试类的功能，应以OOD结果为依据，检测类提供的功能是否满足设计的要求，是否有缺陷。
- 必要时（如通过OOD结果仍不清楚明确的地方）还应该参照OOA的结果，并以其为最终标准。

面向对象的软件测试（续）

3、开发后期的面向对象测试

编程完成之后，需要经历若干个阶段的测试：单元测试、集成测试、验收测试、系统测试

3.1面向对象的单元测试

- 传统的单元测试对象：模块。多采用白盒测试技术。
- 当考虑面向对象软件时，单元的概念发生了变化
  - 最小的可测试单位可能是封装的类或对象，类包含一组不同的操作。

    一个对象有它自己的状态和依赖于状态的行为，对象操作既与对象的状态有关，也可能改变对象的状态。
  - 单元测试的意义发生了较大变化，可能是测试封装操作的类，也可能是测试类中的单个操作。

面向对象的软件测试

3.1 面向对象的单元测试（续）

传统的单元测试主要关注模块的算法；而面向对象软件的类测试主要是测试封装在类中的操作以及类的状态行为。

为此需要分两步走：

1) 测试与对象相关联的单个操作。 它们是一些函数或程序，传统的白盒测试和黑盒测试方法都可以使用。

单独地看类的成员函数，与过程性程序中的函数或过程没有本质的区别，几乎所有传统的单元测试中使用的方法，都可在面向对象的单元测试中使用。

2) 测试单个对象类。 黑盒测试的原理不变，但等价划分的概念要扩展以适合操作序列的情况。

面向对象的软件测试

1. 对象操作的测试

1）在面向对象程序中，对象的操作（成员函数）通常都很小，功能单一，函数间调用频繁，易出现一些不宜发现的错误。如：

- `if (write(fid, buffer, amount) == -1 ) error_out();`

  该语句没有全面检查 write() 的返回值，无意中假设了只有数据被完全写入和没有写入两种情况。此测试还忽略了数据部分写入的情况，就给程序遗留了隐患。

- 按程序的设计，使用函数 strrchr() 查找最后的匹配字符，但程序中误写成了函数 strchr()，使程序功能实现时查找的是第一个匹配字符。

- 程序中将 `if ( strncmp(str1, str2, strlen(str1)) )` 误写成了`if ( strncmp(str1, str2, strlen(str2)) )`。如果测试用例中使用的数据 str1 和 str2 长度相同，就无法检测出。

因此，在设计测试用例时，应对以函数返回值作为条件判断，字符串操作等情况特别注意。

面向对象的软件测试

1. 对象操作的测试（续）

2）面向对象编程的特性使得对成员函数的测试，又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性，Brian Marick 提出了两点：

（1）继承的成员函数可能需要重新测试

对父类中已测试过的成员函数，两种情况需要在子类中重新测试：

- 继承的成员函数在子类中做了改动；
- 成员函数调用了改动过的成员函数。

如：假设父类 Bass 有两个成员函数：Inherited()、Redefined()

若子类 Derived 对 Redefined()做了改动，

Derived::Redefined() 必需重新测试。如果 Derived::Inherited() 调用了 Redefined() （如：x = x / Redefined()），也需要重新测试；反之，则不必重新测试。

面向对象的软件测试

（2）对父类的测试用例不能照搬到子类

- 根据以上的假设，Base::Redefined() 和Derived::Redefined() 是不同的成员函数，它们有不同的说明和实现。对此，应该对 Derived::Redefined() 重新设计测试用例。
- 由于面向对象的继承性，使得两个函数还是有相似之处，故只需在 Base::Redefined() 的测试用例基础上添加对 Derived::Redfined() 的新测试用例。例如：

Base::Redefined() 含有如下语句

```
if ( value < 0 ) message ("less");
else if ( value == 0 ) message ("equal");
   else message ("more");
```

Derived::Redfined() 中定义为

面向对象的软件测试

```
        if ( value < 0 ) message ("less");
else if ( value == 0 ) message ( “It is equal");
   else { message ("more");
                        if ( value == 88 )
 message("luck");
                    }
```

在原有的测试上，对 Derived::Redfined() 的测试只需做如下改动：改动 value == 0 的预期测试结果，并增加 value == 88 的测试。

包含多态和重载多态在面向对象语言程序中通常体现在子类与父类的继承关系上，对这两种多态的测试可参照对父类成员函数继承和重载的情况处理。

面向对象的软件测试

2. 对象类测试

在测试对象时，完全的覆盖测试应当包括：

- 隔离对象中所有操作，进行独立测试。
- 测试对象中所有属性的设置和访问。
- 测试对象的所有可能的状态转换。所有可能引起状态改变的事件都要模拟到。

- 对象类，作为在语法上独立的构件，应当允许在不同应用中使用。每个类都应是可靠的且不需了解任何实现细节就能复用。因此对象类应尽可能孤立地进行测试。

面向对象的软件测试

- 设计操作的测试用例时的要点：
  - 首先定义测试对象各操作的测试用例。
  - 对于一个单独的操作，可通过该操作的前置条件选择测试用例，产生输出，让测试者能够判断后置条件是否能够得到满足。
  - 各个操作的测试与传统对函数过程定义的测试基本相同。
  - 然后再把测试用例组扩充，针对被测操作调用对象类中其他操作的情况，设计操作序列的测试用例组。
  - 测试可以覆盖每个操作的整个输入域。但这不够，还必须测试这些操作的相互作用，才能认为测试是充分的。
  - 各个操作间的相互作用包括类内通信和类间通信。

面向对象的软件测试

- 设计对象类功能测试时的要点：
  - 把对象类当做一个黑盒，确认类的实现是否遵照它的定义

    例如，对于“栈”的测试应当确保 LIFO 原则得以实施。
  - 对于多数对象类，主要检验在类声明的 public 域中的那些操作。
  - 对于子类，要检查继承父类的public 域和protected 域的那些操作。
  - 检查所有public域，protected域及private 域中的操作以完全检查对象中定义的操作。
  - 等价划分的思想也可用到对象类上。将使用对象相同属性的测试归入同一个等价划分集合中。这样可以建立对对象类属性进行初始化、访问、更新等的等价划分。

面向对象的软件测试

- 在设计对象类的行为测试时需要注意：
  - 基于对象的状态模型进行测试时，首先要识别需要测试的状态的变迁序列，并定义事件序列来强制执行这些变迁。
  - 原则上应当测试每一个状态变迁序列，当然这样做测试成本很高。
  - 完全的单元测试应当保证类的执行必须覆盖它的一个有代表性的状态集合。
  - 构造函数和消息序列（线程）的参数值的选择应当满足这个规则。

面向对象的软件测试

3.2、面向对象的集成测试

- 传统的集成测试，有两种方式通过集成完成的功能模块进行测：自顶向下集成、自底向上集成
- 当开发面向对象系统时，集成的层次并不明显。而当一组对象类通过组合行为提供一组服务时，则需将它们一起测试，这就是簇测试。此时不存在自底向上和自顶向下的集成。

面向对象的软件测试

3.2、面向对象的集成测试（续）

对象集成测试又称交互测试，目的是确保对象的消息传递能够正确进行。

- 面向对象系统的集成测试有 3 种可用的方法：

1）类层次测试

这种测试着眼于系统结构，首先测试几乎不使用服务器类的独立类，再测试那些使用了独立类的下一层次的（依赖）类。这样一层一层地持续下去，直到整个系统构造完成。

2）基于线程的测试

它把为响应某一系统输入或事件所需的一组对象类组装在一起。每一条线程将分别测试和组装。因为面向对象系统通常是事件驱动的，因此这是一个特别合适的测试形式。

面向对象的软件测试（续）

3.2、面向对象的集成测试（续）

3）对象交互测试

这个方法提出了集成测试的中间层概念。中间层给出叫做“方法-消息”路径的对象交互序列。所谓“原子系统功能”就是指一些输入事件加上一条“方法-消息”路径，终止于一个输出事件。

- 集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。
- 集成测试只关注于系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

面向对象的软件测试（续）

- 3.2、面向对象的集成测试（续）

  1） 静态测试

  静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求；通过源程序得到类关系图和函数功能调用关系图。

  将得到的结果与OOD的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测OOP是否达到了设计要求。

面向对象的软件测试（续）

- 3.2、面向对象的集成测试（续）

  2） 动态测试

  动态测试在设计测试用例时，通常需要上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。

  测试所要达到的覆盖标准可以是：

  - 达到类所有的服务要求或服务提供的一定覆盖率；
  - 依据类间传递的消息，达到对所有执行线程的一定覆盖率；
  - 达到类的所有状态的一定覆盖率等。
  - 考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。

面向对象的软件测试（续）

- 3.2、面向对象的集成测试（续）

  具体设计测试用例，可参考下列步骤：

  - 先选定检测的类；参考OOD分析结果，仔细列出类的状态和相应的行为、类或成员函数间传递的消息、输入或输出的界定等。
  - 确定覆盖标准。
  - 利用结构关系图确定待测试类的所有关联。
  - 根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

注意：设计测试用例时，要考虑不合理数据。

根据具体情况，动态的集成测试，有时也可以通过系统测试完成。

面向对象的软件测试（续）

- 面向对象软件集成测试的UML支持

在采用UML定义的面向对象软件中，协作图和序列图是集成测试的基础。协作图显示了类之间的信息传递情况。它既支持对方法为单元的测试，也支持相邻类的集成测试。

序列图跟踪通过协作图的执行时间路径（消息的前后相继关系）。序列图有两个层次的，一个是系统/用例级别的，一个是类交互级别的。它们是面向对象集成测试的很好的基础。

实试中，可根据协作图和序列图来建立测试用例，来确定测试覆盖的标准。

面向对象的软件测试（续）

3.3、面向对象的确认测试

与传统的确认测试一样，面向对象软件的有效性集中在用户可见的动作（事件驱动与过程）和用户可识别的系统输出（结果），通过测试检验软件是否满足用户的需求。

在面向对象的确认测试中，通常采用传统的黑盒测试方法，以证明软件功能和需求的一致性。

面向对象的软件测试（续）

3.4、面向对象的系统测试

系统测试应尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段

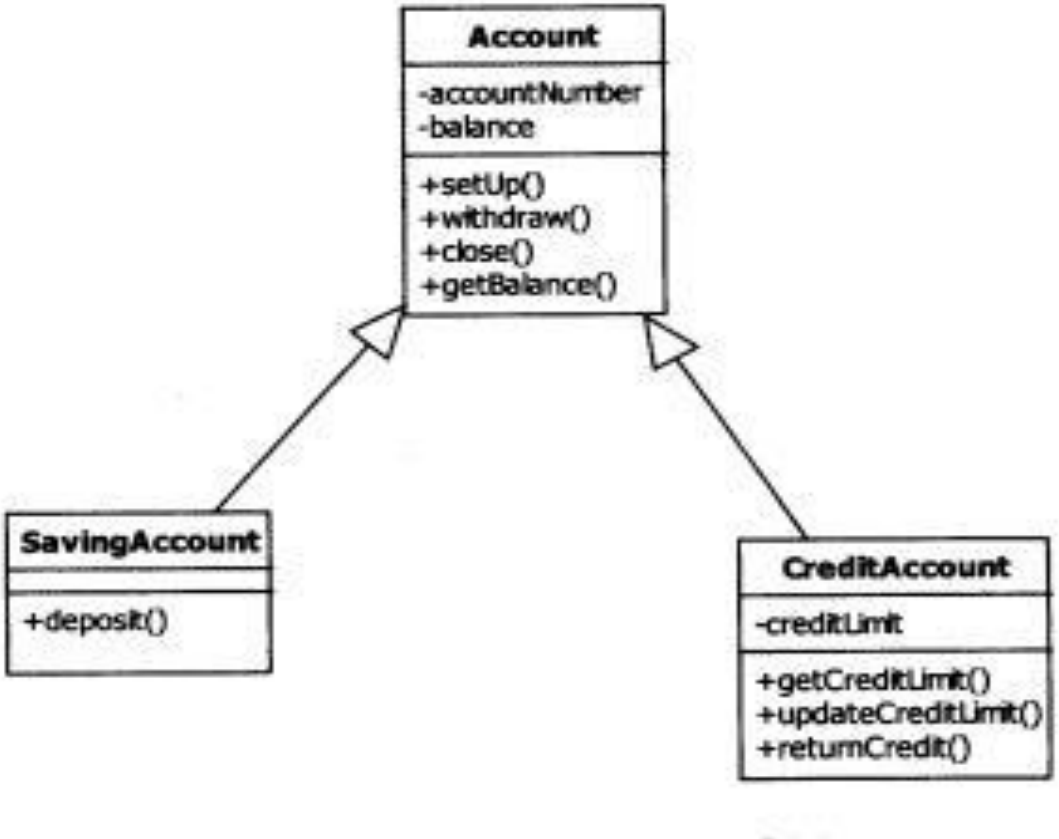
系统测试时，应该参考OOA分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全再现问题空间

系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认

它与传统的系统测试一样，可套用传统的系统测试方法

举例

下图是银行卡应用的部分类图，图中属性和操作前的“+”和“-”分别表示公有成员和私有成员。银行卡Account有两种类型，借记卡SavingAccount和信用卡CreditAccount。

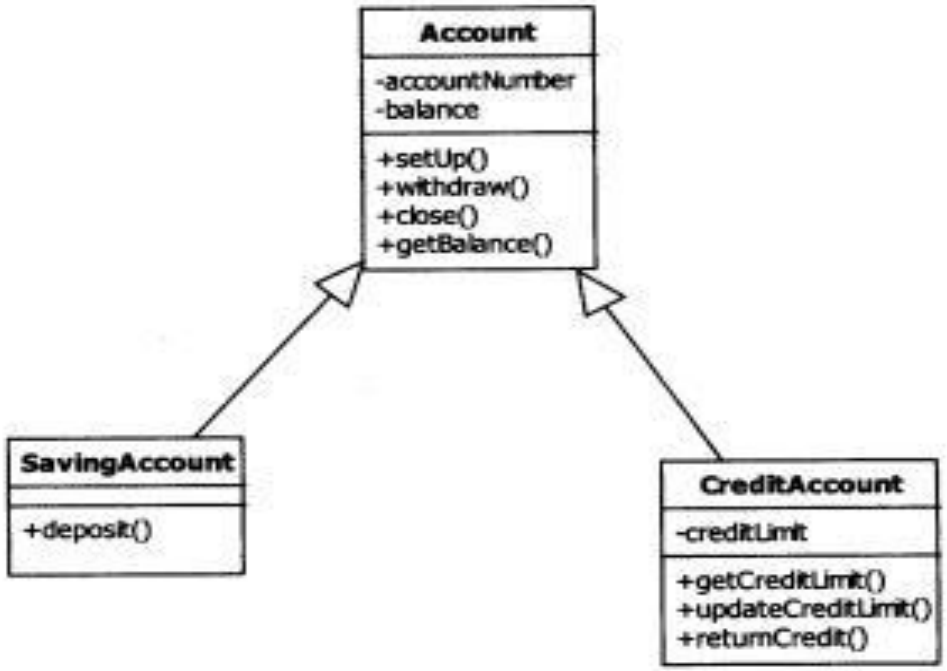


(1)借记卡和信用卡都有卡号accountNumber和余额balance两个属性。借记卡的余额是正常余额，信用卡的余额是目前未还的金额，如果没有未还的金额，则为0；有开户setUp、取款withdraw、查询余额getBalance和销户close四个方法。借记卡取钱时，要求取钱金额不能超过余额；而信用卡取钱金额不能超过信用额度，因此需要在子类中实现该方法。

(2)借记卡可以存钱deposit。

(3)信用卡有信用额度creditLimit属性，可以查询信用额度getCreditLimit、修改信用额度updateCreditLimit和还款retumCredit。

现拟采用面向对象的方法进行测试。

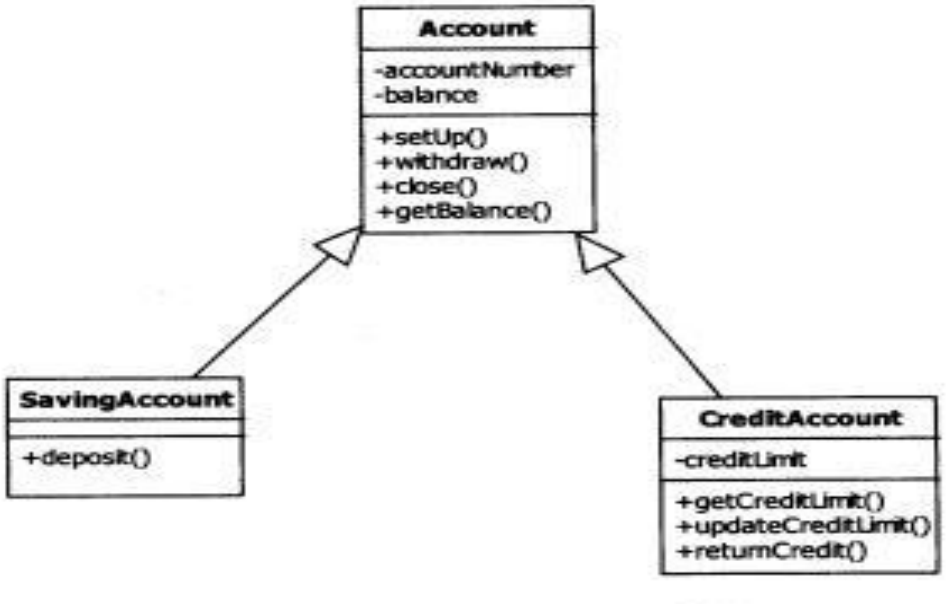


问：1）要测试方法deposit()时，还需要调用什么方法?给出测试序列。

在测试方法deposit()时，由于其属性balance发生了变化，因此在测试方法deposit()前后，调用方法getBalance()。

所以答案是：需要调用方法：getBalance()

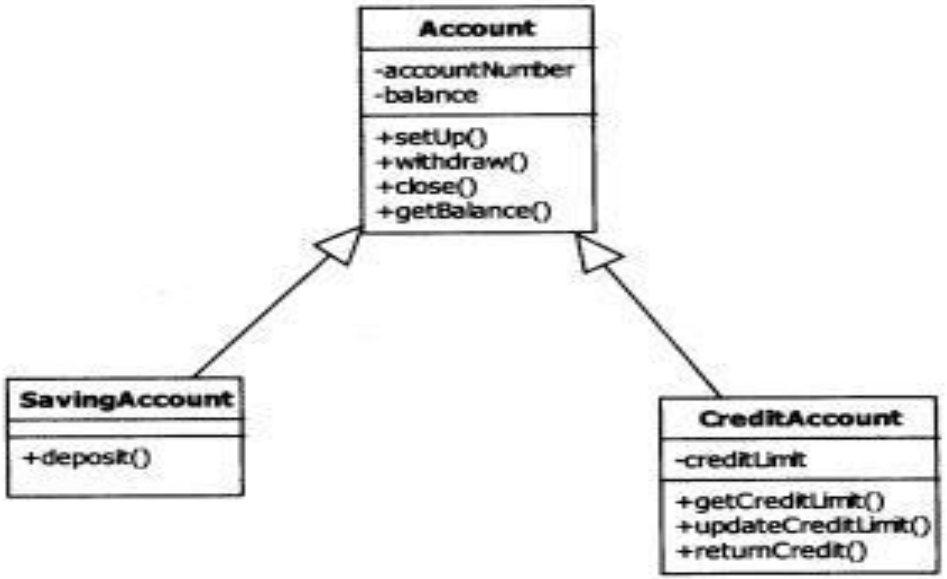
测试序列：getBalance()；depositq()；getBalance()；



2）方法withdraw在基类Account中定义，但在两个子类中有不同的实现。这是面向对象的什么机制?这种情况在测试时如何进行?

面向对象机制：多态

如何测试：在两个子类中均要测试withdraw方法



3)给出类SavingAccount的最小测试序列。

子类SavingAccount有五个方法，可以完成开户、存款、取款、查询余额和销户功能。因此，测试过程中应该包含这些功能。可以根据上述功能序列来设计测试序列，即setUp(); getBalance(); deposit(); getBalance(); withdraw(); getBalance(); close(); 。