软件质量保证与软件测试

--面向对象的测试

主讲教师:

Email:

主要内容

- 测试面向对象软件的问题
- 示例-ooNextDate
- 面向对象的单元测试
- 面向对象的集成测试
- 面向对象的系统测试

面向对象测试的单元

- 适合于面向对象测试的“单元”的定义：
  - 单元是可以编译和执行的最小软件组件。
  - 单元是绝不会指派给多个设计人员开发的软件组件
- 以类为单元有多种优点:
  - 在UML语境下，类和描述其行为的"状态图"关联起来,这对测试用例标识极为有用
  - 面向对象集成测试有更清晰的目标

合成与封装的涵义

- 合成：是面向对象软件开发的核心设计策略
- 封装：只有在类的内聚高，且耦合松时才能发挥作用
- 以下通过挡风玻璃雨刷系统例子说明这一点，三个类:控制杆、雨刷和刻度盘

合成与封装的案例

■ 这些类的接口伪代码是：

```
Class lever(leverPosition;
        private sense Lever Up (),
        private sense Lever Down())
Class dial( dial Positon;
        private sense Dial Up(),
        private sense Dial Down())
Class wiper(wiper Speed;
        Set Wiper Speed(new Speed))
```

■控制杆和刻度盘都是独立设备，并且在控制杆位于“间歇”位置时有交互

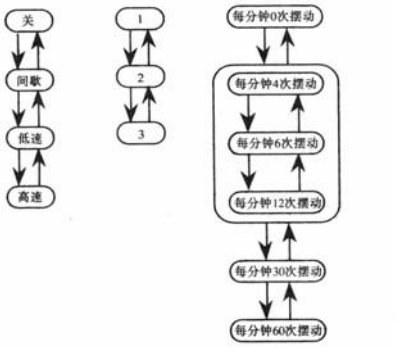


图16-1 挡风玻璃雨刷类的行为

这种交互应该在什么地方控制？

合成与封装的案例

■ 封装规则的要求是：类只了解自己的信息，并根据这些信息进行操作

■ 因此有三种方法解决上面的交互控制问题：

- ■控制杆和刻度盘永远报告各自的位置，由雨刷考虑要做什么
- ■使控制杆类成为“聪明”对象，因为它知道什么时候处于“间歇”位置
- ■建立雨刷主程序，但是使用控制杆和刻度盘类之间的"拥有"关系

■ 第一种方法在类之间的藕合非常少，可以最大限度地提高重用类的潜力；在另外两种方法中，类之间的耦合程度较强，会降低合成能力

很好的封装会得到能够更容易的合成(并因此而重用)和测试的类

继承的涵义

■将类作为单元看起来很自然，但是继承角色使这种选择变得很复杂

■ “扁平类”作为解决这种问题的一种方法

■扁平类有些类似结构化分析中被彻底扁平化了的数据流图

继承的案例

■左图显示的是前面提到过的简单自动柜员机(SATM)系统部分的UML继承图，并增加了一些功能，以使这个例子更完善

■有图给出了已经“扁平化”了的checking Account和savings Account类

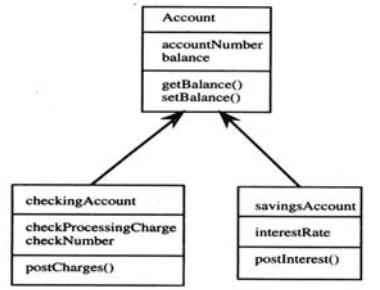


图16-2 类继承

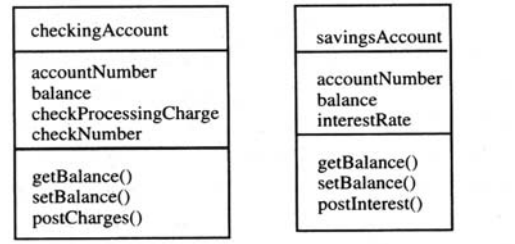
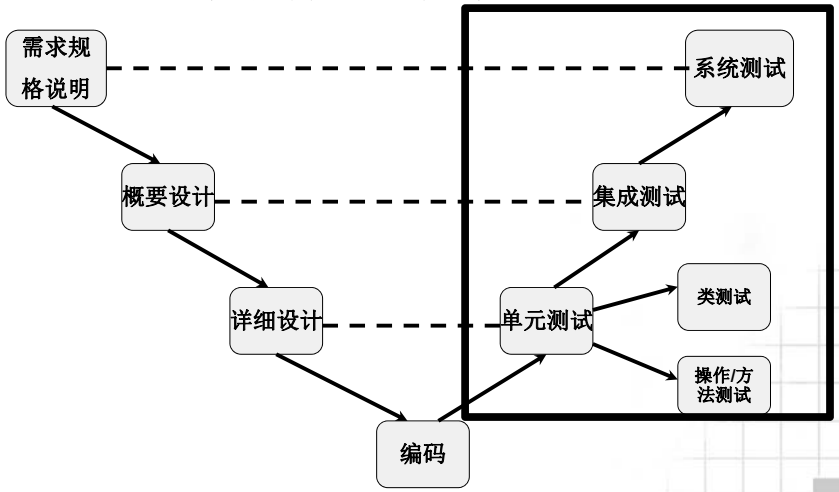


图16-3 被扁平化了的checkingAccount和savingsAccount类

多态性的涵义

- 多态性的本质是同样的方法应用于不同的对象。把类看作单元，意味着多态性的所有问题都要被类/单元测试覆盖

面向对象软件测试层次



面向对象软件的数据流测试

- 数据流测试的基础是标识单元程序图中的定义和使用节点，然后再考虑各种定义/使用路径
- 传统软件中的过程调用使这种方法复杂化，常见的做法是将被调用的过程嵌入到要测试的单元中
- 后面讨论一种事件驱动Petri网的改进版本，能够确切地描述面向对象操作之间的数据流

主要内容

- 测试面向对象软件的问题
- 示例-ooNextDate
- 面向对象的单元测试
- 面向对象的集成测试
- 面向对象的系统测试

NextDate问题的面向对象的实现

■**ooNextDate**程序是第2章中NextDate问题例子的一种面向对象实现

■下图给出了**ooNextDate**问题的UML类图和类的继承及聚合

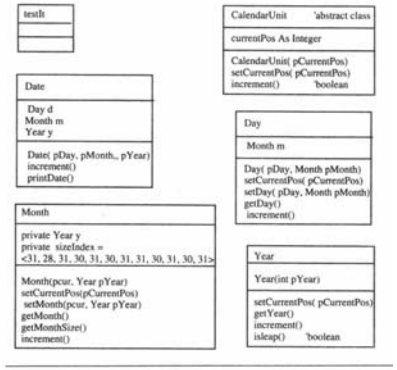
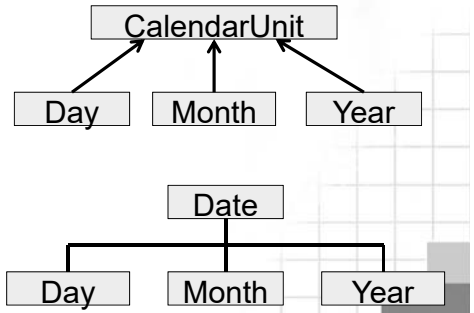


图16-4 o-oCalendar中的类



类继承和聚合

NextDate问题的面向对象实现

■ **Class: CalendarUnit**

责任：提供一个操作在所继承的类中设置值，提供一个布尔操作说明所继承类中的属性是否可以增1

```
class CalendarUnit          'abstract class
            currentPos As Integer
            CalendarUnit (pCurrentPos)
                        current Pos = pCurrentPos
            End   'Calendar Unit
a.          set CurrentPos (pCurentPos)
b.              current Pos = pCurrentPos
            End    'set Current Pos
            abstract protected boolean increment ( )
```


NextDate问题的面向对象实现

■类：testIt

责任：用做测试驱动器，即创建一个测试日期对象，然后请求该对象对其本身增1，最后打印新的日期值

```
class test It
            main()
1.          testdate = instantiate Date(test Month，  test Day，  test Year)     msg1
2.          testdate. increment ( )                                               msg2
3.          testdate. PrintDate ( )                                               msg3
            End     'test It
```


NextDate问题的面向对象实现

■类：Date

责任：Date对象由日、月和年对象组成

```
class Date
            private Day d
            private Month m
            private Year y
4           Date (pMonth，  pDay，  pYear)
5               y = instantiate Year (pYear)                        msg4
6               m = instantiate Month (pMonth，y)                   msg5
7               d = instantiate Day (pDay，  m)                     msg6
            End   'Date constructor
8           increment ( )
9               if  ( NOT ( d. increment ( ))                               msg7
10                      Then
11                              if  ( NOT ( m. increment ( ) )      msg8
12                              Then
13                                      y. increment ( )            msg9
14                                      m. setMonth (1，y)          msg10
15                              Else
16                                      d. setday (1，m)            msg11
17                              End If
18          End If
            End         'increment
19          Print Date ( )
20              Output (d. getDay ( ) + "/" +                       msg12
                        m. getMonth ( ) + "/" +                     msg13
                        y. getYear ( ))                             msg14
            End          'print Date
```

NextDate问题的面向对象实现

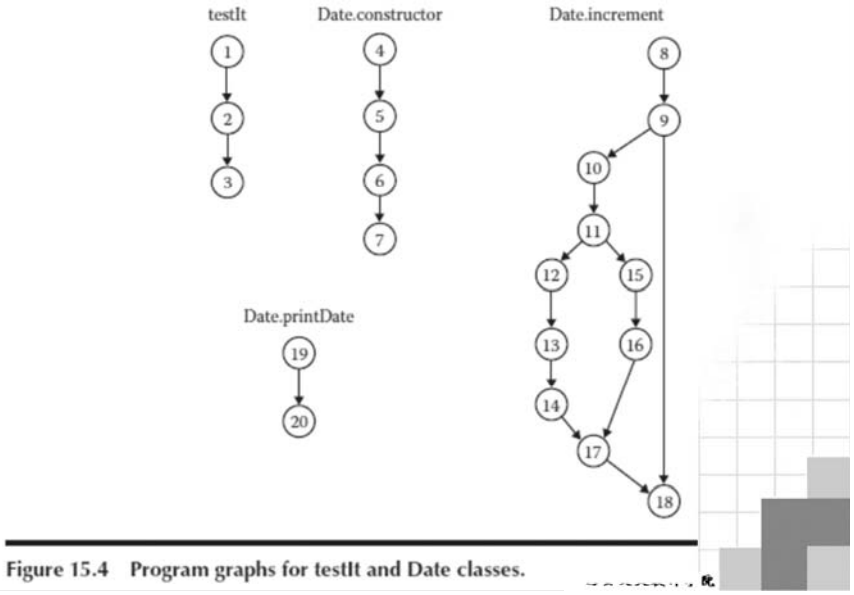


Figure 15.4 Program graphs for testIt and Date classes.

NextDate问题的面向对象实现

■类：Day

责任： Day对象有一个私有月属性，增量方法用来查看日取值是要增1还是复位为1。Day对象也提供get()和set()方法。

```
class Day is A Calendar Unit
          private Month m
21        Day(pDay， Month pMonth)
22                  setDay (pDay, pMonth)        msg15
          End       'Day constructor
23        setDay (pDay, Month pMonth)
24          setCurrentPos (pDay)                 msg16
25           m = pMonth
          End       'setDay
26        getDay ( )
27              return  currentPos
          End       'getDay
28        bolean increment( )
29              currentPos = currentPos + 1
30              if (current Pos <= m.getMonthSize( ))    msg17
31                    Then return true
32                    Else return false
33        EndIf
          End       'increment
```

NextDate问题的面向对象实现

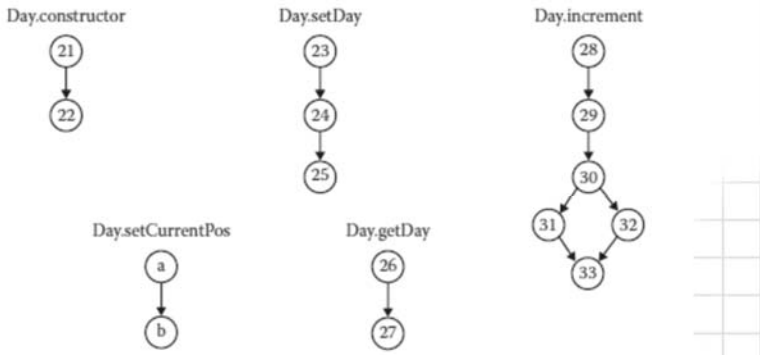


Figure 15.5 Program graphs for Day class.

NextDate问题的面向对象实现

■类：Month

责任：Month对象有一个值属性，用做月最后一日值的数组下标(例如1月的最后一日是31，2月的最后一日是28，等等)。Month对象提供get()和set()服务，以及所继承的布尔增量方法。2月29日的判决，由给Year对象的isLeap消息做出。

```
class Month isA Calendar Unit
          private Year y
          Private sizeIndex =<31,28,31,30,31,30,31,31,30,31,30,31>
34        Month (pcur， Year pYear)
35          setMonth(pCurrentPos， Year pyear)     msg18
          End    'Month constructor
36        Set Month( pcur, Year pYear)
37            setCurrentPos(pcur)                  msg19
38             y = pYear
          End       'setMonth
39        getMonth()
40            return currentPos
          End       'getMonth
41        getMonthSize ()
42                  if (y.isleap())                msg20
43                        Then sizeIndex [1] = 29
44                        Else sizeIndex [1] = 28
45                  EndIf
46                  Return sizeIndex[ currentPos - 1]
          End       'getMontSize
47        Increment ( )
48              currentPos = currentPos  + 1
49              if (currentPos > 12)
50                    Then return false
51                    Else return true
52              EndIf
          End       'increment
```

NextDate问题的面向对象实现

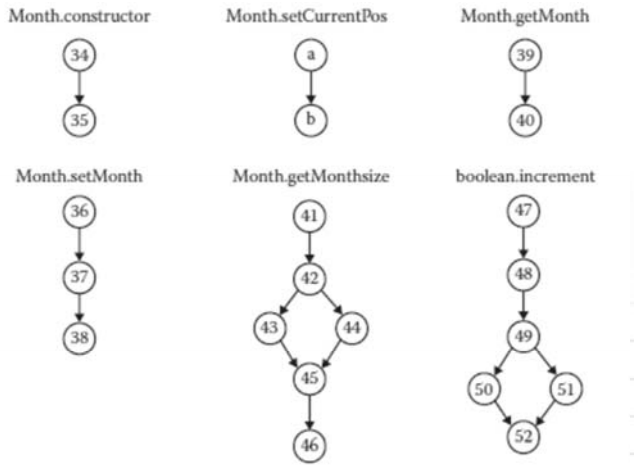


Figure 15.6 Program graphs for Month class.

NextDate问题的面向对象实现

■类：Year

```
class Year isA Calendar Unit
53          Year (pYear)
54                setCurrentPos (pYear)                                    msg21
            End         'Year constructor
55          getYear()
56              return currentPos
            End         'get Year
57          boolean increment()
58                currentPos = currentPos + 1
59          return true
            End         'increment
60          boolean isleap()
61                  if (((currentPos MOD 4 = 0) AND NOT(currentPos MOD 100 = 0)) OR
                          (currentPos MOD 400 = 0))
62                        Then return true
63                        Else return false
64                  EndIf
            End                 'isleap
```


NextDate问题的面向对象实现

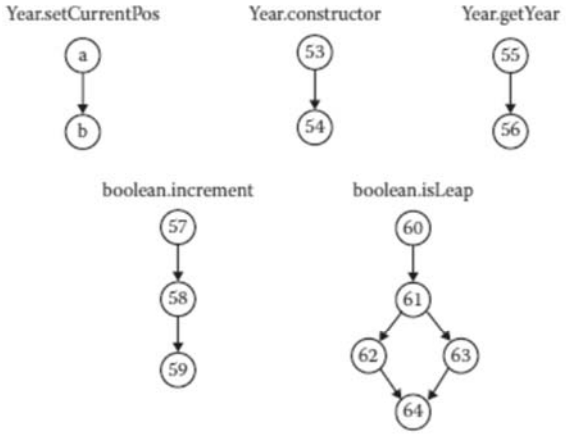


Figure 15.7 Program graphs for Year class.

主要内容

- ■测试面向对象软件的问题
- ■示例-ooNextDate
- ■面向对象的单元测试
- ■面向对象的集成测试
- ■面向对象的系统测试

以方法作为单元

- 这种方法可以将面向对象单元测试归结为传统的(过程的)单元测试
- 方法几乎等价于过程，所以可以使用所有传统功能性测试和结构性测试技术
- 必须提供能够实例化的桩类，以及起驱动器作用的"主程序"类，以提供和分析测试用例

以方法为单元

- 如果更仔细地研究单个方法，就会发现令人高兴的封装结果：方法一般很简单
- 前面给出ooNextDate应用程序类的伪代码和对应的程序图
  - 圈复杂度总是很低，Date类的Increment方法圈复杂度最高，也只有V(G) =3
  - 即使圈复杂度很低，但是接口复杂度仍然很高
  - 大部分负担被转移到集成测试中

以方法为单元

- Date.increment操作处理日的三个等价类：
  - D1 = {日 | 1 ≤ 日 < 月的最后一天}
  - D2 = {日 | 日是非12月的最后一天}
  - D3 = {日 | 日是12月31日}

以类为单元

- 把类作为单元可以解决类内集成问题，但是会产生其他问题
  - 一个问题是与类的各种视图有关
    - 第一种视图是静态视图：类作为源代码存在，继承被忽略，但是通过被充分扁平化了的类可以解决这个问题
    - 第二种视图是编译时视图：继承实际发生在编译时
    - 第三种视图是执行时视图：测试实际上在该视图下发生，但不能测试抽象类，因为不能被实例化
  - 把类作为单元，在没有什么继承，并且类具有我们称之为内部控制复杂性时最有意义

windshieldWiper类的伪代码

■前面讨论过的三个类，在这里被合并为一个类

```
class windshieldWiper
        private wiper Speed
        private lever Position
        private dial Position
        wind shield Wiper (wiperSpeed， leverPosition， dialPosition)
        get WiperSpeed( )
        set WiperSpeed( )
        get LeverPosition( )
        set Lever Position( )
        get DialPosition( )
        set DialPosition( )
        senseLeverUp( )
        senseLeverDown( )
        SenseDialUp( )
        SenseDialDown( )
End class windshieldWiper
```


windshieldWiper类的状态图

■类行为如图中的"状态图"所示

■其中三个设备出现在正交组件中。在Dial和Lever 组件中，转移由事件引起，因此雨刷组件中的转移，都是由与Dial和Lever正交组件中"活动"状态有关的命题引起的

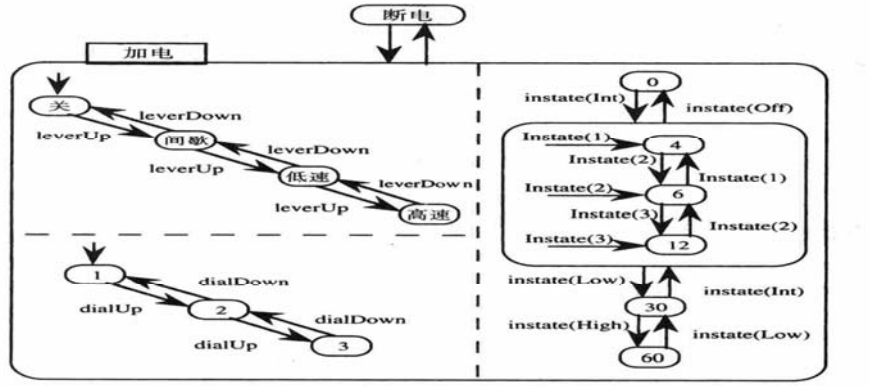


图17-5 windshieldWiper类的 "状态图"

windshieldWiper类的单元测试

■以类为单元选择的困难部分，是因为有单元测试的层次

```
senseLeverUp()
        Case leverPosition Of
                Case 1: Off
                        leverPositon = Int
                        Case dialPosition Of
                        Case 1:1
                                wiperSpeed = 4
                        Case 1:2
                                wiperSpeed = 6
                        Case 1:3
                                wiperSpeed = 12
                        EndCase  'dialPosition
                Case 2: Int
                        leverPosition = Low
                        wiperSpeed = 30
                Case 3: Low
                        leverPosition = High
                        wiperSpeed = 60
                Case 4: High
                        (impossible; error condition)
                EndCase  'leverPositon
```


windshieldWiper类的单元测试

■测试senseLeverUp方法需要Case和嵌套Case语句中的不同分支，测试驱动器类的伪代码与以下给出的代码类似:

```
class testSenseLeverUp
        wiperSpeed
        leverPos
        dialPos
        testResult 'boolean
main( )
        testCase = instantiate windshieldWiper(0, Off, 1)
        windshieldWiper.sensLeverUp( )
        leverPos = windshieldWiper.getLverPosition()
        If leverPos = Int
                Then testResult = Pass
                Else testResult = Fail
        End If
End  'main
```

■ 此外还有两个测试用例，用来测试从"间歇"到"低速"和从"低速"到"高速"的转移。

windshieldWiper类的单元测试

■采用以下伪代码测试windshieldWiper类的其他部分

```
class testWindshieldWiper
        wiperSpeed
        leverPos
        dialPos
        testResult 'boolean
main( )
        testCase = instantiate windshieldWiper(0, Off, 1)
        windshieldWiper.senseLverUp()
        wiperSpeed = windshieldWiper.getWiperSpeed()
        If wiperSpeed = 4
                Then testResult = Pass
                Else testResult = Fail
        End If
End   'main
```


windshieldWiper类的单元测试

■ 较高层次的覆盖实际上适用于方法的类内集成，这看起来与以类为单元的思想矛盾。场景覆盖准则与系统级测试几乎相同。以下是一个测试用例以及在一个测试类中所需的相应消息序列

UC1	正常用法
描述	用户将控制杆推到"间歇"
	用户将刻度盘从位置1转到位置2
	用户将刻度盘从位置2转到位置3
	用户将控制杆推到"低速"
	用户将控制杆推到"间歇"
	用户将控制杆推到"关"
前提	挡风玻璃雨刷在"关"位置
	"刻度盘"在位置1
	雨刷速度为0

windshieldWiper类的单元测试

事件序列	用户行动	系统应答
1	将控制杆推到"间歇"	雨刷速度为4
2	将刻度盘转到2	雨刷速度为6
3	将刻度盘转到3	雨刷速度为12
4	将控制杆推到"低速"	雨刷速度为20
5	将控制杆推到"间歇"	雨刷速度为12
6	将控制杆推到"关"	雨刷速度为0

windshieldWiper类的单元测试

■"控制杆"组件中每个状态覆盖层次的实例化语句(用于建立前提)和预期输出的测试用例

测试用例	前提(实例化语句)	windsheldWiper事件(方法)	预期leverPos的输出值
1	WindshieldWiper(0,Off,1)	senseLeverUp ( )	"间歇"
2	WindshieldWiper(0,Int,1)	senseLeverUp ( )	"低速"
3	WindshieldWiper(0,Low,1)	senseLeverUp ( )	"高速"
4	WindshieldWiper(0,High,1)	senseLeverDown ( )	"低速"
5	WindshieldWiper(0,Low,1)	senseLeverDown ( )	"间歇"
6	WindshieldWiper(0,Int,1)	senseLeverDown ( )	"关"

windshieldWiper类的单元测试

```
class test Scenario
   wiperSpeed
   leverPos
   dialpos
   step1 OK  'boolean
   step2 OK  'boolean
   step3 OK  'boolean
   step4 OK  'boolean
   step5 OK  'boolean
   step6 OK  'boolean
main()
   testCase = instantiate windshieldWiper(0,off,1)
   windshieldWiper.senseLeverUp()
   WiperSpeed = windshieldWiper.getWiperSpeed()
   If wiperSpeed = 4
         Then stepl OK = Pass
         Else stepl OK = Fail
   End If
   windshieldWiper.senseDialUp()
   wiperSpeed = windshieldWiper.getWiperSpeed()
   If wiperSpeed = 6
         Then stp2 OK = Pass
         Else step2 OK = Fail
   End If
```

```
windshieldWiper.senseDialUp()
wiperSpeed=windshieldWiper.getWiperSpeed()
If wiperSpeed = 12
         Then step3 OK = Pass
         Else step3 OK = Fail
End If
windshieldWiper. senseLeverUp()
wiperSpeed=windshieldWiper.getWiperSpeed()
If wiperSpeed = 20
         Then step4 OK = Pass
         Else step4 OK = Fail
End If
windshieldWiper. senseLeverDown()
wiperSpeed=windshieldWiper.getWiperSpeed()
If wiperSpeed = 12
         Then step5 OK = Pass
         Else step5 OK = Fail
End If
windshieldWiper. senseLeverDown()
wiperSpeed=windshieldWiper.getWiperSpeed()
If wiperSpeed = 0
         Then step6 OK = Pass
         Else step6 OK = Fail
End If
End      'main
```

主要内容

- ■测试面向对象软件的问题
- ■示例-ooNextDate
- ■面向对象的单元测试
- ■面向对象的集成测试
- ■面向对象的系统测试

面向对象的集成测试

■对于集成测试，两种单元选择方法都需要进行集成测试：

- ■ 如果采用操作/方法作为单元，则需要进行两级集成：一级是将操作集成到完整类中，另一级是将类与其他类集成
- ■ 以类为单元的方法，一旦完成单元测试，必须执行两个步骤：

(1)如果使用了被扁平化了的类，则必须恢复最初的类层次结构

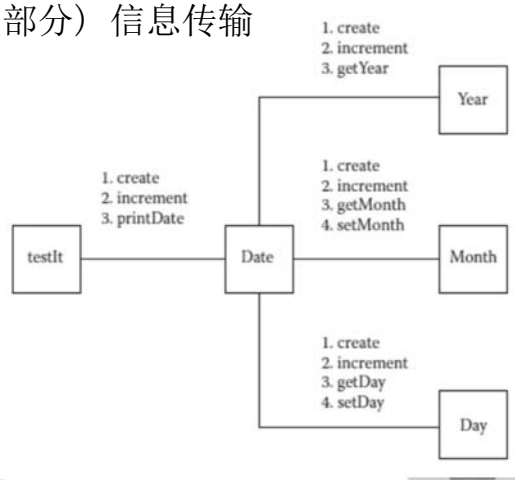
(2)如果增加了测试方法，则必须删除

集成测试的UML支持

■在采用UML定义的面向对象软件中，协作图和时序图是集成测试的基础

■协作图显示类之间的(部分) 信息传输

- ■成对集成测试
- ■相邻集成测试

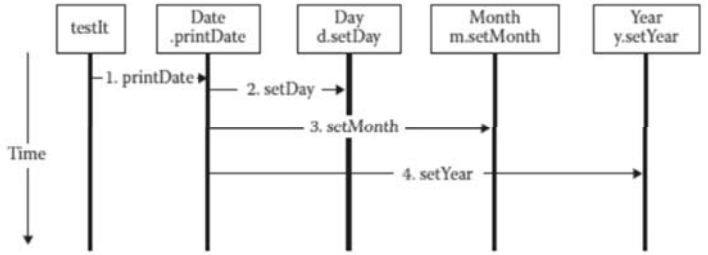


集成测试的UML支持

- 根据上图所示的协作图，可以得到以下要集成的类对:
  - testIt和Date，为Year、Month和Day建立桩
  - Date和Year，为testIt、Month和Day建立桩
  - Date和Month，为testIt、Year和Day建立桩
  - Date和Day，为testIt、Month和Year建立桩
  - Year和Month，为Date和Day建立桩
  - Month和Day，为Date和Year建立桩

集成测试的UML支持

- 顺序图跟踪通过协作图的执行时路径
- ooNextDate应用程序的部分序列图如下所示：



集成测试的UML支持

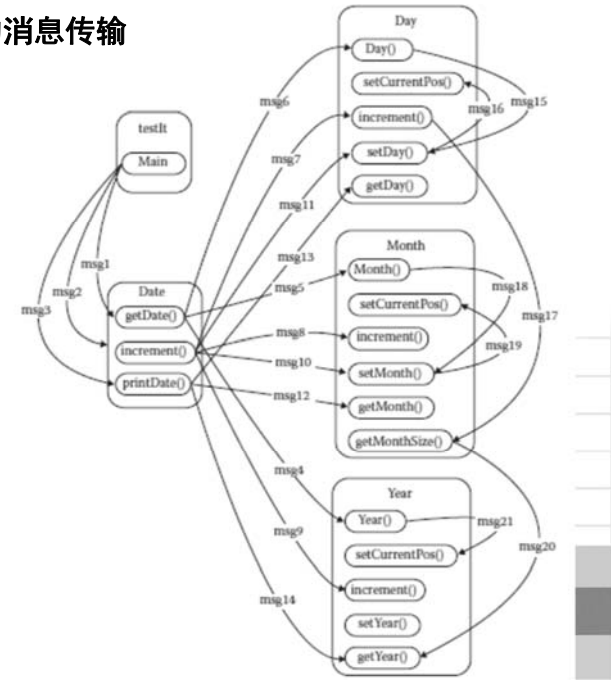
- 在一定程度上，创建时序图是面向对象集成测试的很好基础
- 这些时序图几乎等价于面向对象版的MM-路径。
- 这种时序图的实际测试应该采用与以下类似的伪代码：

```
1.  test Driver
2.      m.setMonth(1)
3.      d.setDay( 15)
4.      y.setYear(2002)
5.      Output ("expected value is 1/15/2002")
6.      Output ("actual output is ")
7.      testIt.printDate()
8.  End test Driver
```


面向对象软件的MM-路径

- **定义1** 面向对象软件中的MM-路径是由消息连接起来的方法执行序列
- **定义2** 原子系统功能(ASF)是一种MM-路径，从输入端口事件开始，到输出端口事件结束

ooNextDate中的消息传输



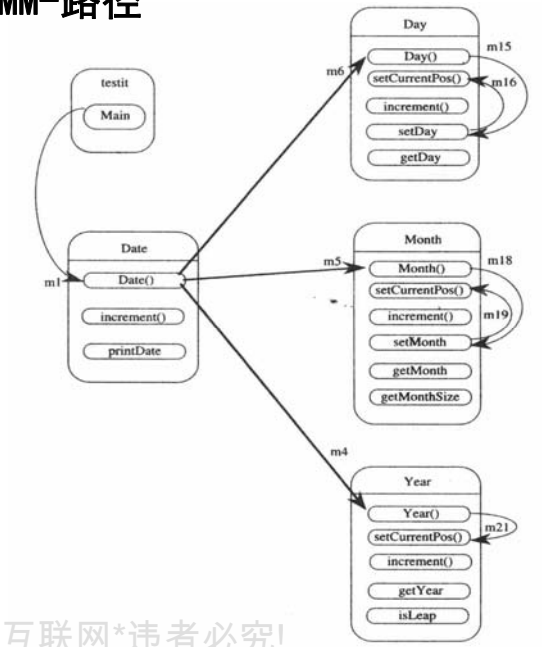
面向对象软件的MM-路径

■以下是采用2002年1月15日实例化Date的部分MM-路径。与第1章一样，这里也直接使用伪代码中的语句和消息编号

```
Test It <1>                                   msg1
Date: test date <4, 5>                        msg4
Year: y <53, 54>                              msg21
Year: y.set Current Pos <a, b>
        (return to Year.y)
        (return to Date:testdate)
Date: test date <6>                           msg5
Month: m <34, 35>                             msg18
Month: m.set Month <36, 37>                   msg19
Month: m.setCurrentPos <a, b>
        (return to Month:m.setMonth)
        (return to Month:m)
        (return to Date:testdate)
Date: test date <7>                           msg6
Day: d <21, 22>                               msg15
Day: d.set Day <23, 24>                       msg16
Day: d.set Current Pos <a, b>
        (return to Day:d.setDay)
Day: d.set Day <25>
        (return to Day:d)
        (return to Date:testdate)
```

面向对象软件的MM-路径

■上面代码的MM-路径如图所示



面向对象软件的MM-路径

■ 以下是用2002年4月30日Date.increment的更有意思的MM-路径（请参阅下图）

```
testIt <2>                                                  msg2
Date: testdate.increment <8，9>                             msg7
Day: d.increment <28, 29>                                   msg17
Month: m.getMonthSize <41, 42>                              msg20
Year: y.sleap <60, 61, 63, 64>     'not a leap year
        (return to Month:m.getMonthSize)
Month: m.getMonthSize <44, 45, 46>
        (return to Day:d.increment)
Day: d.increment <32, 33>          'returns false
(return to Date:testdate.increment)
Date: testdate.increment <10, 11>                           msg8
Month: m.increment <47, 48, 49, 51, 52>     'returns true
        (return to Date:testdate.increment)
Date: testdate.increment <15，16>                           msg11
Day: d.setDay <23, 24, 25>       'now dayis 1, month is 5
        (return to Date:testdate.increment)
Date: testdate.increment <17, 18>
        (return to testIt)
```

面向对象软件的MM-路径

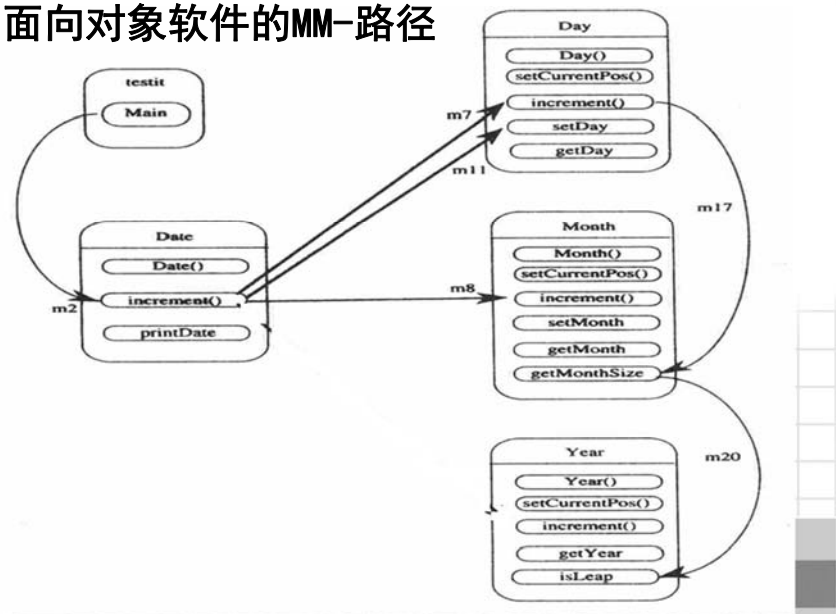


图18-5 Date.increment（2002年4月30日）的MM-路径

面向对象数据流集成测试框架

- 面向对象软件的集成测试优点：
  - 取值可以通过继承树得到；
  - 数据可以在消息传递的不同阶段定义
- 程序图构成过程代码的描述和分析数据流测试的基础
- 面向对象软件的复杂性超出有向(程序)图的表达能力

事件驱动和消息驱动的Petri网

定义： EMDPN是一种由四部分组成的有向图(P、D、M、S、In、Out)，包括四组节点：P、D、 M和S，以及两个映射：In和Out。其中：

- P是端口事件集合
- D是数据库所集合
- M是消息库所集合
- S是转移集合
- In是(P∪D∪M)×S的有序对偶集合
- Out是S×(P∪D∪M)的有序对偶集合

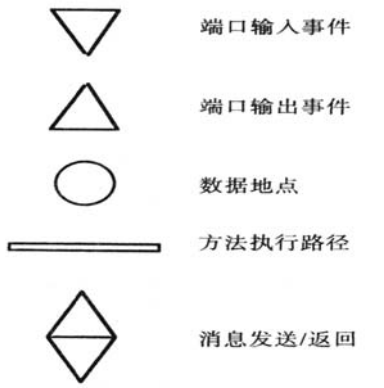


图18-6 EMDPN的符号

事件驱动和消息驱动的Petri网

- 新符号用来获取对象间消息的本质：
  - 这些消息是发送消息对象中的方法执行路径一个输出
  - 这些消息是接收消息对象中的方法执行路径一个输入
  - 返回是接收消息对象中的方法执行路径一个非常微妙的输出
  - 返回是发送消息对象中的方法执行路径一个输入

事件驱动和消息驱动的Petri网

■图给出的是新消息库所可以在EMDPN中出现的惟一方式

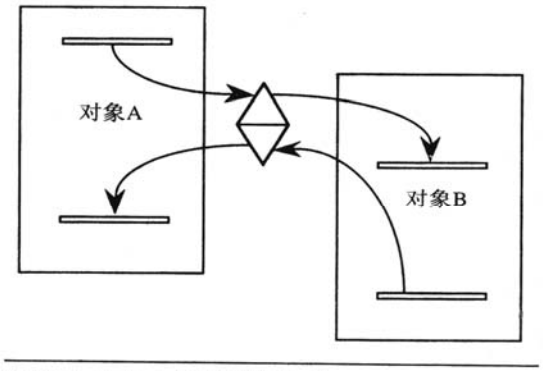


图18-7 对象之间的消息连接

由继承导出的数据流

■这种框架支持多种形式的继承，包括单一继承、多重继承和有选择性的继承

■描述继承的EMDPN仅由数据库所和方法执行路径组成，如图所示

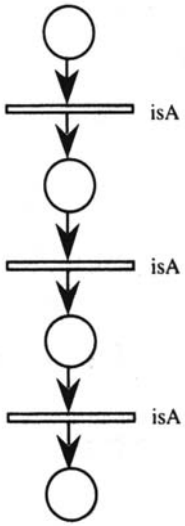


图18-8 有继承的数据流

由消息导出的数据流

■假设mep3是要由mep5转发，在mep6中修改，并最终在使用节点mep2中使用的数据项的“定义”节点。我们可以标识这两条定义-使用路径:

■du1 = <mep3, msg2, mep5, d6, mep6, return(msg2), mep4 , return(msg 1), mep2>

■du2 = <mep6, return(msg2), mep4, return(msg 1), mep2>

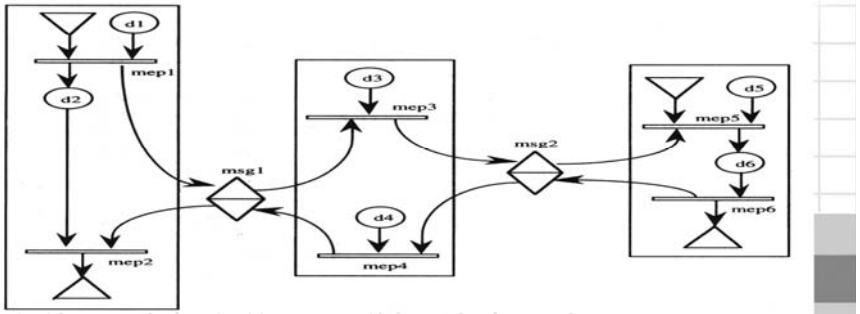


图18-9 通过消息通信的数据流

主要内容

■测试面向对象软件的问题

■示例-ooNextDate

■面向对象的单元测试

■面向对象的集成测试

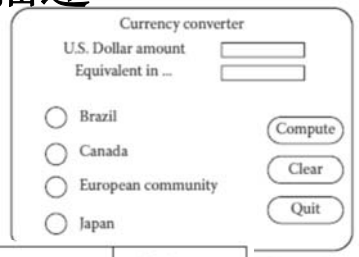
■面向对象的系统测试

面向对象的系统测试

- 系统测试独立于系统实现，测试人员不必知道系统是如何实现的
- 在第14章，系统测试的基础是端口输入和输出事件，采用EDPN来表示系统级线索
- 问题在于如何找到用作测试用例的线索
  - 第14章使用需求规格说明模型、尤其是行为模型
- 对于面向对象软件，借鉴第14章的方法
  - 不同在于，假定系统已经用UML进行了建模
  - 从UML模型中得到系统级线索测试用例

货币转换程序的UML描述

- 应用程序将美元转换为任意四种货币：巴西雷亚尔、加拿大元、欧元和日元



Reference No.	*Function*	*Category*
R1	Start application	Evident
R2	End application	Evident
R3	Input US dollar amount	Evident
R4	Select country	Evident
R5	Perform conversion calculation	Evident
R6	Clear user inputs and program outputs	Evident
R7	Maintain exclusive–or relationship among countries	Hidden
R8	Display country flag images	Frill

货币转换程序的UML描述

- 高层用例

HLUC 1	Start application.
Description	The user starts the currency conversion application in Windows®.

HLUC 2	End application.
Description	The user ends the currency conversion application in Windows.

HLUC 3	Convert dollars.
Description	The user inputs a US dollar amount and selects a country; the application computes and displays the equivalent in the currency of the selected country.

HLUC 4	Revise inputs.
Description	The user resets inputs to begin a new transaction.

HLUC 5	Repeated conversions, same dollar amount.
Description	The user inputs a US dollar amount and selects a country; the application computes and displays the equivalent in the currency of the selected country.

货币转换程序的UML描述

- 高层用例

HLUC 6	Revise inputs.
Description	A US dollar amount has been entered OR a country has been selected.

HLUC 7	Abnormal case: no country selected.
Description	User enters a dollar amount and clicks on the Compute button without selecting a country.

HLUC 8	Abnormal case: no dollar amount entered.
Description	User selects a country and clicks on the Compute button without entering a dollar amount.

HLUC 9	Abnormal case: no dollar amount entered and no country selected.
Description	User clicks on the Compute button without entering a dollar amount and without selecting a country.

货币转换程序的UML描述

- 基础用例
- 在高层用例的基础上增加“actor”和“system”事件；actor是系统级输入的来源

EUC-1	Start application
Description	The user starts the currency conversion application in Windows.
Event Sequence	
Input Events	Output Events
1. The user starts the application, either with a Run ... command or by double clicking the application icon.	
	2. The currency conversion application GUI appears on the monitor and is ready for user input.

货币转换程序的UML描述

- 基础用例

EUC-3	Convert dollars
Description	The user inputs a US dollar amount and selects a country; the application computes and displays the equivalent in the currency of the selected country.
Event Sequence	
Input Events	Output Events
1. The user enters a dollar amount.	
	2. The dollar amount is displayed on the GUI.
3. The user selects a country.	
	4. The name of the country's currency is displayed.
	5. The flag of the country is displayed.
6. The user requests a conversion calculation.	
	7. The equivalent currency amount is displayed.

货币转换程序的UML描述

- 基础用例

EUC-4	Revise inputs
Description	The user resets inputs to begin a new transaction.
Event Sequence	
Input Events	Output Events
1. The user enters a dollar amount.	
	2. The dollar amount is displayed on the GUI.
3. The user selects a country.	
	4. The name of the country's currency is displayed.
	5. The flag of the country is displayed.
6. The user cancels the inputs.	
	7. The name of the country's currency is removed.
	8. The flag of the country is no longer visible.

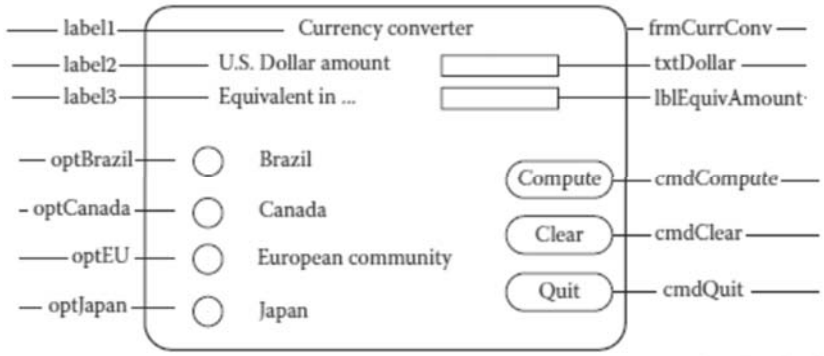
货币转换程序的UML描述

- 基础用例

EUC-7	Abnormal case: no country selected
Description	The user enters a dollar amount and clicks on the cmdCompute button without selecting a country.
Event Sequence	
Input Events	Output Events
1. The user enters a dollar amount.	2. The dollar amount is displayed on the GUI.
3. User clicks the Compute button.	4. A message box appears with the caption "must select a country."
5. The user closes the message box.	6. The message box is no longer visible.
	7. The flag of the country is no longer visible.

货币转换程序的UML描述

- 详细GUI定义



货币转换程序的UML描述

- 扩展用例
- 在基础用例的基础上，增加前置条件、后置条件、备选事件流、交叉引用等
- 更多的用例被找到

EEUC-1	Start application
Description	The user starts the currency conversion application in Windows.
Preconditions	Currency conversion application in (disk) storage
Event Sequence	
Input Events	Output Events
1. User double-clicks currency conversion application icon	2. frmCurrConv appears on screen
Postconditions	1. Currency conversion application is in memory 2. txtDollar has focus

货币转换程序的UML描述

- 扩展用例

EEUC-3	Normal usage (dollar amount entered first)
Description	The user inputs a US dollar amount and selects a country; the application computes and displays the equivalent in the currency of the selected country.
Preconditions	txtDollar has focus
Event Sequence	
Input Events	Output Events
1. User enters US dollar amount on keyboard	2. Dollar amount appears in txtDollar
3. User clicks on a country button	4. Country currency name appears in lblEquiv
5. User clicks cmdCompute button	6. Computed equivalent amount appears in lblEqAmount
Postconditions	cmdClear has focus

货币转换程序的UML描述

- 扩展用例

EEUC-4	Repeated conversions, same country
Description	The user inputs a US dollar amount and selects a country; the application computes and displays the equivalent in the currency of the selected country.
Preconditions	txtDollar has focus
Event Sequence	
Input Events	Output Events
1. User enters US dollar amount on keyboard	2. Dollar amount appears in txtDollar
3. User clicks on a country button	4. Country currency name appears in lblEquiv
5. User clicks cmdCompute button	6. Computed equivalent amount appears in lblEqAmount
7. User clicks on txtDollar	8. txtDollar has focus
9. User enters different US dollar amount on keyboard	10. Dollar amount appears in txtDollar
11. User clicks cmdCompute button	12. Computed equivalent amount appears in lblEqAmount
Postconditions	cmdClear has focus

货币转换程序的UML描述

- 实际用例
- 在扩展用例的基础上，用更精确的描述语句，如：
  “enter a US dollar amount”必须替换为“enter 125 in txtDollar”；“select a country”应该替换为“click on the optBrazil button”
- 根据实际用例，可以机械地得到系统级的测试用例

货币转换程序的UML描述

- 基于UML的系统测试
- 对于GUI应用的覆盖度量，至少可以从4个层次进行：
- 1.测试第一步得到的系统功能（来自于用户的原始功能）
- 由于扩展用例交叉引用这些功能，所以可以构建一个关联矩阵
- 然后设计测试用例来覆盖这些功能即可

货币转换程序的UML描述

- 1.测试第一步得到的系统功能（来自于用户的原始功能）

Table 15.4 Incidence Matrix of Use Cases with System Functions

EEUC	R1	R2	R3	R4	R5	R6	R7
1	×	–	–	–	–	–	–
2	–	×	–	–	–	–	–
3	–	–	×	×	×	–	–
4	–	–	×	×	×	–	–
5	–	–	×	×	×	–	×
6	–	–	×	×	–	×	×
7	–	–	×	–	×	–	–
8	–	–	×	–	×	–	–
9	–	–	–	–	×	–	–

货币转换程序的UML描述

- 2.根据实际用例，得到测试用例；可以机械式得到

System test case 3	Normal usage (dollar amount entered first)
Test performed by	Paul Jorgensen
Preconditions	txtDollar has focus
Event Sequence	
Input Events	Output Events
1. Enters 10 on the keyboard	
	2. Observe 10 appears in txtDollar
3. Click on optEU button	
	4. Observe “euros” appears in label3
5. Clicks cmdCompute button	
	6. Observe “7.60” appears in lblEquivAmount
Postconditions	cmdClear has focus
Test result	Pass (on first attempt)
Date run	May 27, 2013

货币转换程序的UML描述

- 3.根据对GUI应用外部特性的有限状态机模型，得到测试用例

- 4.从基于状态的事件表得到测试用例

总结

- 面向对象软件的特征是什么？
- 面向对象软件的测试层次是什么？
- 以方法为单元的类测试的思想是什么？
- 以类为单元的类测试的思想是什么？
- 面向对象的集成测试有哪些方法？
- 如何进行面向对象的系统测试？