实验 1：程序员视角的软件调试与软件测试

董瑞志
常熟理工学院软件工程系

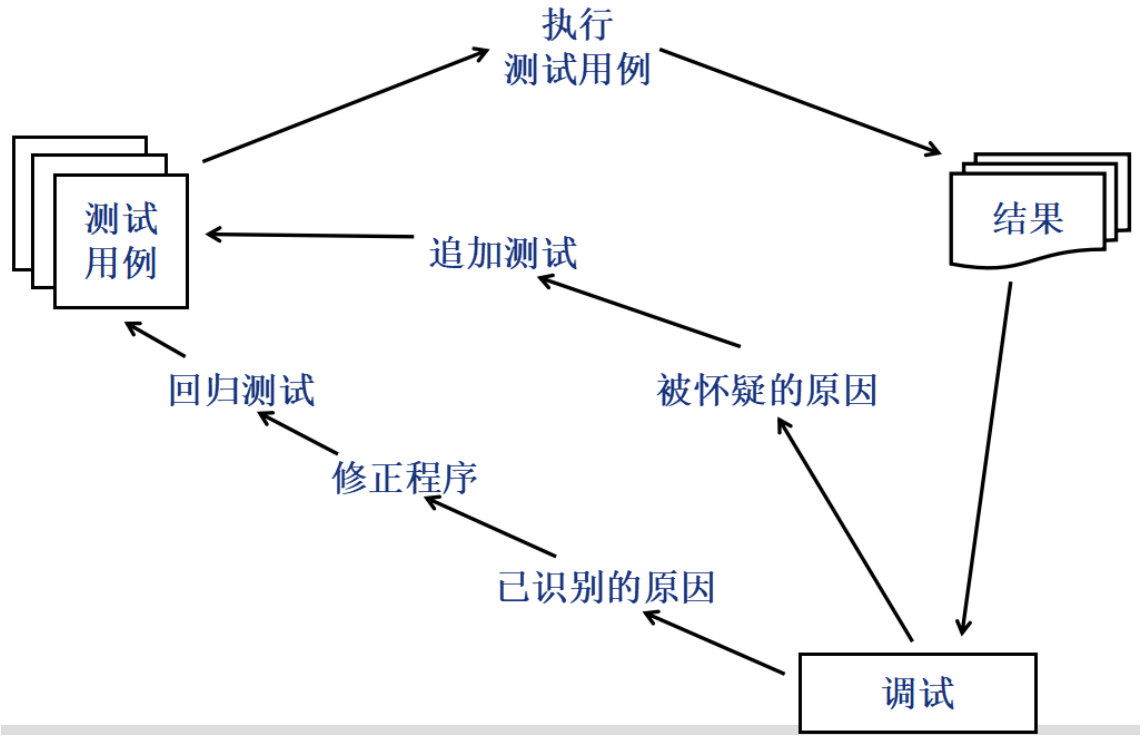
实验目的：

了解软件调试原理；
掌握 Eclipse 调试器的使用方法及技巧。

课时安排：

2 课时

知识准备



Introduction

In this section, we will demonstrate the debugging process using a debugger. We will use the debugger included in Eclipse. The basic functions provided by this debugger are similar to those provided by other debuggers. Therefore, you can also walk through this tutorial with another debugger.

The Example

The class Employee represents an employee in a payroll system. The objects of class Employee contain the name and salary of the employee. The salary of an employee is a function of the hourly wage, the number of hours worked, and the sales commissions. The information of each employee is stored in a string with the following format:

name_hourlyWage_hoursWorked_sale1,sale2,...,salen

where,

- name is the name of the employee
- wagePerHour is the employee's hourly wage
- hoursWorked is the number of hours the employee worked
- sale1,sale2,...,salen are the employee's sales

The earning of an employee is calculated using the following equation.

earning = hourlyWage * hoursWorked + (sale1 + sale2 + ... + salen) * 0.02

Where, the value 0.02 corresponds to a 2% sales commission.

Following is the code of class Employee: Employee.java

The constructor uses a StringTokenizer object to extract each part of the employee's data. It then assigns values to the variables name and earning. The method computeCommission returns the employee's commission of the sales data. This implementation of class Employee is incorrect. In the rest of the page, we will use the Eclipse debugger to find the error and fix it.

Debuging with Eclipse

Create the Project

The first step is to create the Java project and then name the project as Employee.

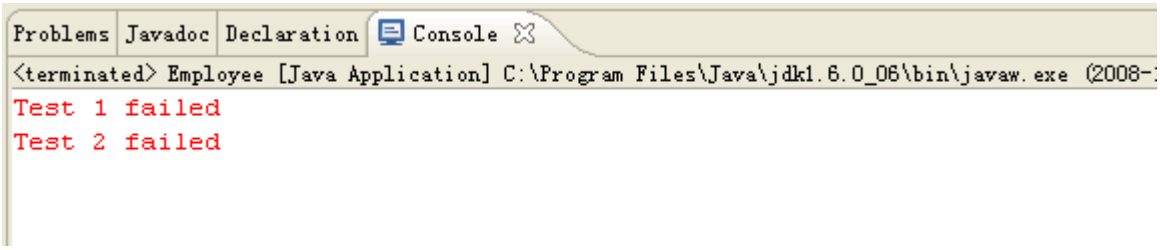
Import Class Employee

On the File menu, click Import.

In the Import wizard, click File System and then click Next.
In the File system dialog box, click Browse. In the Import from directory dialog box, locate the directory that contains the file Employee.java, click the directory name, and then click OK.
The wizard will display in the right pane of the window all the files in the selected directory. Select the check box next to Employee.java and type Employee in the Into folder box. Click Create selected folders only and then click Finish.
In the Package Explorer view at the left hand of the window, double-click Employee and then double-click default-package. This will display the files in the application Employee. Double-click Employee.java. The code will appear in the editor with syntax highlighting.

Execute the Application

Click the arrow to the right of the Run button in the toolbar, point to Run As and then click Java Application.
Eclipse will display the output of the application in the Console view. This output indicates that class Employee has an error. In the following sections, we will show how to find and correct the error using the Eclipse debugger.

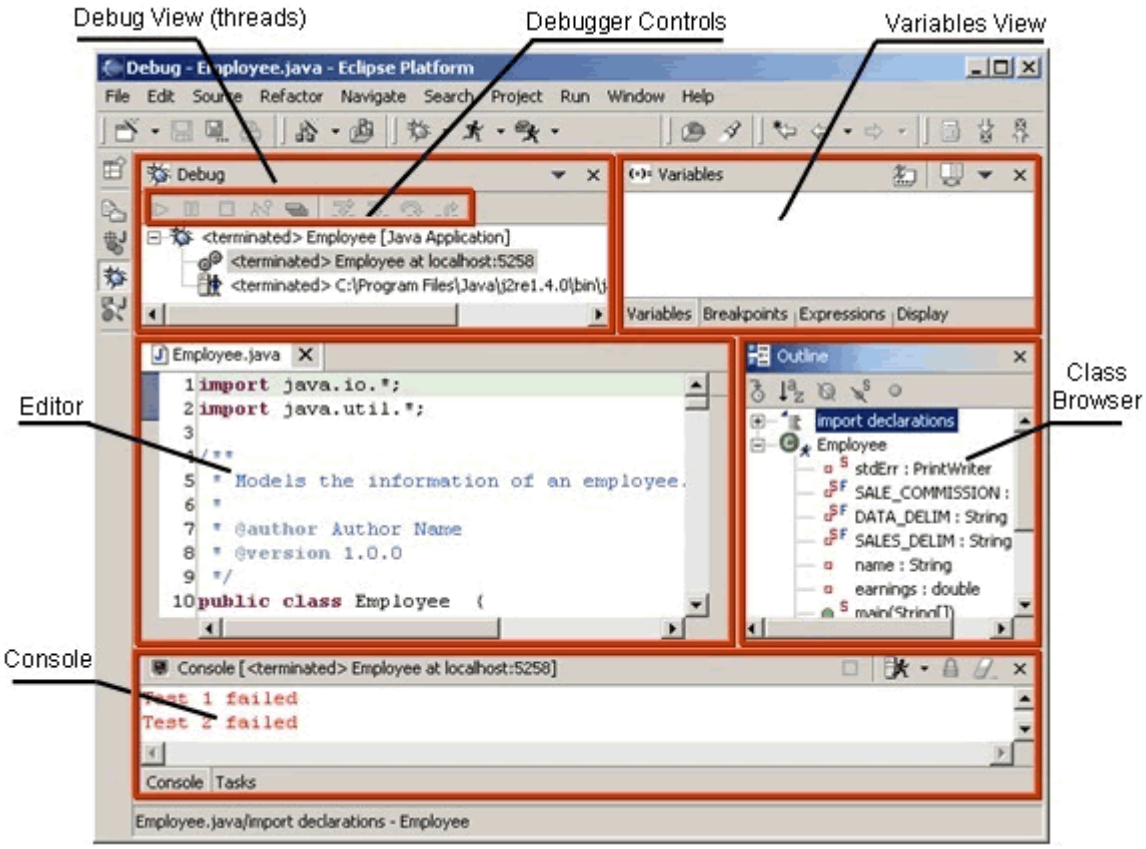


Show Line Numbers

Before looking for errors, modify the Java editor preferences so that the editor will include line numbers.
On the Window menu, click Preferences.
In the left pane of the Preferences dialog box, double-click Java, and then click Editor.
Click the Show line numbers check box and then click OK

Debug the Class

To debug the class, click the arrow to the right of the Debug button on the toolbar. Point to Debug As and then click Java Application.
Eclipse will show the Debug perspective of the project.



The Debug perspective contains an editor and four views:

- Debug view: Displays the methods that have been called
- Variables view: Displays the values of the variables
- Class browser: Displays the variables and methods of the classes
- Console view: Displays the output to System.err and System.out
- The Debug view has a toolbar with buttons that can control the execution of the code

Place a Breakpoint

Class Employee contains a test case that outputs a message when it detects an error. As you can see in this example, it is a good idea to define test cases that verify that the program is working as expected.

The next step is to locate the error and the debugger is a great tool for this task. Typically, breakpoints are placed in the part of the code where we want to suspend execution and examine the values of the variables.

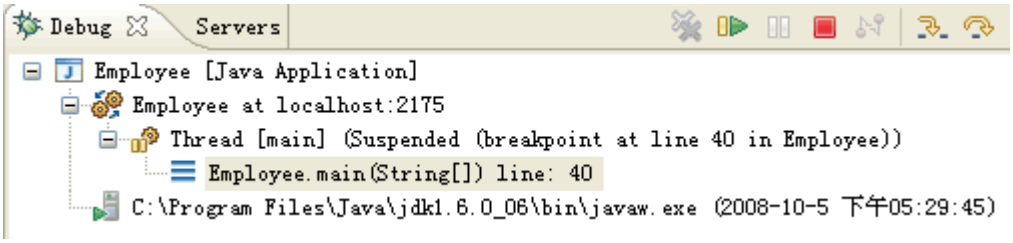
A good point to begin is the line that reports the error. (See debugging technique described in page 1.1.8 Debugging.) In this example, line 40 in method main reports the error.

In the editor, go to line 40. Right-click the line number and click Add Breakpoint Then a blue circle representing the breakpoint will appear to the left of the line number.

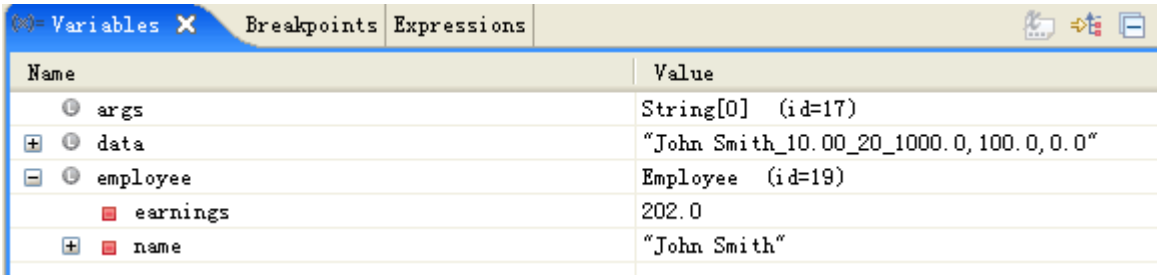
```
36          try {
37              if (employee.getEarnings() == 222.0) {
38                  stdErr.println("Test 1 passed");
39              } else {
40                  stdErr.println("Test 1 failed");
41              }
42          } catch (NumberFormatException  iae)  {
43              stdErr.println("Test 1 failed, " + iae);
44          }
```

Monitor the Execution

1.On the toolbar, click debug to "launch" the application Employee in debug mode.
2.The Debug view shows much information that we can ignore for the moment. We will focus our attention on the "Thread[main]" section. This section shows the methods that were called before execution reached the breakpoint. It also shows that execution was suspended at line 40 of method main in class Employee.



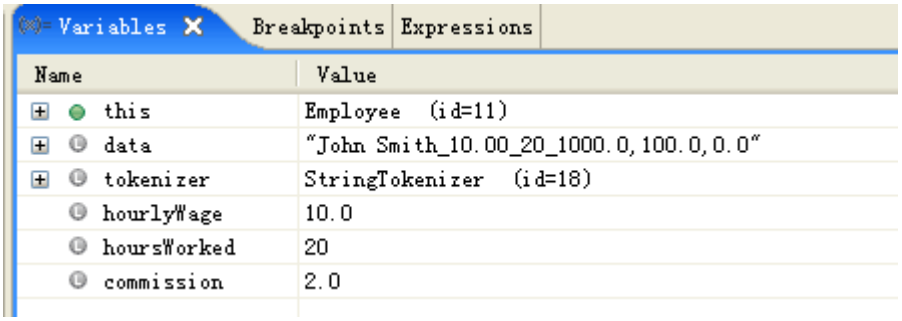
The Variables view shows the value of the variables at this point. Click the plus sign (+) next to the variable employee. This will display the contents of the object.



The value of variable earnings is wrong: it is 202.0 and it should be 222.0. The next step is to look at the code where the value of earnings is modified. Line 95 assigns a value to earnings.
4.On the Debug toolbar in the Debug view, click terminate to abort execution of the application.
5.Double-click the blue mark at line 40 to remove the breakpoint.
6.Go to line 95. Right-click the line number and click Add Breakpoint.

```
94
95              earnings = hourlyWage * hoursWorked + commission;
96
97          } catch (NumberFormatException  nfe)  {
98              throw new NumberFormatException(
99                  "bad employee data: " + data);
```

7.On the toolbar, click bug to launch the application in debug mode again.
8.Execution stops at line 95. The Variables view shows the value of the variables at this point.



The values of hourlyWage and hoursWorked are correct, but the value of commission is 2.0 and it should be 22.0. The next step is to look at the code where the value of commission is modified. Line 93 calls method computeCommission and assigns the value returned by method computeCommission to the variable commission.

9.On the Debug toolbar, click terminate to abort execution of the application.

10.Double-click the blue mark at line 95 to remove the breakpoint.

11.Method computeCommission uses a StringTokenizer object and a while-loop to sum of the sales. Go to line 144 to monitor the behavior of the while-loop. Right-click line 144 and click Add Breakpoint.

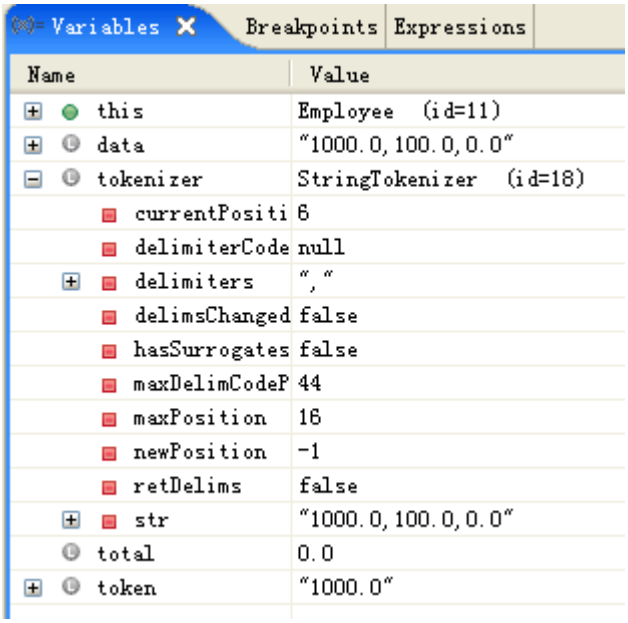
```
142             String token = tokenizer.nextToken();
143
144             while (tokenizer.hasMoreTokens()) {
145                 token = tokenizer.nextToken();
146                 total += Double.parseDouble(token);
147             }
```

12.On the toolbar, click bug to "launch" the application in debug mode again.

13.The execution stops at line 144, before the while-loop.

```
142             String token = tokenizer.nextToken();
143
144             while (tokenizer.hasMoreTokens()) {
145                 token = tokenizer.nextToken();
146                 total += Double.parseDouble(token);
147             }
```

The Variables view shows the value of the variables at this point.

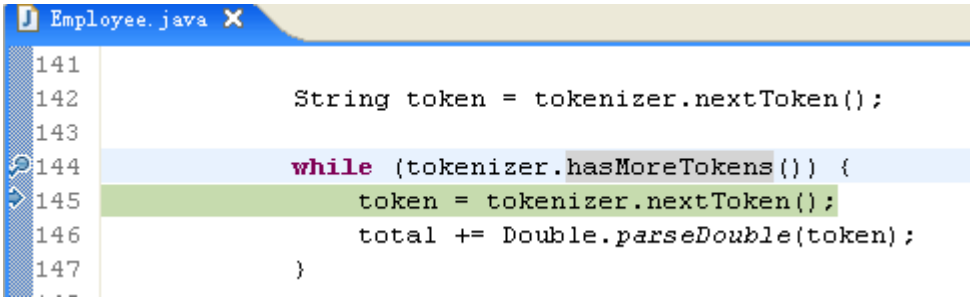


Before execution of the while-loop begins, total contains 0.0, token contains the first token ("1000.0"), and tokenizer indicates the position of the next token in data.

15. We will use the step controls to monitor the behavior of the while-loop. The Debug toolbar in the Debug view contains the following buttons

resume	Continue execution until next breakpoint.
step into	Execute current line. If current line calls a method, the debugger "steps into" the method, that is, execution moves to the *called* method and stops before the first line in the called method.
step over	Execute current line. If current line calls a method, the debugger "steps over" the method, that is, the method is executed and execution stops before the next line in the *current* method. If there are no more lines in the method, execution returns to the *calling* method and stops before the next line in the *calling* method.
step return	Execute until current method completes and return to the *calling* method. Execution stops before the next line in the *calling* method.
terminate	Abort execution.

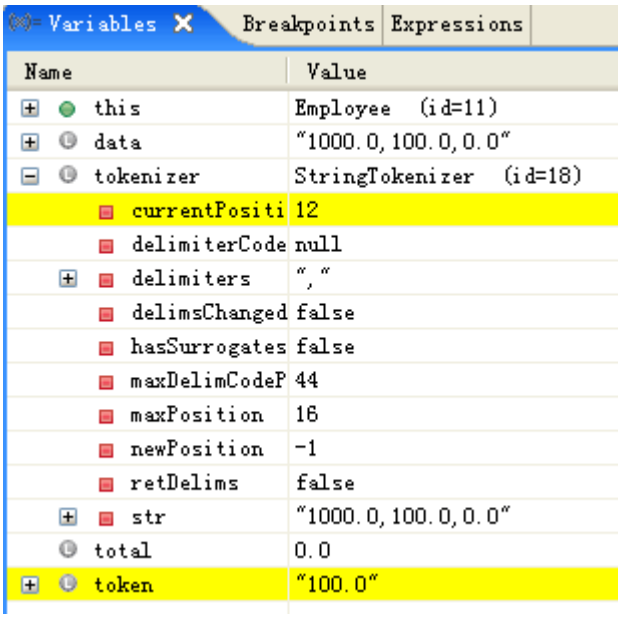
16. On the Debug toolbar, click step over to execute the current line. Execution stops before line 145.



The variable tokenizer indicates the location of the next token in data.

17. Click step over to execute the current line. Execution stops before line 146

18. The variable `token` now contains the second token ("100.0"), but `total` is still 0.0. The information in the first token has been lost.



19.Click terminate to abort execution.

20.Double-click the blue mark at line 144 to remove the breakpoint.

Diagnostic and Fix the Error

We discovered that the value in the first token is not added to total. One common mistake is disguising the error instead of fixing it. For example, we might think the order of lines 145 and 146 is wrong and swap them:

```
140         try {
141
142             String token = tokenizer.nextToken();
143
144             while (tokenizer.hasMoreTokens()) {
145                 total += Double.parseDouble(token);
146                 token = tokenizer.nextToken();
147             }
148
149         } catch (NumberFormatException nfe) {
150             throw new NumberFormatException(
```

1.Swap the lines and save the file.

2.Run the application to test the fix. Click the arrow to the right of the Run button on the toolbar, point to Run As and then click Java Application

3. The Console view shows that the second test case failed.

Before trying to correct the code, it is important to analyze the code so that we have a clear understanding of the error. Reviewing the code, we notice that the structure of the while-loop is wrong. The method hasMoreTokens should be called before reading each token, and the value of the sale should be added to the variable total immediately after the token is read.

4. Correct the code in method `computeCommission` as follows:

```
140         try {
141
142             while (tokenizer.hasMoreTokens()) {
143                 String token = tokenizer.nextToken();
144                 total += Double.parseDouble(token);
145             }
146
147         } catch (NumberFormatException  nfe)  {
148             throw new NumberFormatException(
149                 "bad sales data: " + data);
150         }
```

5. Run the application to test the fix. Click the arrow to the right of the Run button on the toolbar, point to Run As and then click Java Application.

6.The Console view shows the output: The output indicates that the application passed both test cases.

Source of Employee.java

```
import java.io.*;//Line 1
import java.util.*;

/**
 * Models the information of an employee.
 *
 * @author Author Name
 * @version 1.0.0
 *
 */
public class Employee   {

    private static PrintWriter   stdErr =
        new PrintWriter(System.err, true);

    private final static double   SALE_COMMISSION = 0.02;

    private final static String   DATA_DELIM = "_";

    private final static String   SALES_DELIM = ",";

    private String   name;

    private double   earnings;

    /**
```

```
  * Test case for class <code>Employee</code>
  *
  * @param args    not used.
  */
public static void main(String[] args) {

        String    data = "John Smith_10.00_20_1000.0,100.0,0.0";

        Employee    employee = new Employee(data);

        try {
                if (employee.getEarnings() == 222.0) {
                        stdErr.println("Test 1 passed");
                } else {
                        stdErr.println("Test 1 failed");
                }
        } catch (NumberFormatException    iae)    {
                stdErr.println("Test 1 failed, " + iae);
        }

        data = "Ruchi Dong_30.00_40_2000.0,200.0,600.0";

        employee = new Employee(data);

        try {
                if (employee.getEarnings() == 1256.0) {
                        stdErr.println("Test 2 passed");
                } else {
                        stdErr.println("Test 2 failed");
                }
        } catch (NumberFormatException    iae)    {
                stdErr.println("Test 2 failed, " + iae);
        }
}

/**
  * Constructs an <code>Employee</code> object with the
  * information specified in the parameter <code>data</code>.
  * <p>
  * The input data has the following format:
  * </p>
  * <p>
  * name_hourlyWage_hoursWorked_sale1,sale2,...,salen
  * </p>
```

```
     * <p>
     * The sale fields store the amount of each sale made by the
     * employee.
     * </p>
     *
     * @param data    a String with employee information.
     * @throws NumberFormatException    if data contains invalid
     *                                  numbers
     */
    public Employee(String data) throws NumberFormatException    {

        StringTokenizer tokenizer =
            new StringTokenizer(data, DATA_DELIM);

        try {
            name = tokenizer.nextToken();

            double hourlyWage =
                Double.parseDouble(tokenizer.nextToken());
            int hoursWorked =
                Integer.parseInt(tokenizer.nextToken());

            double commission = (tokenizer.hasMoreTokens()) ?
                                    computeCommission(tokenizer.nextToken()) : 0;

            earnings = hourlyWage * hoursWorked + commission;

        } catch (NumberFormatException    nfe)    {
            throw new NumberFormatException(
                "bad employee data: " + data);
        }
    }

    /**
     * Obtains the name of the employee.
     *
     * @return a String with the name of the employee.
     */
    public String    getName()    {

        return name;
    }

    /**
```

```
     * Obtains the earnings of the employee.
     *
     * @return the earnings of the employee.
     */
    public double    getEarnings()    {

        return earnings;
    }

    /*
     * Computes the commission of the employee with the data
     * specified in the parameter.
     *
     * @param data    a String with the employee's sales data.
     * @return the commission earned by the employee.
     * @throws NumberFormatException    if data contains invalid
     *                                          numbers
     */
    private double    computeCommission(String data)
        throws NumberFormatException {

        StringTokenizer tokenizer =
            new StringTokenizer(data, SALES_DELIM);

        double total = 0.0;

        try {

            String token = tokenizer.nextToken();

            while (tokenizer.hasMoreTokens()) {
                token = tokenizer.nextToken();
                total += Double.parseDouble(token);
            }

        } catch (NumberFormatException    nfe)    {
            throw new NumberFormatException(
                "bad sales data: " + data);
        }

        return total * SALE_COMMISSION;
    }
}
```

任务 1

1）阅读《软件测试的艺术（第 3 版）》第 8 章，简介常用的软件调试方法。

2）使用 UML 类图描述 Employee.java 的结构。（用 StarUML 建模后，把模型导出为图片插入到 Word 文档中）

3）结合 Employee.java 调试实践，讲述 Java 程序调试的断点调试过程及技巧。

任务 2

编码实现 NextDay 类（如图 1 所示），可以根据用户输入的 year、 month、 day 的值输出用户输入日期对应的下一天。备注：要求年的取值范围为 1900 年~3000 年。如果用户输入的 year、month、day 不在给定范围内或用户输入不是正整数时，提示“日期格式不正确，请重新输入！”。编码时给出适当的注释。

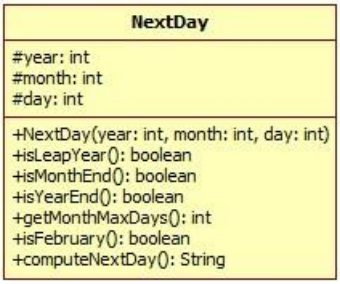


图 1　NextDay 类框架设计

任务 3：

使用表 1 测试用例对该 NextDay 进行单元测试和软件调试，确保所有测试用例都通过。如果发现缺陷，在实验报告中记录软件调试和回归测试的详细过程、结果。

表 1 测试用例一览表

编号	输入数据			预期输出（字符串格式）	备注
	year	month	day		
#1	2001	1	1	2001年1月2日	
#2	2020	2	28	2020年2月29日	
#3	2020	2	29	2020年3月1日	
#4	2022	6	30	2022年7月1日	
#5	2019	2	29	日期格式不正确，请重新输入！	
#6	1800	1	1	日期格式不正确，请重新输入！	提示用户，年的输入范围为1900-3000

任务 4：

将 NextDay 源码、实验报告 PDF 文档打包为 zip 文件提交，文件命名格式为“学号-姓名-实验 1 报告及源码（完成日期）.zip”。

实验报告要求

编写电子稿实验报告（PDF 格式），描述实验目标、相关知识、过程、实验结果（包括阶段性结果）、实验任务求解过程及结果、实验体会。