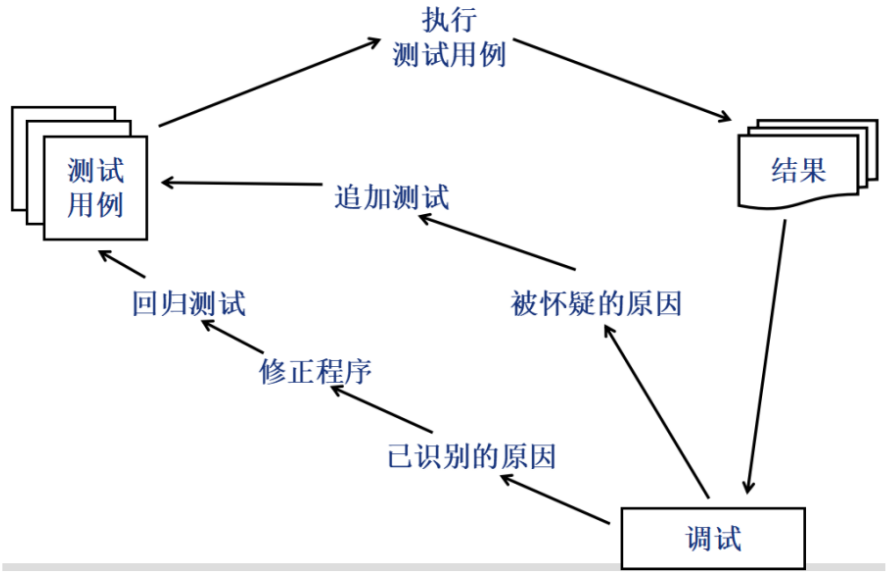
实验名称	实验 一 : 程序员视角的软件调试与软件测试
实验目的	了解软件调试原理； 掌握 Eclipse 调试器的使用方法及技巧。
实验设备	IDEA
实验日期	2024.3.13

一、知识准备



二、实验内容 (原理、方法)

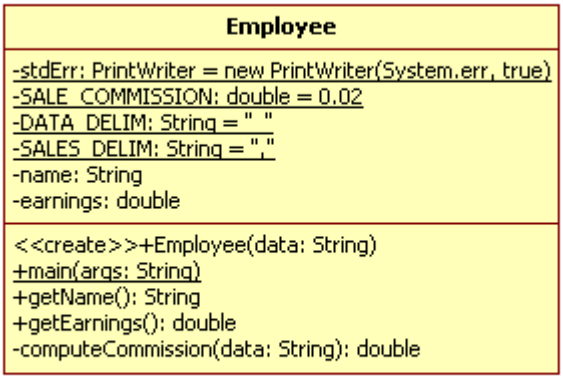
In this section, we will demonstrate the debugging process using a debugger. We will use the debugger included in Eclipse. The basic functions provided by this debugger are similar to those provided by other debuggers. Therefore, you can also walk through this tutorial with another debugger.

三、实验过程

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。

打印调试：在关键代码段中插入打印语句，输出关键变量的值和程序执行的状态，以便跟踪代码执行路径和发现问题。
日志记录：使用日志记录工具，在程序执行过程中记录关键信息和事件，以便在后续分析和调试时使用。
调试器：使用集成开发环境（IDE）或独立的调试器工具，设置断点并逐步执行程序，观察变量的值、执行流程，并进行逐行调试。
故障注入：有意地在代码中引入故障或错误，以测试程序对异常情况的处理能力，并通过调试来修复问题。
追踪和分析：使用追踪工具和性能分析工具来监视程序的执行，收集性能数据和调用堆栈信息，以便定位性能瓶颈和优化代码。
单元测试和集成测试：编写针对特定模块或功能的测试用例，并使用测试框架执行这些测试，以验证代码的正确性和功能是否符合预期。
代码审查：与团队成员一起仔细审查代码，发现潜在的错误、逻辑问题和不一致之处，并进行修复。
崩溃转储分析：当程序崩溃时，生成崩溃转储文件（core dump），然后使用调试器分析转储文件，以了解崩溃原因。

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。

在开发环境中打开 Employee.java 文件，并找到 calculateSalary() 方法。在方法的起始位置或你想要调试的代码行上设置一个断点。你可以在行号旁边单击，或使用快捷键（通常是 F9）来设置断点。

运行程序，启动调试模式。这通常是通过点击运行按钮旁边的调试按钮来完成的。程序将在断点处停止执行，等待你逐行调试。

一旦程序停在断点处，你可以使用调试器提供的各种功能来逐行执行代码并观察变量的值。以下是一些常用的调试器技巧：

单步执行：使用调试器的单步执行功能，逐行执行代码。这可以帮助你观察代码的执行流程，并检查变量的值是否符合预期。

查看变量：在调试器的变量窗口中，查看当前的变量值。你可以监视对象的属性值、局部变量和全局变量的变化。

条件断点：设置断点时，可以为断点添加条件。这样，只有当满足特定条件时，程序才会在该断点处停止执行。这对于调试特定的情况或特定的输入数据很有帮助。

异常断点：除了在特定代码行上设置断点，还可以设置异常断点。这将在异常被抛出时暂停程序的执行，让你可以查看异常的调用堆栈和相关信息。

修改变量：在调试过程中，有些调试器允许你修改变量的值。这样你可以尝试不同的值，以验证对程序行为的影响。

监视表达式：在调试器中，你可以设置监视表达式来跟踪某个表达式的值。这对于观察某个特定条件的变化非常有用。

通过逐行调试和观察变量值，你可以定位代码中的问题，理解程序的执行流程，并验证算法的正确性。

如果你发现了问题，你可以修复代码并重新运行进行调试，直到问题解决。

任务 2

```
package exp1;
import java.util.Scanner;
public class NextDay {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // 提示用户输入年份
        System.out.print("请输入年份（1900 年~3000 年）：");
        int year = getUserInput(scanner);
        // 提示用户输入月份
        System.out.print("请输入月份（1 月~12 月）：");
        int month = getUserInput(scanner);
        // 提示用户输入日期
        System.out.print("请输入日期（1 日~31 日）：");
        int day = getUserInput(scanner);
        // 检查输入的年份、月份和日期是否在给定范围内
        if (isDateValid(year, month, day)) {
            // 调用 getNextDay 方法获取下一天的日期
            String nextDay = getNextDay(year, month, day);
            // 输出下一天的日期
            System.out.println("下一天的日期是：" + nextDay);
        } else {
            System.out.println("日期格式不正确，请重新输入！");
            if(year<1900||year>3000)
                System.out.println("年的输入范围为 1900-3000");
        }
    }
    // 获取用户输入，并检查输入是否为正整数
    public static int getUserInput(Scanner scanner) {
        int input;
        while (true) {
            if (scanner.hasNextInt()) {
                input = scanner.nextInt();
                if (input > 0)
                    break;
            } else
                scanner.next(); // 清除无效输入
            System.out.println("日期格式不正确，请重新输入！");
            System.out.print("请输入正确的数字：");
        }
        return input;
    }
    // 检查年份、月份和日期的范围
    public static boolean isDateValid(int year, int month, int day) {
        return year >= 1900 && year <= 3000 && month >= 1 && month <= 12 && day >= 1 &&
day <= getDaysInMonth(year, month);
    }
    // 获取每个月的天数
    public static int getDaysInMonth(int year, int month) {
```

四、实验结果（遇到的问题及解决方法）

任务 3

请输入年份（1900年~3000年）：*2001*
请输入月份（1月~12月）：*1*
请输入日期（1日~31日）：*1*
下一天的日期是：2001年1月2日

请输入年份（1900年~3000年）：*2020*
请输入月份（1月~12月）：*2*
请输入日期（1日~31日）：*28*
下一天的日期是：2020年2月29日

请输入年份（1900年~3000年）：*2020*
请输入月份（1月~12月）：*2*
请输入日期（1日~31日）：*29*
下一天的日期是：2020年3月1日

请输入年份（1900年~3000年）：*2022*
请输入月份（1月~12月）：*6*
请输入日期（1日~31日）：*30*
下一天的日期是：2022年7月1日

请输入年份（1900年~3000年）：*2019*
请输入月份（1月~12月）：*2*
请输入日期（1日~31日）：*29*
日期格式不正确，请重新输入！

请输入年份（1900年~3000年）：*1800*
请输入月份（1月~12月）：*1*
请输入日期（1日~31日）：*1*
日期格式不正确，请重新输入！
年的输入范围为1900-3000

五、心得体会 (实验收获)

设置断点：在需要调试的代码行上设置断点，这样程序执行到该行时会暂停。可以通过双击代码行或右键单击选择"Toggle Breakpoint"来设置断点。设置合适的断点可以帮助我们聚焦于问题所在的代码。

运行调试模式：选择要调试的 Java 类文件，然后点击 Eclipse 工具栏上的"Debug"按钮或使用快捷键进行调试。程序将在调试模式下运行，遇到断点时会暂停。

观察变量和表达式：在调试模式下，可以查看当前变量的值以及计算表达式的结果。在"Variables"视图中可以查看局部变量和成员变量的值。在"Expressions"视图中可以计算并查看任意表达式的结果，这对于检查条件和计算中间值非常有用。

单步调试：一旦程序暂停在断点处，可以使用调试器提供的单步调试功能逐行执行代码。可以使用"Step Into"进入方法内部，"Step Over"跳过当前方法，"Step Return"从当前方法返回等。

监视变量：在调试过程中，可以选择要监视的变量并将其添加到"Watch"视图中。这样可以实时查看这些变量的值，有助于跟踪变量的变化和调试复杂逻辑。

条件断点：可以为断点添加条件，只有当满足条件时才会触发断点。这对于在特定条件下调试代码非常有用，可以帮助我们定位特定情况下的问题。

异常处理：在调试过程中，可以选择在异常发生时暂停程序执行。这样可以捕获和调试异常，并查看异常抛出时的堆栈跟踪信息。

知识准备	实验过程	问题思考	综合评价