浙江大学

本科实验报告

姓名：

学院：　生物医学工程与仪器科学学院

系：　生物医学工程系

专业：　生物医学工程

学号：

指导教师：　耿晨歌

2024 年 10 月 24 日

浙江大学实验报告

课程名称：高级程序设计　　实验类型：

实验项目名称：计算代码行数

同组学生姓名：

指导老师：耿晨歌

实验地点：　　实验日期：2024 年 10 月 24 日

一、实验目的和要求（必填）

本实验的目的是编写一个 Java 程序，统计指定目录下所有 Java 文件中的代码行数、注释行数和空白行数，验证程序统计结果的准确性。

二、实验内容和原理（必填）

写一个 Java 程序，统计一个项目中一共有多少行代码，多少行注释，多少个空白行，通过命令行参数给定目录。对目录下所有 java 文件，扫描其中有：

- 多少行代码
- 多少行注释
- 多少空白行

参考：cloc

三、主要仪器设备

Vscode

四、操作方法与实验步骤

（一）了解 cloc

Cloc（Count Lines of Code）是一款用 Perl 语言开发的开源代码统计工具，它支持多平台使用，并且能够识别和统计多种编程语言中的代码行数、注释行数以及空白行数。其可通过 npm 安装：npm install -g cloc，当然在这之前先要安装 perl。

基本命令格式为：cloc [选项] <路径>，其中 <路径> 是需要统计的代码库路径。

例如，统计当前目录下的代码行数，可以使用命令：cloc .。

排除特定目录或文件，可以使用：cloc --exclude-dir=<dir> .。

按文件统计代码行数，可以使用：cloc --by-file .。

输出结果到文件，可以使用：cloc . --out=result.txt。

（二）程序架构

1、程序架构

命令行参数和读取文件模块：通过命令行传入目录路径，并通过 nio 递归读取 Java 文件。

遍历目录模块：遍历指定目录下的所有文件，筛选出以“.java”结尾的文件。

判断注释模块：根据正则表达式和逻辑判断代码、注释和空白行的类型。

2、详细步骤

使用 Files.walk()递归遍历目录下的所有文件，筛选出 Java 文件；

逐行读取 Java 文件内容，判断每行是否为空行、代码行或注释行；

对于注释部分：

如果是单行注释，直接计入注释行；

如果是多行注释，需要检查注释的开始和结束符，并正确识别注释块是否跨行；

对结果进行输出，并与 cloc 结果对比。

（三）Junit 测试代码

JUnit 是一个 Java 编程语言的开源测试框架，它是 xUnit 架构的一部分，xUnit 是为多种编程语言创建的单元测试库的通用名称。下面是一些常见的 JUnit 注解：

@Test：该注解表示该方法是一个测试方法，当你运行 JUnit 测试时，这个方法将会被执行。你可以给这个注解提供一个可选的参数来指定预期的异常类型，如果测试方法抛出了该类型的异常，那么测试就会通过。

@BeforeEach（JUnit 5）/ @Before（JUnit 4）：这个注解用于标注一个方法，在每一个测试方法运行前，这个方法都会被执行。这在你需要为每个测试方法做一些准备工作，如初始化一些公共的变量或对象时，会非常有用。

@AfterEach（JUnit 5）/ @After（JUnit 4）：这个注解用于标注一个方法，在每一个测试方法运行后，这个方法都会被执行。这在你需要在每个测试方法后做一些清理工作时，会非常有用。

@BeforeAll（JUnit 5）/ @BeforeClass（JUnit 4）：这个注解用于标注一个方法，在所有测试方法运行前，这个方法会被执行一次。这对于一次性的初始化工作，如打开数据库连接，很有用。注意，这个注解标注的方法必须是静态的。

@AfterAll（JUnit 5）/ @AfterClass（JUnit 4）：这个注解用于标注一个方法，在所有测试方法运行后，这个方法会被执行一次。这对于一次性的清理工作，如关闭数据库连接，很有用。注意，这个注解标注的方法必须是静态的。

@Ignore：这个注解表示该测试方法被忽略，即在运行测试时，这个方法不会被执行。你可以为这个注解提供一个可选的参数来说明为什么这个测试方法被忽略。

@ParameterizedTest（JUnit 5）：这个注解用于表示一个参数化的测试方法。与之配合使用的还有一些注解，如@ValueSource、@EnumSource、@MethodSource、@CsvSource 等，它们用于提供测试数据。

五、实验数据记录和处理

Cloc 处理结果：

```
D:\MyLab\大三秋冬\高级程序设计\作业\hw1>cloc --by-file .
       5 text files.
       5 unique files.
       3 files ignored.

github.com/AlDanial/cloc v 2.02  T=0.03 s (155.9 files/s, 4706.8 lines/s)
-------------------------------------------------------------------------------
File                                     blank        comment           code
-------------------------------------------------------------------------------
.\CodeCounter.java                          12              5             84
.\.vscode\launch.json                        0              0             12
.\测试样例\Test3\Test3.java                  2              2              6
.\测试样例\Test1.java                        4              5              5
.\测试样例\Test2.java                        3              6              5
-------------------------------------------------------------------------------
SUM:                                        21             18            112
-------------------------------------------------------------------------------
```

CodeCounter 处理结果：

```
PS D:\MyLab\大三秋冬\高级程序设计\作业\hw1> javac CodeCounter.java
PS D:\MyLab\大三秋冬\高级程序设计\作业\hw1> java CodeCounter.java D:/MyLab/大三秋冬/高级程序设计/作业/hw1
HW1: 计算代码行数
File:D:\MyLab\大三秋冬\高级程序设计\作业\hw1\CodeCounter.java
Code Lines:84
Comment Lines:5
Blank Lines:12
File:D:\MyLab\大三秋冬\高级程序设计\作业\hw1\测试样例\Test1.java
Code Lines:5
Comment Lines:5
Blank Lines:4
File:D:\MyLab\大三秋冬\高级程序设计\作业\hw1\测试样例\Test2.java
Code Lines:5
Comment Lines:7
Blank Lines:3
File:D:\MyLab\大三秋冬\高级程序设计\作业\hw1\测试样例\Test3\Test3.java
Code Lines:6
Comment Lines:2
Blank Lines:2
File:D:\MyLab\大三秋冬\高级程序设计\作业\hw1\测试样例\Test3\Test4\Test4.java
Code Lines:0
Comment Lines:0
Blank Lines:1
```

junit 处理结果：

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.bubblevan:demo >------------------------
[INFO] Building demo 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.0.2:resources (default-resources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\MyLab\大三秋冬\高级程序设计\作业\hw1\demo\src\main\resources
[INFO]
[INFO] --- compiler:3.8.0:compile (default-compile) @ demo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.0.2:testResources (default-testResources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\MyLab\大三秋冬\高级程序设计\作业\hw1\demo\src\test\resources
[INFO]
[INFO] --- compiler:3.8.0:testCompile (default-testCompile) @ demo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- surefire:2.22.1:test (default-test) @ demo ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.bubblevan.CodeCounterTest
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.02 s - in com.bubblevan.CodeCounterTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.766 s
[INFO] Finished at: 2024-10-25T20:33:01+08:00
[INFO] ------------------------------------------------------------------------
```

六、实验结果与分析（必填）

1、在 Test1 里最开始 cloc 结果是 356，而我的是 359，说明我对内联注释进行判断的代码不够健壮，这导致一些注释行被误判为代码行。

2、在修改代码的过程中，连续几次都得到了一样的结果，在散心回来之后才明白原来是没有再次运行 javac 编译而是直接运行导致。

3、比较结果可以看出，CodeCounter 和 cloc 在 Test2 的判断上有歧义。主要集中在下面：

```
    /* Starting a block comment */ System.out.println("Mixed comment and code"); /* Ending comment */
```

cloc 认为这行是注释，而我的程序将其视为代码行。关于此类情况，统计标准有所不同，编写习惯也并不规范，但我坚持认为代码与注释共存的行应该视为代码行。

4、使用 Junit 单元测试时，出现了末尾空行无法被识别成 blankLine 的情况，但是单独在 java 文件中运行却能识别，经过好一番排查后才发现可能原因是因为输入样例被错误判定成了字符串结尾的空格而非换行符。

```
    @Test
    public void testCountLinesInContent_MixedContent() {
        String content = """
            // 单行注释
            /* 多行注释开始
            多行注释继续
            多行注释结束 */

            public class HelloWorld {
                public static void main(String[] args) { // 内嵌注释
                    System.out.println("Hello, World!");
                }
            }

            """;
        Map<String, Integer> result = CodeCounter.countLinesInContent(content);
        // 根据 CodeCounter 的实现，闭合的大括号也被计为代码行
        assertEquals(5, result.get("codeLines"), "代码行数应为 5");
        assertEquals(4, result.get("commentLines"), "注释行数应为 4");
        assertEquals(1, result.get("blankLines"), "空行数应为 1");
    }
```

5、在判断空白文件的特殊情况时，我只能创建一行空白行的 java 文件，但是 Junit 可以直接用 null 作为测试样例，而我的输入判断里 String.split(String)方法没法处理 content 为 null 的情况，所以需要额外增加判断。

七、讨论、心得

通过本次实验，我加深了对 Java 文件读写、字符串处理的理解。在比较 cloc 工具和自己编写的程序时，发现处理复杂注释场景时容易出现误判，这提醒我在处理真实项目时要更加仔细和全面地考虑各种边界情况。同时，这次实验也让我对命令行工具和代码自动化统计有了更深刻的认识。

八、附件（代码）

```
package com.bubblevan;

import java.io.IOException;
```

```java
import java.nio.file.*;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Stream;
public class CodeCounter {
    public static void main(String args[]) {
        System.out.println("HW1: 计算代码行数");
        if (args.length == 0) {
            System.out.println("请输入命令行参数！");
            return;
        }
        String dipath = args[0]; // 命令行第一个参数为目录路径
        try (Stream<Path> paths = Files.walk(Paths.get(dipath))) {
            paths.filter(Files::isRegularFile)      // 过滤出普通文件
                 .filter(path -> path.toString().endsWith(".java")) // lambda 表达式
                 .forEach(CodeCounter::countInFile);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static void countInFile(Path file) {
        int codeLines = 0;
        int commentLines = 0;
        int blankLines = 0;
        boolean inBlockComment = false; // 是否在多行注释中的 flag

        try {
            String content = new String(Files.readAllBytes(file));
            String[] lines = content.split("\\r?\\n");

            for (String line : lines) {
                line = line.trim();

                if (line.isEmpty()) {
                    blankLines++;
                    continue;
                }

                if (inBlockComment) {
                    commentLines++;
                    if (line.endsWith("*/")) {
                        inBlockComment = false;
                        // 如果行有注释结束符后还有代码，继续计算代码行
                        int endIndex = line.indexOf("*/") + 2;
```

```
                String afterComment = line.substring(endIndex).trim();
                if (!afterComment.isEmpty()) {
                    codeLines++;
                }
            }
            continue;
        }

        // 检查多行注释开始
        if (line.startsWith("/*")) {
            inBlockComment = true;
            commentLines++;
            // 检查是否是单行内的多行注释
            if (line.endsWith("*/") && !line.equals("/*")) {
                inBlockComment = false;
                // 检查是否有代码在注释之前或之后
                int startIndex = line.indexOf("/*");
                String beforeComment = line.substring(0, startIndex).trim();
                if (!beforeComment.isEmpty()) {
                    codeLines++;
                }
                int endIndex = line.indexOf("*/") + 2;
                String afterComment = line.substring(endIndex).trim();
                if (!afterComment.isEmpty()) {
                    codeLines++;
                }
            }
            continue;
        }

        // 处理单行注释
        int commentIndex = line.indexOf("//");
        if (commentIndex != -1) {
            String codePart = line.substring(0, commentIndex).trim();
            if (!codePart.isEmpty()) {
                codeLines++; // 仅计为代码行
            } else {
                commentLines++; // 仅计为注释行
            }
        } else {
            codeLines++; // 如果没有注释，计为代码行
        }
    }
```

```
            System.out.println("File:" + file.toAbsolutePath());
            System.out.println("Code Lines:" + codeLines);
            System.out.println("Comment Lines:" + commentLines);
            System.out.println("Blank Lines:" + blankLines);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // 可测试的核心逻辑
    public static Map<String, Integer> countLinesInContent(String content) {
        int codeLines = 0;
        int commentLines = 0;
        int blankLines = 0;
        boolean inBlockComment = false;

        if (content == null) {
            Map<String, Integer> result = new HashMap<>();
            result.put("codeLines", 0);
            result.put("commentLines", 0);
            result.put("blankLines", 0);
            return result;
        }

        String[] lines = content.split("\\r?\\n");
        for (String line : lines) {
            line = line.trim();
            if (line.isEmpty()) {
                blankLines++;
                continue;
            }
            if (inBlockComment) {
                commentLines++;
                if (line.endsWith("*/")) {
                    inBlockComment = false;
                    String afterComment = line.substring(line.indexOf("*/") + 2).trim();
                    if (!afterComment.isEmpty()) codeLines++;
                }
                continue;
            }
            if (line.startsWith("/*")) {
                inBlockComment = true;
                commentLines++;
```

```
                if (line.endsWith("*/") && !line.equals("/*")) {
                    inBlockComment = false;
                    String beforeComment = line.substring(0, line.indexOf("/*")).trim();
                    String afterComment = line.substring(line.indexOf("*/") + 2).trim();
                    if (!beforeComment.isEmpty()) codeLines++;
                    if (!afterComment.isEmpty()) codeLines++;
                }
                continue;
            }
            int commentIndex = line.indexOf("//");
            if (commentIndex != -1) {
                String codePart = line.substring(0, commentIndex).trim();
                if (!codePart.isEmpty()) codeLines++;
                else commentLines++;
            } else {
                codeLines++;
            }
        }
        Map<String, Integer> result = new HashMap<>();
        result.put("codeLines", codeLines);
        result.put("commentLines", commentLines);
        result.put("blankLines", blankLines);
        return result;
    }
}

// 单行与多行测试
public class Test1 {
    public static void main(String[] args) {
        System.out.println("Hello, World!"); // Inline comment
    }

}



/*
 * This is a block comment
 * It spans multiple lines
 */
```

```
// 复杂混合行测试
```

```java
public class Test2 {
    public static void main(String[] args) { // code with inline comment
        /* Starting a block comment */ System.out.println("Mixed comment and code"); /* Ending comment */
    }
    /*
     * public void commentedMethod() {
     *    // System.out.println("This method is commented out.");
     * }
     */
}
```

```java
// 字符串中的注释符号

public class Test3 {
    public static void main(String[] args) {
        String example = "This is not a comment /* but looks like one */";
        System.out.println(example);
    }
}

// 里面的 Test4 是空文件
```

```java
package com.bubblevan;

import static org.junit.jupiter.api.Assertions.assertEquals;
import java.util.Map;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
public class CodeCounterTest {
    @BeforeEach
    public void setUp() {
        // 初始化
    }
    @Test
    public void testCountLinesInContent_MixedContent() {
        String content = """
            // 单行注释
            /* 多行注释开始
            多行注释继续
            多行注释结束 */
            public class HelloWorld {
                public static void main(String[] args) { // 内嵌注释
                    System.out.println("Hello, World!");
                }
```

```
        }
        """;
    Map<String, Integer> result = CodeCounter.countLinesInContent(content);
    // 根据 CodeCounter 的实现，闭合的大括号也被计为代码行
    assertEquals(5, result.get("codeLines"), "代码行数应为 5");
    assertEquals(4, result.get("commentLines"), "注释行数应为 4");
    assertEquals(1, result.get("blankLines"), "空行数应为 1");
}
@Test
public void testCountLinesInContent_EmptyContent() {
    String content = "";
    Map<String, Integer> result = CodeCounter.countLinesInContent(content);
    assertEquals(0, result.get("codeLines"), "代码行数应为 0");
    assertEquals(0, result.get("commentLines"), "注释行数应为 0");
    assertEquals(1, result.get("blankLines"), "空行数应为 1");
}
@Test
public void testCountLinesInContent_OnlyComments() {
    String content = """
        // 注释行 1
        /* 注释行 2
        注释行 3 */
        // 注释行 4
        """;
    Map<String, Integer> result = CodeCounter.countLinesInContent(content);
    assertEquals(0, result.get("codeLines"), "代码行数应为 0");
    assertEquals(4, result.get("commentLines"), "注释行数应为 4");
    assertEquals(0, result.get("blankLines"), "空行数应为 0");
}
@Test
public void testCountLinesInContent_OnlyCode() {
    String content = """
        public class Test {
            public static void main(String[] args) {
                System.out.println("Test");
            }
        }
        """;
    Map<String, Integer> result = CodeCounter.countLinesInContent(content);
    assertEquals(5, result.get("codeLines"), "代码行数应为 5");
    assertEquals(0, result.get("commentLines"), "注释行数应为 0");
    assertEquals(0, result.get("blankLines"), "空行数应为 0");
}
@Test
```

```
    public void testCountLinesInContent_CodeWithInlineComments() {
        String content = """
            public class Test { // 类注释
                public static void main(String[] args) { // 主方法注释
                    System.out.println("Test"); // 输出注释
                } // 方法结束
            } // 类结束
            """;
        Map<String, Integer> result = CodeCounter.countLinesInContent(content);
        assertEquals(5, result.get("codeLines"), "代码行数应为 5");
        assertEquals(0, result.get("commentLines"), "注释行数应为 0");
        assertEquals(0, result.get("blankLines"), "空行数应为 0");
    }
    @Test
    public void testCountLinesInContent_MultipleBlockComments() {
        String content = """
            /* 块注释 1 开始
            块注释 1 结束 */
            public class Test {
                /* 块注释 2 开始
                块注释 2 结束 */
                public void method() {
                    /* 块注释 3 */
                }
            }
            """;
        Map<String, Integer> result = CodeCounter.countLinesInContent(content);
        assertEquals(4, result.get("codeLines"), "代码行数应为 4");
        assertEquals(5, result.get("commentLines"), "注释行数应为 5");
        assertEquals(0, result.get("blankLines"), "空行数应为 0");
    }
    @Test
    public void testCountLinesInContent_NullContent() {
        String content = null;
        Map<String, Integer> result = CodeCounter.countLinesInContent(content);
        assertEquals(0, result.get("codeLines"), "代码行数应为 0");
        assertEquals(0, result.get("commentLines"), "注释行数应为 0");
        assertEquals(0, result.get("blankLines"), "空行数应为 0");
    }
}
```