

**计算机与信息学院**

**《 软件测试课程考核报告 》**

**报告**

开课学期： 2023 年 春季 学期

网选班号： 02

学 号： 2020162121

姓 名： 梁宸铭

完成日期： 2023 年 5 月 28 日

基于决策表方法的测试

梁宸铭

摘 要：对智能辅助新型冠状病毒肺炎问诊量表程序进行了实现。通过决策表的方式对智能辅助新型冠状病毒肺炎问诊量表程序中的主要方法进行了测试，对其他简单的方法使用了边界值分析法进行了测试，生成了若干测试用例并通过了测试。整个项目通过Gradle进行依赖的导入。在测试过程中使用了JUnit5进行单元测试，针对所测方法调用其他对象的问题，使用了Mockito进行对象的Mock。

**关键词：决策表;单元测试;Gradle;JUnit;Mockito**

# 1 技术栈

## 1.1 Gradle

Gradle是开源的自动化构建工具，与Maven相比更加灵活、简洁。由于Gradle变化十分快，在使用时要考虑版本兼容的问题。JUnit过高的版本在低版本的Gradle上不能使用。

本次项目使用的版本为7.4。

## 1.2 JUnit

本次课程作业使用了JUnit5进行测试。JUnit5是由JUnit Platform、JUnit Jupiter、JUnit Vintage这三个底层项目结合而成，其中JUnit Platform被看作一个在JAVA虚拟机上的测试启动框架，其本身也定义了测试引擎的API，允许开发在平台运行的测试框架。JUnit Jupiter是JUnit为编写测试代码者提供的编程模组和扩展模组，它在JUnit Platform上运行。JUnit Vintage是JUnit的低版本的的引擎，但是它需要类路径上存在JUnit 4.12以上的版本。

本次项目使用的版本为5.9.3。

## 1.3 Mockito

Mockito是一个JAVA的单元测试的模拟框架，和EasyMock和JMock略有相似。Mockito提供了简洁的API，句法和EasyMock类似。Mockito在执行方法前创建模拟的对象，通过mock()方法创建模拟的对象或者使用@mock注解。

本次项目使用的版本为5.3.1。

# 2 题目分析

题目要求设计一个智能辅助新型冠状病毒肺炎问诊量表，首先要求用户选择症状，症状有发热、咽喉痛、咳嗽、鼻塞、流鼻涕、胸闷、气急、呼吸困难、全身酸痛、乏力、腹泻这11种，以及“都没有”共12种选项，其中“都没有”选项与另外11种选项互斥。若没有选择“都没有”选项，则其他11种选项可进行多项选择。在用户采集完症状后，对各已经选择的症状进行详细信息采集，具体分为：

1.若选择“发热”选项，则采集目前提问、历史最高体温、发热时是否产生其他情况，是否服用退烧药，服用退烧药后体温这5项。

2.若选择“咳嗽”选项，则采集是否咳痰。

3.若选择“腹泻”选项，则采集是否腹泻加重。

4.若选择“胸闷”选项，则采集是否胸闷加重。

5.若选择“气急”选项，则采集是否气急加重。

在所有症状以及详细信息采集结束后，将进行时长采集，分为14天以上和14以下两个选项。

进行接触史信息采集，分为：1.有武汉（湖北）等疫区旅游史或居住史。2.有接触过武汉（湖北）等疫区的人员。3.有接触过疑似或确诊新型冠状病毒感染者。4.身边有多人出现发热、乏力、咳嗽、咽痛等。5.都没有。其中选项5与前4个选项互斥，若没有选择选项5，则可以对前4项进行多项选择。

最后进行逻辑判断给出答案，答案由症状和接触史有关，与症状时长和症状详细信息均无关。根据所给予条件，不同答案的产生的条件分别为：

表格 1 条件与输出

|  |  |  |
| --- | --- | --- |
| 序号 | 条件 | 输出 |
| 一 | 以上十一种症状一个或多个与四项接触史一个或多个 | 建议立刻发热门诊就诊，并且佩戴好口罩做好防护。佩戴医用外科口罩或N95口罩，避免乘坐公共交通工具 |
| 二 | 以上十一种症状一个或多个与无接触史 | 结合你提供的病情症状及接触史特点，建议继续在家监测体温，可适当服用治疗感冒的中成药，注意手卫生，多饮水，休息为主。若体温持续不退或体温大于38℃或不适症状加重，及时就诊 |
| 三 | 无症状与四项接触史一个或多个 | 建议你们全家及密切接触者居家隔离至少2周，在家期间建议戴口罩交流，条件允许时，尽量单独居住或居住在通风良好的单人房间。  多休息，多饮水，注意手卫生和常用物品卫生消毒。若出现发热及呼吸道症状，需马上前往医院就诊 |
| 四 | 无症状与无接触史 | 建议您戴口罩、勤洗手，避免人群聚集，减少不必要外出 |

# 3 决策表与测试用例生成

## 3.1 决策表简介

决策表是控制表的一种，它在1960年至1970年流行于商业。决策表是一种通过表格的形式将各种条件列出，并根据条件输出一系列结果的简洁的表示方法。决策表尤其适合if-else语句和switch-case语句的测试。在决策表中，每一个决策可以看作一个变量，它与条件变量有关联。每一个变量的变化都导致动作的改变。为了让决策表表示得更加精确，在优化决策表时要将无关变量进行合并，即无论其真值是多少都不改变结果。决策表有条件桩和动作桩两个部分。[1]

## 3.2 决策表生成

根据本题要求，答案的四项输出与症状、接触史有关。症状包含：发热、咽喉痛、咳嗽、鼻塞、流鼻涕、胸闷、气急、呼吸困难、全身酸痛、乏力、腹泻、都没有，共12种。接触史包含：有武汉（湖北）等疫区旅游史或居住史、有接触过武汉（湖北）等疫区的人员、.有接触过疑似或确诊新型冠状病毒感染者、身边有多人出现发热、乏力、咳嗽、咽痛等、都没有，共5种。输出共有4项，根据条件和输出确定条件桩和动作桩。

表格 2 根据题目生成的决策表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | | **规则** | | | |
| 条件 | *发热* |  |  |  |  |
| *咽喉痛* |  |  |  |  |
| *咳嗽* |  |  |  |  |
| *鼻塞* |  |  |  |  |
| *流鼻涕* |  |  |  |  |
| *胸闷* |  |  |  |  |
| *气急* |  |  |  |  |
| *呼吸困难* |  |  |  |  |
| *全身酸痛* |  |  |  |  |
| *乏力* |  |  |  |  |
| *腹泻* |  |  |  |  |
| *都没有* | 0 | 1 | 1 | 0 |
| *有武汉(湖北)等疫区旅游史或居住史* |  |  |  |  |
| *有接触过武汉(湖北)等疫区的人员* |  |  |  |  |
| *有接触过疑似或确诊新型冠状病毒感染者* |  |  |  |  |
| *身边有多人出现发热,乏力,咳嗽,咽痛等* |  |  |  |  |
| *都没有* | 0 | 1 | 0 | 1 |
| **动作** | *1* | 1 |  |  |  |
| *2* |  |  | 1 |  |
| *3* |  |  |  | 1 |
| *4* |  | 1 |  |  |

根据以上决策表可以看出，如将各个症状和各类接触史均看作一个动作桩输入，那么根据排列组合的数学知识，单计算症状的排列组合将会出现2046种情况。

公式 1 症状条件桩组合

但是根据卡诺图化简无关项的方法，症状条件桩中，前11种进行或逻辑运算在和最后一种情况进行与运算，同时前11种条件桩与最后一种条件桩互斥，即：选择前11种条件后将不能选择最后一种条件桩，同时只要保证前11项任意一项或多项被选择，则输出将会从输出一和输出二中产生。接触史条件桩同理，接触史条件桩的前4项与最后一项互斥，而且前4项中任选一项或者多项均会使输出限制在输出一和输出三中，选择了最后一项后将不能选择前4项。根据以上分析生成新的决策表。

表格 3 修改后的决策表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | | **规则** | | | |
| 条件 | *有症状* | 1 | 1 | 0 | 0 |
| *无症状* | 0 | 0 | 1 | 1 |
| *有接触史* | 1 | 0 | 1 | 0 |
| *无接触史* | 0 | 1 | 0 | 1 |
| **动作** | *1* | 1 | 0 | 0 | 0 |
| *2* | 0 | 1 | 0 | 0 |
| *3* | 0 | 0 | 1 | 0 |
| *4* | 0 | 0 | 0 | 1 |

## 3.3 根据决策表生成测试用例

决策表生成的是最主要的方法的测试用例。在代码实现的过程中还产生了其他方法，现根据决策表将最主要的方法的测试用例给出。其余方法的测试用例在代码实现中介绍。

表格 4 测试用例与预期结果

|  |  |
| --- | --- |
| 测试用例 | 预期结果 |
| (1,1) | 1 |
| (12,1) | 3 |
| (12,5) | 4 |
| (12,5) | 2 |

# 4 代码实现

## 4.1 功能代码

首先实现了症状采集功能和详细信息采集功能。方法命名为selectSymptom，使用switch-case语句，将不同症状分别填入不同case语句中。不同症状类型通过int类型的数字代替进行条件判断，传入switch语句中。针对特定的症状条件，在case语句中添加详细信息采集功能，例如：在发热症状条件中，系统应出现4项详细信息采集功能，每一项信息采集都需要系统打印提示信息，打开输入端口，用户输入信息，将输入信息传入特定的方法进行判断。对其创建单独的JAVA类，命名为Symptom.java。在这个类中另添加接触史信息采集方法。两个方法均为public类型。

对详细信息采集功能进行分析可知，所有的详细信息采集均为二元判断，即输出只有是与否两种。在其中对最高体温的信息采集和症状持续时间采集与其他略有不同，需要通过输入具体的数字并根据输入数字所在的区间对其进行判断，如：根据输入int类型变量是否在0~14这个区间来判断症状持续时间是否超过14天，根据输入float类型变量是否在42.0f~37.3f这个区间中来判断最高温度是否合理。综上所述，针对详细信息创建单独的JAVA类，命名为Detail.java，在这个类中创建三个public类型的方法。这三个方队供症状采集函数调用。

最后设置主类，命名为Main.java。这个类调用以上所需的方法，实现症状多项选择功能和给出对应输出这两个功能。逻辑如下：用户给出输入后判断，若没有选择“都没有”，则进入循环，从而实现多次选择，每次循环中插入一段代码，判断是否选择结束并终止循环，当把全部选项循环一遍后，应当终止循环，执行后面的语句。在判断是否选择“都没有”的if-else语句中添加标志，表明是否执行过循环语句，方便后面输出不同的答案。

功能代码见附录。

## 4.2 测试代码

针对Detail类进行测试，使用@ParameterizedTest进行参数化测试，采用边界值分析法进行测试。使用这个方法的原因是因为，Detail类的逻辑十分简单，每个函数的代码量很小。在离点、上点、内点分别设置输入值，利用assertEquals()方法直接将输入和预期值进行比较。

表格 5 symptomDurationTest测试用例

|  |  |
| --- | --- |
| 输入 | 预期输出 |
| 0 | 持续14天以内 |
| 1 | 持续14天以内 |
| 14 | 持续14天以内 |
| 13 | 持续14天以内 |
| 7 | 持续14天以内 |
| 15 | 持续14天以上 |
| 16 | 持续14天以上 |
| 20 | 持续14天以上 |

表格 6 TemperatureTopTest测试用例

|  |  |
| --- | --- |
| 输入 | 预期输出 |
| 37.3 | true |
| 37.4 | true |
| 38.0 | true |
| 41.9 | true |
| 50.0 | false |

针对Symptom类进行测试，测试其中的contactHistory()和selectSymptom()这两个函数，在测试时利用Mockito进行对象的Mock，设置调用方法后的预期输入和输出。同时，在方法运行时要求用户进行输入，这里利用到了ByteArrayInputStream类，将想要的输入以字符串的形式重定向，来模拟用户输入。

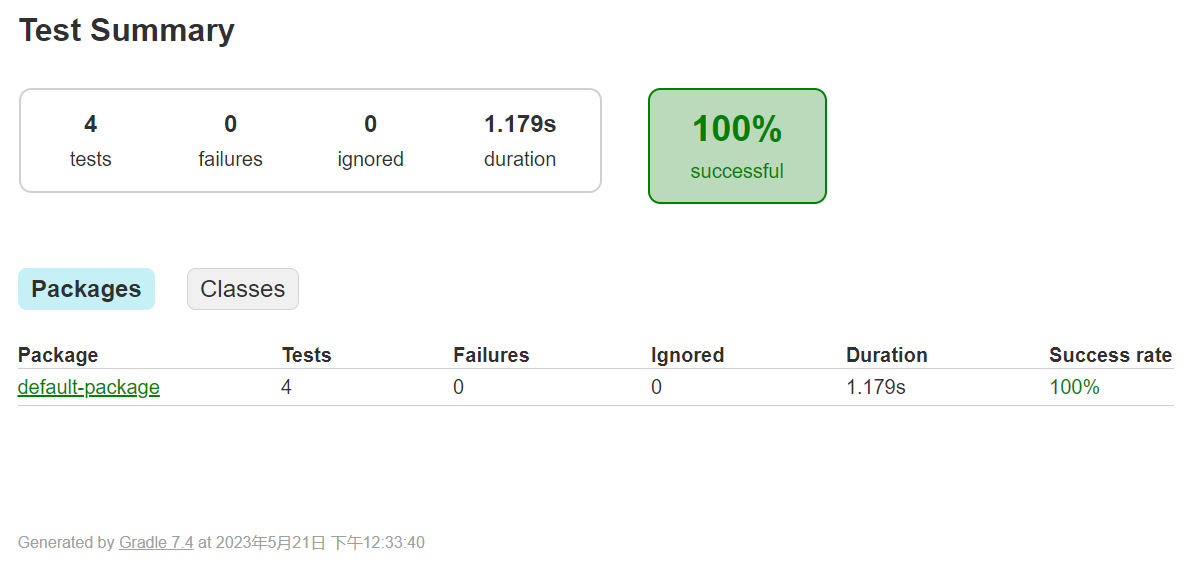
针对Main类进行测试，对函数giveSuggestion()进行测试，将方法中所调用的Detail类和Symptom类进行Mock，对输入流进行重定向，从而进行参数化测试。

表格 7 giveSuggestionTest测试用例

|  |  |
| --- | --- |
| 输入 | 预期输出 |
| (1,1) | 1 |
| (12,1) | 3 |
| (12,5) | 4 |
| (1,5) | 2 |

在编写测试代码时，同时对功能代码进行修改，在类中创建对象时，如果对象创建在类中方法外，则使用Mockito进行Mock时则会报错。故将所有的对象创建移至方法内。

测试代码见附录。



图表 1 测试报告

**参考文献**

[1]. Decision table.(2023/5/21). In Wikipedia, the free encyclopedia. From <https://www.wikiwand.com/en/decision%20table#Overview>

# 附 录

#### 附录一

package org.example;

import java.util.Scanner;

public class Main {

// public Scanner scanner = new Scanner(System.in);

// public Symptom symptom = new Symptom();

// public int symptonflag = 0, contactflag = 0;

public int giveSuggestion(int symptonChosen, int contactChosen) {

Symptom symptom = new Symptom();

int symptonflag = 0, contactflag = 0;

System.out.println("是否有以下症状?");

System.out.println("1.发热、2.咽喉痛、3.咳嗽、4.鼻塞、5.流鼻涕、6.胸闷、");

System.out.println("7.气急、8.呼吸困难、9.全身酸痛、10.乏力、11.腹泻、12.都没有");

System.out.println("如有症状请输入对应数字:");

// int symptonChosen = scanner.nextInt();

int num = 11; //循环次数,最多11次

if (symptonChosen != 12) {

while (num > 0) {

symptom.selectSymptom(symptonChosen);

System.out.println("是否还有其他症状(按0退出)");

// int symptonChosen2 = scanner.nextInt();

int symptonChosen2 = 0;

if (symptonChosen2 == 0)

break;

num--;

}

symptonflag =1;

}

System.out.println("近14天有没有以下情况？");

System.out.println("1.\t有武汉（湖北）等疫区旅游史或居住史\n" +

"2.\t有接触过武汉（湖北）等疫区的人员\n" +

"3.\t有接触过疑似或确诊新型冠状病毒感染者\n" +

"4.\t身边有多人出现发热、乏力、咳嗽、咽痛等\n" +

"5.\t都没有\n");

System.out.println("请输入对应数字:");

// int contactChosen = scanner.nextInt();

int num2 = 4; //循环次数,最多11次

if (contactChosen != 5) {

while (num2 > 0) {

symptom.selectSymptom(contactChosen);

System.out.println("是否还有其他症状(按0退出)");

// int contactChosen2 = scanner.nextInt();

int contactChosen2 = 0;

if (contactChosen2 == 0)

break;

num2--;

}

contactflag = 1;

}

int flag = 0;

if(symptonflag==1&&contactflag==1) {

flag = 1;

System.out.println("建议立刻发热门诊就诊，并且佩戴好口罩做好防护。佩戴医用外科口罩或N95口罩，避免乘坐公共交通工具");

}

else if(symptonflag==1&&contactflag==0) {

flag =2;

System.out.println("结合你提供的病情症状及接触史特点，建议继续在家监测体温，可适当服用治疗感冒的中成药，注意手卫生，多饮水，休息为主。若体温持续不退或体温大于38℃或不适症状加重，及时就诊。");

}

else if (symptonflag==0&&contactflag==1){

flag = 3;

System.out.println("建议你们全家及密切接触者居家隔离至少2周，在家期间建议戴口罩交流，条件允许时，尽量单独居住或居住在通风良好的单人房间。\n" +

"多休息，多饮水，注意手卫生和常用物品卫生消毒。若出现发热及呼吸道症状，需马上前往医院就诊。\n");

}

else if (symptonflag==0&&contactflag==0){

flag = 4;

System.out.println("建议您戴口罩、勤洗手，避免人群聚集，减少不必要外出。");

}

return flag;

}

}

package org.example;

import java.util.Scanner;

public class Symptom {

// public Scanner scanner = new Scanner(System.in);

// public Detail detail;

public Boolean selectSymptom(int i) {

int detailNum = 0;

Detail detail = new Detail();

switch (i) {

case 1: {

System.out.println("目前体温是否大于37.3?(1表示是,0表示否)");

// int temperatureNow = scanner.nextInt();

int temperatureNow = 1;

detailNum = detailNum + detail.symptom(temperatureNow);

System.out.println("最高体温是多少?");

// int temperatureTop = scanner.nextInt();

int temperatureTop = 1;

System.out.println(detail.TemperatureTop(temperatureTop));

System.out.println("发热时有没有发热体温持续不退、体温一直大于38℃情况?(1表示是,0表示否)");

// int temperatureDetail = scanner.nextInt();

int temperatureDetail = 1;

detailNum = detailNum + detail.symptom(temperatureDetail);

System.out.println("发热后有没有吃退热药?(1表示是,0表示否)");

// int feverMedicine = scanner.nextInt();

int feverMedicine = 1;

detailNum = detailNum + detail.symptom(feverMedicine);

break;

}

case 2:

break;

case 3: {

System.out.println("有没有咳痰?(1表示是,0表示否)");

// int cough = scanner.nextInt();

int cough = 1;

detailNum = detailNum + detail.symptom(cough);

break;

}

case 4:

break;

case 5:

break;

case 6: {

System.out.println("有没有胸闷加重?(1表示是,0表示否)");

// int tightChest = scanner.nextInt();

int tightChest = 1;

detailNum = detailNum + detail.symptom(tightChest);

break;

}

case 7: {

System.out.println("有没有气急加重?(1表示是,0表示否)");

// int shortBreath = scanner.nextInt();

int shortBreath = 1;

detailNum = detailNum + detail.symptom(shortBreath);

break;

}

case 8:

break;

case 9:

break;

case 10:

break;

case 11: {

System.out.println("有没有腹泻加重?(1表示是,0表示否)");

// int diarhea = scanner.nextInt();

int diarhea = 1;

detailNum = detailNum + detail.symptom(diarhea);

break;

}

}

return true;

}

public int contactHistory(int i) {

if (i <= 4)

return 0;

else

return 1;

}

}

package org.example;

import java.util.Scanner;

public class Detail {

public float TemperatureTop(float i) {

if (i <= 42.0 && i >= 37.3)

return i;

return 0;

}

public String symptomDuration(int i) { //症状持续14天以内或者14天以上

String a = "";

if (i >= 0 && i <= 14) {

a = "持续14天以内";

System.out.println(a);

} else if (i > 14) {

a = "持续14天以上";

System.out.println(a);

}

return a;

}

public int symptom(int i) {

return i; //1表示有症状,0表示没有症状

}

}

import org.example.Detail;

import org.example.Main;

import org.example.Symptom;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.util.Scanner;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

public class MainTest {

public Scanner scanner = new Scanner(System.in);

// public Main main = new Main();

@ParameterizedTest

@CsvSource({

"1,1,1",

"12,1,3",

"12,5,4",

"1,5,2"

})

public void giveSuggestionTest(int symptonChosen, int contactChosen, int expectation) {

Detail mockedDetail = mock(Detail.class);

Symptom mockedSymptom = mock(Symptom.class);

when(mockedDetail.symptom(1)).thenReturn(1);

when(mockedDetail.TemperatureTop(38.0f)).thenReturn(38.0f);

when(mockedSymptom.selectSymptom(2)).thenReturn(true);

String detailChosen = "1";

ByteArrayInputStream strIn = new ByteArrayInputStream(detailChosen.getBytes());

System.setIn(strIn);

int a;

try {

while ((a = System.in.read()) != -1) {

System.out.print((char) a);

}

} catch (IOException e) {

System.out.println(e.toString());

}

assertEquals(expectation, new Main().giveSuggestion(symptonChosen,contactChosen));

}

}

import org.example.Detail;

import org.example.Symptom;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

public class SymptomTest {

@ParameterizedTest

@CsvSource({

"1",

"2",

"3",

"4",

"5",

"6",

"7",

"8",

"9",

"10",

"11"

})

void selectSymptomTest(int inputValue) {

Boolean functionFlag = new Symptom().selectSymptom(inputValue);

Detail mockedDetail = mock(Detail.class);

when(mockedDetail.symptom(1)).thenReturn(1);

when(mockedDetail.TemperatureTop(38.0f)).thenReturn(38.0f);

String detailChosen = "1";

ByteArrayInputStream strIn = new ByteArrayInputStream(detailChosen.getBytes());

System.setIn(strIn);

int a;

try {

while ((a = System.in.read()) != -1) {

System.out.print((char) a);

}

} catch (IOException e) {

System.out.println(e.toString());

}

assertEquals(true, functionFlag);

}

@ParameterizedTest

@CsvSource({

"0",

"1",

"4",

"3",

"5",

"14",

})

void contactHistoryTest(int a) {

assertEquals(a <= 4, new Symptom().contactHistory(a) == 0);

}

}

import org.example.Detail;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class DetailTest {

@ParameterizedTest

@CsvSource({"37.3", "37.4", "38.0", "41.9", "42.0", "50.0"})

void TemperatureTopTest(float i) {

assertEquals(i <= 42.0 && i >= 37.3, new Detail().TemperatureTop(i) != 0);

}

@ParameterizedTest

@CsvSource({

"0,持续14天以内",

"1,持续14天以内",

"14,持续14天以内",

"13,持续14天以内",

"7,持续14天以内",

"15,持续14天以上",

"16,持续14天以上",

"20,持续14天以上"

})

void symptomDurationTest(int a, String expected) {

assertEquals(expected, new Detail().symptomDuration(a));

}

@ParameterizedTest

@CsvSource({"0, 持续14天以内", "14, 持续14天以内", "15, 持续14天以上", "100, 持续14天以上"})

public void testSymptomDuration(int inputNumber, String expectedResult) {

assertEquals(expectedResult, new Detail().symptomDuration(inputNumber));

}

}

#### 附录二

代码仓库：https://github.com/Real1CM/FinalAssignmentforSoftwareTest.git