Domain Testing

#### 

#### 

Team 16:

Raghav Gupta

Chelsea Jaculina

Bala Nyan Kwaw

Mike Lee

Yihua Li

**March 11th, 2018**

**Professor Badari**

**CMPE 187**

# Table of Contents

[**Table of Contents**](#_fvgztqx5fz61) **1**

[**Overview**](#_8pvx7xuczgkg) **2**

[Introduction](#_ynfxzoxduftu) 2

[Purpose](#_2shjt91jdiwk) 3

[**Part I**](#_2wvt0f6xc8n1) **3**

[Student.java](#_c271ewcu6om2) 4

[StudentTester.java](#_9hc270cjym5n) 6

[**Part II**](#_12gndt7ndpng) **7**

[Test Oracle](#_jlu7nyf28ez) 7

[Environment](#_hxp4hhr1f9w7) 7

[Strategy](#_pde375v0bgqr) 7

[Test Suite](#_gxzblx6galfh) 8

[Inputs](#_x313y9bl1koy) 8

[Outputs](#_pp3ify6te8lq) 8

[Test Cases](#_gy4v0dnanl04) 8

[**Part III**](#_yjk1ob5z6wo) **10**

[Domain Diagram](#_59p96jyyctrz) 10

[**Lessons Learned**](#_zain1bfg61tw) **11**

# Overview

## Introduction

Through the course of completing this assignment we broke it down into 3 individual tasks. The first task that we were asked was to select an implementation unit for testing. The unit that we needed to select had to have a reasonable number of conditionals so that we were able to generate a graphical representation of domains. For our program, we went with a program in which we retrieved from *Java How To Program Early Objects* *10th* *Edition* by Paul and Harvey Deitel. The program demonstrated if...else statements that determined a student’s letter grade based on the student’s average in a course. In the program 2 instance variables are used. A String variable called “name” and a double variable called “average”. The program itself contained 8 predicates. A predicate is defined as a logical function that is evaluated at a decision point.

In the next task, we were asked to draw the boundaries representing domains in the unit under test. Each domain and boundary needed to be labeled in a meaningful way. In addition we also needed to denote the predicates that define each domain. To complete these tasks, we defined testing strategies using the form of an Oracle. The testing strategies included adopting and creating tests cases that would test various types of domain errors.

## **Purpose**

The purpose of this assignment is to use the concept of “Domain Testing” to identify a category of faults, also known as domain errors. In order for us to master the concept of what domain testing is, we needed to have a unit to test that had a sufficient number of conditions. We found that in the Student program that we retrieved from the *Java How To Program Early Objects* *10th* *Edition* by Paul and Harvey Deitel was the perfect unit to test. By obtaining a unit to test that had the right amount of predicates, we were able to perform domain testing by drawing the domain boundaries and denoting all of the predicates that define each domain.

In addition, this assignment allowed us to practice how to write a Test Oracle for a program by adopting and creating test cases for various type of domain errors in which we discussed during class.

# 

# Part I

The predicates that we have identified in our source code are highlighted in yellow.

## 

## *Student.java*

|  |
| --- |
| /\*  \* CMPE 187 - Project #5  \* Domain testing  \* Program - Student record  \* Author: Raghav Gupta  \* Chelsea Jaculina  \* Bala Nyan Kwaw  \* Mike Lee  \* Yihua Li  \*  \* Source:  \* Java How To Program Early Objects   \* 10th Edition   \* By Paul and Harvey Deitel   \* 3/11/2018  \*/  // Student class that stores a student name and average. public class **Student** {  private String name;  private double average;  // constructor initializes instance variables  public **Student**(String name, double average) {  this.name = name;  // validate that average is > 0.0 and <= 100.0; otherwise,  // keep instance variable average's default value (0.0)  if (average > 0 && average <= 100) {  this.average = average; // assign to instance variable  }  }   // sets the Student's name  public void **setName**(String name) {  this.name = name;  }   // retrieves the Student's name  public String **getName**() {  return name;  }   // sets the Student's average  public void **setAverage**(double studentAverage) {  // validate that average is > 0.0 and <= 100.0; otherwise,  // keep instance variable average's current value  if (average > 0 && average <= 100) {  this.average = average; // assign to instance variable  }  }   // retrieves the Student's average  public double **getAverage**() {  return average;  }   // determines and returns the Student's letter grade  public String **getLetterGrade**() {  String letterGrade = ""; // initialized to empty String 52  if (average >= 90.0) {  letterGrade = "A";  } else if (average >= 80.0) {  letterGrade = "B";  } else if (average >= 70.0) {  letterGrade = "C";  } else if (average >= 60.0) {  letterGrade = "D";  } else {  letterGrade = "F";  }  return letterGrade;  } } // end class Student |

## 

## 

## *StudentTester.java*



|  |
| --- |
| /\*  \* CMPE 187 - Project #5  \* Domain testing  \* Program - Student record  \* Author: Raghav Gupta  \* Chelsea Jaculina  \* Bala Nyan Kwaw  \* Mike Lee  \* Yihua Li  \*  \* Source:  \* Java How To Program Early Objects   \* 10th Edition   \* By Paul and Harvey Deitel   \* 3/11/2018  \*/  import java.util.Scanner; // Create and test Student objects. public class **StudentTester** {  public static void **main**(String[] args) {  Scanner scan = new Scanner(System.in);  System.out.println("Please enter your name: ");  String name = scan.nextLine();  System.out.println("Please enter your grade (0-100): ");  double grade = scan.nextDouble();  while (grade < 0 || grade > 100) {  System.out.println("Invalid grade, please re-enter");  grade = scan.nextDouble();  }  Student student = new Student(name, grade);  System.out.println(student.getName() + "'s letter grade is: " + student.getLetterGrade());  } } // end class StudentTest |

# Part II

## Test Oracle

### Environment

The hardware used for programming and testing is a Macbook Pro 13.3 from 2015, with a 2.7GHz Intel Core I5 processor, 8GB 1867MHz DDR4 of memory.

The software used for programming and testing is OS X High Sierra version 10.13.3. The program is written and tested in Java code using the Eclipse version Oxygen, with Java version 8 build 1.8.0\_144. The tests are run in Eclipse IDE.

The program used for creating the Domain flow diagram was draw.io (a plugin by Google) and Code2flow ( <https://code2flow.com/app> ) for verification of the diagram through a third party software.

### Strategy

The strategies that we perform to test the Student program that we obtained *Java How To Program Early Objects* *10th* *Edition* by Paul and Harvey Deitel was to first write a test oracle that includes our inputs and expected outputs. From there, we identified the boundaries of each domain by labeling them in a meaningful way. By using a specifc way of labeling, we were then able to generate a domain diagram for our implementation unit.

## Test Suite

A test suite is a set of tests that tend be in done in units. Each test suite helps validate if the program is working the way that it is expected to. In **Table 1**, shows a table consisting of our test cases that our team has written for our Student average grade program.

### Inputs

The inputs for our Student program will be provided by a user in which they will input a string for their name, and a double for the grade. The user can provide any name they want unless it’s a valid string; but they can provide double values from 0 to 100 because the range of the letter grade is from 0-100

### Outputs

The output for our program will consist of a String that contains the name the user concatenated by the average grade depending on the number value. Below are sample outputs of the Student Program.

|  |  |
| --- | --- |
|  |  |

**Table 1:** Output Table for Student Grade Program

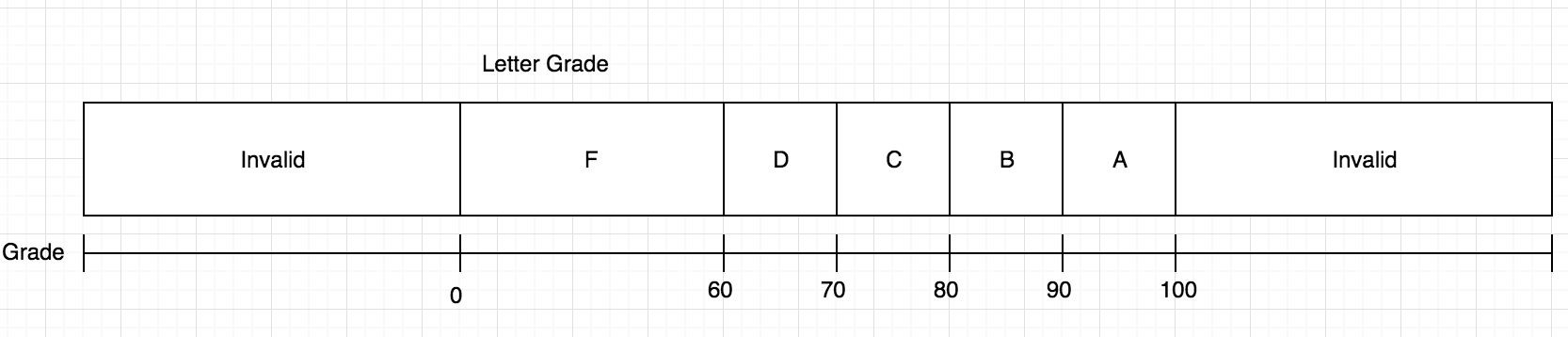
### Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Purpose | Input | Expected Output | Actual Output | Pass/Fail |
| 1 | Test letter grade domain | “John Doe”, -1 | Invalid grade, please re-enter | Invalid grade, please re-enter | Pass |
| 2 | Test letter grade domain | “John Doe”, 0.01 | John Doe’s letter grade is F | John Doe’s letter grade is F | Pass |
| 3 | Test letter grade domain | “John Doe”, 99.99 | John Doe’s letter grade is A | John Doe’s letter grade is A | Pass |
| 4 | Test letter grade domain | “John Doe”, 101 | Invalid grade, please re-enter | Invalid grade, please re-enter | Pass |
| 5 | Test grade “F” | “John Doe”, 0 | John Doe’s letter grade is F | John Doe’s letter grade is F | Pass |
| 5 | Test grade “F” | “John Doe”, 59.99 | John Doe’s letter grade is F | John Doe’s letter grade is F | Pass |
| 6 | Test grade “D” | “John Doe”, 60 | John Doe’s letter grade is D | John Doe’s letter grade is D | Pass |
| 7 | Test grade “D” | “John Doe”, 69.99 | John Doe’s letter grade is D | John Doe’s letter grade is D | Pass |
| 8 | Test grade “C” | “John Doe”, 70 | John Doe’s letter grade is C | John Doe’s letter grade is C | Pass |
| 9 | Test grade “C” | “John Doe”, 79.99 | John Doe’s letter grade is C | John Doe’s letter grade is C | Pass |
| 10 | Test grade “B” | “John Doe”, 80 | John Doe’s letter grade is B | John Doe’s letter grade is B | Pass |
| 10 | Test grade “B” | “John Doe”, 89.99 | John Doe’s letter grade is B | John Doe’s letter grade is B | Pass |
| 11 | Test grade “A” | “John Doe”, 90 | John Doe’s letter grade is A | John Doe’s letter grade is A | Pass |
| 12 | Test grade “A” | “John Doe”, 100 | John Doe’s letter grade is A | John Doe’s letter grade is A | Pass |

**Table 2:** Test Cases for Student Average Grade Program

# Part III

## **Domain Diagram**



**Figure 1:** Domain Diagram for Student Average Grade Program

### Boundaries

An open boundary is said to be open if the points of the boundary do not belong to the domain of interest. In contract, a closed boundary is said to be closed if the points on the boundary are included in the domain of interest.

|  |  |  |
| --- | --- | --- |
| Code Line | Boundaries | Letter Grade |
| 28 (StudentTester) | 100 (**open**) or greater | Invalid |
| 48 | 100 (**closed**) or greater | A |
| 61 | 90 - 100 | A |
| 63 | 80-89 | B |
| 65 | 70-79 | C |
| 67 | 60-69 | D |
| 69 | 0-59 | F |
| 28 (StudentTester) | Negative value to 0 (**open**) | Invalid |
| 28 (StudentTester) | Negative Value to 0 (**closed**) | F |

**Table 3:** Boundary Table for Open and Closed Boundaries

# **Lessons Learned**

There were many lessons that our team learned during this project. The first lesson that our team learned was to be able to identify a unit that contained a sufficient amount of conditionals so that we were able to perform Domain testing on it. The second lession that our team learned was how to draw and label the different boundaries that represented the domains. In addition, we learned we also practiced how to denote predicates that defined each domain and worked on creating test cases in the form of an Oracle.

# 

## 

## 