Week 3 – Test Plan Assignment

Instructions:

- Download and edit this document to fill in your test plans.
- Apply the techniques described in the lesson "Week 3 Equivalence Partitioning, Boundary Value Analysis, and Decision Table Testing" for the following tasks:
    - Testing Task #1: LetterGrade.java.
    - Testing Task #2: HotelBill.java.
    - Testing Task #3: Chapter 3 Programming Project 3.
    - Testing Task #4: Chapter 3 Programming Project 5.
- When editing test cases, fill in the expected I/O (input/output) **prior** to running the program. Fill in the actual I/O and status (Pass or Fail) after running the program.
    - Display the input values in bold. It's ok to wrap a line of output across multiple lines if the table cell is not wide enough to display it on one line. Assume input is entered on the same line as the prompt, even if it is displayed on a separate line in the test case.
    - Refer to the Revel sample runs for exact input/output expectations per project.
- Describe the lessons learned while implementing each assigned programming project. What challenges did you encounter and how did you solve them?

Note: This submission is only for the test plans. The Java code for your programming project solutions should be submitted in the Revel environment for grading.

Testing Task #1 – LetterGrade.java

Click here to download a copy of LetterGrade.java.

Click here to run LetterGrade.java. Create your own editable version by clicking Fork.

A numeric score between 0 and 100 is mapped to a letter grade:
F = 0-59, D = 60-69, C = 70-79, B = 80-89, A = 90-100

0 – 100 is specified as the valid range of scores, thus there are seven equivalence classes:

Invalid	F	D	C	B	A	Invalid
< 0	0 – 59	60 – 69	70 – 79	80 – 89	90 – 100	> 100

The program LetterGrade.java attempts to implement the mapping of numeric score to letter grade, but here is an error in the code.

Fill in the expected I/O for test cases 3-7 by picking a value from the specified equivalence class. Run the program for each test case and fill in the actual I/O and status. One test case should fail.

	Expected I/O	Actual I/O	Status	Equivalence Class
1	Score: **-10** Invalid			< 0
2	Score: **52** F			0 - 59
3				60 - 69
4				70 - 79
5				80 - 89
6				90 - 100
7				> 100

Identify and fix the coding error. Rerun the tests to confirm your solution produces the correct output.

	Expected I/O	Actual I/O	Status	Equivalence Class
1	Score: **-10** Invalid			< 0
2	Score: **52** F			0 - 59
3				60 - 69
4				70 - 79
5				80 - 89
6				90 - 100

7				> 100

Include a screen print of your code solution:

Testing Task #2 – HotelBill.java

Click here to download a copy of HotelBill.java.

Click here to run HotelBill.java. Create your own editable version by clicking Fork.

The base price for a hotel room depends on the bed size (full=$89, queen=$129, king=$159). The hotel gives a $10 senior discount if you're 65 or older, but **only** for rooms with a queen or king bed. There is no senior discount for rooms with a full bed. There are two input conditions that determine the price:

- room type {1 = full, 2 = queen, 3 = king}
- age>= 65 {true, false}
-

The decision table below shows a pricing rule for each of the six combinations of room type and age.

Hotel Room Price Decision Table						
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
CONDITIONS						
room type (1/2/3)	**1**	**1**	**2**	**2**	**3**	**3**
age >= 65 (T/F)	T	F	T	F	T	F
ACTIONS						
price	89	89	119	129	149	159

We can reduce the number of rules to five since rules 1 and 2 show the same price for the same room type. Recall the dash **-** represents "don't care" and can be used to indicate a room type where the action is the same regardless of whether the age condition is true or false.

Complete the simplified table below that merges rules 1 and 2 from the table above into a single rule.

Hotel Room Price Decision Table					
	Rule 1 (merged1&2)	**Rule 2 (old rule 3)**	**Rule 3 (old rule 4)**	**Rule 4 (old rule5)**	**Rule 5 (old rule 6)**
CONDITIONS					
room type (1/2/3)	**1**	**2**	**2**	**3**	**3**
age >= 65 (T/F)	-	T	F	T	F
ACTIONS					
price					

Next, we create 5 test cases, one per rule.

	Expected I/O	Actual I/O	Status	Comment
1	Room 1=full,2=queen,3=king: **1** Age: **78** Price: $89			Rule 1
2	Room 1=full,2=queen,3=king: **2** Age: **70** Price: $119			Rule 2
3	Room 1=full,2=queen,3=king: **2** Age: **37** Price: $129			Rule 3
4	Room 1=full,2=queen,3=king: **3** Age: **68** Price: $149			Rule 4
5	Room 1=full,2=queen,3=king: **3** Age: **28** Price: $159			Rule 5

The HotelBill class attempts to implement the required functionality. However, there is an error involving logical operator precedence.

Run the program and fill out the actual output and status for each test case. The last test should fail to produce the correct output.

Recall && has higher precedence than ||. The expression a && b || c is equivalent to (a && b) || c, which makes the entire expression true whenever c is true.

Identify and fix the error and rerun the tests to confirm your solution produces the correct output.

Include a screen print of your code solution:

Testing Task #3 – Chapter 3 Project 3 Test Plan

NOTE: The textbook instructions for programming project 3 do not describe what to do if the user enters an invalid month such as 0 or 13. The CodeGrade tests also don't check for that error. You can assume the user enters a valid month between 1 and 12 when you implement the code for project 3, i.e. don't worry about boundary value analysis.

You will apply equivalence class partitioning and decision table testing for programming project 3.

Complete the table below by providing 2 sample years for each of the 4 equivalence classes.
A year that does not qualify as a leap year is known as a common year.

Year Equivalence Classes			
	Leap Year	**Common year**	**Common Year**
Divisible by 4 and not divisible by 100	Divisible by 400	Not divisible by 4	Divisible by 100 and not divisible by 400
1. 2.	1. 2.	1. 2.	1. 2.

Fill in the action row in the decision table below. Recall the dash **-** represents "don't care" and indicates a month where the action is the same for both leap and common years.

Days in a Month Decision Table													
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**	**Rule 7**	**Rule 8**	**Rule 9**	**Rule 10**	**Rule 11**	**Rule 12**	**Rule 13**
CONDITIONS													
Month (1-12)	**1**	**2**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Leap Year (T/F)	-	T	F	-	-	-	-	-	-	-	-	-	-
ACTIONS													
Days	31												

Edit the table below to define test cases for the first 5 rules. Select a unique year from the various equivalence classes for each test case. Run your Project 3 solution for each test case and fill in the actual I/O and status. Try to fix errors identified by the tests. If you are unable to get a test case to pass, mention it in the lessons learned.

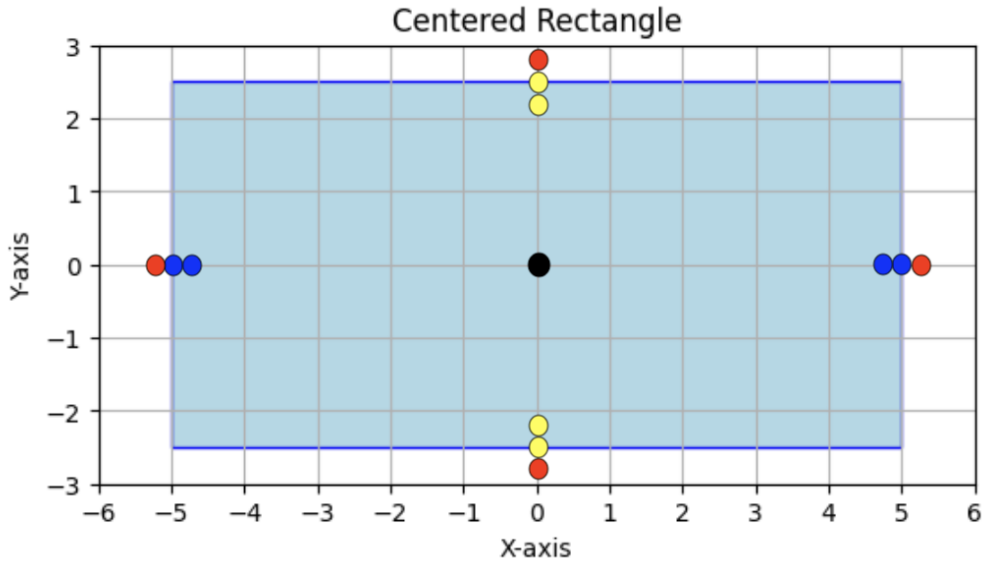
	Expected I/O	Actual I/O	Status	Comment
1	Enter a month in the year (e.g., 1 for Jan): **1** Enter a year: **2000** January 2000 has 31 days			Rule 1
2				Rule 2
3				Rule 3
4				Rule 4

5				Rule 5

Lessons learned while implementing programming project 3:

Testing Task #4– Chapter 2 Project 5 Test Plan

Programming project 5 reads in a point (x, y) and prints whether the point is within the rectangle centered at (0,0) having width 10 and height 5.



Complete the following table based on the inclusive range of x values [-5.0, 5.0].

X Boundary Value Analysis (BVA)		
Outside range min-0.1	x in range [-5.0, 5.0] min, min+0.1, nominal, max-0.1, max	Outside range max + 0.1

Complete the following table based on the inclusive range of y values [-2.5, 2.5].

Y Boundary Value Analysis (BVA)		
Outside range min-0.1	y in range [-2.5, 2.5] min, min+0.1, nominal, max-0.1, max	Outside range max + 0.1

N variables require 6N + 1 test cases for VBA.

- 4N+1 cases on or within the boundaries.
- 2N cases just outside a boundary.

Given 2 variables (x, y), we need 6*2+1 = 13 test cases.

- 9 test cases for points inside the rectangle
- 4 test cases for points outside the rectangle.

Edit the table below to define test cases 1-13 based on results of the 2-variable BVA. Test 1-9 are the various points inside the rectangle (black, blue, yellow), while tests 10-13 are the 4 red points outside the rectangle. Nominal is abbreviated “nom”.

Run your Project 5 solution for each test case and fill in the actual I/O and status. Try to fix errors identified by the tests. If you are unable to get a test case to pass, mention it in the lessons learned.

	Expected I/O	Actual I/O	Status	(x, y)
1	Enter a point with two coordinates: **0 0** Point (0.0, 0.0) is in the rectangle			(nom, nom)
2	Enter a point with two coordinates: **-5 0** Point (-5.0, 0.0) is in the rectangle			(min, nom)
3				(min+0.1, nom)
4				(max-0.1, nom)
5				(max, nom)
6				(nom, min)
7				(nom, min+1)
8				(nom, max-0.1)
9				(nom, max)
10	Enter a point with two coordinates: **-5.1 0** Point (-5.1, 0.0) is not in the rectangle			(min–0.1, nom)
11				(max+ 0.1, nom)
12				(nom, min-0.1)
13				(nom, max+0.1)

Lessons learned while implementing programming project 5: