Homework Assignment 2 (100 Points)
CSE 464, Fall 2020
CIDSE, Arizona State University
Due: By Saturday Sep19th 11:59 pm

Please see the submission instructions given in the last page
Note: This is an individual homework, DO NOT collaborate

Introduction: This assignment helps you to apply equivalent partitioning and develop Junit test cases

1. **[25 Points]** One of functionalities of a password generator/verifier is to determine if a password user selected is strong enough.

   a) Two essential password rules:

   Following two rules are bare minimal that you should follow while creating a password.

   Rule 1 – Password Length: Stick with passwords that are at least 8 characters in length. The more character in the passwords is better, as the time taken to crack the password by an attacker will be longer. 10 characters or longer are better.

   Rule 2 – Password Complexity: Should contain at least one character from each of the following group. At least 4 characters in your passwords should be each one of the following.

   1. Lower case alphabets
   2. Upper case alphabets
   3. Numbers
   4. Special Characters

   We can call the above two rules combined as **"8 4 Rule"** (Eight Four Rule):

   - **8** = 8 characters minimum length
   - **4** = 1 lower case + 1 upper case + 1 number + 1 special character.

(Source: http://www.thegeekstuff.com/2008/06/the-ultimate-guide-for-creating-strongpasswords/)

Suppose you are using equivalent partitioning technique for testing if the password user selected is acceptable.

a) Identify equivalent partitions based on the above description

- **Answer:**

  There are various different ways to come up with equivalent partitioning for the description of this problem. One of the ways to approach this problem is by considering and applying the process of elimination. First step to take is to think of what kind of passwords are valid and what kind are not. Afterwards, one can sub-divide the valid partitioning to equivalent partitions of different possible combinations so that a variety of different test cases/ scenarios can be tested.

  EP1: Password < 8 characters → **Invalid**
  EP2: Password is >= 8 characters, but, doesn't contain at least one of the following 4 characters required: 1 lower case + 1 upper case + 1 number + 1 special character → **Invalid**

  EP3: Password is >= 8 characters and does contain at least one of the following 4 characters required: 1 lower case + 1 upper case + 1 number + 1 special character → **Valid!**

b) Develop test cases in a tabular format given below based on your equivalent partitions above. Use Weak Normal form

Test case #	Partition Tested	Input(s)	Expected output
WN 1	EP3	@Word12.12	Valid password!

2.[50 Points] Equivalent Partitions: Credit Card Verification Problem

Consider the following program that checks if a credit number is a valid card number.

The last digit of a credit card number is the check digit, which protects againts transaction errors. The following method is used to veryfy credit card numbers. Follwing steps explains the algorithm in determining if a credit card number is a valid card.

- Starting from the right most digit, form the sum of every other digit. For example, if the credit card is number is 3125145643589795 then you form the sum 5+7+8+3+6+4+5+1 = 39

- Double each digit that we have not included in the preceding step. Add all digits of resulting numbers. For example, with the number given above, doubling the digits starting with next to last one, yields 18, 18, 10, 8, 10, 2, 4,6. Adding all digits in these values yield 1+8+1+8+1+0+8+1+0+2+4+6 = 40

- Add sum of the two preceding steps. If the last digit of the result is zero, then the number is valid number

The program asks the user 16 digit credit card number and the printout if the credit card is valid or invalid card

a) [20 Pts] How do you test this application using EP technique? Identify your EPs Write test cases based on your EPs (use strong robust classification).

Answer:

Using the Equivalent Partitioning technique, this application can be tested by considering process of elimination. For example, firstly, one can think of valid and invalid credit card numbers that are possible for the application.

EP1: credit card < 16 digits → **Invalid**
EP2: credit card > 16 digits → **Invalid**
EP3: checking length of credit card and making sure both sum1 and sum2 add up to 16 and when and when the sums(sum1 and sum2) get calculated individually, they pick on every other number **(this is where the error is found in the program) and the algorithm total sum is correct**
EP4: credit card = 16 but is made up of other input such as negative numbers, letters, special characters. . →**Invalid**
EP5: credit card = 16 digits; however, the end result of the sum of the two steps for the algorithm doesn't result in 0 → **Invalid**

EP6: credit card = 16 positive numbers and the end result of the sum of the two steps for the algorithm does result in 0 → **Valid!**

Test case #	Partition Tested	Input(s)	Expected output
1	EP 1	16578486	Invalid Card
2	EP 2	3125145643589795869	Invalid Card
3	EP3	4444444444444444 And 9999999999999999	Invalid card
4	EP 4	- 3125145643589795	Invalid Card
5	EP 5	4024007164160267	Invalid Card

6	EP 6	4024007164160268	Valid Card

b) [30 Pts] Develop Junit test cases based on your test cased listed above and test the java program given. This java program has two defects. Identify defects in this program (If your test cases can catch them, you have developed good set of test cases!). You can assume that the user will enter a 16 digit number. No need to consider other types of inputs such as strings, characters …etc.

```
//**********1st ERROR***************
// get the card length
// length = (int) (Math.log10(cardNumber) + 1);
//******************************
```

This is an error/bug because after getting the length of the credit card number using this, in some cases instead of the length resulting in 16 numbers, it would give 17, therefore a length check is necessary for this section for instance: if a credit card has the following 16 digits: 1313131313131313L it will through an error even though the length is 16. Another example is if a credit has the following 16 digits: 9999999999999999L its length will be counted as 17 instead of 16

```
//**********************2nd ERROR****************
// starting from the right most digit add every other digit to sum 1
//change** i>=2 in for loop to i>=0

for(i= CARD_LENGHT - 1;  i >=2 ; i= i -2)
        {
          sum1 =  sum1 + digitArray[i];
        }
```

In the program, this part is used to calculate the first sum of the algorithm, however, instead of i starting from the right most digit and adding every other digit at i>=0 it starts at i>=2,
For instance, if a credit card that has the following digits: 4444444444444444L, the total sum(sum1 and sum2) of the algorithm should add up to 96, however because in the given program i>=2 instead of i>=0, the total sum of the algorithm results in 92.

3.) Equivalent Partitions: Finding roots of a quadratic equation

a) **[25 Pts]** Study the Roots.java and usingRoots.java programs given. Roots.java program finds the roots of a quadratic equation in the form ax^2 + bx + c=0. If you are to design test cases to test the Roots.java program using equivalence partitioning:

i) Identify equivalent classes to test the Roots program. Briefly describe your strategy first.

Answer:

An approach to finding equivalent classes for this problem would be to consider process of elimination. To explain, one can find what is valid and invalid for a quadratic equation in order for it to be considered a quadratic equation. The rule for a quadratic equation to be considered real is that its discriminant must be >= 0 in order for it to have real numbers/roots. The discriminant is what is under the square root of the formula **($\sqrt{b^2 - 4ac}$),** after solving what is under the square root one can determine if a quadratic equation is real(real-numbers) or non-real(has imaginary-numbers). If the discriminant is < 0 then, a solver of a quadratic equation would have imaginary numbers; therefore, it would be considered a solution that has non-real numbers.

EP1: **if a = 0, and one tries to apply the quadratic formula, this will be invalid because**

(0)x^2 = 0: this gets rid of an equation being quadratic for instance: (0)x^2 +4x-3 = 4x-3 and it can't be a quadratic. (not quadratic)

EP2: **$b^2 - 4ac > 0$ = There exists two real roots(real-numbers)**

EP3: **$b^2 - 4ac = 0$ There exists one real root(real-numbers)**

EP 4: **$b^2 - 4ac < 0$ = no real root(s) (Imaginary numbers/non-real numbers)**

Design test cases based on your equivalence classes. Use strong robust classification.

Test case #	Partition Tested	Input(s) a, b, c	Expected output
1	EP1	0, 4, 3	Not quadratic(0 root(s))
2	EP2	2, -11, 5	There exists two real roots(real numbers)
3	EP3	-4, 12, -9	There exists one real root
4	EP4	2, -4, 4	No real root(s): Imaginary numbers
5	EP5	2, -10, 2	There exists two real roots(real numbers)

6	EP6	10, 12, 0	There exists two real roots(real numbers)
7	EP7	10,0,0	There exists one real root
8	EP8	10, 0, 1	No real root(s): Imaginary numbers
9	EP9	10, 1, 1	No real root(s): Imaginary numbers
10	EP10	1, 1, 1	No real root(s): Imaginary numbers
11	EP11	1, 0, 1	No real root(s): Imaginary numbers
12	EP12	1, 0, 0	There exists one real root
13	EP13	12, 12, 12	No real root(s): Imaginary numbers
14	EP14	100, 100, 100	No real root(s): Imaginary numbers
15	EP15	100, 99, 100	No real root(s): Imaginary numbers
16	EP16	12, 6, 2	No real root(s): Imaginary numbers
17	EP17	50, 99, 50	No real root(s): Imaginary numbers
18	EP18	60, 0, 60	No real root(s): Imaginary numbers

Submission Instructions:

Things to submit:

1. Answers to question 1 part a and b in PDF format. Handwritten is okay if readable and clear.

For question 2:

a. PDF document containing answers to part a
b. Junit test program (java file) for part b and a PDF document explaining bugs in the program
c. Corrected java program for part c

For question 3:

a. PDF document containing equivalent partitions and test cases for part (a) Junit test program (java file) for part b

How to submit:

Create a zip file that contain all the submission items listed above and submit online to the blackboard.

NO LATE SUBMISSIONS WILL BE ACCEPTED