Assignment #2

Daylan Quinn

Turn in your work as **a single Word or PDF file**.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Problem 1** | | | | **Problem 2** | | | | | | **Total** |
| **1.1** | **1.2** | **1.3** | **1.4** | **2.1** | **2.2** | **2.3** | **2.4** | **2.5** | **2.6** |
| 15 pts | 10 pts | 7 pts | 6 pts | 15 pts | 15 pts | 10 pts | 10 pts | 6 pts | 6 pts | 100 pts |

Download isbnAssignment.zip and unzip it. The source code consists of seven packages: correct and mutantX(X=1..6). Each package contains a class ISBN10.java. In the “correct” package, ISBN10.java has a correct version of the method “boolean isValidISBN(String isbn)”. In each mutantX package, ISBN10.java contains a modified version of isValidISBN. **Do not make any change to the code.** **If you choose to use a different language, it is your responsibility to translate all the code correctly.**

The isValidISBN method returns true if a given string *isbn* is a valid ISBN-10 number, i.e., meets the following conditions:

(a) the length is 10,

(b) the first 9 characters are all digits,

(c) the 10-th character (check digit) is a digit or ‘X’. ‘X’ means 10 (suppose ‘x’ is invalid).

(d) the weighted sum of the 10 digits, defined below, is a multiple of 11.

where *xi* represents the *i*-th digit. For example, the weighted sum of “0306406152” is:

(0 \*10) + (3\*9) + (0\*8) + (6\*7) + (4\*6) + (0\*5) + (6\*4) + (1\*3) + (5\*2) + (2\*1)

= 0 + 27 + 0 + 42 + 24 + 0 + 24 +3 +10 +2

= 132 = 12 \*11

You may create your test cases by using and modifying known ISBN-10 numbers. Some can be found at: https://en.wikipedia.org/wiki/International\_Standard\_Book\_Number

1. **Decision Table** 
   1. Complete the following decision table, where T/F/DC represent True /False/Don’t Care, respectively. Each entry in the “length=10” column should be either T or F. Each entry of “The 1-9 characters are all digits” or “The weighted sum is a multiple of 11” should be T, F, or DC. The three entries, “is digit”, “is X”, and “is not digit or X”, in each row are mutually exclusive – no more than one of them should be T. (**15 points**)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Variant** | **Condition** | | | | | | **Action**  (valid or invalid ISBN-10 number) |
| length=10 | The 1-9 characters are all digits | The 10th character | | | The weighted sum is a multiple of 11 |
| is digit | is ‘X’ | is not digit or ‘X’ |
| 1 | F | DC | DC | DC | DC | DC | INVALID |
| 2 | T | F | DC | DC | DC | DC | INVALID |
| 3 | T | T | T | F | F | F | INVALID |
| 4 | T | T | F | T | F | F | INVALID |
| 5 | T | T | F | F | T | DC | INVALID |
| 6 | T | T | T | F | F | T | VALID |
| 7 | T | T | F | T | F | T | VALID |

No. combinations = 2\*2\*2\*3 = 24

DC = 2 options combined

Total variants = 24 – 17 = 7 variants

* 1. Create one test case for each variant in the completed decision table in Problem 1.1. List all test cases in the following table. (**10 points**)

|  |  |  |
| --- | --- | --- |
| **Test No** | **Test Input (isbn)** | **Oracle Value** |
| 1 | 030640615 | False |
| 2 | X306406152 | False |
| 3 | 0306406153 | False |
| 4 | 000002002X | False |
| 5 | 000002001x | False |
| 6 | 0306406152 | True |
| 7 | 000002001X | True |

* 1. Write a JUnit class ISBN10DecisionTableTest.java in the “correct” package to implement all the test cases in Problem 1.2. Run and debug this class in the “correct” package **until there is no failure**. If there is a failure, then something is wrong with your tests and you must fix it before moving on to the next problem. Include in this document the source code of your ISBN10DecisitionTableTest.java and a screenshot of your test execution that shows no failure. (**7 points**)

##################################################################################################

package correct;

import static org.junit.Assert.\*;

import org.junit.Test;

public class ISBN10DecisionTableTest {

@Test

public void test1() {

assertFalse(ISBN10.isValidISBN("030640615"));

}

@Test

public void test2() {

assertFalse(ISBN10.isValidISBN("X306406152"));

}

@Test

public void test3() {

assertFalse(ISBN10.isValidISBN("0306406153"));

}

@Test

public void test4() {

assertFalse(ISBN10.isValidISBN("000002002X"));

}

@Test

public void test5() {

assertFalse(ISBN10.isValidISBN("000002001x"));

}

@Test

public void test6() {

assertTrue(ISBN10.isValidISBN("0306406152"));

}

@Test

public void test7() {

assertTrue(ISBN10.isValidISBN("000002001X"));

}

}

##################################################################################################

A picture containing text, electronics, monitor, computer

Description automatically generated

* 1. Copy your ISBN10DecisionTableTest.java in Problem1.3 to each mutantX (X=1..6) package. Make sure the package statement in ISBN10DecisionTableTest.java is correct so that you can test ISBN10.java in the same package. Report your test execution results in the following table and include a screenshot of the test execution for each mutantX package. (**6 points**)

|  |  |  |
| --- | --- | --- |
| **Mutant Version** | **Test failure: Yes or No?** | **If Yes, list the tests that failed** |
| 1 | Yes | 1 |
| 2 | Yes | 6, 7 |
| 3 | Yes | 2 |
| 4 | No |  |
| 5 | Yes | 6, 7 |
| 6 | Yes | 7 |

Mutant 4 could have been failed if I had used a 9 in my isbn tests, but with blackbox testing I did not know to use a 9 or not.

A picture containing text, electronics, computer

Description automatically generated

A picture containing text, computer, electronics, stereo

Description automatically generated

A picture containing text, monitor, electronics, computer

Description automatically generated

A picture containing text, electronics, computer

Description automatically generated

A picture containing text, electronics, computer, stereo

Description automatically generated

A picture containing text, monitor, computer, electronics

Description automatically generated

1. **Equivalence Partitioning and Boundary Value Analysis** 
   1. Complete the following table of equivalence classes. (**15 points**)

|  |  |  |
| --- | --- | --- |
| **Condition** | **Valid classes** | **Invalid classes** |
| Length=10 | 10,  1234567890 | 9, 11  12345678, 12345678910 |
| Each of the first 1-9 characters is a digit | 123456789,  987654321 | 12345678X9, X123456789 |
| The 10th character is a digit or ‘X’ | 1234567890, 123456789X | 123456789!, 123456789x |
| The weighted sum is a multiple of 11 | 0306406152,  000002001X | 0306406153, 0306406151 |

* 1. Combine the valid and invalid classes of all conditions in Problem 2.1. (**15 points**)

Each combination involves one valid or invalid class of each condition in the above equivalent class table. When a combination contains only valid classes of different conditions, it may cover as many uncovered valid classes as possible. When it contains an invalid class, however, it should cover only one invalid class and use valid classes for other conditions. Here is an example combination (called test input requirement), which covers a valid class for each condition:

Length=10,

Each of the first 1-9 characters is a digit,

The 10th character is a digit,

The weighted sum is a multiple of 11

Complete the following table such that each valid and invalid class should appear at least once.

|  |  |  |
| --- | --- | --- |
| **#** | **Test input requirement** | **Expected result** |
| 1 | Length=10,  Each of the first 1-9 characters is a digit,  The 10th character is a digit,  The weighted sum is a multiple of 11 | Valid |
| 2 | Length=10,  Each of the first 1-9 characters is a digit,  The 10th character is a ‘X’,  The weighted sum is a multiple of 11 | Valid |
| 3 | Length!=10,  Each of the first 1-9 characters is a digit,  The 10th character is a ‘X’,  The weighted sum is a multiple of 11 | Invalid |
| 4 | Length=10,  At least one of the first 1-9 characters is NOT a digit,  The 10th character is a digit,  The weighted sum is a multiple of 11 | Invalid |
| 5 | Length=10,  Each of the first 1-9 characters is a digit,  The 10th character is NOT a digit OR ‘X’,  The weighted sum is a multiple of 11 | Invalid |
| 6 | Length=10,  Each of the first 1-9 characters is a digit,  The 10th character is a digit,  The weighted sum is NOT a multiple of 11 | Invalid |

* 1. Complete the following table of boundary values according to the valid and invalid classes in Problem 2.1. (**10 points**)

|  |  |  |
| --- | --- | --- |
| **Condition** | **Boundary values from valid classes** | **Boundary values from invalid classes** |
| Length=10 | L=10 | L<10, L>10 |
| Each of the first 1-9 characters is a digit | char[1-9] = digits  ascii\_val(48) <= Char[1-9] <= ascii\_val(57) | char[1-9] != digits  char[1-9] > ascii\_val(57), char[1-9] < ascii\_val(48) |
| The 10th character is a digit or ‘X’ | char[10] = digit, char[10] = ‘X’  ascii\_val(48) <= Char[10] <= ascii\_val(57)  or  char[10] = ascii\_val(88) | char[10] != digit, char[10] != ‘X’  char[10] < ascii\_val(48), char[10] > ascii\_val(57)  or  char[10] > 88, char[10] <88 |

The valid and invalid classes of “The weighted sum is a multiple of 11” are not listed here. You should consider them when creating test cases in the next problem.

* 1. Complete the following table of test cases based on the test input requirements in Problem 2.2 and the boundary values in Problem 2.3. There should be at least one test case for each test input requirement in Problem 2.2. The test cases should cover each boundary value at least once (**10 points**)

Note: when creating a test case of invalid ISBN-10 number to cover the condition of invalid length or invalid character, you don’t need to ensure that the weighted sum of the test input is a multiple of 11 (it would be great if you can do it, but this is not required).

|  |  |  |  |
| --- | --- | --- | --- |
| **Test case #** | **Test input requirement # in Problem 2.2** | **Test Input (isbn)** | **Expected result** |
| 1 | 1 | 0306406152 | Valid |
| 2 | 2 | 000002001X | Valid |
| 3 | 3 | 12345678 | Invalid |
| 4 | 3 | 12345678910 | Invalid |
| 5 | 4 | 12345678X9 | Invalid |
| 6 | 4 | X123456789 | Invalid |
| 7 | 5 | 123456789! | Invalid |
| 8 | 5 | 123456789x | Invalid |
| 9 | 6 | 0306406153 | Invalid |
| 10 | 6 | 0306406151 | Invalid |

* 1. Write a JUnit class ISBN10PartitioningTest.java in the “correct” package to implement all the test cases in Problem 2.4. Run and debug this class in the “correct” package **until there is no failure.** If there is a failure, then something is wrong with your tests and you must fix it before moving on to the next problem. Include in this document the source code of ISBN10PartitioningTest.java and a screenshot of your test execution that shows no test failure. (**6 points**)

##################################################################################################

package correct;

import static org.junit.Assert.\*;

import org.junit.Test;

public class ISBN10PartitioningTest {

@Test

public void test1() {

assertTrue(ISBN10.isValidISBN("0306406152"));

}

@Test

public void test2() {

assertTrue(ISBN10.isValidISBN("000002001X"));

}

@Test

public void test3() {

assertFalse(ISBN10.isValidISBN("12345678"));

}

@Test

public void test4() {

assertFalse(ISBN10.isValidISBN("12345678910"));

}

@Test

public void test5() {

assertFalse(ISBN10.isValidISBN("12345678X9"));

}

@Test

public void test6() {

assertFalse(ISBN10.isValidISBN("X123456789"));

}

@Test

public void test7() {

assertFalse(ISBN10.isValidISBN("123456789!"));

}

@Test

public void test8() {

assertFalse(ISBN10.isValidISBN("123456789x"));

}

@Test

public void test9() {

assertFalse(ISBN10.isValidISBN("0306406153"));

}

@Test

public void test10() {

assertFalse(ISBN10.isValidISBN("0306406151"));

}

}

##################################################################################################

A picture containing text, computer, electronics, screenshot

Description automatically generated

* 1. Copy your ISBN10PartitioningTest.java in Problem 2.5 to each mutantX (X=1, 2, …, 6) package. Make sure the package statement in ISBN10PartitioningTest.java is correct so that you can test ISBN10.java in the same package. Report your test execution results in the table below and include in a screenshot of the test execution for each mutantX package. (**6 points**)

|  |  |  |
| --- | --- | --- |
| **Mutant Version** | **Test failure: Yes or No?** | **If Yes, list the tests that failed** |
| 1 | Yes | 3 |
| 2 | Yes | 1, 2 |
| 3 | Yes | 6 |
| 4 | No |  |
| 5 | Yes | 1, 2 |
| 6 | Yes | 2 |

Again, mutant 4 could have failed if I had used a 9 in my isbn tests, but with blackbox testing I did not know to use a 9 or not.

A picture containing text, computer, electronics

Description automatically generated

A picture containing text, monitor, computer, electronics

Description automatically generated

A picture containing text, monitor, computer, electronics

Description automatically generated

A picture containing text, computer, electronics, computer

Description automatically generated

A picture containing text, monitor, computer, electronics

Description automatically generated

A picture containing text, computer, monitor, electronics

Description automatically generated

In this assignment I could have adjusted some tests after the fact to allow mutant 4 to show as a failed test, but I decided not to in order to show that I understand that there are possible faults in tests that are created. I could have simply used an isbn that used a 9 in it, however, the testing requirements did not mandate that so I left it as is.