THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
SE4452A
Software Verification and Validation
Fall 2016

Assignments # 2
Due on Nov. 25, 2016

1. **AccountStatus**

   a) **Equivalence Class**
   Not applicable - covered by decision table

   b) **Boundary Values**

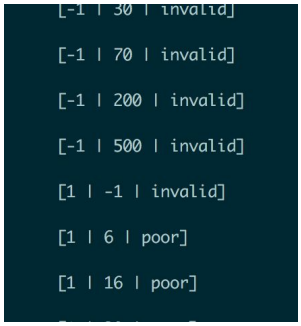
accountFactor	AccountStatus
accountFactor <0	invalid
accountFactor >=0 and accountFactor <=700	poor
accountFactor > 700 and accountFactor <=3000	fair
accountFactor > 3000 and accountFactor <=10k	good
Above 10k	very good

It is very clear from the above chart that the boundaries lie on the values listed explicitly - for example, 0, 700, 3000, and 10k. Some combination which yields these values must be tested - see the decision table testing for more details.

c) decision table

I automatically generated the decision table for the dependant values ageFactor, and balanceFactor -- see classGenerator.js for details. Though I'd like to note that I think this function could simply have been better written to avoid the need for such testing. Why not make it simply take these 2 things as arguments?

In any case we can see the output from the generator looks something like this:



We can see that there is a rule where ageFactor (column 1) is -1, and balanceFactor(column 2) is any positive number -- and so on in this fashion, I wrote our rules based on the table I generated. I would never actually use this method but this example is really contrived and repetitive so I figured this was faster.

2. **getAgeFactor**

    a) **Equivalence Classes**

accountAge	ageFactor
Age <0 or Age >100	-1
Age =0	1
Age >0 and Age<2	5
Age >=2 and Age<5	10
Age >=5 and Age<10	20
Age >=10 and Age<=100	50

The equivalence classes are clearly identified by this chart. There is one class for each row, ignoring the boundaries.

    b) **Boundary Values**

Just as above, the chart clearly lays out the boundaries. The boundary values are the numbers given explicitly on either end of the accountAge, 1, 100, 2, 5, 10, 100, and undefined all need to be tested explicitly.

    c) **Decision Table**

Not necessary for this function, boundary value and equivalence class handle it.

3. **getBalanceFactor**

    a) **Equivalence Classes**

accountBalance	balanceFactor
Balance < -100 or Balance> 1B	-1
Balance >= -100 and Balance < =0	6
Balance > 0 and Balance < 1000	16
Balance >= 1k and Balance < 50k	30
Balance >= 50k and Balance < 100k	70
Balance >= 100k and Balance <= 1M	200
Balance >1M	500

This function is very similar to the above function. The chart lays out the equivalence classes, 1 per row.

b) Boundary Values

Again, the chart defines the boundaries, -100, 1B, 0, 1000, 50k, 100k, 1M, undefined.

c) Decision Table

Not necessary for this function, boundary value and equivalence class handle it.

4. creditStatus

a) Equivalence Classes

Not necessary - decision table supersedes this.

b) Boundary Values

creditStatus	creditCheckMode	creditScore
bad	restricted	less than 750
	default	less than 700
good	restricted	more or equal to 750
	default	more or equal to 700

The boundaries are 750 and 700 exactly - with both creditStatuses. This is obvious from the chart above.

c) Decision Table

The chart above actually *is* a decision table. There is really no need to make another one, I simply will write tests for all combinations listed above

5. productStatus

a) Equivalence Classes

threshold	productQuantity	status
0	> 0	sold-out

0	< 0	sold-out
0	==0	sold-out
N (st N > 0)	< N	available
N	> N	limited
N	== N	available

b) Boundary Values

Boundaries that need to be checked are when items in the store equal to the threshold.

c) Decision Table

Not necessary

6. orderHandling

a) Equivalence Classes

b) Boundary Values

c) Decision Table

client, product, store, storeThreshold, creditCheckMode

accountStatus	creditStatus	productStatus	orderStatus
very good	-	-	accepted
good	good	-	accepted
poor	good	available	accepted
fair	good	available	accepted
fair	good	limited	pending
fair	good	sold-out	pending
poor	good	limited	pending
good	bad	-	underReview

fair	bad	available	underReview
fair	bad	limited	rejected
fair	bad	sold-out	rejected
poor	good	sold-out	rejected
poor	bad	-	rejected

This should pretty much cover all the test cases - I just have to write these out as tests.