Prof. Lichter
Andreas Steffens, Konrad Fögen

OOSC
WS 2016/2017

Submission 3

oosc@swc.rwth-aachen.de

Issued: 21.05.2019 Submission: 04.06.2019 Discussion: 07.06.2019

Submitted by Group 09

Ulfet CETIN	391819
Saud KHAN	392365
Samuel ROY	391822
Charulekha, Besta Venkateswara RAO	391844
Deepak SATEESH	391813

(ordered on lastname basis)

3.1. Task: Boundary-Value Analysis

a) Please use the Boundary-Value Analysis to identify the input space.
Please name all necessary boundary values.
Create and implement a test suite with Junit5 using Boundary Value Testing.

Boundary values are given below for each variable:

Variable Name	min-	min	min+	nom	max-	max	max+
local-part	0	1	2	33	63	64	65
domain-part	0	1	2	65	127	128	129
top-part	0	1	2	32	62	63	64

* in case a during a nominal value calculation that an odd integer division is encountered (e.g. (128+1) / 2), we ceil the value to the nearest integer.

How values are determined:

- range for length of each part of an e-mail is given in the assignment text,
  - local part (local-part) $\Rightarrow$ [1,64]
  - domain part (domain-part) $\Rightarrow$ [1,128]
  - top-level domain (tld) $\Rightarrow$ [1,63]
- for min- and max-, I just decremented the respective value by 1,
- for min+ and max+, I just incremented the respective value by 1,
- for nominal value, I summed max and min together, and then divided the result by 2

Simple Boundary Value Testing:

Test Suite	Local Length	Domain Length	Top Length	email Validator	Actual Result
[Simple]	33	32	32	True	True
[Simple]	1	65	32	True	True
[Simple]	2	65	32	True	True
[Simple]	63	65	32	True	True
[Simple]	64	65	32	True	True
[Simple]	33	1	32	True	True
[Simple]	33	2	32	True	True
[Simple]	33	127	32	True	True
[Simple]	33	128	32	True	True
[Simple]	33	32	1	True	True
[Simple]	33	32	2	True	True
[Simple]	33	32	62	True	True
[Simple]	33	32	63	True	True

We have 3 variables, and each has 5 values {min, min+, nom, max-, max} for simple BVT.
How to run the test:

- for each variable in {local, domain, top}
  - for each {min, min+, max-, max} value of the variable
    - generate a test case with:
      (this variable's value from for,
      other_variable$_{nom}$, another_variable$_{nom}$)
  - do not forget to add the test case including all nominal values:
    - (variable$^1_{nom}$, variable$^2_{nom}$, variable$^3_{nom}$)

This would lead to a total of (4*n + 1) test cases (4 different values for each variable, and 1 case consisting of all nominal values).

Please note that the tests are generated using "@TestFactory" approach of JUnit 5, which allows for creating DynamicTest collections and providing them names for execution, easily allowing creating of separate test cases for combinations of values from variables.

For generation of strings, I created functions that randomly generates strings from a character pool consisting of: { a to z, A to Z, 0 to 9 }(endings inclusive). Another function generates triples of strings for test cases, given three of the lengths wanted. See the code below:

```
String characterPool = new String ("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");
int characterPoolSize = characterPool.length();

public String stringGeneratorbyLength(Random r, int stringLength) {
  if (stringLength == 0)
    return new String("");

  String resultString = new String("");
    for (int i=0; i<stringLength; i++) {
      int pickedValue = r.nextInt(characterPoolSize);
      resultString = resultString + characterPool.charAt(pickedValue);
    }
    return resultString;
}

public Triple<String, String, String> mailGeneratorByLength(Random r, int localLength, int
     domainLength, int topLength) {
  String _local = stringGeneratorbyLength(r, localLength);
  String _domain = stringGeneratorbyLength(r, domainLength);
  String _top = stringGeneratorbyLength(r, topLength);

  Triple<String, String, String> emailTriple = Triple.of(_local, _domain, _top);
  return emailTriple;
}
```

SimpleBVT test case generation part is the code below (for full code, please refer to the Appendix part of this document):

```
@TestFactory
Collection<DynamicTest> simpleBVT() {
  List<Integer> localValues = Arrays.asList( localMinLength, localMinLength_plus, localMaxLength_minus,
       localMaxLength);
  List<Integer> domainValues  = Arrays.asList( domainMinLength  , domainMinLength_plus  ,
       domainMaxLength_minus , domainMaxLength);
  List<Integer> topValues   = Arrays.asList( topMinLength, topMinLength_plus, topMaxLength_minus,
       topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  for (Integer lVal: localValues) {
    Triple<String, String, String> s = mailGeneratorByLength(r, lVal, domainNomLength, topNomLength);
    DynamicTest t = DynamicTest.dynamicTest(testMessage,
      () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
           s.getRight()), "pass"));
    tests.add(t);
  }
  /* similar for loops for domainValues and topValues is done, omitted from the document for shortness */

  return tests;
}
```

b) Create and implement a test suite with Junit5 using Robustness BV Testing. Explain the changes and additions you made to the test suite.

Robustness Boundary Value Testing:

Test Suite	Local Length	Domain Length	Top Length	email Validator	Actual Result
[Robustness]	33	32	32	true	true
[Robustness]	0	65	32	false	false
[Robustness]	1	65	32	true	true
[Robustness]	2	65	32	true	true
[Robustness]	63	65	32	true	true
[Robustness]	64	65	32	true	true
[Robustness]	65	65	32	false	false
[Robustness]	33	0	32	false	true
[Robustness]	33	1	32	true	true
[Robustness]	33	2	32	true	true
[Robustness]	33	127	32	true	true
[Robustness]	33	128	32	true	true
[Robustness]	33	129	32	false	false
[Robustness]	33	32	0	false	false
[Robustness]	33	32	1	true	true
[Robustness]	33	32	2	true	true
[Robustness]	33	32	62	true	true
[Robustness]	33	32	63	true	true
[Robustness]	33	32	64	false	false

For robustness,
instead of using the values { min, min+, max-, max },
we use { min-, min, min+, max-, max, max+ }
(we added min- and max+, to underline)

RobustnessBVT test case generation part is the code below (for full code, refer to the Appendix):

```
@TestFactory
Collection<DynamicTest> robustnessBVT() {
  List<Integer> localValues = Arrays.asList( localMinLength_minus, localMinLength,
      localMinLength_plus, localMaxLength_minus, localMaxLength, localMaxLength_plus);
  List<Integer> domainValues  = Arrays.asList( domainMinLength_minus, domainMinLength ,
      domainMinLength_plus  , domainMaxLength_minus , domainMaxLength, domainMaxLength_plus);
  List<Integer> topValues   = Arrays.asList( topMinLength_minus, topMinLength, topMinLength_plus,
      topMaxLength_minus, topMaxLength, topMaxLength_plus);

  List<Integer> safeL = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
      localMaxLength_minus, localMaxLength);
  List<Integer> safeD = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
      domainMaxLength_minus  , domainMaxLength);
  List<Integer> safeT = Arrays.asList( topMinLength, topMinLength_plus, topNomLength,
      topMaxLength_minus, topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  // all nominal case
  Triple<String, String, String> sNom = mailGeneratorByLength(r, localNomLength, domainNomLength,
      topNomLength);
  String testMessageNom = "[Robustness] -> with [ " + ((Integer)localNomLength) + " " +
      ((Integer)topNomLength) + " " +  ((Integer)topNomLength) + " ]" ;

  DynamicTest tNom = DynamicTest.dynamicTest(testMessageNom,
    () -> Assertions.assertTrue(validator.validateEMailAdress(sNom.getLeft(), sNom.getMiddle(),
       sNom.getRight()), "pass"));
  tests.add(tNom);

  for (Integer lVal: localValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, lVal, domainNomLength, topNomLength);
    Boolean isSafe = safeL.contains(lVal);
    String testMessage = "[Robustness] -> with [ " + lVal.toString() + " " +
        ((Integer)domainNomLength) + " " + ((Integer)topNomLength) + " ]";

    DynamicTest t;
    if (isSafe)
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    else
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    tests.add(t);
  }
  /* similar for loops for domainValues and topValues is done, omitted from the document for shortness
      */
  return tests;
}
```

c) Create and implement a test suite with Junit5 using the Worst Case BV Testing. Explain the changes and additions you made to the test suite. For this provide a generator which computes the needed input/output values. Explain how your generator works.

L	D	T	eV	R
1	1	1	true	true
1	1	2	true	true
1	1	32	true	true
1	1	62	true	true
1	1	63	true	true
1	2	1	true	true
1	2	2	true	true
1	2	32	true	true
1	2	62	true	true
1	2	63	true	true
1	65	1	true	true
1	65	2	true	true
1	65	32	true	true
1	65	62	true	true
1	65	63	true	true
1	127	1	true	true
1	127	2	true	true
1	127	32	true	true
1	127	62	true	true
1	127	63	true	true
1	128	1	true	true
1	128	2	true	true
1	128	32	true	true
1	128	62	true	true
1	128	63	true	true
2	1	1	true	true
2	1	2	true	true
2	1	32	true	true
2	1	62	true	true
2	1	63	true	true
2	2	1	true	true
2	2	2	true	true
2	2	32	true	true
2	2	62	true	true
2	2	63	true	true
2	65	1	true	true
2	65	2	true	true
2	65	32	true	true
2	65	62	true	true
2	65	63	true	true
2	127	1	true	true
2	127	2	true	true
2	127	32	true	true
2	127	62	true	true
2	127	63	true	true
2	128	1	true	true
2	128	2	true	true
2	128	32	true	true
2	128	62	true	true
2	128	63	true	true
33	1	1	true	true
33	1	2	true	true
33	1	32	true	true
33	1	62	true	true
33	1	63	true	true
33	2	1	true	true
33	2	2	true	true
33	2	32	true	true
33	2	62	true	true
33	2	63	true	true
33	65	1	true	true
33	65	2	true	true
33	65	32	true	true

L	D	T	eV	R
33	65	63	true	true
33	65	62	true	true
33	127	1	true	true
33	127	2	true	true
33	127	32	true	true
33	127	62	true	true
33	127	63	true	true
33	128	1	true	true
33	128	2	true	true
33	128	32	true	true
33	128	62	true	true
33	128	63	true	true
63	1	1	true	true
63	1	2	true	true
63	1	32	true	true
63	1	62	true	true
63	1	63	true	true
63	2	1	true	true
63	2	2	true	true
63	2	32	true	true
63	2	62	true	true
63	2	63	true	true
63	65	1	true	true
63	65	2	true	true
63	65	32	true	true
63	65	62	true	true
63	65	63	true	true
63	127	1	true	true
63	127	2	true	true
63	127	32	true	true
63	127	62	true	true
63	127	63	true	true
63	128	1	true	true
63	128	2	true	true
63	128	32	true	true
63	128	62	true	true
63	128	63	true	true
64	1	1	true	true
64	1	2	true	true
64	1	32	true	true
64	1	62	true	true
64	1	63	true	true
64	2	1	true	true
64	2	2	true	true
64	2	32	true	true
64	2	62	true	true
64	2	63	true	true
64	65	1	true	true
64	65	2	true	true
64	65	32	true	true
64	65	62	true	true
64	65	63	true	true
64	127	1	true	true
64	127	2	true	true
64	127	32	true	true
64	127	62	true	true
64	127	63	true	true
64	128	1	true	true
64	128	2	true	true
64	128	32	true	true
64	128	62	true	true
64	128	63	true	true

Instead of only using a variable with the other varibles' nominal values, we generate every possible test case out of the possible values.

Each variable can have 5 different values { min, min+, nom, max-, max}, and we have 3 different variables:

- 5^n test cases
- for our test suite, it makes 125 test cases.

Test case generation has the following format:

```
for v1 in values1:
  for v2 in values2:
    for v3 in values3:
      testCase = (v1, v2, v3)
```

How random string generation works is explained in the SimpleBVT case above.

For generating educated predictions on the outcomes of the tests (not the one emailValidator provides, but the ones that are created to check whether emailValidator is working right), I used the following approach:

```
values1 = {...}
values2 = {...}
values3 = {...}

safeV1 = {...}
safeV2 = {...}
safeV3 = {...}

for v1 in values1:
  for v2 in values2:
    for v3 in values3:

      testCase = (v1, v2, v3)

      isSafe = (v1 elem safeV1) && (v2 elem
          safeV2) && (v3 elem safeV3)

      if (isSafe)
        /*
          assertTrue
        */

      else
        /*
          assertFalse
        */
```

WorstCaseBVT test case generation part is the code below (for full code, refer to the Appendix):

```
@TestFactory
Collection<DynamicTest> worstCaseBVT() {

  List<Integer> localValues = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
      localMaxLength_minus, localMaxLength);
  List<Integer> domainValues  = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
      domainMaxLength_minus  , domainMaxLength);
  List<Integer> topValues   = Arrays.asList( topMinLength, topMinLength_plus, topNomLength,
      topMaxLength_minus, topMaxLength);

  List<Integer> safeL = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
      localMaxLength_minus, localMaxLength);
  List<Integer> safeD = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
      domainMaxLength_minus  , domainMaxLength);
  List<Integer> safeT = Arrays.asList( topMinLength, topMinLength_plus, topNomLength,
      topMaxLength_minus, topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  for (Integer lVal: localValues)
  {
    for (Integer dVal: domainValues)
    {
      for (Integer tVal: topValues)
      {
        Boolean isSafe = (safeL.contains(lVal) && safeD.contains(dVal) && safeT.contains(tVal));
        String testMessage = "[sWorst] -> with [ " + lVal.toString() + " " + dVal.toString() + " " +
            tVal.toString() + " ]";

        Triple<String, String, String> s = mailGeneratorByLength(r, lVal, dVal, tVal);

        Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
        System.out.println(testMessage + " " + isSafe + " | " + actualResult);

        DynamicTest t;
        if ( isSafe )
          t = DynamicTest.dynamicTest(testMessage,
            () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                s.getRight()), "pass"));
        else
          t = DynamicTest.dynamicTest(testMessage,
            () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                s.getRight()), "pass"));
        tests.add(t);
      }
    }
  }

  System.out.println("[WorstCase] #Cases: " + ((Integer)tests.size()).toString());
  System.out.println();
  return tests;
}
```

d) Create and implement a test suite with Junit5 representing the Robustness Worst Case BV Testing. Explain the changes and additions you made to the test suite.

Test Suite	Local Length	Domain Length	Top Length	email Validator	Actual Result
[RobustnessWorst]	0	0	0	false	true
[RobustnessWorst]	0	0	1	false	true
[RobustnessWorst]	0	0	2	false	true
[RobustnessWorst]	0	0	32	false	true
[RobustnessWorst]	0	0	62	false	true
[RobustnessWorst]	0	0	63	false	true
[RobustnessWorst]	0	0	64	false	true
[RobustnessWorst]	1	0	0	false	true
[RobustnessWorst]	1	0	1	false	true
[RobustnessWorst]	1	0	2	false	true
[RobustnessWorst]	1	0	32	false	true
[RobustnessWorst]	1	0	62	false	true
[RobustnessWorst]	1	0	63	false	true
[RobustnessWorst]	1	0	64	false	true
[RobustnessWorst]	1	128	0	false	true
[RobustnessWorst]	2	0	0	false	true
[RobustnessWorst]	2	0	1	false	true
[RobustnessWorst]	2	0	2	false	true
[RobustnessWorst]	2	0	32	false	true
[RobustnessWorst]	2	0	62	false	true
[RobustnessWorst]	2	0	63	false	true
[RobustnessWorst]	2	0	64	false	true
[RobustnessWorst]	33	0	0	false	true
[RobustnessWorst]	33	0	1	false	true
[RobustnessWorst]	33	0	2	false	true
[RobustnessWorst]	33	0	32	false	true
[RobustnessWorst]	33	0	62	false	true
[RobustnessWorst]	33	0	63	false	true
[RobustnessWorst]	33	0	64	false	true
[RobustnessWorst]	63	0	0	false	true
[RobustnessWorst]	63	0	1	false	true
[RobustnessWorst]	63	0	2	false	true
[RobustnessWorst]	63	0	32	false	true
[RobustnessWorst]	63	0	62	false	true
[RobustnessWorst]	63	0	63	false	true
[RobustnessWorst]	63	0	64	false	true
[RobustnessWorst]	64	0	0	false	true
[RobustnessWorst]	64	0	1	false	true
[RobustnessWorst]	64	0	2	false	true
[RobustnessWorst]	64	0	32	false	true
[RobustnessWorst]	64	0	62	false	true
[RobustnessWorst]	64	0	63	false	true
[RobustnessWorst]	64	0	64	false	true
[RobustnessWorst]	65	0	0	false	true
[RobustnessWorst]	65	0	1	false	true
[RobustnessWorst]	65	0	2	false	true
[RobustnessWorst]	65	0	32	false	true
[RobustnessWorst]	65	0	62	false	true
[RobustnessWorst]	65	0	63	false	true
[RobustnessWorst]	65	0	64	false	true

Instead of using { min, min+, nom, max-, max } values for variables (like in WorstCase), we use { min-, min, min+, nom, max-, max, max+ } values for each variable. Test case generation and generator part are, other than the values used, the same as WorstCase part.

Note: the values that are displayed on the left only contains the failed tests, as putting 7^3 = 343 test in the report would not make sense.

RobustnessWorstCaseBVT test case generation part is the code below (for full code, refer to the Appendix):

```
@TestFactory
Collection<DynamicTest> robustnessWorstCaseBVT() {

  List<Integer> localValues = Arrays.asList( localMinLength_minus, localMinLength,
      localMinLength_plus,localNomLength, localMaxLength_minus, localMaxLength, localMaxLength_plus);
  List<Integer> domainValues  = Arrays.asList( domainMinLength_minus, domainMinLength ,
      domainMinLength_plus, domainNomLength, domainMaxLength_minus  , domainMaxLength,
      domainMaxLength_plus);
  List<Integer> topValues = Arrays.asList( topMinLength_minus, topMinLength, topMinLength_plus,
      topNomLength, topMaxLength_minus, topMaxLength, topMaxLength_plus );

  List<Integer> safeL = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
      localMaxLength_minus, localMaxLength);
  List<Integer> safeD = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
      domainMaxLength_minus  , domainMaxLength);
  List<Integer> safeT = Arrays.asList( topMinLength, topMinLength_plus, topNomLength,
      topMaxLength_minus, topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  for (Integer lVal: localValues)
  {

    for (Integer dVal: domainValues)
    {
      for (Integer tVal: topValues)
      {
        Boolean isSafe = (safeL.contains(lVal) && safeD.contains(dVal) && safeT.contains(tVal));
        String testMessage = "[RobustnessWorst] -> with [ " + lVal.toString() + " " + dVal.toString()
            + " " + tVal.toString() + " ]";

        Triple<String, String, String> s = mailGeneratorByLength(r, lVal, dVal, tVal);

        Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
        if (isSafe != actualResult)
        System.out.println(testMessage + " " + isSafe + " | " + actualResult);

        DynamicTest t;
        if ( isSafe )
          t = DynamicTest.dynamicTest(testMessage,
            () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                s.getRight()), "pass"));
        else
          t = DynamicTest.dynamicTest(testMessage,
            () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                s.getRight()), "pass"));
        tests.add(t);
      }
    }
  }
  System.out.println("[RobustnessWorstCase] #Cases: " + ((Integer)tests.size()).toString());
  System.out.println();
  return tests;
}
```

e) Please list all bugs found in the EMailValidator Component.

I have managed to found two bugs in the EMailValidator component:

- in case an input in which the string length of the domain part is zero (e.g. (any, "", any)),
  the EMailValidator would judge this input as a valid input, while it is not at all.

- I found also a single case which is interesting:
  if one provides an input where local-part string of length 1, domain-part string is of length 128, and top-part string is of length is 0,
  the EMailValidator would accept this case as a valid, which is not valid at all.

  – example case:

```
Assertions.assertTrue(
  validator.validateEMailAdress("a", "aaaaa....aaaa" /* of length 128 */, ""),
  "this e-mail should have been marked invalid"
);
```

Moreover, I could have checked whether there are any errors on accepting characters outside the allowed character pool (characters such as *, /, {, }; characters that are not alpha-numeric), yet, the assignment put emphasis on the length of the input parts, not the input sanity check. However, it is in the best intentions that input sanity check should be done, which the test suites I provided fails to do so.

3.2. Task: Equivalence Class Testing

Task: Equivalence Classes (Part – a , b and d):

Equivalence Classes :

a) As a first step, derive suitable equivalence classes from DHL's price model and take the different package characteristics into account. Please provide some information about the equivalence relation you've used for each characteristic.

Solution:

Step 1:

Extract input and output conditions:

1. Output is the price to be paid to DHL for delivery
2. Destination Codes are used such as zone 0 for inside Germany, zone 1 for EU and zone 2 to 8 for rest of the world
3. Weight of the package can be anywhere from 1Kg to 31,5Kg
4. Maximum Dimensions (L*B*H) has to be specified based on Package type such as (PäckchenXS, PaketXL) for zone 0 and based on weight in zone 1-8
5. Price depends on where the payment is made (Online/Branch/DHL Shop)
6. Quantity has to be specified to get separate offers for more sets of packages

Step 2:

Defining equivalence classes:

Conditions	Valid EC	Invalid EC
Destination Zone	**0(1)**	**Zone <0(1a)**
Max Weight	**0<=Weight<=31,5(2)**	**Weight<0(2a), Weight >31,5(2b)**
Max Dimension	**L*B*H: 30 x 30 x 15 cm (upto weight =1kg) (3.1)/ 60 x 30 x 15 cm(upto weight =2kg)(3.2)/ 120 x 60 x 60 cm (upto weight= 31,5)(3.3)(**	**Dimension greater than the maximum limit for specified weight(3a)**
Payment Method	**Online/Branch/ dhl-shop.de valid for Quantity ={10,50,100}(4)**	**Other than three methods of payment(4a)**
Quantity	**1/ 10/ 50/100(5)**	**Quantity other than {1,10,50,100}(5a)**

Conditions	Valid EC	Invalid EC
Destination Zone	**1-8**	**Zone other than {0,1,2,3,4,5,6,7,8}**
Max Weight	**1<=Weight<=31,5**	**Weight<1, Weight >31,5**
Max Dimension	**For Weight<=2kg, L+B+H=90cm && {L,B,H} <60 For Weight > 2kg, L*B*H:**	**For Weight<=2kg, L+B+H >90 cm For Weight > 2kg ,**

	{120 x 60 x60 cm}	**L*B*H dimension greater than {120 x 60 x 60}**
Payment Method	**Online/Branch/ dhl-shop.de valid for Quantity ={10,50,100}**	**Other than three methods**
Quantity	**1/3/10/50/100**	**Quantity other than {1,3,10,50,100}**

b) Select representatives and create a test suite which satisfies the weak equivalence class test criterion (This can be done manually). Enrich the test cases with expected results. You can derive them manually from the document or from the DHL online service. Store the resulting test suite as a CSV file.

Solution:

Representatives test suite for weak equivalence class criteria:

testcase	**Destination Zone**	**Max Weight**	**Max Dimension(L+B+H)**	**Payment Method**	**Quantity**	**EC**	**Expected result**	**Expected Value**
1	0	10	47	Online	10	1,2,3,4,5	Valid	182.83
2	0	10	47	Online	19	1,2,3,4,5a	Invalid	-1
3	0	10	47	Online1	10	1,2,3,4a,5	Invalid	-1
4	0	10	250	Online	10	1,2,3a,4,5	Invalid	-1
5	0	111	47	Online	10	1,2a,3,4,5	Invalid	-1
6	-1	10	47	Online	10	1a,2,3,4,5	Invalid	-1

Here we consider the invalid conditions as basis for weak equivalence class along with the valid conditions which tries to accommodate all the valid conditions possible by extending the range. However, the individual internal conditions can’t be shown because they are not independent.

c) Select representatives and create a test suite which satisfies the weak equivalence class test criterion (This can be done manually). Enrich the test cases with expected results. You can derive them manually from the document or from the DHL online service. Store the resulting test suite as a CSV file.

CalculatePrice Function : The CalculatePrice function takes the parameters type, weight, zone, dimension, paymentMethod and quantity. This function

calculates the price similar to the one shown in the DHL site. The calculation differs based on productType and paymentMethod. The invalid inputs are also handled by returning -1 if they are not in specified range/invalid.

```
price=(weight*(zone+1)*dimension*quantity)/100
```

```
public class calculatePriceClass {

	public double calculatePrice(ProductType type, double weight, int
zone, double dimension, paymentMethod pm, int quantity)
	{
		double price=1.0;
		double pricePerUnit=0.0;
		double total=0.0;
		if( quantity <0 || quantity%10!=0 || quantity>50)
			return -1;
		if( dimension <0 || dimension>240)
			return -1;
		if( zone <0 || zone>8)
			return -1;
		if( weight <0 || weight>31.5)
			return -1;

		switch(type) {
		case pXS:
			if(pm == paymentMethod.online)
				pricePerUnit=3.89;
			else
				pricePerUnit=4.0;
			break;
		case pS:
			if(pm == paymentMethod.online)
				pricePerUnit=4.39;
			else
				pricePerUnit=4.5;
			break;
		case s:
			if(pm == paymentMethod.online)
				pricePerUnit=4.99;
			else
				pricePerUnit=0;
			break;
		case m:
			if(pm == paymentMethod.online)
				pricePerUnit=5.99;
			else
				pricePerUnit=6.99;
			break;
		case l:
			if(pm == paymentMethod.online)
				pricePerUnit=8.49;
			else
				pricePerUnit=9.49;
			break;
```

```
            case xl:
                  pricePerUnit=16.49;
                  break;
            case rgp:
                  pricePerUnit=16.49;
                  break;
            default: pricePerUnit=-1.0;
            }


            price=weight*(zone+1)*dimension*quantity;
            total=(price*pricePerUnit)/100;

            if(total>10 && pm == paymentMethod.shop)
                  return total+9.99;
            return total;
      }
}
```

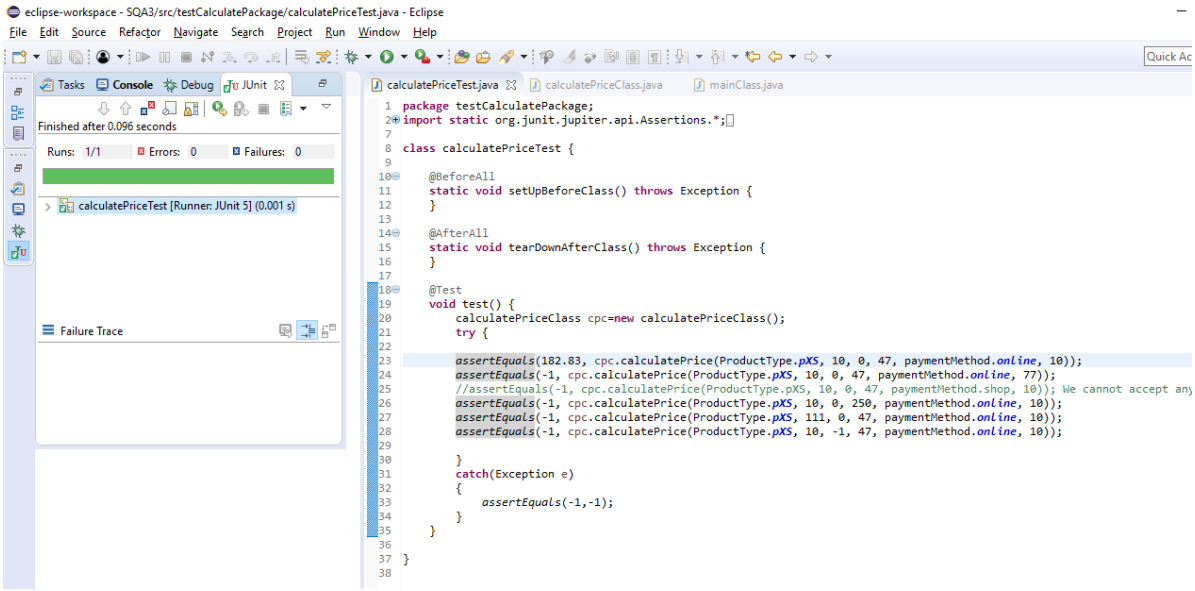
Considering the equivalence class in 2b (the csv file), we are writing test cases as shown below:

```
@Test
void test() {
    calculatePriceClass cpc=new calculatePriceClass();
    try {

    assertEquals(182.83, cpc.calculatePrice(ProductType.pXS, 10, 0, 47, paymentMethod.online, 10));
    assertEquals(-1, cpc.calculatePrice(ProductType.pXS, 10, 0, 47, paymentMethod.online, 77));
    //assertEquals(-1, cpc.calculatePrice(ProductType.pXS, 10, 0, 47, paymentMethod.shop, 10));
    //We cannot accept any other value for ProductType and PaymentMethod than those mentioned in the enum clas
    assertEquals(-1, cpc.calculatePrice(ProductType.pXS, 10, 0, 250, paymentMethod.online, 10));
    assertEquals(-1, cpc.calculatePrice(ProductType.pXS, 111, 0, 47, paymentMethod.online, 10));
    assertEquals(-1, cpc.calculatePrice(ProductType.pXS, 10, -1, 47, paymentMethod.online, 10));

    }
```

Instead of importing the CSV file, we are testing using the same values as shown above. And we can see that all the above test cases have passed as seen below.



The project is there inside the folder SQA3.


SQA3.zip

d) How can the equivalence classes be extended by boundary values? Please note: You don't have to implement this.

Solution:

The equivalence classes can be extended to boundary values by ordering the set of test cases according to each category/condition and then defining the boundaries as follows. This boundary has the maximum range of all the subcategories or conditions.

The equivalence class for all zones can be extended by boundary values as follows:

Condition	min-	min	min+	nom	max-	max	max+
Destination Zone	-1	0	1	4	7	8	9
Max Weight	-1	0	1	16	31	31.5	32
Max Dimension(upto 31.5kg)	-1x-1x-1	0x0x0	1x1x1	60x30x30	119x59x59	120x60x60	121x61x61
Payment Method	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Quantity	0	1	3	10	50	100	101

Payment method and quantity does not yield proper results as the in between range input don’t have valid output.

Max dimensions tries to accommodate all the categories and hence has the highest range of all categories.

3.3. Appendix for Task 3.1

```java
package de.rwth.swc.teaching.sqa;


import org.apache.commons.lang3.tuple.Triple;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestFactory;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.Random;

public class EmailValidatorTest {

  private static EmailValidator validator;

  @BeforeAll
  public static void init(){
  validator =  new EmailValidator();
  }

  // localPart boundary variables
  public static final int localMinLength_minus = 0;
  public static final int localMinLength = 1;
  public static final int localMinLength_plus = 2;

  public static final int localNomLength = 33;

  public static final int localMaxLength_minus = 63;
  public static final int localMaxLength = 64;
  public static final int localMaxLength_plus = 65;

  // domainPart boundary variables
  public static final int domainMinLength_minus = 0;
  public static final int domainMinLength = 1;
  public static final int domainMinLength_plus = 2;

  public static final int domainNomLength = 65;

  public static final int domainMaxLength_minus = 127;
  public static final int domainMaxLength = 128;
  public static final int domainMaxLength_plus = 129;

  // topLevelPart boundary variables
  public static final int topMinLength_minus = 0;
  public static final int topMinLength = 1;
  public static final int topMinLength_plus = 2;

  public static final int topNomLength = 32;

  public static final int topMaxLength_minus = 62;
  public static final int topMaxLength = 63;
  public static final int topMaxLength_plus = 64;

  String characterPool = new String ("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");
  int characterPoolSize = characterPool.length();

  public String stringGeneratorbyLength(Random r, int stringLength) {
    if (stringLength == 0)
      return new String("");

    String resultString = new String("");
    for (int i=0; i<stringLength; i++) {
      int pickedValue = r.nextInt(characterPoolSize);
      resultString = resultString + characterPool.charAt(pickedValue);
    }

    return resultString;
  }

  public Triple<String, String, String> mailGeneratorByLength(Random r, int localLength, int domainLength,
      int topLength) {
    String _local = stringGeneratorbyLength(r, localLength);
    String _domain = stringGeneratorbyLength(r, domainLength);
    String _top = stringGeneratorbyLength(r, topLength);

    Triple<String, String, String> emailTriple = Triple.of(_local, _domain, _top);
    return emailTriple;
  }
```

```
@TestFactory
Collection<DynamicTest> simpleBVT() {
  List<Integer> localValues = Arrays.asList( localMinLength, localMinLength_plus, localMaxLength_minus,
      localMaxLength);
  List<Integer> domainValues  = Arrays.asList( domainMinLength  , domainMinLength_plus  ,
      domainMaxLength_minus , domainMaxLength);
  List<Integer> topValues   = Arrays.asList( topMinLength, topMinLength_plus, topMaxLength_minus,
      topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  // all nominal case
  Triple<String, String, String> sNom = mailGeneratorByLength(r, localNomLength, domainNomLength,
      topNomLength);
  String testMessageNom = "[Simple] -> with [ " + ((Integer)localNomLength) + " " +
      ((Integer)topNomLength) + " " +  ((Integer)topNomLength) + " ]" ;
  System.out.println(testMessageNom);
  DynamicTest tNom;
  tNom = DynamicTest.dynamicTest(testMessageNom,
  () -> Assertions.assertTrue(validator.validateEMailAdress(sNom.getLeft(), sNom.getMiddle(),
      sNom.getRight()), "pass"));
  tests.add(tNom);

  for (Integer lVal: localValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, lVal, domainNomLength, topNomLength);
    String testMessage = "[Simple] -> with [ " + lVal.toString() + " " + ((Integer)domainNomLength) + " "
        + ((Integer)topNomLength) + " ]";
    System.out.println(testMessage);
    DynamicTest t;
    t = DynamicTest.dynamicTest(testMessage,
      () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
          s.getRight()), "pass"));
    tests.add(t);
  }

  for (Integer dVal: domainValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, localNomLength, dVal, topNomLength);
    String testMessage = "[Simple] -> with [ " + ((Integer)localNomLength).toString() + " " +
        dVal.toString() + " " + ((Integer)topNomLength).toString() + " ]";
    System.out.println(testMessage);
    DynamicTest t;
    t = DynamicTest.dynamicTest(testMessage,
      () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
          s.getRight()), "pass"));
    tests.add(t);
  }

  for (Integer tVal: topValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, localNomLength, domainNomLength, tVal);
    String testMessage = "[Simple] -> with [ " + ((Integer)localNomLength).toString() + " " +
        ((Integer)topNomLength).toString() + " " + tVal.toString() + " ]" ;
    System.out.println(testMessage);
    DynamicTest t;
    t = DynamicTest.dynamicTest(testMessage,
      () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
          s.getRight()), "pass"));
    tests.add(t);
  }

  System.out.println("[Simple] #Cases: " + ((Integer)tests.size()).toString());
  System.out.println();
  return tests;
}

@TestFactory
  Collection<DynamicTest> robustnessBVT() {
  List<Integer> localValues = Arrays.asList( localMinLength_minus, localMinLength, localMinLength_plus,
      localMaxLength_minus, localMaxLength, localMaxLength_plus);
  List<Integer> domainValues  = Arrays.asList( domainMinLength_minus, domainMinLength ,
      domainMinLength_plus  , domainMaxLength_minus , domainMaxLength, domainMaxLength_plus);
  List<Integer> topValues   = Arrays.asList( topMinLength_minus, topMinLength, topMinLength_plus,
      topMaxLength_minus, topMaxLength, topMaxLength_plus);

  List<Integer> safeL = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
      localMaxLength_minus, localMaxLength);
  List<Integer> safeD = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
      domainMaxLength_minus  , domainMaxLength);
  List<Integer> safeT = Arrays.asList( topMinLength, topMinLength_plus, topNomLength, topMaxLength_minus,
      topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();
```

```
  // all nominal case
  Triple<String, String, String> sNom = mailGeneratorByLength(r, localNomLength, domainNomLength,
      topNomLength);
  String testMessageNom = "[Robustness] -> with [ " + ((Integer)localNomLength) + " " +
      ((Integer)topNomLength) + " " +  ((Integer)topNomLength) + " ]" ;

  Boolean actualResultNom = validator.validateEMailAdress(sNom.getLeft(), sNom.getMiddle(),
      sNom.getRight());
  Boolean isSafeNom = true;
  System.out.println(testMessageNom + " " + isSafeNom + " | " + actualResultNom);

  DynamicTest tNom;
  tNom = DynamicTest.dynamicTest(testMessageNom,
    () -> Assertions.assertTrue(validator.validateEMailAdress(sNom.getLeft(), sNom.getMiddle(),
        sNom.getRight()), "pass"));
  tests.add(tNom);

  for (Integer lVal: localValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, lVal, domainNomLength, topNomLength);
    Boolean isSafe = safeL.contains(lVal);
    String testMessage = "[Robustness] -> with [ " + lVal.toString() + " " + ((Integer)domainNomLength) +
        " " + ((Integer)topNomLength) + " ]";

    Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
    System.out.println(testMessage + " " + isSafe + " | " + actualResult);

    DynamicTest t;
    if (isSafe)
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    else
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    tests.add(t);
  }

  for (Integer dVal: domainValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, localNomLength, dVal, topNomLength);
    Boolean isSafe = safeD.contains(dVal);
    String testMessage = "[Robustness] -> with [ " + ((Integer)localNomLength).toString() + " " +
        dVal.toString() + " " + ((Integer)topNomLength).toString() + " ]";

    Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
    System.out.println(testMessage + " " + isSafe + " | " + actualResult);

    DynamicTest t;
    if (isSafe)
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    else
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    tests.add(t);
  }

  for (Integer tVal: topValues)
  {
    Triple<String, String, String> s = mailGeneratorByLength(r, localNomLength, domainNomLength, tVal);
    Boolean isSafe = safeT.contains(tVal);
    String testMessage = "[Robustness] -> with [ " + ((Integer)localNomLength).toString() + " " +
        ((Integer)topNomLength).toString() + " " + tVal.toString() + " ]" ;

    Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
    System.out.println(testMessage + " " + isSafe + " | " + actualResult);

    DynamicTest t;
    if (isSafe)
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    else
      t = DynamicTest.dynamicTest(testMessage,
        () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
            s.getRight()), "pass"));
    tests.add(t);
  }

  System.out.println("[Robustness] #Cases: " + ((Integer)tests.size()).toString());
  System.out.println();
  return tests;
}
```

```
@TestFactory
Collection<DynamicTest> worstCaseBVT() {

  List<Integer> localValues = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
       localMaxLength_minus, localMaxLength);
  List<Integer> domainValues  = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
       domainMaxLength_minus  , domainMaxLength);
  List<Integer> topValues   = Arrays.asList( topMinLength, topMinLength_plus, topNomLength,
       topMaxLength_minus, topMaxLength);

  List<Integer> safeL = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
       localMaxLength_minus, localMaxLength);
  List<Integer> safeD = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
       domainMaxLength_minus  , domainMaxLength);
  List<Integer> safeT = Arrays.asList( topMinLength, topMinLength_plus, topNomLength, topMaxLength_minus,
       topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  for (Integer lVal: localValues)
  {
    for (Integer dVal: domainValues)
    {
      for (Integer tVal: topValues)
      {
        Boolean isSafe = (safeL.contains(lVal) && safeD.contains(dVal) && safeT.contains(tVal));
        String testMessage = "[Worst] -> with [ " + lVal.toString() + " " + dVal.toString() + " " +
             tVal.toString() + " ]";

        Triple<String, String, String> s = mailGeneratorByLength(r, lVal, dVal, tVal);

        Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
        System.out.println(testMessage + " " + isSafe + " | " + actualResult);

        DynamicTest t;
        if ( isSafe )
          t = DynamicTest.dynamicTest(testMessage,
            () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                 s.getRight()), "pass"));
        else
          t = DynamicTest.dynamicTest(testMessage,
            () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                 s.getRight()), "pass"));
        tests.add(t);
      }
    }
  }

  System.out.println("[WorstCase] #Cases: " + ((Integer)tests.size()).toString());
  System.out.println();
  return tests;
}

@TestFactory
Collection<DynamicTest> robustnessWorstCaseBVT() {

  List<Integer> localValues = Arrays.asList( localMinLength_minus, localMinLength,
       localMinLength_plus,localNomLength, localMaxLength_minus, localMaxLength, localMaxLength_plus);
  List<Integer> domainValues  = Arrays.asList( domainMinLength_minus, domainMinLength ,
       domainMinLength_plus, domainNomLength, domainMaxLength_minus  , domainMaxLength,
       domainMaxLength_plus);
  List<Integer> topValues = Arrays.asList( topMinLength_minus, topMinLength, topMinLength_plus,
       topNomLength, topMaxLength_minus, topMaxLength, topMaxLength_plus );

  List<Integer> safeL = Arrays.asList( localMinLength, localMinLength_plus,localNomLength,
       localMaxLength_minus, localMaxLength);
  List<Integer> safeD = Arrays.asList( domainMinLength, domainMinLength_plus, domainNomLength,
       domainMaxLength_minus  , domainMaxLength);
  List<Integer> safeT = Arrays.asList( topMinLength, topMinLength_plus, topNomLength, topMaxLength_minus,
       topMaxLength);

  Random r = new Random();
  ArrayList<DynamicTest> tests = new ArrayList<DynamicTest>();

  for (Integer lVal: localValues)
  {
    for (Integer dVal: domainValues)
    {
      for (Integer tVal: topValues)
      {
        Boolean isSafe = (safeL.contains(lVal) && safeD.contains(dVal) && safeT.contains(tVal));
        String testMessage = "[RobustnessWorst] -> with [ " + lVal.toString() + " " + dVal.toString() + "
             " + tVal.toString() + " ]";

        Triple<String, String, String> s = mailGeneratorByLength(r, lVal, dVal, tVal);
```

```
          Boolean actualResult = validator.validateEMailAdress(s.getLeft(), s.getMiddle(), s.getRight());
          if (isSafe != actualResult)
            System.out.println(testMessage + " " + isSafe + " | " + actualResult);

          DynamicTest t;
          if ( isSafe )
            t = DynamicTest.dynamicTest(testMessage,
              () -> Assertions.assertTrue(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                  s.getRight()), "pass"));
          else
            t = DynamicTest.dynamicTest(testMessage,
              () -> Assertions.assertFalse(validator.validateEMailAdress(s.getLeft(), s.getMiddle(),
                  s.getRight()), "pass"));
          tests.add(t);
        }
      }
    }
    System.out.println("[RobustnessWorstCase] #Cases: " + ((Integer)tests.size()).toString());
    System.out.println();
    return tests;
  }
}
```