Assignment 2

**1. A tester defined four characteristics based on the input parameter car: Where Made, Energy**

**Source, Size, and Color. The following partitioning for these characteristics have at least four**

**mistakes. Correct them. (10 pts.)**

**Where Made Energy Source Size Color**

**North America Gas 2-door White**

**Europe Electric 4-door Silver**

**Denmark Hybrid Hatch back Black**

**Africa Blue**

**Hint: each partition must satisfy two properties (Completeness and disjointness).**

**Solution:**

**1. Where Made** is not complete. We need to add **“other”**

**2. Energy Source** is not complete. The energy source value for Africa is not given. It is empty. We need to add a value and not leave it empty. Ex: Gas or Electric or Hybrid.

**3. Size** overlaps,

- a hatch-back could be 2-door or 4-door.

- either add “2-door + hatch-back,” and “4-door + hatch-back,”

- or create two new characteristics.

a) Side Doors: 2, 4

b) Hatch-back: yes, no

**4. Size** is not complete. The size value for Africa is not given. It is empty. We need to add any of the value mentioned in (3).

**5. Color** is not complete. We need to add **“other”**

**2. Answer the following questions for the method search() below: (10 pts.)**

**a) “Location of element in list” fails the disjointness property. Give an example that illustrates this.**

list= [9, 8, 9]; e = 9

list = [5]; e = 5

**b) “Location of element in list” fails the completeness property. Give an example that illustrates this.**

list=[2, 9]; e = 5 ie, If e is not in the list.

**c) Supply one or more new partitions that capture the intent of “Location of e in list” but do not suffer from completeness or disjointness problems.**

1. Whether e is first in the list: true, false

2. Whether e is last in the list: true, false

3. Whether e is in list: true, false

**3. Derive input space partitioning tests for the GenericStack class with the following method signatures: (15 pts.)**

**• public GenericStack ();**

**• public void Push (Object X);**

**• public Object Pop ();**

**• public boolean IsEmt ();**

**Assume the usual semantics for the stack. Try to keep your partitioning simple, choose a small number of partitions and blocks**

**a) Define characteristics of inputs.**

We need test four units the constructor and the three methods push, pop and isempt.There is definitely overlap between the characteristics of each methods. The constructor does not have inputs or implicit parameters.The state of the constructor GenericStack is the implicit parameter for the three methods. The explicit input among the three methods is parameter of push () where the explicit input is Object x.

The characteristics for the implicit state are

• If the stack is empty.

• The size of the stack.

• Does the stack contain null entries

• If x is null.

• Does Object x appear in the stack?

**b) Partition the characteristics into blocks.**

**c) Define values for the blocks**

**Solution for B and C**

**• If the stack is empty.**

true (Value stack = [])

false (Values stack = [”pen”], [”pen”, ”book”])

**• The size of the stack.**

0 (Value stack = [])

1 (Possible values stack = [”pen”], [null])

more than 1 (Possible values stack = [”pen”, ”book”], [”pen”, null], [”pen”, ”book”, ”pencil”])

**• Does the stack contain null entries**

true (Possible values stack = [null], [null, ”pen”, null])

false (Possible values stack = [”pen”, ”book”], [”pen”, ”book”, ”pencil”])

**• If x is null.**

true (Value x = null)

false (Possible values x = ”pen”, ”book”, ””)

**• Does Object x appear in the stack?**

true (Possible values: (null, [null, ”pen”, null]), (”pen”, [”pen”, ”book”]))

false (Possible values: (null, [”pen”]), (”pen”, [”book”, ”pencil”]))

**4. Answer the following questions for the method intersection() below: (15 pts.)**

**(a)  Does the partition “Type of s1” satisfy the completeness property? If not, give a value for s1 that does not fit in any block.**

Yes.

**(b)  Does the partition “Type of s1” satisfy the disjointness property? If not, give a value for s1 that fits in more than one block.**

Yes.

**(c)  Does the partition “Relation between s1 and s2” satisfy the completeness property? If not, give a pair of values for s1 and s2 that does not fit in any block.**

No, the relation between s1 and s2” does not satisfy the completeness property.

Example for a pair of values for s1 and s2 that does not fit in any block.

s1 = {1, 2};

s2 = {1, 3}

**(d)  Does the partition “Relation between s1 and s2” satisfy the disjointness property? If not, give a pair of values for s1 and s2 that fits in more than one block.**

No, the relation between s1 and s2” does not satisfy the disjointness property.

Example for a pair of values for s1 and s2 that fits in more than one block.

S1 = {1, 2}

S2 = {1,2}

The predicates for each of the first three blocks are satisfied.

**(e)  If the “Base Choice” criterion were applied to the two partitions (exactly as written), how many test requirements would result?**

There would be 6 test requirements in total.

1 for base requirement

2 for first characteristic

And 3 for second characteristic

**5. Derive input space partitioning tests for the BoundedQueue class with the following signature: (15 pts.)**

**• public BoundedQueue (int capacity);**

**• public void Enqueue (Object X);**

**• public Object Dequeue ();**

**• public boolean IsEmpty ();**

**• public boolean IsFull ()**

**Assume the usual semantics for a queue with a fixed, maximal capacity. Try to keep your partitioning simple–choose a small number of partitions and blocks.**

**a) Identify several characteristics that suggest partitions.**

1. Is the queue empty

2. Is the queue full

3. Size of the queue

4. Value of cap

5. Value of capacity

6. If x is null

**b) Identify the blocks in the partition for each characteristic. Designate one block in each partition as the “Base” block.**

**1. Is the queue empty**

- true (Value queue = [])

- false (Value queue = [”pencil”, ”book”]) Base

**2. Is the queue full**

- true (Value queue = [”pencil”], cap = 1)

- false (Value queue = [”pencil”, ”book”], cap = 3) Base

**3. Size of the queue**

- 0 (Value queue = [])

- 1 (Value queue = [”pencil”], )

- more than 1 (Values queue = [”pencil”, ”book”]) Base

**4. Value of cap**

- negative (Value cap = -1) (May not be possible.)

- 0 (Value cap = 0) (May not be possible.)

- 1 (Value cap = 1 )

- more than 1 (Value cap = 2) Base

**5. Value of capacity**

-negative (Value cap = -1)

-0 (Value cap = 0)

-1 (Value cap = 1 )

-more than 1 (Value cap = 2) Base

**6. If x is null**

-true (Value x = null)

-false (Value x = ”pencil” ) Base

**c) Define values for the blocks.**

The values of the blocks are defined in the solution (c).

**d) Define a test set that satisfies base choice coverage (BCC).**

The base test is (a2, b2, c3, d4, f2).

For varying each partition, we get 8 more tests:

{(a1, b2, c3, d4, f2), (a2, b1, c3, d4, f2), (a2, b2, c1, d4, f2), (a2, b2, c2, d4, f2),

(a2, b2, c3, d1, f2), (a2, b2, c3, d2, f2), (a2, b2, c3, d3, f2), (a2, b2, c3, d4, f1) }.

**6. Write down all 64 tests to satisfy the All Combinations (ACoC) criterion for the second categorization of triang()’s inputs in Table 4.2. Use the values in Table 4.3. (15 pts.)**

**Solution:**

{

(0,2,2), (0,2,1), (0,2,0), (0,2,−1), (0,1,2), (0,1,1), (0,1,0), (0,1,−1), (0,0,2), (0,0,1), (0,0,0), (0,0,−1), (0, −1, 2), (0, −1, 1), (0, −1, 0), (0, −1, −1),

(1,2,2), (1,2,1), (1,2,0), (1,2,−1), (1,1,2), (1,1,1), (1,1,0), (1,1,−1), (1,0,2),(1,0,1), (1,0,0),(1,0,−1), (1, −1, 2), (1, −1, 1), (1, −1, 0), (1, −1, −1),

(−1, 2, 2), (−1, 2, 1), (−1, 2, 0), (−1, 2, −1), (−1, 1, 2), (−1, 1, 1), (−1, 1, 0), (−1, 1, −1), (−1, 0, 2), (−1, 0, 1), (−1, 0, 0), (−1, 0, −1),(−1, −1, 2), (−1, −1, 1), (−1, −1, 0), (−1, −1, −1)

(2, 2, 2), (2, 2, 1), (2, 2, 0), (2, 2, −1), (2,1,2), (2,1,1), (2,1,0), (2,1,−1), (2,0,2), (2,0,1), (2,0,0), (2,0, −1), (2, −1, 2), (2, −1, 1), (2, −1, 0), (2, −1, −1)

}

**7. Enumerate all 16 tests to satisfy the pair-wise (PWC) criterion for the second categorization of TriTyp’s inputs in Table 4.2. Use the values in Table 4.3 (use the above tables). (10 pts.)**

**Solution:**

{

(2, 0, 0),

(2, −1, −1),

(−1, 1, 0),

(−1, 0, 1),

(−1, −1, 2),

(0, 1, −1),  
(0, −1, 1),

(−1, 2, −1),

(2, 2, 2),

(2, 1, 1),

(1, 2, 1),

(1, 1, 2),

(1, 0, −1),

(1, −1, 0),

(0, 2, 0)

(0, 0, 2),  
}

**8. Enumerate all 16 tests to satisfy the Multiple Base Choice (MBCC) criterion for the second categorization of TriTyp’s inputs in Table 4.2. Use the values in Table 4.3.**

**Solution:**

The 16 tests to satisfy the Multiple Base Choice are:

{

(1, 2, 2),

(2, 1, 2),

(2, 0, 2),

(2, −1, 2),

(1, 2, 0),

(1, 2, −1),

(1, 1, 2),

(2, 2, 2),

(2, 2, 1),

(2, 2, 0),

(2, 2, −1),

(0, 2, 2),

(−1, 2, 2),

(1, 2, 1),

(1, 0, 2),

(1, −1, 2)

}