1. Answer Q4 from Chapter 4, Section 4.2 of your textbook (pp 163-164).

Answer the following questions for the method intersection() below:

public Set intersection (Set s1, Set s2)

// Effects: If s1 or s2 are null throw NullPointerException

// else return a (non null) Set equal to the intersection

// of Sets s1 and s2

// A null argument is treated as an empty set.

Characteristic: Type of s1

- s1 = null

- s1 = {}

- s1 has at least one element

Characteristic: Relation between s1 and s2

- s1 and s2 represent the same set

- s1 is a subset of s2

- s2 is a subset of s1

- s1 and s2 do not have any elements in common

(a) Does the partition “Type of s1” satisfy the completeness property? If not, give a value for *s*1 that does not fit in any block.

* Type of s1 satisfies the completeness property.

(b) Does the partition “Type of s1” satisfy the disjointness property? If not, give a value for *s*1 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 *s*1 and *s*2 that does not fit in any block.

* No. s1 and s2 can have one or multiple elements in common without being a subset of each other. For example, if s1={1,2} and s2={2,3} – it does not satisfy any of the blocks.

(d) Does the partition “Relation between s1 and s2” satisfy the disjointness property? If not, give a pair of values for *s*1 and *s*2 that fits in more than one block.

* No. If s1=s2={4,5,6}, then it satisfies block 1 ( s1 and s2 represents the same set), block 2 ( s1 is a subset of s2) and block 3 ( s2 is a subset of s1).

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

* 1 (base choice) + 2 ( 3-1=2 tests for Characteristic: Type of s1 ) + 3 ( 4-1 =3 tests for Characteristic: Relation between s1 and s2) = 6.

1. Derive input space partitioning tests for the Roman class that you implemented in A1.

**public** **class** Roman {

**public** **int** toDecimal(String romanNumber) **throws** InvalidNumberException;

**public** String toRoman(**int** decimalNumber) **throws** InvalidNumberException;

}

Input space partitioning for toRoman() :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Characteristics | B1 | B2 | B3 | B4 |
| value of decimalNumber | <0 | =0 | 1<=x<=4000 | >4000 |
| Test: romanNumber = | -1 | 0 | 45 | 4001 |

Input space partitioning for toDecimal() :

|  |  |  |  |
| --- | --- | --- | --- |
| Characteristics | B1 | B2 | B3 |
| Type of romanNumber | Null | Empty string (string length = 0 ) | Length >=1 |

Block B3 can be subpartitioned according to the following

|  |  |  |
| --- | --- | --- |
| Characteristics | B4 | B5 |
| Contains invalid character + Length >=1 | Yes | No |

Test case for block B1, romanNumber = null;

Test case for block B2, romanNumber = “”;

Test case for block B4, romanNumber = GXI;

Test case for block B5, romanNumber = XII;