Problem 1

Manideepika Reddy Myaka

a. **Describe the input domain for the intersection() method.**
The input domain for the intersection() method contains two sets, s1 and s2. Each set can take on the following possible values:
   1. null, which is treated as an empty set.
   2. An empty set ({}).
   3. A non-empty set, which may contain one or more elements of any type (e.g., integers, strings, objects).

b. **Does the partition "Type of s1" satisfy the completeness property? If not, give a value for s1 that does not fit in any block.**
No, the partition "Type of s1" does not fully satisfy the completeness property. The existing blocks are:
   1. Block 1: s1 = null
   2. Block 2: s1 = {} (empty set)
   3. Block 3: s1 has at least one element

   A set where s1 contains null as an element, i.e., s1 = {null}. This case does not fit in any of the existing blocks.

c. **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, the partition "Type of s1" satisfies the disjointness property. Each block is clearly defined and there are no overlapping conditions. For example:
   1. A set cannot be both null and {} (empty).
   2. A set with at least one element cannot be considered empty.

d. **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 partition "Relation between s1 and s2" does not satisfy the completeness property. The current blocks are:
   1. Block 1: s1 and s2 represent the same set
   2. Block 2: s1 is a subset of s2

3. Block 3: s2 is a subset of s1
4. Block 4: s1 and s2 do not have any elements in common

These blocks miss a case where both s1 and s2 contain the same elements plus null. For example, {null, 1} and {null, 2} do not fit in any of these blocks.

e. Does the partition "Relation between s1 and s2" satisfy the disjointness property? If not, give a pair of values that fits in more than one block.

No, the partition "Relation between s1 and s2" does not satisfy the disjointness property. For example:

1. If s1 is {1, 2} and s2 is {1}, both "Block 2: s1 is a subset of s2" and "Block 3: s2 is a subset of s1" apply.

f. Change the blocks for the partitions "Type of s1" and "Relation between s1 and s2" such that they do not suffer from any disjointness or completeness problems, but do not just use partitions with two blocks (i.e., blocks that are effectively "true" and "false").

Type of s1:

1. Block 1: s1 = null
2. Block 2: s1 = {} (empty set)
3. Block 3: s1 contains at least one non-null element
4. Block 4: s1 contains null as an element

Relation between s1 and s2:

1. Block 1: s1 and s2 represent the same set
2. Block 2: s1 is a strict subset of s2 (i.e., all elements of s1 are in s2, but s2 has extra elements)
3. Block 3: s2 is a strict subset of s1
4. Block 4: s1 and s2 share some elements, but neither is a subset of the other
5. Block 5: s1 and s2 have no elements in common

g. Create a partition for Characteristic 2, "Type of s2" which is analogous to "Type of s1".

1. Block 1: s2 = null
2. Block 2: s2 = {} (empty set)
3. Block 3: s2 contains at least one non-null element
4. Block 4: s2 contains null as an element

h. Choose a representative input for each block from the three partitions "Type of s1", "Type of s2", and "Relation between s1 and s2".

Type of s1	Type of s2	Relation between s1 and s2
s1 = null	s2 = null	s1 and s2 are the same (empty sets)
s1 = {}	s2 = {}	s1 and s2 are the same (empty sets)
s1 = {1, 2}	s2 = {1, 2}	s1 and s2 are the same
s1 = {1}	s2 = {1, 2}	s1 is a subset of s2
s1 = {1, 2}	s2 = {1}	s2 is a subset of s1
s1 = {1, 2}	s2 = {3, 4}	s1 and s2 have no common elements
s1 = {null}	s2 = {null, 1}	s1 is a subset of s2
s1 = {1, 2}	s2 = {null, 1, 2}	s2 contains additional null

i. Describe the constraints among the three partitions used in (h) above.

The constraints among the partitions are:

1. When s1 or s2 is null, the method should treat them as empty sets.
2. If both s1 and s2 are empty, the result should also be an empty set.
3. If s1 or s2 contains null, special consideration is needed for elements being compared (ensure null is handled correctly in set operations).
4. When s1 is a subset of s2 or vice versa, the intersection should return only the common elements.

j. Construct test cases that obey the constraints among the three partitions and satisfy the pair-wise coverage criteria.

Pair-Wise Coverage (PWC) Criterion : A value from each block for each characteristic must be combined with a value from every block for all other characteristics.

Each test case should have two input sets and an expected output.

Test Case	Input (s1)	Input (s2)	Expected Output
TC1	null	null	null
TC2	{}	{}	{}
TC3	{1, 2}	{1, 2}	{1, 2}
TC4	{1}	{1, 2}	{1}
TC5	{1, 2}	{1}	{1}
TC6	{1, 2}	{3, 4}	{}
TC7	{null}	{null, 1}	{null}
TC8	{1, 2}	{null, 1, 2}	{1, 2}

k. Complete the intersection method in Sets.java.

```
public static Set intersection(Set s1, Set s2) {
  // Effects: Return a (non null) Set equal to the intersection of sets s1 and s2
  // A null argument is treated as an empty set
        // complete the implemenation below
    if (s1 == null) s1 = new HashSet<>();
    if (s2 == null) s2 = new HashSet<>();
    Set result = new HashSet(s1);
    result.retainAll(s2);
    return result;
  }
```

l. Use your answer in (j) to create JUnit tests for intersection in Test*.java files (such as the provided TestPairwise.java file) in the hw-1-solution/coding-problems Maven project.

```
public class TestPairwise {

  @Test
  public void testBothSetsNull() {
    Set result = Sets.intersection(null, null);
    assertEquals(new HashSet<>(), result);
  }

  @Test
  public void testSetOneNull() {
    Set<Object> s2 = new HashSet<>();
    s2.add(1);
    s2.add(2);
    Set result = Sets.intersection(null, s2);
    assertEquals(new HashSet<>(), result);
  }

  @Test
  public void testSetTwoNull() {
    Set<Object> s1 = new HashSet<>();
    s1.add(1);
    s1.add(2);
```

```
    Set result = Sets.intersection(s1, null);
    assertEquals(new HashSet<>(), result);
}

@Test
public void testBothSetsEmpty() {
    Set<Object> s1 = new HashSet<>();
    Set<Object> s2 = new HashSet<>();
    Set result = Sets.intersection(s1, s2);
    assertEquals(new HashSet<>(), result);
}

@Test
public void testSetsWithSameElements() {
    Set<Object> s1 = new HashSet<>();
    s1.add(1);
    s1.add(2);
    Set<Object> s2 = new HashSet<>();
    s2.add(1);
    s2.add(2);
    Set result = Sets.intersection(s1, s2);
    Set<Object> expected = new HashSet<>();
    expected.add(1);
    expected.add(2);
    assertEquals(expected, result);
}

@Test
public void testFirstSetSubsetOfSecond() {
    Set<Object> s1 = new HashSet<>();
    s1.add(1);
    s1.add(2);
    Set<Object> s2 = new HashSet<>();
    s2.add(1);
    s2.add(2);
    s2.add(3);
    Set result = Sets.intersection(s1, s2);
    Set<Object> expected = new HashSet<>();
```

```
    expected.add(1);
    expected.add(2);
    assertEquals(expected, result);
}

@Test
public void testSecondSetSubsetOfFirst() {
    Set<Object> s1 = new HashSet<>();
    s1.add(1);
    s1.add(2);
    s1.add(3);
    Set<Object> s2 = new HashSet<>();
    s2.add(1);
    s2.add(2);
    Set result = Sets.intersection(s1, s2);
    Set<Object> expected = new HashSet<>();
    expected.add(1);
    expected.add(2);
    assertEquals(expected, result);
}

@Test
public void testNoCommonElements() {
    Set<Object> s1 = new HashSet<>();
    s1.add(1);
    s1.add(2);
    Set<Object> s2 = new HashSet<>();
    s2.add(3);
    s2.add(4);
    Set result = Sets.intersection(s1, s2);
    assertEquals(new HashSet<>(), result);
}

@Test
public void testSetWithNullElement() {
    Set<Object> s1 = new HashSet<>();
    s1.add(null);
    Set<Object> s2 = new HashSet<>();
```

```
        s2.add(null);
        s2.add(1);
        Set result = Sets.intersection(s1, s2);
        Set<Object> expected = new HashSet<>();
        expected.add(null);
        assertEquals(expected, result);
    }
}
```