SWE 619 Assignment 7

- *Amish Papneja*
- *Avinash Arunachalam A Murugappan*
- *Rushil Nandan Dubey*

Goal: Polymorphic Abstraction.

Assignment:

A Comparator based on absolute values is problematic. Code up the comparator and then write client code that illustrates the problem. Use a Java 8 *lambda function* implement the comparator. Explain what is wrong in a brief summary statement. Your explanation of the problem must be phrased in terms of a violation of the contract for Comparator.

To emphasize that this contract problem is real, your code should create two Java sets, one a HashSet, and the other a TreeSet. The TreeSet should order items with your absolute value comparator. Your example should add the same integers to both sets, yet still end up with sets that are different. Your summary statement should explain why.

As for other recent assignments, your deliverable is a clear, concise story that demonstrates the completion of the assignment.

1. Writing MyComparator:

We will be writing a class Mycomparator which extends Comparator<> interface. In order to implement it, we need to implement the compare() method with the parameters

```
class MyComparator implements Comparator<Integer>{
   @Override
   public int compare(Integer i1, Integer i2) throws NullPointerException,
ClassCastException {
       return Math.abs(i1.intValue()) - Math.abs(i2.intValue());
   }
}
```

2. Writing compare() method using lambda function:

We implemented the AbsoluteValueComparator above in our classical Java method. However, Comparator is a FunctionalInterface which means it has only one abstract method. This enables us to code up the comparator in a much more concise manner, So we used lambda function.

```
Comparator<Integer> comp = (i1, i2)->Math.abs(i1) - Math.abs(i2);
```

Client code to prove it false:
Now we write the test cases to prove that it is false in the AbsoluteValueComparator. In this, we create instances of a TreeSet and HashSet and add the exact same elements in those 2 sets. Since we add the same elements, it should return the exact same set, however this is where the AbsoluteValueComparator comes into play and highlights this assignment needs.

```
Set<Integer> treeSet, hashSet;
  Comparator<Integer> comp;

  public void test_before() {
    comp = (i1, i2) -> Math.abs(i1) - Math.abs(i2);

    treeSet = new TreeSet<Integer>( (i1, i2)  -> Math.abs(i1) - Math.abs(i2));
    hashSet = new HashSet<Integer>();

    treeSet.add(1);
    treeSet.add(2);
    treeSet.add(-90);
    treeSet.add(7);
    treeSet.add(11);
    treeSet.add(-11);
    treeSet.add(-23);


    hashSet.add(1);
    hashSet.add(2);
    hashSet.add(-90);
    hashSet.add(7);
    hashSet.add(11);
    hashSet.add(-11);
    hashSet.add(-23);
  }
```

Test Cases:

TEST 1(Comparison):

```
@Test
  public void test_after() {
    assertNotEquals(treeSet, hashSet);
 }// this test passed with the above implementation
PASS
```

TEST 2 (Voilation Checker):

The AbsoluteValueComparator, gives wrong output, because it violates the Comparator contract,
The contract says as follows:
"It follows immediately from the contract for compare that the quotient is an *equivalence relation* on S, and that the imposed ordering is a *total order* on S. When we say that the ordering imposed by c on S is *consistent with equals*, we mean that the quotient for the ordering is the equivalence relation defined by the objects' equals(Object) method(s):
{(x, y) such that x.equals(y)}."

This means that a customly created Comparator abides by its contract if it satisfies the equivalence relation between compare(x,y) and x.equals(y).

In mathematical terms,

$$\text{compare}(x,y) == 0 \Leftrightarrow x.\text{equals}(y)$$

How so ever, our AbsoluteValueComparator returns 0 even after comparison between unequal elements.

```
@Test
   public void test_violation() {
      Integer a = 10;
      Integer b = -10;
      assertEquals(0, comp.compare(a, b));
      assertFalse(a.equals(b));

}// This test also passes
```

TEST 3 (NullPointerException Checker):
The compare method does not allow null elements to be compared and throws NullPointerException if it encounters any null elements passed as its parameters, so this test is implemented to check the same.

```
   @Test
(expected = NullPointerException.class)
   public void test_exception(){
      Integer a = null;
```

```
        Integer b = 3;
        comp.compare(a, b);
    } // This test passes
```

CONCLUSION:

Therefore, The AbsoluteValueComparater using the classical format and using lambda expressions was implemented along with the client code. We tested the client code which implied that the AbsoluteValueComparater did not abide by the contract.

Contributions:

Amish Papneja (G01211503)

- - *Implemented comparator()*
- - *Wrote the story for the above methods*

Avinash Arunachalam A Murugappan (G01163980)

- - *Implemented client code*
- - *Wrote story for client code*

Rushil Nandan Dubey (G01203932)

- - *Implemented tests*
- - *Wrote story for the tests*