SWE 619 Assignment 11

// TODO

Goal: Favoring composition over inheritance. Bloch, Item 18.

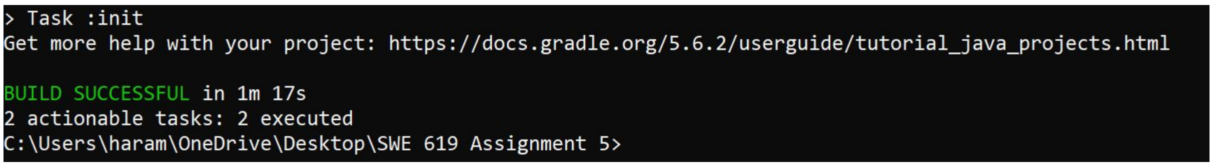
Consider the InstrumentedSet example from Bloch Item 18 (as well as in-class exercise #10A).

1. Replace Set with List. There is no problem with equals(). Why not?
2. Replace Set with Collection. Now equals() does not satisfy its contract.
    - Explain why there is a problem.
    - Demonstrate the problem with a suitable JUnit test.

The GTA will look for correct responses, appropriate JUnit tests, and plausible explanations when doing the grading.

We need to create a few classes and write JUnit tests to test the corresponding classes. Hence, I first created a **new gradle application** as follows:

- Open command prompt
- `cd` to the directory where you want to initialise your project
- Type `gradle init`
- Follow the onscreen instructions and Gradle should create a workspace for you



```
> Task :init
Get more help with your project: https://docs.gradle.org/5.6.2/userguide/tutorial_java_projects.html

BUILD SUCCESSFUL in 1m 17s
2 actionable tasks: 2 executed
C:\Users\haram\OneDrive\Desktop\SWE 619 Assignment 5>
```

Now, we have our project ready to roll!

LET'S GET STARTED!!!

1. Replacing Set with List:

We first refactor the ForwardingSet and InstrumentedSet classes to accommodate List functionality. Hence the two new classes we create are ForwardingList and InstrumentedList

ForwardingList.java:

```
public class ForwardingList<E> implements List<E> {
    private final List<E> s;

    public ForwardingList(List<E> s) {
        this.s = s;
    }

    public boolean add(E e) {
        return s.add(e);
    }

    public boolean remove(Object o) {
        return s.remove(o);
    }

    @Override
    public boolean equals(Object o) {
        return s.equals(o);
```

```java
    }

    @Override
    public int hashCode() {
        return s.hashCode();
    }

    @Override
    public String toString() {
        return s.toString();
    }

    @Override
    public int size() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public boolean isEmpty() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean contains(Object o) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Iterator<E> iterator() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object[] toArray() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public <T> T[] toArray(T[] a) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean containsAll(Collection<?> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean addAll(Collection<? extends E> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean addAll(int index, Collection<? extends E> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean removeAll(Collection<?> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean retainAll(Collection<?> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public void clear() {
        // TODO Auto-generated method stub

    }

    @Override
    public E get(int index) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public E set(int index, E element) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void add(int index, E element) {
        // TODO Auto-generated method stub

    }

    @Override
    public E remove(int index) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int indexOf(Object o) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int lastIndexOf(Object o) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public ListIterator<E> listIterator() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public ListIterator<E> listIterator(int index) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public List<E> subList(int fromIndex, int toIndex) {
        // TODO Auto-generated method stub
        return null;
    }

    // Other forwarded methods from Set interface omitted


}
```

InstrumentedList.java

```java
public class InstrumentedList<E> extends ForwardingList<E> {
    private int addCount = 0;

    public InstrumentedList(List<E> s) {
        super(s);
    }

    @Override
    public boolean add(E e) {
        addCount++;
        return super.add(e);
    }

    public int getAddCount() {
        return addCount;
    }
}
```

2. Testing InstrumentedList:

Now, the task assigned is to test the equals method for this implementation and verify its correctness with respect to contract satisfaction.
We write the following piece of code to test the above implementation:

```java
    // Test for satisfactory implementation of equals method when
replacing Set with List
    @Test
    public void InstrumentedListTest() {
        List<String> l1 = new ArrayList<String>();
        l1.add("Apple");
        l1.add("Ball");

        List<String> l2 = new InstrumentedList<String>(l1);
        l2.add("Apple");
        l2.add("Cat");
        List<String> l3 = new InstrumentedList<String>(l2);
        l3.add("Dog");

        l1.remove("Ball");
        l2.remove("Apple");

        System.out.println(l1);
        System.out.println(l2);
        System.out.println(l3);

        // Transitivity, symmetry properties maintained -> Equals
contract satisfied by
        // List
        assertTrue(l1.equals(l2));
        assertTrue(l2.equals(l3));
        assertTrue(l3.equals(l1));
        assertTrue(l2.equals(l1));
    }
```

After running these tests, we confirm that the equals method contract is satisfied for this implementation, since the properties of transitivity and symmetry are maintained, thus maintaining the contract. Also, even under forwarding, it goes to check whether both objects are Lists.

3. Replacing Set with Collection:

Now we replace Set with Collection in a similar fashion to create two new classes - ForwardingCollection and InstrumentedCollection.

ForwardingCollection.java

```
public class ForwardingCollection<E> implements Collection<E> {
    private final Collection<E> c;

    public ForwardingCollection(Collection<E> col) {
        this.c = col;
    }

    public boolean add(E e) {
```

```java
        return c.add(e);
    }

    public boolean remove(Object o) {
        return c.remove(o);
    }

    @Override
    public boolean equals(Object o) {
        return c.equals(o);
    }

    @Override
    public int hashCode() {
        return c.hashCode();
    }

    @Override
    public String toString() {
        return c.toString();
    }

    @Override
    public int size() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public boolean isEmpty() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean contains(Object o) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Iterator<E> iterator() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object[] toArray() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public <T> T[] toArray(T[] a) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean containsAll(Collection<?> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean addAll(Collection<? extends E> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean removeAll(Collection<?> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean retainAll(Collection<?> c) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public void clear() {
        // TODO Auto-generated method stub
    }
 }
```

InstrumentedCollection.java

```
public class InstrumentedCollection<E> extends
ForwardingCollection<E> {
    private int addCount = 0;

    public InstrumentedCollection(Collection<E> s) {
        super(s);
    }

    @Override
    public boolean add(E e) {
        addCount++;
        return super.add(e);
    }

    public int getAddCount() {
        return addCount;
    }
}
```

4. Testing InstrumentedCollection

The task assigned to us for this implementation is to verify the fallacy in the equals method implementation for this new class formed after replacing Set with Collection.
Hence we write the following JUnit test to test this fallacy:

```
    // Test for violation of equals method contract due to broken
symmetry property when replacing Set with Collection
    @Test
    public void InstrumentedCollectionTest() {
        Collection<String> c1 = new ArrayList<>();
        c1.add("Apple");
        c1.add("Ball");

        Collection<String> c2 = new
InstrumentedCollection<String>(c1);
        c2.add("Apple");
        c2.add("Cat");

        Collection<String> c3 = new
InstrumentedCollection<String>(c2);
        c3.add("Dog");

        c1.remove("Ball");
        c2.remove("Apple");

        // Symmetry is broken here
        // parent.equals(child) != child.equals(parent)

        assertFalse(c1.equals(c2));
        assertTrue(c2.equals(c1));
    }
```

Here, parent.equals(child) returns false, however child.equals(parent) returns true, which breaks the symmetry property. In the above case, c1.equals(c2) calls the equals method for ArrayList and it returns false because Collection is a parent class and not all parents are the same as their children. However, c2.equals(c1) calls the equals method for InstrumentedCollection but since c2 contains c1 (since that is how we instantiate it), it will actually dynamically dispatch and call c1's equals method which will return true.

Conclusion:

Thus we replaced Set with List and Collection and tested for satisfactory implementation of equals method. However, after running JUnit tests, we came to the conclusion that replacing Set with List doesn’t really fiddle with the equals contract but when we replace it with Collection, it creates an issue due to dynamic dispatching.