SWE619 Assignment-10

-Amish Papneja

-Avinash Arunachalam A Murugappan

- Rushil Nandan Dubey

Assignment Question:
Goal: Object class contracts.

As it happens, Liskov's implementation of clone() for the IntSet class (see figure 5.10, page 97) is wrong.

- Use the version of IntSet from the in-class exercise. Implement a subtype of IntSet to demonstrate the problem. Your solution should include appropiate executable code in the form of JUnit tests.
- Provide a correct implementation of clone() for IntSet. Again, give appropriate JUnit tests.
- Correctly override hashCode() and equals(). As discussed in the class exercise, the standard recipe is not appropriate in this (unusual) case.
- In addititon to code and tests, your deliverable is a **story**. Explain what is going on at each stage of the exercise. The GTA will primarily grade your story.

Implementation of IntSet Class:

1. Class Members and Constructor:

```
public class IntSet implements Cloneable {
  private List<Integer> els;

   public IntSet () { els = new ArrayList<Integer>(); }

  private IntSet (List<Integer> list) { els = list; }
```

IntSet class consists of a private List<Integer> data type named els which contains all the Integers passed in the constructor. The Non-parametrized constructor creates a new arrayList<Integer> which is assigned to els. The parameterized constructor assigns the list passed in it to els.

2. equals(Object obj) method:

```
@Override public boolean equals(Object obj) {
    if (!(obj.getClass()==this.getClass()))
       return false;
    IntSet s = (IntSet) obj;
    return els.equals(s.els);
```

```
   }
```

Our version of the equals method checks for the class type of the object which is passed as a parameter. If it is not the same as the calling (this) object it returns false otherwise the passed object obj is casted and assigned to a new object s. In the end the List of Integer(els) is compared for both the objects and returned true or false.

3. hashCode() method:

```
@Override public int hashC0ode() {
      int result = 0;
      for (Integer i : els) {
//          result += i.hashCode();  // from class
          result += 31 * result + i.hashCode();  // bloch
      }
      return result;
   }
```

If a client wants to store our object in a container, such as Hash Maps, and we override equals(), then we must also override hashCode(). Returning a value of 42 was the default return value, which is correct, but a terrible choice because it ensures that every object hashes to the same bucket. This results in a linked list, which results in programs that run in quadratic time instead of linear time. Our implementation of hashCode() leverages Bloch's recipe for hashCode() and the implementation discussed in class.

4. clone() method:

```
   // previous implementation
//   @Override public IntSet clone() {
//       return new IntSet (new ArrayList<Integer>(els));
//   }


   // our implementation
   @Override public IntSet clone() {
      try {
          IntSet s = (IntSet) super.clone();
          s.els.addAll(els);
          return s;
      }catch(CloneNotSupportedException e) {
          throw new IllegalStateException();
      }
   }
```

The problem with the clone method is that it doesn't allow for any subtypes.
The current implementation of it is correct if we don't create any.
If we call the method from the subtype (currently commented out in IntSet),

it would raise a ClassCastException (as shown in the failing test testOldClone)
due to the fact that it creates a new IntSet instead.

By changing the implementation of clone to call super's clone method, the exception doesn't occur. It would create a clone of the super (in this case is Object) which is cast to an IntSet.
Then we would only need to copy over any necessary values, which in this case is els.

Implementation of IntSetSub Class:

1. Class Members and Constructors:

```
public class IntSetSub extends IntSet{

   public IntSetSub() {
      super();
   }

}
```

TEST CASES:

1. Test 1:

```
@Test
   public void testOldClone() {
      IntSetSub set = new IntSetSub();
      IntSetSub sub =(IntSetSub) set.clone();
      assertTrue(sub!=set);
      assertEquals(set.getClass(),sub.getClass());

   }
```

This test is to show that the clone method is not supported by any subclass. We create an object of the IntSetSub class named sub. After that, we try to clone it in a new object named sub. ClassCastException is the reason for (sub!=set) to be True in this test case. Whereas getClass for the object and the clone are the same.

2. Test 2:

```
 @Test
   public void testClone() {
      IntSet set = new IntSet();
      IntSet clone = set.clone();

      assertTrue(set!=clone);
```

```
        assertEquals(set.getClass(),clone.getClass());
    }
```

This test is to show that the clone is implemented to call the super's clone method, so the exception will not occur. Where the clone created of the super's will cast to inset.

3. Test 3:

```
 @Test
    public void testEquals() {
        IntSet set = new IntSet();
        IntSet clone = set.clone();

        //reflexive
        assertTrue(set.equals(set));

        // symmetry
        assertTrue(set.equals(clone));
        assertTrue(clone.equals(set));

        // transitivity
        IntSet clone2 = clone.clone();
        assertTrue(set.equals(clone2));
        assertTrue(clone2.equals(clone));
    }
```

This test if for the correctness of the equals method implementation. An object of IntSet is created named set. A new object clone which is initialized via set.clone(). Now we have checked multiple cases where equals can fail but every test passes. Our implementation of the equal supports reflexive property (*set.equals(set)* , symmetry property (*(clone.equals(set))* and transitive property (a new object clone2 is initialized as clone.clone() and *(clone2.equals(clone))* comes out to be true).

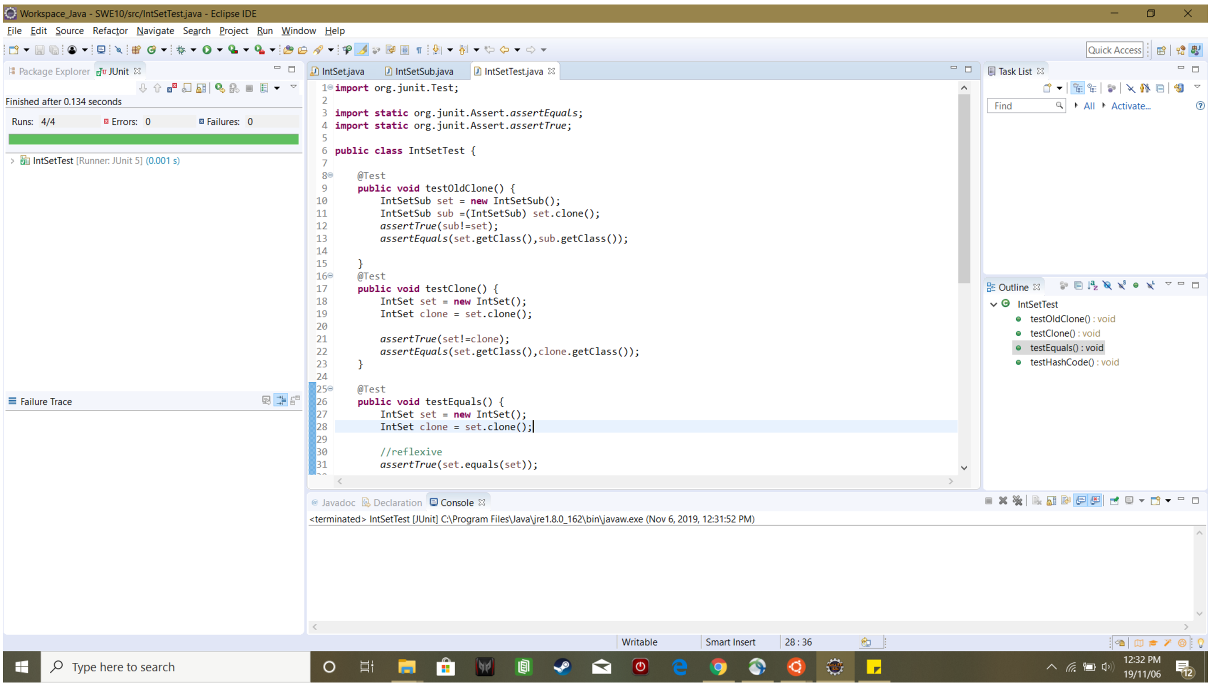
4. Test 4:

```
    @Test
    public void testHashCode() {
        IntSet set = new IntSet();
        IntSet clone = set.clone();

        // reflexive
        assertTrue(set.hashCode() == set.hashCode());

        // symmetry
        assertEquals(set.hashCode(), clone.hashCode());

        // transitivity
        IntSet clone2 = clone.clone();
        assertEquals(set.hashCode(), clone2.hashCode());
        assertEquals(clone2.hashCode(), clone.hashCode());
    }
```

This test is for the correctness of the hasCode() function. A new object of IntSet type is created named set. ANother object clone which is initialzed as set.clone(). We have checked multiple cases where hashCode implementation can fail but it passes all the times. The first check is reflexive where we assertTrue the hashcodes of set and clone. The second check is for symmetry property where we assertTrue(*set.hashCode(), clone.hashCode()*). The Third check is for transitivity where we create a new object clone which is initialized as clone.clone() and *assertEquals(clone2.hashCode(), clone.hashCode());* comes out to be true as well.

Code Screenshots and Test Case Results:

1. IntSetTest.java



Workspace_Java - SWE10/src/IntSetTest.java - Eclipse IDE

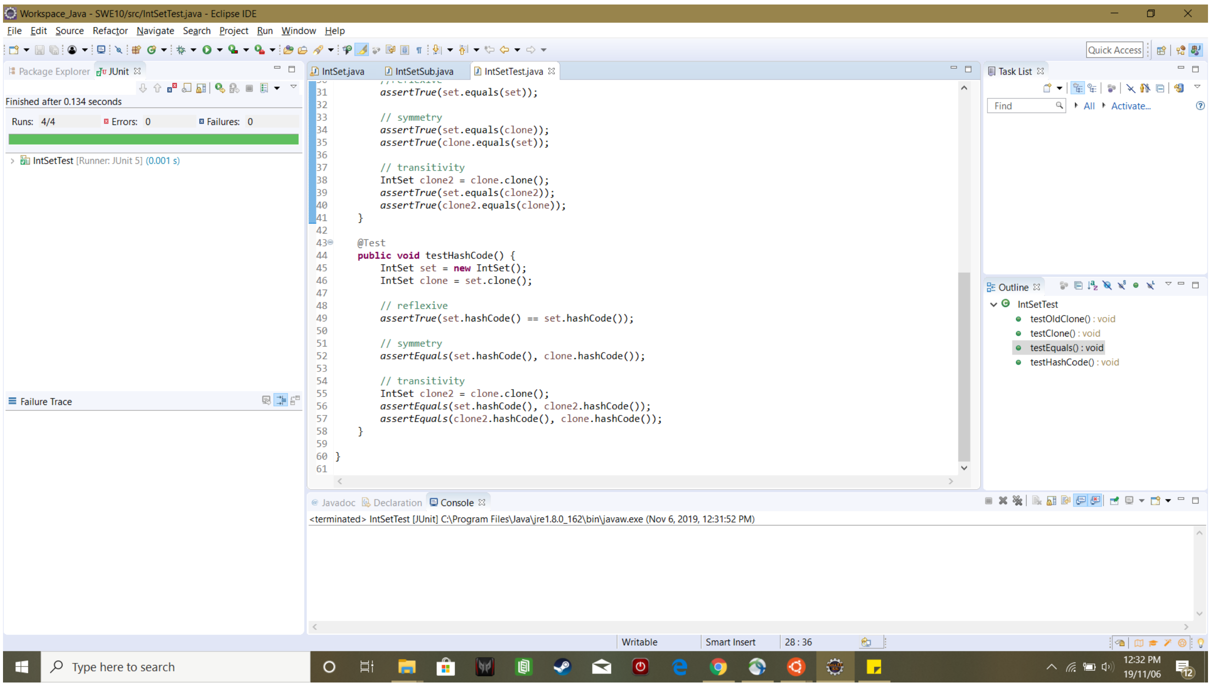
Finished after 0.134 seconds

Runs: 4/4 Errors: 0 Failures: 0

IntSetTest [Runner: JUnit 5] (0.001 s)

```java
import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class IntSetTest {

    @Test
    public void testOldClone() {
        IntSetSub set = new IntSetSub();
        IntSetSub sub =(IntSetSub) set.clone();
        assertTrue(sub!=set);
        assertEquals(set.getClass(),sub.getClass());

    }
    @Test
    public void testClone() {
        IntSet set = new IntSet();
        IntSet clone = set.clone();

        assertTrue(set!=clone);
        assertEquals(set.getClass(),clone.getClass());
    }

    @Test
    public void testEquals() {
        IntSet set = new IntSet();
        IntSet clone = set.clone();

        //reflexive
        assertTrue(set.equals(set));
```

```
<terminated> IntSetTest [JUnit] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (Nov 6, 2019, 12:31:52 PM)
```



```java
        assertTrue(set.equals(set));

        // symmetry
        assertTrue(set.equals(clone));
        assertTrue(clone.equals(set));

        // transitivity
        IntSet clone2 = clone.clone();
        assertTrue(set.equals(clone2));
        assertTrue(clone2.equals(clone));
    }

    @Test
    public void testHashCode() {
        IntSet set = new IntSet();
        IntSet clone = set.clone();

        // reflexive
        assertTrue(set.hashCode() == set.hashCode());

        // symmetry
        assertEquals(set.hashCode(), clone.hashCode());

        // transitivity
        IntSet clone2 = clone.clone();
        assertEquals(set.hashCode(), clone2.hashCode());
        assertEquals(clone2.hashCode(), clone.hashCode());
    }

}
```

Workspace_Java - SWE10/src/IntSetTest.java - Eclipse IDE

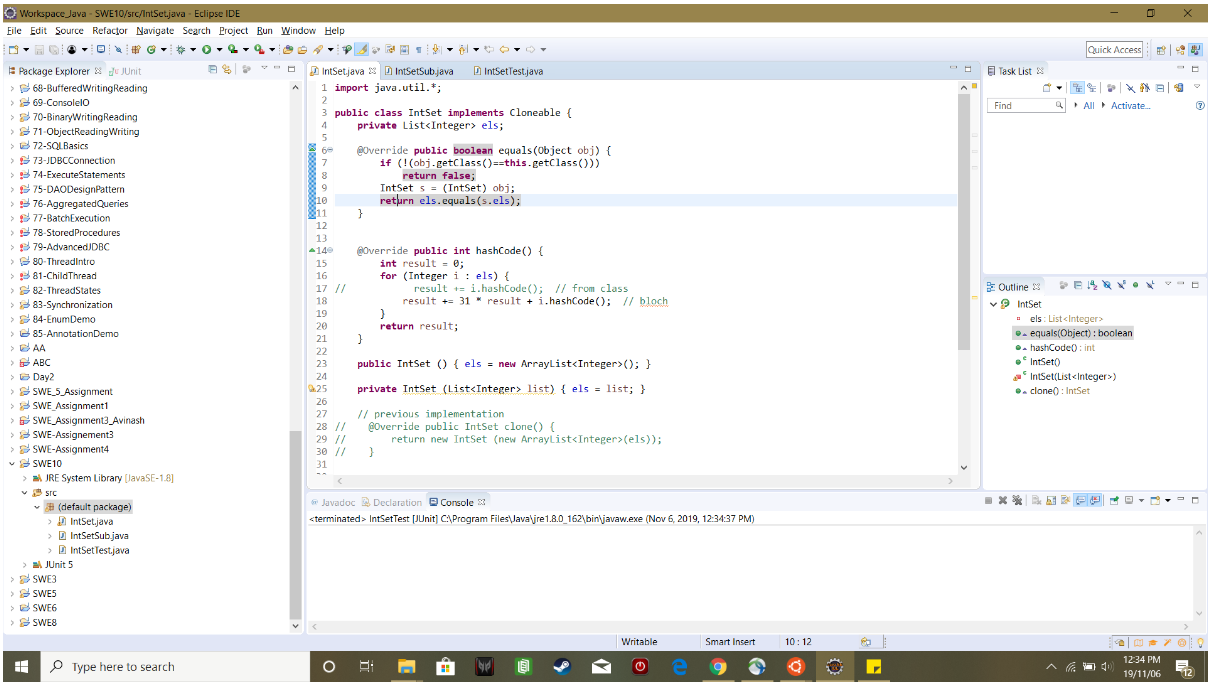
Finished after 0.134 seconds

Runs: 4/4 Errors: 0 Failures: 0

IntSetTest [Runner: JUnit 5] (0.001 s)

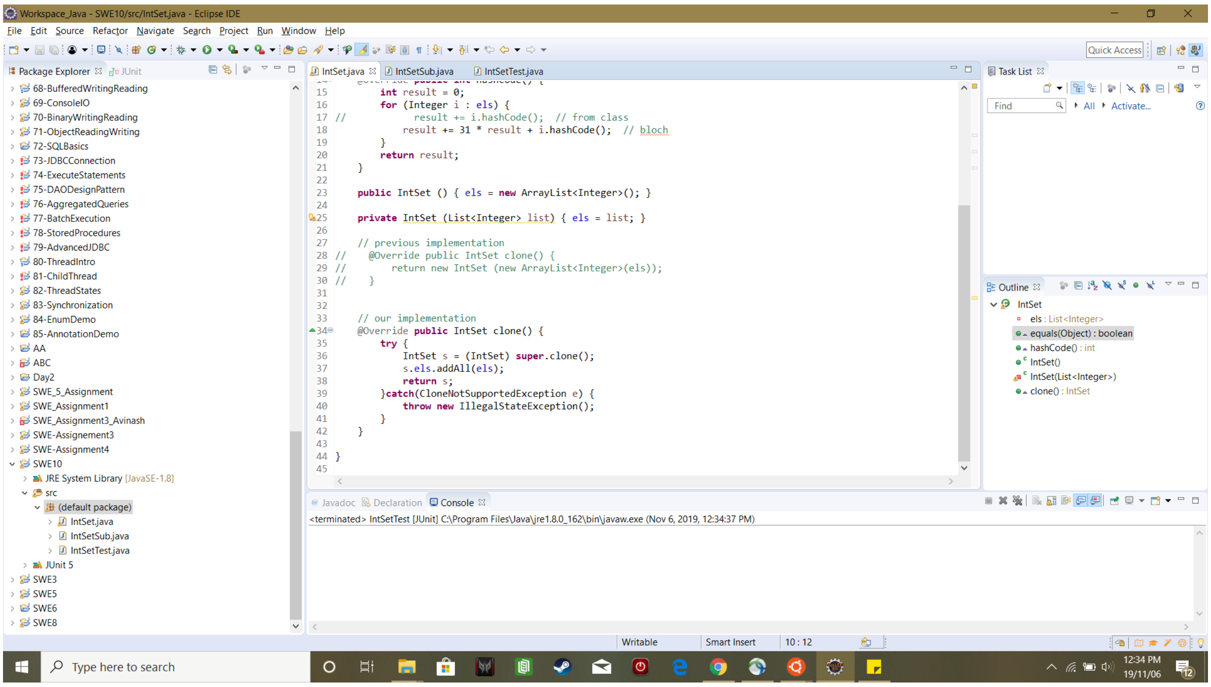
<terminated> IntSetTest [JUnit] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (Nov 6, 2019, 12:31:52 PM)

2. IntSet.java



```java
import java.util.*;

public class IntSet implements Cloneable {
    private List<Integer> els;

    @Override public boolean equals(Object obj) {
        if (!(obj.getClass()==this.getClass()))
            return false;
        IntSet s = (IntSet) obj;
        return els.equals(s.els);
    }


    @Override public int hashCode() {
        int result = 0;
        for (Integer i : els) {
//              result += i.hashCode();  // from class
            result += 31 * result + i.hashCode();  // bloch
        }
        return result;
    }

    public IntSet () { els = new ArrayList<Integer>(); }

    private IntSet (List<Integer> list) { els = list; }

    // previous implementation
//    @Override public IntSet clone() {
//        return new IntSet (new ArrayList<Integer>(els));
//    }

```

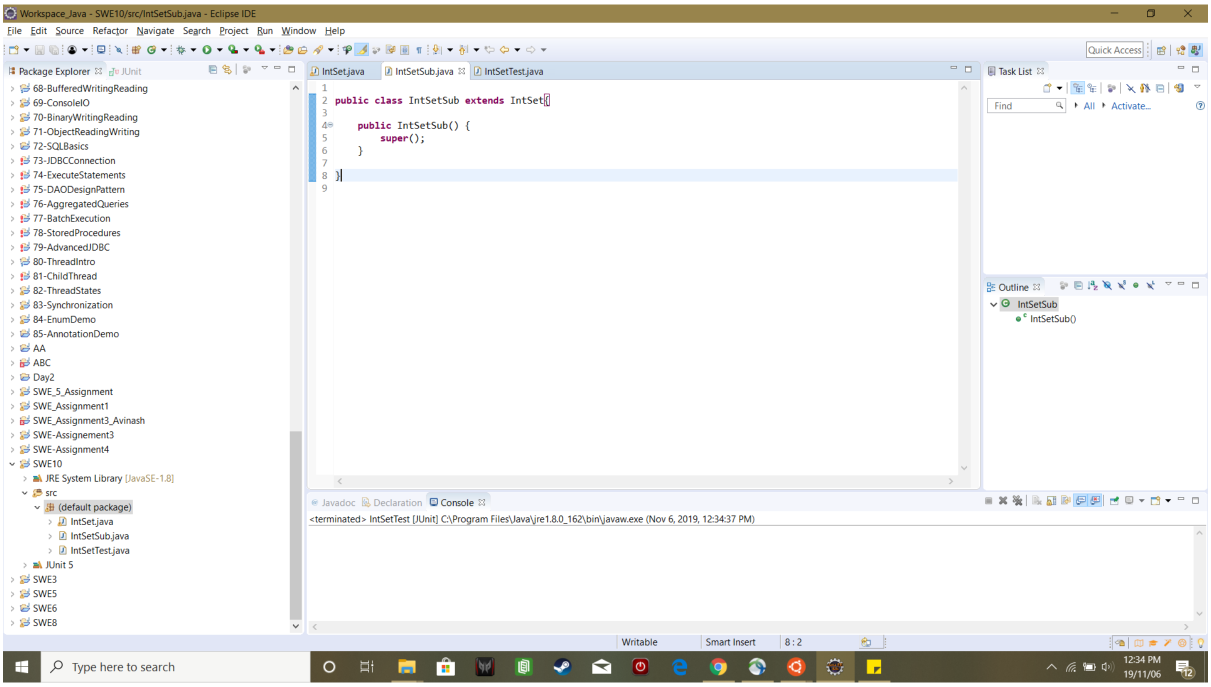
```
<terminated> IntSetTest [JUnit] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (Nov 6, 2019, 12:34:37 PM)
```



```java
        int result = 0;
        for (Integer i : els) {
//              result += i.hashCode();  // from class
            result += 31 * result + i.hashCode();  // bloch
        }
        return result;
    }

    public IntSet () { els = new ArrayList<Integer>(); }

    private IntSet (List<Integer> list) { els = list; }

    // previous implementation
//    @Override public IntSet clone() {
//        return new IntSet (new ArrayList<Integer>(els));
//    }


    // our implementation
    @Override public IntSet clone() {
        try {
            IntSet s = (IntSet) super.clone();
            s.els.addAll(els);
            return s;
        }catch(CloneNotSupportedException e) {
            throw new IllegalStateException();
        }
    }

}
```

```
<terminated> IntSetTest [JUnit] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (Nov 6, 2019, 12:34:37 PM)
```

2. IntSetSub.java:



```java

public class IntSetSub extends IntSet{

	public IntSetSub() {
		super();
	}

}

```

```
<terminated> IntSetTest [JUnit] C:\Program Files\Java\jre1.8.0_162\bin\javaw.exe (Nov 6, 2019, 12:34:37 PM)
```

Contributions

-Amish Papneja

- Implemented clone and test for clone

- Drafted the story for clone and test clone methods

-Avinash Arunachalam A Murugappan

- Implemented Equals method and test for equals

- Drafted the story for equals and test equals methods

- Rushil Nandan Dubey

-Implemented hash code method and test for hash code

- Drafted the story for hash code and test hash methods