Lab 7 Instructions

Testing Ranges by Embedding Invariant Methods

Yuichi Hamamoto

Book Pages Number: 82,83,84,85,86

There are six classes in the zip folder

1. Car.java
2. Gear.java
3. InvariantException.java
4. Moveable.java
5. SparseArray.java
6. Transmission.java

Part 1
Testing Ranges by Embedding Invariant Methods

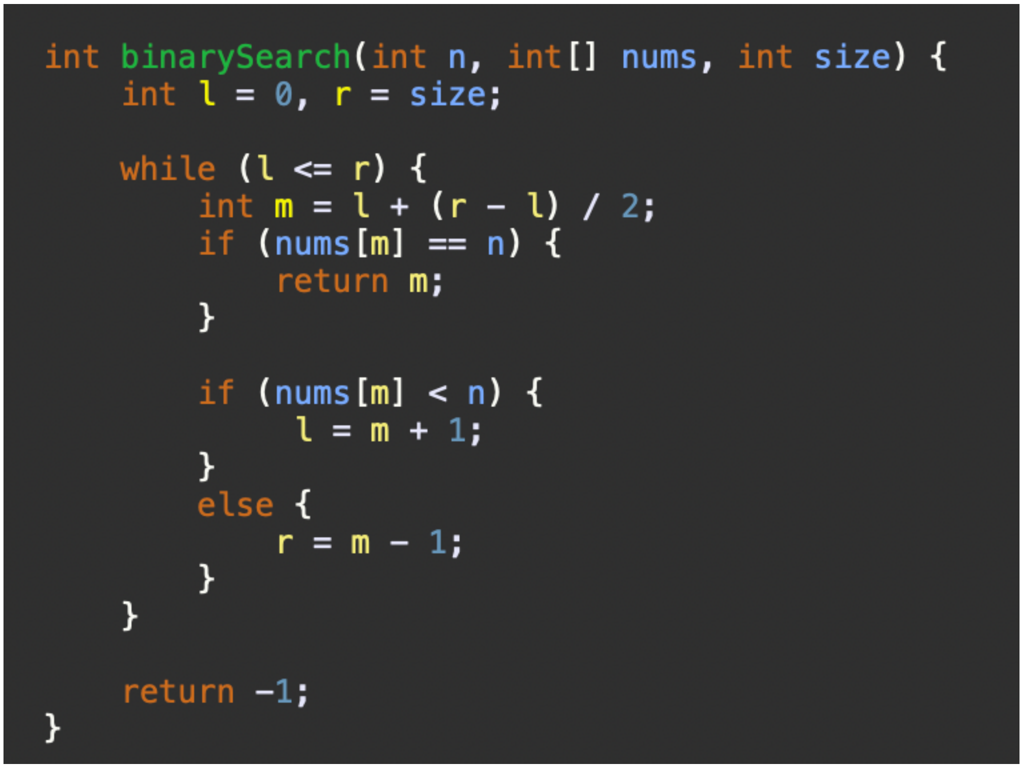
The most common ranges you'll test will likely depend on data-structure concerns, not application-domain constraints.

Let's look at a questionable implementation of a sparse array—a data structure designed to save space. The sweet spot for a sparse array is a broad range of indexes where most of the corresponding values are null. It accomplishes this goal by storing only non-null values, using a pair of arrays that work in concert: an array of indexes corresponds to an array of values.

1- Get the source for the "SparseArray" class from the zip file

2- TODO 1: Implement the iterative Binary search function in SparseArray.java and take screenshot of the code

```
//Returns index of n if it is present in nums else return -1

int binarySearch(int n, int[] nums, int size)
```



```
int binarySearch(int n, int[] nums, int size) {
    int l = 0, r = size;

    while (l <= r) {
        int m = l + (r - l) / 2;
        if (nums[m] == n) {
            return m;
        }

        if (nums[m] < n) {
             l = m + 1;
        }
        else {
            r = m - 1;
        }
    }

    return -1;
}
```

3- **Add** the following line of code snippet to **SparseArray.java**

```
public void checkInvariants() throws InvariantException

{

long nonNullValues = Arrays.stream(values).filter(Objects::nonNull).count();

 if (nonNullValues != size)

throw new InvariantException("size " + size + " does not match value count of " + nonNullValues);

}
```

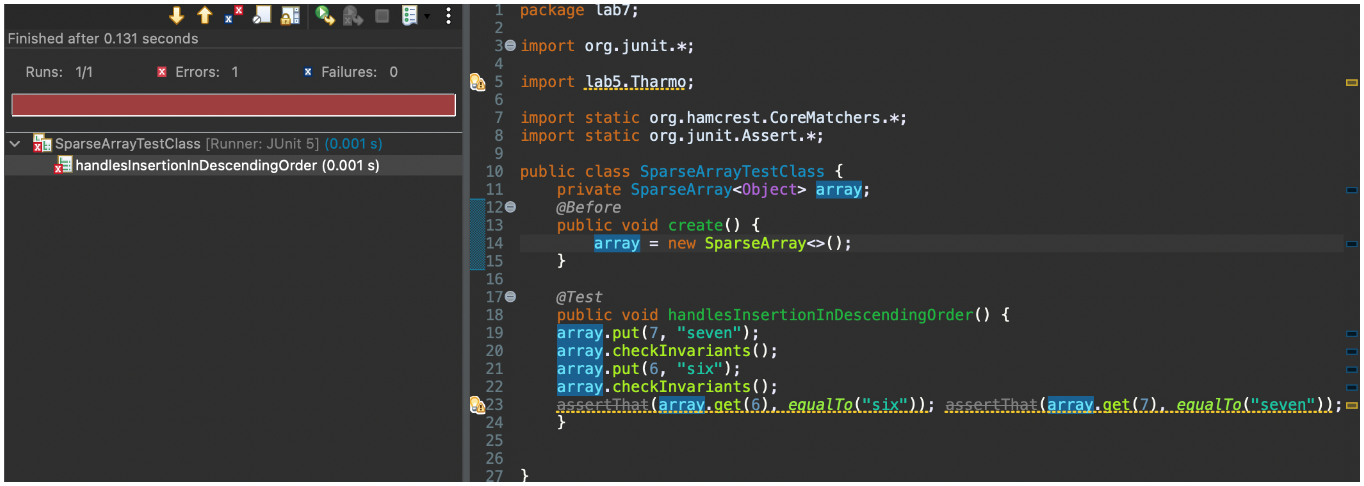
4- TODO 2: Write the following new test class in SparseArrayTestClass.java. Run the code, and take screenshots of the code and test case output

@Test

public void handlesInsertionInDescendingOrder() {

array.put(7, "seven");
array.checkInvariants();
array.put(6, "six");
array.checkInvariants();
assertThat(array.get(6), equalTo("six"));
assertThat(array.get(7), equalTo("seven"));
}

Add @Before annotation. public void create() method sets the new SparseArray() object. Try to create that method with @before annotation creating new object SparseArray<Object> setting it equal to array. Also, think about importing static hamcrest and Assert. (import static org.junit.Assert.*; import static org.hamcrest.CoreMatchers.*;).



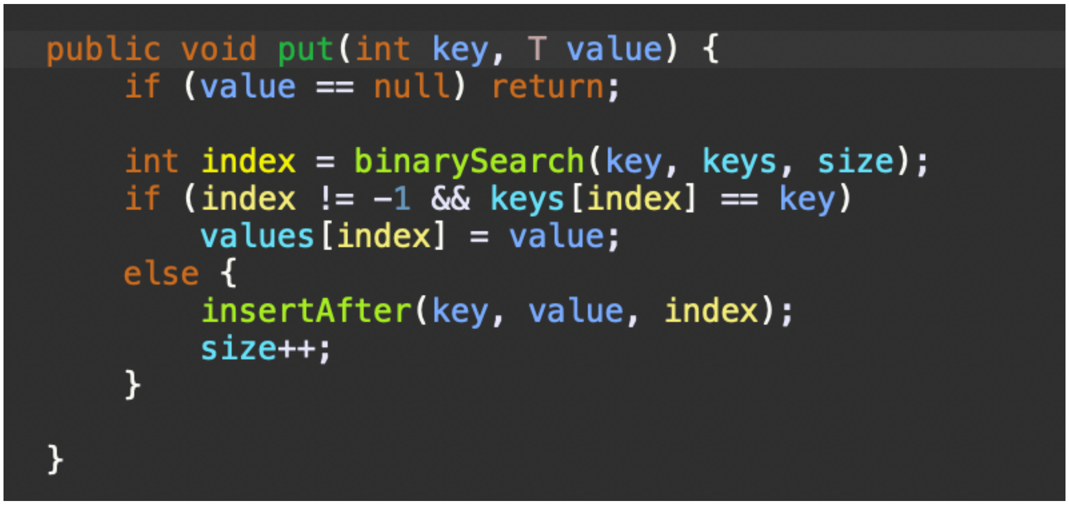
```
Finished after 0.131 seconds

Runs: 1/1        Errors: 1        Failures: 0

SparseArrayTestClass [Runner: JUnit 5] (0.001 s)
    handlesInsertionInDescendingOrder (0.001 s)
```

```java
package lab7;

import org.junit.*;

import lab5.Tharmo;

import static org.hamcrest.CoreMatchers.*;
import static org.junit.Assert.*;

public class SparseArrayTestClass {
    private SparseArray<Object> array;
    @Before
    public void create() {
        array = new SparseArray<>();
    }

    @Test
    public void handlesInsertionInDescendingOrder() {
    array.put(7, "seven");
    array.checkInvariants();
    array.put(6, "six");
    array.checkInvariants();
    assertThat(array.get(6), equalTo("six")); assertThat(array.get(7), equalTo("seven"));
    }


}
```

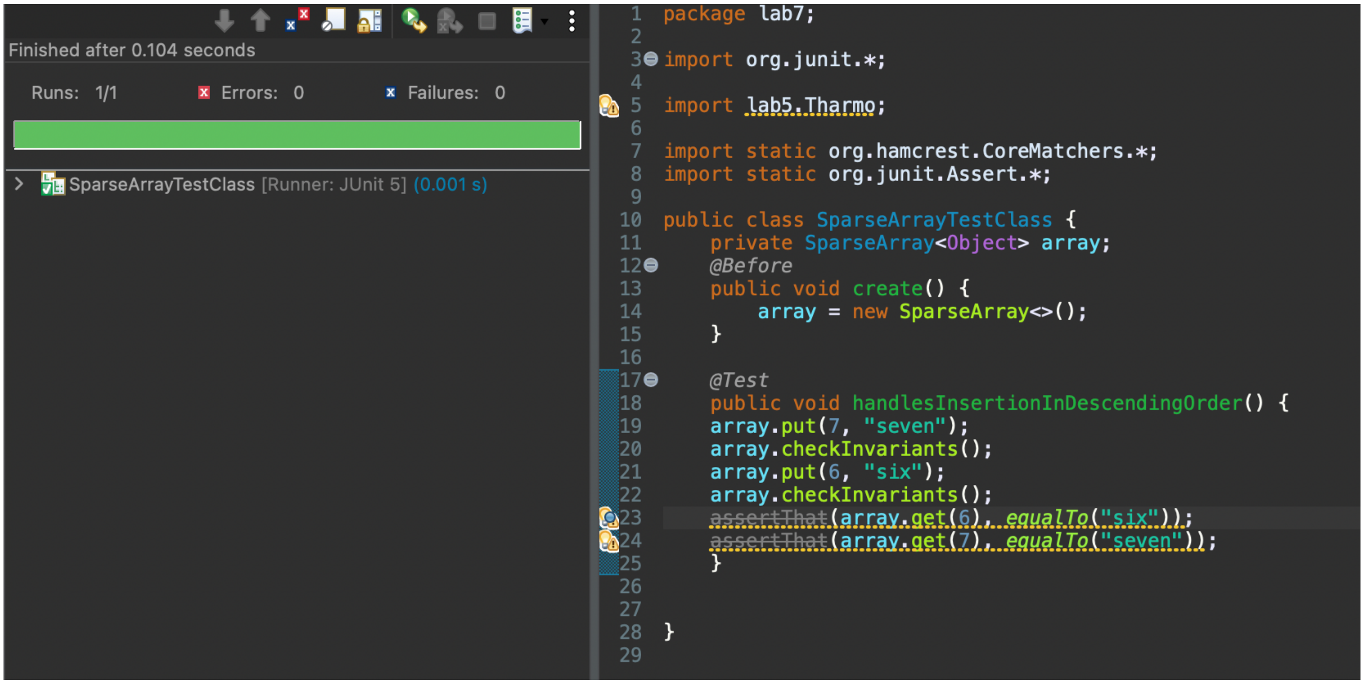
The test errors out with an InvariantException:
util.InvariantException: size 0 does not match value count of 1 at util.SparseArray.checkInvariants(SparseArray.java:48) at util.SparseArrayTest .handlesInsertionInDescendingOrder(SparseArrayTest.java:65) …

Our code indeed has a problem with tracking the internal size.
What is the problem? Answer this in your lab report and fix the code in order to make the test pass.
Take screenshots of the passed test cases and your fixed code and upload it to the lab report.Explain in your lab report what your approach to fix the code
Hint: Take a look at the put method in SparseArray class.

The problem was it didn't update the size. Therefore, I added size ++; after putting a new element in the array.



```java
public void put(int key, T value) {
     if (value == null) return;

     int index = binarySearch(key, keys, size);
     if (index != -1 && keys[index] == key)
          values[index] = value;
     else {
          insertAfter(key, value, index);
          size++;
     }

}
```



Finished after 0.104 seconds

Runs: 1/1 Errors: 0 Failures: 0

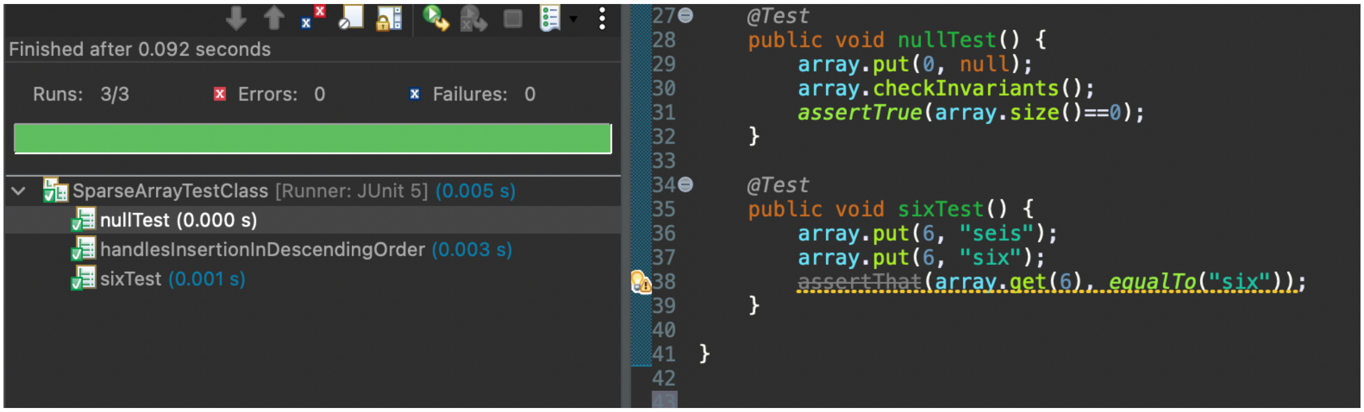
SparseArrayTestClass [Runner: JUnit 5] (0.001 s)

```java
package lab7;

import org.junit.*;

import lab5.Tharmo;

import static org.hamcrest.CoreMatchers.*;
import static org.junit.Assert.*;

public class SparseArrayTestClass {
    private SparseArray<Object> array;
    @Before
    public void create() {
        array = new SparseArray<>();
    }

    @Test
    public void handlesInsertionInDescendingOrder() {
    array.put(7, "seven");
    array.checkInvariants();
    array.put(6, "six");
    array.checkInvariants();
    assertThat(array.get(6), equalTo("six"));
    assertThat(array.get(7), equalTo("seven"));
    }


}
```

5- TODO 3: Write two test cases for the following conditions. Run the 2 test codes, and take screenshots of the code and test cases outputs

- Test case 1 (insert null value): insert (key: 0, value: null) then call checkInvariants method
  - Expected result: array size should equal to 0

- Test case 2 (insert replace value): insert (key: 6, value: “seis”) and insert again with (key: 6, value: “six”)
  - Expected result: array.get(6) should equal to “six”



```
Finished after 0.092 seconds

Runs:  3/3        Errors:  0        Failures:  0

SparseArrayTestClass [Runner: JUnit 5] (0.005 s)
    nullTest (0.000 s)
    handlesInsertionInDescendingOrder (0.003 s)
    sixTest (0.001 s)
```

```
    @Test
    public void nullTest() {
        array.put(0, null);
        array.checkInvariants();
        assertTrue(array.size()==0);
    }

    @Test
    public void sixTest() {
        array.put(6, "seis");
        array.put(6, "six");
        ~~assertThat~~(array.get(6), equalTo("six"));
    }

}
```

Part 2
Correct Reference (Correct Initial State)

When testing a method, consider:

- What it references outside its scope

- What external dependencies it has

- Whether it depends on the object being in a certain state

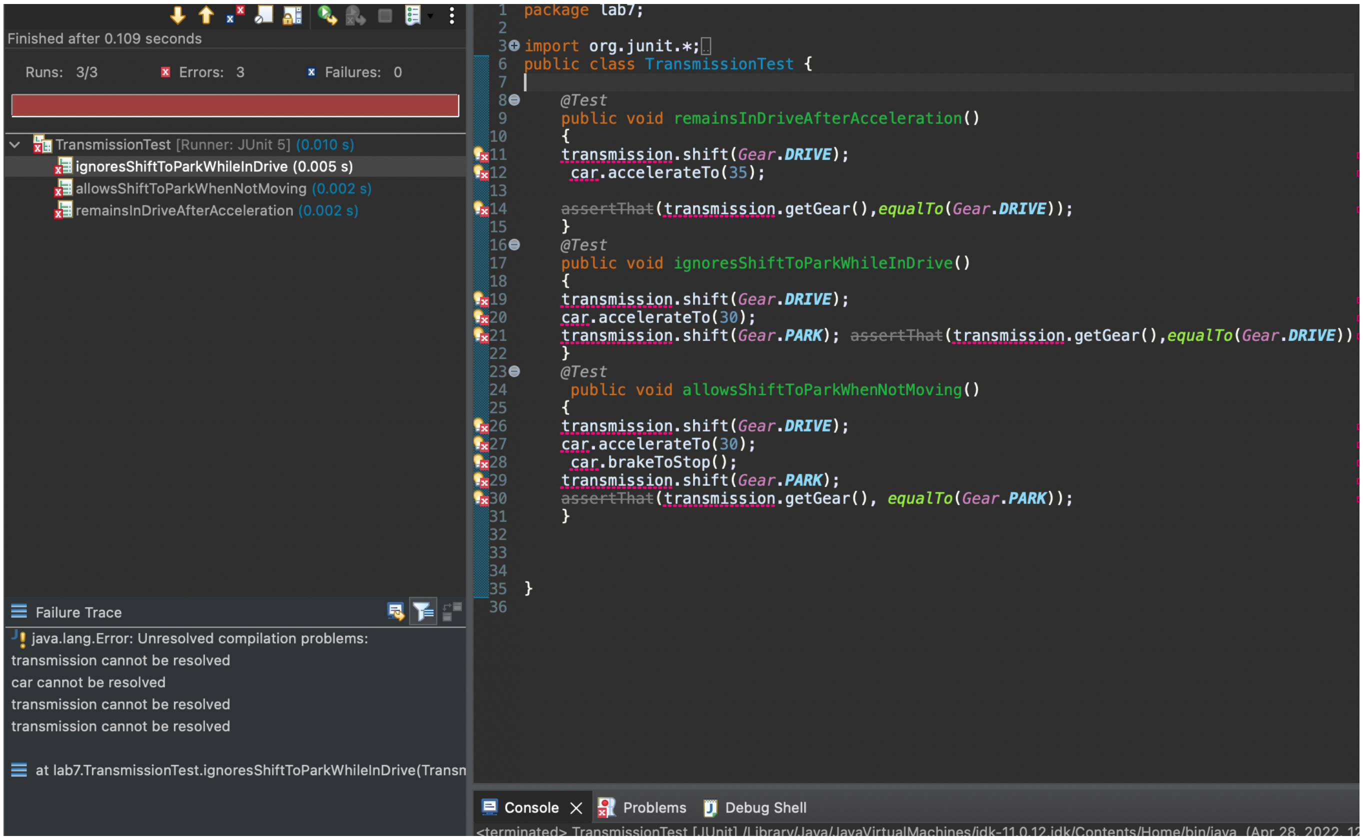
- Any other conditions that must exist

A web app that displays a customer's account history might require the cus- tomer to be logged on. The pop() method for a stack requires a nonempty stack. Shifting your car's transmission from Drive to Park requires you to first stop—if your transmission allowed the shift while the car was moving, it'd likely deliver some hefty damage to your fine Geo Metro.

When you make assumptions about any state, you should verify that your code is reasonably well-behaved when those assumptions are not met. Imagine you're developing the code for your car's microprocessor-controlled transmission. You want tests that demonstrate how the transmission behaves when the car is moving versus when it is not. Our tests for the Transmission code cover three critical scenarios: that it remains in Drive after accelerating, that it ignores the damaging shift to Park while in Drive, and that it *does* allow the shift to Park once the car isn't moving.

6- TODO 4: Inspect and run the following TransmissionTest code. Take a screenshot of the code and output

```
@Test

public void remainsInDriveAfterAcceleration()

{

transmission.shift(Gear.DRIVE);
 car.accelerateTo(35);


assertThat(transmission.getGear(),equalTo(Gear.DRIVE));

}

@Test
public void ignoresShiftToParkWhileInDrive()
{

transmission.shift(Gear.DRIVE);
car.accelerateTo(30);
transmission.shift(Gear.PARK);
assertThat(transmission.getGear(),equalTo(Gear.DRIVE));
}

@Test

 public void allowsShiftToParkWhenNotMoving()
{
 transmission.shift(Gear.DRIVE);
 car.accelerateTo(30);
  car.brakeToStop();
 transmission.shift(Gear.PARK);
 assertThat(transmission.getGear(), equalTo(Gear.PARK));
}
```



Finished after 0.109 seconds

Runs: 3/3   Errors: 3   Failures: 0

```
TransmissionTest [Runner: JUnit 5] (0.010 s)
    ignoresShiftToParkWhileInDrive (0.005 s)
    allowsShiftToParkWhenNotMoving (0.002 s)
    remainsInDriveAfterAcceleration (0.002 s)
```

Failure Trace

```
java.lang.Error: Unresolved compilation problems:
transmission cannot be resolved
car cannot be resolved
transmission cannot be resolved
transmission cannot be resolved

at lab7.TransmissionTest.ignoresShiftToParkWhileInDrive(Transn
```

```java
package lab7;

import org.junit.*;
public class TransmissionTest {

    @Test
    public void remainsInDriveAfterAcceleration()
    {
    transmission.shift(Gear.DRIVE);
     car.accelerateTo(35);

    assertThat(transmission.getGear(),equalTo(Gear.DRIVE));
    }
    @Test
    public void ignoresShiftToParkWhileInDrive()
    {
    transmission.shift(Gear.DRIVE);
    car.accelerateTo(30);
    transmission.shift(Gear.PARK); assertThat(transmission.getGear(),equalTo(Gear.DRIVE))
    }
    @Test
     public void allowsShiftToParkWhenNotMoving()
    {
    transmission.shift(Gear.DRIVE);
    car.accelerateTo(30);
     car.brakeToStop();
    transmission.shift(Gear.PARK);
    assertThat(transmission.getGear(), equalTo(Gear.PARK));
    }



}
```

```
<terminated> TransmissionTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin/java (Apr 28, 2022, 12
```

The *preconditions* for a method represent the state things must be in for it to run. The precondition for putting a transmission in Park is that the car must be at a standstill. We want to ensure that the method behaves gracefully when its precondition isn't met (in our case, we ignore the Park request).

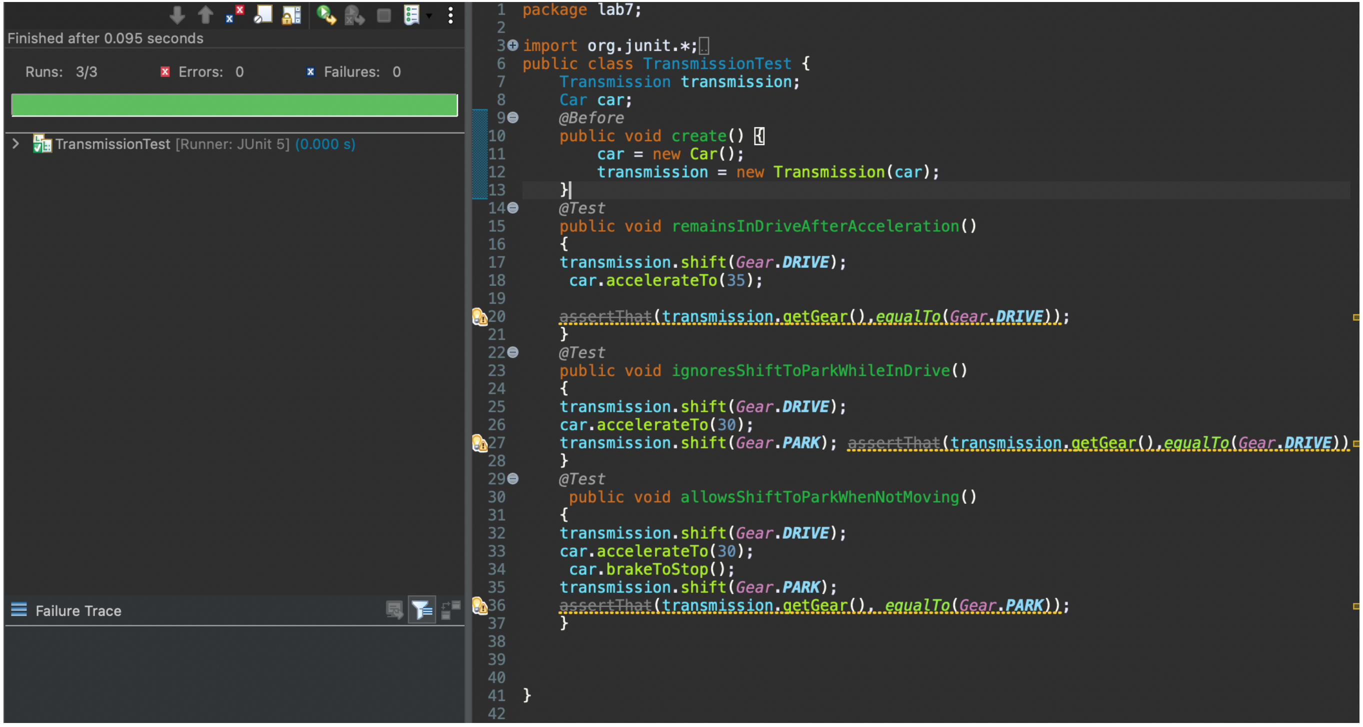
Postconditions state the conditions that you expect the code to make true— essentially, the assert portion of your test. Sometimes this is simply the return value of a called method. You might also need to verify other *side effects*—changes to state that occur as a result of invoking behavior. In the allowsShiftToParkWhenNotMoving test case, calling brakeToStop() on the car instance has the side effect of setting the car's speed to 0

7- - TODO 5: The code above will not run. You will need to fix the test code as follows and take a screenshot:

Add @before annotation and think about public void create() method.

You are creating new Car object and a new transmission where newly created car object should pass in it.

Also, think about importing static hamcrest and Assert.



```
Finished after 0.095 seconds

Runs: 3/3        Errors: 0        Failures: 0

> TransmissionTest [Runner: JUnit 5] (0.000 s)

Failure Trace
```

```java
package lab7;

import org.junit.*;
public class TransmissionTest {
    Transmission transmission;
    Car car;
    @Before
    public void create() {
        car = new Car();
        transmission = new Transmission(car);
    }
    @Test
    public void remainsInDriveAfterAcceleration()
    {
    transmission.shift(Gear.DRIVE);
     car.accelerateTo(35);

    assertThat(transmission.getGear(),equalTo(Gear.DRIVE));
    }
    @Test
    public void ignoresShiftToParkWhileInDrive()
    {
    transmission.shift(Gear.DRIVE);
    car.accelerateTo(30);
    transmission.shift(Gear.PARK); assertThat(transmission.getGear(),equalTo(Gear.DRIVE))
    }
    @Test
     public void allowsShiftToParkWhenNotMoving()
    {
    transmission.shift(Gear.DRIVE);
    car.accelerateTo(30);
     car.brakeToStop();
    transmission.shift(Gear.PARK);
    assertThat(transmission.getGear(), equalTo(Gear.PARK));
    }



}

```

Part 3
Submission

Upload the following:

1. Screenshots of test passing and the answers to the mentioned questions above in the different parts.

   There are the screenshots and answers under each TODO.

2. Answer to “what does checkvariants() method do?”

   It tests that probe at the value stored in the array, and if any invariants fail to hold true, it throws an exception

3. Answer to “what Transmission.Java class does?”

   It checks the current car’s state and then allow us to shift to either DRIVE or PARK. For instance, it will not shift to PARK unless the car is currently not moving.