Exercise 1

Running both the assertTrue and assertFalse makes the test pass and fail. In ecliplise, passing is a checkmark and failing is an x and gives you some details o while it fails.

```
@Test(timeout = 1000)
   public void infinity() {
      while (true) ;
   }
```

This test is designed to test how long it takes to run. I made it fail on purpose and that's exactly what it did. The timeout triggered since its an infinite loop.

```
@Test(expected = IndexOutOfBoundsException.class)
        public void testIndexOutOfBoundsException() {
           ArrayList emptyList = new ArrayList();
           Object a = emptyList.get(0);
        }
```

This test checks to see if an index is out of bounds. Since I created an empty array and asked to get the first index of that array, it failed the test.

Exercise 2

Clear

```
@Test
public void testClear() {
       while (!testArray.isEmpty()) {
              testArray.remove(0);
       }

       List<Integer> expected =
           new ArrayList<Integer>(Arrays.asList());
              assertEquals(testArray, expected);
}
```

Contains-True

```
@Test
public void testContains() {
       assertEquals(testArray.contains(3), true);
   }
```

Contains – False

```
@Test
public void testContainsFalse() {
       assertEquals(testArray.contains(0), false);
   }
```

Get Test where the 2nd in TestArray is 1

```
@Test
public void testGet() {
       assertEquals(testArray.get(1), 1);
   }
```

Exercise 3

TimeParser Test Statement Coverage & Branch (Since 3 ifs and one else if, it explores all branches)

```
import org.junit.Test;
import java.sql.Time;

import static org.junit.Assert.*;
public class TimeParserTestStatementCoverage {

   @Test (expected = NumberFormatException.class)
   public void testStatementCoverage() throws Exception {
      TimeParser.parseTimeToSeconds("00");
   }

   @Test (expected = NumberFormatException.class)
   public void testOneColon() throws Exception {
      TimeParser.parseTimeToSeconds("1:00");
   }

   @Test (expected = IllegalArgumentException.class)
   public void testInvalidTime() throws Exception {
      TimeParser.parseTimeToSeconds("24:00:00");
   }

   @Test
   public void testAM() throws Exception {
```

```
    assertEquals(TimeParser.parseTimeToSeconds("12:00:00am"), 0);
  }

  @Test
  public void testPM() throws Exception {
    assertEquals(TimeParser.parseTimeToSeconds("1:00:00pm"), 46800);
  }

}
```

TimeParser Test Path Coverage

```
import org.junit.Test;

import static org.junit.Assert.*;
public class TimeParserTestPathCoverage {

  @Test (expected = NumberFormatException.class)
  public void testStatementCoverage() throws Exception {
    TimeParser.parseTimeToSeconds("00");
  }

  @Test (expected = NumberFormatException.class)
  public void testOneColon() throws Exception {
    TimeParser.parseTimeToSeconds("1:00");
  }

  @Test (expected = IllegalArgumentException.class)
  public void testInvalidHours() throws Exception {
    TimeParser.parseTimeToSeconds("24:00:00");
  }

  @Test (expected = IllegalArgumentException.class)
  public void testInvalidMinutes() throws Exception {
    TimeParser.parseTimeToSeconds("00:61:00");
  }

  @Test (expected = IllegalArgumentException.class)
  public void testInvalidSeconds() throws Exception {
    TimeParser.parseTimeToSeconds("00:00:61");
  }

  @Test
  public void testAM() throws Exception {
    assertEquals(TimeParser.parseTimeToSeconds("12:00:00am"), 0);
  }
```

```
    @Test
    public void testPM() throws Exception {
        assertEquals(TimeParser.parseTimeToSeconds("1:00:00pm"), 46800);
    }

    @Test
    public void test24Hour() throws Exception {
        assertEquals(TimeParser.parseTimeToSeconds("13:00:00"), 46800);
    }

}
```

Exercise 4

```
@Test
  public void testInvariant {
      x = array[n];
      y = array[2*n+1];
      if (x <= y){
            z = true;
      }
      assertEquals(z, true);
  }

@Test
  public void testInvariant {
      x = array[n];
      y = array[2*n+2];
      if (x <= y){
            z = true;
      }
      assertEquals(z, true);
  }
```

If I were to find an error, I would resort after every iteration or remake the list to suit the changes that come across when I modify the array.