Lecture 2 - In Class Exercise

Goal: Understand the difference between fault, error, and failure. Practice for the homework.

1 Faults, Errors, Failures, oh my!

Instructions: Work with your neighbors in groups of 2.

```java
/**
* Find last index of zero
*
* @param x array to search
* @return last index of 0 in x; -1 if absent
* @throws NullPointerException if x is null
*/
public static int lastZero (int[] x) {
   for (int i = 0; i < x.length; i++) {
      if (x[i] == 0) {
         return i;
      }
   }
   return -1;
}
// test: x = [0, 1, 0];
// expect: return 2
```

Based on the code above, answer the following questions:

1. Explain what is wrong with the given code.
2. If possible, identify a test case that does **not** execute the fault.
3. If possible, identify a test case that executes the fault, but does **not** result in an error.
4. If possible, identify a test case that results in an error, but **not** a failure.
5. Identify the first error state for the given test.

```java
/**
* Count odd or postive elements
*
* @param x array to search
* @return count of odd/positive values in x
* @throws NullPointerException if x is null
*/
 public static int oddOrPos(int[] x) {
   int count = 0;
   for (int i = 0; i < x.length; i++) {
      if (x[i]%2 == 1 || x[i] > 0) {
         count++;
      }
   }
   return count;
}
// test: x = [-3, -2, 0, 1, 4];
// expect: return 3
```

Based on the code above, answer the following questions:

1. Explain what is wrong with the given code.
2. If possible, identify a test case that does **not** execute the fault.
3. If possible, identify a test case that executes the fault, but does **not** result in an error.
4. If possible, identify a test case that results in an error, but **not** a failure.
5. Identify the first error state for the given test.

```java
public class BigDecimalTest {
   BigDecimal x = new BigDecimal ("1.0");
   BigDecimal y = new BigDecimal ("1.00");
   // Fact: !x.equals(y), but x.compareTo(y) == 0

   Set <BigDecimal> BigDecimalTree = new TreeSet <BigDecimal> ();
   BigDecimalTree.add (x);
   BigDecimalTree.add (y);
   // TreeSet uses compareTo(), so BigDecimalTree now has 1 element

   Set <BigDecimal> BigDecimalHash = new HashSet <BigDecimal> ();
   BigDecimalHash.add (x);
   BigDecimalHash.add (y);
   // HashSet uses equals(), so BigDecimalHash now has 2 elements
}
// Test:
//    System.out.println ("BigDecimalTree = " + BigDecimalTree);
//    System.out.println ("BigDecimalHash = " + BigDecimalHash);
// Expect: BigDecimalTree = 1; BigDecimalHash = 1

// The problem is that in BigDecimal, equals() and compareTo() are inconsistent.
// Let's suppose we decide that compareTo() is correct, and that equals() is faulty.
```

Based on the code above, answer the following questions:

1. Explain what is wrong with the given code.
2. If possible, identify a test case that does **not** execute the fault.
3. If possible, identify a test case that executes the fault, but does **not** result in an error.
4. If possible, identify a test case that results in an error, but **not** a failure.
5. Identify the first error state for the given test.