CSE 331 Section 04: Testing

I. Blackbox & Whitebox Tests

```
/**
 * Takes an integer array and an integer number and returns the index
 * that the number appears in the array. Returns -1 if the number is
 * not found in the array.
 *
 * @param array an integer array that contains elements
 * @param num the integer number to be searched for in the array
 * @throws IllegalArgumentException if given array is null
 * @return the index of the given number in the given array, -1 if
 *         not found
 */
public static int search(int[] array, int num) {
        boolean smallIndex = false;
        int countIfStartAtOne = 1;
        if (array == null) {
                throw new IllegalArgumentException();
        }
        int index = -1;
        for (int i = 0; i < array.length; i++) {
                if (array[i] == num) {
                        index = i;
                }
        }
        smallIndex = (index < 10);
        countIfStartAtOne += index;
        return index;
}
```

Give four different black-box and white box tests for the method search as seen above. You do not need to write complete test codes. Simply describe what you are testing, the input, and the expected output. Each test must verify a distinct behavior of the method.

Black Box Test (give at least 4):

White Box Test (give at least 4):

II. Some Unreliable Co-worker

Your co-worker was writing a Java class and its unit tests shown below. **You know TAs will not be happy about what he wrote,** but you still decided to read through them and try to help your co-worker out because you are awesome.

Read the following code and answer the questions on the next page.

```
public class Direction {

  // Some unreliable programmer made this

  private int option;

  // create a direction with specified start and end
  public Direction(Place start, Place end) {
    // Your co-worker used dark magic to hide his bad implementation.
  }


  /**
   * Set the transportation user want to use
   *
   * @param option: 1 for bus
   *                2 for car
   *                3 for walking
   * @return true if the option is valid and set successfully
   *         false if the option was invalid and the option was not set
   */
  public boolean pickTransportation(int option) {
    // Your co-worker used dark magic to hide his bad implementation.
  }


  /**
   * Provide user suggestions base on the transportation that was picked previously
   *
   * @return a list of 4 suggestions if the bus option was picked
   *         a list of 2 suggestions if the car option was picked
   *         a list of 1 suggestion  if the walking option was picked
   */
  public List<Sugggestion> provideSuggestions(){
      // Your co-worker used dark magic to hide his bad implementation.
  }
}
```

Base on this piece of test code written by your partner, answer the following question.

*****Note: the test code is the same for all questions, but your co-worker might have different implementation in different question (that's his style)**

```
15 Direction dir = new Direction(plc01, plc02);
16 assertTrue(dir.pickTransportation(2));
17 assertTrue(dir.pickTransportation(1));
18 assertTrue(dir.pickTransportation(4));
19 assertEquals(dir.getSuggestions.size(), 4);
20 System.out.println("done!");
```

1. What are the probable problems if it outputs "done!" without exceptions? (more than one answer)
   A. There were no problems at all
   B. The test code is wrong
   C. The pickTransportation() method may have a bug
   D. The getSuggestions() method may have a bug

2. What is most likely to be the problem if assertion fails at Line 16?
   A. There were no problems at all
   B. The test code is wrong
   C. The pickTransportation() method may have a bug
   D. The getSuggestions() method may have a bug

3. What is most likely to be the problem if assertion fails at Line 17?
   A. There were no problems at all
   B. The test code is wrong
   C. The pickTransportation() method may have a bug
   D. The getSuggestions() method may have a bug

4. What is most likely to be the problem if assertion fails at Line 18?
   A. There were no problems at all
   B. The test code is wrong
   C. The pickTransportation() method may have a bug
   D. The getSuggestions() method may have a bug

TO BE CONTINUED ….

III. Tips when you write tests.

- Partition your tests by purposes and give detailed comments on what your tests are doing.
- When testing equals and hashcode method, mainly test the three properties of these two functions mentioned in slides (coming soon).
- When writing assertions on each element of a collection, add a message to the assertion with arguments.
- Assertion meesage's string can also include variables such as "Error from "+arr.get(i)
- Try to use the most precise assertion method Ex: No assertequals(obj, null). Use assertNotNull(obj).
- Overriding the toString() method for your classes can help you visualize elements and data structure.
- Use println methods to monitor test code execution when needed (but delete all of them before turning in)
- Use System.out.println and System.err.println together so that you can highlight some particular outputs since eclipse display them in different color by default.
- Do not only test viable inputs, but also make sure the wrong inputs do give back proper errors.
- For unit tests, NEVER rely on the execution of other unit tests.
- For black box testing, write the test as if you are seeing the method from the client's view
- For white box testing, write the test as if you are seeing the method from the implementor's view.
- IT IS YOUR OWN TEST, THERE IS NOT A SINGLE CORRECT ANSWER.