# Testing null and Avoiding NullPointerException

- bad: assertEquals(null, nullObject);
- bad: assertTrue(nullObject == null);
- good: assertNull(nullObject);

## Think hard about where things can go wrong…

- Pay close attention to both adding and removing.
- Make sure that size stays consistent as things are added and removed
- For structures that expand or contract, like an array-based bag, make sure you add/remove enough to trigger the change in size, and then making sure the resized structure is consistent.
- Make sure that a structure that once contained elements but is now empty behaves the same as a newly created structure.
- Look at behavior for empty collections, null values, and exceptions as specified in documentation

# Often helpful to use a loop to fill a collection for testing

```
Stack.push(”Chipolte”);
assertEquals(“Chipolte”, stack.peek());
assertEquals(1,stack.size());

for (int i = 1; i <= 10 ; i++) {
  stack.push(“restaurant number “ + i);
}
assertEquals(“restaurant number 10”, stack.peek());
assertEquals(11,stack.size());

stack.push(“Qdoba”);
assertEquals(“Qdoba”, stack.peek());
assertEquals(12,stack.size());

assertEquals(“Qdoba”, stack.pop());
assertEquals(11,stack.size());

assertEquals(“restaurant number 10”, stack.pop());
assertEquals(10,stack.size());
```

# Some tests for a stack data structure

```
ArrayStack<String> shortStack;
ArrayStack<String> tallStack;

public void setUp()

{
        shortStack = new ArrayStack<String>();
        tallStack = new ArrayStack<String>();

        for (int i = 0; i < 9; i++)
        {
                tallStack.push ("" + (i + 1));
        }

        shortStack.push ("A is for Array");
        tallStack.push ("B is for Boolean");
}
```

# Test clear() for a stack

```
public void testClear()

{
        // Testing basic clear() behavior, shortStack has 2 elements in it from setUp
        shortStack.clear();
        assertEquals(0, shortStack.getSize());
        assertTrue(shortStack.isEmpty());

        shortStack.push("C is for C++");
        assertEquals("C is for C++”,shortStack.peek());
        assertEquals("C is for C++”,shortStack.pop());

        // Testing clear()'s behavior after ensureCapacity has been run
        tallStack.clear();

        for (int i = 0; i < 15; i++)
        {
                tallStack.push("" + i);
        }

        tallStack.clear();

        // Testing that the stack had its max length reset to default
        assertEquals(11, tallStack.getLength());
}
```

# Just checking for exceptions is not enough!

```
/**
  * sets up for the tests
  */
 public void setUp() {

   tower = new Tower(Position.OTHER);
   discW10 = new Disc(10);

   Disc discW5 = new Disc(5);
   tower.push(discW5);
 }


 /**
  * test the push method
  */
 public void testPush() {

   Exception thrown = null;

   try {

     tower.push(discW10);
   }
   catch (Exception exception) {

     thrown = exception;
   }

   assertNotNull(thrown);
   assertTrue(thrown instanceof IllegalStateException);
 }
```

- Test pushing many discs.
- Test pushing after a clear.
- Test pushing in conjunction with remove() and peek().
- Use size() in asserts.
- Also use toArray() and equals() methods when available

# General Tips

1. Write small test methods so easier to debug

2. Remove duplicate code
   – We can remove all duplicate code from our test class by moving it to the setUp() methods.

3. Do not print anything out in unit tests
   – you will never need to add any print statement in your test cases. If you feel like having one, revisit your test case(s), you have done something wrong.
   – if you find yourself wanting to print that something succeeded, then perhaps that something should be factored into its own (well-named) test case.
   – If you find yourself wanting to print that something failed, you should probably be using an assert.