

Testing

CSE 331: Software Design and Implementation, Autumn 2026

Your job is to deliver code that you know works.

TODO

- Homework 1: due Friday, October 9, 11:59pm
- Reading 2: due Friday, October 16, 11:59pm

Correctness

Your job is to deliver code that you know works.

Definition

Code is *correct* if it meets its specification.

- Note this depends on having a spec!

Knowledge

*Your job is to deliver code that you **know** works.*

Three ways to gain knowledge:

- **Testing:** run the code on some inputs and check for expected behavior
- **Tools:** type check the code
- **Reasoning:** use logic to convince yourself the code satisfies the spec

Today: testing

Testing

Definition

A *test* runs code on a chosen input and compares the actual behavior to the expected behavior.

A test **passes** when the behaviors match and **fails** when they don't match.

How to write a test

1. Choose an input
2. Use the specification to figure out the expected behavior
3. Run the code on the input, and observe the actual behavior
4. Compare: the test passes if they match, and fails if not

Expected behavior first

Determine the expected behavior *before* running the code and observing the actual behavior. (Why?)

Specification D

```
7  /**
8   * Searches for a value in part of an array.
9   *
10  * @param a the array to search; must be non-null
11  * @param lo the index where the range to search begins
12  * @param hi the index just past where the range to search ends
13  * @param x the value to search for
14  * @requires 0 <= lo <= hi <= a.length
15  * @return some index i with lo <= i < hi and a[i] == x,
16  *         or -1 if there is none
17  */
18 static int indexOf(int[] a, int lo, int hi, int x) {
```

Implementation 1

```
18  static int indexOf(int[] a, int lo, int hi, int x) {  
19      for (int i = lo; i < hi; i++) {  
20          if (a[i] == x) {  
21              return i;  
22          }  
23      }  
24      return -1;  
25  }
```

Taking the steps

```
7  /**
8   * Searches for a value in part of an array.
9   *
10  * @param a the array to search; must be non-null
11  * @param lo the index where the range to search begins
12  * @param hi the index just past where the range to search ends
13  * @param x the value to search for
14  * @requires 0 <= lo <= hi <= a.length
15  * @return some index i with lo <= i < hi and a[i] == x,
16  *         or -1 if there is none
17  */
18 static int indexOf(int[] a, int lo, int hi, int x) {
```

```
18 static int indexOf(int[] a, int lo, int hi, int x) {
19     for (int i = lo; i < hi; i++) {
20         if (a[i] == x) {
21             return i;
22         }
23     }
24     return -1;
25 }
```

1. Choose an input

- $a = \{3, 0, 7\}$, $lo = 0$, $hi = 3$, $x = 7$

2. Use the specification to figure out the expected behavior

- 2

3. Run the code on the input, and observe the actual behavior

- Implementation 1 returns 2

4. Compare: the test passes if they match, and fails if not

- Test passes

A test in JUnit

```
6  public class IndexOfTest {  
7      @Test  
8      void findsTheOnlyMatch() {  
9          int[] a = {3, 0, 7};  
10         assertEquals(2, IndexOf.indexOf(a, 0, 3, 7));  
11     }
```

@Test: JUnit runs this method as a test

Step 1: the input you chose

Steps 2 to 4: expect 2, run the call, and compare

Unit, integration, end-to-end

- **Unit test:** one method
- **Integration test:** multiple methods working together
- **End-to-end test:** the whole program

Four tips for choosing inputs

1. Only test allowed inputs
 - Spec says nothing about disallowed inputs!
2. Test each method separately
 - Assume methods you call are correct. They have their own tests.
3. Test exhaustively when possible
 - Every allowed input, say when there are 10 or fewer
4. Cover each case and each boundary
 - Element appears in the list 0 times, 1 time, many times; empty list

What should the test expect?

- Input: the array {5, 1, 5, 2, 5}, the whole range, $x = 5$
- The spec allows any return value of 0, 2, or 4
- Implementation 1 searches from the front, and returns 0
- Implementation 4 searches from the back, and returns 4

A test that expects 0

```
19  @Test
20  void findsTheFirstMatch() {
21      int[] a = {5, 1, 5, 2, 5};
22      assertEquals(0, IndexOf.indexOf(a, 0, 5, 5));
23  }
```

Expects 0, the first match

A test that expects some match

```
25  @Test
26  void findsSomeMatch() {
27      int[] a = {5, 1, 5, 2, 5};
28      int i = IndexOf.indexOf(a, 0, 5, 5);
29      assertTrue(0 <= i && i < 5 && a[i] == 5);
30  }
```

Checks only D's promise: in the range, and holding 5

Specification-based tests

Definition

A **specification-based** test checks only behavior that the specification promises, so it passes on every correct implementation.

- **Implementation-based**: checks more than the spec promises

Half sheet (3 minutes)

- 1. A test calls `indexOf` on `{2, 2}`, the whole range, `x = 2`
 - It expects 0: is it specification-based, and why?
- 2. When must a test's input satisfy the precondition?
- Hand it in at the door as you leave

What tests tell us

- A test checks for some behavior on one execution
- **No set of tests** guarantees full correctness
 - Unless we achieve exhaustive testing
- Still, testing is a useful tool in the toolbox

Your job is to deliver code that you know works.

- Testing gives us knowledge that the code works on some executions.

Coming up

- Friday: testing, continued
 - How many tests are enough
- Monday: error handling, and interactive programs
- Homework 1: due Friday, October 9, 11:59pm
- Hand in your half sheet at the door