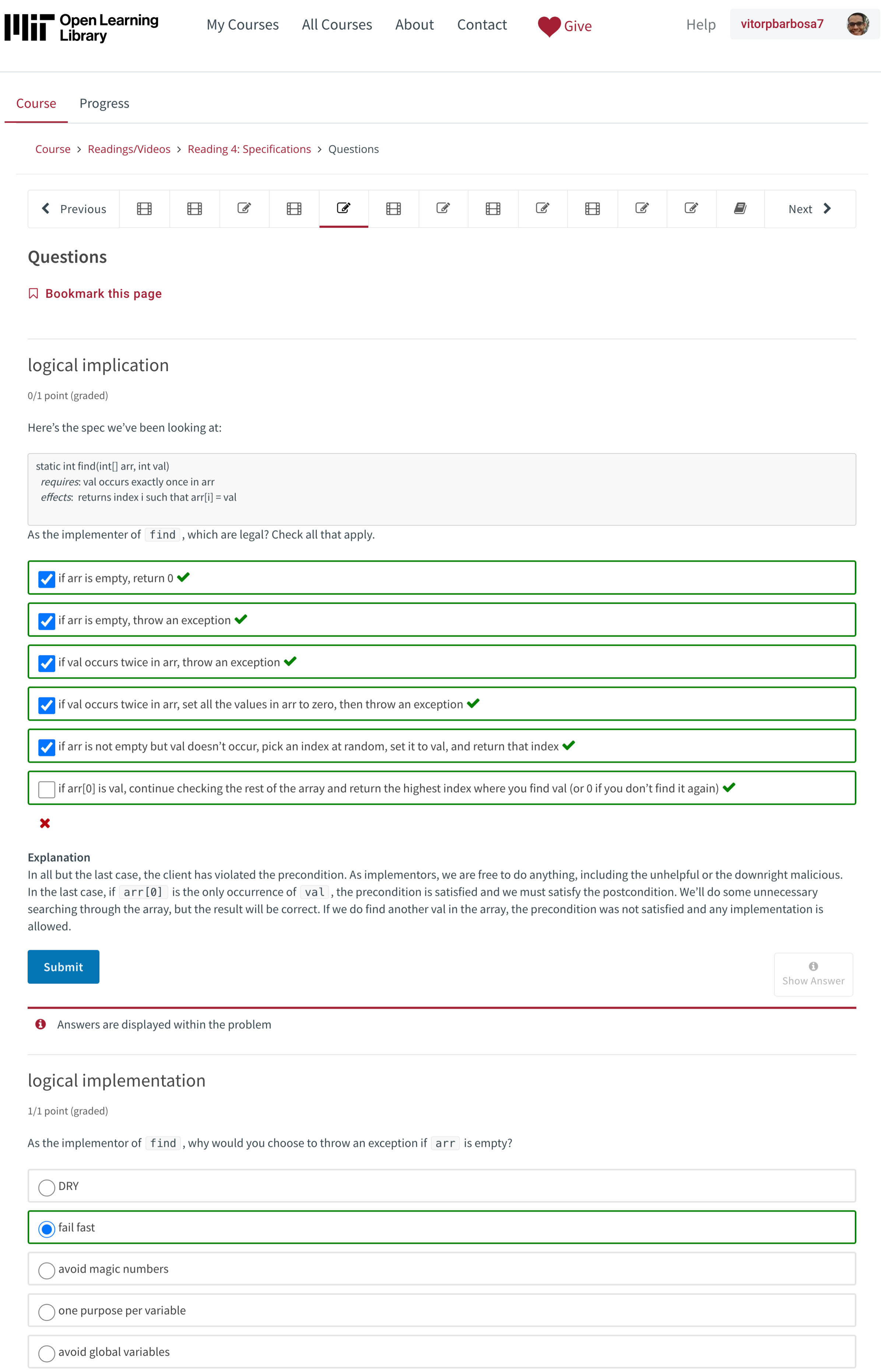


## Questions

### logical implication

0/1 point (graded)

Here's the spec we've been looking at:

```
static int find(int[] arr, int val)
  requires: val occurs exactly once in arr
  effects:  returns index i such that arr[i] = val
```

As the implementer of `find`, which are legal? Check all that apply.

- ☑ if arr is empty, return 0 ✓
- ☑ if arr is empty, throw an exception ✓
- ☑ if val occurs twice in arr, throw an exception ✓
- ☑ if val occurs twice in arr, set all the values in arr to zero, then throw an exception ✓
- ☑ if arr is not empty but val doesn't occur, pick an index at random, set it to val, and return that index ✓
- ☐ if arr[0] is val, continue checking the rest of the array and return the highest index where you find val (or 0 if you don't find it again) ✓

✖

**Explanation**

In all but the last case, the client has violated the precondition. As implementors, we are free to do anything, including the unhelpful or the downright malicious. In the last case, if `arr[0]` is the only occurrence of `val`, the precondition is satisfied and we must satisfy the postcondition. We'll do some unnecessary searching through the array, but the result will be correct. If we do find another val in the array, the precondition was not satisfied and any implementation is allowed.

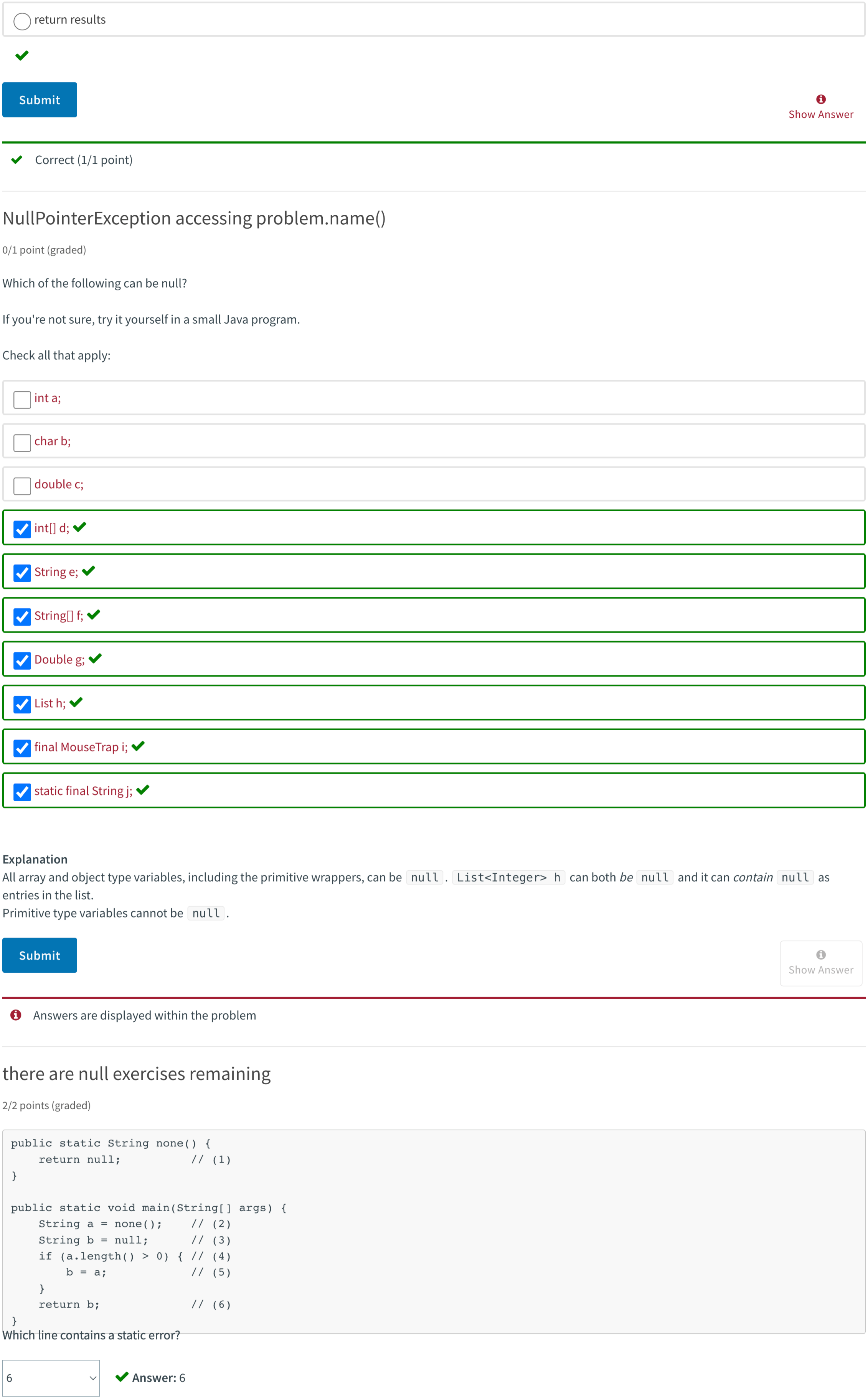
Answers are displayed within the problem

### logical implementation

1/1 point (graded)

As the implementor of `find`, why would you choose to throw an exception if `arr` is empty?

- ○ DRY
- ● fail fast
- ○ avoid magic numbers
- ○ one purpose per variable
- ○ avoid global variables



○ return results

✓

✓ Correct (1/1 point)

## NullPointerException accessing problem.name()

0/1 point (graded)

Which of the following can be null?

If you're not sure, try it yourself in a small Java program.

Check all that apply:

- □ int a;
- □ char b;
- □ double c;
- ☑ int[] d; ✓
- ☑ String e; ✓
- ☑ String[] f; ✓
- ☑ Double g; ✓
- ☑ List h; ✓
- ☑ final MouseTrap i; ✓
- ☑ static final String j; ✓

**Explanation**
All array and object type variables, including the primitive wrappers, can be `null`. `List<Integer> h` can both *be* `null` and it can *contain* `null` as entries in the list.
Primitive type variables cannot be `null`.

Answers are displayed within the problem

## there are null exercises remaining

2/2 points (graded)

```
public static String none() {
    return null;          // (1)
}

public static void main(String[] args) {
    String a = none();    // (2)
    String b = null;      // (3)
    if (a.length() > 0) { // (4)
        b = a;            // (5)
    }
    return b;             // (6)
}
```

Which line contains a static error?

6 ✓ **Answer:** 6

If we comment out that line and run main...

Which line contains a dynamic error?

4 ⌄ ✓ **Answer:** 4

**Explanation**

The line marked (6) is a static error because `null` is not the same as `void`, and a `void` method like `main` cannot return `null`.
On the line marked (4), since `none()` returns `null`, `a` will be `null`, and calling `length()` will cause a `NullPointerException`.

Submit

Show Answer

Answers are displayed within the problem

Previous Next

Open Learning Library
About
Accessibility
All Courses
Why Support MIT Open Learning?
Help

Connect
Contact
Twitter
Facebook

Privacy Policy Terms of Service