Questions

Bookmark this page

distinguished

2/2 points (graded)

We just saw the following implementations:

```
static int findFirst(int[] arr, int val) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == val) return i;
    }
    return arr.length;
}
```

```
static int findLast(int[] arr, int val) {
    for (int i = arr.length -1 ; i >= 0; i--) {
        if (arr[i] == val) return i;
    }
    return -1;
}
```

Now consider this possible specification of `find`:

```
static int find(int[] arr, int val)
  requires: nothing
  effects: returns largest index i such that
              arr[i] = val, or -1 if no such i
```

Which input(s) demonstrate that `findFirst` does not satisfy this spec? Check all that apply.

- [✓] `[ 1, 2, 2 ], 2`
- [ ] `[ 1, 2, 3 ], 2`
- [✓] `[ 1, 2, 3 ], 4`
- [ ] none of the above, `findFirst` does satisfy this spec!

✓

Which input(s) demonstrate that `findLast` does not satisfy this spec? Check all that apply.

- [ ] `[ 1, 2, 2 ], 2`
- [ ] `[ 1, 2, 3 ], 2`

☐ `[ 1, 2, 3 ], 4`

[✓ none of the above, `findLast` does satisfy the spec!]

✓

Submit

Show Answer

✓ Correct (2/2 points)

over/under (a)

1/1 point (graded)

Previously we saw this *under-determined* version of the *find* specification: (let's call it *findUnderdet*)

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

This spec allows multiple possible return values for arrays with duplicate values.

For each spec below, say whether it is *less* fully-determined than *findUnderdet*, *more* fully-determined, or deterministic.

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

[✓ deterministic]

○ more fully-determined than *findUnderdet*, but not deterministic

○ less fully-determined than *findUnderdet*

✓

Explanation

This is our original, deterministic spec.

Submit

Show Answer

Answers are displayed within the problem

over/under (b)

1/1 point (graded)

```
static int find(int[] arr, int val)
  requires: nothing
  effects: returns largest index i such that arr[i] = val, or -1 if no such i
```

[✓ deterministic]

○ more fully-determined than *findUnderdet*, but not deterministic

○ less fully-determined than *findUnderdet*

✓

Explanation
This is our version of the spec that distinguishes the two implementations in the reading. It is also determinstic.

Submit

Show Answer

Answers are displayed within the problem

over/under (c)

1/1 point (graded)

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

● deterministic

○ more fully-determined than *findUnderdet*, but not deterministic

○ less fully-determined than *findUnderdet*

✓

Explanation
Once again, the spec is fully-determined.

Submit

Show Answer

Answers are displayed within the problem

Previous Next

Questions

Bookmark this page

joint declaration

1/1 point (graded)

Given this specification:

```
static String join(String delimiter, String[] elements)
  effects: append together the strings in elements, but at each step,
           if there are more elements left, insert delimiter
```

Rewrite the spec so it is declarative, not operational.

- ○
```
effects: returns the result of adding all elements to a
         new StringJoiner(delimiter)
```
- ○
```
effects: returns the result of looping through elements and
         alternately appending an element and the delimiter
```
- ● ✓
```
effects: returns elements joined together with copies of delimiter, i.e.
         elements[0] + delimiter + elements[1] + delimiter
         + ... + delimiter + elements[elements.length-1]
```

✓

Explanation
The first and second options clearly talk about implementation.
`join` is, in fact, a static method of class `String`.
`StringJoiner` is also part of the standard API.

Submit Show Answer

Answers are displayed within the problem

out of joint

1/1 point (graded)

Which are valid criticisms of the declarative spec from the previous problem? Check all that apply.

- [ ] Be more clear about the empty delimiter special case
- [✓] Be more clear about the empty elements special case
- [ ] Be less deterministic, implementors need more freedom
- [ ] Be more deterministic, clients need more specific results

✓

Explanation

Behavior with the empty array is one aspect where none of the specs are immediately clear, including the third one.
The behavior with the empty delimiter, however, doesn't seem in doubt.
The spec was deterministic to begin with, and an underdetermined spec seems unlikely to be nearly as useful.

Submit

Show Answer

Answers are displayed within the problem

Previous Next

Questions

Bookmark this page

bulking up

1/1 point (graded)

When a specification is strengthened (check all that apply):

- [✓] fewer implementations satisfy it
- [ ] more implementations satisfy it
- [ ] fewer clients can use it
- [✓] more clients can use it
- [ ] none of the above

✓

Submit Show Answer

✓ Correct (1/1 point)

strength is truth

1/1 point (graded)

Which of the following can be true about a pair of specifications *A* and *B*? Check all that apply.

- [✓] 1. *A* can be stronger than *B* and have a weaker precondition
- [✓] 2. *A* can be stronger than *B* and have the same precondition
- [ ] 3. *A* can be stronger than *B* and have a stronger precondition
- [ ] 4. *A* can be stronger than *B* and have an incomparable precondition
- [✓] 5. *A* can be incomparable to *B*

✓

Explanation
If *A* has a stronger precondition, either it is weaker than *B*, or there is not a containment relationship (option 3).
If the preconditions are not comparable, then the specs will not be comparable either (4).

Submit Show Answer

Answers are displayed within the problem

finding find1

1/1 point (graded)

Here are the *find* specifications from the reading:

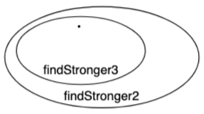
```
static int find1(int[] a, int val)
  requires: val occurs exactly once in a
  effects:  returns index i such that a[i] = val
```

```
 static int findStronger2(int[] a, int val)
  requires: val occurs at least once in a
  effects:  returns index i such that a[i] = val
```

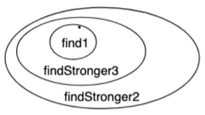
```
 static int findStronger3(int[] a, int val)
  requires: val occurs at least once in a
  effects:  returns lowest index i such that a[i] = val
```

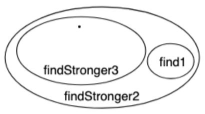
```
static int find4(int[] a, int val)
  requires: nothing
  effects:  returns index i such that a[i] = val,
              or -1 if no such i
```

We already know that *findStronger3* is stronger than *findStronger2*, which is stronger than *find1*.

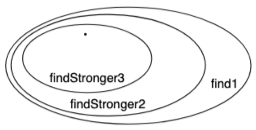


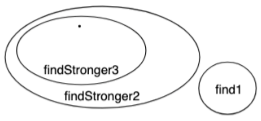
Where is *find1* on the diagram?





[✓]





✓

Submit

Show Answer

✓ Correct (1/1 point)

finding find4

1/1 point (graded)

Let's determine where *find4* is on the diagram.

How does find1 compare to find4?

- find4 is weaker
- [✓ find4 is stronger]
- find4 and find1 are incomparable

✓

Explanation
find4 has a weaker precondition.
For inputs that satisfy find1's precondition, find4's postcondition is equal.
So find4 is stronger.

Submit

Show Answer

Answers are displayed within the problem

vs. findStronger2

1/1 point (graded)

How does findStronger2 compare to find4?

- ○ find4 is weaker
- [✓ find4 is stronger]
- ○ find4 and findStronger2 are incomparable

✓

Submit

Show Answer

✓ Correct (1/1 point)

vs. findStronger3

0/1 point (graded)

How does findStronger3 compare to find4?

- ○ find4 is weaker
- [✓ find4 is stronger]
- ○ find4 and findStronger3 are incomparable

✗

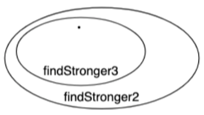
Submit

Show Answer

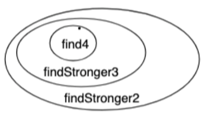
✗ Incorrect (0/1 point)

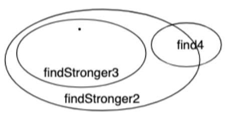
found

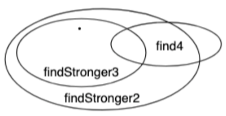
0/1 point (graded)

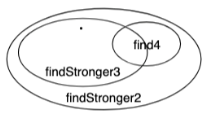


Where is find4 on the diagram?

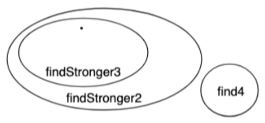
[✓ ]

○ 

○ 



✓



✗

Explanation
Since find4 is stronger than findStronger2, it must be contained within that region of the space.
Then the question is its relationship to findStronger3.
There exist implementations that satisfy findStronger3 but not find4: for example, they do not return -1 when `val` is not in `a`, which is excluded by findStronger3's precondition.
There also exist implementations that satisfy find4 but not findStronger3: for example, they do not return the lowest index when `val` occurs multiple times.
And there exist implementations that satisfy both: they can handle the weaker precondition, and the stronger parts of each postcondition.
So find4 overlaps findStronger3, but neither contains the other.

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

Questions

show me a sign

1/1 point (graded)

Which of the following are signs of an excellent specification (check all that apply):

- [x] 1. the specification is declarative
- [ ] 2. the specification is operational
- [ ] 3. the specification is as super-strong as possible
- [ ] 4. the specification is as super-weak as possible
- [ ] 5. the implementation is allowed to ignore invalid arguments
- [x] 6. the implementation is allowed to use different algorithms depending on the arguments
- [ ] 7. the specification utilizes the reader's knowledge of the implementation

✓

Explanation
We prefer declarative specs (option 1) to operational ones (2).
We want a spec that is neither too strong (3) nor too weak (4) to balance the constraints of implementor and client.
We would rather have clear specs and implementations that fail fast than allow the implementation to quietly fail (5).
And we do not want the client to have to read the implementation at all (7).

Submit Show Answer

Answers are displayed within the problem

that's an odd way of looking at it

1/1 point (graded)

```
public static int secondToLastIndexOf(int[] arr, int val)
  requires: val appears in arr an odd number of times
  effects:  returns the 2nd-largest i such that arr[i] == val
```

Which of the following are reasonable criticisms of this spec? Check all that apply.

- [✓] The spec is not well-defined, we cannot implement it
- [✓] The spec is not coherent
- [ ] The spec is not deterministic
- [ ] The spec is not operational

✓

Submit

Show Answer

✓ Correct (1/1 point)

behavioral oddities

4/4 points (graded)

Consider the following test cases for `secondToLastIndexOf` :

`[ 1, 3, 4 ], 3` returns `1`

- ○ valid test case
- ○ could be valid with a weaker precondition, same postcondition
- ○ could be valid with a weaker precondition, stronger postcondition
- [✓ ● could be valid with same precondition, weaker postcondition]

✓

Explanation
The current postcondition doesn't admit any output value with only one occurence of `val` .

`[ 1, 3, 3, 4 ], 3` returns `1`

- ○ valid test case
- [✓ ● could be valid with a weaker precondition, same postcondition]
- ○ could be valid with a weaker precondition, stronger postcondition
- ○ could be valid with same precondition, stronger postcondition

✓

Explanation
We would need to weaken the precondition to allow even occurences of `val` .

`[ 1, 3, 3, 3, 4 ], 3` returns `2`

- [✓ ● valid test case]
- ○ could be valid with a weaker precondition, same postcondition

○ could be valid with a weaker precondition, stronger postcondition

○ could be valid with same precondition, stronger postcondition

✓

Explanation

Satisfies the precondition, and the postcondition is deterministic.

`[ 3, 3, 3, 3 ], 3` throws an exception

○ valid test case

○ could be valid with a weaker precondition, same postcondition

● could be valid with a weaker precondition, stronger postcondition

○ could be valid with same precondition, stronger postcondition

✓

Explanation

This is not (currently) a valid test because it violates the precondition; maybe the method throws an exception in such cases, but that behavior is beyond the spec. So at least we need to weaken the precondition to allow this input. Having done that, we also need to change the postcondition to specify that an exception is thrown when `val` appears an even number of times, because the current postcondition doesn't mention any exceptions at all.

Submit

Show Answer

Answers are displayed within the problem

odd doc

1/1 point (graded)

Here's the spec again:

```
public static int secondToLastIndexOf(int[] arr, int val)
  requires: val appears in arr an odd number of times
  effects:  returns the 2nd-largest i such that arr[i] == val
```

Choose lines below to complete one possible Javadoc version of this terrible spec:

☐ /*

☑ /**

☑ * Finds the second-to-last occurence of a value in an array.

☐ * Find j, the largest index such that arr[j] == val.

☐ * Then find i, the largest index such that i < j and arr[i] == val.

☑ * @param arr array to search

☐ * @param arr fixed-size array of integers to search

☐ * @param val value to search for

- [✓] * @param val value to search for, requires val appears in arr an odd number of times
- [ ] * @return index i
- [✓] * @return second-largest index i such that arr[i] == val
- [✓] */

✓

Explanation
The business about computing `j` and then `i` in terms of `j` is operational so we don't need that.
We don't need to explain Java semantics (e.g. `arr` is a fixed-size array of integers), but we definitely do need to explain the precondition.
It's good to include a summary of the function as the first line of the Javadoc, but we should still explain the postcondition on the return value clearly and succinctly.
And let's just be clear: regardless of how well we write it up, *this is a terrible specification.*

Submit

Show Answer

Answers are displayed within the problem

Previous Next