Questions

Bookmark this page

behave nicely

1/1 point (graded)

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

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

If clients only care about calling the find method when they know `val` occurs exactly once in `a`, are `findFirst` and `findLast` behaviorally equivalent?

- (•) Yes
- ( ) No

✔

Explanation
If `val` occurs exactly once in `a`, then it doesn't matter whether we search from start to end or end to start; and it doesn't matter what we do when we don't find `val`.
Once we define how specifications are structured, we'll see that they are equivalent in this case because a strong precondition hides their potential differences in behavior.

Submit Show Answer

Answers are displayed within the problem

best behavior

1/1 point (graded)

Suppose clients only care that the find method should return:

- any index `i` such that `a[i] == val`, if `val` is in `a`

- any integer `j` such that `j` is not a valid array index, otherwise

In this case, are `findFirst` and `findLast` behaviorally equivalent?

- (•) Yes
- ( ) No

✔

Explanation
Both specifications satisfy these very minimal requirements on the return value.
Once we define how specifications are structured, we'll see that they are equivalent in this case because a weak postcondition permits their differences in behavior.

Submit Show Answer

Answers are displayed within the problem

Questions

Bookmark this page

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.

- [x] if arr is empty, return 0 ✔
- [x] if arr is empty, throw an exception ✔
- [x] if val occurs twice in arr, throw an exception ✔
- [x] if val occurs twice in arr, set all the values in arr to zero, then throw an exception ✔
- [x] 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.

Submit Show Answer

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

✔

Submit

Show Answer

✔ 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`.

Submit

Show Answer

ℹ 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

Questions

Bookmark this page

get in here

1/1 point (graded)

Which of the following are part of a function's specification? Check all that apply.

- [x] 1. return type
- [x] 2. restrictions on return values
- [x] 3. number of arguments
- [x] 4. argument types
- [x] 5. restrictions on argument values

✔

Explanation
1, 3, and 4 are statically checked by Java; 2 and 5 are usually not.
1 and 2 are postconditions; 3 through 5 are preconditions.

Submit Show Answer

Answers are displayed within the problem

greatest common denominator

4/4 points (graded)

Alice writes the following code:

```
public static int gcd(int a, int b) {
    if (a > b) {
        return gcd(a-b, b);
    } else if (b > a) {
        return gcd(a, b-a);
    }
    return a;
}
```

Bob writes the following test:

```
@Test public void gcdTest() {
    assertEquals(6, gcd(24, 54));
}
```

The test passes!

Alice should write `a > 0` in the precondition of `gcd`

- (•) True
- ( ) False

✔

Alice should write `b > 0` in the precondition of `gcd`

- (●) True
- ( ) False

✔

Alice should write `gcd(a, b) > 0` in the precondition of `gcd`

- ( ) True
- (●) False

✔

Alice should write `a and b are integers` in the precondition of `gcd`

- ( ) True
- (●) False

✔

Explanation
The function required `a > 0` and `b > 0` in order to return a correct answer.
`gcd(a, b) > 0` is not a precondition, it would be a postcondition.
The compiler already checks that `a` and `b` are integers, so writing it again would be duplicative.

Submit

Show Answer

Answers are displayed within the problem

gcd, cont'd

2/2 points (graded)

If Alice adds `a > 0` to the precondition, Bob should test negative values of `a`

- ( ) True
- (●) False

✔

If Alice does not add `a > 0` to the precondition, Bob should test negative values of `a`

- (●) True
- ( ) False

✔

Explanation
Bob should not test inputs that violate the precondition. But without the precondition, a reasonable partitioning of input `a` would certainly include negative values.

Submit

Show Answer

Answers are displayed within the problem

Previous Next

Questions

Bookmark this page

get to the point

1/1 point (graded)

Suppose we're building a robot and we want to specify the function

```
public static List<Point> findPath(Point initial, Point goal)
```

which is responsible for path-finding: determining a sequence of `Point`s that the robot should move through to navigate from `initial` to `goal`, past any obstacles that might be in the way.

In the postcondition, we say that `findPath` will search for paths only up to a bounded length (set elsewhere), and that it will throw an exception if it fails to find one.

Which of these would we choose?

- ◯ throws a checked `NoPathException`
- ◯ throws an unchecked `NoPathException`
- ◉ throws a checked `PathNotFoundException`
- ◯ throws an unchecked `PathNotFoundException`

✔

Explanation
`NoPathException` is a bad name, since there might be very long paths the function doesn't find.
Unless we're working in a domain where not having a path signals a bug, it seems like not finding a path is a condition that the client must anticipate.
And it's not feasible for the client to check beforehand whether there is a path (that's what this function does)... so a *checked* exception is appropriate.

Submit Show Answer

Answers are displayed within the problem

don't point that thing at me

1/1 point (graded)

When we define our exception for `findPath`, which will we choose as our superclass?

- ◯ Throwable
- ◉ Exception
- ◯ Error
- ◯ RuntimeException

✔

Explanation
When we declare new checked exceptions, we subclass `Exception`.

Submit

Show Answer

Answers are displayed within the problem

Previous Next

Questions

Bookmark this page

throw all the things!

1/1 point (graded)

Examine this code for analyzing some `Thing` objects:

```
static void analyzeEverything() {
    analyzeThingsInOrder();
}

static void analyzeThingsInOrder() {
    try {
        for (Thing t : ALL_THE_THINGS) {
            analyzeOneThing(t);
        }
    } catch (AnalysisException e) {
        return;
    }
}

static void analyzeOneThing(Thing t) throws AnalysisException {
    // ...
    // ... maybe go off the end of an array
    // ...
}
```

`AnalysisException` is an unchecked exception.

Which exceptions could be thrown by a call to `analyzeEverything`? Check all that apply.

- [x] ArrayIndexOutOfBoundsException
- [ ] IOException
- [x] NullPointerException
- [ ] AnalysisException
- [x] OutOfMemoryError

✔

Explanation

`ArrayIndexOutOfBoundsException`, `NullPointerException`, and `OutOfMemoryError` are unchecked exceptions; they could be thrown by `analyzeOneThing`, propagate up through `analyzeThingsInOrder`, and be thrown by `analyzeEverything`.
`AnalysisException` is unchecked, but there is no code in `analyzeThingsInOrder` to throw it, and any of them thrown by `analyzeOneThing` will be caught by the try-catch.
`IOException` is a checked exception, so these methods would be required by the compiler to declare it if it could be thrown.

Submit | Show Answer

Answers are displayed within the problem

a terrible thing

1/1 point (graded)

What might happen if `analyzeOneThing` throws an `AnalysisException` in this code? Check all that apply.

- [ ] The program might crash
- [x] We might fail to call analyzeOneThing on all of the Things in ALL_THE_THINGS
- [ ] We might call analyzeOneThing multiple times on some Thing(s)

✔

Explanation

Notice how the try-catch is outside the for loop in the code.
If we are only partially through our iteration over `ALL_THE_THINGS`, the exception will bubble up out of the loop and prematurely end our iteration.
If we wanted to continue iterating past `Thing`s that cause `AnalysisException`s, we would need to put the try-catch inside the loop.
The program will not crash, because we do catch the `AnalysisException` before it can propagate further up the call stack.

Submit

Show Answer

Answers are displayed within the problem

Previous Next

Questions

Bookmark this page

words with specs

2/2 points (graded)

Suppose we're working on the method below:

```
/**
 * Requires: tiles has length 7 & contains only uppercase letters.
 *           crossings contains only uppercase letters, without duplicates.
 * Effects: Returns a list of words where each word can be made by taking
 *          letters from tiles and at most 1 letter from crossings.
 */
public static List<String> scrabble(String tiles, String crossings) {
    if (tiles.length() != 7) { throw new RuntimeException(); }
    return new ArrayList<>();
}
```

Which are parts of the *postcondition* of `scrabble`?

- [ ] `tiles` has only uppercase letters
- [ ] `crossings` has no duplicates
- [ ] `scrabble` takes two arguments
- [x] `scrabble` returns a list of strings

✔

Explanation
The first three choices are preconditions. Only the last choice is a postcondition.

Which are parts of the *precondition* of `scrabble`?

- [x] `tiles` has length 7
- [x] `crossings` is a string of uppercase letters
- [x] `scrabble`'s arguments are of type `String` and `String`
- [ ] `scrabble` returns an empty `ArrayList`

✔

Explanation
The last answer is true about the implementation, but it is not part of the spec.

Submit

Show Answer

Answers are displayed within the problem

Scrabble scramble

2/2 points (graded)

Which are *parts of the spec that are checked statically* by Java?

- [ ] `tiles` is a string of uppercase letters
- [ ] `crossings` has no duplicates
- [ ] when `tiles.length() != 7`, `scrabble` throws a `RuntimeException`
- [x] `scrabble` takes two arguments

✔

Does the `scrabble` implementation satisfy its specification?

- (•) Yes
- ( ) No, because it throws a `RuntimeException` when `tiles` has invalid length
- ( ) No, because it returns an empty result list even if we call it with tiles and crossings that can be combined to form valid words

✔

Explanation
The specification doesn't say that the method returns *all* words that can be made from the inputs, or even *some* words that can be made.
This is a clear place where the specification could be improved.

Submit

Show Answer

Answers are displayed within the problem

Previous Next