Ask me anything

0 questions
0 upvotes
My tests cover every line of code, and all the tests are green. Can I be sure there are no bugs?

Yes. The code is correct.

Maybe. Tests can't cover all cases.

No. There are always bugs.
Should you write tests for private or protected methods? Why or why not?

Private methods are called from public methods, so test the public methods that call these private methods. Protected methods should be tested as they are used in the subclasses as well.
Why are bugs common at boundaries?

- Because sometimes we forget trivial things like a +1 or a -1.
- Most implementations are vulnerable to strange input.
- When writing the method we tend to forget about such input, as it's unusual.
- Tests special and unexpected user input, to see if the program can deal with those.
- Sometimes we count from 0 and we are sometimes not familiar with that.
- Because they are special cases and are not always thought about when writing code.
- Boundaries are edge cases which are sometimes hard to understand what really happens. Example when handling integer.MAX_VALUE, it could overflow to a negative integer.
What would be a suitable class invariant for ArrayStack?

```java
public class ArrayStack<T> implements StackInterface<T> {
    protected T store[];
    protected int capacity;
    protected int size;

    public ArrayStack() {
        store = null; // default value
        capacity = 0; // available slots
        size = 0; // used slots
    }
}
```

- `size <= store.size()`
- `size > capacity`
- `size <= store.size()`
- `capacity >= 0`
- `capacityy >= 0`
- `size >= 0`
- `size < capacity`
- `return size >= 0 && capacity > 0;`
- `size <= capacity`
What would be a suitable class invariant for ArrayStack?

```java
public class ArrayStack<T> implements StackInterface<T> {
    protected T store[];
    protected int capacity;
    protected int size;

    public ArrayStack() {
        store = null; // default value
        capacity = 0; // available slots
        size = 0; // used slots
    }
}
```

- store.size == capacity
- if not empty, store.size==capacity
- store.size <= capacity
- if (store == null && capacity == 0) return true
- else if (store.size() == capacity) return true
- 0 <= size <= capacity
- store.size==capacity || store.size==null

When you think the bug is in that method you should step into
How would you implement the `ArrayStack.grow()` method?

- Init new Array of Size 2x old
- `ArrayCopy` the elements using a for-loop
- if you think that there is a problem in this line you should step into otherwise not
Demo — debugging ParenMatchTest with a broken LinkStack.pop()
Where should I set a breakpoint?
When should I step into rather than over code?

- if the problem occurs after that line
- if you need more information
- when you expect the bug to be close
- If a line calls multiple methods
- when you suspect a bug or don't understand what that part of the code does
- when you don't know where the problem is and don't know what it do's
How do you explain the benchmarks?

Popping from the ArrayStack is very fast as there isn't any pointer relocation.

wrapperStack has more return statements.

<table>
<thead>
<tr>
<th>Stack Implementation</th>
<th>10M pushes</th>
<th>10M pops</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2.stack.LinkStack</td>
<td>909</td>
<td>98</td>
</tr>
<tr>
<td>p2.stack.ArrayStack</td>
<td>342</td>
<td>24</td>
</tr>
<tr>
<td>p2.stack.WrappedStack</td>
<td>816</td>
<td>529</td>
</tr>
<tr>
<td>java.util.Stack</td>
<td>253</td>
<td>175</td>
</tr>
</tbody>
</table>
Last chance for questions