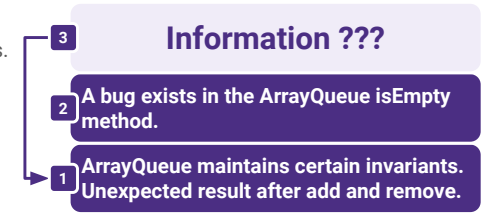
The Role of Information

How are bugs fixed? Here's one proposal.

- Productive changes fix bugs.
- Information gathered about the system informs productive changes.
- A hypothesis guides information gathering and testing.
- Things we know about the problem inform how we choose hypotheses.

The point here is that information is the most important thing and you need to do whatever's necessary to get information.

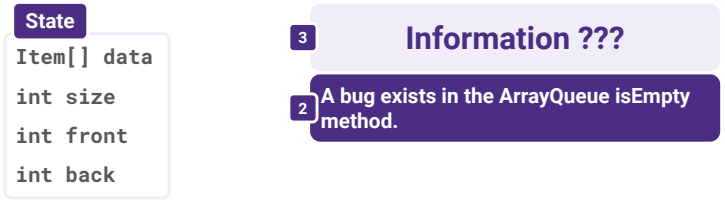


What does debugging a program look like? (Julia Evans); The Debugging Mindset (Devon H. O'Dell/ACM Queue)

?: Do we have reason to question the things we know about the problem? Or the hypothesis?

Generating Hypotheses

A good hypothesis identifies the cause of failure separately from where and when the program actually fails. The **state** of the ArrayQueue determines the behavior of isEmpty.



The Debugging Mindset (Devon H. O'Dell/ACM Queue)

A good hypothesis describes a problem and is both testable and falsifiable. While this hypothesis is both, it doesn't really yield much more information about the problem or how to fix it.

Descriptive Hypotheses

A good hypothesis identifies the cause of failure separately from where and when the program actually fails. The **state** of the ArrayQueue determines the behavior of isEmpty.

The hypothesis on the left suggests more about the problem than the one on the right.

The size variable is not set correctly, causing isEmpty to return false.

A bug exists in the ArrayQueue isEmpty method.

Q1: Which of the hypotheses below do you think will yield more information upon answering?

```
ArrayQueue1<Integer> queue = new ArrayQueue1<>();
queue.add(1);
queue.remove();
queue.add(3);
queue.remove();
queue.remove();
queue.add(6);
queue.remove();
System.out.println(
    "isEmpty() expected true, got " + queue.isEmpty());
```

Tests as a Source of Information

?: What does this test check?

Gathering Information

Debugging is about integrating various different sources of information. Let's try out a few methods of gathering information to understand the bug.

- Trying new inputs.
- Writing a unit test to reproduce the bug.
- Explaining to yourself the behavior of each line of code.
- Searching online to understand what error messages mean.
- Changing or removing code.
- Poking at memory values with a debugger or print statements.

?: Bugs often appear away from their root causes. How does each information gathering method help us learn more about the problem?

JUnit Testing

Testing is a means of gathering information. But because unit tests are just programs, we can automate it and continuously gather information.

Better JUnit Testing

```
@org.junit.Test
```

ArrayQueueTest

The messages output by JUnit are kind of ugly, and invoking each test manually is annoying.

IntelliJ has built-in support for JUnit.

1. Annotate each test (Java method) with @org.junit.Test.
2. Change all test methods to non-static.
3. Use IntelliJ's built-in JUnit runner to run all tests and tabulate results.

This is called **boilerplate code**. IntelliJ can generate this code for you!

To have IntelliJ generate the boilerplate code in a class ArrayQueue Test,

1. With the cursor placed somewhere inside the class's curly braces, press Alt+Insert (on Windows/Linux) or Command+N (on Mac) to open the options for generating code.
2. In the pop-up menu, select "Test Method" and then choose the JUnit test option.
3. Fill in a name for the test.
4. Write the body of the test.

Protip: You can also generate all sorts of other boilerplate from this keyboard shortcut, such as "public static void main(String[] args)".

The Role of Information

The point here is that information is the most important thing and you need to do whatever's necessary to get information.

How are bugs fixed? Here's one proposal.

- Productive changes fix bugs.
- Information gathered about the system informs productive changes.
- A hypothesis guides information gathering and testing.
- Things we know about the problem inform how we choose hypotheses.

→ **Two new regression tests**

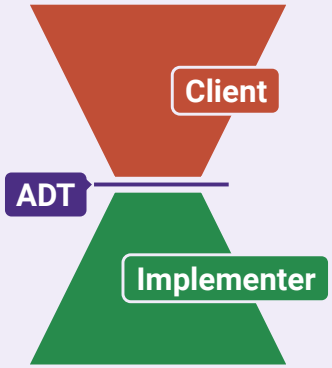
What does debugging a program look like? (Julia Evans); The Debugging Mindset (Devon H. O'Dell/ACM Queue)

They're called regression tests because we want to ensure that future changes don't break these test cases that we know our program has had difficulty with in the past.

Q Testing as Planning

Suppose we're implementing ArrayDeque from HW 2.

1. Describe a unit test we might want to write for ArrayDeque.
2. What behaviors does this test check? Describe in terms of the methods it checks as well as concepts like contracts, invariants, etc. that we've discussed in class.



?: How did the ad-hoc tests for ArrayQueue1 and ArrayQueue2 expose particular bugs? What was special about those tests?

Q1: Describe a unit test we might want to write for ArrayDeque. Recall that the Deque interface expects methods such as addFirst, addLast, removeFirst, removeLast, and get.

Q2: What behaviors does this test check? Describe in terms of the methods it checks as well as concepts like contracts, invariants, etc. that we've discussed in class.

Testing as Planning

The point here is that information is the most important thing and you need to do whatever's necessary to get information.

Not only does running a test improve our understanding of a problem, **but so does writing a test**!

"I'm almost done, I just need to make sure it works."
– Famous last words

Tests are hard to write, but easy to run.

Maximize the benefit of testing by writing tests first (or early) and code afterwards.

What does debugging a program look like? (Julia Evans)

If testing is left until after all of the code is written, we lose any opportunities to gather information along the way and fix bugs as they come up! We may even be solving the wrong problem altogether.

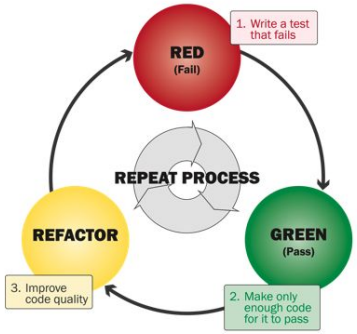
?: How does testing serve as a form of planning?

?: What makes writing good tests so challenging?

Test-Driven Development

The point here is that information is the most important thing and you need to do whatever's necessary to get information.

1. Identify a new feature.
2. Write a unit test for that feature.
3. **RED**: Run the test. It should fail.
4. **GREEN**: Write code that passes test.
5. **REFACTOR**: Improve code quality.



What does debugging a program look like? (Julia Evans); Red-Shirt, Red, Green, Refactor - A TDD Fairytale (Ryan Tablada)

?: Testing is just one tool in the information gathering toolbox. We've seen how testing can be a forming of planning. How do other information gathering methods inform our planning processes?

The Role of Information

The point here is that information is the most important thing and you need to do whatever's necessary to get information.

How are bugs fixed? Here's one proposal.

- Productive changes fix bugs.
- Information gathered about the system informs productive changes.
- A hypothesis guides information gathering and testing.
- Things we know about the problem inform how we choose hypotheses.

3 **Modify the remove method to handle the special case of removing if empty.**

2 **The remove method decrements the size variable even when the queue is empty.**

1 **ArrayQueue maintains certain invariants. Unexpected result after add and remove.**

What does debugging a program look like? (Julia Evans); The Debugging Mindset (Devon H. O'Dell/ACM Queue)

?: What are the differences between this new hypothesis and the hypothesis that we started with? How did we get from the starting hypothesis to this new hypothesis?