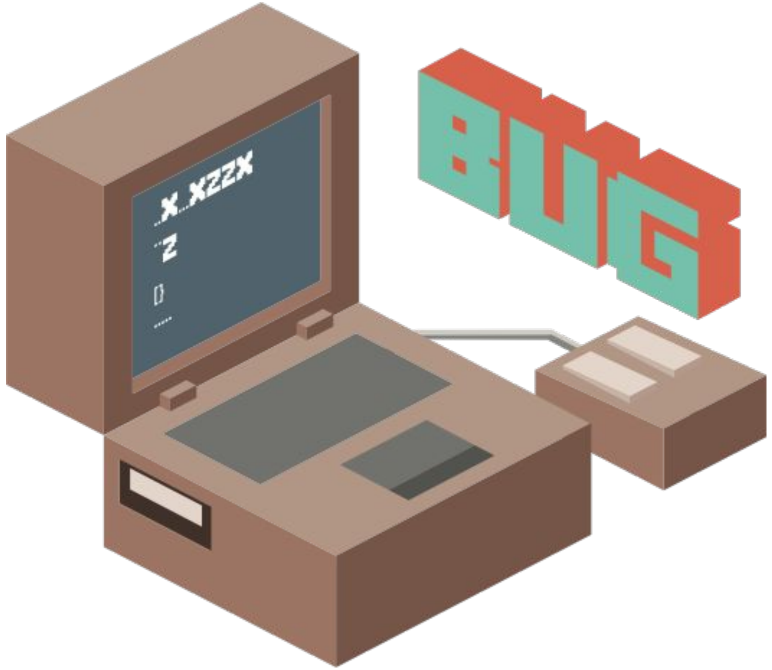
MAKE SCHOOL

Debugging 2: Techniques

SPD 2.3

Agenda

- Learning Outcomes
- **Activity 1**: What is Debugging?
- Teacher Talk
- **Interactive Debugging Demo**
- Break
- **Activity 2**: Stepping Through



Learning Outcomes

By the end of today, you should be able to…

1. Compare and contrast the **pros and cons** of **different debugging techniques**.
2. **Use breakpoints** to debug code.
3. Apply debugging techniques in a project of their own.

Warm-Up: Read & Discuss

MAKE SCHOOL

"Once a problem is described in sufficient detail, its solution is obvious."

- https://wiki.c2.com/?RubberDucking

Read & Discuss (10 minutes)

Read this article on the psychology of "rubber duck debugging". Then, in a group of 3, answer the following questions:

- Have you used "rubber duck debugging" before? Was it useful?
- According to the article, how could you make your "rubber duck debugging" more useful in the future?

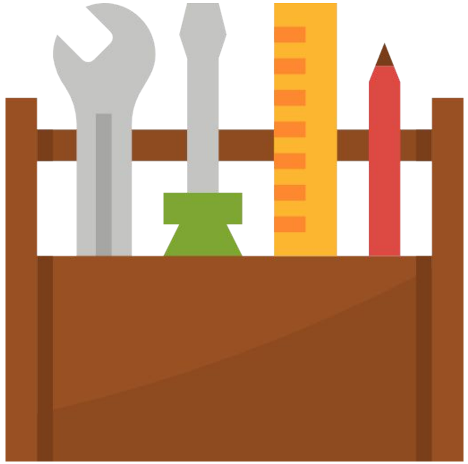
MAKE SCHOOL

Techniques

Advanced Techniques

It's important to have some techniques in your toolbox that enable you to solve your own problems. We'll go over a few:

- Trace Backward
- Trace Forward
- Divide and Conquer



Trace Backward

To **trace backward**, figure out which line of code is producing the error. Then, **work backwards one line at a time** until you find the source of the bad data.

Use this technique when...

- The **error is thrown from a known location**
- You **understand the program well enough** to know what it's supposed to be doing

Trace Forward

MAKE SCHOOL

To **trace forward**, start at the beginning of your program. Then, **work forwards one line at a time** and **verify that your mental model matches the result**, using a debugger or log statements.

Use this technique when...

- The **problem line isn’t known**
- You **can't narrow down the problem** to one specific area of the code

Divide and Conquer

To **divide and conquer**:

- **Identify different code sections** that could have caused the error
- Use **breakpoints or log statements at the boundaries** to test which area is the problem
- **Focus your efforts** on that area using Trace Forward or Trace Backward

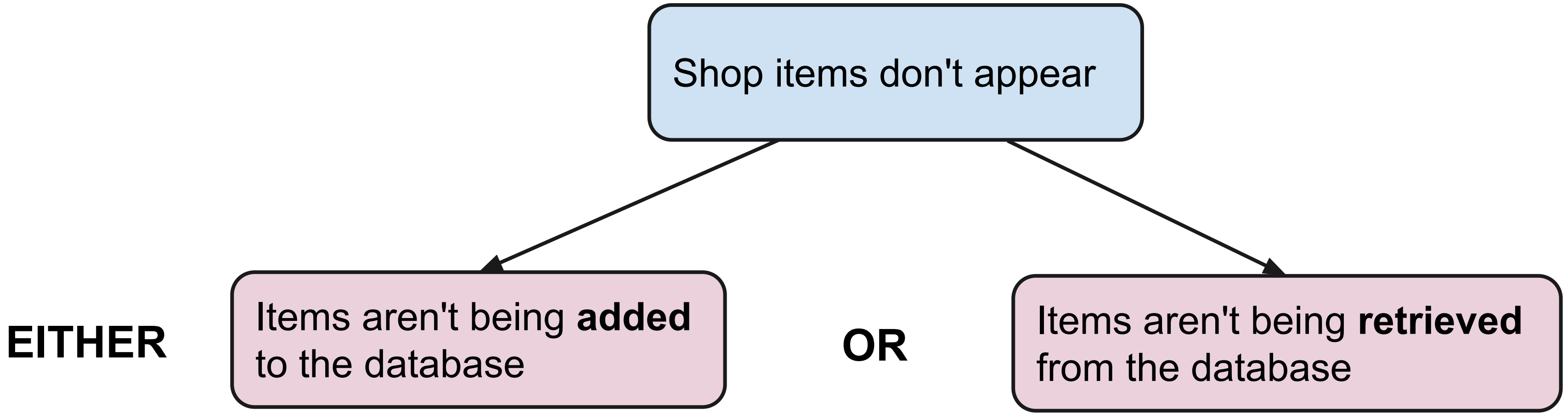
Use this technique when...

- There are **multiple areas where the error could have come from**

Divide and Conquer - Example

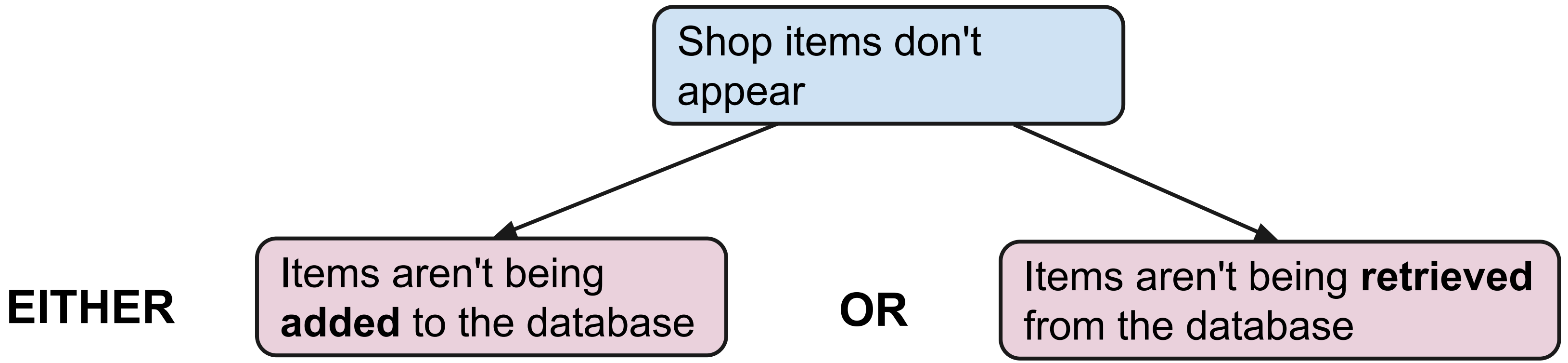
Scenario: You are writing a website for an online store. You are connecting to a database that will hold the items.

Bug: When the store owner adds a new item, it doesn't appear on the website.


Shop items don't appear
EITHER
Items aren't being **added** to the database
OR
Items aren't being **retrieved** from the database

Divide and Conquer - Example

How do we know **which of these scenarios** is causing the bug??


Shop items don't appear
EITHER
Items aren't being **added** to the database
OR
Items aren't being **retrieved** from the database

We need to **check the boundary conditions** for each of the scenarios.

In this case, we could **query the database** directly to see if the data is there.

Choose a Technique

Which technique would you use for the following scenarios?

- Your sort algorithm gives the wrong result, but there are no errors thrown.

  Trace Forward

- You make an API call to retrieve some data, but the data is not being displayed properly.

  Divide and Conquer

- When running your code, you get a NullPointerException. The stack trace doesn't immediately tell you the cause of the error.

  Trace Backward

Choose and Apply a Technique

50 minutes

Work with a partner!

For each problem in the starter code, **choose and apply a technique** that would work best for identifying and fixing the error:

- **Trace Backward**: If there is a specific line number causing the error
- **Trace Forward**: If you aren't sure which line is causing the error
- **Divide and Conquer**: If there are multiple areas which may have caused the error

It's ok to **change your mind** or **use more than one strategy**!

Break - 10 min

Credits & Additional Resources

- Deck adapted from **Debugging Effectively** (2019)