Unit 10: Debugging

Monday

Unit Overview

- **Why debugging?**
  - **Benefits & risks avoided**
- **Review of understanding a program and problem**
- **Types of errors**
- Error messages & context clues
- Narrowing down a bug
  - Breaking down large programs
  - Stepping through a program
  - Additional debugging tools and strategies

Learning Objectives

After the completion of this unit, learners will be able to:

1. Describe the different types of common programming errors
2. Classify errors based on the common error types
3. Identify context clues in an error message
4. Apply the procedure of stepping through a program to find a bug
5. Explain how to break down a large program to isolate an error
6. Determine the cause of a bug in a program

Why Debugging?

Debugging (n): The process of finding the cause of a bug or error in a program.

Review: Program scope questions

- What are some questions you can ask to understand a program?
- How can they be adapted to understand a problem or error that is occurring?

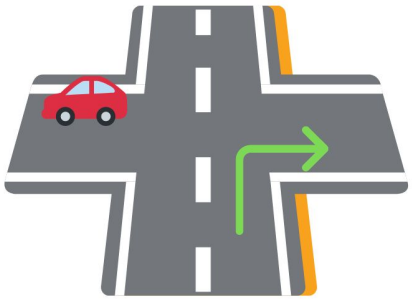
Learning Strategies

- Ask clarifying questions
- Write down what you know
- Break down the problem
- Reference handouts for tips

Types of Errors

Build error

Runtime error



Logic error

Types of Errors

Error Type	Definition	Key Features
Build Error 	An error that prevents a program from building properly.	- Can find multiple errors at once - Error message shows up where the program was built
Runtime Error 	An error that is thrown during the running of a program.	- Usually results in a stack track with filename and line number - Program will build correctly - Will only find the first error
Logic Error 	An error that is caused by an incorrect algorithm.	- Output will not match expectations - No error message - Can narrow down incorrect logic by stepping through the program

Types of Build Errors 

Error Type	Definition	Key Features
Syntax Error	An error in how you are writing your code.	- May be highlighted before building depending on the IDEs - Should contain a filename and line number
Missing imports	An error that occurs when you try to use code from another file or class that is not properly included.	- May be highlighted before building depending on the IDEs - Tells you what class is missing

Types of Runtime Errors 

Error Type	Definition	Key Features
Index out of bounds error	An error that is thrown when trying to access an item in a list that is shorter than the index being accessed.	- May contain index that was being accessed - Located in the program logs - Can be easy to find by stepping through the program
Null pointer exception	An error that is thrown when trying to access data from or call a method on an object that does not exist.	- Should contain a file name and line number - May not contain which object is null - Can be the result of not setting a value for a variable
Out of memory (OOM)	An error that is thrown when the program runs out of space to store the objects used in the program.	- Does not contain information about what objects are using the memory - Error message located in program logs - May run fine on some datasets but not others
Infinite Loop	An error that results in a program running forever unless it is canceled.	- No error message - Can be found by checking loops in the program - Easy to find by stepping through the program

Types of Logic Errors 

- Tend to be unique to the program
- Cannot be easily classified into subtypes

Types of Errors (example)

You're writing a program to sort a list of numbers. You encounter the following errors. What type of errors are they?

1. When you try to run the program with the list [1, 4, 6, 2, 3], it starts running but you get an error message that "combinedSorted" may not have been initialized before the program completes.
2. When you try the program with the list [1, 4, 6, 2, 3], you get an error message before the program runs that the symbol "inputArray" cannot be found.
3. When you try the program with the list [1, 4, 6, 2, 3], the program runs successfully but the output is [1, 2, 4, 6].

Small Group Activity

Using the descriptions of the errors your group has received and your error types handout, discuss what type of errors you think they are and why. Be prepared to report back to the class.

Small Group Activity (cont.)

- What classification did you decide on? What strategies did you use to reach that classification? Are there any strategies that you didn’t use that could have been helpful?

Big Ideas

1. Take an index card
2. Write down ~ 2-3 “big ideas” you want to remember about the types of errors and classifying them
3. Add anything else you think is important

Wednesday

Review: Monday's class

- Why is debugging important?
- What are some types of errors?

Unit Overview

- Why debugging?
  - Benefits & risks avoided
- Review of understanding a program and problem
- Types of errors
- **Error messages & context clues**
- Narrowing down a bug
  - Breaking down large programs
  - Stepping through a program
  - Additional debugging tools and strategies

Learning Strategies

- Ask clarifying questions
- Write down what you know
- Break down the problem
- Reference handouts for tips

Error messages

Build error

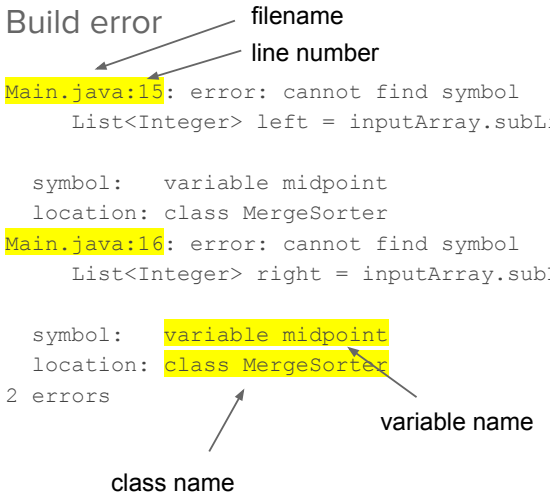
```
Main.java:15: error: cannot find symbol
     List<Integer> left = inputArray.subList(0, midpoint);
                                             ^
  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                              ^
  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

Stack trace

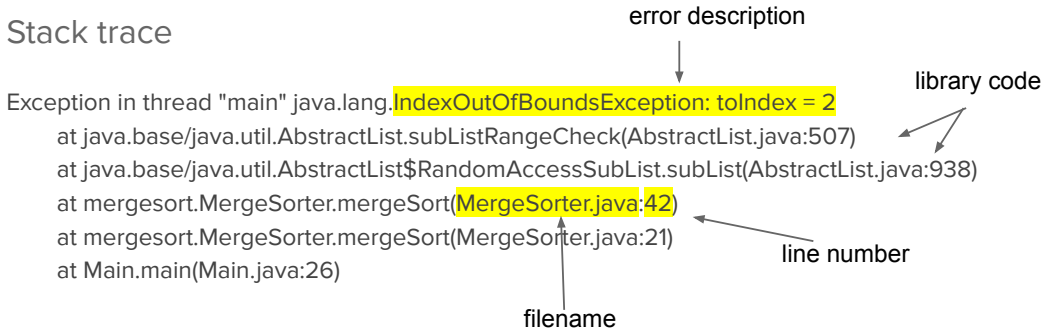
```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 4
    at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
    at java.base/java.util.ArrayList.subList(ArrayList.java:1138)
    at MergeSorter.mergeSort(Main.java:40)
    at MergeSorter.mergeSort(Main.java:19)
    at Main.main(Main.java:56)
```

Context clues

Build error



Stack trace



Context Clues (cont.)

Build error

```
Main.java:15: error: cannot find symbol
     List<Integer> left = inputArray.subList(0, midpoint);
                                             ^
  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                              ^
  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

```java
5 class MergeSorter {
6
7   List<Integer> mergeSort(List<Integer> inputArray) {
8      // There's only 1 element so return the array.
9      if (inputArray.size() < 2) {
10         return inputArray;
11      }
12
13      // Find the midpoint and split into two lists.
14      int midpoints = inputArray.size() / 2;
15      List<Integer> left = inputArray.subList(0, midpoint);
16      List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
```

Context Clues (cont.)

Build error

```
Main.java:15: error: cannot find symbol
     List<Integer> left = inputArray.subList(0, midpoint);
                                             ^
  symbol:   variable midpoint
  location: class MergeSorter
Main.java:16: error: cannot find symbol
     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
                                              ^
  symbol:   variable midpoint
  location: class MergeSorter
2 errors
```

```
5  class MergeSorter {
6
7    List<Integer> mergeSort(List<Integer> inputArray) {
8      // There's only 1 element so return the array.
9      if (inputArray.size() < 2) {
10         return inputArray;
11     }
12
13     // Find the midpoint and split into two lists.
14     int midpoints = inputArray.size() / 2;
15     List<Integer> left = inputArray.subList(0, midpoint);
16     List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
```

Context Clues (cont.)

Stack trace

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 2
    at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
    at java.base/java.util.AbstractList$RandomAccessSubList.subList(AbstractList.java:938)
    at mergesort.MergeSorter.mergeSort(MergeSorter.java:42)
    at mergesort.MergeSorter.mergeSort(MergeSorter.java:21)
    at Main.main(Main.java:26)
```

```java
package mergesort;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MergeSorter {

  public List<Integer> mergeSort(List<Integer> inputArray) {
.
.
.


    // Add the remaining items in the list where the end was not reached.
    if (leftIndex == leftSorted.size()) {
        combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size() + 1));
    } else {
        combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
    }

    return combinedSorted;
  }
}
```

Context Clues (cont.)

Stack trace

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: toIndex = 2
    at java.base/java.util.AbstractList.subListRangeCheck(AbstractList.java:507)
    at java.base/java.util.AbstractList$RandomAccessSubList.subList(AbstractList.java:938)
    at mergesort.MergeSorter.mergeSort(MergeSorter.java:42)
    at mergesort.MergeSorter.mergeSort(MergeSorter.java:21)
    at Main.main(Main.java:26)
```

```
1 package mergesort;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class MergeSorter {
8
9   public List<Integer> mergeSort(List<Integer> inputArray) {
.
.
.
39
40      // Add the remaining items in the list where the end was not reached.
41      if (leftIndex == leftSorted.size()) {
42          combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size() + 1));
43      } else {
44          combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
45      }
46
47      return combinedSorted;
48   }
49 }
```

Small Group Activity

Using the error messages your group has received and your context clues handout, find as many context clues as you can and discuss how each can be used. Be prepared to report back to the class.

Small Group Activity (cont.)

- What were some context clues you identified? How can they be used to find the error? Are there any context clues or strategies that you didn't use that could have been helpful?

Big Ideas

1. Take an index card
2. Write down ~ 2-3 “big ideas” you want to remember about error messages and context clues
3. Add anything else you think is important

Friday

Review: Monday's and Wednesday's classes

- Why is debugging important?
- What are some types of errors?
- What are error messages?
- What are some examples context clues?

Unit Overview

- Why debugging?
  - Benefits & risks avoided
- Review of understanding a program and problem
- Types of errors
- Error messages & context clues
- **Narrowing down a bug**
  - **Breaking down large programs**
  - **Stepping through a program**
  - **Additional debugging tools and strategies**

Learning Strategies

- Ask clarifying questions
- Write down what you know
- Break down the problem
- Reference handouts for tips

Splitting Large Problems

LetterSorter.java × Main.java × MergeSorter.java ×

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import lettersort.LetterSorter;
import mergesort.MergeSorter;

class Main {
  public static void main(String args[]) {

    List<String> input = Arrays.asList("c", "b", "e", "A", "M", "i");

    LetterSorter letterSorter = new LetterSorter();
    MergeSorter mergeSorter = new MergeSorter();

    List<Integer> numberList = letterSorter.convertLettersToNumbers(input);
    List<Integer> sortedList = mergeSorter.mergeSort(numberList);
    List<String> sortedLetterList = letterSorter.convertNumbersToLetters(sortedList);

    System.out.println("Sorted Array:");
    System.out.println(sortedLetterList.toString());
  }
}
```

- Functions
- Loops
- Conditional statements
- Files
- Classes

Testing Sub-components

- convertLettersToNumbers(List<String> letterList)
  - Input: ["c", "b", "e", "A", "M", "i"]
  - Output: [2, 1, 4, 26, 38, 8]
- mergeSort(List<Integer> inputArray)
  - Input: [2, 1, 4, 26, 38, 8]
  - Output: [1, 2, 4, 26, 38, 8]
- convertNumbersToLetters(List<Integer> numberList)
  - Input: [1, 2, 4, 26, 38, 8]
  - Output: [b, c, e, A, M, i]

```
CHAR_LIST = Arrays.asList(
      "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",
      "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
      "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
      "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z")
```

Stepping Through

```
22      // Merge the sorted lists by keeping a pointer to the first unused number in each list.
23      // Compare the first unused item in each list and add the lowest to the combined list.
24      // Stop when the end of either list has been reached.
25      ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
26      int leftIndex = 0;
27      int rightIndex = 0;
28      while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
29          if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
30              combinedSorted.add(leftSorted.get(leftIndex));
31          } else {
32              combinedSorted.add(rightSorted.get(rightIndex));
33              rightIndex++;
34          }
35      }
```

leftSorted = [1, 2, 4], rightSorted = [8, 26, 38]

Small Group Activity

Using the error description, error message, and program code your group has received find the error in the program. Practice breaking the problem down and stepping through the code. Be prepared to report back to the class.

Small Group Activity (cont.)

- Were you able to find the error? What strategies did you use to find the error? Are there any strategies that you didn't use that could have been helpful?

Debugging Tools & Other Strategies

- Debuggers
- Diffs

Debuggers

```
5   class MergeSorter {
6
7     List<Integer> mergeSort(List<Integer> inputArray) {
8       // There's only 1 element so return the array.
9       if (inputArray.size() < 2){
10          return inputArray;
11      }
12
13      // Find the midpoint and split into two lists.
14      int midpoint = inputArray.size() / 2;
15      List<Integer> left = inputArray.subList(0, midpoint);
16      List<Integer> right = inputArray.subList(midpoint, inputArray.size());
17
18      // Sort both lists.
19      List<Integer> leftSorted = mergeSort(left);
20      List<Integer> rightSorted = mergeSort(right);
21
22      // Merge the sorted lists by keeping a pointer to the first unused number in each list.
23      // Compare the first unused item in each list and add the lowest to the combined list.
24      // Stop when the end of either list has been reached.
25      ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
26      int leftIndex = 0;
27      int rightIndex = 0;
28      while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size() + 1) {
```

.
.
.

inputArray = [6, 5, 12, 10, 9, 1, 7]
midpoint = 3
left = [6, 5, 12]
right = [10, 9, 1, 7]

.
.
.

inputArray = [6, 5, 12]
midpoint = 1
left = [6]
right = [5, 12]

Diff

```
    // Find the midpoint and split into two lists.
    int midpoint = inputArray.size() / 2;
    List<Integer> left = inputArray.subList(0, midpoint);
    List<Integer> right = inputArray.subList(midpoint, inputArray.size());

    // Sort both lists.
    List<Integer> leftSorted = mergeSort(left);
    List<Integer> rightSorted = mergeSort(right);

    // Merge the sorted lists by keeping a pointer to the first unused number in each list.
    // Compare the first unused item in each list and add the lowest to the combined list.
    // Stop when the end of either list has been reached.
    ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
    int leftIndex = 0;
    int rightIndex = 0;


    while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size() + 1) {
        if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
            combinedSorted.add(leftSorted.get(leftIndex));
            leftIndex++;
        } else {
            combinedSorted.add(rightSorted.get(rightIndex));
            rightIndex++;
        }
    }

    // Add the remaining items in the list where the end was not reached.
    if (leftIndex == leftSorted.size()) {
        combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
    } else {
        combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
    }
```

```
    // Find the midpoint and split into two lists.
    int midpoint = inputArray.size() / 2;
    List<Integer> left = inputArray.subList(0, midpoint + 1);
    List<Integer> right = inputArray.subList(midpoint, inputArray.size());

    // Sort both lists.
    List<Integer> leftSorted = mergeSort(left);
    List<Integer> rightSorted = mergeSort(right);

    // Merge the sorted lists by keeping a pointer to the first unused number in each list.
    // Compare the first unused item in each list and add the lowest to the combined list.
    // Stop when the end of either list has been reached.
    ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
    int leftIndex = 0;
    int rightIndex = 0;
    int leftSize = leftSorted.size();
    int rightSize = rightSorted.size();
    while (leftIndex < leftSize) && rightIndex < rightSize + 1) {
        if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
            combinedSorted.add(leftSorted.get(leftIndex));

        } else {
            combinedSorted.add(rightSorted.get(rightIndex));
            rightIndex++;
        }
    }

    // Add the remaining items in the list where the end was not reached.
    if (leftIndex == leftSorted.size()) {
        combinedSorted.addAll(rightSorted.subList(rightIndex, right.size()));
    } else {
        combinedSorted.addAll(leftSorted.subList(leftIndex, left.size()));
    }
```

Big Ideas

1. Take an index card
2. Write down ~ 2-3 “big ideas” you want to remember about debugging large or complex programs
3. Add anything else you think is important

Advance Organizer for Unit 11: Linked Lists

1. Defining the problem scope
2. Algorithms
3. Testing
4. Data - Variables & primitive types
5. Conditionals
6. Loops
7. Data - Arrays
8. Functions
9. Classes
10. Debugging
11. **Data - Linked lists**
    a. Definition of a linked list
    b. Comparison to arrays (covered in unit 7)
    c. Types of linked lists (e.g. single linked list, doubly linked list, circular linked list)
    d. Advantages and disadvantages of linked lists
    e. When to use linked lists
12. Data - Maps
13. Data - Stacks & queues
14. Real world problems