Inf1B

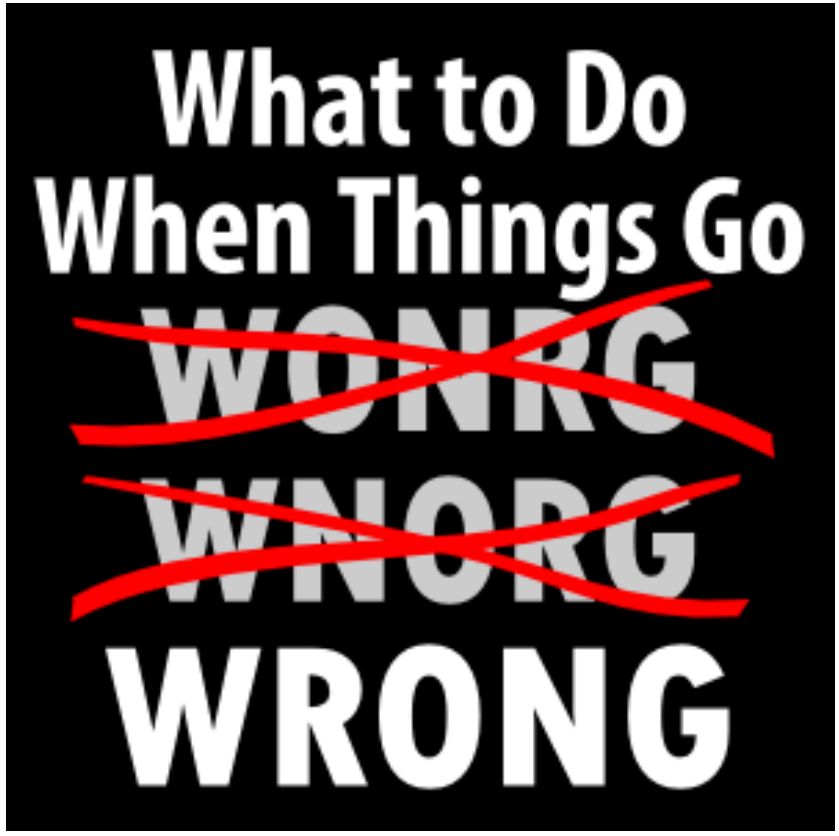
Testing and Debugging

Volker Seeker

School of Informatics

January 29, 2020

Things will go wrong



There is usually an error in your code somewhere.

https://4pyz335b69-flywheel.netdna-ssl.com/wp-content/uploads/2014/05/things-go-wrong.png

Types of Errors

Ordered by difficulty to detect and fix them.

- Syntax Errors
- Runtime Errors
- Logical Errors

Syntax Errors

Comparable to a spelling mistake in a text.

This is a speling mistake!

```
int value = 5;
if (value < 10
    Systm.out.println("Here we are.")
```

Syntax Errors

Comparable to a spelling mistake in a text.

This is a speling mistake!

```
int value = 5;
if (value < 10
    Systm.out.println("Here we are.")
```

An IDE can help you detect them.

Syntax Errors

Syntax errors are detected at compile time.

Compiler Output

```
Main.java:5: error: ')' expected
        if (value < 10
                    ^
Main.java:6: error: ';' expected
           Systm.out.println("Here we are.")
                                             ^
2 errors
```

Syntax Errors

Not always easy to identify despite compiler and IDE help.

```
1  public class Main {
2      public static int add(int a, int b) {
3          return a + b;
4
5      public static void main(String[] args) {
6          System.out.println(add(5,5));
7      }
8  }
```

Compiler Output

```
Main.java:5: error: illegal start of expression
    public static void main(String[] args) {
    ^
1 error
```

Runtime Errors

Comparable to a grammar mistake in a text.

There taking they're kids their.

```
int[] arr = { 1, 2, 3, 4 };
System.out.println(arr[4]);
```

Runtime Errors

Comparable to a grammar mistake in a text.

There taking they're kids their.

```
int[] arr = { 1, 2, 3, 4 };
System.out.println(arr[4]);
```

Compiler and IDE are unable to detect them.

Runtime Errors

The Java Runtime will detect them and crash your program.

```
int[] arr = { 1, 2, 3, 4 };
System.out.println(arr[4]);
```

Runtime Output

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 4 out of bounds for length 4
at Main.main(Main.java:5)
```

Logical Errors

Comparable to an incorrect or unintended statement in a text.

The swan is an orange bird.

```
public static int add(int a, int b) {
    return a - b;
}
```

Logical Errors

Comparable to an incorrect or unintended statement in a text.

The swan is an orange bird.

```
public static int add(int a, int b) {
    return a - b;
}
```

Neither compiler, nor IDE or Java Runtime can detect them.

Logical Errors

You need to test your code to catch them.

```
public static int add(int a, int b) {
    return a - b;
}

public static void main(String[] args) {
    if (add(5,5) != 10)
        System.out.println("Unexpected sum!");
}
```

Types of Errors

Ordered by difficulty to detect and fix them.

- Syntax Errors
- Runtime Errors
- Logical Errors

Types of Errors

Ordered by difficulty to detect and fix them.

- Syntax Errors **Caught at compile time**
- Runtime Errors
- Logical Errors

Types of Errors

Ordered by difficulty to detect and fix them.

- Syntax Errors **Caught at compile time**
- Runtime Errors **Caught at runtime**
- Logical Errors

Types of Errors

Ordered by difficulty to detect and fix them.

- Syntax Errors **Caught at compile time**
- Runtime Errors **Caught at runtime**
- Logical Errors **Caught via testing**

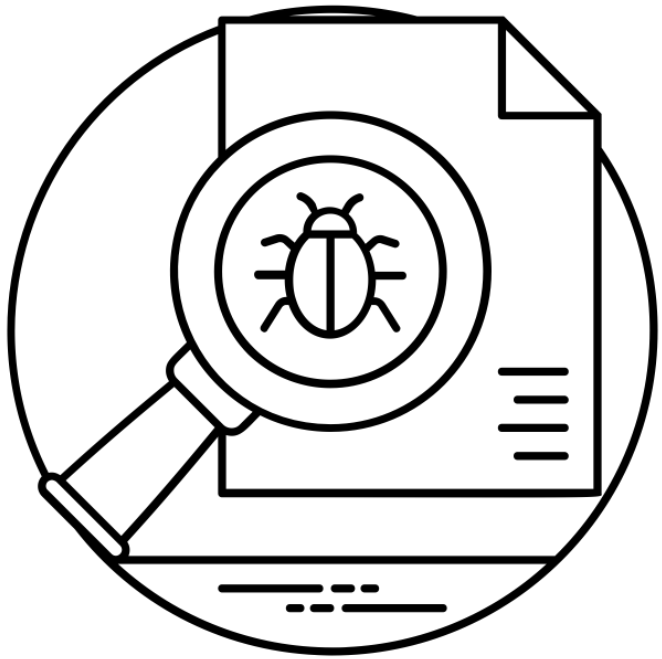
Types of Errors

Ordered by difficulty to detect and fix them.

- Syntax Errors **Caught at compile time**
- Runtime Errors **Caught at runtime**
- Logical Errors **Caught via testing**

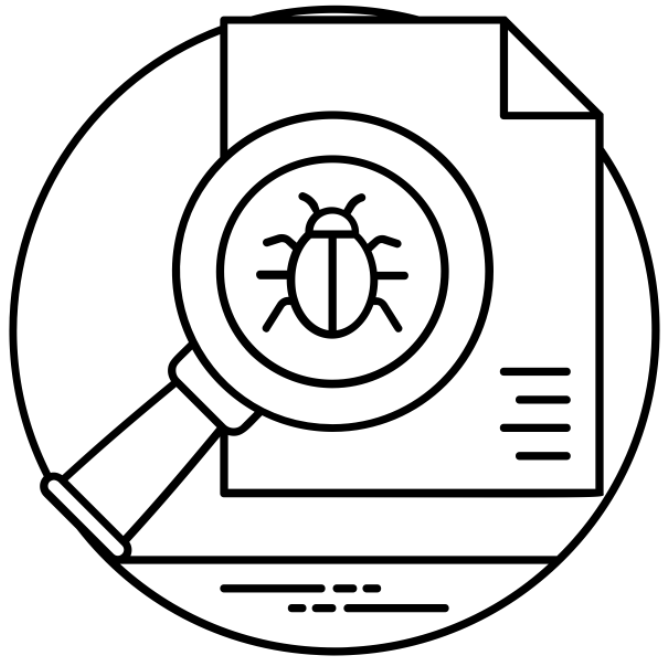
NB Since tests execute your code, they will also catch runtime errors.

Let's hunt some bugs!



Created by Vectors Market
from Noun Project

Let's hunt some bugs!

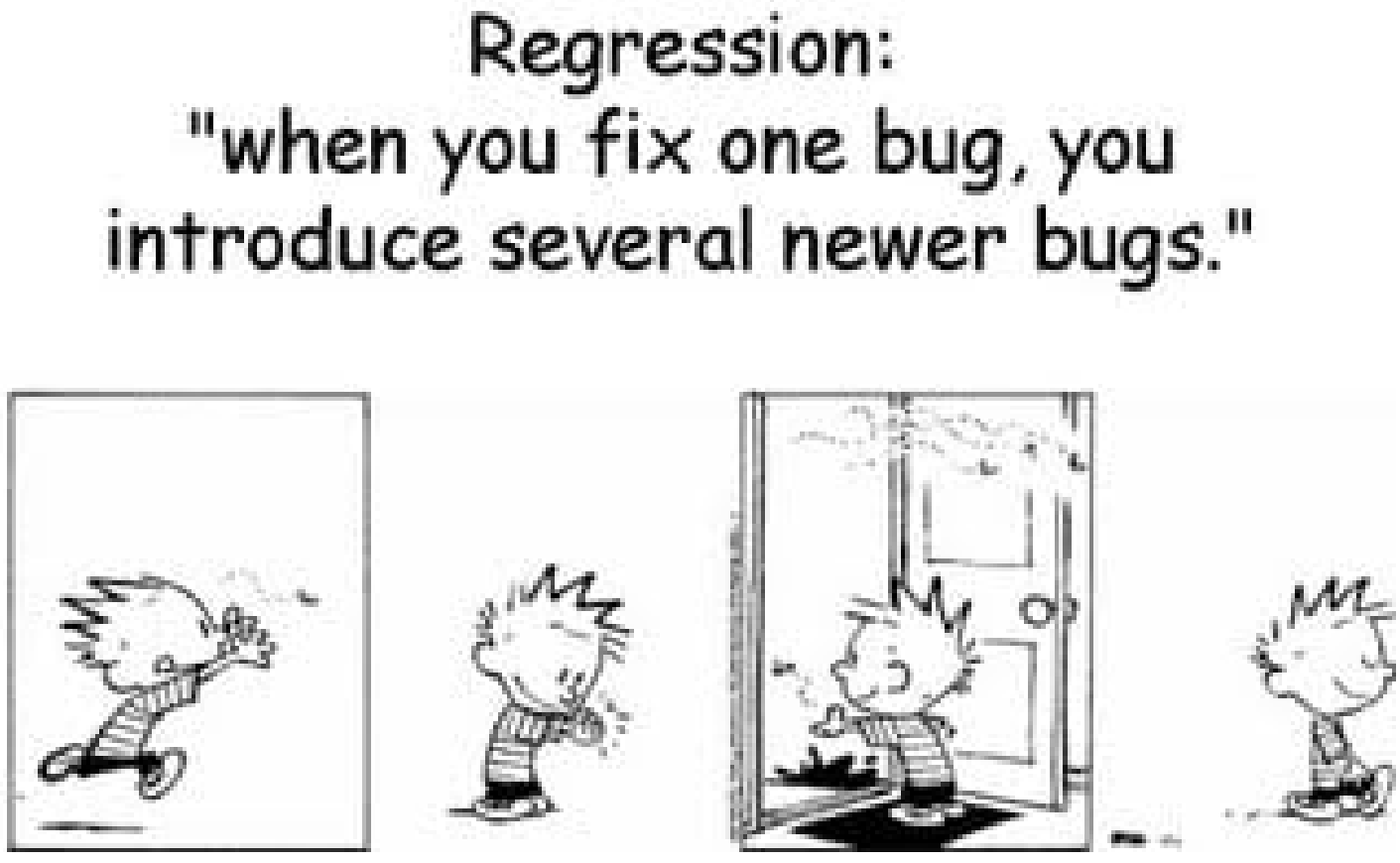


Created by Vectors Market
from Noun Project

1. Testing *detect the errors*
2. Debugging *find and fix the errors*

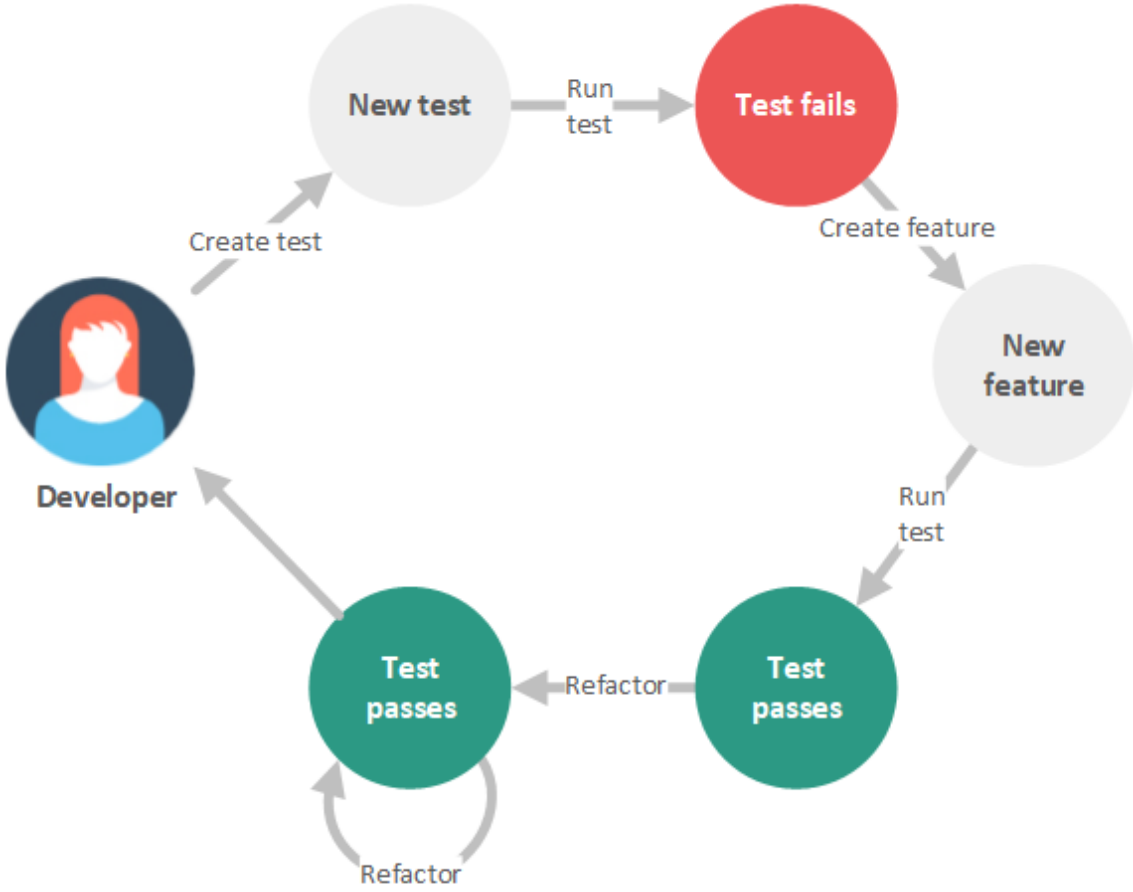
Testing

Regression Testing



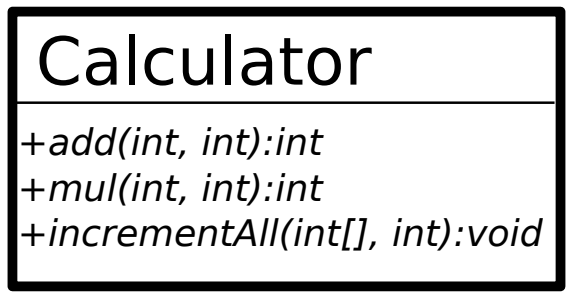
Source: https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/

Test Driven Development



Source: https://dzone.com/articles/what-is-refactoring

Simple Calculator



Implement a utility class with calculator functionality.

How would you test the functionality of a class?

Demo

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

They have, however, a few drawbacks:

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

They have, however, a few drawbacks:

- Using console output to evaluate test results requires manual effort and is error prone for more complex tests

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

They have, however, a few drawbacks:

- Using console output to evaluate test results requires manual effort and is error prone for more complex tests

  $\rightarrow$ use assertions instead!

Automatic evaluation with assertions

Demo

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

They have, however, a few drawbacks:

- Using console output to evaluate test results requires manual effort and is error prone for more complex tests

  $\rightarrow$ use assertions instead!

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

They have, however, a few drawbacks:

- Using console output to evaluate test results requires manual effort and is error prone for more complex tests

  $\rightarrow$ use assertions instead!
- tests are unorganised, no easy way to test only certain methods

Main Method as Test Client

Main methods can be used to quickly evaluate the functionality of your code.

They have, however, a few drawbacks:

- Using console output to evaluate test results requires manual effort and is error prone for more complex tests

  $\rightarrow$ use assertions instead!
- tests are unorganised, no easy way to test only certain methods

  $\rightarrow$ use a test framework instead!

Organising Tests with a Test Framework

Demo

Testing Strategies

- test for regular use cases
- test for corner cases
- test for invalid input (how should it be handled?)
- positive testing vs. negative testing

Debugging

Manual walk through

Something is wrong with this array rotation code.

```
1  int[] arr = { 1, 2, 3, 4, 5 };
2  int tmp = arr[arr.length - 1];
3  for (int i = 0; i < arr.length - 1; i++) {
4      arr[i + 1] = arr[i];
5  }
6  arr[0] = tmp;
```

Let's find out what without the help of machines.

Logging

With Compiler and Runtime, we can use a logging approach.

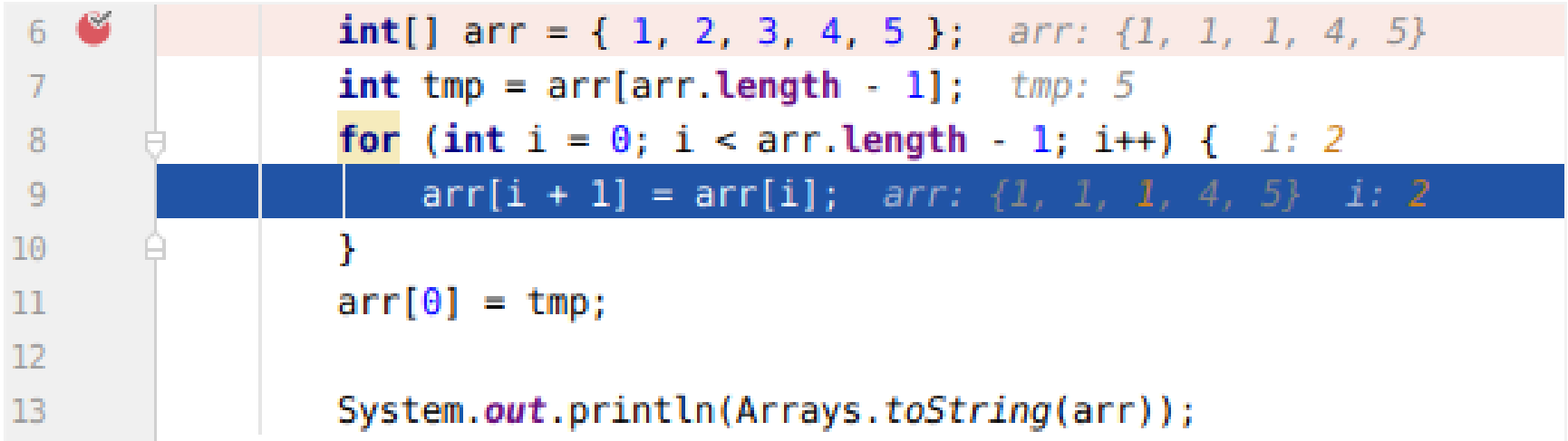
```
int[] arr = { 1, 2, 3, 4, 5 };
int tmp = arr[arr.length - 1];
for (int i = 0; i < arr.length - 1; i++) {
    arr[i + 1] = arr[i];
    System.out.println(Arrays.toString(arr));
}
arr[0] = tmp;
System.out.println(Arrays.toString(arr)):
```

Output

```
[1, 1, 3, 4, 5]
[1, 1, 1, 4, 5]
[1, 1, 1, 1, 5]
[1, 1, 1, 1, 1]
[5, 1, 1, 1, 1]
```

Using a Debugger

With the help of a debugger, we can get a lot of information without much effort from our side.



Demo

Debugging Strategies

- Manual Walk Through
- Logging
- Debugger

Bug Hunting

1. Testing *detect the errors*
2. Debugging *find and fix the errors*

Bug Hunting

0. Write Robust and Maintainable Code

avoid errors in the first place

1. Testing *detect the errors*
2. Debugging *find and fix the errors*

Error Handling

Handling Invalid Input

Given a function that generates a sequence of numbers ...

What could go wrong here?

```
1  public static int[] sequence(int start, int end) {
2   int[] result = new int[end - start];
3   int index = 0;
4   while (start < end) {
5    result[index++] = start++;
6   }
7   return result;
8  }
```

Handling Invalid Input

Given a function that generates a sequence of numbers ...

What could go wrong here?

```
1  public static int[] sequence(int start, int end) {
2   int[] result = new int[end - start];
3   int index = 0;
4   while (start < end) {
5    result[index++] = start++;
6   }
7   return result;
8  }
```

Start could be smaller than end!

Handling Invalid Input

Given a function that generates a sequence of numbers ...

What could go wrong here?

```
1  public static int[] sequence(int start, int end) {
2   int[] result = new int[end - start];
3   int index = 0;
4   while (start < end) {
5    result[index++] = start++;
6   }
7   return result;
8  }
```

Start could be smaller than end!

How could we handle this best?

Handling Invalid Input

Make a note in the function documentation.

```
1  /** Start must always be smaller or equal to end! */
2  public static int[] sequence(int start, int end) {
3   int[] result = new int[end - start];
4   int index = 0;
5   while (start < end) {
6    result[index++] = start++;
7   }
8   return result;
9  }
```

Handling Invalid Input

Make a note in the function documentation.

```
1  /** Start must always be smaller or equal to end! */
2  public static int[] sequence(int start, int end) {
3   int[] result = new int[end - start];
4   int index = 0;
5   while (start < end) {
6    result[index++] = start++;
7   }
8   return result;
9  }
```

Helpful but not a good way to enforce rules.

Handling Invalid Input

Add a check and print an error message.

```
1  /** Start must always be smaller or equal to end! */
2  public static int[] sequence(int start, int end) {
3   if (start > end)
4    System.err.println("ERROR: Start must be smaller end!");
5
6   int[] result = new int[end - start];
7   int index = 0;
8   while (start < end) {
9    result[index++] = start++;
10  }
11  return result;
12 }
```

Handling Invalid Input

Add a check and print an error message.

```
1  /** Start must always be smaller or equal to end! */
2  public static int[] sequence(int start, int end) {
3   if (start > end)
4    System.err.println("ERROR: Start must be smaller end!");
5
6   int[] result = new int[end - start];
7   int index = 0;
8   while (start < end) {
9    result[index++] = start++;
10  }
11  return result;
12 }
```

More helpful, but this will still crash.

Handling Invalid Input

For internal code during development, a crash might be sufficient. But you should use an assertion in that case.

```
1  /** Start must always be smaller or equal to end! */
2  public static int[] sequence(int start, int end) {
3   assert start < end : "Start must be smaller end.";
4
5   int[] result = new int[end - start];
6   int index = 0;
7   while (start < end) {
8    result[index++] = start++;
9   }
10  return result;
11 }
```

Handling Invalid Input

For internal code during development, a crash might be sufficient. But you should use an assertion in that case.

```
1  /** Start must always be smaller or equal to end! */
2  public static int[] sequence(int start, int end) {
3   assert start < end : "Start must be smaller end.";
4
5   int[] result = new int[end - start];
6   int index = 0;
7   while (start < end) {
8    result[index++] = start++;
9   }
10  return result;
11 }
```

Not enough for publicly exposed function used by others.

Handling Invalid Input

Return an error value.

```
1  /** Start must always be smaller or equal to end!
2   * Null will be returned otherwise. */
3  public static int[] sequence(int start, int end) {
4   if (start > end) return null;
5
6   int[] result = new int[end - start];
7   int index = 0;
8   while (start < end) {
9    result[index++] = start++;
10  }
11  return result;
12 }
```

Handling Invalid Input

Return an error value.

```
1  /** Start must always be smaller or equal to end!
2   * Null will be returned otherwise. */
3  public static int[] sequence(int start, int end) {
4   if (start > end) return null;
5
6   int[] result = new int[end - start];
7   int index = 0;
8   while (start < end) {
9    result[index++] = start++;
10  }
11  return result;
12 }
```

This will avoid the error and report it to the calling code but it does not always work.

Handling Invalid Input

Return an error value.

```
1  public static int sum(int[] data) {
2      if (data.length == 0) return ??????
3
4      int result = 0;
5      for(int d : data) {
6          result += data;
7      }
8      return data;
9  }
```

This will avoid the error and report it to the calling code but it does not always work.

Handling Invalid Input

Throw an Exception.

```
1   /** Start must always be smaller or equal to end!
2    * IllegalArgumentException is thrown otherwise. */
3   public static int[] sequence(int start, int end) {
4    if (start > end)
5        throw new IllegalArgumentException("Start must be smaller end.");
6
7    int[] result = new int[end - start];
8    int index = 0;
9    while (start < end) {
10    result[index++] = start++;
11   }
12   return result;
13  }
```

This reports the error without contaminating the return value.

There is a short exercise on handling errors in the calling code in the labs.

Inf1B Coding Conventions:

For private methods:
Use assertions if it helps you during development.

For public methods:

- Note error handling in the documentation.
- Throw IllegalArgumentExceptions for illegal arguments.
- Throw NullPointerExceptions for null arguments.
- If explicitly stated: handle via return value.

Summary

- Three types of errors:
  syntax, *runtime* and *logical*
- Three testing strategies:
  main, *assert*, *unit*
- Three debugging strategies:
  manual, *print*, *debugger*
- Three ways for error handling:
  assert, *return*, *exception*

Reading

Objects First

Chapter 9 (some *BlueJ* specifics and techniques I have not yet fully taught you but good examples. Feel free to ignore functional bit.)

Java Tutorial

Chapter 10 (Mostly about exceptions and exception handling.)