Principles of Software Construction: Objects, Design, and Concurrency (Part 1: Designing Classes)

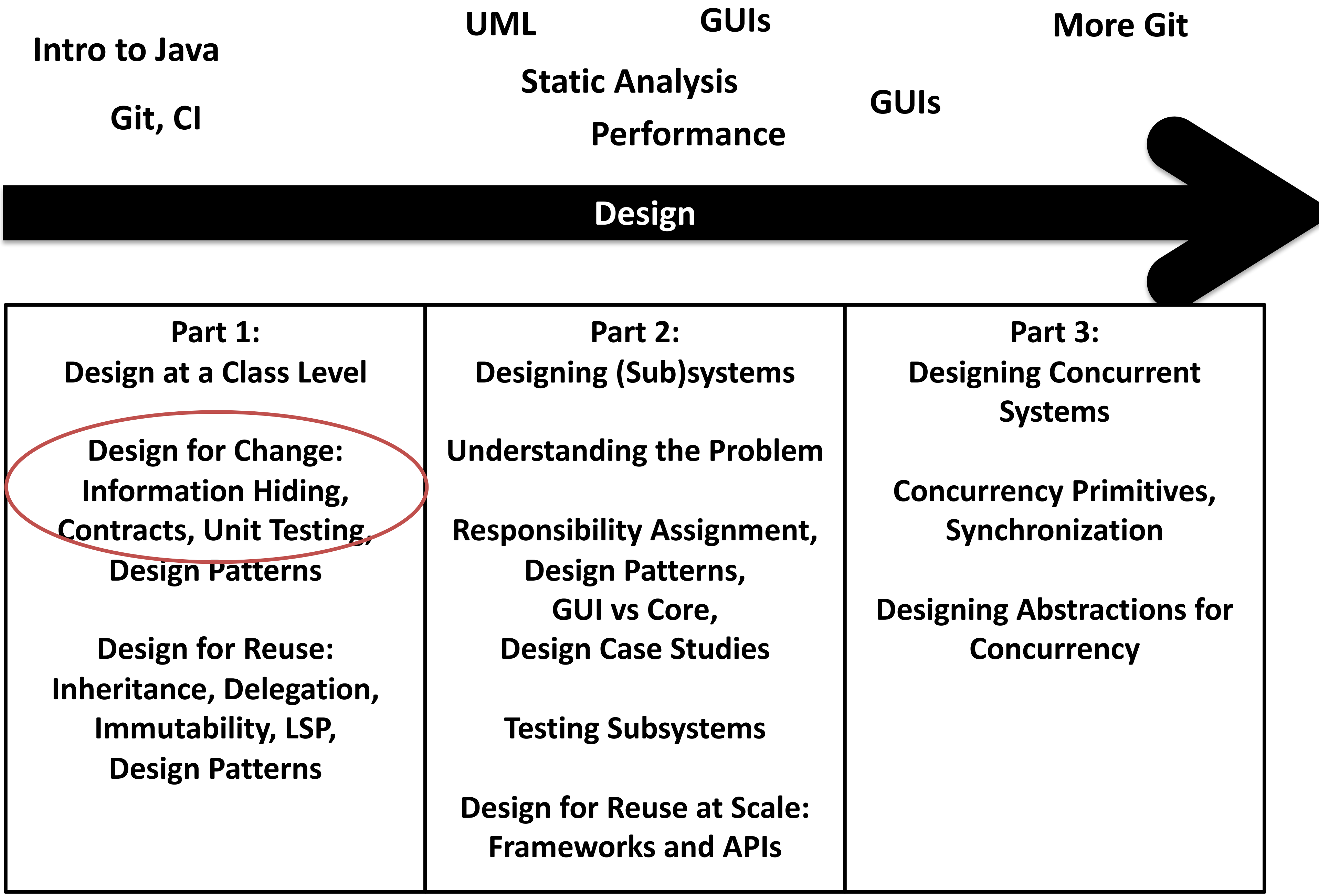
Design for Change (class level)

Charlie Garrod **Bogdan Vasilescu**

School of Computer Science

Administrivia

- Homework 1 due on Thursday 11:59pm
  - Everyone must read and sign our collaboration policy
- Reading assignment due today
- (Optional) reading due Thursday
- Next reading due next Tuesday



Intro to Java

Git, CI

UML

Static Analysis

Performance

GUIs

GUIs

More Git

Design

Part 1: Design at a Class Level	Part 2: Designing (Sub)systems	Part 3: Designing Concurrent Systems
Design for Change: Information Hiding, Contracts, Unit Testing, Design Patterns	Understanding the Problem	Concurrency Primitives, Synchronization
Design for Reuse: Inheritance, Delegation, Immutability, LSP, Design Patterns	Responsibility Assignment, Design Patterns, GUI vs Core, Design Case Studies	Designing Abstractions for Concurrency
	Testing Subsystems	
	Design for Reuse at Scale: Frameworks and APIs	

Design Goals for Today

- **Design for Change** (flexibility, extensibility, modifiability)

also

- Design for Division of Labor
- Design for Understandability
- Design for Reuse (more later)

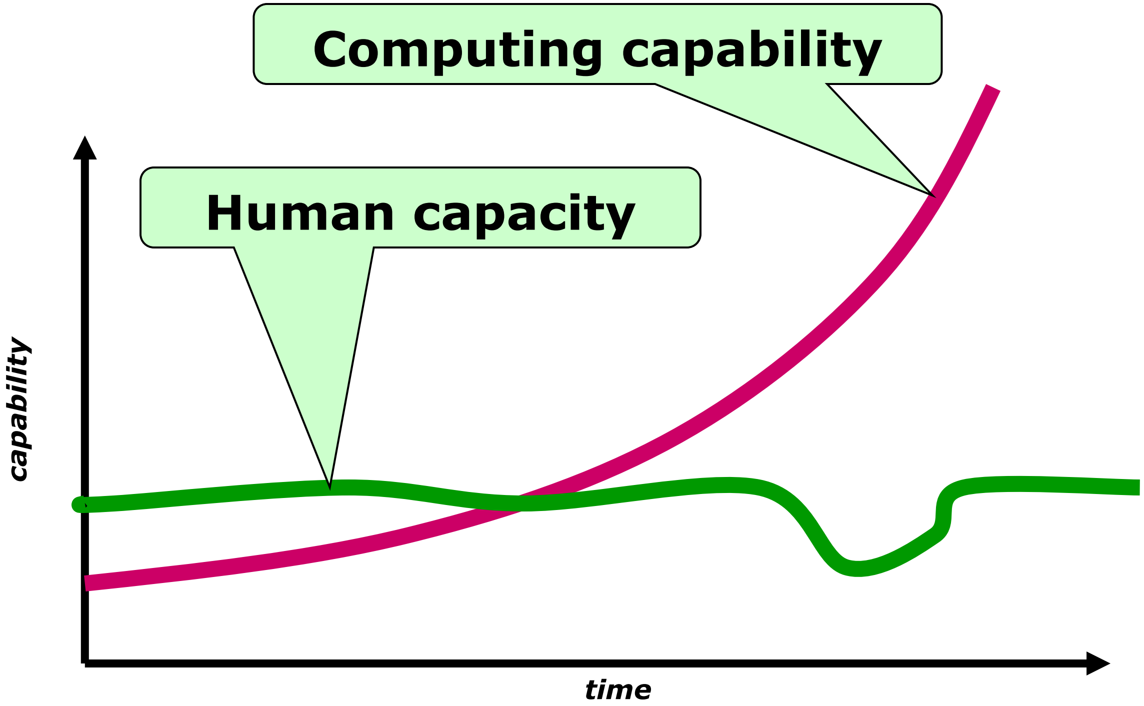
Software Change

- …accept the fact of change as a way of life, rather than an untoward and annoying exception.
  —Brooks, 1974

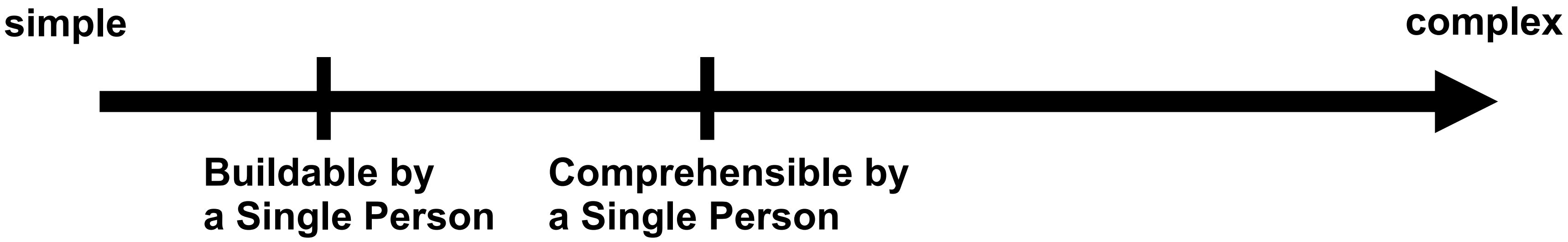
- Software that does not change becomes useless over time.
  —Belady and Lehman

- For successful software projects, most of the cost is spent evolving the system, not in initial development
  - Therefore, reducing the cost of change is one of the most important principles of software design

The limits of exponentials



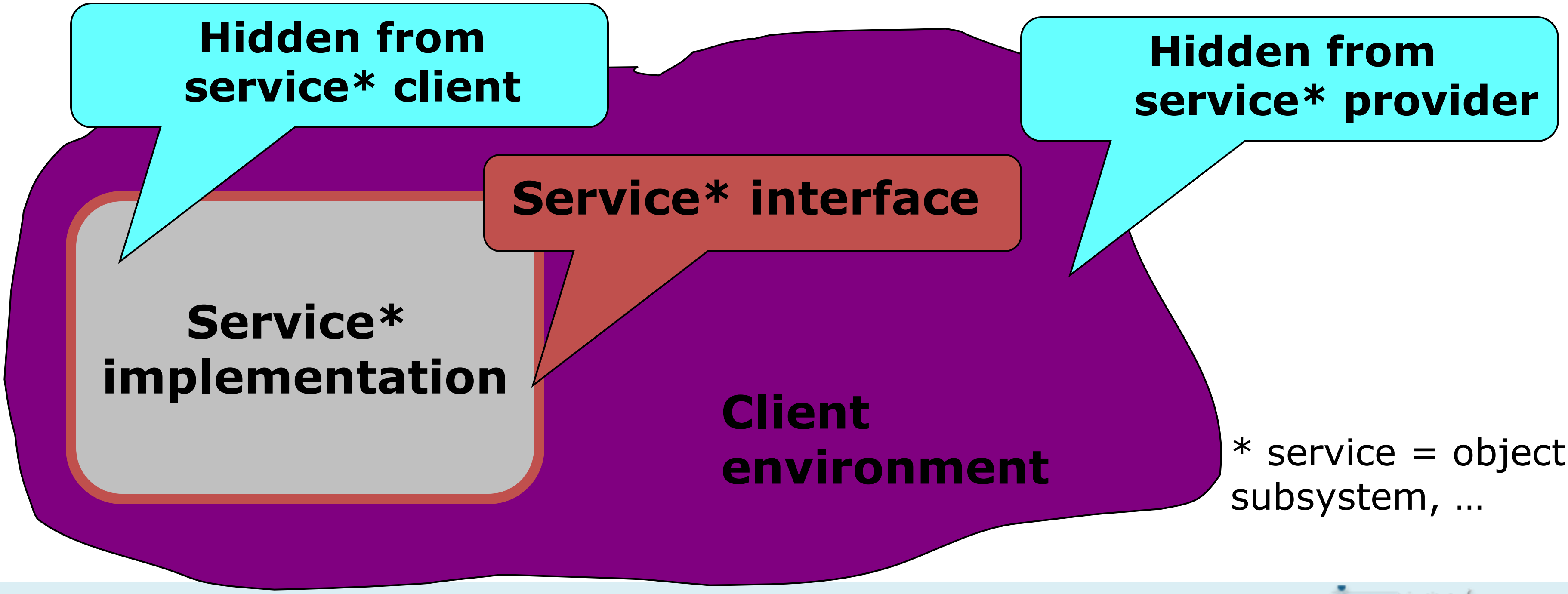
Building Complex Systems



- Division of Labor
- Division of Knowledge and Design Effort
- Reuse of Existing Implementations

Fundamental Design Principle for Change: Information Hiding

- Expose as little implementation detail as necessary
- Allows to change hidden details later



* service = object, subsystem, …

Information hiding

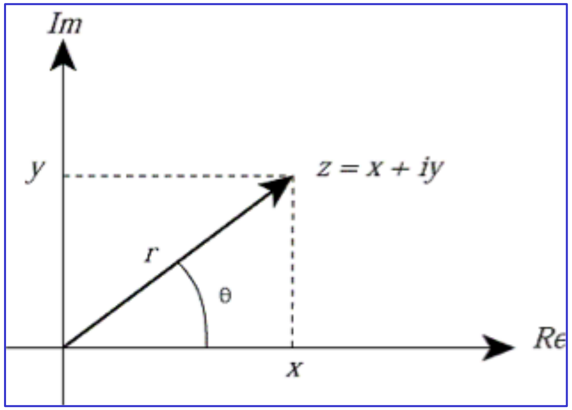
- Visibility modifiers in Java ("encapsulation")
  - private
  - "package private"
  - protected
  - public
- Interface types vs. class types

Information hiding is more general than visibility

- Use interfaces to separate expectations from implementation
  - Create interfaces to define your API
  - Declare variables, arguments, and return values as interface type
    - Write API in terms of other interfaces, not implementations
- Do not publicly document implementation details

Review

A more complex example



```
public class Complex {
  private final double re;  // Real part
  private final double im;  // Imaginary part

  public Complex(double re, double im) {
    this.re = re;
    this.im = im;
  }

  public double realPart()      { return re; }
  public double imaginaryPart() { return im; }
  public double r()             { return Math.sqrt(re * re + im * im); }
  public double theta()         { return Math.atan(im / re); }

  public Complex plus(Complex c) {
    return new Complex(re + c.re, im + c.im);
  }
  public Complex minus(Complex c) { ... }
  public Complex times(Complex c) { ... }
  public Complex dividedBy(Complex c)   { ... }
}
```

Using the `Complex` class

```
public class ComplexUser {
    public static void main(String args[]) {
        Complex c = new Complex(-1, 0);
        Complex d = new Complex(0, 1);

        Complex e = c.plus(d);
        System.out.println(e.realPart() + " + "
                           + e.imaginaryPart() + "i");
        e = c.times(d);
        System.out.println(e.realPart() + " + "
                           + e.imaginaryPart() + "i");
    }
}
```

When you run this program, it prints

```
-1.0 + 1.0i
-0.0 + -1.0i
```

Review

Extracting an interface from our class

```
public interface Complex {
    // No constructors, fields, or implementations!

    double realPart();
    double imaginaryPart();
    double r();
    double theta();

    Complex plus(Complex c);
    Complex minus(Complex c);
    Complex times(Complex c);
    Complex dividedBy(Complex c);
}
```

An interface defines but does not implement API

Review

Modifying our earlier class to use the interface

```
public class OrdinaryComplex implements Complex {
  private final double re;  // Real part
  private final double im;  // Imaginary part

  public OrdinaryComplex(double re, double im) {
    this.re = re;
    this.im = im;
  }

  public double realPart()      { return re; }
  public double imaginaryPart() { return im; }
  public double r()             { return Math.sqrt(re * re + im * im); }
  public double theta()         { return Math.atan(im / re); }

  public Complex plus(Complex c) {
    return new OrdinaryComplex(re + c.realPart(), im + c.imaginaryPart());
  }
  public Complex minus(Complex c) { ... }
  public Complex times(Complex c) { ... }
  public Complex dividedBy(Complex c)   { ... }
}
```

Review

Modifying our earlier client to use the interface

```
public class ComplexUser {
    public static void main(String args[]) {
        Complex c = new OrdinaryComplex(-1, 0);
        Complex d = new OrdinaryComplex(0, 1);

        Complex e = c.plus(d);
        System.out.println(e.realPart() + " + "
                           + e.imaginaryPart() + "i");
        e = c.times(d);
        System.out.println(e.realPart() + " + "
                           + e.imaginaryPart() + "i");
    }
}
```

When you run this program, it still prints

```
-1.0 + 1.0i
-0.0 + -1.0i
```

Review

Interfaces permit multiple implementations

```
public class PolarComplex implements Complex {
  private final double r;      // Radius
  private final double theta;  // Angle

  public PolarComplex(double r, double theta) {
    this.r = r;
    this.theta = theta;
  }

  public double realPart()      { return r * Math.cos(theta) ; }
  public double imaginaryPart() { return r * Math.sin(theta) ; }
  public double r()             { return r; }
  public double theta()         { return theta; }

  public Complex plus(Complex c)      { ... } // Completely new impls
  public Complex minus(Complex c)     { ... }
  public Complex times(Complex c)     { ... }
  public Complex dividedBy(Complex c) { ... }
}
```

Review

Interface decouples client from implementation

```
public class ComplexUser {
    public static void main(String args[]) {
        Complex c = new PolarComplex(Math.PI,   1);  // -1
        Complex d = new PolarComplex(Math.PI/2, 1);  //  i

        Complex e = c.plus(d);
        System.out.println(e.realPart() + " + "
                           + e.imaginaryPart() + "i");
        e = c.times(d);
        System.out.println(e.realPart() + " + "
                           + e.imaginaryPart() + "i");
    }
}
```

When you run this program, it STILL prints

```
-1.0 + 1.0i
-0.0 + -1.0i
```

Information hiding facilitates change, promotes reuse

- Think in term of abstractions, not implementations
  - Abstractions are more likely to be reused
- Can change implementations more easily
  - Different performance
  - Different behavior
- Prevents bad programmer behavior, unnecessary dependencies

Other benefits of information hiding

- Decoupled subsystems are easier to understand in isolation
- Speeds up system development
- Reduces cost of maintenance
- Improves effectiveness of performance tuning

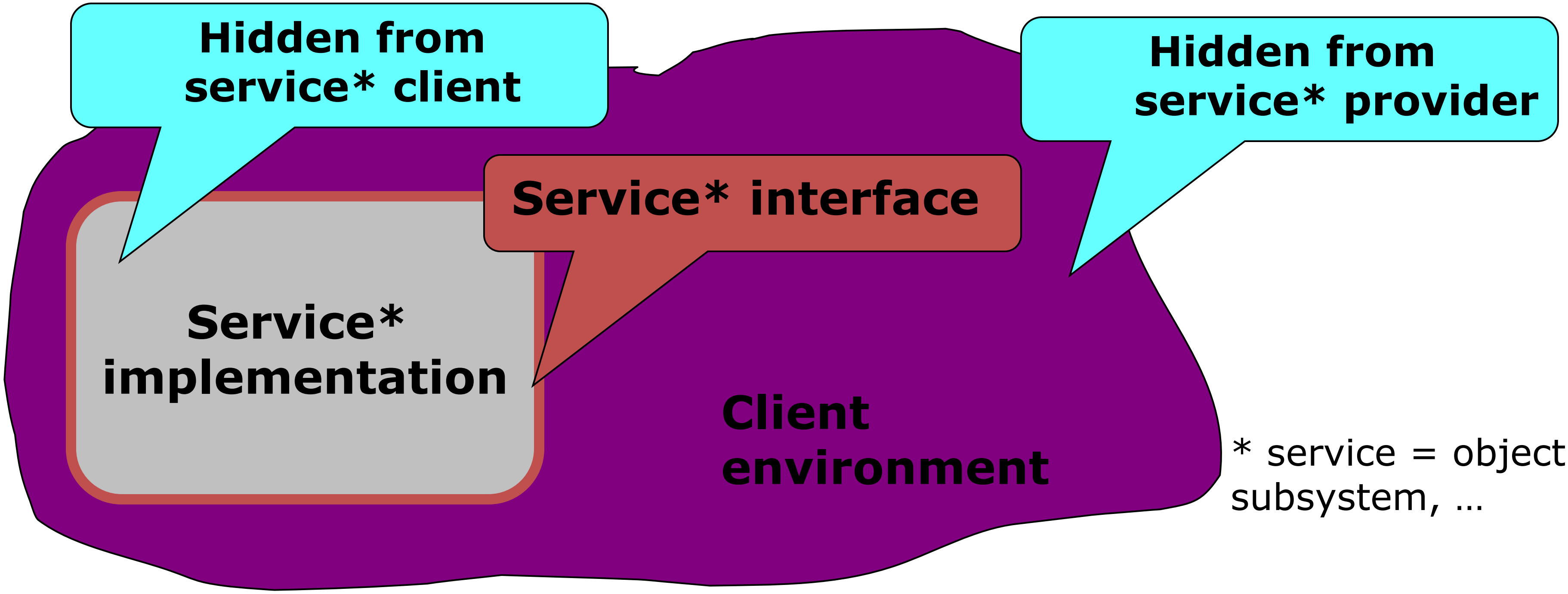
- But:
  - Requires anticipation of change (judgment)
  - Not all change can be anticipated

Best practices for information hiding

- Carefully design your API
- Provide *only* functionality required by clients
  - *All* other members should be private
- You can always make a private member public later without breaking clients
  - But not vice-versa!

CONTRACTS
(BEYOND TYPE SIGNATURES)

Contracts and Clients



* service = object, subsystem, …

Contracts

- Agreement between provider and users of an object
- Includes
  - Interface specification (types)
  - Functionality and correctness expectations
  - Performance expectations
- What the method does, not how it does it
  - Interface (API), not implementation

Who’s to blame?

```
Algorithms.shortestDistance(g, “Tom”, “Anne”);
```

```
> ArrayOutOfBoundsException
```

Who’s to blame?

```
Algorithms.shortestDistance(g, “Tom”, “Anne”);


> -1
```

Who’s to blame?

```
Algorithms.shortestDistance(g, “Tom”, “Anne”);


> 0
```

Who’s to blame?

```
class Algorithms {
    /**
     * This method finds the
     * shortest distance between to
     * verticies. It returns -1 if
     * the two nodes are not
     * connected. */
    int shortestDistance(…) {…}
}
```

Who’s to blame?

```
Math.sqrt(-5);
```

```
> 0
```

Who’s to blame?

```
/**
  * Returns the correctly rounded positive square root of a
  * {@code double} value.
  * Special cases:
  * <ul><li>If the argument is NaN or less than zero, then the
  * result is NaN.
  * <li>If the argument is positive infinity, then the result
  * is positive infinity.
  * <li>If the argument is positive zero or negative zero, then
  * the result is the same as the argument.</ul>
  * Otherwise, the result is the {@code double} value closest to
  * the true mathematical square root of the argument value.
  *
  * @param a a value.
  * @return the positive square root of {@code a}.
  * If the argument is NaN or less than zero, the result is NaN.
  */

public static double sqrt(double a) { …}
```

Textual Specification

public int read(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.
- If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.
- The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let *k* be the number of bytes actually read; these bytes will be stored in elements b[off] throughb[off+*k*-1], leaving elements b[off+*k*] through b[off+len-1] unaffected.
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

- Throws:
  - IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If off is negative, len is negative, or len is greater than b.length - off

Textual Specification

public int read(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data f
  attempt is made to read as ma
  The number of bytes actually r
  until input data is available, er
- If len is zero, then no bytes ar
  attempt to read at least one by
  end of file, the value -1 is retu
  into b.
- The first byte read is stored in
  on. The number of bytes read
  bytes actually read; these byte
  1], leaving elements b[off+*k*]
- In every case, elements b[0] through b[off] and
  elements b[off+len] through b[b.length-1] are unaffected.

- **Specification of return**
- **Timing behavior (blocks)**
- **Case-by-case spec**
  - **len=0 → return 0**
  - **len>0 && eof → return -1**
  - **len>0 && !eof →return >0**
- **Exactly where the data is stored**
- **What parts of the array are not affected**

- Throws:
  - IOException - If the first byte
    or if the input stream has beer
  - NullPointerException - If b is n
  - IndexOutOfBoundsException -
    than b.length - off

- **Multiple error cases, each with a precondition**
- **Includes “runtime exceptions” not in throws clause**

Specifications

- Contains
  - Functional behavior
  - Erroneous behavior
  - Quality attributes (performance, scalability, security, …)
- Desirable attributes
  - Complete
    - Does not leave out any desired behavior
  - Minimal
    - Does not require anything that the user does not care about
  - Unambiguous
    - Fully specifies what the system should do in every case the user cares about
  - Consistent
    - Does not have internal contradictions
  - Testable
    - Feasible to objectively evaluate
  - Correct
    - Represents what the end-user(s) need

Functional Specification

- States method’s and caller’s responsibilities
- Analogy: legal contract
  - If you pay me this amount on this schedule…
  - I will build a with the following detailed specification
  - Some contracts have remedies for nonperformance
- Method contract structure
  - **Preconditions**: what method requires for correct operation
  - **Postconditions**: what method establishes on completion
  - **Exceptional behavior**: what it does if precondition violated
- Defines what it means for impl to be correct

Functional Specificati

- State
- Analo
  - If yo
  - I wil ing detailed specification
  - Som contracts have remedies for nonperformance

What does the implementation have to fulfill if the client violates the precondition?

- Method contract structure
  - **Preconditions**: what method requires for correct operation
  - **Postconditions**: what method establishes on completion
  - **Exceptional behavior**: what it does if precondition violated
- Defines what it means for impl to be correct

Formal Specifications

```
/*@ requires len >= 0 && array != null && array.length == len;
  @
  @ ensures \result ==
  @              (\sum int j;  0 <= j && j < len;  array[j]);
  @*/
int total(int array[], int len);
```

Advantage of formal specifications:

* runtime checks (almost) for free
* basis for formal verification
* assisting automatic analysis tools

JML (Java Modelling Language) as specifications language in Java (inside comments)

Disadvantages?

Runtime Checking of Specifications with Assertions

```
/*@ requires len >= 0 && array.length == len
   @ ensures \result ==
   @              (\sum int j;  0 <= j && j < len;  array[j])
   @*/
float sum(int array[], int len) {
    assert len >= 0;
    assert array.length == len;
    float sum = 0.0;
    int i = 0;
    while (i < len) {
        sum = sum + array[i]; i = i + 1;
    }
    assert sum …;
    return sum;
}
```

Enable assertions with -ea flag, e.g.,

```
> java -ea Main
```

Specifications in the real world

Javadoc

```
/**
 * Returns the element at the specified position of this list.
 *
 * <p>This method is <i>not</i> guaranteed to run in constant time.
 * In some implementations, it may run in time proportional to the
 * element position.
 *
 * @param index position of element to return; must be non-negative and
 *              less than the size of this list.
 * @return the element at the specified position of this list
 * @throws IndexOutOfBoundsException if the index is out of range
 *         ({@code index < 0 || index >= this.size()})
 */
E get(int index);
```

Postcondition

Precondition

Exceptional behavior

Javadoc contents

- Document
  - Every parameter
  - Return value
  - Every exception (checked and unchecked)
  - What the method does, including
    - Purpose
    - Side effects
    - Any thread safety issues
    - Any performance issues
- Do **not** document implementation details

Write a Specification

- Write
  - a type signature,
  - a textual (Javadoc) specification, and
  - a formal specification

for a function **slice(list, from, until)** that returns all values of a list between positions <from> and <until> as a new list

Reminder: Formal specification

```
/*@ requires len >= 0 && array != null &&
  @                  array.length == len;
  @
  @ ensures \result ==
  @            (\sum int j;  0 <= j &&
  @                  j < len;  array[j]);
  @*/
int total(int array[], int len);
```

Reminder: Javadoc specification

```
/**
 * Returns …
 * @param index position of element …
 * @return the element at the specified posi
 * @throws IndexOutOfBoundsException if the
 *        ({@code index < 0 || index >= thi
 */
E get(int index);
```

Contracts and Interfaces

- All objects implementing an interface must adhere to the interface's contracts
  - Objects may provide different implementations for the same specification
  - Subtype polymorphism: Client only cares about interface, not about the implementation

p.getX() **s.read()**

=> **Design for Change**

FUNCTIONAL CORRECTNESS (UNIT TESTING AGAINST INTERFACES)

Context

- **Design for Change** as goal
- **Encapsulation** provides technical means
- **Information Hiding** as design strategy
- **Contracts** describe behavior of hidden details
- **Testing** helps gaining confidence in functional correctness (w.r.t. contracts)

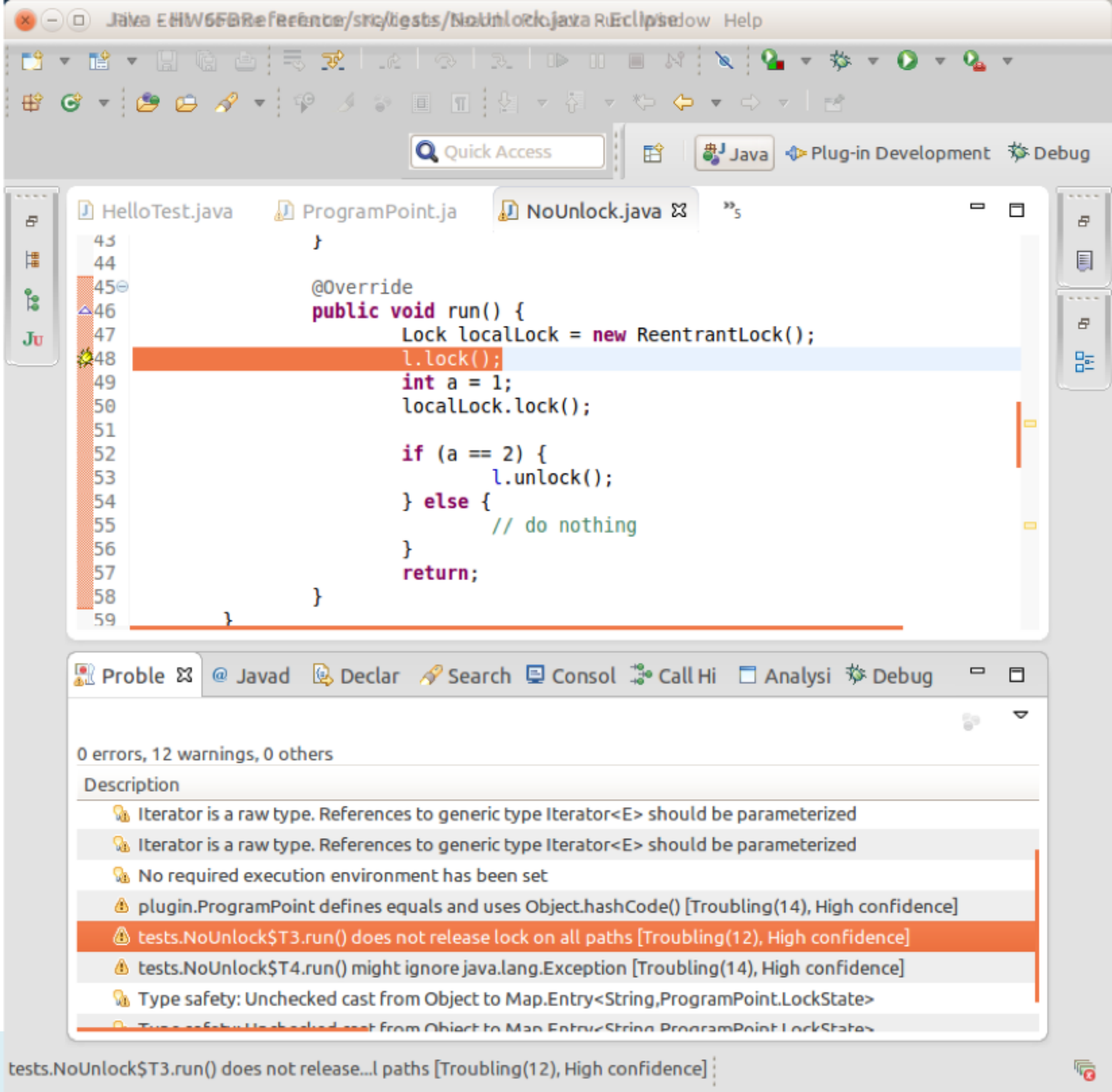
Functional correctness

- Compiler ensures types are correct (type-checking)
  - Prevents many runtime errors, such as “Method Not Found” and “Cannot add boolean to int”

Functional correctness

- Compiler ensures types are correct (type-checking)
  - Prevents many runtime errors, such as “Method Not Found” and “Cannot add boolean to int”
- Static analysis tools (e.g., FindBugs) recognize many common problems (*bug patterns*)
  - Warns on possible NullPointerExceptions or forgetting to close files

FindBugs



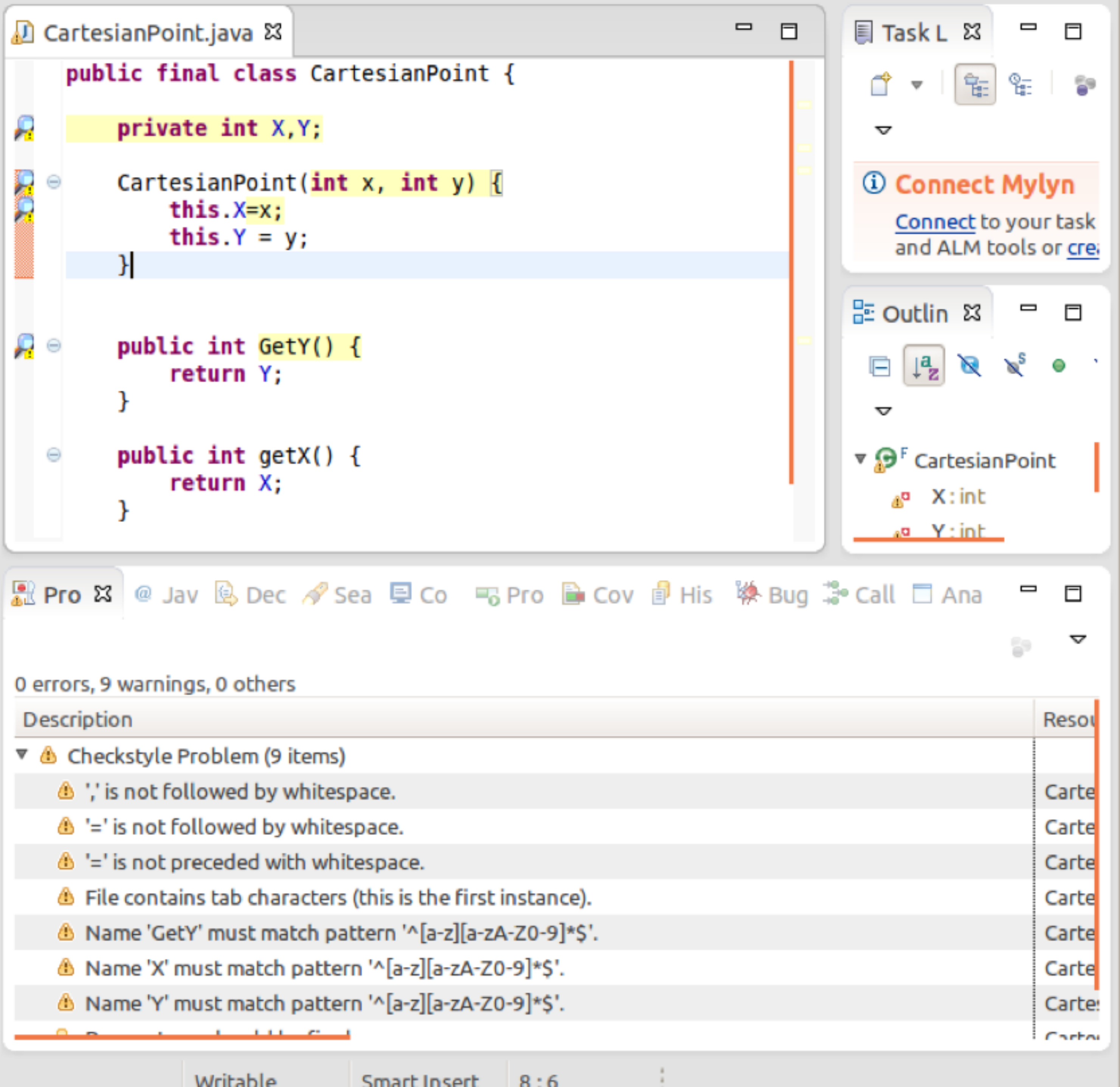
```
            }

            @Override
            public void run() {
                Lock localLock = new ReentrantLock();
                l.lock();
                int a = 1;
                localLock.lock();

                if (a == 2) {
                    l.unlock();
                } else {
                    // do nothing
                }
                return;
            }
        }
```

0 errors, 12 warnings, 0 others

Description
Iterator is a raw type. References to generic type Iterator<E> should be parameterized
Iterator is a raw type. References to generic type Iterator<E> should be parameterized
No required execution environment has been set
plugin.ProgramPoint defines equals and uses Object.hashCode() [Troubling(14), High confidence]
tests.NoUnlock$T3.run() does not release lock on all paths [Troubling(12), High confidence]
tests.NoUnlock$T4.run() might ignore java.lang.Exception [Troubling(14), High confidence]
Type safety: Unchecked cast from Object to Map.Entry<String,ProgramPoint.LockState>
Type safety: Unchecked cast from Object to Map.Entry<String,ProgramPoint.LockState>

tests.NoUnlock$T3.run() does not release...l paths [Troubling(12), High confidence]

CheckStyle



```
public final class CartesianPoint {

    private int X,Y;

    CartesianPoint(int x, int y) {
        this.X=x;
        this.Y = y;
    }


    public int GetY() {
        return Y;
    }

    public int getX() {
        return X;
    }
```

0 errors, 9 warnings, 0 others

Description	Resou
Checkstyle Problem (9 items)	
',' is not followed by whitespace.	Carte
'=' is not followed by whitespace.	Carte
'=' is not preceded with whitespace.	Carte
File contains tab characters (this is the first instance).	Carte
Name 'GetY' must match pattern '^[a-z][a-zA-Z0-9]*$'.	Carte
Name 'X' must match pattern '^[a-z][a-zA-Z0-9]*$'.	Carte
Name 'Y' must match pattern '^[a-z][a-zA-Z0-9]*$'.	Carte

Functional correctness

- Compiler ensures types are correct (type-checking)
  - Prevents many runtime errors, such as “Method Not Found” and “Cannot add boolean to int”
- Static analysis tools (e.g., FindBugs) recognize many common problems (*bug patterns*)
  - Warns on possible NullPointerExceptions or forgetting to close files
- How to ensure functional correctness of contracts beyond type correctness and bug patterns?

Formal verification

- Use mathematical methods to prove correctness with respect to the formal specification
- Formally prove that all possible executions of an implementation fulfill the specification
- Manual effort; partial automation; not automatically decidable

Testing

- Executing the program with selected inputs in a controlled environment
- Goals
  - Reveal bugs, so they can be fixed (main goal)
  - Assess quality
  - Clarify the specification, documentation

Re: Formal verification, Testing

“Beware of bugs in the above code; I have only proved it correct, not tried it.”

Donald Knuth, 1977

"Testing shows the presence, not the absence of bugs.”

Edsger W. Dijkstra, 1969

Q: Who’s right, Dijkstra or Knuth?

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
             low = mid + 1
         else if (midVal > key)
             high = mid - 1;
         else
             return mid; // key found
     }
     return -(low + 1);  // key not found.
 }
```

Binary search from java.util.Arrays

Q: Who's right, Dijkstra or Knuth?

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
             low = mid + 1
         else if (midVal > key)
             high = mid - 1;
         else
             return mid; // key found
     }
     return -(low + 1);  // key not found.
 }
```

Spec: sets `mid` to the average of `low` and `high`, truncated down to the nearest integer.

Fails if
`low + high` > MAXINT ($2^{31} - 1$)
Sum overflows to negative value

Binary search from java.util.Arrays

A: They’re both right

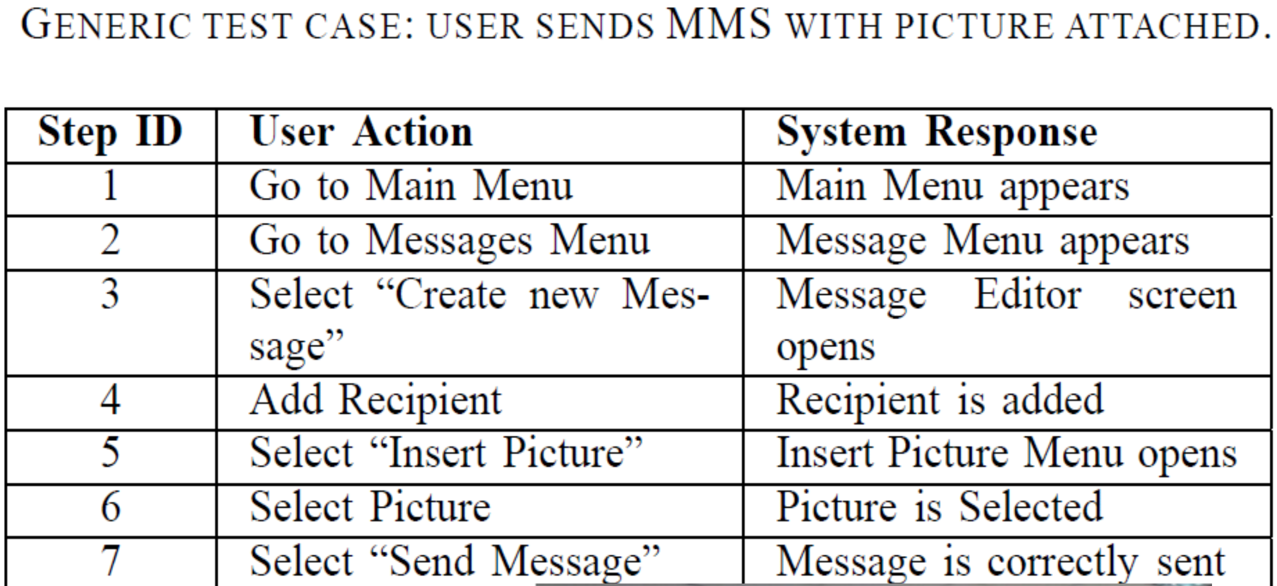
- There is no silver bullet!
- Use all the tools at your disposal
  - Careful design
  - Testing
  - Formal methods (where appropriate)
  - Code reviews
  - …
- You’ll still have bugs, but hopefully fewer.

What to test?

- Functional correctness of a method (e.g., computations, contracts)
- Functional correctness of a class (e.g., class invariants)
- Behavior of a class in a subsystem/multiple subsystems/the entire system
- Behavior when interacting with the world
  - Interacting with files, networks, sensors, …
  - Erroneous states
  - Nondeterminism, Parallelism
  - Interaction with users
- Other qualities (performance, robustness, usability, security, …)

Our focus now

Manual testing



GENERIC TEST CASE: USER SENDS MMS WITH PICTURE ATTACHED.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select "Create new Message"	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select "Insert Picture"	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select "Send Message"	Message is correctly sent

- Live System?
- Extra Testing System?
- Check output / assertions?
- Effort, Costs?
- Reproducible?

Automated testing

- Execute a program with specific inputs, check output for expected values
- Easier to test small pieces than testing user interactions
- Set up testing infrastructure
- **Execute tests regularly**
  - After *every* change

Example

Black box testing

```
/**
 * computes the sum of the first len values of the array
 *
 * @param array array of integers of at least length len
 * @param len number of elements to sum up
 * @return sum of the array values
 */
int total(int array[], int len);
```

Example

Black box testing

```
/**
 * computes the sum of the first len values of the array
 *
 * @param array array of integers of at least length len
 * @param len number of elements to sum up
 * @return sum of the array values
 */
int total(int array[], int len);
```

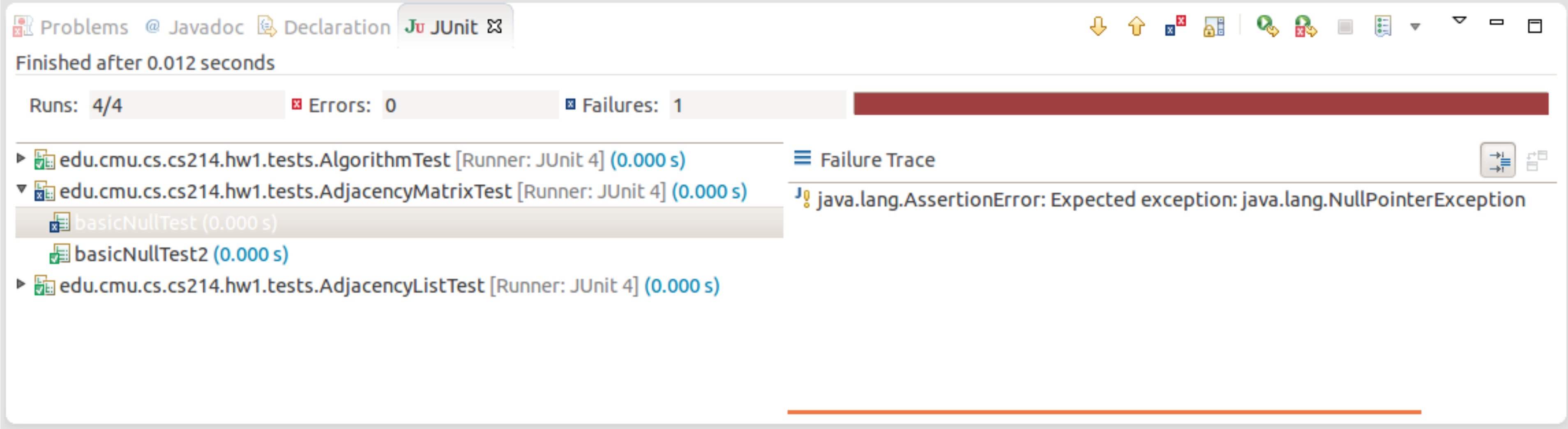
- Test empty array
- Test array of length 1 and 2
- Test negative numbers
- Test invalid length (negative / longer than array.length)
- Test null as array
- Test with a very long array

Unit Tests

- Tests for small units: functions, classes, subsystems
  - Smallest testable part of a system
  - Test parts before assembling them
  - Intended to catch local bugs
- Typically written by developers
- Many small, fast-running, independent tests
- Little dependencies on other system parts or environment
- Insufficient but a good starting point, extra benefits:
  - Documentation (executable specification)
  - Design mechanism (design for testability)

JUnit

- Popular unit-testing framework for Java
- Easy to use
- Tool support available
- Can be used as design mechanism



Finished after 0.012 seconds

Runs: 4/4 Errors: 0 Failures: 1

```
▸ edu.cmu.cs.cs214.hw1.tests.AlgorithmTest [Runner: JUnit 4] (0.000 s)
▾ edu.cmu.cs.cs214.hw1.tests.AdjacencyMatrixTest [Runner: JUnit 4] (0.000 s)
    basicNullTest (0.000 s)
    basicNullTest2 (0.000 s)
▸ edu.cmu.cs.cs214.hw1.tests.AdjacencyListTest [Runner: JUnit 4] (0.000 s)
```

Failure Trace

```
java.lang.AssertionError: Expected exception: java.lang.NullPointerException
```

JUnit

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class AdjacencyListTest {
        @Test
        public void testSanityTest(){
                Graph g1 = new AdjacencyListGraph(10)
                Vertex s1 = new Vertex("A");
                Vertex s2 = new Vertex("B");
                assertEquals(true, g1.addVertex(s1));
                assertEquals(true, g1.addVertex(s2));
                assertEquals(true, g1.addEdge(s1, s2));
                assertEquals(s2, g1.getNeighbors(s1)[0]);
        }

        @Test
        public void test….

        private int helperMethod…
}
```

Set up tests

Check expected results

JUnit conventions

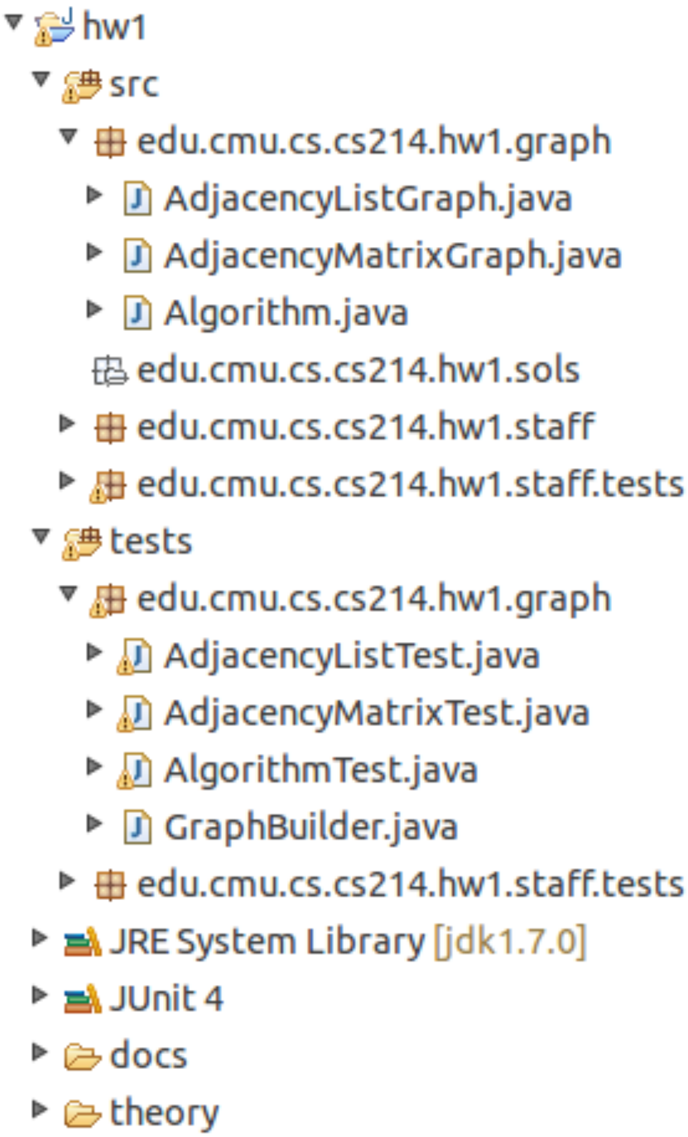
- TestCase collects multiple tests (in one class)
- TestSuite collects test cases (typically package)
- Tests should run fast
- Tests should be independent

- Tests are methods without parameter and return value
- AssertError signals failed test (unchecked exception)

- Test Runner knows how to run JUnit tests
  - (uses reflection to find all methods with @Test annotat.)

Test organization

- Conventions (not requirements)
- Have a test class FooTest for each public class Foo
- Have a source directory and a test directory
  - Store FooTest and Foo in the same package
  - Tests can access members with default (package) visibility



- hw1
  - src
    - edu.cmu.cs.cs214.hw1.graph
      - AdjacencyListGraph.java
      - AdjacencyMatrixGraph.java
      - Algorithm.java
    - edu.cmu.cs.cs214.hw1.sols
    - edu.cmu.cs.cs214.hw1.staff
    - edu.cmu.cs.cs214.hw1.staff.tests
  - tests
    - edu.cmu.cs.cs214.hw1.graph
      - AdjacencyListTest.java
      - AdjacencyMatrixTest.java
      - AlgorithmTest.java
      - GraphBuilder.java
    - edu.cmu.cs.cs214.hw1.staff.tests
  - JRE System Library [jdk1.7.0]
  - JUnit 4
  - docs
  - theory

Selecting test cases: common strategies

- Read specification
- Write tests for
  - Representative case
  - Invalid cases
  - Boundary conditions
- Are there difficult cases? (error guessing)
  - Stress tests?
  - Complex algorithms?
- Think like an attacker
  - The tester’s goal is to find bugs!
- How many test should you write?
  - Aim to cover the specification
  - Work within time/money constraints

Testable code

- Think about testing when writing code
- Unit testing encourages you to write testable code
- Separate parts of the code to make them independently testable
- Abstract functionality behind interface, make it replaceable

- Test-Driven Development
  - A design and development method in which you write tests before you write the code

Write testable code

```
//700LOC
public boolean foo() {
    try {
        synchronized () {
            if () {
            } else {
            }
            for () {
                if () {
                    if () {
                        if () {
                            if ()?
                            {
                                if () {
                                    for () {
                                    }
                                }
                            }
                        } else {
                            if () {
                                for () {
                                    if () {
                                    } else {
                                    }
                                    if () {
                                    } else {
                                        if () {
                                        }
                                    }
                                    if () {
                                        if () {
                                            if () {
                                                for () {
                                                }
                                            }
                                        }
                                    } else {
                                    }
                                }
```

Unit testing as design mechanism

* Code with low complexity

* Clear interfaces and specifications

Source: http://thedailywtf.com/Articles/Coding-Like-the-Tour-de-France.aspx

When should you stop writing tests?

- When you run out of money…
- When your homework is due…
- When you can't think of any new test cases...
- The *coverage* of a test suite
  - Trying to test all parts of the implementation
  - Statement coverage
    - Execute every statement, ideally
    - Compare to: method coverage, branch coverage, path coverage

When to stop writing tests?

- Outlook: statement coverage
  - Trying to test all parts of the implementation
  - Execute every statement, ideally

Does 100% coverage guarantee correctness?

A: No

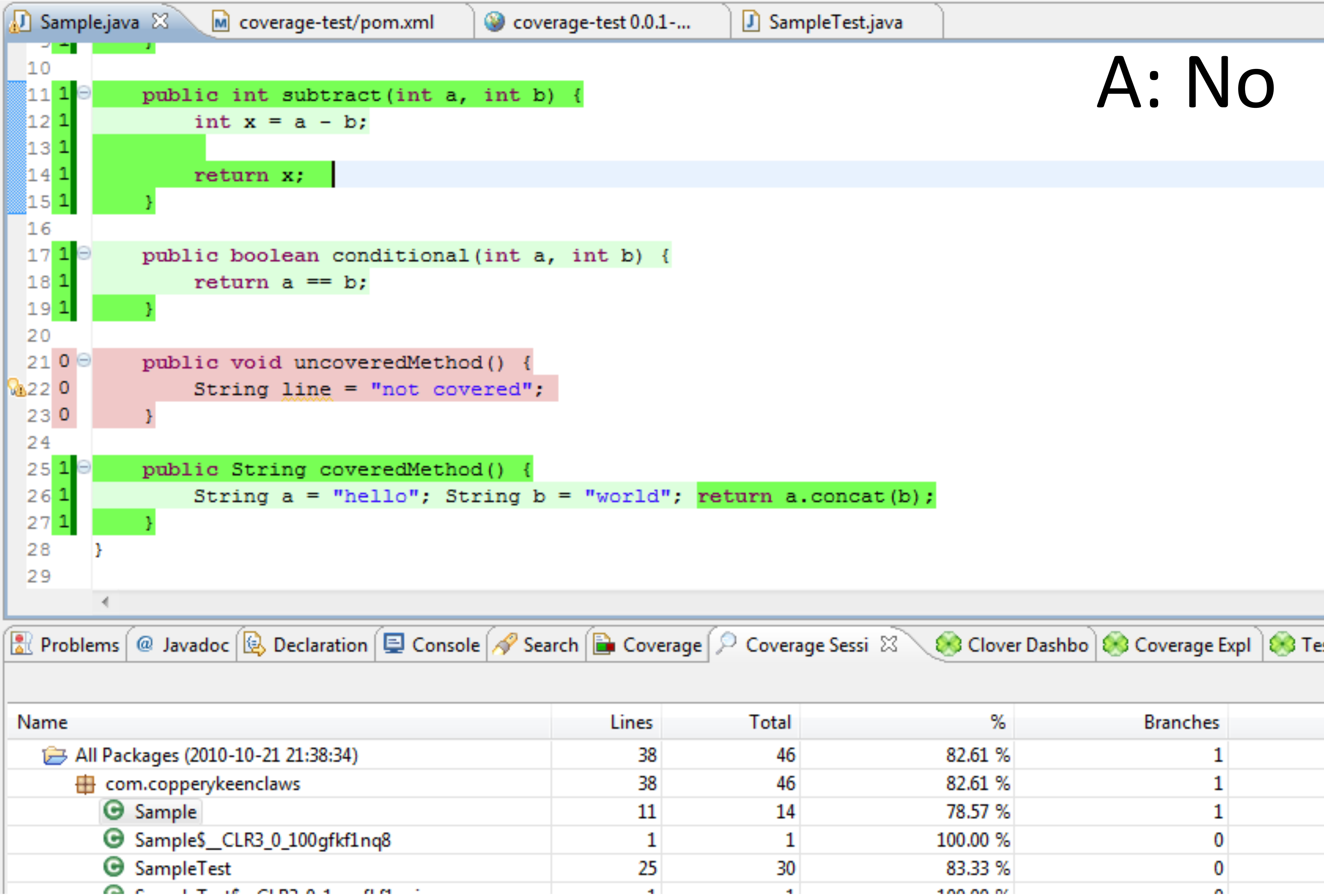
```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
             low = mid + 1
         else if (midVal > key)
             high = mid - 1;
         else
             return mid; // key found
     }
     return -(low + 1);  // key not found.
 }
```

When to stop writing tests?

- Outlook: statement coverage
  - Trying to test all parts of the implementation
  - Execute every statement, ideally

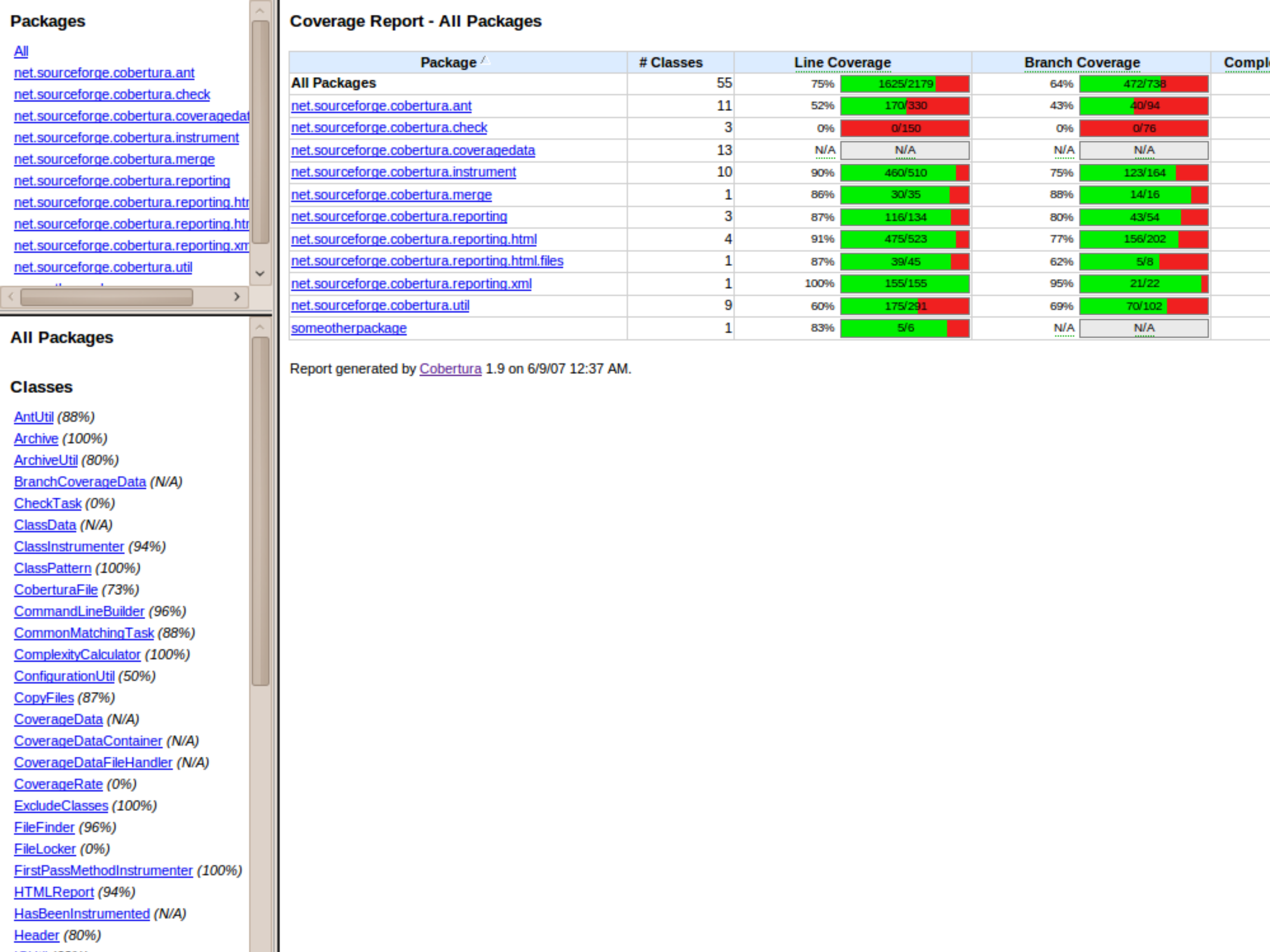
Does less than 100% coverage guarantee incorrectness?

A: No



```
    public int subtract(int a, int b) {
        int x = a - b;

        return x;
    }

    public boolean conditional(int a, int b) {
        return a == b;
    }

    public void uncoveredMethod() {
        String line = "not covered";
    }

    public String coveredMethod() {
        String a = "hello"; String b = "world"; return a.concat(b);
    }
}
```

Name	Lines	Total	%	Branches
All Packages (2010-10-21 21:38:34)	38	46	82.61 %	1
com.copperykeenclaws	38	46	82.61 %	1
Sample	11	14	78.57 %	1
Sample$__CLR3_0_100gfkf1nq8	1	1	100.00 %	0
SampleTest	25	30	83.33 %	0



Packages

- All
- net.sourceforge.cobertura.ant
- net.sourceforge.cobertura.check
- net.sourceforge.cobertura.coveragedat
- net.sourceforge.cobertura.instrument
- net.sourceforge.cobertura.merge
- net.sourceforge.cobertura.reporting
- net.sourceforge.cobertura.reporting.ht
- net.sourceforge.cobertura.reporting.ht
- net.sourceforge.cobertura.reporting.xm
- net.sourceforge.cobertura.util

All Packages

Classes

- AntUtil *(88%)*
- Archive *(100%)*
- ArchiveUtil *(80%)*
- BranchCoverageData *(N/A)*
- CheckTask *(0%)*
- ClassData *(N/A)*
- ClassInstrumenter *(94%)*
- ClassPattern *(100%)*
- CoberturaFile *(73%)*
- CommandLineBuilder *(96%)*
- CommonMatchingTask *(88%)*
- ComplexityCalculator *(100%)*
- ConfigurationUtil *(50%)*
- CopyFiles *(87%)*
- CoverageData *(N/A)*
- CoverageDataContainer *(N/A)*
- CoverageDataFileHandler *(N/A)*
- CoverageRate *(0%)*
- ExcludeClasses *(100%)*
- FileFinder *(96%)*
- FileLocker *(0%)*
- FirstPassMethodInstrumenter *(100%)*
- HTMLReport *(94%)*
- HasBeenInstrumented *(N/A)*
- Header *(80%)*

Coverage Report - All Packages

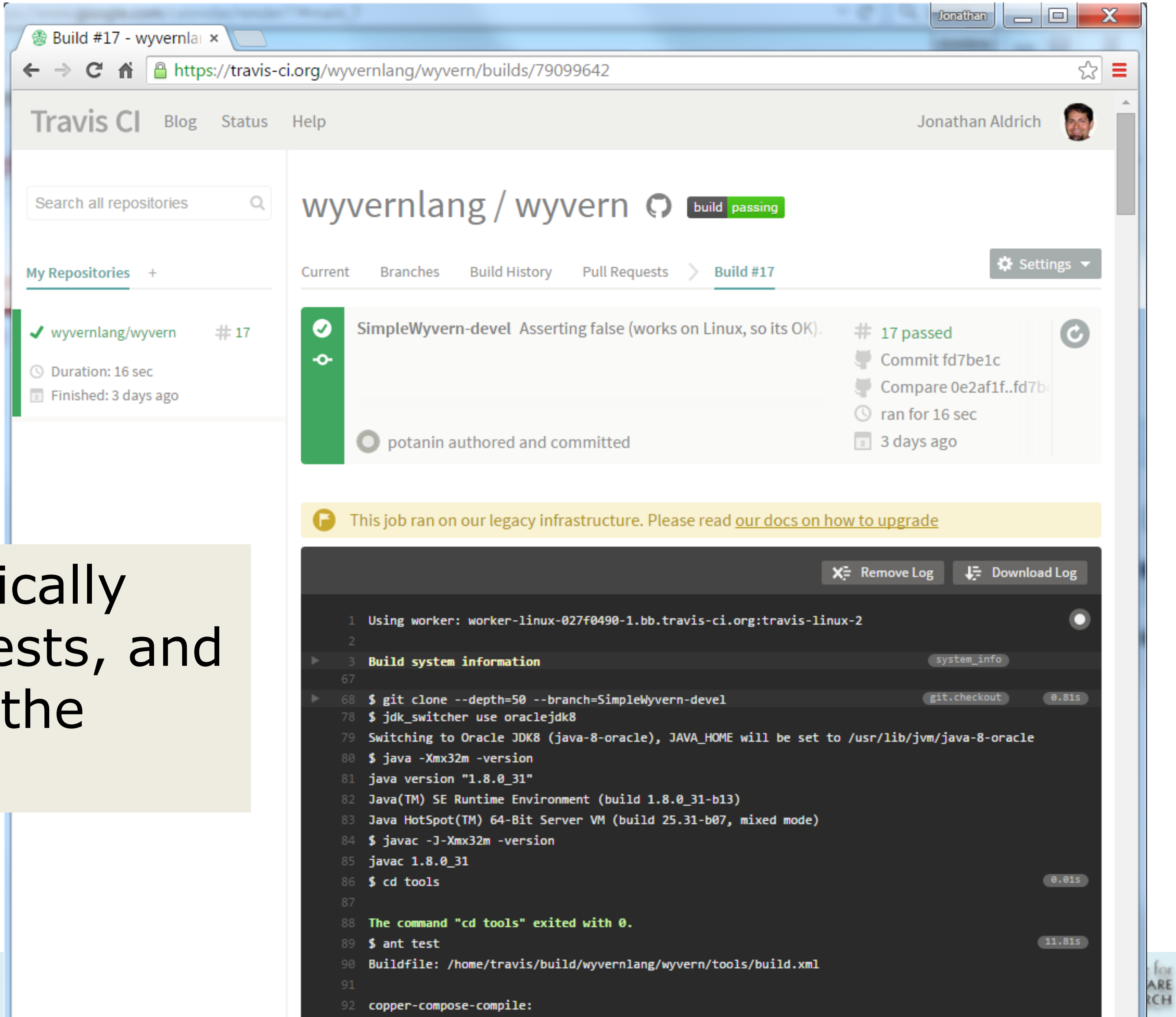
Package	# Classes	Line Coverage		Branch Coverage		Compl
All Packages	55	75%	1625/2179	64%	472/738	
net.sourceforge.cobertura.ant	11	52%	170/330	43%	40/94	
net.sourceforge.cobertura.check	3	0%	0/150	0%	0/76	
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	N/A	N/A	
net.sourceforge.cobertura.instrument	10	90%	460/510	75%	123/164	
net.sourceforge.cobertura.merge	1	86%	30/35	88%	14/16	
net.sourceforge.cobertura.reporting	3	87%	116/134	80%	43/54	
net.sourceforge.cobertura.reporting.html	4	91%	475/523	77%	156/202	
net.sourceforge.cobertura.reporting.html.files	1	87%	39/45	62%	5/8	
net.sourceforge.cobertura.reporting.xml	1	100%	155/155	95%	21/22	
net.sourceforge.cobertura.util	9	60%	175/291	69%	70/102	
someotherpackage	1	83%	5/6	N/A	N/A	

Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

Run tests frequently

- You should only commit code that is passing all tests
- Run tests before every commit
- If entire test suite becomes too large and slow for rapid feedback:
  - Run local tests ("smoke tests", e.g. all tests in package) frequently
  - Run all tests nightly
  - Medium sized projects easily have 1000s of test cases and run for minutes
- Continuous integration servers help to scale testing

Continuous integration - Travis CI



✓ wyvernlang/wyvern # 17

Duration: 16 sec

Finished: 3 days ago

wyvernlang / wyvern

build passing

SimpleWyvern-devel Asserting false (works on Linux, so its OK).

potanin authored and committed

17 passed

Commit fd7be1c

Compare 0e2af1f..fd7b

ran for 16 sec

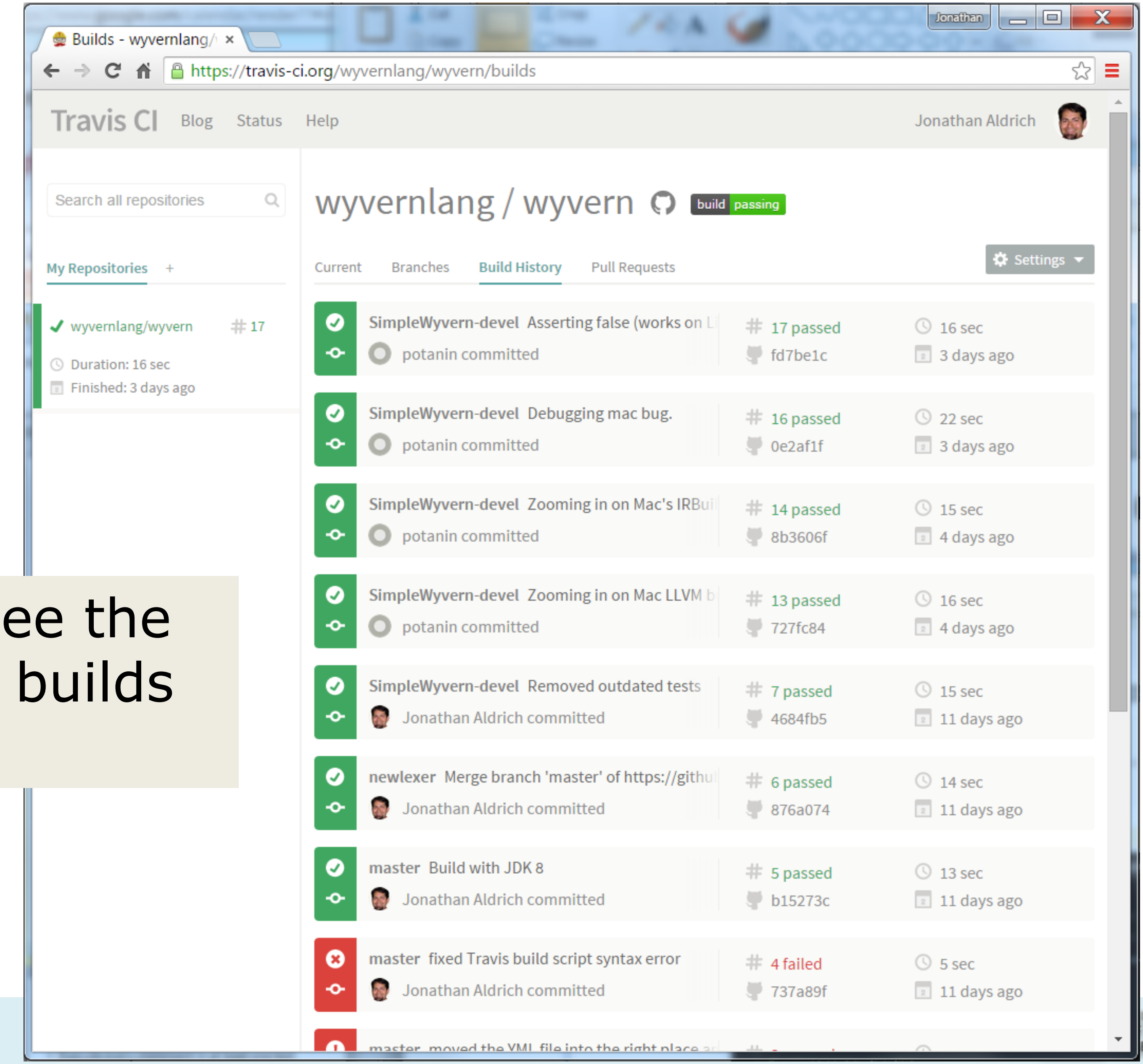
3 days ago

This job ran on our legacy infrastructure. Please read our docs on how to upgrade

```
Using worker: worker-linux-027f0490-1.bb.travis-ci.org:travis-linux-2

Build system information

$ git clone --depth=50 --branch=SimpleWyvern-devel
$ jdk_switcher use oraclejdk8
Switching to Oracle JDK8 (java-8-oracle), JAVA_HOME will be set to /usr/lib/jvm/java-8-oracle
$ java -Xmx32m -version
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
$ javac -J-Xmx32m -version
javac 1.8.0_31
$ cd tools

The command "cd tools" exited with 0.
$ ant test
Buildfile: /home/travis/build/wyvernlang/wyvern/tools/build.xml

copper-compose-compile:
```

Automatically builds, tests, and displays the result

Continuous integration - Travis CI



✓ wyvernlang/wyvern # 17
Duration: 16 sec
Finished: 3 days ago

wyvernlang / wyvern

Build	Commit	#	Time
SimpleWyvern-devel Asserting false (works on L potanin committed	17 passed fd7be1c	16 sec 3 days ago	
SimpleWyvern-devel Debugging mac bug. potanin committed	16 passed 0e2af1f	22 sec 3 days ago	
SimpleWyvern-devel Zooming in on Mac's IRBui potanin committed	14 passed 8b3606f	15 sec 4 days ago	
SimpleWyvern-devel Zooming in on Mac LLVM b potanin committed	13 passed 727fc84	16 sec 4 days ago	
SimpleWyvern-devel Removed outdated tests Jonathan Aldrich committed	7 passed 4684fb5	15 sec 11 days ago	
newlexer Merge branch 'master' of https://githu Jonathan Aldrich committed	6 passed 876a074	14 sec 11 days ago	
master Build with JDK 8 Jonathan Aldrich committed	5 passed b15273c	13 sec 11 days ago	
master fixed Travis build script syntax error Jonathan Aldrich committed	4 failed 737a89f	5 sec 11 days ago	

You can see the results of builds over time

Testing, Static Analysis, and Proofs

- Testing
  - Observable properties
  - Verify program for one execution
  - Manual development with automated regression
  - Most practical approach now
  - Does not find all problems (unsound)

- Static Analysis
  - Analysis of all possible executions
  - Specific issues only with conservative approx. and bug patterns
  - Tools available, useful for bug finding
  - Automated, but unsound and/or incomplete

- Proofs (Formal Verification)
  - Any program property
  - Verify program for all executions
  - Manual development with automated proof checkers
  - Practical for small programs, may scale up in the future
  - Sound and complete, but not automatically decidable

What strategy to use in your project?

SUMMARY: DESIGN FOR CHANGE/ DIVISION OF LABOR

Design Goals

- Design for Change such that
  - Classes are *open for extension* and modification without invasive changes
  - Classes encapsulate details likely to change behind (small) stable interfaces
- Design for Division of Labor such that
  - Internal parts can be *developed* independently
  - Internal details of other classes do not need to be *understood*, contract is sufficient
  - Test classes and their contracts separately (unit testing)