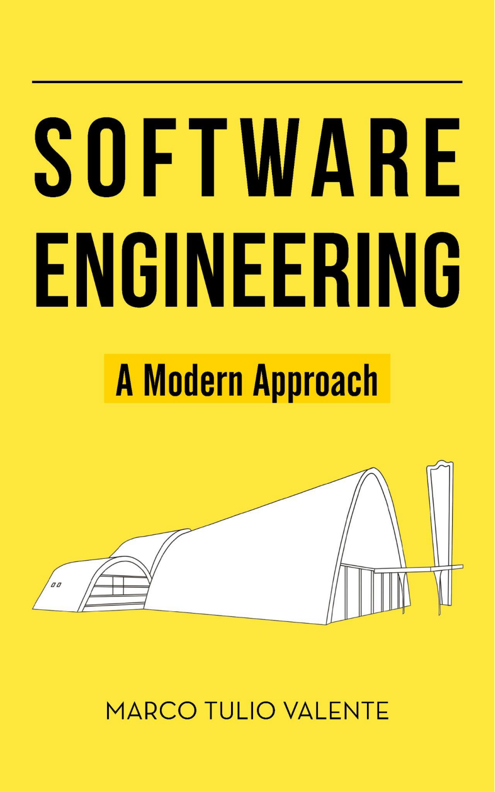


Chapter 8 - Testing

Prof. Marco Tulio Valente

https://softengbook.org

```java
public class Math {

  public static long factorial(int n) {
    if (n == 0 || n == 1) {
       return 1;
    } else {
       long result = 1;
       for (int i = 2; i <= n; i++) {
         result *= i;
       }
       return result;
    }
  }
}
```

```
public class Math {

  public static long factorial(int n) {
    if (n == 0 || n == 1) {
       return 1;
    } else {
       long result = 1;
       for (int i = 2; i <= n; i++) {
         result *= i;
       }
       return result;
    }
  }
}
```

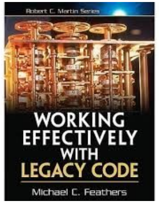
What code is missing here?

```
public class Math {

  public static long factorial(int n) {
    if (n == 0 || n == 1) {
       return 1;
    } else {
       long result = 1;
       for (int i = 2; i <= n; i++) {
         result *= i;
       }
       return result;
    }
  }
}
```

```
public class MathTest {

  @Test
  public void testFactorial() {
    assertEquals(1, Math.factorial(0));
    assertEquals(1, Math.factorial(1));
    assertEquals(120, Math.factorial(5));
  }
}
```

Unit Testing is strongly encouraged and widely practiced at Google. All code used in production is expected to have unit tests.

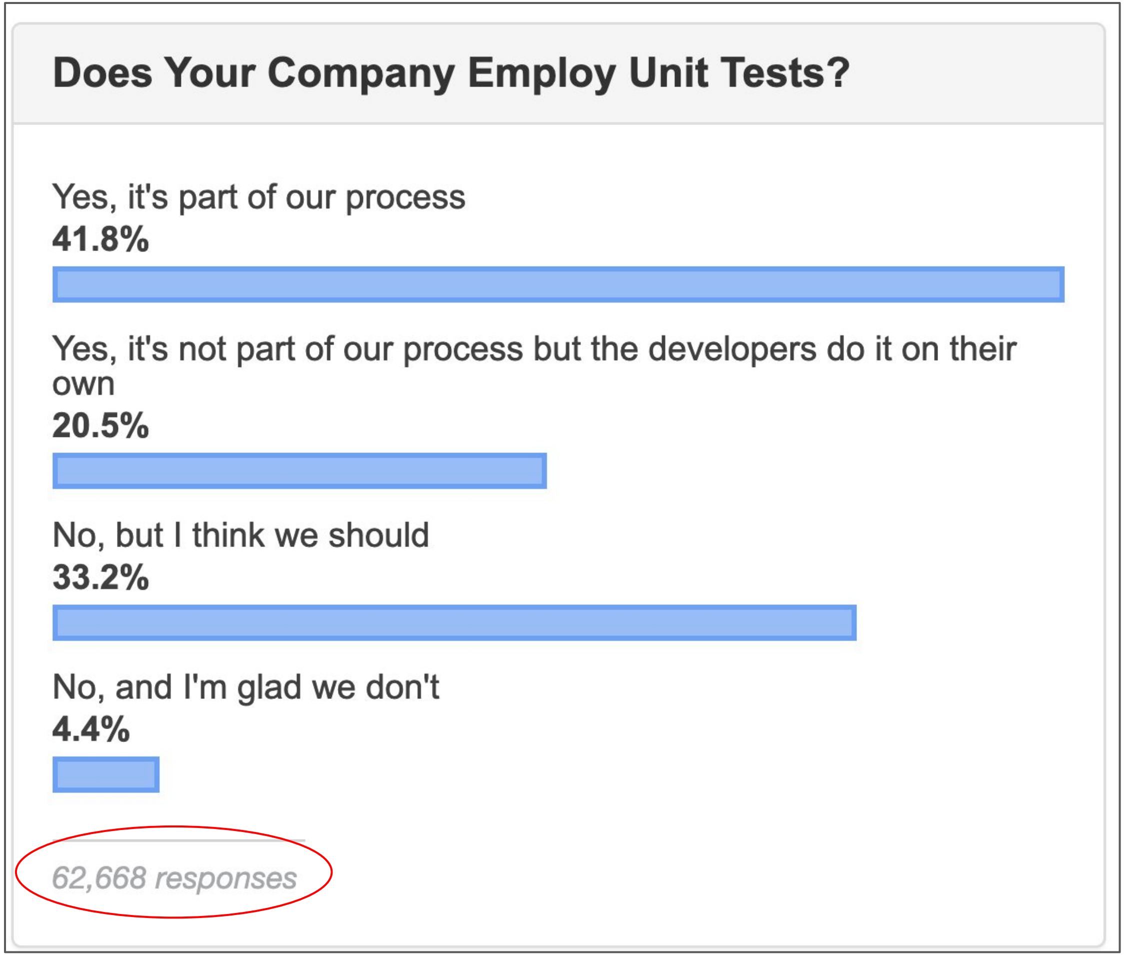
At Facebook, engineers conduct any unit tests for their newly developed code.



Code without tests is bad code.

-- Michael Feathers

Stack Overflow Survey 2019



Does Your Company Employ Unit Tests?

Yes, it's part of our process
41.8%

Yes, it's not part of our process but the developers do it on their own
20.5%

No, but I think we should
33.2%

No, and I'm glad we don't
4.4%

62,668 responses

https://survey.stackoverflow.co/2019#development-practices

Recalling Chapter 1 (Introduction)

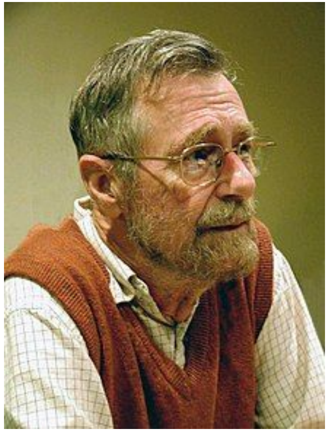
Software Testing

- Check if a program produces an expected result when executed with some test cases
- Tests can be:
  - Manual
  - Automated

Software testing can reveal the presence of bugs, but not their absence.

limitation

goal



Edsger W. Dijkstra

Defects, Bugs, and Failures

- Example of defect or bug:

```
if (condition)
    area = pi * radius * radius * radius;
```

- The correct is “area = pi * radius * radius”
- When it is executed, it will cause a failure, meaning an incorrect result.

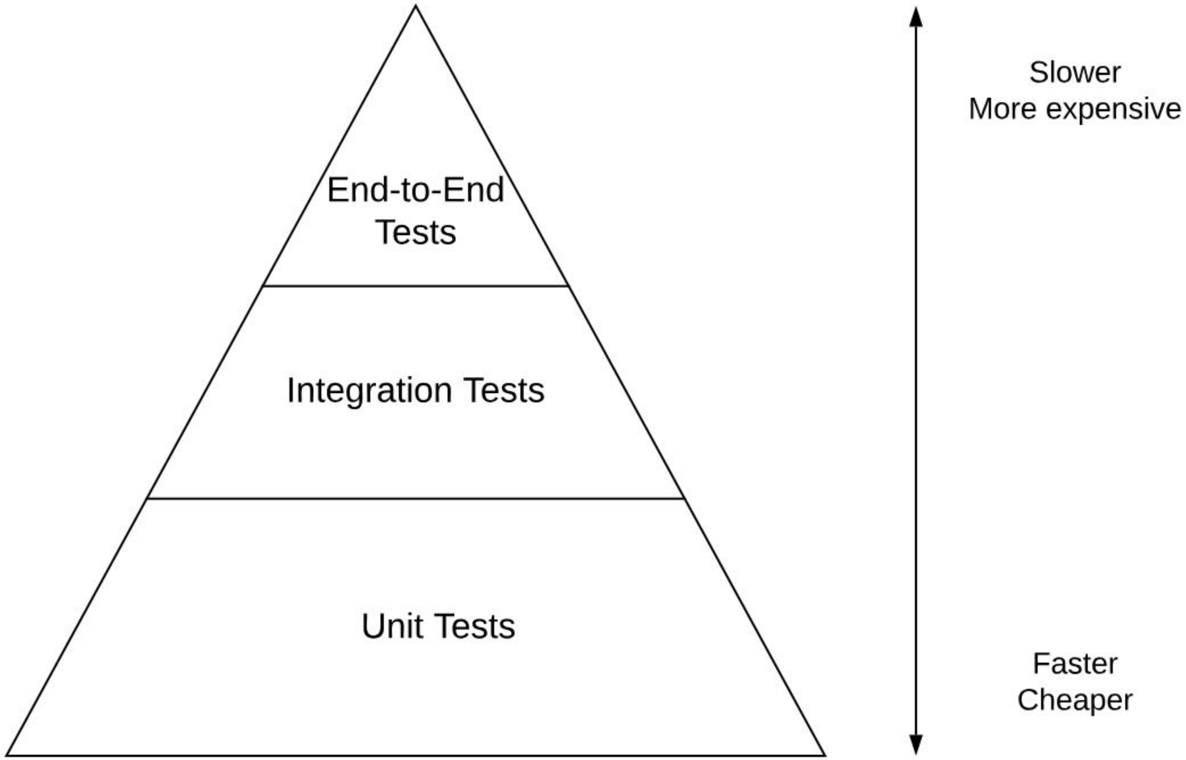
Verification vs Validation

- Verification: Are we building the product right?
    - According to the specification
- Validation: Are we building the right product?
    - The one that meets the customer needs

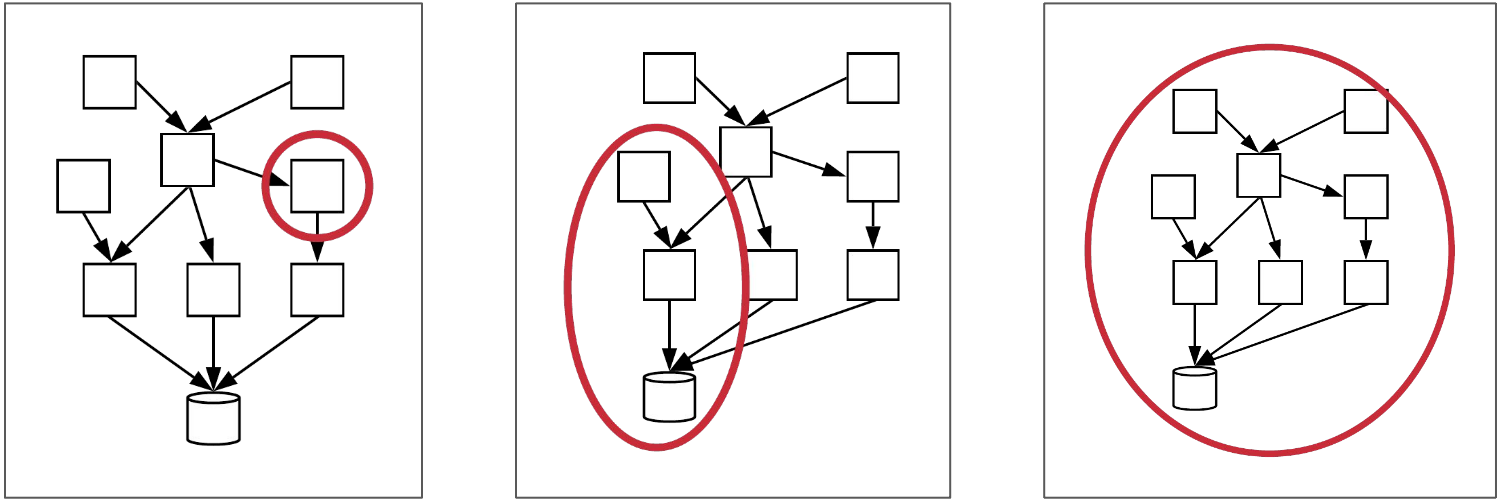
Testing & Agile Methods

- Automated
- Written by the developer of the code under testing

Test Pyramid



Types of Automated Tests



Unit

Integration

End-to-End

Unit Tests

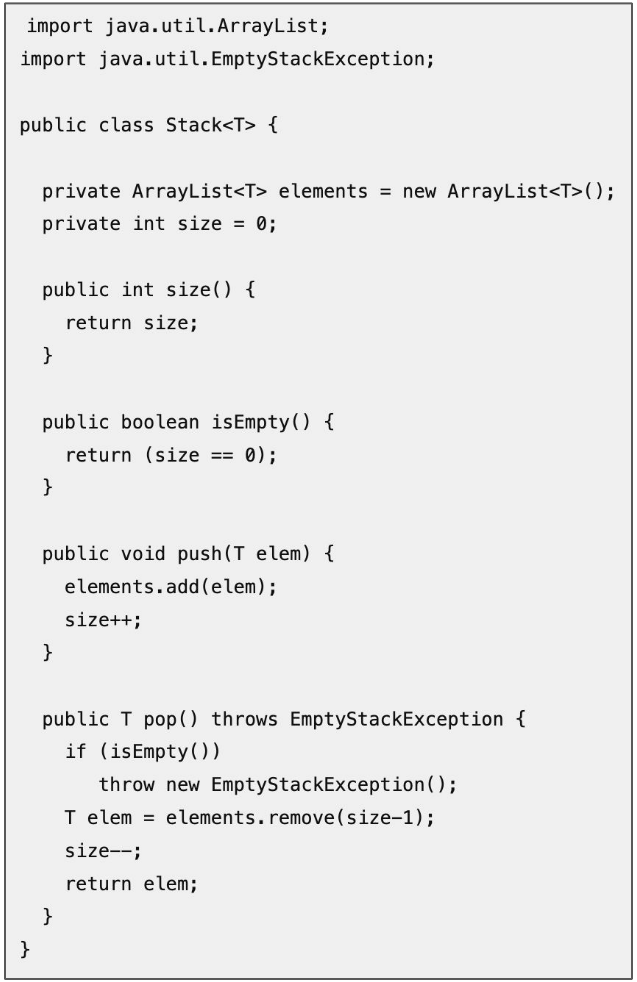
(our main subject of study)

Unit Tests

- Automated tests of small units of code (typically, classes)

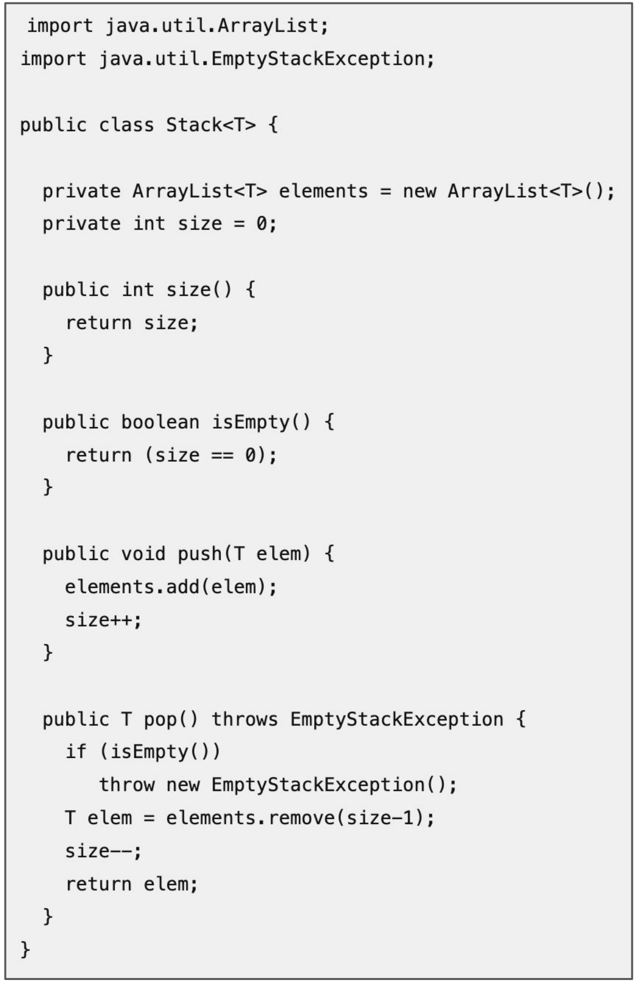
First Example: unit test for a Stack class

Class Under Test



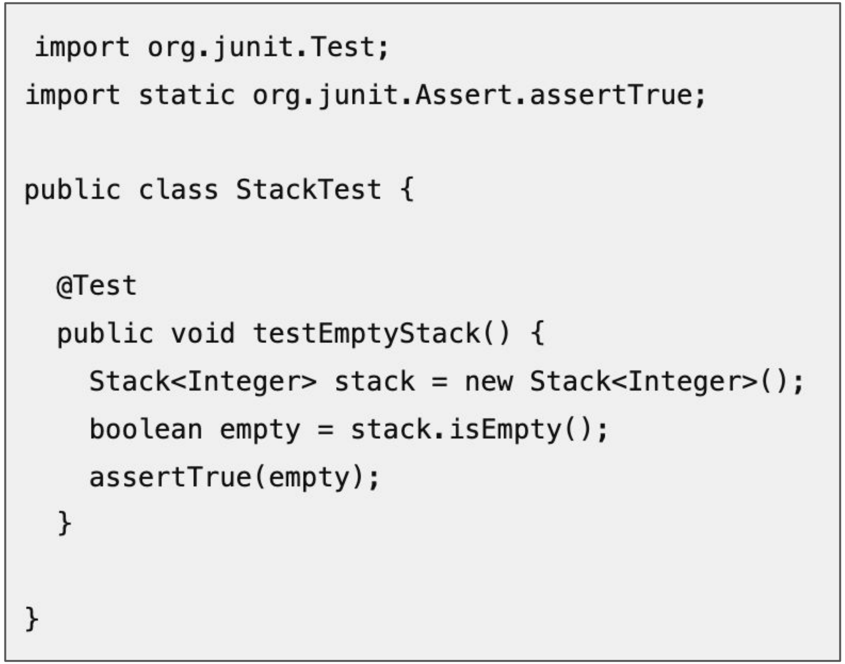
```
 import java.util.ArrayList;
import java.util.EmptyStackException;

public class Stack<T> {

  private ArrayList<T> elements = new ArrayList<T>();
  private int size = 0;

  public int size() {
    return size;
  }

  public boolean isEmpty() {
    return (size == 0);
  }

  public void push(T elem) {
    elements.add(elem);
    size++;
  }

  public T pop() throws EmptyStackException {
    if (isEmpty())
       throw new EmptyStackException();
    T elem = elements.remove(size-1);
    size--;
    return elem;
  }
}
```

Class Under Test



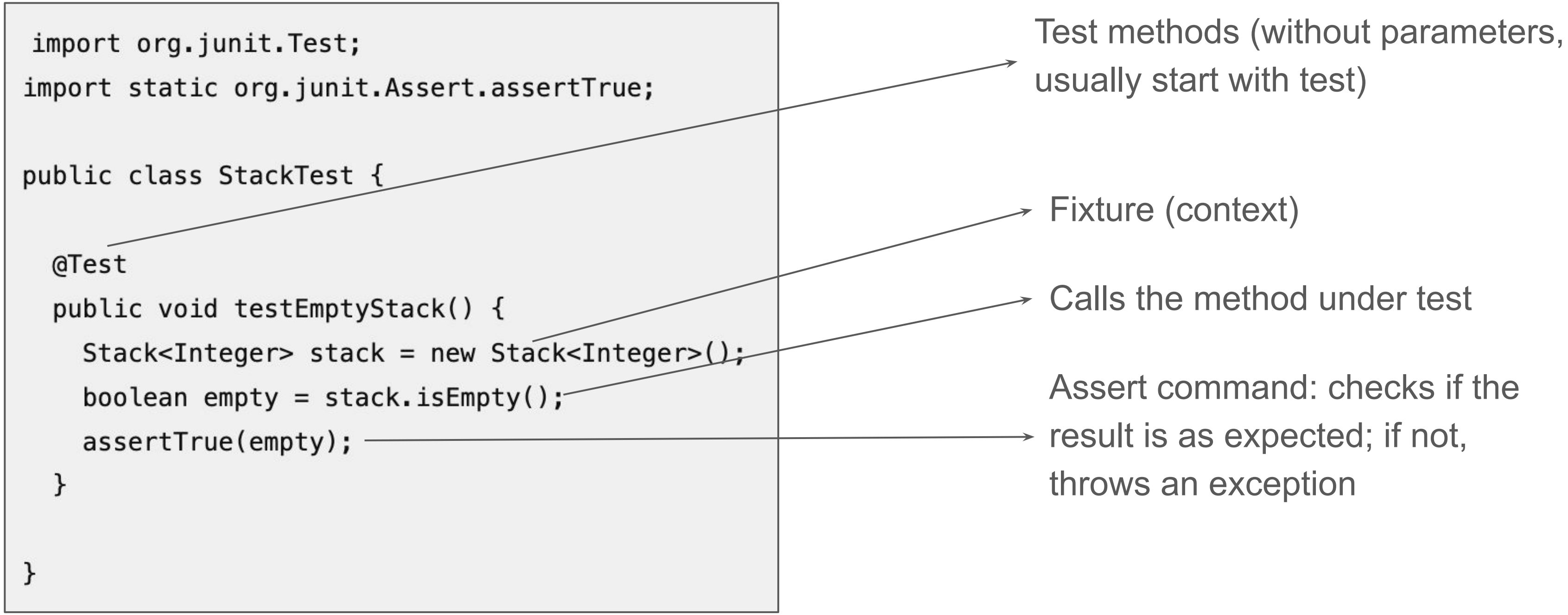
```
 import java.util.ArrayList;
import java.util.EmptyStackException;

public class Stack<T> {

  private ArrayList<T> elements = new ArrayList<T>();
  private int size = 0;

  public int size() {
    return size;
  }

  public boolean isEmpty() {
    return (size == 0);
  }

  public void push(T elem) {
    elements.add(elem);
    size++;
  }

  public T pop() throws EmptyStackException {
    if (isEmpty())
       throw new EmptyStackException();
    T elem = elements.remove(size-1);
    size--;
    return elem;
  }
}
```

Test (which is also a class)



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

public class StackTest {

  @Test
  public void testEmptyStack() {
    Stack<Integer> stack = new Stack<Integer>();
    boolean empty = stack.isEmpty();
    assertTrue(empty);
  }

}
```

Anatomy of a Unit Test



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

public class StackTest {

  @Test
  public void testEmptyStack() {
    Stack<Integer> stack = new Stack<Integer>();
    boolean empty = stack.isEmpty();
    assertTrue(empty);
  }

}
```

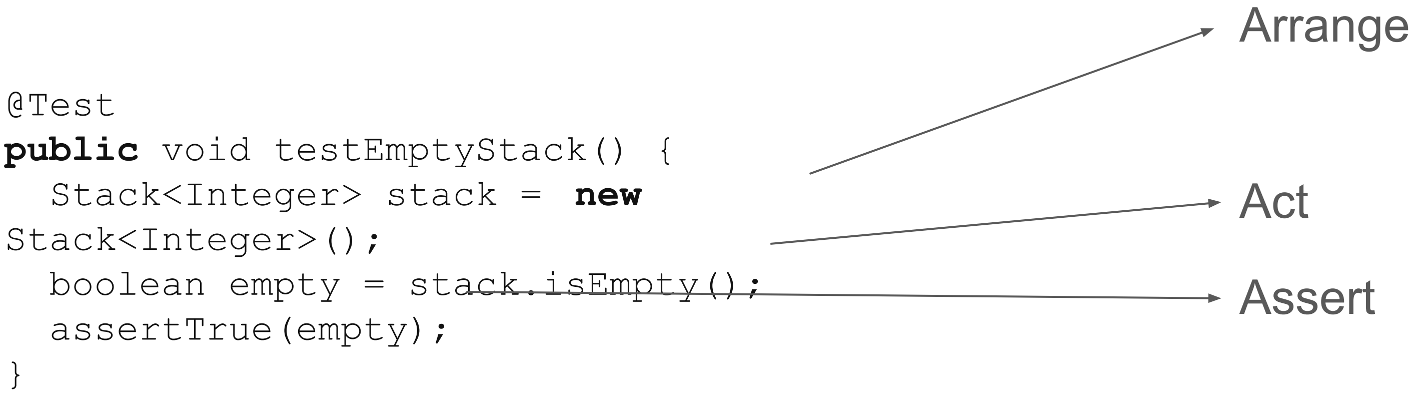
Test methods (without parameters, usually start with test)

Fixture (context)

Calls the method under test

Assert command: checks if the result is as expected; if not, throws an exception

AAA Pattern



```
@Test
public void testEmptyStack() {
  Stack<Integer> stack = new
Stack<Integer>();
  boolean empty = stack.isEmpty();
  assertTrue(empty);
}
```

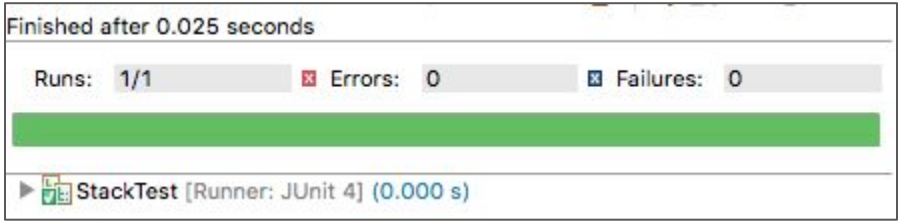
Arrange

Act

Assert

Testing framework: xUnit

JUnit

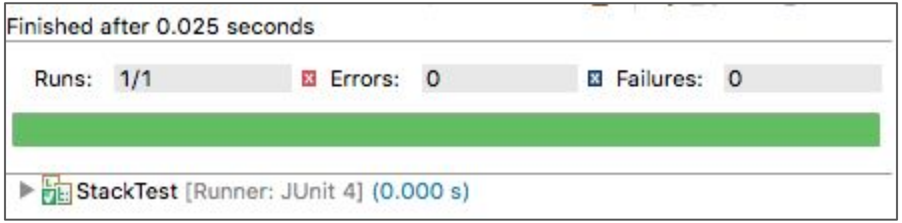


```
Finished after 0.025 seconds

Runs:  1/1     ☒ Errors:  0     ☒ Failures:  0

▶ StackTest [Runner: JUnit 4] (0.000 s)
```

Tests passed!

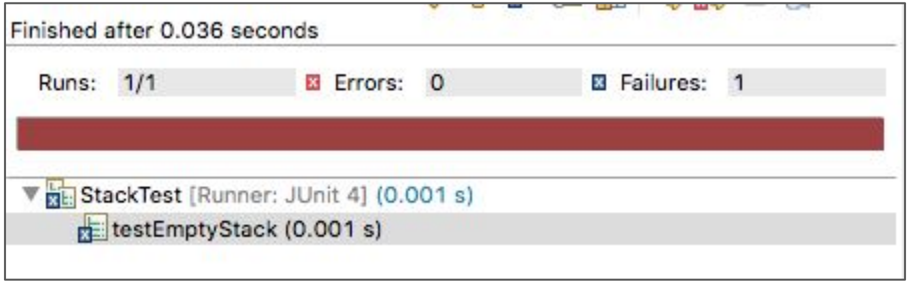
Testing framework: xUnit

JUnit



```
Finished after 0.025 seconds

Runs:  1/1      ☒ Errors:  0      ☒ Failures:  0

▶ StackTest [Runner: JUnit 4] (0.000 s)
```

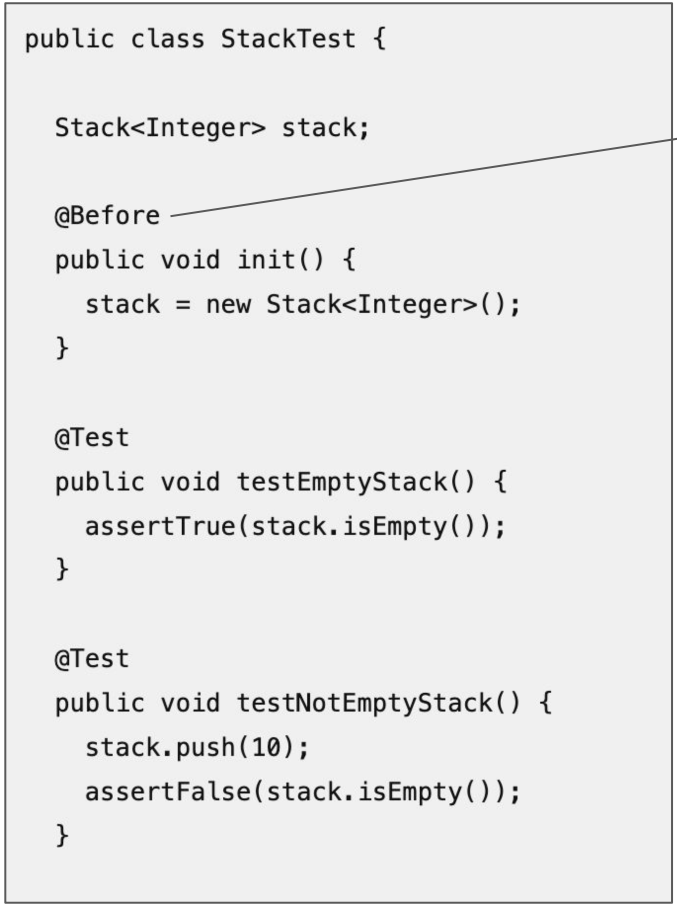
Tests passed!



```
Finished after 0.036 seconds

Runs:  1/1      ☒ Errors:  0      ☒ Failures:  1

▼ StackTest [Runner: JUnit 4] (0.001 s)
      testEmptyStack (0.001 s)
```

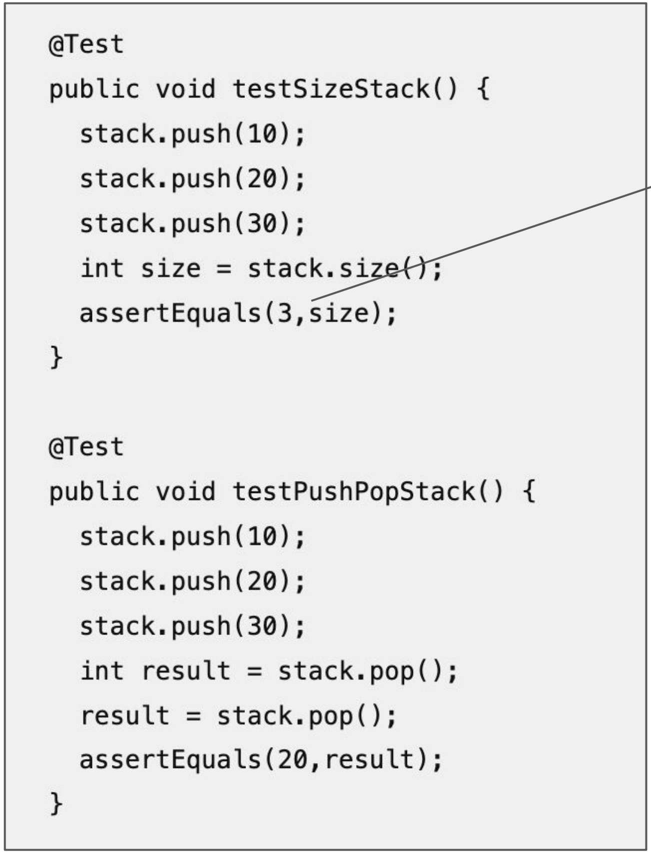
Some test failed!

More test methods



```
public class StackTest {

  Stack<Integer> stack;

  @Before
  public void init() {
    stack = new Stack<Integer>();
  }

  @Test
  public void testEmptyStack() {
    assertTrue(stack.isEmpty());
  }

  @Test
  public void testNotEmptyStack() {
    stack.push(10);
    assertFalse(stack.isEmpty());
  }
```

Executed before any @Test method

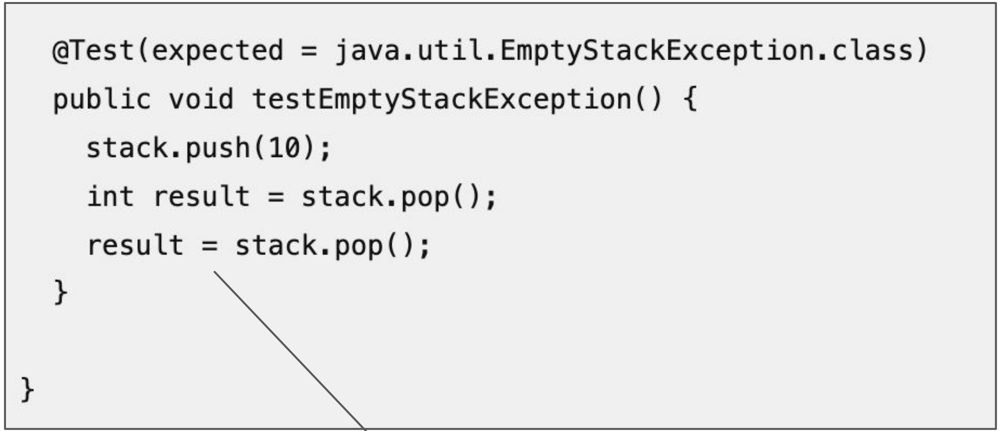


```
@Test
public void testSizeStack() {
  stack.push(10);
  stack.push(20);
  stack.push(30);
  int size = stack.size();
  assertEquals(3,size);
}

@Test
public void testPushPopStack() {
  stack.push(10);
  stack.push(20);
  stack.push(30);
  int result = stack.pop();
  result = stack.pop();
  assertEquals(20,result);
}
```

Expected value (3) and found value (size), in this order.

Message when the assert fails:

```
Expected 3 but found [value]
```



```
  @Test(expected = java.util.EmptyStackException.class)
  public void testEmptyStackException() {
    stack.push(10);
    int result = stack.pop();
    result = stack.pop();
  }

}
```

assert is not useful here; as it wouldn't be reached

More concepts about testing

Benefits

- Detecting bugs
  - In the class under test C
  - In another class (regressions)
- Documentation

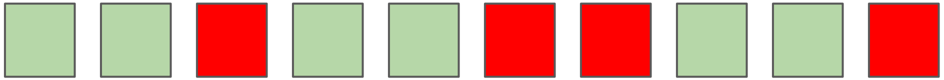
FIRST Principles (good characteristics of unit tests)

- **F**ast
- **I**ndependent (execution order does not matter)
- **R**epeatable (deterministic, non-flaky or non-erratic)
- **S**elf-checking (green vs red)
- **T**imely (written as soon as possible)

Flaky Tests

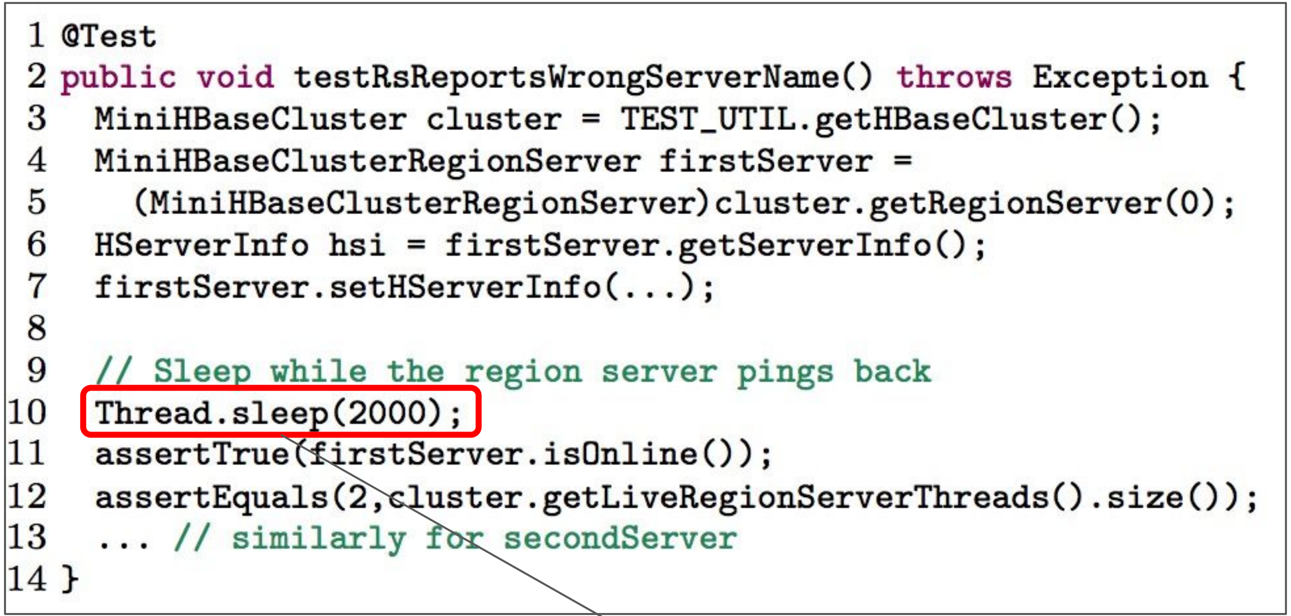
- Non-deterministic tests: Sometimes they pass, sometimes they fail
- Example:

Results of successive executions of the same test T in a program
that has not undergone any modifications:



Why do some tests are flaky?

Source: An Empirical Analysis of Flaky Tests, FSE 2014.



```
@Test
public void testRsReportsWrongServerName() throws Exception {
  MiniHBaseCluster cluster = TEST_UTIL.getHBaseCluster();
  MiniHBaseClusterRegionServer firstServer =
    (MiniHBaseClusterRegionServer)cluster.getRegionServer(0);
  HServerInfo hsi = firstServer.getServerInfo();
  firstServer.setHServerInfo(...);

  // Sleep while the region server pings back
  Thread.sleep(2000);
  assertTrue(firstServer.isOnline());
  assertEquals(2,cluster.getLiveRegionServerThreads().size());
  ... // similarly for secondServer
}
```

Concurrency

(65% of cases)

And if the server takes more than 2 seconds to respond?

Exercises

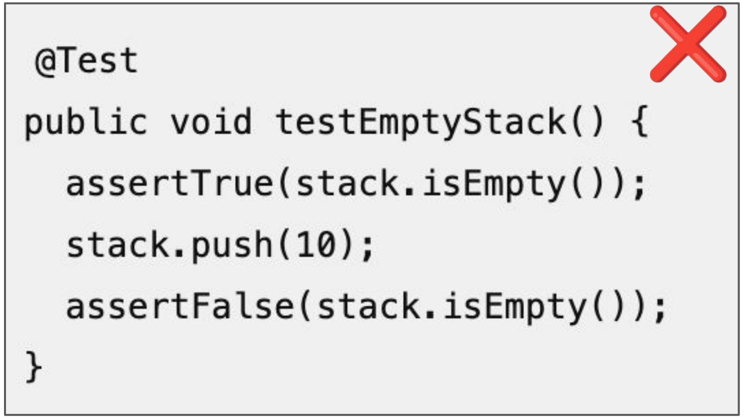
1. If end-to-end tests check "the most important" (i.e., the entire system), why is it not recommended to implement only such tests?

2. What will be printed by the following test?

```
class ExampleTest {
  int i = 10;

  @Test
  public void test1() {
    i++;
    println(i);
  }

  @Test
  public void test2() {
    i++;
    println(i);
  }
}
```

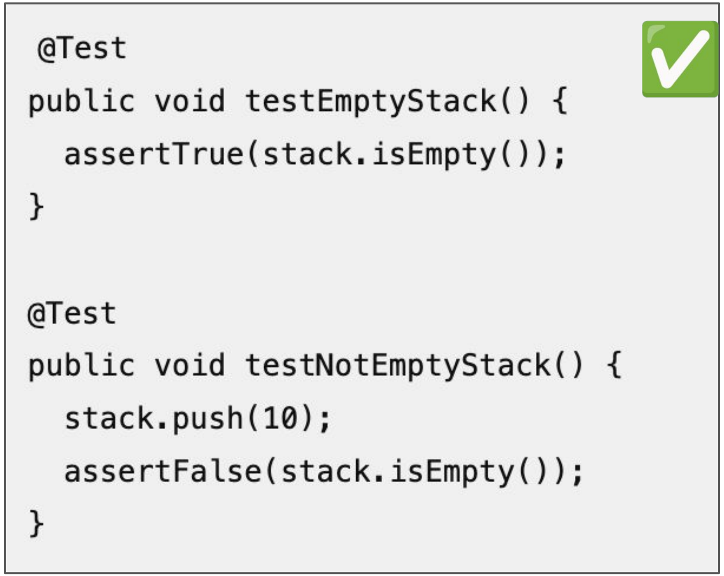
Number of asserts per test

- Most of the time, a single assert / test



❌

```
@Test
public void testEmptyStack() {
  assertTrue(stack.isEmpty());
  stack.push(10);
  assertFalse(stack.isEmpty());
}
```



✅

```
@Test
public void testEmptyStack() {
  assertTrue(stack.isEmpty());
}

@Test
public void testNotEmptyStack() {
  stack.push(10);
  assertFalse(stack.isEmpty());
}
```

But there are exceptions ...

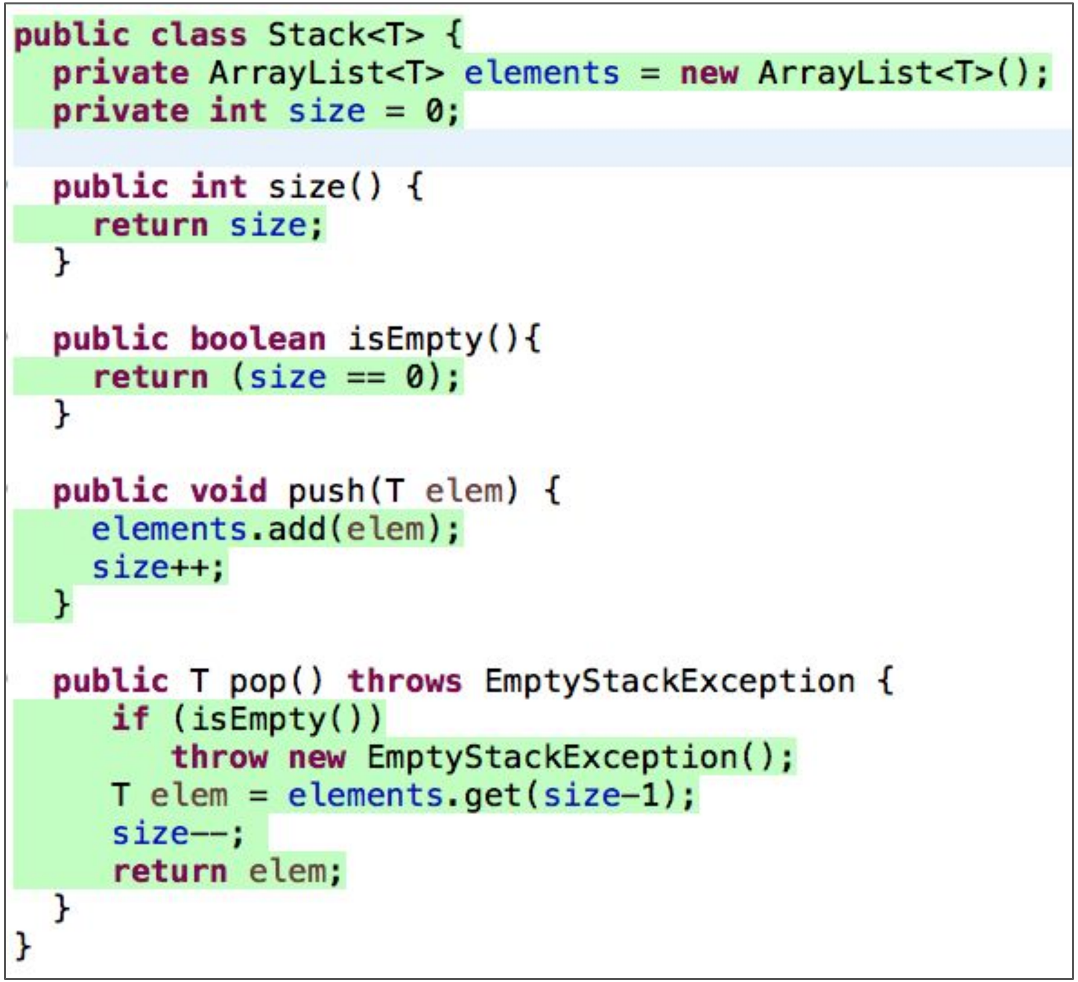
✅

```
 @Test
public void testRepeat() {
  String input = "20";
  assertEquals("", Strings.repeat(input,0));
  assertEquals("20", Strings.repeat(input,1));
  assertEquals("2020", Strings.repeat(input,2));
  assertEquals("202020", Strings.repeat(input,3));
  ...
}
```

How many tests do I have to write?

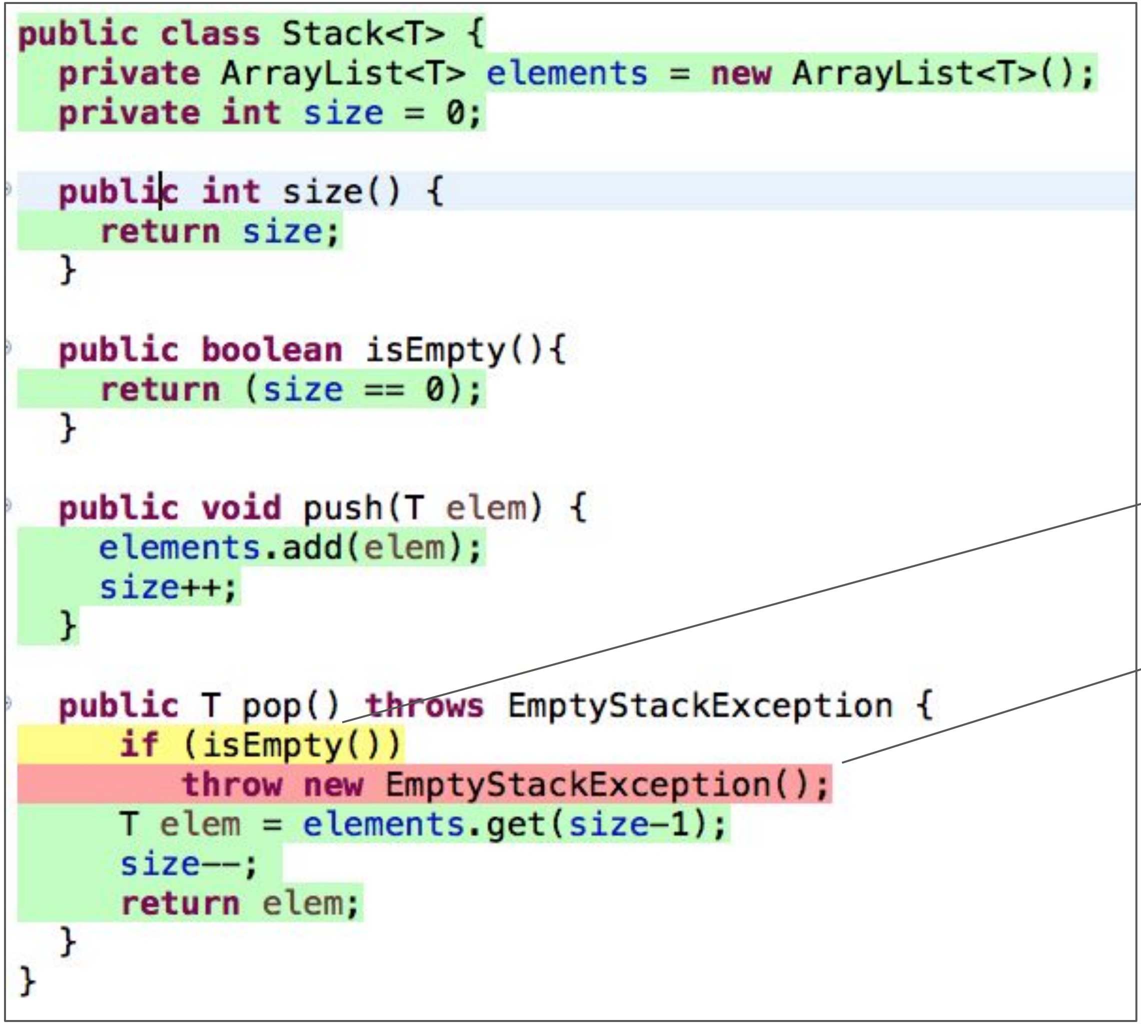
Test Coverage

- Test coverage = (number of statements executed by the tests) / (total number of statements)



```
public class Stack<T> {
  private ArrayList<T> elements = new ArrayList<T>();
  private int size = 0;

  public int size() {
    return size;
  }

  public boolean isEmpty(){
    return (size == 0);
  }

  public void push(T elem) {
    elements.add(elem);
    size++;
  }

  public T pop() throws EmptyStackException {
      if (isEmpty())
          throw new EmptyStackException();
      T elem = elements.get(size-1);
      size--;
      return elem;
  }
}
```

100% coverage



```
public class Stack<T> {
  private ArrayList<T> elements = new ArrayList<T>();
  private int size = 0;

  public int size() {
    return size;
  }

  public boolean isEmpty(){
    return (size == 0);
  }

  public void push(T elem) {
    elements.add(elem);
    size++;
  }

  public T pop() throws EmptyStackException {
     if (isEmpty())
        throw new EmptyStackException();
     T elem = elements.get(size-1);
     size--;
     return elem;
  }
}
```

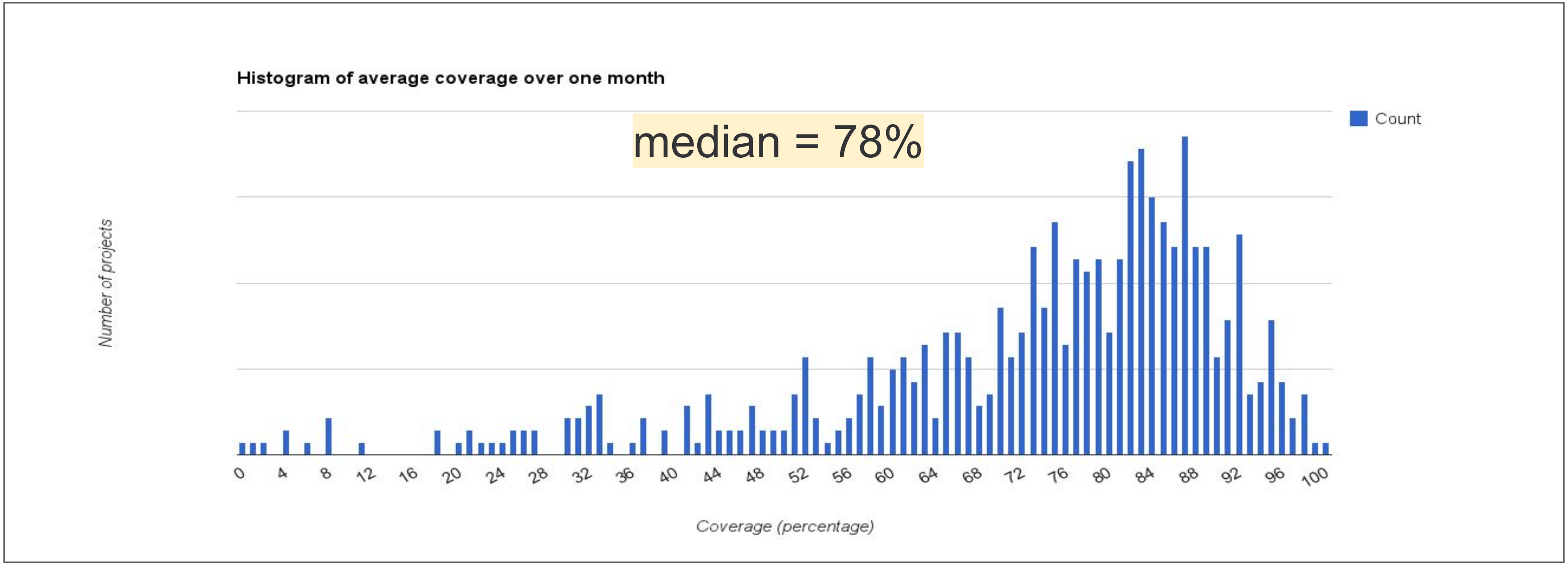
yellow: only one branch is tested;
any “if” has two branches: T and F

red: command not covered by tests

What is the ideal test coverage?

- Varies from project to project, but doesn't need to be 100%
- At least 60%, according to some authors

Example: Google



median = 78%

https://docs.google.com/presentation/d1god5/fDDd1aP6PwhPodOnAZSPpD80lqYDrHhuhyD7Tvg/edit#slide=id.g3f5c82004_99_135

Exercises

1. For the following function, fill in the table with the statements and branch coverage results.

```
void f(int x, int y) {
  if (x > 0) {
     x = 2 * x;
     if (y > 0) {
        y = 2 * y;
     }
  }
}
```

Test	Stm	Brch
f(0,0)		
f(1,1)		
f(0,0) e f(1,1)		

2. In a university, students receive score A if they have a grade greater than or equal to 90. This function implements this requirement:

```
boolean isScoreA(int grade) {
  if (grade > 90)
    return true;
  else return false;
}
```

(a) Does this implementation have a bug? If so, when it result in a failure?

(b) Suppose this function is tested with grades 85 and 95. What is the statement coverage of this test? And the branch coverage?

3. Consider the following statement:

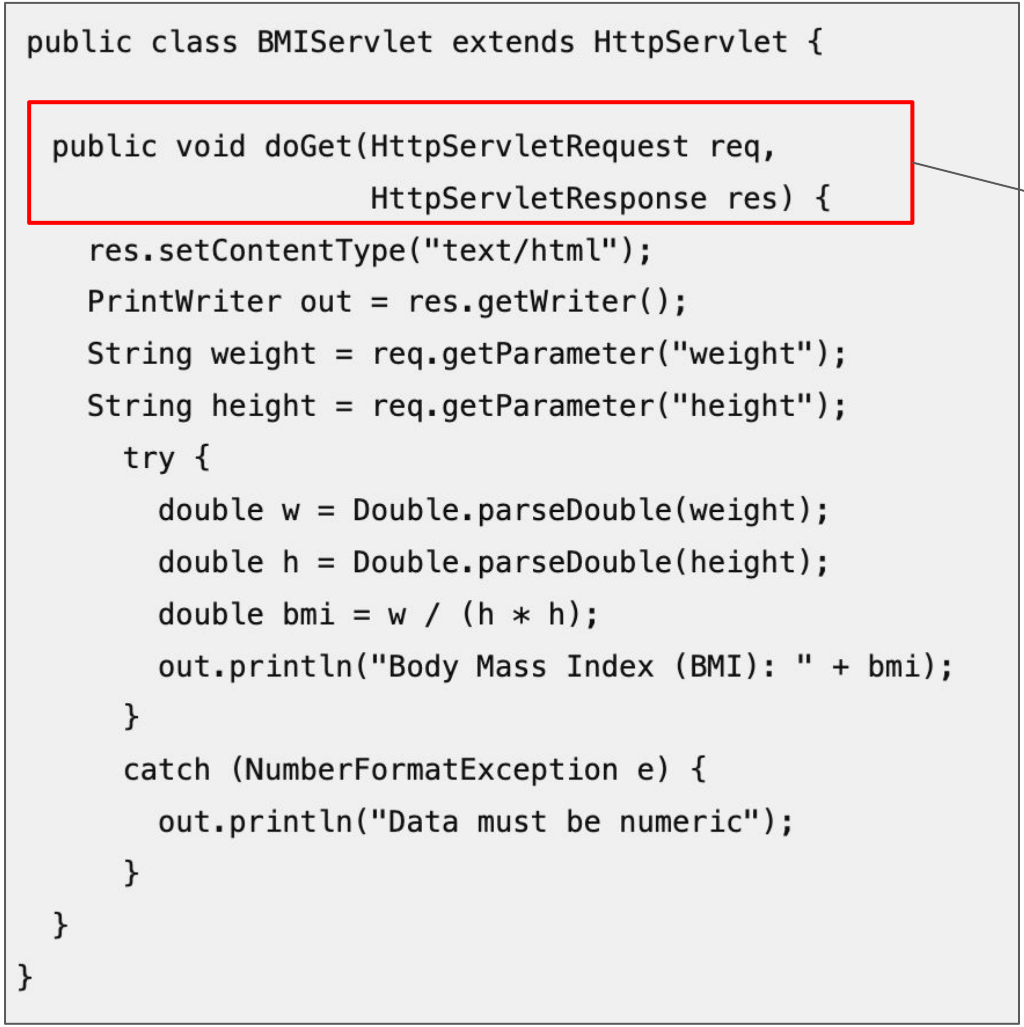
if a program has 100% statememt coverage, it is bug-free.

Is this statement true or false? Justify.

4. Why is it usually not necessary to achieve 100% statement coverage?

Testability

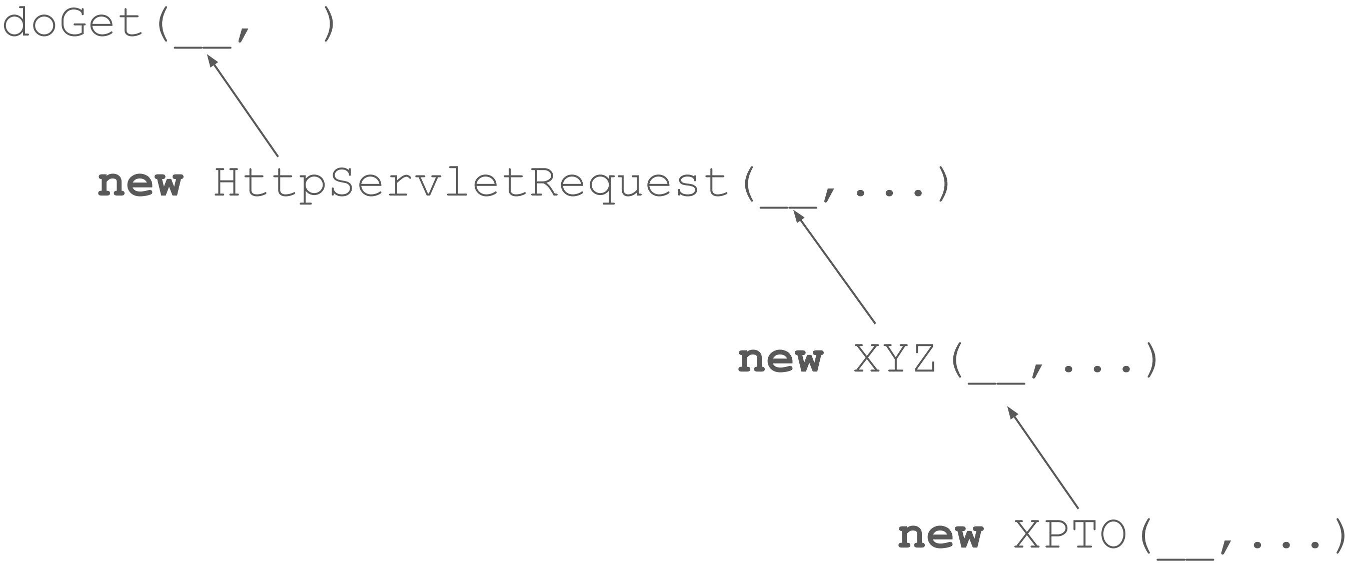
Example: Servlet



```
public class BMIServlet extends HttpServlet {

  public void doGet(HttpServletRequest req,
                    HttpServletResponse res) {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String weight = req.getParameter("weight");
    String height = req.getParameter("height");
      try {
        double w = Double.parseDouble(weight);
        double h = Double.parseDouble(height);
        double bmi = w / (h * h);
        out.println("Body Mass Index (BMI): " + bmi);
      }
      catch (NumberFormatException e) {
        out.println("Data must be numeric");
      }
  }
}
```

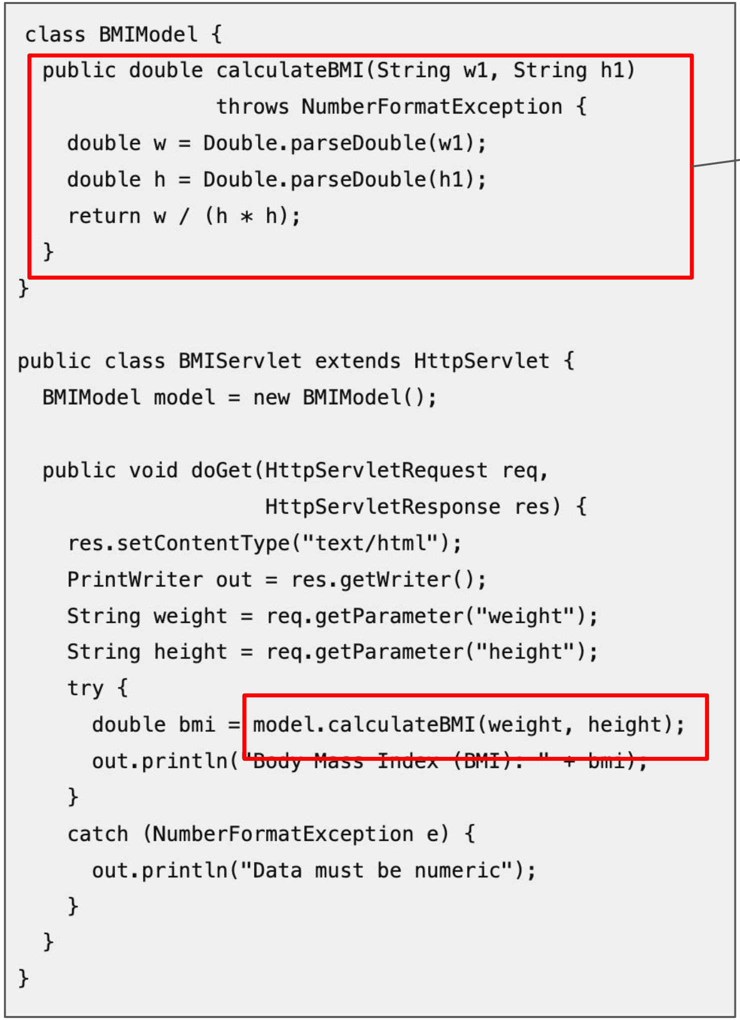
Difficult to test as it has dependencies (parameters) to the Java Servlets package

Problem: calling doGet(...) is not easy...



```
doGet(__,  )

    new HttpServletRequest(__,...)

                    new XYZ(__,...)

                        new XPTO(__,...)
```

Testability of doGet() is low

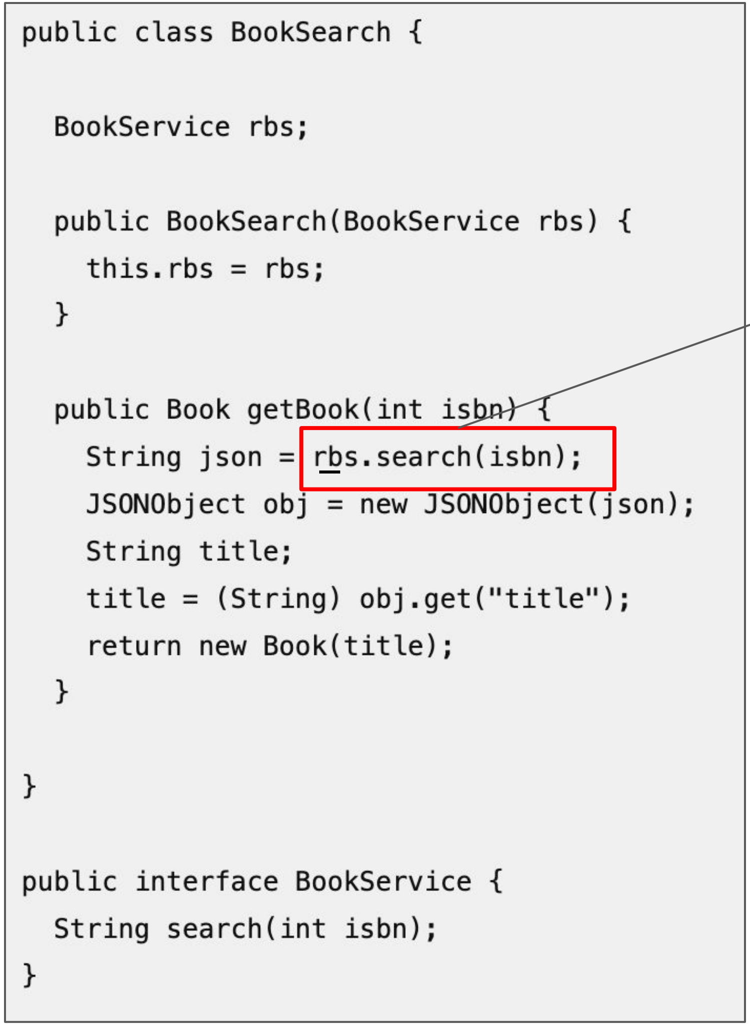


```
class BMIModel {
  public double calculateBMI(String w1, String h1)
                throws NumberFormatException {
    double w = Double.parseDouble(w1);
    double h = Double.parseDouble(h1);
    return w / (h * h);
  }
}

public class BMIServlet extends HttpServlet {
  BMIModel model = new BMIModel();

  public void doGet(HttpServletRequest req,
                    HttpServletResponse res) {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String weight = req.getParameter("weight");
    String height = req.getParameter("height");
    try {
      double bmi = model.calculateBMI(weight, height);
      out.println("Body Mass Index (BMI): " + bmi);
    }
    catch (NumberFormatException e) {
      out.println("Data must be numeric");
    }
  }
}
```

Solution: extraction of the domain rule to a separate and easier to test class

Mocks

Motivating Example



```java
public class BookSearch {

  BookService rbs;

  public BookSearch(BookService rbs) {
    this.rbs = rbs;
  }

  public Book getBook(int isbn) {
    String json = rbs.search(isbn);
    JSONObject obj = new JSONObject(json);
    String title;
    title = (String) obj.get("title");
    return new Book(title);
  }

}

public interface BookService {
  String search(int isbn);
}
```

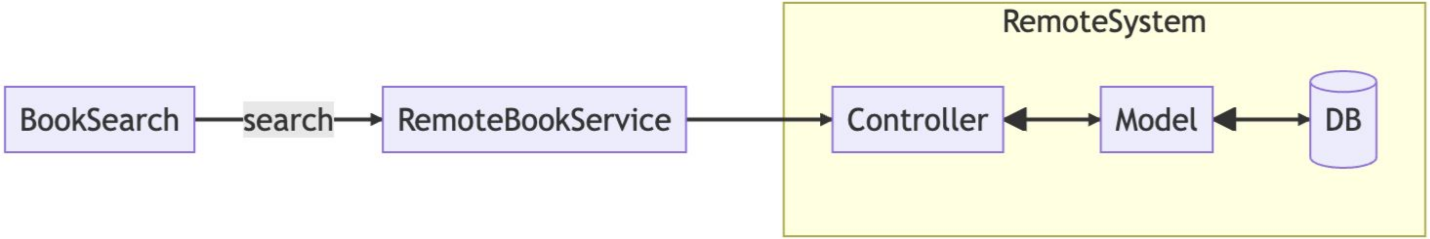
Method that searches for a book in a remote service

Problem: Unit tests should be fast!

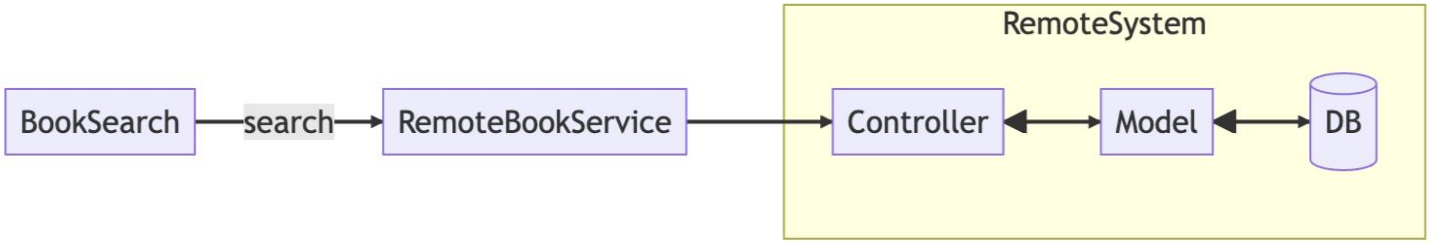
Solution: Mocks

- Object that emulates a real object
- But, it's much simpler than this object

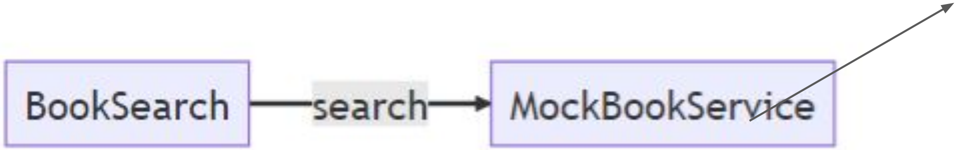
Solution without mocks (slow test as it accesses a remote server)



Solution without mocks (slow test as it accesses a remote server)



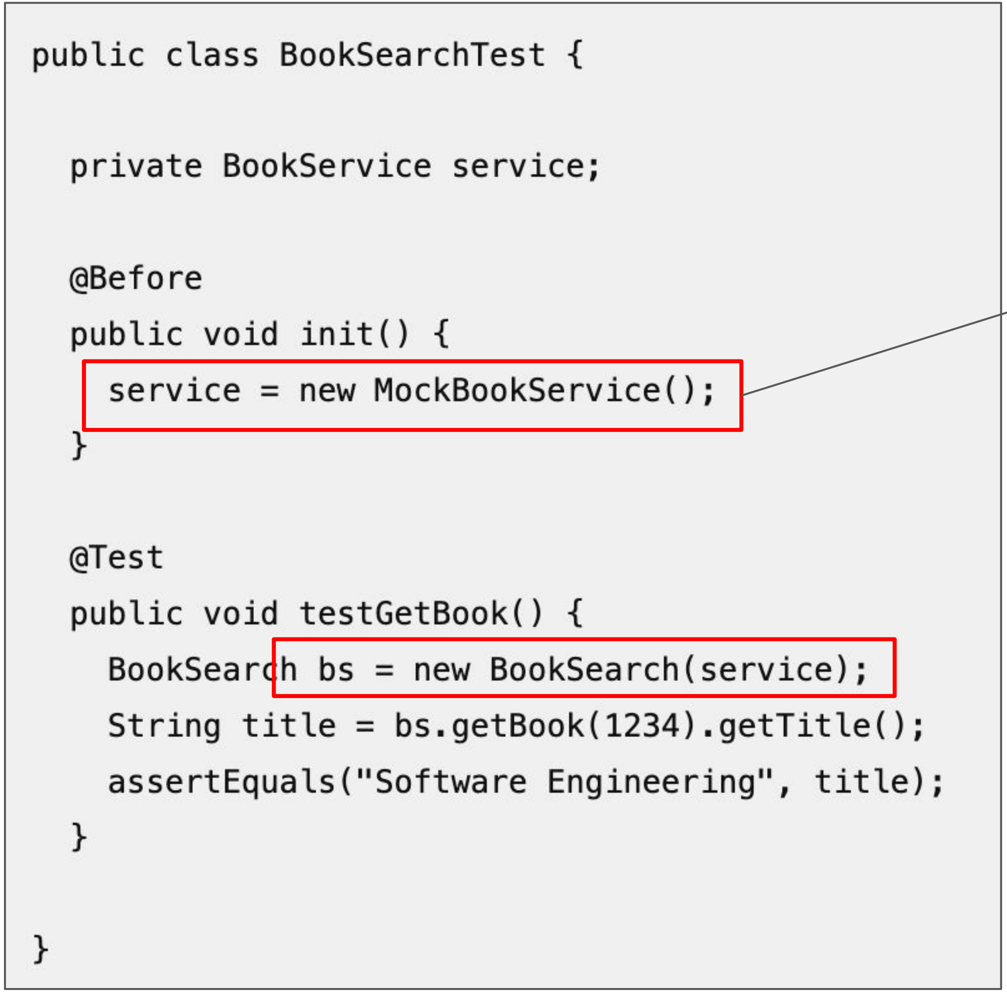
Solution with a mock



Only works with two books:
SOFTENG and NULLBOOK

```
class BookConst {

  public static String SOFTENG =
          "{ \"title\": \"Software Engineering\" }";

  public static String NULLBOOK =
          "{ \"title\": \"NULL\" }";

}

class MockBookService implements BookService {

   public String search(int isbn) {
      if (isbn == 1234)
        return BookConst.SOFTENG;
      return BookConst.NULLBOOK;
   }

}
```

Search on a single ISBN



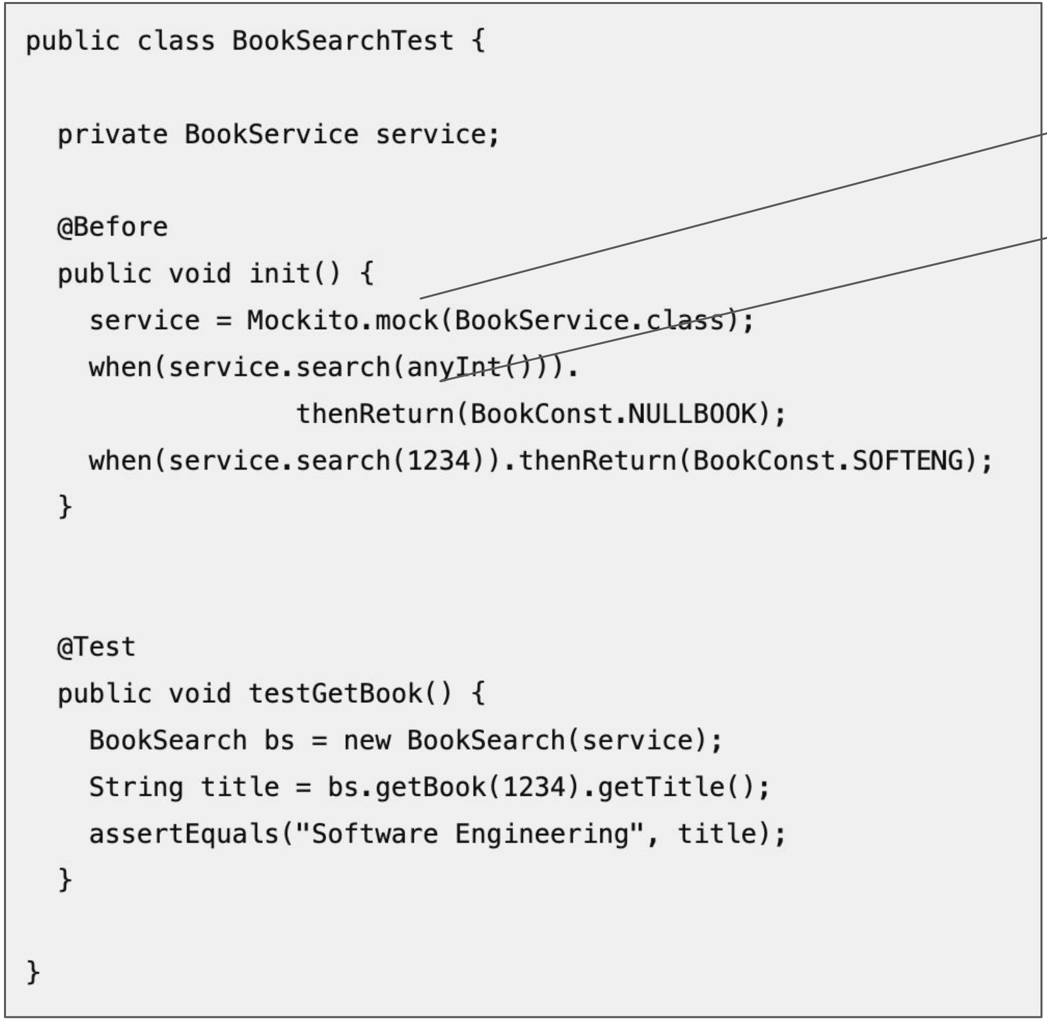
```
public class BookSearchTest {

  private BookService service;

  @Before
  public void init() {
    service = new MockBookService();
  }

  @Test
  public void testGetBook() {
    BookSearch bs = new BookSearch(service);
    String title = bs.getBook(1234).getTitle();
    assertEquals("Software Engineering", title);
  }

}
```

Test uses the mock

Mock Frameworks

Example: Mockito

- Facilitates the implementation of mocks via a domain-specific language
- Dispenses the manual implementation of mocks



```
public class BookSearchTest {

  private BookService service;

  @Before
  public void init() {
    service = Mockito.mock(BookService.class);
    when(service.search(anyInt())).
                 thenReturn(BookConst.NULLBOOK);
    when(service.search(1234)).thenReturn(BookConst.SOFTENG);
  }


  @Test
  public void testGetBook() {
    BookSearch bs = new BookSearch(service);
    String title = bs.getBook(1234).getTitle();
    assertEquals("Software Engineering", title);
  }

}
```

Creates a mock

Programs the mock's behavior

Mock Manual

```
@Before
public void init() {
  service = new MockBookService();
}
```

```
class MockBookService
      implements BookService {

   public String search(int isbn) {
      if (isbn == 1234)
         return BookConst.SOFTENG;
      return BookConst.NULLBOOK;
   }
}
```

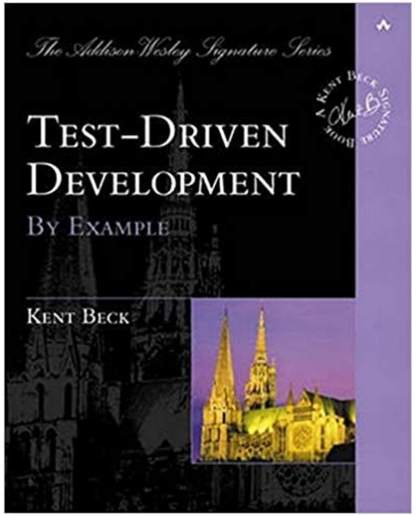
Mockito

```
@Before
public void init() {
  service = Mockito.mock(BookService.class);
  when(service.search(anyInt()).thenReturn(BookConst.NULLBOOK);
  when(service.search(1234)).thenReturn(BookConst.SOFTENG);
}
```

Test-Driven Development (TDD)

TDD

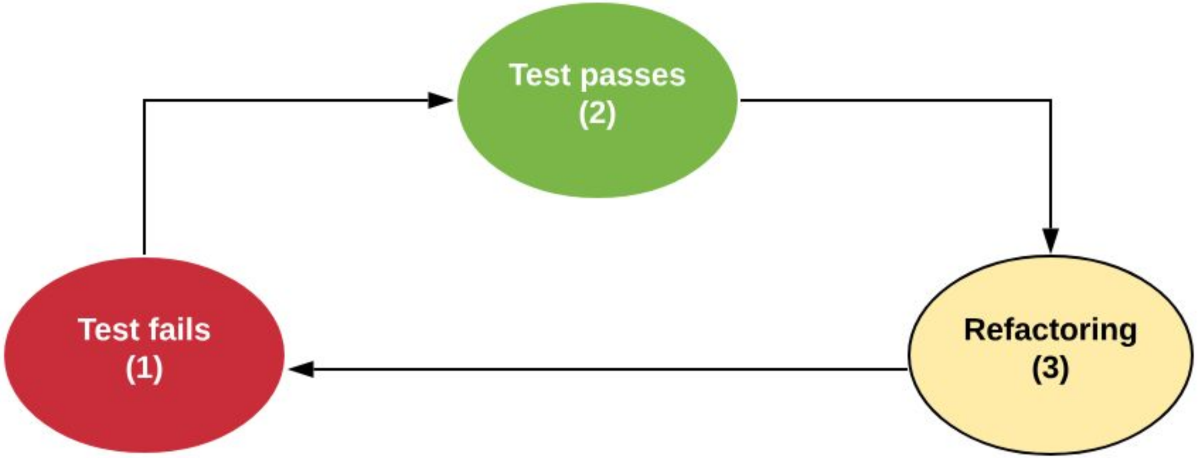
- One of the programming practices proposed by XP
- Idea: write test T before class C


The Addison Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

Benefits

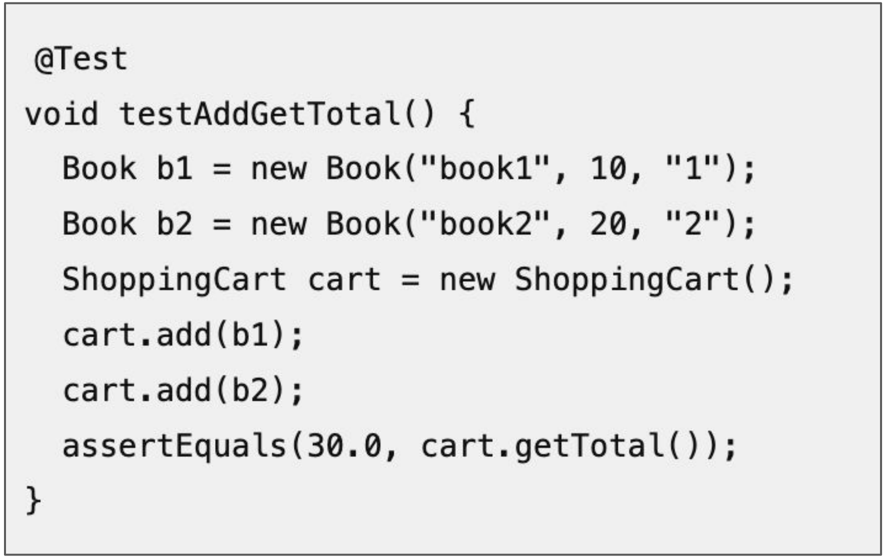
- Prevents devs from forgetting to write tests
- Encourages the writing of code with testability
- Improves the design of the code, as the developer becomes the first user of their code
- Coverage can reach 90%

TDD cycle



Example of TDD: Shopping Cart

Red



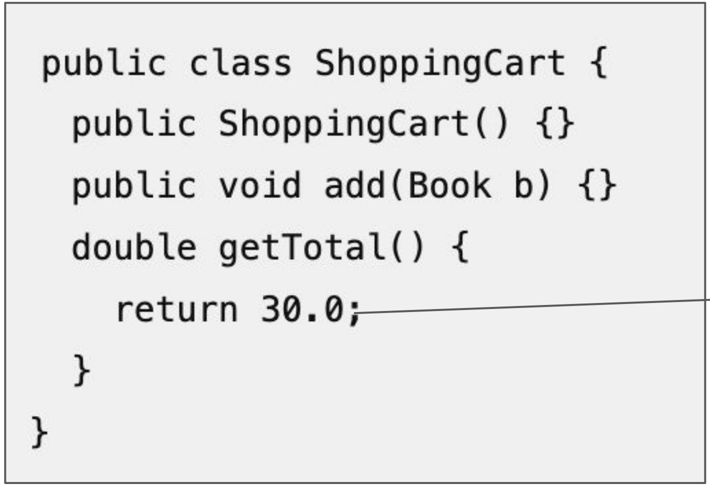
```
@Test
void testAddGetTotal() {
  Book b1 = new Book("book1", 10, "1");
  Book b2 = new Book("book2", 20, "2");
  ShoppingCart cart = new ShoppingCart();
  cart.add(b1);
  cart.add(b2);
  assertEquals(30.0, cart.getTotal());
}
```

Still red, but at least compiling

```
public class Book {
  public String title;
  public double price;
  public String isbn;

  public Book(String title, double price, String isbn) {
    this.title = title;
    this.price = price;
    this.isbn = isbn;
  }

}

public class ShoppingCart {

  public ShoppingCart() {}

  public void add(Book b) {}

  double getTotal() {
    return 0.0;
  }
}
```

Temporary implementations

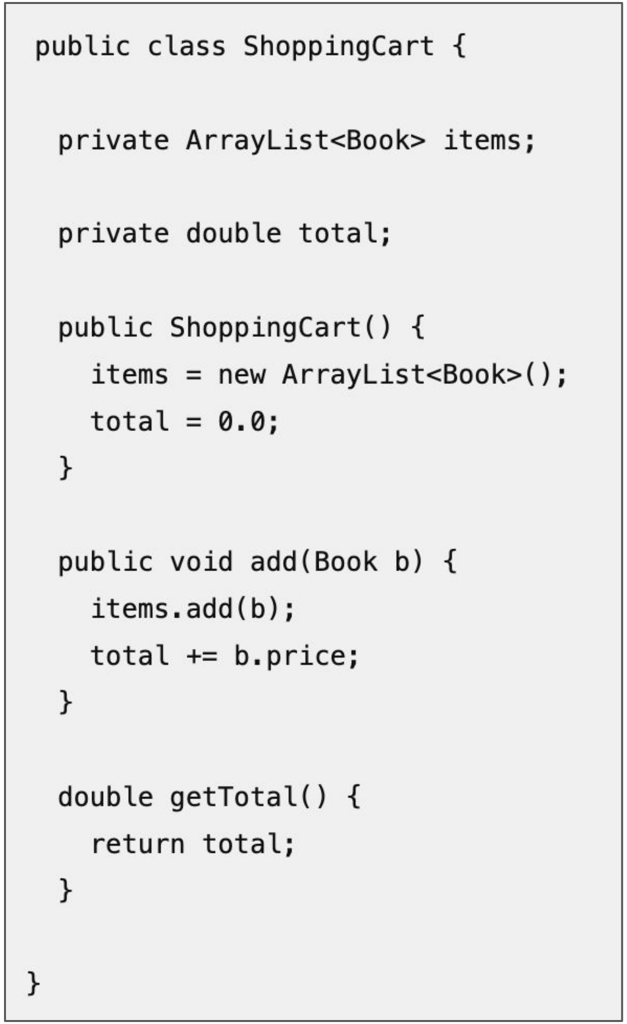
First Green



```
public class ShoppingCart {
  public ShoppingCart() {}
  public void add(Book b) {}
  double getTotal() {
    return 30.0;
  }
}
```

Just for a "small victory" ...
baby steps

Now, a real **green**



```
public class ShoppingCart {

  private ArrayList<Book> items;

  private double total;

  public ShoppingCart() {
    items = new ArrayList<Book>();
    total = 0.0;
  }

  public void add(Book b) {
    items.add(b);
    total += b.price;
  }

  double getTotal() {
    return total;
  }

}
```

Yellow: can we refactor and improve the code?

```
public class Book {
  private String title;
  private double price;
  private String isbn;

  public Book(String title, double price, String isbn)
{
    this.title = title;
    this.price = price;
    this.isbn = isbn;
  }
}
```

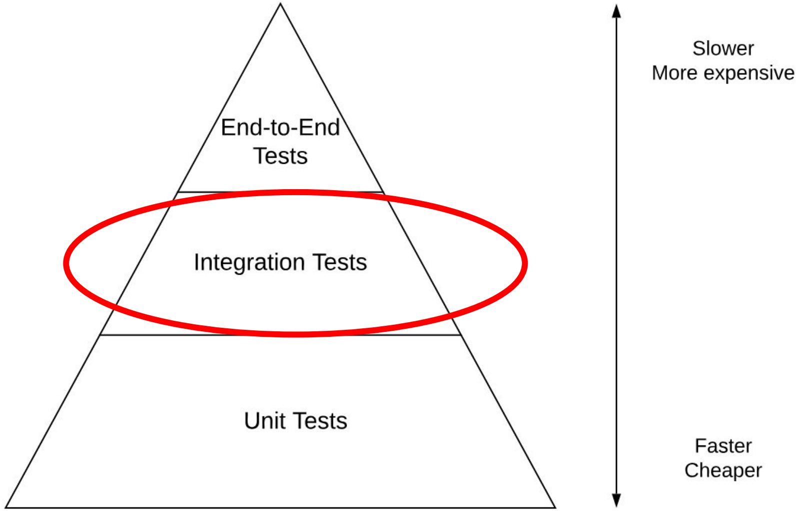
Next Step?

- Do we need more features?
- If yes, new TDD cycle (red-green-yellow)

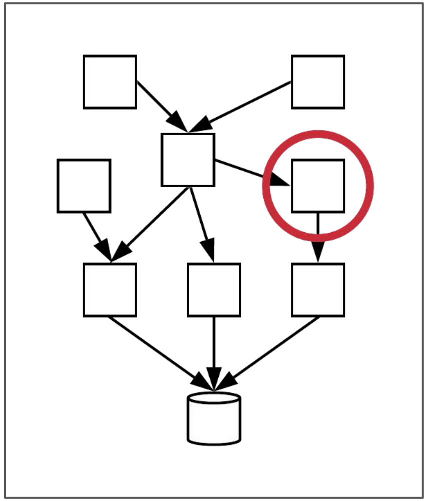
Integration Tests

Integration Tests

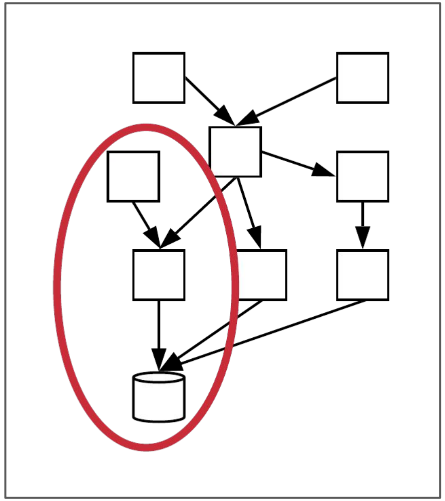
- Test a feature or service
- Including external services (DB, for example)



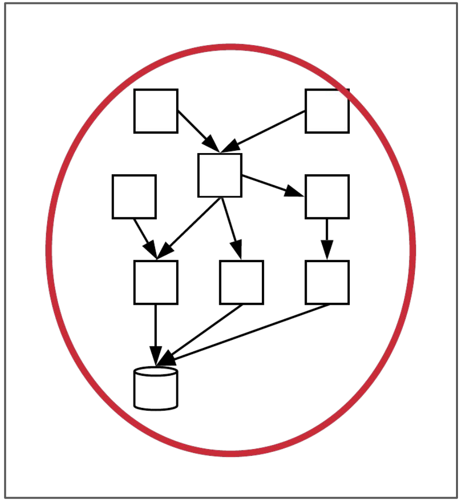
Remembering ...



Unit



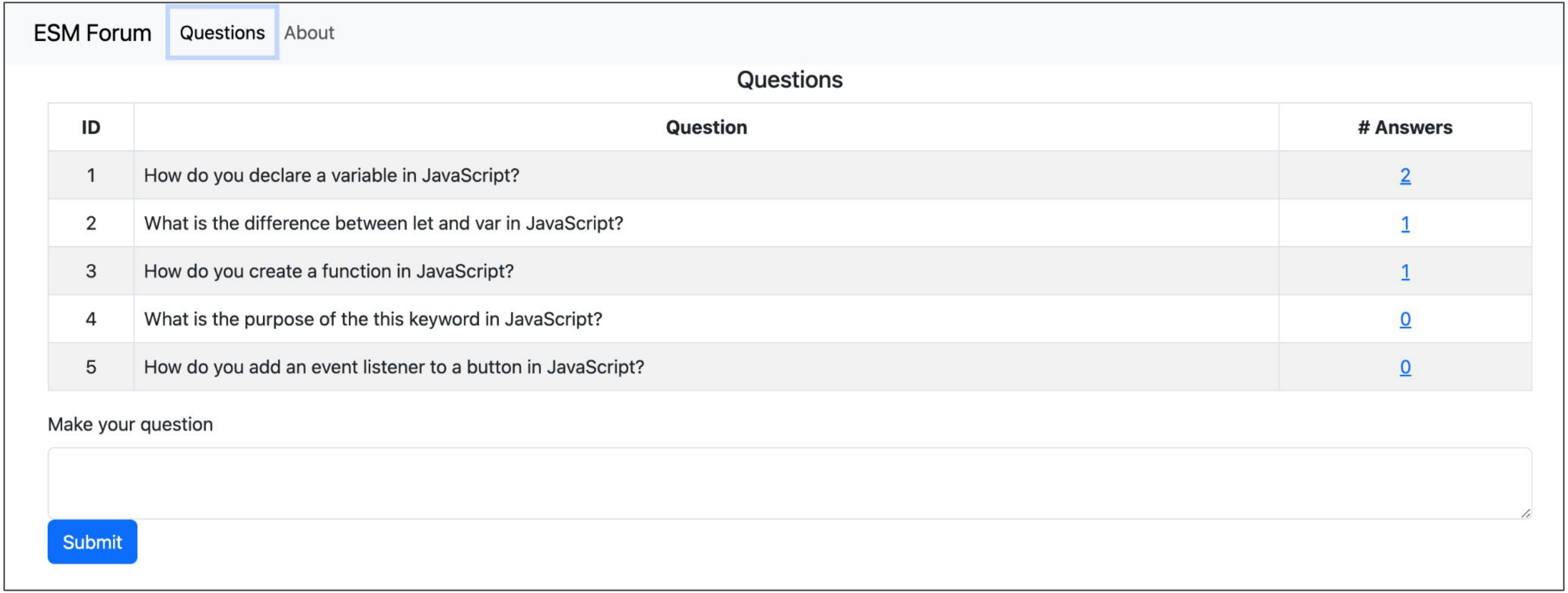
Integration



End-to-End

Example of Integration Test

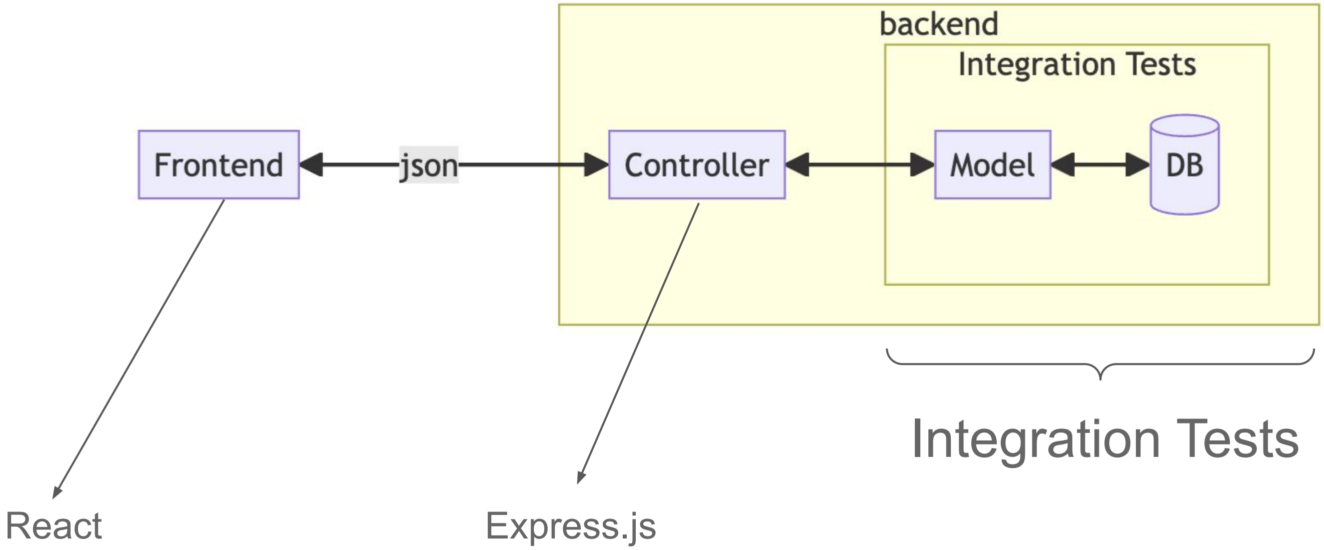
Q&A Forum with a frontend (React) and a backend (Express.js)



Questions

ID	Question	# Answers
1	How do you declare a variable in JavaScript?	2
2	What is the difference between let and var in JavaScript?	1
3	How do you create a function in JavaScript?	1
4	What is the purpose of the this keyword in JavaScript?	0
5	How do you add an event listener to a button in JavaScript?	0

Make your question



React

Express.js

Integration Tests

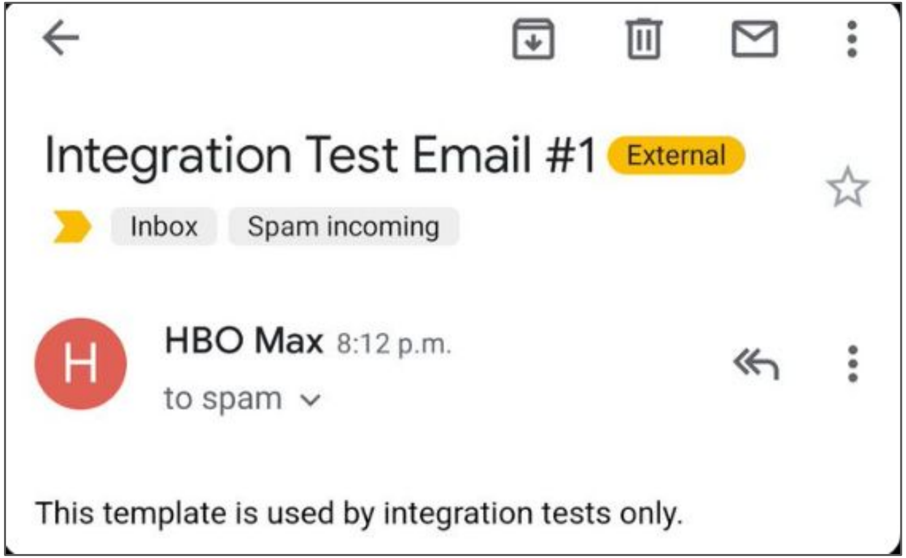
Jest

```
beforeEach(() => {
  bd.reconfig('./db/qa-forum-test.db');

  // clears all tables
  bd.exec('delete from questions', []);
  bd.exec('delete from answers', []);
});

test('Testing empty database', () => {
   expect(model.list_questions().length).toBe(0);
});
```

```
test('Creating three questions', () => {
  model.add_question('1 + 1 = ?');
  model.add_question('2 + 2 = ?');
  model.add_question('3 + 3 = ?');
  const questions = model.list_questions();

  expect(questions.length).toBe(3);
  expect(questions[0].text).toBe('1 + 1 = ?');
  expect(questions[1].text).toBe(2 + 2 = ?');
  expect(questions[2].num_answers).toBe(0);
  expect(questions[1].id_question).toBe(questions[2].id_question-1);
});
```

Exercises

1. In June 2021, the following email was sent by mistake to thousands of HBO subscribers. What might have happened for this mail to reach HBO's end users?



Integration Test Email #1 External

Inbox Spam incoming

HBO Max 8:12 p.m.

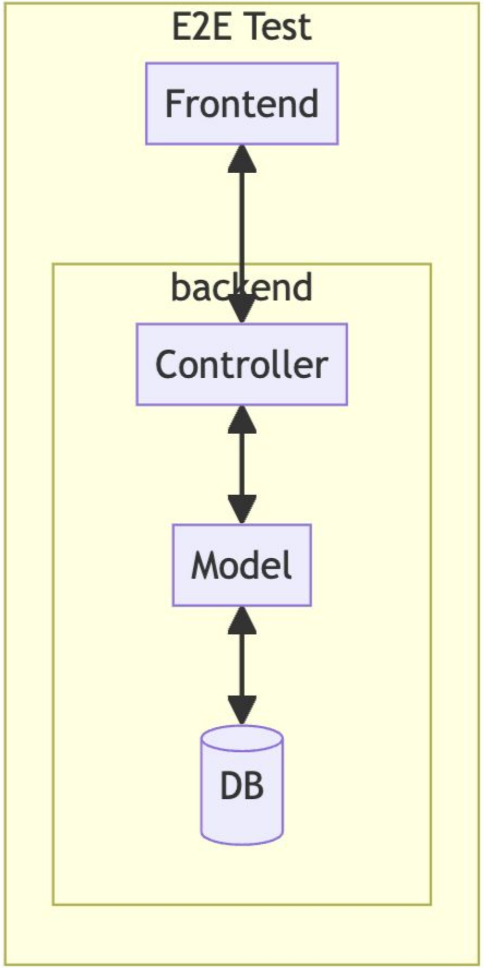
to spam

This template is used by integration tests only.

End-to-End Tests

End-to-End Tests

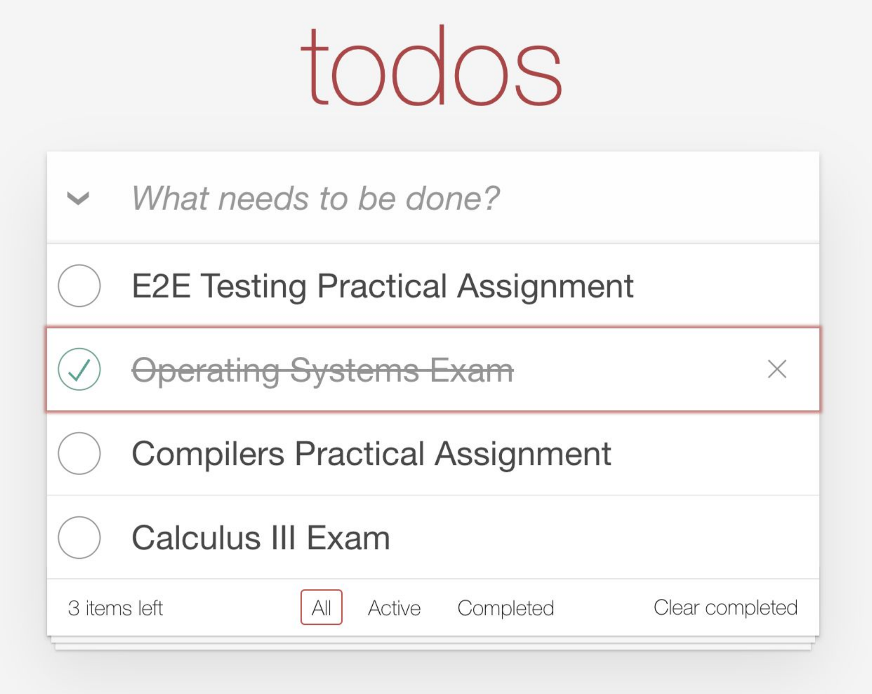
- Test the whole system via its external interface
- The test simulates a person using the system (filling in data, clicking on buttons, etc.).
- Also called: system tests, frontend tests, web UI tests



E2E Test

cypress

Example of E2E Test



What needs to be done?

○ E2E Testing Practical Assignment

✓ ~~Operating Systems Exam~~ ×

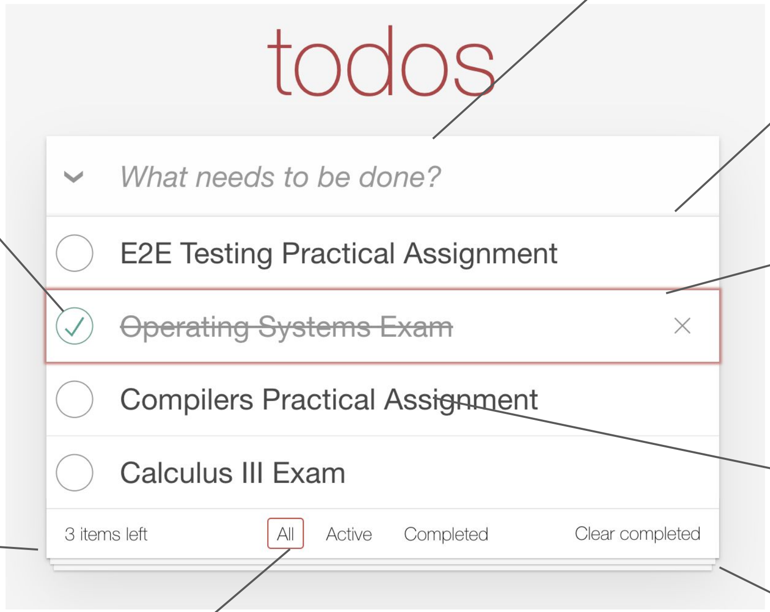
○ Compilers Practical Assignment

○ Calculus III Exam

https://todomvc.com

HTML & CSS Selectors

```
<input class="new-todo" placeholder="What needs to be done?"/>
...
<input class="toggle" type="checkbox" {{checked}}>
...
<button class="destroy"></button>
...
<span class="todo-count"></span>
...
<input class="toggle-all" type="checkbox" />
...
<button class="clear-completed">Clear completed</button>
...
```



todos

What needs to be done?

○ E2E Testing Practical Assignment

✓ ~~Operating Systems Exam~~ ×

○ Compilers Practical Assignment

○ Calculus III Exam

3 items left | All | Active | Completed | Clear completed

.new-todo (input)

.todo-list (container)

.todo-list li .toggle
(checkbox)

.todo-list li (Item)

.todo-list li .destroy
(visible on hover)

.todo-list li label (text)

.todo-count

.toggle-all
(checkbox)

.clear-completed (button)

```
it('Checking if the app is opening', () => {
    cy.visit('http://127.0.0.1:7001/')
  })
```

```
it('Inserting a task', () => {
    cy.visit('http://127.0.0.1:7001');

    cy.get('.new-todo')
      .type('E2E Testing Practical Assignment{enter}');

    cy.get('.todo-list li')
      .should('have.length', 1)
      .first()
      .should('have.text', 'E2E Testing Practical Assignment');
  });
```

```
it('Inserting and deleting a task', () => {
    cy.visit('http://127.0.0.1:7001');

    cy.get('.new-todo')
      .type('E2E Testing Practical Assignment{enter}');

    cy.get('.todo-list li .destroy')
      .invoke('show')
      .click();

    cy.get('.todo-list li')
      .should('have.length', 0);
  });
```

```
it('Selecting completed and active tasks', () => {
    cy.visit('http://127.0.0.1:7001');

    cy.get('.new-todo')
      .type('SoftEng Practical Assignment{enter}')
      .type('SoftEng Exam{enter}');

    cy.get('.todo-list li .toggle')
      .first()
      .click();

    cy.contains('Active').click();
    cy.get('.todo-list li')
      .should('have.length', 1)
      .first()
      .should('have.text', 'SoftEng Exam');
```

Continue…

Continuation…

```
    cy.contains('Completed').click();
    cy.get('.todo-list li')
      .should('have.length', 1)
      .first()
      .should('have.text', 'SoftEng Practical Assignment');

    cy.contains('All').click();
    cy.get('.todo-list li')
      .should('have.length', 2);
  });
```

End