UNIT TESTING IN JAVA

JUPITER API

JUnit 5

GREEN LEARNER ARVIND

INTRODUCTION

What is unit testing??

JUNIT??

- Java library for writing unit tests
- Version history
    - Under 4.x > 4.13
    - Then 4.x > 5 >> Major changes done

JUNIT4.x vs JUNIT5.x

- Architecture level difference
- Annotation name difference

JUNIT4.x vs JUNIT5.x

- Architecture level difference
- Annotation name difference
- What’s new
  - Java 8 lambda
  - Different style of testing

JUNIT4.x vs JUNIT5.x

Architecture level difference

- Minimum JDK version for **Junit4 -> Jdk5** but for **Junit5 -> Jdk8**
- Earlier it was single .jar
- Now it's 3 different .jar

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

- **JUnit platform**
  - Launch testing frameworks
  - Define test engines
  - Console launcher
  - Junit4 based runner
  - Integration with 3rd party

- **JUnit Jupiter**
  - New programming model + extension to write test and extension
  - It's subproject provides test engine to run the Jupiter based tests

- **JUnit Vintage**
  - To run tests that are written in junit4 and junit3

JUNIT4.x VS JUNIT5.x

Annotation names difference

Junit4	Junit5
@Before	@BeforEach
@After	@AfterEach
@BeforClass	@BeforAll
@AfterClass	@AfterAll
@Ignore	@Disabled
@Category	@Nested

New Features/Annotations in JUnit5.x

- @TestFactory
- @Nested
- @ExtendWith
- New way to test exceptions
- @SelectPackages and @SelectClasses

#2

UNIT TESTING IN JAVA

JUPITER API

JUnit 5

GREEN LEARNER ARVIND

ENVIRONMENT SETUP & WRITING FIRST UNIT TEST

#3

UNIT TESTING IN JAVA

JUPITER API

JUnit 5

GREEN LEARNER ARVIND

Test Classes, Life Cycle Methods, Test Methods

TEST CLASS

- Any class that contains at least on test method
- *Test classes must not be abstract and*
- *must have a single constructor.*

TEST METHODS

- Any instance method that is directly annotated or meta-annotated with
  - @Test,
  - @RepeatedTest,
  - @ParameterizedTest,
  - @TestFactory, or
  - @TestTemplate.

LIFE CYCLE METHODS

- any method that is directly annotated or meta-annotated with
  - @BeforeAll,
  - @AfterAll,
  - @BeforeEach, or
  - @AfterEach.

LIFE CYCLE METHODS

- any method that is directly annotated or meta-annotated with
  - @BeforeAll,
  - @AfterAll,
  - @BeforeEach, or
  - @AfterEach.

FEW POINTS TO NOTE

- test methods and lifecycle methods must not be abstract and must not return a value.
- Test classes, test methods, and lifecycle methods are not required to be public, but they must not be private.

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

Writing tests for our Application

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

Exception Testing

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

Timeout testing

@Timeout | assertTimeout()

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

Assertions | assertAll | @Nested

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

@RepeatTest

#9

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

@ParameterizedTest

UNIT TESTING IN JAVA

JUPITER API

Test Execution Order

&

Test Instance

FEW POINTS..

- Execution order
  - Default – Fixed, non-deterministic
  - Random
  - Alphanumeric
  - Order annotation
- Test Instance
  - Per method – default
  - Per class

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

@TempDir & Creating Custom Test Annotation

#12

UNIT TESTING IN JAVA

JUnit 5

JUPITER API

Tagging and Filtering

&

??What Next??

What Next???

Tasty mocking framework for unit tests in Java

https://site.mockito.org/

Introduction What/Why

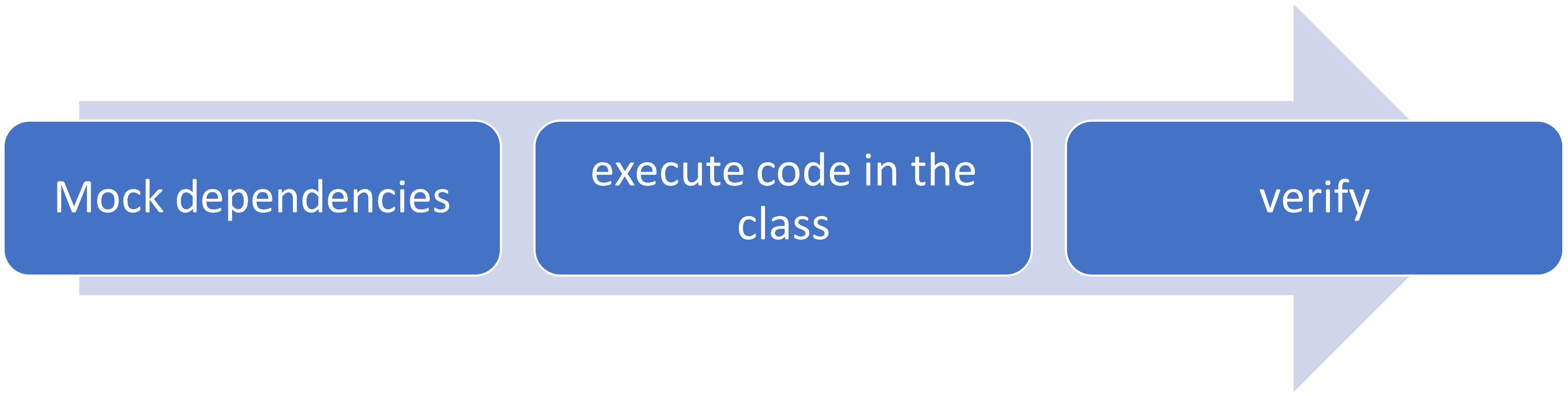
WHY MOCKING IS NEEDED??

- A unit test should test functionality in isolation. Side effects from other classes or the system should be eliminated for a unit test, if possible.

TEST DOUBLES

- Test replacement of real/external dependencies
  - Dummy Object
    - Passed around but never used
  - Fake objects
    - Like memory db for actual db
  - A stub class
    - Initialize > exercises > verify
    - To test data
  - Mock object
    - Initialize > set expectations > exercise > verify
    - To test behavior

Mockito is a mocking framework for Java. Mockito allows convenient creation of substitutes of real objects for testing purposes.



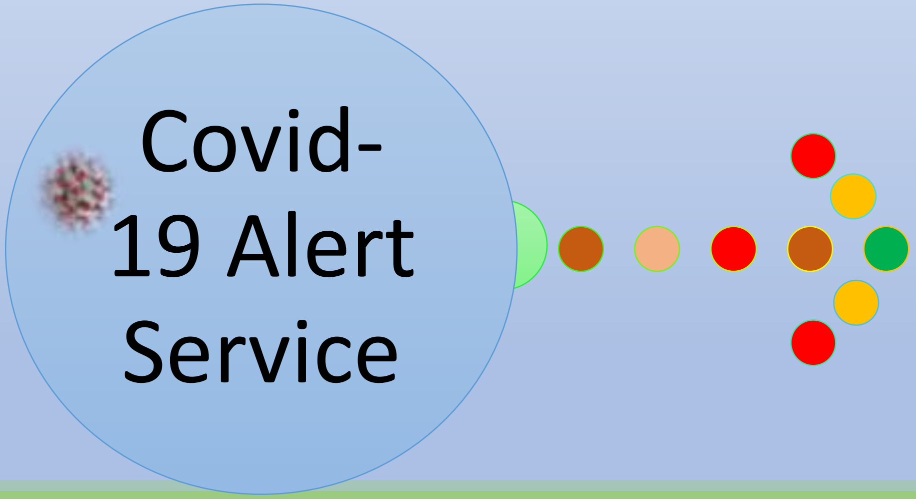
Let’s see some code examples

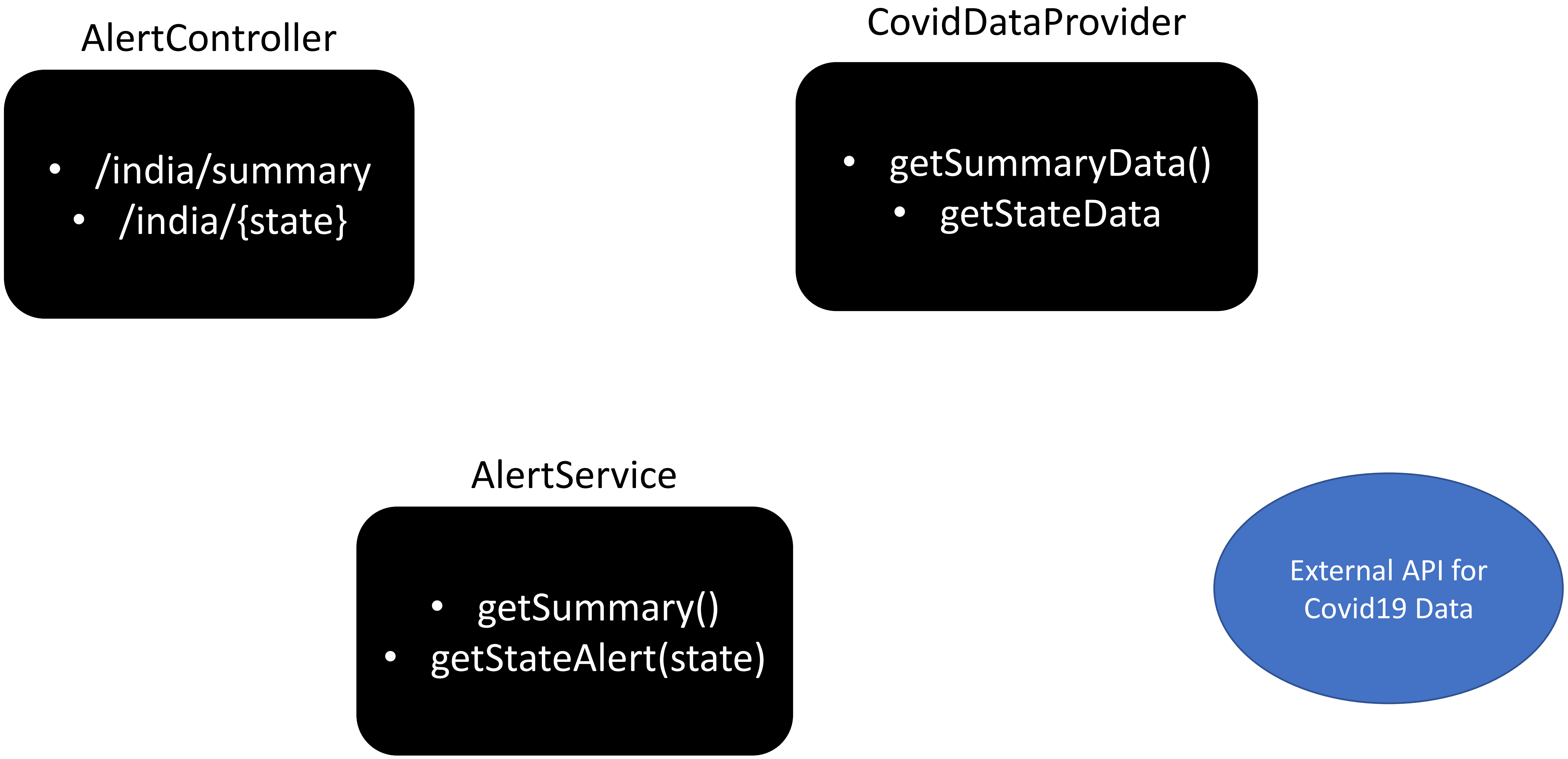
LIMITATIONS OF MOCKITO LIBRARY

- Can’t mock
  - Static methods
  - Private methods
- Can’t mock constructors
- Detailed –
  - https://github.com/mockito/mockito/wiki/FAQ#what-are-the-limitations-of-mockito

ARVIND

WRITING TESTS FOR MICROSERVICES




AlertController
• /india/summary
• /india/{state}
CovidDataProvider
• getSummaryData()
• getStateData
AlertService
• getSummary()
• getStateAlert(state)
External API for Covid19 Data

REFS

- https://api.rootnet.in/
- https://api.rootnet.in/covid19-in/stats/latest

Complete course about TESTING MICROSERVICES (Daily video from19-Jul-2020 at 8.00 PM IST)

- ❖Junit5/Jupiter API - Published
- ❖Mockito – 8.00PM IST, 19-Jul-2020
- ❖Microservice(Covid19AlertService) – creating from scratch with spring boot - 8.00PM IST, 20-Jul-2020
- ❖Writing tests for microservice – spring boot - 8.00PM IST, 21-Jul-2020
- ❖Code coverage – Introduction and integration with spring boot - 8.00PM IST, 22-Jul-2020
- ❖Sonarqube – Introduction/Local Env setup / Integration with spring boot - 8.00PM IST, 23-Jul-2020
- ❖Mutation testing - 8.00PM IST, 24-Jul-2020
- ❖Wiremock - 8.00PM IST, 26-Jul-2020
- ❖Spring cloud contract - 8.00PM IST, 28-Jul-2020

JUNIT??

ARVIND

spring boot

CODE COVERAGE

INTRODUCTION &

INTEGRATION WITH SPRING BOOT

JACOCO

Java Code Coverage

Covid-19 Alert Service

What is Code Coverage??

- Code coverage is a measurement of how many lines/blocks of your code are executed while the automated tests are running.

How It's Measured??

- Code coverage is collected by using a specialized tool to instrument the binaries to add tracing calls and run a full set of automated tests against the instrumented product.
- Steps
    - Source/Intermediate code instrumentation
    - Runtime information collection

Code Coverage Criteria

- Function Coverage –
  - The functions in the source code that are called and executed at least once.
- Statement Coverage –
  - The number of statements that have been successfully validated in the source code.
- Path Coverage –
  - The flows containing a sequence of controls and conditions that have worked well at least once.
- Branch or Decision Coverage –
  - The decision control structures (loops, for example) that have executed fine.
- Condition Coverage –
  - The Boolean expressions that are validated and that executes both TRUE and FALSE as per the test runs.

Integration with spring boot and Gradle

https://docs.gradle.org/current/userguide/jacoco_plugin.html

Why do we measure code coverage??

- Following reasons:
  - To know how well our tests actually test our code
  - To know whether we have enough testing in place
  - To maintain the test quality over the lifecycle of a project
- While code coverage is a good metric of how much testing you are doing, it is not necessarily a good metric of how well you are testing your product.

ARVIND

sonarqube

Your teammate for

Code Quality and Security

INTRODUCTION &

INTEGRATION WITH SPRING BOOT

Covid-19 Alert Service

SonarQube

- An open-source platform developed by SonarSource for continuous **inspection of code quality**
- Perform automatic reviews
- Static analysis of code to **detect bugs, code smells, and security vulnerabilities** on 20+ programming languages.
- SonarQube offers reports on **duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities**.

ARVIND

MUTATION TESTING

TESTING THE UNIT TESTS

pitest.org

Real world mutation testing

Covid-19 Alert Service

WHAT IS MUTATION TESTING

- Faults (or mutations) are automatically seeded into your code, then your tests are run. If your tests fail then the mutation is killed, if your tests pass then the mutation lived.
- The quality of your tests can be gauged from ***the percentage of mutations killed***.

WHY??

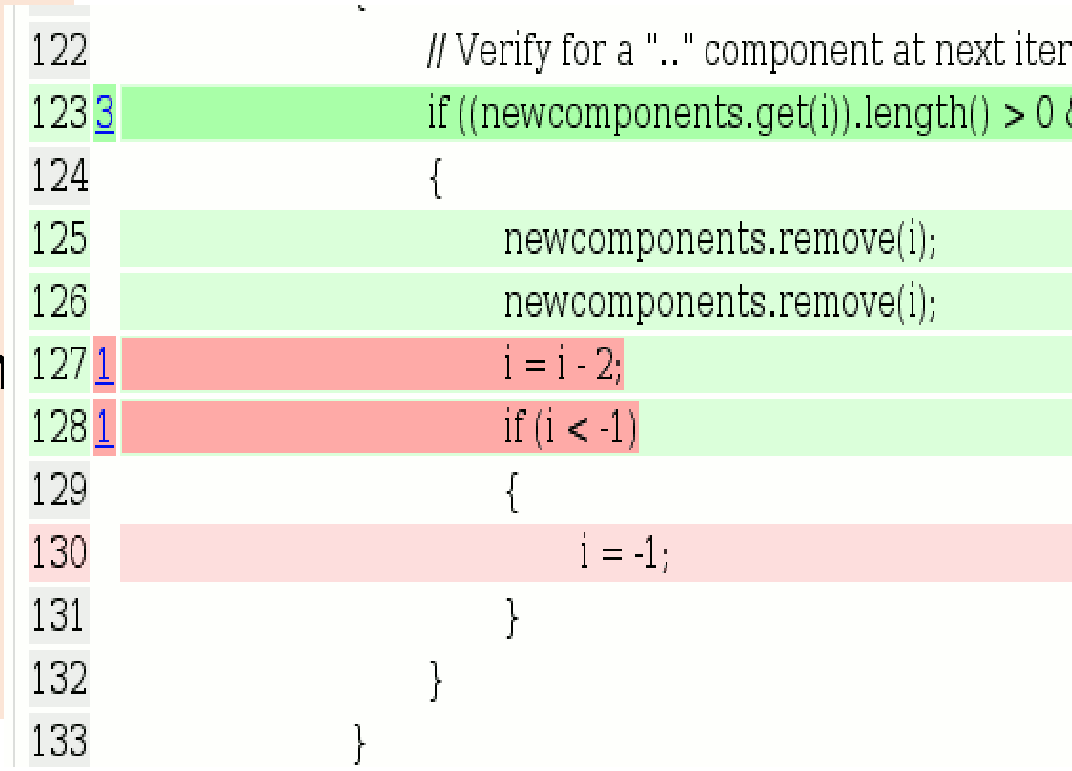
- Traditional test coverage (i.e line, statement, branch, etc.) measures only which code is **executed** by your tests.
- It does **not** check that your tests are actually able to **detect faults** in the executed code. It is therefore only able to identify code that is definitely **not tested**.
- The most extreme examples of the problem are tests with no assertions. Fortunately these are uncommon in most code bases.
- As it is actually able to detect whether each statement is meaningfully tested, mutation testing is the gold standard against which all other types of coverage are measured.

GOAL

- identify weakly tested pieces of code (those for which mutants are not killed)
- identify weak tests (those that never kill mutants)
- compute the mutation score
- learn about error propagation and state infection in the program

PIT – Java lib for mutation testing

- PIT is
  - **Fast** - can analyse in minutes what would take earlier systems days
  - Easy to use - works with ant, maven, gradle and others
  - **Actively developed**
  - **Actively supported**
- The reports produced by PIT are in an easy to read format combining line coverage and mutation coverage information.

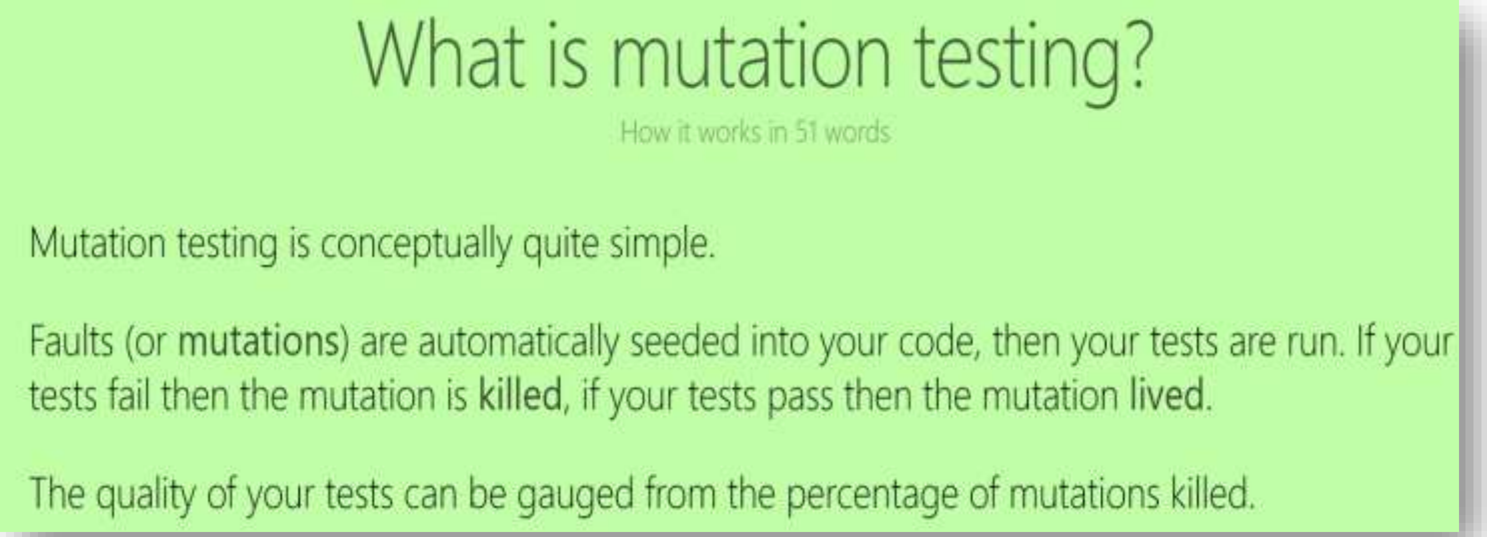


```
                // Verify for a ".." component at next iter
                if ((newcomponents.get(i)).length() > 0 &
                {
                    newcomponents.remove(i);
                    newcomponents.remove(i);
                    i = i - 2;
                    if (i < -1)
                    {
                        i = -1;
                    }
                }
            }
```

Integration with spring boot and Gradle

https://github.com/devcon5io/mutation-analysis-plugin/wiki
https://nwillc.wordpress.com/2016/12/26/together-pit-sonarqube-and-gradle/
https://pitest.org/

JUNIT??



What is mutation testing?

How it works in 51 words

Mutation testing is conceptually quite simple.

Faults (or **mutations**) are automatically seeded into your code, then your tests are run. If your tests fail then the mutation is **killed**, if your tests pass then the mutation **lived**.

The quality of your tests can be gauged from the percentage of mutations killed.

JUNIT??