Unit testing

In Java

Hany Ahmed
i@namozag.com

Unit testing, Maven, JUnit, AssertJ, Mockito

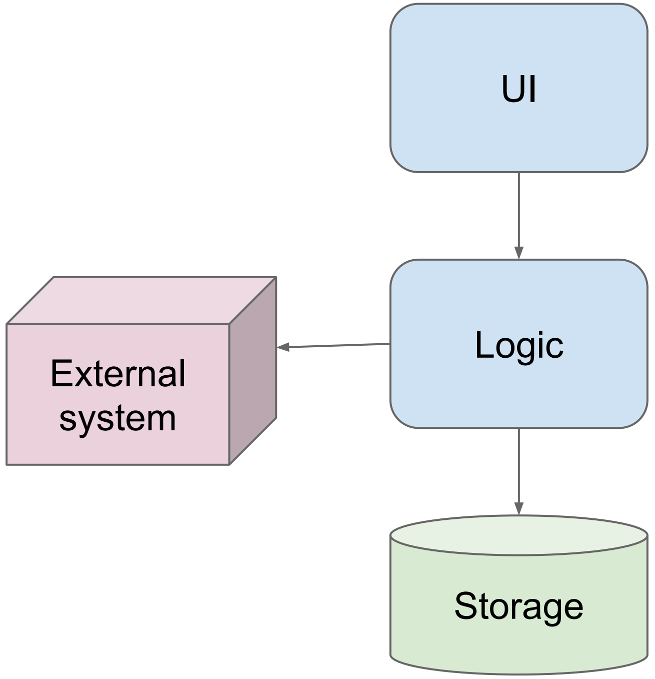
Motivation

Normal application structure

Modules

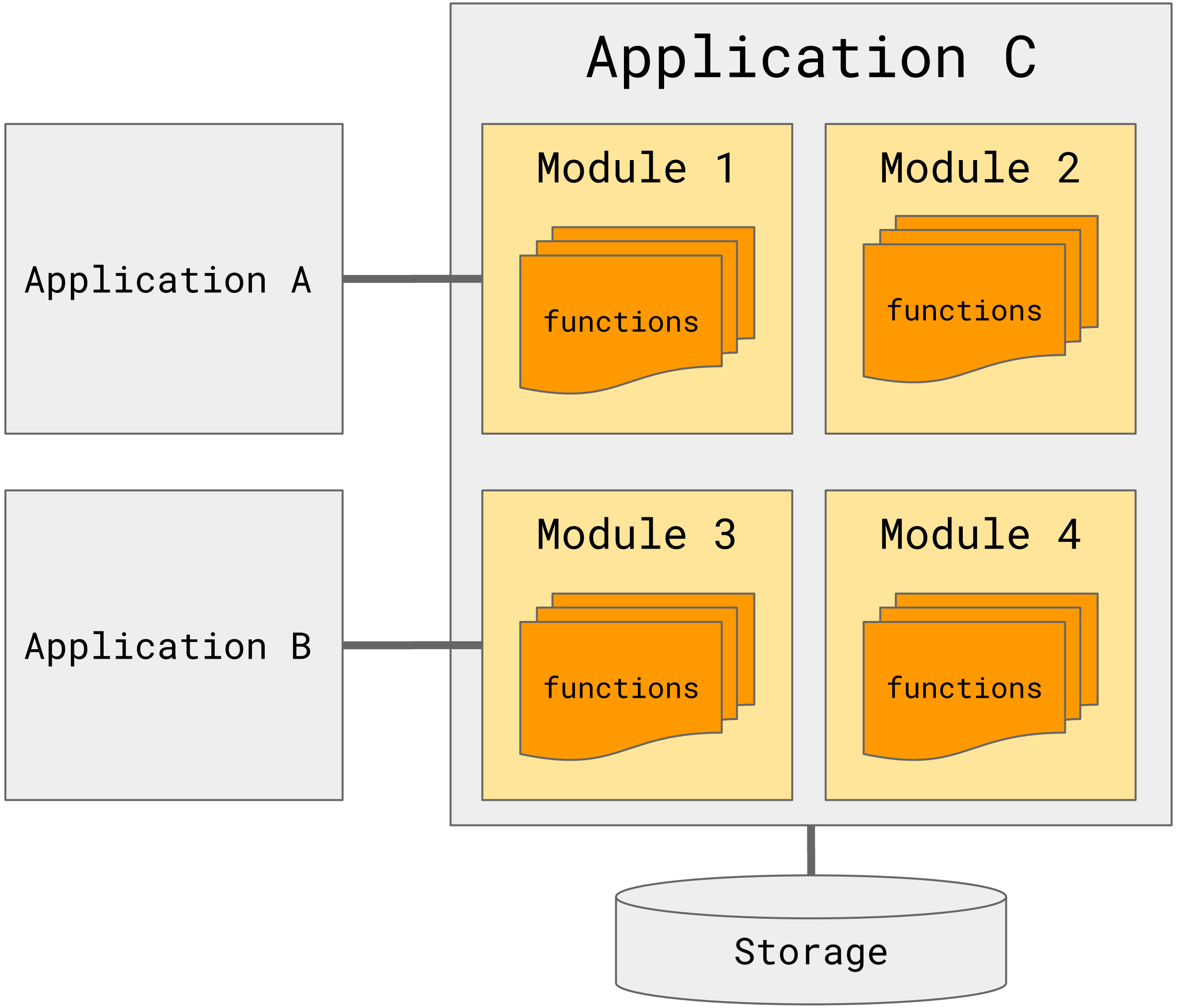
Integrations

Resources

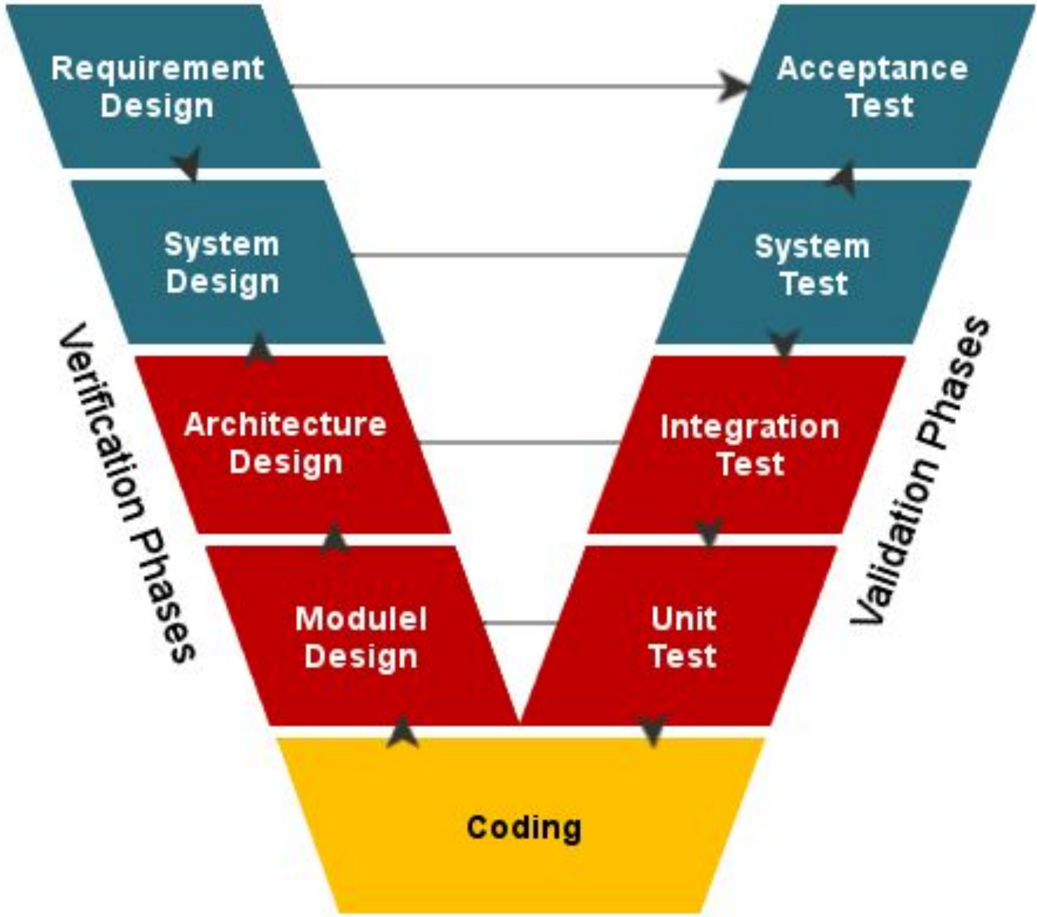


System

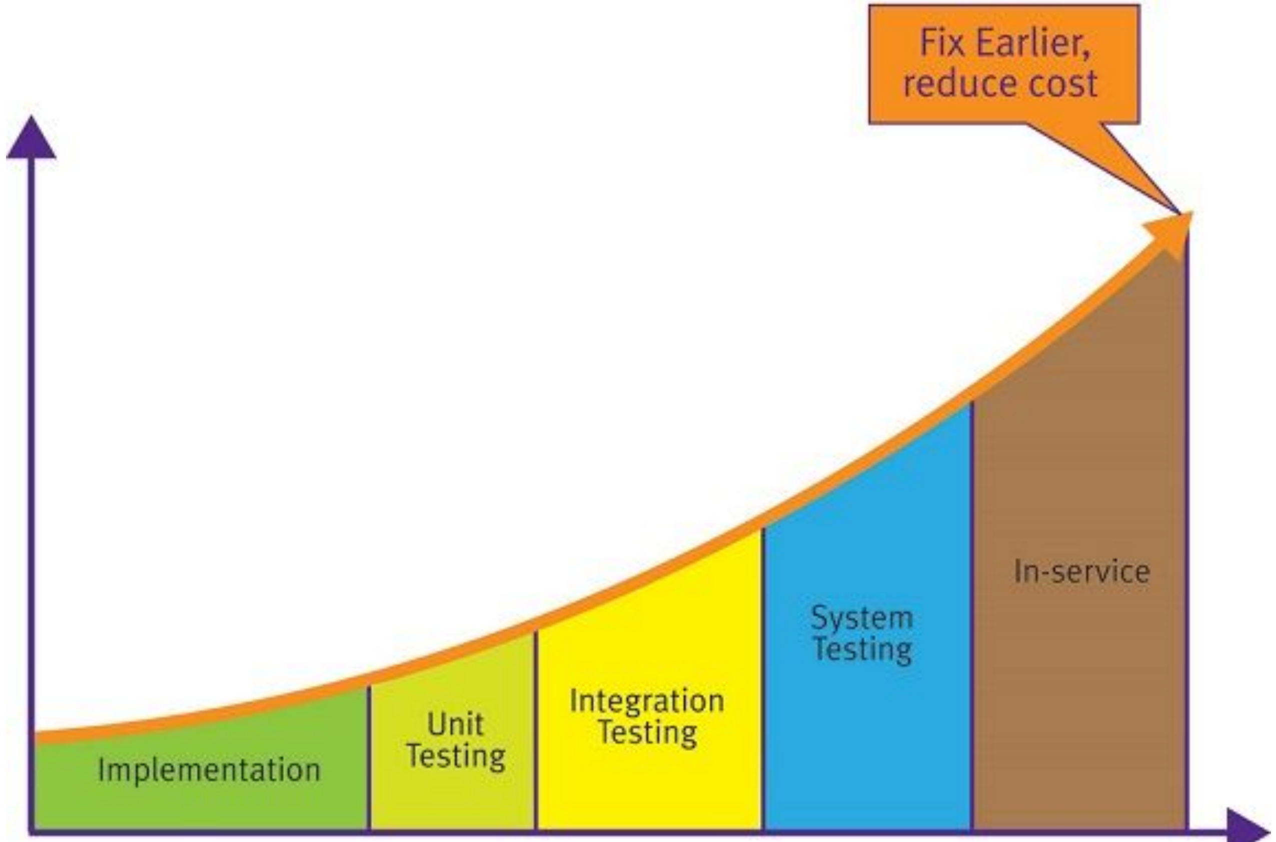
What should be tested?



Testing levels



Issues fixing cost



Unit test

Scope

Test a small piece of code (usually a function)

independently from other parts

Test steps

	BDD	AAA
Preconditions	Given	Arrange
Action	When	Act
Assertions	Then	Assert

Unit testing principles

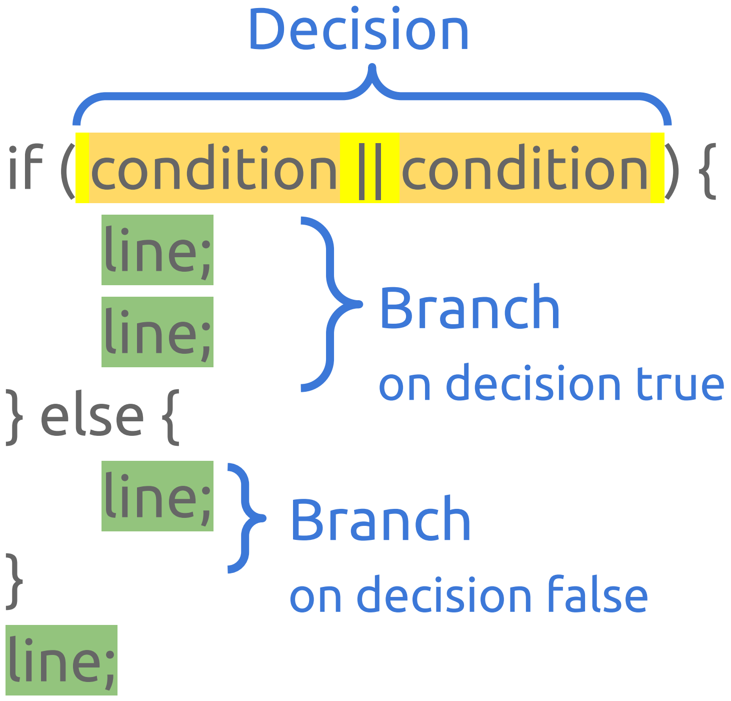
Self-validating

No manual inspection required to check whether the test has passed or failed.

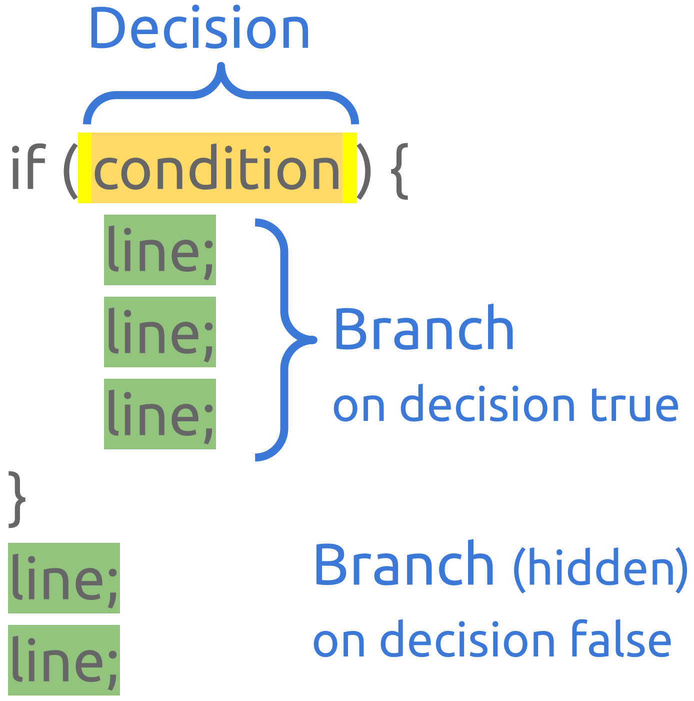
Thorough

Should cover every scenarios

Test coverage



```
if (condition || condition) {
    line;
    line;
} else {
    line;
}
line;
```



```
if (condition) {
    line;
    line;
    line;
}
line;
line;
```

Repeatable

should NOT depend on any data in the environment/instance

Deterministic results

Each test should setup or arrange it's own data.

Isolated /Independent

No order-of-run dependency.

F.I.R.S.T

Fast

Isolated/Independent

Repeatable

Self-validating

Thorough

Unit testing challenges

Dependencies	Depend on interfaces
Long complex methods	SRP, refactoring
`new` keyword	Dependency Injection
100% Coverage	

Not to test

"Don't test already tested things"

Trivial code

- Getters & Setters

Database

Frameworks/libraries (well-tested)

Project structure

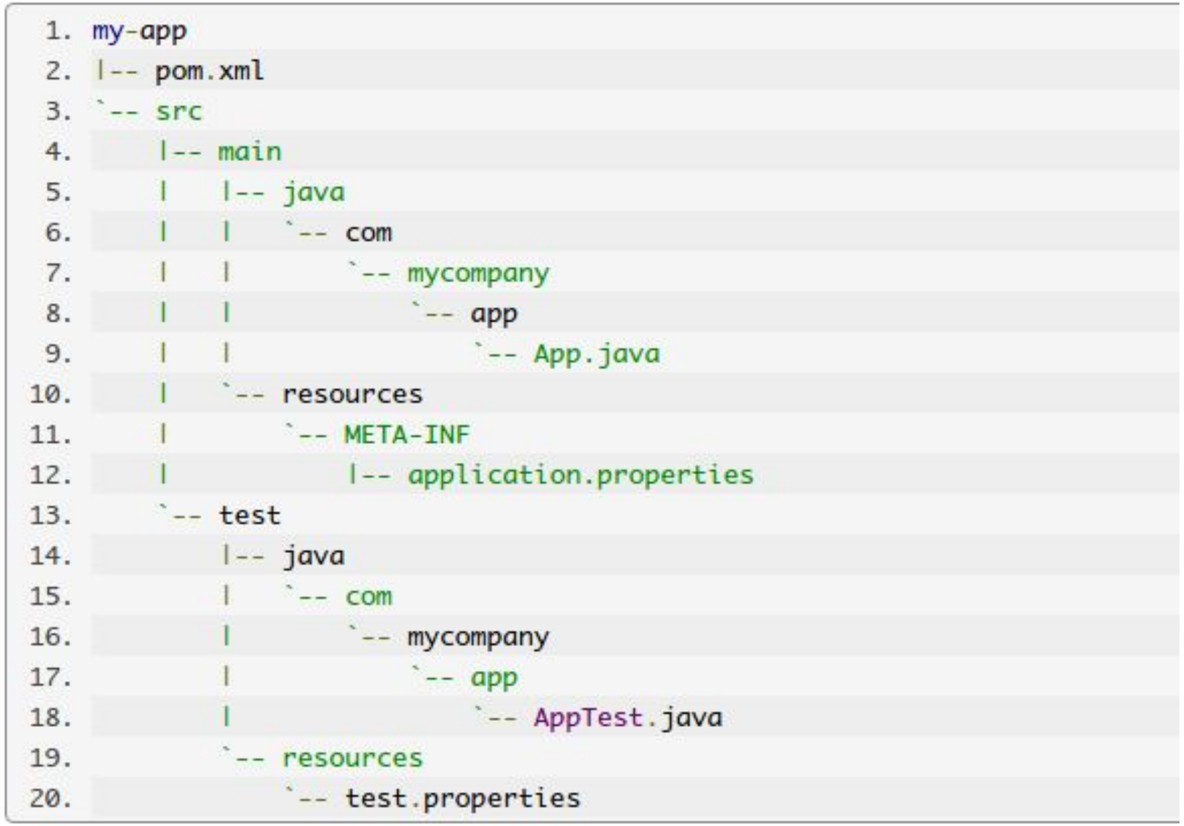
Maven

What's Maven™ ?

Uniform build system

- Simple project setup
- Dependency management
- Extensible (plugins)
- Easy to work with multiple projects/modules
- Start with ready made templates (archtypes)
- Maintain project quality (run tests)
- Supports multiple profiles

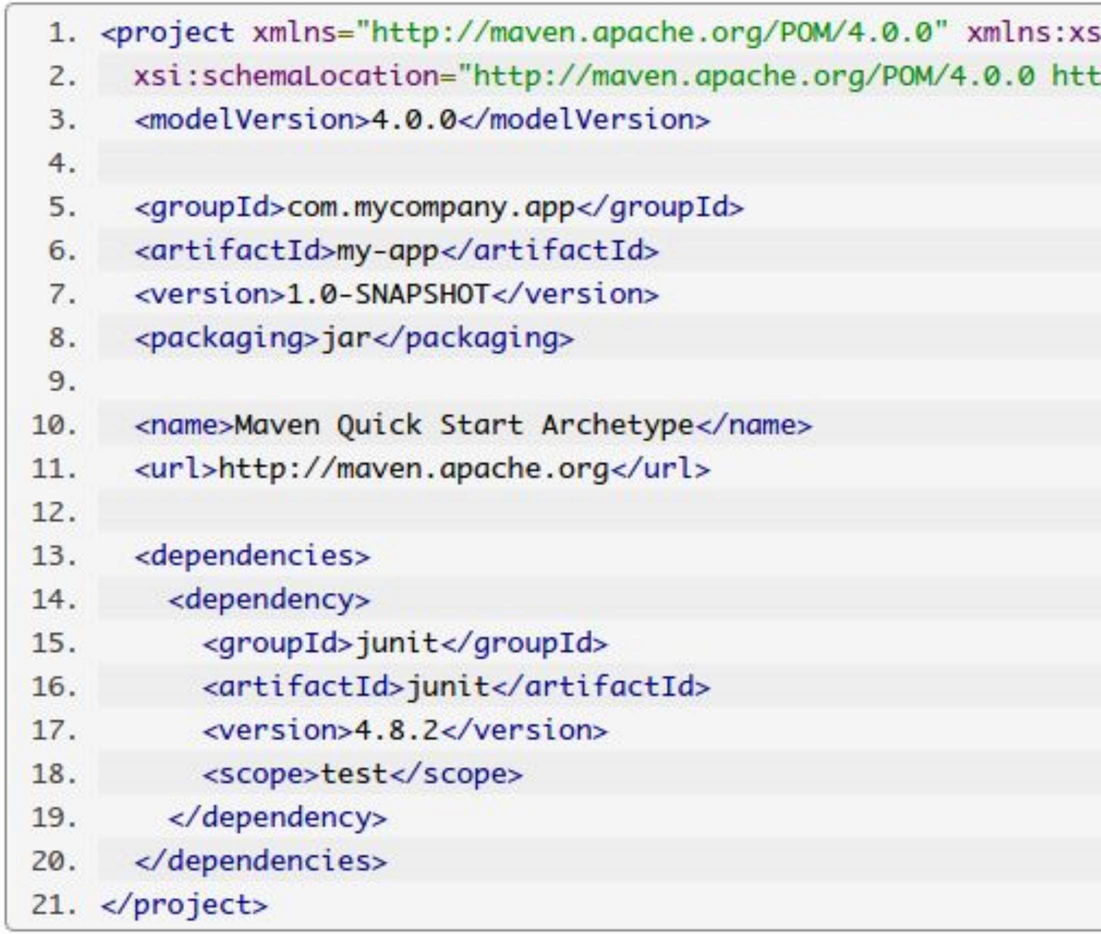
Maven Directory structure



```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- com
    |   |       `-- mycompany
    |   |           `-- app
    |   |               `-- App.java
    |   `-- resources
    |       `-- META-INF
    |           |-- application.properties
    `-- test
        |-- java
        |   `-- com
        |       `-- mycompany
        |           `-- app
        |               `-- AppTest.java
        `-- resources
            `-- test.properties
```

Maven POM

- artifact
- packaging
  - jar
  - war
  - pom
- version
  - SNAPSHOT
  - RELEASE
- dependencies

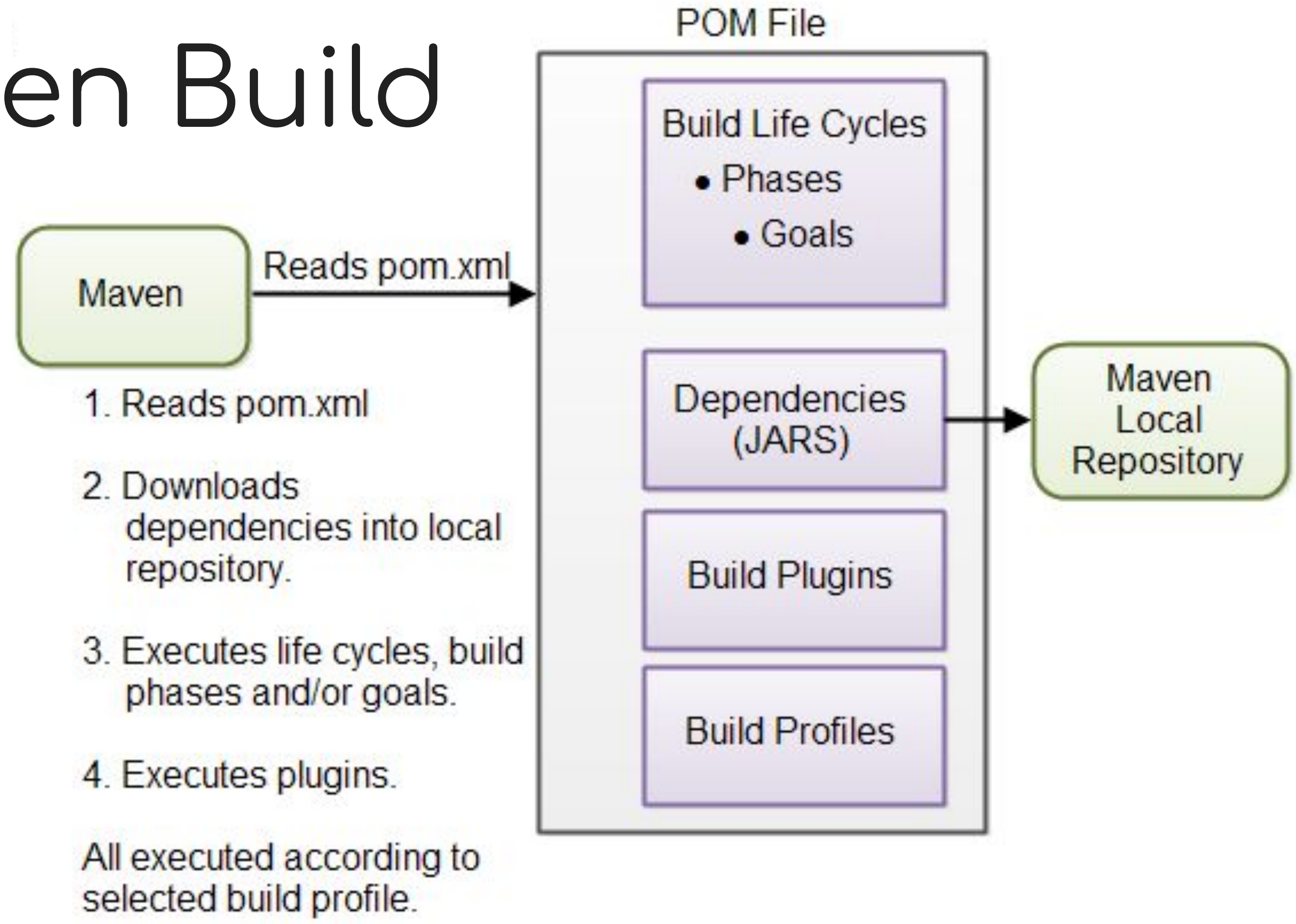


```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xs
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 htt
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Maven Dependencies

- Repository
  - Local [cache] *(~/.m2/repository)*
  - Remote
    - Internal (Hosted on company)
    - Remote (Hosted globally)
      - Central *(http://repo.maven.apache.org/maven2/)*
- Scope
  - compile
  - test
  - provided

Maven Build



1. Reads pom.xml

2. Downloads dependencies into local repository.

3. Executes life cycles, build phases and/or goals.

4. Executes plugins.

All executed according to selected build profile.

Maven lifecycle

Validate	pom.xml
generate-resources	resources (src/main/resources)
process-resources	resources (src/main/resources)
compile	sources (src/main/java) → class
test-compile	test sources (src/test/java) → class
test	surefire plugin (Unit testing)
package	jar/war/...
verify	failsafe plugin (Integration testing)
install	→ Local repository
deploy	→ Remote repository

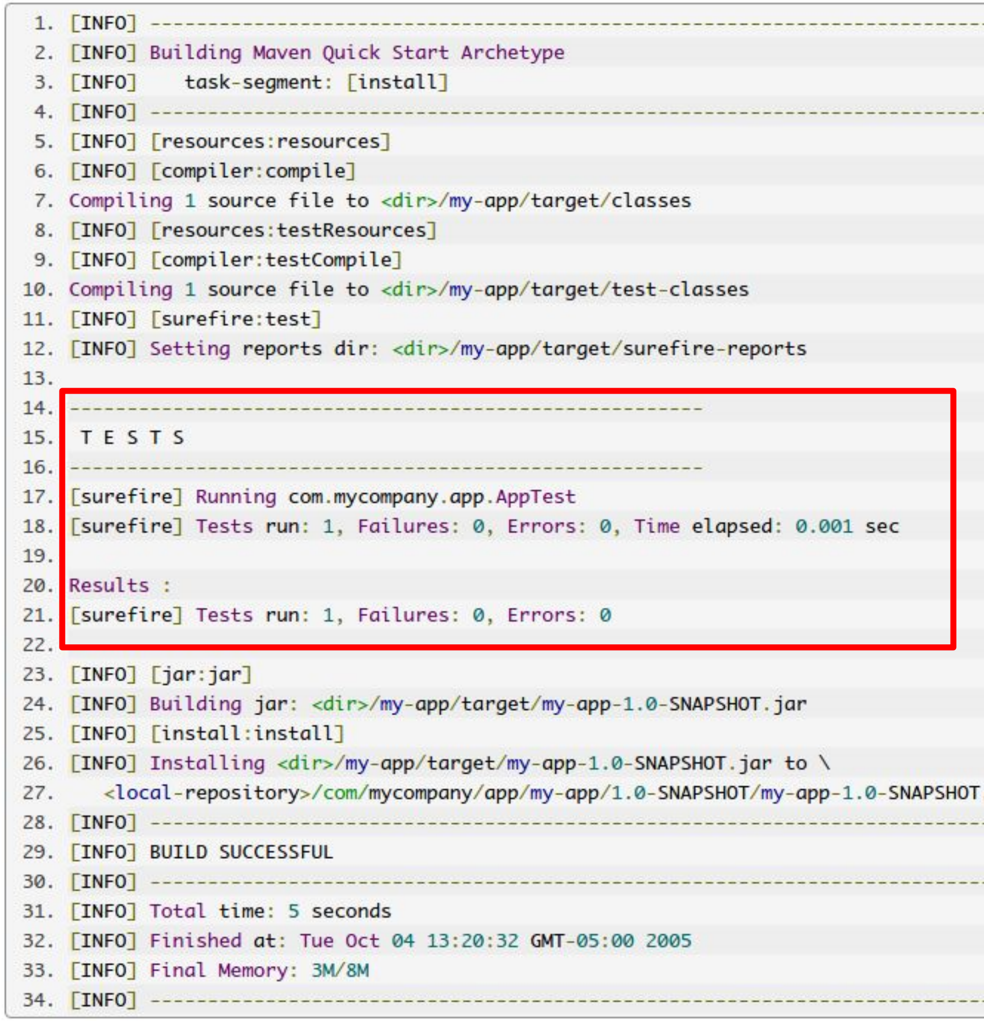
Mvn install

Applying Unit testing on

**/*Test.java

**/Test*.java

**/*TestCase.java



```
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] [compiler:compile]
Compiling 1 source file to <dir>/my-app/target/classes
[INFO] [resources:testResources]
[INFO] [compiler:testCompile]
Compiling 1 source file to <dir>/my-app/target/test-classes
[INFO] [surefire:test]
[INFO] Setting reports dir: <dir>/my-app/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
[surefire] Running com.mycompany.app.AppTest
[surefire] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.001 sec

Results :
[surefire] Tests run: 1, Failures: 0, Errors: 0

[INFO] [jar:jar]
[INFO] Building jar: <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar
[INFO] [install:install]
[INFO] Installing <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar to \
   <local-repository>/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5 seconds
[INFO] Finished at: Tue Oct 04 13:20:32 GMT-05:00 2005
[INFO] Final Memory: 3M/8M
[INFO] ------------------------------------------------------------------------
```

Test runner

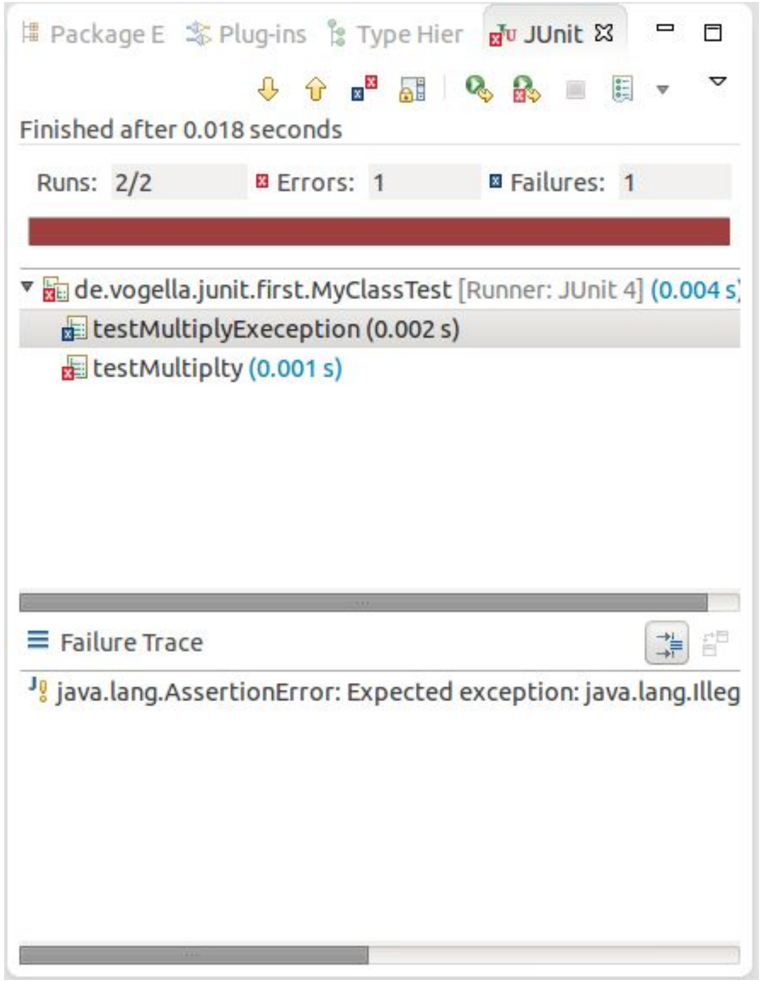
JUnit

Unit testing

Unit testing scope

Test cases

- Execution order
- Predictability



```
Finished after 0.018 seconds

Runs: 2/2    Errors: 1    Failures: 1

▾ de.vogella.junit.first.MyClassTest [Runner: JUnit 4] (0.004 s)
    testMultiplyExeception (0.002 s)
    testMultiplty (0.001 s)

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

JUnit

Test method

    @org.junit.jupiter.api.Test

Test class

    *Test

Test Suite

Test lifecycle

`@BeforeAll`	`@BeforeClass`
`@BeforeEach`	`@Before`
`@Test`	
`@AfterEach`	`@After`
`@AfterAll`	`@AfterClass`

Naming convention

- Feature
  - `registerAddsUser`
  - `throwsExceptionWhenRegisterUser/IfMailIsInvalid`
  - `failToWithdrawMoney/IfAccountIsInvalid`
- Should
  - `userShouldBeCreated`
  - `userShouldNotBeCreated/IfMailIsInvalid`
  - `moneyShouldNotBeWithdrawn/IfAccountIsInvalid`
- BDD
  - `givenValidUser_whenRegister_thenSucceed`
  - `givenUserWithInvalidMail_whenRegister_thenThrowException`
  - `givenAnInvalidAccount_whenWithdraw_thenFail`

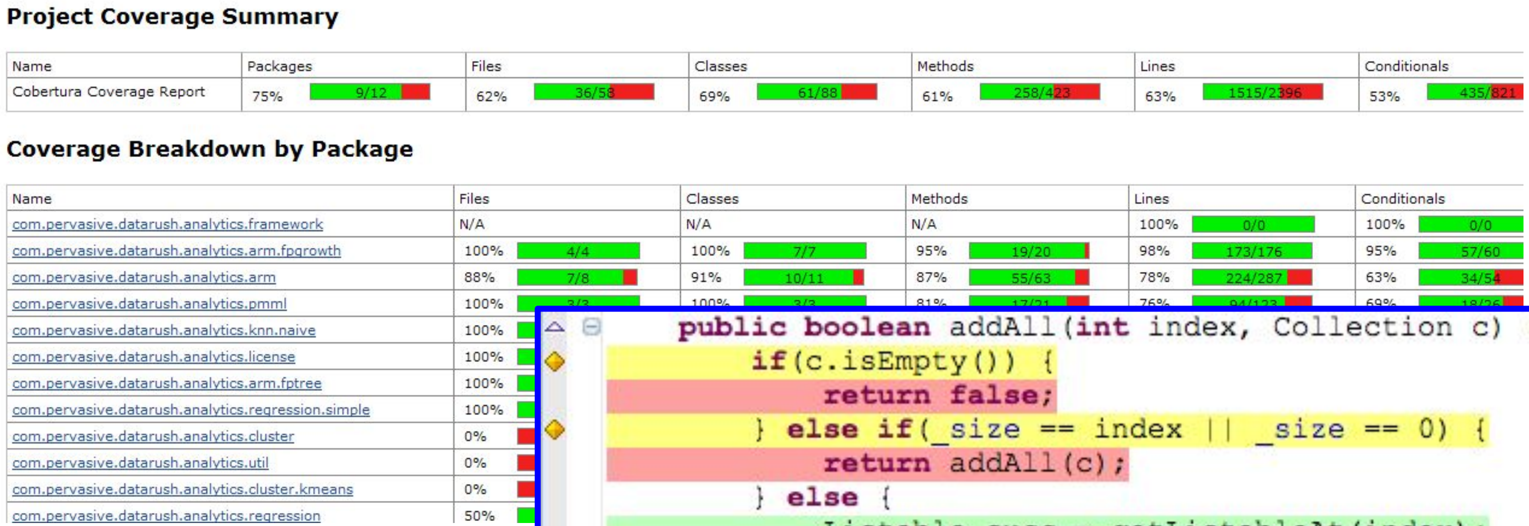
JUnit assertions

```
fail(message)

assertTrue/assertFalse([message,] boolean condition)

assertEquals([message,] expected, actual)

assertNull/assertNotNull([message,] object)

assertSame/assertNotSame([message,] expected, actual)
```

Code coverage

JaCoCo

Code coverage

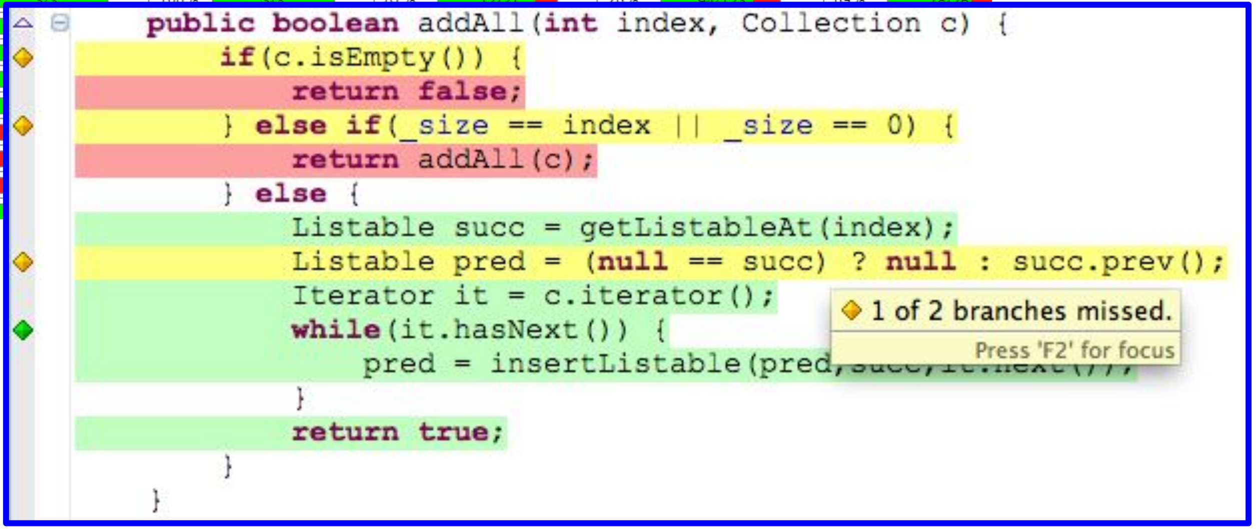


Project Coverage Summary

Name	Packages		Files		Classes		Methods		Lines		Conditionals	
Cobertura Coverage Report	75%	9/12	62%	36/58	69%	61/88	61%	258/423	63%	1515/2396	53%	435/821

Coverage Breakdown by Package

Name	Files		Classes		Methods		Lines		Conditionals	
com.pervasive.datarush.analytics.framework	N/A		N/A		N/A		100%	0/0	100%	0/0
com.pervasive.datarush.analytics.arm.fpgrowth	100%	4/4	100%	7/7	95%	19/20	98%	173/176	95%	57/60
com.pervasive.datarush.analytics.arm	88%	7/8	91%	10/11	87%	55/63	78%	224/287	63%	34/54
com.pervasive.datarush.analytics.pmml	100%	3/3	100%	3/3	81%	17/21	76%	94/123	69%	18/26
com.pervasive.datarush.analytics.knn.naive	100%									
com.pervasive.datarush.analytics.license	100%									
com.pervasive.datarush.analytics.arm.fptree	100%									
com.pervasive.datarush.analytics.regression.simple	100%									
com.pervasive.datarush.analytics.cluster	0%									
com.pervasive.datarush.analytics.util	0%									
com.pervasive.datarush.analytics.cluster.kmeans	0%									
com.pervasive.datarush.analytics.regression	50%									



```
    public boolean addAll(int index, Collection c) {
        if(c.isEmpty()) {
            return false;
        } else if(_size == index || _size == 0) {
            return addAll(c);
        } else {
            Listable succ = getListableAt(index);
            Listable pred = (null == succ) ? null : succ.prev();
            Iterator it = c.iterator();
            while(it.hasNext()) {
                pred = insertListable(pred,succ,it.next());
            }
            return true;
        }
    }
```

1 of 2 branches missed.
Press 'F2' for focus

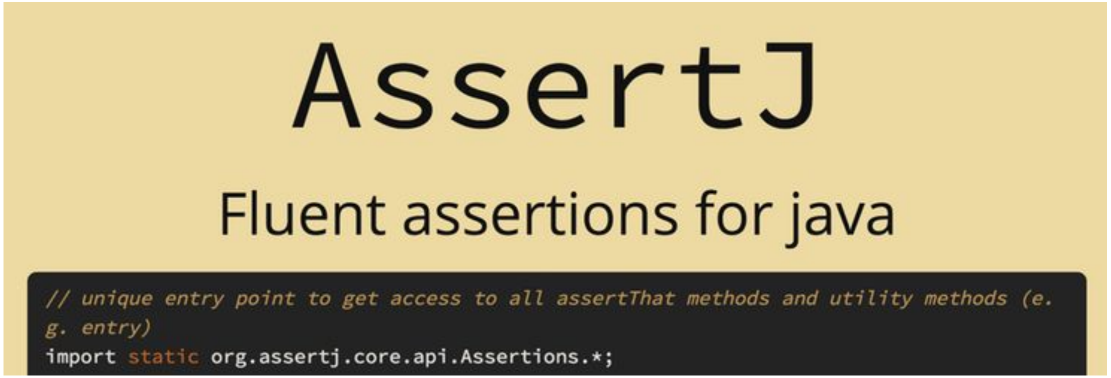
Fluent Assertions

AssertJ

AssertJ

A Java library that provides a fluent interface for writing assertions.

Goal: improve test code Readability & Maintainability.



```
// unique entry point to get access to all assertThat methods and utility methods (e.
g. entry)
import static org.assertj.core.api.Assertions.*;
```

First look

```
assertThat(frodo.getAge()).isEqualTo(100);


assertThat(frodo.getAge()).as("check %s's age",
frodo.getName()).isEqualTo(100);


// Assertion error
[check Frodo's age] expected:<33> but was:<100>
```

basic assertions

```
assertThat(frodo.getName()).isEqualTo("Frodo");

assertThat(frodo).isNotEqualTo(sauron)
                 .isIn(fellowshipOfTheRing);


assertThat(frodo.getName()).startsWith("Fro")
                           .endsWith("do")
                           .isEqualToIgnoringCase("frodo");
```

Collection specific assertions

```
assertThat(fellowshipOfTheRing)

   .hasSize(9)

   .contains(frodo, sam)

   .doesNotContain(sauron);
```

Exception Assertion

```
// Java 8 exception assertion
assertThatThrownBy(() -> { throw new Exception("boom!");
}).isInstanceOf(Exception.class)
 .hasMessageContaining("boom");

// Java 8 BDD style exception assertion
Throwable thrown = catchThrowable(() -> { throw new Exception("boom!");
});
assertThat(thrown).isInstanceOf(Exception.class)
                  .hasMessageContaining("boom");
```

Extracting

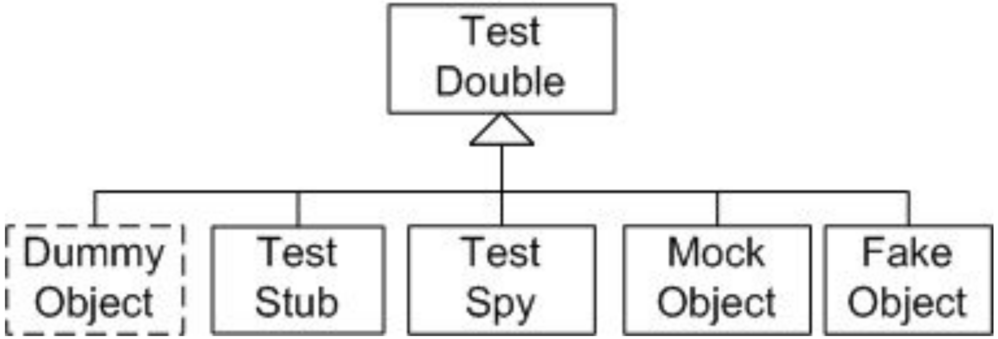
```
assertThat(fellowshipOfTheRing).extracting("name")
                               .contains("Boromir", "Gandalf", "Frodo")
                               .doesNotContain("Sauron", "Elrond");

// Java 8 (type safe)
assertThat(fellowshipOfTheRing).extracting(TolkienCharacter::getName)
                               .contains("Boromir", "Gandalf", "Frodo")
                               .doesNotContain("Sauron", "Elrond");

// multiple values at once (using a tuple)
assertThat(fellowshipOfTheRing).extracting("name", "age", "race.name")
                               .contains(tuple("Boromir", 37, "Man"),
                                         tuple("Sam", 38, "Hobbit"),
                                         tuple("Legolas", 1000, "Elf"));
```

Mocking

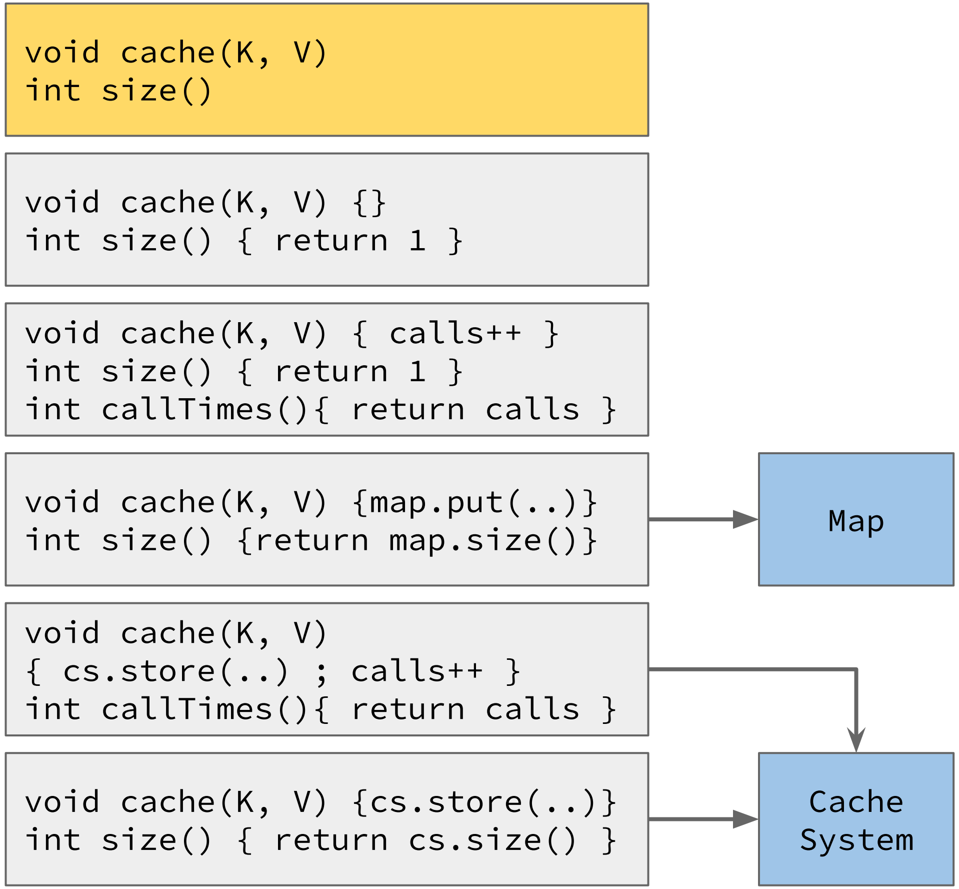
Mockito

Test Doubles

Double is a generic term for any case where you replace a production object for testing purposes



Ex.



Interface

```
void cache(K, V)
int size()
```

Stub

```
void cache(K, V) {}
int size() { return 1 }
```

Mock

```
void cache(K, V) { calls++ }
int size() { return 1 }
int callTimes(){ return calls }
```

Fake

```
void cache(K, V) {map.put(..)}
int size() {return map.size()}
```

Spy

```
void cache(K, V)
{ cs.store(..) ; calls++ }
int callTimes(){ return calls }
```

Real

```
void cache(K, V) {cs.store(..)}
int size() { return cs.size() }
```

System layers/modules

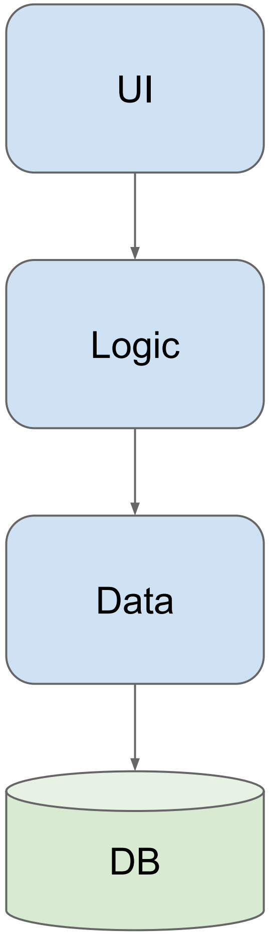
// Testing UI, Logic, ...

Every layer should mock layer below → **Mock**

// Testing Data Access Layer

Mocking DB is a hard task

We can depend on a light DB (in-memory) → **Fake**



Mockito

A framework that enables you to easily generate test doubles for testing purpose

It allows you to verify execution and write logic inside your mocked objects

First look

```
// mock creation
List list = mock(List.class);

// using mock object
list.add("one");
list.clear();

// selective, explicit, highly readable verification
verify(list).add("one");
verify(list).clear();
```

Stub method calls

```
// you can mock concrete classes, not only interfaces
LinkedList list = mock(LinkedList.class);

// stubbing appears before the actual execution
when(list.get(0)).thenReturn("first");

// prints "first"
System.out.println(list.get(0));

// prints "null" because get(999) was not stubbed
System.out.println(list.get(999));
```

Stubbing - throw exceptions

```
LinkedList list = mock(LinkedList.class);

//stubbing
when(list.get(1)).thenThrow(new RuntimeException());

//or

doThrow(new RuntimeException()).when(list).get(1);

//following throws runtime exception
System.out.println(list.get(1));
```

Verify

```
//using mocks - only mockOne is interacted
mockOne.add("one");

//ordinary verification
verify(mockOne).add("one");

//verify that method was never called on a mock
verify(mockOne, never()).add("two");

//verify that other mocks were not interacted
verifyZeroInteractions(mockTwo, mockThree);
```

Callbacks

```
when(mock.someMethod(anyString())).thenAnswer(new Answer() {
     Object answer(InvocationOnMock invocation) {
         Object[] args = invocation.getArguments();
         Object mock = invocation.getMock();
         return "called with arguments: " + args;
     }
 });

 //the following prints "called with arguments: foo"
 System.out.println(mock.someMethod("foo"));
```