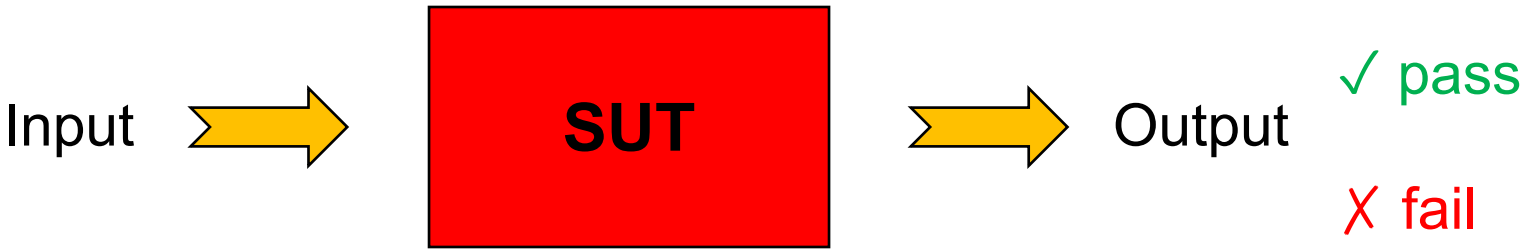
Unit Testing Lab

Mariano Ceccato

mariano.ceccato@univr.it

Testing



Test phases

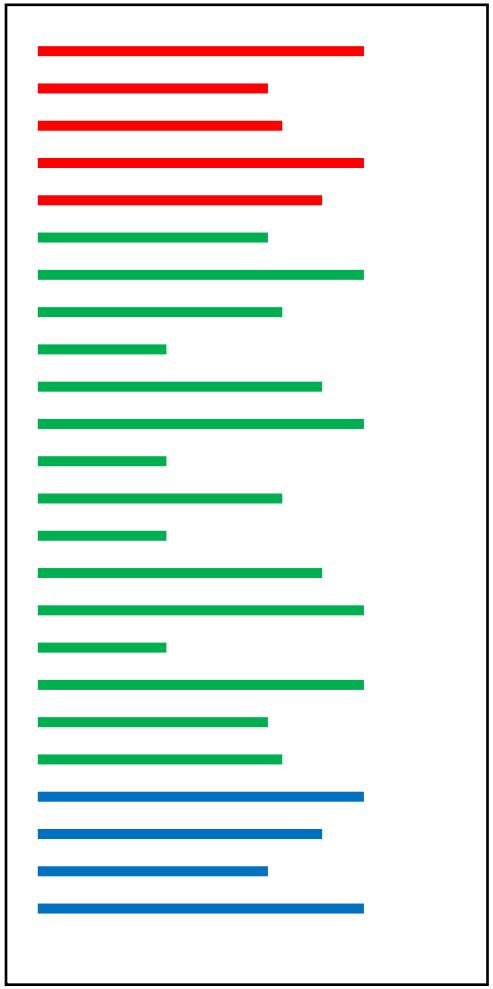
Code testing generally falls into three distinct phases:

- **Unit testing** – this is basically testing of a single function, procedure, class.
- **Integration testing** – this checks that units tested in isolation work properly when put together.
- **System testing** – here the emphasis is to ensure that the whole system can cope with real data, monitor system performance, test the system's error handling and recovery routines.

Testing with Junit

- Junit is a unit test environment for Java programs developed by Erich Gamma and Kent Beck.
  - Writing test cases
  - Executing test cases
  - Pass/fail? (expected result = obtained result?)
- Consists in a framework providing all the tools for testing.
  - Test engine: set of classes and conventions to use them.
  - Launcher: launch the platform from the command line
- Integrated in all the major IDE
  IntelliJ IDEA, Eclipse, NetBeans, Visual Studio Code
- Integrate in build tools
  - Gradle, Maven, Ant

Automation of Unit testing



1. A setup part
   - The system is initialized and brought in a testable state
2. A call part,
   - Functionality to be tested are exercised
3. An assertion part
   - Actual result are compared with expected result
   - The test passes/failts

Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention!

Annotations in Java

- J2SE 5 introduced the Metadata feature (data about data)
- Annotations allow you to add decorations to your code (remember javadoc tags: @author)
- Annotations are used for code documentation, compiler processing (@Deprecated), code generation, runtime processing
- New annotations can be created by developers

Annotations in Java … an example

- `@Override` is a predefined annotation used by the Java compiler
- It informs the compiler that the element (a method) is meant to override an element declared in a superclass

```
// mark method as a superclass method
// that has been overridden

@Override
public int overriddenMethod() {
    …
}
```

- While it is not required to use this annotation when overriding a method, it helps to prevent errors.
- If a method marked with `@Override` fails in correctly overriding the original method in its superclass, the compiler emits an error.

Test Class in Junit 4

```
import org.junit.Test;

public class StackTests {

  @Test
  public void testStack() {
     Stack aStack = new Stack();
     // Stack should be empty
     aStack.push(10);
     aStack.push(-4);
     // Last element should be -4
     // First element should be 10
     }
}
```

- Annotation to mark test method(s)
- Test class
  - Public visibility
  - At least a public constructor
- Test method
  - `@Test`
  - Public visibility
  - No formal parameter
  - Void return type

Assertions

- Method family to check conditions
  - whether the actual value corresponds to the expected value
- Their names begin with “assert” and are used in test methods

```
assertTrue("stack should be empty", aStack.empty());
```

- If the condition is true:
  - execution continues normally
- If the condition is false:
  - test fails
  - execution skips the rest of the test method
  - the message (if any) is printed

Assertions

- for a boolean condition
  - `assertTrue("message for fail", condition);`
- for object, int, long, and byte values
  - `assertEquals(expected_value, expression);`
- for float and double values
  - `assertEquals(expected, expression, error);`
- for objects references
  - `assertNull(reference)`
  - `assertNotNull(reference)`
- …

Test Class

Come creare le asserzioni per questo caso.

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

public class StackTestes {

  @Test
  public void testStack() {
    Stack aStack = new Stack();
    // Stack should be empty
    assertTrue("Stack should be empty!", aStack.isEmpty());
    aStack.push(10);
    assertTrue("Stack should not be empty!",!aStack.isEmpty());
    aStack.push(-4);
    // Last element should be -4
    assertEquals(-4, aStack.pop());
    // First element should be 10
    assertEquals(10, aStack.pop());
    }
}
```

Gradle integration

Tramite gradle utilizzo JUnit

```
plugins {
    id 'java'
}

group 'it.univr'
version '1.0-SNAPSHOT'

repositories {
    jcenter()
}

dependencies {
    testImplementation 'junit:junit:4.12'
}
```

definisco una dipendenza per fare usare a Gradle Junit

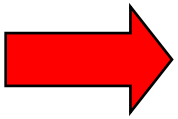
Il sistema utilizza Junit solo nella fase di test

Junit

- Test framework
  - test cases are Java code
  - test case = “sequence of operations +inputs + expected values”

- Production code

```
int doubleOf( ) {
    …
}
```



- Test code

```
testDobleOf( ) {
    …
}
```

Separate methods

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

public class StackTester {

  @Test
  public void testStackEmpty() {
    Stack aStack = new Stack();
    assertTrue("Stack should be empty!", aStack.isEmpty());
    aStack.push(10);
    assertTrue("Stack should not be empty!", !aStack.isEmpty());
  }

  @Test
  public void testStackOperations() {
    Stack aStack = new Stack();
    aStack.push(10);
    aStack.push(-4);
    assertEquals(-4, aStack.pop());
    assertEquals(10, aStack.pop());
  }
}
```

Test suite

- Explicit aggregation of test cases
- When a test suite is run, all the test cases in the suite will be executed

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
  TestFeatureLogin.class,
  TestFeatureLogout.class,
  TestFeatureNavigate.class,
  TestFeatureUpdate.class
})

public class FeatureTestSuite {
  // the class remains empty,
  // used only as a holder for the above annotations
}
```

Test execution order

```
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class TestMethodOrder {

    @Test
    public void testA() {
        System.out.println("first");
    }
    @Test
    public void testB() {
        System.out.println("second");
    }
    @Test
    public void testC() {
        System.out.println("third");
    }
}
```

- Well-written test code should not assume any order of test case execution
  - Each test should bring the code in the *testable* state, and make no other assumption
- Sometime, however, a predictable failure is better than a random failure on certain platforms

Exception testing

- To test that code throws exceptions when supposed to:
  - E.g, `pop` an empty stack

```
public class ExceptionTest {

    @Test
    public void exteption1(){
        Stack stack = new Stack();
        try {
            stack.pop();
            fail();
        }
        catch (IndexOutOfBoundsException exception){
            assertTrue(true);
        }
    }
}
```

```
@Test(expected = IndexOutOfBoundsException.class)
public void exteption2(){
  Stack stack = new Stack();
  stack.pop();
}
```

Setting up & tearing down

- Known and fixed environment in which tests are run so that results are repeatable
- Examples:
  - Preparation of input data and setup/creation of objects
  - Loading a database with a specific, known set of data
  - Copying a specific known set of files initialized to certain states.

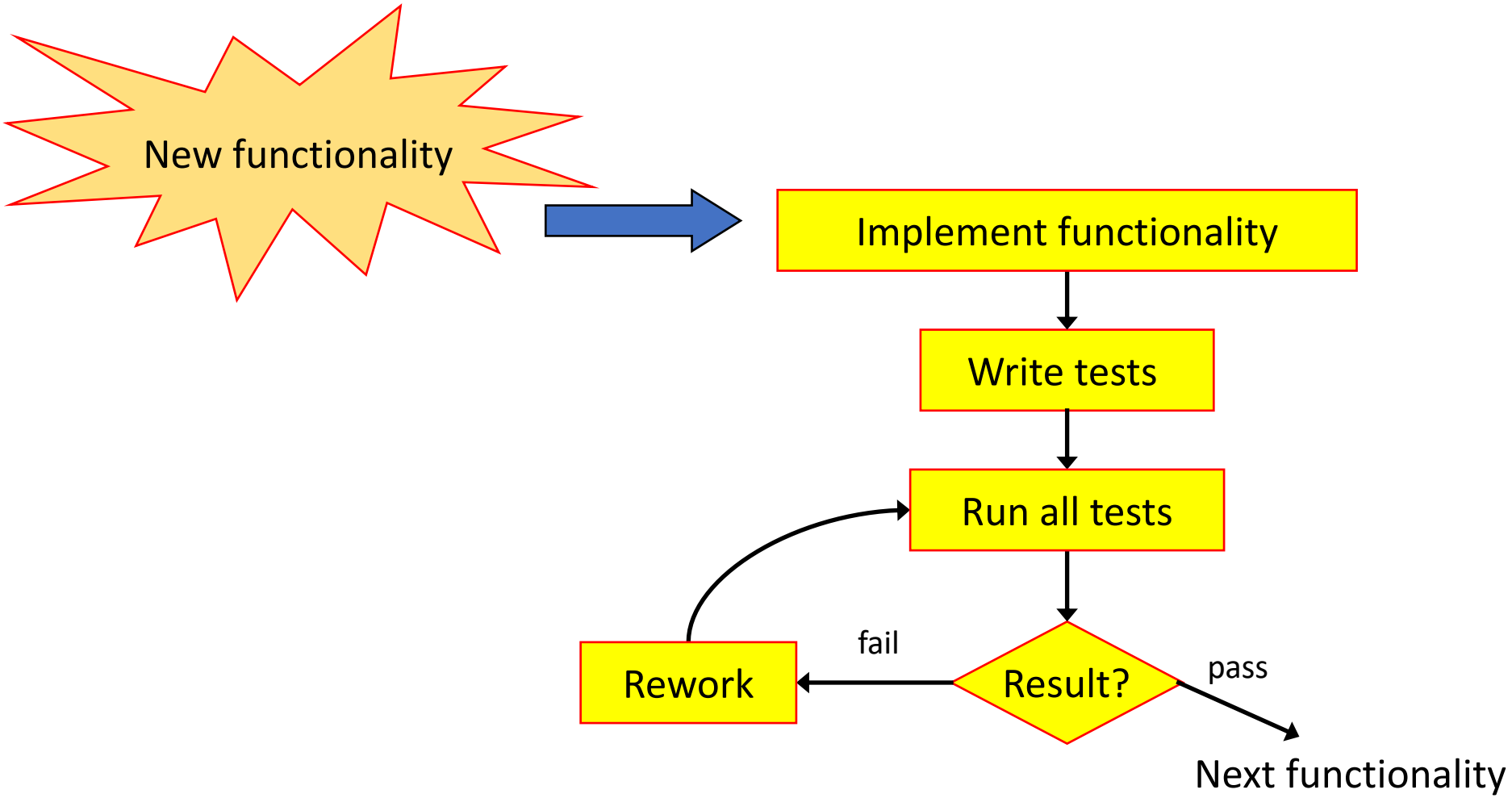
- Class level
  - `@BeforeClass` static method is run once, before running test methods
    - E.g., initialize object(s) under test
  - `@AfterClass` static method is run once, after having run test methods
    - E.g., release object(s) under test
- Method/test level
  - `@Before` method is run many times, once before each test method
  - `@After` method is run many times, once after each test method

Additional Features of @Test

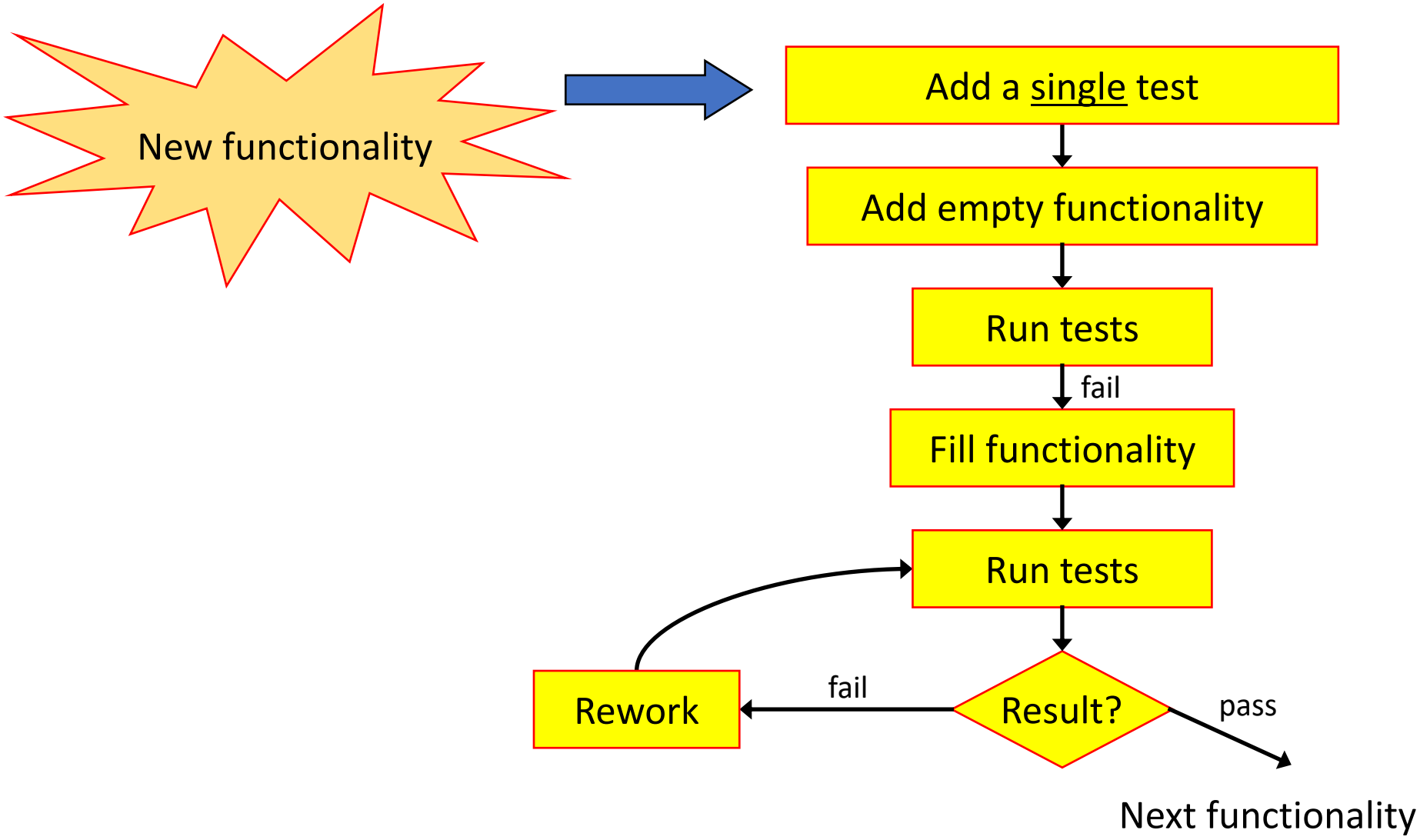
- To avoid infinite loops, an execution time limit can be used.
- The time limit is specified in milliseconds.
- The test fails if the method takes too long.

```
@Test(timeout=10)
public void greatBig() {
    assertTrue(program.compute(5, 5) > 10e12);
}
```

Test last


New functionality
Implement functionality
Write tests
Run all tests
Result?
fail
pass
Rework
Next functionality

Test first


New functionality
Add a single test
Add empty functionality
Run tests
fail
Fill functionality
Run tests
Result?
fail
Rework
pass
Next functionality

Bank account

- Status: a bank account supports two types of transactions, they are “deposit” and “withdraw”.
- **Task1: Implement a new feature to compute the balance (some of transactions)**
- Tests:
  - On a fresh bank account the balance is 0
  - Deposit 10 and check the balance
  - Deposit 10, withdraw 8 and check the balance
  - Deposit and withdraw 2024 times (for loop) and check the balance

```
git clone https://github.com/Fondamenti-di-ingegneria-del-software/TestLab
```

Bank account

- Status: a bank account supports two types of transactions, they are “deposit” and “withdraw”.
- Task1: Implement a new feature to compute the balance (some of transactions)
- Tests:
  - On a fresh bank account the balance is 0
  - Deposit 10 and check the balance
  - Deposit 10, withdraw 8 and check the balance
  - Deposit and withdraw 2024 times (for loop) and check the balance
- **Task2: change the transaction database from fixed length (array) to dynamic length (list)**

```
git clone https://github.com/Fondamenti-di-ingegneria-del-software/TestLab
```

Junit Javadoc

- https://junit.org/junit4/javadoc/latest/index.html

junit.org/junit4/javadoc/latest/index.html

Apps

Other Bookmarks

All Classes

Packages
org.hamcrest
org.hamcrest.core
org.junit
org.junit.experimental
org.junit.experimental.categories
org.junit.experimental.max
org.junit.experimental.results
org.junit.experimental.runners
org.junit.experimental.theories
org.junit.experimental.theories.supp
org.junit.function

org.junit

Classes
Assert
Assume
Test.None

Exceptions
AssumptionViolatedException
TestCouldNotBeSkippedException

Errors
ComparisonFailure

Annotation Types
After
AfterClass

Constructor Summary

protected	**Assert**() Protect constructor since it is a static only class

Method Summary

static void	**assertArrayEquals**(boolean[] expecteds, boolean[] actuals) Asserts that two boolean arrays are equal.
static void	**assertArrayEquals**(byte[] expecteds, byte[] actuals) Asserts that two byte arrays are equal.
static void	**assertArrayEquals**(char[] expecteds, char[] actuals) Asserts that two char arrays are equal.
static void	**assertArrayEquals**(double[] expecteds, double[] actuals, double delta) Asserts that two double arrays are equal.
static void	**assertArrayEquals**(float[] expecteds, float[] actuals, float delta) Asserts that two float arrays are equal.
static void	**assertArrayEquals**(int[] expecteds, int[] actuals) Asserts that two int arrays are equal.
static void	**assertArrayEquals**(long[] expecteds, long[] actuals) Asserts that two long arrays are equal.
static void	**assertArrayEquals**(Object[] expecteds, Object[] actuals) Asserts that two object arrays are equal.
static void	**assertArrayEquals**(short[] expecteds, short[] actuals) Asserts that two short arrays are equal.
static void	**assertArrayEquals**(String message, boolean[] expecteds, boolean[] actuals) Asserts that two boolean arrays are equal.
static void	**assertArrayEquals**(String message, byte[] expecteds, byte[] actuals) s that two byte arrays are equal.

https://junit.org/junit4/javadoc/latest/org/junit/Assert.html#assertArrayEquals(int[], int[])

Coverage

- Coverage measures describe the degree to which a program has been tested
  - tell the effectiveness in terms of coverage of the test set, help improve software quality
  - inform quantitatively the project manager about the progress of testing
- Many type of coverage measures
  - statements
  - paths
  - methods, classes
  - requirement specifications, etc.

Coverage tool features

- Compile time: automatic instrument code for coverage recording
- Runtime: structure-based coverage recording
  - lines, blocks, conditions, methods and classes
- Reporting
  - quantitative coverage measure
    - percentage of code executed
    - most executed VS never executed code
  - visual navigation
    - quickly navigate to code that is not executed to improve the test set

JaCoCo

- Edit build.gradle to integrate JaCoCo in your project

```
plugins {
    id 'java'
    id 'jacoco'
}
```

```
test {
    finalizedBy jacocoTestReport
}

jacocoTestReport {
    dependsOn test
}
```

Report is always generated after tests run

tests are required to run before generating the report

- Run Tests
- Generate JaCoCo test coverge report
  - `gradle jacocoTestReport`
- The report is available in folder: build/reports/jacoco/test/html/

Coverage Report

TestLab — Sessions

TestLab

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
it.univr		95%		100%	1	11	2	23	1	9	0	2
Total	4 of 90	95%	0 of 4	100%	1	11	2	23	1	9	0	2

TestLab > it.univr — Source Files — Sessions

it.univr

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BankAccount		91%		100%	1	6	2	13	1	5	0	1
Stack		100%		100%	0	5	0	10	0	4	0	1
Total	4 of 90	95%	0 of 4	100%	1	11	2	23	1	9	0	2

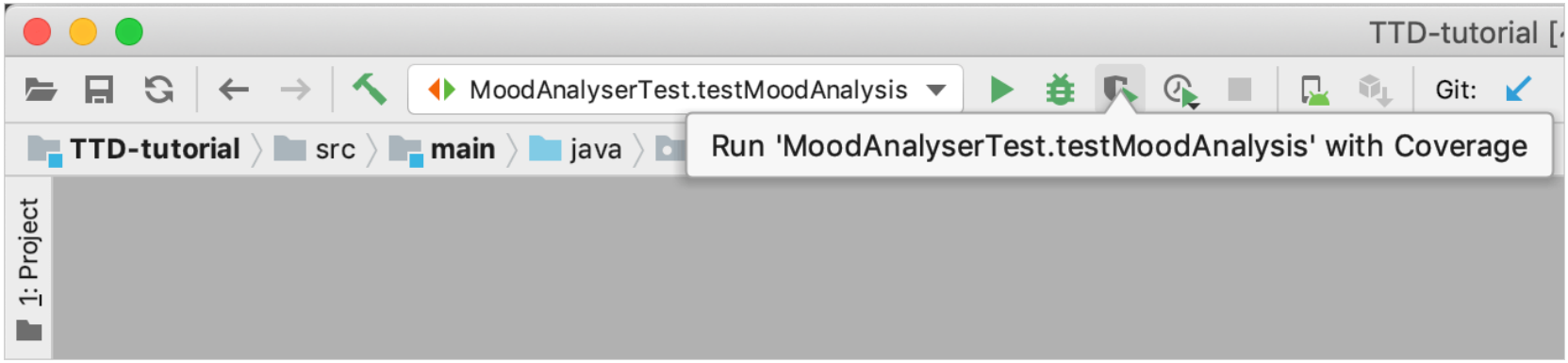
TestLab > it.univr > BankAccount

BankAccount

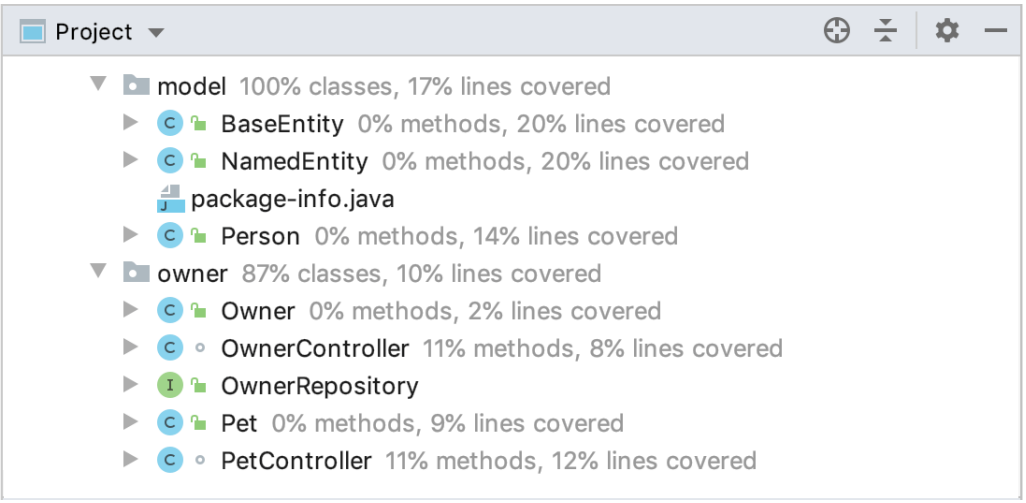
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
close()		0%		n/a	1	1	2	2	1	1
balance()		100%		100%	0	2	0	4	0	1
BankAccount()		100%		n/a	0	1	0	3	0	1
withdraw(int)		100%		n/a	0	1	0	2	0	1
deposit(int)		100%		n/a	0	1	0	2	0	1
Total	4 of 48	91%	0 of 2	100%	1	6	2	13	1	5

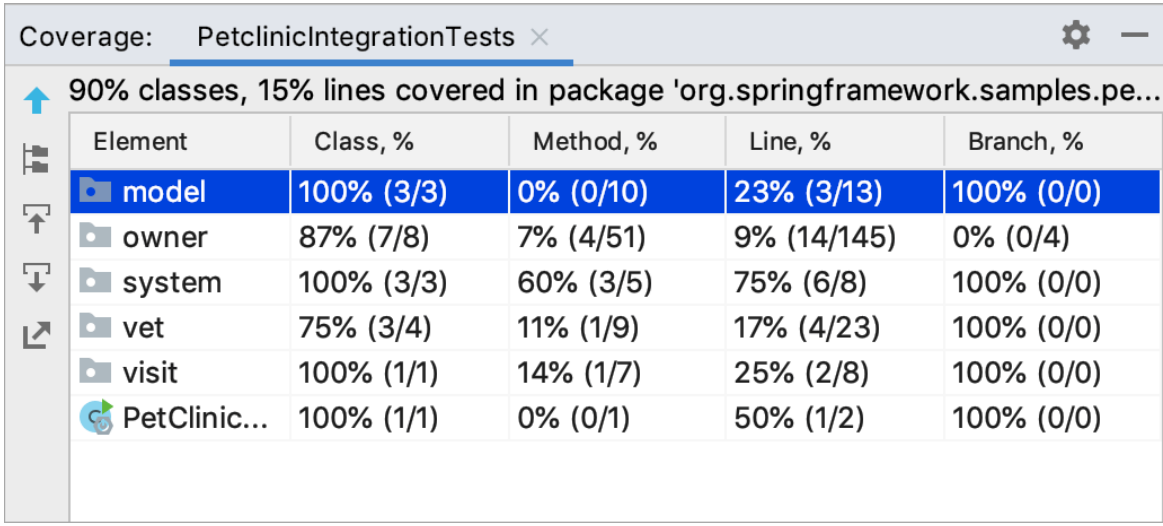
Created with JaCoCo 0.

IntelliJ Coverage



Coverage result





Element	Class, %	Method, %	Line, %	Branch, %
model	100% (3/3)	0% (0/10)	23% (3/13)	100% (0/0)
owner	87% (7/8)	7% (4/51)	9% (14/145)	0% (0/4)
system	100% (3/3)	60% (3/5)	75% (6/8)	100% (0/0)
vet	75% (3/4)	11% (1/9)	17% (4/23)	100% (0/0)
visit	100% (1/1)	14% (1/7)	25% (2/8)	100% (0/0)
PetClinic...	100% (1/1)	0% (0/1)	50% (1/2)	100% (0/0)

Coverage in the editor

BankAccount.java

di ingegneria del software/code-done/TestLab

```
2
3   import java.util.LinkedList;
4   import java.util.List;
5
6   public class BankAccount {
7
8       List<Integer> transactions;
9
10
11      public BankAccount() { transactions = new LinkedList<>(); }
14
15      public void deposit (int value) { transactions.add(value); }
18
19      public void withdraw (int value) { transactions.add(-value); }
22
23      public int balance(){
24          int result =0;
25          for (int value: transactions)
26              result += value;
27          return result;
28      }
29
30      public void close() { transactions = null; }
33
34  }
35
```

HTML report

Coverage: PetclinicIntegrationTests

90% classes, 15% lines covered in package 'org.springframework.samples.pe...

Element	Class, %	Method, %	Line, %	Branch, %
model	100% (3/3)	0% (0/10)	23% (3/13)	100% (0/0)
owner	87% (7/8)	7% (4/51)	9% (14/145)	0% (0/4)
system	100% (3/3)	60% (3/5)	75% (6/8)	100% (0/0)
vet	75% (3/4)	11% (1/9)	17% (4/23)	100% (0/0)
visit	100% (1/1)	14% (1/7)	25% (2/8)	100% (0/0)
PetClinic...	100% (1/1)	0% (0/1)	50% (1/2)	100% (0/0)

Report

[all classes] [it.univr]

Coverage Summary for Package: it.univr

Package	Class, %	Method, %	Line, %
it.univr	100% (2/ 2)	88.9% (8/ 9)	91.3% (21/ 23)

Class
BankAccount
Stack

Coverage Summary for Class: BankAccount (it.univr)

Class	Class, %	Method, %	Line, %
BankAccount	100% (1/ 1)	80% (4/ 5)	84.6% (11/ 13)

```
 1  package it.univr;
 2
 3  import java.util.LinkedList;
 4  import java.util.List;
 5
 6  public class BankAccount {
 7
 8      List<Integer> transactions;
 9
10
11      public BankAccount(){
12          transactions = new LinkedList<>();
13      }
14
15      public void deposit (int value){
16          transactions.add(value);
17      }
18
19      public void withdraw (int value){
20          transactions.add(-value);
21      }
22
23      public int balance(){
24          int result =0;
25          for (int value: transactions)
26              result += value;
27          return result;
28      }
29
30      public void close(){
31          transactions = null;
32      }
33
34  }
```

References

- Reference WebSite: www.junit.org
- https://github.com/junit-team/junit4/wiki
- https://junit.org/junit4/javadoc/latest/index.html