OAK 3 ACADEMY

TECHNICAL END-USER PERSONAL SKILLS

Junit

Table of contents

Introduction JUnit

What is JUnit?

Framework to test software

Make stable software

- **Unit test:** test seperate objects/classes
- **Functional test:** test a functionality (many objects)
- **Integration test:** test full system

Test Driven Development

First write tests, afterwards class that meets conditions.

BENEFIT?

→You will think more about your code, write better and stable code.

JUnit

Focus is on testing UNITS

→ Test each unit (class) individually.

Exercise

- Make new Maven Project: HelloTest
- Add Junit as dependency
- Run **mvn test**

Lifecycle

Test class

Lifecycle

Testrunner → searching for **@Test**

Each test method must be able to be tested separately!

Order testing may not play a role!

Lifecycle

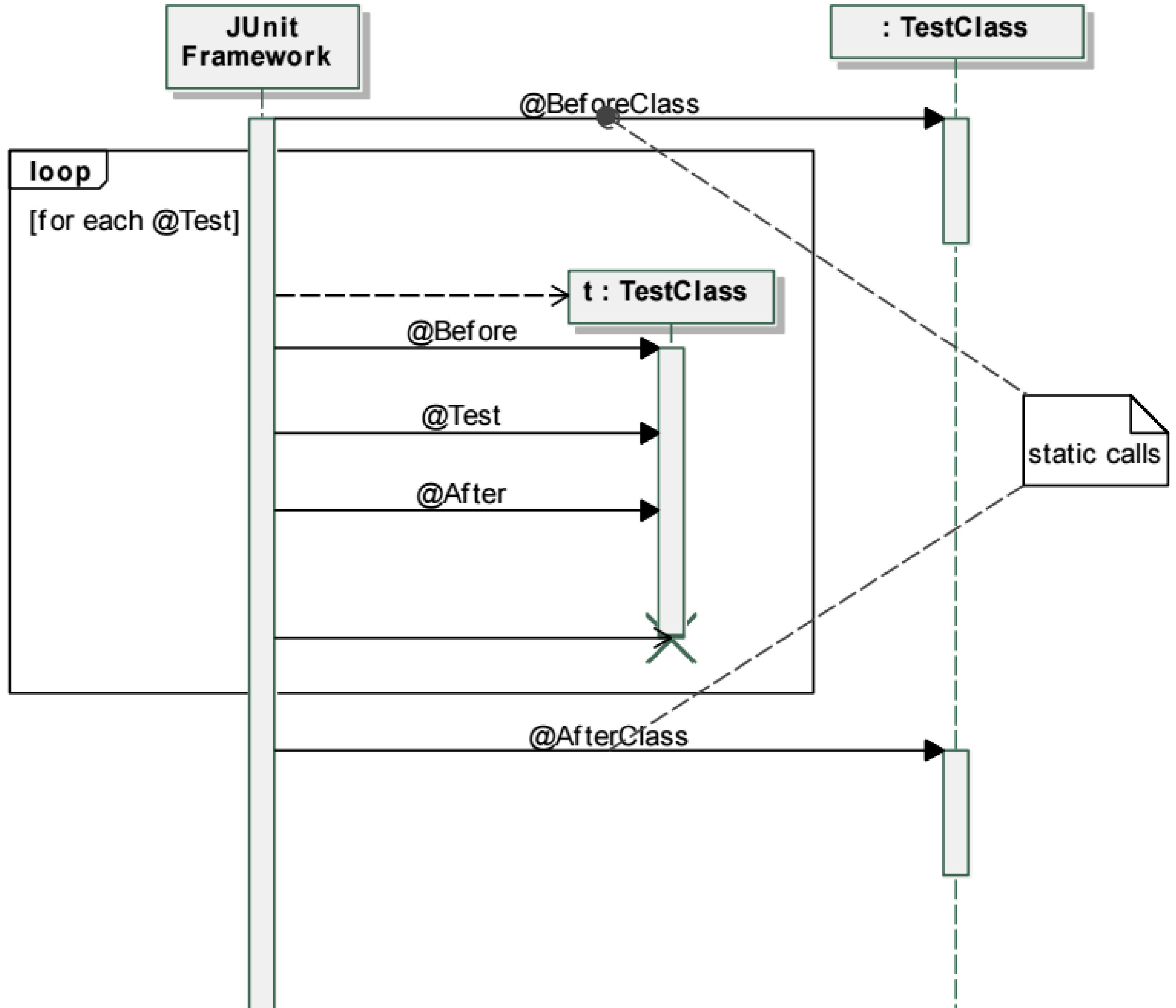
@Before: method is executed before every test

@After: method is executed after every test

@BeforeClass: one time for all tests

@AfterClass: one time after all tests

Schematic


JUnit Framework
: TestClass
@BeforeClass
loop
[for each @Test]
t : TestClass
@Before
@Test
@After
static calls
@AfterClass

Conditions

- public
- void
- No arguments
- May throw exceptions→ test fail!
- Name doesn't matter

@Ignore: ignore one test or all tests in the class

Annotaties

Java 5

Meta information in classes/interfaces

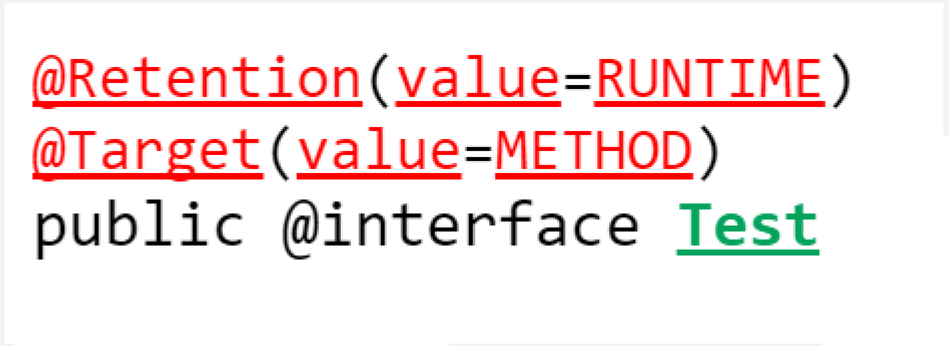
Source: only available in source code (@Override)

Class: Present during compilation but not during execution

Runtime: Also present during execution. (JUnit annotations)

Annotaties

- Annotation test: (interface with @ symbole in front of it)



```
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface Test
```

→ See documentation

Assert methods

Assert class

Class with only static methods

Used to compare

E.G.: Assert.assertEquals(“Java”, “Java”);

http://junit.org/junit4/javadoc/latest/

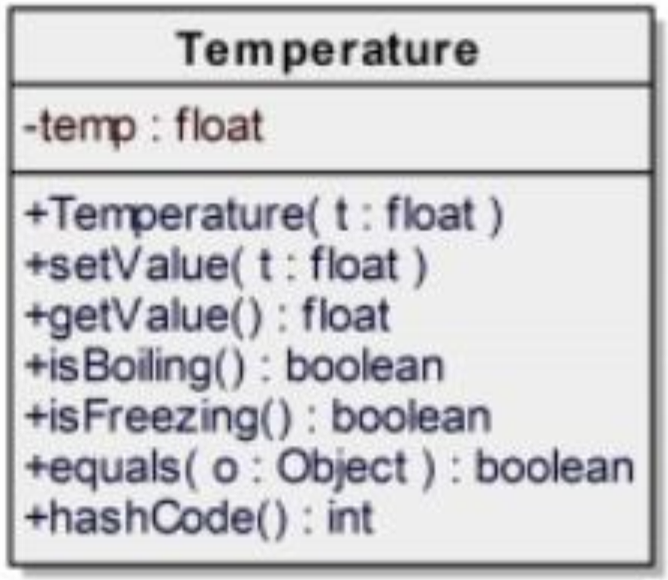
Demo

- HelloWorld → sayHello()
- HelloWorldTest → testSayHello()
- Run **mvn test**

Exercise

- Make class Temperature:

- Make test class.
  - testConstructor()
  - testSetValue()



Fixtures

Bring similar code together

→@Before en @After: code die voor of na elke test gebruikt moet worden.

→@BeforeClass en @AfterClass: code die eenmalig voor testen of na testen moet uitgevoerd worden
Bv: connectie maken en sluiten

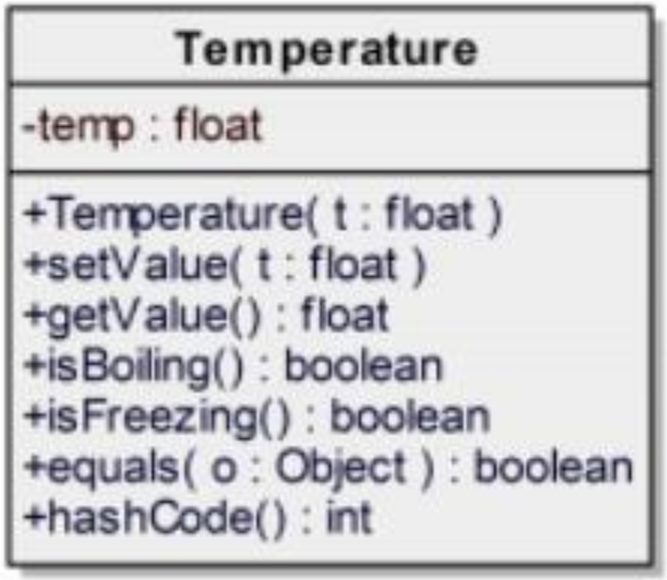
Test boundaries

Difficult to test everything

→ Usefull tests: bv limit values, null, exceptions …

Exercise

- TemperatureTest
  - init(): make Temperature object
  - testBoiling(): Use normal values and values around boiling point.
  - testFreezing(): same



Test Exceptions

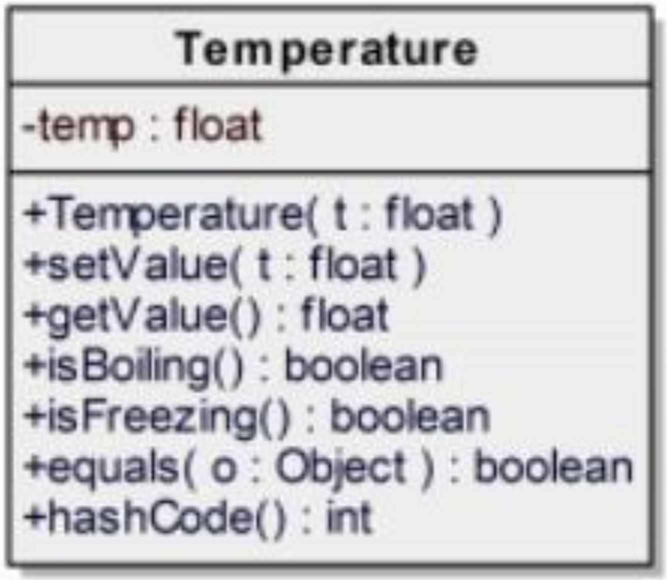
You can test of some methods throw an exception.

→E.G.: @Test(expected=IOException.class)

We expect that this test throws an exception

Exercise

- InvalidTemperatureException (RuntimeException)
- Throw exception if value is below -273,15°
- TemperatureTest
  - testException()



Stub objects

Object A depend on Object B.

You only want to test Object A?

→Make Fake Object B = Stub or dummy-object

→Empy implementation

→Test an isolated piece.

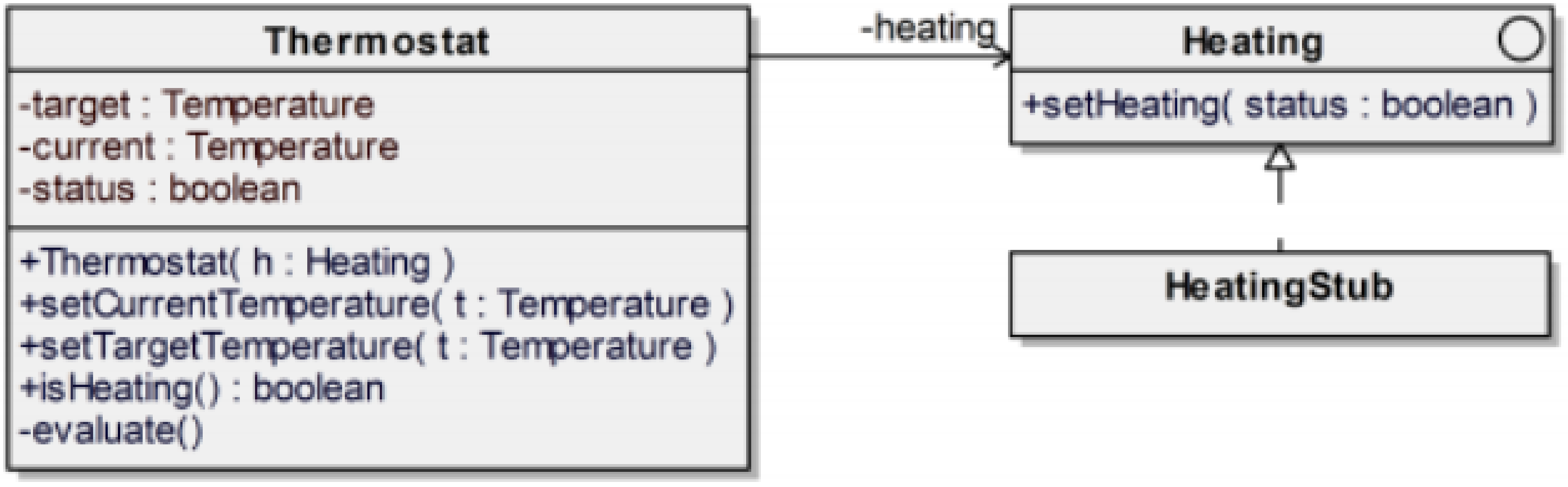
Mock objects

= Stub object with functionality → Mimic certain expectations.

Use Framework for this.

Exercise Stub

Make class ThermostatTest. Use a stub implementation for the Heating interface.



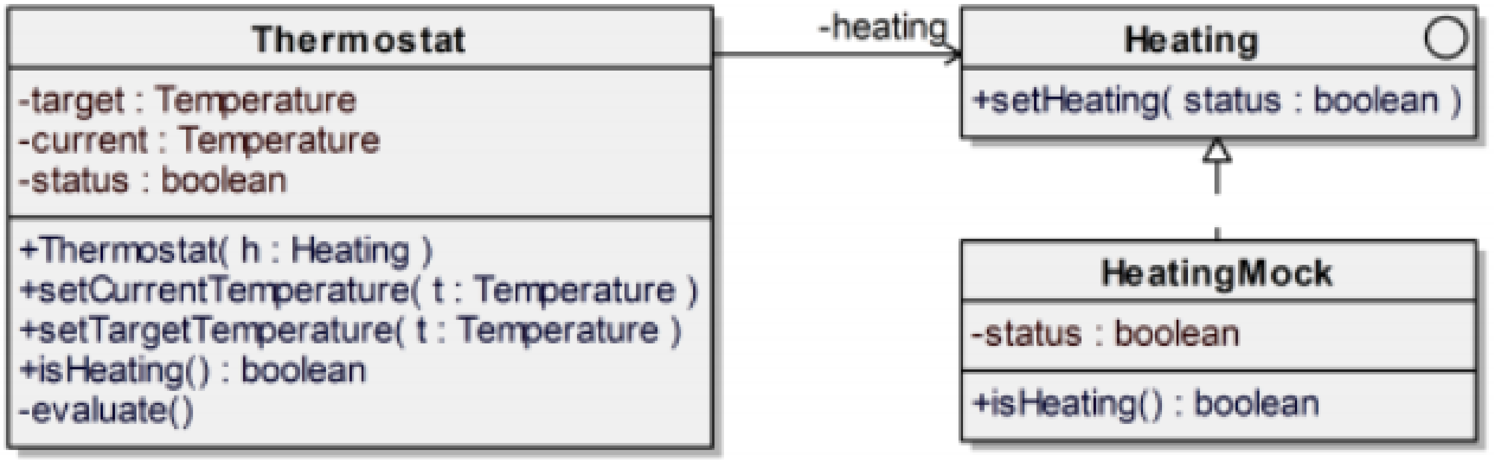
Thermostat
-target : Temperature
-current : Temperature
-status : boolean
+Thermostat(h : Heating)
+setCurrentTemperature(t : Temperature)
+setTargetTemperature(t : Temperature)
+isHeating() : boolean
-evaluate()
-heating
Heating
+setHeating(status : boolean)
HeatingStub

Exercise Stub

- testChangeCurrent(): targetTemperature is 21°, current temperature changes. Test isHeating() method.
- testChangeTarget: current temperature is 20°. Target temperature changes. Test isHeating() method.

Exercise Mock

- Make HeatingMock with method isHeating() to see if the Thermostat is activated or not.
- Make new test to see if the thermostat works correct.



Mockito

Framework to make Mock objects

Framework provides implementation

Test suites - Categoriën

Test class for executing other test classes together.

Can be divided into categories.

→ In one test suite all tests of a certain category can then be performed

Maven commando’s

Only test one class:

```
mvn test –Dtest=MijnTest
```

Ignore tests:

```
mvn clean install -DskipTests
```

Opdracht: JUnit in AutoApp

Maak van de AutoApp een Maven applicatie.
Voeg JUnit toe als Dependency.

Maak voor elke functionaliteit in de auto klasse een aparte test.