HBO Graduaat Informatica Optie Programmeren

jUnit

Test Classes

cvo leerstad

volwassenenonderwijs

What is a Unit test

- A unit test is a test of a single isolated component in a repeatable way. A test consists of four phases:

Prepare	Sets up a baseline for testing and defines the expected results.
Execute	Running the test.
Validate	Validates the results of the test against previously defined expectations.
Reset	Resets the system to the way it was before Prepare.

- JUnit is a popular framework for creating unit tests for Java. It provides a simple yet effective API for the execution of all four phases of a unit test.
- Free download at http://www.junit.org

Test case

- A test case is the basic unit of testing in JUnit and is defined by extending `junit.framework.TestCase`.
- The TestCase class provides a series of methods that are used over the lifecycle of a test.
- When creating a test case, it is required to have one or more test methods. A test method is defined by any method that fits the following criteria:
  - It must be public.
  - It must return void.
  - The name must begin with “test”.

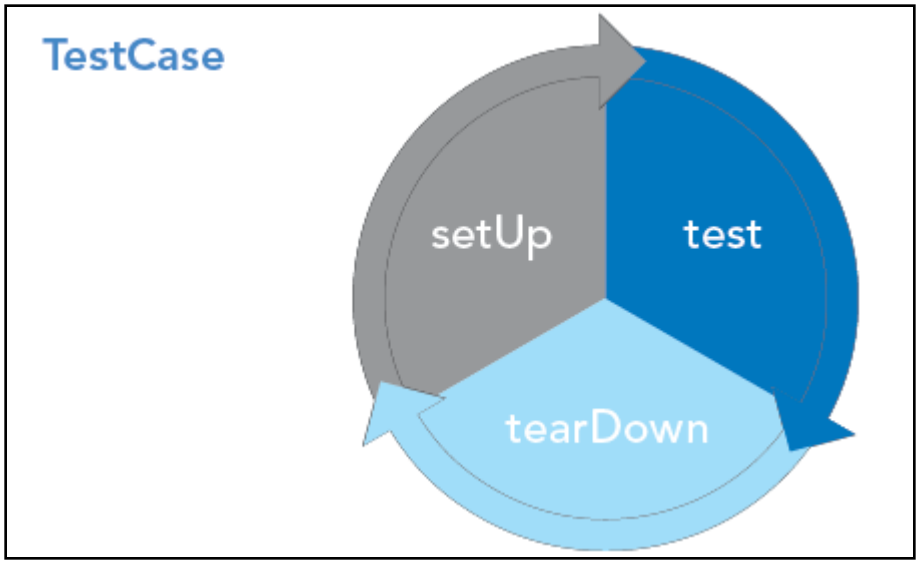
Test case

- Optional lifecycle methods include `public void setUp()` and `public void tearDown()`.`setUp()` is executed before each test method, `tearDown()` is executed after each test method and the execution of both `setUp()` and `tearDown()` are guaranteed.

Place test classes in the same package but different source folder as the class they are testing. That allows the test to have access to protected methods and attributes.

JUnit lifecycle

- A JUnit test case can contain many test methods. Each method identified as a test will be executed within the JUnit test lifecycle. The lifecycle consists of three pieces: setup, test and teardown, all executed in sequence.



JUnit lifecycle

- All of the test methods are guaranteed to be executed. In JUnit 4 two more phases of the lifecycle were added, `beforeClass()` and `afterClass()`. These methods are executed once per test class (instead of once per test method as `setUp` and `tearDown` are), before and after respectively.

Lifecycle stage	Method called	Method description
Setup	`public void setUp()`	Called to do any required preprocessing before a test. Examples include instantiating objects and inserting test data into a database
Test	`public void testXYZ()`	Each test method is called once within the test lifecycle. It performs all required testing. Test results are recorded by JUnit for reporting to the test runner upon completion.
Teardown	`public void tearDown()`	Called to do any required post processing after a test. Examples include cleaning up of database tables and closing database connections.

Assertions

Statement	Description
fail(message)	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)	Checks that the boolean condition is true.
assertFalse([message,] boolean condition)	Checks that the boolean condition is false.
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message,] object)	Checks that the object is null.
assertNotNull([message,] object)	Checks that the object is not null.
assertSame([message,] expected, actual)	Checks that both variables refer to the same object.
assertNotSame([message,] expected, actual	Checks that both variables refer to different objects.

Junit 4 annotations

Annotation	Parameters	Use
`@After`	None	Method will be executed after each test method (similar to the **`tearDown()`** method in JUnit 3.x). Multiple methods may be tagged with the **`@After`** annotation, however no order is guaranteed.
`@AfterClass`	None	Method will be executed after all of the test methods and teardown methods have been executed within the class. Multiple methods may be tagged with the **`@AfterClass`** annotation, however no order is guaranteed.
`@Before`	None	Method will be executed before each test method (similar to the **`setUp()`** method in JUnit 3.x). Multiple methods may be tagged with the **`@Before`** annotation, however no order is guaranteed.
`@BeforeClass`	None	Executed before any other methods are executed within the class. Multiple methods may be tagged with the **`@BeforeClass`** annotation, however no order is guaranteed.

Junit 4 annotations

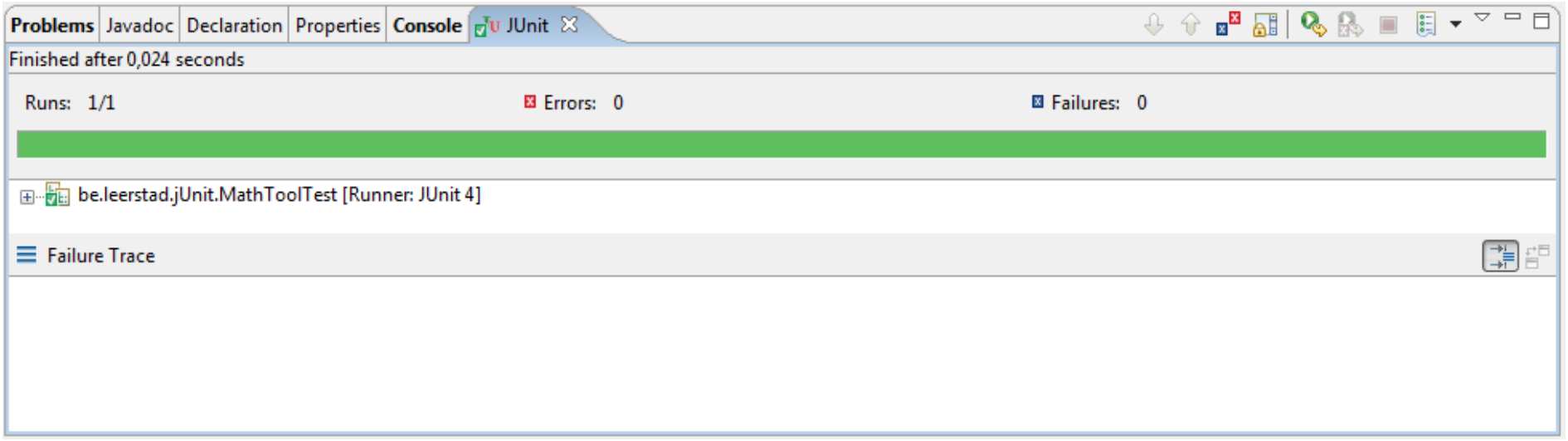
`@Ignore`	String (optional)	Used to temporarily exclude a test method from test execution. Accepts an optional String reason parameter.
`@Parameters`	None	Indicates a method that will return a Collection of objects that match the parameters for an available constructor in your test. This is used for parameter driven tests.
`@RunWith`	Class	Used to tell JUnit the class to use as the test runner. The parameter must implement the interface `junit.runner.Runner`.
`@SuiteClasses`	Class []	Tells JUnit a collection of classes to run. Used with the `@RunWith(Suite.class)` annotation is used.
`@Test`	■ Class(optional) ■ Timeout(optional)	Used to indicate a test method. Same functionality as naming a method `public void testXYZ()` in JUnit 3.x. The class parameter is used to indicate an exception is expected to be thrown and what the exception is. The timeout parameter specifies in milliseconds how long to allow a single test to run. If the test takes longer than the timeout, it will be considered a failure.

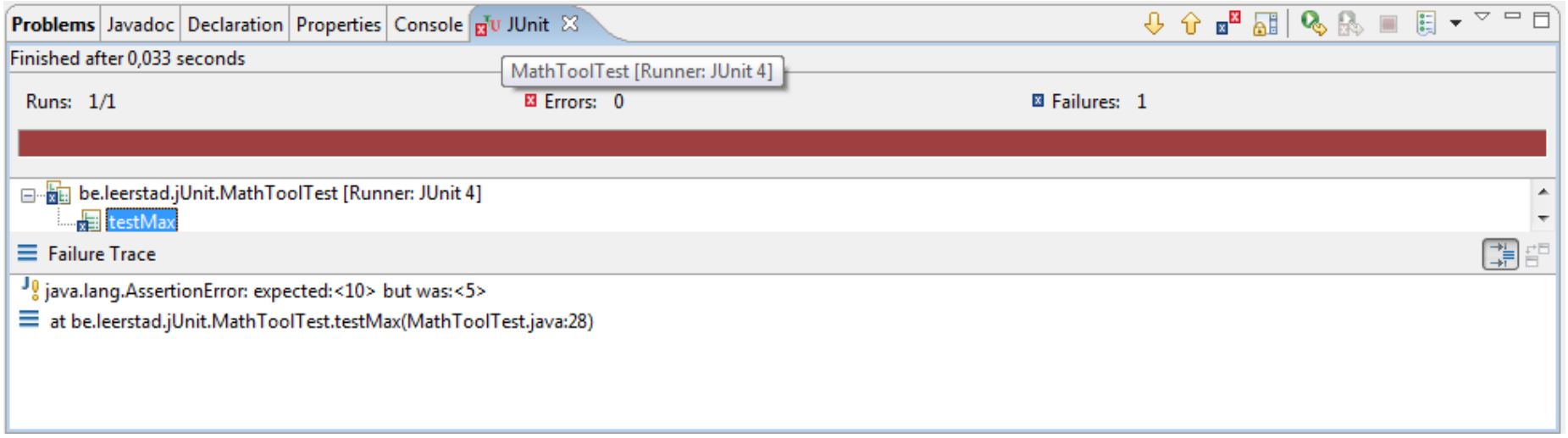
JUnit 4

```
public class FooTestCase {
   private Foo foo;

   @Before
   public void buildFoo() {
     foo = new Foo();
   }

   @Test
   public void testGoodResultsBar() {
     String param1 = "parameter1";
     String results = foo.bar(param1);
     assertNotNull("results was null", results);
     assertEquals("results was not 'good'", "good",results);
   }

   @Test
   public void testBadResultsBar() {
     try {
          String results = foo.bar(null);
     } catch (NullPointerException npe) {
          return;
     }
     fail();
     }

   @After
   public void closeFoo() {
     foo.close();
   }
}
```

When using JUnit 4, you do not need to extend junit.framework.TestCase. Any plain old java object (POJO) can be be run as a test with the appropriate annotations.

Result


Problems
Javadoc
Declaration
Properties
Console
JUnit
Finished after 0,024 seconds
Runs: 1/1
Errors: 0
Failures: 0
be.leerstad.jUnit.MathToolTest [Runner: JUnit 4]
Failure Trace


Problems
Javadoc
Declaration
Properties
Console
JUnit
Finished after 0,033 seconds
MathToolTest [Runner: JUnit 4]
Runs: 1/1
Errors: 0
Failures: 1
be.leerstad.jUnit.MathToolTest [Runner: JUnit 4]
testMax
Failure Trace
java.lang.AssertionError: expected:<10> but was:<5>
at be.leerstad.jUnit.MathToolTest.testMax(MathToolTest.java:28)

Test Suite

- A test suite is a collection of tests cases.
  - It is used to run a collection of tests and aggregate the results.
- There are two ways to create a test suite, programmatically and with annotations.

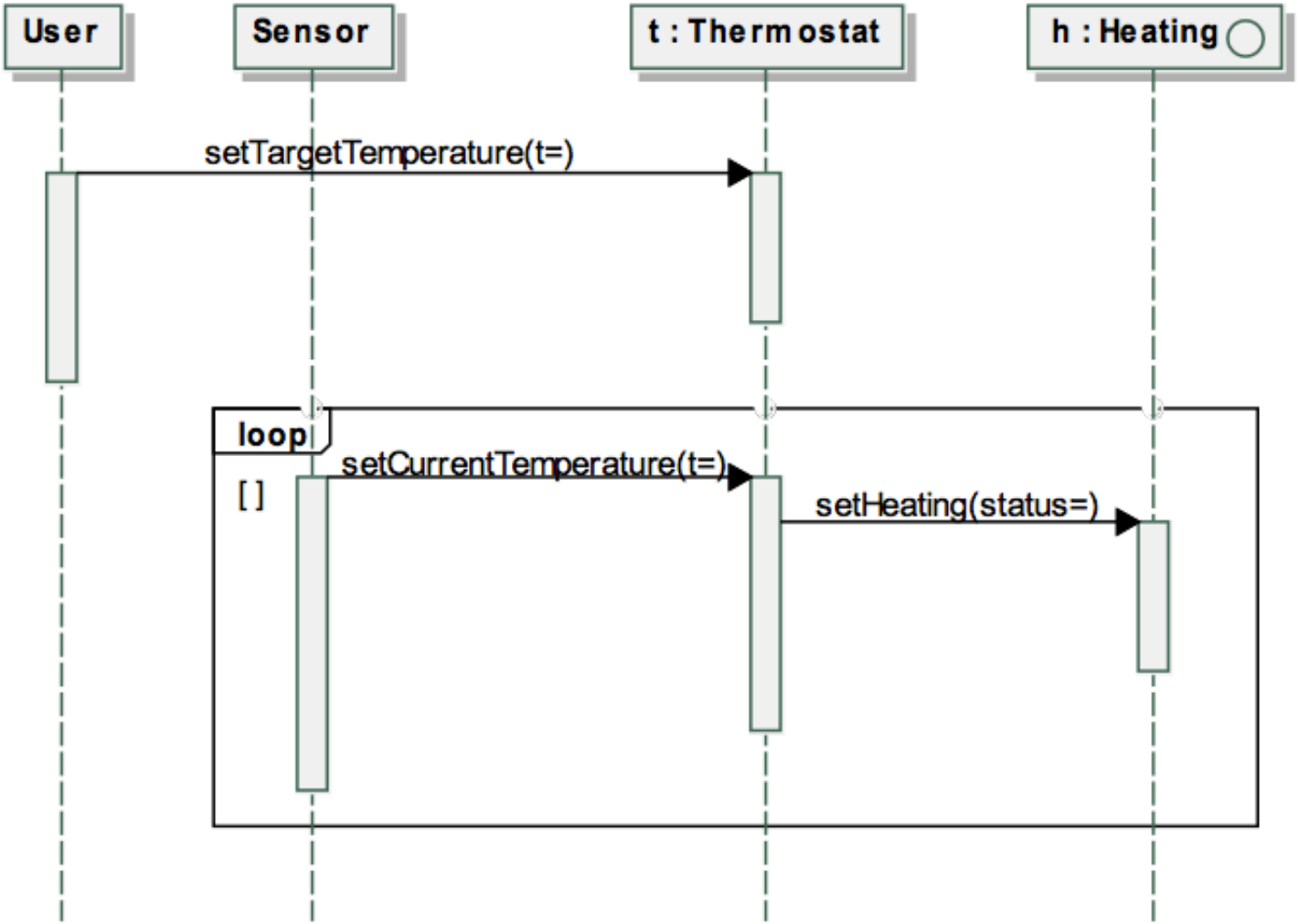
Test Suite

```
@RunWith(Suite.class)
@Suite.SuiteClasses({BasePlusCommissionEmployeeTest.class,
        CommissionEmployeeTest.class,
        HourlyEmployeeTest.class,
        SalariedEmployeeTest.class,})
public class AllTests {

}
```

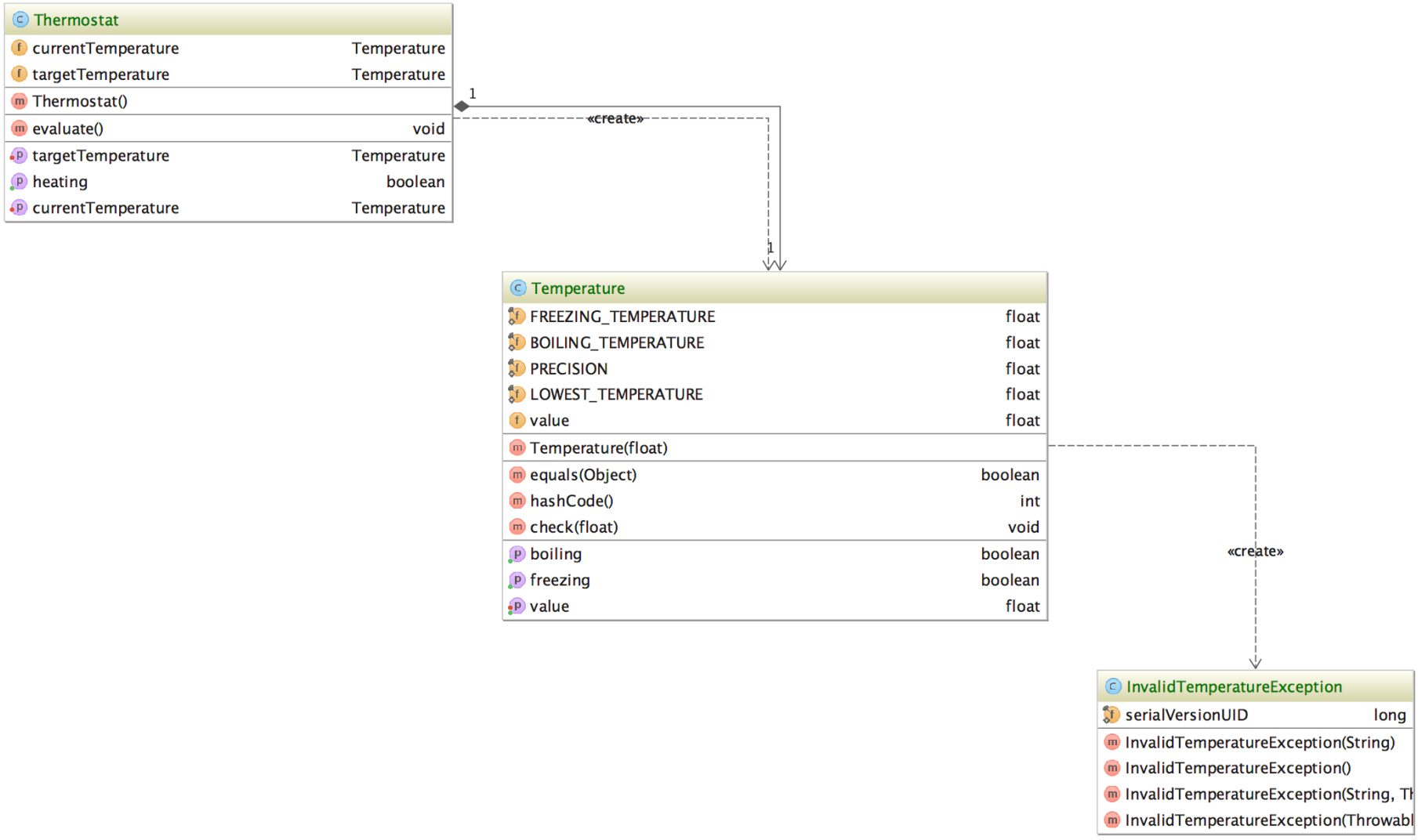
Testing Exceptions

```
@Test(expected = InvalidTemperatureException.class)
public void testException() {
    t.setValue(-274F);
}
```

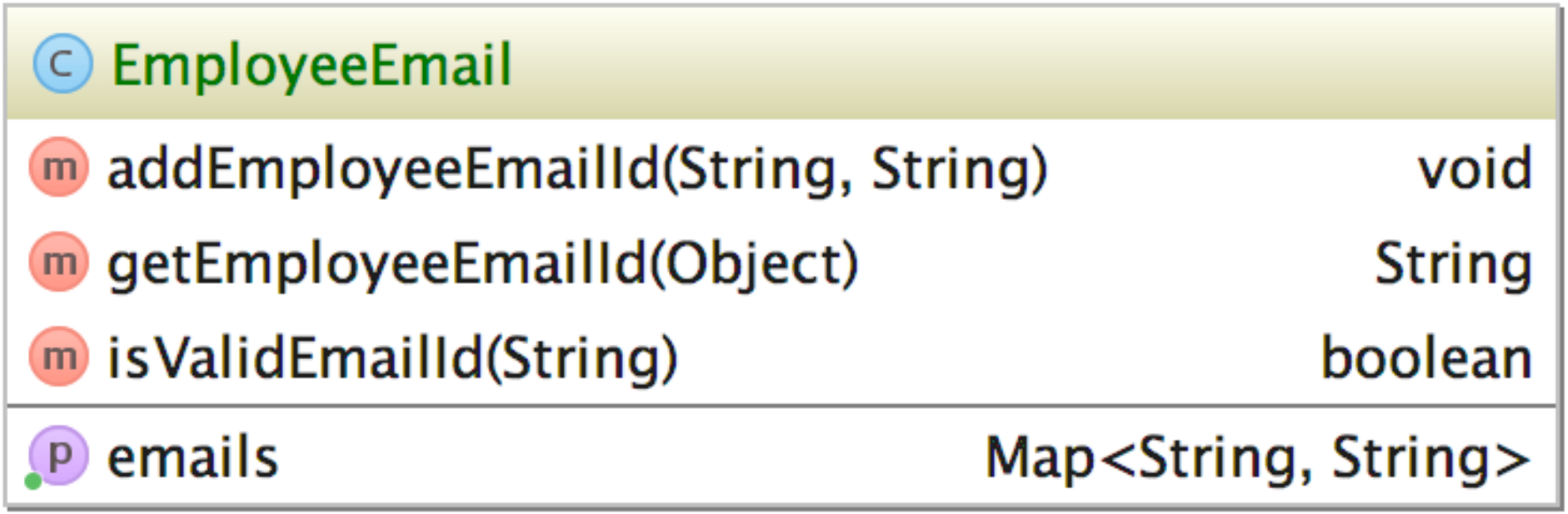
Thermostat exercise



Thermostat exercise



TDD


EmployeeEmail
addEmployeeEmailId(String, String) void
getEmployeeEmailId(Object) String
isValidEmailId(String) boolean
emails Map<String, String>

JUnit extensions

Add-on	URL	Use
DbUnit	http://dbunit.sourceforge.net/	Provides functionality relevant to database testing including data loading and deleting, validation of data inserted, updated or removed from a database, etc.
HttpUnit	http://httpunit.sourceforge.net/	Impersonates a browser for web based testing. Emulation of form submission, JavaScript, basic http authentication, cookies and page redirection are all supported.
EJB3Unit	http://ejb3unit.sourceforge.net/	Provides necessary features and mock objects to be able to test EJB 3 objects out of container.
JUnitPerf	http://clarkware.com/software/JUnitPerf.html	Extension for creating performance and load tests with JUnit.

Questions ?