

Contents

1	Package extension.annotations	2
1.1	Interfaces	3
1.1.1	INTERFACE Depends	3
2	Package extension	4
2.1	Classes	5
2.1.1	CLASS ComposedTestRunner	5
2.1.2	CLASS CycleDetector	5
2.1.3	CLASS DependencyParser	6
2.1.4	CLASS DependencyValidator	7
2.1.5	CLASS MethodCollector	8
2.1.6	CLASS MethodValidator	9
2.1.7	CLASS TestClass	10
2.1.8	CLASS TestGraph	12
2.1.9	CLASS TestMethod	13

Chapter 1

Package extension.annotations

<i>Package Contents</i>	<i>Page</i>
<hr/>	
Interfaces	
Depends	3
<i>The Depends Annotation defines the dependencies of a test method.</i>	
<hr/>	

1.1 Interfaces

1.1.1 INTERFACE Depends

The **Depends** Annotation defines the dependencies of a test method.

DECLARATION

<pre>public interface Depends implements java.lang.annotation.Annotation</pre>

METHODS

- *value*
public String **value**()
 - **Returns** - a String representing the Method 's the declaring Method depends on.

Chapter 2

Package extension

<i>Package Contents</i>	<i>Page</i>
<hr/>	
Classes	
ComposedTestRunner	5
<i>The ComposedTestRunner class is the Runner for composed JUnit Tests.</i>	
CycleDetector	5
<i>The CycleDetector class checks the test dependencies for cycles.</i>	
DependencyParser	6
<i>The DependencyParser class parses a String for the Method 's it represents.</i>	
DependencyValidator	7
<i>The DependencyValidator class validates the specified dependencies between tests.</i>	
MethodCollector	8
<i>The MethodCollector class collects all Method 's involved in a test run.</i>	
MethodValidator	9
<i>The MethodValidator class validates all test methods in testClass.</i>	
TestClass	10
<i>A wrapper for the Class under test.</i>	
TestGraph	12
<i>The TestGraph class takes the responsibility delegated from ComposedTestRunner : validating Method 's, running tests and returning Description 's.</i>	
TestMethod	13
<i>The wrapper for the Method 's to be run.</i>	

2.1 Classes

2.1.1 CLASS `ComposedTestRunner`

The `ComposedTestRunner` class is the Runner for composed JUnit Tests. It delegates everything to the Singleton `TestGraph` .

DECLARATION

```
public class ComposedTestRunner
extends Runner
```

CONSTRUCTORS

- *ComposedTestRunner*
 public **ComposedTestRunner**(java.lang.Class **underTest**)
 – **Parameters**
 * **underTest** - the Class to be run as a test

METHODS

- *getDescription*
 public Description **getDescription**()
- *run*
 public void **run**(RunNotifier **notifier**)

METHODS INHERITED FROM CLASS `Runner`

2.1.2 CLASS `CycleDetector`

The `CycleDetector` class checks the test dependencies for cycles.

DECLARATION

```
public class CycleDetector
extends java.lang.Object
```

CONSTRUCTORS

- *CycleDetector*
 public **CycleDetector**(java.util.Collection testMethods)
 - **Parameters**
 - * testMethods - the Collection of TestMethod 's that have to be checked for cycles

METHODS

- *hasCycle*
 public boolean **hasCycle**()
 - **Usage**
 - * Does for all not visited TestMethod 's a depth-first search and marks the visited nodes. Nodes whos dependencies were all visited are marked as 'done'. If you encounter a visited node, there is a cycle.
 - **Returns** - **true** if the dependencies are cyclic, **false** otherwise.

2.1.3 CLASS DependencyParser

The `DependencyParser` class parses a String for the Method 's it represents.

DECLARATION

```
public class DependencyParser
extends java.lang.Object
```

CONSTRUCTORS

- *DependencyParser*

```
public DependencyParser( extension.TestClass myTestClass )
```

- **Parameters**

- * myTestClass - the TestClass that is to be run

METHODS

- *getDependencies*

```
public List getDependencies( java.lang.String value,  
java.lang.reflect.Method method )
```

- **Usage**

- * Name and arguments of the Method's defined in value are extracted and used to get the Method object from it's declaring class.

- **Parameters**

- * value - the value from the Annotation Depends .
 - * method - the Method that depends on the Method's to return.

- **Returns** - a List of the Method's method depends on.

- **Exceptions**

- * java.lang.ClassNotFoundException -
 - * java.lang.SecurityException -
 - * java.lang.NoSuchMethodException -

2.1.4 CLASS *DependencyValidator*

The *DependencyValidator* class validates the specified dependencies between tests.

DECLARATION

```
public class DependencyValidator  
extends java.lang.Object
```

CONSTRUCTORS

- *DependencyValidator*
`public DependencyValidator()`

METHODS

- *dependencyIsValid*
`public List dependencyIsValid(java.lang.reflect.Method method,
java.lang.reflect.Method [] dependencies)`
 - **Usage**
 - * The following checks are made and have to be passed: - all the dependencies have to be test methods - a Method cannot have itself as a dependency - if `method` takes arguments, the number of dependencies have to be the same and all the dependencies have to return the appropriate object
 - **Parameters**
 - * `method` - the Method that has dependencies
 - * `dependencies` - the dependencies of `method`
 - **Returns** - `true`, if all the dependencies are valid, `false` otherwise

2.1.5 CLASS MethodCollector

The `MethodCollector` class collects all `Method` 's involved in a test run.

DECLARATION

```
public class MethodCollector
extends java.lang.Object
```

CONSTRUCTORS

- *MethodCollector*
`public MethodCollector(extension.TestClass testClass, java.util.Map
alreadyCollectedMethods)`

- **Parameters**

- * **testClass** - the TestClass the Method 's have to be collected from
- * **alreadyCollectedMethods** - a Map of already collected Method 's, e.g. when a TestSuite is run

METHODS

- *collectTestMethods*

```
public Map collectTestMethods( )
```

- **Usage**

- * All Method 's are collected in a non-recursive way, so no endless-loop is risked. For all Method 's of **this.testClass** is checked, whether the dependencies are already collected. If not, they are processed in a second loop and so on, until there are no new Method 's to process anymore.

- **Returns** - a Map with all the collected Method 's as keys and TestMethod 's as values in it

- **Exceptions**

- * `java.lang.SecurityException` -
- * `java.lang.NoSuchMethodException` -
- * `java.lang.ClassNotFoundException` -

2.1.6 CLASS MethodValidator

The MethodValidator class validates all test methods in **testClass**.

DECLARATION

```
public class MethodValidator
extends java.lang.Object
```

CONSTRUCTORS

- *MethodValidator*

```
public MethodValidator( java.util.Set  methodUnderTest,
extension.TestClass  testClass )
```

- **Parameters**

- * `methodUnderTest` - a Set of Method under test
- * `testClass` - the `TestClass` to be run

METHODS

- *assertValid*

```
public void assertValid( )
```

- **Usage**

- * Checks if the list of errors is empty, if not, an `InitializationError` is thrown

- **Exceptions**

- * `InitializationError` -

- *validateMethodsForComposedRunner*

```
public List validateMethodsForComposedRunner( )
```

- **Usage**

- * Checks, if there is a default constructor, if there are test methods and if they are all public and if their declaring classes are also public. In the end the declared dependencies are validated.

- **Returns** - a List of all encountered errors

2.1.7 CLASS `TestClass`

A wrapper for the Class under test.

DECLARATION

```
public class TestClass
extends java.lang.Object
```

CONSTRUCTORS

- *TestClass*

```
public TestClass( java.lang.Class klass )
```

- **Parameters**

- * `klass` - the `Class` under test

METHODS

-
- *equals*
`public boolean equals(java.lang.Object obj)`
 - *getConstructor*
`public Constructor getConstructor()`
 - **Returns** - the `Constructor` of `fClass`
 - **Exceptions**
 - * `java.lang.SecurityException` -
 - * `java.lang.NoSuchMethodException` -
-
- *getDependenciesFor*
`public List getDependenciesFor(java.lang.reflect.Method testMethod)`
 - **Parameters**
 - * `testMethod` - the `Method` whos dependencies have to be returned
 - **Returns** - a `List` of `Method` 's `testMethod` depends on
 - **Exceptions**
 - * `java.lang.NoSuchMethodException` -
 - * `java.lang.SecurityException` -
 - * `java.lang.ClassNotFoundException` -
-
- *getJavaClass*
`public Class getJavaClass()`
 - **Returns** - the `Class` object of `fClass`
-
- *getName*
`public String getName()`
 - **Returns** - the name of `fClass`
-
- *getTestMethods*
`public List getTestMethods()`
 - **Returns** - a `List` of all `Method` 's annotated with `Test`

2.1.8 CLASS TestGraph

The **TestGraph** class takes the responsibility delegated from **ComposedTestRunner** : validating Method 's, running tests and returning Description 's.

DECLARATION

```
public class TestGraph
extends java.lang.Object
```

CONSTRUCTORS

- *TestGraph*
public **TestGraph**()

METHODS

- *addClass*
public void **addClass**(extension.TestClass testClass)
 - **Usage**
 - * All Method 's are collected, checked for cycles, validated and then added to Map of all the Method 's to be run.
 - **Parameters**
 - * testClass - the TestClass to be added
 - **Exceptions**
 - * **InitializationError** -
- *descriptionForClass*
public Description **descriptionForClass**(extension.TestClass testClass)
 - **Usage**
 - * The Description 's for the Method 's of this class are added as children to the class description. If there are dependencies from TestMethod 's which are not declared in a TestClass that is run in this turn, the Description of the declaring Class is also added as a child.

– **Parameters**

* **testClass** - the TestClass to get the Description from

– **Returns** - the description for testClass;

- *getClasses*

```
public Set getClasses( )
```

– **Usage**

* Only for testing purposes

– **Returns** - a Set of TestClass Objects

- *getInstance*

```
public static TestGraph getInstance( )
```

- *getTestMethods*

```
public Map getTestMethods( )
```

– **Usage**

* Only for testing purposes

– **Returns** - a Map with the mapping Method ->TestMethod

- *runClass*

```
public void runClass( extension.TestClass testClass, RunNotifier notifier )
```

– **Usage**

* All TestMethod 's of testClass are run, inclusive their dependencies.

– **Parameters**

* **testClass** - the TestClass to be run

* **notifier** - RunNotifier

2.1.9 CLASS TestMethod

The wrapper for the Method 's to be run.

DECLARATION

```
public class TestMethod
extends java.lang.Object
```

CONSTRUCTORS

• *TestMethod*

```
public TestMethod( java.lang.reflect.Method  method )
```

– **Parameters**

* `method` - the `Method` to be run

METHODS

• *addDependency*

```
public void addDependency( extension.TestMethod  testMethod )
```

– **Usage**

* If the `TestMethod` doesn't already have the dependency `testMethod`, `testMethod` is added as a dependency.

– **Parameters**

* `testMethod` - the `TestMethod` to be added as a dependency

• *belongsToClass*

```
public boolean belongsToClass( extension.TestClass  testClass )
```

– **Usage**

* Checks, if this `TestMethod` belongs to `testClass`

– **Parameters**

* `testClass` - the `TestClass` to be compared

– **Returns** - true, if the `TestMethod` belongs to `testClass`, false otherwise

• *createDescription*

```
public Description createDescription( )
```

– **Returns** - the test `Description` of this `TestMethod`

• *equals*

```
public boolean equals( java.lang.Object  obj )
```

• *extractDependencies*

```
public List extractDependencies( extension.TestClass  testClass )
```

– **Parameters**

- * `testClass` - the `TestClass` the method is declared in
 - **Returns** - a `List` of `Method` 's, which are the dependencies of this `TestMethod`
 - **Exceptions**
 - * `java.lang.SecurityException` -
 - * `java.lang.ClassNotFoundException` -
 - * `java.lang.NoSuchMethodException` -

- *getDeclaringClass*
`public Class getDeclaringClass()`
 - **Returns** - the declaring `Class` of `javaMethod`

- *getDependencies*
`public List getDependencies()`
 - **Returns** - a `List` of `TestMethod` 's, being the dependencies

- *run*
`public void run(RunNotifier notifier)`
 - **Usage**
 - * Runs this `TestMethod` after it run all of its dependencies.
 - **Parameters**
 - * `notifier` - the `RunNotifier`