Table of Contents

Alle unsere Java Klassen verwenden einen einheitlichen Style. Außerdem verwenden wir Checkstyle bzw. das Checkstyle Eclipse Plugin, mit vordefinierten Rules, um keine Zeit bei der Entwicklung zu verschwenden.

Die Checkstyle rules sind im **JVx Repository** unter dem Dateinamen

```
<jvx>/trunk/java/library/checkstyle_opensource.xml
```

abgelegt.

Klassen

Für Klassen verwenden wir folgenden Style:

ClassTemplate.java

```
/*
 * Copyright 2018 SIB Visions GmbH
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not
 * use this file except in compliance with the License. You may obtain
a copy of
 * the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See
the
 * License for the specific language governing permissions and
limitations under
 * the License.
 *
 *
 * History
 *
 * dd.MM.yyyy - [XX] - creation
 */
package com.sibvisions.foo;

/**
 * I do that and that ....
 *
 * @author First Last
 */
public class Bar
{
```

```java
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Class members
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /** The foo type. */
    public static final int TYPE_FOO = 1;


    /** The value of foo bar. */
    private Object oValue;

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Initialization
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /**
     * Creates a new instance of <code>Bar</code>.
     */
    public Bar()
    {
    }

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Abstract methods implementation
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Interface implementation
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Abstract methods
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Overwritten methods
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /**
     * {@inheritDoc}
     */
    @Override
    public String toString()
    {
        return "Foo";
    }

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // User-defined methods
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
    /**
     * Sets the value.
     *
     * @param pValue the value.
     */
    public void setValue(Object pValue)
    {
        this.oValue = pValue;
    }

    /**
     * Gets the value.
     *
     * @return the value.
     */
    public Object getValue()
    {
        return oValue;
    }

    //*************************************************************************
    // Subclass definition
    //*************************************************************************

}    // Bar
```

Folgende Regeln werden durch diese Vorlage definiert:

- Variablendeklaration zu Beginn (zuerst Konstante, danach veränderbare Variablen)
- danach Konstrukten und Initialisierungsmethoden
- danach die Implementierungen von abstrakten Methoden
- danach die Implementierungen von Interface Methoden
- danach die Definition von abstrakten Methoden
- danach alle überschriebenen Methoden (gekennzeichnet mit @Override)
- danach alle Methoden der Klasse
- Sub/Inner Klassen am Ende

- Jeder Parameter einer Methode wird mit dem Prefix “p” gekennzeichnet
- Für Instanz Variablen wird ebenfalls ein Prefix verwendet wie z.B.:

```
String sValue = "bar";
```

- Im Header erstellen wir bei wichtigen Änderungen einen Hinweis mit Zeitstempel und Autor
- Dokumentation für die Klassen Deklaration, ALLE Methoden und Instanz Variablen bzw. Konstante

Interfaces

Für Interfaces verwenden wir folgenden Style:

InterfaceTemplate.java

```java
/*
 * Copyright 2018 SIB Visions GmbH
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not
 * use this file except in compliance with the License. You may obtain
a copy of
 * the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See
the
 * License for the specific language governing permissions and
limitations under
 * the License.
 *
 *
 * History
 *
 * dd.MM.yyyy - [XX] - creation
 */
package com.sibvisions.foo;

/**
 * I do that and that ....
 *
 * @author First Last
 */
public interface IBar
{
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Constants
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /** The foo type. */
    public static final int TYPE_FOO = 1;

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Method definitions
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /**
     * Sets the value.
     *
     * @param pValue value.
```

```
     */
    public void setValue(Object pValue);

    //////////////////////////////////////////////////////////////////
    // Subinterface definition
    //////////////////////////////////////////////////////////////////

}    // IBar
```

Folgende Regeln werden durch diese Vorlage definiert:

- Konstante werden zu Beginn definiert
- danach Interface Methoden
- Sub/Inner Interfaces am Ende

- Jedes Interface beginnt mit “I”
- Im Header erstellen wir bei wichtigen Änderungen einen Hinweis mit Zeitstempel und Autor
- Dokumentation für die Interface Deklaration, ALLE Methoden und Konstante

Unit Tests

Durch den Einsatz von Unit Tests stellen wir sicher das die Basisfunktionalitäten wie erwartet funktionieren. Ein Unit Test kann niemals die komplette Funktionalität in allen erdenkbaren Konstellationen testen, doch ohne Unit Tests können die Qualitätsanforderungen nicht erfüllt werden. Wir setzen daher ein funktionierendes Set an Unit Tests voraus.

Die Unit Tests werden getrennt vom Core Source Code gespeichert:

```
<jvx>/trunk/java/library/src/com/sibvisions/foo
<jvx>/trunk/java/library/test/com/sibvisions/foo
```

Als Testing Framework kommt JUnit zum Einsatz.

Für Unit Tests verwenden wir folgenden Style:

TestTemplate.java

```
/*
 * Copyright 2018 SIB Visions GmbH
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not
 * use this file except in compliance with the License. You may obtain
a copy of
 * the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
```

```
 * distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT
 * WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See
the
 * License for the specific language governing permissions and
limitations under
 * the License.
 *
 *
 * History
 *
 * dd.MM.yyyy - [XX] - creation
 */
package com.sibvisions.foo;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 * Tests the functionality of ...
 *
 * @author First Last
 */
public class TestBar
{
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Class members
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Initialization
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /**
     * Initializes the unit test.
     *
     * @throws Exception if initialization fails
     */
    @BeforeClass
    public static void beforeClass() throws Exception
    {
    }

    /**
     * Resets the unit test.
     *
     * @throws Exception if reset fails
```

```
     */
    @AfterClass
    public static void afterClass() throws Exception
    {
    }

    /**
     * Sets values before each test.
     *
     * @throws Exception if set values fails
     */
    @Before
    public void beforeTest() throws Exception
    {
    }

    /**
     * Reset values after each test.
     *
     * @throws Exception if reset values fails
     */
    @After
    public void afterTest() throws Exception
    {
    }

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Abstract methods implementation
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Interface implementation
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Overwritten methods
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // User-defined methods
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Test methods
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    /**
     * Tests the ... method.
     */
    @Test
    public void testGet()
```

```
    {
    }

    //****************************************************************
    // Subclass definition
    //****************************************************************

}    // TestBar
```

Folgende Regeln werden durch diese Vorlage definiert:

- Variablendeklaration zu Beginn (zuerst Konstante, danach veränderbare Variablen)
- danach Methoden für die Test Initialisierung
- danach die Implementierungen von abstrakten Methoden
- danach die Implementierungen von Interface Methoden
- danach alle überschriebenen Methoden (gekennzeichnet mit @Override)
- danach alle Methoden der Klasse
- danach alle Test Methoden (gekennzeichnet mit @Test)
- Sub/Inner Klassen am Ende

- Jede Test Klasse beginnt mit “Test”
- Jede Test Methode beginnt mit “test”
- Im Header erstellen wir bei wichtigen Änderungen einen Hinweis mit Zeitstempel und Autor
- Dokumentation für die Klassen Deklaration, ALLE Methoden und Instanz Variablen bzw. Konstante