chem-bla-ics

State of CDK 1.2.0...

Egon Willighagen

Published December 26, 2008

Citation
Willighagen, E. (2008). State of CDK 1.2.0... In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/xgy88-kfs02

Keywords
Cdk, Junit, Java

Abstract
The reason why I have not blogged in more than two weeks, was that I was hoping to blog about the CDK 1.2.0 release. This was originally aimed at September, slipped into October, November and then December. There were only three show stoppers (see this wiki page), one of which the IChemObject interfaces were not properly tested.

Copyright
Copyright © Egon Willighagen 2008. Distributed under the terms of the Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

The reason why I have not blogged in more than two weeks, was that I was hoping to blog about the CDK 1.2.0 release. This was originally aimed at September, slipped into October, November and then December. There were only three show stoppers (see this wiki page), one of which the IChemObject interfaces were not properly tested.

The problem was that the unit tests for the methods in superinterfaces were not applied to implementations of subinterfaces. For example, the unit test for `IElement.getSymbol()` was not applied to the class `Atom`, which implements `IAtom` which is a subinterfaces of `IElement`.

In fixing this, I had to take some hurdles. For example: the unit test classes used a set up following the implementations; CDK 1.2.x has three implementations of the interfaces: data, datadebug and nonotify. The last does not send around update notifications, and rough tests indicate it is about 10% faster. The second implementation sends messages to the debugger for every modification of the data classes, which is, clearly, useful for debugging purposes.

However, the JUnit4 test classes were basically doing the same. The unit test `DebugAtomTest` inherited form `AtomTest`, and only overwrote customizations. `AtomTest`, itself, inherited from `ElementTest`. That's where things got broken. In the single implementation set up, this would have been fine, but to allow testing of all three implementations, `getBuilder()` had to be used.

And when I implemented that, I did not realize that `ElementTest` would do a test like:

```
IElement element = builder.newElement();
// test IElement functionality
```

However, while the use of builder ensure testing of all three implementations, it does **not** run these tests on `IAtom` implementations.

The followed a long series of patches to get this fixed. One major first patch, was to define unit test frameworks like `AbstractElementTest` which formalized running unit tests on any implementation, as I noticed that quite a few tests were still testing one particular implementation. This allowed `DebugElementTest` to extend `AbstractElementTest`, instead of `ElementTest`, which would now extend `AbstractElementTest` too.

OK, with that out of the way, it was time to fix running the unit test for `IElement.getSymbol()` on `IAtom.getSymbol()`, which required the removal of the use of `IChemObjectBuilder` implementations. So, I introduced `newChemObject()` which would return a fresh instance of the actually tested implementation. That is, `DebugAtomTest` would return a new `DebugAtom`, and the `getSymbol()` test would now run on `DebugAtom` and not `DebugElement`. Good.

No, not good. The actual implementation I was using, looks like:

```
public class DebugElementTest extend AbstractElementTest {
  @BeforeClass public static void setup() {
    setChemObject(new DebugElement());
```

```
  }
}

public abstract class AbstractElementTest extend AbstractChemObjectTest {
  @Test public void testGetSymbol() {
    IElement element = (IElement)newChemObject();
    // do testing
  }
}

public abstract class AbstractChemObjectTest {
  private IChemObject testedObject;
  public static setChemObject(IChemObject object) {
    this.testedObject = object;
  }
  public IChemObject setChemObject(IChemObject object) {
    return (IChemObject)testedObject.clone();
  } // just imagine it has try/catch here too

  // and here the tests for the IChemObject API
  @Test public void testGetProperties() {
    IChemObject element = (IChemObject)newChemObject();
    // do testing
  }
}
```

Excellent! No.

Well, yes. The above system works, but made many unit tests fail, because of bugs in `clone()` methods. The full scope has to be explored, but at least `IPolymer.clone()` is not doing what I would expect it to do. Either I am wrong, and need to overwrite the clone unit tests of superinterfaces in `AbstractPolymerTest`, or the implementations needs fixing. I emailed the cdk-devel mailing list and filed a bug report. But having about 1000 unit tests fail, because of clone broken, is something I did not like. For example, as it makes bug fixing more difficult.

So, next step was to find an approach that did not require clone, but give some interesting insights in the Java language. JUnit4 requires the `@BeforeClass` method to be static. This means I cannot have a non-static `DebugElementTest` method return an instance. And, you cannot overwrite a static method! That had never occured to me in the past. `DebugElementTest.newChemObject()` does not overwrite `AbstractChemObjectTest.newChemObject` which is somewhere upstream.

But, after discussing matters with Carl, I ended up with this approach:

chem-bla-ics

```
public abstract class AbstractChemObjectTest extends CDKTestCase {
  private static ITestObjectBuilder builder;
  public static void setTestObjectBuilder(ITestObjectBuilder builder) {
    AbstractChemObjectTest.builder = builder;
  }
  public static IChemObject newChemObject() {
    return AbstractChemObjectTest.builder.newTestObject();
  }
}

public interface ITestObjectBuilder {
  public IChemObject newTestObject();
}

public class DebugAtomTest extends AbstractAtomTest {
  @BeforeClass public static void setUp() {
    setTestObjectBuilder(new ITestObjectBuilder() {
      public IChemObject newTestObject() {
        return new DebugAtom();
      }
    });
  }
}
```