chem-bla-ics

The Art of Programming: do not leak implementation details

Egon Willighagen

Published September 9, 2009

Citation

Willighagen, E. (2009). The Art of Programming: do not leak implementation details. In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/vkj7r-zc905

Keywords

Java, Cdk, Jchempaint

Abstract

At the JChemPaint workshop here in Uppsala, where we have Mark from Chris' group as our guest, we encountered an inconsistency in CDK 1.2, where the bond stereochemistry did not yet follow the pattern recently adopted of having Class fields, to allow using null to have the semantics of undefined. Previously, the defaults for native values were confounded with set values.

At the JChemPaint workshop here in Uppsala, where we have Mark from Chris' group as our guest, we encountered an inconsistency in CDK 1.2, where the bond stereochemistry did not yet follow the pattern recently adopted of having Class fields, to allow using *null* to have the semantics of undefined. Previously, the defaults for native values were confounded with set values. For example, the formal charge *unset* and 0 would be have a field value *int = 0*.

So, I am now writing a patch which replaces the use of *int* in `IBond.getStereo()`. But instead of going for `Integer`, the patch is actually going to use a enumeration.

Now, getting the the Art of Programming... while writing patches in the CDK, you run into those lovely bits of code, where intention is mixed with implementation details. They should not, and often do not need to, but they typically do. This is actually one reason why we now have a more strict peer-review installed. Below are two nice examples where intention is mixed with implementation detail.

Example 1

```
int stereo = container.getBond(chiralNeighbours.get(i), atom).getStereo();
if (stereo == 0) {
  // do something
}
```

This code is bad because we have no clue of what this code is supposed to do. When should the if-clause kick in? Be reminded that the *int = 0* has the confounded meanings of *no stereochemistry* and perhaps *has stereochemistry, but no one ever bothered telling me*. So, which of the two situations does the if clause apply to. So, my patch can only assume that both were applicable (following the actual implementation), though I don't think that makes sense on an algorithmic level. Had the author used `CDKConstants.STEREO_BOND_NONE` (which is the implementation for *int = 0* for *no stereochemistry*, then I had known what the implementation was doing. Instead, the author chose to reuse implementation details: a hardcoded 0.

Example 2

There is another instance of this problem. Look at this lovely piece of code:

```
IBond bond = molecule.getBond(atomA, unplacedAtom);
if (Math.abs(bond.getStereo()) < 2
    && Math.abs(bond.getStereo()) != 0) {
}
```

This example also uses hardcoded value, instead of the matching constants. Remember that *int = 0* had the meaning of no stereochemistry, so I assume this code is determining is stereochemistry is defined for the bond, making nice use that those situations at some point were coded as non-zero values. Moreover, it is only interested in a few stereochemistry definitions, and from the implementation I learn (and that actually makes sense at this

location) that it is only interested in those stereochemistry for which the first bond atom is the stereochemical center. This again is leaking implementation details, instead of using semantically meaningful constants.