chem-bla-ics

CDK-JChemPaint #4: embedding the renderer into a Swing panel

Egon Willighagen

Published April 18, 2010

Citation
Willighagen, E. (n.d.). In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/2148c-n1102

Keywords
Cdk, Jchempaint

Now that we covered the utmost basics of using the CDK-JChemPaint patch (see #1 , #2 , #3), it is time to move on. I am happy to hear that so many people have started using the new rendering architecture, either via the EBI JChemPaint Swing applet/application branch, or via the CDK-JChemPaint patch.

A couple of issues and questions came up (scaling not working as expected; how to layout reactions; how to get charges to show up), and I will look at those shortly. But before I get into those matters, I'll first show how to use the renderer with a Swing JPanel (I'll do the SWT alternative later). First, we need to subclass the JPanel:

```
class JCPPanel extends JPanel {

  IMolecule mol;
  AtomContainerRenderer renderer;
  int width;
  int height;

  public JCPPanel(IMolecule mol, int width, int height) {
    super();
    this.setSize(width, height);
    this.mol = mol;
    this.width = width;
    this.height = height;

    // generators make the image elements
    List generators = new ArrayList();
    generators.add(new BasicSceneGenerator());
    generators.add(new BasicBondGenerator());
    generators.add(new BasicAtomGenerator());

    // the renderer needs to have a toolkit-specific font manager
    renderer = new AtomContainerRenderer(
      generators, new AWTFontManager()
    );
  }

  public Dimension getPreferredSize() {
    return new Dimension(width, height);
  }

  public void paint(Graphics graphics) {
    // the call to 'setup' only needs to be done on the first paint
    renderer.setup(mol, new Rectangle(getWidth(), getHeight()));
```

```
    // paint the background
    graphics.setColor(Color.WHITE);
    graphics.fillRect(0, 0, getWidth(), getHeight());

    // the paint method also needs a toolkit-specific renderer
    renderer.paint(mol, new AWTDrawVisitor(graphics));
  }

}
```

The panel does not implement resizing, and it could consider caching the image too, to speed things up a bit. But, we'll use this as a starting point.

We can then embed this panel into a JFrame to make a small runable application:

```
int WIDTH = 600;
int HEIGHT = 600;

// create molecule
IMolecule triazole = MoleculeFactory.make123Triazole();
StructureDiagramGenerator sdg = new StructureDiagramGenerator();
sdg.setMolecule(triazole);
sdg.generateCoordinates();
triazole = sdg.getMolecule();

// create the frame
JFrame frame = new JFrame("Swinging CDK-JChemPaint");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JCPPanel panel = new JCPPanel(triazole, WIDTH, HEIGHT);
frame.getContentPane().add(panel);

frame.pack();
frame.setVisible(true);
```

The result is pretty much the same as with the created PNG, just with a window. But, this should get you started with using the new code base in your Swing-based application. If you need an impression on where this can get you, have a look at the applet developed by Chris' team. Likewise, a SWT-based application can be developed, of which Bioclipse is a full example. This shows one of the features of this new JChemPaint code base: it is widget set-independent. I am not aware of applications using other widget toolkits yet, though, but I am still hoping someone will use QtJambi to create a Qt-based JChemPaint port.