The binner package

Jonathan.Paul Cook <jcook@sciops.esa.int> 4 October 2018 at 13:43
To: "G. Belanger (ESA)" <gbelanger@sciops.esa.int>

Hi,

I had a bit of a look and I think it would be better if you were to use composition (https://en.wikipedia.org/wiki/Composition_over_inheritance) in your AbstractIntensityBin, e.g) it has a bin and it can have an intensity then you don't need to duplicate the code all the methods. You can still implement the two interfaces and simply delegate/forward the method calls to the intensity or bin reference that this class contains. I think it is cleaner.

So your AbstractIntensityBin would be like this:

```
public abstract class AbstractIntensityBinJC implements IBin,
IIntensity {

    private IIntensity intensity;
    private IBin bin;

    public AbstractIntensityBinJC(IIntensity intensity, IBin bin) {
        this.intensity = intensity;
        this.bin = bin;
    }

    @Override
    public double getValue() {
        return intensity.getValue();
    }

    @Override
    public double getError() {
        return intensity.getError();
    }

    @Override
    public double getVariance() {
        return intensity.getVariance();
    }

    @Override
    public boolean errorIsSet() {
        return intensity.errorIsSet();
    }

    @Override
    public String getUnits() {
        return intensity.getUnits();
    }
```

```
    @Override
    public String getDescription() {
        return intensity.getDescription();
    }

    @Override
    public double[] getEdges() {
        return bin.getEdges();
    }

    @Override
    public double getLeftEdge() {
        return bin.getLeftEdge();
    }

    @Override
    public double getRightEdge() {
        return bin.getRightEdge();
    }

    @Override
    public double getWidth() {
        return bin.getWidth();
    }

    @Override
    public double getCentre() {
        return bin.getCentre();
    }

    @Override
    public boolean contains(double value) {
        return bin.contains(value);
    }

    @Override
    public boolean contains(IBin bin) {
        return bin.contains(bin);
    }

    @Override
    public boolean overlaps(IBin bin) {
        return bin.overlaps(bin);
    }


}
```

But you need to construct the class with the references to an intensity or bin, rather than the values. e.g) change a bit IntensityBinSplitter (I think I did it correctly)

```
   //  Contruct and return the two new bins
   AbsoluteQuantityBin leftBin = new AbsoluteQuantityBin(new
AbsoluteQuantity(whereToSplit), new Bin(thisBin.getLeftEdge(), whereToSplit));
```

AbsoluteQuantityBin rightBin = new AbsoluteQuantityBin(new AbsoluteQuantity(intensityPlus), new Bin(whereToSplit, thisBin.getRightEdge()));

You could pass in the values but then you would have to create the bin and intensity objects internally if you see what I mean.

One other thing I would I suggest to make the code a bit more readable is to use @Overrides on the methods that are implementations in subclasses. Then it is easy to see at a glance which ones are inherited from an interface of abstract class.

Does it make sense and help a bit? I am missing some of the dependencies so can't run the testers.

Thanks
Jon

From: "G. Belanger (ESA)" <gbelanger@sciops.esa.int>
To: "Jonathan.Paul Cook" <jcook@sciops.esa.int>
Sent: Thursday, September 27, 2018 5:22:47 PM
Subject: The binner package

[Quoted text hidden]