Coding Conventions

Table of contents

1. About conventions in JPL

FIXME (nikitin):
CONTENT OF THIS PAGE TO UPDATE !!!!

2. Coding Style

Coding style, formatter and code templates.

```
Writing clean code helps other developers read, understand,
and maintain the code you write. However, not everyone
agrees on the definitions of pretty, nice, or clean.
Different developers possess different styles and aesthetic
sensibilities (reference).
```

3. Eclipse Style

- A new 'JPLStyleProfile.xml' has been created and stored in JPLlib/lib that we will need to discuss format and at the end apply everywhere in the project
- A new 'JPLCodeTemplates.xml'
- A new 'JPLFormatterProfile.xml'
- Share profiles for importation in JPLLib/profile project
- DO NOT FORGET TO CONFIGURE ECLIPSE EDITOR (prop->Java Code ->Style->Clean up Code Template and Formatter) + prop->Java Editor ->Save Actions to define previously set prefs to couple with saving file

To apply an external profile from an XML file to your project, you must import it first. Click Import in the main cleanup preferences, select the file, and click OK.

4. Naming Conventions

We try in the jpl to call methods in a consistency way:

Creating instances:

- with dedicated `Builders` when many optional parameters implement the `build()` method.
- with `Static Factory` methods, when we want to control instanciation with `getInstance()` else `newInstance()`
- with classic constructors (rarely)

In general, we try to follow the naming conventions given by java. The most proper way to do that is to implement interfaces proposed in the core java, propagating the contracts and propagating naming conventions.

> **Note:**
> If it is not possible we try to fit the naming convention of an approching concept.

For example, every `interval` met in jpl follows the same rules defined in core java. The `start` index is inclusive and the `end` index is exclusive (`[begin, end[`).

As another example, `Sequence<T>` is a generalisation of `CharSequence` and as such methods names are pretty close.

```
public interface CharSequence {
  char charAt(int index);
  CharSequence subSequence(int start, int end);
  int length();
}

public interface Sequence<T> {
   T valueAt(int index);
  Sequence<T> subSequence(int start, int end);
  int length();
}
```

> **Note:**
> Each time a concept is develop in a method via the naming mechanism, it has to be propagated everywhere the same concept appears.

```
In older version of jpl, in peaklist object, we had a method named
getPeakNumber(), and even getNumberOfPeak()
else where in a closed object that returned the number of peaks in
the object. As peaklist are basically list of peaks it is more
consistent to name them size() (ref).
```