JPL Todo-List

Table of contents

1. Reimplement fragmentation

Status: Open.

Actually, each fragmenter share a static array of N fragments (N extensible if more fragment needed). When a peptide is fragmented, the fragmenter is setting the current fragment in the array. After the fragmentation is finished, a call to getPeakList([filter]) return a new instance of JPLMSPeakList (subarray fragments[0, last fragment] is sorted first).

The main problem of this implementation may be caused by the intrication calls of many fragmenters. For example, F1 fragments P1, F2 fragments P2, then F1 get a peak list that is coming from P2 ... sic !!

As fragmentation is computed and fragment generated, mzs will be sorted progressively and annotations positions will have to be updated in the same time. I am going to implement an in-place sort (merge sort for instance) that link an array with another one. At the end of the fragmentation, a new instance of peaklist will be created with all fragments (that pass the optional filter). The computation should be quicker and the internal fragments object will be useless and deprecated.

2. Remove class JPLIonizedPeptide

Status: Open.

JPLIonizedPeptide is simply a JPLPeptide with a charge ! I am going to add "charge" into JPLPeptide and get rid off JPLIonizedPeptide.

3. Get peak annotation indices in JPLFragmentAnnotations

Status: Open.

4. Remove peaks from indices in JPLFragmentAnnotations

Status: Open.

5. Handle losses in Fragmentation

Status: Closed - Done.

```
Specific losses pattern depends on various parameters
including the amino acid composition and size of the
peptide, excitation method, time scale of the instrument,
```

```
the charge state of the ion , ...
Paizs and Suhai in Fragmentation pathways of protonated
peptides, 2004.
```

We will first generate loss water/ammonium fragments with really simple general rules (we will not include the type of instrument parameter). Here are the simple rules based on sequence composition or specific sequence patterns:

- amonium loss from the side chains of R, K, N and Q (lots of mechanisms there).
- water loss from the side chains of S and T attacked by the nearest Nt backbone amide oxygens.
- water loss involving dehydratation of the C-terminal (oxazolone pathway).
- water loss from (H-(X)m[DE](X)n-OH and H-E(X)n-OH (pyroglutamate pathway).

Here is more or less the algorithm to include in the fragmenter:

1. Locate the amino-acids able to lose material and the type of losses.
2. Generate every sequences w/wo loss depending on variability of the loss (i.e. if 3 loss sites (fixed) -> 3^2 seqs.), see `JPLPeptideEditorFactory`.
3. All sequences to fragment normally where all peaks go in one peak list.

6. Polymer/Peptide Manager

Status: Closed - done.

If signals or motifs are detected on polymers, the manager can trigger specific operations of edition on the sequence (cutting or adding modifications at detected amino-acid sites). see how to deal with JPLDigester as it is a subtask of this manager. Here are what a manager need:

1. A sequence matcher to find AA target site
2. A modif editor to add modifications on polymer sequence if sites have matched the motif

The manager will handle the combinatory editing for *variable*- typed modifications may be with the help of a Polymer Factory (see JPLPeptideEditorFactory).

7. Handle internal Fragmentation

Status: Open.

8. More methods for JPLMSRenderer

Status: Closed - done.

- Add methods to filter intensity

- Add methods to filter fragment type (a, b, y, prec, ...)
- Add methods to transform intensity
- Possible to choose peak color for fragment types

9. Generic Symbol Sequence and Polymer Generator

Status: Open.

This class will generate sequence from specified parameters following specific distributions. List of parameters to care about: alphabet from which we pick monomer, monomer frequencies, modification frequencies (monomer (in)dependant).

Distributions

- Homogeneous distribution
- Monomer-dependent distribution in a specific alphabet
- Previous state-dependence (Markov chain, bayesian conditional probs).

10. Bio-Sequence Declaration

Status: Closed - canceled.

Extends the language of bioseq to enable support of average accuracy. Every future modification will be converted to the same accuracy.

- in builder through method accuracy()
- in string definition by explicitly surrounding diamonds (like for molecule "CH3" and "<CH3>").

Examples:

```
bioSeq =
  new JPLBioSeqBuilder<JPLAminoAcid>("MQRSTATGCFKL",
    JPLAASymbol.getSymbolType()).accuracy(AVG).build();

bioSeq =
  new JPLBioSeqBuilder<JPLAminoAcid>("<MQRSTATGCFKL>",
    JPLAASymbol.getSymbolType()).build();
```

11. Mass Calculator

Status: Closed - done.

Find a way to get mass accuracy type mass from any symbols with JPLMassCalculator.getAvgAccuracyInstance() or

JPLMassCalculator.getMonoAccuracyInstance().

```
        bioSeq = new JPLBioSeqBuilder<JPLAminoAcid>("MQRSTATGCFKL",
          JPLAASymbol.getSymbolType()).build();
        Assert.assertTrue(MONO_MASS_CALC.getMass(bioSeq) != AVG_MASS_CALC
          .getMass(bioSeq));
```

12. Multiple language (Mixing alphabet and tokens)

Status: Open.

Macromolecules are instanciated from a language defined by a molecular alphabet (symbolism). For example proteins are defined from an alphabet of amino-acids. But then how to define in this sequence other infos aside symbols like modifications ? We propose to handle a multi-language sequence to deal with this problem. Each symbol defining an core element of the macromolecule and each token defining an annotation that apply to the previous symbol. A token can express hierarchical annotation.

Let's take the example of modifications. For example, we want to define tokens for modification then more specific token for different modif format:

```
        ACWRKTS({136.0})O(PO4(3-))R([P])T
        11111112333333321244444421255521
```

Alphabets and Tokens

1. amino-acid (monomer symbol)
2. a modif apply to previous symbol (token)
3. mass value type modif (token)
4. formula molecule type modif (token)
5. amino-acid modif (token)

Here is an example of grammar and tokens:

```
        MODIF := BEGIN_MODIF MODIF_CONTENT END_MODIF;
        BEGIN_MODIF := '(';
        END_MODIF := ')';
        MODIF_CONTENT := MODIF_MASS | MODIF_FORMULA | MODIF_AA;
        MODIF_MASS := BEGIN_MODIF_MASS [0-9]+ END_MODIF_MASS;
        BEGIN_MODIF_MASS := '{'
        END_MODIF_MASS := '}'
        MODIF_FORMULA := grammar(formula, molecule);
        MODIF_AA := ALPHABET_AA;
        ALPHABET_AA := [AB:WY];
```