Recipe 1.1. Creating a custom symbol

Table of contents

1. Problem

You need to create a symbol defined over a custom alphabet.

2. Solution

A `Symbol` is represented as a `character` and is of a specific `type`.

A symbol `type` links the symbol with a specific data type. It defined an alphabet over letters (symbol characters) and a data manager that makes the correct coupling with the data.

In the following, we will take the example of cards. A card is simply evaluated by an integer value that correspond to its rank value.

In other word, each card have a possible rank from deuce ('2') to Ace ('A') and each rank has an associated integer value. A '7' has value 7, 'J' the value 10 and 'A' the value 13. Here is the code that could correspond to this case:

```
SymbolType<Integer> cardRankType =
        new AbstractSymbolType<Integer>("CardRankType",
CardRankSymbol.class,
                "*(23456789TJQKA)") {

                public DataSymbolRegister<Integer> getDataSymbolManager() {
                        return CardRankManager.getInstance();
                }
        };
```

The third parameter is the alphabet itself. Alphabets are defined from a string representing a tree of symbols. In our example about card symbol, the alphabet is really simple as no hierarchy has been defined here.

Nevertheless, it is often needed to define a hierarchical alphabet. For instance, the tree string that define nucleotides could be written `N(R(AG)Y(CT))` where N is the root (any symbol), RY the nodes and ACGT the leaves (the terminal symbols).

This tree representation allows us to handle ambiguous alphabets that we often found in molecular biology.

Let's go back to our cards and define our symbol class. A symbol class have to extends the `AbstractSymbol` abstract class:

```
// a card rank has a integer type data from 2 (rank 1) to Ace (rank 13)
public class CardRankSymbol extends AbstractSymbol<Integer> {

        private static final SymbolType<Integer> TYPE =
                new AbstractSymbolType<Integer>("CardRankType",
CardRankSymbol.class,
                        "*(23456789TJQKA)") {

                        public DataSymbolRegister<Integer>
getDataSymbolManager() {
                                return CardRankManager.getInstance();
                        }
                };

        public CardRankSymbol() {}

        public CardRankSymbol(char name) {
                super(name, TYPE);
        }

        static CardRankSymbol newInstance(char name) {
                return new CardRankSymbol(name);
        }

        public static SymbolType<Integer> getSymbolType() {
                return TYPE;
        }
}
```

Note:

All molecular monomer symbols are built on the same principle. In the JPL, there are symbols for the main monomers (AASymbol, RiboNucSymbol and DeoxyRiboNucSymbol) and respective biological data (AminoAcid, RiboNucleotide and DeoxyRiboNucleotide) that all extends Molecule.

3. Discussion

Each symbol implements the light-weight pattern.

In the JPL, all symbols are instanciated at building time of symbol types. Internally, the alphabet is created on the fly and each symbols are built by reflection.

4. See Also

See also how to create data symbol manager and how to build sequence of symbols.