BioJava

Análise e breve comparação com outras Bio* frameworks

Rui Marques

Bioinformática - MIERSI

Faculdade de Ciências da Universidade do Porto

up030307060@alunos.dcc.fc.up.pt

BioJava

- Criado em 1999 por **Thomas Down e Matthew Pocock** (Wellcome Trust Sanger Institute / Cambridge University).
- É uma ***Application Programming Interface*** (API) para simplificar o desenvolvimento de software de bioinformática usando Java.
- **Objectivos:**
- Facilitar a **reutilização** de código,
- Implementar uma **base genérica** para programas de bioinformática.
- Implementar **algoritmos chave** na bioinformática.

- Faz parte de uma série de Bio* *toolkits*:
  - BioPerl, BioPython e o BioRuby.

Funcionalidades

- Alfabetos de nucleotídeos e amino-ácidos
  - Manipulação de sequências
  - I/O e parsing de ficheiros de sequências
  - Distribuições de probabilidade sobre grupos de sequências
  - Programação dinâmica
  - Algoritmos genéticos
  - Support Vector Machine – classificação e regressão
  - Serialização para bases de dados.
  - Suporte para interfaces gráficos (GUIs)

BioJava API

- As sequências são representadas como **listas de objectos**, não como **strings**.
- Strings têm várias **desvantagens**:
  - **Validação:** é possível passar uma sequência de RNA a uma função que deveria receber como parâmetro uma sequência de DNA.
  - **Ambiguidade:** numa sequência de DNA a letra 'T' significa timina e numa sequência proteica significa treonina.
  - **Propriedades:** não é possível associar facilmente propriedades e anotações a uma letra de uma string.
  - DNA A != RNA A != Protein A
  - Usando strings: “A”.equals(“A”);

Symbols

- No BioJava, um nucleotídeo de DNA corresponde a um objecto (Symbol)
- As instâncias de um Symbol têm:
  - Um **nome** (ex: tiamina).
  - Uma **anotação**, opcional (ex: informação sobre as propriedades químicas).
  - Um método **getMatches**, só relevante para o caso de símbolos ambíguos.
- ProteinTools.a() != DNATools.a();

Métodos Symbol

- Package: **org.biojava.bio.symbol**
- String **getName**() - nome do Symbol (ex. Tiemina)
- Annotation **getAnnotation**() - anotação, opcional (ex: informação sobre as propriedades químicas)
- Alphabet **getMatches**() - usado no caso de símbolos ambiguos

Alphabets

- Conjunto de **objectos do tipo Symbol** que pode ser encontrado em determinada sequência
- Existem alguns **alfabetos *standard.***
- É possível criar **novos alfabetos.**
- Pode ser **infinito**: IntegerAlphabet, DoubleAlphabet.
- Outros são **finitos**: DNA, RNA, PROTEIN...

Métodos Alphabet

- Package: **org.biojava.bio.symbol**
- boolean **contains**(Symbol s) – testar se o alfabeto contem o Symbol 's'.
- List **getAlphabets**() - retorna uma lista ordenada de todos os alfabetos que compôem um alfabeto composto.
- Symbol **getGapSymbol**() - retorna o Symbol que representa o 'gap' neste alfabeto.
- String **getName**() - retorna o nome do alfabeto.
- void **validate**(Symbol s) – lança uma exceção se o Symbol 's' não pertence a este alfabeto.
- ...

Métodos FiniteAlphabet

- Package: **org.biojava.bio.symbol**
- Além dos métodos anteriores:
- void **addSymbol**(Symbol s) – acrescenta um Symbol a este alfabeto.
- Iterator **iterator**() - cria um iterador sobre os simbolos deste alfabeto.
- void **removeSymbol**(Symbol s) – remove um Symbol deste alfabeto.
- int **size**() - o número de simbolos do alfabeto.

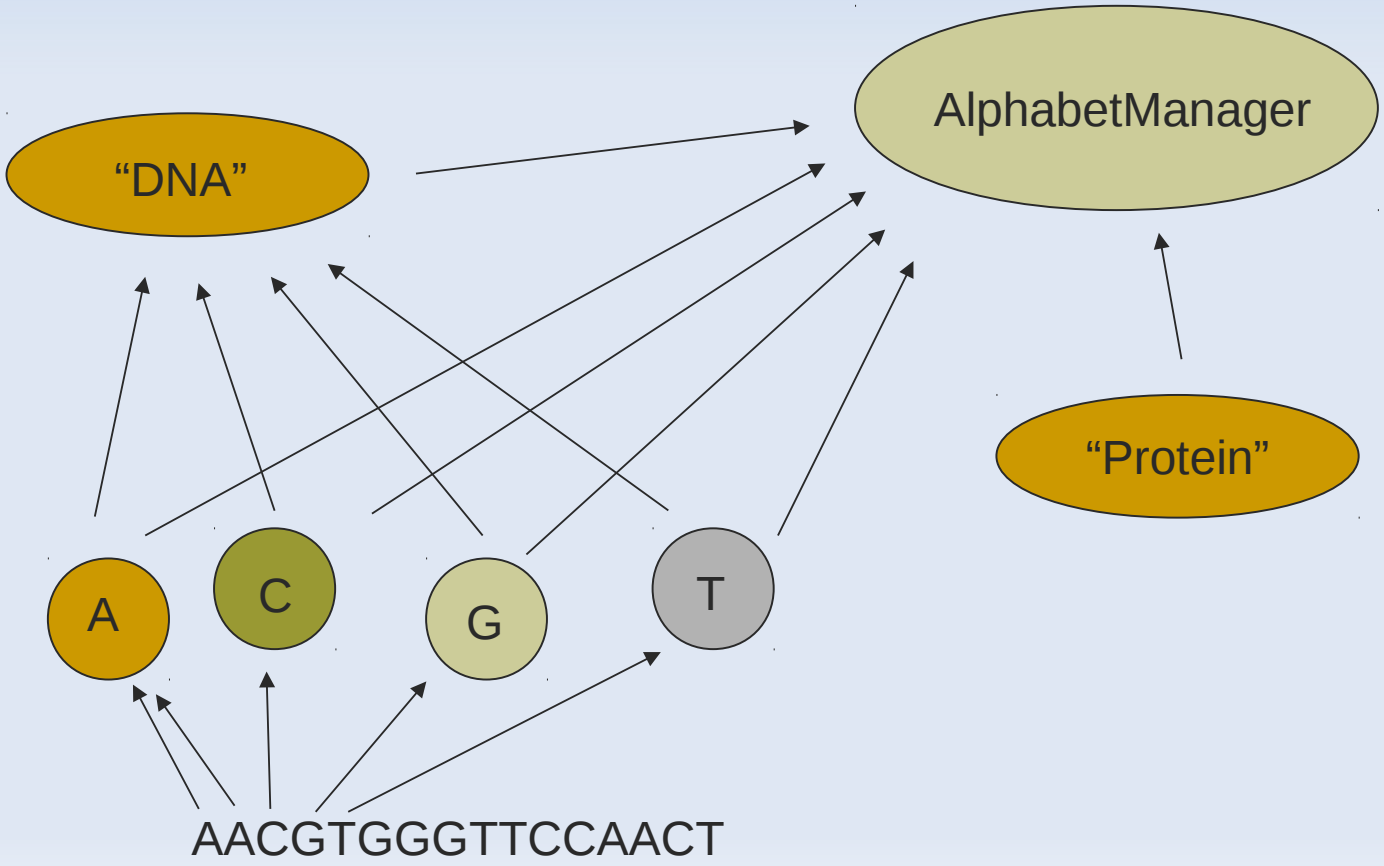
```
// get all the DNA symbols
FiniteAlphabet dna = DNATools.getDNA();
Iterator dnaI = dna.iterator();
while (dnaI.hasNext()) {
    Symbol dnaSymbol = (Symbol) dnaI.next();
    System.out.println(dnaSymbol.getName());
}
```

Modelar Sequências

- O BioJava representa sequências como sendo **listas de Symbol's** (SymbolList).
- Cada SymbolList está **associada a um alfabeto** e só pode conter simbolos desse alfabeto.
- SymbolLists podem ser vistas como strings que são compostas por **objectos Symbol**, em vez de caracteres.

Desperdício de memória?

- Não é na realidade uma lista de objectos.
- É uma lista de referências para objectos Symbol



Métodos SymbolList

- **getAlphabet**() - o alfabeto a que esta lista pertence.
- **iterator**() - um iterador sobre todos os símbolos desta lista.
- **length**() - o número de símbolos da lista.
- **subList**(int start, int end) - retorna uma nova SymbolList contendo os simbolos entre 'start' e 'end', inclusivé.
- **symbolAt**(int index) – retorna o símbolo na posição 'index', contando a partir de 1.
- ...

```
   //create a DNA SymbolList from a String
SymbolList dna = DNATools.createDNA("atcggtcggctta");
//convert dna into a String
String s = dna.seqString();
```

Sequence

- Uma Sequence é uma SymbolList com mais informação, estende-a.
- Para além dos métodos que herda, tem os seguintes:
  - **getName**() - o nome da sequência.
  - **getURN**() - uma URL ou URI que identifica a sequência representada por este objecto.

```
//create a DNA sequence with the name dna_1
Sequence dna = DNATools.createDNASequence("atgctg", "dna_1");
```

SeqIOTools

- Ferramentas para ler e escrever ficheiros, que contêm sequências, formatados nos formatos mais comuns de bioinformática.
- SeqIOTools pode ler e escrever nos formatos:
  - Fasta
  - EMBL
  - GenBank
  - SwissProt
  - GenPept
  - MSF
  - Fasta Alignments
  - ...

Métodos SeqIOTools

- **readEmbl** (java.io.BufferedReader br) – iterar sobre sequências numa stream com o formato EMBL.
- **readFastaDNA** (java.io.BufferedReader br) - iterar sobre sequências DNA numa stream com o formato FASTA.
- **readFastaProtein** (java.io.BufferedReader br) - iterar sobre sequências proteicas numa stream com o formato FASTA.
- **readGenbank** (java.io.BufferedReader br) - iterar sobre sequências DNA numa stream com o formato GenBank.
- **readSwissprot** (java.io.BufferedReader br) - iterar sobre sequências DNA numa stream com o formato Swissprot.
- **writeFasta** (java.io.OutputStream os, Sequence seq) – escrever a sequência 'seq' no output stream 'os'.
- ...

DNATools

- Ferramentas para lidar com sequências de DNA.
- Exemplos de alguns métodos:
  - **a**(): Retorna um AtomicSymbol estático para 'a'.
  - **createDNA**(String dna): Retorna uma nova SymbolList DNA usando a string 'dna'.
  - **getDNA**(): Retorna o alfabeto do DNA.

RNATools

- Conjunto de ferramentas semelhantes ao DNATools mas para RNA.

- Exemplos de alguns métodos:
  - **transcribe**(SymbolList list): Transcreve DNA para RNA.
  - **translate**(SymbolList list): Traduz RNA para proteína.

ProteinTools

- Conjunto de ferramentas para lidar com proteínas.

- Exemplos de alguns métodos:
  - **createProtein**(String prot): Retorna uma nova SymbolLista Protein usando a string 'prot'.
  - **getTAlphabet**(): retorna o alfabeto da proteína incluído os símbolos de terminação da tradução.

Distributions e Counts

- **Distribuições** são usadas, por ex., em programação dinâmica, podem ser treinadas ou podem se definir pesos para os símbolos.
- **Distributions** associam Symbols a frequências.
- **Counts** associam AtomicSymbols a contagens.
- Alguns métodos de Distribution:
  - **Distribution getNullModel**() - retorna o modelo nulo que esta distribuição reconhece.
  - **double getWeight**(Symbol s) – retorna a probabilidade da emissão do símbolo 's' nesta distribuição.

Conclusões e Trabalhos Futuros

Conclusões:

- BioJava é uma *framework* madura que pode ser muito útil.
- Ser baseada em Java é um bom compromisso entre a performance e a legibilidade e robustez do código.
- A documentação pode melhorar (principalmente a da versão 3).
- Reescrever a *framework* na versão 3 tornou-a mais difícil de aprender e pode afastar programadores mais inexperientes.

Trabalho futuro:

- Estudo mais detalhado do BioJava 3 e das outras Bio* *frameworks*.
- Fazer *benchmarks* das versões mais recentes das Bio* *frameworks*

Perguntas?