IUT Sénart-Fontainebleau, UPEC
CPOO
IUT Info 2A

Florent Madelaine

2022-23

Semaine 4

Les notions

1. usage : synopsis et DCU.
2. structure : DC, DO.
3. comportement : DSS et DC.
4. Abstraction : DC avec classes abstraites et interfaces, API, couplage faible.
5. Patron : design patterns (patrons de conception), reconnaître et mettre en oeuvre quelques réponses élégantes apportées à des problèmes récurrents.

Cette semaine nous étudions en détail un premier patron : iterateur.

Il y a deux difficultés.

D'une part, il convient de comprendre comment on code des liste chaînées. C'est quelque chose que vous avez vu en APL l'an dernier mais peut-être pas bien assimilé. C'est l'occasion de bien comprendre comment on fait : il convient d'utiliser une structure récursive.

D'autre part, il y a la subtilité du patron itérateur, pas simple à digérer tant que vous n'avez pas essayé de tout coder.

TD n° 4

Exercice 1. Attention dans cet exercice nous considérons des versions simplifiées des interfaces portant le même nom en java. L'interface `Iterable` propose une méthode `iterator():Iterator`. L'interface `Iterator` propose deux méthodes `hasNext():Boolean` et `next():E`.

Ici le E indique une classe quelconque car ces interfaces sont paramétrée par un objet E (la notation devrait être quelque chose comme `Iterable<E>` et `Iterator<E>`.

Si vous voulez tester votre réalisation en java, vous pouvez toujours ne pas vraiment implémenter les autres méthodes de ces interfaces et renvoyer une exception adaptée.

Proposez une modélisation et une esquisse de code java pour permettre la modélisation de séquences d'ADN (pour des raisons pédagogiques on s'interdira d'utiliser les classes concrètes de java-util comme `ArrayList` etc).

Pour vous aider à réfléchir, considérez par exemple le brin GCTTAG pour lequel vous pouvez esquissez un diagramme objet (pour bien comprendre la structure récursive) puis simulez ce qui se passe en cas d'itération.

> *Les molécules d'ADN des cellules vivantes sont formées de deux brins antiparallèles. [...] Un brin d'ADN est un polymère appelé polynucléotide. Chaque monomère qui le constitue est un nucléotide, lequel est formé d'une base nucléique, ou base azotée — adénine (A), cytosine (C), guanine (G) ou thymine (T) [...] Les bases nucléiques d'un brin d'ADN peuvent interagir avec les bases nucléiques d'un autre brin d'ADN à travers des liaisons hydrogène, qui déterminent des règles d'appariement entre paires de bases : l'adénine et la thymine [... ou] la guanine et la cytosine [...]. Lorsque les séquences des deux brins sont complémentaires, ces brins peuvent s'apparier en formant une structure bicaténaire hélicoïdale caractéristique qu'on appelle double hélice d'ADN. [...] Le génome humain représente environ trois milliards de paires de bases distribués dans 46 chromosomes [...]* Extrait de wikipedia.

TP n° 4

Ce TP est évalué. Il s'agit ici de mettre en oeuvre en java ce qu'on a vu en TD. Le travail est à rendre sur devoir (le lien est accessible depuis le site du département informatique) à l'horaire de votre soutenance.

Vous devez rendre votre travail à la fin du TP.

Vous devez soumettre une archive tar.gz **qui reprend exactement la structure du stub donné**, c'est-à-dire une archive contenant un répertoire par exercice avec uniquement des classes .java dans chaque répertoire.

NB : si tous vos fichiers sont dans un répertoire stub et que vous vous placez dans le répertoire contenant stub, il suffit de faire

```
tar -czvf stubTPADN.tar.gz stub/
```

Pensez à vérifier que votre travail est soumis correctement.

Faites en sorte que votre travail compile, quitte à commenter des choses qui ne compilent pas.

Barème indicatif :

- exo 0 : 4 pts
- exo 1 : 7 pts
- exo 2 : 6 pts
- exo 3 : 4 pts

Un fichier `stub.tar.gz` avec la bonne structure est disponible sur git.
Votre travail doit fonctionner avec mes fichiers `Exemple.java` inchangés.
NB. Pour desarchiver on fait

```
tar -xzvf stubTPADN.tar.gz
```

Exercice 0 (Structure)**.** Mettez en oeuvre la structure de données pour l'ADN vu en TD, c'est-à-dire sans utiliser `java.util`, ni les tableaux natifs de java.
Testez avec le fichier `Exemple.java`.

Exercice 1 (Abstraction)**.** Sans utiliser de classe concrète de `java.util`, ni les tableaux natifs de java, améliorez votre réponse à l'exercice précédent pour que votre classe implémente l'interface `Iterator<Base>`.
Testez avec le fichier `Exemple.java`.
Vous devez traiter **une** question en plus parmi ces questions (voir consigne individuelle donnée en TP noté).

1. Comment faire si on souhaite pouvoir mémoriser certains index intéressants pendant la lecture pour pouvoir y revenir directement plus tard ? Par exemple, pouvoir définir 10 tels « marques-pages » ?
2. Comment faire si on souhaite que trois personnes puissent naviguer indépendamment ?
3. Comment faire si on souhaite que trois personnes puissent naviguer indépendamment ?

Changez votre code pour adresser cette question et proposez des tests adaptés à la fin du fichier `Exemple.java`.

Exercice 2 (Patron)**.** Toujours sans utiliser de classe concrète de `java.util`, ni les tableaux natifs de java, améliorez votre réponse à l'exercice précédent pour dissocier la structuration de la navigation (indication : une classe implémentera `Iterator<Base>`, l'autre `Iterable<Base>`, c'est le patron de conception Iterateur vu en cours).
Testez avec le fichier `Exemple.java`.
Notez que pour un objet l de type `Iterable<Base>`, on peut écrire :

```
l.forEach(b -> System.out.println(b));
```

voir le code suivant qui est presque aussi lisible que du Python.

```
for(Base b: l){
  System.out.println(b);
}
```

Exercice 3 (**Pour aller plus loin**)**.** Reprenez l'exo précédent en changeant la structure de donnée dans la classe implémentant `Iterable<Base>` (et donc il faudra probablement changer aussi des choses dans la classe implémentant `Iterator<Base>` puisque les deux classes sont fortement couplées).
On propose de ne pas utiliser la liste simplement chaînée mais d'utiliser les tableaux natifs de java. On n'aura donc plus de structure récursive : un objet Brin stocke l'ensemble de la structure.
Attention, il faut penser à dimensionner le tableau quand il n'y a plus de place.
Suggestion : Initialement le tableau compte 12 cellules. Il faut connaître la taille actuelle du tableau, et la quantité de cellules utilisées. En cas de besoin on double la taille du tableau. Il faut pouvoir récupérer le tableau sous-jacent d'un Brin.
NB. Pour recopier efficacement un tableau, il y a
`System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`