

Introduction à la Programmation Orientée Objet en JAVA – L2 MIASHS

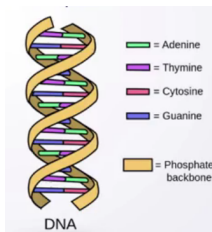
Nicolas HERBAUT

9 Octobre 2018

Les Chaînes de caractère

Notre fil rouge: le code ADN en tant que Chaîne

- Les informations du génomes sont stockées à l'aide d'un alphabet de 4 lettres ATCG (Adenine Thymine Cytosine et Guanine)
TGGGAATAGTACGGGGATTCAATGGTACCAGTATAGC
- Le Génome est composé de paires de ces *nucléotiques* (environ 3.10^9)
- Des algorithmes de recherche de gènes sont indispensable pour exploiter l'information contenue dans notre ADN



Notre fil rouge: le code ADN en tant que Chaîne

ATCGTATACTGAAAACAGCTTTGAGATTGTTAAACACCGAAGAGTTA
TCAGCGTTGAGGTCTAACCCTCAGCGATTATAGATGTGGAGGGTCT
TGTAAGCATACGACGGTCTAGAGCTGGGATGAGGCCCGAA
ACGGTTGTAAGTCTTGAAACGTCTTGAGGGGTGGGCCGCC **Gene**
CAGGCGCGGGGTACCCGTATGCTTATCTTAAGGAGAGCGCGGTGAGAG
GAAAGCCCTGGATTTCATCTTAGCATGCGGGATATCCGAATTTGGAAG
GACAGGAAACAATCTGATATGACCCGTAGATCAACTCTGAACCCCG
CCGAGCGATACCGACTCTACCGGGTGATGCTATCGTTGCGCTCTC
GAGATGATGCTGA **ATG** GAAAACCCACCCATCTCTAAGCGAAC
AATAATGGAACCGGCTACTATTTTCATAGAATGCAACGACGTTTGA
ATGGCGTTCTATCCACTCMAATCTCCGTATACTAGCGTTATCACAGT
ATTAAACGCCAAAAACAAAACGTATATGGCGTTGTAACGCTGCACAT
ACATCGTACAGTGCATCATTCTCCGGGAACCAAGCACAAATGACTACT
TAGCGGGAACGCAGATGTCTATCAGCACACCCGTTTTGATTGAGAG
TAA TACGCAATTTGAGTAATACACCCTTCATGGTAGGGGACATGG
TACTGCAACCCTAGTATCACCTTAGAACGGCTACACACATTCGCACT
TACGCGGCAACTTGTCGACGTTCTTGAGACGCTGTTCGAGTGTTCCCA
CTGGTCGGGACAATTATGACAACGGCAGTCCAGCATCATATGCCGCG
ACATTGGCTCCGTGTACGCGCGGATTGCTAGATCCGGGCA

ONNE

Pourquoi travailler avec les Chaînes (String)

- lettre, chiffre, ponctuation... n'importe quel caractère
- enregistrer une information lisible pour stocker, lire et la traiter

```
>JX477166v1
CGGACACACAAAAAGAAAAAGGTTTTTAAAGATTTTTGTGTGCGAGTAACTATGAGGAAGATTAACAG
TTTTCTCAGTTAAAGGTATACACTGAAATTGAGATTGAGATTCTCTCTTTGCTATTCTGTAACTTCC
CTGGTTGTGACAATTGAATCAGTTTTATCTATTACCAATTACCATCAACATGGTATGTCTAGTGATCTTG
GGACTCTTCTCATCTGGTTTTCTAGAGCTCTGAATCTATTTGTGAGAAGTTCATCCAAACGACCCA

<div class="split-3-layout layout theme-base">
<h2 class="section-heading">
</h2>
<div class="column">
  <article class="story theme-summary

cdatetime,address,district,beat,grid,crimedescr,ucr_ncic_code,latitude,longitude
1/1/06 0:00,2082 EXPEDITION WAY,5,5A,1512,459 PC BURGLARY RESIDENCE,2204,38.47350069,-121.4901858
1/1/06 0:00,4 PALEN CT,2,2A,212,10851(A)VC TAKE VEH W/O OWNER,2404,38.65784584,-121.4621009
1/1/06 0:00,22 BECKFORD CT,6,6C,1443,476 PC PASS FICTICIOUS CHECK,2501,38.50677377,-121.4269508
```

Objectifs

- Comprendre la class String de Java
 - manipuler les idiomes de programmation
 - utiliser les méthodes courantes de String
- En apprendre plus sur les types Java
 - Les types numériques et les opérateurs
- Programmer pour découvrir des patterns et de l'information dans des Strings
 - tous les liens sur une page web?
 - tous les gènes dans un brin d'ADN?

Les Strings avec l'ADN

Travailler avec des Strings représentant de l'ADN

- Chercher des gènes dans des Strings
 - Un exemple réel (très simplifié!)
 - Généralisable à de nombreuses applications (HTML, Email, tout ce qui est textuel)
- D'autres sujets importants
 - Math en Java
 - Pratique de la méthode en 7 étapes!

Concepts sur l'ADN

- 4 lettres ATGC (nucléotides), les briques de l'ADN
- 3 nucléotides = 1 codon
- Début du gène : “start codon” *ATG*
- Fin du gène : “stop codon” *TAA*
- Gène: tous les codons entre un start codon et un stop codon (inclus)

Premier problème

```
ATCGTATACTGAAAACAGCTTTGAGATTGTTAAACACCGAAGAGTTATGATTT
TCAGCGTTGAGGTCTAACCCTCAGCGATTATAGATGTGGAGGGTCTCTTCGC
TGTAAGCATACGACGGTCTAGAGCTGGGATGAGGCCCGAACACTGTTATGGGT
ACGGTTGTAAGTCTTGAAACGTCTTGGAGGGTGGGCCGCCCAAGTACTTGTCC
CAGGCGCGGGGTACCCGTATGCTTATCTTAAGGAGACGCGGTGAGAGTGGTCC
GAAAGCCCTGGATTCATCTTAGCATGCGGGAAATCCGAAGTTGGAAGGTGAGG
```

FIGURE 2 – small_adn.png

- Trouver un gène
 - tout le texte en *ATG* et *TAA*
- Simplification!
 - les vrais gènes ne contiennent que des codons (multiple de 3)
 - il existe d'autres stop codons que *TAA*

Méthode en 7 étapes:

- ① Un exemple à la main
- ② Ecrire ce que l'on a fait
- ③ Généraliser
- ④ Vérifier les modèles à la main
- ⑤ Coder! (mais nous avons besoin de nouveaux concepts en Java)
 - Comment trouver ATG dans une chaîne?
 - Comment représenter les positions dans une chaîne?
 - Comment obtenir toutes les lettres dans un intervalle?
- ⑥ Vérifier par le code
- ⑦ Débugger

Les index en Java

Comment représenter la position de qqchse dans une chaîne?

```
      111111111122  
0123456789012345678901
```

sorbonnepantheonMIASHS

- facile à coder, mais il est mieux d'utiliser une fonction déjà faite
- nous retourne *l'index* des éléments du tableau.

Substring

```
String s = "sorbonnepantheonMIASHS";  
String x = String.substring(4,7);
```

- Le premier indice est inclus

Substring

```
String s = "sorbonnepantheonMIASHS";  
String x = String.substring(4,7);
```

- Le premier indice est inclus
- le deuxième est *exclus*


Substring

```
String s = "sorbonnepantheonMIASHS";  
String x = String.substring(4,7);  
s.substring(4,7)
```

- Le premier indice est inclus
- le deuxième est *exclus*
- peu paraître arbitraire mais $4 - 7 = 3$ est la taille de la chaîne retournée.

Autres méthodes intéressantes de String

```
String s = "sorbonnepantheonMIASHS";
```



méthodes	valeur
s.substring(8,16)	pantheon
s.length()	22
s.indexOf("bonne")	3
s.indexOf("n")	5
s.indexOf("CRI")	-1
s.indexOf("n",7)	10
s.startsWith("sorbonne")	true
s.endsWith("L1")	false

commencement de la recherche sur la chaîne

Où trouver l'info?

- <https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

Live Coding!

- Télécharger et importer dans bluej le fichier `v02_live_code_01.zip`
- On cherche un volontaire pour coder devant nous.

Améliorons notre algorithme pour trouver
plusieurs gènes

Objectifs

- Vous connaissez déjà la class FileResource du précédent TD
- l'utilisation des iterables vous a permis de facilement accéder aux données d'un fichier
- Nous allons utiliser le même principe d'itération dans cette partie.
- Idée: Utiliser le même algorithme que pour trouver 1 gène et répéter.
- Nous allons aussi présenter un nouvel itérable capable de stocker temporairement des informations.

Problème

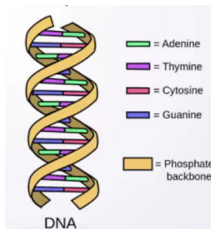
ATCGTATACTGAAAACAGCTTTGAGATTGTTAAACACCGAAGAGTTATGATTT
TCAGCGTTGAGGTCTAACCCTCAGCGATTATAGATGTGGAGGGTCTCTTCGC
TGTAAGCATACGACGGTCTAGAGCTGGGATGAGGCCCGAACACTGTTATGGGT
ACGGTTGTAAGTCTTGAAACGTCTTGAGGGGTGGGCCGCCCAAGTACTTGTCC
CAGGCGCGGGGTACCCGTATGCTTATCTTAAGGAGACGCGGTGAGAGTGGTCC
GAAAGCCCTGGATTCATCTTAGCATGCGGGAAATCCGAAGTTGGAAGGTGAGG
GACAGGAAACAATCTGATATGACCCTGTAGATCAACTCTGAACCCCGACATGT
CCGAGCGATACCGACTCTACACGGGTGATGCATATCGTTGCGCTCTCTTTATA
GAGATGATGCTGA**ATGGAAGAAAACCGCCACCCATCTCTAAGCGAACAGATTC**
AATAATGGAACCGGCCGAACTATTT**CATAGAATGCAACGACGTTTGACAAATA**
ATGGCGTTCTATCCACTCAAATCTCCGTATACTAGCGTTATCACAGTCGCATA
ATTAAACGCCAAAAACAAAACGTATATGGCGTTGTAACGCTGCACATTACCCG
ACATCGTACAGTGCATCATTCTCCGGGAACCAAGCACAATGACTACTAAGCAT
TACCAGGGAACGCAGATGTCTATCAGCACACCCGTTTTGATTGAGAGACAGCT
TAATGTACGCAATTTGAGTAATACACCCTTCATGGTAGGGGACATGGAAGCCA
TACTGCAACCCTAGTATCACCTTAGAACGGCTACACACATTCGCACTTTCTCC
TACGCGGCAACTTGTCGACGTTCTTGAGACGCTGTCGAGTGTTCCCAGCTAGC
CTGGTCGGGACAATTATGACAACGGCAGTCCAGCATCATATGCCGCGAGCCGC
ACATTGGCTCCGTGTCACGCGCGATTGCTAGATCCGGGCA

ONNE

FIGURE 3 – genes2.png

Vous allez apprendre

- l'utilisation d'une boucle while infinie
- l'utilisation de la class ArrayList (ajout de valeurs, comptage, filtrage)



Boucle While

chercher le multiple de trois soit :

distance = indexstopcodon-indexstartcodon

distance%3==0 ==> gene valide

												11	13	15	17	19	21	23					
												v	v	v	v	v	v	v					
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^		^		^		^		^		^		^	
0	1	2	3	4	5	6	7	8	9	10		12	14	16	18	20	22						

Boucle While

												11	13	15	17	19	21	23					
												v	v	v	v	v	v	v					
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10		12	14	16	18	20	22						

- ① on trouve la première occurrence de ATG
- ② on trouve le TAA après ATG
- ③ Vérifie que la distance est un multiplicateur de 3
- ④ ce n'était pas le cas, donc on trouve le prochain TAA

Boucle While

											11	13	15	17	19	21	23						
											V	V	V	V	V	V	V						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- ① on trouve la première occurrence de ATG
- ② on trouve le TAA après ATG
- ③ Vérifie que la distance est un multiplicateur de 3
- ④ ce n'était pas le cas, donc on ~~trouver~~ recherche le prochain TAA
- ⑤ on vérifie que la distance entre les 2 est un multiple de 3
- ⑥ c'est le cas, donc ma réponse est tout ce qui se trouve entre les deux codons

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- ❶ on trouve la première occurrence de ATG
- ❷ on trouve le TAA après ATG
- ❸ **Vérifie que la distance est un multiplicateur de 3**
- ❹ ce n'était pas le cas, donc on trouve le prochain TAA
- ❺ **on vérifie que la distance entre les 2 est un multiple de 3**
- ❻ c'est le cas, donc ma réponse est tout ce qui se trouve entre les deux codons

Boucle While

- Combien de fois vérifier que la distance est bien un multiple de 3?
 - Dans le cas, général, on ne sait pas.
 - A la place on doit écrire un algo qui vérifie autant de fois que nécessaire.
- Les répétitions dans nos algo se transforment en boucle
 - il faut transformer nos répétitions en étapes identiques
 - Trouver sur quel élément boucler
 - on a déjà vu **for**
 - on va voir les boucles **while**

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- nous avons trouvé que la première occurrence d'ATG était à l'**index 0**

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- nous avons trouvé que la première occurrence d'ATG était à l'**index 0**
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à l'**index 8**

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- nous avons trouvé que la première occurrence d'ATG était à l'**index 0**
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à l'**index 8**
- nous avons vérifié si la distance entre eux était un multiple de 3

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- nous avons trouvé que la première occurrence d'ATG était à l'**index 0**
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à l'**index 8**
- nous avons vérifié si la distance entre eux était un multiple de 3
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à l'**index 9**

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- nous avons trouvé que la première occurrence d'ATG était à l'**index 0**
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à l'**index 8**
- nous avons vérifié si la distance entre eux était un multiple de 3
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à l'**index 9**
- nous avons vérifié si la distance entre eux était un multiple de 3

Boucle While

											11	13	15	17	19	21	23						
											v	v	v	v	v	v	v						
A	T	G	A	T	C	G	C	T	A	A	T	G	C	T	T	A	A	G	C	T	A	T	G
^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^
0	1	2	3	4	5	6	7	8	9	10	12	14	16	18	20	22							

- nous avons trouvé que la première occurrence d'ATG était à l'**index 0**
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à l'**index 8**
- nous avons vérifié si la distance entre eux était un multiple de 3
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à l'**index 9**
- nous avons vérifié si la distance entre eux était un multiple de 3
- c'était le cas, donc tout ce qui est entre ceux deux codons est ma réponse

Généralisation

- nous avons trouvé que la première occurrence d'ATG était à **l'index 0**
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à **l'index 8**
- nous avons vérifié si la distance entre eux était un multiple de 3
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à **l'index 9**
- nous avons vérifié si la distance entre eux était un multiple de 3
- c'était le cas, donc tout ce qui est entre ceux deux codons est ma réponse

Généralisation

- trouver la première occurrence d'ATG et l'appeler startIndex
- nous avons trouvé que le "TAA" après l'ATG à **partir de l'index 3** commençait à l'index 8
- nous avons vérifié si la distance entre eux était un multiple de 3
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à l'index 9
- nous avons vérifié si la distance entre eux était un multiple de 3
- c'était le cas, donc tout ce qui est entre ceux deux codons est ma réponse

Généralisation

- trouver la première occurrence d'ATG et l'appeler startIndex
- trouver le TAA commençant après l'index (startIndex + 3), appeler ce résultat currIndex
- nous avons vérifié si la distance entre eux était un multiple de 3
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à l'index 9
- nous avons vérifié si la distance entre eux était un multiple de 3
- c'était le cas, donc tout ce qui est entre ceux deux codons est ma réponse

Généralisation

- trouver la première occurrence d'ATG et l'appeller `startIndex`
- trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- **Vérifier que $(currIndex - startIndex)$ est un multiple de 3**
- ça n'était pas le cas, donc nous avons trouvé le "TAA" suivant commençant à l'index 9
- **Vérifier que $(currIndex - startIndex)$ est un multiple de 3**
- c'était le cas, donc tout ce qui est entre ceux deux codons est ma réponse

Généralisation

- trouver la première occurrence d'ATG et l'appeller `startIndex`
- trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- **Vérifier que $(currIndex - startIndex)$ est un multiple de 3**
- si FAUX, mettre à jour `currIndex` à l'index du prochain "TAA", en commençant à (`currIndex + 1`)
- **Vérifier que $(currIndex - startIndex)$ est un multiple de 3**
- c'était le cas, donc tout ce qui est entre ceux deux codons est ma réponse

Généralisation

- trouver la première occurrence d'ATG et l'appeler `startIndex`
- trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- Vérifier que (`currIndex-startIndex`) est un multiple de 3
- si FAUX, mettre à jour `currIndex` à l'index du prochain "TAA", en commençant à (`currIndex + 1`)
- Vérifier que (`currIndex-startIndex`) est un multiple de 3
- Si VRAI, le texte entre `startIndex` et `currIndex + 3` est la réponse

Généralisation

- trouver la première occurrence d'ATG et l'appeler `startIndex`
- trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- Tant que _____
 - Vérifier que $(currIndex - startIndex)$ est un multiple de 3
 - Si VRAI, le texte entre `startIndex` et `currIndex + 3` est la réponse
 - Si FAUX, mettre à jour `currIndex` à l'index du prochain "TAA", en commençant à $(currIndex + 1)$

Généralisation (enfin)

- trouver la première occurrence d'ATG et l'appeler `startIndex`
- trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- Tant que **`currIndex != -1`**
 - Vérifier que $(currIndex - startIndex)$ est un multiple de 3
 - Si VRAI, le texte entre `startIndex` et `currIndex + 3` est la réponse
 - Si FAUX, mettre à jour *currIndex* à l'index du prochain "TAA", en commençant à $(currIndex + 1)$
- **Votre réponse est une chaîne vide**

Syntaxe de while

Tant que ...

```
while( x < y ){  
    System.out.println(x);  
    x=x+3;  
}
```

- Si la condition est vraie, entrer dans le corps de la boucle
 - exécuter les instructions
- A la fin de la boucle, réévaluer la condition et recommencer.

Live Coding!

- Télécharger et importer le fichier `v02_live_code_02.zip` dans bluej
- On cherche un volontaire pour coder en live.

3 codons de fin

- on rajoute ici une nouvelle couche de complexité: la multiplicité des codons de fin
- TAA, TGA, TAG



FIGURE 4 – Quel codon choisir?

- On veut celui qui arrive en premier (ici TAG)

3 codons de fin: résolution en repartant de l'algo d'avant

- ❶ trouver la première occurrence d'ATG et l'appeler `startIndex`
- ❷ si `startIndex = -1` retourne la chaîne vide.
- ❸ trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- ❹ Tant que `currIndex \neq -1`
 - ❺ Vérifier que $(currIndex - startIndex)$ est un multiple de 3
 - ❻ Si VRAI, le texte entre `startIndex` et `currIndex + 3` est la réponse
 - ❼ Si FAUX, mettre à jour `currIndex` à l'index du prochain "TAA", en commençant à $(currIndex + 1)$
- ❺ Votre réponse est une chaîne vide

3 codons de fin: résolution en repartant de l'algo d'avant

- ❶ trouver la première occurrence d'ATG et l'appeler `startIndex`
- ❷ si `startIndex = -1` retourne la chaîne vide.
- ❸ trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- ❹ Tant que `currIndex \neq -1`
 - ❺ Vérifier que $(currIndex - startIndex)$ est un multiple de 3
 - ❻ Si VRAI, le texte entre `startIndex` et `currIndex + 3` est la réponse
 - ❼ Si FAUX, mettre à jour `currIndex` à l'index du prochain "TAA", en commençant à $(currIndex + 1)$
- ❺ Votre réponse est une chaîne vide

Peut-on réutiliser cet algo?

Nous allons découper le problème et abstraire la partie qui cherche le codon de fin dans sa propre méthode

Abstraction de la recherche de codon de fin

- 1 trouver la première occurrence d'ATG et l'appeler `startIndex`
- 2 si `startIndex = -1` retourne la chaîne vide.

`findStopCodon(dnaStr, startIndex, codon)`

- trouver le TAA commençant après l'index (`startIndex + 3`), appeler ce résultat `currIndex`
- Tant que `currIndex \neq -1`
- Vérifier que `(currIndex - startIndex)` est un multiple de 3
- Si VRAI, le texte entre `startIndex` et `currIndex + 3` est la réponse
- Si FAUX, mettre à jour `currIndex` à l'index du prochain "TAA", en commençant à `(currIndex + 1)`
- Votre réponse est une chaîne vide

Abstraction de la recherche de codon de fin

- 1 trouver la première occurrence d'ATG et l'appeler startIndex
- 2 si startIndex = -1 retourne la chaîne vide.
- 3 findStopCodon(dnaStr,startIndex,"TAA") et appeler le résultat taalIndex
- 4 findStopCodon(dnaStr,startIndex,"TAG") et appeler le résultat tagIndex
- 5 findStopCodon(dnaStr,startIndex,"TGA") et appeler le résultat tgalIndex
- 6 prendre le plus petit entre taalIndex, tagIndex, tgalIndex et l'appeler minIndex
- 7 La réponse est le texte commençant à startIndex jusqu'à minIndex + 3

findStopCodon(dnaStr, startIndex, stopCodon)

- ❶ trouver le TAA commençant après l'index ($\text{startIndex} + 3$), appeler ce résultat *currIndex*
- ❷ Tant que *currIndex* $\neq -1$
 - ❸ Vérifier que $(\text{currIndex} - \text{startIndex})$ est un multiple de 3
 - ❹ Si VRAI, le texte entre startIndex et $\text{currIndex} + 3$ est la réponse
 - ❺ Si FAUX, mettre à jour *currIndex* à l'index du prochain TAA), en commençant à $(\text{currIndex} + 1)$
- ❸ Votre réponse est une chaîne vide

findStopCodon(dnaStr, startIndex, stopCodon)

- ❶ trouver le **stopCodon** commençant après l'index ($\text{startIndex} + 3$), appeler ce résultat *currIndex*
- ❷ Tant que *currIndex* $\neq -1$
 - ❸ Vérifier que $(\text{currIndex} - \text{startIndex})$ est un multiple de 3
 - ❹ Si VRAI, *currIndex* est la réponse
 - ❺ Si FAUX, mettre à jour *currIndex* à l'index du prochain **stopCodon**, en commençant à $(\text{currIndex} + 1)$
- ❸ retourner `dnaStr.length()`

Live Coding

(on cherche un volontaire pour venir coder la fonction)

Algorithme Final

- ① trouver la première occurrence d'ATG et l'appeler startIndex
- ② si startIndex = -1 retourne la chaîne vide.
- ③ findStopCodon(dnaStr, startIndex, "TAA") et appeler le résultat taalIndex
- ④ findStopCodon(dnaStr, startIndex, "TAG") et appeler le résultat tagIndex
- ⑤ findStopCodon(dnaStr, startIndex, "TGA") et appeler le résultat tgaIndex
- ⑥ prendre le plus petit entre taalIndex, tagIndex, tgaIndex et l'appeler minIndex
- ⑦ si minIndex = startIndex alors la réponse est une chaîne vide \$
- ⑧ La réponse est le texte commençant à startIndex jusqu'à minIndex + 3

Trouver tous les Gènes!

Nous souhaitons afficher tous les gènes

- les brins d'ADN contiennent de nombreux gènes
 - on veut tous les afficher (1 par ligne)
- On peut déjà trouver le premier gène
 - on doit légèrement changer la position de début de recherche
- Nous sommes habitués aux boucles maintenant
 - tant qu'il y a encore des gènes... → boucle while
- Difficulté, on ne sais pas

Algorithme

- Difficile? travailler les étapes 1,2 et 3 à la maison

CGATGATCGCATGATTTCATGCTTAAATAAAGCTCA

FIGURE 5 – break

- ① prendre startIndex=0
- ② tant qu'il y a encore des gènes après startIndex
- ③ Trouver le gène après startIndex
- ④ Afficher le gène
- ⑤ startIndex \leftarrow la fin du gène.

Algorithme 2

- ① prendre `startIndex=0`
- ② **répéter les étapes suivantes**
- ③ Trouver le gène après `startIndex`
- ④ **si aucun gène n'est trouvé, quitter la boucle**
- ⑤ Afficher le gène
- ⑥ `startIndex` \leftarrow la fin du gène.

Algorithme 2

répéter les étapes suivantes

```
while(true){ ... }
```

si aucun gène n'est trouvé, quitter la boucle

```
while(...){  
    ...  
    break; //quitte la boucle et sort des {...}  
}
```

Lorsque Break ne suffit plus

- Il est également possible de stopper l'itération en cours et de reprendre au début de la boucle.
- Pour cela on utilise le mot clé **continue**

```
while(...){  
    ...  
    continue; // ignore toutes les instructions  
              // suivantes dans la boucle et  
              // reprend au début de la boucle  
}
```

Stocker les Chaînes de caractères dans une liste

- (Documentation

ArrayList)[<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>]

- Création d'une liste.

```
java.util.ArrayList<String> list = java.util.ArrayList<String>();
```

- Ici on spécifie que la liste va convenir des Chaînes de caractères

Operations sur les listes

```
java.util.ArrayList<String> list = new java.util.ArrayList<String>();
```

méthodes	valeur
list.add("elt1")	void
list.add("elt2")	void
list.get(0)	"elt1"
list.get(1)	"elt2"
list.get(1)	"elt2"
list.size()	2

ATG/ ... / TAA

Iteration sur les Listes

```
java.util.ArrayList<String> list = new java.util.ArrayList<String>();  
list.add("elt1");  
list.add("elt2");  
for ( String elt : list){  
    System.out.println(elt)  
}
```

```
elt1  
elt2
```

hasGene :

on va appeller la fonction getAllgene () parcours la liste des éléments

// comme il y a pas de starcodon et de end sur le agene on peut pas utiliser equal on utilise donc :
.containe(ogene)