Exercices d'algorithmique et Java
1ère Informatique IPL

Matière
- Tableaux non triés à une dimension de taille fixe

Objectifs
- Pouvoir écrire une boucle *for* qui permet de parcourir tous les éléments d'un tableau
- Pouvoir écrire des boucles avec sortie prématurée de parcours de table
- Pouvoir écrire des méthodes classiques telles que des recherches de min, des calculs de moyenne, des vérifications d'existence tout en se plongeant dans un contexte.
- Pouvoir écrire des méthodes simples mais moins classiques car vraiment liées à un contexte
- Ecrire des premières méthodes qui font intervenir 2 tables

Attention : les tableaux que vous allez manipuler sont maintenant des attributs de classe.

Avertissement
Bien souvent les professeurs fournissent des classes de tests. Il faut pouvoir tirer des conclusions de ces tests et corriger les méthodes en conséquence.
Attention ces classes reprennent de nombreux tests, mais ne peuvent pas tout prévoir.
Ce n'est pas parce que tous les tests ont réussi que votre méthode est correcte !

Exercices obligatoires

.

A Méthodes simples dans une table de taille fixe qui contient des variables de type primitif

A1 Classe *Temperatures*

Vous allez écrire une application qui permet de tirer quelques statistiques sur les températures d'un mois.

D'une part, vous allez compléter la classe *Temperatures*.
Cette classe contient comme attribut une table des températures d'un mois donné.
Elle possède les méthodes `moyenne()`, `temperatureMin()`, `temperatureMax()`, …

D'autre part, vous allez compléter la classe *StatistiquesTemperatures*.
Celle-ci, après avoir chargé les températures d'un mois, va présenter le menu :
1 : afficher toutes les températures
2 : moyenne
3 : température la plus haute
4 : température la plus basse
…

Remarques :

1) Procédez à une statistique à la fois !
Passez, dans un premier temps les méthodes supplémentaires.
Pour chaque statistique :
Vous écrivez la méthode correspondante dans la classe *Temperatures*.
Vous ajoutez cette statistique dans le menu.
Vous testez !

2) La méthode `chargerTemperatures()` devrait s'occuper de l'encodage des températures.
Pour faciliter les tests, des données ont été hard codées.

Voici les statistiques attendues pour celles-ci :

Moyenne = 1.032258064516129
Min = -3
Nombre jours de gel : 7
Jours de gel : 1, 5, 6, 7, 8, 10, 25
Toutes positives : `false`
Au moins une négative : `true`
Au moins une supérieure à 5 : `false`, au moins une supérieure à 0 : `true`
Max = 5
Jours max : 15, 21, 31
Jours min : 6, 7

3) Pour mieux tester les méthodes qui renvoient un booléen, modifiez la table testée :

```
double[]tableTemperatures = {3,5,6} ;
```

Toutes positives : `true`
Au moins une négative : `false`
Au moins une supérieure à 5 : `true`

4) Pour afficher le contenu d'une table et non son adresse :

```
System.out.println(Arrays.toString(table));
```

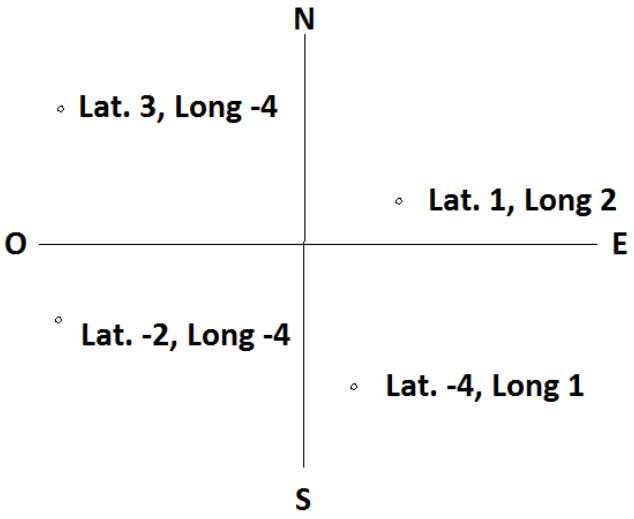
B Méthodes simples dans une table de taille fixe qui contient des objets

B1 classe *Vol*

Contexte :

Tommy est un passionné de parapente. Chaque fois qu'il effectue un vol, il emporte un gps. Cet appareil enregistre à intervalles de temps réguliers sa position. Pendant toute la durée de son vol, son parcours est ainsi mémorisé sous forme d'une suite de coordonnées gps.
Une fois à la maison, Tommy, après transfert de toutes les données du gps sur son ordinateur, pourra les analyser.

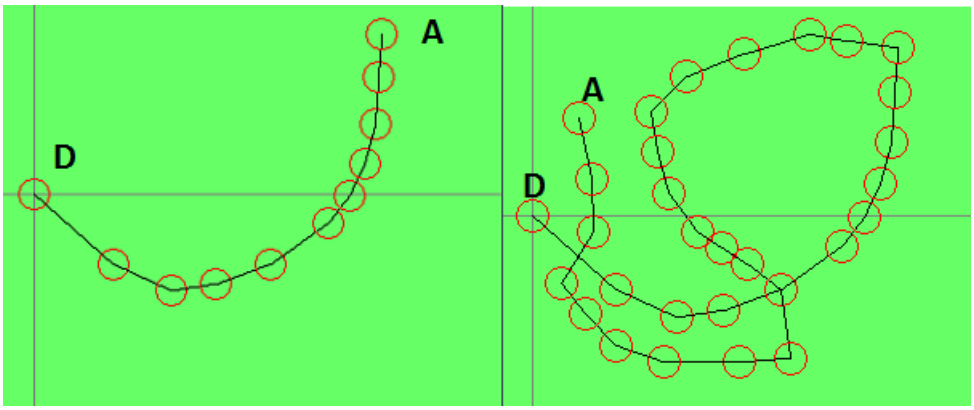
Exemples de coordonnées gps (système simplifié)



Exemples de vols

Dans les schémas suivants, chaque point représente un lieu qui a été survolé et mémorisé par le gps. Le point D représente le point de départ. Le point A, le point d'arrivée.

Vol 1 : Vol 2 :



La classe *Coordonnees*

Cette classe vous est fournie, ne la modifiez pas.
Chaque coordonnée possède une latitude et une longitude
Cette classe contient les méthodes `distance()`, `equals()` et `segmentsCroises()`.
(La méthode `segmentsCroises()` va uniquement servir pour un défi.)

La classe *Vol*

Cette classe contient un tableau de coordonnées gps.
Dans un premier temps, ne faites pas les méthodes mises comme exercice supplémentaire ou en défi.
Suivez bien la *JavaDoc* et les indications données.
Ne modifiez pas les en-têtes des méthodes car elles seront testées par le programme de tests.
Vous pouvez ajouter de nouvelles méthodes si nécessaire.

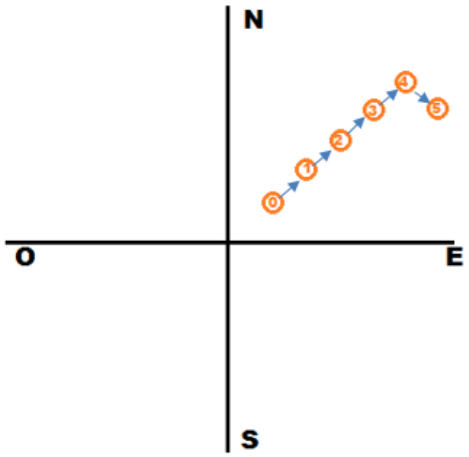
La classe *TraitementVol*

Cette classe vous est fournie. Elle va vous permettre de faire vos tests.
Le programme demande à l'utilisateur de choisir un vol.
Ensuite via menu, il propose les 10 fonctionnalités suivantes :

```
Traitement d'un vol
*******************

1) afficher le lieu d'arrivee
2) verifier si le lieu de depart correspond au lieu d'arrivee
3) afficher le lieu survole apres n intervalles de temps
4) afficher le lieu survole le plus au sud
5) afficher le lieu survole le plus eloigne (a vol d'oiseau) du lieu de depart
6) verifier si le parapentiste a atteint une cible donnee
7) afficher le nombre de cibles atteintes
8) afficher la distance parcourue
9) verifier si le parapentiste a croise son propre parcours
10) verifier si le parapentiste a suivi un parcours impose
```

Pour faciliter les tests, les données de 3 vols ont été hard codées.
Voici ces vols et les résultats attendus :

Vol1 :

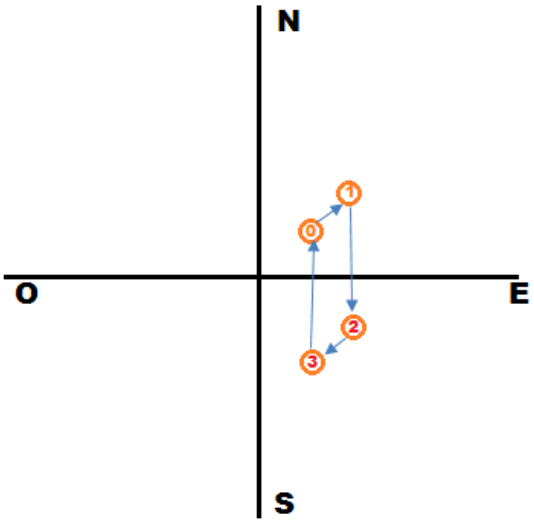


0	1	2	3	4	5
(1,1)	(2,2)	(3,3)	(4,4)	(5,5)	(4,6)

```
Votre choix : 1
le lieu d'arrivee :
Latitude : 4 Longitude : 6

Votre choix : 2
le parapentiste n'est pas revenu a son point de depart

Votre choix : 3
Entrez n : 0
Apres 0 unites de temps, le lieu survole :
Latitude : 1 Longitude : 1

Votre choix : 3
Entrez n : 2
Apres 2 unites de temps, le lieu survole :
Latitude : 3 Longitude : 3

Votre choix : 3
Entrez n : 5
Apres 5 unites de temps, le lieu survole :
Latitude : 4 Longitude : 6

Votre choix : 3
Entrez n : 6
vol trop court

Votre choix : 4
le lieu le plus au sud :
Latitude : 1 Longitude : 1

Votre choix : 5
le lieu le plus eloigne :
Latitude : 4 Longitude : 6

Votre choix : 6
Entrez la cible :
```

```
Entrez la latitude : 3
Entrez la longitude : 3
la cible a ete atteinte

Votre choix : 6
Entrez la cible :
Entrez la latitude : 9
Entrez la longitude : 9
la cible n'a pas ete atteinte

Votre choix : 7
Entrez le nombre de lieux : 3
Coordonnees n°1
Entrez la latitude : 4
Entrez la longitude : 6
Coordonnees n°2
Entrez la latitude : 4
Entrez la longitude : 4
Coordonnees n°3
Entrez la latitude : 9
Entrez la longitude : 9
nombre de cibles atteintes : 2

Votre choix : 8
la distance du vol : 0.22

Votre choix : 9
le parapentiste n'a jamais croise son propre parcours

Votre choix : 10
Entrez le nombre de lieux : 3
Coordonnees n°1
Entrez la latitude : 2
Entrez la longitude : 2
Coordonnees n°2
Entrez la latitude : 4
Entrez la longitude : 6
Coordonnees n°3
Entrez la latitude : 5
Entrez la longitude : 5
le parcours n'a pas ete suivi

Votre choix : 10
Entrez le nombre de lieux : 3
Coordonnees n°1
Entrez la latitude : 2
Entrez la longitude : 2
Coordonnees n°2
Entrez la latitude : 4
Entrez la longitude : 4
Coordonnees n°3
Entrez la latitude : 4
Entrez la longitude : 6
le parcours a ete suivi
```

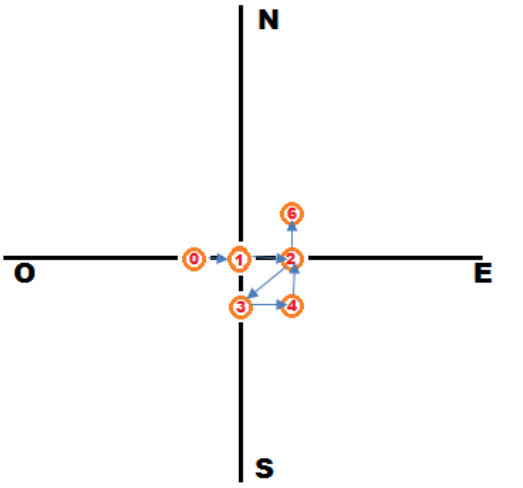
Vol2



0	1	2	3	4
(1,1)	(2,2)	(-1,2)	(-2,1)	(1,1)

```
Votre choix : 1
le lieu d'arrivee :
Latitude : 1 Longitude : 1

Votre choix : 2
le parapentiste est revenu a son point de depart

Votre choix : 4
le lieu le plus au sud :
Latitude : -2      Longitude : 1

Votre choix : 5
le lieu le plus eloigne :
Latitude : -2      Longitude : 1

Votre choix : 8
la distance du vol : 0.27

Votre choix : 9
le parapentiste a croise son propre parcours
```

Vol3



0	1	2	3	4	5	6
(0,-1)	(0,0)	(0,1)	(-1,0)	(-1,1)	(0,1)	(1,1)

```
Votre choix : 1
le lieu d'arrivee :
Latitude : 1 Longitude : 1

Votre choix : 2
le parapentiste n'est pas revenu a son point de depart

Votre choix : 4
le lieu le plus au sud :
Latitude : -1      Longitude : 0

Votre choix : 5
le lieu le plus eloigne :
Latitude : -1      Longitude : 1
Votre choix : 8
la distance du vol : 0.2

Votre choix : 9
le parapentiste a croise son propre parcours
```

Exercices supplémentaires

A1 Complétez les méthodes mises en exercices supplémentaires de la classe *Temperature*.

A2 Classe *Etudiant*

a) Complétez la classe *Etudiant*.

Vous avez à votre disposition 3 classes de tests.
Elles proposent toutes les mêmes tests, mais se déclinent de différentes façons.
Ces 3 versions vous donnent une bonne panoplie des différentes classes de tests qui vous seront proposées pendant tout le Bloc1.

TestEtudiant1 : Cette classe s'écrit très vite mais l'utilisateur doit vérifier les résultats affichés. C'est très pénible. Il s'agit souvent de classe de tests que le programmeur complète au fur et à mesure de sa programmation. Ce sont de telles classes de tests que vous écrivez aux cours d'APOO.

TestEtudiant2 : Cette classe s'écrit très vite. Au premier test échoué, le programme s'arrête.
L'utilisateur reçoit un message d'erreur, le résultat attendu et le résultat fourni par sa méthode.
Attention, pour tester une méthode, il faut donc nécessairement que toutes les méthodes testées précédemment fonctionnent.

TestEtudiant3 : Cette classe est agréable à utiliser. On peut tester les méthodes dans n'importe quel ordre. Par contre, elle est assez longue à écrire.

b) Ajoutez la méthode `nombreUEValidees()`.

Seules les UEs dont la cote est >=10 sont validées.
Cependant, lors de la délibération, le jury peut décider de valider des UEs en échec.
Voici un critère « inventé» pour cet exercice :
Un échec sera validé si l'étudiant a une moyenne supérieure ou égale à 12, que cet échec est unique et qu'il n'est pas inférieur à 9.

Pour tester cette méthode, il vous faudra d'abord compléter une classe de tests.
Complétez la classe *TestEtudiant3*.

A3 Classe *Cryptographie*

Les services secrets de sa très gracieuse majesté vous ont engagé pour réaliser une classe Java de message crypté.
a) Votre première mission est de prendre la classe *MessageCrypte* et d'implémentez la méthode `decalerCaracteres()` qui est décrite dans sa *Javadoc*.
Ensuite exécutez la classe *TestMessageCrypte* pour connaître vos prochaines missions !

B1 Complétez les méthodes mises comme exercices supplémentaires de la classe *Vol*.

Exercices défis

A3 Cryptographie : améliorations

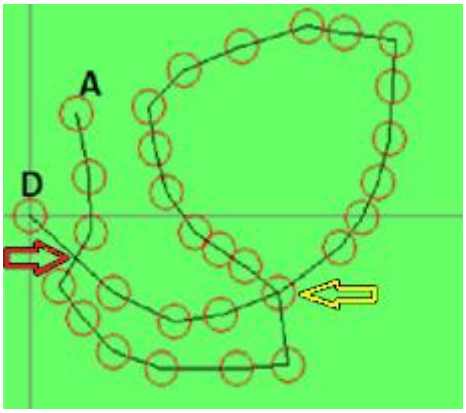
a) Après la lettre ‘a’ se trouve la lettre ‘b’, après la lettre ‘b’ se trouve la lettre ‘c’, …
MAIS après la lettre ‘z’, on ne trouve pas la lettre ‘a’ ! Pour que la méthode `decalerCaracteres()` fonctionne, il faudrait que l’alphabet soit circulaire.
Remédiez à ce problème.

b) Implémentez la méthode `substitution()` de la classe *MessageCrypte*. Ecrivez des tests pour cette méthode

c) La méthode `substitution()` crypte le message. Comment pourrait-on le décrypter ? Implémentez cette méthode de décryptage.

B1 Complétez les méthodes `aSurvoleUnMemeLieu()` et `parcoursSuivi()` de la classe *Vol*.

B1 Il y a croisement lorsque 2 lieux enregistrés sont les mêmes (flèche jaune), mais aussi lorsque 2 segments s’entrecroisent (flèche rouge).



Ecrivez la méthode *aCroiseSonParcours()*.
Il vous faudra utiliser la méthode *segmentsCroises()* de la classe *Coordonnee*.
Pour tester cette méthode, ajoutez, par exemple, le lieu (2,1) à la fin du vol1 et appelez cette méthode pour le choix 9 de la classe *TraitementVol*.