

AUCUN DOCUMENT AUTORISÉ

Il est très fortement recommandé de soigner la présentation de la copie et la qualité des explications fournies.

Algorithme de correction : Illisible => Pas lu ; Pas expliqué => Pas compris ; A peu près = 0

Questions de cours (9') : Question de cours

- 1) Yaml et Marshall, à quoi ça sert ? Donnez un exemple d'utilisation.
- 2) Qu'est-ce qu'un mixin ? Donnez un exemple.
- 3) Expliquez la différence entre polymorphisme adhoc et polymorphisme d'héritage.
- 4) Citez une autre forme de polymorphisme non utilisée par Ruby
- 5) En Ruby les méthodes renvoient le résultat de l'évaluation de la dernière instruction. Quel problème cela pose t'il ? quelle solution proposez-vous ?
- 6) La méthode initialize est-elle nécessaire dans une classe abstraite ? Expliquez
- 7) Rappeler la convention utilisée en ruby pour les méthodes destructives.
- 8) Expliquer le fonctionnement de la méthode inject du module Enumerable. Donnez un exemple.

Exercice 1 (6') : Réveil Musculaire

Soit la classe suivante

```
class LesVoyelles
  @@voyelles = ["a", "e", "i", "o", "u", "y"]
end
```

- 1) Quel est (approximativement) le message d'erreur résultant de l'exécution de ce programme :


```
x = LesVoyelles.new
x.each { |v| print v}
```
- 2) Corrigez la classe LesVoyelles, pour que l'affichage du programme soit : yuoiea
- 3) Quel est (approximativement) le message d'erreur résultant de l'exécution de ce programme :


```
x = LesVoyelles.new
print x.collect { |i| i + "x" }
```
- 4) Corrigez la classe LesVoyelles, pour que l'affichage soit le suivant :


```
["yx", "ux", "ox", "ix", "ex", "ax"]
```

Exercice 3 (30') : Gymnastique

- 1) Ajouter à la classe Array la méthode pasQuUn qui retourne les éléments présents plusieurs fois.


```
[[1, 2], [3, [4, 5]], 6, [1, 2], 6, [4, 5], [6], 6].pasQuUn ➔ [[1, 2], 6]
```
- 2) Ajouter à la classe Array la méthode aplatir ramenant les éléments au même niveau.


```
[1, 2, 3, [4, [5, 6], [7]]].aplatir ➔ [1, 2, 3, 4, 5, 6, 7]
```
- 3) La méthode to_a de la classe Hash permet de convertir une table de Hashage en tableau (Array). Ecrire l'opération inverse to_h de la classe Array, qui convertit un tableau en table de hashage.


```
[['A', 'a'], ['B', 'b'], ['C', 'c']].to_h ➔ {"A"=>"a", "B"=>"b", "C"=>"c"}
```

- 4) On parle d'itérateur externe lorsque l'itérateur est un objet séparé du conteneur de collection.
Ecrire la classe ArrayIterator qui implémente un itérateur externe (« à la Java ») pour la classe Array. Cette classe est dotée des méthodes classiques des itérateurs externes : has_next?(), next_item().

On parle d'itérateur interne quand l'itérateur est « intégré » au conteneur.

Ecrire l'itérateur interne pourChacunDesElements() de la classe Array.

Exemple d'utilisation et d'exécution des itérateurs interne et externe

```

* TestArrayIterator.rb
1 # Un programme de test
2 # pour les Iterateurs Externe et Interne
3
4 load "ArrayIterator.rb"
5
6 tableau=['rouge', 'vert', 'bleu']
7 i=ArrayIterator.new(tableau)
8 puts("# Iterateur Externe : Résultat de l'exécution ")
9 while (i.has_next?)
10   puts("#{\t}item : #{i.next_item()}")
11 end
12
13 load "ArrayIteratorInterne.rb"
14
15 tableau=['rouge', 'vert', 'bleu']
16 puts("\n# Iterateur Interne : Résultat de l'exécution ")
17 tableau.pourChacunDesElements() { |item|
18   puts("#{\t}item : #{item}"}
19 }

Running “TestArrayIterator.rb”...
# Iterateur Externe : Résultat de l'exécution
#     item : rouge
#     item : vert
#     item : bleu

# Iterateur Interne : Résultat de l'exécution
#     item : rouge
#     item : vert
#     item : bleu

Program exited with code #0 after 0.14 seconds.

```

Exercice 4 (30') : TDA Ensemble

Un ensemble est un type de données permettant de représenter une collection de valeurs de même type.
 On appelle élément chacune de ces valeurs.

On rappelle les propriétés principales d'un ensemble :

- Un ensemble est non ordonné (la place des éléments n'a pas d'importance) et un élément ne peut apparaître qu'une fois.
 - La suppression d'un élément n'appartenant pas à l'ensemble est sans effet.
 - Un ensemble peut être vide. La cardinalité d'un ensemble est le nombre de ses éléments. Une valeur appartient à un ensemble si elle fait partie de ses éléments.
 - Un ensemble A est inclus dans un ensemble B si tous les éléments de A appartiennent à B, deux ensembles A et B sont égaux s'ils contiennent les mêmes éléments.
 - La Classe Ensemble fournit une méthode permettant d'obtenir l'ensemble union de deux ensembles passés en paramètres. La Classe Ensemble fournit aussi une méthode permettant d'obtenir l'ensemble intersection de deux ensembles passés en paramètres.
- Donnez en Ruby une réalisation du TDA Ensemble en expliquant la structure de donnée concrète retenue.
1. Ecrire les méthodes de création/initialisation d'ensembles
 2. Ecrire les méthodes permettant l'ajout et la suppression d'élément dans un ensemble, ainsi que le calcul de la cardinalité
 3. Ecrire les méthodes de test (égalité d'ensemble, inclusion d'ensemble, appartenance d'un élément, ensemble vide)
 4. Ecrire les méthodes d'union et d'intersection de la classe Ensemble
 5. L'ensemble contenant les éléments A,B,C,D,E doit s'afficher de la façon suivante : (A, B, C, D, E). Ecrire le code correspondant.
 6. Ecrire un programme de test créant au minimum deux ensembles, et utilisant les méthodes que vous aurez écrites.

Exercice 5 (45) : Robot Pollueur

Le monde est constitué d'une matrice de cases. Différents types de robots agissent dans ce monde : des robots pollueurs et des robots nettoyeurs. Les robots pollueurs se baladent dans le monde et déposent des papiers gras sur les cases où ils se trouvent. Les robots nettoyeurs, quant à eux, ne supportent pas la vue d'un papier gras et l'enlèvent dès qu'ils en voient un sur une case.

Les robots pollueurs sont de deux sortes : robots sauteurs et robots qui vont tout droit. Chaque robot pollueur suit son parcours et dépose un papier gras sur chaque case rencontrée. Le parcours précis des robots pollueurs sera donné dans la suite.

Les robots nettoyeurs parcourent méthodiquement le monde en "boustrophédon", c'est à dire qu'ils parcourent la première ligne case par case, puis arrivé en bout de ligne font demi-tour, parcourent la deuxième ligne en sens inverse, et ainsi de suite jusqu'à la dernière case de la dernière ligne. Ils enlèvent, s'ils sont présent, le papier gras de la case où ils se trouvent. Parmi les robots nettoyeurs, certains sont distraits et n'enlèvent qu'un papier sur deux.

Les méthodes de créations, les initialiseurs, les accesseurs et les méthodes `to_s()` seront créés dans chaque classe si besoin.

Question 1 : Dessiner proprement et complètement la hiérarchie des classes correspondant à la description ci-dessus.

Question 2 : Ecrire la classe `Monde` qui contient les variables d'instance, les méthodes de construction et les méthodes suivantes (préciser en commentaire si elles sont d'instance ou de classe) :

- le nombre de lignes `nbL`, le nombre de colonnes `nbC`, et une matrice booléenne `mat` de `nbL` lignes et `nbC` colonnes (true signifiant la présence d'un papier gras).
- `monde()` : cette méthode de création d'instance crée un monde sans papiers gras.
- La méthode `to_s()` retourne une chaîne de caractères décrivant le monde. On représentera un papier gras par le caractère "o", rien par le caractère "." (point).
- `metPapierGras(i, j)` : met un papier gras dans la case (i, j) .
- `prendPapierGras(i, j)` : enlève le papier gras de la case (i, j) .
- `estSale?(i, j)` : teste si la case (i, j) a un papier gras.

1) Ecrire dans une classe `TestRobot` une méthode `test` et y créer un `Monde` de 10 lignes et 10 colonnes. Y tester les méthodes `metPapierGras`, `prendPapierGras`, `estSale` et affiche. `test` est-elle une méthode de classe ou d'instance ?

2) Définir la classe `Robot` qui contient les champs et méthodes suivantes :

- `posx`, `posy` : position du robot sur le monde.
- `monMonde` : référence sur le `Monde`. En effet il faut que le robot connaisse le monde pour pouvoir s'y déplacer et agir.
- On devra pouvoir créer un robot en fournissant ces coordonnées et le monde par la méthode `robot(x, y, m)`
- `vaEn(i, j)` : se déplace en (i, j) .
- `parcourir()`

Quelle est la caractéristique de cette classe par rapport à l'héritage ?

3) Ecrire la classe `RobotPollueur` qui contient la méthode :

- `polluer()` : met un papier gras là où ce robot se trouve dans `monMonde`.

Définir les méthodes de création/initialisation

4) Ecrire la classe PollueurToutDroit qui contient les champs et méthodes suivantes :

- colDepart, numéro de la colonne de départ.
- parcourir() : Elle décrit un parcours complet de ce robot dans le monde. Le départ est en case (0, ColDepart), puis il va tout droit vers le sud en visitant chaque case de la colonne et dépose un papier gras sur chacune d'elle. Il s'arrête dans la dernière case de la colonne.

Définir les méthodes de création/initialisation

Créer dans la méthode test deux PollueurToutdroit et tester leur méthode parcourir(). Afficher l'état du monde après chaque parcours.

5) Ecrire la classe PollueurSauteur, qui contient les champs et méthodes suivants :

- colDepart, numéro de la colonne de départ.
- deltax représentant la taille du saut effectué à chaque déplacement du sauteur.
- parcourir() : Elle décrit un parcours complet de ce robot dans le monde. Le départ est en case (0, coldepart), puis il saute de façon analogue au cavalier des échecs et va en (1, coldepart+deltax), puis en (2, colDepart), puis en (3, colDepart+deltax), etc. Si la colonne sort de l'échiquier, on revient au début. Par exemple, la colonne 10, qui n'existe pas, sera transformée en 10 modulo 10, c'est à dire colonne 0. De même la colonne 15 (qui n'existe pas) sera transformée en 15 modulo 10, c'est à dire 5, etc. Chaque case rencontrée est souillée d'un papier gras. Le robot s'arrête lorsqu'il atteint la dernière ligne.

Définir les méthodes de création/initialisation

Créer dans la méthode test deux PollueurSauteur avec des colDepart et des deltax différents et tester leur méthode parcourir().

6) Ecrire la classe RobotNettoyeur, qui contient les méthodes suivantes :

- nettoyer() : enlève le papier gras de la case où se trouve ce robot,
- parcourir() : cette méthode définit la méthode parcourir abstraite de la classe Robot. Elle décrit un parcours complet de ce robot dans le monde. Il part de la case (0, 0) et parcourt le monde en "boustrophédon". Si la case est sale, il enlève le papier gras qui s'y trouve.

Définir les méthodes de création/initialisation

Créer dans la méthode main deux RobotNettoyeur et tester leur méthode parcourir() après le passage des robots pollueurs.

7) Ecrire la classe NettoyeurDistrait qui contient les méthodes suivantes :

- parcourir() : Elle décrit un parcours complet de ce robot dans le monde. C'est le même parcours que celui des robots nettoyeurs mais comme il est distrait, il n'enlève qu'un papier sur deux.

Définir les méthodes de création/initialisation

Créer dans le main un NettoyeurDistrait et tester sa méthode parcourir() après le passage d'un pollueur.