Formation JAVA - Application 2

L'objectif de cette application est de créer un jardin virtuel. Dans ce jardin, vous pourrez semer et récolter différents végétaux que vous verrez grandir de saison en saison. Vous pourrez également gérer un panier qui contiendra l'ensemble de vos graines restantes. L'application sera réalisée en mode console.

Partie 1 : Création du projet

Créer un nouveau projet nommé « Jardin ».
Créer une classe **Terre** qui contiendra votre fonction principale **main**.
En général, la fonction principale est une des méthodes les plus courtes du programme, son seul rôle sera d'instancier un objet **Jardin** et d'appeler ses méthodes nécessaires au fonctionnement de notre application.

Partie 2 : Classe Jardin

Avant de commencer à créer la classe, il convient de se demander de quoi est un constitué un jardin. Un jardin a **une longueur**, **une largeur** et est composé de différents **emplacements** qui permettront de semer **les végétaux**. Nous voulons également gérer **un panier** pour y stocker nos graines qui n'ont pas encore été semées.

Nous devons donc créer **4 attributs** : longueur, largeur, emplacement et panier. Ces attributs seront **privés** afin de respecter le principe **d'encapsulation**.
Mais de quels types seront emplacement et panier ?

Nous partons du principe que le nombre d'emplacements revient à la largeur multipliée par la longueur. Nous pouvons déduire que ce premier attribut sera **un tableau bidimensionnel**.
Le type d'emplacement sera **Emplacement**, une classe que nous allons créer ensuite.

En ce qui concerne le panier, il n'est pas nécessaire de créer un objet spécifique. En effet, il s'agit simplement d'associer un nom de végétal à une quantité et nous aurons notre panier. Coup de chance, Java fournit déjà une classe permettant cela : il s'agit de **HashMap**.
HashMap est une classe générique, c'est-à-dire que nous pouvons choisir les types de données que nous voulons stocker ; en l'occurrence, nous souhaitons associer **une chaine de caractères à un entier**. Nous devons déclarer notre panier de la manière suivante :

```
private HashMap<String, Integer> panier;
```

Notre panier pourra stocker des données de la manière suivante :

String (nom du végétal)	Integer (quantité de graines)
Ail	4
Betterave	7
Carotte	2

Partie 3 : Constructeur de la classe Jardin

Il peut être intéressant de créer **un constructeur paramétré** à la classe Jardin afin de permettre à la classe qui l'utilise d'indiquer la longueur et la largeur du jardin.
Nous allons créer **un constructeur public** prenant en **paramètre deux entiers** (longueur et largeur) et qui **instanciera et initialisera l'ensemble de nos attributs**. Pour instancier un objet de type HashMap, la syntaxe est la suivante :

```
this.panier = new HashMap<String, Integer>();
```

Nous pouvons désormais instancier un objet Jardin depuis notre fonction main en indiquant la longueur et la largeur du jardin que nous souhaitons.

Partie 4 : Classe Emplacement

Créer une nouvelle classe que vous appellerez **Emplacement**. Cette classe correspond, comme son nom l'indique, a un emplacement du jardin.
Posons-nous les mêmes questions que pour le jardin : que contient un emplacement ?
La réponse est assez simple ; un emplacement contient **un végétal**. Cette classe n'aura qu'**un seul attribut privé** de **type Vegetal**. Afin de pouvoir y accéder facilement, nous y ajouterons un **accesseur public**.
Concernant le **constructeur**, il prendra **en paramètre un objet de type Vegetal** afin de pouvoir initialiser son attribut dès la construction de l'objet.

Partie 5 : Package Flore
Créer un nouveau **package** que nous appellerons Flore et qui contiendra l'ensemble des classes relatives aux végétaux. Dans ce package, créer une nouvelle classe Vegetal.

Partie 6 : Classe Vegetal et ses classes filles
Nous avons vu qu'un emplacement était composé d'un végétal. Seulement, semer un végétal n'a pas de sens. En effet, lorsque l'on sème un végétal, il s'agit soit de tomate, de carotte, de betterave, mais jamais d'un végétal uniquement. Par conséquent, notre classe Vegetal sera **abstraite** et contiendra **les méthodes et attributs communs** à tous les végétaux.

Quels sont les éléments communs à tous les végétaux ? Dans notre programme, nous souhaitons voir grandir des légumes. Par conséquent, nous pouvons imaginer deux attributs : **etat** correspondant à l'état actuel du végétal (en graine, germé, en feuille, …) et **dessin** qui nous permettra d'afficher le végétal.
Concernant le type, nous allons créer **une classe Etat** pour l'attribut etat et dessin correspond à un ensemble de caractères, par conséquent, il s'agit d'**un tableau unidimensionnel de caractères**. Voici un exemple de dessins que peut prendre un végétal :

État du végétal	Dessin
En graine	_
En germe	.
En tige	I
En feuille	Dépend du végétal
En fleur	Dépend du végétal
Mort	#

Nous pouvons dès à présent créer le constructeur de Vegetal qui permettra d'initialiser le tableau dessin : on indiquera qu'il s'agit d'un tableau contenant **6 cases** et spécifiera **le contenu des 3 premières et de la dernière case.**

Ensuite, nous allons créer **4 classes filles** : Ail, Betterave, Carotte et Tomate. Pour chacune d'entre elles, il faudra créer **un constructeur qui complétera les quatrième et cinquième cases** du tableau dessin :

Etat du végétal		Ail	Betterave	Carotte	Tomate
En feuille		a	b	c	t
En fleur		A	B	C	T

Partie 7 : Classe Etat
La classe Etat est une classe particulière. En effet, nous voulons simplement indiquer que notre végétal est en feuille, en fleur, mort, … Il s'agit là d'**une énumération**.

Une énumération est un type de données constitué d'un ensemble de constantes que l'on appelle énumérateurs. Une variable de ce type peut alors recevoir un et un seul de ces énumérateurs comme valeur. Par conséquent, nous allons définir une énumération ayant comme énumérateurs : GRAINE, GERME, TIGE, FEUILLE, FLEUR et MORT.

Comme en Java, tout est une classe, les énumérations n'échappent pas à la règle. En effet, elles héritent toutes de la classe **java.lang.Enum**. Pour créer une énumération, il convient de faire un clic droit sur le paquetage censé contenir cette dernière et choisir New … > Enum. Le code sera le suivant :

```
public enum Etat {
	GRAINE,
	GERME,
	TIGE,
	FEUILLE,
	FLEUR,
	MORT
}
```

Nous pouvons maintenant finaliser le constructeur de la classe Vegetal puisque lorsque l'on sème un végétal, ce dernier est forcément à l'état de graine : this.etat = Etat.GRAINE;

Partie 8 : Exécution du programme

Nous avons créé toutes nos classes mais pour le moment, si nous lançons l'application, il ne se passe rien. Nous allons donc revenir à la méthode principale et réfléchir à l'algorithme qu'elle doit réaliser.

La première étape consiste à créer le jardin, c'est-à-dire appeler le constructeur de la classe Jardin.
La seconde étape est d'ajouter des graines dans notre panier ; en effet, sans graine, il va être difficile de semer dans notre jardin. Notre panier étant un attribut de la classe Jardin, nous allons créer **une méthode ajouterPanier** à la classe Jardin. Cette méthode prendra en paramètre une chaine de caractères correspondant au nom du végétal et un entier correspondant à la quantité de graines que l'on souhaite ajouter au panier.

Pour ajouter un élément dans un objet de type HashMap, la syntaxe est la suivante :

```
this.panier.put(nomDuVegetal, quantite);
```

Nous pouvons depuis le main ajouter des betteraves, des carottes, … à notre panier.
La troisième étape consiste à afficher le jardin. Jardin héritant implicitement de la classe Object, ce dernier possède la méthode **toString**. Cependant, nous aimerions que cette méthode nous affiche le contenu du jardin ainsi que le contenu du panier. Il va nous falloir **réécrire cette méthode** pour qu'elle affiche ce que l'on souhaite. Puisque notre jardin est vide pour le moment, nous afficherons le caractère « o » pour signifier un emplacement vide. Le résultat que l'on doit obtenir sera similaire au suivant :

```
Voici notre jardin :
ooooo
ooooo
ooooo
Et notre panier contient :
Tomate : 5 graine(s)
Carotte : 5 graine(s)
Ail : 10 graine(s)
Betrave : 5 graine(s)
```

Par la suite, nous aurons besoin d'afficher le contenu des différents emplacements. Il nous faut donc créer également une méthode **toString dans la classe Emplacement** qui affichera le dessin du végétal qu'il contient en fonction de son état. Nous parlons alors de polymorphisme puisque plusieurs classes n'ayant pas de lien ont une méthode du même nom et de même signature.
Enfin, la dernière étape consiste à demander à l'utilisateur ce qu'il souhaite faire. Il a le choix entre :

1. Semer une graine ;
2. Récolter toutes les plantes qui sont matures ;
3. Passer à la saison suivante (toutes les plantes grandissent) ;
4. Quitter l'application.

Nous allons donc afficher un menu et récupérer ce que l'utilisateur entre au clavier. Pour cela, **la classe Scanner** va nous être utile :

```
Scanner scanner = new Scanner(System.in);
int res = scanner.nextInt();
```

La variable res contiendra le nombre que l'utilisateur a entré. Attention, ce code ne fonctionnera pas si l'utilisateur entre une chaine de caractères.
Après cela, il convient d'appeler la méthode correspondant à ce que l'utilisateur a choisi.

Partie 8.1 : Semer une graine

Dans la classe Jardin, nous allons créer **une méthode semer** qui permettra à l'utilisateur de semer une graine. Nous allons encore utiliser la classe Scanner pour demander la position de l'emplacement où l'utilisateur souhaite placer sa graine (x et y) ainsi que le végétal qu'il souhaite semer.
Nous pouvons alors instancier un nouvel emplacement dans notre jardin qui prendra en paramètre une instance du végétal que nous voulons semer. Par exemple, pour de l'Ail :

```
this.emplacement[x][y] = new Emplacement(new Ail());
```

Après avoir semé une graine, notre jardin devrait ressembler à ceci :

```
Voici notre jardin :
_oooo
ooooo
ooooo
Et notre panier contient :
Tomate : 5 graine(s)
Carotte : 5 graine(s)
Betterave : 5 graine(s)
Ail : 9 graine(s)
```

Partie 8.2 : Passer à la saison suivante

Lorsque l'on passe à la saison suivante, tous nos végétaux grandissent. Il va donc falloir créer **une méthode grandir** dans la classe Vegetal.

Cette méthode va passer à l'état suivant dans le cas où le végétal n'est pas mort. Pour nous aider, la classe Enum nous fournit une méthode appelée ordinal qui nous retourne la position de l'énumérateur au sein de l'énumération. Il convient alors d'incrémenter cette valeur :

```
this.etat = Etat.values()[this.etat.ordinal() + 1];
```

Ensuite, nous allons créer **une méthode saisonSuivante** dans la classe Jardin qui va appeler la méthode grandir de chacun des végétaux présents dans les emplacements du jardin. Pour cela, nous allons parcourir le tableau emplacement de la classe Jardin et faire grandir tous les végétaux associés :

```
this.emplacement[x][y].getVegetal().grandir();
```

Attention, certains emplacements peuvent être vides, cela peut provoquer une erreur à l'exécution !

Notre jardin doit être similaire à celui-ci après quelques saisons :

```
Voici notre jardin :
Aiooo
b.ooo
ioooo
Et notre panier contient :
Tomate : 4 graine(s)
Carotte : 4 graine(s)
Betterave : 4 graine(s)
Ail : 8 graine(s)
```

Partie 8.3 : Récolter

Lorsqu'un végétal est à l'état FLEUR, il peut être récolté. La récolte consiste à **passer à null le végétal** contenu à un emplacement. En effet, une fois récolté, notre légume ne présente plus d'intérêt pour notre application, il est simplement supprimé.

La récolte consiste alors à parcourir notre tableau emplacement et à vérifier l'état de chacun des végétaux. Si le végétal est à l'état FLEUR, nous le supprimons :

```
this.emplacement[x][y] = null;
```

Si nous lançons la récolte dans le jardin présenté dans la partie 8.2, nous obtenons :

```
Voici notre jardin :
oiooo
b.ooo
ioooo
Et notre panier contient :
Tomate : 4 graine(s)
Carotte : 4 graine(s)
Betterave : 4 graine(s)
Ail : 8 graine(s)
```

Partie 9 : Interfaces

Notre application fonctionne désormais correctement. Cependant, toutes nos graines se comportent de la même manière : on les sème, elles grandissent et peuvent être récoltées. J'aimerais que mon programme prenne en compte les 3 types de graines que j'utilise :

• Race pure : il s'agit de végétaux qui produisent des graines réutilisables lors de la récolte. C'est-à-dire que lorsque l'on récolte notre végétal, il faut ajouter à notre panier de nouvelles graines que l'on vient de récolter également ;

• Hybride F1 : il s'agit des végétaux par défaut : lorsque nous les récoltons, aucune graine n'est ajoutée au panier ;

• OGM : il s'agit de végétaux venant d'un autre monde et qui ont un comportement très particulier : lorsque nous récoltons un végétal OGM, une graine est immédiatement semée à un emplacement choisi au hasard dans le jardin.

Pour nous permettre d'appliquer ces comportements à certains de nos végétaux, nous allons utiliser des interfaces.

Créer deux interfaces dans le paquetage Flore : IRacePure et IOgm.

IRacePure implémente la **méthode seReproduire** qui ne retourne rien et prend en paramètre un objet de type HashMap (il s'agit du panier de la classe Jardin).

IOgm implémente la **méthode seDupliquer** qui prend la longueur et la largeur du jardin et retourne un objet de type **SimpleEntry<Integer, Integer>**. La classe SimpleEntry correspond à un tuple ; dans notre cas, la méthode prendra en paramètre la longueur et la largeur du jardin et retournera les coordonnées x et y de l'emplacement où le végétal récolté se dupliquera.

Maintenant, il va falloir implémenter l'interface à nos différents végétaux :

	Ail	Betterave	Carotte	Tomate
… implémente	IRacePure	IOgm	IRacePure	Aucune interface

Les classes Ail et Carotte devront donc implémenter la méthode seReproduire. Cette méthode permettra d'ajouter dans le panier les graines récoltées :

```
panier.put(NOM, panier.get(NOM) + 3);
```

Mais comment appeler cette méthode ? Car dans la méthode recolter de la classe Jardin, nous utilisons des objets de type Vegetal et un végétal n'a pas de méthode seReproduire.

C'est pour cela que Java met à notre disposition le mot-clé **instanceof** qui permet de vérifier si un objet est une instance d'une classe :

```
if(emplacement.getVegetal() instanceof IRacePure)
{
    IRacePure v = (IRacePure) e.getVegetal();
    v.seReproduire(this.panier);
}
```

Dans cet exemple, nous vérifions que le végétal de l'emplacement est une race pure. Si tel est le cas, nous pouvons convertir l'objet Vegetal en un objet de type IRacePure. Nous manipulerons les mêmes données mais nous pourrons appeler les méthodes issues de l'interface IRacePure telle que seReproduire.

Vous pouvez désormais faire la même chose avec l'interface IOgm puis mettre une dernière fois à jour vos diagrammes.