Java - Examen session 2 du 2 juillet 2024. Durée 2 heures.

Pour cet examen, votre fond d'écran doit être vert clair. Vous devez absolument mettre tous les fichiers que vous souhaitez rendre dans le répertoire `EXAM` qui est déjà présent sur votre compte à l'ouverture de la session. Tout ce qui ne se trouve pas dans ce dossier sera perdu lorsque vous vous déconnectez.

Sous Eclipse, votre workspace doit être dans `EXAM` sinon configurez le workspace d'Eclipse (File > Switch WorkSpace) pour qu'il corresponde à un répertoire dans `EXAM`). Attention, les noms des classes, des champs et des méthodes que l'on vous demande doivent impérativement être respectés. De plus, le code doit être indenté correctement.

Commencez par configurer Eclipse. Pour cela aller dans le menu window, preferences, installed JRE. Faire add, standard VM. Cherchez ensuite le répertoire /usr/local/apps/java21 et ajouter le. Régler ensuite le compilateur, menu preferences, java compiler. Mettre le compilateur en java 21. Créez ensuite votre Java project qui porte votre nom : `NOM_PRENOM`.

La javadoc 21 et les slides du cours (seuls documents autorisés) sont accessibles ici :
`http://igm.univ-mlv.fr/~juge/javadoc-21/`
`https://igm.univ-mlv.fr/~beal/JavaL3.html`

Toutes les classes seront écrites dans un package `fr.uge.postoffice` dans la partie 1 et dans un package `fr.uge.postoffice2` dans la partie 2. On écrira une méthode `main` dans une classe `Main` dans chaque package pour faire les tests. Les tests des exemples donnés dans chaque question doivent être faits (les mettre en commentaire s'ils ne marchent pas). Ces tests sont pris en compte dans la notation. **Dans toutes les questions, on utilisera quand c'est possible et adapté des streams et/ou des lambdas.**

On cherche à modéliser un bureau de poste dans lequel peuvent être déposés des colis.

PARTIE 1

Exercice 1.

Définissez un type `Person` qui représente une personne qui contient deux champs, son nom `name`, une chaîne de caractères, et son adresse `address`, une autre chaîne de caractères. Les champs seront non mutables. Écrire le type `Person` de telle sorte que le code suivant fonctionne. Attention on veut exactement l'affichage demandé.

```
public static void main(String[] args) {
  var person1 = new Person("Dupont",
                  "5 boulevard Descartes, 77454 Marne-la-Vallée, France");
  System.out.println("person1");
  System.out.println(person1);
}
```

qui donne

```
person1
Dupont, 5 boulevard Descartes, 77454 Marne-la-Vallée, France
```

Exercice 2.

On va définir un type `Box` qui représente un colis qui contient quatre champs, son poids positif ou nul `weight`, de type `double`, deux personnes `Person` pour l'expéditeur `sender` et le destinataire `recipient`, et un dernier champ pour un timbre de type `Stamp` pour indiquer que le colis aura (plus tard) un prix réduit.

On va créer différents types de timbres :

1. un type `StarStamp` qui représente un timbre sous forme d'étoile. Lorsque l'on affiche un colis qui a un timbre sous forme d'étoile, une étoile apparait à la fin de la ligne d'affichage.

2. un type `PlusStamp` qui représente un timbre sous forme de plus. Un timbre sous forme de plus contient une catégorie (`category`) qui est un nombre entre 1 et 5 (les deux compris) et qui indique le nombre de plus sur le timbre. Il ne doit pas être possible de créer des timbres sous forme de plus avec une valeur de catégorie invalide. Lorsque l'on affiche un colis qui a un timbre sous forme de plus, autant de plus que la catégorie apparaissent à la fin de la ligne d'affichage : par exemple, si la catégorie est 2, deux `+` sont affichés.

3. un type `NullStamp` qui représente un timbre pour les colis qui n'ont pas de timbre.

Les champs seront non mutables.

Écrire le code pour les timbres et faites en sorte qu'il ne soit pas possible d'avoir d'autres types de timbres que ceux décrits plus haut. Écrire le type `Box` pour les colis. Pour l'affichage de l'expéditeur et du destinataire dans un colis, on n'affichera que leur nom comme ci-dessous. L'affichage du timbre est à la fin.

```
public static void main(String[] args) {
  ...
  var person2 = new Person("Dupond",
                   "10 rue Galilée, 77454 Marne-la-Vallée, France");
  var box1 = new Box(500.0, person1, person2,  new PlusStamp(1));
  System.out.println("box1");
  System.out.println(box1);
```

donne en sortie

```
box1
Box: 500.0, Dupont -> Dupond, +
```

Exercice 3. On veut maintenant écrire une classe `PostOffice` pour représenter un bureau de poste qui contient une liste `boxes` de colis. La liste peut contenir plusieurs fois le même colis. Écrire la classe avec une méthode `add` qui permet d'ajouter un colis à la liste. Ajouter une méthode `toString` pour afficher la liste de colis comme ci-dessous (un colis par ligne).

```
public static void main(String[] args) {
  ...
  var person3 = new Person("Forax",
                    "7 rue de la Paix, 75002 Paris, France");
  var person4 = new Person("Beal",
                   "24 avenue de la Bourdonnais, 75007 Paris, France");
  var box2 = new Box(200.0, person3, person4, new StarStamp());
  var box3 = new Box(500.0, person4, person3, new PlusStamp(3));
  var box4 = new Box(1000.0, person1, person4, new NullStamp());
  var postOffice = new PostOffice();
  postOffice.add(box1);
  postOffice.add(box2);
  postOffice.add(box3);
  postOffice.add(box4);
  System.out.println("postOffice");
  System.out.println(postOffice);
}
```

donne

```
postOffice
Box: 500.0, Dupont -> Dupond, +
Box: 200.0, Forax -> Beal, *
Box: 500.0, Beal -> Forax, +++
Box: 1000.0, Dupont -> Beal,
```

Exercice 4. Ajouter une méthode `weight` qui renvoie la somme des poids des colis du bureau de poste. La méthode renverra 0 si le bureau de poste ne contient aucun colis.

```java
public static void main(String[] args) {
  System.out.println("weight of all boxes of postOffice");
  System.out.println(postOffice.weight()); // 2200.0
}
```

Exercice 5.

Écrire une méthode `allBoxesToRecipient` qui renvoie une liste non modifiable de tous les colis qui doivent être envoyés à une personne. S'il n'y a pas de colis pour cette personne, la liste renvoyée sera vide. On vérifiera que le code suivant fonctionne.

```java
public static void main(String[] args) {
  ...
  var list = postOffice.allBoxesToRecipient(person4);
  System.out.println("list");
  System.out.println(list);
```

donne

```
list
[Box: 200.0, Forax -> Beal, *, Box: 1000.0, Dupont -> Beal, ]
```

Exercice 6.

Écrire une méthode `boxesByRecipient()` qui renvoie une map donnant pour chaque `nom` de destinataire la liste des colis qui lui sont destinés. On vérifiera que le code suivant fonctionne.

```java
public static void main(String[] args) {
  ...
  var map = postOffice.boxesByRecipient();
  System.out.println("map");
  System.out.println(map);
```

donne

```
map
{Beal=[Box: 200.0, Forax -> Beal, *, Box: 1000.0, Dupont -> Beal, ],
Dupond=[Box: 500.0, Dupont -> Dupond, +],
Forax=[Box: 500.0, Beal -> Forax, +++]}
```

L'affichage ci-dessus contient des retours à la ligne mais c'est juste pour la présentation du sujet. N'écrivez pas un code qui produit ces retours à la ligne.

Exercice 7.

Écrire une méthode `weightByRecipient()` qui renvoie une map donnant pour chaque `nom` de destinataire la somme des poids de tous les colis qui lui sont destinés. On vérifiera que le code suivant fonctionne.

```java
public static void main(String[] args) {
  ...
  var map2 = postOffice.weightByRecipient();
  System.out.println("map2");
  System.out.println(map2);
```

donne

```
map2
{Beal=1200.0, Dupond=500.0, Forax=500.0}
```

Exercice 8.

On souhaite maintenant pouvoir calculer le prix total pour l'envoi de tous les colis du bureau de poste. Le prix d'un colis dépend uniquement du poids et du timbre. Les prix seront des `int`.

Écrire une méthode `boxPrice` qui renvoie le prix d'un colis. On utilisera le pattern matching des types pour avoir le prix d'un colis. Le prix d'un colis est un entier obtenu à partir de son

poids (sans tenir compte de ce qu'il y a après la virgule) auquel on applique une réduction en fonction du timbre. Les réductions de prix sont des réduction maximales car un colis ne peut pas avoir un prix négatif. Quand on a un timbre sous forme d'étoile, il y a une réduction sur le prix de 200. Dans le cas où l'on a un timbre sous forme de plus, le colis à une réduction de prix de 10% par plus (par exemple `++++` correspond à une réduction de 40%).

Écrire le code de `price` qui doit renvoyer la somme des prix de tous les colis du bureau de poste. En tenant compte des timbres les prix deviennent :

```
0 pour Box: 200.0, Forax -> Beal, *,
1000 pour Box: 1000.0, Dupont -> Beal,
450  pour Box: 500.0, Dupont -> Dupond, +
350 pour Box: 500.0, Beal -> Forax, +++
```

```
public static void main(String[] args) {
 ...
 System.out.println("price");
 System.out.println(postOffice.price();
 // price
 // 1800
}
```

PARTIE 2

Faites un copier-coller du package `fr.uge.postoffice` dans un package `fr.uge.postoffice2` et travaillez dans `fr.uge.postoffice2` pour cette partie. On va modifier uniquement les classes `PostOffice` et `Main`.

On désire dans cette partie stocker les colis dans le bureau de poste de telle sorte que l'on ait un accès en temps constant à la liste des colis à destination d'une personne identifiée par son nom. Pour cela on utilisera une map qui associe à chaque `nom` de destinataire (de type `String`) la liste des colis qui lui sont destinés.

Exercice 9.

Faites les modifications dans la classe `PostOffice` pour prendre en compte ces conditions. On aura un unique champ dans `PostOffice` et on ne conservera que les méthodes `add`, `toString`, `weight` (en les modifiant).

Ajouter une méthode `getByRecipient(String name)` qui renvoie la liste non modifiable des colis adressés à une personne et donnant en argument le nom de la personne (de type `String`).

Ajouter une méthode `getBySenderAndRecipient(String name1, String name2)` qui renvoie la liste non modifiable des colis adressés à une personne de nom `name1` et d'expéditeur une personne de nom `name2`.

Modifier aussi la méthode `toString` pour que l'affichage d'un bureau de poste soit comme ci-dessous.

```
public static void main(String[] args) {
  var person1 = new Person("Dupont",
                   "5 boulevard Descartes, 77454 Marne-la-Vallée, France");
  var person2 = new Person("Dupond",
                   "10 rue Galilée, 77454 Marne-la-Vallée, France");
  System.out.println("person1");
  System.out.println(person1);
  var box1 = new Box(500.0, person1, person2,  new PlusStamp(1));
  System.out.println("box1");
  System.out.println(box1);
  var person3 = new Person("Forax",
                    "7 rue de la Paix, 75002 Paris, France");
  var person4 = new Person("Beal",
                    "24 avenue de la Bourdonnais, 75007 Paris, France");
  var box2 = new Box(200.0, person3, person4, new StarStamp());
  var box3 = new Box(500.0, person4, person3, new PlusStamp(3));
  var box4 = new Box(1000.0, person1, person4, new NullStamp());
```

```
  var postOffice = new PostOffice();
  postOffice.add(box1);
  postOffice.add(box2);
  postOffice.add(box3);
  postOffice.add(box4);
  System.out.println("postOffice");
  System.out.println(postOffice);
  System.out.println("weight of all boxes of postOffice");
  System.out.println(postOffice.weight());
  System.out.println("getByRecipient");
  System.out.println(postOffice.getByRecipient("Beal"));
  System.out.println("getBySenderAndRecipient -> Beal");
  System.out.println(postOffice.getBySenderAndRecipient("Forax", "Beal"));}
```

donne

```
person1
Dupont, 5 boulevard Descartes, 77454 Marne-la-Vallée, France
package1
Box: 500.0, Dupont -> Dupond, +
postOffice
Beal: [Box: 200.0, Forax -> Beal, *, Box: 1000.0, Dupont -> Beal, ]
Dupond: [Box: 500.0, Dupont -> Dupond, +]
Forax: [Box: 500.0, Beal -> Forax, +++]
weight of all boxes of postOffice
2200.0
getByRecipient -> Beal
[Box: 200.0, Forax -> Beal, *, Box: 1000.0, Dupont -> Beal, ]
getBySenderAndRecipient Forax -> Beal
[Box: 200.0, Forax -> Beal, *]
```