Java - Examen du 22 janvier 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.

Si vous utilisez Eclipse, configurez le workspace d'Eclipse (File > Switch WorkSpace) pour qu'il corresponde au répertoire `EXAM` que vous avez dans le répertoire home de votre session de TP noté. 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.

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 de la partie 1 seront écrites dans un package `fr.uge.sport` et celles de la partie 2 dans un package `fr.uge.sport2`. On écrira une méthode `main` dans une classe `Main` pour faire les tests.

On cherche à modéliser un magasin de sport qui vend des vêtements et des chaussures de sport. Dans la première partie les articles du magasin sont stockés dans une liste. On peut avoir plusieurs fois le même article dans la liste. Dans la deuxième partie on utilise une map qui indique pour chaque article combien il y en a en stock dans le magasin.

PARTIE 1

Exercice 1. Commencez par configurer eclipse. Vérifier que l'environnement d'exécution est bien Java-21, changer si ce n'est pas le cas. Le JDK 21 est situé dans /usr/local/apps/java21. Vérifier que la version pour la compilation est bien 21 (dans les paramètres du compilateur), changer si ce n'est pas le cas. Créez ensuite votre Java project qui porte votre nom : `NOM_PRENOM`, et votre package `fr.uge.sport`.

Exercice 2. On souhaite définir un type `Clothing` qui modélise des vêtements de sport. Chaque vêtement a un `type` de type `ClothingType` qui est l'enum suivant

```
public enum ClothingType {
  T_SHIRT, POLO, SWEAT_SHIRT, PANT
}
```

Un vêtement a aussi une marque (`brand`) de type `String` et deux champs entiers `size` et `price` pour la taille et le prix. Les tailles devront être comprises entre 1 et 5 et le prix doit être positif ou nul.

Écrire le type `Clothing` avec le constructeur compact. Écrire une deuxième constructeur qui prend en argument le type, la marque et le prix et crée le vêtement en taille 1. Faites en sorte que l'on puisse afficher les vêtements comme dans le code suivant :

```
public static void main(String[] args) {
  var polo = new Clothing(ClothingType.POLO, "Colmar", 3, 40);
  System.out.println(polo);
  var polo2 = new Clothing(ClothingType.POLO, "Colmar", 40);
  System.out.println(polo2);
}
```

donne

```
Clothing[type=POLO, brand=Colmar, size=3, price=40]
Clothing[type=POLO, brand=Colmar, size=1, price=40]
```

Exercice 3. On souhaite définir un type `ShoePair` qui modélise des chaussures de sport. Chaque paire de chaussures a une marque `brand` de type `String`, une couleur `color` de type `String` et deux champs entiers `size` et `price` pour la taille et le prix. Les tailles devront être

comprises entre 37 et 44 et le prix doit être positif ou nul. Écrire le type `ShoePair` avec le constructeur compact. Faites en sorte que l'on puisse afficher les chaussures comme dans le code suivant :

```
public static void main(String[] args) {
  var shoePair1 = new ShoePair("Nike", "black", 38, 300);
  var shoePair2 = new ShoePair("Nike", "white", 44, 300);
  System.out.println(shoePair1);
  System.out.println(shoePair2);
}
```

donne

```
ShoePair[brand=Nike, color=black, size=38, price=300]
ShoePair[brand=Nike, color=white, size=44, price=300]
```

Exercice 4. Écrire une classe `SportsShop` qui modélise un magasin de sport. Un magasin aura un champ `name` de type `String` et il contiendra une liste d'articles qui seront des vêtements de sport ou des chaussures de sport. Les articles pourront figurer plusieurs fois dans la liste. On fera en sorte qu'il ne puisse pas y avoir d'autres types d'articles que les vêtements de sport et les chaussure de sport. Le type commun pour les articles du magasin sera `Sportswear`. Écrire une méthode `add` pour ajouter un article dans le magasin.

Exercice 5. Écrire une méthode d'affichage dans `SportsShop` qui permet d'afficher le magasin. On affichera le nom du magasin sur une ligne puis chaque article sur une ligne. On devra utiliser des streams. Le code suivant devra donc fonctionner :

```
public static void main(String[] args) {
  var polo = new Clothing(ClothingType.POLO, "Colmar", 3, 40);
  var polo2 = new Clothing(ClothingType.POLO, "Colmar", 40);
  var shirt1 = new Clothing(ClothingType.T_SHIRT, "Burton", 4, 50);
  var shirt2 = new Clothing(ClothingType.T_SHIRT,"Burton", 4, 50);
  var shoePair1 = new ShoePair("Nike", "black", 38, 300);
  var shoePair2 = new ShoePair("Nike", "white", 44, 300);
  var shop1 = new SportsShop("Italie2");
  shop1.add(polo);
  shop1.add(shirt1);
  shop1.add(shoePair1);
  shop1.add(shoePair2);
  var shop2 = new SportsShop("Jaude");
  shop2.add(shirt2);
  shop2.add(polo);
  shop2.add(shoePair1);
  shop2.add(shoePair1);
  shop2.add(shoePair2);
  System.out.println(shop1);
  System.out.println(shop2);
  }
```

donne

```
Italie2
Clothing[type=POLO, brand=Colmar, size=3, price=40]
Clothing[type=T_SHIRT, brand=Burton, size=4, price=50]
ShoePair[brand=Nike, color=black, size=38, price=300]
ShoePair[brand=Nike, color=white, size=44, price=300]
Jaude
Clothing[type=T_SHIRT, brand=Burton, size=4, price=50]
Clothing[type=POLO, brand=Colmar, size=3, price=40]
ShoePair[brand=Nike, color=black, size=38, price=300]
ShoePair[brand=Nike, color=black, size=38, price=300]
ShoePair[brand=Nike, color=white, size=44, price=300]
```

Exercice 6. Écrire une méthode `public long totalPrice()` qui calcule le prix total des tous les articles du magasin. On renverra bien un `long` et pas un `int`. On devra utiliser des streams.

```
public static void main(String[] args) {
  ...
  System.out.println(shop1.totalPrice()); // 690
}
```

Exercice 7. Écrire une méthode `onSale()` qui renvoie la liste non modifiable des articles du magasin qui sont en solde. Un vêtement est en solde si sa taille est supérieure ou égale à 3 et une paire de chaussures est en solde si sa taille est supérieure ou égale à 43. On utilisera le polymorphisme et la méthode `onSale()` sera écrite avec des streams. Dans l'affichage de la liste ci-dessous des sauts de lignes ont été ajoutés.

```
public static void main(String[] args) {
  ...
  System.out.println(shop1.onSale());
}
```

donne

```
 [Clothing[type=POLO, brand=Colmar, size=3, price=40],
  Clothing[type=T_SHIRT, brand=Burton, size=4, price=50],
  ShoePair[brand=Nike, color=white, size=44, price=300]]
```

Exercice 8. Écrire une méthode `shoesBySize()` qui renvoie une map de type `Map<Integer, List<ShoePair>>` dont les clés sont des tailles de chaussures et la valeur associée à une clé est la liste des chaussures de cette taille. On utilisera des streams, le pattern matching de Java 19 et un `flatMap`. L'idée dans le `flatMap` est que, si c'est un vêtement, on renvoie un stream vide et que, si c'est une paire de chaussures, on renvoie un stream contenant la paire de chaussures. Dans l'affichage de la map ci-dessous des sauts de lignes ont été ajoutés.

```
public static void main(String[] args) {
  ...
  System.out.println(shop2.shoesBySize());
}
```

donne

```
{38=[ShoePair[brand=Nike, color=black, size=38, price=300],
 ShoePair[brand=Nike, color=black, size=38, price=300]],
 44=[ShoePair[brand=Nike, color=white, size=44, price=300]]}
```

Exercice 9. Écrire une méthode `selectedItems` qui renvoie la liste non modifiable des articles du magasin qui vérifient une condition. On utilisera des streams. Si on présuppose qu'il existe dans `SportsShop` une méthode `priceTooHigh` définie comme ceci

```
static boolean priceTooHigh(Sportswear sportswear) {
  return sportswear.price() >= 300;
}
```

Par exemple on pourra appeler `selectedItems` avec `priceTooHigh` en paramètre, et on aura

```
System.out.println(shop2.selectedItems(SportsShop::priceTooHigh));
```

qui donne

```
[ShoePair[brand=Nike, color=black, size=38, price=300],
 ShoePair[brand=Nike, color=black, size=38, price=300],
 ShoePair[brand=Nike, color=white, size=44, price=300]]
```

Exercice 10. Écrire une méthode `occurrences` qui renvoie une map de type `Map<Sportswear, Long>` indiquant pour chaque article combien de fois il figure dans le magasin. On demande explicitement des `Long` et pas des `Integer`. On utilisera des streams.

```
public static void main(String[] args) {
  ...
   System.out.println(shop2.occurrences());
}
```

donne

```
{ShoePair[brand=Nike, color=white, size=44, price=300]=1,
 ShoePair[brand=Nike, color=black, size=38, price=300]=2,
 Clothing[type=POLO, brand=Colmar, size=3, price=40]=1,
 Clothing[type=T_SHIRT, brand=Burton, size=4, price=50]=1}
```

Exercice 11. Écrire une méthode `public static boolean sameItems(SportsShop shop1, SportsShop shop2)` qui teste si deux magasins ont les mêmes articles sans tenir compte des répétitions.

```
public static void main(String[] args) {
  ...
   System.out.println(SportsShop.sameItems(shop1, shop2)); // true
}
```

PARTIE 2

Recopiez votre package `fr.uge.sport` dans un package `fr.uge.sport2`. Les classes `Clothing`, `ShoePair`, `Sportswear` ne vont pas changer. Vous allez modifier la classe `SportsShop` pour gérer les articles du magasin dans une map dont les clés sont les articles. La valeur associée à un article est le nombre d'exemplaires (toujours strictement positif) de l'article dans le magasin. Les valeurs associées aux clés seront des `Long` et pas des `Integer`. Dans les méthodes ré-écrites, on gardera les mêmes type de retour que pour la partie 1.

Exercice 12. Changer la liste en map dans `SportsShop` et modifier la méthode `add`.

Exercice 13. Modifier la méthode d'affichage pour tenir compte de la nouvelle implémentation. L'affichage du magasin se fera on mettant le nom sur une ligne et chaque article suivi de `": "` et de son nombre d'exemplaires dans le magasin sur une ligne. On utilisera toujours des streams.

```
public class Main {
  public static void main(String[] args) {
    var polo = new Clothing(ClothingType.POLO, "Colmar", 3, 40);
    var polo2= new Clothing(ClothingType.POLO, "Colmar", 40);
    var shirt1 = new Clothing(ClothingType.T_SHIRT, "Burton", 4, 50);
    var shirt2= new Clothing(ClothingType.T_SHIRT,"Burton", 4, 50);
    var shoePair1 = new ShoePair("Nike", "black", 38, 300);
    var shoePair2 = new ShoePair("Nike", "white", 44, 300);
    var shop1 = new SportsShop("Italie2");
    shop1.add(polo);
    shop1.add(shirt1);
    shop1.add(shoePair1);
    shop1.add(shoePair2);
    var shop2 = new SportsShop("Jaude");
    shop2.add(shirt2);
    shop2.add(polo);
    shop2.add(shoePair1);
    shop2.add(shoePair1);
    shop2.add(shoePair2);
    System.out.println(shop1);
    System.out.println(shop2);
```

```
  }
}
```

donne

```
Italie2
Clothing[type=T_SHIRT, brand=Burton, size=4, price=50]: 1
ShoePair[brand=Nike, color=white, size=44, price=300]: 1
Clothing[type=POLO, brand=Colmar, size=3, price=40]: 1
ShoePair[brand=Nike, color=black, size=38, price=300]: 1
Jaude
Clothing[type=T_SHIRT, brand=Burton, size=4, price=50]: 1
ShoePair[brand=Nike, color=white, size=44, price=300]: 1
Clothing[type=POLO, brand=Colmar, size=3, price=40]: 1
ShoePair[brand=Nike, color=black, size=38, price=300]: 2
```

Exercice 14. Modifier la méthode `long totalPrice()` pour tenir compte de la nouvelle implémentation. On utilisera toujours des streams.

```
public class Main {
  public static void main(String[] args) {
    ...
    System.out.println(shop1.totalPrice()); // 690
  }
}
```

Exercice 15. Modifier la méthode `onSale()` pour tenir compte de la nouvelle implémentation. La méthode renverra comme dans la première partie une `List<Sportswear>` et elle pourra contenir plusieurs fois les mêmes articles. Par exemple si un vêtement en solde figure trois fois dans le magasin, il apparaîtra trois fois dans la liste renvoyée. On utilisera toujours des streams. On pourra utiliser les méthodes `IntStream.range()` ou `Collections.nCopies()`.

```
public class Main {
  public static void main(String[] args) {
    ...
    System.out.println(shop1.onSale());
  }
}
```

donne

```
[Clothing[type=T_SHIRT, brand=Burton, size=4, price=50],
 ShoePair[brand=Nike, color=white, size=44, price=300],
 Clothing[type=POLO, brand=Colmar, size=3, price=40]]
```

Exercice 16. Modifier la méthode `selectedItems` pour tenir compte de la nouvelle implémentation. On utilisera toujours des streams.

```
public class Main {
  public static void main(String[] args) {
    ...
    System.out.println(shop2.selectedItems(SportsShop::priceTooHigh));
  }
}
```

donne

```
[ShoePair[brand=Nike, color=white, size=44, price=300],
 ShoePair[brand=Nike, color=black, size=38, price=300],
 ShoePair[brand=Nike, color=black, size=38, price=300]]
```

Exercice 17. Modifier la méthode `occurrences` pour tenir compte de la nouvelle implémentation. Quelle est la complexité de cette méthode désormais (répondre en commentaire au-dessus de la méthode) ?

```
public class Main {
  public static void main(String[] args) {
    ...
    System.out.println(shop2.occurrences());
  }
}
```

donne

```
{ShoePair[brand=Nike, color=white, size=44, price=300]=1,
 Clothing[type=POLO, brand=Colmar, size=3, price=40]=1,
 Clothing[type=T_SHIRT, brand=Burton, size=4, price=50]=1,
 ShoePair[brand=Nike, color=black, size=38, price=300]=2}
```

Exercice 18. Ajouter une méthode `get` qui prend en argument un article et renvoie le nombre d'exemplaires de l'article en magasin (0 si l'article n'est pas présent). Quelle est la complexité (répondre en commentaire au-dessus de la méthode) ?

```
public class Main {
  public static void main(String[] args) {
    ...
    System.out.println(shop2.get(new ShoePair("Nike", "black", 38, 300))); // 2
  }
}
```

Exercice 19. Modifier la méthode `sameItems` pour tenir compte de la nouvelle implémentation.

```
public class Main {
  public static void main(String[] args) {
    ...
    System.out.println(SportsShop.sameItems(shop1, shop2)); // true
  }
}
```