L3 2024-2025

Programmation orientée objet - Java - Examen session 2. 1er Juillet 2025.
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 déconnecterez. Ne créez pas vous-même de répertoire `EXAM`. Mettez tous vos fichiers dans celui qui existe.

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 vérifier la configuration d'Eclipse.

- Vérifiez que le JRE est /usr/local/apps/java23.
- Vérifez que le compilateur est java 23 avec enable preview.
- Créez ensuite votre Java project qui porte votre nom : `NOM_PRENOM`.

La javadoc 23 et les slides du cours (seuls documents autorisés) sont accessibles ici :
`http://igm.univ-mlv.fr/~juge/javadoc-23/`
`https://igm.univ-mlv.fr/~beal/JavaL3.html`

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.**

Toutes les classes seront écrites dans un package `fr.uge.medical`. On écrira une méthode `main` dans une classe `Main` dans le package pour faire les tests.

On cherche à modéliser une pharmacie qui stocke des ordonnances de patients.

1. Écrire un record `Patient` pour représenter un patient. Une patient contiendra deux champs : un champ `name` pour son nom de type `String`, et un champ `age` pour son age (un `int`). Le contructeur devra tester que l'âge est toujours positif ou nul.

   Compléter le record de telle sorte que le code suivant fonctionne avec un affichage comme ci-dessous.

   ```
   IO.println("test 1");
   var patient1 = new Patient("Bob", 32);
   var patient2 = new Patient("Alice", 30);
   IO.println(patient1);
   IO.println(patient2);
   ```

   donne

   ```
   test 1
   Bob (32)
   Alice (30)
   ```

2. On souhaite maintenant écrire des types pour des médicaments (`Medication`). Les médicaments ont deux types possibles (et seulement deux types) : les médicaments génériques (`Generic`) et les non-génériques (`Brand`). Écrire les deux types `Generic` et `Brand`. Chaque objet `Generic` ou `Brand` contient un nom `name` de type `Name`, où `Name` est l'enum ci-dessous, et un prix `price` de type `int`. Les champs seront tous non mutables.

```
public enum Name {
  AMOXICILLIN, AMOXIL,
  IBUPROFEN, ADVIL,
  LOSARTAN, COZAR,
  SERTRALINE, ZOLOFT,
  OMEPRAZOLE, PRILOZEC
}
```

Écrire les types de telle sorte que le code suivant fonctionne.

```
IO.println("test 2");
var medication = new Generic(Name.AMOXICILLIN, 10);
var medication2 = new Brand(Name.AMOXIL, 50);
IO.println(medication);
IO.println(medication2);
```

donne

```
test 2
AMOXICILLIN (Generic) 10 euros
AMOXIL (Brand) 50 euros
```

3. Écrire un record `Prescription` pour représenter une ordonnance. Chaque ordonnance contient une champ `patient` de type `Patient` et un champ `prescription` de type `Set<Medication>`. L'ensemble sera non mutable.

   Compléter le record de telle sorte que le code suivant fonctionne. Le constructeur devra faire une copie défensive de l'ensemble passé en argument.

```
IO.println("test 3");
var prescription1 = new Prescription(patient1, Set.of(
                            new Generic(Name.AMOXICILLIN, 10),
                            new Generic(Name.IBUPROFEN, 10),
                            new Brand(Name.ZOLOFT, 50)
                        ));
var prescription2 = new Prescription(patient2, Set.of(
                             new Generic(Name.AMOXICILLIN, 10),
                             new Generic(Name.OMEPRAZOLE, 10),
                             new Brand(Name.ADVIL, 60)
                         ));
IO.println(prescription1);
IO.println(prescription2);
```

   donne

```
test 3
Bob (32)
AMOXICILLIN (Generic) 10 euros
IBUPROFEN (Generic) 10 euros
ZOLOFT (Brand) 50 euros
Alice (30)
AMOXICILLIN (Generic) 10 euros
ADVIL (Brand) 60 euros
OMEPRAZOLE (Generic) 10 euros
```

4. Écrire une méthode `price` qui renvoie le prix global d'une ordonnance. Testez le code ci-dessous.

```
IO.println("test 4");
IO.println(prescription1.price());
```

donne

```
test 4
70
```

5. Écrire une méthode `Set<Medication> generics()` qui renvoie l'ensemble non modifiable des médicaments génériques d'une ordonnance.

6. Écrire une méthode `Set<Generic> generics2()` qui renvoie l'ensemble non modifiable des médicaments génériques d'une ordonnance. C'est le même résultat que pour la question précédente mais le type renvoyé doit être cette fois `Set<Generic>`. Sautez cette question si vous ne savez pas faire. Les casts ne sont pas autorisés.

   Testez le code ci-dessous.

```
IO.println("test 5 et 6");
IO.println(prescription1.generics());
IO.println(prescription1.generics2());
```

   donne

```
test 5 et 6
[IBUPROFEN (Generic) 10 euros, AMOXICILLIN (Generic) 10 euros]
[IBUPROFEN (Generic) 10 euros, AMOXICILLIN (Generic) 10 euros]
```

7. Écrire une méthode `List<Medication> filter` qui prend en argument un prédicat pour des médicaments et renvoie la liste non modiable des médicaments de l'ordonnance qui vérifient le prédicat. Testez le code ci-dessous.

```
IO.println("test 7");
IO.println(prescription2.filter(m -> m.name().toString().startsWith("A")));
```

   donne

```
test 7
[AMOXICILLIN (Generic) 10 euros, ADVIL (Brand) 60 euros]
```

8. Écrire une classe `Pharmacy` pour représenter une pharmacie qui contient une map dont les clés sont de type `Patient` et la valeur associée à une clé est l'ordonnance du patient. Chaque patient a une seule ordonnance et si on ajoute une ordonnance à un patient qui en a déjà une, l'ancienne ordonnance est remplacée et il n'y a pas d'exception levée.

   Ajouter à la classe une méthode `add` qui permet d'ajouter une ordonnance dans la pharmacie et une méthode `toString` pour que le code ci-dessous fonctionne.

```
IO.println("test 8");
var pharmacy = new Pharmacy();
pharmacy.add(prescription1);
pharmacy.add(prescription2);
IO.println(pharmacy);
```

   donne

```
test 8
Alice (30)
AMOXICILLIN (Generic) 10 euros
OMEPRAZOLE (Generic) 10 euros
ADVIL (Brand) 60 euros
Bob (32)
AMOXICILLIN (Generic) 10 euros
ZOLOFT (Brand) 50 euros
IBUPROFEN (Generic) 10 euros
```

9. Écrire une méthode `List<Patient> filterPatient` qui prend en argument un prédicat pour des patients et renvoie la liste non modifiable des patients de la pharmacie qui vérifient ce prédicat.

   Testez le code ci-dessous

```
IO.println("test 9");
IO.println(pharmacy.filterPatient(p -> p.name().p.name().startsWith("B")));
IO.println(pharmacy.filterPatient(p -> p.name().length() > 0));
```

   donne

```
test 9
[Bob (32)]
[Alice (30), Bob (32)]
```

10. Écrire une méthode `List<Patient> filterPatient2` qui prend en argument deux prédicats : un prédicat pour des patients et un prédicat pour des médicaments, et renvoie la liste non modifiable des patients de la pharmacie qui vérifient le prédicat des patients et dont l'ordonnance contient au moins un médicament qui vérife le prédicat des médicaments.

    Testez le code ci-dessous

```
IO.println("test 10");
IO.println(pharmacy.filterPatient2(p -> true, m -> m.name()== Name.OMEPRAZOLE));
```

    donne

```
test 10
[Alice (30)]
```

11. Écrire une méthode `Prescription mostExpensive` qui renvoie l'ordonnance la plus coûteuse de la phramacie. La méthode lèvera une exception si la pharmacie ne contient aucune ordonnance.

    Testez le code ci-dessous

```
IO.println("test 11");
IO.println(pharmacy.mostExpensive());
```

    donne

```
test 11
Alice (30)
ADVIL (Brand) 60 euros
OMEPRAZOLE (Generic) 10 euros
AMOXICILLIN (Generic) 10 euros
```