L3 2025-2026

Programmation orientée objet - Java - Examen Janvier 2026. Durée 2 heures.

Pour cet examen, votre fond d'écran doit être vert clair. Vous devez obligatoirement placer 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 le répertoire `EXAM`. Placez tous vos fichiers dans celui qui existe déjà.

Sous Eclipse, votre workspace doit être dans `EXAM`. Sinon, configurez-le (via File > Switch Workspace) pour qu'il corresponde à un répertoire dans `EXAM`. Attention : les noms des classes, des champs et des méthodes demandés doivent impérativement être respectés. De plus, le code doit être correctement indenté.

Commencez par vérifier la configuration d'Eclipse.

- Vérifiez que le JRE est /usr/local/apps/java25.
- Vérifiez que le compilateur est java 25
- Créez ensuite votre Java project qui portera votre nom : `NOM_PRENOM`.

La javadoc 25 et les slides du cours (seuls documents autorisés) sont accessibles ici :
`https://ligm.univ-eiffel.fr/~juge/javadoc-25/`
`https://ligm.univ-eiffel.fr/~beal/JavaL3.html`

Les tests des exemples donnés dans chaque question doivent être réalisés (les mettre en commentaire s'ils ne fonctionnent pas). Ces tests sont pris en compte dans la notation. Vous devrez utiliser des streams et des lambdas lorsque c'est opportun. **L'utilisation de `instanceof` est interdite.** Vous n'aurez pas à écrire les méthodes `equals` et `hashCode`.

Dans la première partie, toutes les classes seront écrites dans le package `fr.uge.spacetravel`. Dans la deuxième partie, toutes les classes seront écrites dans le package `fr.uge.spacetravel2`. Vous écrirez une méthode `main` dans une classe `Main` dans chaque package pour effectuer les tests.

Dans cet examen, on modélise des vaisseaux spatiaux. Trois types de passagers peuvent embarquer :

- des membres d'équipage (`Crew`),
- des touristes (`Tourist`),
- des droïdes (`Droid`).

Partie 1

Exercice 1.

1. Membre d'équipage (`Crew`)

   Un membre d'équipage (`Crew`) est caractérisé par

   - son nom (`name`) (une chaîne de caractères),
   - son âge (`age`, un nombre entier d'années) qui ne peut pas dépasser 60 ans,
   - son grade de type `Rank`, qui est un type énuméré qui peut prendre une des trois valeurs `ROOKIE`, `STAFF` ou `OFFICER`.

   Le type énuméré `Rank` se définit comme suit :

```
public enum Rank {
  ROOKIE, STAFF, OFFICER
}
```

Le code est à placer dans un fichier `Rank.java`.

Une variable de type `Rank` peut donc prendre les valeurs `Rank.ROOKIE`, `Rank.STAFF`, `Rank.OFFICER`.

Écrire le code permettant de créer et d'afficher un objet de type `Crew` possédant ces trois champs. L'affichage devra être de la forme :

```
IO.println("test 1");
var crew = new Crew("James", 40, Rank.ROOKIE);
IO.println(crew);
```

Sortie attendue :

```
Crew James (40 years ROOKIE)
```

2. Touriste (`Tourist`).

   Un touriste est caractérisé par :

   - son nom (`name`) (une chaîne de caractères),
   - un booléen (`vip`) indiquant s'il est VIP,
   - un nombre de crédits (`credits`, un entier au maximum 30 000).

   Écrire le code permettant de créer et afficher un objet de type `Tourist`.

```
var tourist = new Tourist("Jeff", true, 30_000);
IO.println(tourist);
var tourist2 = new Tourist("Elon", false, 10_000);
IO.println(tourist2);
```

   Sortie attendue :

```
Tourist Jeff (vip 30000)
Tourist Elon (10000)
```

3. Droïde (`Droid`)

   Un droïde est un robot possédant

   - un identifiant (`id`, un entier),
   - un rôle (`role`, une chaîne de caractères).

   Écrire le code permettant de créer et afficher un objet de type `Droid` de la façon suivante :

```
var droid = new Droid(42, "chemist");
IO.println(droid);
```

   Sortie attendue :

```
Droid 42 (chemist)
```

4. Vaisseau spatial (`SpaceShip`)

   Créer une classe pour modéliser un vaisseau spatial `SpaceShip` contenant :

   - un nom (`name` de type `String`)
   - un ensemble de passagers (`passengers`, de type `LinkedHashSet`). Les passagers peuvent être `Crew`, `Tourist` ou `Droid`. Le type commun pour les passagers sera une interface scellée (sealed) `Passenger`.

   Un `LinkedHashSet` se manipule comme un `HashSet` et permet de conserver l'ordre d'insertion pour un affichage.

   Écrivez une méthode `add` permettant d'ajouter un passager. Une exception sera levée si le passager est déjà présent.

5. Affichage du vaisseau

   Ajoutez une méthode pour afficher le vaisseau en commençant par son nom, suivi de ses passagers (un par ligne) :

```
IO.println("test 2");
var enterprise = new SpaceShip("Enterprise");
enterprise.add(new Tourist("Jeff", true, 30_000));
enterprise.add(new Tourist("Elon", false, 10_000));
enterprise.add(new Crew("James", 40, Rank.ROOKIE));
enterprise.add(new Crew("Naomi", 50, Rank.STAFF));
enterprise.add(new Crew("Amos", 45, Rank.OFFICER));
enterprise.add(new Droid(3451435, "fire-gunner"));
enterprise.add(new Droid(42, "fire-gunner"));
enterprise.add(new Droid(44, "astronaut"));
enterprise.add(new Droid(4111, "astronaut"));
enterprise.add(new Droid(40, "icebreaker"));
enterprise.add(new Crew("Camina", 30, Rank.OFFICER));
IO.println(enterprise);
```

   Sortie attendue :

```
Enterprise
Tourist Jeff (vip 30000)
Tourist Elon (10000)
Crew James (40 years ROOKIE)
Crew Naomi (50 years STAFF)
Crew Amos (45 years OFFICER)
Droid 3451435 (fire-gunner)
Droid 42 (fire-gunner)
Droid 44 (astronaut)
Droid 4111 (astronaut)
Droid 40 (icebreaker)
Crew Camina (30 years OFFICER)
```

6. Méthode hibernation dans `Passenger`

   Ajoutez une méthode statique `int hibernation(Passenger passenger)` dans l'interface `Passenger` qui calcule le temps maximal d'hibernation pour chaque passager. Ce temps dépend du type des passagers et pour les membres d'équipage, de leur grade :

   - pour les droïdes, 100 ans,
   - pour les touristes VIP, 60 ans,
   - pour les touristes non VIP, 50 ans,
   - pour les membres d'équipage :
     - 10 ans pour un `ROOKIE`,
     - 15 ans pour un `STAFF`,
     - 30 ans pour un `OFFICER`.

   On peut effectuer du pattern-matching sur des `Enum` (voir le slide 37 du cours poo5.pdf sur les interfaces et le polymorphisme).

7. Méthode hibernation dans `SpaceShip`

   Écrivez une méthode `hibernation` qui calcule le **minimum** des temps maximaux d'hibernation de tous les passagers. Une exception sera levée si le vaisseau est vide.

```
IO.println("test 3");
IO.println(enterprise.hibernation());
```

   Sortie attendue :

```
test 3
10
```

8. Liste des membres d'équipage

   Écrivez une méthode `crew` qui renvoie une liste non modifiable de type `List<Crew>` des membres d'équipage. Vous pourrez écrire une méthode privée statique qui prend en argument un passager et renvoie un `Stream<Crew>` contenant ce passager si ce passager est un membre d'équipage, sinon `null`.

```
IO.println("test 4");
IO.println(enterprise.crew());
```

   Sortie attendue :

```
[Crew James (40 years ROOKIE), Crew Naomi (50 years STAFF),
 Crew Amos (45 years OFFICER), Crew Camina (30 years OFFICER)]
```

   La sortie est donnée ici sur plusieurs lignes uniquement pour la présentation du sujet.

9. Membres d'équipage par grade

   Écrivez une méthode `crewByRank` qui renvoie une map non modifiable (de type `Map<Rank, List<Crew>>`), associant à chaque grade la liste des membres d'équipage correspondants.

```
IO.println("test 5");
IO.println(enterprise.crewByRank());
```

   Sortie attendue :

```
{OFFICER=[Crew Amos (45 years OFFICER), Crew Camina (30 years OFFICER)],
 STAFF=[Crew Naomi (50 years STAFF)],
 ROOKIE=[Crew James (40 years ROOKIE)]}
```

   La sortie est donnée ici sur plusieurs lignes uniquement pour la présentation du sujet.

10. Liste des officiers

    Écrivez une méthode `officers` qui renvoie une liste non modifiable (de type `List<Crew>`) des membres d'équipage de grade `Rank.OFFICER`.

```
IO.println("test 6");
IO.println(enterprise.officers());
```

    Sortie attendue :

```
[Crew Amos (45 years OFFICER), Crew Camina (30 years OFFICER)]
```

11. Filtrage des membres d'équipage

    Écrivez une méthode `crewWithPredicate` qui prend en argument un prédicat sur des `Crew` et renvoie une liste non modifiable (de type `List<Crew>`) des membres d'équipage vérifiant ce prédicat.

```
IO.println("test 7");
IO.println(enterprise.crewWithPredicate(c -> c.age() >= 45));
```

    Sortie attendue :

```
[Crew Naomi (50 years STAFF), Crew Amos (45 years OFFICER)]
```

Partie 2

Exercice 2.

Copiez-collez le package `fr.uge.spacetravel` dans `fr.uge.spacetravel2` et travaillez dans ce nouveau package. Gardez les records `Crew`, `Tourist`, `Droid` et l'enum `Rank`. Videz le `main` de `Main`.

Dans la classe `SpaceShip`, ne gardez que les champs (`name` et `passengers`), et les méthodes `add` et `toString`.

1. Ajout d'une map pour les droïdes

   Ajoutez un **troisième champ** `droids` (une map associant à chaque rôle (de type `String`) une liste des droïdes ayant ce rôle).

   Modifiez la méthode `add` pour mettre à jour cette map lors de l'ajout d'un droïde. Vérifiez que les tests suivants fonctionnent toujours :

   Cette map pourra servir à effectuer des recherches plus rapides sur les droïdes du vaisseau sans avoir à parcourir l'ensemble de tous les passagers.

   Vérifier que les tests suivants fonctionnent toujours :

```
IO.println("test 1");
var crew = new Crew("James", 40, Rank.ROOKIE);
IO.println(crew);
var tourist = new Tourist("Jeff", true, 30_000);
IO.println(tourist);
var tourist2 = new Tourist("Elon", false, 10_000);
IO.println(tourist2);
var droid = new Droid(42,"chemist");
IO.println(droid);
// -------------------------
IO.println("test 2");
var enterprise = new SpaceShip("Enterprise");
enterprise.add(new Tourist("Jeff", true, 30_000));
enterprise.add(new Tourist("Elon", false, 10_000));
enterprise.add(new Crew("James", 40, Rank.ROOKIE));
enterprise.add(new Crew("Naomi", 50, Rank.STAFF));
enterprise.add(new Crew("Amos", 45, Rank.OFFICER));
enterprise.add(new Droid(3451435, "fire-gunner"));
enterprise.add(new Droid(42, "fire-gunner"));
enterprise.add(new Droid(44, "astronaut"));
enterprise.add(new Droid(4111, "astronaut"));
enterprise.add(new Droid(40, "icebreaker"));
enterprise.add(new Crew("Camina", 30, Rank.OFFICER));
IO.println(enterprise);
```

   Sortie attendue :

```
test 1
Crew James (40 years ROOKIE)
Tourist Jeff (vip 30000)
Tourist Elon (10000)
Droid 42 (chemist)
test 2
Enterprise
Tourist Jeff (vip 30000)
Tourist Elon (10000)
Crew James (40 years ROOKIE)
Crew Naomi (50 years STAFF)
Crew Amos (45 years OFFICER)
```

```
Droid 3451435 (fire-gunner)
Droid 42 (fire-gunner)
Droid 44 (astronaut)
Droid 4111 (astronaut)
Droid 40 (icebreaker)
Crew Camina (30 years OFFICER)
```

2. Droïdes par rôle

   Écrivez une méthode `droidsWithRole(String role)` qui prend un rôle en argument et renvoie une liste non modifiable des droïdes ayant ce rôle. Si aucun droïde n'a ce rôle, on renverra une liste vide.

```
IO.println("test 3");
IO.println(enterprise.droidsWithRole("astronaut"));
```

   Sortie attendue :

```
[Droid 44 (astronaut), Droid 4111 (astronaut)]
```

3. Nombre total de droïdes

   Écrivez une méthode `numberOfDroids()` qui renvoie le nombre de droïdes dans le vaisseau.

```
IO.println("test 4");
IO.println(enterprise.numberOfDroids());
```

   Sortie attendue :

```
test 4
5
```

4. Nombre de droïdes par rôle

   Écrire une méthode `numberOfDroidsWithRole(String role)` qui renvoie le nombre de droïdes ayant un rôle donné passé en argument.

```
IO.println("test 5");
IO.println(enterprise.numberOfDroidsWithRole("astronaut"));
```

   Sortie attendue :

```
test 5
2
```

5. Filtrage des droïdes

   Écrivez une méthode `droidsWithPredicate` qui renvoie une liste non modifiable des droïdes vérifiant un prédicat sur les droïdes passé en argument.

```
IO.println("test 6");
IO.println(enterprise.droidsWithPredicate(d -> d.role().endsWith("r")));
```

   Sortie attendue :

```
test 6
[Droid 40 (icebreaker), Droid 3451435 (fire-gunner), Droid 42 (fire-gunner)]
```