L3 2025-2026

Programmation orientée objet - Java
Examen session 2 du 29 juin 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 de méthodes `equals` et `hashCode`.

Toutes les classes seront écrites dans le package `fr.uge.library`.

Vous écrirez une méthode `main` dans une classe `Main` pour effectuer les tests.

Dans cet examen, on modélise une bibliothèque municipale qui gère les emprunts de livres de ses adhérents.

1. Adhérent (`Member`)

   Un adhérent (`Member`) est caractérisé par :

   - son nom (`name`, une chaîne de caractères),
   - son âge (`age`, un nombre entier d'années) qui doit être positif ou nul.

   Écrire le code permettant de créer et d'afficher un objet de type `Member`.

   ```
   IO.println("test member");
   var member1 = new Member("Emma", 22);
   var member2 = new Member("Lucas", 14);
   IO.println(member1);
   IO.println(member2);
   ```

   Sortie attendue :

   ```
   test member
   Emma (22)
   Lucas (14)
   ```

2. Livres papier ou numériques (`PaperBook` et `EBook`)

   Un livre papier (`PaperBook`) est caractérisé par trois champs :

   - son titre (`title`, une chaîne de caractères),

- son numéro ISBN (`isbn`, une chaîne de caractères),
- son nombre de pages (`pages`, un entier strictement positif).

Un livre est affiché avec son titre et son nombre de pages entre parenthèses.

Un livre numérique (`EBook`) possède les mêmes champs.

Écrire le code permettant de créer et d'afficher des objets de type `PaperBook` et `EBook`.

Un livre numérique est affiché avec un `[E]` après son titre et son nombre de pages entre parenthèses..

```
IO.println("test book");
var book1 = new PaperBook(
    "Dune",
    "9780441172719",
    450);
var book2 = new EBook(
    "Harry Potter and the Philosopher's Stone",
    "9780747532699",
    320);
var book3 = new PaperBook(
    "Foundation",
    "9780553293357",
    280);
var book4 = new EBook(
    "Le Petit Prince",
    "9780156013987",
    96);
IO.println(book1);
IO.println(book2);
IO.println(book3);
IO.println(book4);
```

Sortie attendue :

```
test book
Dune (450)
Harry Potter and the Philosopher's Stone [E] (320)
Foundation (280)
Le Petit Prince [E] (96)
```

3. Emprunt (`Loan`)

   Un emprunt (`Loan`) est caractérisé par deux champs :

   - un adhérent (`member`, de type `Member`),
   - une liste de livres (`list`, de type `List<Book>`).

   où `Book` est un type commun pour `PaperBook` et `EBook`. On prendra une interface scellée.

   Un même livre pourra a priori être stocké plusieurs fois dans la liste.

   Écrire le code permettant de créer un objet de type `Loan`. **On créera un `record` qui stocke la liste de façon non modifiable**. Le constructeur devra faire une copie défensive de la liste passée en argument.

```
IO.println("test loan");
var loan1 = new Loan(member1, List.of(book1, book2));
var loan2 = new Loan(member2, List.of(book2, book3, book4));
IO.println(loan1);
IO.println(loan2);
```

Sortie attendue :

```
test loan
Emma (22) [Dune (450), Harry Potter and the Philosopher's Stone [E] (320)]
Lucas (14) [Harry Potter and the Philosopher's Stone [E] (320),
            Foundation (280), Le Petit Prince [E] (96)]
```

L'affichage ci-dessus est sur plusieurs lignes pour la présentation du sujet uniquement.

4. `filter()` dans `Loan`

   Écrire une méthode `public List<Book> filter(Predicate<Book> predicate)` qui prend en argument un prédicat pour des livres et renvoie la liste non modifiable des livres de l'emprunt qui vérifient ce prédicat.

   On testera par exemple :

```java
IO.println("test filter");
IO.println(loan1.filter(b -> b.title().startsWith("H")));
```

   Sortie attendue :

```
test filter
[Harry Potter and the Philosopher's Stone [E] (320)]
```

5. `ebooks()` dans `Loan`

   Écrire une méthode `public List<EBook> ebooks()` qui renvoie la liste non modifiable des livres numériques d'un emprunt. Le type de retour devra être `List<EBook>`.

   Cette liste peut être vide.

```java
IO.println("test ebooks()");
IO.println(loan1.ebooks());
IO.println(loan2.ebooks());
```

   Sortie attendue :

```
test ebooks()
[Harry Potter and the Philosopher's Stone [E] (320)]
[Harry Potter and the Philosopher's Stone [E] (320), Le Petit Prince [E] (96)]
```

6. `noDuplicates()` dans `Loan`

   Écrire une méthode `private static boolean noDuplicates(List<Book> list)` qui renvoie faux si la liste contient plusieurs livres ayant le même ISBN.

   Modifier le constructeur compact de `Loan` afin qu'un emprunt ne puisse pas être créé avec une telle liste.

   Vérifiez que le code ci-dessous lève une exception et commentez ensuite ces lignes.

```java
IO.println("test no duplicates");
var loan3 = new Loan(member2,
        List.of(book2, book3, book4, new PaperBook("Le Petit Prince",
        "9780156013987",
        98)));
```

7. Bibliothèque (`Library`)

   Écrire une classe `Library`, pour représenter une bibliothèque, qui contient un champ

   - `HashMap<Member, List<Loan>> map`.

La map permet d'associer à un adhérent une liste de ses emprunts. Le dernier emprunt ajouté est en fin de liste.

Écrire une méthode `public void add(Loan loan)` permettant d'ajouter un emprunt dans la map.

Ajoutez une méthode `toString` dans `Library`. On affichera un emprunt par ligne.

```
IO.println("test library");
var library = new Library();
var loan4 = new Loan(member2, List.of(book1, book4));
library.add(loan1);
library.add(loan2);
library.add(loan4);
IO.println(library);
```

Sortie attendue :

```
test library
Emma (22) [Dune (450), Harry Potter and the Philosopher's Stone [E] (320)]
Lucas (14) [Harry Potter and the Philosopher's Stone [E] (320),
            Foundation (280), Le Petit Prince [E] (96)]
Lucas (14) [Dune (450), Le Petit Prince [E] (96)]
```

8. `lastLoan()` dans `Library`

   Écrire une méthode `Optional<Loan> lastLoan(Member member)` qui prend en argument un adhérent et renvoie son dernier emprunt sous forme d'un optional.

   Si l'adhérent n'est pas présent dans la map, la méthode renverra `Optional.empty()`.

```
IO.println("test lastLoan");
IO.println(library.lastLoan(member2));
```

   Sortie attendue :

```
test lastLoan
Optional[Lucas (14) [Dune (450), Le Petit Prince [E] (96)]]
```

9. `filterMember()` dans `Library`

   Écrire une méthode `List<Member> filterMember(Predicate<Member> predicate)` qui renvoie la liste non modifiable des adhérents de la bibliothèque vérifiant le prédicat.

```
IO.println("test filter member");
IO.println(library.filterMember(m -> m.age() >= 18));
```

   Sortie attendue :

```
test filter member
[Emma (22)]
```

10. `filterMember2()` dans `Library`

    Écrire une méthode `List<Member> filterMember2(Predicate<Member> memberPredicate, Predicate<Book> bookPredicate)` qui renvoie la liste non modifiable des adhérents vérifiant le premier prédicat et ayant au moins un emprunt contenant un livre vérifiant le second prédicat.

```
IO.println("test filter member 2");
IO.println(library.filterMember2(_ -> true, b -> b.title().startsWith("F")));
```

    Sortie attendue :

`[Lucas (14)]`

11. `memberWithFewestLoans()` dans `Library`

    Écrire une méthode `public Member memberWithFewestLoans()` qui renvoie un adhérent ayant un nombre minimal d'emprunts dans la bibliothèque. Attention, il s'agit du nombre minimal d'emprunts, pas du nombre minimal de livres empruntés. Si la bibliothèque est vide, la méthode lèvera l'exception `NoSuchElementException`.

```
IO.println("test member with fewest loans");
IO.println(library.memberWithFewestLoans());
```

    Sortie attendue :

```
test member with fewest loans
Emma (22) % car Emma a 1 emprunt et Lucas 2
```