BUT 1 2025-2026 SESSION 2

Développement orienté objet - Java - TP noté 2 session 2 du 7 mai 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/JavaBUT1.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. **L'utilisation de `instanceof` est interdite.** Vous n'aurez pas à écrire les méthodes `equals` et `hashCode`.

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

Dans ce sujet on va modéliser des hôtels et des palaces et savoir combien rapportent les hôtels et les palaces que l'on possède.

Tous les exemples de code proposés devront être inclus dans la fonction `main` d'une classe `Main`.

Exercice 1.

Écrire un type (`Hotel`) qui permet de représenter des hôtels. Un hôtel possède une recette (le montant qu'il rapporte, `income`, un `double`), et un nom de rue (`streetName`). Tous les champs seront non mutables. Attention à gérer correctement le fait qu'un hôtel doit forcément avoir un nom de rue et ne peut pas avoir une recette strictement négative. Ajouter une méthode permettant d'afficher un hôtel de façon à ce que le code suivant fonctionne.

```
var hotel = new Hotel(10_000, "rue de Matignon");
IO.println(hotel);  // rue de Matignon = 10000.0
```

Exercice 2.

On souhaite ajouter une classe qui modélise un livre de comptes, ou `Ledger` en anglais, dans lequel on va pouvoir ajouter nos hôtels.

Ecrire une classe `Ledger`, qui stocke une liste d'hôtels et dispose d'une méthode `add` permettant d'ajouter un hôtel au livre de comptes ainsi qu'une méthode `remove` permettant de supprimer un hôtel présent dans le livre de comptes.

Il est possible d'ajouter plusieurs fois le même hôtel (pour des raisons comptables). Il est possible qu'un hôtel appartienne à plusieurs livres de comptes.

Pour la suppression, s'il y a plusieurs fois le même hôtel dans un livre de comptes, la suppression devra concerner l'hôtel qui a été ajouté en premier dans ce livre de comptes. Enfin, si un hôtel n'est pas dans le livre de comptes, il n'est pas possible de le supprimer et cela doit provoquer une levée d'exception.

Vérifier aussi que le code suivant fonctionne

```
var ledger = new Ledger();
ledger.add(hotel); // l'hôtel est ajouté
ledger.remove(new Hotel(10_000, "rue de Matignon"));  // l'hôtel est supprimé
// ledger.remove(new Hotel(10_000, "rue de Matignon"));
// Exception in thread "main" java.lang.IllegalStateException
```

Exercice 3. Faites en sorte que l'on puisse afficher un livre de comptes avec une ligne pour chaque hôtel, avec en début de ligne le nom de l'hôtel, un `=`, puis la recette de cet hôtel.

Vous pouvez vérifier que le code suivant fonctionne.

```
var ledger2 = new Ledger();
ledger2.add(new Hotel(10_000, "rue de Matignon"));
ledger2.add(new Hotel(5_000, "rue de la Gare"));
ledger2.add(new Hotel(15_000, "Beau Rivage"));
ledger2.add(new Hotel(15_000, "Beau Rivage"));
ledger2.add(new Hotel(10_001, "boulevard Descartes"));
IO.println(ledger2); // affiche:
//  rue de Matignon = 10000.0
//  rue de la Gare = 5000.0
//  Beau Rivage = 15000.0
//  Beau Rivage = 15000.0
//  boulevard Descartes = 10001.0
```

Exercice 4.

Ajouter dans la classe `Ledger` une méthode `double totalIncome` qui prend en paramètre un booléen `withTax` et qui renvoie la recette de l'ensemble des hôtels (la somme des recettes des hôtels) avec les taxes, appliquées ou non, selon l'argument.

Si la taxe est appliquée, alors la recette d'un hôtel est 80% de la recette initiale dans le cas où la recette est strictement supérieure à 10000 (c'est-à-dire qu'il y a une taxe de 20%), alors que dans le cas ou la recette est inférieure ou égale à 10000, la taxe ne s'applique pas (i.e., c'est la recette initiale).

Vous pouvez vérifier que le code suivant fonctionne.

```
IO.println(ledger2.totalIncome(true));
// 47000.8
```

Exercice 5. On souhaite maintenant ajouter un type `Palace` représentant un hôtel qui offre un meilleur service pour un prix supérieur.

De plus, il y a deux sortes de palaces : le palace normal et le palace VIP qui donne, en plus des services classiques d'un palace, un accès illimité au mini-bar.

Écrire un type `Palace` sachant qu'un palace est créé à partir d'une recette `income` (toujours un `double`), d'un nom de rue `streetName` et d'un booléen `vip`. Tous les champs seront non mutables.

Faites en sorte que l'affichage pour un palace donne le nom de la rue, la recette ainsi que la mention "palace" et que dans le cas d'un palace VIP, la mention soit "palace" suivie d'une étoile `*`.

```
var palace = new Palace(50_000, "rue de la Paix", false);
var palaceVIP = new Palace(100_000, "rue de la Paix", true);
IO.println(palace);     // affiche rue de la Paix = 50000.0 palace
IO.println(palaceVIP);  // affiche rue de la Paix = 100000.0 palace *
```

Exercice 6. On veut maintenant pouvoir ajouter/retirer des palaces ou des hôtels dans notre livre de comptes. On aura besoin d'un type commun `Asset` pour les hôtels et les palaces. Ajoutez le code nécessaire et modifiez la classe `Ledger` pour le faire. Vérifiez qu'il est possible d'ajouter/supprimer un palace dans un livre de comptes.

```
var ledger3 = new Ledger();
ledger3.add(palace);
ledger3.add(palaceVIP);
ledger3.add(new Palace(200_000, "rue de la Paix", true));
IO.println("ledger3 avant remove");
IO.println(ledger3);
//  rue de la Paix = 50000.0 palace
//  rue de la Paix = 100000.0 palace *
//  rue de la Paix = 200000.0 palace *
ledger3.remove(new Palace(200_000, "rue de la Paix", true));
IO.println("ledger3 après remove");
IO.println(ledger3);
// rue de la Paix = 50000.0 palace
// rue de la Paix = 100000.0 palace *
```

Exercice 7. La recette d'un palace est calculée de la même façon que pour un hôtel, mais le seuil à partir duquel on applique la taxe n'est pas de 10000 comme pour les hôtels mais de 50000 pour un palace et 100000 pour un palace VIP. De plus, la taxe n'est pas de 20% comme pour les hôtels mais uniquement 5%. Le seuil est toujours strict, c'est-à-dire qu'il s'applique si la recette est strictement supérieure à 50000.

Modifiez la méthode `double totalIncome` pour que le code suivant fonctionne. On utilisera le polymorphisme. Bien sûr `instanceof` et `getClass()` sont interdits ici.

```
ledger3.add(new Palace(70_000, "avenue des Champs Elysées", false));
ledger3.add(new Palace(70_000, "avenue de Champs sur Marne", true));
IO.println("nouveau ledger3");
IO.println(ledger3);
// rue de la Paix = 50000.0 palace
// rue de la Paix = 200000.0 palace *
// avenue des Champs Elysées = 70000.0 palace
// avenue de Champs sur Marne = 70000.0 palace *
IO.println(ledger3.totalIncome(true));
// 286500.0
```

Exercice 8.

Ajouter une méthode `int numberOfPalaces` à la classe Ledger qui renvoie le nombre de palaces parmi les propriétés enregistrées dans le livre de comptes. On utilisera le polymorphisme. Bien sûr, `instanceof` et `getClass()` sont interdits ici.

```
var ledger4 = new Ledger();
ledger4.add(new Hotel(10_000, "rue de Matignon"));
ledger4.add(palace);
ledger4.add(palaceVIP);
IO.println(ledger4.numberOfPalaces());
// 2
```

Exercice 9.

Ajouter une méthode `Map<String, List<Asset» assetsByStreetName()` à la classe `Ledger` qui renvoie une map donnant pour chaque nom de rue la liste des propriétés figurant dans cette rue. Une même propriété pourra figurer deux fois dans la liste associée à une clé.

```
IO.println(ledger4.assetsByStreetName());
// {rue de la Paix=[rue de la Paix = 50000.0 palace, rue de la Paix = 100000.0 palace *],
//   rue de Matignon=[rue de Matignon = 10000.0]}
```

Exercice 10.

Écrire une méthode `static boolean sameAssets(Ledger ledger1, Ledger ledger2)` qui teste si deux livres de comptes ont les mêmes propriétés sans compter les répétitions.

```
var ledger5 = new Ledger();
ledger5.add(new Hotel(10_000, "rue de Matignon"));
ledger5.add(palace);
ledger5.add(palaceVIP);
ledger5.add(palace);
IO.println(Ledger.sameAssets(ledger4, ledger5)); // true
IO.println(Ledger.sameAssets(ledger5, ledger4)); // true
```

Exercice 11.

Faites un copier-coller du package `fr.uge.monopoly` en `fr.uge.monopoly2`. Gardez les records `Hotel`, `Palace` et l'interface `Asset`. Videz `Ledger` et `Main`.

La classe `Ledger` aura maintenant un unique champ qui est une map de type `HashMap<String, List<Asset>>`. On stocke les hôtels et palaces avec cette map qui associe à une adresse (un nom de rue) la liste des hôtels ou palaces qui ont cette adresse.

Écrire une méthode `add(Asset asset)` dans `Ledger` qui permet d'ajouter un objet `Asset` dans `Ledger`.

Exercice 12.

Écrire une méthode `toString` qui permet d'afficher la map de `Ledger` comme ci-dessous.

```
var ledger = new Ledger();
ledger.add(new Hotel(10_000, "rue de Matignon"));
ledger.add(new Hotel(5_000, "rue de la Gare"));
ledger.add(new Hotel(15_000, "Beau Rivage"));
ledger.add(new Hotel(15_000, "Beau Rivage"));
ledger.add(new Hotel(10_001, "boulevard Descartes"));
ledger.add(new Palace(70_000, "rue de Matignon", false));
ledger.add(new Palace(70_000, "rue de Matignon", true));
IO.println(ledger);
// affiche sur 4 lignes
//Beau Rivage: [Beau Rivage = 15000.0, Beau Rivage = 15000.0]
//rue de la Gare: [rue de la Gare = 5000.0]
// rue de Matignon: [rue de Matignon = 10000.0, rue de Matignon = 70000.0 palace,
   rue de Matignon = 70000.0 palace *]
// boulevard Descartes: [boulevard Descartes = 10001.0]
```