ESIEE E3 2025-2026

Développement orienté objet - Java
TP noté session 2 du 28 août 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/JavaESIEE.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.esiee.ferry`. Vous écrirez une méthode `main` dans une classe `Main` pour effectuer les tests.

Dans cet examen, on modélise des ferries assurant le transport de voitures et de camions.

---

1. Voiture (`Car`)

   Une voiture (`Car`) est caractérisée par

   - le nom du propriétaire `ownerName` de type `String`.
   - le nombre de passagers `passengers`, un entier. Ce nombre doit être positif ou nul.
   - le nombre d'enfants parmi les passagers `children`. Ce nombre doit être positif ou nul et inférieur ou égal au nombre de passagers.

   Écrire un record `Car` permettant de créer et d'afficher un objet de type `Car` possédant ces champs. L'affichage devra être de la forme :

   ```
   IO.println("test record Car");
   var car1 = new Car("John", 2, 1);
   IO.println(car1);
   IO.println();
   ```

   Sortie attendue :

   ```
   test record Car
   Car John 2 (1)
   ```

2. Camion (`Truck`)

   Un camion (`Truck`) est caractérisé par

- le nom de l'entreprise `companyName` de type `String`.
- son poids en kilos `weight`, un entier. Ce nombre doit être positif ou nul.

Écrire un record `Truck` permettant de créer et d'afficher un objet de type `Truck` possédant ces champs. L'affichage devra être de la forme :

```
IO.println("test record Truck");
var truck1 = new Truck("World Inc", 2000);
IO.println(truck1);
IO.println();
```

Sortie attendue :

```
test record Truck
Truck World Inc 2000
```

3. Interface `Vehicle`

   On aura besoin d'un type commun pour les voitures et les camions. Ajouter pour cela une interface scellée `Vehicle`.

   Ajouter dans cette interface une méthode `String name()` qui donne, pour une voiture, le nom du propriétaire, et, pour un camion, le nom de l'entreprise.

4. Ferry (`Ferry`)

   Écrivez une classe `Ferry` qui contient :

   - une liste de véhicules (`ferry`), de type `ArrayList<Vehicle>`,

   La classe ne contiendra qu'un seul champ non statique.

   Écrivez une méthode `add` permettant d'ajouter un véhicule au ferry. On ne vérifiera pas si un véhicule est déjà présent avant de l'ajouter (un même véhicule pourra donc figurer plusieurs fois dans la liste).

   Ajoutez une méthode `toString` pour afficher le ferry en affichant un véhicule par ligne :

```
IO.println("test class Ferry");
var car2 = new Car("David", 4, 0);
var car3 = new Car("John", 5, 3);
var truck2 = new Truck("FedEx", 1000);
var ferry = new Ferry();
ferry.add(car1);
ferry.add(truck1);
ferry.add(truck2);
ferry.add(car3);
ferry.add(car1);
ferry.add(car2);
ferry.add(truck1);
ferry.add(truck1);
IO.println(ferry);
IO.println();
```

   Sortie attendue :

```
test class Ferry
Car John 2 (1)
Truck World Inc 2000
Truck FedEx 1000
Car John 5 (3)
Car John 2 (1)
Car David 4 (0)
Truck World Inc 2000
Truck World Inc 2000
```

5. Voitures sans enfants `carsWithNoKids`

   Écrire une méthode `public List<Car> carsWithNoKids()` qui renvoie la liste des voitures sans enfants du ferry.

   La méthode renverra une liste non modifiable et le type de retour doit être `List<Car>`.

```
IO.println("test cars with no kids");
IO.println(ferry.carsWithNoKids());
IO.println();
```

   Sortie attendue :

```
test cars with no kids
[Car David 4 (0)]
```

6. Calcul de la flotte `fleet()` dans `Ferry`

   Écrire une méthode `Map<String, List<Vehicle>>` qui calcule une map donnant pour chaque nom de propriétaire, ou chaque nom d'entreprise, la liste des véhicules appartenant à ce propriétaire ou à cette entreprise.

   La méthode renverra une map non modifiable.

```
IO.println("test fleet");
IO.println(ferry.fleet());
IO.println();
```

   Sortie attendue :

```
test fleet
{FedEx=[Truck FedEx 1000],
 World Inc=[Truck World Inc 2000, Truck World Inc 2000, Truck World Inc 2000],
 David=[Car David 4 (0)],
 John=[Car John 2 (1), Car John 5 (3), Car John 2 (1)]}
```

   La sortie ci-dessus est donnée sur plusieurs lignes uniquement pour la présentation du sujet.

7. Calcul de la taille de la flotte `fleetSizes()` dans `Ferry`

   Écrire une méthode `public Map<String, Long> fleetSizes()` qui calcule une map donnant, pour chaque nom de propriétaire, ou chaque nom d'entreprise, combien de véhicules possède chaque propriétaire ou entreprise dans le ferry.

   La méthode renverra une map non modifiable.

```
IO.println("test fleetSizes");
IO.println(ferry.fleetSizes());
IO.println();
```

   Sortie attendue :

```
test fleetSizes
{FedEx=1, World Inc=3, David=1, John=3}
```

8. Ajout de constantes dans `Ferry`

   Pour les questions suivantes, on utilisera les constantes suivantes. Ajoutez-les dans la classe `Ferry`.

```
private static final int PASSENGER_FARE = 100;
private static final int TRUCK_FARE = 2;
```

9. Prix du transport `computeFare()` dans `Ferry`

   Le prix du transport pour un véhicule est

   - pour une voiture : le nombre de passagers multiplié par `PASSENGER_FARE`.
   - pour un camion : son poids multiplié par `TRUCK_FARE`.

   Écrire une méthode `public long computeFare()` qui calcule le prix total du transport de tous les véhicules du ferry.

```
IO.println("test computeFare");
IO.println(ferry.computeFare());
IO.println();
```

   Sortie attendue :

```
test computeFare
15300
```

10. `computeFareByFleet()` : prix du transport par nom

    On souhaite calculer le prix du transport en le regroupant par nom de propriétaire pour les voitures, et par nom d'entreprise pour les camions.

    Ajoutez une nouvelle méthode
    `public Map<String, Long> computeFareByFleet()` donnant, pour chaque nom de propriétaire (resp. d'entreprise), le prix du transport de tous les véhicules de sa flotte. La méthode renverra une map non modifiable.

```
IO.println("test computeFareByFleet");
IO.println(ferry.computeFareByFleet());
IO.println();
```

    Sortie attendue :

```
test computeFareByFleet
{FedEx=2000, World Inc=12000, David=400, John=900}
```

    On vérifie que la somme des valeurs est bien la valeur renvoyée par `computeFare()`, 15300.