ESIEE INFO1 Examen Java du 5 juin 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`.
- Vérifiez 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>

Toutes les classes seront écrites dans un package `fr.esiee.farm` dans la partie 1 et dans un package `fr.esiee.farm2` dans la partie 2. On écrira une méthode `main` dans une classe `Main` dans chaque package pour faire les tests. 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. Chaque fois que cela est adapté on **devra utiliser des streams et des lambdas**.

PARTIE 1

On souhaite modéliser une ferme d'un ancien temps qui a des agriculteurs et aussi des vaches. Une des particularités est leur système de calendrier qui compte le temps qui passe en saisons depuis la saison 0. Chaque année compte quatre saisons qui se répètent, hiver, printemps, été et automne. Les saisons commencent ici par l'hiver. On a donc

- saison 0 : hiver
- saison 1 : printemps
- saison 2 : été
- saison 3 : automne
- saison 4 : hiver
- saison 5 : ...

Par exemple, si une vache est née à la saison 2, elle est née à l'été de la première année, tandis qu'une vache née à la saison 12 est née à l'hiver de la 3^{ème} année.

Dans tout ce qui suit, les dates sont toutes calculées depuis la saison 0.

Exercice 1.

Dans un premier temps, on souhaite créer un type pour des vaches (`Cow`). Chaque vache a un nom (`name`) ainsi qu'une date de naissance (`date`) respectant le système de comptage du temps (un entier `int` positif ou nul). Pour afficher une vache, on affiche le nom de la vache suivie de (`Cow`). Implémentez le code de `Cow` en pensant à vérifier les pré-conditions. Voici un exemple de code qui doit fonctionner.

```
System.out.println("Test 1");
var daisy = new Cow("daisy", 2);
System.out.println(daisy);
```

donne

Test 1
daisy (Cow)

Exercice 2. On souhaite maintenant savoir si une vache est plus vieille au sens large qu'une autre vache. Pour cela, nous allons ajouter une méthode `isOlder` qui permet de savoir si la vache courante est plus vieille qu'une vache passée en paramètre.

```
...  
System.out.println("Test 2");  
var ina = new Cow("ina", 12);  
System.out.println(daisy.isOlder(ina));  
System.out.println(ina.isOlder(ina));
```

donne

Test 2
true
true

Note : daisy est née à la 2ième saison, elle est donc plus vieille que ina qui est née à la 12ième saison. Implémentez le code de la méthode `isOlder`.

Exercice 3.

On souhaite maintenant ajouter des agriculteurs (`Farmer`). Implémentez le code de `Farmer` sachant que l'on crée un agriculteur avec juste son nom (`name`) de type `String` et une date de naissance (`date`) respectant le système de comptage du temps (donc un entier positif ou nul). L'affichage d'un agriculteur se fera avec son nom suivi de (`Farmer`).

```
...  
System.out.println("Test 3");  
var gimly = new Farmer("gimly", 10);  
System.out.println(gimly);
```

donne

Test 3
gimly (Farmer)

Exercice 4. On souhaite maintenant créer une ferme (`Farm`) et ajouter des vaches et des agriculteurs à cette ferme. La ferme n'acceptera que ces deux types de résidents. Les résidents seront stockés dans une liste par ordre d'insertion et un résident pourra figurer plusieurs fois dans la liste. On ajoutera une méthode `toString` pour afficher la ferme avec un résident par ligne comme ci-dessous.

```
...  
System.out.println("Test 4");  
var farm = new Farm();  
farm.add(daisy);  
farm.add(ina);  
farm.add(gimly);  
System.out.println(farm);
```

donne

Test 4
daisy (Cow)
ina (Cow)
gimly (Farmer)

Exercice 5.

Écrire une méthode `double meanDate()` qui renvoie la moyenne des dates de naissance (en saisons) des résidents. On lèvera une exception si la ferme est vide.

```
...
System.out.println("Test 5");
System.out.println(farm.meanDate());
```

donne

```
Test 5
8.0
```

Exercice 6.

Écrire une méthode `List<Resident> cows` qui renvoie la liste non modifiable des vaches de la ferme. Dans cette question le type de retour sera `List<Resident>`. On utilisera le pattern matching de types.

```
...
System.out.println("Test 6");
System.out.println(farm.cows());
```

donne

```
Test 6
[daisy (Cow), ina (Cow)]
```

Exercice 7. (Si vous ne savez pas faire cette question, sautez là)

Écrire une méthode `List<Cow> cows2` qui renvoie la liste non modifiable des vaches de la ferme comme à la question précédente, mais dans cette question le type de retour sera `List<Cow>`

```
...
System.out.println("Test 7");
System.out.println(farm.cows2());
```

donne

```
Test 7
[daisy (Cow), ina (Cow)]
```

Exercice 8.

Écrire une méthode `residentsByDate` qui renvoie une map non modifiable dont les clés sont des entiers qui sont les dates des résidents et la valeur associée à une clé est la liste des résidents dont la date est cette clé.

Par exemple ont aura le code suivant.

```
...
System.out.println("Test 8");
farm.add(new Cow("bella", 2));
farm.add(new Cow("marguerite", 12));
farm.add(new Farmer("tho", 12));
farm.add(new Farmer("dwa", 14));
System.out.println(farm.residentsByDate());
```

donne (en passant à la ligne uniquement pour la présentation du sujet)

```
Test 8
{12=[ina (Cow), marguerite (Cow), tho (Farmer)],
 10=[gimly (Farmer)], 2=[daisy (Cow), bella (Cow)], 14=[dwa (Farmer)]}
```

Exercice 9.

Écrire une méthode `int apples()` qui renvoie le nombre de pommes nécessaires pour nourrir la ferme en sachant que chaque agriculteur mange 2 pommes et chaque vache mange autant de pommes que sa date de naissance en saisons.

```
...
System.out.println("Test 9");
System.out.println(farm.apples());
```

donne

Test 9

34

Exercice 10.

Écrire une méthode `Set<Resident> residents(List<Farm> list)` qui prend en paramètre une liste de `Farm` et qui demande l'ensemble de tous les résidents de toutes les fermes de la liste.

```
...
System.out.println("Test 10");
var farm2 = new Farm();
var thorin = new Farmer("thorin", 12);
var dwalin = new Farmer("dwalin", 12);
farm2.add(thorin);
farm2.add(dwalin);
System.out.println(farm2);
System.out.println(Farm.residents(List.of(farm, farm2)));
```

donne

Test 10

thorin (Farmer)

dwalin (Farmer)

[tho (Farmer), gimly (Farmer), ina (Cow), dwalin (Farmer), bella (Cow),
marguerite (Cow), dwa (Farmer), thorin (Farmer), daisy (Cow)]

PARTIE 2

Faites un copier-coller du package `fr.esiee.farm` dans un package `fr.esiee.farm2` et travaillez dans `fr.esiee.farm2` pour cette partie.

Exercice 11.

Dans ce nouveau package, on conserve les records, l'interface mais les classes `Farm` et `Main` sont changées. Les résidents sont stockés dans une map dont les clés sont des dates de naissance en saisons de type `Integer`. La valeur associée à une clé est la liste des résidents dont la date est cette clé.

Modifier la classe en laissant uniquement la map comme champs. Modifier la méthode `add` pour ajouter un résident et la méthode `toString` pour afficher chaque entrée de la map sur une ligne.

```
System.out.println("Test 11");
var daisy = new Cow("daisy", 2);
var ina = new Cow("ina", 12);
var gimly = new Farmer("gimly", 10);
var farm = new Farm();
farm.add(daisy);
farm.add(ina);
farm.add(gimly);
farm.add(new Cow("bella", 2));
farm.add(new Cow("marguerite", 12));
farm.add(new Farmer("tho", 12));
farm.add(new Farmer("dwa", 14));
var thorin = new Farmer("thorin", 12);
var dwalin = new Farmer("dwalin", 12);
farm.add(thorin);
farm.add(dwalin);
System.out.println(farm);
```

donne

```
2: [daisy (Cow), bella (Cow)]
10: [gimly (Farmer)]
12: [ina (Cow), marguerite (Cow), tho (Farmer), thorin (Farmer), dwalin (Farmer)]
14: [dwa (Farmer)]
```

Exercice 12.

Écrire une méthode boolean `isInFarm(Resident resident)` qui renvoie `true` si un résident est dans la ferme et `false` sinon.

```
...
System.out.println("Test 12");
System.out.println(farm.isInFarm(new Cow("marguerite", 12)));
```

donne

```
Test 12
true
```

Exercice 13.

On souhaite ajouter une méthode `forEach(consumer)` à `Farm`, qui prend une fonction `consumer` en paramètre et appelle cette fonction sur chaque résident de la ferme. Écrire le code de la méthode `forEach(consumer)` de telle sorte que le code suivant fonctionne.

```
System.out.println("Test 13");
farm.forEach(System.out::println);
```

donne

```
Test 13
daisy (Cow)
bella (Cow)
gimly (Farmer)
ina (Cow)
marguerite (Cow)
tho (Farmer)
thorin (Farmer)
dwalin (Farmer)
dwa (Farmer)
```