TP : Trier des collections en réalisant des interfaces du JDK

<u>Être capable de</u> trier une collection *pour enrichir le comportement du système*
<u>Se familiariser</u> *avec la documentation du JDK (javadoc)*

Cet exercice de TP reprend le TD du même nom
« Trier des collection en réalisant des interfaces du JDK »
Avoir l'énoncé et la correction de TD sous la main devrait vous aider 😉

Dans cet exercice, nous nous attacherons à montrer qu'en implémentant une interface, une classe va enrichir son comportement et son offre de services (***<u>être capable de</u>*** faire quelque chose de plus…)

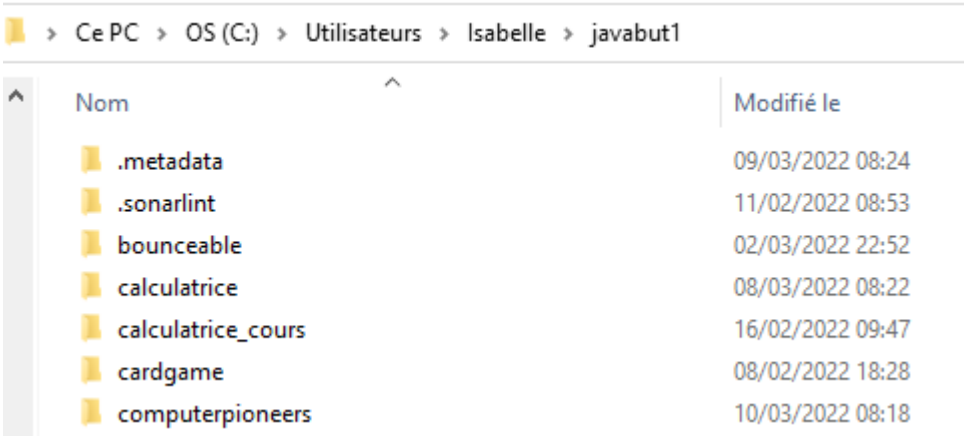
Exercice 1 : Importer un projet dans Eclipse …

Le projet que vous devez récupérer se trouve dans le répertoire ressources du dépôt :
https://github.com/iblasquez/enseignement-but1-developpement

1. **Un peu de git pour commencer :**
   - Si cela n'est pas déjà fait, vous clonerez ce dépôt sur votre machine (local)
   - Si vous avez déjà cloné le dépôt synchroniser le dépôt remote avec votre dépôt local.

   ⇒ Pour effectuer ces manipulations, vous pouvez utilisé la ligne de commande ou GitKraken (en vous aidant du TP Git de la semaine dernière…)

2. Dans un explorateur de fichiers, ouvrez le **`workspace`** dans lequel vous stockez vos projets java. Par exemple, mon chemin est : `C:\Users\Isabelle\javabut1`
   Il faut que vous voyez le **`.metadata`** qui est à la racine de votre **`workspace`**

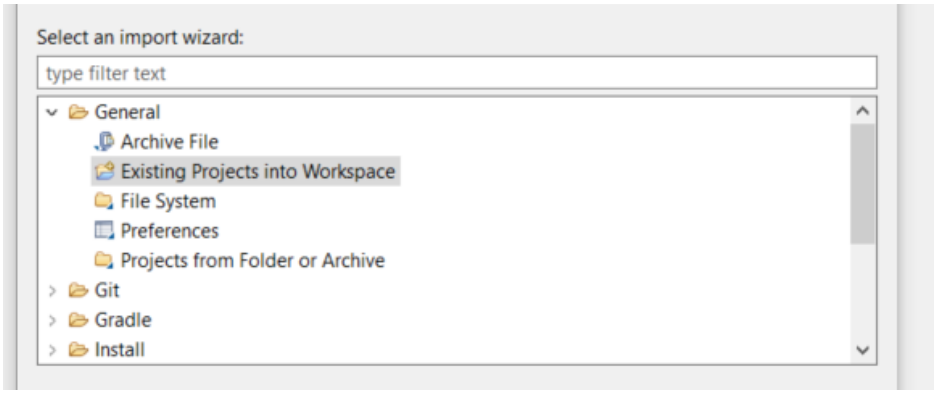


3. Ouvrez un autre explorateur de fichiers et recherchez sur votre ordinateur le dossier **`montagne`** dans le dossier **`ressources`** du dépôt local que vous venez de descendre ou de synchroniser.

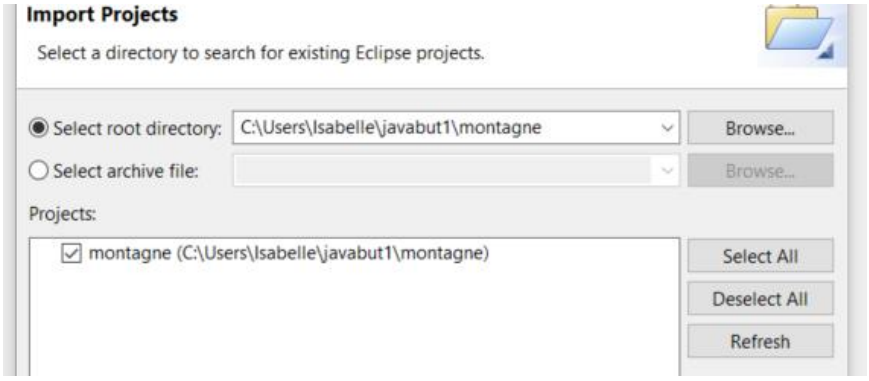
4. Copiez le répertoire montagne et collez-le dans votre **`workspace`**, de manière à ce qu'il se trouve au même niveau que le **`.metadata`** et vos autres projets.

5. Dans votre IDE, choisissez le menu **`File → Import…`**

Puis dans **`General`**, choisissez **`Existing Project into Workspace`**

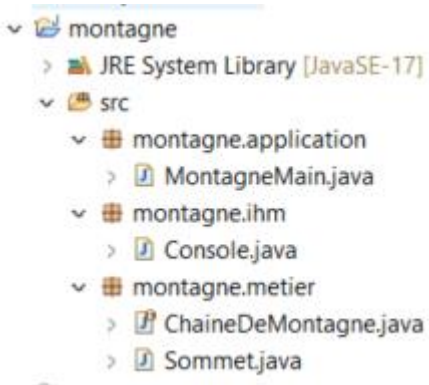


Cliquez sur **`Next,`** puis en cliquant sur **`Browse`** allez chercher votre projet **`montagne`** dans le **`workspace`** dans la première option **`Select root directory`**.

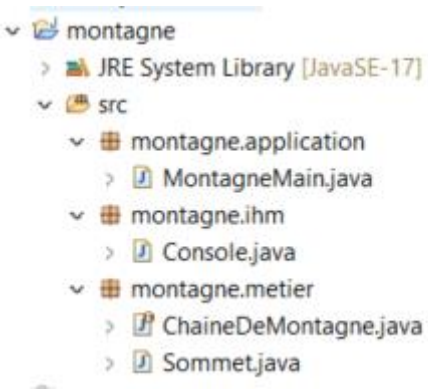


Cliquez sur **Finish**.
Le projet montagne apparaît alors dans la vue **Packahg Explorer** 😉

6. Mettez-vous sur le projet **`montagne,`** fermez tous les autres projets (**close Unrelated Projects**) et dépliez le projet **`montagne`**.



Exercice 2 : Se familiariser avec un projet existant …



La vue **`Package Explorer`**, en indiquant la structure actuelle du projet, vous permet de rentrer rapidement dans ce projet.

Juste en lisant le nom de classes, vous pouvez déjà supposé que le code de ce projet est écrit en français…

➔ Consultez la classe **`Console`**

➔ Dirigez-vous ensuite vers le cœur du projet : le package **`metier`** pour découvrir :

- L'énumération **`ChaineDeMontagne`**
- La classe `Sommet`

```
//public int hashCode() {…}
```

➔ Terminez par la classe **`MontagneMain`** qui est vide pour l'instant.

Exercice 3 : Ouvrir la javadoc !

➔ **Onglet n°1 :** Ouvrir la **page d'accueil de la Javadoc** prête à accueil vos recherches : **https://docs.oracle.com/en/java/index.html**

➔ Dans la barre de recherche de recherche de la barre d'accueil de la javadoc, tapez **`sort`**
Onglet n°2 : Ouvrez le lien de **`Collections`** (Java SE *x* & JDK *x*) dans un nouvel onglet
où *x* est la version de Java que vous utilisée, sûrement la dernière 😉

➔ Dans la barre de recherche de recherche de la barre d'accueil de la javadoc, tapez **`comparator`**

- **Onglet n°3 :** Ouvrez le lien de **`Comparator`** (Java SE *x* & JDK *x*) dans un nouvel onglet
- **Onglet n°4 :** Ouvrez le lien de **`Comparable`** (Java SE *x* & JDK *x*) dans un nouvel onglet

Gardez ces quatre onglets ouverts tout au long du TP et référez-vous à la javadoc aussi souvent que possible !

Exercice 4 : Être capable de trier une collection

(Illustration d'implémentions d'interfaces du **JDK**
qui permettent enrichir le comportement du système)

2. Une collection de sommets

Dans la classe **MontagneMain**, implémentez une collection de **sommets.**
Cette collection de sommets doit vous permettre d'obtenir un affichage similaire à l'affichage suivant sur la console lorsque vous exécutez le projet **montagne** avec comme point d'entrée, le **main** la classe **MontagneMain**

```
-----------------------------------------
Mes sommets
-----------------------------------------
Mont Blanc - altitude : 4809 - Massif du Mont Blanc
Aiguille des Grands Montets - altitude : 3295 - Massif du Mont Blanc
La Rhune - altitude : 905 - Pyrénées
Pic du Midi - altitude : 2877 - Pyrénées
Pic d'Aneto - altitude : 3404 - Pyrénées
Pic du Canigou - altitude : 2785 - Pyrénées
Puy de Sancy - altitude : 1579 - Massif Central
Puy de Dôme - altitude : 1465 - Massif Central
Le Grand Ballon - altitude : 1424 - Massif des Vosges
```

3. Tri d'une collection selon un seul critère : Méthode statique Collections.sort à 1 paramètre & Comparable

a. Documentation Time ! Zoom sur la méthode sort de la classe Collection

Allez dans la javadoc de la **class Collections** et
recherchez la méthode **sort à un seul paramètre** qui indique pour pouvoir utiliser cette méthode, il faut donc que les éléments de la collection à trier implémentent l'interface **Comparable**.

Depuis la signature de la méthode **sort** (dans la rubrique Method Summary) vous pouvez cliquer directement sur **Comparable** pour vous rendre dans la javadoc de cette classe
et prendre connaissance **de(s) méthode(s) abstraite(s**) de cette classe qu'il est nécessaire d'implémenter pour **réaliser l'interface !** …

Remarque : *N'hésitez pas à reprendre l'énoncé de TD pour avoir des explications détaillées sur la javadoc que vous avez sous les yeux* 😉

b. En résumé (ce que nous avons appris en lisant la javadoc) :

Trier une collection de sommets avec la méthode `Collections.sort` à 1 seul paramètre

❑ **Etape n°1** : La classe **`Sommet`** doit implémenter l'interface **`Comparable`**.

⇨ L'interface `Comparable` par la redéfinition de la méthode **`compareTo`** va permettre à la classe `Sommet` d'enrichir son comportement (**être capable de trier** selon un critère).

❑ **Etape n°2** : Une **collection** d'objets de type **`Sommet`** peut désormais être triée par un appel à la méthode **`sort`** de la classe **`Collections`** à un paramètre !

A noter également que seules les listes peuvent être triées 😉

c. Trier la collection de sommets selon une altitude décroissante :

Votre client souhaite pouvoir **trier ses sommets par altitude :**
le tri devant s'effectuer du sommet la plus haute altitude vers le sommet de la plus basse altitude c-a-d **selon un ordre décroissant des altitudes** !

Une fois les sommets ***capable d'être triés*** (c-a-d **`Comparable`** implémentée), écrire l'instruction qui vous permet de trier la collection de sommets de manière à ce que lorsque vous afficherez maintenant votre collection de sommets de manière séquentielle (via un foreach), vous visualiserez sur la console un affichage similaire au suivant.

```
-------------------------------------------
Mes sommets triés par altitude décroissante
-------------------------------------------
Mont Blanc - altitude : 4809 - Massif du Mont Blanc
Pic d'Aneto - altitude : 3404 - Pyrénées
Aiguille des Grands Montets - altitude : 3295 - Massif du Mont Blanc
Pic du Midi - altitude : 2877 - Pyrénées
Pic du Canigou - altitude : 2785 - Pyrénées
Puy de Sancy - altitude : 1579 - Massif Central
Puy de Dôme - altitude : 1465 - Massif Central
Le Grand Ballon - altitude : 1424 - Massif des Vosges
La Rhune - altitude : 905 - Pyrénées
```

4. Disposer de plusieurs critères de tri pour trier une collection au gré de ses envies Méthode statique `Collections.sort` à 2 paramètres & `Comparator`

a. Toujours lire la documentation avant de commencer à implémenter 😉

Allez dans la javadoc de la **`class Collections`** et recherchez
la méthode **`sort` à un deux paramètres** qui indique pour pouvoir utiliser cette méthode,
il n'y a rien à modifier sur les éléments de la collection à trier
mais le second paramètre sera le critère de tri et devra implémenter l'interface **`Comparator`**

Depuis la signature de la méthode **`sort`** (dans la rubrique Method Summary) vous pouvez cliquer directement sur **`Comparator`** pour vous rendre dans la javadoc de cette classe
et prendre connaissance de(s) **méthode(s) abstraite(s)** de cette classe qu'il est nécessaire d'implémenter pour **réaliser l'interface !** …

Remarque : *N'hésitez pas à reprendre l'énoncé de TD pour avoir des explications détaillées sur la javadoc que vous avez sous les yeux* 😉

b. En résumé (ce que nous avons appris en lisant la javadoc)

Trier une collection de sommets avec la méthode `Collections.sort` à 2 paramètres

❑ **Etape n°1** : Un comparateur doit être implémenté par critère de tri.
Ce **comparateur** est une classe qui doit redéfinir la méthode ……..
de ……..

⇨ Cette fois-ci, **c'est le comparateur qui est « responsable du tri »** (qui est **capable de trier selon un critère**) car il connaît le comportement à adopter (**`compare`**)
si on l'appelle pour trier une collection.

❑ **Etape n°2** : Une collection peut être triée selon le critère de tri que l'on souhaite dès lors que la méthode **`sort`** sera appelée avec **2 paramètres** :
⇨ La collection à trier
⇨ Le comparateur capable de mettre en place le critère de tri souhaité sur la collection.

c. Trier les sommets par altitude

❑ Implémentez dans la classe **`ComparateurAltitude`** un comparateur qui permet de **trier la collection de sommets par altitude croissante (« ordre naturel de l'altitude »).**

Puis dans la méthode **`main`**, implémentez l'instruction qui permet de trier la collection sommets avec ce comparateur de manière à obtenir après l'exécution de cette instruction un affichage de la collection de sommets similaire à :

```
-----------------------------------------
Mes sommets triés par altitude croissante
-----------------------------------------
La Rhune - altitude : 905 - Pyrénées
Le Grand Ballon - altitude : 1424 - Massif des Vosges
Puy de Dôme - altitude : 1465 - Massif Central
Puy de Sancy - altitude : 1579 - Massif Central
Pic du Canigou - altitude : 2785 - Pyrénées
Pic du Midi - altitude : 2877 - Pyrénées
Aiguille des Grands Montets - altitude : 3295 - Massif du Mont Blanc
Pic d'Aneto - altitude : 3404 - Pyrénées
Mont Blanc - altitude : 4809 - Massif du Mont Blanc
```

Remarque : En principe, les comparateurs sont **implementés « à la volée »** c-a-d au moment où on a en besoin, ou plutôt dans la classe qui les utilisent.
Dans le cadre du projet **`montagne`**, le comparateur sera implémenté dans le fichier **`MontagneMain.java`** (rappel : fichier du même nom de la classe qui doit être **`public`**)
Comme, il ne peut y avoir qu'une seule classe public par fichier en Java, on ne mettre pas le mot `public` devant un comparateur qu'on décide d'implémenter à la volée, mais on écrira simplement :

```
class ComparateurAltitude ……{
      //…
}
```

❑ Dans votre `main`, écrivez maintenant le code qui permet de **trier la collection de sommets par altitude décroissante** en utilisant un comparateur pour le tri .
Ne vous lancez tête baissée dans ce code, mais consultez la javadoc avant 😉 …

d. Trier les sommets par nom

Implémentez maintenant un comparateur (**ComparateurNom)** qui permet de trier la collection de sommets **par nom selon l'ordre alphabétique**.

Implémentez ensuite dans le `main`, l'instruction qui permet de trier la collection sommets avec ce comparateur de manière à obtenir après l'exécution de cette instruction un affichage de la collection de sommets similaire à :

```
-----------------------------------------
Mes sommets triés par nom (ordre alpha)
-----------------------------------------
Aiguille des Grands Montets - altitude : 3295 - Massif du Mont Blanc
La Rhune - altitude : 905 - Pyrénées
Le Grand Ballon - altitude : 1424 - Massif des Vosges
Mont Blanc - altitude : 4809 - Massif du Mont Blanc
Pic d'Aneto - altitude : 3404 - Pyrénées
Pic du Canigou - altitude : 2785 - Pyrénées
Pic du Midi - altitude : 2877 - Pyrénées
Puy de Dôme - altitude : 1465 - Massif Central
Puy de Sancy - altitude : 1579 - Massif Central
```

e. Trier les sommets par chaine de montagnes

Implémentez maintenant un comparateur (**ComparateurChaineDeMontagne)** qui permet de trier la collection de sommets par le nom des chaines de montagne (ordre alphabétique).

Implémentez ensuite dans le **main**, l'instruction qui permet de trier la collection sommets avec ce comparateur de manière à obtenir après l'exécution de cette instruction un affichage de la collection de sommets similaire à :

```
-----------------------------------------
Mes sommets triés par chaine de montagne
-----------------------------------------
Puy de Dôme - altitude : 1465 - Massif Central
Puy de Sancy - altitude : 1579 - Massif Central
Le Grand Ballon - altitude : 1424 - Massif des Vosges
Aiguille des Grands Montets - altitude : 3295 - Massif du Mont Blanc
Mont Blanc - altitude : 4809 - Massif du Mont Blanc
La Rhune - altitude : 905 - Pyrénées
Pic d'Aneto - altitude : 3404 - Pyrénées
Pic du Canigou - altitude : 2785 - Pyrénées
Pic du Midi - altitude : 2877 - Pyrénées
```

f. Trier les sommets par chaine de montagnes et par altitude décroissante (tri multi-critère)

Un comparateur peut même être capable d'effectuer un **tri multicritère.**
Le dernier souhait de notre client est qu'il puisse **Trier ses sommets par chaine de montagnes *et* par altitude décroissante** (à l'intérieur de chaque chaine de montagnes)

Implémentez maintenant un comparateur (**`ComparateurChaineAltitude`**) qui permet de trier la collection de sommets par chaine de montagne et par altitude au sein d'une même chaine de montagne.

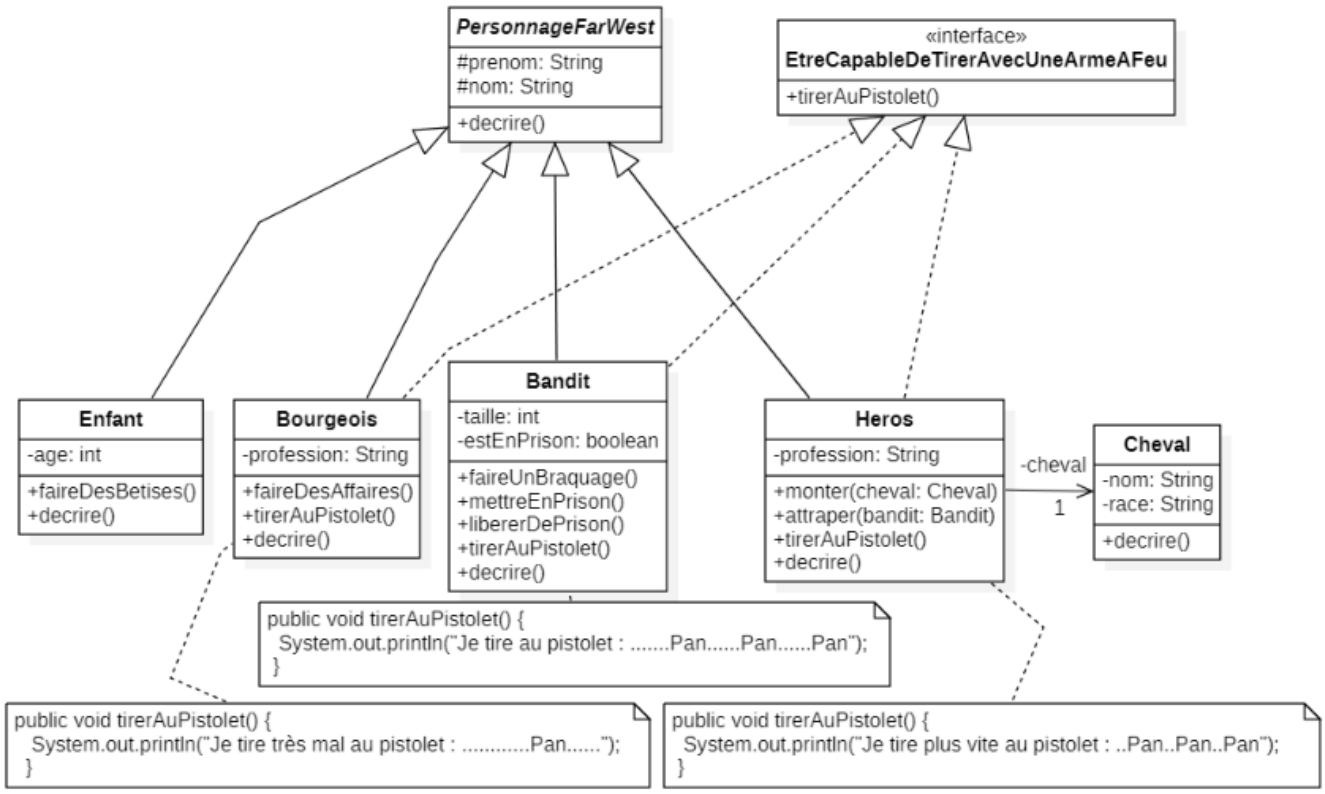
```
-----------------------------------------
Mes sommets par chaine et altitude décroissante
-----------------------------------------
Puy de Sancy - altitude : 1579 - Massif Central
Puy de Dôme - altitude : 1465 - Massif Central
Le Grand Ballon - altitude : 1424 - Massif des Vosges
Mont Blanc - altitude : 4809 - Massif du Mont Blanc
Aiguille des Grands Montets - altitude : 3295 - Massif du Mont Blanc
Pic d'Aneto - altitude : 3404 - Pyrénées
Pic du Midi - altitude : 2877 - Pyrénées
Pic du Canigou - altitude : 2785 - Pyrénées
La Rhune - altitude : 905 – Pyrénées
```

Exercice 5 : Un dernier petit tour au Rendez-vous au Far West

1. Si vous n'avez pas fini **l'exercice 2 Rendez-vous au Far West …** du TP ***Implémenter des interfaces à partir d'un diagramme de classes simple (Implémenter des conceptions simples),*** terminez-le avant de passer à la question 2.

Rappel de cet énoncé :

❑ Rendez-vous dans votre projet **`farwest`** ou créez le, si vous n'en disposez pas encore d'un 😉
❑ Implémentez le diagramme de classes suivant
(vous vérifierez l'architecture avec une petite **rétro-conception** 😉)



Remarques :

→ Pour les méthode `faireDesBetises`, `faireDes Affaires` et `faireUnBraquage` , un simple `//TODO` suffira comme implémentaion.

→ Pour la méthode `décrire` le résultat attendu à la console donné ci-dessous devrait vous aider 😉

❑ Récupérerez la classe **`FarWestMain`** disponible sur le **gist** : https://unil.im/farwest (https://gist.github.com/iblasquez/5597413cb37a30a2e7e8129f2d6aedf1)

Faites-en sorte que ce code compile, puis exécutez-le pour vérifier et valider le comportement de votre application. Si votre implémentation est correcte, vous devriez obtenir sur la console un affichage similaire à :

```
--------------------------------------------------
Les personnages de la caravanne sont :
Lucky Luke! Je suis cow-boy et mon cheval est Jolly Jumper de race appaloosa
Joe Dalton! Je mesure 150 cm et je suis Libre
Averell Dalton! Je mesure 190 cm et je suis Libre
Zacharie Martins! Je suis inventeur
Phineas ! J'ai 10ans
--------------------------------------------------
Les personnages capable de tirer au pistolet sont :
Lucky Luke ! Je tire plus vite au pistolet : ..Pan..Pan..Pan
Joe Dalton ! Je tire au pistolet : .......Pan......Pan......Pan
Averell Dalton ! Je tire au pistolet : .......Pan......Pan......Pan
Zacharie Martins ! Je tire très mal au pistolet : ............Pan......
```

2. Complétez le programme **`FarWestMain`** de manière que les 4 Bandits Dalton soient instanciés. Puis proposez un classement des Daltons selon une taille croissante, puis selon une taille décroissante. En vous inspirant de l'exercice précédent sur les montagnes, pour effectuer cette comparaison, vous devez implémenter une interface du JDK : vous êtes libre de choisir la solution qui vous convient 😉

Effet Tampon :

S'il vous reste du temps sur cette séance, profitez-en pour finir :

- tout autre exercice des TPs précédents qui n'aurait pas été terminé.
- si tous les TPs sont terminés, profitez du temps restant pour avancer sur votre **S.A.E** 😉