TP02

**Contraintes impératives**

Tous les affichages doivent être strictement identiques à ceux produit dans ce document. Cela fera partie des critères d’évaluation.

## PictureOrganizer

Dans cet exercice, il vous est demandé de finir le développement d’un lecteur d’image. Le projet actuel est composé de 3 classes dont le détail est :

* PictureOrganizer

C’est la classe principale permettant d’effectuer toutes les actions (lecture de l’image, previous, next, lectureAleatoire).

* PictureReader

Cette classe permet de lire et stocker dans un ArrayList les images. Aucun travail n’est demandé dans cette classe.

* PicturePlayer

Cette classe est en charge de créer une fenêtre pour l’affichage. Aucun travail n’est demandé dans cette classe.

Picture

String picturePath

getPicturePath()

La première étape consiste à compléter une classe Picture ayant comme seul attribut PicturePath qui est le chemin d’accès au fichier qui sera fourni lors de la création d’une instance de Picture. Cette classe aura une méthode getPicturePath() qui permettra d’obtenir le chemin d’accès au fichier. (Cela résoudra les erreurs).

Il vous est ensuite demandé de développer les fonctions previous, next et lectureAleatoire dans la classe PictureOrganizer.

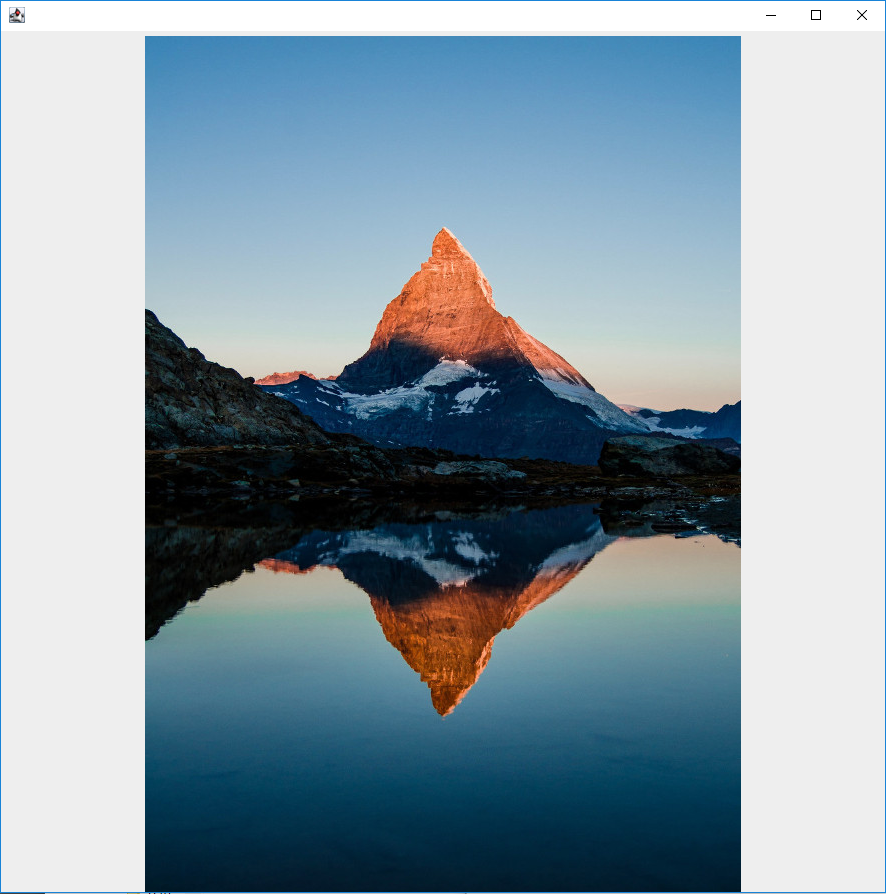
Pour le previous et le next, si aucun affichage d’image a été réalisé depuis le début de l’exécution, le message « You have to display a picture first ! » doit être affiché dans la console.

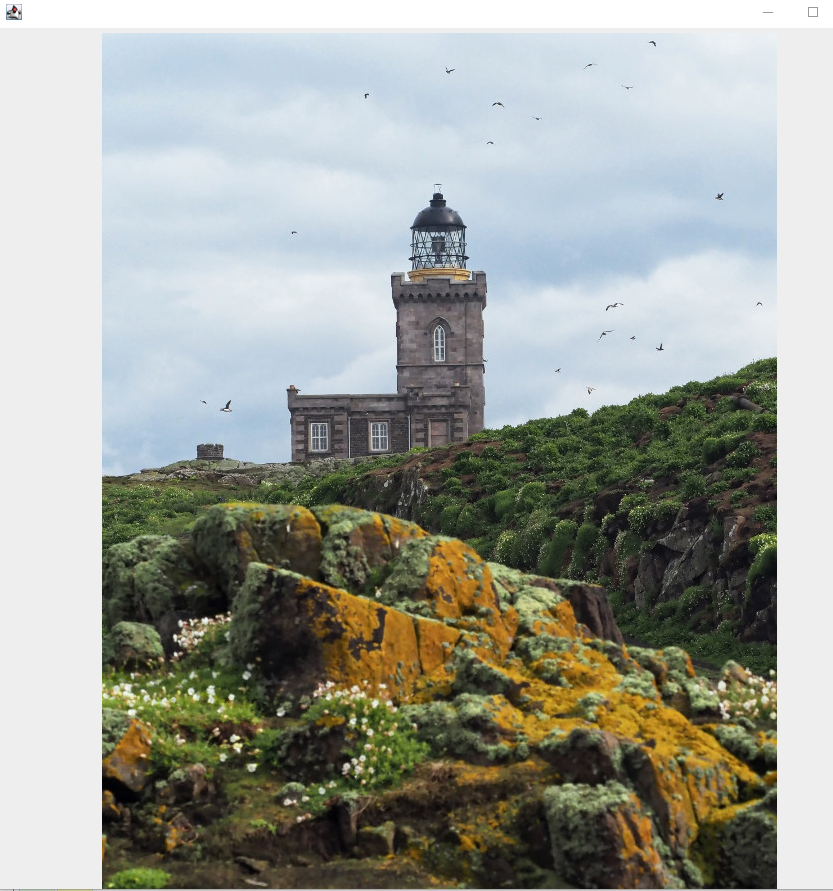
Pour la lectureAléatoire, cette fonction doit mélanger l’ArrayList pictures et affectera donc les résultats obtenus avec next et previous.

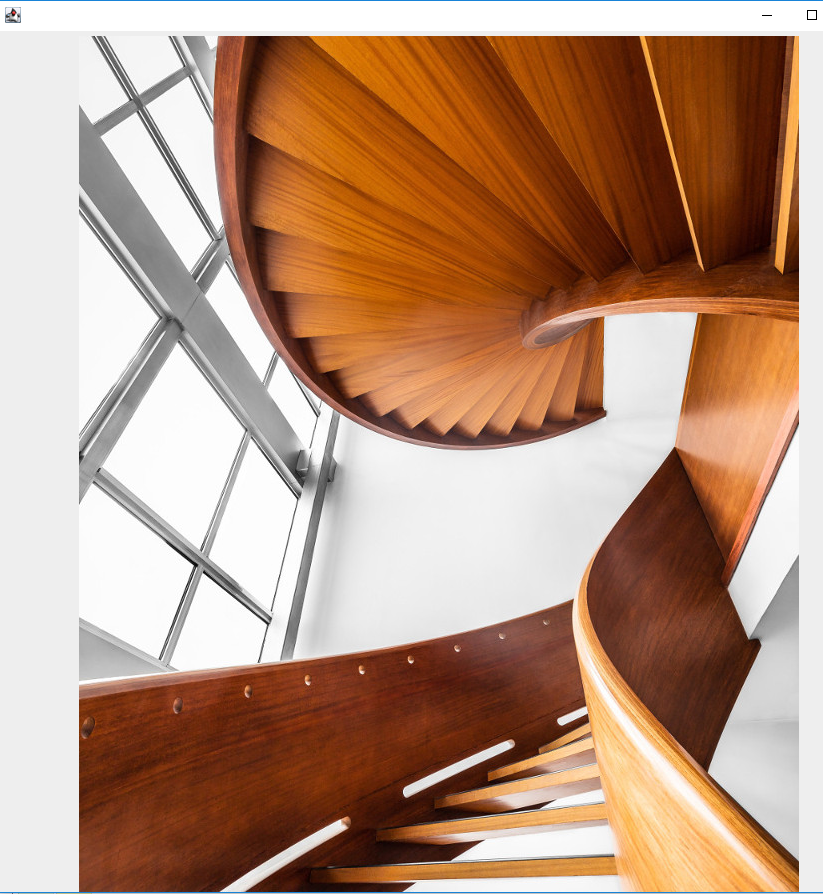
Il est conseillé de s’inspirer sur le projet music\_organizer\_v5 vu en cours avec M.Teodoro car cela peut vous aider.

Exemples de sorties :

(Il faut exécuter les méthodes dans cet ordre et sans lectureAléatoire pour atteindre ces résultats)

 (Après l’appui sur showFirstPicture() )

 (Après l’appui sur next() )

 (Après 2 fois l’appui sur previous() )

## GestionnairePersonne

Dans cet exercice il vous est demandé d’implémenter des outils pour gérer une liste de personne. Pour faire cela vous devrez compléter la classe Personne ainsi :

GestionnairePersonne

ArrayList aLPersonnes<Personne>

Personne

String nom

String prenom

int age

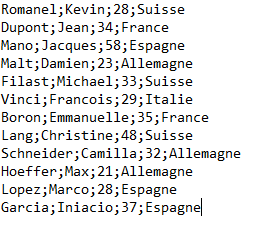
String nationalite

Dans la classe GestionnairePersonne, il vous est demandé d’implémenter les fonctions :

* chargerPersonnes()

Cette fonction va permettre de remplir l’ArrayList personnes d’instance de Personne par rapport au contenu du fichier texte donné en paramètre.

Le paramètre String textePersonnes de cette méthode contient toutes les informations pour créer des instances de Personne. Celles-ci sont sous le format suivant :



Chaque ligne est donc séparée par un retour à la ligne.

* afficheMoinsTrente()

Cette fonction va afficher les personnes ayant moins de 30 ans en utilisant la fonction toString() de Personne.

* afficheNationalite()

Cette fonction va afficher toutes les personnes de l’ArrayList groupées par nationalité en utilisant la fonction toString() de Personne.

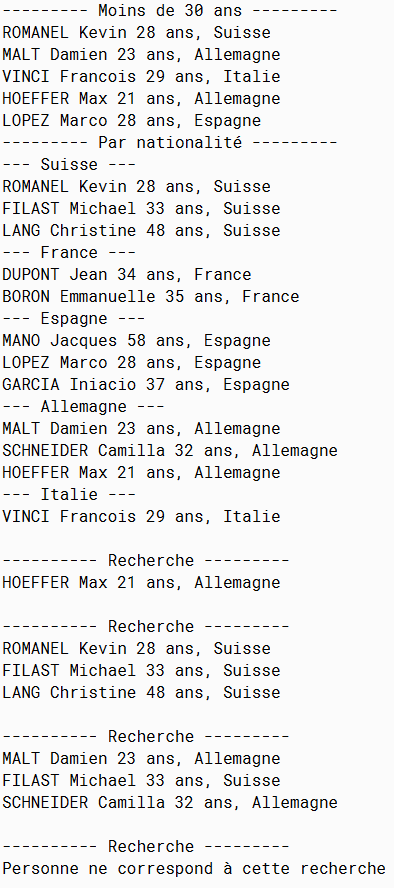
* recherchePersonne()

Cette fonction va afficher toutes les personnes comportant (mais pas forcément 100% identique) la valeur donnée en paramètre (que ce soit l’age, le nom, le prenom ou la nationalité) en utilisant la fonction toString() de Personne.

Par exemple la recherche de « Jean » devrait retourner « Jean-Vincent » comme résultat également (il ne figure pas dans la liste de personne).

Exemples de sorties :

Exécution de la méthode main()



Recherche de « Hoeffer »

Recherche de « Suisse »

Recherche de « mi »

Recherche de « Test »

## Garage

Programme permettant de gérer les entrées et sorties d’un Garage.

La classe Voiture contient les attributs suivants :

Voiture

String marque

String modele

int annee

String plaqueImmatriculation

Int anneeRevision

La classe Garage contient plusieurs fonctions à développer :

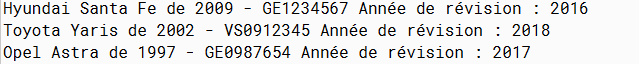
* entreeVoiture qui modélise l’entrée d’une voiture dans le garage en prenant une instance de voiture comme paramètre
* sortieVoiture qui modélise la sortie d’une voiture du garage en prenant une instance de voiture comme paramètre
* reviserToutesLesVoitures qui révise chaque voiture présente dans le garage (met l’année de révision à 2020)
* afficherVoitures qui affiche chaque voiture présente dans le garage
* voiturePresente qui test grâce à une plaqueImmatriculation passée en paramètre si le véhicule est présent dans le garage ou non au moyen de equals.

Il vous faudra donc implémenter des fonctions supplémentaires dans Voiture afin de pouvoir réaliser les tâches ci-dessus (notamment toString et equals).

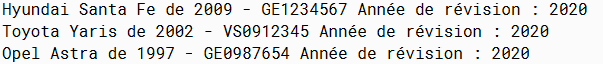
De plus, la liste des voitures dans le garage sera modélisée par une ArrayList voitures définie comme variable d’instance de Garage.

Exemples de sorties :

Après l’ajout de 3 instances de véhicules, afficherVoiture() :



Après l’appel de reviserToutesLesVoitures(), afficherVoiture() :



Appel de voiturePresente() en donnant « GE0987654 » :



Après la sortie de l’Opel « GE0987654 » du garage, rappel de voiturePresente() en donnant « GE0987654 » :

