632-1 - Algorithmes et structures de données
Teodoro Douglas, Stettler Christian, Humbert Jérôme,
David Issom

h e g | Haute école de gestion de Genève

TP02

Contraintes impératives

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

1. 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.

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).

Picture
String picturePath
getPicturePath()

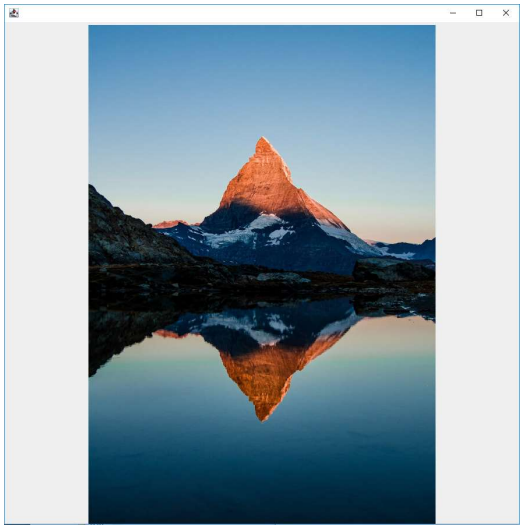
Il vous est ensuite demandé de développer les fonctions `previous`, `next` et `lectureAleatoire` dans la classe `PictureOrganizer`.

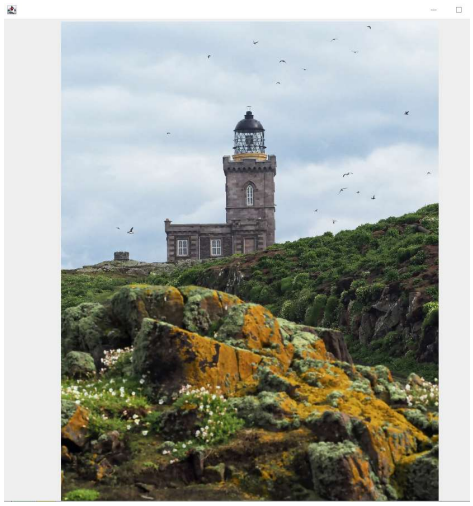
Pour le `previous` et le `next`, si aucun affichage d'image a été réalisé depuis le début de l'exécution, le message « You have to display a picture first ! » doit être affiché dans la console.

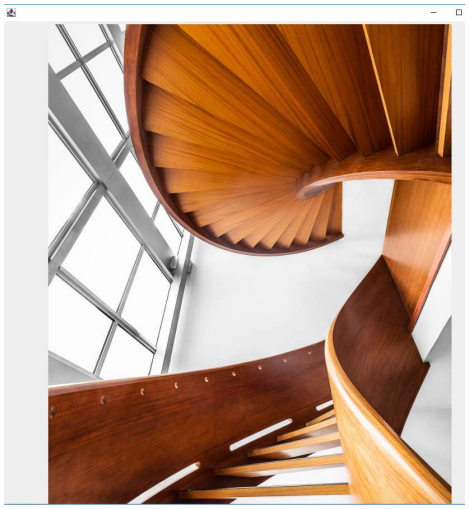
Pour la `lectureAléatoire`, cette fonction doit mélanger `l'ArrayList pictures` et affectera donc les résultats obtenus avec `next` et `previous`.

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)

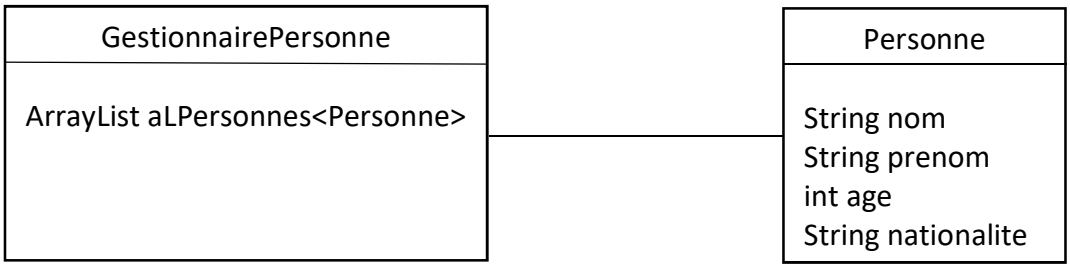

(Après l'appui sur `showFirstPicture()`)


(Après l'appui sur `next()`)


(Après 2 fois l'appui sur `previous()`)

2. 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 :



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 donnees` de cette méthode contient toutes les informations pour créer des instances de `Personne`. Celui-ci est sous le format suivant :

```
Romanel;Kevin;28;Suisse
Dupont;Jean;34;France
Mano;Jacques;58;Espagne
Malt;Damien;23;Allemagne
Filast;Michael;33;Suisse
Vinci;Francois;29;Italie
Boron;Emmanuelle;35;France
Lang;Christine;48;Suisse
Schneider;Camilla;32;Allemagne
Hoeffer;Max;21;Allemagne
Lopez;Marco;28;Espagne
Garcia;Iniacio;37;Espagne
```

  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()`

```
--------- Moins de 30 ans ---------
ROMANEL Kevin 28 ans, Suisse
MALT Damien 23 ans, Allemagne
VINCI Francois 29 ans, Italie
HOEFFER Max 21 ans, Allemagne
LOPEZ Marco 28 ans, Espagne
--------- Par nationalité ---------
--- Suisse ---
ROMANEL Kevin 28 ans, Suisse
FILAST Michael 33 ans, Suisse
LANG Christine 48 ans, Suisse
--- France ---
DUPONT Jean 34 ans, France
BORON Emmanuelle 35 ans, France
--- Espagne ---
MANO Jacques 58 ans, Espagne
LOPEZ Marco 28 ans, Espagne
GARCIA Iniacio 37 ans, Espagne
--- Allemagne ---
MALT Damien 23 ans, Allemagne
SCHNEIDER Camilla 32 ans, Allemagne
HOEFFER Max 21 ans, Allemagne
--- Italie ---
VINCI Francois 29 ans, Italie

---------- Recherche ---------
HOEFFER Max 21 ans, Allemagne

---------- Recherche ---------
ROMANEL Kevin 28 ans, Suisse
FILAST Michael 33 ans, Suisse
LANG Christine 48 ans, Suisse

---------- Recherche ---------
MALT Damien 23 ans, Allemagne
FILAST Michael 33 ans, Suisse
SCHNEIDER Camilla 32 ans, Allemagne

---------- Recherche ---------
Personne ne correspond à cette recherche
```

Recherche de « Hoeffer »

Recherche de « Suisse »

Recherche de « mi »

Recherche de « Test »

3. 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`() :

```
Hyundai Santa Fe de 2009 - GE1234567 Année de révision : 2016
Toyota Yaris de 2002 - VS0912345 Année de révision : 2018
Opel Astra de 1997 - GE0987654 Année de révision : 2017
```

Après l'appel de `reviserToutesLesVoitures`(), `afficherVoiture`() :

```
Hyundai Santa Fe de 2009 - GE1234567 Année de révision : 2020
Toyota Yaris de 2002 - VS0912345 Année de révision : 2020
Opel Astra de 1997 - GE0987654 Année de révision : 2020
```

Appel de `voiturePresente()` en donnant « GE0987654 » :

```
Opel Astra de 1997 - GE0987654 Année de révision : 2020 est présente dans le garage.
```

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

```
La voiture ayant la plaque GE0987654 n'est pas présente.
```