Pratique de la programmation orientée-objet
Examen final

3 juin 2022

Indications :

— l'examen dure de 12h15 à 16h00,

— indiquez votre nom, prénom et numéro SCIPER ci-dessous et sur toutes les éventuelles feuilles additionnelles que vous rendriez,

— placez votre carte d'étudiant sur la table.

Seuls les documents suivants sont autorisés, en version papier uniquement :

— les notes de cours,

— les énoncés et les corrigés des séries d'exercices,

— une feuille de résumé de format A4 au maximum, ne couvrant que le cours.

Aucun document concernant le projet n'est autorisé !

Bon travail !

Nom : _____

Prénom : _____

SCIPER : _____

1 Liste de points [38 points]

Dans le projet, les points d'une route sont représentés par une valeur de type `List<PointCh>`. Bien que cette solution fonctionne, elle utilise beaucoup de mémoire pour représenter relativement peu d'information, car chaque point est stocké dans un objet séparé, et chacun de ces objets contient deux attributs de type **`double`**. Le but de cet exercice est de développer une représentation plus compacte d'une telle liste de points.

L'idée est de stocker les coordonnées de tous les points de la liste dans un seul tableau de type **`float[]`**. Ces coordonnées sont représentées comme des différences par rapport à l'origine du système suisse, dont les coordonnées sont $(E = 2\,600\,000, N = 1\,200\,000)$.

Par exemple, une liste de deux points dont les coordonnées sont :

$$[(E = 2\,600\,010, N = 1\,199\,900), (E = 2\,533\,000, N = 1\,152\,000)]$$

serait représentée à l'aide du tableau suivant :

```
new float[]{ 10, -100, -67_000, -48_000 };
```

La classe immuable `PointChList` ci-dessous, à compléter, représente une liste de points en utilisant cette idée, les coordonnées des points étant stockées dans l'attribut `cs` :

```
public final class PointChList implements Iterable<PointCh> {
  private static final double ORIGIN_E = 2_600_000;
  private static final double ORIGIN_N = 1_200_000;

  private final float[] cs;

  private PointChList(float[] cs) { this.cs = cs; }
  … méthodes (à faire)
}
```

Notez que le constructeur de la classe est privé, et ne copie pas le tableau de coordonnées qu'il reçoit en argument. Par conséquent, pour assurer l'immuabilité de la classe, tout code invoquant le constructeur doit garantir que le tableau qu'il lui passe en argument n'est jamais modifié par la suite.

Partie 1 [5 points] Ajoutez à `PointChList` une méthode statique appelée `of`, prenant en argument une liste de points de type `List<PointCh>` et retournant une instance de `PointChList` représentant la même liste de points.

Par exemple, la liste contenant les deux points mentionnés ci-dessus pourrait être créée comme suit en utilisant cette méthode :

```
PointCh p1 = new PointCh(2_600_010, 1_199_900);
PointCh p2 = new PointCh(2_533_000, 1_152_000);
PointChList twoPoints = PointChList.of(List.of(p1, p2));
```

Souvenez-vous que `PointCh` est un enregistrement avec deux attributs de type **`double`** contenant les coordonnées, appelés `e` et `n`.

Réponse :

Réponse (suite) :

Partie 2 [6 points] Ajoutez à `PointChList` les quatre méthodes suivantes :

— `size`, qui ne prend aucun argument et retourne le nombre de points contenus dans la liste,

— `e`, qui prend l'index d'un point dans la liste et retourne sa coordonnée E (Est) sous la forme d'un **`double`**, ou lève une `IndexOutOfBoundsException` si l'index est invalide,

— `n`, similaire à `e` mais pour la coordonnée N (nord),

— `get`, qui prend l'index d'un point dans la liste et le retourne sous la forme d'un `PointCh`, ou lève une `IndexOutOfBoundsException` si l'index est invalide.

Par exemple, appliquée à la variable `twoPoints` définie ci-dessus, `size` devrait retourner 2, et lorsqu'on lui donne l'index 0, `e` devrait retourner 2 600 010, `n` devrait retourner 1 199 900, et `get` devrait retourner un point égal à `p1`.

Réponse :

Partie 3 [4 points] Selon vous, la classe `PointChList` doit-elle redéfinir les méthodes `equals` et `hashCode` ? Si oui, écrivez-les, sinon expliquez pourquoi.

Réponse :

Partie 4 [5 points] Ajoutez à `PointChList` une redéfinition de la méthode `toString`. Dans la chaîne de caractères qu'elle renvoie, chaque point doit être représenté par la paire de ses coordonnées E/N séparées par une virgule et entourées de parenthèses. Les points doivent être séparés les uns des autres par une virgule et un espace, et entourés de crochets.

Par exemple, appliquée à la variable `twoPoints` définie ci-dessus, `toString` devrait produire la chaîne `[(2600010.0,1199900.0), (2533000.0,1152000.0)]`

Pour obtenir la représentation textuelle d'une coordonnée, utilisez la méthode (statique) `toString` de `Double`, ou quelque chose d'équivalent.

Réponse :

Partie 5 [9 points] Ajoutez à `PointChList` une redéfinition de la méthode `iterator` de `Iterable`. La classe étant immuable, la méthode `remove` de l'itérateur qu'elle retourne doit simplement lever une `UnsupportedOperationException`.

Réponse :

Partie 6 [9 points] Ajoutez à `PointChList` un bâtisseur, sous la forme d'une classe imbriquée statiquement nommée `Builder` et n'offrant que deux méthodes :

— `Builder add(double e, double n)`, qui ajoute un point avec les coordonnées données à la fin de la liste en cours de construction, et retourne le récepteur (`this`),

— `PointChList build()`, qui retourne une liste de points contenant tous les points ajoutés jusqu'à présent avec `add`.

Votre bâtisseur *doit* stocker les coordonnées qu'il reçoit comme des différences par rapport à l'origine du système de coordonnées suisse, dans un tableau de type `float[]` dont la taille croît (par copie) en fonction des besoins.

En utilisant ce bâtisseur, la liste `twoPoints` définie ci-dessus pourrait être définie de manière équivalente comme suit :

```
PointChList twoPoints = new PointChList.Builder()
  .add(2_600_010, 1_199_900)
  .add(2_533_000, 1_152_000)
  .build();
```

Réponse :

2 Fournisseur de tuiles [25 points]

Dans le projet, les tuiles de la carte sont téléchargées depuis les serveurs d'OpenStreetMap, puis mises en cache sur le disque et en mémoire par une classe unique et monolithique appelée `TileManager`. Le but de cet exercice est d'explorer une approche plus « compositionnelle », basée sur l'interface `TileProvider` ci-dessous, qui représente un objet capable de fournir une tuile en fonction de son identité (niveau de zoom et index x/y) :

```
public interface TileProvider {
  record TileId(int zoom, int x, int y) { }
  byte[] imageForTileAt(TileId tileId) throws IOException;
}
```

La méthode `imageForTileAt` est très similaire à celle du projet, la seule différence étant qu'elle retourne les données de la tuile sous la forme d'un tableau d'octets plutôt que d'une image. Notez que, comme tous les tableaux Java, celui-ci est modifiable, mais il ne doit *jamais* être modifié, et peut être traité comme immuable. Il n'est donc jamais nécessaire de le copier.

Nous supposons l'existence des trois classes ci-dessous, qui implémentent toutes l'interface `TileProvider` :

1. `NetworkTP`, un fournisseur de tuiles qui les télécharge depuis un serveur,
2. `DiskCachingTP`, un fournisseur de tuiles qui met en cache disque les tuiles fournies par un autre fournisseur,
3. `MemoryCachingTP`, un fournisseur de tuiles qui met en cache mémoire les tuiles fournies par un autre fournisseur.

`NetworkTP` offre deux constructeurs : le premier prend le nom du serveur sous forme d'une chaîne de caractères, le second ne prend aucun argument et appelle le premier avec le nom du serveur principal d'OpenStreetMap (`tile.openstreetmap.org`).

`DiskCachingTP` et `MemoryCachingTP` stockent dans un cache — disque ou mémoire — les tuiles fournies par un autre fournisseur passé à leur constructeur comme seul argument. Lorsqu'une tuile demandée est présente dans le cache, elle est simplement retournée, sinon elle est obtenue de cet autre fournisseur, stockée dans le cache et finalement retournée.

Partie 1 [6 points] En utilisant les trois classes susmentionnées, montrez comment compléter la définition de la variable `tileProvider` ci-dessous afin qu'elle contienne un fournisseur qui, lorsqu'on lui demande une tuile :

1. essaie tout d'abord de l'obtenir du cache mémoire,
2. si cela échoue, essaie de l'obtenir du cache disque,
3. si cela échoue également, l'obtient du serveur principal d'OpenStreetMap.

Les tuiles obtenues à partir du serveur sont mises en cache à la fois sur le disque et en mémoire, tandis que celles obtenues à partir du cache disque sont mises en cache en mémoire.

```
TileProvider tileProvider =
```

Partie 2 [13 points] Écrivez une classe instanciable appelée `ErrorSilencingTP` implémentant l'interface `TileProvider` et dont le constructeur prend deux arguments : un fournisseur de tuiles de type `TileProvider` et un tableau de type **`byte`**`[]`.

La méthode `imageForTileAt` de cette classe doit utiliser le fournisseur donné à son constructeur pour obtenir les tuiles et les retourner simplement, sauf si une `IOException` est levée, auquel cas elle doit retourner le tableau donné à son constructeur à la place.

Votre classe devrait être utilisable comme suit pour créer un fournisseur de tuiles obtenant ses tuiles depuis le réseau mais retournant une tuile par défaut — sur laquelle un message d'erreur pourrait par exemple figurer — lorsqu'une exception d'entrée/sortie se produit.

```
byte[] ERROR_TILE_BYTES = new byte[] { ... };
TileProvider p = new ErrorSilencingTP(new NetworkTileProvider(),
                                      ERROR_TILE_BYTES);
```

Notez bien que la méthode `imageForTileAt` de votre classe ne doit *jamais* lever une exception de type `IOException`.

Réponse :

Partie 3 [6 points] Nommez les patrons de conception utilisés par :

1. `DiskCachingTP` : _____
2. `MemoryCachingTP` : _____
3. `ErrorSilencingTP` : _____

3 Empaquetage d'entiers [35 points]

Dans plusieurs parties du projet, nous avons emballé plusieurs petites valeurs entières dans une seule valeur de type **`int`** (32 bits).

Par exemple, le nombre d'arêtes sortantes d'un nœud du graphe et l'index de la première d'entre elles ont été empaquetés dans une seule valeur de type **`int`**. Les 4 bits les plus significatifs de cette valeur contiennent le nombre d'arêtes sortantes, tandis que les 28 bits restants contiennent l'index de la première d'entre elles.

Le but de cet exercice est de compléter la définition de la classe immuable `IntPacking` ci-dessous, qui permet d'empaqueter et de dépaqueter des petites valeurs positives dans des entiers de type **`int`**.

```
public final class IntPacking {
  private final int[] sizes;

  public IntPacking(int... sizes) { à faire }
  public int pack(int... values) { à faire }
  public int unpack(int packedValue, int index) { à faire }
}
```

L'extrait de test JUnit suivant, qui devrait s'exécuter avec succès, illustre comment cette classe pourrait être utilisée pour empaqueter et dépaqueter le nombre d'arêtes sortantes et l'index de la première d'entre elles, comme décrit ci-dessus :

```
IntPacking packing = new IntPacking(4, 28);
int packed = packing.pack(12, 3456);
assertEquals(12, packing.unpack(packed, 0));
assertEquals(3456, packing.unpack(packed, 1));
```

Partie 1 [10 points] Écrivez le corps du constructeur de la classe, qui reçoit les tailles des valeurs à empaqueter sous la forme d'un nombre variable de valeurs de type **`int`** et lève une `IllegalArgumentException` à moins que toutes les conditions suivantes ne soient vraies :

1. au moins deux tailles ont été passées au constructeur,
2. toutes ces tailles sont strictement supérieures à 0 et strictement inférieures à 32,
3. la somme de toutes les tailles est inférieure ou égale à 32.

La *dernière* taille donnée est celle de la valeur stockée dans les bits les moins significatifs, la précédente celle de la valeur stockée juste au-dessus, et ainsi de suite. Par conséquent, si la somme de toutes les tailles données au constructeur est inférieure à 32, certains des bits les plus significatifs ne contiendront aucune valeur.

N'oubliez pas que la classe est immuable !

Réponse :

Réponse (suite) :

Partie 2 [15 points] Écrivez le corps de la méthode `pack`, qui prend les valeurs à empaqueter dans le même ordre que les tailles données au constructeur, et retourne la valeur empaquetée.

Si le nombre de valeurs données n'est pas égal au nombre de tailles données au constructeur, ou si une des valeurs est négative ou ne peut être représentée avec le nombre de bits qui lui est attribué, `pack` lève une `IllegalArgumentException`.

Réponse :

Partie 3 [10 points] Écrivez le corps de la méthode `unpack`, qui prend en arguments l'entier contenant les valeurs empaquetées et l'index de la valeur à extraire, et retourne cette valeur.

Si l'index est invalide, `unpack` lève une `IndexOutOfBoundsException`.

Réponse :

4 Vue de matrice [27 points]

Un tableau unidimensionnel d'entiers peut être vu comme une matrice $m \times n$ en utilisant la formule suivante pour faire correspondre les index de ligne et de colonne (r, c) d'un élément de la matrice à l'index i d'un élément du tableau :

$$i = r \times r_s + c \times c_s$$

où r_s et c_s sont les **enjambées** (*strides* en anglais) de ligne et de colonne, qui sont des entiers constants et non négatifs (≥ 0).

Par exemple, le tableau unidimensionnel $[1,2,3,4,5,6]$ peut être vu comme la matrice 2×3 suivante en posant $r_s = 3, c_s = 1$:

$$M_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

On peut s'en convaincre en vérifiant, par exemple, qu'avec ces enjambées, l'élément de la matrice à l'index (1, 2), dont la valeur est 6, correspond à l'élément du tableau à l'index 5, car :

$$i = 1 \times 3 + 2 \times 1 = 5$$

Le même tableau unidimensionnel peut également être vu comme la matrice 3×2 suivante en posant $r_s = 1, c_s = 3$:

$$M_2 = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Le but de cet exercice est de compléter la définition de la classe `MatrixView` ci-dessous, qui utilise cette technique pour voir un tableau (d'entiers) comme une matrice.

```
public final class MatrixView {
  private final int[] array;
  private final int rStride, cStride, rows, columns;

  public static MatrixView ofRow(int[] row, int rows) { à faire }
  public static MatrixView ofColumn(int[] column, int columns) { à faire }

  public MatrixView(int[] array,
                    int rStride, int cStride,
                    int rows, int columns) { code omis }

  public int get(int r, int c) { à faire }
  public MatrixView transposed() { à faire }
}
```

Le code du constructeur a été omis car il ne fait que valider les arguments avant de les stocker dans les attributs correspondants. Les arguments sont validés en vérifiant que le nombre de lignes et de colonnes est strictement positif, que les enjambées sont positives ou nulles, et que tous les index valides de la matrice correspondent à des index valides du tableau. Remarquez que, puisque cette classe est une vue, le tableau passé au constructeur n'est pas copié.

En utilisant cette classe, l'équivalent de la matrice M_1 ci-dessus peut être défini ainsi :

```
int[] array = new int[]{1, 2, 3, 4, 5, 6};
MatrixView m1 = new MatrixView(array, 3, 1, 2, 3);
```

Partie 1 [4 points] Écrivez le corps de la méthode statique `ofRow`, qui renvoie une vue sur une matrice avec le nombre de lignes donné, qui sont toutes égales à la ligne donnée. Par exemple, le code suivant :

```
MatrixView.ofRow(new int[]{7, 8, 9}, 2);
```

définit l'équivalent de la matrice suivante :

$$\begin{pmatrix} 7 & 8 & 9 \\ 7 & 8 & 9 \end{pmatrix}$$

Votre méthode n'est pas autorisée à créer un nouveau tableau, et doit uniquement utiliser celui qui lui a été donné en argument.

Réponse :

Partie 2 [4 points] Écrivez le corps de la méthode statique `ofColumn`, qui renvoie une vue sur une matrice avec le nombre donné de colonnes, qui sont toutes égales à la colonne donnée. Par exemple, le code suivant :

```
MatrixView.ofColumn(new int[]{7, 8}, 4);
```

définit l'équivalent de la matrice suivante :

$$\begin{pmatrix} 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 \end{pmatrix}$$

Votre méthode n'est pas autorisée à créer un nouveau tableau, et doit uniquement utiliser celui qui lui a été donné en argument.

Réponse :

Partie 3 [4 points] Écrivez le corps de la méthode `get`, qui retourne l'élément de la matrice à la ligne `r` et à la colonne `c` données, ou lance une `IndexOutOfBoundsException` si l'un des index est invalide.

Réponse :

Partie 4 [5 points] Écrivez le corps de la méthode `transposed`, qui renvoie la vue transposée de la même matrice. Par exemple, lorsqu'elle est appelée sur `m1` (l'équivalent de la matrice M_1), elle doit retourner l'équivalent de la matrice M_2.

Votre méthode n'est pas autorisée à créer un nouveau tableau.

Réponse :

Partie 5 [10 points] Écrivez une redéfinition de la méthode `toString` pour `MatrixView` qui retourne une chaîne de caractères dans laquelle :

1. les éléments d'une ligne sont séparés par une virgule et un espace, et entourés de crochets,
2. les lignes sont séparées par une virgule, un séparateur de ligne (`\n`) et un espace, et également entourées de crochets.

Par exemple, appliquée à `m1`, `toString` devrait produire la chaîne suivante :

```
[[1, 2, 3],
 [4, 5, 6]]
```

Réponse :

Formulaire

Ce formulaire présente toutes les parties de la bibliothèque Java nécessaires à cet examen. De nombreuses méthodes ont été omises et les types parfois simplifiés.

Classe `Arrays`

La classe `java.util.Arrays`, non instanciable, contient des méthodes statiques de manipulation de tableaux.

```
public class Arrays {
  // Retourne une copie du tableau array de longeur newLength.
  // Les éventuels nouveaux éléments valent null.
  static <E> E[] copyOf(E[] array, int newLength);

  // Retourne vrai ssi les tableaux array1 et array2 contiennent les mêmes éléments.
  static boolean equals(Object[] array1, Object[] array2);

  // Retourne une valeur de hachage basée sur le contenu du tableau array.
  static int hashCode(Object[] array);
}
```

Interface `Iterator`

L'interface `java.lang.Iterator` représente un itérateur, c.-à-d. un objet permettant de parcourir les éléments d'une collection.

```
public interface Iterator<E> {
  // Retourne vrai ssi l'itérateur a encore un élément à fournir.
  boolean hasNext();

  // Retourne le prochain élément, ou lève NoSuchElementException s'il n'y en a plus.
  E next();

  // Supprime l'élément retourné par le dernier appel à next.
  // Lève IllegalStateException si next n'a pas encore été appelée, ou si remove
  // a déjà été appelée après le dernier appel à next.
  void remove();
}
```

Interface `Iterable`

L'interface `java.lang.Iterable` représente un objet itérable, c-à-d dont le contenu peut être parcouru au moyen d'un itérateur, ou de la boucle *for-each*.

```
interface Iterable<E> {
  // Retourne un itérateur sur les éléments de l'objet.
  Iterator<E> iterator();
}
```

(suite à la page suivante)

Classe `StringJoiner`

La classe `java.lang.StringJoiner` représente un «joigneur» de chaîne, qui construit des chaînes de caractères constituées d'une séquence de chaînes séparées par un délimiteur et entourées d'un préfixe et d'un suffixe.

```
public class StringJoiner {
  // Construit un « joigneur » de chaînes avec le préfixe p, le suffixe s et
  // le séparateur d.
  StringJoiner(String d, String p, String s);

  // Ajoute la chaîne s à la séquence.
  void add(String s);

  // Retourne la chaîne constituée de la concaténation du prefixe, des chaînes
  // ajoutées jusqu'à présent et séparées par le séparateur, et du suffixe.
  String toString();
}
```