Pratique de la programmation orientée-objet
Examen final

23 mai 2014

Indications :
— l'examen dure de 11h15 à 15h00,
— 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.

Seul les documents suivants sont autorisés, en version papier uniquement :
— les transparents du 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 Fonctions [11 points]

Une fonction réelle d'une variable réelle fait correspondre à tout élément de $\mathbb{R}$ un autre élément de $\mathbb{R}$. Si f est une telle fonction, ce que l'on note $f : \mathbb{R} \to \mathbb{R}$, alors $f(x)$ est l'image (unique) de $x \in \mathbb{R}$ par la fonction f. En Java, une telle fonction peut être représentée par l'interface suivante :

```
public interface FunctionRR {
  public double valueAt(double x);
}
```

où la méthode `valueAt` retourne l'image de son argument par la fonction représentée par l'objet auquel on l'applique. Par exemple, la classe ci-dessous représente la fonction cosinus :

```
public final class Cos implements FunctionRR {
  @Override
  public double valueAt(double x) { return Math.cos(x); }
}
```

Partie 1 [3 points] Ecrivez une classe instanciable et immuable nommée `Linear` modélisant une fonction affine de la forme $f(x) = ax + b$ où a et b sont des constantes réelles passées en paramètres au constructeur de la classe.

Réponse :

Partie 2 [3 points] La dérivée d'une fonction $f : \mathbb{R} \to \mathbb{R}$, notée f', est également une fonction réelle d'une variable réelle. Sa valeur en un point x peut être approximée de différentes manières, par exemple au moyen de la formule suivante :

$$f'(x) = \frac{f(x+\delta) - f(x-\delta)}{2\delta}$$

où δ est une valeur proche de 0.

Ecrivez une classe instanciable et immuable nommée `Derivative` modélisant la dérivée d'une fonction donnée en l'approximant au moyen de cette formule. En plus de la fonction à dériver, cette classe doit être paramétrée par la valeur de δ.

Réponse :

Partie 3 [3 points] La composition de deux fonctions $f : \mathbb{R} \to \mathbb{R}$ et $g : \mathbb{R} \to \mathbb{R}$, notée $f \circ g$, est également une fonction réelle d'une variable réelle. Sa valeur en un point x est donnée par la formule suivante :

$$(f \circ g)(x) = f(g(x))$$

Ecrivez une classe instanciable et immuable nommée `Composition` et modélisant la composition de deux fonctions qu'on lui fournit en paramètres.

Réponse :

Partie 4 [2 points] Nommez le patron de conception utilisé par chacune des deux classes suivantes :

1. `Derivative` : _____
2. `Composition` : _____

2 Fonctions interpolées [19 points]

Lorsqu'une fonction $f : \mathbb{R} \to \mathbb{R}$ n'est connue qu'en un certain nombre de points $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))$, il est possible de l'*interpoler* entre ces points afin d'estimer sa valeur en tout point $x \in \mathbb{R}$. Différentes techniques d'interpolation existent, la plus simple étant l'interpolation linéaire, qui consiste à utiliser des segments de droites liant les points connus.

La figure 1 montre un exemple de fonction interpolée linéairement à partir de trois points connus. Notez que la fonction interpolée est constante pour tout $x \leq x_1$, et vaut alors $f(x_1)$, et également pour tout $x \geq x_3$, et vaut alors $f(x_3)$.

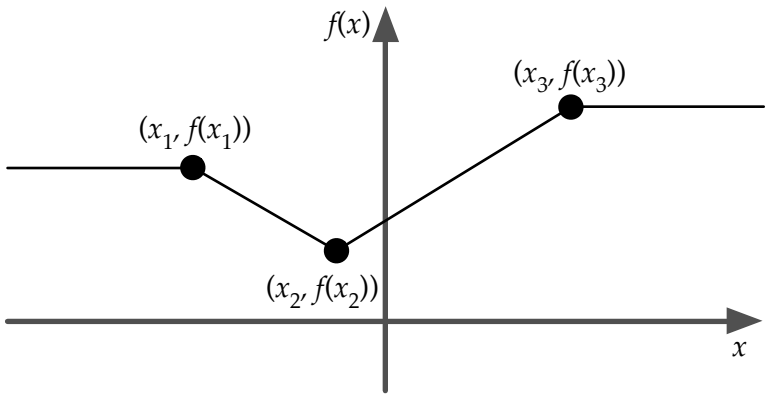

FIGURE 1 – Fonction interpolée linéairement

Partie 1 [10 points] Ecrivez une classe `InterpolatedFunction`, instanciable et immuable, implémentant l'interface `FunctionRR` de l'exercice 1 et modélisant une fonction interpolée linéairement.

Le constructeur de cette classe est paramétré par deux tableaux de réels (de type **`double`**), le premier contenant les valeurs x des points connus, le second les valeurs $f(x)$ correspondantes. Il lève l'exception `IllegalArgumentException` si :

- la longueur des deux tableaux est différente, ou
- les tableaux ont une longueur nulle, ou
- le premier tableau, contenant les valeurs x, n'est pas trié en ordre croissant, sans doublons.

La méthode `valueAt` de cette classe retourne quant à elle l'image de la valeur qu'on lui passe en argument. Cette image est calculée par interpolation linéaire à partir des points passés au constructeur, selon la technique décrite plus haut.

Réponse :

(suite à la page suivante)

Réponse :

Partie 2 [8 points] La classe `InterpolatedFunction` étant immuable, il est judicieux de fournir un bâtisseur permettant d'en construire petit-à-petit des instances. Ecrivez pour ce faire une classe instanciable `Builder`, imbriquée statiquement dans la classe `InterpolatedFunction`, et dotée des deux méthodes suivantes :

1. **`void`** `addPair(`**`double`** `x, `**`double`** `y)`, qui ajoute la paire (x, y) où $y = f(x)$ à l'ensemble des points connus de la fonction à interpoler. Lorsque plusieurs paires de même composante x sont ajoutées à un bâtisseur, seule la dernière est prise en compte.
2. `InterpolatedFunction build()`, qui retourne une fonction interpolée avec les points ajoutés jusqu'ici.

Réponse :

Partie 3 [1 point] L'interpolation linéaire n'est qu'une des nombreuses techniques d'interpolation existante. Quel patron de conception pourrait-on utiliser pour offrir la possibilité de paramétrer la classe `InterpolatedFunction` avec une technique d'interpolation ?

Réponse : _____

3 Tableau et table associative [13 points]

Un tableau peut être vu comme une table associative associant chacun de ses éléments à leur indice. Par exemple, un tableau contenant les chaînes `"zero"`, `"one"` et `"two"` (dans cet ordre) peut être vu comme une table associative associant la chaîne `"zero"` à l'entier 0, la chaîne `"one"` à l'entier 1, et la chaîne `"two"` à l'entier 2.

Pour cet exercice, les tables associatives sont décrites par l'interface `Map` ci-dessous, identique à celle du cours si ce n'est que les méthodes `put` et `remove` en ont été supprimées. L'interface `Iterable` qu'elle étend est celle de la bibliothèque Java, rappelée en annexe.

```
public interface Map<K, V> extends Iterable<Map.Entry<K, V>> {
  boolean isEmpty();
  int size();
  boolean containsKey(K key);
  V get(K key);
  public Iterator<Map.Entry<K, V>> iterator();

  public interface Entry<K, V> {
    K key();
    V value();
  }
}
```

Partie 1 [1 point] Quel patron peut-on utiliser pour concevoir une classe permettant de voir un tableau comme une table associative ?
Réponse : _____

Partie 2 [12 points] Ecrivez une classe instanciable `ArrayToMap` qui implémente l'interface `Map` ci-dessus et permette de voir un tableau comme une table associative. Il doit être possible de l'utiliser ainsi :

```
String[] a = new String[] { "zero", "one", "two" };
Map<Integer, String> m = new ArrayToMap<>(a);
System.out.println(m.get(1));
```

la dernière ligne affichant `one` à l'écran.

La méthode `remove` de l'itérateur retourné par la méthode `iterator` doit bien entendu lever l'exception `UnsupportedOperationException`.

Réponse :

(suite à la page suivante)

Réponse :

4 Transformation MTF [10 points]

La tranformation *move-to-front*, abrégée MTF, est une transformation de flot souvent utilisée dans le domaine de la compression. En elle-même, elle ne compresse pas les données, mais elle peut les rendre plus faciles à compresser.

La transformation MTF consiste à maintenir une table dans laquelle chaque valeur qu'il est possible de rencontrer apparaît exactement une fois, et à encoder chaque valeur par sa position dans cette table. De plus, chaque fois qu'une valeur donnée est encodée, elle est déplacée en tête de la table, d'où le nom.

Cette technique peut être illustrée par un exemple consistant à encoder une chaîne de caractères composée uniquement à l'aide des 26 lettres minuscules de l'alphabet. La table d'encodage initiale pourrait être celle donnée dans la table 1, où les caractères sont triés par ordre alphabétique. Chaque caractère qu'il est possible de rencontrer apparaît exactement une fois dans la table et sa position détermine son encodage.

Position	0	1	2	3	...	13	...	25
Lettre	a	b	c	d	...	n	...	z

TABLE 1 – Table d'encodage initiale

Pour encoder la chaîne `banane` au moyen de cette table, on commence par sa première lettre, `b`, dont la position dans la table d'encodage (et donc le code) est 1. Une fois cette lettre encodée, on modifie la table pour placer la valeur encodée, `b`, à sa tête, ce qui peut bien entendu changer la position d'autres lettres. On obtient alors la table 2.

Position	0	1	2	3	...	13	...	25
Lettre	**b**	a	c	d	...	n	...	z

TABLE 2 – Table après encodage de la première lettre (`b`)

La deuxième lettre à encoder est `a`, dont le code est 1 dans la table 2. Une fois l'encodage effectué, la dernière lettre encodée est à nouveau placée en tête, et on se retrouve avec la table d'encodage initiale (table 1).

La troisième lettre à encoder est `n`, dont le code est 13 dans la table 1. Après l'avoir encodée, on la place en tête de la table, pour obtenir la table 3.

Position	0	1	2	3	...	13	...	25
Lettre	**n**	a	b	c	...	m	...	z

TABLE 3 – Table après encodage de la troisième lettre (`n`)

En poursuivant ainsi, on détermine que l'encodage de la chaîne `banane`, étant donnée la table initiale présentée ci-dessus, est la séquence 1, 1, 13, 1, 1, 5. Le décodage de cette séquence pour obtenir la chaîne `banane` se fait de manière symétrique, en utilisant bien entendu la même table initiale.

Partie 1 [10 points] Ecrivez un décorateur de flot `InputStream` qui, étant donné un flot d'octets encodé par la technique MTF, produit la version décodée de ce flot. Comme d'habitude, cela implique de redéfinir les méthodes `read` et `close` héritées de `InputStream`.

La table de décodage doit contenir la totalité des valeurs qu'il est possible de rencontrer. Les flots `InputStream` étant des flots d'octets, il s'agit des valeurs comprises

dans l'intervalle $[0; 255]$. La table initiale contient simplement ces valeurs dans l'ordre naturel, c-à-d que l'octet n est à la position n.

Réponse :

5 Systèmes de coordonnées géographiques [6 points]

La figure 2 montre le système de coordonnées OSM au niveau de zoom 2. Le carré, quadrillé pour faciliter votre travail, représente la carte du monde entier à ce niveau de zoom.

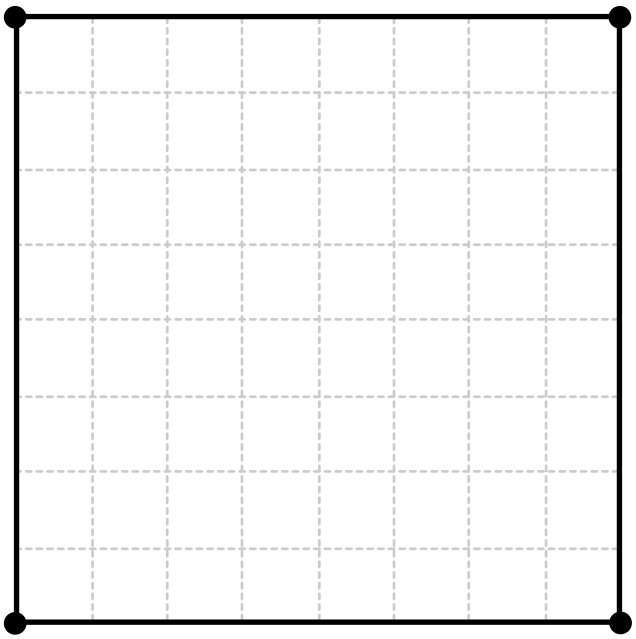

FIGURE 2 – Système de coordonnées OSM au niveau de zoom 2

Partie 1 [2 points] Ecrivez, directement sur la figure 2, les coordonnées (x, y) des quatre sommets du carré, repérés par de petits disques.

Partie 1 [4 points] Dessinez, directement sur la figure 2, les points A à D décrits plus bas, en les convertissant au besoin dans le bon système de coordonnées. Représentez chaque point par un petit disque annoté avec son nom, comme dans la figure 4 de la page 14.

A le point de coordonnées $(320, 0)$ dans le système OSM au niveau de zoom 1.

B le point de coordonnées $(0°, 0°)$ dans le système WGS 84.

C le point de coordonnées $(3072, 3584)$ dans le système OSM au niveau de zoom 4.

D le point de coordonnées $(128, 256)$ dans le système OSM au niveau de zoom 2.

6 Algorithme de Dijkstra [10 points]

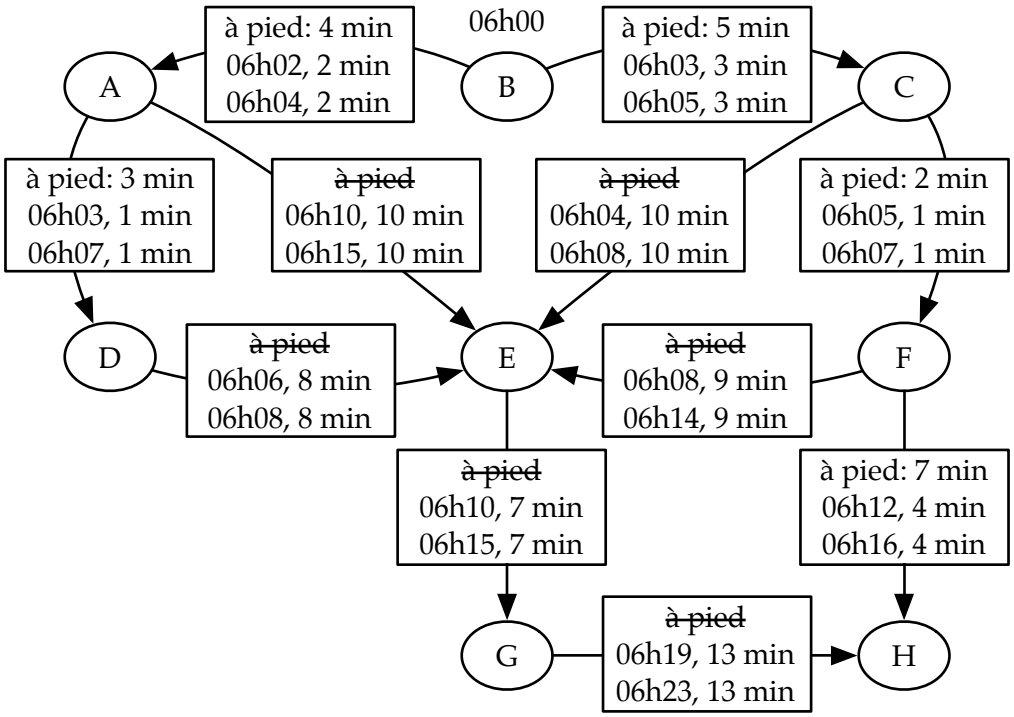

FIGURE 3 – Graphe d'horaires

Partie 1 [8 points] Exécutez manuellement la version modifiée de l'algorithme de Dijkstra utilisée dans le projet afin de trouver les heures de première arrivée pour tous les arrêts du graphe d'horaire de la figure 3 en partant de l'arrêt B à 6h00. Ce faisant, remplissez la table ci-dessous dont chaque ligne correspond à une itération. Les colonnes A à H donnent les heures de première arrivée pour les différents arrêts tandis que la colonne *Visité* donne le nom de l'arrêt visité durant cette itération.

Remplissez la première ligne dans son intégralité, mais ne remplissez ensuite que les cases dont le contenu est différent de celui de l'itération précédente.

Nº	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**Visité**
1									
2									
3									
4									
5									
6									
7									
8									

Partie 2 [2 points] Dessinez l'arbre des trajets les plus rapides pour le résultat de la recherche effectuée ci-dessus, en utilisant le même format que dans la figure 5 de l'exercice 7 (en page 14).

Réponse :

7 Dessin de tuiles isochrones [8 points]

La figure 4 montre une tuile d'une carte isochrone et 6 arrêts désignés par les lettres A à F. Elle a été quadrillée dans le but de faciliter votre travail. La figure 5 montre quant à elle un arbre de trajets les plus rapides.

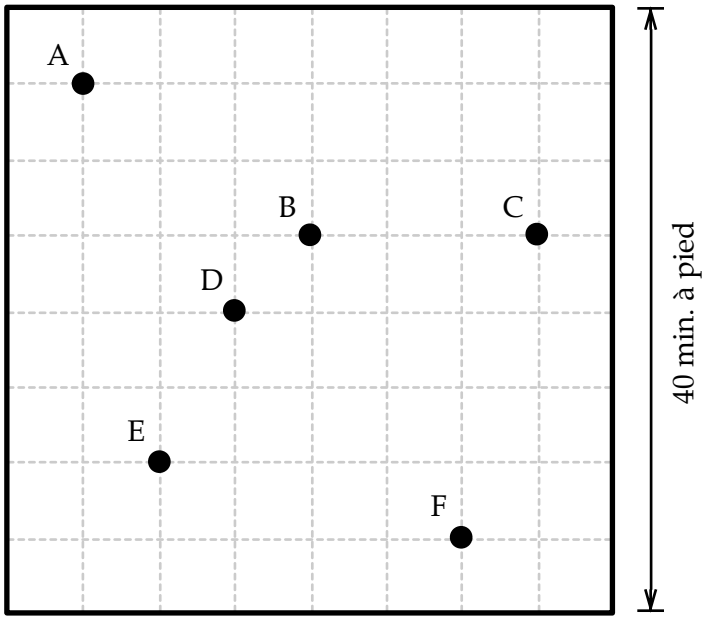

FIGURE 4 – Tuile isochrone

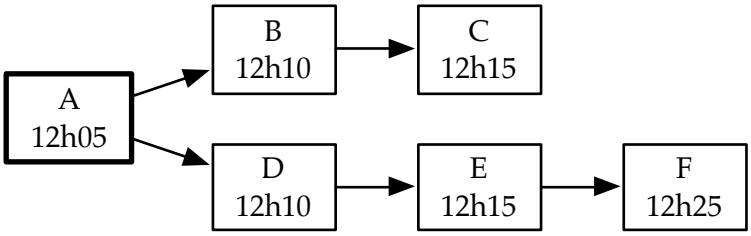

FIGURE 5 – Arbre des trajets les plus rapides

Partie 1 [6 points] Dessinez, directement sur la figure 4, les cercles englobant tous les points atteignables en moins de 15 minutes en partant de l'arrêt A à 12h05, sachant qu'il est possible de marcher d'un côté de la tuile au côté opposé en 40 minutes exactement.

Partie 2 [2 points] Afin d'optimiser le dessin des tuiles isochrones, on pourrait imaginer ignorer, lors du dessin d'une tuile, tous les arrêts qui ne se trouvent pas sur cette tuile. Cette optimisation est-elle correcte ? Justifiez votre réponse.
Réponse :

Formulaire

Ce formulaire présente toutes les parties de la bibliothèque standard Java dont vous avez besoin pour cet examen. Notez que de nombreuses méthodes inutiles ont été omises afin de ne pas encombrer la présentation.

Classe `Arrays`

La classe `java.util.Arrays` contient des méthodes statiques utiles pour manipuler les tableaux.

```
class Arrays {
  // Retourne une copie du tableau a en le tronquant ou en ajoutant des 0 à la fin
  // au besoin pour qu'elle ait une longueur l.
  static double[] copyOf(double[] a, int l);

  // Trie le tableau a en ordre croissant.
  static void sort(double[] a);

  // Recherche par dichotomie l'élément e dans le tableau (trié !) a et retourne sa
  // position s'il s'y trouve ou -(i + 1) sinon, où i est sa position d'insertion.
  // Cette position est celle à laquelle e devrait être inséré dans a pour le garder trié.
  static int binarySearch(double[] a, double e);
}
```

Interface `List`

L'interface `java.util.List` représente les listes. Elle est implémentée, entre autres, par les classes `LinkedList` (listes chaînées) et `ArrayList` (tableaux-listes).

```
interface List<E> extends Iterable<E> {
  // Ajoute l'élément e à la fin de la liste et retourne vrai.
  boolean add(E e);

  // Insère l'élément e à la position i de la liste.
  void add(int i, E e);

  // Retourne l'élément d'indice i de la liste.
  E get(int i);

  // Supprime et retourne l'élément d'indice i de la liste.
  E remove(int i);

  // Retourne un itérateur sur les éléments de la liste.
  Iterator<E> iterator();
}
```

Interface `Map`

L'interface `java.util.Map` représente les tables associatives. Elle est implémentée, entre autres, par la classe `HashMap`.

```
interface Map<K, V> {
  // Associe la valeur v à la clef k, en remplaçant l'ancienne valeur associée s'il y en
  // a une. Retourne l'ancienne valeur associée, ou null s'il n'y en avait pas.
  V put(K k, V v);

  // Retourne la valeur associée à k, ou null s'il n'y en a aucune.
  V get(K k);

  // Retourne vrai si et seulement si la table contient la clef k.
  boolean containsKey(K k);

  // Retourne l'ensemble des clefs de la table.
  Set<K> keySet();
}
```

Interface `Iterator`

```
interface Iterator<E> {
  // Retourne vrai ssi cet itérateur peut encore livrer des éléments.
  boolean hasNext();

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

  // Supprime la valeur retournée par le dernier appel à next, ou lève
  // l'exception IllegalStateException si next n'a pas encore été
  // appelée, ou si remove est appelée deux fois de suite.
  void remove();
}
```

Interface `Iterable`

L'interface `java.lang.Iterable` représente les objets itérables, c-à-d ceux 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();
}
```

Class `InputStream`

La classe `java.io.InputStream` représente les flots (d'octets) d'entrée.

```
abstract class InputStream {
  // Lit et retourne le prochain octet du flot. Si ce dernier est épuisé, retourne -1.
  public abstract int read() throws IOException;

  // Ferme le flot.
  public void close() throws IOException;
}
```