Pratique de la programmation orientée-objet
Examen intermédiaire

15 avril 2015

Indications :
— l'examen dure de 13h15 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.

Seuls 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 Polynômes [20 points]

Un polynôme à une indéterminée et à coefficients entiers est une expression de la forme :

$$a_0 + a_1\,x + a_2\,x^2 + \cdots + a_n\,x^n$$

où x est l'indéterminée, $a_i \in \mathbb{Z}$ est le coefficient de degré i et $n \in \mathbb{N}$.

Un tel polynôme peut être représenté par une table associative associant chaque coefficient non nul (la valeur) à son degré (la clef). Ainsi, le polynôme $3 - 2x + x^4$ peut être représenté par la table associant 3 à 0 (car $a_0 = 3$), -2 à 1 (car $a_1 = -2$) et 1 à 4 (car $a_4 = 1$).

La classe *immuable* `Polynomial` ci-dessous, à compléter, représente un polynôme de cette manière, l'attribut `coefs` contenant la table des coefficients non nuls.

```
public final class Polynomial {
  private final Map<Integer, Integer> coefs;

  public Polynomial(Map<Integer, Integer> coefs) { à faire }
  public Polynomial add(Polynomial that) { à faire }
  public double valueAt(double x) { à faire }
}
```

Partie 1 [4 points] Ecrivez le corps du constructeur de la classe ci-dessus. La table qui lui est passée peut contenir des coefficients nuls, mais ceux-ci ne doivent pas être stockés dans l'attribut `coefs`.

Ce constructeur lève l'exception `IllegalArgumentException` si l'un des degrés de la table passée en argument est négatif.

Réponse :

Partie 2 [3 points] Ecrivez le corps de la méthode `add` qui retourne la somme du polynôme auquel on l'applique et de celui passé en argument.

Par exemple, appliquée au polynôme $3 - x$ en lui passant le polynôme $-x + x^4$, elle retourne le polynôme $3 - 2x + x^4$.

Réponse :

Partie 3 [3 points] Ecrivez le corps de la méthode `valueAt` qui retourne la valeur du polynôme étant donnée la valeur de son indéterminée.

Par exemple, appliquée au polynôme $3 - 2x + x^4$ en lui passant la valeur 2, elle retourne 15 car $3 - 2 \cdot 2 + 2^4 = 15$.

Réponse :

Partie 4 [3 points] Redéfinissez les méthodes `equals` et `hashCode` pour la classe `Polynomial`, afin qu'elles soient compatibles et que l'égalité soit structurelle, c-à-d que deux polynômes mathématiquement identiques soient considérés comme égaux par `equals`, même s'il s'agit de deux objets Java différents.

Souvenez-vous que l'égalité sur les collections Java est structurelle. Par exemple, deux tables associatives sont considérées comme égales par `equals` si elles contiennent le même ensemble de paires clef/valeur. Leurs méthode `equals` et `hashCode` sont bien entendu compatibles entre elles.

Réponse :

Partie 5 [1 point] A votre avis, serait-il judicieux que la classe `Polynomial` implémente l'interface `Comparable` ? Justifiez votre réponse.

Réponse :

Partie 6 [6 points] Redéfinissez la méthode `toString` pour la classe `Polynomial` afin qu'elle retourne une représentation textuelle du polynôme utilisant la lettre `x` pour l'indéterminée et le caractère `^` pour l'exponentiation. Les exemples ci-dessous illustrent le fonctionnement de cette méthode en donnant la chaîne qu'elle retourne pour différents polynômes :

1. la chaîne `2-x+x^3` pour le polynôme $2 - x + x^3$,
2. la chaîne `-3+2x^2` pour le polynôme $-3 + 2x^2$,
3. la chaîne `x^3` pour le polynôme x^3,
4. la chaîne `0` pour le polynôme 0.

Comme ces exemples l'illustrent, la chaîne retournée est minimale — elle ne contient aucun coefficient, exposant ou signe + superflu — et les termes triés par degré croissant.

Réponse :

2 Entropie d'un flot [13 points]

En théorie de l'information, l'entropie d'une source de valeurs permet de quantifier la quantité d'information qu'elle transmet.

Un flot d'octets d'entrée Java — de type `InputStream` — est une source produisant des octets, c-à-d des valeurs comprises entre 0 et 255. L'entropie H d'une telle source est donnée par :

$$H = -\sum_{i=0}^{255} p(i)\log_2(p(i))$$

où $p(i)$ est la probabilité d'apparition de l'octet i, qui est simplement le rapport entre le nombre d'occurrences de la valeur dans la séquence fournie par le flot et la longueur de cette dernière. Par convention, si un octet n'apparaît jamais dans le flot, c-à-d si $p(i) = 0$, alors le terme $p(i)\log_2(p(i))$ est remplacé par 0 dans la somme ci-dessus.

Par exemple, un flot émettant toujours le même octet c (quelconque) a une entropie nulle, car $p(c) = 1$ et donc $\log_2(p(c)) = 0$.

Un exemple plus intéressant est celui d'un flot émettant la séquence de sept octets $(4,5,6,5,6,5,6)$. On a alors $p(4) = 1/7$, $p(5) = p(6) = 3/7$ et $p(i) = 0$ pour les autres octets, c-à-d $i \in \{0,\ldots,3\} \cup \{7,\ldots,255\}$. L'entropie de ce flot vaut :

$$\begin{aligned} H &= -\big(p(4)\log_2(p(4)) + p(5)\log_2(p(5)) + p(6)\log_2(p(6))\big) \\ &= -\big(1/7\,\log_2(1/7) + 3/7\,\log_2(3/7) + 3/7\,\log_2(3/7)\big) \\ &\approx 1.449 \end{aligned}$$

Le but de cet exercice est de définir un nouveau type de flot d'entrée permettant de connaître à tout moment l'entropie d'un flot d'entrée sous-jacent.

Partie 1 [13 points] Ecrivez une classe nommée `EntropyInputStream` héritant de `InputStream` et ayant les caractéristiques suivantes :

- son constructeur prend un seul argument, un autre flot d'entrée que nous nommerons le flot sous-jacent,
- sa méthode `read` produit les mêmes octets que la méthode `read` du flot sous-jacent, dans le même ordre,
- elle possède une méthode nommée `entropy` retournant l'entropie de la séquence d'octets fournie jusqu'à présent par le flot sous-jacent.

Cette classe peut s'utiliser pour calculer l'entropie de l'exemple donné plus haut :

```
byte[] bs = new byte[]{ 4, 5, 6, 5, 6, 5, 6 };
try (EntropyInputStream s =
       new EntropyInputStream(new ByteArrayInputStream(bs))) {
  System.out.println(s.entropy()); // affiche 0
  for (int i = 0; i < 7; ++i)
    s.read();
  System.out.println(s.entropy()); // affiche 1.449...
}
```

Pour écrire votre solution, souvenez-vous que le logarithme dans une base b quelconque peut se calculer au moyen du logarithme naturel ln, grâce à l'égalité suivante :

$$\log_b(x) = \frac{\ln(x)}{\ln(b)}$$

Réponse :

3 Comparateurs partiels [7 points]

L'interface `Comparator` de la bibliothèque Java fait l'hypothèse que les objets qu'elle compare sont totalement ordonnés. C'est-à-dire que, étant donné deux objets à comparer, l'un d'eux est forcément plus petit ou égal à l'autre. (Si chacun des deux objets est plus petit ou égal à l'autre, alors ils sont égaux.)

Certains types d'objets ne peuvent pas être ordonnés totalement de la sorte, mais peuvent l'être partiellement. C'est-à-dire que deux objets peuvent être soit comparables, auquel cas l'un des deux est plus petit ou égal à l'autre, soit incomparables, auquel cas aucun des deux n'est plus petit ou égal à l'autre.

Par exemple, les ensembles mathématiques ne sont pas ordonnés totalement mais peuvent être ordonnés partiellement par la relation d'inclusion $\subseteq$. Ainsi, l'ensemble $\{1,2\}$ est plus petit ou égal à l'ensemble $\{1,2,3\}$ car $\{1,2\} \subseteq \{1,2,3\}$. Par contre, les ensembles $\{1,2\}$ et $\{2,3\}$ sont incomparables, car aucun des deux n'est plus petit ou égal à (c-à-d inclus dans) l'autre.

Le but de cet exercice est de définir une interface fonctionnelle permettant de représenter cette variante partielle des comparateurs.

Partie 1 [2 points] Définissez une interface fonctionnelle représentant un comparateur partiel et nommée `PartialComparator`. Cette interface est bien entendu très similaire à l'interface `Comparator`, mais sa méthode abstraite, nommée `lessOrEqual`, retourne une valeur booléenne qui n'est vraie que si le premier des deux objets qu'on lui passe est plus petit ou égal au second.

Cette interface peut s'utiliser ainsi pour définir le comparateur partiel sur les ensembles d'entiers fondé sur l'inclusion mentionné plus haut :

```
PartialComparator<Set<Integer>> c =
  (s1, s2) -> s2.containsAll(s1);
```

Réponse :

Partie 2 [3 points] Ajoutez à `PartialComparator` une méthode statique nommée `ofTotal`, prenant en argument un comparateur total de type `Comparator` et retournant le comparateur partiel correspondant.

Notez qu'un comparateur total est un cas particulier d'un comparateur partiel, pour lequel toute paire d'objets est comparable.

Réponse :

Partie 3 [2 points] Ajoutez à `PartialComparator` une méthode par défaut nommée `comparable` prenant en arguments deux objets du type de ceux comparés par ce comparateur et retournant vrai si (et seulement si) ils sont comparables, c-à-d que l'un est plus petit ou égal à l'autre.

Réponse :

Formulaire

Ce formulaire présente toutes les parties de la bibliothèque Java nécessaires à cet examen. De nombreuses méthodes inutiles ont été omises pour alléger la présentation.

Interface `Map`

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

```
public interface Map<K, V> {
  // Associe la valeur value à la clef key. Retourne la valeur qui était associée
  // à la clef, ou null s'il n'y en avait pas.
  V put(K key, V value);

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

  // Retourne la valeur associée à key, ou defaultValue s'il n'y en a aucune.
  V getOrDefault(K key, V defaultValue);

  // Retourne l'ensemble des paires clef/valeur.
  Set<Map.Entry<K, V>> entrySet();
}
```

Les classes implémentant cette interface offrent toutes un constructeur de copie (c-à-d un constructeur qui prend une valeur de type `Map<K, V>` en argument et l'utilise pour initialiser la table associative construite).

Interface `Map.Entry`

L'interface `java.util.Map.Entry` représente une paire clef/valeur.

```
public interface Map.Entry<K, V> {
  // Retourne la clef de la paire.
  K getKey();

  // Retourne la valeur de la paire.
  V getValue();
}
```

Interface `Comparator`

L'interface fonctionnelle `java.util.Comparator` représente les comparateurs.

```
public interface Comparator<T> {
  // Compare les objets reçus et retourne un entier négatif si le premier est strictement
  // inférieur au second, zéro s'ils sont égaux et un entier positif sinon.
  int compare(T o1, T o2);
}
```

Classe `StringBuilder`

La classe `java.lang.StringBuilder` est un bâtisseur pour les chaînes de caractères.

```
public class StringBuilder {
  // Retourne la longueur de la chaîne en cours de construction.
  public int length();

  // Ajoute la représentation textuelle de l'objet passé à la chaîne en cours
  // de construction et retourne this.
  public StringBuilder append(Object o);

  // Retourne une nouvelle chaîne avec le contenu ajouté jusqu'à présent.
  public String toString();
}
```

Classe `InputStream`

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

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

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

Classe `Collections`

La classe `java.util.Collections`, non instanciable, contient des méthodes statiques travaillant sur les collections.

```
public class Collections {
  // Retourne une vue non modifiable sur la table associative donnée.
  public static <K, V> Map<K, V> unmodifiableMap(Map<K, V> m);
}
```

Classe `Math`

La classe `Math`, non instanciable, contient des méthodes statiques représentant des fonctions mathématiques courantes.

```
public class Math {
  // Retourne la valeur absolue de x, c-à-d |x|.
  public static int abs(int x);

  // Retourne x élevé à la puissance y, c-à-d x^y.
  public static double pow(double x, double y);

  // Retourne le logarithme naturel de x, c-à-d ln x.
  public static double log(double x);
}
```