Pratique de la programmation orientée-objet
Examen final

29 mai 2015

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.

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 Nombres avec unité [22 points]

Lorsqu'on écrit un programme qui manipule des nombres, il peut être utile de connaître l'unité dans laquelle chacun d'entre-eux est exprimé. Ainsi, plutôt que de représenter une vitesse par un simple nombre de type **`double`**, on peut la représenter par ce nombre accompagné de son unité, p.ex. $\mathrm{m\,s^{-1}}$ (mètres par seconde). Cela permet, entre autres, de détecter certains calculs invalides comme l'addition d'une vitesse en $\mathrm{m\,s^{-1}}$ et d'une distance en m.

Le but de cet exercice est d'écrire une classe représentant un nombre accompagné de son unité. Pour simplifier les choses, les préfixes servant à désigner les multiples des unités (p.ex. le k dans kg ou km) ne sont pas traités. Dès lors, deux unités de nom différent sont toujours considérées comme différentes, même dans le cas où elles sont en réalité liées comme kg et g.

Partie 1 [3 points] Ecrivez la définition de la classe instanciable et immuable `DoubleU` représentant un nombre avec unité, et de ses deux attributs privés, le premier contenant le nombre, le second l'unité.

L'attribut contenant le nombre a le type **`double`**, tandis que celui contenant l'unité est une table associant à chaque nom d'unité (une chaîne) l'exposant correspondant (un entier). Par exemple, l'unité $\mathrm{m\,s^{-1}}$ est représentée par la table associant l'entier 1 à la chaîne `m` et l'entier -1 à la chaîne `s`.

Réponse :

Partie 2 [5 points] Ajoutez à la classe `DoubleU` les trois constructeurs suivants :

- le premier prenant un nombre et son unité, chacun du même type que l'attribut qui lui correspond,
- le second prenant un nombre et un *nom* d'unité (de type `String`), et construisant ce nombre avec l'unité en question élevée à la puissance 1,
- le troisième prenant uniquement un nombre et construisant ce nombre sans unité (un nombre « pur »).

Le premier de ces constructeurs doit tolérer la présence d'exposants nuls dans la table représentant l'unité, mais ne doit pas les stocker dans l'attribut correspondant.

Ces trois constructeurs doivent pouvoir s'utiliser ainsi (dans l'ordre) :

```
DoubleU surface =
  new DoubleU(20, Collections.singletonMap("m", 2)); // 20 m²
DoubleU time = new DoubleU(10, "s");                  // 10 s
DoubleU pi = new DoubleU(Math.PI);                    // π
```

Réponse :

Partie 3 [2 points] Ajoutez à la classe `DoubleU` une méthode nommée `add` retournant la somme du nombre avec unité à laquelle on l'applique et de celui passé en argument.

Cette méthode doit lever l'exception `IllegalArgumentException` lorsque les unités des deux nombres ne sont pas égales, et doit pouvoir s'utiliser ainsi :

```
DoubleU d1 = new DoubleU(10, "m"); // 10 m
DoubleU d2 = new DoubleU(20, "m"); // 20 m
DoubleU sum = d1.add(d2);          // 30 m
```

Réponse :

Partie 4 [4 points] Ajoutez à la classe `DoubleU` une méthode nommée `multiply` retournant le produit du nombre avec unité à laquelle on l'applique et de celui passé en argument. Cette méthode doit pouvoir s'utiliser ainsi :

```
DoubleU sideLen = new DoubleU(5, "m");     // 5 m
DoubleU area = sideLen.multiply(sideLen); // 25 m^2
DoubleU volume = sideLen.multiply(area);  // 125 m^3
```

Réponse :

Partie 5 [4 points] Redéfinissez les méthodes `equals` et `hashCode` pour la classe `DoubleU` afin que les nombres avec unité soient comparés de manière structurelle, c'est-à-dire que deux nombres égaux et de même unité soient considérés comme égaux par `equals`, même s'il s'agit de deux objets Java différents.

Réponse :

Partie 6 [4 points] Redéfinissez la méthode `toString` pour la classe `DoubleU` afin qu'elle retourne une représentation textuelle du nombre avec unité auquel on l'applique.

Dans cette représentation, les composants de l'unité doivent apparaître par ordre alphabétique de leur nom, séparés du nombre et les uns des autres par une espace. De plus, le caractère `^` doit être utilisé pour représenter l'exponentiation, et l'exposant 1 ne doit jamais apparaître.

Les exemples ci-dessous illustrent le fonctionnement de cette méthode en donnant la chaîne qu'elle retourne pour différents nombres avec unité :

1. la chaîne `9.0` pour le nombre 9 « pur » (sans unité),
2. la chaîne `5.0 m s^-2` pour le nombre avec unité 5 $\mathrm{m\,s^{-2}}$,
3. la chaîne `3.0 g l^-1` pour le nombre avec unité 3 $\mathrm{g\,l^{-1}}$.

Notez le `.0` qui suit les nombres et qui provient de la méthode (statique) `toString` de la classe `Double`, qu'il vous est suggéré d'utiliser pour transformer les nombres en chaînes.

Réponse :

2 Ensembles d'entiers [12 points]

L'interface générique `SimpleSet` ci-dessous représente un ensemble très simple :

```
public interface SimpleSet<E> {
    public int size();
    public void add(E e);
    public void remove(E e);
    public boolean contains(E e);
}
```

Lorsqu'un tel ensemble contient des entiers et que ceux-ci font partie d'un intervalle relativement petit, il est possible de le représenter avantageusement au moyen d'un tableau de bits.

Par exemple, un ensemble d'entiers dont tous les éléments potentiels font partie de l'intervalle $[-100; 100]$ peut être représenté par un tableau de 201 bits. Le bit d'index 0 de ce tableau vaut 1 si l'entier -100 fait effectivement partie de l'ensemble, 0 sinon. Le bit d'index 1 correspond quant à lui à l'entier -99 et ainsi de suite jusqu'au bit d'index 200, qui correspond à l'entier 100.

Ce tableau de bits peut lui-même être représenté de manière compacte au moyen d'un tableau d'entiers de type **`long`**, dont chaque élément contient donc 64 bits. Ainsi, le premier entier contient les bits d'index 0 à 63, le second les bits d'index 64 à 127, le troisième les bits d'index 128 à 191 et le quatrième (et dernier) les bits d'index 192 à 200, plus quelques bits inutilisés.

Le but de cet exercice est de compléter la définition de la classe `DenseIntSet`, qui utilise l'idée ci-dessus pour représenter des ensembles d'entiers :

```
public final class DenseIntSet implements SimpleSet<Integer> {
    private final int min, max;   // bornes
    private final long[] bits;    // tableau des bits
    private int size;             // taille de l'ensemble

    constructeur (à faire)

    @Override
    public int size() { return size; }

    méthodes add, remove et contains (à faire)
}
```

Partie 1 [3 points] Ecrivez le constructeur de la classe `DenseIntSet`, qui prend en paramètres les bornes (inclusives) `min` et `max` de l'intervalle des éléments potentiels et construit un ensemble vide.

Le constructeur doit lever l'exception `IllegalArgumentException` si le nombre d'éléments potentiels, calculé en fonction des bornes, est négatif.

Réponse :

Partie 2 [5 points] Ecrivez la méthode `add` qui ajoute l'élément reçu à l'ensemble ou lève l'exception `IllegalArgumentException` si celui-ci ne fait pas partie des éléments potentiels, c-à-d qu'il n'est pas dans l'intervalle passé au constructeur.

N'oubliez pas de mettre à jour la taille de l'ensemble stockée dans le champ `size`!

Réponse :

Partie 3 [2 points] Ecrivez la méthode `remove` qui supprime l'élément reçu de l'ensemble ou lève l'exception `IllegalArgumentException` si celui-ci ne fait pas partie des éléments potentiels.

Réponse :

Partie 4 [2 points] Ecrivez la méthode `contains` qui retourne vrai ssi l'élément reçu appartient à l'ensemble ou lève l'exception `IllegalArgumentException` si celui-ci ne fait pas partie des éléments potentiels.

Réponse :

3 Transformation OSM en géométrie [4 points]

La figure 1 montre un ensemble de nœuds P_1 à P_{20} d'un fichier OSM. Les lignes les reliant représentent les chemins, listés également dans la table 1. Pour faciliter la lecture, des traits différents sont utilisés chaque fois que deux chemins distincts se partagent un nœud. Finalement, la table 2 liste les relations de type `multipolygon` du fichier.

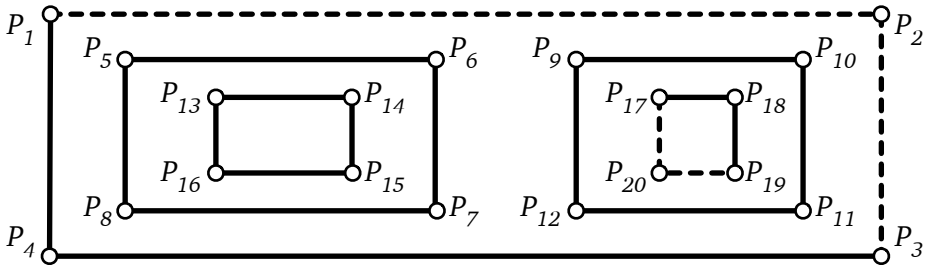


FIGURE 1 – Données OSM

Nom	Nœuds	Attributs
W_1	P_1, P_2, P_3	—
W_2	P_1, P_4, P_3	—
W_3	P_5, P_6, P_7, P_8, P_5	—
W_4	$P_9, P_{10}, P_{11}, P_{12}, P_9$	—
W_5	$P_{13}, P_{14}, P_{15}, P_{16}, P_{13}$	`natural=water`
W_6	P_{17}, P_{18}, P_{19}	`natural=water`
W_7	P_{19}, P_{20}, P_{17}	`natural=water`

TABLE 1 – Chemins

Nom	`outer`	`inner`	Attributs
R_1	W_1, W_2	W_3, W_4	`building=yes`
R_2	W_3	W_5	`natural=grass`

TABLE 2 – Relations de type `multipolygon`

La transformation de ces données OSM en entités géométriques attribuées produit un certain nombre de polygones attribués. Listez les tous en remplissant la table ci-dessous, en utilisant le même format que la ligne d'exemple. Notez que celle-ci utilise des points fictifs et ne correspond à aucun polygone réelement produit par la transformation.

Pour mémoire, les attributs `building` et `natural` font partie de ceux qui, lorsqu'ils sont attachés à un chemin fermé, garantissent que celui-ci décrit une surface.

Enveloppe	Trous (s'il y en a)	Attributs
$(P_{30}, P_{31}, P_{34}, P_{35})$	$(P_{41}, P_{42}, P_{43}, P_{44}), (P_{55}, P_{57}, P_{58}, P_{56})$	`natural=water`

4 Ensembles vérifiants [11 points]

Toute classe implémentant l'interface `SimpleSet` de l'exercice 2 doit satisfaire (au moins) les conditions suivantes pour représenter un ensemble valide :

1. l'entier retourné par la méthode `size` est toujours positif ou nul,
2. après que la méthode `add` ait été appelée avec un élément donné, la méthode `contains` appliquée à ce même élément retourne vrai,
3. lorsque la méthode `add` est appelée avec un élément pour lequel `contains` retourne vrai avant l'appel, la méthode `size` retourne le même entier avant et après l'appel,
4. lorsque la méthode `add` est appelée avec un élément pour lequel `contains` retourne faux avant l'appel, la méthode `size` retourne après l'appel le successeur de l'entier qu'elle retournait avant l'appel,
5. après que la méthode `remove` ait été appelée avec un élément donné, la méthode `contains` appliquée à ce même élément retourne faux,
6. lorsque la méthode `remove` est appelée avec un élément pour lequel `contains` retourne faux avant l'appel, la méthode `size` retourne le même entier avant et après l'appel,
7. lorsque la méthode `remove` est appelée avec un élément pour lequel `contains` retourne vrai avant l'appel, la méthode `size` retourne après l'appel le prédécesseur de l'entier qu'elle retournait avant l'appel.

Le but de cet exercice est d'écrire une classe implémentant l'interface `SimpleSet` et vérifiant qu'un ensemble existant satisfait bien ces conditions.

Partie 1 [10 points] Ecrivez une classe instanciable nommée `CheckingSet` implémentant l'interface `SimpleSet` et vérifiant la totalité des conditions mentionnées plus haut pour un ensemble de type `SimpleSet` passé à son constructeur.

La première de ces conditions doit être vérifiée après chaque appel à la méthode `size`, les trois suivantes après chaque appel à la méthode `add` et les trois dernières après chaque appel à la méthode `remove`. Si l'une d'entre-elles est violée, l'exception `UnsatisfiedCondition`, définie ci-dessous, doit être levée.

```
public final class UnsatisfiedCondition
    extends RuntimeException { }
```

Pour mémoire, le fait que cette exception soit une sous-classe de `RuntimeException` rend facultatif l'ajout d'un attribut **`throws`** aux méthodes qui peuvent la lever.

Partie 2 [1 point] Quel patron est utilisé par la classe `CheckingSet` ?

Réponse : _____

Réponse :

5 Encodage Base32 [10 points]

L'encodage appelé « Base32 » permet d'encoder une séquence d'octets sous la forme d'une séquence de caractères tirés d'un alphabet de 32 lettres. Cet alphabet est composé des 26 lettres majuscules de l'alphabet latin suivies des chiffres de 2 à 7 :

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 2, 3, 4, 5, 6, 7

L'idée de cet encodage est très simple : les octets de la séquence à encoder sont découpés en groupes de 5 bits interprétés comme des entiers compris entre 0 et 31, et chacun d'entre-eux est encodé par la lettre de l'alphabet dont la position lui correspond. Par exemple, la séquence de 5 octets (40 bits) ci-dessous :

`00000000 11111111 01010101 11110000 00001111`

se découpe ainsi en 8 groupes de 5 bits :

`00000 00011 11111 10101 01011 11100 00000 01111`.

Interprété comme un entier, le premier de ces groupes vaut 0 et est donc encodé par la lettre `A` ; le second vaut 3 et est donc encodé par la lettre `D` ; le troisième vaut 31 et est donc encodé par la lettre `7` ; et ainsi de suite. L'encodage Base32 de cette séquence de 5 octets est donc finalement la séquence de 8 lettres `AD7VL4AP`.

Dans le cas où le nombre de bits à encoder n'est pas un multiple de 5, des bits nuls sont ajoutés jusqu'à ce que ce soit le cas. Par exemple, deux bits nuls sont ajoutés à une séquence composée d'un seul octet (8 bits) pour en obtenir un total de 10.

Le but de cet exercice est d'utiliser le patron *Adapter* pour écrire une classe adaptant un écrivain pour en faire un flot de sortie d'octets, en encodant en Base32 les octets du flot.

Partie 1 [10 points] Ecrivez la classe instanciable `Base32OutputStream` héritant de `OutputStream` et dont le constructeur prend en argument l'écrivain de type `Writer` dans lequel les caractères résultants de l'encodage des octets sont écrits.

Pour faciliter votre travail, vous pouvez utiliser l'interface `Base32` ci-dessous, définissant l'alphabet Base32 sous la forme d'une chaîne :

```
public interface Base32 {
    public static final String alphabet =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567";
}
```

Réponse :

6 Peintre [4 points]

Soit un peintre défini par l'expression suivante :

```
line(2, WHITE).when(tagged("highway", "primary"))
  .above(line(4, BLACK).when(tagged("highway", "primary")))
  .above(line(2, BLACK).when(tagged("railway")))
  .layered();
```

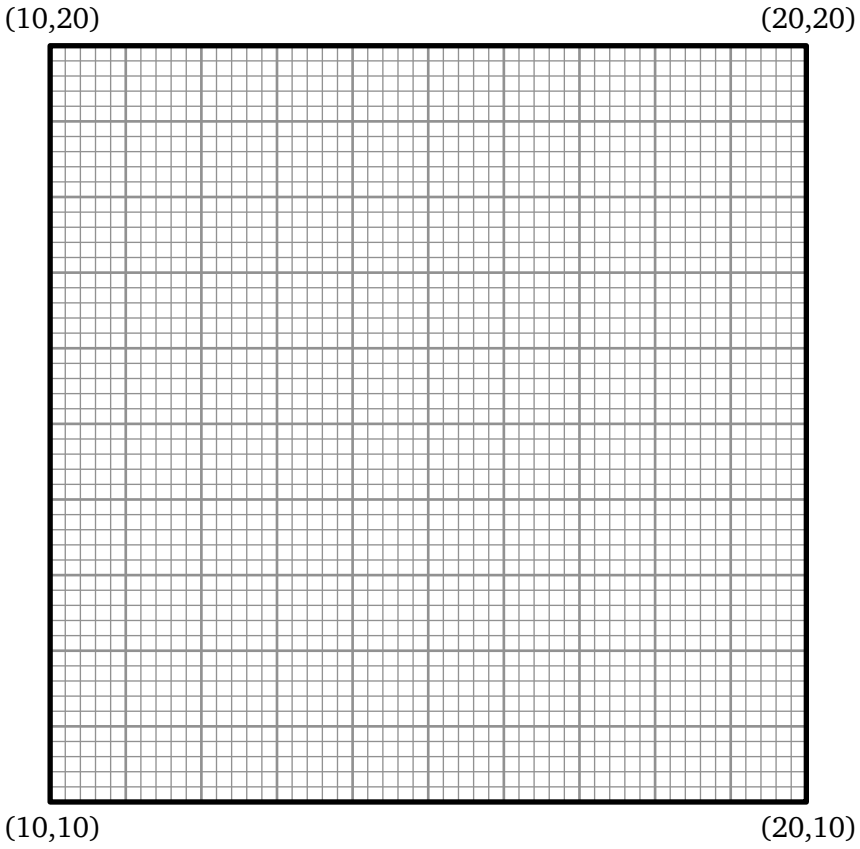
et une carte contenant les polylignes ouvertes attribuées suivantes :

Points	**Attributs**
$(0,13),(30,13)$	`highway=primary`
$(0,17),(30,17)$	`highway=secondary`
$(12,30),(12,0)$	`railway=tram`
$(15,0),(15,30)$	`railway=rail`, `layer=1`

Dessinez sur la toile ci-dessous l'image obtenue en appliquant le peintre à la carte. Pour faciliter votre travail, cette toile a été quadrillée et agrandie de manière à ce que l'espacement entre les lignes de la grille corresponde à un point pica. Pour mémoire, le point pica est l'unité dans laquelle la largeur des lignes est spécifiée.

Vous pouvez supposer que la résolution de la toile est assez grande pour que les pixels individuels soient invisibles à l'œil nu.

Utilisez la couleur de votre stylo ou crayon, quelle qu'elle soit, pour colorier les zones que le peintre dessine en noir (`BLACK`) mais laissez vides les zones qu'il ne dessine pas ou dessine en blanc (`WHITE`).



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();
}
```

Classe `StringBuilder`

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

```
public class StringBuilder {
  // 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 `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);

  // Retourne une table associative immuable associant la valeur v à la clef k.
  public static <K, V> Map<K, V> singletonMap(K k, V v);

  // Retourne une table associative immuable vide.
  public static <K, V> Map<K, V> emptyMap();
}
```

Classe `OutputStream`

La classe `java.io.OutputStream` représente les flots (d'octets) de sortie.

```
abstract public class OutputStream {
  // Ecrit les 8 bits de poids faible de b dans le flot.
  abstract public void write(int b) throws IOException;

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

Classe `Writer`

La classe `java.io.Writer` représente les écrivains, c-à-d les flots de caractères de sortie.

```
abstract public class Writer {
  // Ecrit le caractère c dans le flot et retourne this.
  public Writer append(char c) throws IOException;

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