Pratique de la programmation orientée-objet
Examen final

1 juin 2018

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 Titre de ROM [33 points]

La mémoire morte (ROM) d'une cartouche de Game Boy contient non seulement le code et les données de son jeu, mais également un en-tête décrivant la cartouche, p.ex. le type de son contrôleur de banque mémoire (MBC).

Une information présente dans l'en-tête est le titre du jeu, stocké dans 15 octets à partir de l'adresse 308 et qui doit être composé exclusivement de caractères ASCII dont le code est compris entre 20_{16} (inclu) et 60_{16} (exclu). Ces caractères sont présentés dans la table ci-dessous, dans laquelle le code hexadécimal d'un caractère peut être déterminé en combinant l'en-tête de sa ligne et de sa colonne. Par exemple, le caractère `C` se trouve à l'intersection de la ligne intitulée 4x et de la colonne intitulée x3, ce qui signifie que son code ASCII est 43_{16}.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
2x		!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_

Si le titre fait moins de 15 caractères, alors les emplacements restants de la partie de l'en-tête le contenant sont remplis avec des octets nuls (c-à-d des octets valant 0).

En plus du titre, l'en-tête de la cartouche contient aussi ce que l'on nomme une *somme de contrôle*, stockée à l'adresse 333. Il s'agit simplement d'une valeur calculée à partir d'autres éléments de l'en-tête, et dont le but est de déterminer la validité de l'en-tête. Le programme contenu dans la mémoire de démarrage du Game Boy recalcule la valeur de cette somme de contrôle et empêche le jeu de s'exécuter si la valeur calculée diffère de la valeur stockée.

La somme de contrôle s est calculée par la formule suivante, où r_i est l'octet à l'adresse i, interprété comme un entier positif compris entre 0 et 255 :

$$s = \left[-\sum_{i=308}^{332} (r_i + 1) \right] \bmod 256$$

Par exemple, dans le cas où tous les octets entre les adresses 308 et 332 valent 0, $s = -25 \bmod 256 = 231$.

Le programme ci-dessous, à compléter, lit un fichier ROM existant et en écrit un nouveau identique au premier mais avec un titre différent, et la somme de contrôle correcte correspondante :

```
public final class SetRomTitle {
  public static void main(String[] args) throws IOException {
    setRomTitle(args[0], args[1], args[2]);
  }
  static void setRomTitle(String inFileName,
                          String outFileName,
                          String title) throws IOException {
    à faire
  }
}
```

Par exemple, en supposant que le fichier `tetris.gb` existe et contienne un fichier ROM valide, l'exécution du programme ci-dessus avec les trois arguments :

```
tetris.gb new-tetris.gb CS108EPFL
```

produit un nouveau fichier ROM nommé `new-tetris.gb`, identique au fichier `tetris.gb` à l'exception des octets de l'en-tête contenant le titre, qui valent :

```
43 53 31 30 38 45 50 46 4C 00 00 00 00 00 00
```

et de la somme de contrôle stockée à l'adresse 333, qui est celle correspondant à ce nouveau titre.

Écrivez le corps de la méthode `setRomTitle` afin que le programme ci-dessus ait le comportement décrit. Votre méthode peut faire l'hypothèse que le fichier d'entrée existe, peut être lu sans erreur et est un fichier ROM valide. Si le titre qu'on lui donne fait plus de 15 caractères, elle doit ignorer ceux en trop. S'il fait moins de 15 caractères, alors tous les octets inutilisés du titre doivent valoir 0 dans le fichier ROM de sortie. Finalement, tout caractère invalide contenu dans le titre, c-à-d tout caractère hors de la plage 20_{16} (inclu) à 60_{16} (exclu), doit être remplacé par un caractère d'espacement (code ASCII 20_{16}).

Souvenez-vous qu'en Java les caractères sont des entiers, et que tous les caractères ASCII ont une valeur égale à leur code ASCII. Par exemple, l'expression suivante est vraie en Java : `'C'== 0x43`.

Réponse :

2 Vecteurs de bits [35 points]

La classe `BitVector` du projet représente des vecteurs de bits de longueur finie en stockant leurs bits dans un tableau d'entiers. Le but de cet exercice est d'explorer une représentation alternative des vecteurs de bits basée sur l'interface fonctionnelle ci-dessous, qui représente un vecteur de bits de longueur *infinie* :

```
@FunctionalInterface
public interface BitVector {
  boolean testBit(int i);

  static BitVector constant(boolean c) { return i -> c; }
  static BitVector ofBits(byte[] b, boolean d) { à faire }

  default BitVector not() { à faire }
  default BitVector and(BitVector that) { à faire }
  default BitVector or(BitVector that) { à faire }

  default BitVector shift(int d) { à faire }
  default BitVector join(BitVector that, int j) { à faire }
  default BitVector cycling(int s, int l) { à faire }

  default String toString(int s, int l) { à faire }
}
```

La méthode `testBit` retourne vrai si et seulement si le bit d'index donné (`i`) vaut 1. Notez que cet index peut être quelconque, y compris négatif, étant donné que le vecteur est infini[1].

La méthode `constant` définit un vecteur infini dont tous les bits ont la même valeur.

Partie 1 [2 points] Écrivez le corps de la méthode `not`, qui retourne un vecteur qui est l'inversion bit à bit du récepteur (`this`).

Réponse :

1. Comme le type `int` de Java ne contient qu'un nombre fini de valeurs, les vecteurs de bits représentés par l'interface ci-dessus sont en fait finis, mais pour cet exercice nous ferons l'hypothèse que le type `int` peut représenter tous les entiers mathématiques.

Partie 2 [2 points] Écrivez le corps des méthodes `and` et `or`, qui retournent un vecteur qui est la conjonction (pour `and`) ou la disjonction (pour `or`) bit à bit du récepteur (`this`) et de l'argument (`that`).

Réponse :

Partie 3 [4 points] Écrivez le corps de la méthode `shift`, qui retourne un vecteur égal au récepteur (`this`) mais décalé du nombre de bits donné (`d`). Comme d'habitude, le décalage peut être soit positif, auquel cas le vecteur est décalé vers la gauche, soit négatif, auquel cas il est décalé vers la droite.

Votre méthode doit traiter séparément le cas dans lequel le décalage `d` vaut 0, et retourner alors simplement le récepteur (`this`).

Réponse :

Partie 4 [4 points] Écrivez le corps de la méthode `join`, qui retourne un vecteur infini dont les bits sont ceux du récepteur (`this`) si leur index est strictement inférieur à l'index donné (`j`), et ceux de l'argument (`that`) sinon.

Réponse :

Partie 5 [9 points] Écrivez le corps de la méthode `cycling`, qui prend un intervalle exprimé sous la forme d'un index de départ (`s`) et d'une longueur (`l`) et retourne un vecteur égal au récepteur dans cet intervalle et qui se répète indéfiniment des deux côtés, ou lève `IllegalArgumentException` si la longueur n'est pas strictement positive.

Par exemple, soit le vecteur ci-dessous, où les valeurs des 12 bits présentés sont identifiées par une lettre allant de a à l :

Bit	…	9	8	7	6	5	4	3	2	1	0	−1	−2	…
Valeur	…	a	b	c	d	e	f	g	h	i	j	k	l	…

En appelant `cycling(2, 3)` sur ce vecteur, on obtient le vecteur :

Bit	…	9	8	7	6	5	4	3	2	1	0	−1	−2	…
Valeur	…	g	h	f	g	h	$\mathbf{f}$	$\mathbf{g}$	$\mathbf{h}$	f	g	h	f	…

qui est égal au vecteur original aux index 2, 3 et 4 (en gras), et se répète à l'infini des deux côtés de cet intervalle.

(Notez que la méthode `cycling` est similaire à la méthode `extractWrapped` du projet, mais produit un vecteur infini.)

Réponse :

Partie 6 [5 points] Écrivez le corps de la méthode `toString`, qui retourne une chaîne représentant la portion du vecteur commençant au bit d'index `s` et ayant une longueur `l`, ou lève `IllegalArgumentException` si la longueur est négative.

Comme d'habitude, les bits doivent apparaître ordonnés par index décroissant.

Réponse :

Partie 7 [9 points] Écrivez le corps de la méthode `ofBits`, qui crée un vecteur infini à partir d'un tableau d'octets.

Les bits 0 à 7 du vecteur qu'elle retourne doivent être égaux aux bits 0 à 7 de l'élément 0 du tableau, si cet élément existe. Les bits 8 à 15 doivent être égaux aux bits 0 à 7 de l'élément 1 du tableau, si cet élément existe. Et ainsi de suite. Tous les bits du vecteur pour lesquels aucun élément n'existe dans le tableau doivent avoir la même valeur, donnée par le second argument de `ofBits`, à savoir `d`.

Par exemple, en faisant l'hypothèse que la méthode `toString` ci-dessus ait été mis en œuvre correctement, l'exemple suivant devrait afficher `0100100010110111` :

```
byte[] bits =
  new byte[]{ (byte) 0b1011_0111, (byte) 0b0100_1000 };
BitVector v = BitVector.ofBits(bits, false);
System.out.println(v.toString(0, 16));
```

Réponse :

(suite à la page suivante)

Réponse (suite) :

3 Fondu au blanc [6 points]

Pour faire disparaître une image de l'écran, de nombreux jeux Game Boy utilisent un effet nommé *fondu au blanc* consistant à progressivement rendre l'image blanche. Cet effet peut facilement être obtenu par changement de la palette.

La figure 1 montre un exemple d'une image rendue progressivement blanche. L'image tout à gauche est l'image originale, tandis que celle tout à droite est totalement blanche. La seule différence entre ces quatre images est la palette utilisée pour les colorier.

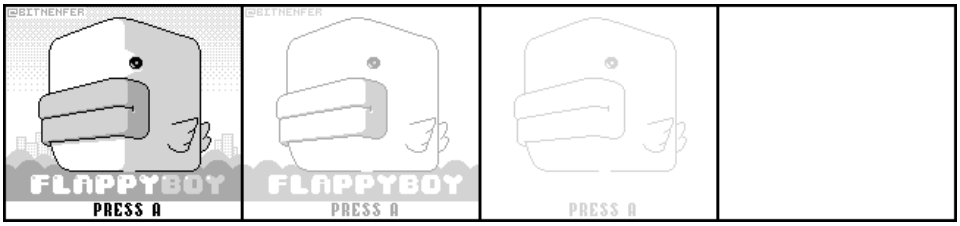

Fig. 1: Images 1 à 4 (de gauche à droite)

La table ci-dessous donne les palettes associées à chacune de ces quatre images, écrites en syntaxe Java. Seule la palette correspondant à l'image originale est donnée, et votre but est de donner les palettes manquantes.

Image	**Palette**
1	`0b11_10_01_00`
2	
3	
4	

4 Dessin d'image [16 points]

Supposons que le contrôleur LCD s'apprête à dessiner une ligne à l'écran, et que ses principaux registres aient les valeurs suivantes, données en syntaxe Java (les registres non importants ont été omis) :

Registre	**Valeur**
SCX	4
SCY	1
LY	2
BGP	0b11_10_01_00

et supposons de plus que le registre LCDC soit configuré de manière à ce que :

— l'écran LCD soit allumé,

— seule l'image de fond soit affichée — la fenêtre et les *sprites* sont désactivés,

— les index des tuiles pour l'image de fond proviennent de la zone comprise entre les adresses 9800_{16} et $9C00_{16}$,

— les images des tuiles pour l'image de fond proviennent de la zone comprise entre les adresses 8000_{16} et 9000_{16}.

Finalement, supposons que le contenu de la mémoire, dans la plage d'adresses allant de 9800_{16} à $980F_{16}$, soit[2] :

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
980x	00	01	02	03	00	01	02	03	00	01	02	03	00	01	02	03

et que celui dans la plage d'adresses allant de 8000_{16} à $803F_{16}$ soit :

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
800x	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
801x	FF	00	FF	00	FF	00	FF	00	FF	00	FF	00	FF	00	FF	00
802x	00	FF	00	FF	00	FF	00	FF	00	FF	00	FF	00	FF	00	FF
803x	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Souvenez-vous que cette seconde plage contient les images des tuiles, qui font chacune 8×8 pixels et occupent 16 octets : le premier donne les bits de poids faible des pixels de la première ligne, le second donne les bits de poids fort de ces pixels, et ainsi de suite.

Remplissez la table suivante en donnant la couleur — sous la forme d'un chiffre allant de 0 à 3 — de chacun des 16 premiers pixels de la ligne dessinée par le contrôleur LCD. Souvenez-vous que LY vaut 2, ce qui signifie que la ligne en cours de dessin est la troisième depuis le haut de l'écran.

Pixel	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Couleur																

2. Dans cette table, chaque ligne présente 16 octets dont la valeur est donnée en hexadécimal. L'adresse d'un octet peut être déterminée en combinant l'en-tête de sa ligne et de sa colonne, comme à l'exercice 1.

5 Conversion d'image [35 points]

La classe `ImageConverter` du projet convertit une image Game Boy — de type `LcdImage` — en une image JavaFX — de type `Image`. Elle fait cette conversion pixel par pixel, ce qui est simple mais relativement lent.

Pour accélérer la conversion, il est possible de la faire ligne par ligne, ce que fait la classe `ImageConverter` ci-dessous. Cela implique d'ajouter une méthode `getLine` à `LcdImage`, qui retourne une ligne de l'image, et d'utiliser la méthode `setPixels` de `PixelWriter` pour écrire en un appel tous les pixels d'une ligne.

Il n'est pas nécessaire de comprendre le code ci-dessous de manière détaillée, seule la méthode `convertLine` importe réellement. Elle prend en argument une ligne à convertir, et retourne le résultat de la conversion sous la forme d'un tableau d'entiers `int` dont chaque élément contient la couleur empaquetée d'un pixel. La conversion proprement dite est effectuée par la méthode `reallyConvertLine`.

```
public final class ImageConverter {
  private static final int[] COLOR_MAP = new int[]
    { 0xFFFFFFFF, 0xFFD3D3D3, 0xFFA9A9A9, 0xFF000000 };
  private static final PixelFormat<IntBuffer> PIX_FMT =
    PixelFormat.getIntArgbInstance();

  public static Image convert(LcdImage l) {
    int w = l.width(), h = l.height();
    WritableImage j = new WritableImage(w, h);
    PixelWriter pixW = j.getPixelWriter();
    for (int y = 0; y < h; ++y) {
      int[] c = convertLine(l.getLine(y));
      pixW.setPixels(0, y, w, 1, PIX_FMT, c, 0, 0);
    }
    return j;
  }

  private static Map<LcdImageLine, int[]> cache =
    new HashMap<>();
  private static int[] convertLine(LcdImageLine l) {
    return reallyConvertLine(l); // à améliorer
  }

  private static int[] reallyConvertLine(LcdImageLine l) {
    int[] b = new int[l.size()];
    for (int i = 0; i < l.size(); ++i) {
      int c = (l.msb().testBit(i) ? 0b10 : 0)
        | (l.lsb().testBit(i) ? 1 : 0);
      b[i] = COLOR_MAP[c];
    }
    return b;
  }
}
```

Une observation importante que l'on peut faire à propos de `convertLine` est qu'on lui demande souvent de convertir une ligne qu'elle a déjà converti par le passé.

Il y a deux raisons à cela : premièrement, les images contiennent souvent des lignes dupliquées ; deuxièmement, les images successives affichées à l'écran ont souvent des lignes en commun.

Par exemple, l'image du logo Nintendo affichée au démarrage contient de nombreuses lignes blanches identiques, et comme elle reste à l'écran durant plusieurs secondes, les lignes du logo sont communes à plusieurs images successives.

Il est possible de tirer parti de cette observation pour accélérer encore la méthode `convertLine`. L'idée est d'introduire ce que l'on nomme un *cache*, une collection contenant les lignes converties par le passé et le résultat de leur conversion.

Initiallement, le cache est vide. Ensuite, lorsqu'une ligne doit être convertie, le cache est consulté pour voir s'il contient une entrée pour cette ligne. Si oui, alors le résultat de la précédente conversion en est extrait et réutilisé. Sinon, la ligne est convertie comme d'habitude, et le cache est mis à jour pour mémoriser le résultat de cette conversion.

Par exemple, admettons que la toute première image affichée à l'écran, celle contenant le logo Nintendo, soit en cours de conversion. Sa première ligne est totalement blanche. Comme le cache est vide lorsqu'elle est convertie, la conversion est faite par la méthode `reallyConvertLine`. Le cache est ensuite mis à jour pour mémoriser le résultat de cette conversion. Lorsque la seconde ligne de l'image, aussi totalement blanche, est convertie, le cache est examiné. Cette fois, une entrée correspondant à la ligne totalement blanche y est trouvée, et le résultat de la conversion précédente est réutilisé, évitant ainsi un appel à `reallyConvertLine`.

Partie 1 [7 points] Montrez comment modifier la méthode `convertLine` plus haut pour qu'elle utilise un cache afin de ne pas appeler `reallyConvertLine` lorsqu'une ligne déjà convertie par le passé l'est à nouveau. Pour cette première version, vous pouvez laisser le cache grandir indéfiniment.

Souvenez-vous que les instances de `LcdImageLine` sont comparées par structure, et notez que le cache est déjà défini dans la classe ci-dessus et stocké dans l'attribut `cache`, il ne reste plus qu'à l'utiliser.

Réponse :

Partie 2 [10 points] Un problème évident d'un cache qui grandit indéfiniment est qu'il peut petit à petit consommer toute la mémoire disponible et provoquer le plantage du programme. Une solution simple à ce problème consiste à attendre que le cache contienne un nombre maximum donné d'éléments avant d'en supprimer un certain pourcentage, choisi aléatoirement.

Montrez comment augmenter votre méthode `convertLine` afin de mettre cette idée en œuvre, en laissant le cache grandir jusqu'à 1000 éléments avant d'en supprimer 100 choisis aléatoirement.

Notez que pour choisir les éléments à supprimer, il n'y a pas vraiment de meilleure solution que de placer les clefs du cache dans une liste, de la mélanger aléatoirement puis de supprimer du cache les 100 premières clefs qu'elle contient.

Réponse :

Partie 3 [18 points] Supprimer des éléments choisis au hasard est mieux que de laisser le cache grandir indéfiniment, mais une solution encore meilleure consiste à supprimer du cache non pas des éléments choisis au hasard, mais plutôt ceux utilisés le moins récemment.

Montrez comment modifier votre méthode `convertLine` de la partie 1 pour mettre cette idée en œuvre, en laissant le cache grandir jusqu'à 1000 éléments avant d'en supprimer les 100 utilisés le moins récemment. Notez que cela implique d'avoir *deux* informations à propos de chaque ligne stockée dans le cache : premièrement, le résultat de sa conversion, comme précédemment ; deuxièmement, une indication sur sa dernière utilisation.

Vous êtes libres de définir une nouvelle classe pour stocker ces informations, et d'ajouter au besoin de nouveaux attributs à la classe `ImageConverter`. Vous noterez qu'un simple compteur incrémenté à chaque appel de `convertLine` fournit une notion abstraite de temps qui suffit aux besoins de cet exercice.

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 `Integer`

La classe `java.lang.Integer` contient entre autres des méthodes statiques travaillant sur les entiers de type `int`.

```
public class Math {
  // Compare x et y et retourne un entier négatif si x < y, zéro si x == y,
  // et un entier positif sinon.
  static int compare(int x, int y);
}
```

Classe `Math`

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

```
public class Math {
  // Retourne le minimum de x et y.
  static int min(int x, int y);

  // Retourne la division entière par défaut de x par y (⌊x/y⌋).
  static int floorDiv(int x, int y);
  // Retourne le reste de la division entière par défaut de x par y (x − y⌊x/y⌋).
  static int floorMod(int x, int y);
}
```

Classe `Collections`

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

```
public class Collections {
  // Mélange aléatoirement les éléments de la liste l.
  static void shuffle(List<?> l);

  // Trie les éléments de la liste l selon leur ordre naturel.
  static <T extends Comparable<T>> void sort(List<T> l);
  // Trie les éléments de la liste l selon l'ordre du comparateur c.
  static <T> void sort(List<T> l, Comparator<T> c);
}
```

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);
}
```

Interface `List`

L'interface `java.util.List` représente les listes. Elle est entre autres implémentée par la classe `ArrayList`, qui possède un constructeur de copie prenant une collection en argument.

L'interface `List` étend l'interface `Iterable`, donc ses éléments peuvent être parcourus au moyen d'un itérateur ou d'une boucle *for-each*.

```
public interface List<E> extends Collection<E> {
  // Retourne l'élément d'index i ou lève IndexOutOfBoundsException
  // s'il est invalide.
  E get(int i);
}
```

Interface `Map`

L'interface `java.util.Map` représente les tables associatives. Elle est implémentée, entre autres, par les classes `HashMap` et `TreeMap`. Ces classes possèdent un constructeur de copie prenant une table associative en argument.

```
public interface Map<K, V> {
  // Retourne le nombre de paires clef/valeur présentes dans la table.
  int size();

  // Retourne vrai ssi la table contient la clef k.
  boolean containsKey(K k);

  // Associe la valeur v à la clef k.
  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);

  // Supprime la valeur associée à la clef k.
  void remove(K k);

  // Retourne une vue sur l'ensemble des clefs.
  Set<K> keySet();
}
```

Classe `InputStream`

La classe `java.io.InputStream` représente un flot (d'octets) d'entrée. Parmi ses sous-classes figure `FileInputStream`, qui lit les octets d'un fichier dont le nom est passé à son constructeur.

```
abstract public class InputStream {
  // Lit et retourne la totalité des octets du flot.
  byte[] readAllBytes() throws IOException;
  // Ferme le flot.
  void close() throws IOException;
}
```

Classe `OutputStream`

La classe `java.io.OutputStream` représente un flot (d'octets) de sortie. Parmi ses sous-classes figure `FileOutputStream`, qui écrit les octets dans un fichier dont le nom est passé à son constructeur.

```
abstract public class OutputStream {
  // Écrit la totalité des octets de b dans le flot.
  void write(byte[] b) throws IOException;
  // Ferme le flot.
  void close() throws IOException;
}
```

Classe `String`

La classe `java.lang.String`, immuable, représente une chaîne de caractères.

```
public class String {
  // Retourne la longueur de la chaîne.
  int length();

  // Retourne le caractère d'index i ou lève IndexOutOfBoudsException
  // s'il est invalide.
  char charAt(int i);
}
```

Classe `StringBuilder`

La classe `java.lang.StringBuilder` représente un bâtisseur de chaîne, qui permet de construire petit à petit une chaîne de caractères.

```
public class StringBuilder {
  // Ajoute le caractère donné à la chaîne en cours de construction et retourne
  // this.
  StringBuilder append(char c);

  // Retourne la chaîne en cours de construction.
  String toString();
}
```