Conditions de distribution et de copie

Exercices (III) : Chaînes de caractères

Fabrice Rossi

8 janvier 2002

1 Analyse de programmes

1.1 Mémoire

Exercice 1.1 :

On considère le programme suivant :

Truc

```
1  public class Truc {
2    public static void a(String s) {
3      s+="DSQKJ";
4    }
5    public static String b(String s) {
6      s+="GHIJK";
7      return s;
8    }
9    public static void c(StringBuffer sb) {
10     sb.append("KNTOP");
11   }
12   public static StringBuffer d(StringBuffer sb) {
13     sb=new StringBuffer("ERTBG");
14     return sb;
15   }
16   public static void main(String[] args) {
17     String s="JNHGV";
18     a(s);
19     System.out.println(s);
20     s="GHIJK";
21     b(s);
22     System.out.println(s);
23     StringBuffer sb=new StringBuffer("ABCDE");
24     c(sb);
25     System.out.println(sb);
26     sb=new StringBuffer("ERTBG");
27     d(sb);
28     System.out.println(sb);
29   }
30 }
```

1. Dessiner l'état complet de la mémoire (en supposant que l'éboueur a fait son travail) juste après l'exécution de la ligne 10 (et avant le retour dans la méthode `main`).
2. Donner l'affichage produit par le programme.

Exercice 1.2 :

Indiquer l'affichage produit par le programme suivant :

Analyse

```
1  public class Analyse {
2    public static void modif1(int x) {
3      x+=2;
4    }
5    public static void modif2(int x) {
6      x=5;
7    }
8    //
9    public static String ajoute1(String s) {
10     s+=" toto";
11     return s;
12   }
13   public static String ajoute2(String s) {
14     s="toto";
15     return "titi";
16   }
17   //
18   public static StringBuffer concat1(StringBuffer sb) {
19     sb.append(" +++");
20     return sb;
21   }
22   public static StringBuffer concat2(StringBuffer sb) {
23     sb= new StringBuffer("+++");
24     return sb;
25   }
26   public static void main(String[] args) {
27     int x=24;
28     modif1(x);
29     System.out.println(x);
30     x=24;
31     modif2(x);
32     System.out.println(x);
33     //
34     String s="titi";
35     System.out.println(ajoute1(s));
36     System.out.println(s);
37     s="titi";
38     System.out.println(ajoute2(s));
39     System.out.println(s);
40     //
41     StringBuffer sb=new StringBuffer("---");
```

```
42      System.out.println(concat1(sb));
43      System.out.println(sb);
44      sb=new StringBuffer("---");
45      System.out.println(concat2(sb));
46      System.out.println(sb);
47    }
48  }
```

1.2 Manipulations

Exercice 1.3 :

On considère la méthode suivante :

```
1  public static String e(String a) {
2    String r="";
3    for(int i=0;i<a.length();i++)
4      r=a.charAt(a.length()-i)+r;
5    return r;
6  }
```

1. Cette méthode :
   (a) renvoie le miroir de la chaîne paramètre (c'est-à-dire la chaîne constituée des mêmes caractères que la chaîne paramètre mais dans le sens inverse)
   (b) renvoie une chaîne identique à la chaîne paramètre
   (c) plante quand la chaîne paramètre contient au moins un caractère
   (d) plante toujours
2. On remplace la ligne 3 par la ligne `for(int i=1 ;i<=a.length() ;i++)`. La nouvelle méthode :
   (a) renvoie le miroir de la chaîne paramètre (c'est-à-dire la chaîne constituée des mêmes caractères que la chaîne paramètre mais dans le sens inverse)
   (b) renvoie une chaîne identique à la chaîne paramètre
   (c) plante quand la chaîne paramètre contient au moins un caractère
   (d) plante toujours
3. On remplace la ligne 3 par la ligne `for(int i=a.length() ;i>0 ;i--)`. La nouvelle méthode :
   (a) renvoie le miroir de la chaîne paramètre (c'est-à-dire la chaîne constituée des mêmes caractères que la chaîne paramètre mais dans le sens inverse)
   (b) renvoie une chaîne identique à la chaîne paramètre
   (c) plante quand la chaîne paramètre contient au moins un caractère
   (d) plante toujours
4. Dans la méthode **d'origine**, on remplace la ligne 4 par :
   `r=a.charAt(a.length()-i-1)+r ;`.
   La nouvelle méthode :

(a) renvoie le miroir de la chaîne paramètre (c'est-à-dire la chaîne constituée des mêmes caractères que la chaîne paramètre mais dans le sens inverse)

(b) renvoie une chaîne identique à la chaîne paramètre

(c) plante quand la chaîne paramètre contient au moins un caractère

(d) plante toujours

Exercice 1.4 :

On considère la méthode suivante :

```
1  public static int d(String a,String b) {
2    int k=Math.min(a.length(),b.length());
3    for(int i=0;i<k;i++) {
4      if(a.charAt(i)<b.charAt(i)) {
5        return -1;
6      } else {
7        if (a.charAt(i)>b.charAt(i)) {
8          return 1;
9        }
10     }
11   }
12   if(a.length()<b.length()) {
13     return -1;
14   } else {
15     if(a.length()>b.length()) {
16       return 1;
17     } else {
18       return 0;
19     }
20   }
21 }
```

1. Cette méthode renvoie :
   (a) -1 si la chaîne `a` est **avant** la chaîne `b` dans le dictionnaire, 1 si `a` est **après** `b` et 0 si les chaînes sont identiques
   (b) -1 si la chaîne `a` est **après** la chaîne `b` dans le dictionnaire, 1 si `a` est **avant** `b` et 0 si les chaînes sont identiques
   (c) -1 si le premier caractère de `a` est **avant** le premier caractère de `b` dans l'alphabet, 1 si le premier caractère de `a` est **après** le premier caractère de `b` et 0 si les deux caractères sont égaux
   (d) -1 si le premier caractère de `a` est **après** le premier caractère de `b` dans l'alphabet, 1 si le premier caractère de `a` est **avant** le premier caractère de `b` et 0 si les deux caractères sont égaux
2. On remplace la ligne 2 de la méthode par `int k=Math.max(a.length(),b.length()) ;`. Suite à cette modification, la méthode fonctionne *en général* exactement comme la méthode d'origine. Il arrive cependant que la méthode plante. Donner un exemple de chaînes paramètres qui provoquent un plantage. Donner une caractérisation précise de tous les cas qui entraînent un plantage.

3. Dans la méthode **d'origine**, on remplace les lignes 12 à 20 par la ligne `return a.length()-b.length() ;` :
   (a) la méthode ne compile plus
   (b) la méthode peut parfois planter (provoquer l'arrêt du programme)
   (c) la méthode fonctionne comme la version d'origine
   (d) le signe du résultat de la nouvelle méthode est identique au signe du résultat de la méthode d'origine
   (e) la méthode fonctionne mais donne des résultats complètement différents de ceux de la méthode d'origine

Exercice 1.5 :

On considère la méthode :

```
1  public static String f(String s) {
2    StringBuffer sb=new StringBuffer(s);
3    int i;
4    for(i=0;i<sb.length();i++) {
5      char c=sb.charAt(i);
6      sb.setCharAt(i,sb.charAt(sb.length()-i-1));
7      sb.setCharAt(sb.length()-i-1,c);
8    }
9    return sb.toString();
10 }
```

1. La méthode devrait renvoyer le miroir de la chaîne paramètre, c'est-à-dire une chaîne de caractères obtenue en retournant la chaîne paramètre. En fait, la méthode ne fonctionne pas :
   (a) elle renvoie une chaîne égale à la chaîne d'origine
   (b) elle ne compile pas
   (c) elle fonctionne seulement avec les chaînes de longueur strictement supérieure à 1 et plante sinon
   (d) elle plante toujours
2. L'erreur principale est située à la ligne :
   (a) 4
   (b) 6
   (c) 5
   (d) 7
3. Donner une version correcte de cette méthode, en modifiant le moins possible la version proposée. Il est interdit d'utiliser la méthode `reverse` des `StringBuffer`s.

Exercice 1.6 :

On considère la méthode suivante :

```
1 public static String buildString(String s) {
2   String r="";
3   for(int i=0;i<s.length();i++) {
4     if(keepChar(s.charAt(i),i)) {
5       r=r+s.charAt(i);
6     }
7   }
8   return r;
9 }
```

On remarque que la méthode utilise la méthode `keepChar`.

1. On suppose que la méthode `keepChar` est la suivante :

```
1 public static boolean keepChar(char c,int position) {
2   return true;
3 }
```

   Que fait alors la méthode `buildString` (justifiez votre réponse) ?

2. On suppose que la méthode `keepChar` est la suivante :

```
1 public static boolean keepChar(char c,int position) {
2   if(position%2==1) {
3     return true;
4   } else {
5     return false;
6   }
7 }
```

   Quel est le résultat de la méthode `buildString` si on lui transmet la chaîne `"NBVCX"` ? Que fait la méthode `buildString` ?

3. Pour chaque résultat souhaité pour la méthode `buildString` proposez une méthode `keepChar` qui permette son obtention :

   (a) La chaîne renvoyée par `buildString` doit contenir un caractère sur trois de la chaîne paramètre, en conservant le caractère de position 1, puis celui de position 4, etc. L'image de `"ABCDEF"` est donc `"BE"`.

   (b) La chaîne renvoyée par `buildString` doit contenir les caractères qui ne sont pas des chiffres contenus dans la chaîne paramètre. L'image de `"AB2C3D6EF"` est donc `"ABCDEF"`.

   Vous pourrez utiliser la classe `Character` qui propose une méthode `isDigit` qui à un `char` associe `true` si et seulement si ce caractère correspond à un chiffre.

2 Programmation

Exercice 2.1 :

Ecrire une méthode qui compare deux chaînes de caractères au regard de l'ordre lexicographique (l'ordre du dictionnaire pour les caractères non accentués). On n'utilisera pas la méthode `compareTo` mais on cherchera à produire des résultats identiques à ceux de cette méthode.

Exercice 2.2 :

Ecrire une méthode qui à une chaîne de caractères associe le nombre de caractères distincts que celle-ci contient.

Exercice 2.3 :

Ecrire une méthode qui permet la saisie d'une chaîne de caractères ne contenant que des chiffres. Ecrire une méthode qui à une chaîne de caractères ne contenant que des chiffres associe le nombre correspondant (sous forme d'un `int`). Ecrire une méthode réalisant l'opération inverse.

Exercice 2.4 :

Écrire un programme qui demande à l'utilisateur un texte et un mot, puis affiche pour chaque lettre du mot, le nombre d'occurrences de cette lettre dans le texte de départ.

Exercice 2.5 :

Ecrire une méthode qui à une chaîne de caractères associe le nombre de mots que celle-ci contient. Ecrire ensuite une méthode qui calcule la longueur moyenne des mots d'un texte.

Exercice 2.6 :

Ecrire une méthode qui à une chaîne de caractères associe `true` si et seulement si la phrase qu'elle contient vérifie les règles élémentaires de typographie : la phrase commence par une majuscule et termine par un point. Les mots sont séparés par exactement un espace (sauf en cas de symbole de ponctuation). La virgule et le point sont collés au mot qui les précède et sont suivis par un espace. Le point-virgule et les deux points sont précédés et suivis par un espace.

Exercice 2.7 :

Ecrire une méthode qui reçoit une chaîne de caractères de longueur impaire et l'affiche sous la forme d'un sablier et d'un noeud papillon. Par exemple :

```
bonjour          b     r
 onjou           bo   ur
  njo            bon our
   j             bonjour
  njo            bon our
 onjou           bo   ur
bonjour          b     r
```

Exercice 2.8 :

Ecrire une méthode qui à deux chaînes de caractères `s1` et `s2` associe la chaîne constituée des caractères de `s1` qui n'apparaissent pas dans `s2` (dans le même ordre).

Exercice 2.9 :

Ecrire une méthode qui reçoit en paramètre une chaîne de caractères et l'affiche sur deux lignes, en affichant sur la première ligne les caractères de rang impair et sur la seconde les caractères de rang pair. Par exemple, pour `"ALGORITHME"`, on obtient :

```
A G R T M
 L O I H E
```

Exercice 2.10 :

On donne l'algorithme de cryptage élémentaire (et très inefficace) suivant :

Données :
- une chaîne de caractères `t` : le texte à coder ;
- une chaîne de caractères `c` : la clé (de longueur strictement inférieure à celle de `t`).

Résultat : une chaîne de caractères cryptée.

1. créer une chaîne vide `s`.
2. pour chaque caractère `x` de `t` (dans l'ordre) :
   (a) extraire de `c` le caractère de position `i` définie comme suit : soit `j` la position de `x` dans `t`. `i` est le reste de la division de `j` par la longueur de `c`.
   (b) ajouter au code unicode de `x` celui de `c`.
   (c) ajouter à la chaîne `s` le caractère correspondant au code unicode qui vient d'être obtenu.
3. Résultat : la chaîne `s`.

Programmez une méthode de codage et une méthode de décodage utilisant cet algorithme.

Exercice 2.11 :

Etant donnés une chaîne de caractères `s` et un caractère `c`, on peut découper `s` selon `c` de la manière suivante : si `s` ne contient pas `c`, l'unique sous-chaîne u_0 est la chaîne `s` entière. Sinon, `s` est formée d'une première partie u_0 ne contenant pas `c`, suivie de `c`, suivie d'une seconde partie s_1. Si s_1 n'est pas la chaîne vide, on peut recommencer l'opération sur celle-ci, en cherchant `c` dans s_1, ce qui permet de construire u_1 et s_2, etc.

Ecrire une méthode qui à une chaîne de caractères `s`, un caractère `c` et un entier `n` associe la sous-chaîne u_n (éventuellement vide).

3 Problèmes

Problème 3.1 :

Pour répondre aux questions de ce problème, on pourra en général donner deux solutions : une avec les `String`s, l'autre avec les `StringBuffer`s. Les deux solutions sont intéressantes à comparer, tant au niveau de l'efficacité que de la simplicité de l'écriture.

1. Écrire une méthode `build` prenant comme paramètres un caractère `c` et un entier positif `n`, et renvoyant la chaîne constituée de `n` fois le caractère `c`. Si le paramètre entier est négatif ou nul, la méthode devra renvoyer la chaîne de caractères vide.
   Exemple : le résultat de `build('a',3)` est donc la chaîne de caractères `"aaa"`.

2. Écrire une méthode `combine` prenant comme paramètres deux caractères `a` et `b`, et deux entiers `n` et `p`. En utilisant la méthode `build`, `combine` devra associer à ces paramètres la chaîne de caractères constituée de `n` fois `a`, suivi de `p` fois `b`, puis encore `n` fois `a`.

   Exemple : le résultat de `combine('u','V',3,2)` est donc la chaîne `"uuuVVuuu"`. La méthode doit fonctionner même si `n` ou `p` est nul.

3. Écrire une méthode `pyramide` prenant comme paramètre un entier strictement positif `n`. Cette méthode devra afficher une pyramide de hauteur `n` à l'écran (en utilisant **impérativement** la méthode `combine`). Un tel affichage comprend `n` lignes. La première ligne comprend `n-1` espaces, un `+` et encore `n-1` espaces. Chaque ligne est obtenue en remplaçant par des `+` les espaces qui entourent la chaîne des `+` dans la ligne précédente. Par exemple, la seconde ligne comporte `n-2` espaces, suivis de 3 `+`, suivis de `n-2` espaces. La dernière `ligne` contient donc `2n-1` fois le caractère `+`.

   Exemple : ceci donne pour `n` égal à 4 l'affichage suivant :

   ```
   ___+___
   __+++__
   _+++++_
   +++++++
   ```

   Pour rendre l'affichage plus clair, les espaces ont été remplacés par le caractère `_` (on utilisera cette notation dans tout l'énoncé).

4. Calculer (en justifiant mathématiquement votre résultat) u_n, le nombre de caractères `+` affiché par l'appel `pyramide(n)`.

5. Le caractère `'\n'` s'affiche à l'écran sous forme d'un passage à la ligne (il s'agit dien d'un seul caractère). Si on exécute par exemple :

   ```
   System.out.println("a\nb") ;
   ```

   On obtiendra l'affichage suivant :

   ```
   a
   b
   ```

   En utilisant cette propriété, écrire une méthode `pyramide2` qui effectue les mêmes opérations que `pyramide`, mais place le dessin de la pyramide dans une chaîne de caractères au lieu de l'afficher, et renvoie la chaîne résultat.

   Exemple : le résultat de `pyramide2(2)` sera donc la chaîne `_+_\n+++` (dans laquelle `_` remplace le caractère espace).

6. Écrire une méthode `compte` qui à une chaîne de caractères `s` et un caractère `c` associe le nombre d'apparitions de `c` dans `s`.

   Exemple : l'appel `compte("bbbAABBb",'b')` renvoie donc `4`.

7. Écrire une méthode `remplace` prenant comme paramètres une chaîne de caractères `s`, un entier positif ou nul `n` et deux caractères `a` et `b`. La méthode devra renvoyer une nouvelle chaîne obtenue en remplaçant dans `s` la `n`-ième apparition du caractère `a` par le caractère `b`.

   Exemple : l'appel `remplace("AAABBBAA",2,'B','-')` donne ainsi comme résultat la chaîne `"AAAB-BAA"`.

8. Écrire une méthode `modifie` prenant comme paramètres une chaîne de caractères `s` et deux caractères `a` et `b`. La méthode renvoie une chaîne de caractères obtenue en remplaçant **une** apparition de `a` par `b` dans `s`, choisie aléatoirement. On utilisera impérativement les méthodes `random` (de la classe `Math`) et `remplace`.

Exemple : le résultat de l'appel `modifie("AAABBBAA",'B','-')` pourra être `"AAA-BBAA"`, `"AAAB-BAA"` ou `"AAABB-AA"`, selon l'apparition choisie.

9. Écrire une méthode `pyramide3` prenant comme paramètre un entier `n`. Cette méthode commencera par utiliser `pyramide2` pour fabriquer une pyramide dans une chaîne de caractères, puis elle appellera plusieurs fois `modifie` afin de remplacer 25% des caractères `+` par `*`. Elle aura pour résultat la chaîne de caractères ainsi obtenue.

   Exemple : comme `modifie` est aléatoire, nous n'indiquons ici qu'un seul résultat possible pour l'appel `pyramide3(3)`, à savoir la chaîne

   ```
   __+__\n_***_\n+++++
   ```

   Cette chaîne s'affiche de la façon suivante :

   ```
   __+__
   _***_
   +++++
   ```

   Notons bien que `pyramide3` **ne fait pas d'affichage**.

10. Écrire une méthode `développe` prenant comme paramètre deux chaînes de caractères `s` et `t`, et un caractère `c`. Le résultat de la méthode est la chaîne `s` dans laquelle chaque apparition de `c` a été remplacée par la chaîne `t`.

    Exemple : le résultat de `développe("bAbAAd","XoY",'A')` est donc la chaîne `"bXoYbXoYXoYd"`.

11. On appelle $p_{\texttt{n}}$ la chaîne de caractères produite par l'appel `pyramide2(n)`, c'est-à-dire la pyramide de hauteur `n`. On peut montrer que $p_{\texttt{n+1}}$ s'obtient à partir de $p_{\texttt{n}}$ par l'algorithme suivant :

    (a) `u` s'obtient en ajoutant un espace au début et à la fin de $p_{\texttt{n}}$;

    (b) `v` est le résultat du remplacement dans `u` de `\n` par la chaîne `"_\n_"` (l'espace est ici remplacé par `"_"`) ;

    (c) $p_{\texttt{n+1}}$ s'obtient enfin en ajoutant à `v` le caractère `\n` puis la dernière ligne de la pyramide (celle qui ne contient que des `+`).

    Programmer une méthode `développe2` qui applique à une chaîne `s` et un entier `n` l'algorithme qui vient d'être présenté, et renvoie la chaîne résultat. Pourquoi le paramètre `n` est-il indispensable ? Notons qu'il correspond à $p_{\texttt{n}}$ et que l'appel `développe2(`$p_{\texttt{n}}$`,n)` doit donc produire comme résultat la chaîne $p_{\texttt{n+1}}$.

12. Programmer une méthode `pyramide4` qui produit le même résultat que `pyramide2` mais en utilisant la méthode `développe2`.

13. Programmer une méthode `randomChar`, prenant comme paramètre un entier `n` et renvoyant soit le caractère `+`, soit le caractère `*`. Plus précisément, la méthode choisit aléatoirement un entier compris entre 1 et `n` et renvoie `*` si et seulement si cet entier est égal à `1`. En moyenne on obtient donc le caractère `+` `n-1` fois sur `n`. Programmer une méthode `combineAléa` prenant comme paramètres deux entiers `n` et `p` et fabrique une chaîne de `p` caractères obtenue en utilisant `randomChar` avec comme paramètre `n`. Programmer enfin une méthode `pyramide5` qui obtient directement un résultat similaire à `pyramide3`. Quelles sont les différences entre les résultats de deux méthodes ?

Problème 3.2 :

On souhaite étudier dans une chaîne de caractères les séquences de caractères identiques, c'est-à-dire les séquences de la forme `"aaaa"`. Il faut bien sûr qu'une séquence de caractères identiques comporte au moins deux caractères pour qu'elle soit prise en compte !

Questions :

1. Écrire une méthode qui à une chaîne de caractères associe une nouvelle chaîne de caractères obtenue en remplaçant chaque séquence de lettres identiques par la première d'entre elles (par exemple `"aabb"` devient `"ab"`, `"aaabbc"` devient `"abc"`, etc.).
2. Écrire une méthode qui à une chaîne de caractères associe le nombre de séquences de caractères identiques qu'elle contient (pour `"aaabbc"` on obtient 2).
3. Écrire une méthode qui à une chaîne de caractères associe un booléen `true` si elle contient au moins une séquence de caractères identiques et `false` sinon. On **n'utilisera pas** la méthode solution de la question précédente.
4. Écrire une méthode qui à une chaîne de caractères associe la longueur de la plus longue séquence de caractères identiques qu'elle contient et 0 si elle n'en contient pas (pour `"aaabbc"` on obtient 3).
5. Écrire une méthode qui à une chaîne de caractères associe la moyenne des longueurs des séquences de caractères identiques qu'elle contient et 0 si elle n'en contient pas. Pour la chaîne `"aaabbcbbaad"`, on obtient 2.25. En effet, elle contient 4 séquences, trois de 2 lettres et une de 3 lettres, et la moyenne des longueurs est bien 2.25.

Problème 3.3 :

Dans ce problème, on programme un ensemble de méthodes qui permettent d'évaluer une expression contenue dans une chaîne de caractères :

1. écrire une méthode qui à une chaîne de caractères associe `true` si et seulement si elle ne contient que des chiffres, des lettres et des symboles mathématiques (`+`, `/`, `*`, `-` et les parenthèses) ;
2. écrire une méthode qui à une chaîne de caractères associe `true` si elle contient une expression correctement parenthèsée (à chaque parenthèse ouvrante correspond une parenthèse fermante), et `false` sinon ;
3. écrire une méthode qui à une expression correctement parenthèsée associe la chaîne de caractères contenu dans la première paire de parenthèses ne contenant pas d'autre paire de parenthèses. Si on a par exemple la chaîne `((a+(b+c))+(a*b))`, on obtient `b+c` ;
4. écrire une méthode qui calcule la valeur numérique d'une chaîne de caractères contenant un calcul élémentaire de la forme "opérande numérique 1 opérateur opérande numérique 2", où les opérandes numériques sont des nombres entiers ;
5. écrire une méthode qui calcule la valeur numérique d'une chaîne de caractères contenant un calcul comportant éventuellement des parenthèses mais dans lequel chaque expression sans parenthèse se réduit à un calcul de la forme élémentaire utilisé dans la question précédente.

Problème 3.4 :

Dans ce problème, nous allons apprendre à transformer une `String` en un nombre entier ou réel. S'il existe déjà une méthode `Java` qui réalise exactement (ou presque exactement) ce qui est demandé dans l'énoncé, son utilisation est interdite dans le problème. Si on vous demandais par exemple d'écrire une méthode qui calcule x^n, vous n'auriez pas le droit d'utiliser la méthode `pow` de la classe `Math`. Par contre, sauf si le contraire est précisé dans l'énoncé, vous pouvez écrire une méthode en vous servant d'une méthode définie avant (même si vous n'avez pas su répondre à la question). Vous pouvez aussi écrire d'autres méthodes si cela simplifie la solution.

1. Ecrire une méthode `int charToInt(char t)` qui transforme un chiffre représenté par un `char` en sa valeur sous forme de `int`. Le résultat de `charToInt('2')` sera par exemple `2`. Si le paramètre `t` n'est pas un chiffre, la méthode devra renvoyer la valeur `-1`.
2. Ecrire une méthode `int toPositiveInt(String s)` qui transforme une chaîne de caractères contenant exclusivement des chiffres en l'entier correspondant. Pour cela, on utilisera la méthode `charToInt` définie dans la question précédente. De plus, on rappelle que si un entier s'écrit $A_{n-1}A_{n-2}\dots A_1A_0$, où les A_i désignent les chiffres de l'entier, alors la valeur numérique de l'entier est donnée par $\sum_{k=0}^{n-1} A_k 10^k$. Il faudra être très attentif au fait que les caractères d'une `String` sont numérotés de gauche à droite, soit le contraire de la numérotation utilisée pour les chiffres d'un nombre. Enfin, si la chaîne de caractères ne représente pas un nombre entier positif, la méthode devra renvoyer `-1`.
3. On souhaite maintenant traduire une chaîne de caractères en un nombre réel. Nous allons procéder par étapes :
   (a) écrire une méthode `int position(String s,char c)` qui donne la position de la première apparition du caractère `c` dans la chaîne `s`, et renvoie `-1` si le caractère n'apparaît pas dans la chaîne.
   (b) écrire une méthode `String substring(String s,int début,int fin)` qui fabrique la chaîne de caractères constituée des caractères de `s` d'indices `début`, `début+1`, etc. jusqu'à `fin-1`. Le résultat de `substring("ABCDEFGH",3,6)` est donc la chaîne `"DEF"`. Si les paramètres numériques ne sont pas adaptés à la chaîne de départ (trop grand, trop petit, etc.), la méthode devra renvoyer une chaîne vide.
   (c) écrire une méthode `double toPositiveReal(String s)` qui convertit son paramètre en un réel en procédant de la façon suivante :
      i. la méthode repère la position dans `s` du point décimal, le caractère `'.'`
      ii. la méthode découpe `s` en deux chaînes : la partie fractionnaire (après le `'.'`) et la partie entière (avant le `'.'`)
      iii. la méthode transforme les deux parties en `int`
      iv. la méthode combine les deux entiers obtenus afin de fabriquer le réel attendu

      On doit bien sûr utiliser les méthodes définies dans les questions précédentes.
   (d) écrire une méthode `double toPositiveReal2(String s)` qui réalise la même opération que la méthode précédente mais en se basant sur le fait suivant : si un réel s'écrit
   $$A_{n-1}A_{n-2}\dots A_1A_0.A_{-1}A_{-2}\dots A_{-p}$$
   où les A_i désignent les chiffres du réel, alors la valeur numérique de ce réel est donnée par $\sum_{k=-p}^{n-1} A_k 10^k$. Il faut donc déterminer `p` et `n`, puis travailler de façon assez similaire à celle utilisée pour la méthode `toPositiveInt`, en faisant de nouveau attention au sens de la numérotation des caractères dans les `String`s. Il est bien entendu interdit d'utiliser la méthode `toPositiveReal` pour construire la nouvelle méthode.