Université de Paris

L1 informatique - Année 2020–2021

TD de Concepts Informatique n° 2

Exercice 1. *Manipulation de la mémoire : références et objets.*
Considérer les extraits de programme suivants

```
1  int n = 5;
2  int [] t = new int [3];
3  for (int i = 0; i < 3; i++) {
4    t[i] = i * i ;
5  }
6
7  System.out.println(t);
8  System.out.println(t[2]);
9  System.out.println(n);
10
11 int [][] m = new int[2][3];
12 m[0] = t;
13 m[1][0] = n;
14
15 for (int i = 0; i < 2; i++) {
16   for(int j = 0; j < 3; j++) {
17     m[i][j] = i+j;
18   }
19 }
20
21 System.out.println(t);
22 System.out.println(t[2]);
23 System.out.println(n);
```

Extrait 1.a)

```
1 int[][] p = new int[4][];
2 for(int i=0; i<4; i++){
3     p[i] = new int[i+1];
4     p[i][0] = 1;
5     for(int j=1; j<i; j++)
6         p[i][j] = p[i-1][j-1]+p[i-1][j];
7     p[i][i] = 1;
8 }
```

Extrait 1.b)

```
1  public class Point2D {
2    double x;
3    double y;
4
5    public Point2D(double x, double y) {
6      this.x = x;
7      this.y = y;
8    }
9
10   public Point2D milieu(Point2D p) {
11     double xm = (this.x + p.x) / 2;
12     double ym = (this.y + p.y) / 2;
13     return new Point2D(xm, ym);
14   }
15
16   public static void main(String[] args){
17     Point2D a = new Point2D(1,4.5);
18     Point2D b = new Point2D(3,2.5);
19     Point2D c = a.milieu(b);
20   }
21 }
```

1. En distinguant pile et tas, faire deux schémas pour illustrer le contenu de la mémoire après avoir exécuté la ligne 13, respectivement 19 dans l'extrait 1.a).
2. La même valeur sera-t-elle affichée sur les lignes 7 et 21 dans l'extrait 1.a) ? La même question pour les lignes 8 et 22, respectivement pour les lignes 9 et 23.
3. Faire un schéma pour illustrer l'évolution du contenu de la mémoire lors de l'exécution des extraits 1.b) et 1.c) (en faisant des hypothèses pour les adresses des variables, en décidant aussi que le contenu de la mémoire est représenté en hexadécimal, et en détaillant les étapes les plus importantes des constructions opérées).

Exercice 2. *Piles et expressions bien parenthésées.*
Cet exercice s'intéresse aux expressions bien parenthésées.

1. Écrire une fonction de prototype `boolean bienParenthesee(String)` qui vérifie si une chaîne de caractères est bien parenthésée en utilisant une pile. En particulier, elle ignore tous les caractères autres que `(` et `)` (non pertinents). La fonction :

- renvoie `true` pour des mots tels que `()`, `()(())` ou `(((())()))`
- renvoie `false` pour des mots tels que `)(`, `()(()` ou `(((()))))`.

Pourrait-on facilement se passer d'une pile ?

2. Modifier la fonction pour qu'elle puisse reconnaître les expressions bien parenthésées utilisant deux types de parenthèses, `( )` et `[ ]`. Pour cette question, la fonction :
   - renvoie `true` pour des mots tels que `()[]`, `([()])` ou `[(()())]([[]()])`
   - renvoie `false` pour des mots tels que `([)]`, `()[` ou `[()[]]]`.

   Pourrait-on facilement se passer d'une pile ?

Exercice 3. *Partiel 2014 (à faire chez vous).*
Un document XML est un document texte structuré tel le document suivant :

```
<concepts>
  <enseignants>
    <enseignant>Machin</enseignant>
    <enseignant>Bidule</enseignant>
  </enseignants>
  <etudiants>
    <etudiant>Eleanor</etudiant>
    <etudiant>Fares</etudiant>
    <etudiant>Thobias</etudiant>
  </etudiants>
</concepts>
```

Nous nous intéressons dans cet exercice aux *balises*, c'est-à-dire aux éléments structurants du document, comme `<etudiant>` ou `</enseignant>`. Ces balises viennent par couples : à toute balise ouvrante `<balise>` correspond nécessairement une balise fermante `</balise>` plus loin dans le document. D'autre part, entre deux balises ouvrante/fermante associées, on peut imbriquer autant de couples ouvrant/fermant que l'on veut, du type que l'on veut ; mais il est interdit d'entremêler deux couples. L'autre particularité est qu'un document XML est toujours constitué d'un couple de balises englobant tout le reste (le couple `<concepts></concepts>` dans l'exemple).
On suppose qu'un document XML est encodé dans une classe `DocumentXML` et on dispose d'une méthode de prototype `String getNextTag()` qui, appelée itérativement, renvoie dans l'ordre les balises XML d'un document (le reste du document est proprement ignoré). Lorsqu'il n'y a plus de balises à lire, la référence renvoyée est simplement `null`. Par exemple, si la variable `doc` de type `DocumentXML` contient une référence vers le document XML présenté ci-dessus, le premier appel `doc.getNextTag()` renverra la chaîne de caractères `"<concepts>"` et l'appel suivant `doc.getNextTag()` renverra la chaîne `"<enseignants>"`.

1. Donner trois exemples de documents mal formés du point de vue de la structure XML. Attention, les malformations doivent être de nature différente.
2. Écrire une méthode de prototype `boolean estCorrect(DocumentXML)` qui teste si le document lu via des appels à `getNextTag` est un document XML correctement formé.