Introduction à la Programmation 1 JAVA

51AE011F

Contrôle Continu 1

Durée: 1h

Documents et équipements électroniques interdits

1 Booléens

Exercice 1 *Simplifier les expressions booléennes suivantes :*

1. `x > 5 && x > 7`
2. `x == 17 || (x != 17 && x == 42)`
3. `x > 5 || (x <= 5 && y > 5)`
4. `!(b1 == true && b2 != false)`

Réponse □

Exercice 2 *Proposer un équivalent au code Java suivant utilisant un seul* `if`.

```
if (a==b) { d=1; }
else {
  if (c==d) { }
  else { d=1; }
}
```

Réponse □

2 Boucles

Exercice 3 *Que fait la fonction suivante ? Donner ses affichages et sa valeur de retour si on la lance avec l'argument 8.*

```
public static int mystere (int n) {
  int r=0;
  for (int i=1; i<=n; i=i+2) {
    r = r + i;
    System.out.println(Integer.toString(r));
  }
  return(r);
}
```

Réponse □

Exercice 4 *Ecrire une fonction* `testCarre` *qui répond* `true` *si et seulement si si le nombre entier reçu en argument est un carré d'entier. On n'utilisera pas la fonction racine carrée approchée de Java (*`Math.sqrt`*). Que répondra* `testCarre(mystere(100))`*, où* `mystere` *est la fonction de l'exercice précédent ?* Réponse □

3 Chaînes

On rappelle que la longueur d'une chaîne de caractère `s` s'obtient via `s.length()`, et que l'on compare deux chaînes `s1` et `s2` via : `s1.equals(s2)`. On suppose également disposer ici de la fonction suivante : `public static String characterAtPos(String s, int i)`.

Exercice 5 *Ecrire une fonction* `miroir` *qui prend en argument une chaîne de caractère et retourne une version renversée de cette chaîne. Par exemple :*

argument : "abracadabra" réponse : "arbadacarba"

Réponse □

Exercice 6 *Ecrire une procédure* `occurences` *qui prend en paramêtre deux chaînes de caractères* `s1` *et* `s2`*, et affiche toutes les positions de* `s1` *à partir desquelles* `s2` *apparaît comme sous-chaîne. Par exemple :*

s1 : "abracadabra" s2 :"abra" affichage : 0 7

N'hésitez pas à écrire et utiliser toute sous-fonction qui vous semblerait nécéssaire.

Réponse □

4 Etoiles

Exercice 7 *Ecrire des procédures Java permettant de réaliser chacun des affichages suivants, généralisés à* `n` *lignes au lieu de 5.*

1.
```
*****
****
***
**
*
```

2.
```
*    *
 * *
  *
 * *
*    *
```

3.
```
    *
   ***
  *****
 *******
*********
```

Réponse □

5 Réponses aux exercices

Exercice 1, page 1

Les trois premières expressions ont été vues en cours : `x>7` `x==17||x==42` `x>5||y>5`.
Et pour la dernière : `!b1 || !b2` ou bien encore `!(b1 && b2)`

Exercice 2, page 1

```
if (a==b || c!=d) { d=1; }
```

Exercice 3, page 1

La fonction mystere calcule la somme des nombres impairs compris entre 0 et n (inclus). Ce n'était pas vraiment demandé, mais on peut déterminer que cette somme vaut $(ceil(n/2))^2$ si $ceil$ donne l'entier arrondi vers le haut ($ceil(2) = 2$, $ceil(2.5) = 3$). mystere(8) affiche 1 4 9 16 et retourne le résultat final 16.

Exercice 4, page 1

Il y avait de multiples manières possibles de faire, voici un exemple de réponse qui convenait ici :

```java
public static boolean testCarre (int n) {
    for (int i=0; i<=n; i++) {
        if (i*i == n) { return true; }
    }
    return false;
}
```

Plus de détails, ainsi que d'autres solutions plus efficaces (et plus correctes quand n grandit !) ici : http://www.irif.fr/~letouzey/ip1/SquareRoot.java. Enfin, pour `testCarre(mystere(100))`, il fallait remarquer que mystere répondait toujours des nombres carrés, donc testCarre va ici répondre `true`.

Exercice 5, page 2

Vu en cours. Par exemple :

```java
public static String miroir (String s) {
    String res = "";
    for (int i=0; i < s.length(); i++) {
        res = characterAtPos(s,i) + res;
    }
    return res;
}
```

Exercice 6, page 2

Il est pratique d'écrire d'abord une fonction auxiliaire `sub` telle que `sub(s,p,n)` donne la sous-chaîne de s commençant à la position p et de longueur n. Ensuite on visite toutes les positions de s1 qui ne sont pas trop proches de la fin de s2.

```java
public static String sub(String s, int p, int n) {
    String res = "";
    for (int i=0; i < n; i++) {
        res = res + characterAtPos(s,p+i);
    }
    return res;
}

public static void occurrences (String s1, String s2) {
    int n1 = s1.length();
    int n2 = s2.length();
    for (int i=0; i <= n1-n2; i++) {
        if (s2.equals(sub(s1,i,n2))) {
            System.out.print(Integer.toString(i) + " ");
        }
    }
    System.out.println();
}
```

Remarque : pour la recherche d'une chaîne dans une autre, il y a nettement plus efficace, voir par exemple l'algorithme KMP : https://fr.wikipedia.org/wiki/Algorithme_de_Knuth-Morris-Pratt

Exercice 7, page 2

```java
public static void etoiles1 (int n) {
    for (int i=n; i > 0; i--) {
        for (int j=0; j<i; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
}

public static void etoiles2 (int n) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (i==j || i+j == n-1) {
                System.out.print("*");
            } else {
                System.out.print(" ");
            }
        }
        System.out.println();
    }
}

/* Une petite fonction auxiliaire qui affiche n fois la chaîne s */
public static void n_affiche (String s, int n) {
    for (int i=0; i<n; i++) { System.out.print(s); }
}

public static void etoiles3 (int n) {
    for (int i=0; i<n; i++) {
        n_affiche(" ", n-i-1);
        n_affiche("*", 2*i+1);
        System.out.println();
    }
}
```