université PARIS DIDEROT PARIS 7

Examen	CI2	24/6/2016 13^{00}–16^{00}

Aucun document ou support autre que le sujet ou les copies d'examen n'est autorisé.
(la copie ou les brouillons du voisin ne sont pas des supports autorisés).
Éteignez impérativement vos mobiles.

Lorsque des calculs sont nécessaires, il est impératif de les présenter sur la feuille d'examen. Il est aussi nécessaire de **justifier** ses réponses.

1 Exercice

Soit la fonction suivante :

```
public static int rec(int n,int m) {
    if (m==0) return n;
    if (m%2==0) return 2*rec(n,m/2);
    return rec(n,m-1);
}
```

1. Quelle valeur est produite par un appel à `rec(1,7)` ? `rec(3,6)` ? Détaillez.
2. Quelles valeurs produisent (probablement) un `StackOverflowError` ?
3. Qu'est-ce qu'un appel récursif ? récursif terminal ?
4. Transformez cette fonction en une fonction de nom `rect` qui soit récursive terminale.
5. Transformez la fonction `rect` en une fonction de nom `recti` qui ne contienne qu'une itération.

2 Exercice

```
public class T {
  public static int eu(int n,int m) {
    while (n!=0 && m!=0) {
      if (n<m) m -= n;
      else     n -= m;
    }
    if (n==0) return m;
    return n;
  }
  public static int a;
  public static int b;
  public static void main(String []args) {
    a = 18;
    b = 27;
    System.out.println(eu(a,b));
  }
}
```

1. Combien d'entiers sont nécessaires au fonctionnement de ce programme ? Détaillez.
2. Traduisez ce programme tel que vu en cours et TD, c'est-à-dire en n'utilisant qu'un tableau statique d'entiers appelé `memoire`, une variable entière appelée `instructionCourante` (ou `ic`), un seul `switch` et un seul `while`. Précisez comment l'appel est réalisé.

3 Exercice

On souhaite mélanger un jeu de n cartes chacune représentée par un entier (de 0 à $n-1$) et en n'utilisant que des piles de type `Stack<Integer>`. On n'utilisera de la classe `Stack` que les méthodes `void push(Integer i);` (qui permet d'ajouter un élément en sommet), `Integer pop();` (qui permet d'enlever et retrouver l'élément en sommet) et `boolean empty();` (qui permet de déterminer si une pile est vide). **Aucune autre méthode de**

la classe `Stack` ne doit être employée. Il est interdit, pour chacune de ses fonctions, d'utiliser autre chose que des variables entières ou booléennes.
Au départ le jeu est ordonné et stocké dans une unique pile. Ensuite, chaque étape du mélange consiste à : distribuer les cartes une par une sur t tas (où $1 < t \leq n$ est déterminé au hasard), puis à regrouper ces t tas en un seul.

1. Écrire une fonction de prototype `public static Stack<Integer> creerJeu(int n)` qui permet d'obtenir le jeu initial trié.
2. Écrire une fonction de prototype
   `public static Stack<Integer> [] distribue(Stack<Integer> jeu,int t` qui renvoie un tableau de `t` piles dans lesquelles se trouvent les ≪cartes≫ du `jeu` reçu en paramètre et distribuées de sorte que la première est déposée dans le premier tas, la seconde dans le second, etc. Lorsqu'on arrive au dernier tas, on ≪repart≫ au premier, jusqu'à épuisement du jeu.

   Si, par exemple, on découpe le jeu $(0, 1, 2, 3, 4, 5, 6)$ en 3 tas on obtient $(6, 3, 0)$, $(5, 2)$, $(4, 1)$
3. Écrire une fonction de prototype `public static Stack<Integer> regroupe(Stack<Integer> []tas)` qui reçoit un tableau de ≪tas≫ et renvoie un unique jeu constitué de l'ensemble des cartes des différents tas, regroupées tas par tas.

   Si, par exemple, on regroupe les 3 tas $(6, 3, 0)$, $(5, 2)$, $(4, 1)$, on obtient le jeu $(0, 3, 6, 2, 5, 1, 4)$
4. En utilisant la classe `java.util.Random` qui doit être instanciée par `new Random()` et sa méthode `int nextInt(int n)` qui permet d'obtenir un entier tiré aléatoirement dans l'ensemble $[0, n[$, écrire la fonction `Stack<Integer> melange(int n,int m);` qui permet d'obtenir un jeu de n cartes mélangé m fois.