**ASD**

*Corrigée du Contrôle continu n°*2

- Définition de la classe TNode et du début de la classe BookTree :

 (2 points)

**public** **class** TNode {

 **private int** num, copies;

 **private** TNode left, right;

 **public** TNode(**int** num, **int** copies, TNode left, TNode right) {

 **this**.num = num;

 **this**.copies = copies;

 **this**.left = left;

 **this**.right = right;

 }

 /\*\*

 \* getNum(), setNum(int), getCopies(), setCopies(int),

 \* getLeft(), setLeft(TNode), getRight(), setRight(TNode)

 \*/

 ...........................

}

**public** **class** BookTree {

 **private** TNode root;

 **public** BookTree () {

 root = **null**;

 }

- (2 points)

 **public** **void** available() {

 available(root);

 }

 **private** **static void** available(TNode r) {

 **if** (r != **null**) {

 available(r.getLeft());

 **if** (r.getCopies() > 0)

 System.***out***.println(r.getNum());

 available(r.getRight());

 }

 }

- (2 points)

 **public** **int** amount() {

 **return** *amount*(root);

 }

 **private** **static** **int** amount(TNode r) {

 **if** (r == **null**)

 **return** 0;

 **else**

 **return** *amount*(r.getLeft()) + *amount*(r.getRight())

 + r.getCopies();

 }

- (2 points)

 **private** TNode searchBook(**int** num) {

 TNode r = root;

 **while** (r != **null**)

 **if** (r.getNum() == num)

 **return** r;

 **else** **if** (num < r.getNum())

 r = r.getLeft();

 **else**

 r = r.getRight();

 **throw** **new** NoSuchElementException();

 }

- (1,5 points)

 **public** **void** lend(**int** num) {

 TNode r = searchBook(num);

 **if** (r.getCopies() > 0)

 r.setCopies(r.getCopies() - 1);

 **else**

 System.***out***.println(num + " n'est pas disponible !");

 }

- (1 point) Une solution :

2051 3

Root

5124 0

2001 1

3655 2

1635 2

1932 0 00

- Définition de la classe LNode et du début de la classe BookList :

 (2 points)

**public** **class** LNode {

 **private int** num, copies;

 **private** LNode next, prec;

 **public** LNode(**int** num, **int** copies, LNode next, LNode prec) {

 **this**.num = num;

 **this**.copies = copies;

 **this**.next = next;

 **this**.prec = prec;

 }

 /\*\*

 \* getNum(),getCopies(),

 \* getNext(), setNext(LNode), getPrec(), setPrec(LNode)

 \*/

 ...........................

}

**public** **class** BookList {

 **private** LNode first, last;

 **public** BookList() {

 first = last = **null**;

 }

 **public** LNode getLast() {

 **return** last;

 }

- (1,5 points)

 **public** **void** removeLast() {

 **if** (last == **null**)

 **throw** **new** NoSuchElementException();

 **else** **if** (first == last)

 first = last = **null**;

 **else** {

 last.getPrec().setNext(**null**);

 last = last.getPrec();

 }

 }

- (1,5 points)

 **public** **int** size() {

 LNode p = first;

 **int** size = 0;

 **while** (p != **null**) {

 size++;

 p = p.getNext();

 }

 **return** size;

 }

- (3 point)

 **private** **static** **void** load(BookList blist, TNode r) {

 **if** (r != **null**) {

 *load*(blist, r.getLeft());

 r.setNum(blist.getLast().getNum());

 r.setCopies(blist.getLast().getCopies());

 blist.removeLast();

 *load*(blist, r.getRight());

 }

 }

 Les éléments sont extraits de blist dans l’ordre du dernier nœud au premier nœud donc dans l’ordre croissant des numéros.

Ils sont ensuite ajoutés dans l’arbre de racine r suivant le parcours *gauche*-*racine*-*droit*.

 Par conséquence, les éléments sont triés par ordre croissant si l’on parcourt l’arbre dans l’ordre *gauche*-*racine*-*droit*. L’arbre obtenu est donc un arbre binaire de recherche.

- (1,5 point)

 **public** BookTree(BookList blist) {

 root = createTree(blist.size());

 *load*(blist, root);

 }

 La méthode load ajoute les livres dans l’arbre de racine root dans l’ordre croissant de leur numéro suivant le parcours *gauche*-*racine*-*droit*. L’arbre obtenu est donc trié selon le numéro des livres comme dans l’exemple.

- Cette méthode n’est pas à définir :

 **private** **static** TNode createTree(**int** n) {

 **if** (n > 0) {

 TNode r = **new** TNode(0, 0, **null**, **null**);

 **int** nLeft = (n - 1)/2;

 r.setLeft(*createTree*(nLeft));

 r.setRight(*createTree*(n - 1 - nLeft));

 **return** r;

 } **else**

 **return** **null**;

 }

 }