TD 11

UFR Math-Info
UE – L2/S4 – Programmation 4

Exercice 1
Après analyse des morceaux de code ci-dessous, remplissez le tableau ci-joint en mettant une croix (X) dans les cases correspondantes aux visibilités des attributs (a, b, c, d) dans les différentes classes.

package p1;
class Clas1{
int a;
public int b;
private int c;
protected int d;
}
	package p1;
	class Clas2 extends Clas1{
	}

package p1;
class Clas3 extends Clas1{
}
	package p2;
	class Clas4 extends Clas1{
	}

package p2;
class Clas5{
}
	class Clas6 extends Clas4{
	}

Visibilité dans	Clas2	Clas3	Clas4	Clas5	Clas6
a					
b					
c					
d					

Exercice 2
Répondez par Vrai ou par Faux aux questions suivantes :

Q1 Une classe ne peut être déclarée *abstract* et *final*
Q2.1 Une méthode déclarée *final* est une méthode qui ne peut pas être surchargée
Q2.2 Une méthode déclarée *final* est une méthode qui ne peut pas être redéfinie
Q3.1 Il est possible de surcharger une méthode déclarée public en la déclarant private
Q3.2 Il est possible de redéfinir une méthode déclarée public en la déclarant private
Q4 Une méthode déclarée private ne peut être déclarée abstract
Q5 Les méthodes d'une interface doivent être explicitement déclarées abstract et public
Q6 Les données membres déclarées dans le corps d'une interface sont implicitement public final et static
Q7 Un bloc try contient un ensemble d'instruction susceptibles de lever des exceptions durant leur exécution

Exercice 3

L'interface *Iterator* de Java

Java propose une API appelée ***Iterator*** dont la structure est présentée ci-dessous :

```
public interface Iterator {
/**
* Returns < tt >true </ tt > if the iteration has more elements.
*
* @return < tt >true </ tt > if the iterator has more elements.
*/
boolean hasNext ();

/**
* Returns the next element in the interation .
*
* @returns the next element in the interation .
* @exception NoSuchElementException iteration has no more elements.
*/
Object next ();

/**  Removes from the underlying collection the last element returned by the iterator
(optional operation).
*/
void remove();
}
```

Dans l'API , il existe les classes Vector et LinkedList qui sont toutes les deux des listes mais dont les structures sont différentes : la première stocke ses éléments dans un tableau et la seconde dans une liste chaînées. Leur comportement étant le même, elles ***implémentent*** toutes les deux l'interface List et possèdent toutes les deux la méthode Iterator iterator() qui retourne un objet de type Iterator, c'est-à-dire un objet qui permet de parcourir la liste avec les méthodes de l'interface Iterator.

Question 1.
Ecrire le morceau de code qui permet de parcourir un vecteur à l'aide d'un objet de type Iterator.
Quel est d'après votre analyse, l'intérêt de cette approche ?

Question 2
Ecrire la classe *TableauIterator* qui implémente l'interface Iterator pour un tableau, on n'implémentera uniquement les méthodes hasNext() et next().