TP JAVA n°4: Collection et Généricité

Exercice 1 : Manipulation des Collections : Comparaison et tri des objets …

1. Dans un nouveau projet **`TestCollections`**, implémenter l'exercice traité en TD sur le ***tri des montagnes***, de manière à obtenir le jeu d'essai ci-dessous. Vous pouvez récupérer la classe **`Montagne.java`** sur la zone libre.

```
--- Des Montagnes ---
Mont Blanc       --> 4807 m d'altitude
Puy de Sancy     --> 1886 m d'altitude
Puy de Dôme      --> 1464 m d'altitude
Pic du Midi      --> 2877 m d'altitude
Pic d'Aneto      --> 3404 m d'altitude
Pic du Canigou   --> 2784 m d'altitude

 --- Tri suivant nom --> appel sort avec 1 paramètre
--- Affichage de la collection après tri suivant nom ---
Mont Blanc       --> 4807 m d'altitude
Pic du Canigou   --> 2784 m d'altitude
Pic du Midi      --> 2877 m d'altitude
Pic d'Aneto      --> 3404 m d'altitude
Puy de Dôme      --> 1464 m d'altitude
Puy de Sancy     --> 1886 m d'altitude

 --- Tri suivant hauteur --> appel sort avec 2 paramètres
--- Affichage de la collection après tri suivant la hauteur ---
Puy de Dôme      --> 1464 m d'altitude
Puy de Sancy     --> 1886 m d'altitude
Pic du Canigou   --> 2784 m d'altitude
Pic du Midi      --> 2877 m d'altitude
Pic d'Aneto      --> 3404 m d'altitude
Mont Blanc       --> 4807 m d'altitude

 --- Tri suivant nom --> appel sort avec 2 paramètres
--- Affichage de la collection après tri suivant nom ---
Mont Blanc       --> 4807 m d'altitude
Pic du Canigou   --> 2784 m d'altitude
Pic du Midi      --> 2877 m d'altitude
Pic d'Aneto      --> 3404 m d'altitude
Puy de Dôme      --> 1464 m d'altitude
Puy de Sancy     --> 1886 m d'altitude
```

2. Modifier ensuite votre comparateur de hauteur pour trier les sommets selon l'ordre décroissant de leur hauteur (du plus haut au plus bas)

Exercice 2 : Cabinet Médical

1. Ecrire l'application **`EssaiCabMed_v3.java`** dans le package **`com.iut.cabinet.essai`**
Cette application va permettre de manipuler une liste d'objets de type `Personne` à l'aide d'une `Collection` (par exemple de type `ArrayList`)

→ Instancier une collection `maListe` qui permet de manipuler une liste d'objets de type `Personne` et peut donc stocker aussi bien des objets de type `Patient` que de type `Professionnel`

→ Ajouter à cette liste, des patients et des professionnels …

→ Afficher le contenu de `maListe` avec une boucle `foreach`.

→ Dans la classe Personne rajouter la méthode suivante :
`public String affichageSimplifie(Personne unePersonne)`
qui renvoie dans un `String` : le nom, le prénom, la date de Naissance et le code postal de la Personne passée en paramètre…
Utiliser cette méthode dans le `foreach`, l'affichage sera ainsi plus lisible

```
--- Contenu de maListe de Personnes  ---
Nom : DUPONT
Prenom : Julie
DateNaissance : 21/05/1960
Code Postal : 87000
-----------------------------------------
Nom : DUPONT
Prenom : Toto
DateNaissance : 25/12/1991
Code Postal : 87065
-----------------------------------------
Nom : LEDOC
Prenom : Paul
DateNaissance : 10/07/1976
Code Postal : 87170
-----------------------------------------
Nom : CHILDREN
Prenom : Rose
DateNaissance : 16/02/1970
Code Postal : 87170
-----------------------------------------
```

2. ***Tri de la collection suivant différents critères*** :
Nous souhaitons trier cette classe suivant plusieurs critères :

→ 2.1 Ecrire un comparateur (classe **`ComparateurNom`**) qui permet de trier la collection suivant le ***nom*** de la `Personne` (pour l'instant nom uniquement, on ne s'occupe pas du prénom…).
Tester ce tri en appelant ce comparateur dans votre fichier **`EssaiCabMed_v3.java`** puis en procédant à un affichage simplifié de votre collection….

→ 2.3 Ecrire un comparateur (classe **`ComparateurCodePostal`**) qui permet de trier la collection suivant le ***code postal*** de la `Personne.` Rajouter ce tri dans votre programme et tester.

→ Ecrire un comparateur (classe **`ComparateurAge`**) qui permet de trier la collection suivant la ***date de naissance*** de la `Personne.` On souhaite afficher en premier la plus jeune personne. Rajouter ce tri dans votre programme et tester.

→ Ecrire un comparateur (classe **`ComparateurOrdreAlpha`**) qui permet de trier la collection suivant le ***nom et le prénom*** de la `Personne.` Rajouter ce tri dans votre programme et tester.

Exercice 3 (pour les plus rapides): Créons une collection générique…

1. Voici le code correspondant à l'**interface** générique **`Pile`** qui propose les méthodes publiques suivantes :
   - → la méthode **`empiler`** qui empile un élément sur la pile
   - → la méthode **`dépiler`** qui dépile un élément de la pile, et renverra cet élément
   - → la méthode **`sommet`** qui renvoie l'élément au sommet de la pile
   - → la méthode **`getNbElement`** qui renvoie le nombre d'éléments dans la pile (sous forme d'**`Integer`**)
   - → la méthode **`estVide`** qui teste si la pile est vide et renverra un booléen

```
public interface Pile<E> {

public void empiler(E elt); //empile un élément sur la pile
public E depiler(); // dépile un élément de la pile
public E sommet(); // renvoie l'élément au sommet de la pile
public Integer getNbElement(); // renvoie le nombre d'éléments dans la pile
public boolean estVide(); // teste si la pile est vide
}
```

Dans un nouveau projet **`TestGenericite`**, écrire l'interface **`Pile.`** (faire **`New`** → **`Interface`**)

2. Créer ensuite une classe **`PileListe`** qui implémente l'interface **`Pile`**.
Utiliser une liste chaînée (**`LinkedList`**) pour stocker les éléments de la **`Pile`**.
Utiliser les méthodes de la classe **`LinkedList`** pour implémenter rapidement les méthodes de l'interface `Pile`. Vous trouverez en annexe la javadoc de l'interface `List` et de la classe `LinkedList`

3. Ecrire une application **`TestPile`** qui permet d'obtenir le jeu d'essai suivant c-a-d :
- Instancier une variable **`pile`** permettant de stocker une **`PileListe`** d'**`Integer`**
- Initialiser cette `pile` avec 10 valeurs (carré des chiffres de 0 à 9)
- Dépiler et afficher au fur et à mesure les valeurs dépilées.

```
--- 1. pile instanciee---
--- 2. pile initialisee avec les carres---
--- 3. on dépile et on affiche ---
81
64
49
36
25
16
9
4
1
0
```


4. On souhaite maintenant vérifier le contenu de la pile après initialisation.
Pour cela, on souhaite effectuer un affichage de la pile en utilisant une boucle **`foreach`**.
   4.1 Quelle ligne de code faut-il rajouter dans le fichier **`TestPile`** pour obtenir l'affichage
   4.2 Doit-on apporter des modifications dans la classe générique **`PileListe`** ? si oui lesquelles ?
Le jeu d'essai précédent devient alors :

```
-------------------------------------------
--- 1. pile d'Integer instanciee---
--- 2. pile initialisee avec les carres---
--- Affichage du contenu de la Pile avec une boucle forEach---
0
1
4
9
16
25
36
49
64
81
--- 3. on dépile et on affiche ---
81
64
49
36
25
16
9
4
1
0
-------------------------------------------
```

5. Implémenter le même jeu d'essai, mais cette fois-ci avec une pile de **`String`**

```
--- 1. pile de String instanciee---
--- 2. pile initialisee avec 10 valeurs---
--- Affichage du contenu de la Pile avec une boucle forEach---
a0
a1
a2
a3
a4
a5
a6
a7
a8
a9
--- 3. on dépile et on affiche ---
a9
a8
a7
a6
a5
a4
a3
a2
a1
a0
-------------------------------------------
```

Exercice 4 (pour les plus courageux et les plus curieux…) : Manipulation des jokers avec la classe *`Paire`* vue en cours

Afin de manipuler le notion de ***Joker***, implémenter et tester la classe `Paire` présentée dans le cours n°5 ainsi que les exemples associés à cette classe, et effectuer des tests…
(transparent n°17, n°21, n°42, n°44 à 48)

Annexe

Constructor Summary

LinkedList()	Constructs an empty list.
LinkedList(Collection<? extends E> c)	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Method Summary

public class **LinkedList<E>**
extends AbstractSequentialList<E>
implements List<E>, Queue<E>, Cloneable, Serializable

boolean	**add**(E o) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**addFirst**(E o) Inserts the given element at the beginning of this list.
void	**addLast**(E o) Appends the given element to the end of this list.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this LinkedList.
boolean	**contains**(Object o) Returns true if this list contains the specified element.
E	**element**() Retrieves, but does not remove, the head (first element) of this list.
E	**get**(int index) Returns the element at the specified position in this list. IndexOutOfBoundsException - if the specified index is out of range (index < 0 || index >= size()).
E	**getFirst**() Returns the first element in this list.
E	**getLast**() Returns the last element in this list. NoSuchElementException - if this list is empty.
int	**indexOf**(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
int	**lastIndexOf**(Object o) Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
ListIterator<E>	**listIterator**(int index) Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
boolean	**offer**(E o) Adds the specified element as the tail (last element) of this list.
E	**peek**() Retrieves, but does not remove, the head (first element) of this list.
E	**poll**() Retrieves and removes the head (first element) of this list.
E	**remove**() Retrieves and removes the head (first element) of this list. IndexOutOfBoundsException - if the specified index is out of range (index < 0 || index >= size()).
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(Object o) Removes the first occurrence of the specified element in this list.
E	**removeFirst**() Removes and returns the first element from this list.
E	**removeLast**() Removes and returns the last element from this list. NoSuchElementException - if this list is empty.
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in the correct order.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

java.util
Interface List<E>

public interface **List<E>**
extends Collection<E>

Method Summary

boolean	**add**(E o) Appends the specified element to the end of this list (optional operation).
void	**add**(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	**clear**() Removes all of the elements from this list (optional operation).
boolean	**contains**(Object o) Returns true if this list contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this list contains all of the elements of the specified collection.
boolean	**equals**(Object o) Compares the specified object with this list for equality.
E	**get**(int index) Returns the element at the specified position in this list.
int	**hashCode**() Returns the hash code value for this list.
int	**indexOf**(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
boolean	**isEmpty**() Returns true if this list contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(Object o) Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.
ListIterator<E>	**listIterator**() Returns a list iterator of the elements in this list (in proper sequence).
ListIterator<E>	**listIterator**(int index) Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.
E	**remove**(int index) Removes the element at the specified position in this list (optional operation).
boolean	**remove**(Object o) Removes the first occurrence in this list of the specified element (optional operation).
boolean	**removeAll**(Collection<?> c) Removes from this list all the elements that are contained in the specified collection (optional operation).
boolean	**retainAll**(Collection<?> c) Retains only the elements in this list that are contained in the specified collection (optional operation).
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).
int	**size**() Returns the number of elements in this list.
List<E>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array.

Correction TP JAVA n°4: Interfaces, Exceptions, Polymorphisme

Exercice 2 : Cabinet Médical

```
package com.iut.cabinet.essai;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import com.iut.cabinet.metier.*;
import com.iut.cabinet.util.DateUtil;

public class EssaiCabMed_v3 {


      public static void main(String[] args) {
            new EssaiCabMed_v3 ();
      }

      EssaiCabMed_v3 ()
      {

      // Instanciation de la liste de personnes
      // qui pourra accueillir auusi bien des Patient que des Professionnels
      //Collection<Personne> maListe = new ArrayList<Personne>();
      ArrayList<Personne> maListe = new ArrayList<Personne>();
//ArrayList sinon sort ne marche pas, à priori il lui faut une classe concrète …

      ////////////////////////////
      // Ajout des patients
      ////////////////////////////
      // Instanciation des patients
            Patient patient1=null;
            try {
                  patient1 = new  Patient(1,"DUPONT","Julie",
                  DateUtil.toDate("21/05/1960",DateUtil.FRENCH_DEFAUT),
                  false,"0555434355","0606060606","julie.dupont@tralala.fr",
            new Adresse("15","avenue Jean Jaurès",null,null,"87000","Limoges",
"France"),
                  null, "260058700112367","MARTIN Paul");
            } catch (CabinetMedicalException e) {
                  System.out.println(e.getMessage());
            }

            Patient patient2=null;
            try {
                  patient2 = new Patient(2,"DUPONT","Toto",
                  DateUtil.toDate("25/12/1991",DateUtil.FRENCH_DEFAUT),
                  true,"0555430000","0605040302","toto.dupont@etu.unilim.fr",
                  new Adresse("185","avenue Albert Thomas",null,"Résidence La Borie",
                  "87065","Limoges","France"),
                  patient1,"260058700112367","MARTIN Paul");
            } catch (CabinetMedicalException e) {
                  System.out.println(e.getMessage());
            }

            // Ajout des patients dans la liste...
            if (patient1!= null) maListe.add(patient1);
            if (patient2!= null)maListe.add(patient2);

            ////////////////////////////
            // Ajout des professionnels
            ////////////////////////////
            // Instanciation des professionnels
            Professionnel pro1=null;
```



```
            try {
                  pro1 = new Professionnel(3,"LEDOC","Paul",
                  DateUtil.toDate("10/07/1976",DateUtil.FRENCH_DEFAUT),
                  true,"0555434343","0612345678","paul.ledoc@lesmedecins.fr",
                   new Adresse("3","rue de Limoges",null,null,"87170","Isle","France"),
                   null, "871255358","generaliste");
            } catch (CabinetMedicalException e) {
                  System.out.println(e.getMessage());
            }

            Professionnel pro2=null;
            try {
                  pro2 = new Professionnel(4,"CHILDREN","Rose",
                  DateUtil.toDate("16/02/1970",DateUtil.FRENCH_DEFAUT),
                  true,"0555434343","0687654321","rose.children@lesmedecins.fr",
                  new Adresse("3","rue de Limoges",null,null,"87170","Isle","France"),
                  null,"312444555","pediatrie");
            } catch (CabinetMedicalException e) {
                  System.out.println(e.getMessage());
            }

            // Ajout des professionels dans la liste...
            if (pro1!= null) maListe.add(pro1);
            if (pro2!= null)maListe.add(pro2);

            ////////////////////////////////////////
            // Affichage du contenu de la collection
            ////////////////////////////////////////
            System.out.println("--- Contenu de maListe de Personnes  ---");
             for(Personne unePersonne : maListe)
                  {System.out.println(unePersonne.affichageSimplifie());
                  System.out.println("-----------------------------------------------
");
                  }

            ////////////////////////////////////////////////
            // Tri suivant le nom et Affichage après tri ...
            ////////////////////////////////////////
            ComparateurNom  compareNom = new ComparateurNom();
            Collections.sort(maListe,compareNom);
//Remarque : il faut que maListe soit ArrayList (concrete) et non
Collection(abstraite)
// pour pouvoir être appelée par sort ...

System.out.println("--- Affichage simplifié de la collection après tri suivant le nom
");
             for(Personne unePersonne : maListe)
                  {System.out.println(unePersonne.affichageSimplifie());
                  System.out.println("-----------------------------------------------
");
                  }


             ////////////////////////////////////////////
            // Tri suivant le code Postal
            ////////////////////////////////////////
            ComparateurCodePostal  compareCodePostal  = new ComparateurCodePostal ();
            Collections.sort(maListe,compareCodePostal );

System.out.println("--- Affichage simplifié de la collection après tri suivant le code
postal");
            for(Personne unePersonne : maListe)
                  {System.out.println(unePersonne.affichageSimplifie());
                  System.out.println("-----------------------------------------------
");
                  }
```

```
            ////////////////////////////////////////////
            // Tri suivant l'age et Affichage après tri ...
            ////////////////////////////////////////
            ComparateurAge  compareAge = new ComparateurAge();
            Collections.sort(maListe,compareAge);

System.out.println("--- Affichage simplifié de la collection après tri suivant la date
de naisance ---");
            for(Personne unePersonne : maListe)
                  {System.out.println(unePersonne.affichageSimplifie());
            System.out.println("-----------------------------------------------");
                  }


             ////////////////////////////////////
            // Tri suivant Ordre Alpha ...
            ////////////////////////////////////
            ComparateurOrdreAlpha  compareOrdreAlpha = new ComparateurOrdreAlpha();
            Collections.sort(maListe,compareOrdreAlpha);

System.out.println("--- Affichage simplifié de la collection après tri suivant la date
de naisance ---");
            for(Personne unePersonne : maListe)
                  {System.out.println(unePersonne.affichageSimplifie());
            System.out.println("-----------------------------------------------");
                  }

      }

      ////////////////////////////////////////////////
      /// Implémentation des comparateurs ....
      ////////////////////////////////////////////////
      class ComparateurNom implements Comparator<Personne>
      {
            public int compare(Personne pers1, Personne pers2)
            {
                  return pers1.getNom().compareTo(pers2.getNom());
            }
      }

      class ComparateurAge implements Comparator<Personne>
      {
            public int compare(Personne pers1, Personne pers2)
            {
                  return -
pers1.getDateNaissance().compareTo(pers2.getDateNaissance());
                  // - moins pour avoir l'ordre inverse des dates de naissance
            }
      }

      class ComparateurCodePostal implements Comparator<Personne>
      {
            public int compare(Personne pers1, Personne pers2)
            {
                  return
pers1.getAdresse().getCodePostal().compareTo(pers2.getAdresse().getCodePostal());
            }
      }


      class ComparateurOrdreAlpha implements Comparator<Personne>
      {
            public int compare(Personne pers1, Personne pers2)
            {
                  int res= pers1.getNom().compareTo(pers2.getNom());
                  if (res!=0) return res;
```



```
                  else // si non égaux, on va voir prénom....
                   {
                    return pers1.getPrenom().compareTo(pers2.getPrenom());
                   }
            }
      }


}
```

Exercice 3 (pour les plus rapides): Créons une collection générique…

```
public interface Pile<E> {

      void empiler(E elt); // empile un élément sur la pile
      E depiler(); // dépile un élement de la pile si celle-ci
      E sommet(); // renvoie l'élement au sommet de la pile
      Integer getNbElement(); // renvoie le nombre d'éléments dans la pile
      boolean estVide(); // teste si la pile est vide
}
```

La classe ***`PileListe<E>`*** écrite de manière la plus concise => 1 ligne par méthode ! ! !
Dans le cas où la pile est vide et on essaye de dépiler, on se contentera de l'exception levée par les méthodes de `LinkedList NoSuchException`...

```
import java.util.Iterator;
import java.util.LinkedList;

public class PileListe<E> implements Pile<E>, Iterable<E>{

      private LinkedList<E> contenu;

       // Ne pas oublier le constructeur !!!
      public PileListe()
            {contenu = new LinkedList<E>();}

      // empile un élément sur la pile
        public void empiler(E elt){
             contenu.add(elt);
        }
      // dépile un élement de la pile
        public E depiler(){
         return contenu.removeLast();// si liste vide renvoie une
NoSuchElementException

        }

      // renvoie l'élement au sommet de la pile
        public E sommet(){
            return contenu.getLast();
      // si liste vide renvoie une NoSuchElementException qui est une RunTimeException
      // et que l'on pourra attraper lors de l'appel de dépiler
        }

        public Integer getNbElement(){
              return contenu.size();
        }


        // teste si la pile est vide
        public boolean estVide(){
         return contenu.isEmpty();
        }
      public Iterator<E> iterator() {
            return contenu.iterator();
      }

}
```

Remarque :… si on souhaite traiter « les cas à part » dans le code avec des if , on peut écrire…

```
// dépiler un élement de la pile
        public E depiler(){
             if (contenu.size() == 0)
            return null; //on aurait pu choisir de lancer une exception...
          else
            {
            return contenu.removeLast();
            }
        }

// renvoie l'élement au sommet de la pile
        public E sommet(){

            if (contenu.size() == 0)
           return null; //new EmptyStackException();
          else
            {
            E elt = contenu.getLast();
            return elt;
            }
      }

// teste si la pile est vide
public boolean estVide(){
          if (contenu.size() == 0) return true;
                else return false;
      }
```

---

3. Programme test qui permet d'obtenir le jeu d'essai suivant c-a-d :

```
public class TestPile {
       public static void main(String[] args){

      //Instanciation de la variable pile
      PileListe<Integer> pile = new PileListe<Integer>();
      System.out.println("--- 1. pile instanciee---");

       //Initialisation de la pile avec 10 valeurs
       for(int i=0; i<10; i++)
         {pile.empiler(new Integer(i*i));}
       System.out.println("--- 2. pile initialisee avec les carres---");

      //Dépiler et Afficher ...
       System.out.println("--- 3. on dépile et on affiche ---");
           while(! pile.estVide())
                 System.out.println(pile.depiler());
            }
      }
```

```
-------------------------------------------------------------------------------
Programme TestPile complet avec les 2 jeux d'essais : Pile d'Integer et Pile de String
-------------------------------------------------------------------------------

public class TestPile {


          public static void main(String[] args){
             System.out.println("-----------------------------------------");
            //Instanciation de la variable pile
              PileListe<Integer> pile = new PileListe<Integer>();
              System.out.println("--- 1. pile d'Integer instanciee---");

              //Initialisation de la pile avec 10 valeurs
              for(int i=0; i<10; i++)
                  {pile.empiler(new Integer(i*i));}
              System.out.println("--- 2. pile initialisee avec les carres---");

             // Affichage avec boucle foreach
System.out.println("--- Affichage du contenu de la Pile avec une boucle forEach---");
              for(Integer elt : pile)
                        {System.out.println(elt);}

            //Dépiler et Afficher ...
              System.out.println("--- 3. on dépile et on affiche ---");
              while(! pile.estVide())
                  System.out.println(pile.depiler());

              System.out.println("-----------------------------------------");


            //Instanciation de la variable pile
              PileListe<String> pile2 = new PileListe<String>();
              System.out.println("--- 1. pile de String instanciee---");

            //Initialisation de la pile avec 10 valeurs
              for(int i=0; i<10; i++)
                  pile2.empiler(new String("a"+i));
              System.out.println("--- 2. pile initialisee avec 10 valeurs---");

              // Affichage avec boucle foreach
System.out.println("--- Affichage du contenu de la Pile avec une boucle forEach---");
               for(String elt : pile2)
                         {System.out.println(elt);}

            //Dépiler et Afficher ...
             System.out.println("--- 3. on dépile et on affiche ---");
               while(! pile2.estVide())
                  System.out.println(pile2.depiler());

              System.out.println("-----------------------------------------");

          }


}
```



```
-------------------------------------------------------------------------------
EXPLICATIONS …
-------------------------------------------------------------------------------
```

4. On souhaite maintenant vérifier le contenu de la pile après initialisation.
4.1 Quelle ligne de code faut-il rajouter dans le fichier **`TestPile`** pour obtenir l'affichage avec une boucle **`foreach`** :

```
            for(Integer elt : pile)
                  {System.out.println(elt);}
```

4.2 Doit-on apporter des modifications dans le classe **`PileListe`** ? si oui lesquelles ?
Pour pouvoir écrire une boucle `foreach` il faut que la classe utilisée dans la boucle `foreach` implémente `Iterable` … (méthode `Iterator<T> iterator();`),.

```
public class PileListe<E> implements Pile<E>, Iterable<E>{

      public Iterator<E> iterator() {
            // Il faut renvoyer un Iterator
            // ou une sous-classe de Iterator ce qui reviendrait dans ce cas à créer
      son propre itérateur …}
}
```

Une classe qui implémente `Iterable` doit founir un `Iterator`. **Il faut donc fournir un `Iterator` :**

- **Solution n°1 : Soit en créant son propre itérateur (implémentation de la classe Iterator)**
- **Solution n°2 : Soit en renvoyant un itérateur déjà existant**

Rappel sur les itérateurs (*voir cours transparent n*37*) :
Un itérateur (objet de la classe `Iterator de java.util)` permet de parcourir les différents éléments d'un conteneur (collection). Un itérateur devra implémenter les 3 méthodes suivantes :

→ **boolean hasNext()** qui renvoie vrai s'il y a un suivant
(ici si on l'implémentait il faudrait aller lire les élements de la LinkedList)
→ **E next()** qui renvoie l'élement courant et décale sur l'élément suivant
→**void remove()** qui retire un élement précédemment envoyé par next()

Est-ce nécessaire de déclarer un nouvel type `monIterator` en implémentant la classe suivante ?
(Solution n°1)
The iterator method should return a new Iterator object each time it's called. The best way to accomplish this is to make the Iterator an inner class, like (`http://forum.java.sun.com/thread.jspa?threadID=590473&messageID=3937060`)

```
class monIterator<E> implements Iterator<E>
{
      int position = 0 ;
      public boolean hasNext() {
            // … à faire : renvoie vrai s'il y a un suivant
            return false;
      }

      public E next() {
            // … à faire :
            // renvoie l'élément courant et décale
            // sur le suivant …
            return …;
      }

      public void remove() {
            // … à faire : retire un élément précédemment envoyé
            // par next…
      }
}
```

Classe dans la classe **`Pile<E>`**
(pas de `private`, ni de `public` car une seule classe `public` par fichier)

```
public class PileListe<E> implements Pile<E>, Iterable<E>{

      public Iterator<E> iterator() {
            return new monIterator<E>();
      }
}
```

… non il n'est pas nécessaire de déclarer une classe `Iterator`

Nous allons **adopter la solution n°2** et modifier directement la classe `PileListe` simplement
En effet, dans la méthode ***`iterator`***, on a seulement besoin d'écrire **`contenu.iterator();`**
car **contenu** est de type **`LinkedList`**, `et` **`LinkedList`** implemente déjà `Iterable` et fournit déjà un `Iterator`, on n'a plus qu'à récuperer cet `Iterator.`

Il faut donc modifier la classe `PileListe`

```
import java.util.Iterator;
public class PileListe<E> implements Pile<E>, Iterable<E>{

      public Iterator<E> iterator() {
            return contenu.iterator();
      }
}
```

Remarque : si temps à la fin, on peut demander un affichage avec un while (en s'aidant des transparents du cours)
Dans ce cas-là, on voit bien la déclaration de l'Iterator typé :

```
Iterator <Integer> it= pile.iterator();
```

Il ne faut pas oublier non plus d'indiquer en haut du fichier `PileTest` le :

A voir si on le rajoute ou pas ?

```
import java.util.Iterator;

// Affichage avec un while
       Iterator <Integer> it = pile.iterator();
       while(it.hasNext()){
            Integer elt = it.next();
            System.out.println(elt);
       }
```