UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE
FACULTÉ D'INFORMATIQUE

Cours

Programmation Orientée Objet 2

Pour

ING 2

Chap 03:

Gestion des collections et Génériticités

MEKAHLIA Fatma Zohra LAKRID
Maître de Conférences Classe B

Laboratoire de Modélisation, Vérification et Evaluation des Performances des systèmes complexes (MOVEP)
Bureau 123

Collections

INTRODUCTION

- On appelle *collections* un ensemble **de classes** et **d'interfaces** fournies par Java et disponibles pour la plupart dans le package **java.util**.

- Parmi ces collections, on trouve les listes (*lists*), les ensembles (*sets*) et les tableaux associatifs (*maps*). Elles forment ce que l'on appelle le **Java Collections Framework**.

- En effet, une collection est un objet servant de conteneur à d'autres objets. Exemples :

– les tableaux,

– les listes et leurs variantes (piles, queues),

– les ensembles,

– les tables associatives,

Introduction

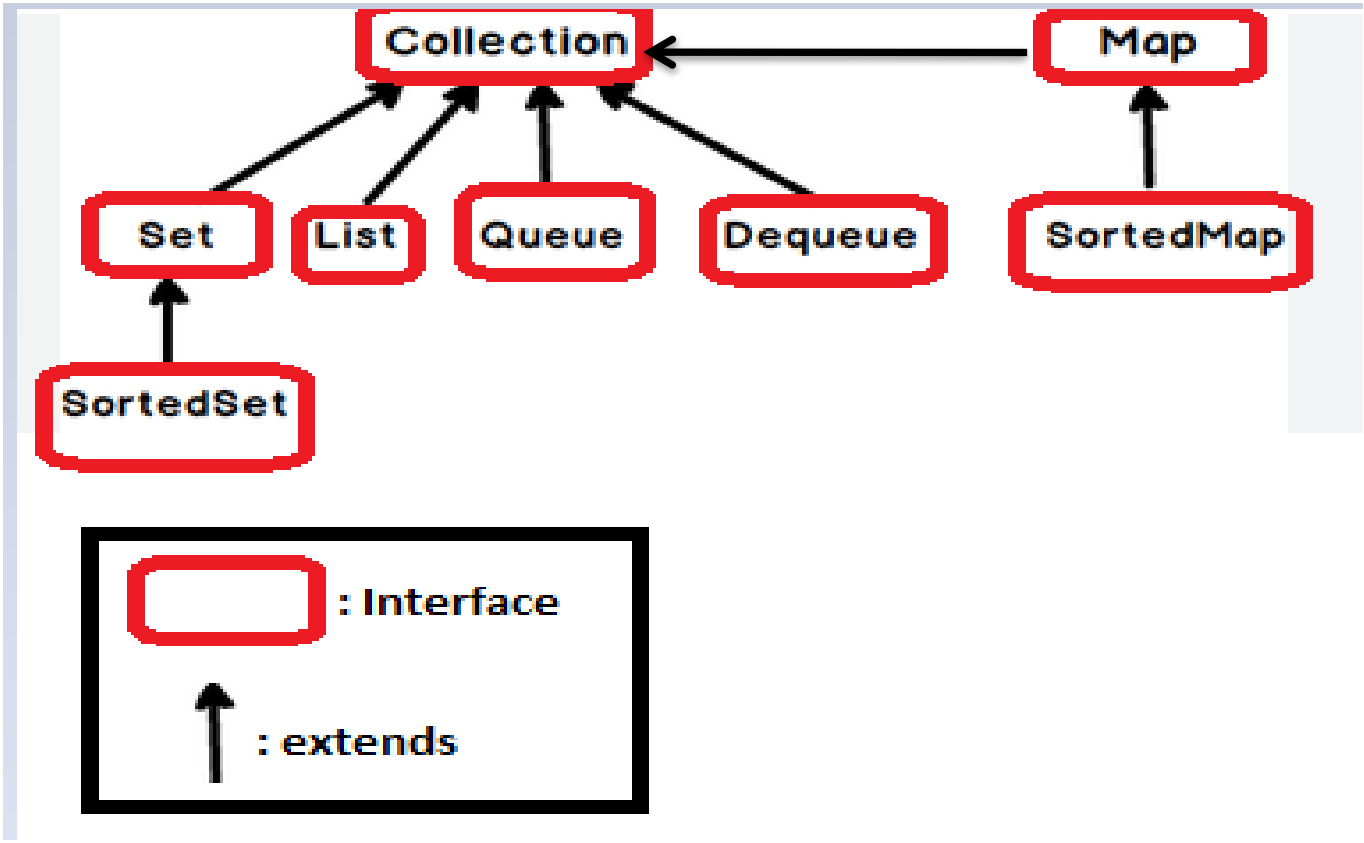
- Chaque genre de collection a ses caractéristiques propres, ses forces et ses faiblesses.
- en commun :

- **mêmes questions** : est-ce qu'elles contiennent des éléments ? combien ?
- **mêmes opérations** : on peut ajouter ou enlever un élément à la structure, on peut vider la structure. On peut aussi parcourir les éléments contenus dans la structure.
- **Mais**, avec des implémentations différentes.

- Le choix de la collection à utiliser dans un cas particulier dépend de ce qu'on veut en faire.

Hiérarchie d'interface

- Toutes ces structures sont organisés sous forme d'une hiérarchie d'interface.



Hiérarchie d'interface

- **Collection:** une interface qui contient des méthodes de base pour parcourir, ajouter, enlever des éléments.
- **Set :** cette interface représente un ensemble, et donc, ce type de collection n'admet aucun doublon.
- **List :** cette interface représente une séquence d'éléments : l'ordre d'ajout ou de retrait des éléments est important (doublons possibles).
- **Queue : file d'attente**, il y a l'élément en tête et les éléments qui suivent. L'ordre d'ajout ou de retrait des éléments est important (doublons possibles).
- **D**ouble-**e**nded**que**ue **:** cette interface ressemble aux files d'attente, mais les éléments importants sont les éléments en tête et en queue.
- **SortedSet :** est la version ordonnée d'un ensemble.

Hiérarchie d'interface

- **Map :** cette interface représente une relation binaire (surjective) : chaque élément est associé à une clé et chaque clé est unique (mais on peut avoir des doublons pour les éléments).
- **SortedMap :** est la version ordonnée d'une relation binaire ou les clés sont ordonnées.

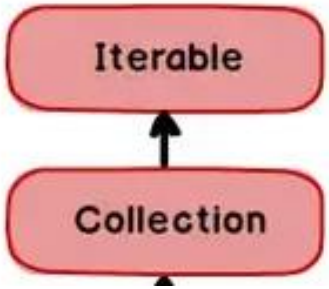
- .....

Ces interfaces sont **génériques**, i.e. on peut leur donner un paramètre pour indiquer qu'on a une collection d'Integer, de String, etc..

- .....

Hiérarchie d'interface

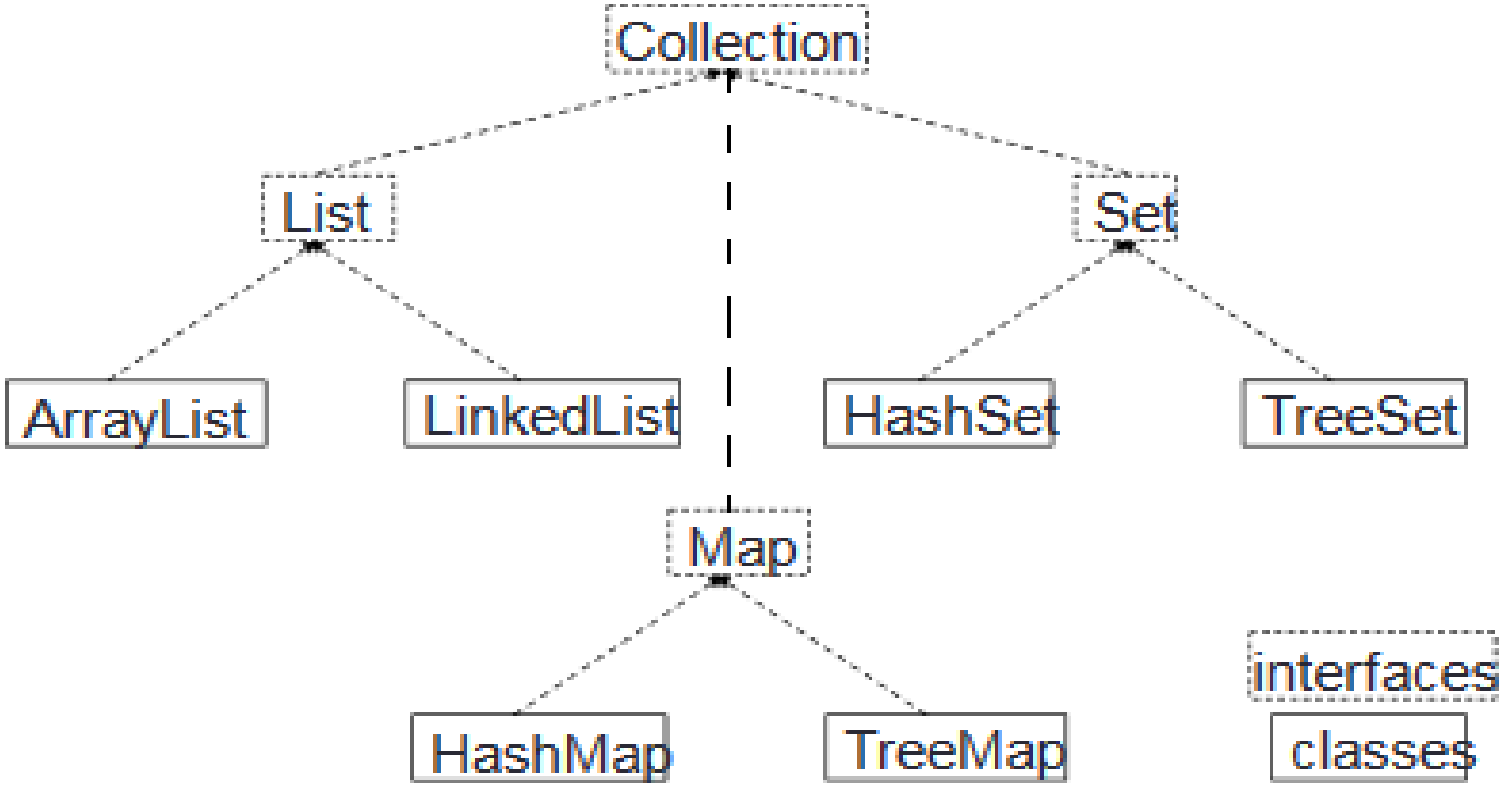
- L'interface **Iterable** (java.lang.Iterable) est l'une des interfaces racine de l'arborescence des collections. **Une classe** qui implémente l'interface **Iterable** peut être itérée avec la boucle for-each.




```
List list = new ArrayList();
list.add("Java"); list.add("Pascal");
list.add("PHP"); list.add("C++");
for(Object obj : list){
System.out.println(obj.toString()); }
```

Implémentation des interfaces collection

- Pour chacune des interfaces, il existe plusieurs implémentations



L'Interface Collection

- L'interface **Collection** définit la notion de collection d'objets d'une façon assez générale.

- **Les opérations sont :**

- Obtenir le nombre d'éléments de la collection,
- Rechercher un objet donné
- Ajouter un objet
- supprimer un objet,
- … etc.

LINTERFACE MAP

- Les collections de type Map, tableau associatif ou dictionnaire en Java, sont définies à partir de la racine Interface Map <K, V>.

- La raison est qu'une telle collection est un ensemble de paires d'objets, chaque paire associant un objet de l'ensemble de départ K à un objet de l'ensemble d'arrivée V ; on parle de paires (clé, valeur)

L'Interface List

- Interface pour des objets qui autorisent des doublons et un accès direct à un élément.

- **Plusieurs implémentations possibles tq:**
  - **ArrayList :** Liste implantée dans un tableau.
  - **LinkedList:** Liste doublement chaînée.

- **Quelques méthodes de ArrayList:**

L'Interface List

- **add(Object element)** permet d'ajouter un élément ;
- **add(int index, Object element)** permet d'ajouter un élément à l'indice index ;
- **get(int index)** retourne l'élément à l'indice demandé.
- **remove(int index)** efface l'elt à l'indice demandé.
- **isEmpty()** renvoie « vrai » si l'objet est vide ;
- **size()** retourne la taille de l'ArrayList;
- **contains(Object element)** retourne « vrai » si l'élément passé en paramètre est dans l'ArrayList.

L'OBJET ARRAYLIST

- Une ArrayList stocke ses éléments dans un tableau dynamique, donc il n'a pas de taille limite, et en plus, ils acceptent n'importe quel type de données ! **null** y compris !

- Vous devez par contre importer la classe **ArrayList**.

```java
package RepertoireDeTelephone;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;

public class MyArrayList {

public static void main(String[] args) {
// TODO Auto-generated method stub
ArrayList al = new ArrayList();
al.add("Bonjour");
al.add(5);
al.add('.');

// afficher ArrayList par une boucle for
 System.out.println("Boucle for");
 for (int i =0;i<al.size(); i++)
 System.out.println(al.get(i));

 // afficher ArrayList par une boucle for avancée
 System.out.println("Boucle for avancée");
 for (Object n : al)
 System.out.println(n);

 //afficher ArrayList par une boucle while+iterator
 System.out.println("Boucle while & iterator");
 Iterator it = al.iterator();
 while (it.hasNext()){
 System.out.println(it.next());
 }}}
```

Problems Javadoc Declaration Console

<terminated> MyArrayList [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (3

```
Boucle for
Bonjour
5
.
Boucle for avancée
Bonjour
5
.
Boucle while & iterator
Bonjour
5
.
```

L'OBJET LINKEDLIST

- Une LinkedList utilise une liste double**ment** chaînée .
- Une liste chaînée est une liste dont chaque élément est relié au suivant par une référence à ce dernier, sa taille n'est pas fixe : on peut ajouter et enlever des éléments selon nos besoins.
- Les LinkedList acceptent tout type d'objet.
- Chaque élément contient une référence sur l'élément suivant sauf pour le dernier : son suivant est en fait **null**.
- Vous devez importer la classe LinkedList .

L'OBJET ARRAYLIST VS LINKEDLIST

- Puisque la recherche dans ArrayList est basée sur l'index de l'élément alors elle est très rapide. La méthode get(index) a une complexité de O(1), mais la suppression est coûteuse parce que vous devez décaler tous les éléments. Dans le cas de LinkedList, elle ne possède pas un accès direct aux éléments, vous devez parcourir toute la liste pour récupérer un élément, sa complexité est égale à O(n).

L'OBJET ARRAYLIST VS LINKEDLIST

- Les insertions sont faciles dans LinkedList par rapport à ArrayList parce qu'il n y a aucun risque lors de la redimensionnement et l'ajout de l'élément à LinkedList et sa complexité est égale à O(1), tandis que ArrayList décale tous les éléments avec une complexité de O(n) dans le pire des cas.
- La suppression est comme l'insertion, meilleure dans LikedList que dans ArrayList.
- LinkedList a plus de mémoire que ArrayList parce que dans ArrayList chaque index est relié avec l'objet actuel, mais dans le cas de LinkedList, chaque nœud est relié avec l'élément et l'adresse du nœud suivant et précédent.

L'OBJET ARRAYLIST VS LINKEDLIST

- On utilise le plus souvent **ArrayList** si l'ajout et l'accès sont direct (indicé).

- Mais, **LinkedList** est utile s'il y a beaucoup d'opérations d'insertions / suppressions afin d'éviter les décalages.

L'interface Set

- Eléments non dupliqués

- **Plusieurs implémentations possibles tq:**
  - **HashSet :** table de hashage (très utilisée).
  - **TreeSet :** arbre binaire de recherche.

- **Quelques méthodes de HashSet:**

L'INTERFACE SET

- **add(Object element)** ajoute un élément.
- **contains(Object element)** retourne « vrai » si l'objet contient element.
- **isEmpty()** retourne « vrai » si l'objet est vide.
- **iterator()** renvoie un objet de type Iterator.
- **remove(Object element)** retire l'objet element de la collection ;
- **toArray()** retourne un tableau d'Object.

L'OBJET HASHSET

- Un Set est une collection qui n'accepte pas les doublons. Elle n'accepte qu'une seule fois la valeur **null**, car deux fois cette valeur est considérée comme un doublon.
- On peut dire que cet objet n'a que des éléments différents.
- Certains Set sont plus restrictifs que d'autres, n'acceptent pas null ou un certain type d'objet.
- On peut parcourir ce type de collection avec un objet **Iterator** où, cet objet peut retourner un tableau d'**Object**.

Généricité

GÉNÉRICITÉ

- **Exemple:**

```
public class CompteBancaire  {
      private String nomProp;
      private double solde;
      private char devise;
}
// Constructeur

public CompteBancaire(String nomProp, double solde, char devise){
this. nomProp= nomProp; this.solde=solde; this.devise=devise;
}
```

Généricité

// Méthodes

```
public String getNomProp(){ return nomProp;}
public double getSolde(){ return Solde;}
public char getDevise(){ return devise;}


public void addSolde(double solde){ this.solde += solde; }
public void removeSolde(double solde){ this.solde -= solde; }
Public void showCompte(){
System.out.println (" votre solde est "+solde+ " "  + devise);
}
```

GÉNÉRICITÉ

```
Public class Main{
public static void main (String[] args){
CompteBancaire monCompte = new CompteBancaire(“Douaa ”, 100, ‘€’);
CompteBancaire autreCompte = new CompteBancaire(“Douaa”, 100,
  “DA”);
}
```

- **Attention**

. Dans le deuxième objet j’aurai une erreur car le troisième paramètre demande un caractère et non pas une chaine .

- Si je change le type de devise de char en String !!
- ça va pas fonctionné dans le premier objet car le troisième paramètre demande char au lieu String!!
- On se retrouve dans **une boucle infinie d’erreur !!**

GÉNÉRICITÉ D'OBJET

Solution: on doit créer un objet générique.

- On lieu de se limiter a un seul type (char) pour l'attribut devise on le rendre générique.

```
public class CompteBancaire <T>  { / * les types génériques sont
   représentés avec un T par convention */
     private String nomProp;
     private double solde;
     private T devise;   }

public CompteBancaire(String nomProp, double solde, T devise){
   this. nomProp= nomProp; this.solde=solde; this.devise=devise;
   }

public T getDevise(){ return devise;}
```

GÉNÉRICITÉ D'OBJET

Maintenant on doit passer le type de T quand on créer des objets dans le main !!

- Par substitution de T on créer un système dynamique.

Public **class Main**{

public static void main (String[] args){

CompteBancaire **<Character>** cpmt1=new CompteBancaire **< >** ("Douaa", 100, '€');

CompteBancaire **<String>** cmpt2=new CompteBancaire **< >** ("Douaa", 100, 'DA');

}

Généricité et héritage

GÉNÉRICITÉ ET HÉRITAGE

Soit la classe paramétrée (générique) Paire suivante:

```
public class Paire <T> {   // classe générique
   T premier ;
   T second ;
   public Paire (T a, T b){  premier=a; second = b; }
   public T getPremier (){ return premier ;   }
   public T getSecond (){ return second ;  }
}
```

GÉNÉRICITÉ ET HÉRITAGE

Soit la classe exécutable TableauG:

```
class TableauG {   ...
public static Paire<T> getDeux(T[ ] tab,int i, int j){
        return new Paire<T> (tab[i],tab[j]);
    }
...
} // fin classe TableauG
```

GÉNÉRICITÉ ET HÉRITAGE

Et soit la classe Personne suivante:

```
class Personne {
   private String nom, prénom;
   public Personne(String nom, String prénom) {
            this.nom=nom; this.prénom=prénom; }
   public String getNom() { return nom; }
   public String getPrénom() { return prénom; }

// méthode .....
}
```

GÉNÉRICITÉ ET HÉRITAGE

Et soit la classe fille **Elève** de Personne:

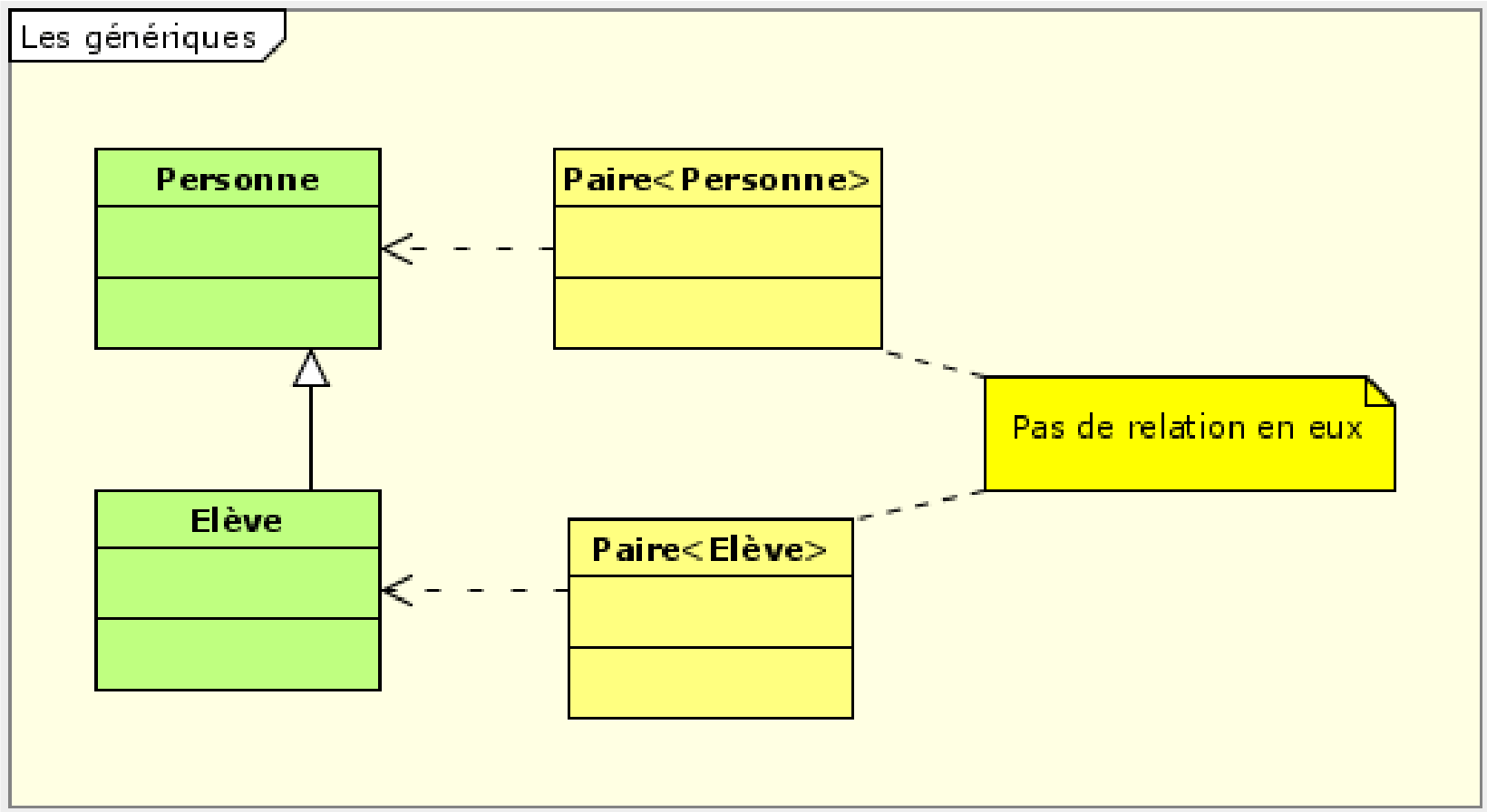
```
class Elève extends Personne {
  private double[] notes = new double[10];
  private int nombreNote = 0;
  public Elève(String nom, String prénom) {
            super(nom, prénom); }
…
}
```

Généricité et héritage

- La question qui se pose:

"Est-ce que Paire<Elève> est une sous-classe de Paire<Personne>" ?.

GÉNÉRICITÉ ET HÉRITAGE



GÉNÉRICITÉ ET HÉRITAGE

- La réponse est « **Non** »

Par exemple, **le code suivant ne sera pas compilé** :

```
Elève[] élèves = ... ;
Paire <Personne>personne =
                    TableauG.getdeux(élèves,1,2);
```

Généricité et héritage

En général, il n'existe pas de relation entre Paire<S> et Paire<T>,

quels que soient les éléments auxquels S et T sont reliés (même s'il y a de l'héritage entre S et T)

Généricité et interface

GÉNÉRICITÉ ET INTERFACE

- Exemple d'une interface de pile d'entiers qui est dynamique mais elle est limité pour représenter que des entiers.
- **NB:** une pile sert à stocker des valeurs de même type.

```
public interface IntStack {
// Ajoute l'élément au sommet de la pile.
public void addFirst(int value);
// déplier et empliler l'élément au sommet de la pile.
// Lève IllegalStateException si la pile est vide.
public int removeFirst();
// Retourne vrai ssi la pile est vide.
public boolean isEmpty();
}
```

Généricité et interface

- les classes qui implémentent l'interface IntStack ne peuvent représenter que des piles d'entiers.

- **Comment faire pour représenter des piles d'un autre type ????**

- **Solution:** écrire autant d'interfaces et de classes qu'il existe de types (spécialisation).

Généricité et interface

- **interface IntStack {**

void addFirst(int value); … }

- **interface BooleanStack {**

void addFirst(boolean value); … }

- **interface StringStack {**

void addFirst(String value); … }

duplication de code !

Généricité et interface

Solution 2: faire des piles de Object

- **interface Stack {**

```
    public void addFirst(Object value);
    public Object removeFirst();
    public boolean isEmpty();
}
```

- possible mais nécessitera des grande quantité de transtypages (*casts)* explicites !!

GÉNÉRICITÉ ET INTERFACE

- En raison des problèmes posés par la spécialisation et la solution basée sur le type Object, la notion de **généricité** (genericity), aussi appelée **polymorphisme paramétrique** (parametric polymorphism) a été introduite dans la version 5 de Java.

- Au moyen de la généricité, il est possible de définir des piles génériques, c'est-à-dire capables de contenir des éléments de différents type.

Généricité et interface

- Une interface peut se définir ainsi :
- **interface Stack<E> {**

```
 public void addFirst(E value);
public E removeFirst();
public boolean isEmpty();
}
```

- E est un **paramètre de type** de cette interface. Il s'agit d'une variable (de type !) représentant le type des éléments de la pile.

GÉNÉRICITÉ ET INTERFACE

Cas d'utilisation:

- **public class LinkedStack <E> implements Stack <E> { … }**

- Stack <Character> s = **new LinkedStack <Character> ();**

Types de base

- Java possède 8 types dits *de base qui ne sont pas* des objets (boolean, byte, short, int, long, char, float, double). Malheureusement, les types de base ne peuvent pas être utilisés comme paramètres de type d'un type générique. Dès lors, le code suivant est erroné :

- Stack<int> s = …; // interdit !
- s.add(2);
- s.add(3);
- Que faire si l'on désire créer une pile d'entiers ?

EMBALLAGE

- **Solution** : stocker chaque valeur de type int dans un objet de type **java.lang.Integer**, et créer une pile de valeurs de ce type. L'exemple devient :
- Stack<Integer> s = …;

  s.add(**new Integer(2));**

  s.add(**new Integer(3));**
- On dit alors que les entiers ont été **emballés (*wrapped ou*** *boxed) dans des objets de type Integer*.
- Le paquetage java.lang contient une class d'emballage par type de base (Boolean pour boolean, Character pour char, Double pour double, etc.)

L'OBJET HASHMAP: EXERCICE

- on utilise **HashMap** pour simuler un répertoire dans lequel le numéro de téléphone est la clé et le nom du propriétaire est la valeur. Les clés ne sont jamais dupliquées.

- 2eme version en TP: maintenant on vous demande d'insérer deux clés identiques. Expliquez le résultat.

```java
package RepertoireDeTelephone;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class Main {

public static void main(String[] args) {
// TODO Auto-generated method stub
 //Créer et remplir  HashMap

 Map <String, String> repPhonne = new HashMap<>();
 repPhonne.put("0556235689", "Mohamed");
 repPhonne.put("0556235690", "Amine");
 repPhonne.put("0556235691", "Mohamed");

 // afficher HashMap par une boucle for
 System.out.println("Boucle for");
 for (Map.Entry mp: repPhonne.entrySet())
 System.out.println("clé est:" +mp.getKey()+"| valeur est:"+mp.getValue());

 //afficher HashMap par une boucle while+iterator
 System.out.println("Boucle while & iterator");
 Iterator it = repPhonne.entrySet().iterator();
 while (it.hasNext()){
 Map.Entry mp =(Map.Entry)it.next();
 System.out.println("clé est:" +mp.getKey()+"| valeur est:"+mp.getValue());
 }}}
```

Problems Javadoc Declaration Console

<terminated> Main (11) [Java Application] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (3 mars 2024 à 10:24:28)

```
Boucle for
clé est:0556235689| valeur est:Mohamed
clé est:0556235690| valeur est:Amine
clé est:0556235691| valeur est:Mohamed
Boucle while & iterator
clé est:0556235689| valeur est:Mohamed
clé est:0556235690| valeur est:Amine
clé est:0556235691| valeur est:Mohamed
```

L'OBJET HASHSET

```
10    public static void main(String[] args) {
11        HashSet<String> hset = new HashSet<String>();
12        hset.add("h1");
13        hset.add("h2");
14        hset.add("h3");
15
16        System.out.println("Boucle for avancée");
17        for(String s : hset)
18          System.out.println(s);
19
20        System.out.println("Boucle While+Iterator");
21        Iterator it = hset.iterator();
22        while(it.hasNext())
23        System.out.println(it.next());
24
25        System.out.println("Boucle While+Ennumération");
26        // récupérer l'objet Ennumeration
27        Enumeration enumeration = Collections.enumeration(hset);
28        // lire à travers les éléments de HashSet
29        while(enumeration.hasMoreElements())
30           System.out.println(enumeration.nextElement());}}
```

L'OBJET HASHSET

```
Boucle for avancée
h1
h2
h3
Boucle While+Iterator
h1
h2
h3
Boucle While+Ennumération
h1
h2
h3
```