



Chapitre 2

Les collections

Classes génériques permettant de gérer des collections d'objets

Les collections

1. Type générique

Sans Généricité

```
public class ObjectBox {  
    private Object object;  
    public void set(Object object) { this.object = object; }  
    public Object get() { return object; }  
}
```

Utilisation

```
ObjectBox objectBox = new ObjectBox();
```

```
objectBox.set(4);
```

```
Integer integer = (Integer) objectBox.get(); // Casting obligatoire
```

```
ObjectBox objectBox2 = new ObjectBox();
```

```
objectBox2.set("John");
```

```
String string = (String) objectBox2.get(); // Casting obligatoire
```

Qu'est-ce que la généricité?

Un type générique est une classe ou une interface paramétrisée
(qui utilisent des **typages en paramètres**)

Avantages

Vérification plus forte à la compilation

Elimination de casting

Qu'est-ce que la généricité?

Exemple:

```
public class Box<T> {  
    private T t;  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}
```

```
Box<Integer> integerBox = new Box<>();
```

```
integerBox.set("John");
```

```
integerBox.set(3);
```

```
Integer integer = integerBox.get();
```

```
Box<String> stringBox = new Box<>();
```

```
stringBox.set(3);
```

```
stringBox.set("John");
```

```
String string = stringBox.get();
```

Instanciation

Compilation pas OK

Pas de casting

Compilation pas OK

Pas de casting

Types de paramètre

Les types de paramètre les plus courants

- **E** - Element
- **K** - Key
- **N** - Number
- **T** - Type
- **V** - Value
- **S,U,V** etc. – 2^{ème}, 3^{ème}, 4^{ème} types

Paramètres multiples

Paramètres multiples

```
public interface KeyValue<K, V> {  
    public K getKey();  
    public V getValue();  
}
```

```
public class Pair<K, V> implements KeyValue<K, V> {  
    private K key;  
    private V value;  
    ...                // Constructeur et méthodes
```

```
Pair<String, Integer> pair1 = new Pair<>("Even", 8);  
Pair<String, String> pair2 = new Pair<>("hello", "world");
```

Les collections

1. Type générique

2. ArrayList

ArrayList

```
public class Personne {  
    private String nom;  
    private GregorianCalendar dateNaiss;  
  
    public Personne (...) { ... }  
  
    ...      Getters et setters  
  
    public int age( ) { ... }  
  
    public String toString( ) {  
        return "La personne " + nom + " agée de " + age() + " ans";  
    }  
}
```

ArrayList

```
public class Etudiant extends Personne {  
    private int annee;  
    private String section;  
  
    public Etudiant (...)  
    { ... }  
  
    public String toString( ) {  
        return super.toString( ) + " est inscrit en " + annee + "e " + section;  
    }  
}
```

```
import java.util.*;
```

```
public class Principal {
```

```
    public static void main(String[ ] args)
```

```
{ ArrayList <Personne> personnes = new ArrayList < > ( );
```

```
    personnes.add(new Personne ("Anne Petit", 12));
```

```
    ...
```

```
    personnes.add(new Etudiant ("Pol Louis", 19,2,"info"));
```

```
    for (int i = 1; i<= personnes.size( ); i++)
```

```
        System.out.println("Elément " + i + " : " + personnes.get(i-1) );
```

```
    for (int i = 1; i<= personnes.size( ); i++)
```

```
        System.out.println("age de la personne numéro " + i + " : "
```

```
            + personnes.get(i-1).age( ) );
```

Retourne un objet **Personne**

Typage

ArrayList

```
Personne personne = new Personne ("Pol Castor", 50);  
personnes.set(2, personne);
```

```
if (personnes.contains(personne))  
    System.out.println("La liste contient: " + personne);
```

```
personnes.remove(1);
```

```
personnes.clear( );
```

```
if (personnes.isEmpty( ))  
    System.out.println("Liste vide");
```

Les collections

1. Type générique
2. ArrayList
3. Collections d'objets et casting

```
ArrayList<Object> liste= new ArrayList<> ( );  
liste.add(new Personne ("Anne Petit", 12));
```

...

```
for (int i = 1; i<= liste.size( ); i++)  
    System.out.println("Elément " + i + " : " +liste.get(i-1) );
```

Polymorphisme:
toString() de Personne

```
for (int i = 1; i<= liste.size( ); i++)  
    System.out.println("age de la personne numéro " + i + " : " +  
        liste.get(i-1).age( ) );
```

De type Object

```
System.out.println("age de la personne numéro " + i + " : " +  
    ( (Personne) (liste.get(i-1))).age( ));
```

CASTING

Les collections

1. Type générique
2. ArrayList
3. Collections d'objets et casting
4. Boucler sur une collection
 - Iterateur

Iterator Pattern

Objectif du pattern *itérateur*

Fournir un moyen d'accéder séquentiellement à une collection d'objets sans révéler son implémentation

Boucler sur tous les éléments de la collection sans connaître son implémentation

Exemples d'implémentation de collection

- *Tableau d'objets*
- *Liste chaînée*
- *ArrayList*
- *HashMap*

Iterator Pattern

⇒ Il faut pouvoir

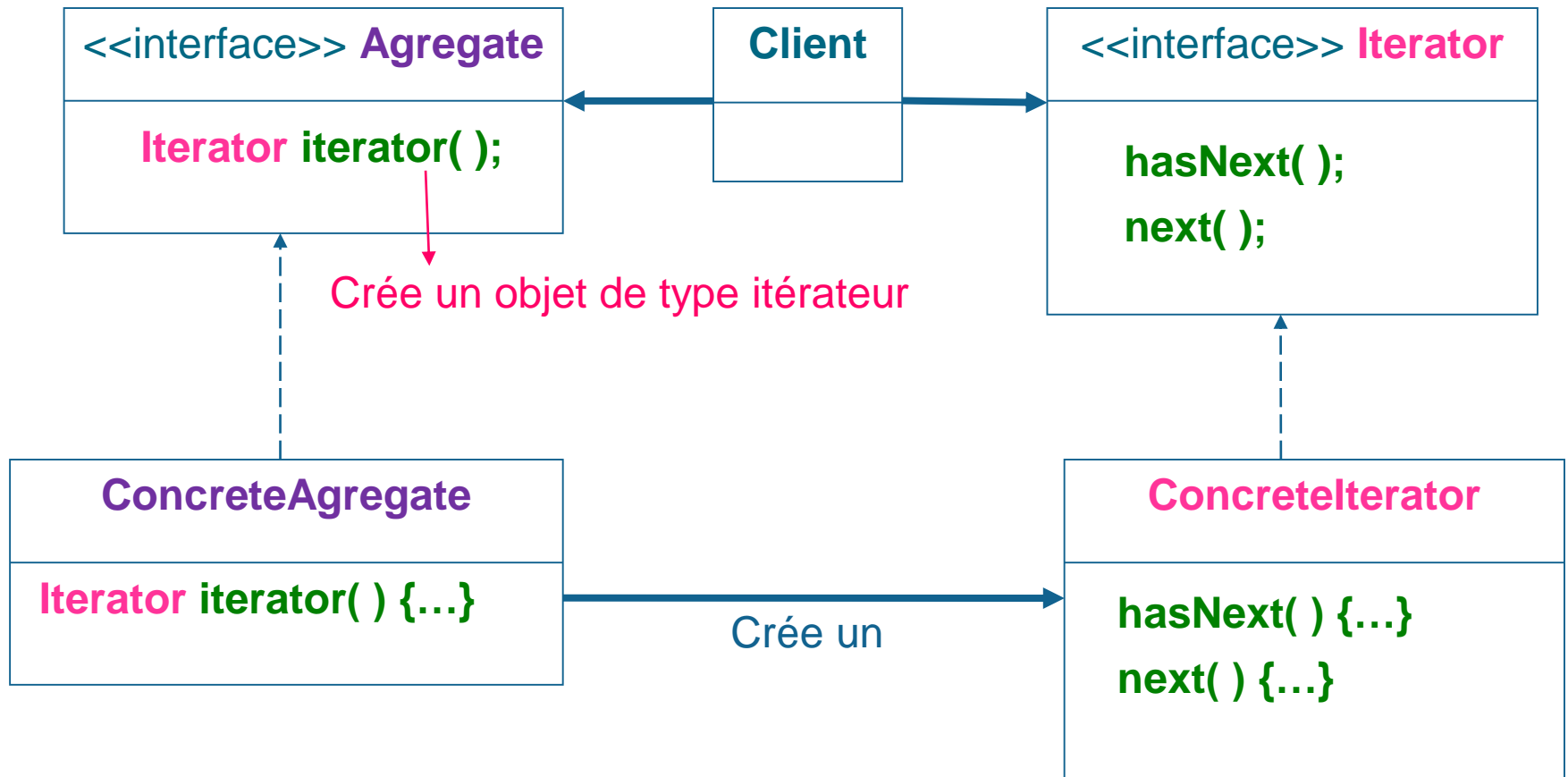
- demander l'élément **sui**vant
- savoir s'il y a **encore** des éléments dans la collection

⇒ Utiliser un objet **itérateur** sur la collection

1. Créer un itérateur en lui fournissant la collection
2. Cet itérateur propose les méthodes

- **hasNext** ⇒ vrai s'il existe encore au moins un élément dans la collection
- **next** ⇒ retourne l'élément suivant de la collection

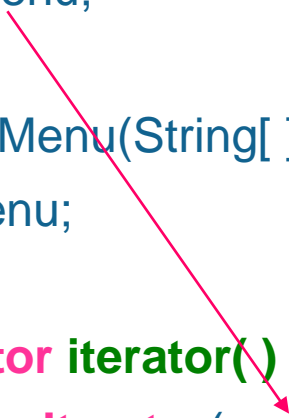
Iterator Pattern



Iterator Pattern

```
public interface Iterator {  
    Object next( );  
    boolean hasNext( );  
}
```

```
public class RestaurantMenu {  
    private String[ ] menu;  
  
    public RestaurantMenu(String[ ] menu) {  
        this.menu = menu;  
    }  
    public MenuIterator iterator( ) {  
        return new MenuIterator(menu);  
    }  
}
```



Iterator Pattern

```
public class MenuIterator implements Iterator {  
    private String[] menu;  
    private int position;  
  
    public MenuIterator(String[] menu) {  
        this.menu = menu;  
        position = 0;  
    }  
  
    public Object next() {  
        return menu[position++];  
    }  
  
    public boolean hasNext() {  
        return !(position >= menu.length || menu[position]==null)  
    }  
}
```

Iterator Pattern

```
public class IteratorDesignPattern {  
  
    public static void main(String[ ] args) {  
  
        String[ ] menu = { "Choucroute 14,5 euros", "Spaghetti bolo 9 euros",  
                            "Pizza 4 fromages 10 euros" };  
  
        RestaurantMenu restoMenu = new RestaurantMenu(menu);  
  
        MenuIterator iterateur = restoMenu.iterator( );  
  
        while ( iterateur.hasNext( ) ) {  
            System.out.println( iterateur.next( ) );  
        }  
    }  
}
```

Iterateur sur ArrayList

```
ArrayList <Personne> personnes = new ArrayList < > ( );
```

```
personnes.add(new Personne ("Pierre Leloup", 23));
```

```
...
```

```
Iterator <Personne> iterator = personnes.iterator( );
```

```
while (iterator.hasNext( ))
```

```
    System.out.println("Elément : " + iterator.next( ));
```

```
    // ou System.out.println("Age: " + iterator.next( ).age( ));
```

OK

Retourne un objet de type Personne

Les collections

1. Type générique
2. ArrayList
3. Collections d'objets et casting
4. Boucler sur une collection
 - Iterateur
 - Boucler for sur une collection

Boucle for sur une collection

Collection <E> liste

```
for ( E variable : liste) {  
    ... variable ...  
}
```


Boucle for sur une collection

Exemple

Soit : `ArrayList< Book > allBooks`

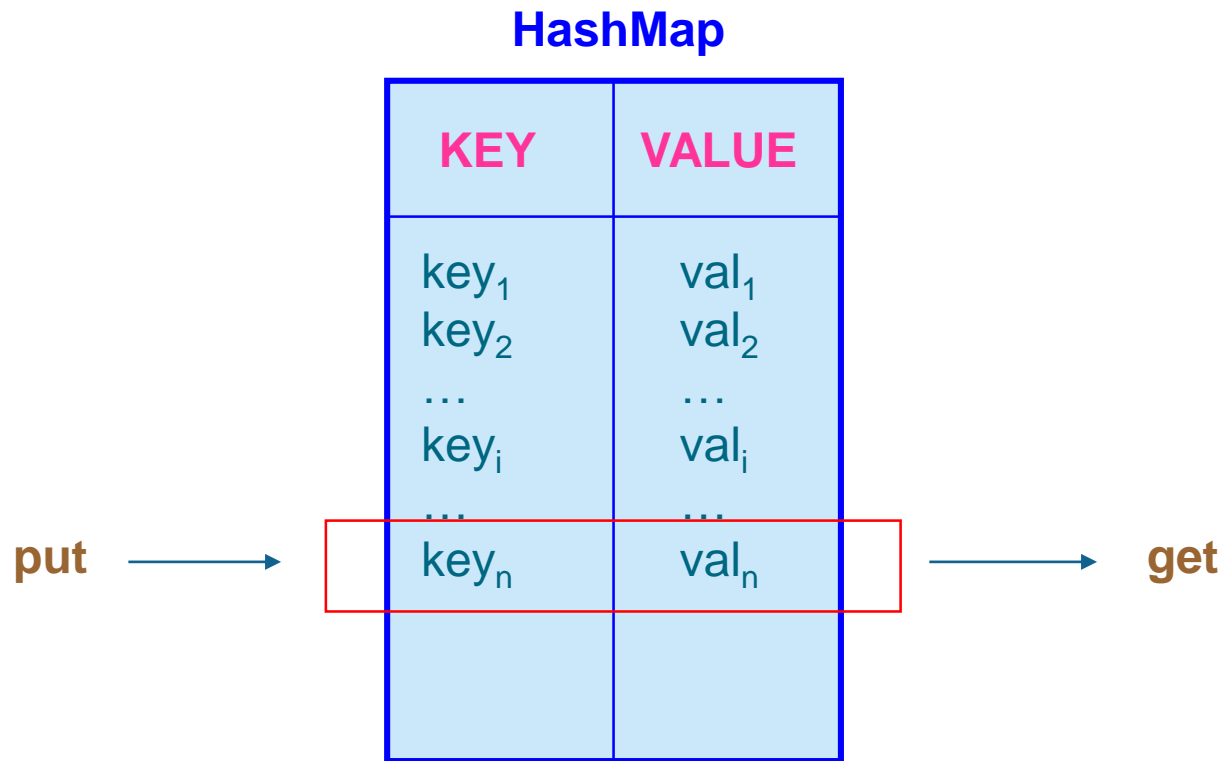
`for (Book book : allBooks)`

`System.out.println(book);`

Les collections

1. Type générique
2. ArrayList
3. Collections d'objets et casting
4. Boucler sur une collection
5. HashMap

HashMap



HashMap

KEY **VALUE**
↑ ↑
HashMap <**String**, **Personne**> **personnes** = new **HashMap** < > ();

KEYS **VALUES**
↑ ↑
personnes.put("key1", new **Personne** ("Anne Petit", 8));
personnes.put("key2", new **Personne** ("Pierre Leloup", 35));
personnes.put("key3", new **Personne** ("Jules Bastin", 88));

KEY
↑
System.out.println("Valeur correspondant à key1 " + **personnes.get** ("key1"));
System.out.println("Valeur correspondant à key2 " + **personnes.get**("key2"));
System.out.println("Valeur correspondant à key3 " + **personnes.get**("key3"));

HashMap

```
System.out.println("Nombre de paires clé-valeur : " + personnes.size( ));
```

```
personnes.remove("key3");
```

```
if (personnes.containsKey("key3"))
```

```
    System.out.println(personnes.get("key3"));
```

```
else System.out.println("La clé key3 n'existe pas dans la map.");
```

```
personnes.clear( );
```

```
if (personnes.isEmpty( ))
```

```
    System.out.println(" La map est vide");
```

Les collections

1. Type générique
2. ArrayList
3. Collections d'objets et casting
4. Boucler sur une collection
5. HashMap
 - Boucler sur une hashMap

```
HashMap <String, Personne> personnes = new HashMap < > ( );
```

```
personnes.put("key1",new Personne ("Anne Petit", 8) );
```

```
personnes.put("key2",new Personne ("Pierre Leloup", 35));
```

```
personnes.put("key3",new Personne ("Jules Bastin", 88));
```

```
for (Personne personne : personnes.values( ) ) {
```

```
    System.out.println ( "La personne est " + personne ) ;
```

```
}
```

```
for (String cle : personnes.keySet( ) ) {
```

```
    System.out.println ( "La clé identifiante est " + cle );
```

```
}
```

```
for (Entry<String,Personne> entree: personnes.entrySet( ) {
```

```
    System.out.println ( "La clé identifiante " + entree.getKey( ) +
```

```
        " correspond à la personne " + entree.getValue( ) );
```

```
}
```

Les collections

1. Type générique
2. ArrayList
3. Collections d'objets et casting
4. Boucler sur une collection
5. HashMap
6. Enumération

Enumération

Déclaration

```
enum EnumName {  
    VALUE1, VALUE2, VALUE3, ...  
}
```

Utilisation

EnumName.VALUE

Enumération

Exemple

Déclaration

```
enum Couleur {  
    ROUGE, BLEU, VERT, JAUNE  
}
```

Utilisation

Couleur.ROUGE

Énumération

Boucler sur les valeurs d'une énumération

```
for ( Couleur couleur : Couleur.values( ) )
```

```
    System.out.println(couleur);
```

