



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() ;

Instanciation

Compilation pas OK

Pas de casting

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

stringBox.set(3) ;

stringBox.set("John") ;

String string = stringBox.get() ;

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 Person {  
    private String name ;  
    private LocalDate birthDate ;  
  
    public Person (...) { ... }  
  
    ...      Getters et setters  
  
    public int age( ) { ... }  
  
    public String toString( ) {  
        return "La personne " + name + " agée de " + age() + " ans" ;  
    }  
}
```

ArrayList

```
public class Student extends Person {  
    private int year ;  
    private String section ;  
  
    public Student (...)  
    { ... }  
  
    public String toString( ) {  
        return super.toString( ) + " est inscrit en " + year + "e " + section ;  
    }  
}
```

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

```
public class Main {
```

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

```
{ ArrayList <Person> persons = new ArrayList <> ( ) ;
```

```
    persons.add(new Person ("Anne Petit",12)) ;
```

```
    ...
```

```
    persons.add(new Student ("Pol Louis",19,2,"info")) ;
```

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

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

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

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

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

retourne un objet Person

Typage

ArrayList

```
Person person = new Person ("Pol Castor", 50) ;
```

```
persons.set(2, person) ;
```

```
if (persons.contains(person))
```

```
    System.out.println("La liste contient: " + person) ;
```

```
persons.remove(1) ;
```

```
persons.clear( ) ;
```

```
if (persons.isEmpty( ))
```

```
    System.out.println("Liste vide") ;
```

Les collections

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

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

...

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

Polymorphisme :
toString() de Person

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

De type Object

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

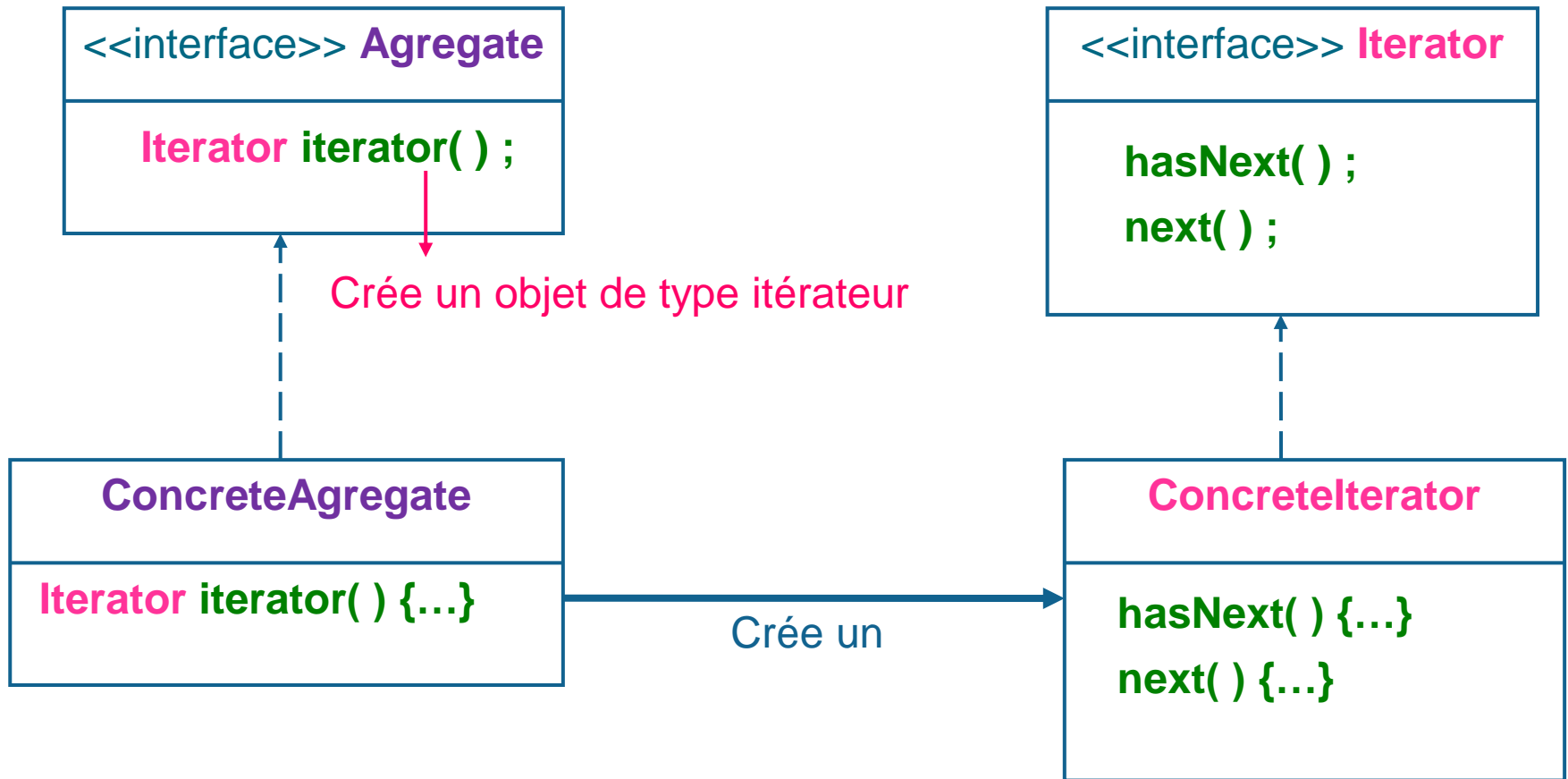
- demander l'élément **suisant**
- savoir s'il y a **encore** des éléments dans la collection

⇒ 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

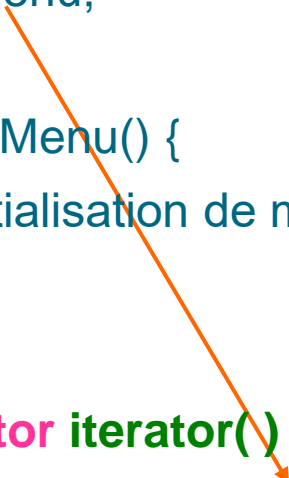
Iterator Pattern



Iterator Pattern

```
public interface Agregate {  
    Iterator iterator( );  
}
```

```
public class RestaurantMenu implements Agregate {  
    private String[ ] menu;  
  
    public RestaurantMenu() {  
        menu = ... // initialisation de menu, par exemple via accès à une BD  
    }  
  
    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)  
    }  
}
```

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

Iterator Pattern

```
public class IteratorDesignPattern {  
  
    public static void main(String[ ] args) {  
  
        RestaurantMenu restaurantMenu = new RestaurantMenu() ;  
  
        MenuIterator menuiterator = restaurantMenu.iterator( ) ;  
  
        while ( menuiterator.hasNext( ) ) {  
            System.out.println(menuiterator.next( ) ) ;  
        }  
    }  
}
```

Iterateur sur ArrayList

```
ArrayList <Person> persons = new ArrayList < > ( ) ;
```

```
persons.add(new Person ("Pierre Leloup", 23)) ;
```

```
...
```

```
Iterator <Person> iterator = persons.iterator( ) ;
```

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

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

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

OK

Retourne un objet de type Person

Les collections

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

Boucle for sur une collection

Collection <E> collection

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


Boucle for sur une collection

Exemple

Soit : `ArrayList< Book > allBooks`

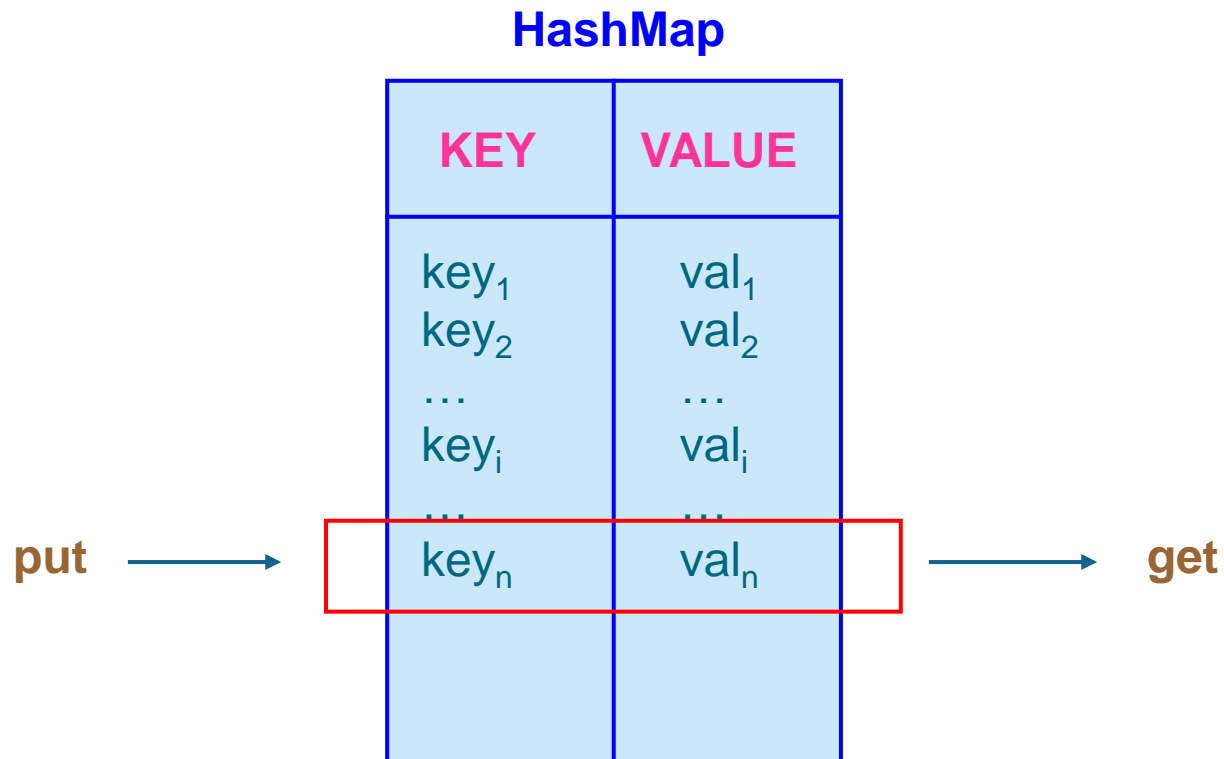
`for (Book book : allBooks)`

`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, Person> persons = new HashMap <> () ;

KEYS VALUES
↑ ↑
persons.put("AP", new Person ("Anne Petit",8)) ;
persons.put("PL", new Person ("Pierre Leloup",35)) ;
persons.put("JB", new Person ("Jules Bastin",88)) ;

KEY
↑
System.out.println("Valeur correspondant à JB : "+ persons.get ("JB")) ;
System.out.println("Valeur correspondant à AP : "+ persons.get("AP")) ;
System.out.println("Valeur correspondant à PL : "+ persons.get("PL")) ;

HashMap

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

```
persons.remove("PL") ;
```

```
if (persons.containsKey("XZ"))
```

```
    System.out.println(persons.get("XZ")) ;
```

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

```
persons.clear( ) ;
```

```
if (persons.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, Person> persons = new HashMap < > ( ) ;
```

```
persons.put("AP", new Person ("Anne Petit",8) ) ;
```

```
persons.put("PL", new Person ("Pierre Leloup",35)) ;
```

```
persons.put("JB", new Person ("Jules Bastin",88)) ;
```

```
for (Person person : persons.values()) {
```

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

```
}
```

```
for (String key : persons.keySet()) {
```

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

```
}
```

```
for (Entry<String,Person> entry : persons.entrySet()) {
```

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

```
        " correspond à la personne " + entry.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.VALUE1

Enumération

Exemple

Déclaration

```
enum Color {  
    RED, BLU, GREEN, YELLOW  
}
```


Utilisation

Color.RED

Énumération

Boucler sur les valeurs d'une énumération

```
for ( Color color : Color.values() ) {  
    ... color...  
}
```



Enumération

Associer une chaîne de caractères à chaque constante de l'énumération

⇒ Accessible via appel au `toString()`

```
public enum Department {  
    PRODUCT("Product Factory"),  
    SALE("Marketing"),  
    RESEARCH("Research and Development");
```

```
    private String text;
```

```
    Department (String text) {  
        this.text = text;  
    }
```

```
    @Override  
    public String toString() {  
        return this.text;  
    }
```

```
}
```