

# Object-Oriented Programming: Application to the Java Language

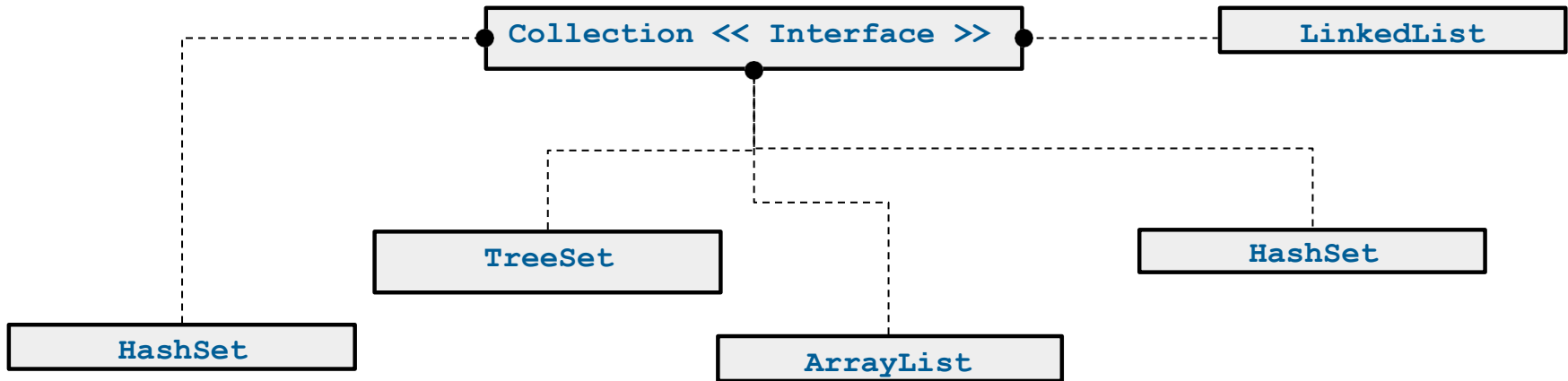
The collections

# The collections

- For now, we've studied the table to structure the data
  - Static size
  - Slow to find specific elements
  - Impossibility of using a displacement pattern in elements
- Since version 2, Java has offered classes for manipulating the main data structures
  - Dynamic arrays implemented by *ArrayList* and *Vector*
  - Lists implemented by *LinkedList*
  - Sets implemented by *HashSet* and *TreeSet*

# The collections

- These classes all indirectly implement the same *Collection* interface, which they complement with their own functionalities



- Since Java version 5, generics can be used *to type* the contents of Collections
  - Before : *Car myCar = (Car)myList.get(2)*
  - Now: *Car myCar = myList.get(2)*

**No more explicit  
conversion problems**



# The collections

## ➤ The Collection interface

- *Genericity and references*: elements of any type can be stored, provided they are objects. A new element introduced into a Java collection is a reference to the object, not a copy.
- *Iterators*: let you go through the different elements of a collection one by one.
- Efficiency of operations on collections
- Operations common to all collections: the collections we're going to study all implement at least the *Collection* interface, so they have common features

# Collections: Java generics

- With Java version 5, generics can be used in collections and other aspects of the language.
- A special syntax has been added to take generics into account.
  - `< ? >` : indicates that the class type must be specified
  - `< ? , ? >` : indicates that two types must be specified
- With generics, it will be possible to set the type of content stored in collections when building the collection.
- Benefits
  - All accessor and modifier methods that manipulate the elements of a collection are *signed* according to the type defined when the collection was built.
  - Checking types during development (before problems with *CastClassException*)

## The collections: Iterator

- Iterators allow you to browse the elements of a collection without precise knowledge of the collection's type:

### Polymorphism

- There are two families of iterators

- *one-way*

The collection is browsed from beginning to end; an item is accessed only once.

- *bidirectional*

The collection can be explored in both directions; you can move backwards and forwards through the collection as you

wish.

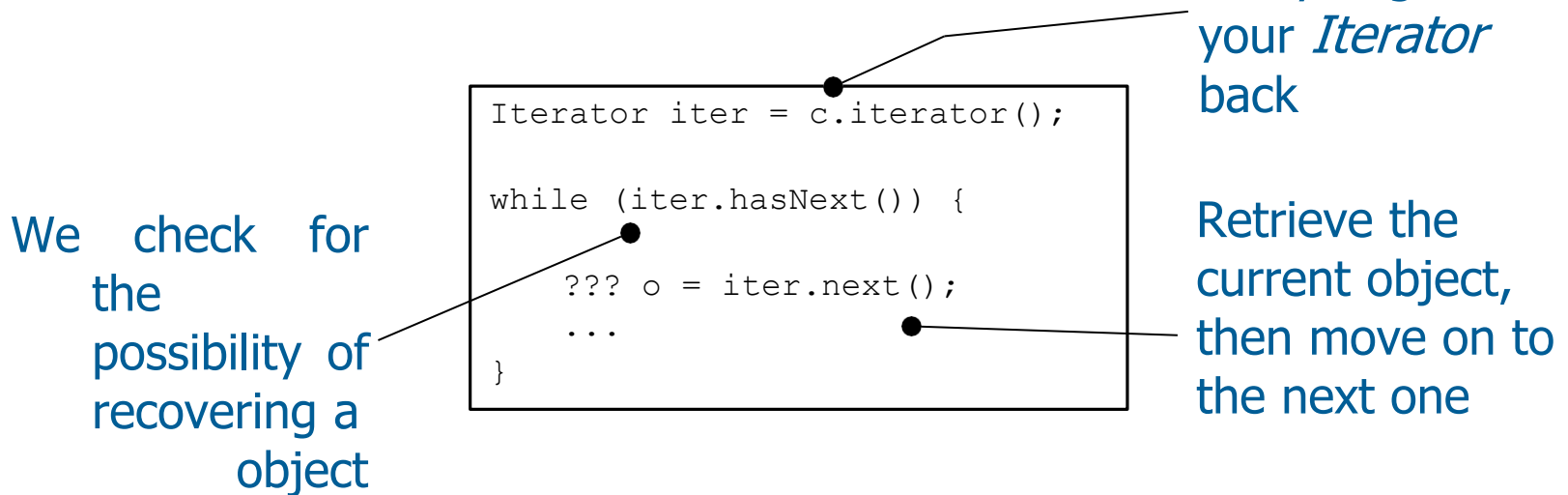
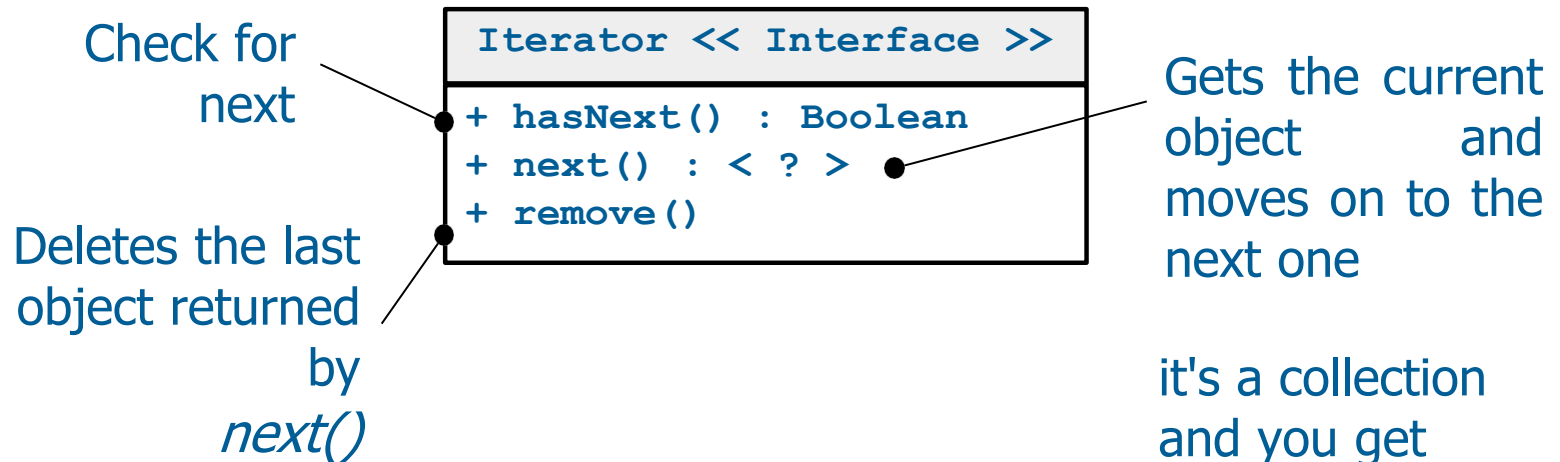
**The notion of Iterator  
is part of the set of  
Design Patterns**



# The collections: Iterator

## ➤ One-way iterator: *Iterator* interface

- By default, all collections have an *Iterator* attribute



# The collections: Iterator

## ➤ Bidirectional iterator: *ListIterator* interface

➤ This applies to lists and dynamic tables

➤ Add and remove objects

**Iterator << Interface >>**

Check for  
previous

**ListIterator << Interface >>**

```
+ previous() : < ? >  
+ hasPrevious() : Boolean  
+ add(< ? >) ●  
+ set(< ? >) ●  
+ ...
```

Adds or modifies  
an item in the  
collection at the  
current position

We check  
whether it is  
possible to  
retrieve an object  
previously

Retrieve the  
object previously  
we move  
on to the  
previous

```
Iterator iter = c.listIterator();  
while (iter.hasPrevious()) {  
    ??? o = iter.previous();  
    ...  
}
```

it's a collection  
and we get its  
*ListIterator*  
Initialise at the  
start of the list



## The collections : LinkedList

- This class can be used to manipulate double-chained lists.
- Each collection element is implicitly associated with two pieces of information: the references to the previous and next elements.



Nothing  
after  
these  
elements  
, we go  
backward  
s

```
LinkedList<String> l1 = new LinkedList<String>();  
ListIterator iter = l1.listIterator();  
  
iter.add("Hello");  
iter.add("Cuckoo");  
  
while(iter.hasPrevious()) {  
    String o = iter.previous();  
    System.out.println(o);  
}
```

Adding  
elements  
through the  
iterator

Using  
*LinkedList* is  
transparent

# The collections : LinkedList

- Possibility of using collections (here *LinkedList* is an example) without iterators but less efficient!!!!

```
LinkedList<String> l1 = new LinkedList<String>();  
  
l1.add("Hello");  
l1.add("Cuckoo");  
  
for (int i = 0; i < l1.size(); i++) {  
    String o = l1.get(i);  
    System.out.println(o);  
}
```

The use of *LinkedList* is not transparent. Knowledge of these methods is mandatory

Using *the add* method of the class *LinkedList*

**Do not modify the collection (*add* from *LinkedList*) while using the iterator (*next()*)**



## Collections : ArrayList

- The *ArrayList* class is an encapsulation of the array with the ability to make it dynamic in size
- Possibility of using *ListIterators*, but we prefer to use them for an element of a given rank.

```
ArrayList<Object> myArrayList = new ArrayList<Object>();

myArrayList.add("Cuckoo")
; myArrayList.add(34);

for (int i = 0; i < myArrayList.size(); i++) {
    Object myObject = myArrayList.get(i);
    if (myObject instanceof String) {
        System.out.println("String:" + ((String)myObject));
    }

    if (my_object instanceof Integer) {
        System.out.println("Integer:" + ((Integer)myObject));
    }
}
```

Use the *ArrayList* class  
instead of the location of  
the *Vector* class



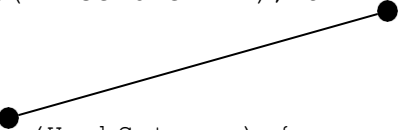
## Collections: HashSet

- The *HashSet* class is used to manage sets of
- Two elements cannot be identical
- Plan two things for your classes
  - Redefinition of *the hashCode()* method, which is used to order the elements of a set (calculating an object's hash table).
  - Redefinition of the *equals(Object)* method, which compares objects of the same class to determine whether an element belongs to the set

# Collections: HashSet

## ➤ Example: managing points with *HashSet*

```
public class TestHashSet {  
    public static void main(String[] argv) {  
        Point p1 = new Point(1,3); Point p2 = new Point(2,2);  
        Point p3 = new Point(4,5); Point p4 = new Point(1,8);  
        Point p[] = {p1, p2, p1, p3, p4, p3}  
  
        HashSet<Point> ens = new HashSet<Point>();  
        for (int i = 0; i < p.length; i++) {  
            System.out.println("Le Point "); p[i].affiche();  
            boolean ajoute = ens.add(p[i]);  
            if (ajoute)  
                System.out.println("has been added");  
            else  
                System.out.println("is already present");  
            System.out.print("Ensemble = "); affiche(ens);  
        }  
    }  
  
    public static void affiche(HashSet ens) {  
        Iterator iter = ens.iterator();  
        while(iter.hasNext()) {  
            Point p =  
                iter.next();  
            p.affiche();  
        }  
        System.out.println();  
    }  
}
```

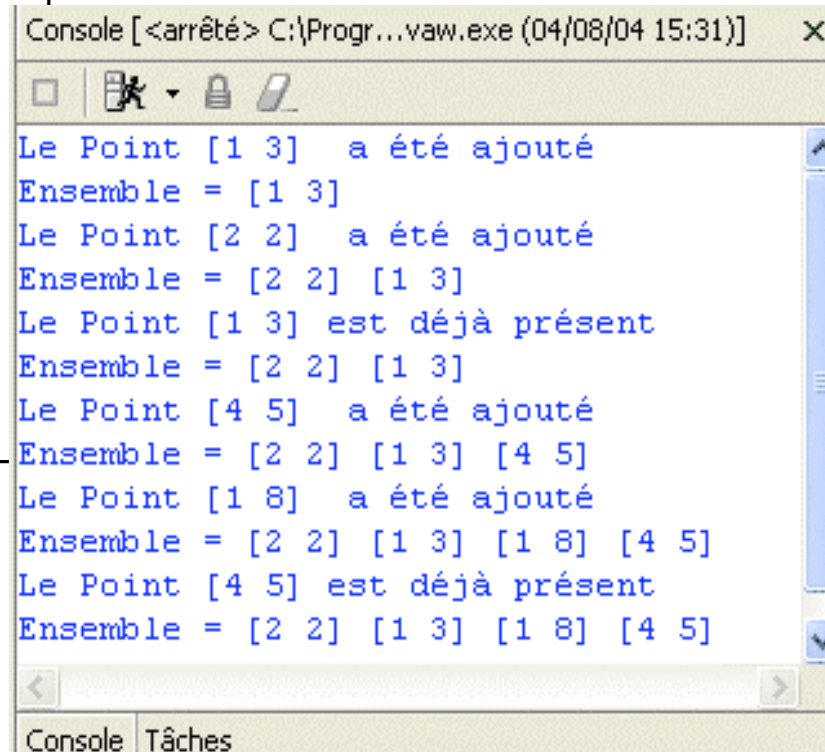
A diagram consisting of two black dots connected by a straight line. One dot is located at the end of the line of code 'System.out.print("Ensemble = "); affiche(ens);' in the main method. The other dot is located at the beginning of the line of code 'public static void affiche(HashSet ens) {' in the affiche method.

# Collections: HashSet

## ► Example: point management with *HashSet*

```
public class Point {  
    private int x,y;  
  
    Point(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public int hashCode() {  
        return x+y;  
    }  
    public boolean equals(Object pp) {  
        Point p = (Point)pp;  
        return ((this.x == p.x) &  
            (this.y == p.y));  
    }  
    public void affiche() {  
        System.out.print "[" + x + " " + y + "] ";  
    }  
}
```

Redefinition of  
*hashCode()* and  
*equals(Object)*  
methods



```
Console [ <arrêté> C:\Progr...vaw.exe (04/08/04 15:31)]  
  
Le Point [1 3] a été ajouté  
Ensemble = [1 3]  
Le Point [2 2] a été ajouté  
Ensemble = [2 2] [1 3]  
Le Point [1 3] est déjà présent  
Ensemble = [2 2] [1 3]  
Le Point [4 5] a été ajouté  
Ensemble = [2 2] [1 3] [4 5]  
Le Point [1 8] a été ajouté  
Ensemble = [2 2] [1 3] [1 8] [4 5]  
Le Point [4 5] est déjà présent  
Ensemble = [2 2] [1 3] [1 8] [4 5]  
  
Console  Tâches
```