JAVA COLLECTIONS

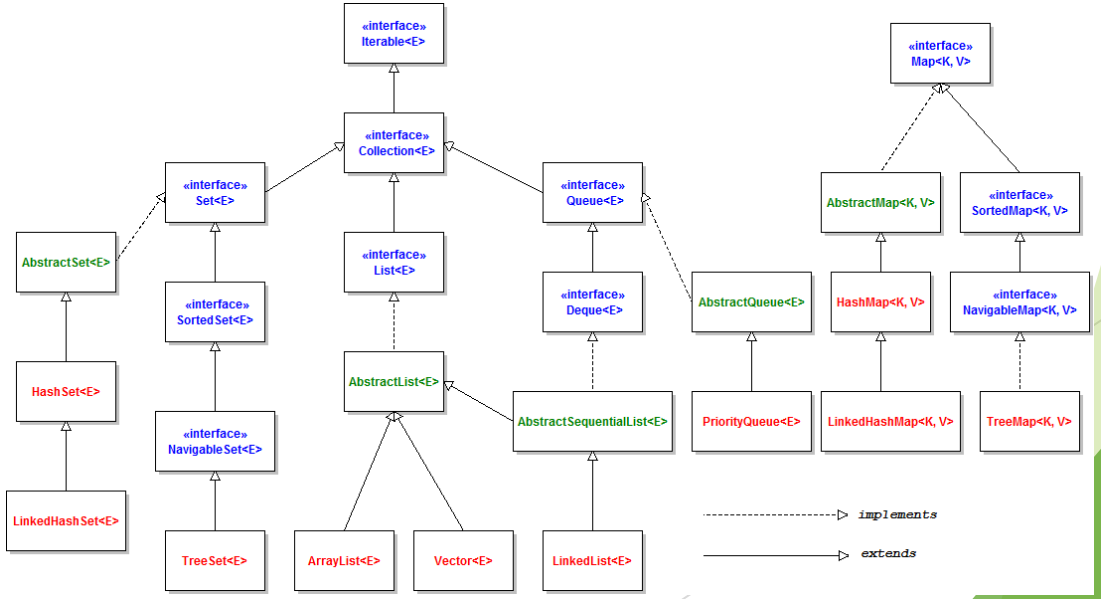
Abdelaaziz EL Hibaoui
Computer Science Department
Faculty of Science – Tetouan
aelhibaoui@uae.ac.ma

Chapter Objectives

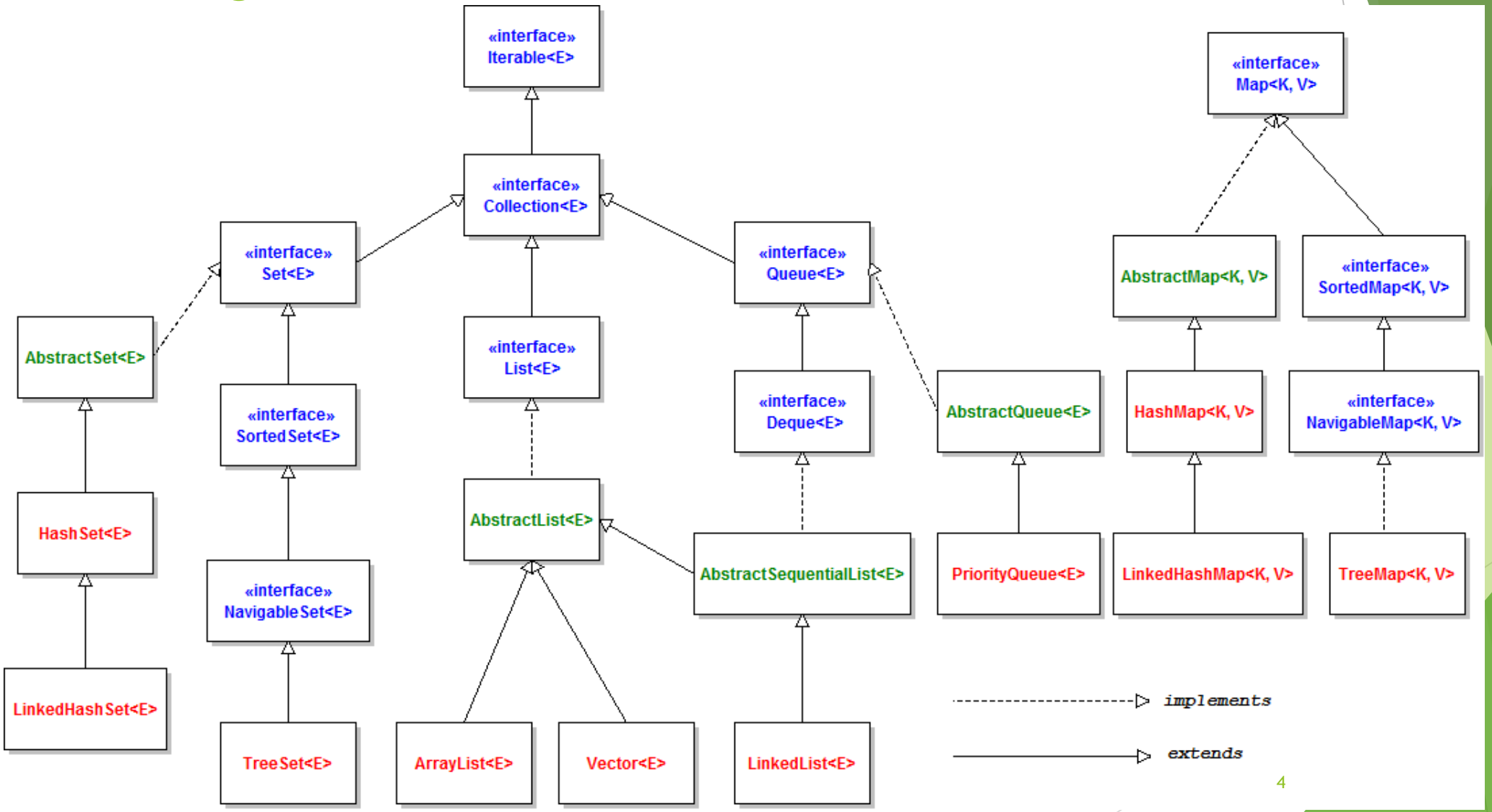
- To learn how to use the collection classes supplied in the Java library
- To understand the idea of Java Generics
- To use iterators to traverse collections
- To use Collection Algorithms
- To choose appropriate collections for solving programming problems

Collections Overview

- When you need to organize multiple objects in your program, you can place them into a collection.
- **Collection classes in Java are containers of Objects which by polymorphism can hold any class that derives from Object (which is actually, any class)**
- **Collections are used to store, retrieve, manipulate, and communicate aggregate data.**
- **There are two main interfaces for all the collection types in Java:**
  - **Collection<E>**
  - **Map<K,V>**

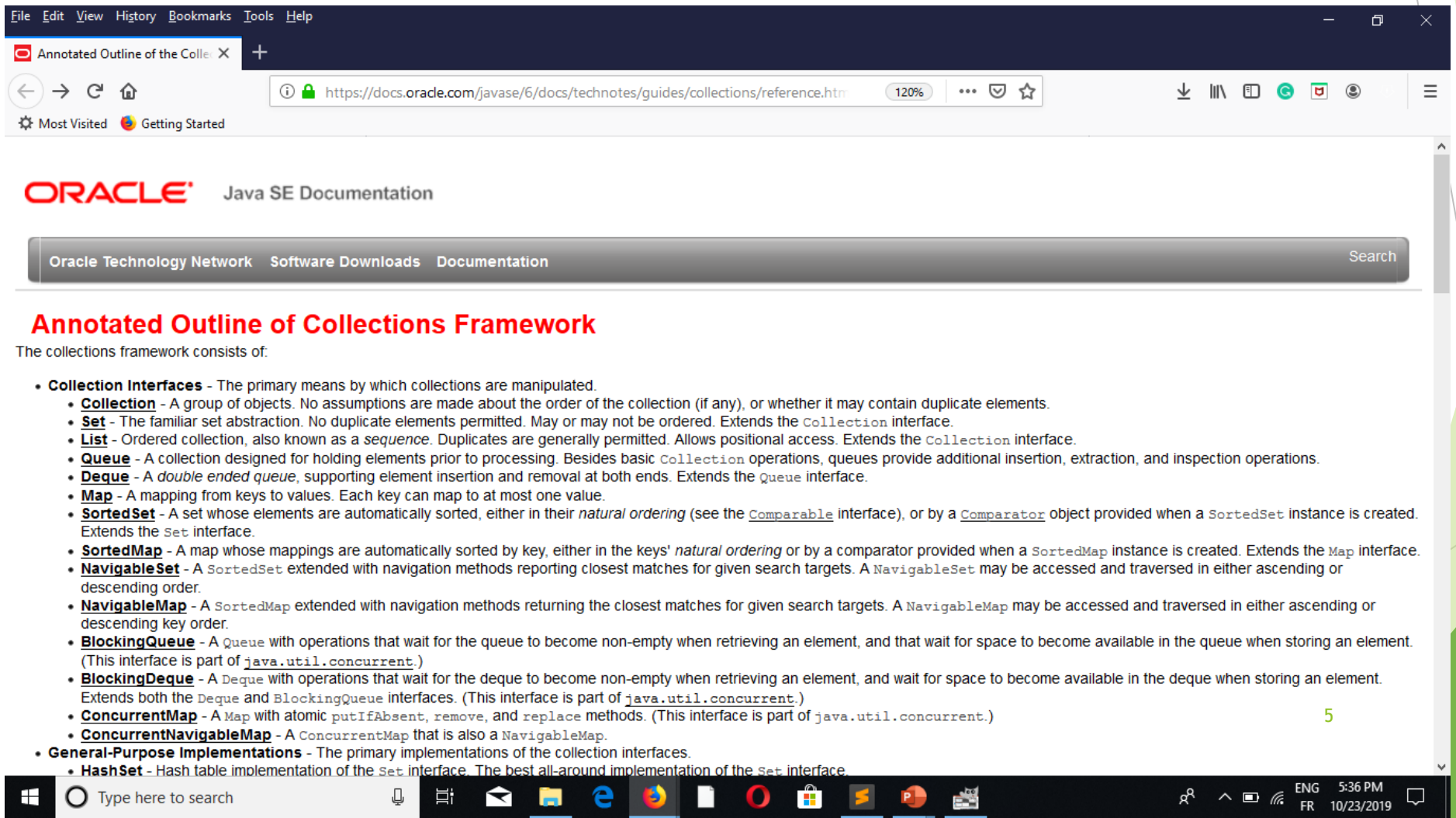


Class diagram of Java Collections framework



List of all Collections and related frameworks:

http://java.sun.com/javase/6/docs/technotes/guides/collections/reference.html



Collection Interface

```
import java.util.Iterator;
interface Collection<E> {
    boolean add(E o);
    boolean addAll(Collection<? extends E> c);
    void clear();
    boolean contains(Object o);
    boolean containsAll(Collection<?> c);
    boolean equals(Object o);
    int hashCode();
    boolean isEmpty();
    Iterator<E>  iterator();
    boolean remove(Object o);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    int size();
    Object[ ] toArray();
    <T> T[ ] toArray(T[ ] a);
```

Collection Interface Methods

boolean	add(Object o) **Ensures that this Collection contains the specified element (optional operation).**
boolean	addAll(Collection c) Adds all of the elements in the specified Collection to this Collection (optional operation).
void	clear() Removes all of the elements from this Collection (optional operation).
boolean	contains(Object o) Returns true if this Collection contains the specified element.
boolean	containsAll(Collection c) Returns true if this Collection contains all of the elements in the specified Collection.
boolean	equals(Object o) Compares the specified Object with this Collection for equality.
int	hashCode() Returns the hash code value for this Collection.
boolean	isEmpty() Returns true if this Collection contains no elements.

Collection Interface Methods

Iterator	**iterator()** **Returns an Iterator over the elements in this Collection.**
boolean	**remove**(Object o) Removes a single instance of the specified element from this Collection, if it is present (optional operation).
boolean	**removeAll**(Collection c) Removes from this Collection all of its elements that are contained in the specified Collection (optional operation).
boolean	**retainAll**(Collection c) Retains only the elements in this Collection that are contained in the specified Collection (optional operation).
int	**size**() Returns the number of elements in this Collection.
Object[]	**toArray**() Returns an array containing all of the elements in this Collection.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this Collection, whose runtime type is that of the specified array.

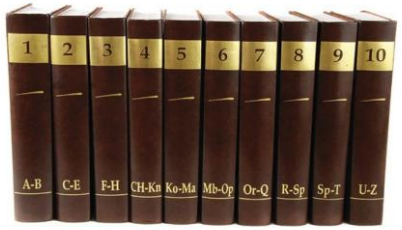
Generic Collections

- The Collection Framework uses Generics
  - Each list is declared with a type field in **< >** angle brackets
  - **Using Generics the Collection classes can be aware of the types they store**

```
ArrayList<String> employeeNames = . . .;
```

```
LinkedList<String>
LinkedList<Employee>
```

A **list** is a collection that maintains the order of its elements.



- Ordered Lists
  - `ArrayList`
    - Stores a list of items in a dynamically sized array
  - `LinkedList`
    - Allows speedy insertion and removal of items from the list

Iterators

- Iterators allow you to move through a list easily
  - Similar to an index variable for an array
- Iterators are often used in while and “for-each” loops
  - `hasNext` returns true if there is a next element
  - `next` returns a reference to the value of the next element

```
while (iterator.hasNext())
{
    String name = iterator.next();
    // Do something with name
}
```

```
for (String name : employeeNames)
{
    // Do something with name
}
```

  - Where is the iterator in the “for-next” loop?
    - It is used ‘behind the scenes’

Adding and Removing with Iterators

- Adding
  ```
  iterator.add("Iyad");
  ```
  - A new node is added AFTER the Iterator
  - The Iterator is moved past the new node
- Removing
  - Removes the object that was returned with the last call to `next` or `previous`
  - It can be called only once after `next` or `previous`
  - You cannot call it immediately after a call to `add`.

If you call the `remove` method improperly, it throws an `IllegalStateException`.

```
while (iterator.hasNext())
{
  String name = iterator.next();
  if (condition is true for name)
  {
    iterator.remove();
  }
}
```

Setting with Iterators

- set (elem): replace by elem the current element, that is the last return by **next ()** or **previous()**

```
ListIterator it = c . listIterator ( ) ;
While ( it . hasNext ( ) ) {
      Object o = it . next ( ) ;
      If ( Condition ) it . set ( nul l ) ;
} //
```

Stacks and Queues

- Another way of gaining efficiency in a collection is to reduce the number of operations available.
- Two examples are:
  - Stack
    - Remembers the order of its elements, but it does not allow to insert elements in every position.
    - It allows only add and remove elements at the top
    - A stack is a collection of elements with "**last-in, first-out**" retrieval.

  - Queue
    - Add items to one end (the tail)
    - Remove them from the other end (the head)
    - A queue is a collection of elements with "first-in, first-out" retrieval.
    - Example: A line of people waiting for a bank teller

Working with Stacks

Table 7 Working with Stacks

`Stack<Integer> s = new Stack<Integer>();`	Constructs an empty stack.

`s.push(1);`
`s.push(2);`
`s.push(3);`
	Adds to the top of the stack; `s` is now `[1, 2, 3]`. (Following the `toString` method of the `Stack` class, we show the top of the stack at the end.)

`int top = s.pop();`	Removes the top of the stack; `top` is set to 3 and `s` is now `[1, 2]`.
`head = s.peek();`	Gets the top of the stack without removing it; `head` is set to 2.

Stack class

- The Java library provides a `Stack` class that implements the abstract stack type's push and pop operations.
  - The Stack is not technically part of the Collections framework, but uses generic type parameters

`Stack<Integer> s = new Stack<Integer>();`	Constructs an empty stack.

`s.push(1);`
`s.push(2);`
`s.push(3);`
	Adds to the top of the stack; `s` is now `[1, 2, 3]`. (Following the `toString` method of the `Stack` class, we show the top of the stack at the end.)

`int top = s.pop();`	Removes the top of the stack; `top` is set to 3 and `s` is now `[1, 2]`.
`head = s.peek();`	Gets the top of the stack without removing it; `head` is set to 2.

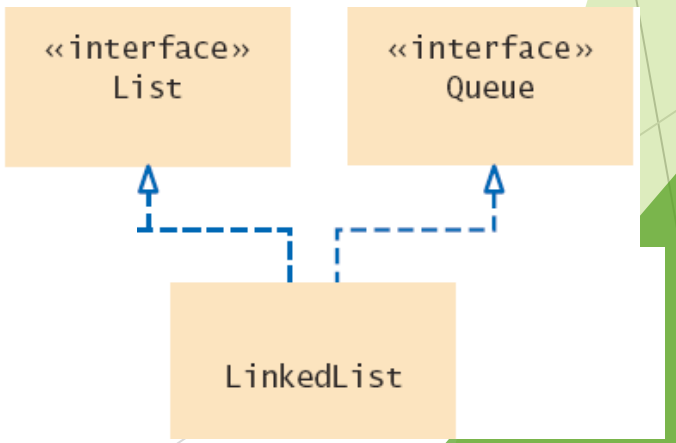
Stack example of use

```
Stack<String> s = new Stack<String>();
s.push("A");
s.push("B");
s.push("C");
// The following loop prints C, B, and A
while (s.size() > 0)
{
  System.out.println(s.pop());
}
```

Linked Lists

- Linked lists use references to maintain an ordered lists of ‘nodes’
  - The ‘head’ of the list references the first node
  - Each node has a value and a reference to the next node

  - They can be used to implement
    - A `List` Interface
    - A Queue Interface



ListIterators

- When traversing a `LinkedList`, use a `ListIterator`
  - Keeps track of where you are in the list.

```
LinkedList<String> employeeNames = . . .;
ListIterator<String> iter = employeeNames.listIterator()
```

- Use an iterator to:
  - Access elements inside a linked list
  - Visit other than the first and the last nodes

```
ListIterator iter = c.ListIterator ( c .size ( ) ) ;
// Current position : end of the list.
While ( iter.hasPrevious ( ) ) {
        Object o = iter.previous ( ) ;
        System.out.println ( o ) ;
}
```

ListIterators methods

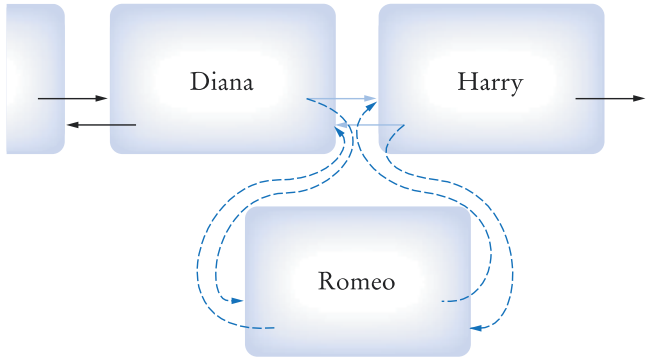
`String s = iter.next();`	Assume that `iter` points to the beginning of the list `[Sally]` before calling `next`. After the call, `s` is `"Sally"` and the iterator points to the end.
`iter.previous();` `iter.set("Juliet");`	The `set` method updates the last element returned by `next` or `previous`. The list is now `[Juliet]`.
`iter.hasNext()`	Returns `false` because the iterator is at the end of the collection.

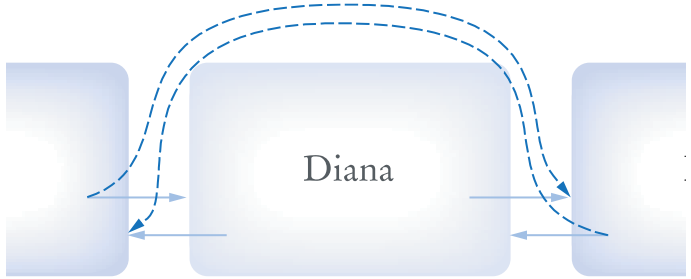
`if (iter.hasPrevious())`
`{`
`  s = iter.previous();`
`}`
	`hasPrevious` returns `true` because the iterator is not at the beginning of the list. `previous` and `hasPrevious` are `ListIterator` methods.

`iter.add("Diana");`	Adds an element before the iterator position (`ListIterator` only). The list is now `[Diana, Juliet]`.
`iter.next();` `iter.remove();`	`remove` removes the last element returned by `next` or `previous`. The list is now `[Diana]`.

Linked Lists Operations

- Efficient Operations
  - Insertion of a node
    - Find the elements it goes between
    - Remap the references
  - Removal of a node
    - Find the element to remove
    - Remap neighbor's references
  - Visiting all elements in order
- Inefficient Operations
  - Random access





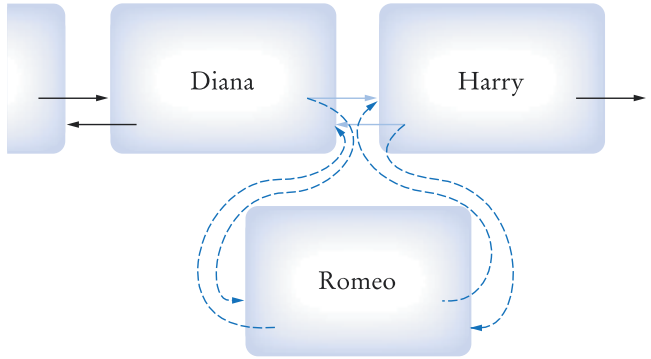
Each instance variable is declared just like other variables we have used.

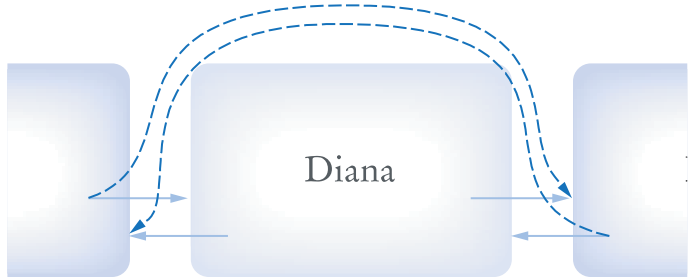
LinkedList: Important Methods

`list.addLast("Harry");`	Adds an element to the end of the list. Same as `add`.
`list.addFirst("Sally");`	Adds an element to the beginning of the list. `list` is now `[Sally, Harry]`.
`list.getFirst();`	Gets the element stored at the beginning of the list; here `"Sally"`.
`list.getLast();`	Gets the element stored at the end of the list; here `"Harry"`.
`String removed = list.removeFirst();`	Removes the first element of the list and returns it. `removed` is `"Sally"` and `list` is `[Harry]`. Use `removeLast` to remove the last element.
`ListIterator<String> iter = list.listIterator()`	Provides an iterator for visiting all list elements

Linked Lists Operations

- Efficient Operations
  - Insertion of a node
    - Find the elements it goes between
    - Remap the references
  - Removal of a node
    - Find the element to remove
    - Remap neighbor's references
  - Visiting all elements in order
- Inefficient Operations
  - Random access





Each instance variable is declared just like other variables we have used.

List Example

```java
import java.util.*;
public class ListExample {
    public static void main(String args[]) {
        List list = new ArrayList();
        list.add("Fatima");
        list.add("Aicha");
        list.add("Hafsa");
        list.add("Aicha");
        list.add("Mariam");
        System.out.println(list);
        System.out.println("2: " + list.get(2));
        System.out.println("0: " + list.get(0));
        LinkedList queue = new LinkedList();
        queue.addFirst("Fatima");
        queue.addFirst("Aicha");
        queue.addFirst("Hafsa");
        queue.addFirst("Aicha");
        queue.addFirst("Mariam");
        System.out.println(queue);
        queue.removeLast();
        queue.removeLast();
        System.out.println(queue);
    }
}
```

```
>> java ListExample
[Fatima, Aicha, Hafsa, Aicha, Mariam]
2: Hafsa
0: Fatima
[Mariam, Aicha, Hafsa, Aicha, Fatima]
[Mariam, Aicha, Hafsa]
```

Example

```
import java.util.*;
public class Liste {
   public static void main(String args[]) {
        LinkedList l = new LinkedList();
        System.out.print("Liste A:"); affiche(l);
        l.add("a"); l.add("b");
        System.out.print("Liste B:"); affiche(l);
        ListIterator it = l.listIterator();
        it.next();
        it.add("c"); it.add("b");
        System.out.print("Liste C:"); affiche(l);
        it = l.listIterator();
        it.next();
it.add("b"); it.add("d");
        System.out.print("Liste D:"); affiche(l);
        it = l.listIterator(l.size());
        while (it.hasPrevious()) {
            String ch = (String) it.previous();
            if (ch.equals("b")) {
                it.remove();
                break;
            }
        }
        while (it.hasPrevious()) {
        String ch = (String) it.previous();
        if (ch.equals("b")) {
            it.remove();
            break;
        }
      }
      System.out.print("Liste E:"); affiche(l);
      it = l.listIterator();
      it.next(); it.next();
      it.set("x");
      System.out.print("Liste F:"); affiche(l);
    }
    public static void affiche (LinkedList l) {
        ListIterator iter = l.listIterator();
        while (iter.hasNext())
        System.out.print (iter.next() + " ");
        System.out.println();
    }
}
```

```
>> java Liste
Liste A:
Liste B:a b
Liste C:a c b b
Liste D:a b d c b b
Liste E:a b d c b
Liste F:a x d c b
```

LinkedList Example

```
import java.util.*;
public class Array {
   public static void main(String args[]) {
        ArrayList v = new ArrayList();
        for (int i=0;i<10;i++)
            v.add(new Integer(i));
        System.out.println("En 0: contenu de v" + v);
        v.add(2,"AAA");
        v.add(4,"BBB");
        v.add(8,"CCC");
        v.add(5,"DDD");

        System.out.println("En I: contenu de v" + v);
        for (int i=0;i<v.size(); i++)
          if (v.get(i) instanceof String) v.set(i,null);
        System.out.println("En II: contenu de v" + v);
        LinkedList l = new LinkedList();
            l.add(new Integer (5));
            l.add(null);
            v.removeAll(l);
            System.out.println("En III contenu v" + v);
      }
   }
```

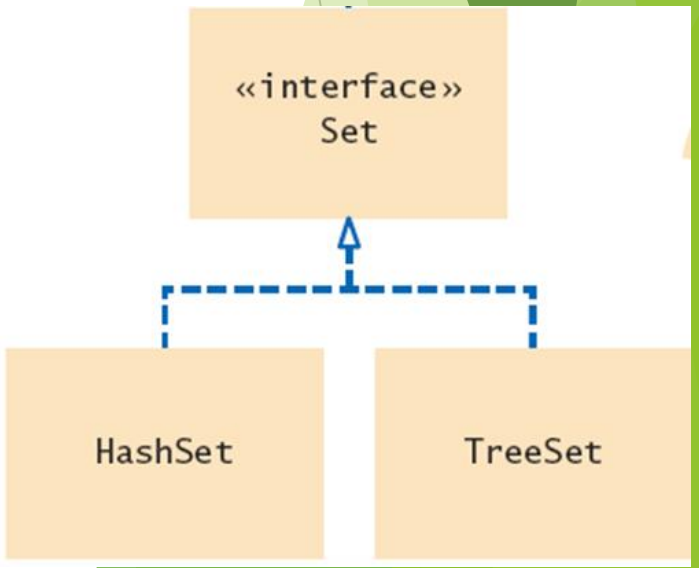
```
>> java Array
En 0: contenu de v[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
En I: contenu de v[0, 1, AAA, 2, BBB, DDD, 3, 4, 5, CCC,
6, 7, 8, 9]
En II: contenu de v[0, 1, null, 2, null, null, 3, 4, 5, null,
6, 7, 8, 9]
En III contenu v[0, 1, 2, 3, 4, 6, 7, 8, 9]
```

Sets

A **set** is an unordered collection of unique elements.

- The collection does not keep track of the order in which elements have been added
  - Therefore, it can carry out its operations more efficiently than an ordered collection

- Classes implement the Set interface
  - `HashSet`
    - Uses hash tables to speed up finding, adding, and removing elements
  - `TreeSet`
    - Uses a binary tree to speed up finding, adding, and removing elements



Iterators and Sets

- Iterators are also used when processing sets
  - `hasNext` returns true if there is a next element
  - `next` returns a reference to the value of the next element
  - `add` via the iterator is not supported for TreeSet and HashSet

```
Iterator<String> iter = names.iterator();
while (iter.hasNext())
{
   String name = iter.next();
   // Do something with name
}
```

```
for (String name : names)
{
   // Do something with name
}
```

  - Note that the elements are not visited in the order in which you inserted them.
  - They are visited in the order in which the set keeps them:
    - Seemingly random order for a HashSet
    - Sorted order for a TreeSet

Working With Sets (1)

`Set<String> names;`	Use the interface type for variable declarations.
`names = new HashSet<String>();`	Use a `TreeSet` if you need to visit the elements in sorted order.
`names.add("Romeo");`	Now `names.size()` is 1.
`names.add("Fred");`	Now `names.size()` is 2.
`names.add("Romeo");`	`names.size()` is still 2. You can't add duplicates.
`if (names.contains("Fred"))`	The `contains` method checks whether a value is contained in the set. In this case, the method returns `true`.

Working With Sets (2)

`System.out.println(names);`	Prints the set in the format `[Fred, Romeo]`. The elements need not be shown in the order in which they were inserted.

`for (String name : names)`
`{`
`. . .`
`}`
	Use this loop to visit all elements of a set.

`names.remove("Romeo");`	Now `names.size()` is 1.
`names.remove("Juliet");`	It is not an error to remove an element that is not present. The method call has no effect.

HashSet Example 1

```
1  import java.util.*;
2  public class Ens {
3      public static void main (String args[]) {
4          String phrase = "Le prof n'est ni ange ni bête et le"
5                  + " malheur veut que qui veut faire l'ange fait la bête";
6          String voy="aeiouyê";
7          HashSet lettres = new HashSet();
8          for (int i=0;i<phrase.length();i++)
9              lettres.add(phrase.substring(i,i+1));
10         System.out.println("lettres presentes:"+lettres);
11
12         HashSet voyelles = new HashSet();
13         for (int i=0;i<voy.length();i++)
14             voyelles.add(voy.substring(i,i+1));
15
16         lettres.removeAll (voyelles);
17         System.out.println("lettres sans les voyelles " + lettres);
18     }
19 }
```

```
>> java Ens
lettres presentes:[f, g,  , e, b, ê, ', a, L, n, o, l, m, h, i, v, u, t, s, r, q, p]
lettres sans les voyelles [f, g,  , b, ', L, n, l, m, h, v, t, s, r, q, p]
```

HashSet Example 2

```java
import java.util.*;
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 String toString() {
          return "[" + x + " " + y + "]";
   }
}
```

```java
import java.util.*;
public class EnsPt1 {
    public static void main (String args[]) {
        Point p1 = new Point (1,3), p2 = new Point (2,2);
        Point p3 = new Point(4,5), p4 = new Point (1,8);

        Point p[] = {p1, p2, p1, p3, p4, p3};
        HashSet ens = new HashSet();
        for (int i=0;i<p.length;i++) {
              System.out.print("le point");
              System.out.println(p[i]);
              boolean ajoute = ens.add(p[i]);
              if (ajoute)
                   System.out.println("a ete ajouté");
              else System.out.println("est déjà présent");
              System.out.print("ensemble=");
              affiche(ens);
         }
    }

    public static void affiche (HashSet ens){
        Iterator iter = ens.iterator();
        while (iter.hasNext()) {
               Point p = (Point) iter.next();
               System.out.print(p);
        }
        System.out.println();
   }
}
```

HashSet Example 2

```
import java.util.*;
class Point {
```

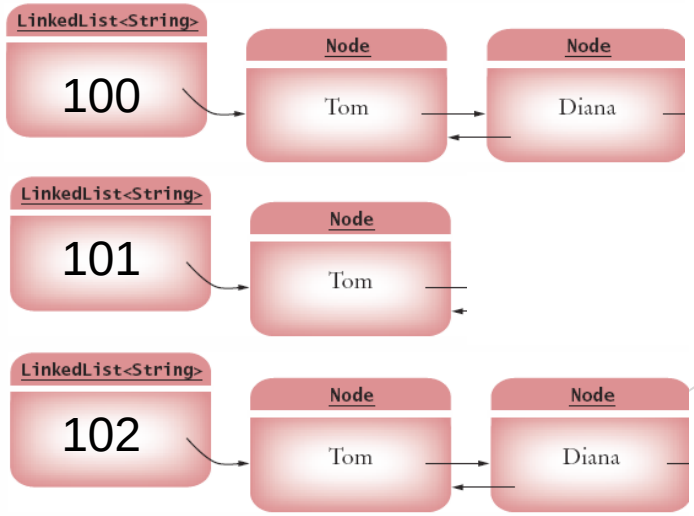
```
>>java EnsPt1
le point[1 3]
a ete ajouté
ensemble=[1 3]
le point[2 2]
a ete ajouté
ensemble=[2 2][1 3]
le point[1 3]
est déjà présent
ensemble=[2 2][1 3]
le point[4 5]
a ete ajouté
ensemble=[2 2][1 3][4 5]
le point[1 8]
a ete ajouté
ensemble=[2 2][1 3][1 8][4 5]
le point[4 5]
est déjà présent
ensemble=[2 2][1 3][1 8][4 5]
```

```
= p.y));
```

```
import java.util.*;
public class EnsPt1 {
    public static void main (String args[]) {
        Point p1 = new Point (1,3), p2 = new Point (2,2);
        Point p3 = new Point(4,5), p4 = new Point (1,8);

        Point p[] = {p1, p2, p1, p3, p4, p3};
        HashSet ens = new HashSet();
        for (int i=0;i<p.length;i++) {
              System.out.print("le point");
              System.out.println(p[i]);
              boolean ajoute = ens.add(p[i]);
              if (ajoute)
                   System.out.println("a ete ajouté");
              else System.out.println("est déjà présent");
              System.out.print("ensemble=");
              affiche(ens);
         }
    }

    public static void affiche (HashSet ens){
        Iterator iter = ens.iterator();
        while (iter.hasNext()) {
               Point p = (Point) iter.next();
               System.out.print(p);
        }
        System.out.println();
    }
}
```

Hash Table Concept

- Set elements are grouped into smaller collections of elements that share the same characteristic
  - It is usually based on the result of a mathematical calculation on the contents that results in an integer value
  - In order to be stored in a hash table, elements must have a method to compute their integer values



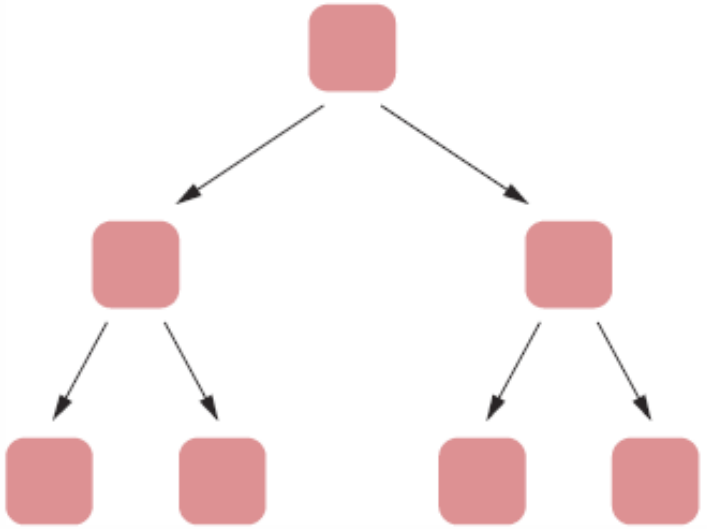
HashTable Example

```java
import java.util.Hashtable;
import java.util.Enumeration;

public class HashtableExample {

 public static void main(String[] args) {

    Enumeration names;
    String key;

    // Creating a Hashtable
    Hashtable<String, String> hashtable =
                new Hashtable<String, String>();

    // Adding Key and Value pairs to Hashtable
    hashtable.put("Key1","Chaitanya");
    hashtable.put("Key2","Ajeet");
    hashtable.put("Key3","Peter");
    hashtable.put("Key4","Ricky");
    hashtable.put("Key5","Mona");

    names = hashtable.keys();
    while(names.hasMoreElements()) {
       key = (String) names.nextElement();
       System.out.println("Key: " +key+ " & Value: " +
       hashtable.get(key));
    }
 }
}
```

```
>>java HashTableExample
Key: Key4 & Value: Ricky
Key: Key3 & Value: Peter
Key: Key2 & Value: Ajeet
Key: Key1 & Value: Chaitanya
Key: Key5 & Value: Mona
```

Tree Concept

- Set elements are kept in sorted order
  - Nodes are not arranged in a linear sequence but in a tree shape



  - In order to use a `TreeSet`, it must be possible to compare the elements and determine which one is "larger"

TreeSet

- Use `TreeSet` for classes that implement the `Comparable` interface
  - `String` and `Integer`, for example
  - The nodes are arranged in a ‘tree’ fashion so that each ‘parent’ node has up to two child nodes.
    - The node to the left always has a ‘smaller’ value
    - The node to the right always has a ‘larger’ value

```
Set<String> names = new TreeSet<String>();
```

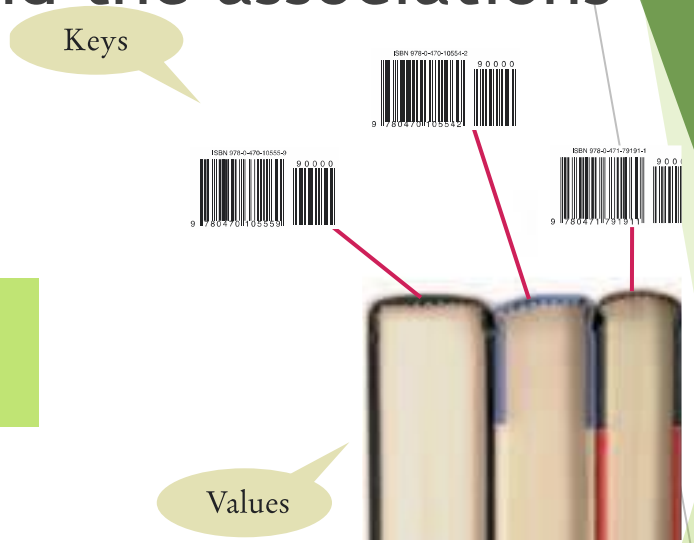
TreeSet Example

```
1   import java.util.*;
2   class TreeSet1{
3    public static void main(String args[]){
4     //Creating and adding elements
5     TreeSet<String> al=new TreeSet<String>();
6     al.add("Ravi");
7     al.add("Vijay");
8     al.add("Ravi");
9     al.add("Ajay");
10    //Traversing elements
11    Iterator<String> itr=al.iterator();
12    while(itr.hasNext()){
13     System.out.println(itr.next());
14    }
15   }
16  }
17
```

```
>>java TreeSet1
Ajay
Ravi
Vijay
```

Maps

- A map stores keys, values, and the associations between them
  - Example:
  - Barcode keys and books



A map keeps associations between key and value objects.

- Keys
  - Provides an easy way to represent an object (such as a numeric bar code)
- Values
  - The actual object that is associated with the key

Example of TreeMap

```
1  import java.util.* ;
2  public class ExempleTreeMap{
3      public ExempleTreeMap (){
4          TreeMap tree = new TreeMap () ;
5          tree.put ("Omar",new Integer (26));
6          tree.put ("Mohamed", new Integer (1)) ;
7          tree.put ("Ali", new Integer (2)) ;
8          Iterator itercle = tree.keySet().iterator() ;
9          Iterator itervaleurs = tree.values().iterator() ;
10         while (itercle.hasNext()) {
11             System.out.println (itercle.next() + " --> " + itervaleurs.next()) ;
12         }
13     }
14     public static void main(String[] args){
15         new ExempleTreeMap () ;
16     }
17 }
```

```
>>java ExempleTreeMap
Ali --> 2
Mohamed --> 1
Omar --> 26
```

Threads and Collections

- If multiple threads can access a collection object, there is a need to synchronize with one of the static methods of the java.util.Collections class:

```
static Collection synchronizedCollection (Collection c)
    static List synchronizedList (List list)
    static Map synchronizedMap (Map m)
    static Set synchronizedSet (Set s)
    static SortedMap synchronizedSortedMap (SortedMap m)
    static SortedSet synchronizedSortedSet (SortedSet s)
```

Remark: Previous methods do not synchronize iterators. So you have to do it manually:

```
synchronized (objet){
    Iterator iter = objet.iterator () ;
      {
          // Work with the iterator
      }
}
```

The Collection Algorithms

- The collections framework defines several algorithms that can be applied to collections and maps.
- These algorithms are defined as static methods within the Collections class.
  - **Sorting** (e.g. sort)
  - **Shuffling** (e.g. shuffle)
  - **Routine Data Manipulation** (e.g. reverse, addAll)
  - **Searching** (e.g. binarySearch)
  - **Composition** (e.g. frequency)
  - **Finding Extreme Values** (e.g. max)

Algorithms of Collection Interface

- The methods defined in collection framework's algorithm are summarized in the following link

  https://www.tutorialspoint.com/java/java_collection_algorithms.htm

- **static void copy(List list1, List list2)** Copies the elements of list2 to list1.
- **static Object max(Collection c)** Returns the maximum element in **c** as determined by natural ordering. The collection need not be sorted.
- **static Object max(Collection c, Comparator comp)** Returns the maximum element in **c** as determined by comp.
- **static void reverse(List list)** Reverses the sequence in list.
- **static void shuffle(List list)** Shuffles (i.e., randomizes) the elements in list.
- **static void sort(List list)** Sorts the elements of the list as determined by their natural ordering.
- **static void sort(List list, Comparator comp)** Sorts the elements of list as determined by comp.

Working with Collection Algorithms

```
1  import java.util.*;
2  public class Tri {
3      public static void main(String[] args) {
4      int nb [] = {4,5,2,1,6,8,3};
5      ArrayList t = new ArrayList();
6          for (int i=0;i<nb.length;i++) t.add(new Integer(nb[i]));
7          System.out.println("t initial="+t);
8
9          Collections.sort(t);
10         System.out.println("t trié="+t);
11
12         Collections.shuffle(t);
13         System.out.println("t mélangé="+t);
14
15         Collections.sort(t, Collections.reverseOrder()); // un comparateur prédéfini
16         System.out.println("t trié="+t);
17     }
18 }
```

```
>>java Tri
t initial=[4, 5, 2, 1, 6, 8, 3]
t trié=[1, 2, 3, 4, 5, 6, 8]
t mélangé=[2, 4, 3, 8, 1, 6, 5]
t trié=[8, 6, 5, 4, 3, 2, 1]
```

Interface Comparable

- Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only.

- **compareTo(Object obj) method**

public int compareTo(Object obj): It is used to compare the current object with the specified object.

It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

Example 1 of the method compareTo

```
class Student implements Comparable<Student>{
    int cne;
    String name;
    int age;
    Student(int rollno,String name,int age){
    this.cne=rollno;
    this.name=name;
    this.age=age;
    }

    public int compareTo(Student st){
    if(age==st.age)
    return 0;
    else if(age>st.age)
    return 1;
    else
    return -1;
    }
    }
```

```
import java.util.*;
public class TestSort1{
public static void main(String args[]){
ArrayList<Student> al=new ArrayList<Student>();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

Collections.sort(al);
for(Student st:al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

```
105 Jai 21
101 Vijay 23
106 Ajay 27
```

Example 2 of the method compareTo

```
import java.util.*;

public class EmployeC implements Comparable{
        int id;
        EmployeC (int i){
            id = i;
       }
       public int compareTo(Object o) {
              if ((this.id) < (((EmployeC)(o)).id))
                     return -1;
              else if ((this.id) > (((EmployeC)(o)).id))
                       return 1;
                    else
                       return 0;
      }
      void imprimer(){
               System.out.println("EmployeC "+id);
      }
      public String toString(){
                return "EmployeC :"+id;
      }
}
```

Example 2 of the method compareTo

```
class Main_EmployeC1 {
    public static void main(String[] args) {
            ArrayList c= new ArrayList();
            Random r = new Random();
            for (int i=0;i<10;i++) c.add(new EmployeC (r.nextInt(100)));
            for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));
            System.out.println("max " + Collections.max(c));
           Collections.sort(c);
            for(int i=0; i<c.size();i++) System.out.println(i + " " + c.get(i));
    }
}
```

Exemple : Méthode **compareTo**

```
class Main_EmployeC1 {
    public static void main(String[] args) {
            ArrayList c= new ArrayList();
            Random r = new Random();
            for (int i=0;i<10;i++) c.add(new EmployeC (r.nextInt(100)));
            for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));
            System.out.println("max " + Collections.max(c));
           Collections.sort(c);
            for(int i=0; i<c.size();i++) System.out.println(i + " " + c.get(i));
    }
}
```

```java
import java.util.*;

public class Employe {
      int id;
      int salaire ;
      Employe (int i, int s){
         id = i;
         salaire =s;
      }
      void imprimer(){
          System.out.println("Employe "+id +" salaire "+salaire);
      }
      public String toString(){
          return "Employer :"+id+" salaire "+salaire;
      }
}
```

```
import java.util.Comparator;
class EmployeComparator implements Comparator{
    public int compare (Object o1, Object o2){
      if ((((Employe)(o1)).salaire) < (((Employe)(o2)).salaire)) return -1;
      else if ((((Employe)(o1)).salaire) > (((Employe)(o2)).salaire)) return 1;
        else return 0;
    }
}
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Random;

class Main_Employe {
    public static void main(String[] args) {
        ArrayList c= new ArrayList();
        Random r= new Random();
        for (int i=0;i<10;i++) c.add(new Employe(i,r.nextInt(5000)));
        for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));

        Comparator comp = new EmployeComparator();
        System.out.println("max " + Collections.max(c,comp));
        Collections.sort(c,comp);
        for(int i=0; i<c.size();i++)
        System.out.println(i + " " + c.get(i));

    }
}
```

Résultats :

```
0 Employer :0 salaire 210
1 Employer :1 salaire 4785
2 Employer :2 salaire 3696
3 Employer :3 salaire 843
4 Employer :4 salaire 1633
5 Employer :5 salaire 2723
6 Employer :6 salaire 582
7 Employer :7 salaire 1398
8 Employer :8 salaire 629
9 Employer :9 salaire 2581
max Employer :1 salaire 4785
10 Employer :0 salaire 210
1 Employer :6 salaire 582
2 Employer :8 salaire 629
3 Employer :3 salaire 843
4 Employer :7 salaire 1398
5 Employer :4 salaire 1633
6 Employer :9 salaire 2581
7 Employer :5 salaire 2723
8 Employer :2 salaire 3696
9 Employer :1 salaire 4785
```

Combinaison des deux : Tri sur différents critères

```
import java.util.*;
class Employe implements Comparable{i
        nt id;
        String nom;
        float  salaire;
        Employe (int i, String N, float S){
              id=i;
              nom = N;
              Salaire = S;
        }
        public int compareTo(Object o) {
              if this.salaire<((Employe)(o).salaire) return -1;
             else  if this.salaire>((Employe)(o).salaire) return 1;
             else return 0;
        }
        void imprimer(){
              System.out.println("Employe : "+nom);
        }
        public String toString(){
              return "Employé -- id : "+id+ " nom : "+ nom+ " salaire : ''+salaire;
        }
}
```

```
class EmployeComparator implements Comparator{
       public int compare (Object o1, Object o2){
          return (Employe)(o1).nom.equals((Employe)(o2).nom);
        }
}
```

```
public class Main_Employe_2 {
public static void main(String[] args) {
            ArrayList  c= new ArrayList ();
            Random r  = new Random();
            for (int i=0;i<10;i++)
            c.add(new Employe(r.nextInt(100),"empoloye"+r.next(100), r.nextFtoat()));f
            or(int i=0; i<c.size();i++)
           System.out.println(i  + " "  +((Employe) c.get(i)).nom+":"+((Employe)

            Comparator  comp = new EmployeComparator();
           System.out.println("max comparateur" + Collections.max(c,comp));
           Collections.sort(c,comp);
           for(int i=0; i<c.size();i++)
           System.out.println(i  + " "  +((Employe) c.get(i)).nom+":"+((Employe)

        System.out.println("max comparable" + Collections.max(c));
         Collections.sort(c);
         for(int i=0; i<c.size();i++)
                    System.out.println(i  + " "  +((Employe) c.get(i)).nom+":"+((Employe)

    }
}
```

c

c

Exemple : Méthode **compareTo**

```
class Main_EmployeC1 {
    public static void main(String[] args) {
            ArrayList c= new ArrayList();
            Random r = new Random();
            for (int i=0;i<10;i++) c.add(new EmployeC (r.nextInt(100)));
            for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));
            System.out.println("max " + Collections.max(c));
           Collections.sort(c);
            for(int i=0; i<c.size();i++) System.out.println(i + " " + c.get(i));
    }
}
```

```
import java.util.Comparator;
class EmployeComparator implements Comparator{
    public int compare (Object o1, Object o2){
      if ((((Employe)(o1)).salaire) < (((Employe)(o2)).salaire)) return -1;
      else if ((((Employe)(o1)).salaire) > (((Employe)(o2)).salaire)) return 1;
        else return 0;
    }
}
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Random;

class Main_Employe {
    public static void main(String[] args) {
        ArrayList c= new ArrayList();
        Random r= new Random();
        for (int i=0;i<10;i++) c.add(new Employe(i,r.nextInt(5000)));
        for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));

        Comparator comp = new EmployeComparator();
        System.out.println("max " + Collections.max(c,comp));
        Collections.sort(c,comp);
        for(int i=0; i<c.size();i++)
        System.out.println(i + " " + c.get(i));

    }
}
```

```
import java.util.*;

public class Employe {
      int id;
      int salaire ;
      Employe (int i, int s){
         id = i;
         salaire =s;
      }
      void imprimer(){
          System.out.println("Employe "+id +" salaire "+salaire);
      }
      public String toString(){
          return "Employer :"+id+" salaire "+salaire;
      }
}
```

Résultats :

```
0 Employer :0 salaire 210
1 Employer :1 salaire 4785
2 Employer :2 salaire 3696
3 Employer :3 salaire 843
4 Employer :4 salaire 1633
5 Employer :5 salaire 2723
6 Employer :6 salaire 582
7 Employer :7 salaire 1398
8 Employer :8 salaire 629
9 Employer :9 salaire 2581
max Employer :1 salaire 4785
10 Employer :0 salaire 210
1 Employer :6 salaire 582
2 Employer :8 salaire 629
3 Employer :3 salaire 843
4 Employer :7 salaire 1398
5 Employer :4 salaire 1633
6 Employer :9 salaire 2581
7 Employer :5 salaire 2723
8 Employer :2 salaire 3696
9 Employer :1 salaire 4785
```

Combinaison des deux : Tri sur différents critères

```
import java.util.*;
class Employe implements Comparable{i
        nt id;
        String nom;
        float  salaire;
        Employe (int i, String N, float S){
              id=i;
              nom = N;
              Salaire = S;
        }
        public int compareTo(Object o) {
              if this.salaire<((Employe)(o).salaire) return -1;
             else  if this.salaire>((Employe)(o).salaire) return 1;
             else return 0;
        }
        void imprimer(){
              System.out.println("Employe : "+nom);
        }
        public String toString(){
              return "Employé -- id : "+id+ " nom : "+ nom+ " salaire : ''+salaire;
        }
}
```

```
class EmployeComparator implements Comparator{
       public int compare (Object o1, Object o2){
          return (Employe)(o1).nom.equals((Employe)(o2).nom);
        }
}
```

```
public class Main_Employe_2 {
public static void main(String[] args) {
            ArrayList  c= new ArrayList ();
            Random r  = new Random();
            for (int i=0;i<10;i++)
            c.add(new Employe(r.nextInt(100),"empoloye"+r.next(100), r.nextFtoat()));f
            or(int i=0; i<c.size();i++)
           System.out.println(i  + "  " +((Employe) c.get(i)).nom+":"+((Employe)

            Comparator  comp = new EmployeComparator();
           System.out.println("max comparateur" + Collections.max(c,comp));
           Collections.sort(c,comp);
           for(int i=0; i<c.size();i++)
           System.out.println(i  + "  " +((Employe) c.get(i)).nom+":"+((Employe)

        System.out.println("max comparable" + Collections.max(c));
         Collections.sort(c);
         for(int i=0; i<c.size();i++)
                    System.out.println(i  + "  " +((Employe) c.get(i)).nom+":"+((Employe)


    }
}
```

c

c

Exercise 1 – Word Frequency

- Write a Java program that calculates the frequency of each word present in an input file text.

Exercise 2 – Character Frequency

- Write a Java program that computes the frequency of each character present in an input file text.