JAVA Collections Framework

- https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html

The collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details.

- **Reduces programming effort** by providing data structures and algorithms so you don't have to write them yourself.
- **Increases performance** by providing high-performance implementations of data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be tuned by switching implementations.
- **Fosters software reuse** by providing a standard interface for collections and algorithms with which to manipulate them.

compact1, compact2, compact3
java.util

Interface Collection<E>

Type Parameters:
`E` - the type of elements in this collection

All Superinterfaces:
`Iterable<E>`

All Known Subinterfaces:
`BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>`

All Known Implementing Classes:
`AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector`

JAVA Collections Framework

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

…(r)esistono ancora Vector, HashTable

ArrayList vs Vector

ArrayList and Vector both implements List interface and maintains insertion order.

However, there are many differences between ArrayList and Vector classes that are given below.

ArrayList

ArrayList is **not synchronized**.

ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity.

ArrayList is **not a legacy** class. It is introduced in JDK 1.2.

ArrayList is **fast** because it is non-synchronized.

ArrayList uses the **Iterator** interface to traverse the elements.

Vector

Vector is **synchronized**.

Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity.

Vector is a **legacy** class.

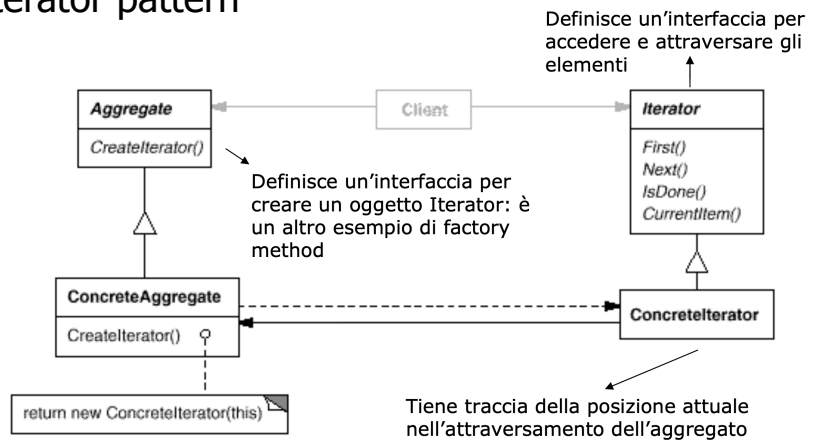
Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads until current thread releases the lock of the object.

A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements.

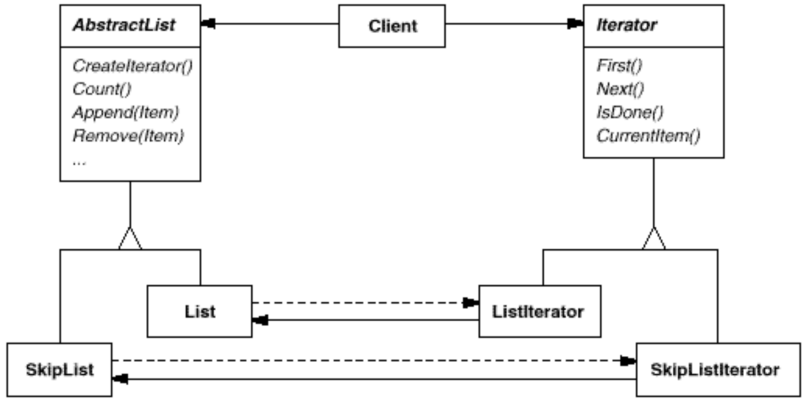
Pattern Iterator

- Attraversare gli elementi di un aggregato senza esporre la sua struttura
- l'iterator conosce i dettagli della classe che visita, mentre al client restano nascosti
- E' possibile definire più *concrete iterator* che visitano lo stesso contenitore con politiche diverse
- Ad esempio strutture complesse come grafi o alberi
- Sia Java che la STL del linguaggio C++ fanno ampio uso del pattern iterator

Iterator pattern



Iterator example



Vantaggi dell' iteratore

```
void print(ArrayList list)
   {
   for( int k = 0; k < list.size(); k++ )
      System.out.println( list.get(k) );
   }

void print(LinkedList list )
   {
   Node current = list.first();
   System.out.println( current );
   while (current.hasNext() )
      {
      current = current.next();
      System.out.println( current );
      }
   }

//Contro…

void print(Collection list )
   {
   Iterator items = list.iterator();
   while (items.hasNext() )
      {
      System.out.println( items.next() );
      }
   }
```

- Iterator abstracts out underlying representation of collection
- Programmer does not have to know implementation details of each type of collection
- Can write code that works for wide range of collects
- Do not have to change code if change the type of collection used

Iterator pattern

- Fornisce un modo per accedere agli elementi di un oggetto aggregato in maniera **sequenziale**, senza esportarne la rappresentazione interna
- Idea: togliere le funzionalità di accesso e attraversamento dall'aggregato e inserirle all'interno di un oggetto **iterator**
- Un oggetto iterator è responsabile di tener traccia dell'**elemento corrente**
  - Può anche tener **traccia** di quali elementi sono già stati attraversati
- E' possibile utilizzare contemporaneamente **più iterator**i sulla stessa struttura dati
  - L'iteratore ha **accesso privilegiato** agli elementi dell'aggregato, senza esporre la loro interfaccia al client
  - Evita di appesantire l'interfaccia dell'aggregato con operazioni per i **diversi tipi di attraversamento** (ad. es. reverse o filtrato)
  - Svincola il codice del client dall'implementazione della str. dati: l'aggregato è visto come astratto
    - Il codice del **client** diventa facilmente **riusabile** in quanto generico

Ways to traverse collections in Java:

1. By using Iterable interface
2. By using Iterator object
3. By using ListIterator object (List only)
4. By using Enumeration (legacy only)
5. By using for-each
6. By using for loop

Property	Enumeration	Iterator	ListIterator
Applicable for	Only legacy classes	Any Collection classes	Only List classes
Movement	Only forward direction(single direction)	Only forward direction(single direction)	Both forward and backward direction(bi directional)
Accessibility	Only read access	Both read and remove	Read ,remove, replace and addition of new objects
How to get it?	By using elements() method of Vector class	By using iterator() method of Collection interface	By using listIterator() method of List interface
Methods	2 methods hasMoreElements() nextElement()	3 methods hasNext () next() remove()	9 methods
Is it legacy	"yes" (1.0v)	"no" (1.2V)	"no" (1.2V)

Enumeration, Iterator, and Streams in Java are iterators

```
List listOfStrings = new Vector<String>();

//con for loop tradizionale…
        for (int i=0; i<listOfStrings.size();i++)
             System.out.println(listOfStrings.get(i));

//con Iterable interface…

        Iterable<String> sameList = listOfStrings;
        for(String s: sameList) System.out.println(s);

//con Iterator hasNext()…

        Iterator<String> list=listOfStrings.iterator();
        while (list.hasNext()) {
            String x = list.next();
            System.out.println(x);
        }


codice su visualstudio…
```

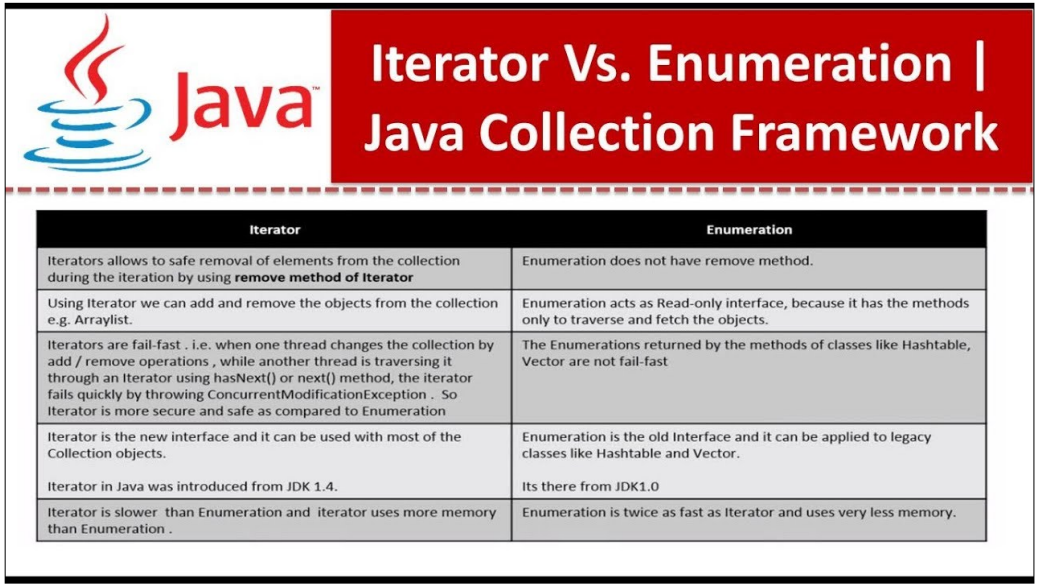
- An Iterable is a simple representation of a series of elements that can be iterated over.
- It does not have any iteration state such as a "current element".
- Instead, it has one method that produces an Iterator.
- An Iterator is the object with iteration state.

Enumeration, Iterator, and Streams in Java are iterators

Import java.util.stream
// definisce Interface Stream<T>

```
//con Enhanced for loop (iteratore implicito)…

        for(Studente s: listOfStrings)  System.out.println(s);

//con metodo forEach dell'interfaccia Iterable…

        listOfStrings.forEach(System.out::println);

//con metodo forEach() degli streams…

        listOfStrings.stream().forEach((c) -> System.out.println(c));
```

metodo stream() definito nell'interfaccia Collection torna uno stream



Iterator	Enumeration
Iterators allows to safe removal of elements from the collection during the iteration by using **remove method of Iterator**	Enumeration does not have remove method.
Using Iterator we can add and remove the objects from the collection e.g. Arraylist.	Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects.
Iterators are fail-fast . i.e. when one thread changes the collection by add / remove operations , while another thread is traversing it through an Iterator using hasNext() or next() method, the iterator fails quickly by throwing ConcurrentModificationException . So Iterator is more secure and safe as compared to Enumeration	The Enumerations returned by the methods of classes like Hashtable, Vector are not fail-fast
Iterator is the new interface and it can be used with most of the Collection objects. Iterator in Java was introduced from JDK 1.4.	Enumeration is the old Interface and it can be applied to legacy classes like Hashtable and Vector. Its there from JDK1.0
Iterator is slower than Enumeration and iterator uses more memory than Enumeration .	Enumeration is twice as fast as Iterator and uses very less memory.