# *Collection Framework in Java 5*

Angelo Gargantini

# Array per tipi primitivi

- Fino ad ora quando dovevamo costruire un array di interi o di oggetti usavamo il costrutto degli array
  - gli array sono oggetti e le variabili array saranno riferimenti
- Per creare un array di 20 interi

```
int nomi[] = new int[20];
```

- se dobbiamo inserire 21 interi in nomi dobbiamo ridimensionare l'array nomi. Si può fare in questo modo:

```
int temp[21] = new int[21];
// copio i 20 interi da nomi a temp TODO
temp[20] = 4;
nomi = temp;
```

# Array di oggetti

- Gli array di oggetti sono oggetti e ogni elemento è un riferimento ad un oggetto
- Per creare un array di Auto:

```
Auto deposito[] = new Auto[40];
```

- Crea 40 riferimenti a Auto (inizializzati a `null`)
- Per ridimensionare il deposito devo fare come nome
- Devo tener traccia di quanti oggetti effetivamente ho presente
  - con possibili “buchi” (riferimenti a null seguiti da riferimenti a non null)
  - con possibile spreco di spazio

# Uso di Vector

- classe `Vector` del package `java.util`
- un oggetto Vector è concettualmente simile ad un array, ma a differenza di quest'ultimo prevede un utilizzo esclusivamente Object Oriented

1. Creazione
   ```
   Vector v = new Vector();
   ```
2. Aggiungere elementi
   metodo `add(Object o)`,
   che aggiunge in coda al vettore l'oggetto passato come parametro:

```
String nome = "Un nome"; v.add(nome);
```

## Specificare il tipo degli elementi in Java 5

- Fino a Java 5 i Vector contengono Object
- In Java 5 si può specificare di che tipo sono gli elementi del vettore.
- Per un vettore di stringhe:

```
Vector<String> v = new Vector<String>();
v.add("Un nome");
```

- In `v` posso inserire solo `String` (o sottoclassi)
- A differenza degli array, un Vector
  - ha dimensione dinamica: posso continuare ad aggiungere elementi senza preoccuparmi di creare spazio
  - (se usato correttamente) non “ha buchi”: gli elementi sono tutti dalla posizione 0 alla dimensione dell’array -1

# Metodi Fondamentali di Vector<T>

- `boolean add(T o)`: aggiunge l'elemento in coda al vettore
- `void add(int index , T element)`: aggiunge un elemento nella posizione specificata; gli elementi successivi vengono spostati in avanti di uno.
- `void clear()`: svuota completamente il vettore.
- `boolean isEmpty()`: restituisce true se il vettore è vuota
- `T get(int i)`: restituisce l'i-esimo elemento
- `int indexOf(T o)`: restituisce l'indice dell'elemento passato come parametro, o -1 se l'elemento non è presente nella lista.

# Metodi Fondamentali di Vector<T>

- `T remove(int i)`: rimuove l'I-esimo elemento della lista, e sposta tutti gli elementi successivi. Il metodo restituisce l'elemento appena rimosso.
- `T set(int i , T element)`: mette l'elemento specificato in I-esima posizione, in sostituzione dell'elemento preesistente. Il metodo restituisce l'elemento appena rimosso.
- `int size()`: restituisce la dimensione dell'array: numero degli elementi effettivamente inseriti
- `int capacity()`: quanti elementi può contenere (prima si essere nuovamente allargato)

## Scandire vector

- Per effettuare una determinata operazione su tutti gli elementi di un vettore dinamico, è possibile ricorrere ad un ciclo for come nel caso di un array:

```
for(int i=0;i<v.size();i++)
      System.out.println(v.get(i));
```

- Oppure usare un apposito oggetto denominato `Iterator`

# Iterator

- L'oggetto `Iterator` può essere prelevato dal Vector usando un apposito metodo:

```
Iterator i = nomi.iterator();
```

- L'Iterator permette di scandire l'array dal primo all'ultimo elemento allo scopo di effettuare una determinata operazione su ciascuno dei suoi elementi. L'utilizzo canonico di un Iterator ha la forma:

```
while(i.hasNext()){
   // hasNext():  c'è un ulteriore elemento?
   // preleva l'elemento e lo utilizza
   System.out.println(i.next());
}
```

## uso di for each

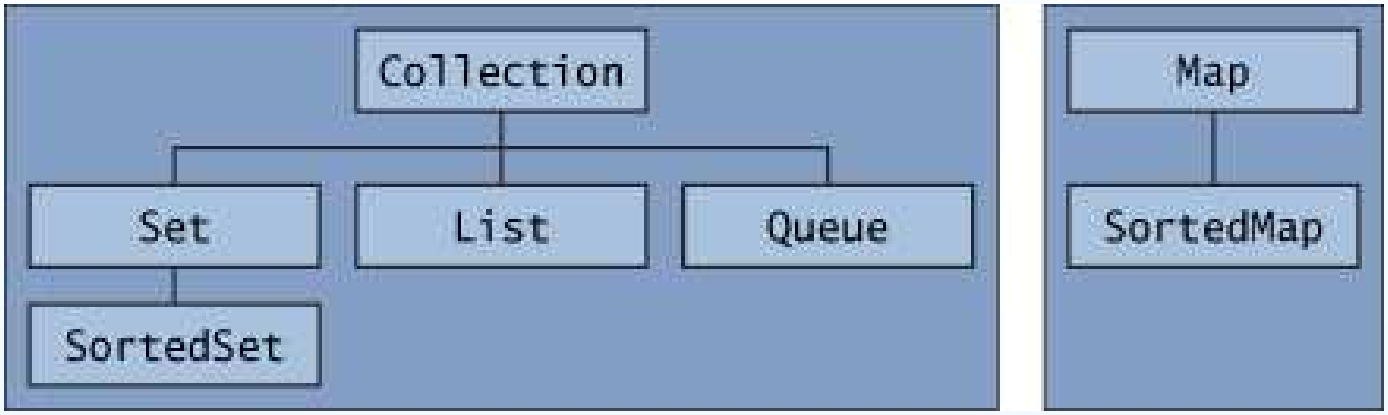
- Invece che l'iterator, Java 5 propone l'uso di for each

```
for(<TIPO> <varibile>:  <Iterable<TIPO>>)
<ISTRUZIONE>
```

- Vector<T> è un Iterable<T> (anche T[])

```
Vector<String> nomi;
...
for (String s:  nomi) System.out.println(s);
```

# Java Collection Framework: Interfaces




**Collection** the root of the collection hierarchy. A collection represents a group of objects known as its elements.

**Set** a collection that cannot contain duplicate elements.

**List** an ordered collection (sometimes called a sequence).

**Queue** additional insertion, extraction, and inspection operations.

**Map** an object that maps keys to values.

**SortedSet** a Set that maintains its elements in ascending order.

**SortedMap** a Map that maintains its mappings in ascending key order.

# Implementations

| Interfaces | Implementazioni | | | |
|---|---|---|---|---|
| | hash table | Resizable array | Tree | Linked list |
| Set | **HashSet** | | **TreeSet** | |
| List | | **ArrayList** | | **LinkedList** |
| Queue | | | | |
| Map | **HashMap** | | **TreeMap** | |

Oppure Vector ... che però sono più complessi

# Algorithms*

**Sorting** List<String> l = Arrays.asList(args); Collections.sort(l);

**Shuffling**

**Routine Data Manipulation**

**Searching**

**Composition**

**Finding Extreme Values**

# Esercizi