COLLEZIONI IN JAVA: ArrayList

APPUNTI PRATICI PER L'UTILIZZO DELLE ARRAY LIST

1. Introduzione

JAVA, come tutti i linguaggi di programmazione OOP, rende disponibili specifiche classi predefinite per la gestione di **contenitori** di oggetti. In JAVA **un contenitore di oggetti** (che sia una lista, una coda, un albero..) **è chiamato COLLEZIONE di oggetti (Collection).** Le classi delle collezioni JAVA espongono i metodi, già pronti e sicuri, per operare su di esse, ossia per aggiungere elementi, eliminarli ecc..

Una delle cose più importanti è che tali collezioni, accettano come **parametro** il **tipo di oggetto che devono contenere** (o meglio la classe degli oggetti che devono contenere). Quindi se vogliamo realizzare una collezione di oggetti di classe Libro basta istanziare un oggetto "collezione" e passare come parametro la classe Libro. Se vogliamo una collezione di Prodotti basta passare all'oggetto "collezione" la classe Prodotto e così via…

Le classi Collezione più utilizzate in Java sono la classe ArrayList **e la classe** LinkedList.

In questi appunti studiamo l'ArrayList. Gli indici dell'ArrayList iniziano da 0 come per gli array.

Quale è il vantaggio dell'ArrayList rispetto agli array studiati fin'ora?

1. Abbiamo visto che per gli array è necessario definirne la dimensione quando vengono istanziati. Tale dimensione può essere definita a runtime, però, nel caso si voglia incrementare tale dimensione, è necessario istanziare un nuovo array più grande. Questo ridimensionamento della dimensionen nell'ArrayList, non è necessario poiché l'ArrayList svolge il ridimensionamento dell'array in maniera automatica, trasparente al programmatore.

2. L'ArrayList espone dei metodi già pronti per le operazioni di inserimento ed eliminazione di elementi. Quando si elimina un elemento da un ArrayList non è dunque necessario "sistemare" gli elementi "spostando indietro" quelli successivi all'elemento eliminato.

A cosa bisogna porre attenzione nell'utilizzo degli ArrayList?

1. Come per gli array già visti, anche negli ArrayList i singoli elementi devono essere tutti della stessa classe, ma negli ArrayList tali elementi non possono essere tipi di dato nativi ma solamente oggetti. Per creare ArrayList di tipi di dato nativi è necessario incapsularne i dati nelle relative classi Wrapper.

2. La sintassi per istanziare ed utilizzare un ArrayList è diversa rispetto a quella per gli array visti fin'ora, e va imparata.

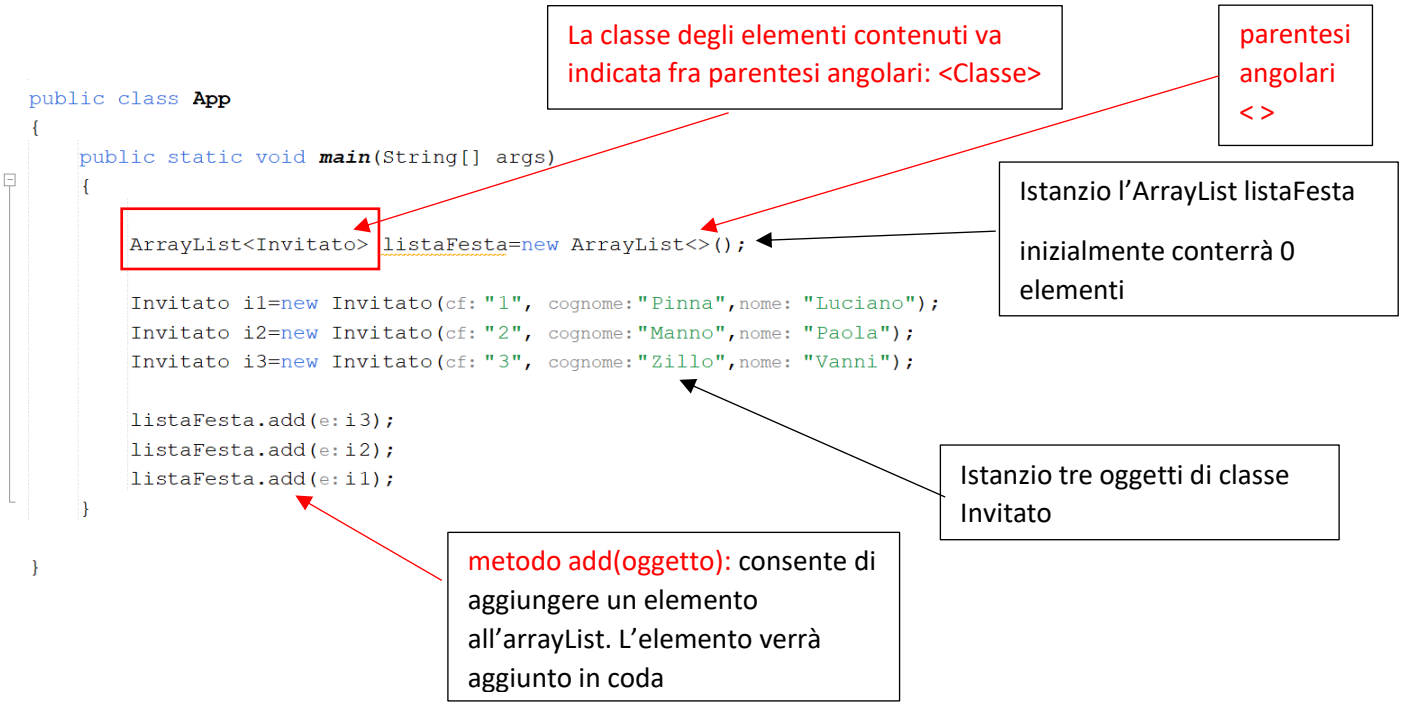
2. Esempio guidato per l'utilizzo degli ArrayList (metodi principali)

Si vuole creare un ArryList chiamato "listaFesta" che contiene un elenco di invitati ad una festa.

A. Creo la classe "Invitato" con attributi codiceFiscale, Cognome, Nome, metodi getter, setter, costruttore di copia e toString. (il codice di questa classe non viene qui riportato)

B. Nella classe App istanzialo l'ArrayList "listaFesta" e vediamo i metodi principali che esso espone:

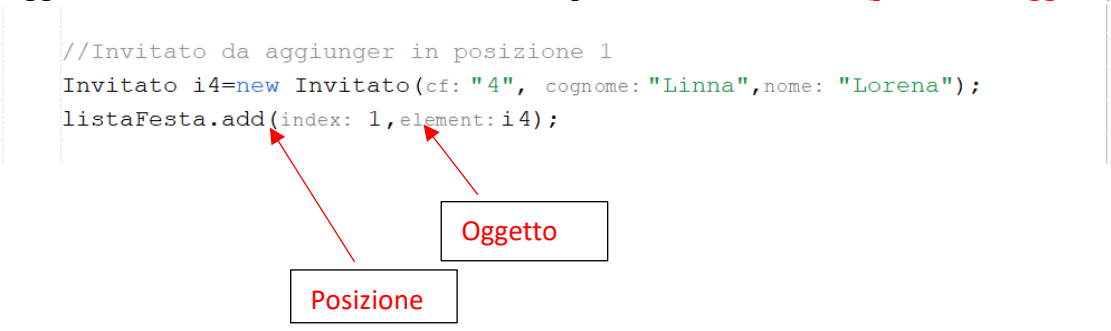
- Istanziazione e metodo add(oggetto)




```
public class App
{
    public static void main(String[] args)
    {
        ArrayList<Invitato> listaFesta=new ArrayList<>();

        Invitato i1=new Invitato(cf: "1", cognome:"Pinna",nome: "Luciano");
        Invitato i2=new Invitato(cf: "2", cognome:"Manno",nome: "Paola");
        Invitato i3=new Invitato(cf: "3", cognome:"Zillo",nome: "Vanni");

        listaFesta.add(e:i3);
        listaFesta.add(e:i2);
        listaFesta.add(e:i1);
    }

}
```

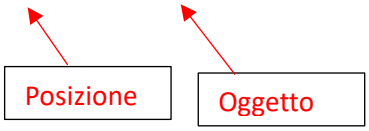
OSSERVAZIONE: gli elementi aggiunti alla lista non sono delle copie, quindi non c'è indipendenza degli oggetti. E' il programmatore che deve eventualmente occuparsi di garantire tale indipendenza aggiungendo all' ArrayList una copia dell'oggetto anzichè l'oggetto.

- Aggiunta di un elemento in una determinata posizione: metodo add(posizione, oggetto)



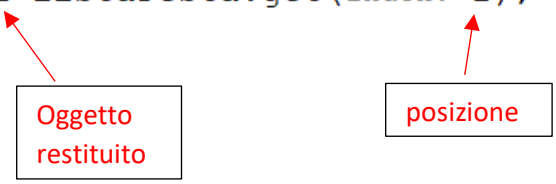
- Set di un elemento in una determinata posizione (setter): metodo set(posizione, oggetto)
  Questo metodo **sostituisce** un elemento già presente nella posizione indicata.
  Se la posizione non esiste viene sollevata un'eccezione IndexOutOfBoundException.

```
Invitato i5=new Invitato(cf: "4", cognome: "Pesce",nome: "Udlerico");
listaFesta.set(index: 1,element: i5);
```



- Get di un elemento in una determinata posizione (getter): metodo get(posizione)

```
Invitato i=listaFesta.get(index: 1);
```



Per ottenere il primo e l'ultimo elemento della lista esistono i metodi getFirst e getLast.

- Restituzione della posizione della prima occorrenza di un oggetto: metodo indexOf (reference dell'oggetto)

```
int posizioneOggettoi2=listaFesta.indexOf(o: i2);
```

Se l'oggetto non è presente, il metodo restituisce -1.

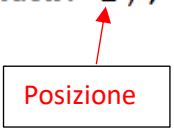
- Restituzione della posizione dell'ultima occorrenza di un oggetto: metodo lastIndexOf(oggetto)

```
int ultimaPosizioneOggettoi2=listaFesta.lastIndexOf(o: i2);
```

Se l'oggetto non è presente, il metodo restituisce -1.

- Eliminazione dell'oggetto che si trova in una determinata posizione: metodo boolean remove (posizione)
  Gli oggetti che presenti, successivi all'elemento eliminato, vengono "spostati in dietro" di una posizione automaticamente.
  Se la posizione non esiste viene sollevata un'eccezione IndexOutOfBoundException.

```
listaFesta.remove(index: 2);
```



- Eliminazione di un oggetto: metodo boolean remove (oggetto)
  Se l'oggetto da rimuovere non è presente, il metodo restituisce false, se l'oggetto viene rimosso il metodo restituisce true.

```
System.out.println(x: listaFesta.remove(o: i5));
```

- Eliminazione di tutti gli elementi dalla collezione: metodo clear()

```
listaFesta.clear();
```

- Verifica della presenza di un elemento nella collezione: metodo boolean contains(oggetto)
  Se l'oggetto è presente restituisce true, altrimenti false

```
System.out.println(x: listaFesta.contains(o: i5));
```

- Verifica del fatto che una collezione sia vuota metodo boolean isEmpty()
  Se la collezione è vuota restituisce true, altrimenti false

```
System.out.println(x: listaFesta.isEmpty());
```

- Restituzione del numero di elementi presenti nella collezione: metodo int size()

```
System.out.println(x: listaFesta.size());
```

- Esportazione degli elementi della collezione in un array (array "tradizionale"): metodo toArray (Invitato arrayInvitati[ ])

```
Invitato[] arrayInvitati=new Invitato[listaFesta.size()];
listaFesta.toArray(a: arrayInvitati);
```

Istanzio un array di invitati della dimensione giusta

Esporto gli elementi dall'ArrayList all'Array

- Mostrare il contenuto di un ArrayList: metodo toString()
  Diversamente dagli array, per mostrare gli oggetti contenuti nell'ArrayList non è necessario scorrere tutti gli elementi presenti ed invocare il metodo toString di ciascun elemento. Questa operazione viene svolta automaticamente dal metodo toString dell'ArrayList. Naturalmente nella classe che costituisce gli elementi contenuti nell'ArrayList (nel nostro caso Invitato) deve essere opportunamente definito il metodo toString().

```
System.out.println("Elenco invitati:\n"+listaFesta.toString());
```

Risultato:

```
Elenco invitati:
[Invitato{cf=3, cognome=Zillo, nome=Vanni}, Invitato{cf=4, cognome=Pesce, nome=Udlerico}, Invitato{cf=2, cognome=Man
no, nome=Paola}, Invitato{cf=1, cognome=Pinna, nome=Luciano}]
```

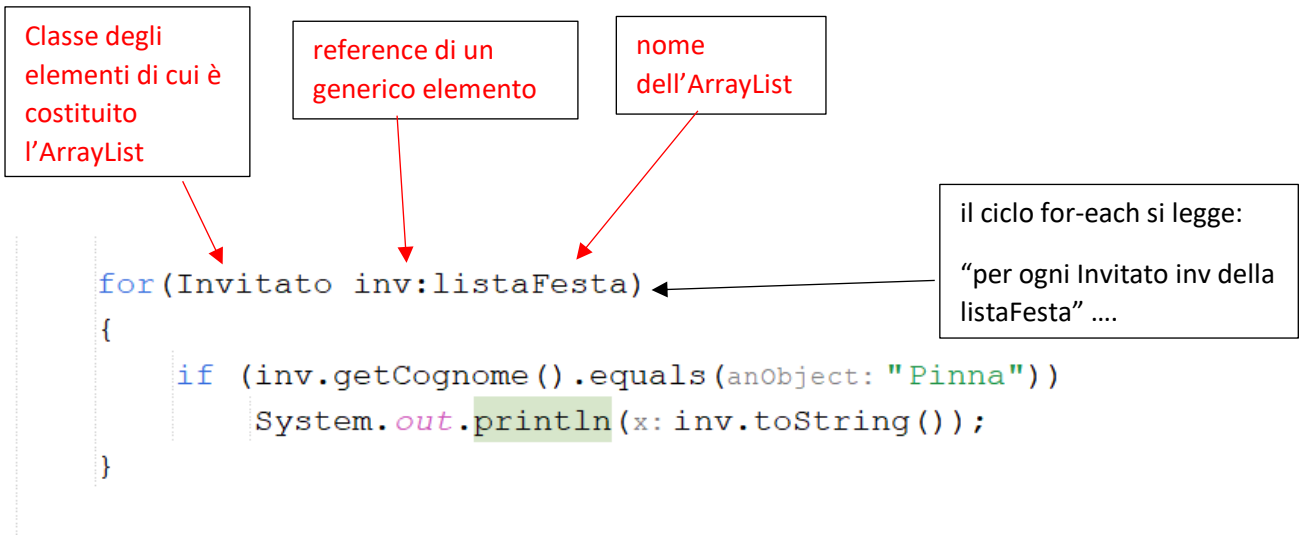
Riassumendo, l'elenco dei metodi visti è:

add (oggetto), add (posizione, oggetto)
set (posizione, oggetto)
get(posizione)
indexOf(oggetto) lastIndexOf(oggetto)
boolean remove (posizione) boolean remove (oggetto)
clear ()
boolean contains()
boolean isEmpty()
int size()
toArray(Oggetto a[])
toString()

3. Ciclo for-each per “scorrere” gli elementi di un ArrayList

Oltre al normale ciclo for per scorrere gli elementi di un ArrayList è possibile utilizzare un’altra sintassi chiamata ciclo **for-each** (per-ogni).

La sintassi del ciclo for-each è mostrata nel seguente codice in cui si “scorrono gli elementi dell’ArrayList “listaFesta” per cercare gli invitati con cognome “Pinna” e stamparli a video:



4. Ordinare un ArrayList

Come per qualsiasi collezione di oggetti, anche per un array list potrebbe essere necessario ordinare gli elementi secondo un qualsiasi criterio di ordinamento, ad esempio per ordine alfabetico di cognome e nome, oppure per ordine alfabetico di codice fiscale, e via dicendo. I possibili criteri di ordinamento sono infiniti. Come si fa ad ordinare un array list specificando il criterio in base al quale si vuole effettuare l'ordinamento?

E' necessario **creare diverse classi** (una per ogni criterio di comparazione) ciascuna delle quali implementa l'interfaccia Comparator. All'interno di ciascuna classe viene stabilito il criterio di comparazione e infine, un'istanza della classe viene passata come parametro al metodo **sort** della **classe Collection**.

Ad esempio, se vogliamo ordinare l'ArrayList listaFesta in base al cognome e al nome degli invitati creiamo la seguente classe "ComparatorCognomeNome"

Interfaccia Comparator (da importare) con parametro la classe degli elementi da comparare (nel nostro caso Invitato)

```
13  public class ComparatorCognomeNome implements Comparator<Invitato>
14  {
15      @Override
16      public int compare(Invitato i1, Invitato i2)
17      {
18          if (i1.getCognome().equals(anObject: i2.getCognome()))
19              return (i1.getNome().compareTo(anotherString: i2.getNome()));
20          else
21              return (i1.getCognome().compareTo(anotherString: i2.getCognome()));
22      }
23  }
```

L'interfaccia "chiede" di ridefinire il metodo compare che restituisce un valore >0, <0 o =0 in seguito al confronto fra due generici elementi da comparare.

Ora per ottenere un ArrayList ordinata nel criterio scelto (Cognome, Nome), scriviamo le seguenti istruzioni nel metodo main:

Creiamo una copia della listaFesta

```
ArrayList<Invitato> listaOrdinataCognomeNome=new ArrayList<>(c: listaFesta);
Collections.sort(list: listaOrdinataCognomeNome, new ComparatorCognomeNome());
System.out.println("Elenco lista ordinata per cognome e nome:\n"+listaOrdinataCognomeNome.toString());
```

Invochiamo il metodo statico sort della classe Collections passando come parametri:

l'ArrayList da ordinare

un'istanza della classe che contiene il criterio di ordinamento (nel nostro caso ComparatorCognomeNome)

Se vogliamo ordinare la lista in base ad un altro criterio, ad esempio in base all'ordine alfabetico del codice fiscale, si deve creare un'altra classe "Comparator", chiamata, ad esempio, ComparatorCF, e ordinare poi nel metodo main un'altra copia dell'ArrayList di partenza utilizzando questo secondo criterio:

Classe ComparatorCF:

```
13    public class ComparatorCF implements Comparator<Invitato>
14    {
15        @Override
          public int compare(Invitato i1, Invitato i2)
17        {
18            return (i1.getCf().compareTo(anotherString: i2.getCf()));
19        }
20
21    }
```

Ordinamento in base al codice fiscale nel metodo main:

```
ArrayList<Invitato> listaOrdinataCF=new ArrayList<>(c: listaFesta);
Collections.sort(list: listaOrdinataCF, new ComparatorCF());
System.out.println("Elenco lista ordina per CF:\n"+listaOrdinataCF.toString());
```

Grazie all'interfaccia Comparator, in questo modo possiamo creare infiniti criteri di confronto e quindi ordinare l'ArrayList secondo qualsiasi criterio a nostro piacere.

ESERCIZIO GARA:

Realizzare un'applicazione per la gestione di una gara podistica utilizzando l'ArrayList. Si prenda spunto dal seguente diagramma delle classi e dei casi d'uso.