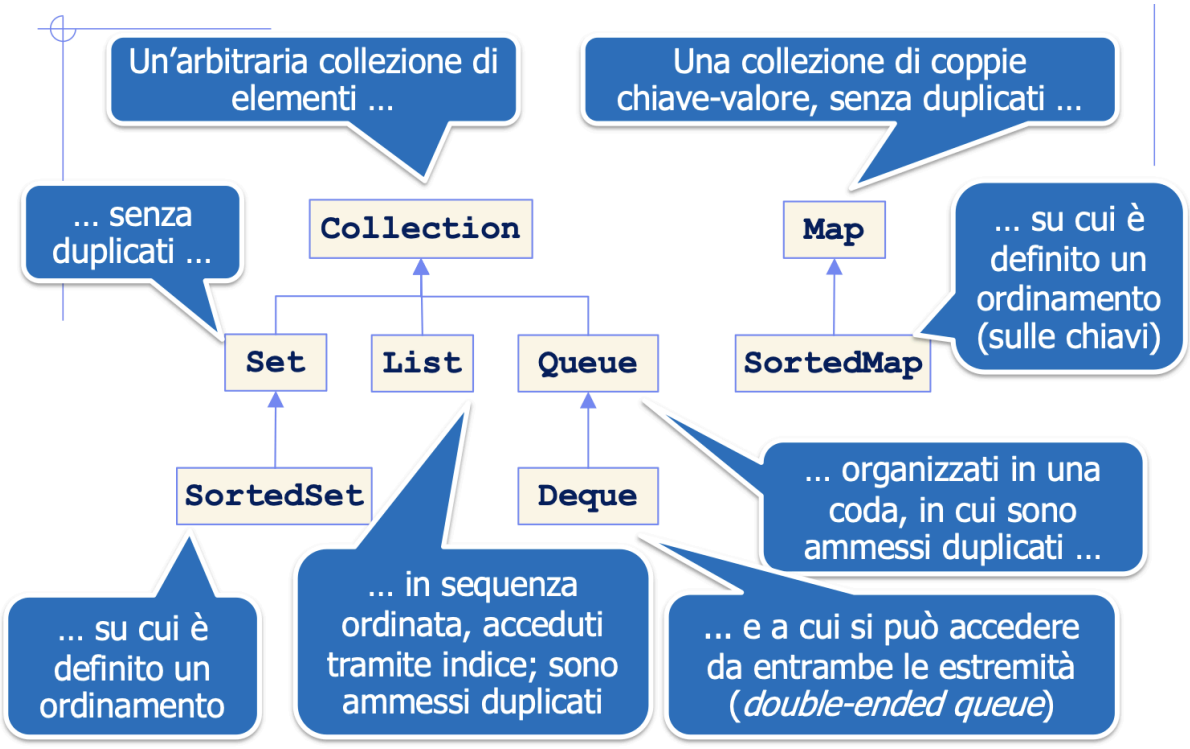
Java Collection Framework

Il Java Collection Framework è un raggruppamento di funzionalità volte ad ampliare ed implementare il funzionamento delle **collections**. Tale Framework è composto da:

- **interfacce**: che specificano insiemi di servizi associati a diversi tipi di collections
- **implementazioni**: implementazioni di specifiche strutture dati di uso comune, implementanti le interfacce citate.
- **algoritmi**: codificati in metodi, implementano operazioni comuni a più strutture dati, ne sono un esempio: algoritmi di ricerca, algoritmi di ordinamento ecc…

Il vantaggio di Java di avere *built-in* un Framework che definisce determinate strutture dati molto efficienti e già predisposte con diversi algoritmi ottimizzati fa si che all'utente sia risparmiato il compito di implementarle da sé, andando quindi a inficiare potenzialmente l'ergonomia ed efficienza del codice.
Al fine di poter usare il Java Collection Framework è necessario **importare la libreria** " java.util.* ".

La **gerarchia di interfacce** che si crea è la seguente:



Come intuibile, ogni tipo di collection gode dei sui metodi completamente implementabili.

Collection e Array

Collection<E> offre i seguenti metodi:

- Object [ ] toArray( )
- <T> T [ ] toArray(T [ ] a) → metodo **generico**

che consentono di ottenere il contenuto di una collection in un array, quindi in un contenitore di oggetti di dimensione prefissata. Ipotizziamo una variabile " **C** " di tipo **collection**:

- Object [ ] a = C.toArray( )
- String [ ] a = C.toArray(new String [0])

In particolare nell'ultimo caso una stringa viene suddivisa per caratteri e ogni carattere viene messo in una cella dell'array.

Iteratori

Ad ogni oggetto di tipo Collection<E> è **associato un oggetto di tipo Iterator<E>** in particolare: **Iterator<Integer> i = C.iterator( )**. Questo operatore ci consente di **scandire gli elementi della collection** uno a uno. In particolare **Iterator** è di per sé un'interfaccia con al suo interno i seguenti metodi:

- **boolean hasNext( )** che ritorna il valore true se ci sono ancora elementi da scandire
- **E next( )** che ritorna l'elemento successivo alla posizione attuale
- **void remove( )** che rimuove l'ultimo elemento ritornato da next( ), è quindi invocabile soltanto una volta.

Un semplice esempio di utilizzo degli iteratori è il seguente: si vogliono rimuovere da una certa collezione determinati oggetti che rispettano una certa condizione (criterio determinato dal metodo " cond( ) " definito altrove.

```
void filter(Collection<E> x) {
  Iterator<E> i = x.iterator();
  while(i.hasNext()) {
    if(!cond(i.next()))
      i.remove();
  }
}
```

Possiamo notare che in un certo senso il codice è **polimorfico** ovvero funziona per ogni tipo di collezione che supporta la rimozione di elementi, indipendentemente dalla specifica implementazione.

Un problema che può sorgere utilizzando gli iteratori è la macchinosità del codice, si rischia di utilizzare tante righe quando il tutto può essere compattato a meno istruzioni. Da Java 5 è possibile sostituire questi elementi con il costrutto **for - each** che letteralmente opera come se si dicesse: *per ogni elemento di un certo tipo nella collection, fai qualcosa*. Tutto questo viene fatto **indicizzando** i singoli elementi della collection e quindi sostituendo il lavoro svolto dagli iteratori.

Si supponga di voler stampare tutte le combinazioni di facce che possono uscire dal lancio di due dadi. Si propone una soluzione con l'utilizzo degli iteratori ed una analoga ma con l'utilizzo del costrutto for - each.

```
for(Iterator<String> i = faces.iterator(); i.hasNext();) {
  String s = i.next();
  for(Iterator<String> j = faces.iterator(); j.hasNext();)
    System.out.println(s + " " + j.next());
}
```

- Oppure, con "for-each":

```
for(String i : faces)
  for(String j : faces)
    System.out.println(i + " " + j);
```

Ambedue le soluzioni sono corrette

La seconda è più leggibile e intuitiva

```
ONE ONE
ONE TWO
ONE THREE
ONE FOUR
ONE FIVE
ONE SIX
… …
SIX FOUR
SIX FIVE
SIX SIX
```

Si noti che il *for-each* può essere usato con qualsiasi oggetto che implementa l'interfaccia *iterable*. Tuttavia il *for-each* può risultare non del tutto utilizzabile. Si osservino i casi per esempio in cui si deve chiamare il metodo **remove**: tale metodo può essere chiamato solamente sull'iteratore. Oppure si ipotizzi si voglia sostituire l'elemento corrente: è necessario sapere la posizione cioè il valore dell'iteratore (o indice dell'Array).

Un accorgimento da tenere in considerazione è il fatto che **non è possibile modificare una collection attraverso i suoi metodi mentre si utilizza un iteratore su di essa**, l'operazione va delegata all'**iteratore**.