



Università degli Studi dell'Aquila



Dipartimento di Ingegneria e Scienze
dell'Informazione e Matematica

Università degli Studi dell'Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

La libreria standard: JCF

La libreria standard

- ▶ Java possiede un'enorme e lussuosa libreria di classi standard, che costituisce uno dei punti di forza del linguaggio.
- ▶ Essa è organizzata in vari **package** e **subpackage** (fisicamente corrispondono a cartelle e sottocartelle) che raccolgono le classi secondo un'organizzazione basata sul campo d'utilizzo.

La libreria standard

- ▶ I principali package sono:
 - **java.io** contiene classi per realizzare l'input - output in Java
 - **java.awt** contiene classi per realizzare interfacce grafiche, come `Button`
 - **java.net** contiene classi per realizzare connessioni, come `Socket`
 - **java.applet** contiene un'unica classe: `Applet`. Questa permette di realizzare applet
 - **java.util** raccoglie classi d'utilità, come `Date`
 - **java.lang** è il package che contiene le classi nucleo del linguaggio, come `Object`, `System`, `String`, l'interfaccia `Comparable`...

Il comando `import`

- ▶ In qualunque programma Java ci si può riferire a tali classi tramite il loro nome "lungo", cioè avente come prefisso anche il nome del package e del subpackage.

Es:

```
java.util.Scanner tastiera =  
    new java.util.Scanner(System.in);
```

- ▶ Tuttavia nomi così "lunghi" sono scomodissimi!
- ▶ Le dichiarazioni `import`, che possono essere messe solo all'inizio di un file `.java`, prima di ogni dichiarazione di classe, permettono di usare in quel file i nomi "corti" delle classi importate, senza il prefisso del package.

Il comando `import`

- ▶ Per utilizzare il nome “corto” di una classe della libreria all’interno di una nuova classe bisogna dunque importarla.
- ▶ Supponiamo di voler utilizzare la classe `Date` del package `java.util`. Prima di dichiarare la classe in cui abbiamo intenzione di utilizzare `Date` dobbiamo scrivere:

```
import java.util.Date;
```
- ▶ oppure, per importare tutte le classi del package `java.util`:
- ▶

```
import java.util.*;
```

 // è detto **wild card**

Il comando `import`

- ▶ Di default in ogni file Java è importato automaticamente tutto il package `java.lang`, senza il quale non potremmo utilizzare classi fondamentali quali `System`, `String`, `Math`.
- ▶ Notiamo che questa è una delle caratteristiche che rende Java definibile come "semplice". Quindi, nel momento in cui compiliamo una classe Java, il compilatore anteporrà alla dichiarazione della nostra classe il comando:

```
import java.lang.*;
```

Il comando `import`: `remark`

- ▶ Le dichiarazioni di `import` NON copiano programmi né in formato sorgente né in formato compilato; rendono semplicemente utilizzabili nomi corti invece di nomi lunghi. Pertanto, importare tutte le classi di un package non è penalizzante rispetto a importarne una sola.
- ▶ L'asterisco non implica l'importazione delle classi appartenenti ai “sottopackage”
 - `import java.awt.*` non importa `java.awt.event.*`
- ▶ Quindi l'istruzione `import java.*` non importa tutti i package fondamentali.

La libreria standard

- ▶ Per conoscere tutte le classi (che sono in continua evoluzione), basta consultare la documentazione del JDK–Java Development Kit (da installare a parte).
- ▶ Aprire il file "index.html" che si trova nella cartella "API" della cartella "Docs" del J.D.K.
- ▶ Se non si trova la cartella fare una ricerca sul disco rigido.
- ▶ Se la ricerca fallisce procurarsi la documentazione (www.java.sun.com)

L'interfaccia `java.lang.Comparable`

- ▶ L'interfaccia `Comparable` impone un **ordinamento totale** sugli oggetti della classe che la implementa (ordinamento naturale della classe).
- ▶ `Comparable` contiene il solo metodo di confronto naturale:

```
public interface Comparable {  
    int compareTo(Object obj);  
}
```

- Ne vedremo in seguito la versione parametrica

L'interfaccia Comparable

- ▶ Il metodo `int compareTo(Object obj)` confronta l'oggetto corrente con l'oggetto specificato `obj`
- ▶ `o1.compareTo(o2)` restituisce:
 - un numero positivo se $o1 > o2$ (`o1` **segue** `o2` nell'ordinamento)
 - un numero negativo se $o1 < o2$ (`o1` **precede** `o2` nell'ordinamento)
 - 0 se $o1 == o2$, it returns 0 (`o1` e `o2` sono uguali)

L'interfaccia Comparable

- ▶ L'ordinamento naturale per una classe C è **consistente con equals ()** se e solo se `o1.compareTo(o2) == 0` ha lo stesso valore booleano di `o1.equals(o2)` per ogni `o1` ed `o2` della classe C
$$(x.compareTo(y) == 0) == (x.equals(y))$$
- ▶ È fortemente raccomandato, ma non strettamente richiesto, che `compareTo()` sia consistente (o compatibile) con `equals()`
- ▶ Ogni classe che implementa l'interfaccia **Comparable** e viola questa condizione dovrebbe dichiararlo esplicitamente
 - *"Note: this class has a natural ordering that is inconsistent with equals."*

L'interfaccia Comparable

Classi di Java che implementano Comparable:

- ▶ String
- ▶ File
- ▶ Date
- ▶ Byte, Character, Short, Integer, Long, Float, Double (**classi wrapper**)

Classi Wrapper (richiami)

- ▶ In varie situazioni, può essere necessario poter trattare i **tipi primitivi come oggetti**
 - per passarli per riferimento a una funzione
 - quando una funzione pretende come parametro (o restituisce) un Object (vedi Collection, List, ...)
- ▶ Una classe “wrapper” incapsula una variabile di un tipo primitivo
 - la classe `Boolean` incapsula un `boolean`
 - la classe `Double` incapsula un `double`
 - ...
- ▶ La classe wrapper ha nome (quasi) identico al tipo primitivo che incapsula, ma con l’iniziale maiuscola.

Classi Wrapper

- ▶ Le classi wrapper sono **immutabili**
- ▶ Ogni classe wrapper definisce metodi per estrarre il valore della variabile incapsulata e viceversa
- ▶ Per estrarre il valore incapsulato:
 - `Integer` fornisce il metodo `intValue()`
 - `Double` fornisce il metodo `doubleValue()`
 - `Boolean` fornisce il metodo `booleanValue()`
 - ...
- ▶ Per creare un oggetto da un valore primitivo:
 - `Integer i = new Integer(valore int)`
 - `Double d = new Double(valore double)`
 - ...

Classi Wrapper

- ▶ **Remark:** le sei classi wrapper `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double` relative ai tipi numerici estendono la classe astratta `Number`
- ▶ Grazie a **autoboxing** e **auto-unboxing**, il compilatore si occupa di inserire le istruzioni di conversione dei valori dei tipi base in oggetti e viceversa laddove queste sono necessarie

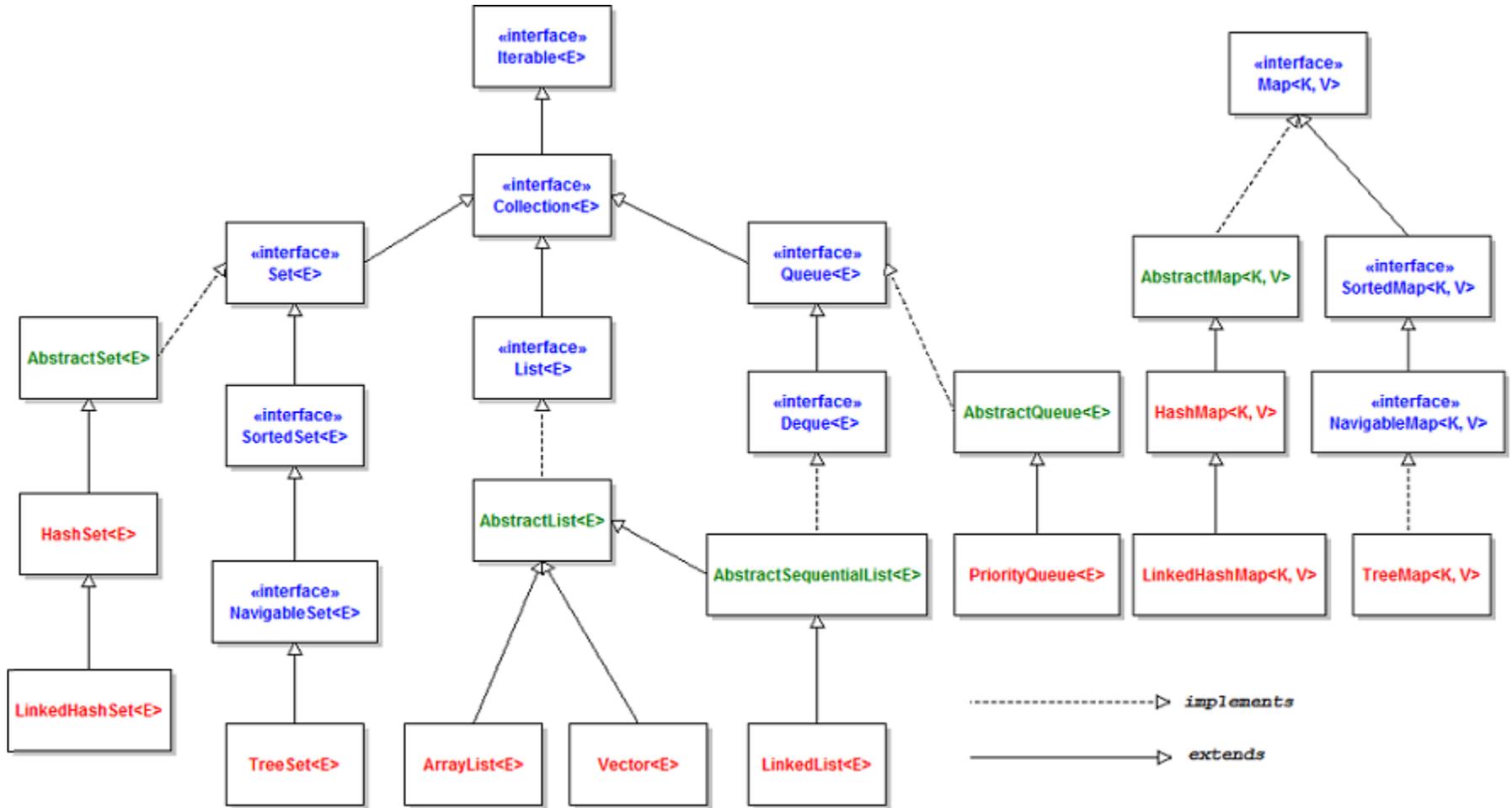
Java Collections Framework: overview

- ▶ L'infrastruttura Java Collections Framework (JCF) è una parte della libreria standard dedicata alle **collezioni**
- ▶ È una raccolta di interfacce e classi, tra loro correlate, appartenenti al pacchetto **java.util.**
- ▶ Un esemplare di una classe del JCF rappresenta generalmente una **collezione di oggetti (collection).**

Java Collections Framework

- ▶ Offre strutture dati di supporto molto utili alla programmazione, come array di dimensione dinamica, liste, insiemi, mappe associative (anche chiamate dizionari) e code
- ▶ Il JCF è costituito in pratica da una gerarchia che contiene classi astratte e interfacce ad ogni livello tranne l'ultimo, dove sono presenti soltanto classi che implementano interfacce e/o estendono classi astratte

Java Collections Framework



Java Collections Framework

- ▶ In cima alla gerarchia troviamo le interfacce `Collection` e `Map`
- ▶ L'interfaccia `Collection` estende la versione parametrica di `Iterable`.
- ▶ L'interfaccia `Collection` non va confusa con la classe `Collections` che contiene numerosi algoritmi di supporto (metodi statici che operano su collezioni)
 - ad esempio, metodi che effettuano l'ordinamento

Cos'è una collection?

- ▶ Una raccolta (collection) è un oggetto composto da elementi.
- ▶ Esempio: un array è una raccolta di elementi dello stesso tipo che vengono memorizzati in aree contigue della memoria

```
String[] names = new String[5];
```

Array

- ▶ **Vantaggio:**
 - Si può accedere ad un singolo elemento dell'array in modo diretto (proprietà di accesso casuale)
- ▶ **Svantaggi:**
 - La dimensione è fissa. Se la dimensione si rivela insufficiente occorre creare un array più grande e copiarvi il contenuto di quello più piccolo.
 - L'inserimento o la rimozione di un elemento può richiedere lo spostamento di molti elementi.
 - Queste operazioni di mantenimento devono essere gestite dal programmatore

“Collection class”

- ▶ Un’alternativa all’uso degli array ?
L’uso di esemplari di classi che rappresentano collezioni (con abuso di terminologia “*collection class*”)
- ▶ Una collection class è una classe i cui singoli esemplari sono raccolte di elementi
- ▶ Gli elementi in un’istanza di una collezione devono essere riferimenti ad un oggetto.
 - Non possiamo creare un esemplare di una raccolta i cui singoli elementi siano di tipo primitivo
 - Possiamo usare le classi wrapper

Strutture di memorizzazione

- ▶ Possibili strutture di memorizzazione per collection class sono:
 1. **Contiguous collection (Raccolta contigua):** Il modo più semplice per archiviare in memoria una raccolta prevede di memorizzare in un array i riferimenti ai singoli elementi: in pratica la classe ha un array come campo

Strutture di memorizzazione

2. **Linked collection:** invece di usare la contiguità, gli elementi possono essere correlati tra loro mediante collegamenti (link), ovvero riferimenti.
 - In una classe che realizza una raccolta mediante collegamenti, ciascun elemento presente in un suo esemplare è memorizzato in una entry o nodo, che contiene almeno un collegamento ad un altro nodo.

Linked collection

- ▶ **Singly-linked list** (lista semplicemente concatenata): raccolta realizzata mediante collegamenti, dove ciascun nodo contiene un elemento ed un riferimento al nodo successivo presente nella raccolta
- ▶ **Doubly-linked list** (lista doppiamente collegata): raccolta realizzata mediante collegamenti, dove ciascun nodo contiene un elemento, un riferimento al nodo precedente ed un riferimento al nodo successivo presente nella raccolta
- ▶ **Binary search tree...**