



Fondamenti di Informatica

(L-Z)

Corso di Laurea in Ingegneria Gestionale

OOP: Strutture Dati e Classi Notevoli

Prof. Stefano Mariani

Dott. Alket Cecaj



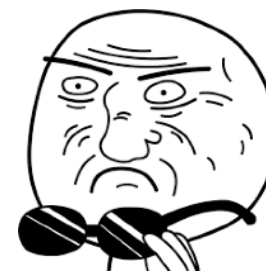
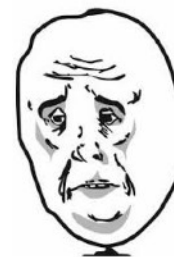
- Il framework Collections
 - ArrayList
 - HashMap
 - il costrutto for-each
- Classi notevoli:
 - String vs. StringBuilder
 - Date-Time API
- Digressione su tipi parametrici e classi wrapper



JAVA COLLECTIONS FRAMEWORK



- Per un qualunque programma non banale, gli array non bastano per gestire **collezioni** di dati
 - ▶ lunghezza fissa
 - ▶ singoli dati in ogni cella
- Java fornisce una varietà di classi (dunque *tipi di dato*) in grado di gestire collezioni di dati
 - ▶ raccolte nel **framework*** “**Collections**”
 - liste, mappe, insiemi, alberi, ...
 - ▶ package `java.util`



*Insieme di classi logicamente correlate e delle loro relazioni



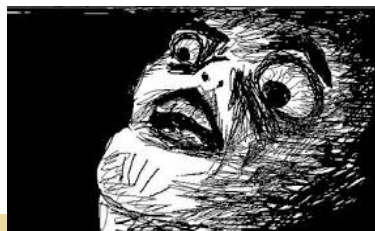
ArrayList

- Sostanzialmente, sono **array** di **dimensione variabile**
 - non occorre specificarne la lunghezza
 - si possono *aggiungere/rimuovere elementi*
 - la collezione si espande/contrae automaticamente

```
ArrayList<Integer> lista = new ArrayList<>();  
lista.add(1);  
lista.add(2);  
lista.add(3);  
for (int i = 0; i < lista.size(); i++) {  
    System.out.println("lista @ " + i + ") " + lista.get(i));  
}
```

- Notate due cose:

- il tipo di dato Integer
- la notazione <>





- I tipi primitivi *non* sono oggetti
 - in Java però, “*everything is an object*”
- Per ogni tipo primitivo è definita una corrispondente **classe wrapper** (contenitore) che *incapsula* in un oggetto un valore di tipo primitivo
 - **Integer** (per int), **Double** (per double), **Float** (per float), **Long** (per long), **Boolean** (per boolean), **Character** (per Char)
- Opportune operazioni consentono la *traduzione* tra classe wrapper e tipo primitivo, anche *automaticamente*:
 - **autoboxing** – da primitivo a oggetto
 - **unboxing** – da oggetto a primitivo
 - Integer.parseInt()
 - Double.parseDouble()
 - etc.

```
ArrayList<Integer> lista = new ArrayList<>();  
lista.add(1); // auto-boxing  
int n = 2;  
lista.add(n); // auto-boxing  
Integer nn = new Integer(3);  
lista.add(nn);  
int nnn = lista.get(2); // unboxing
```



- Le collezioni possono operare sugli elementi che contengono *indipendentemente* dal tipo degli elementi stessi
 - su un ArrayList di Integer potete svolgere le stesse operazioni che su un ArrayList di Studente
- Ogni tipo (classe) nel framework Collections è **parametrico**
 - funziona su qualunque *tipo di dato* passato come *parametro*
- Parametrizzare un tipo garantisce la *type-safety a tempo di compilazione*
 - tradotto: vi accorgete degli errori mentre scrivete il codice

```
ArrayList<Integer> interi = new ArrayList<>();  
interi.add(1);  
interi.add("ciao"); // ERRORE  
ArrayList<String> parole = new ArrayList<>();  
parole.add("ciao");  
parole.add(4); // ERRORE
```

Ci sarebbero decine di ore da spendere sull'argomento, ma NON in un corso di fondamentali :)



- Sostanzialmente, sono *tabelle*
 - ▶ *associano* un oggetto (valore) a un altro (chiave)
 - ▶ consentono di recuperare il valore data la chiave
 - senza sapere *dove* si trova
 - ▶ *non* ammettono chiavi duplicate
 - inserire un secondo valore con una stessa chiave *sovrascrive* il primo valore

```
HashMap<Integer, String> numeri = new HashMap<>();  
numeri.put(0, "zero");  
numeri.put(1, "uno");  
numeri.put(2, "due");  
for (Integer i : numeri.keySet()) {  
    System.out.println(i + " -> " + numeri.get(i));  
}
```

- *Notate:*
 - ▶ Cos'è quel 'for'??



Esempi di metodi offerti da HashMap nel file Slide08.java della lezione 11



Il costrutto **for-each**

- **Sintassi:**

```
for (<tipo> <variabile> : <collezione>) {  
    <action1>;  
    <action2>;  
    ...  
}
```

- E.g.:

```
HashMap<Integer, String> numeri = new  
HashMap<>();  
for (Integer i : numeri.keySet()) {  
    System.out.println(numeri.get(i));  
}
```

- **Iterazione**

- ogni elemento della collezione viene iterato...

- **Terminazione**

- ...fino al termine della collezione stessa :)

Esempio di utilizzo nel file Slide08.java della lezione 11



Il costrutto for-each

- Sintassi alternativa al for conveniente quando si devono *leggere* elementi di una collezione
- Da non usare se volete
 - ▶ modificare o ri-assegnare l'elemento iterato
 - solo lettura
 - ▶ iterare due collezioni in parallelo
 - un for-each per ogni collezione
 - ▶ accedere a diversi elementi nella stessa collezione
 - ▶ saltare elementi o iterare in altro ordine
 - solo iterazione sequenziale "in avanti"



CLASSI NOTEVOLI



- `boolean equals(String str)`
- `boolean equalsIgnoreCase(String str)`
- `boolean startsWith(String str)`
- `boolean endsWith(String str)`
- `boolean contains(String str)`
- `String trim()`
- `int length()`
- `int indexOf(String str)`
- `String substring(int begin, int end)`
- `char[] toCharArray()`
- `String[] split(String regex)`
 - <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>*

*La stringa fornita rappresenta un oggetto Pattern (non lo useremo)
Esempi di utilizzo nel file Slide12.java della lezione 11



- Gli oggetti di tipo String sono *immutabili*
 - quando scriviamo cose tipo

```
String s1 = "ciao";  
s1 = s1 + "ciao";
```
 - non viene modificata s1, ma viene creata una nuova stringa, poi messa in s1
- La classe **StringBuilder** fornisce stringhe *mutabili*
 - *velocizzando* notevolmente operazioni quali la *concatenazione* di molte stringhe
 - fornendo le stesse operazioni viste per String
 - offrendone altre
 - e.g., `append()` e `insert()`



- La *versione 8* di Java* ha introdotto nuove classi per la gestione del tempo (package `java.time`)
 - **LocalDate** – per gestire giorni, mesi, anni, ...
 - **LocalTime** – per gestire ore, minuti, secondi, ...
 - **LocalDateTime** – unione delle precedenti
- Noi useremo queste API molto limitatamente
 - e.g. non considereremo le problematiche (dunque le API) legate ai fusi orari (time zones, in inglese)
 - Guardate gli esempi di metodi offerti nel file *Slide14.java* della lezione 11

*Quella usata da noi, che è anche l'ultima release stabile



Fondamenti di Informatica

(L-Z)

Corso di Laurea in Ingegneria Gestionale

OOP: Strutture Dati e Classi Notevoli

Prof. Stefano Mariani

Dott. Alket Cecaj