

ottimizzazione applicazioni dati  
realizzazione basi soluzioni risolvere comprensione tipo  
specifiche codice interfaccia alocazione librerie oggetti swing  
problem problemi linguaggio tortile laboratorio greedy  
gestione gesto corso complessità algoritmi ottimale standard esercitazioni  
grado diversa conoscerà impara programma informatico riconoscere min-max  
corsi corsi diversi programmi informatico aula mediante solving risoluzione  
vista particolare vista questo particolare utilizzo divide proprie  
grafici complesse accesso

## Lists

Arrays reloaded



# Problem

- ▶ Store a set of *unique* words (duplicates shall be ignored)
- ▶ Class “interface”

```
public class WordSet {  
    public Boolean add(String str);  
    public void delete(String str);  
    public void dump();  
}
```



# Main (driver)

```
public static void main(String[] args) {  
    Scanner keyboard = new Scanner(System.in);  
    String str;  
  
    do {  
        str = keyboard.next();  
        if(!str.equals("-")) {  
            if(!ws.add(str)) {  
                System.out.println("Yeuch");  
            }  
        }  
    } while(!str.equals("-"));  
    keyboard.close();  
    ws.dump();  
  
    ws.remove("foo");  
    ws.dump();  
}
```

# Solution 1 (Array)

---

- ▶ Array of String
- ▶ Check whether a word is already present in the array before inserting it
- ▶ Shift the array after deleting an element



# Data and constructor

```
final static int A_BIG_NUMBER = 9999;  
String[] words;  
int numWords;  
  
public WordSet() {  
    numWords = 0;  
    words = new String[A_BIG_NUMBER];  
}
```



# dump() method

```
public void dump() {  
    System.out.println("WORDS");  
    for(int t=0; t<numWords; ++t) {  
        System.out.printf("%d %s\n", t+1, words[t]);  
    }  
}
```



# add() method

```
public Boolean add(String str) {  
    Boolean newWord = true;  
    for(int t=0; t<numWords; ++t) {  
        if(str.equals(words[t])) {  
            newWord = false;  
        }  
    }  
    if(newWord) {  
        words[numWords++] = str;  
    }  
    return newWord;  
}
```



# remove() method

```
public void remove(String str) {  
    int t;  
    t=0;  
    while(!str.equals(words[t]))  
        ++t;  
    for(++t; t<numWords; ++t) {  
        words[t-1] = words[t];  
    }  
    --numWords;  
}
```

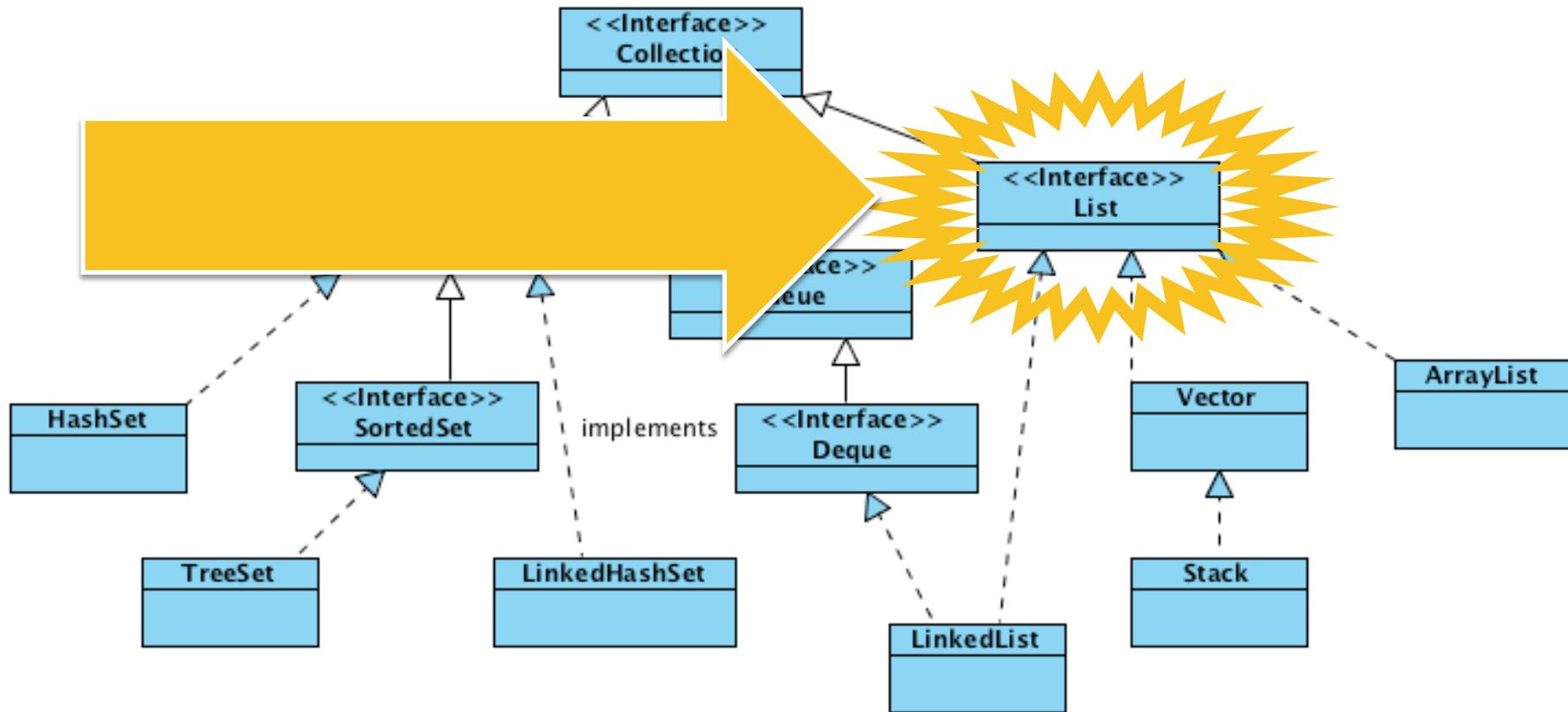


# remove() method

```
public void remove(String str) {  
    int t;  
    t=0;  
    while(!str.equals(words[t]))  
        ++t;  
    for(++t; t<numWords; ++t) {  
        words[t-1] = words[t];  
    }  
    --numWords;  
}
```



# Collection Family Tree



# Lists == Arrays “Reloaded”

---

- ▶ Lists are (probably) the most widely used Java collections
- ▶ Like arrays
  - ▶ full visibility and control over the ordering of its elements
  - ▶ may contain duplicates
- ▶ Unlike arrays
  - ▶ resize smoothly

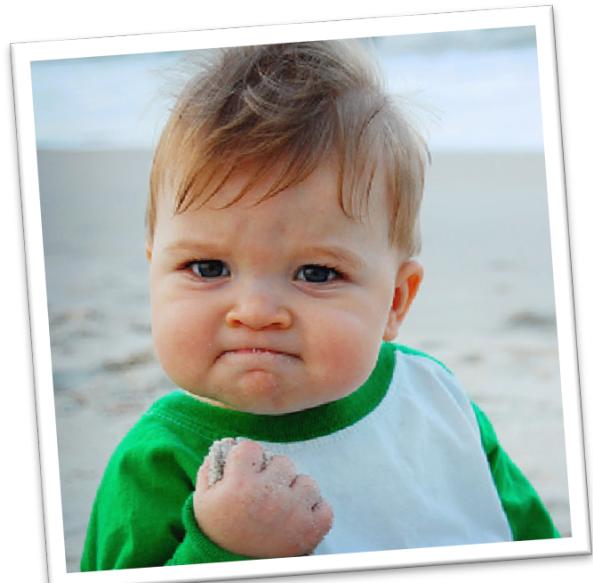
# List interface

---

- ▶ Add/remove elements
  - ▶ boolean **add(element)**
  - ▶ boolean **remove(object)**
- ▶ Positional Access
  - ▶ element **get(index)**
  - ▶ element **set(index, element)**
  - ▶ void **add(index, element)**
  - ▶ element **remove(index)**
- ▶ Search
  - ▶ boolean **contains(object)**
  - ▶ int **indexOf(object)**

# remove() method

```
public void remove(String str) {  
    words.remove(str);  
}
```





# dump() method

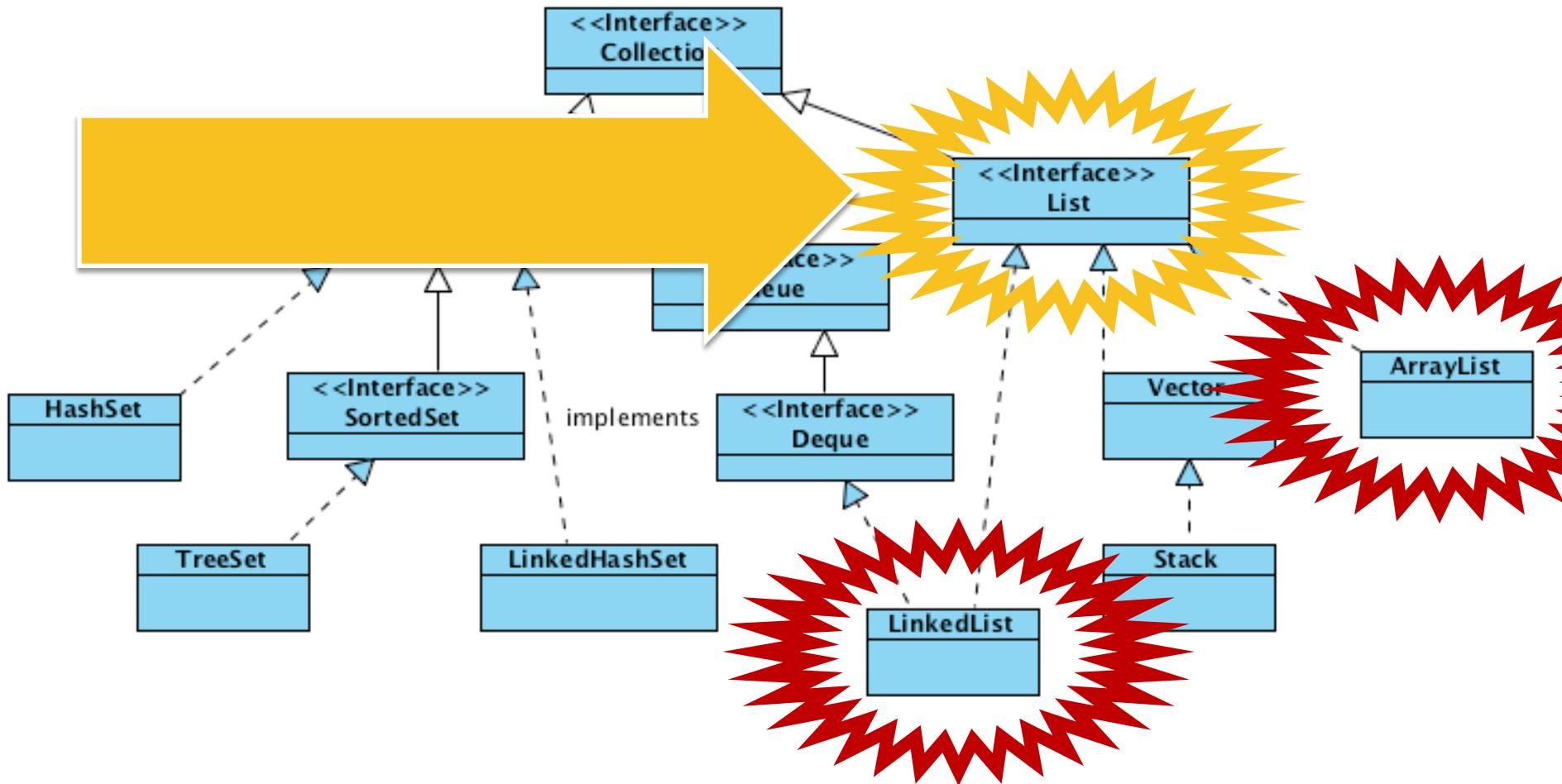
```
public void dump() {  
    System.out.println("WORDS");  
  
    Iterator<String> i = words.iterator();  
    while(i.hasNext()) {  
        System.out.println(i.next());  
    }  
}
```



# add() method

```
public Boolean add(String str) {  
    if(!words.contains(str)) {  
        words.add(str);  
        return true;  
    } else {  
        return false;  
    }  
}
```

# Collection Family Tree





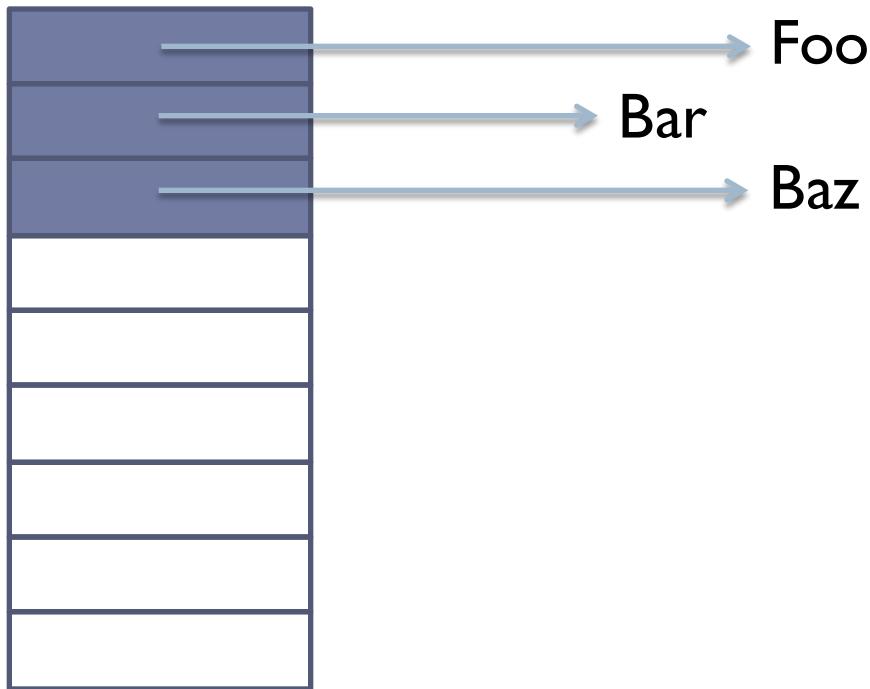
# Data and constructor

## ▶ ArrayList

```
List<String> words;  
  
public WordSet() {  
    words = new ArrayList<String>();  
}
```

# ArrayList

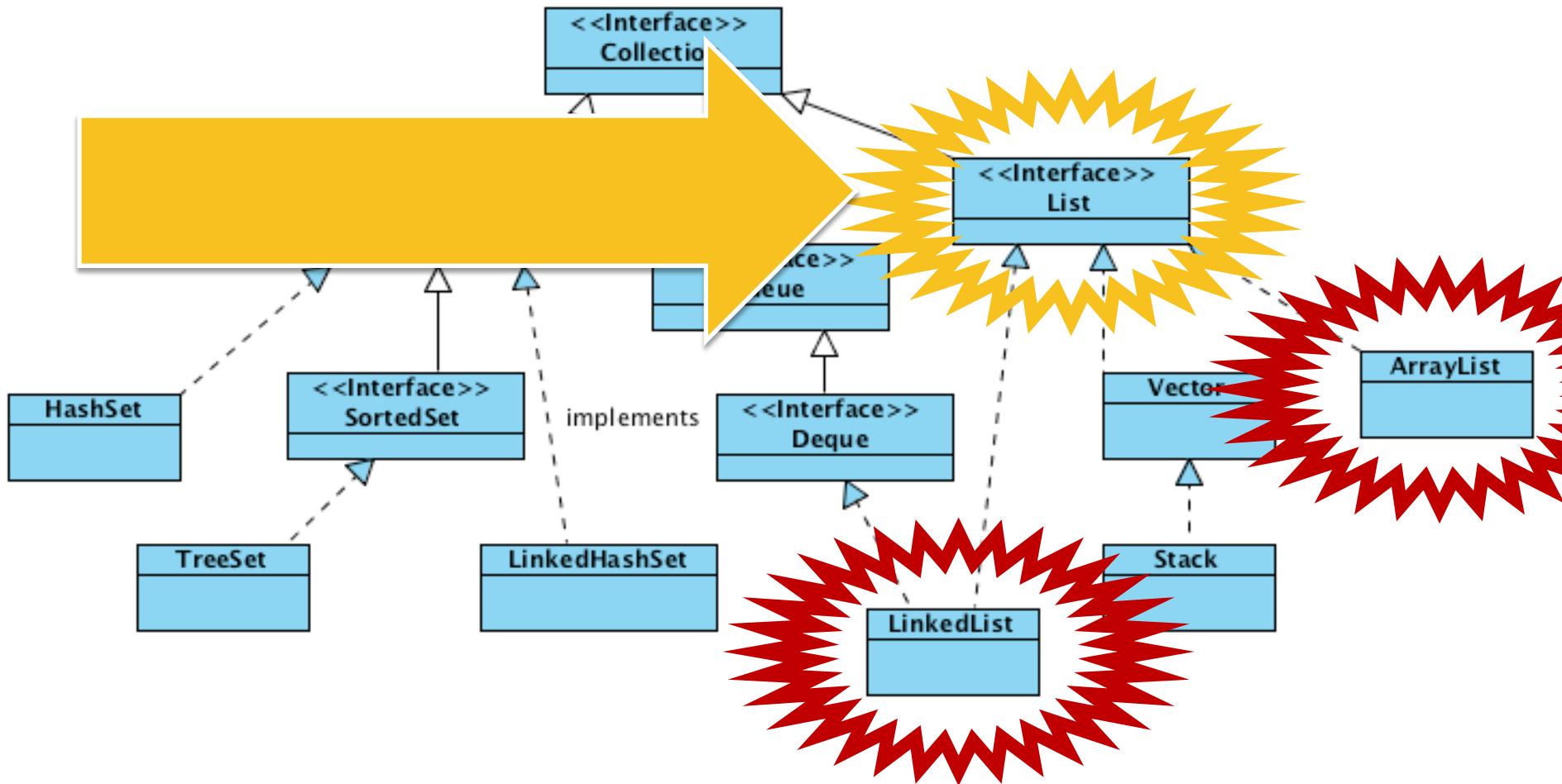
---



# ArrayList – Delete



# Collection Family Tree





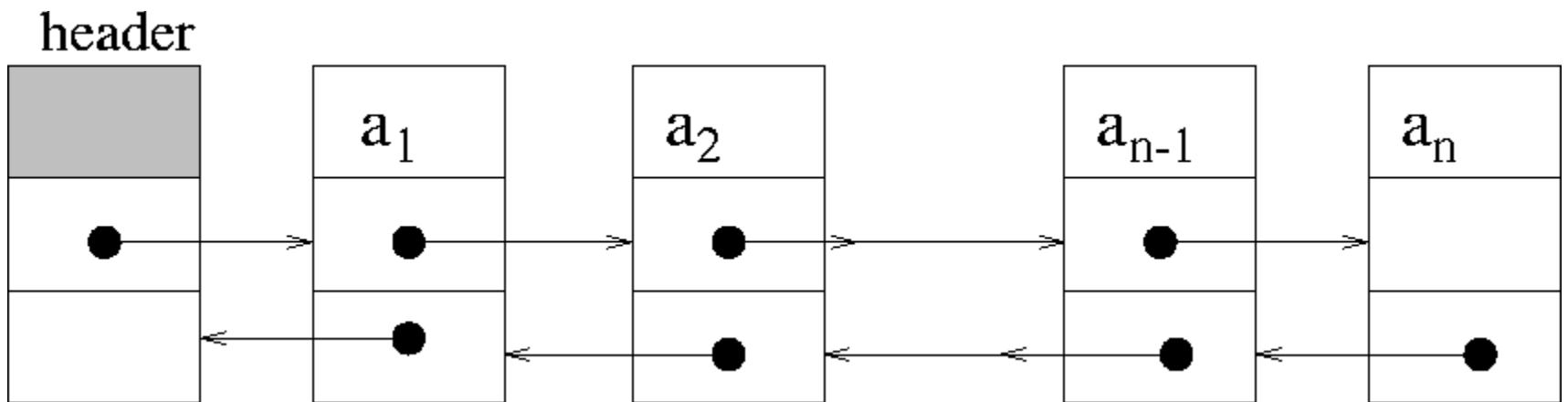
# Data and constructor

## ▶ **LinkedList**

```
List<String> words;  
  
public WordSet() {  
    words = new LinkedList<String>();  
}
```

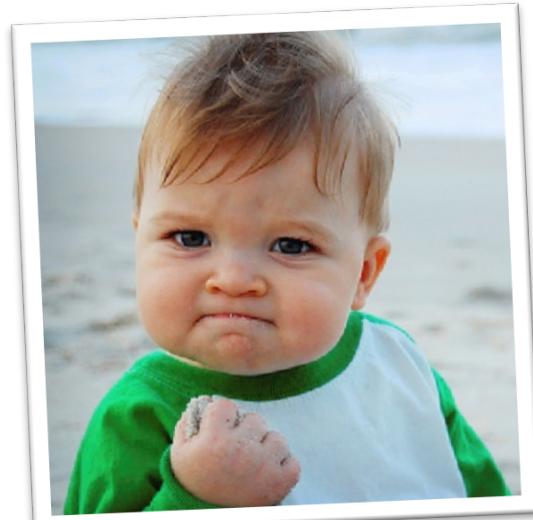
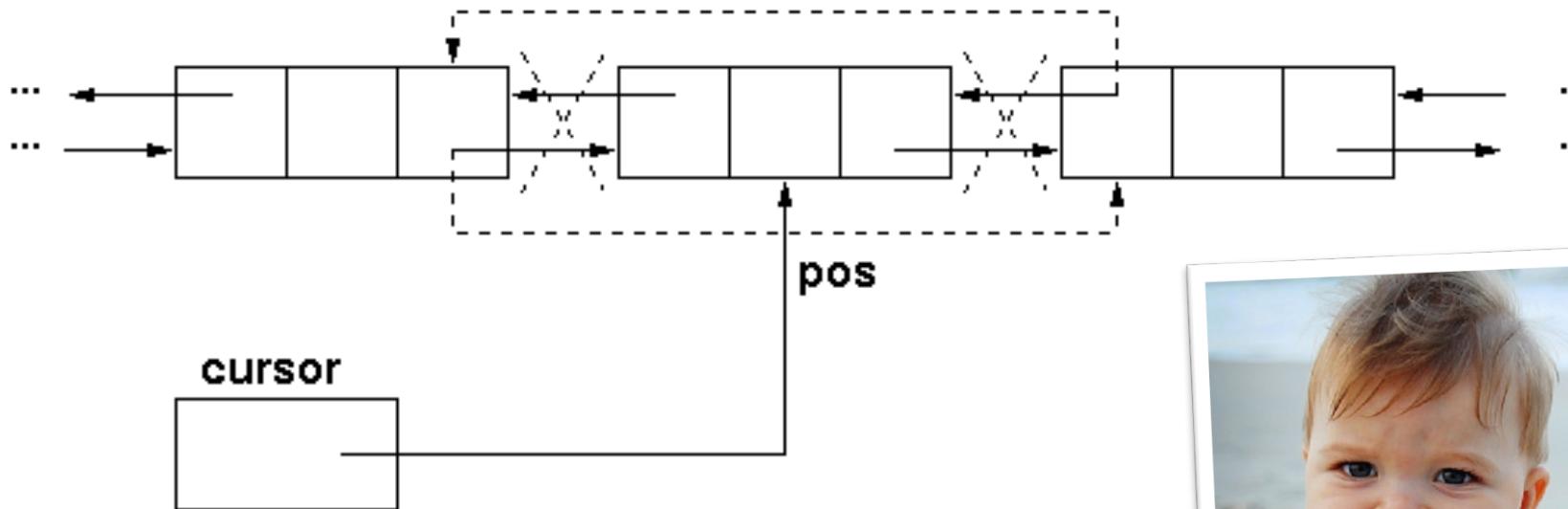
# LinkedList

---



# LinkedList – Delete

Removal of an element of a doubly-linked list



# ArrayList vs. LinkedList

|                            | ArrayList | LinkedList |
|----------------------------|-----------|------------|
| <b>add(element)</b>        |           |            |
| <b>remove(object)</b>      |           |            |
| <b>get(index)</b>          |           |            |
| <b>set(index, element)</b> |           |            |
| <b>add(index, element)</b> |           |            |
| <b>remove(index)</b>       |           |            |
| <b>contains(object)</b>    |           |            |
| <b>indexOf(object)</b>     |           |            |

# ArrayList vs. LinkedList

|                            | ArrayList        | LinkedList       |
|----------------------------|------------------|------------------|
| <b>add(element)</b>        | <b>IMMEDIATE</b> | <b>IMMEDIATE</b> |
| <b>remove(object)</b>      |                  |                  |
| <b>get(index)</b>          |                  |                  |
| <b>set(index, element)</b> |                  |                  |
| <b>add(index, element)</b> |                  |                  |
| <b>remove(index)</b>       |                  |                  |
| <b>contains(object)</b>    |                  |                  |
| <b>indexOf(object)</b>     |                  |                  |

# ArrayList vs. LinkedList

|                                  | ArrayList | LinkedList    |
|----------------------------------|-----------|---------------|
| <code>add(element)</code>        | IMMEDIATE | IMMEDIATE     |
| <code>remove(object)</code>      | SLUGGISH  | LESS SLUGGISH |
| <code>get(index)</code>          |           |               |
| <code>set(index, element)</code> |           |               |
| <code>add(index, element)</code> |           |               |
| <code>remove(index)</code>       |           |               |
| <code>contains(object)</code>    |           |               |
| <code>indexOf(object)</code>     |           |               |

# ArrayList vs. LinkedList

|                                  | ArrayList | LinkedList    |
|----------------------------------|-----------|---------------|
| <code>add(element)</code>        | IMMEDIATE | IMMEDIATE     |
| <code>remove(object)</code>      | SLUGGISH  | LESS SLUGGISH |
| <code>get(index)</code>          | IMMEDIATE | SLUGGISH      |
| <code>set(index, element)</code> | IMMEDIATE | SLUGGISH      |
| <code>add(index, element)</code> |           |               |
| <code>remove(index)</code>       |           |               |
| <code>contains(object)</code>    |           |               |
| <code>indexOf(object)</code>     |           |               |

# ArrayList vs. LinkedList

|                                  | ArrayList | LinkedList    |
|----------------------------------|-----------|---------------|
| <code>add(element)</code>        | IMMEDIATE | IMMEDIATE     |
| <code>remove(object)</code>      | SLUGGISH  | LESS SLUGGISH |
| <code>get(index)</code>          | IMMEDIATE | SLUGGISH      |
| <code>set(index, element)</code> | IMMEDIATE | SLUGGISH      |
| <code>add(index, element)</code> | SLUGGISH  | SLUGGISH      |
| <code>remove(index)</code>       | SLUGGISH  | SLUGGISH      |
| <code>contains(object)</code>    |           |               |
| <code>indexOf(object)</code>     |           |               |

# ArrayList vs. LinkedList

|                                  | ArrayList | LinkedList    |
|----------------------------------|-----------|---------------|
| <code>add(element)</code>        | IMMEDIATE | IMMEDIATE     |
| <code>remove(object)</code>      | SLUGGISH  | LESS SLUGGISH |
| <code>get(index)</code>          | IMMEDIATE | SLUGGISH      |
| <code>set(index, element)</code> | IMMEDIATE | SLUGGISH      |
| <code>add(index, element)</code> | SLUGGISH  | SLUGGISH      |
| <code>remove(index)</code>       | SLUGGISH  | SLUGGISH      |
| <code>contains(object)</code>    | SLUGGISH  | SLUGGISH      |
| <code>indexOf(object)</code>     | SLUGGISH  | SLUGGISH      |



# Timing

## ▶ Class System – current time

```
static long currentTimeMillis(); // in milliseconds  
static long nanoTime(); // in nanoseconds
```

**current value of the  
most precise  
available system  
timer**



# Random string

```
String val = "tag_" + num;
```

- ▶ Not quite random...

# Universally unique identifier

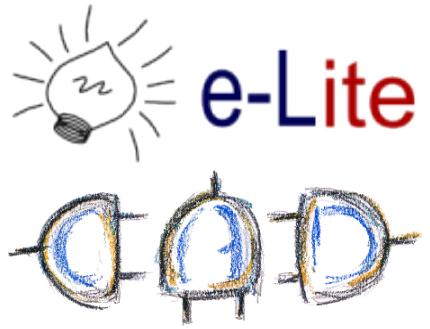
---

- ▶ *Open Software Foundation (OSF) standard*
- ▶ *Part of the Distributed Computing Environment (DCE)*
- ▶ **Five versions**
- ▶ **Version 4 (completely random)**
- ▶ **xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx**
  - ▶ x is any hexadecimal digit
  - ▶ y is one of 8, 9, a, or b.
- ▶ **E.g.,**
  - ▶ **f47ac10b-58cc-4372-a567-0e02b2c3d479**



# Random string

```
import java.util.UUID;  
  
String val = UUID.randomUUID().toString());
```

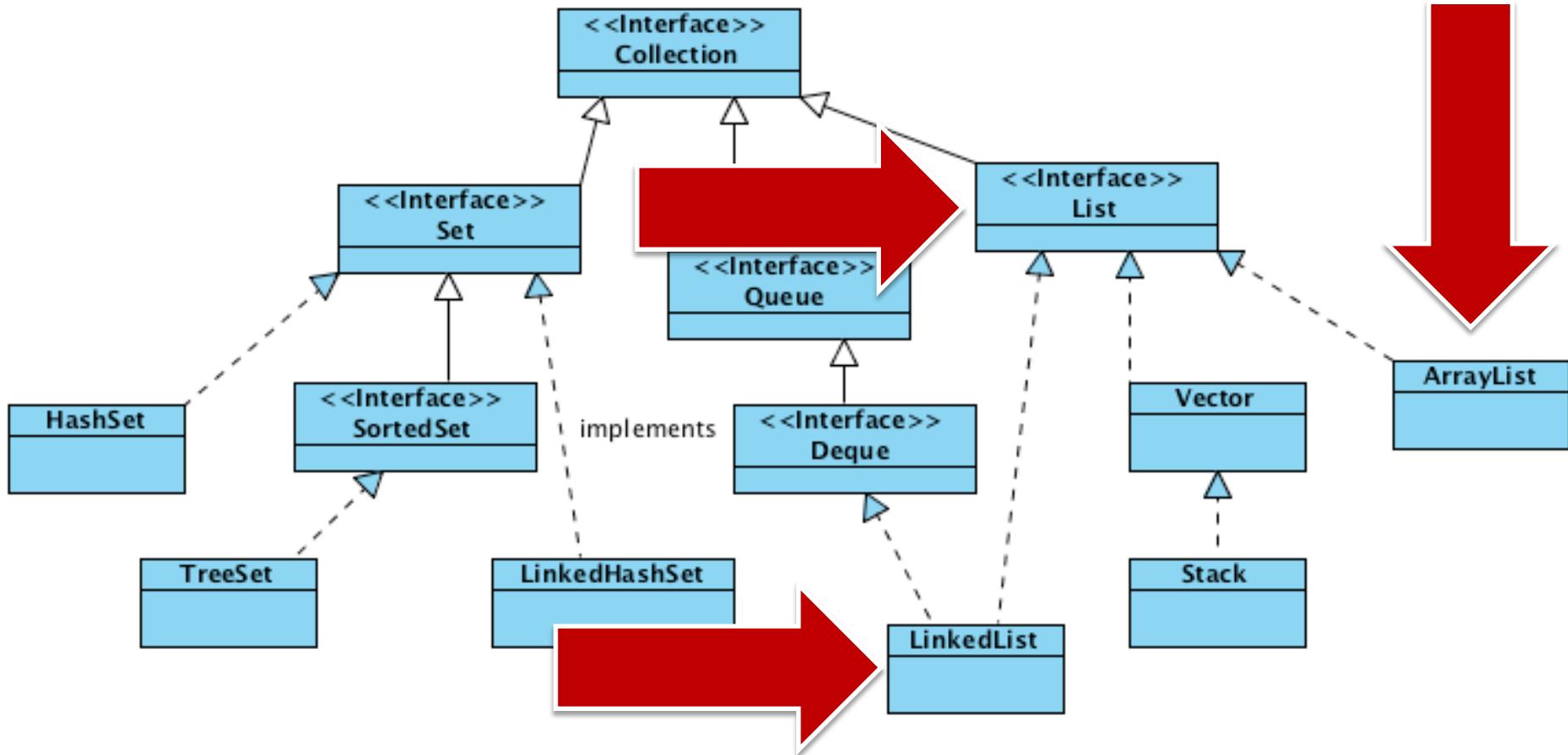


A large, dense word cloud centered on the right side of the slide. The words are primarily in Italian, with some English terms mixed in. The most prominent words are "ottimizzazione" (optimization), "algoritmi" (algorithms), "applicazioni" (applications), "dati" (data), and "java". Other visible words include "problemi" (problems), "realizzazione" (implementation), "gestione" (management), "programmazione" (programming), "simulazione" (simulation), "tecniche" (techniques), "ArrayList" (ArrayList), and "LinkedList" (LinkedList). The words are colored in various shades of green, orange, red, and yellow, and their size varies to represent frequency or importance.

List

ArrayList vs. LinkedList

# Collection Family Tree



# ArrayList vs. LinkedList

|                                  | ArrayList | LinkedList    |
|----------------------------------|-----------|---------------|
| <code>add(element)</code>        | IMMEDIATE | IMMEDIATE     |
| <code>remove(object)</code>      | SLUGGISH  | LESS SLUGGISH |
| <code>get(index)</code>          | IMMEDIATE | SLUGGISH      |
| <code>set(index, element)</code> | IMMEDIATE | SLUGGISH      |
| <code>add(index, element)</code> | SLUGGISH  | SLUGGISH      |
| <code>remove(index)</code>       | SLUGGISH  | SLUGGISH      |
| <code>contains(object)</code>    | SLUGGISH  | SLUGGISH      |
| <code>indexOf(object)</code>     | SLUGGISH  | SLUGGISH      |
| <code>it.add()</code>            | SLUGGISH  | IMMEDIATE     |
| <code>it.remove()</code>         | SLUGGISH  | IMMEDIATE     |



# Big O notation

- ▶

## **O( $n$ )**

- ▶ Used to compare different implementation of a Collection
- ▶ **O( $n$ )** is used to note that the time required for the execution of an algorithm rises like  $n$
- ▶  $n$  is usually intended as the dimension of the data.

- ▶

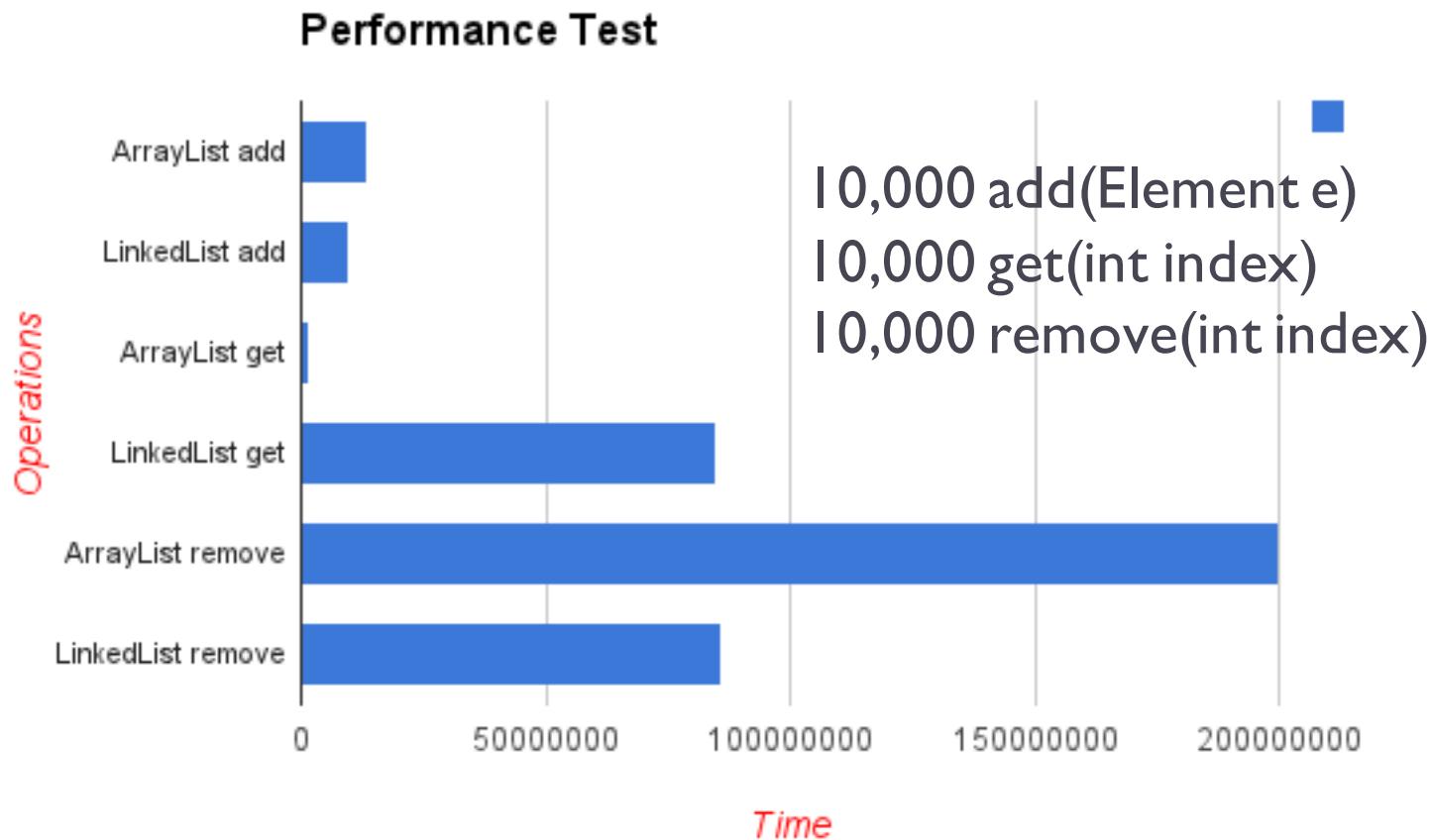
## Examples

- ▶ **O( $n^2$ )** takes a time that is quadratic-dependent by  $n$
- ▶ **O( $n$ )** takes a time that is linear-dependent by  $n$
- ▶ **O(log  $n$ )** takes a time that is dependent from the log  $n$
- ▶ **O(C)** or **O(1)** is a constant-time operation

# ArrayList vs. LinkedList

|                               | ArrayList     | LinkedList    |
|-------------------------------|---------------|---------------|
| <code>add(element)</code>     | $O(1)$        | $O(1)$        |
| <code>remove(object)</code>   | $O(n) + O(n)$ | $O(n) + O(1)$ |
| <code>get(index)</code>       | $O(1)$        | $O(n)$        |
| <code>set(index, elem)</code> | $O(1)$        | $O(n) + O(1)$ |
| <code>add(index, elem)</code> | $O(1) + O(n)$ | $O(n) + O(1)$ |
| <code>remove(index)</code>    | $O(n)$        | $O(n) + O(1)$ |
| <code>contains(object)</code> | $O(n)$        | $O(n)$        |
| <code>indexOf(object)</code>  | $O(n)$        | $O(n)$        |
| <code>it.add()</code>         | $O(n)$        | $O(1)$        |
| <code>it.remove()</code>      | $O(n)$        | $O(1)$        |

# ArrayList vs. LinkedList



\*source: <http://www.programcreek.com/2013/03/arraylist-vs-linkedlist-vs-vector/>



# ArrayList vs. LinkedList

- ▶ **ArrayList**
  - ▶ **get(index)** and **set(index, element)** are **O(1)**
  - ▶ **adding or removing** an element in last position are **O(1)**
  - ▶ **add(element)** with resize could cost **O(n)**
- ▶ **LinkedList**
  - ▶ **iterator.remove()** and **listIterator.add()** are **O(1)**
  - ▶ **adding or removing** an element in first position are **O(1)**
- ▶ **Memory footprint**
  - ▶ **LinkedList** uses more memory than an **ArrayList**

# Licenza d'uso



- ▶ Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)”
- ▶ Sei libero:
  - ▶ di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - ▶ di modificare quest'opera
- ▶ Alle seguenti condizioni:
  - ▶ **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - ▶ **Non commerciale** — Non puoi usare quest'opera per fini commerciali.
  - ▶ **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- ▶ <http://creativecommons.org/licenses/by-nc-sa/3.0/>