e-Lite

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;
}
```

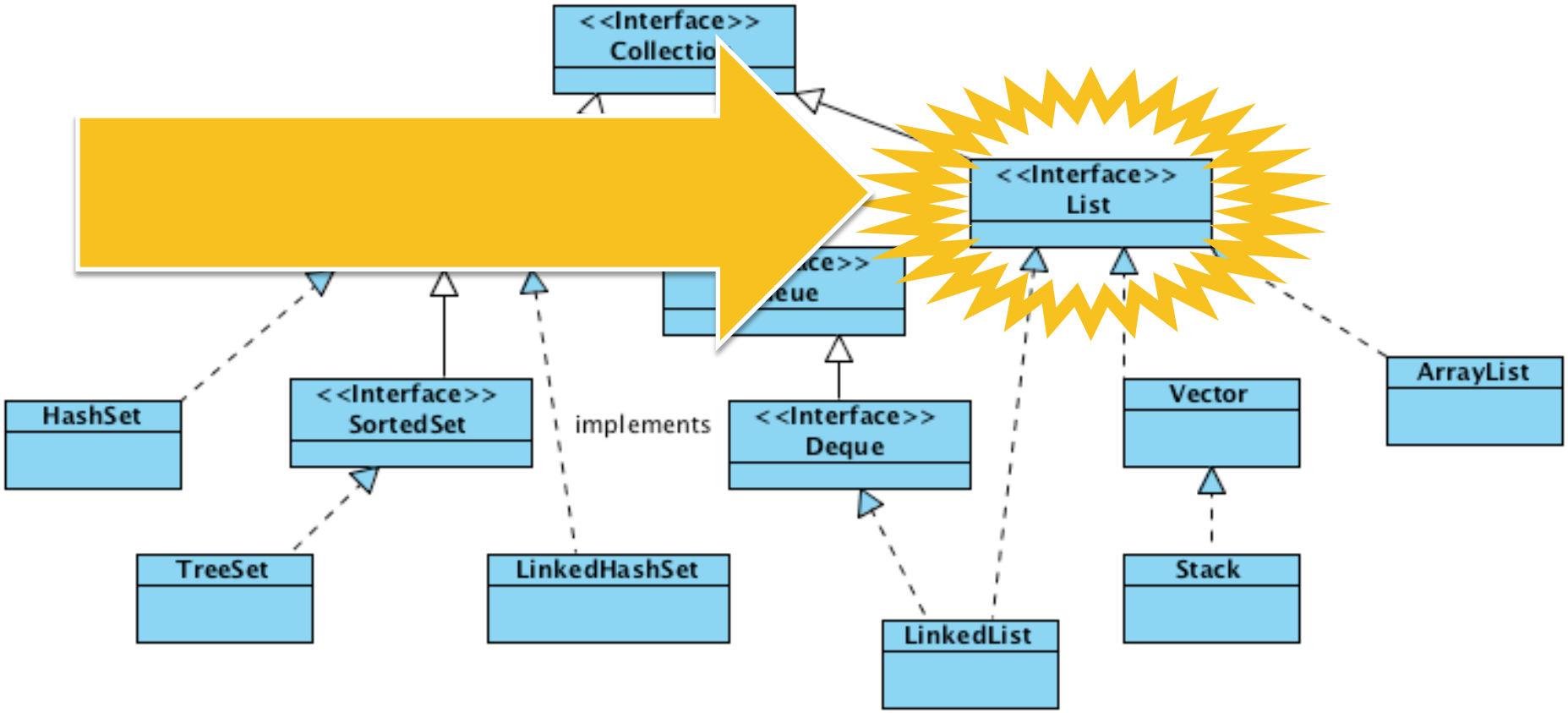
remove() method

```
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

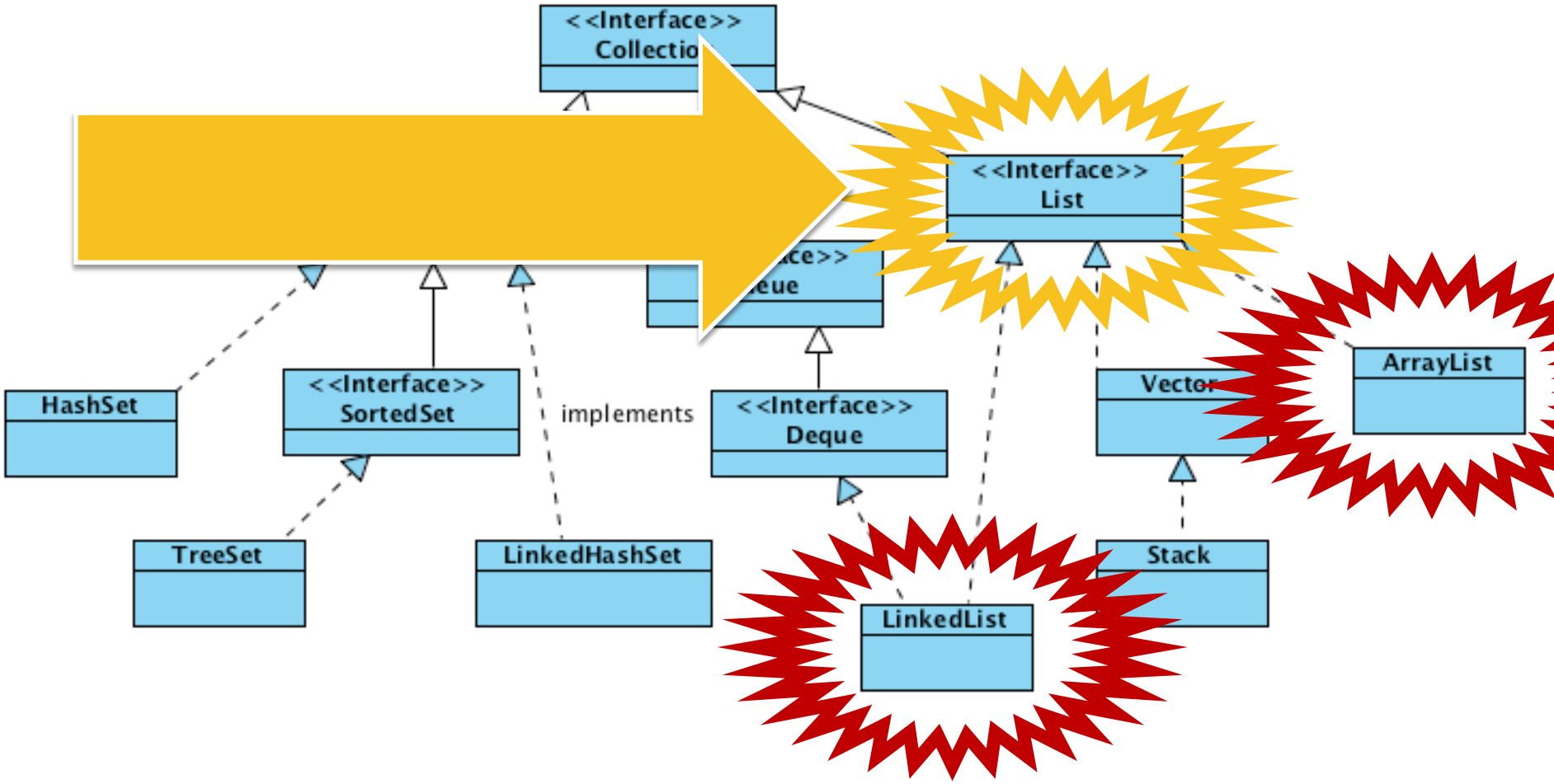
```
public void dump() {
    System.out.println("WORDS");

    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



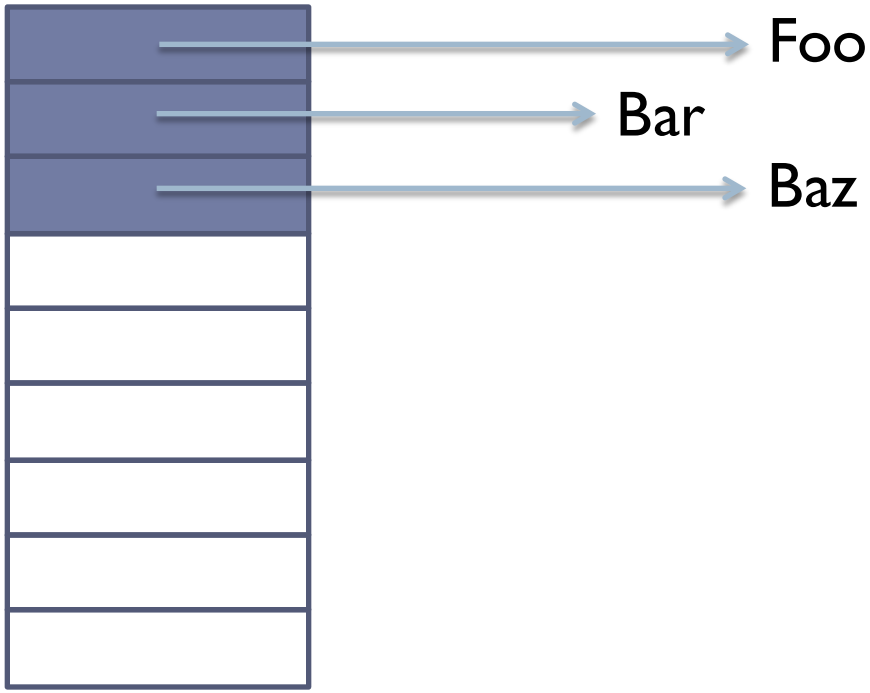
Data and constructor

- ArrayList

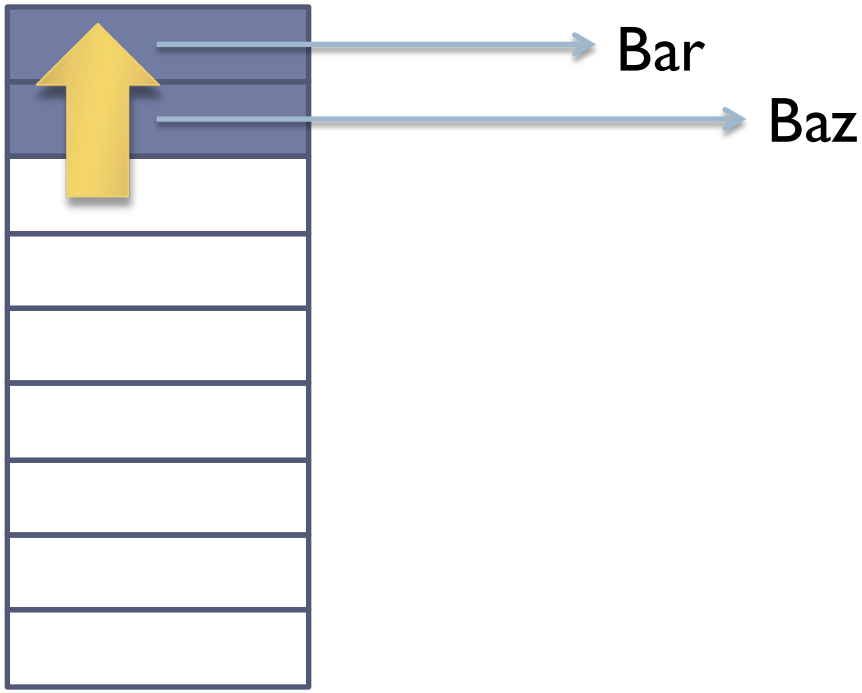
```
List<String> words;

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

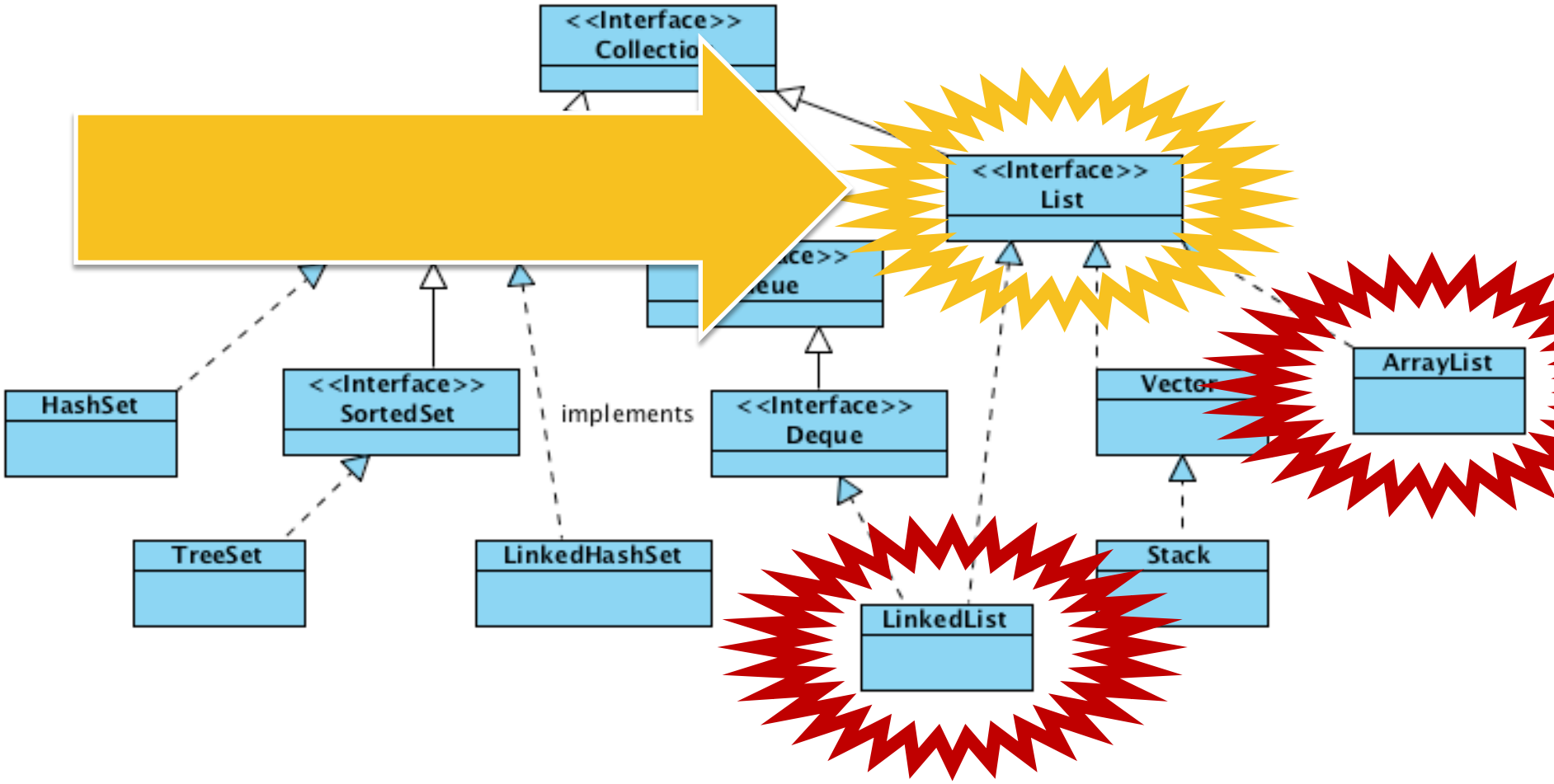
ArrayList



ArrayList – Delete



Collection Family Tree



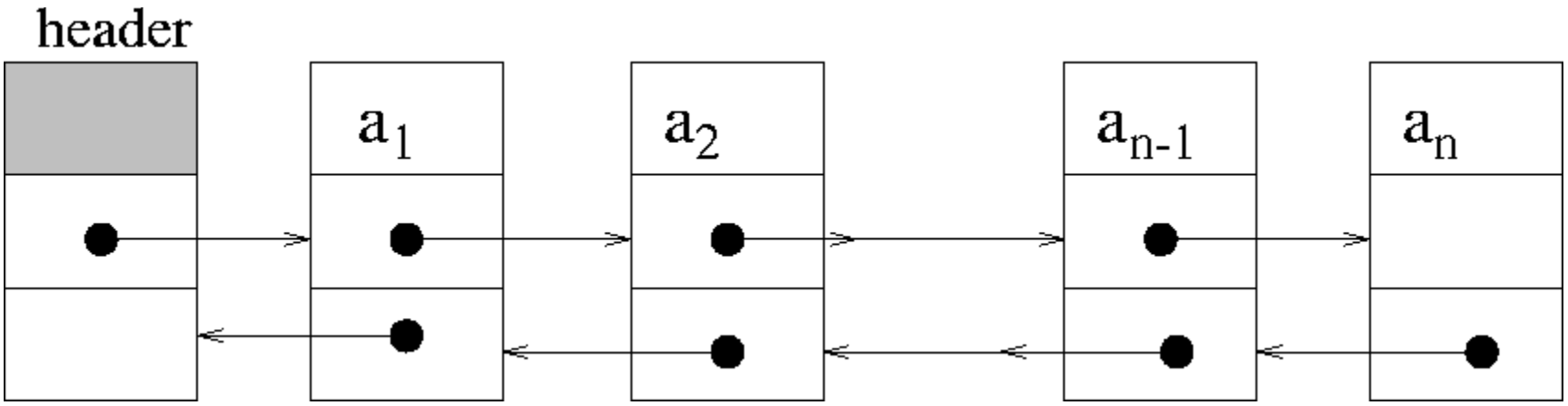
Data and constructor

- LinkedList

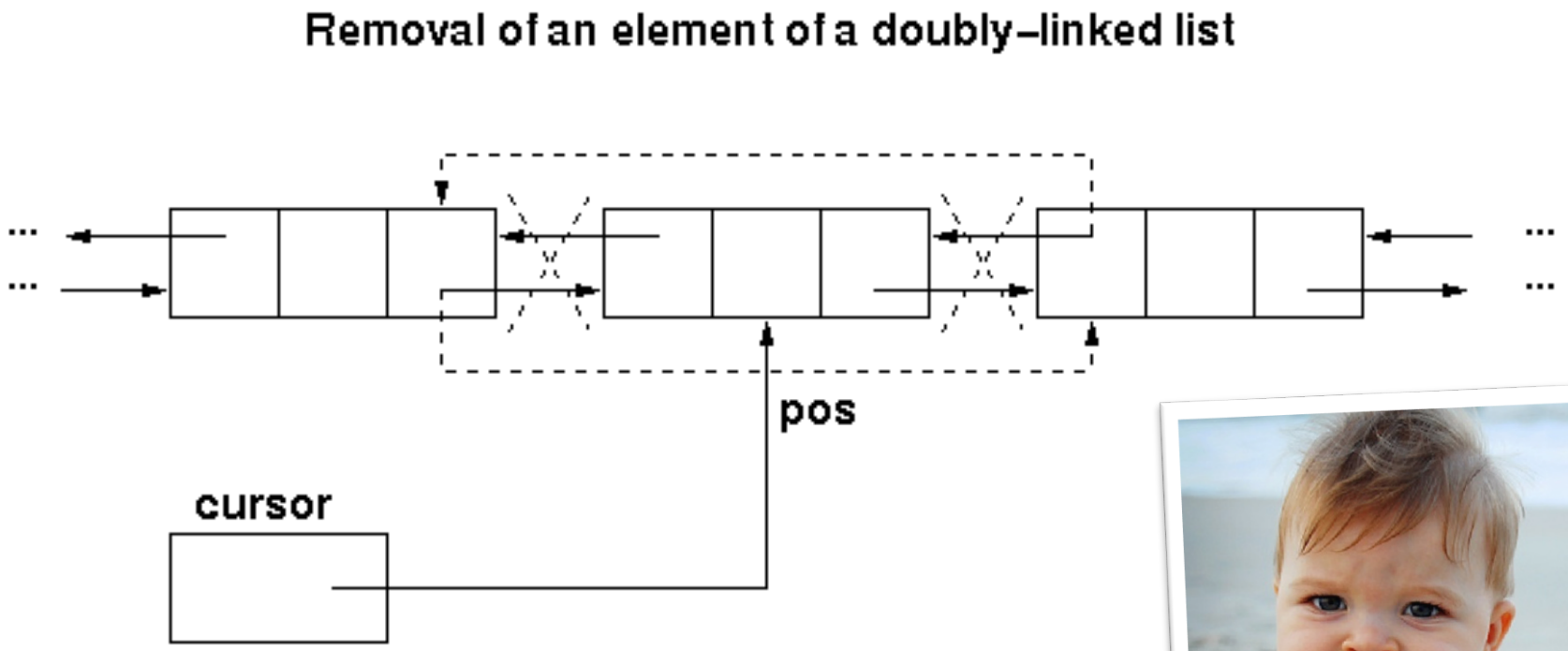
```
List<String> words;

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

LinkedList



LinkedList – Delete



ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)		
remove(object)		
get(index)		
set(index, element)		
add(index, element)		
remove(index)		
contains(object)		
indexOf(object)		

ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)	IMMEDIATE	IMMEDIATE
remove(object)		
get(index)		
set(index, element)		
add(index, element)		
remove(index)		
contains(object)		
indexOf(object)		

ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)	IMMEDIATE	IMMEDIATE
remove(object)	SLUGGISH	LESS SLUGGHISH
get(index)		
set(index, element)		
add(index, element)		
remove(index)		
contains(object)		
indexOf(object)		

ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)	IMMEDIATE	IMMEDIATE
remove(object)	SLUGGISH	LESS SLUGGHISH
get(index)	IMMEDIATE	SLUGGISH
set(index, element)	IMMEDIATE	SLUGGISH
add(index, element)		
remove(index)		
contains(object)		
indexOf(object)		

ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)	IMMEDIATE	IMMEDIATE
remove(object)	SLUGGISH	LESS SLUGGHISH
get(index)	IMMEDIATE	SLUGGISH
set(index, element)	IMMEDIATE	SLUGGISH
add(index, element)	SLUGGISH	SLUGGISH
remove(index)	SLUGGISH	SLUGGISH
contains(object)		
indexOf(object)		

ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)	IMMEDIATE	IMMEDIATE
remove(object)	SLUGGISH	LESS SLUGGHISH
get(index)	IMMEDIATE	SLUGGISH
set(index, element)	IMMEDIATE	SLUGGISH
add(index, element)	SLUGGISH	SLUGGISH
remove(index)	SLUGGISH	SLUGGISH
contains(object)	SLUGGISH	SLUGGISH
indexOf(object)	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

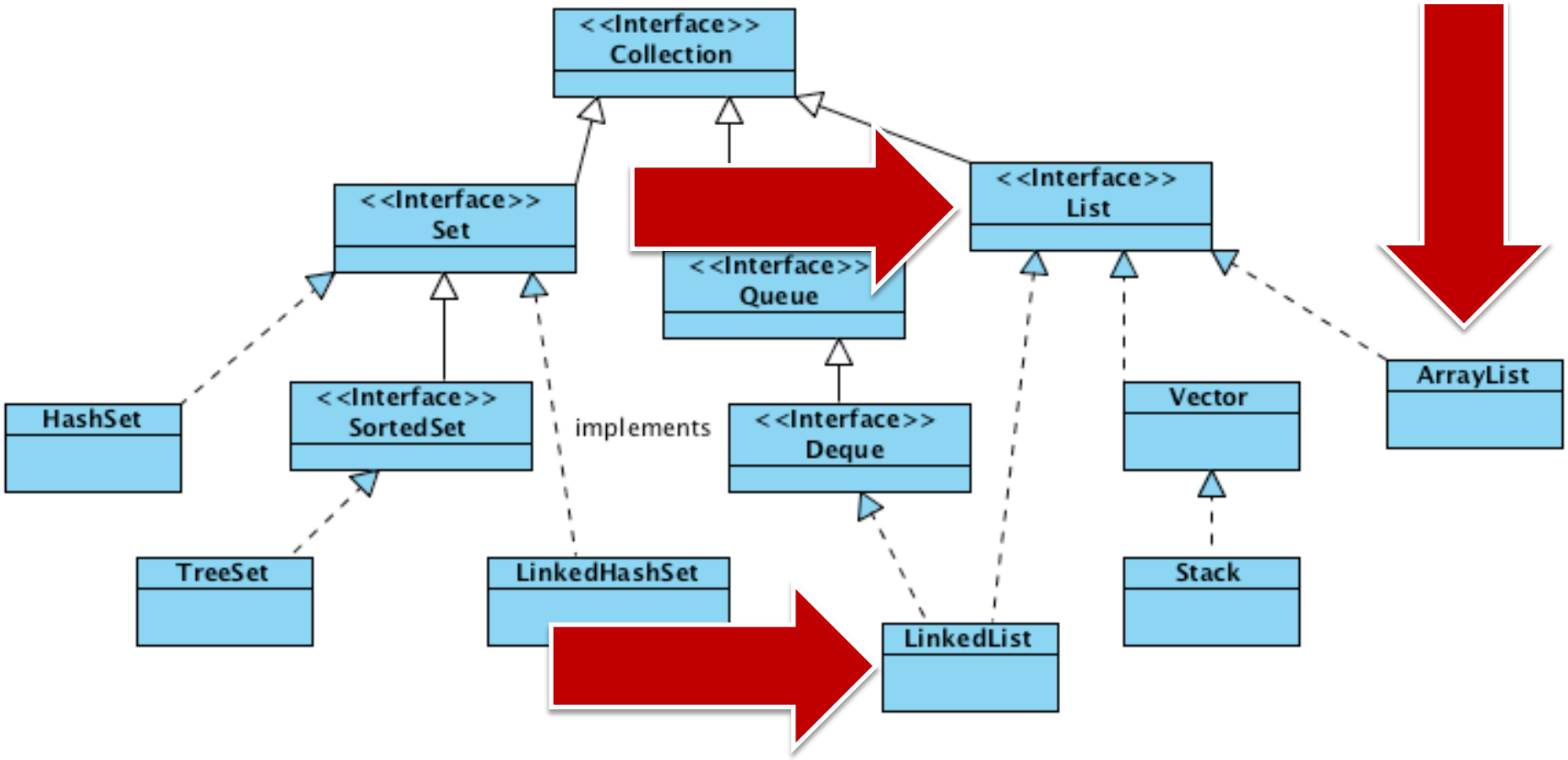
```
import java.util.UUID;

String val = UUID.randomUUID().toString());
```

List

ArrayList vs. LinkedList

Collection Family Tree



ArrayList vs. LinkedList

	ArrayList	LinkedList
add(element)	IMMEDIATE	IMMEDIATE
remove(object)	SLUGGISH	LESS SLUGGISH
get(index)	IMMEDIATE	SLUGGISH
set(index, element)	IMMEDIATE	SLUGGISH
add(index, element)	SLUGGISH	SLUGGISH
remove(index)	SLUGGISH	SLUGGISH
contains(object)	SLUGGISH	SLUGGISH
indexOf(object)	SLUGGISH	SLUGGISH
it.add()	SLUGGISH	IMMEDIATE
it.remove()	SLUGGISH	IMMEDIATE

Big O notation

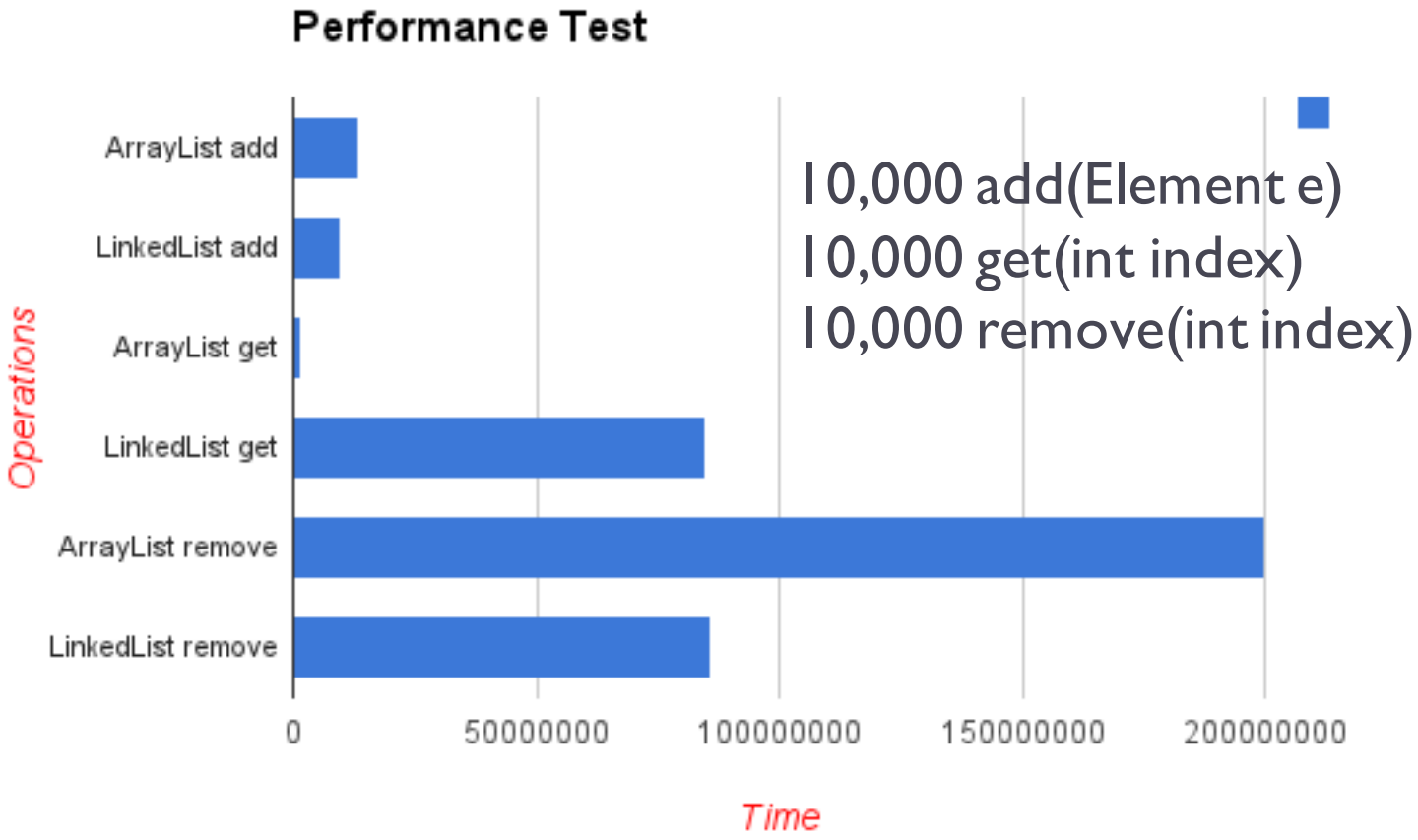
- O(*n*)
  - Used to compare different implementation of a Collection
  - **O(*n*)** is used to note that the time required for the execution of an algorithm rises like *n*
  - *n* is usually intended as the dimension of the data.

- 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
add(element)	O(1)	O(1)
remove(object)	O(n) + O(n)	O(n) + O(1)
get(index)	O(1)	O(n)
set(index, elem)	O(1)	O(n) + O(1)
add(index, elem)	O(1) + O(n)	O(n) + O(1)
remove(index)	O(n)	O(n) + O(1)
contains(object)	O(n)	O(n)
indexOf(object)	O(n)	O(n)
it.add()	O(n)	O(1)
it.remove()	O(n)	O(1)

ArrayList vs. LinkedList

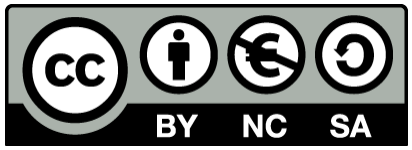


*source: http://www.programcreek.com/2013/03/arraylist-vs-linkedlist-vs-vector/

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 i 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 un licenza identica o equivalente a questa.

- http://creativecommons.org/licenses/by-nc-sa/3.0/