dati
ottimizzazione
problemi
applicazioni
strutture
java
algoritmi
programmazione
tecniche
simulazione
ricerca
gestione
laboratorio
problem
linguaggio
relazionali
problema
corso
conoscenza
complessità
software
realizzazione
basi
capacità
grafiche
liste
soluzione
tipo
librerie
oggetti
comprensione
risolvere
progettazione
scheduling
swing
efficace
specifiche
code
greedy
interfacce
allocazione
esercitazioni
standard
complementi
ottimale
filone
min-max
risoluzione
solving
ricorsione
efficiente
mediante
aula
campo
gestionale
programmi
particolare
quattro
vista
complesse
alberi
grafi
utilizzo
divide
accesso
propone
punti
reti
sviluppo
applicativi
impera
grado
diversi
jdbc
informatico
attraverso
programma
algoritmica
richiami
testi
reali
filoni
verifica
risorse
durante
partire
algoritmiche
installazione
svolgimento

Wrap-up

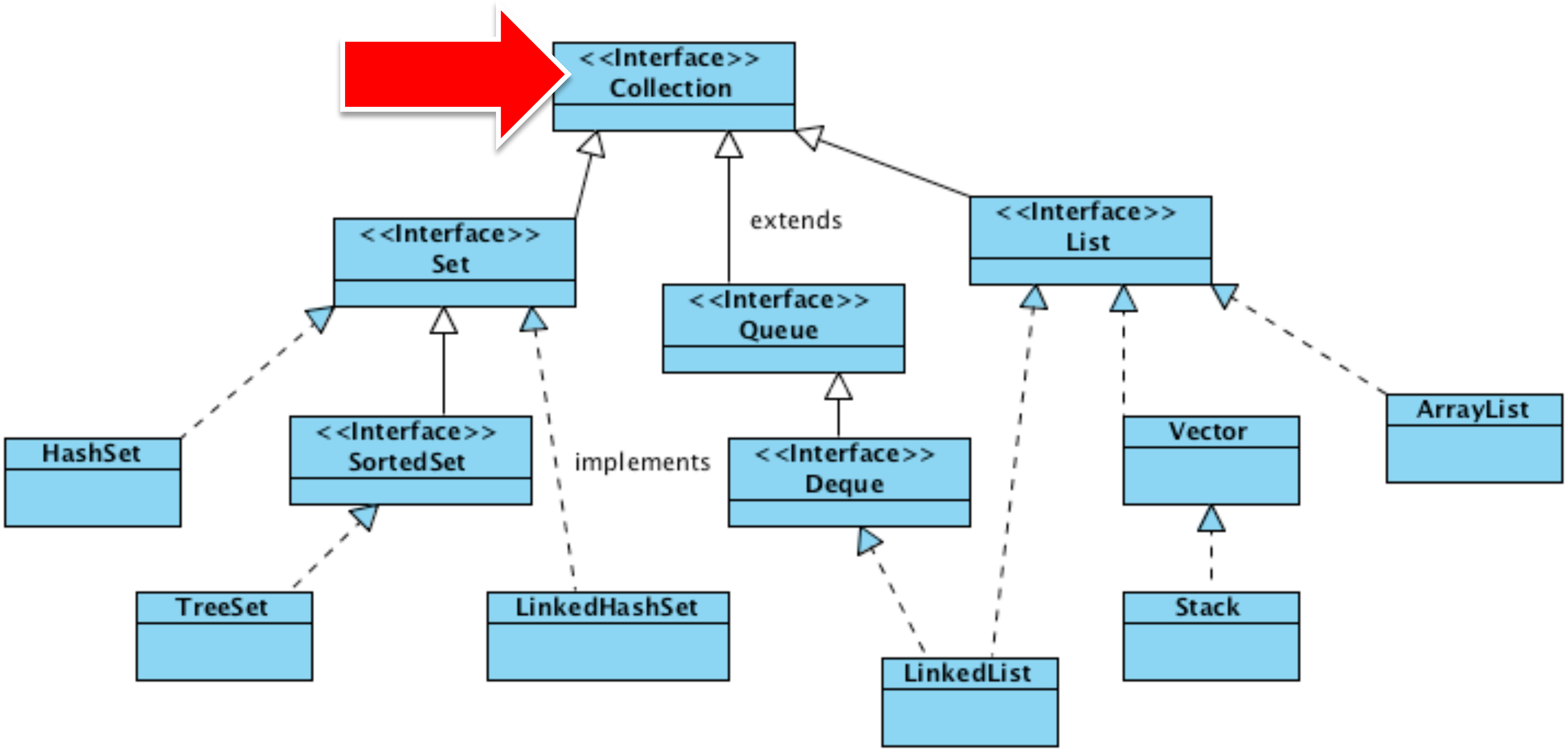
Java Collection Framework

- The core collection interfaces are the foundation of the Java Collections Framework (JCF).

- The Java Collections Framework hierarchy consists of two distinct interface trees:
  - The first tree starts with the **Collection** interface, which provides for the basic functionality used by all collections (e.g. add, remove)
  - The second tree starts with the **Map** interface, which maps keys and values.

- These interfaces allow collections to be manipulated independently of the details of their representation.

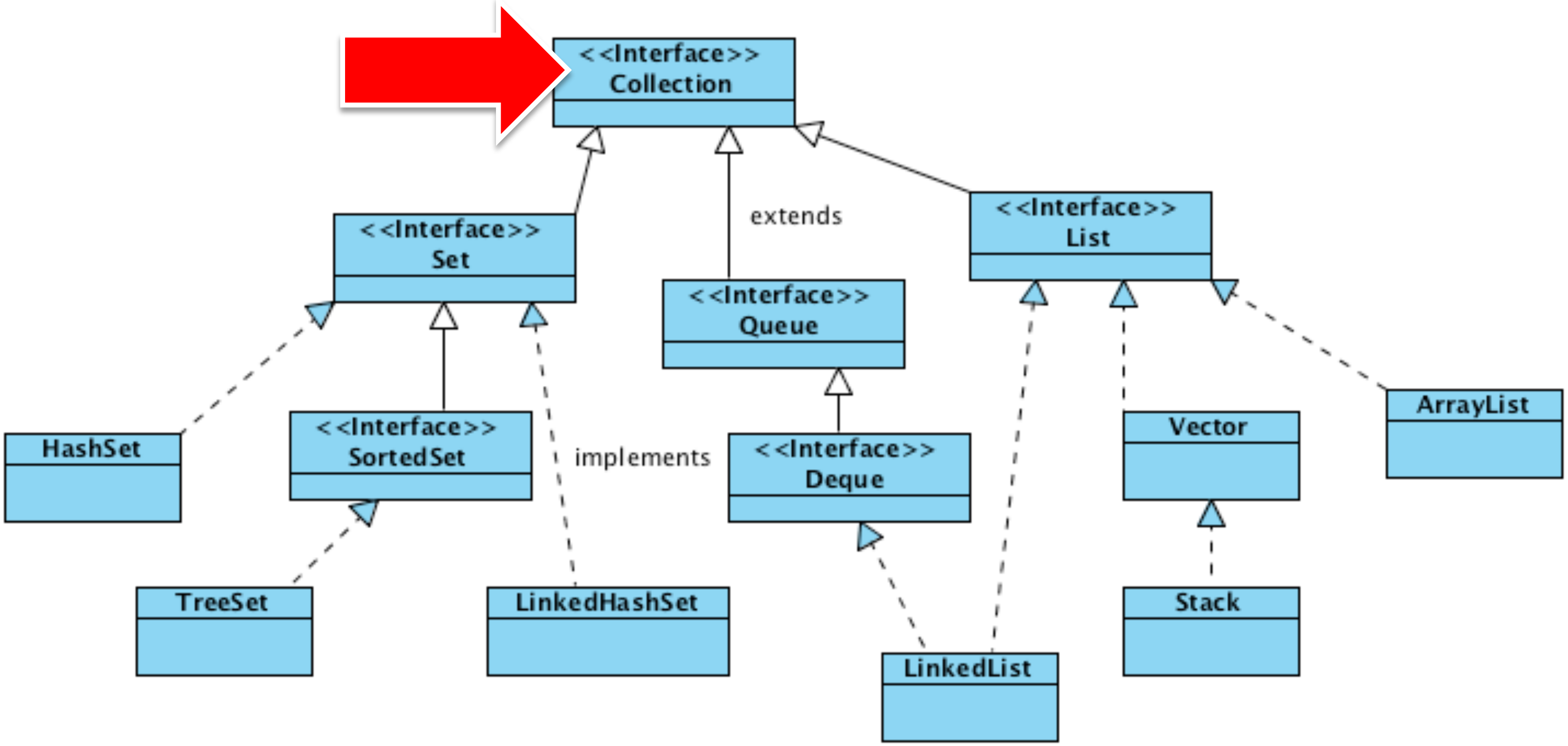
Java Collection Framework



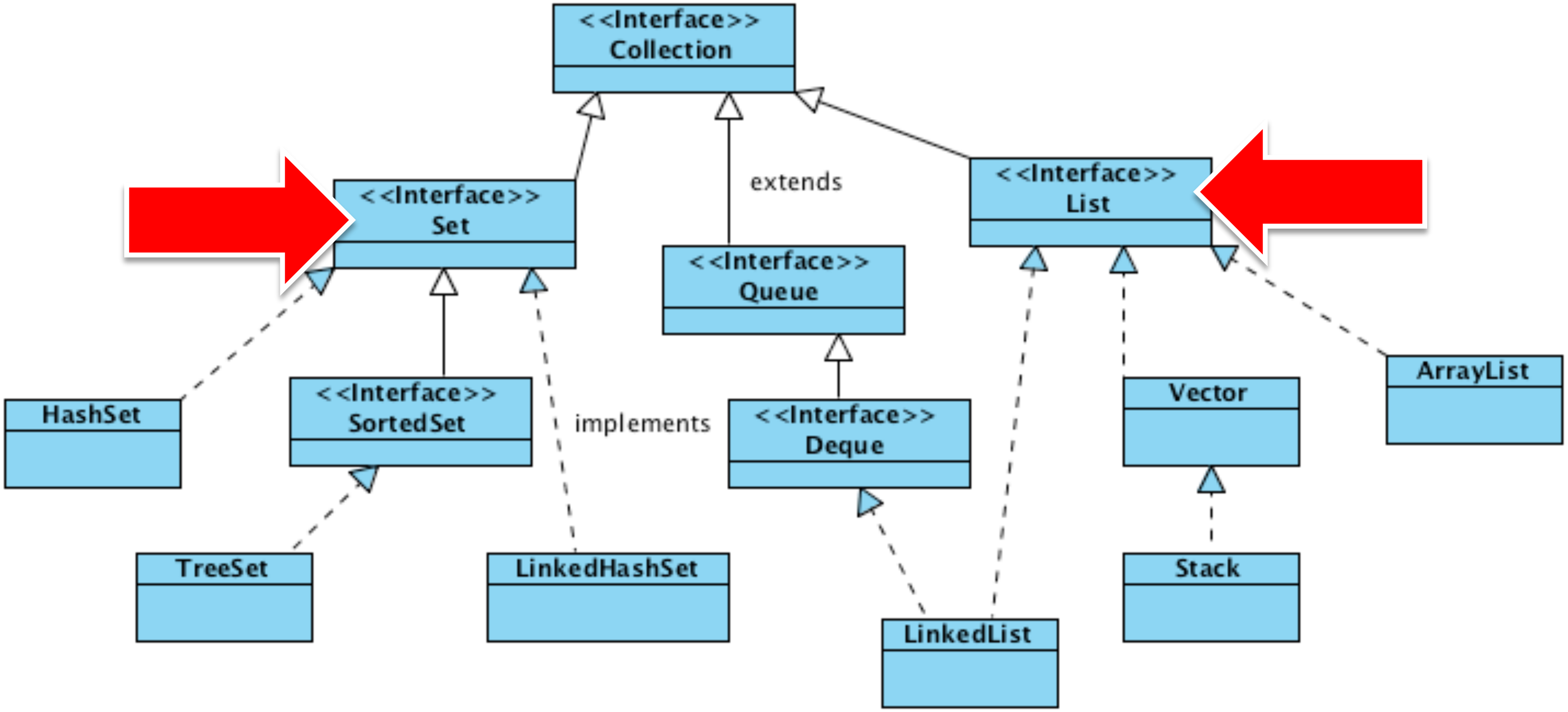
The first tree: the Collection interface

- Its subinterfaces provide for more specialized collections
- The Set interface does not allow duplicate elements.This can be useful for storing collections such as a deck of cards or student records. The Set interface has a subinterface, SortedSet, that provides for ordering of elements in the set
- The List interface provides for an ordered collection, for situations in which you need precise control over where each element is inserted. You can retrieve elements from a List by their exact position
- The Queue interface enables additional insertion, extraction, and inspection operations. Elements in a Queue are typically ordered in on a FIFO basis.
- The Deque interface enables insertion, deletion, and inspection operations at both the ends. Elements in a Deque can be used in both LIFO and FIFO.

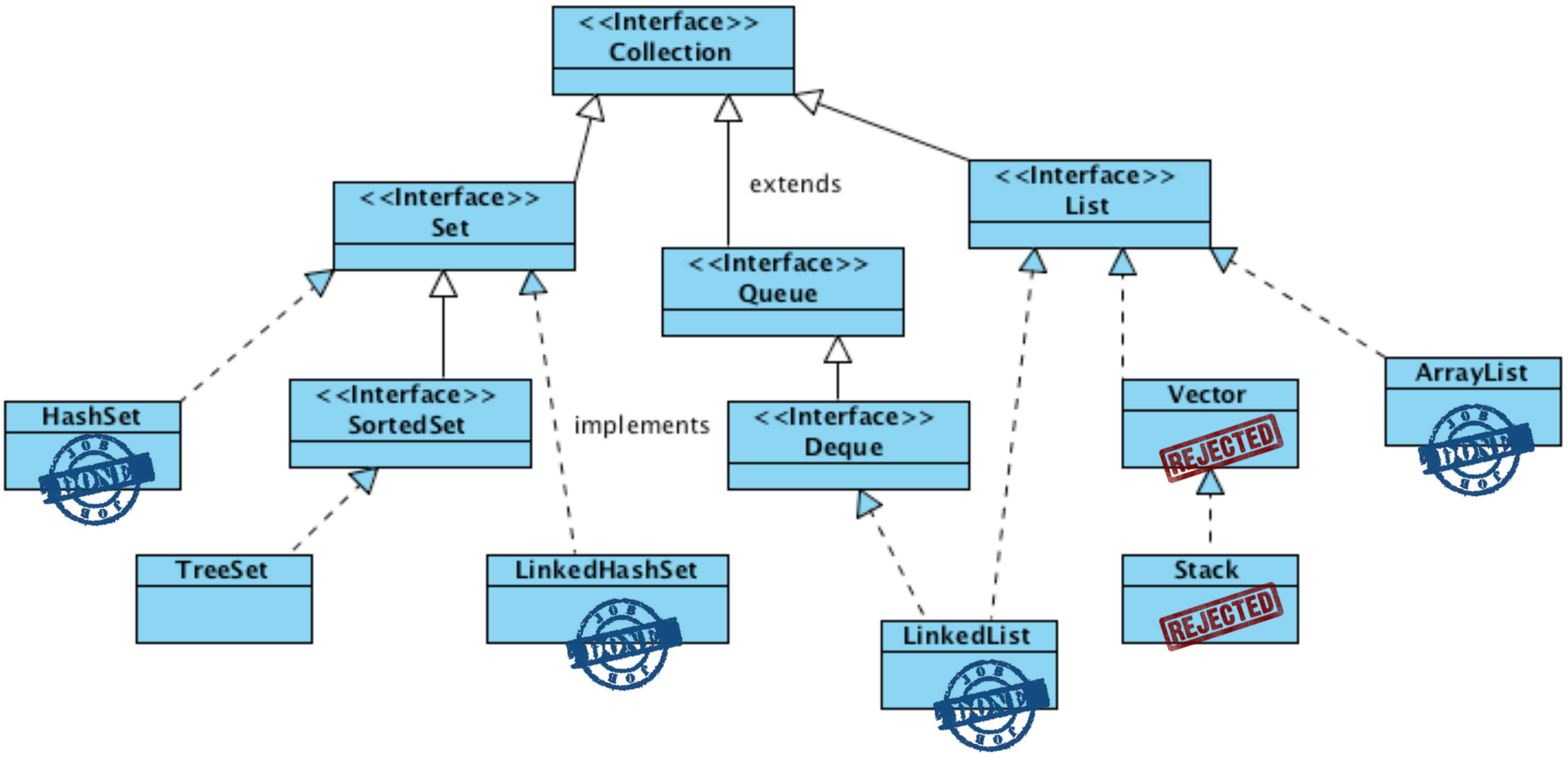
Java Collection Framework



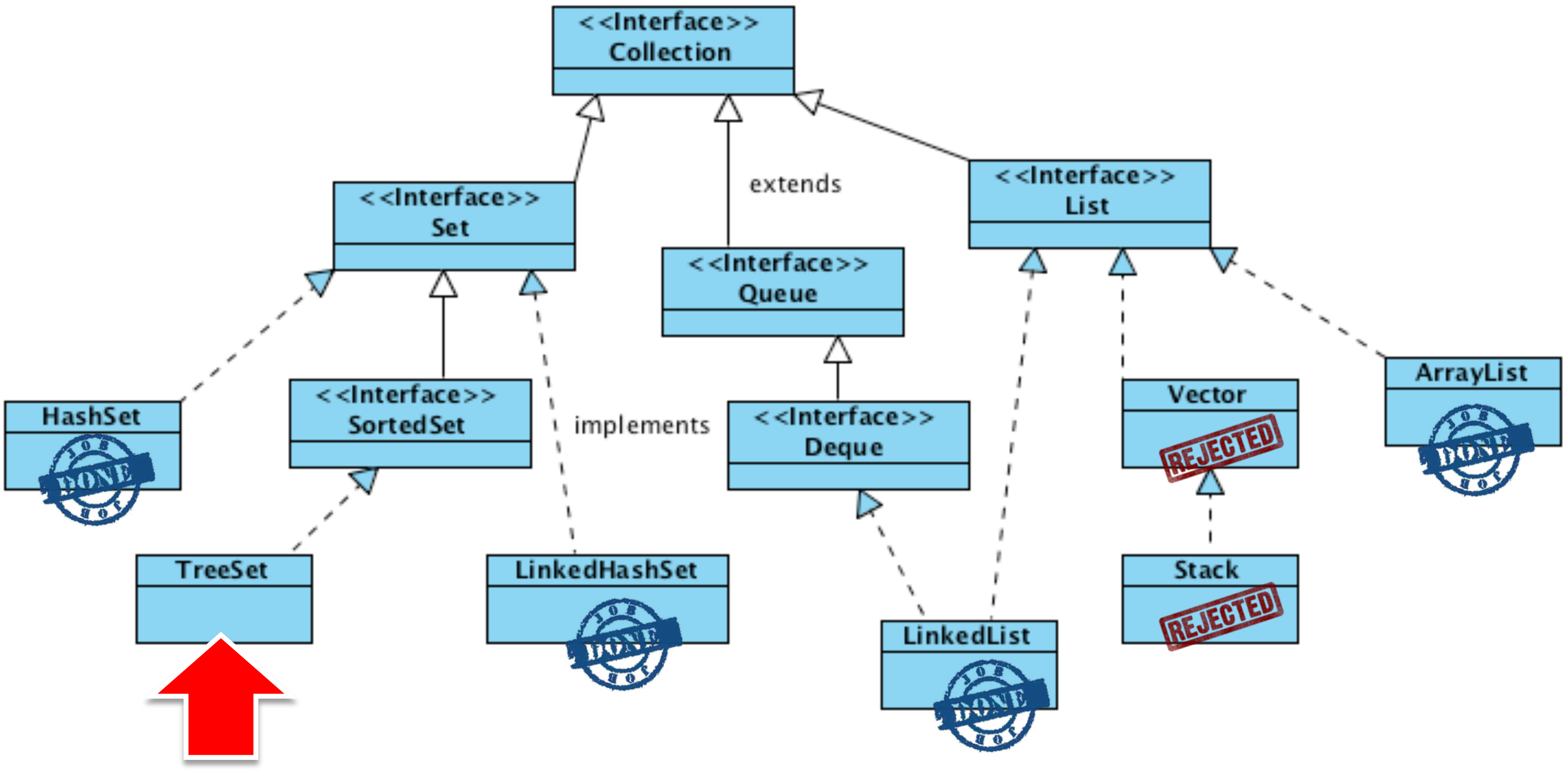
Java Collection Framework



Java Collection Framework



Java Collection Framework

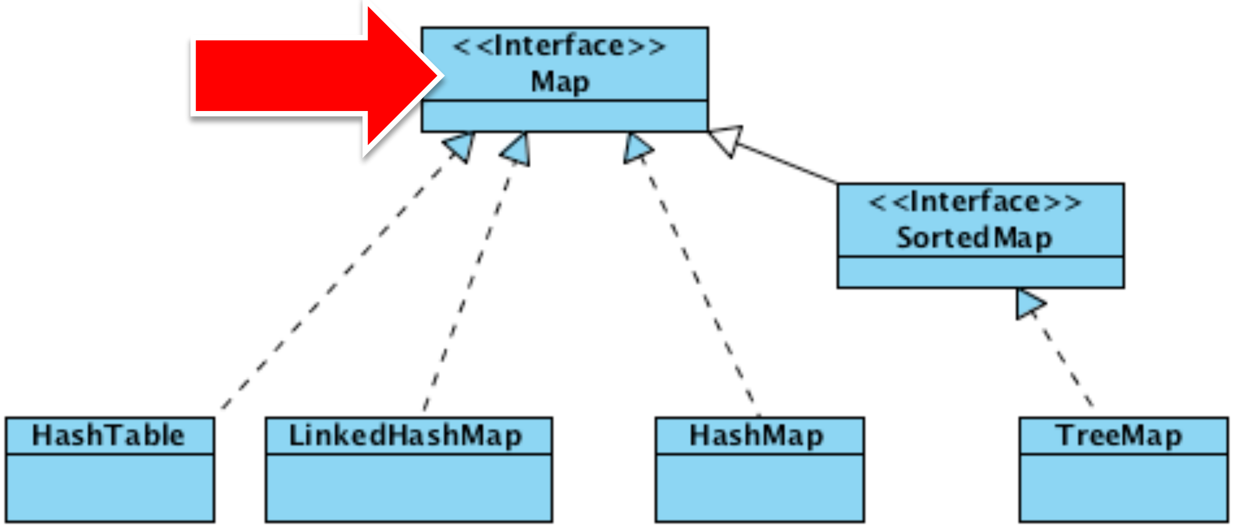


TODO

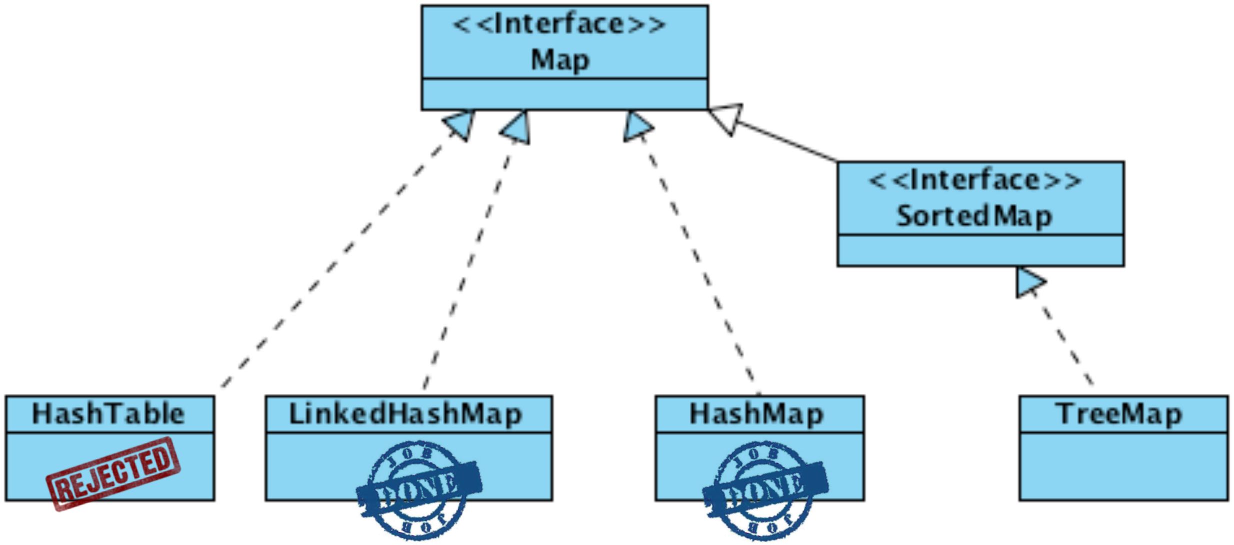
The second tree: the Map interface

- The second tree starts with the Map interface, which maps keys and values similar to a Hashtable
- Map's subinterface, SortedMap, maintains its key-value pairs in ascending order or in an order specified by a Comparator.

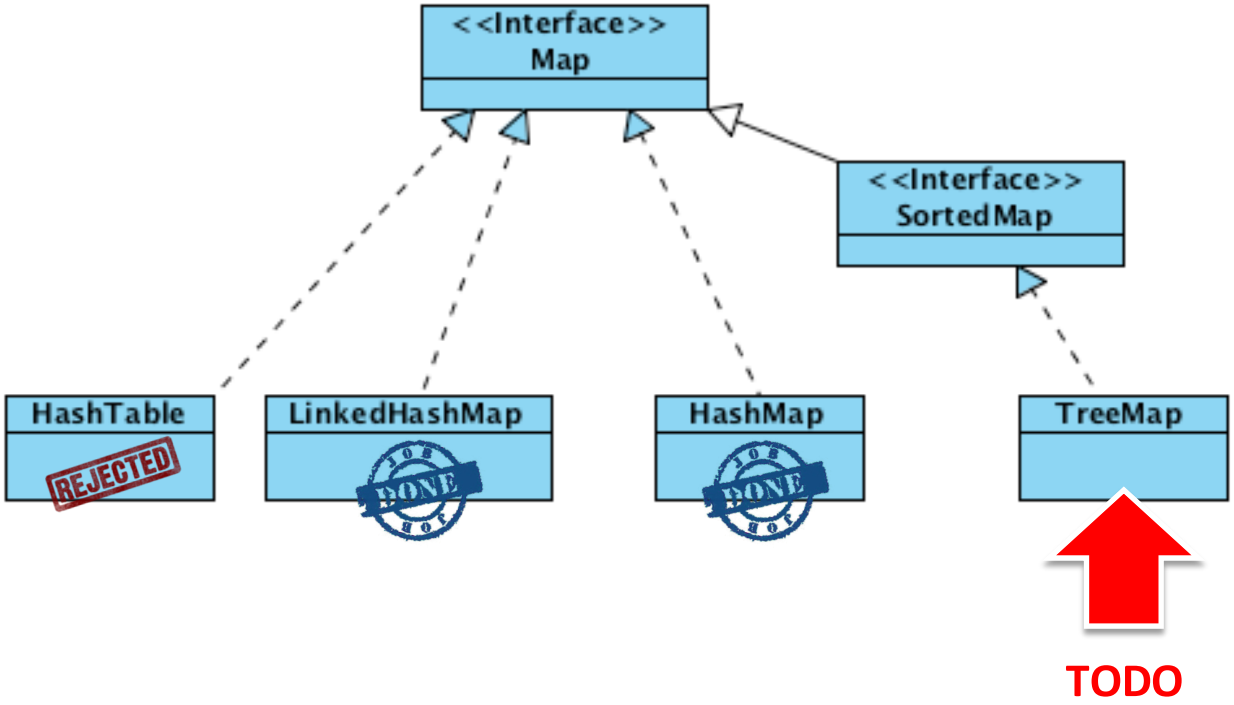
Java Collection Framework



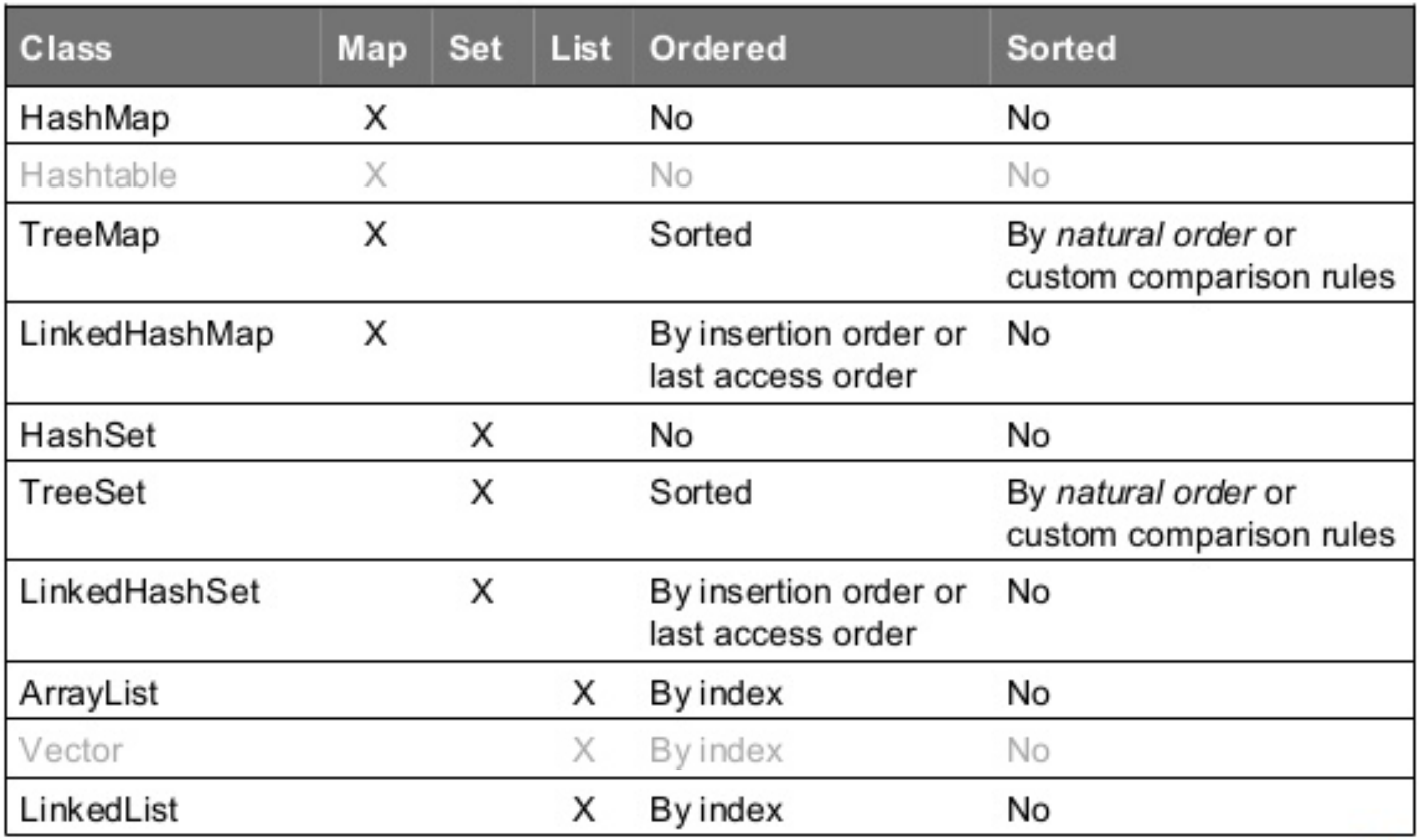
Java Collection Framework



Java Collection Framework



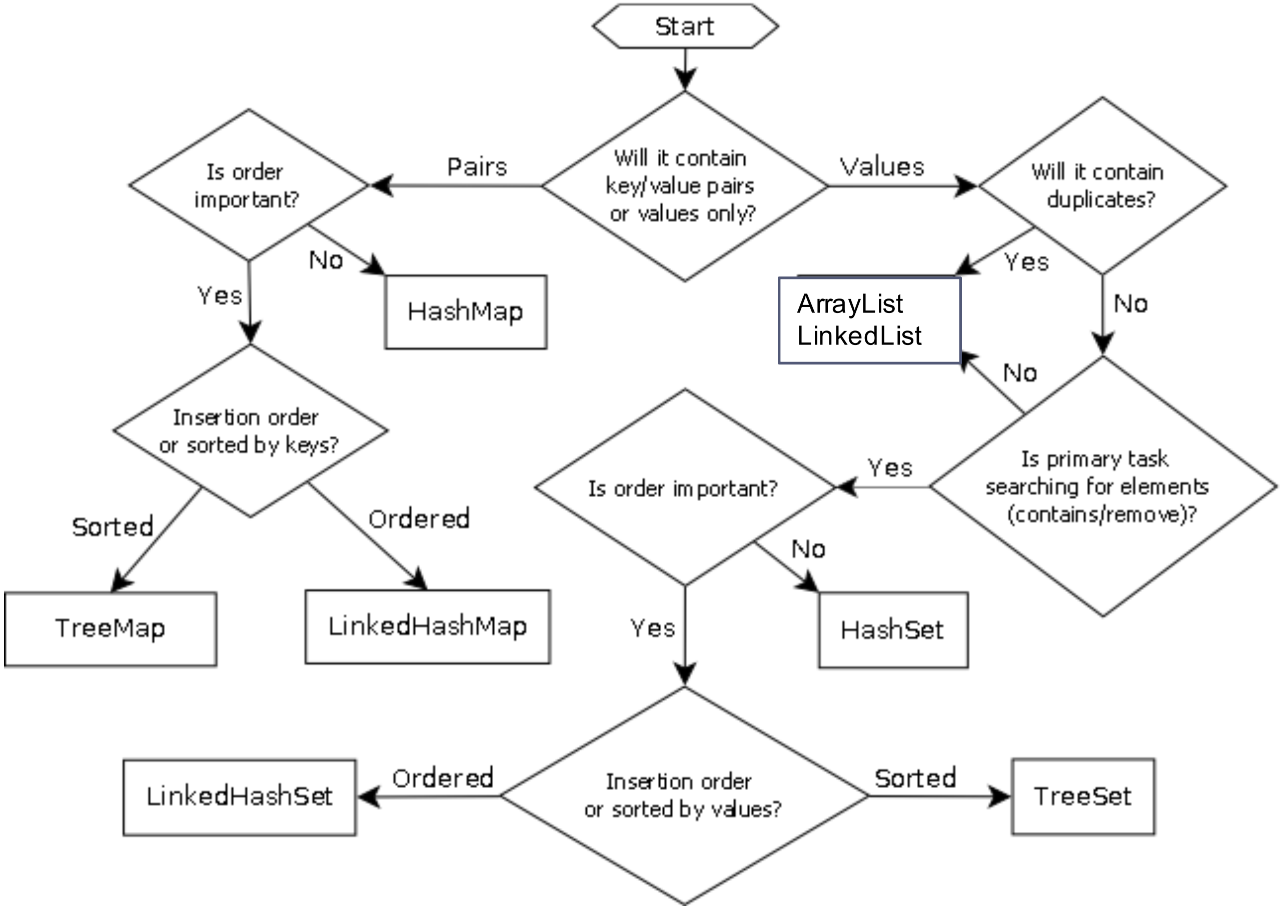
Java Collection Framework



Class	Map	Set	List	Ordered	Sorted
HashMap	X			No	No
Hashtable	X			No	No
TreeMap	X			Sorted	By *natural order* or custom comparison rules
LinkedHashMap	X			By insertion order or last access order	No
HashSet		X		No	No
TreeSet		X		Sorted	By *natural order* or custom comparison rules
LinkedHashSet		X		By insertion order or last access order	No
ArrayList			X	By index	No
Vector			X	By index	No
LinkedList			X	By index	No

*source: https://www.slideshare.net/cpdindia2/collection-framework-in-java

Java Collection Framework


Start
Will it contain key/value pairs or values only?
Pairs
Values
Is order important?
No
HashMap
Yes
Insertion order or sorted by keys?
Sorted
Ordered
TreeMap
LinkedHashMap
Will it contain duplicates?
Yes
ArrayList
LinkedList
No
No
Is primary task searching for elements (contains/remove)?
Yes
Is order important?
No
HashSet
Yes
Insertion order or sorted by values?
Ordered
LinkedHashSet
Sorted
TreeSet

*source: https://stackoverflow.com/questions/48442/rule-of-thumb-for-choosing-an-implementation-of-a-java-collection

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

Lists vs. Sets

	ArrayList	LinkedList	Set
add(element)	O(1)	O(1)	O(1)
remove(object)	O(n) + O(n)	O(n) + O(1)	O(1)
get(index)	O(1)	O(n)	n.a.
set(index, elem)	O(1)	O(n) + O(1)	n.a.
add(index, elem)	O(1) + O(n)	O(n) + O(1)	n.a.
remove(index)	O(n)	O(n) + O(1)	n.a.
contains(object)	O(n)	O(n)	O(1)
indexOf(object)	O(n)	O(n)	n.a.

Map

	HashMap
put(key, object)	O(1)
get(key)	O(1)
remove(key)	O(1)
containsKey(key)	O(1)
containsValue(object)	O(N)

Recap

- == or !=
  - Used to compare the references of two objects

```
MyData foo = new MyData();
MyData bar = new MyData();

if(foo != bar) {
    System.out.println("References are different");
}

if(foo == bar){
    System.out.println("References are equal");
}
```

Recap

- **equals()**
  - Used to give **equality** information about the objects

```
MyData foo = new MyData();
MyData bar = new MyData();

if(foo.equals(bar)) {
    System.out.println("Objects have the same values");
} else {
    System.out.println("Objects have different values");
}
```

Recap

- hashCode()
  - Return the hash value of an object
  - Must behave in a way consistent with the same object equals() method

```
MyData foo = new MyData();
MyData bar = new MyData();

if(foo.equals(bar)) {
    if(foo.hashCode() == bar.hashCode()) {
        System.out.println("Hash code must be equal")
    }
}
```

Recap

- compareTo()
  - Gives the ordering of objects
  - Must be used only if need to order the object in a collection

```
MyData foo = new MyData();
MyData bar = new MyData();

if (foo.compareTo(bar) == 0){
   // WRONG!!
}
```

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 ur licenza identica o equivalente a questa.
- http://creativecommons.org/licenses/by-nc-sa/3.0/