# RIPRENDENDO GLI ESEMPI: Java

Java

Nell'esercizio n. 1 (**`Set`**) si può scegliere fra:

- **`HashSet`**: insieme non ordinato, tempo d'accesso costante
- **`TreeSet`**: insieme ordinato, tempo di accesso non costante

Output con `HashSet`:

```
>java FindDups Io sono Io esisto Io parlo
Parola duplicata: Io
Parola duplicata: Io
4 parole distinte: [Io, parlo, esisto, sono]
```

ordine qualunque

Output con `TreeSet`:

```
>java FindDups Io sono Io esisto Io parlo
Parola duplicata: Io
Parola duplicata: Io
4 parole distinte: [Io, esisto, parlo, sono]
```

ordine alfabetico!

# RIPRENDENDO GLI ESEMPI: Java

Negli esercizi n. 2 e 3 (**`List`**) si può scegliere fra:

Java

- **`ArrayList`**: i principali metodi eseguono in tempo costante, mentre gli altri eseguono in un tempo lineare, ma con una costante di proporzionalità molto più bassa di **`LinkedList`**.
- **`LinkedList`**: il tempo di esecuzione è quello di una tipica realizzazione basata su puntatori; implementa anche le interfacce **`Queue`** e **`Deque`**, offrendo così una coda FIFO
- `Vector`: versione reingegnerizzata e sincronizzata di **`ArrayList`**

L'output ovviamente *non varia* al variare dell'implementazione, in ossequio sia al concetto di lista come *sequenza* di elementi, sia alla semantica di **`add`** come *append.*

# RIPRENDENDO GLI ESEMPI: Java

Java

Negli esercizi n. 4 e 5 (**`Map`**) si può scegliere fra:

- **`HashMap`**: tabella non ordinata, tempo d'accesso costante
- **`TreeMap`**: tabella ordinata, tempo di accesso non costante

Output con `HashMap`:

```
>java ContaFrequenza cane gatto cane pesce gatto gatto cane
3 parole distinte: {cane=3, pesce=1, gatto=3}
```

Output con `TreeMap` *(elenco ordinato)*:

```
>java ContaFrequenzaOrd cane gatto cane pesce gatto gatto cane
3 parole distinte: {cane=3, gatto=3, pesce=1}
```