

# Programmazione ad Oggetti

---

Collezioni:  
Mappe generiche

# Concetti introdotti

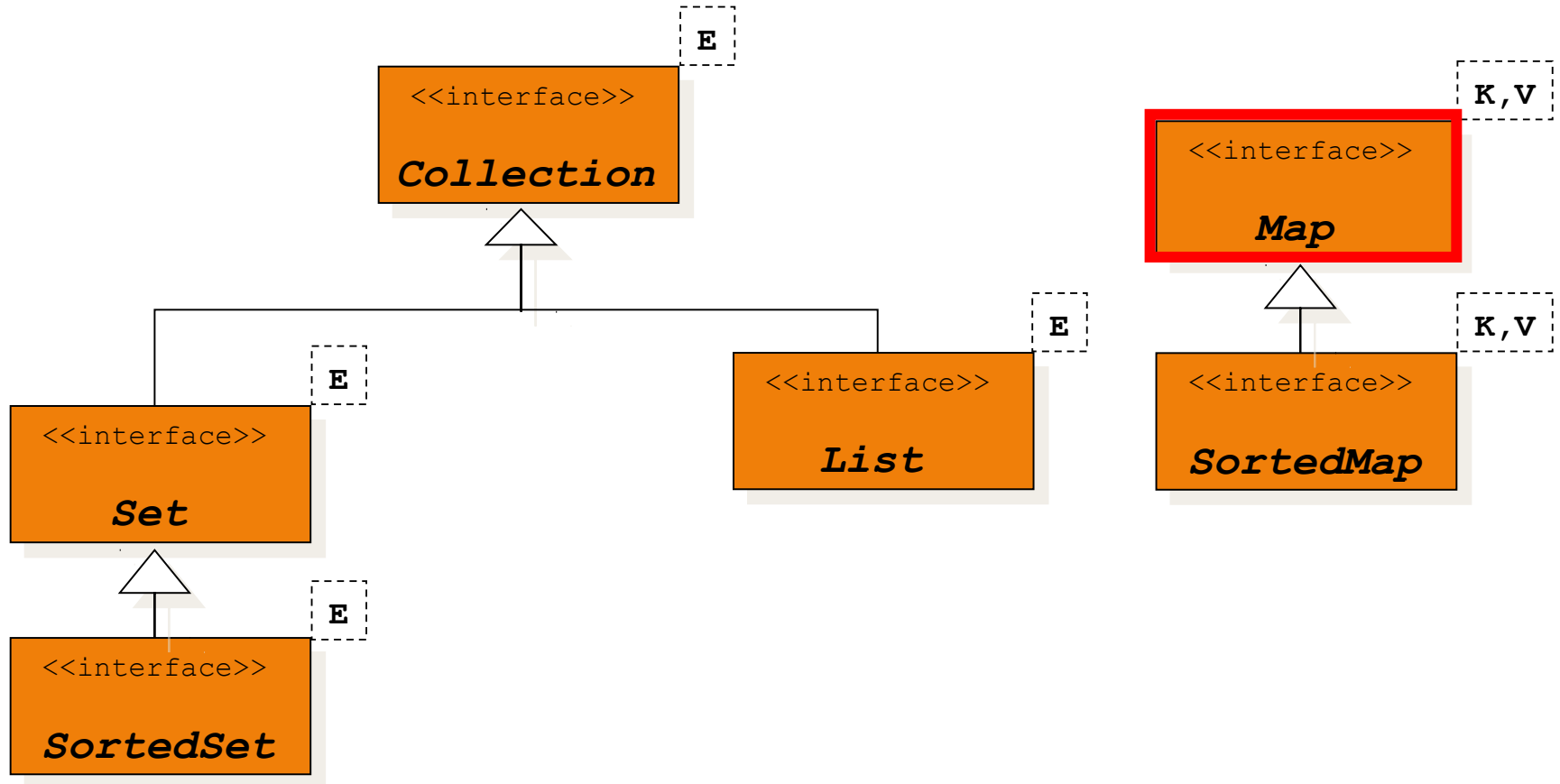
- Interface **Map<K, V>**
  - Operazioni tipiche sulle mappe
- Implementazioni di **Map<K, V>**
  - unicità delle chiavi

# Concetti introdotti

- **Interface Map<K, V>**
  - Operazioni tipiche sulle mappe
- Implementazioni di **Map<K, V>**
  - unicità delle chiavi

# Collezioni: Interface

- Le principali interface del package `java.util`



- Per ognuna di queste interface il package offre diverse implementazioni




# Mappe

- Una mappa (o dizionario, o array associativo) è una collezione di coppie chiave-valore (entry)
- Le chiavi, in quanto tali, sono uniche
  - In una mappa non possono esistere due chiavi eguali
- Ad ogni chiave può essere associato un solo valore
  - Ma il valore può essere una collezione
- Esempi:
  - rubrica telefonica: a ciascuna persona (chiave) è associato un numero di telefono (il numero è il valore)
  - Indice analitico di un libro: ad ogni voce (chiave) è associato un insieme di numeri di pagine (l'insieme è il valore)

# Mappe: operazioni principali

- Operazioni tipiche su una mappa sono:
  - Inserimento di una coppia chiave-valore
    - Inserisci nella rubrica: "Paolo Rossi" -> 0288821212
  - Richiesta del valore associato ad una chiave
    - Dammi il numero di telefono di "Elena Rondi"
  - Aggiornamento di un valore
    - Cambia il numero di telefono di "Paolo Rossi" in 063232122
  - Rimozione di una coppia chiave-valore
    - Rimuovi "Fabio Capello" dalla mia rubrica
  - Verifica di esistenza di una chiave
    - C'è "Bill Gates" nella mia rubrica?
  - Richiesta dell'insieme delle chiavi
    - Dammi i nomi di tutte le persone della mia rubrica

# I metodi base di Map<K, V>

- **V put(K chiave, V valore) ;**  
inserisce la coppia chiave-valore nella mappa;
  - se la chiave esiste già, allora il valore viene aggiornato e il metodo ritorna il valore vecchio
  - se la chiave non esiste viene inserita una nuova coppia, e il metodo ritorna `null`
- **V get(Object  chiave) ;**  
restituisce il valore associato alla chiave, `null` se la chiave non esiste nella mappa
- **V remove(Object  chiave) ;**  
rimuove la coppia associata alla chiave e lo ritorna il valore
- **boolean containsKey(Object  chiave) ;**  
verifica se la chiave è presente nella mappa

# Metodi bulk e viste di Map<K, V>

- **void putAll (Map<K, V> mappa)**  
inserisce nella mappa tutte le coppie della mappa passata come parametro
- **void clear ()**  
Elimina tutte le coppie dalla mappa
- **Set<K> keySet ()**  
Restituisce in un insieme tutte le chiavi
- **Collection<V> values ()**  
Restituisce in una collezione tutti i valori



# Concetti introdotti

- Interface **Map<K, V>**
  - Operazioni principali
- Implementazioni di **Map<K, V>**
  - unicità delle chiavi

# Mappe: implementazioni

- Nelle librerie del Collection Framework abbiamo diverse implementazioni di **Map<K,V>**:
  - **HashMap<K,V>**
  - **TreeMap<K,V>**
- Nelle due implementazioni l'unicità delle chiavi viene garantita da meccanismi analoghi a quelli delle implementazioni di **Set**. In particolare:
  - l'unicità delle chiavi in **HashMap<K,V>** è garantita attraverso i metodi **hashCode()** e **equals()** delle chiavi
  - l'unicità delle chiavi in **TreeMap<K,V>** è garantita attraverso il metodo **compareTo()** delle chiavi (e le chiavi devono implementare **Comparable<K>**) oppure tramite il metodo **compare()** di un comparatore esterno **Comparator<K>** (passato al costruttore della mappa)

# Esercizio

- Scrivere una batteria di metodi di test per comprendere a fondo la semantica di tutti i metodi citati nelle slide precedenti con riferimento alle due implementazioni di **Map<K, V>**
  - **HashMap<K, V>**
  - **TreeMap<K, V>**

# Esempio d'uso

```
import java.util.*;

public class Rubrica {
    private Map<String,Integer> rubrica;

    public Rubrica() {
        this.rubrica = new HashMap<String,Integer>();
    }

    public void inserisci(String nome, Integer numero) {
        this.rubrica.put(nome, numero);
    }

    public void rimuovi(String nome) {
        this.rubrica.remove(nome);
    }

    public Set<String> nomiInRubrica() {
        return this.rubrica.keySet();
    }

    public Integer dammiIlNumeroDi(String nome) {
        return this.rubrica.get(nome);
    }
}
```

# Esempio d'uso

```
public Integer dammiIlNumeroDi(String nome) {
    return this.rubrica.get(nome);
}

public Integer aggiornaNumero(String nome, Integer numero) {
    return this.rubrica.put(nome, numero);
}

public void toString(){
    String str = new String ();
    str += "-----\n";
    str += "Rubrica\n";
    Set<String> nomi = nomiInRubrica();
    for(String s : nomi) {
        str += s;
        str += ": ";
        str += this.rubrica.get(s);
        str += "\n";
    }
    str += "-----\n";
}
```

...

# Esempio d'uso (cont)

```
... public static void main(String[] args) {  
    Rubrica r = new Rubrica();  
    String s1 = new String("Paolo"), s2 = new String("Fabio");  
    String s3 = new String("Anna"), s4 = new String("Carla");  
  
    r.inserisci(s1, 390112461); // inserisco Paolo->390112461 in rubrica  
    r.inserisci(s2, 390108361); // inserisco Fabio->390108361 in rubrica  
    r.inserisci(s3, 39062888); // inserisco Anna->39062888 in rubrica  
    System.out.println(r.toString()); // stampo la rubrica  
    // verifico se ho il numero di Carla  
    Integer numeroCercato = r.dammiIlNumeroDi(s4);  
    if (numeroCercato == null)  
        System.out.println("Il numero di "+s4+" non esiste");  
    else  
        System.out.println("Il numero di "+s4+" è "+numeroCercato);  
  
    r.rimuovi(s2); // tolgo i dati relativi a Fabio dalla rubrica  
  
    // cambio il numero di Paolo (e mi faccio stampare il numero vecchio)  
    Integer nuovoNumero = 39066777;  
    Integer vecchioNumero = r.aggiornaNumero(s1, nuovoNumero);  
    System.out.println("Aggiornato il numero di "+s1+  
        " da "+vecchioNumero+" a "+nuovoNumero);  
  
    System.out.println(r.toString()); // stampo la rubrica  
}
```

# Esercizio Mappe

- Date le classi **Studente** e **Aula**

```
public class Studente {
    private String matricola;
    private String meseDiNascita;

    public Studente(String matricola, String meseDiNascita) {
        this.matricola = matricola;
        this.meseDiNascita = meseDiNascita;
    }

    public String getMatricola() {return this.matricola;}

    public String getMeseDiNascita() {return this.meseDiNascita;}

    public int hashCode() {return this.matricola.hashCode();}

    public boolean equals(Object o) {
        Studente s = (Studente)o;
        return this.matricola.equals(s.getMatricola());
    }
}
```

# Esercizio Mappe (cont.)

```
public class Aula {  
    private Set<Studente> studenti;  
  
    public Aula() {  
        this.studenti = new HashSet<Studente>();  
    }  
  
    public boolean addStudente(Studente studente) {  
        studenti.add(studente);  
    }  
  
    public Map<String, List<Studente>> meseDiNasciata2studenti()  
    {  
        // scrivere il codice di questo metodo  
    }  
}
```

- Scrivere il codice del metodo `cds2studenti()` :
  - ritorna una mappa che associa ad un mese una lista con gli studenti nati in quel mese



# Esercizio Mappe (cont.)

- Esempio:
  - se un'aula contiene un insieme con i seguenti studenti:  
`{<"00","gen">, <"11","gen">,  
 <"54","ott">,<"24","mar">,<"32", "mar">}`
  - il metodo **`meseDiNascita2studenti()`** deve restituire una mappa con le seguenti coppie chiave→valore:  
  
    `"gen" → {<"00", "gen">,<"11", "gen">}`  
    `"ott" → {<"54", "ott">}`  
    `"mar" → {<"32", "mar">,<"24", "mar">}`

# Una Soluzione

```
public Map<String, List<Studente>>
    meseDiNascita2studenti() {
    List<Studente> tmp;
    Map<String, List<Studente>> mappa;
    mappa = new HashMap<String, List<Studente>>();

    for(Studente stud : this.studenti){
        tmp = mappa.get(stud.getMeseDiNascita());
        if(tmp==null)
            tmp = new ArrayList<Studente>();
        tmp.add(stud);
        mappa.put(stud.getCds(), tmp);
    }
    return mappa;
}
```

# Un'altra soluzione

```
public Map<String, List<Studente>>
    meseDiNascita2studenti() {
    List<Studente> tmp;
    Map<String, List<Studente>> mappa;
    mappa = new HashMap<String, List<Studente>>();
    for(Studente stud : this.studenti){
        tmp = mappa.get(stud.getgetMeseDiNascita());
        if(tmp==null) {
            tmp = new ArrayList<Studente>();
            mappa.put(stud.getgetMeseDiNascita(), tmp);
        }
        tmp.add(stud);
    }
    return mappa;
}
```

# Un'altra soluzione ancora

```
public Map<String, List<Studente>>
    meseDiNascita2studenti() {
    List<Studente> tmp;
    Map<String, List<Studente>> mappa;
    mappa = new HashMap<String, List<Studente>>();
    for(Studente stud : this.studenti){
        if(mappa.containsKey(stud.getMeseDiNascita()) {
            tmp = mappa.get(stud.getgetMeseDiNascita());
            tmp.add(stud)
        }
        else {
            tmp = new ArrayList<Studente>();
            tmp.add(stud);
            mappa.put(stud.getgetMeseDiNascita(), tmp);
        }
    }
    return mappa;
}
```

# Mappe, Liste, Insiemi

- Spesso è più comodo usare una mappa al posto di una lista o di un insieme
- Proviamo ad esempio a scrivere il codice della classe **Borsa**: per memorizzare la collezione degli attrezzi nella borsa, confrontiamo le due soluzioni:
  - uso una lista `List<Attrezzo>`
  - uso un mappa `Map<String, Attrezzo>`, dove la chiave è il nome dell'attrezzo `nomeAttrezzo → Attrezzo`  
In pratica la mappa funziona da *array associativo*:  
(il nome dell'attrezzo agisce come un indice)
- NB: poiché la chiave è unica, non è possibile memorizzare duplicati