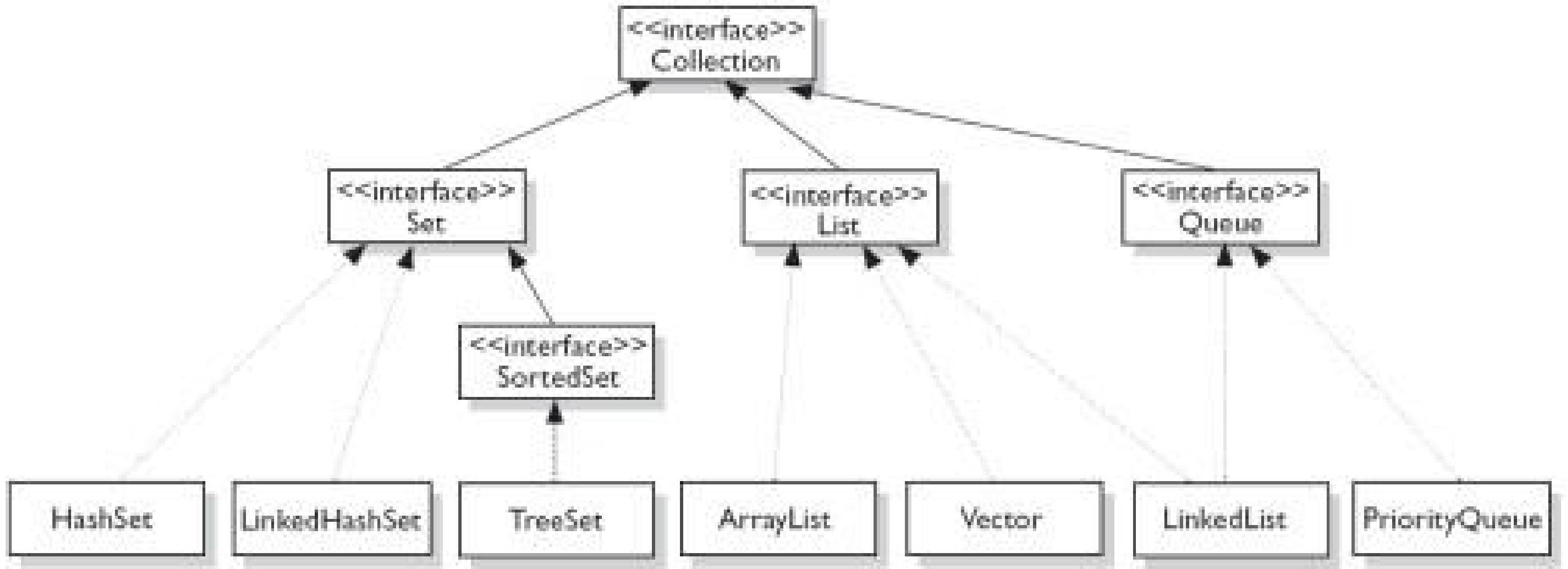
Università degli Studi dell’Aquila

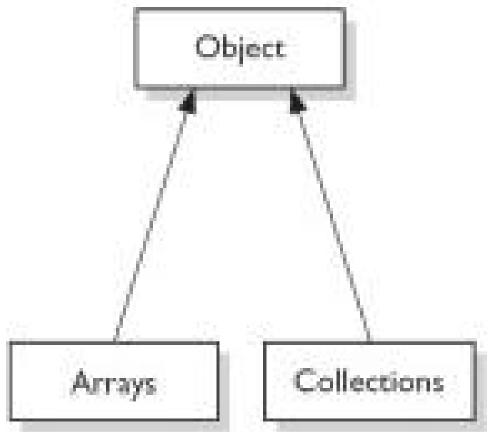
Dipartimento di Ingegneria e Scienze dell’Informazione e Matematica

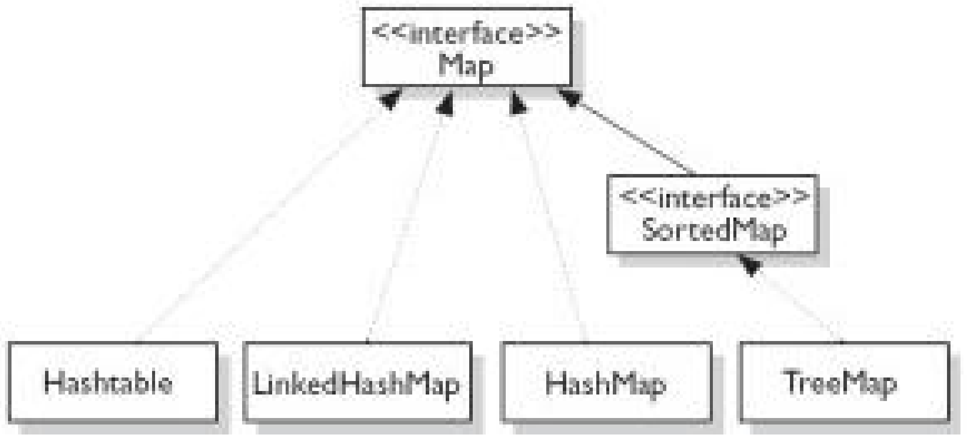
Università degli Studi dell’Aquila

Corso di Algoritmi e Strutture Dati con Laboratorio

The JCF(continua): L’interfaccia Map


<<interface>> Collection
<<interface>> Set
<<interface>> List
<<interface>> Queue
<<interface>> SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue


Object
Arrays
Collections


<<interface>> Map
<<interface>> SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap

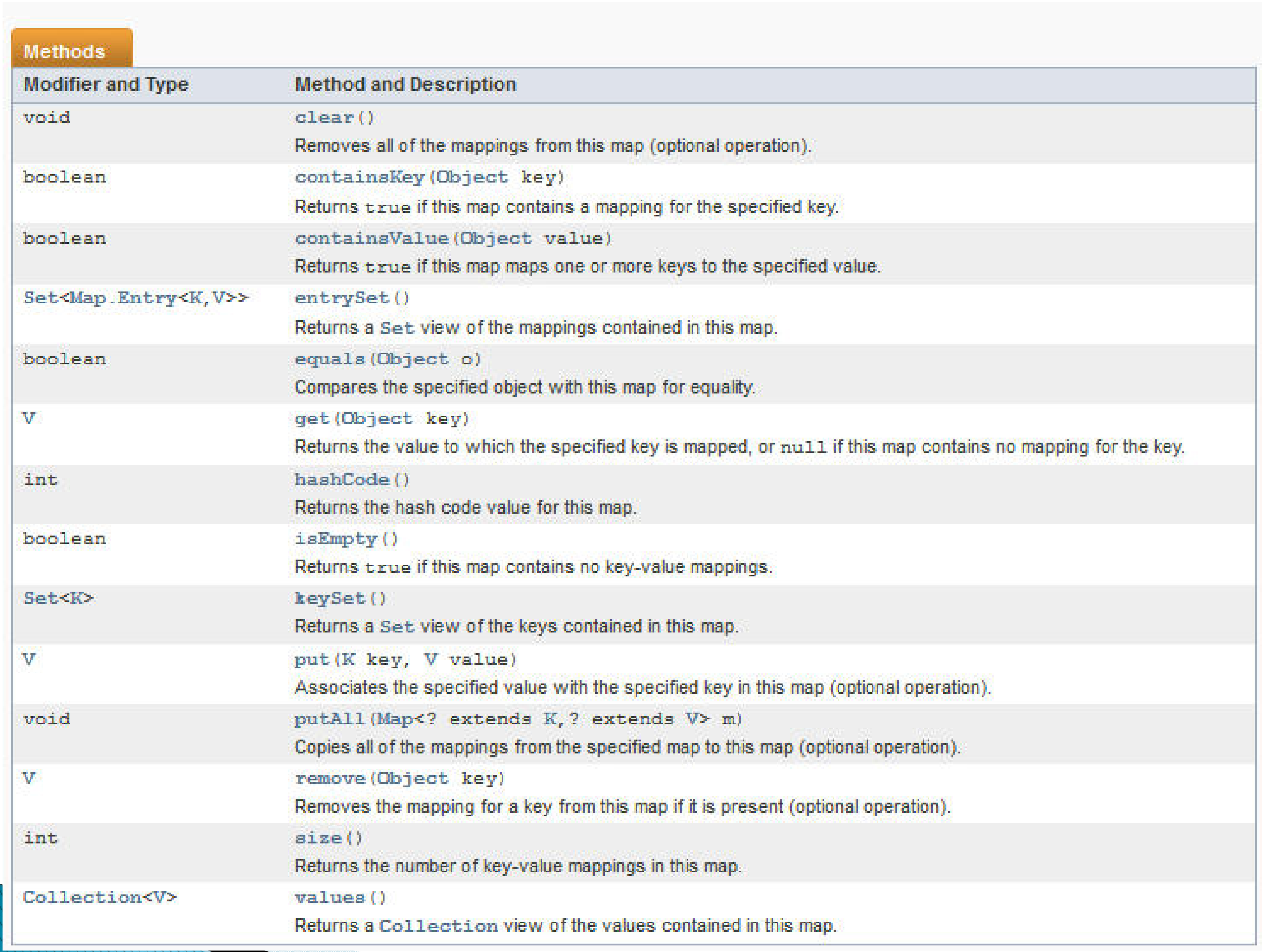
L’interfaccia `Map<K,V>`

- Una mappa è una raccolta (oggetto che contiene elementi: es: array, oggetto di tipo Collection) in cui ogni elemento ha due parti:
  1. Una chiave univoca (es. Codice fiscale)
  2. Un valore (es. Nomi)

1. Il JCF contiene un’interfaccia Map che definisce le intestazioni dei metodi per il dato astratto “mappa”
2. L’interfaccia Map non estende l’interfaccia Collection

L’interfaccia `Map<K,V>`

- Un dizionario è un esempio di mappa:
  - la chiave è la parola che viene definita
  - Il valore è costituito dalla sua definizione e dall’etimologia
- Talvolta i termini dizionario e mappa vengono usati come sinonimi
- Esempio: possiamo creare una mappa di studenti in cui ogni chiave è l’ID (matricola) ed il valore il voto medio dello studente



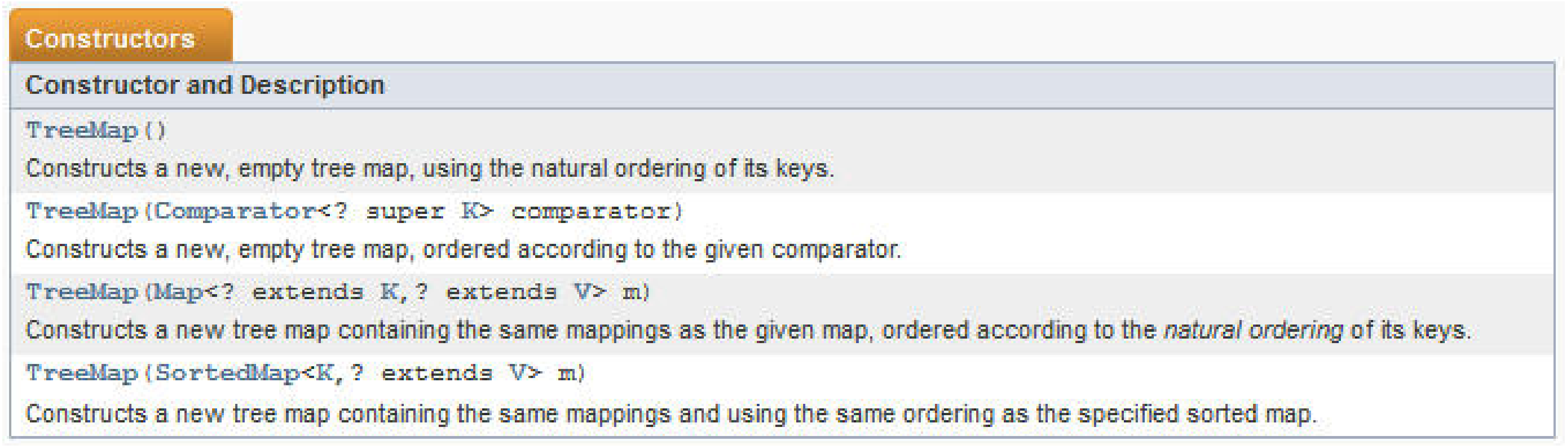
Methods

Modifier and Type	Method and Description
`void`	`clear()` Removes all of the mappings from this map (optional operation).
`boolean`	`containsKey(Object key)` Returns `true` if this map contains a mapping for the specified key.
`boolean`	`containsValue(Object value)` Returns `true` if this map maps one or more keys to the specified value.
`Set<Map.Entry<K,V>>`	`entrySet()` Returns a `Set` view of the mappings contained in this map.
`boolean`	`equals(Object o)` Compares the specified object with this map for equality.
`V`	`get(Object key)` Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
`int`	`hashCode()` Returns the hash code value for this map.
`boolean`	`isEmpty()` Returns `true` if this map contains no key-value mappings.
`Set<K>`	`keySet()` Returns a `Set` view of the keys contained in this map.
`V`	`put(K key, V value)` Associates the specified value with the specified key in this map (optional operation).
`void`	`putAll(Map<? extends K,? extends V> m)` Copies all of the mappings from the specified map to this map (optional operation).
`V`	`remove(Object key)` Removes the mapping for a key from this map if it is present (optional operation).
`int`	`size()` Returns the number of key-value mappings in this map.
`Collection<V>`	`values()` Returns a `Collection` view of the values contained in this map.

La classe `TreeMap<K,V>`

- La classe alloca una mappa in un albero red-black, ordinato in base alle chiavi
- Nell’intestazione, I parametri di tipo K e V stanno per: “Key” e “Value”.

```
public class TreeMap<K, V>
    implements SortedMap<K, V>
    extends AbstractMap<K, V>
```



Constructors

Constructor and Description
`TreeMap()` Constructs a new, empty tree map, using the natural ordering of its keys.
`TreeMap(Comparator<? super K> comparator)` Constructs a new, empty tree map, ordered according to the given comparator.
`TreeMap(Map<? extends K,? extends V> m)` Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys.
`TreeMap(SortedMap<K,? extends V> m)` Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map.

La classe `TreeMap<K,V>`

Specifiche dei principali metodi:

```
/**
 *  Initializes this TreeMap object to
 *  be an empty map.
 */
public TreeMap( )
```

- Esempio:

```
TreeMap<String, Double> students =
              new TreeMap<String, Double>( );
```

```
/**
 *  Ensures that there is an element in this TreeMap object
 *  with the specified key&value pair. If this TreeMap
 *  object had an element with the specified key before
 *  this method was called, the previous value associated
 *  with that key has been returned.  Otherwise, null
 *  has been returned.
 *  The worstTime (n) is O (log n).
 *  @param key - the specified key
 *  @param value - the specified value
 *  @return the previous value associated with key, if
 *                 there was such a mapping; otherwise, null.
 * @throws ClassCastException - if key cannot be compared
 *                 with the keys currently in the map.
 * @throws NullPointerException - if key is null and this Map
 *                 object uses the natural order, or the comparator
 *                 does not allow null keys.
 */
public V put (K key, V value)
```

Esempi:

```
students.put (“L00000000”, 3.7);
students.put (“L11111111”, 2.0);
students.put (“L22222222”, 3.5);
students.put (“L44444444”, 3.7);
students.put (“L33333333”, 4.0);
students.put (“L22222222”, 3.8);
```

Ora il voto medio per L22222222 is 3.8

```
/**
 *  Returns the value associated with a specified key in
 *  this TreeMap object, or null if this TreeMap object has
 *  no mapping with the specified key.
 *  The worstTime (n) is O (log n).
 *
 *  @param key - the specified key
 *
 *  @return the value associated with key, or null if this
 *               TreeMap object has no mapping with this key.
 *
 * @throws ClassCastException - if key cannot be compared
 *               with the keys currently in the map.
 * @throws NullPointerException - if key is null and this Map
 *               object uses the natural order, or the comparator
 *               does not allow null keys.
 */
public V get (Object key)
```

- Esempio:

```
System.out.println (
        students.get  (“L11111111”));
System.out.println (
        students.get  (“L55555555”));
```

Output:

```
2.0
null
```

```
/**
* Determines if this TreeMap object contains a mapping
* with a specified key.
* The worstTime (n) is O (log n).
*
* @param key - the specified key
*
* @return true - if this TreeMap object contains a mapping
*               with the specified key; otherwise, false.
* @throws ClassCastException - if key cannot be compared
*                with the keys currently in the map.
* @throws NullPointerException - if key is null and this Map
*          object uses the natural order, or the comparator
*                does not allow null keys.
*/
public boolean containsKey (Object key)
```

```
/**
* Determines if this TreeMap object contains a
* mapping with a specified value.
* The worstTime (n) is O (n).
*
* @param value - the specified value
* @return true - if this TreeMap object
* contains a mapping
* with the specified value; otherwise, false.
*/
public boolean containsValue (Object value)
```

```
/**
 *  Ensures that there is no mapping in this TreeMap object
 *  with the specified key. If this TreeMap object had such
 *  a mapping before this method was called, the value
 *   has been returned.  Otherwise, null has been returned.
 *  The worstTime (n) is O (log n).
 *
 *  @param key - the specified key
 *  @return the value associated with key, if
 *                 there was such a mapping; otherwise, null.
 * @throws ClassCastException - if key cannot be compared
 *                 with the keys currently in the map.
 * @throws NullPointerException - if key is null and this Map
 *     object uses the natural order, or the  comparator
 *                 does not allow null keys.
 */
public V remove (Object key)
```

Il metodo Map.entrySet()

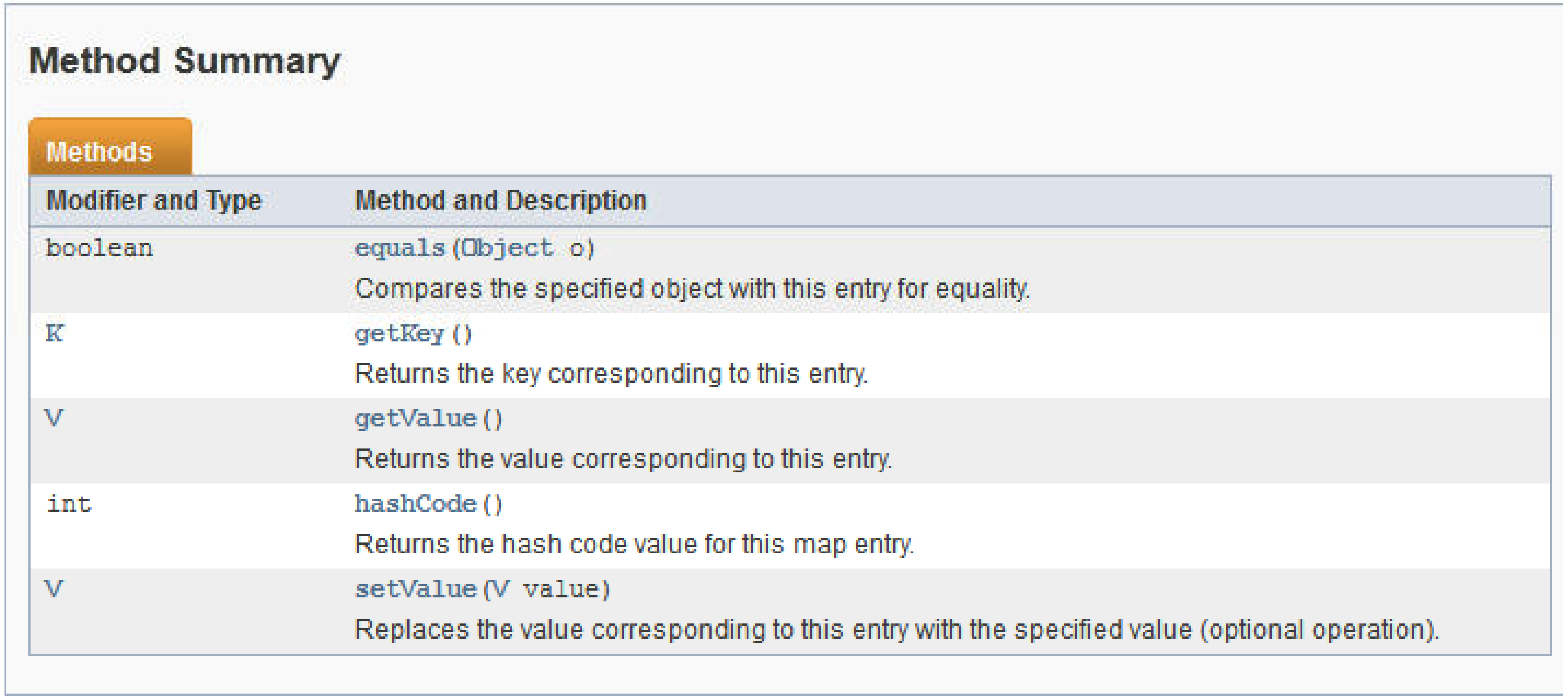
- Possiamo vedere un oggetto `TreeMap` come un set of entries
- Possiamo "estrarre" dal set di entries un set of keys, e una collection of values.

```
/**
 *  @return a Set view of the mappings in
 *  this TreeMap object.
 */
public Set entrySet( )
```

Il metodo Map.entrySet()

- Una mappa non è iterabile!!!
- Il metodo **`Map.entrySet()`** consente di iterare le coppie di una mappa nell'ambito del set di entries associato all'oggetto TreeMap; non esiste altro modo per consentire un'iterazione delle coppie.
- Se si modifica la mappa mentre è in corso un'iterazione sul corrispondente set, l'esito dell'iterazione è indefinito (tranne quando le modifiche avvengono mediante l'operazione **`remove`** dell'iteratore, oppure mediante l'operazione **`setValue`** su una map entry restituita dall'iteratore).

public static interface Map.Entry<K,V>



Method Summary

Methods

Modifier and Type	Method and Description
boolean	`equals(Object o)` Compares the specified object with this entry for equality.
K	`getKey()` Returns the key corresponding to this entry.
V	`getValue()` Returns the value corresponding to this entry.
int	`hashCode()` Returns the hash code value for this map entry.
V	`setValue(V value)` Replaces the value corresponding to this entry with the specified value (optional operation).

Il metodo Map.entrySet(): Esempio

- Esempio: per stampare gli studenti con voto medio maggiore di 3.5 :

```
for (Map.Entry<String, Double> entry :
  students.entrySet())
  if (entry.getValue() > 3.5)
          System.out.println (entry);
```

- Oppure:

```
Iterator<Map.Entry<String, Double>> itr =
  students.entrySet().iterator();
while (itr.hasNext()) {
     Map.Entry<String, Double> entry = itr.next();
  if (entry.getValue() > 3.5)
      System.out.println (entry);
} // while
```

- Output:

```
L00000000=3.7
L22222222=3.8
L33333333=4.0
L44444444=3.7
```

- Esempio: Stampare l’ID di ogni studente con voto maggiore di 3.5

```
for (Map.Entry<String, Double> entry :
       students.entrySet())
  if (entry.getValue() > 3.5)
    System.out.println(entry.getKey());
```

- Oppure:

```
Iterator<Map.Entry<String, Double>> itr =
  students.entrySet().iterator();
while (itr.hasNext()) {
   Map.Entry<String, Double> entry = itr.next();
  if (entry.getValue() > 3.5)
      System.out.println (entry.getKey());
} // while
```

- Esempio: Stampare ogni voto medio > 3.5

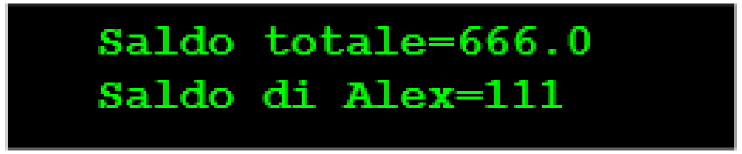
```
for (Double gpa : students.values())
  if (gpa > 3.5) System.out.println (gpa);
```

Oppure:

```
Iterator<Double> itr =
  students.values().iterator();
while (itr.hasNext())  {
   Double gpa = itr.next();
  if (gpa.doubleValue() > 3.5)
  System.out.println (gpa);
} // while
```

ESEMPIO

```
import java.util.*;
public class ContoCorrente {
        String numCC,nome;
        int saldo;
        public ContoCorrente (String nome, String numCC, int saldo){
                this.nome=nome;
                this.saldo=saldo;
                this.numCC=numCC;
        }
        public int getSaldo(){return saldo;}
        public static void main(String[] args ){
                Map<String, ContoCorrente> tm=
                                                        new TreeMap<String,ContoCorrente>();
                tm.put("Alex", new ContoCorrente("Alex", "100A", 111));
                tm.put("Max", new ContoCorrente("Max", "200B", 222));
                tm.put("Pippo", new ContoCorrente("Pippo", "100B", 333));
                Collection<ContoCorrente> collezioneCC= tm.values();
                double saldoTot = 0.0;
                for (ContoCorrente cc : collezioneCC ) {
                        saldoTot += cc.getSaldo();
                }
                System.out.println("Saldo totale="+saldoTot);
                System.out.println("Saldo di Alex="+tm.get("Alex").getSaldo());      } }
```



```
Saldo totale=666.0
Saldo di Alex=111
```

Esercizio 1

- Suppose we are given the name and division number for each employee in a company. There are no duplicate names. We would like to store this information alphabetically, by name.

How should this be done?

TreeMap? TreeSet? Comparable? Comparator?

Esercizio 1: Esempio

Misino,John	8
Nguyen,Viet	14
Panchenko,Eric	6
Dunn,Michael	6
Deusenbery,Amanda	14
Taoubina,Xenia	6

We want these elements stored in the following order:

Deusenbery,Amanda	14
Dunn,Michael	6
Misino,John	8
Nguyen,Viet	14
Panchenko,Eric	6
Taoubina,Xenia	6

Esercizio 1

- We can use a TreeMap object.
- The keys will be names, of type String, and the values will be division numbers of type Integer.
- The String class implements the Comparable interface, so the elements will be stored in alphabetical order of keys.

- Rif. CompanyMap.java

Esercizio 2

- Re-do Programming Exercise 1, but now the ordering should be by increasing division numbers, and within each division number, by alphabetical order of names.

Esercizio 2: Esempio

Misino,John	8
Nguyen,Viet	14
Panchenko,Eric	6
Dunn,Michael	6
Deusenbery,Amanda	14
Taoubina,Xenia	6

We want these elements stored in the following order:

Dunn,Michael	6
Panchenko,Eric	6
Taoubina,Xenia	6
Misino,John	8
Deusenbery,Amanda	14
Nguyen,Viet	14

Esercizio 2

- We can use a TreeSet object.
- The elements will be objects in an Employee class, which implements the Comparable interface.

```
class Employee implements Comparable<Employee>  {
  protected String name;
  protected int division;

  public Employee() { }

  public Employee (String name, Integer division)  {
       this.name = name;
       this.division = division;
  } // 2-parameter constructor

  public int compareTo (Employee otherEmployee)  {
       if (division != otherEmployee.division)  return division -
  otherEmployee.division;
       return name.compareTo (otherEmployee.name);
  } // method compareTo

  public String toString()  {
       return name + " " + division;
  } // method toString

} // class Employee
```

II prova intermedia A.A. 2014/15

Esercizio 1 Si scriva un metodo **`String piuFrequente(List<String>)`** che, data una lista di stringhe, restituisce quella con il maggior numero di occorrenze.
Suggerimento: si usi una mappa per calcolare le frequenze delle stringhe.
Si ricorda che la classe **`TreeMap<K,V>`** mette a disposizione, tra le altre, le operazioni:

- **`boolean containsKey(Object key)`**
- **`Boolean containsValue(Object value)`**
- **`Object get(Object key)`**
- **`Object put(Object key, Object value)`**
- **`int size()`**