Corso di Laurea in Ingegneria dell'Informazione
(sede di Latina)
A. A. 2013/2014

Progettazione del Software

Luca Iocchi

J11 – Java Collections Framework

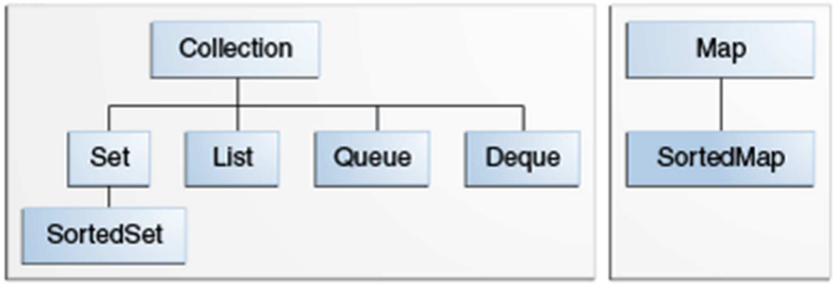
Java Collections Framework

JCF è una libreria formata da un insieme di interfacce e di classi che implementano tipi di dati astratti che rappresentano collezioni di oggetti.

JCF comprende:

- Interfacce per vari tipi di collezioni
- Classi che implementano tali interfacce
- Algoritmi di uso comune (ricerca, ordinamento, ecc.)

Interfacce del JCF



Classi e interfacce generiche

Le interfacce e le classi del JCF sono definite generiche, ovvero nel nome viene specificato un tipo generico che sarà istanziato all'atto della creazione degli oggetti.

Esempio:

```
public interface I<E> {
    E …
}
public class C<E> implements I<E> {
    E …
}

C<String> o = new C<String>();
```

Interfaccia Collection

```
public interface Collection <E> {
    // Basic Operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(<E> element);    // Optional
    boolean remove(Object element); // Optional
    Iterator <E> iterator();
    boolean equals(Object o);
    int hashCode();

    // Bulk Operations
    boolean containsAll(Collection <?> c);
    boolean addAll(Collection<? extends E> c); // Opt.
    boolean removeAll(Collection <?> c); // Optional
    boolean retainAll(Collection <?> c); // Optional
    void clear(); // Optional

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```


Interfaccia Set

```
public interface Set <E> extends Collection <E>{
    // Basic Operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(<E> element);    // Optional
    boolean remove(Object element); // Optional
    Iterator <E> iterator();
    boolean equals(Object o);
    int hashCode();

    // Bulk Operations
    boolean containsAll(Collection <?> c);
    boolean addAll(Collection<? extends E> c); // Opt.
    boolean removeAll(Collection <?> c); // Optional
    boolean retainAll(Collection <?> c); // Optional
    void clear(); // Optional

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```


Interfaccia List

```
public interface List <E> extends Collection <E>{
    /*...
    Metodi ereditati da Collection
    ...*/

    // Positional Access
    E get(int index);
    E set(int index, E element);     // Optional
    void add(int index, E element);  // Optional
    E remove(int index); // Optional
    boolean addAll(int ind,
                   Collection <? extends E> c);//Opt.

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator <E> listIterator();
    ListIterator <E> listIterator(int index);

    // Range-view
    List <E> subList(int from, int to);
}
```


Classi del JCF

Interfaccia	Classe
Set	HashSet TreeSet
List	ArrayList LinkedList

Classi HashSet e TreeSet

Per effettuare una ricerca efficiente

- HashSet con la definizione del metodo hashCode()
- TreeSet con l'implementazione dell'interfaccia Comparable

La classe String fornisce sia la definizione di hashCode che l'implementazione di Comparable, quindi si possono usare HashSet<String> e TreeSet<String> senza ulteriori definizioni.

Per le classi definite dall'utente bisogna invece realizzare hashCode() e/o implementare l'interfaccia Comparable.

Esempio di uso del JCF

Dichiarazione di variabili di tipo interfaccia.

Assegnazione di variabili di tipo interfaccia a oggetti delle classi che implementano tale interfaccia.

Esempio

```
Set<String> s = new HashSet<String>();
```

Vantaggi: estendibilità e riusabilità, in quanto l'implementazione di Set può essere cambiata senza modificare altre parti del codice oltre la chiamata al costruttore.

Esempio di uso del JCF

```
import java.util.*;

public class TestJCF {
    public static void main(String[] args) {
        Set<String> s = new HashSet<String>();
        s.add("Mario"); s.add("Anna"); s.add("Luigi"); s.add("Anna");
        System.out.println(s);
    }
}
```

Output: [Mario, Luigi, Anna]

L'ordine di stampa non corrisponde a quello di inserimento
Anna compare solo una volta.

OK! è un insieme, quindi l'ordine non è rilevante e i valori multipli non sono considerati

Esempio di uso del JCF

```
import java.util.*;

public class TestJCF {
    public static void main(String[] args) {
        List<String> l = new LinkedList<String>();
        l.add("Mario"); l.add("Anna"); l.add("Luigi"); l.add("Anna");
        System.out.println(l);
    }
}
```

Output: [Mario, Anna, Luigi, Anna]

L'ordine di stampa corrisponde a quello di inserimento
Anna compare due volte

OK! è una lista, quindi l'ordine è rilevante e sono ammessi duplicati

Iterator

Un iteratore è un oggetto che rappresenta un cursore per accedere sequenzialmente agli elementi di una collezione.

La funzione di Collection
    Iterator<E> iterator()
restituisce un iteratore con cui si può scandire la collezione

Anche Iterator è una interfaccia e non una classe.

```
public interface Iterator <E> {
    boolean hasNext();
    E next();
    void remove();    // Optional
}
```


ListIterator

Un iteratore di lista estende le funzionalità dell'iteratore consentendo un accesso bidirezionale (si usa sulle Liste)

```
public interface ListIterator <E>
            extends Iterator <E> {
   /* Metodi ereditati da Iterator:
   boolean hasNext();
   E next();
   void remove();
   */

    boolean hasPrevious();
    E previous();

    int nextIndex();
    int previousIndex();

    void set(E element);     // Optional
    void add(E element);     // Optional
}
```


Esempio di uso degli iteratori

```
import java.util.*;

public class TestJCF {
    public static void main(String[] args) {
        Set<String> s = new HashSet<String>();
        …
        Iterator<String> is = s.iterator();
        while (is.hasNext()) {
            String e = is.next();
            …
        }
    }
}
```

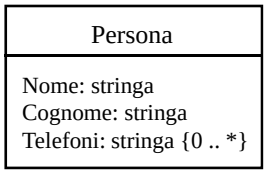

Esempio di uso degli iteratori

```
import java.util.*;

public class TestJCF {
    public static void main(String[] args) {
        List<String> l = new LinkedList<String>();
        …
        Iterator<String> il = l.iterator();
        while (il.hasNext()) {
            String e = il.next();
            …
        }
    }
}
```

JCF per realizzare attributi multivalore

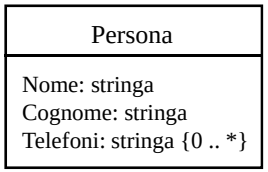
"*Una persona può avere 0, 1 o più numeri di telefono*"



La proprietà telefoni di Persona può essere rappresentata quindi come un insieme di stringhe, ovvero Set<String>

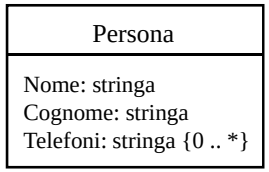
In corrispondenza di un attributo multivalore si definiscono i metodi get, add e remove (add e remove solo se la proprietà è mutabile)

JCF per realizzare attributi multivalore




```
public class Persona {
    // campi dati
    private final String nome, cognome;
    private Set<String> telefoni;
    // campi operazione
    public Persona(String nome, String cognome) { ... }
    public String getNome() {  ... }
    public String getCognome() {  ...  }
    public Set<String> getTelefoni() {  ...  }
    public void addTelefono(String tel) {  ...  }
    public void removeTelefono(String tel) {  ...  }
    ...
}
```


Realizzazione



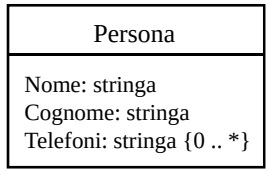

```
public Persona(String nome, String cognome) {
    ...
    telefoni = new HashSet<String>();
}

public Set<String> getTelefoni() {
    return telefoni;  // ERRORE: consente side-effect!!!
}

public void addTelefono(String tel) {
    telefoni.add(tel);
}

public void removeTelefono(String tel) {
    telefoni.remove(tel);
}
```


Realizzazione




```
public Persona(String nome, String cognome) {
    ...
    telefoni = new HashSet<String>();
}

public Set<String> getTelefoni() {
    return new HashSet<String>(telefoni);  // CORRETTO: copia profonda
}

public void addTelefono(String tel) {
    telefoni.add(tel);
}

public void removeTelefono(String tel) {
    telefoni.remove(tel);
}
```

Insiemi per classi definite dall'utente

Per usare una classe del JCF che implementa l'interfaccia Set su oggetti di una classe definita dall'utente è necessario:

- implementare l'interfaccia Comparable per usare TreeSet
- definire il metodo hashCode per usare HashSet

Implementazione di Comparable

```
public class Persona implements Comparable<Persona> {

        // -1 se this precede p; 0 se sono uguali; +1 se this segue p
        public int compareTo(Persona p) {
                if (this.cognome.equals(p.cognome))
                        return this.nome.compareTo(p.nome);
                else
                        return this.cognome.compareTo(p.cognome);
        }

}
```


Definizione di hashCode

```
public class Persona implements Comparable {

        // ritorna un valore di hash per l'oggetto
        public int hashCode() {
                return 7;  // implementazione non efficiente
        }

        // ritorna un valore di hash per l'oggetto
        public int hashCode() {
                return eta;  // implementazione un po' più efficiente
        }

        …
}
```


Esempi

- Creare una lista di persone

```
List<Persona> rubrica = new LinkedList<Persona>();
Persona p = new Persona(…);
rubrica.add(p);
```

- Trovare una persona in una lista di Persone

```
Persona p = new Persona(…);
rubrica.contains(p) // true se p è presente nella lista rubrica
```

Nota: Bisogna implementare equals nella classe Persona per la verifica dell'uguaglianza profonda.

Quali campi bisogna considerare nell'uguaglianza profonda?

Esempio

Ricerca di una persona in una lista.

```
List<Persona> rubrica = ….
Persona p = …;
bool found = false;
Iterator<Persona> i = rubrica.iterator();
while (i.hasNext() && ! false) {
    Persona q = i.next();
    if (q.equals(p)) {
        System.out.println("Persona "+p+" trovata"); found=true;
    }
}
if (!found)
    System.out.println("Persona "+p+" non trovata");
```


Esempio

Data una persona, stampare i suoi numeri di telefono.

```
List<Persona> rubrica = ….
Persona p = …;

Iterator<Persona> i = rubrica.iterator();
while (i.hasNext()) {
    Persona q = i.next();
    if (q.equals(p))
        System.out.println(q.getTelefoni());
}
```


Esercizio J11.1

Scrivere un metodo statico che data una lista di persone, effettui le seguenti operazioni:

- individuazione dei duplicati, ovvero elementi della lista corrispondenti alla stessa persona identificata da nome e cognome;
- rimozione dei duplicati, lasciando un solo elemento della lista per ogni persona ed effettuando l'unione di tutti i numeri di telefono presenti nei vari elementi duplicati