Programmazione Orientata agli Oggetti in Linguaggio Java

Tecniche di Programmazione: Collezioni Parte b

versione 2.3

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Mappe
  - ⇨HashMap
- Insiemi
- Ordinamenti
- Java Collections Framework

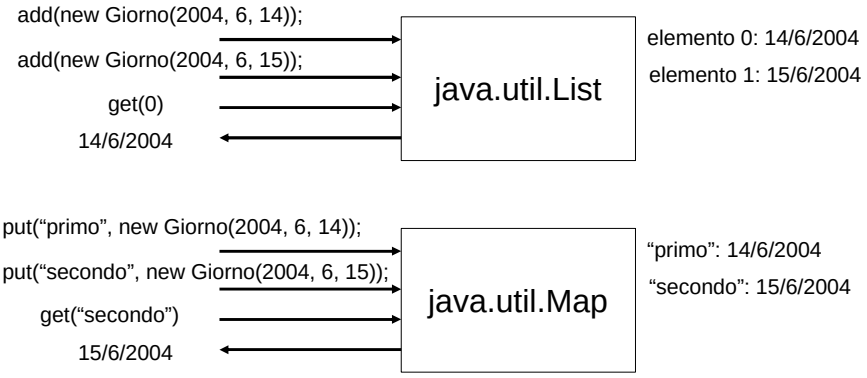
Mappe

- Utilizzo delle collezioni nell'applicazione
  - ⇨in ogni Giorno, una lista di impegni, come già discusso
  - ⇨nell'Agenda, una collezione di giorni
- Il tipo di collezione
  - ⇨una mappa, ovvero un dizionario associativo
  - ⇨classe java.util.HashMap
  - ⇨implementa l'interfaccia java.util.Map

Mappe

- Dizionario associativo
  - ⇨collezione in cui i riferimenti sono salvati con un "nome", detto chiave
  - ⇨tipicamente una stringa
  - ⇨possono successivamente essere recuperati rapidamente utilizzando la chiave
- Analogo
  - ⇨le liste sono associative rispetto agli interi

Mappe



Mappe

- I metodi principali di java.util.Map
  - ⇨void put(Object chiave, Object riferimento)
  - ⇨Object get(Object chiave)
  - ⇨Object remove(Object chiave)
  - ⇨int size()
- Le principali implementazioni
  - ⇨java.util.HashMap
  - ⇨java.util.TreeMap
  - ⇨java.util.Hashtable (versione legacy)

```
package it.unibas.appuntamenti.modello;

public class Agenda {

    private String utente;
    private java.util.HashMap giorni = new java.util.HashMap();

    public java.lang.String getUtente() {   return utente;  }

    public void setUtente(java.lang.String utente) { this.utente = utente; }

    public void addGiorno(Giorno giorno) {
        this.giorni.put(giorno.toShortString(), giorno);
    }

    public void removeGiorno(String data) {
        this.giorni.remove(data);
    }

    public Giorno getGiorno(String data) {
        return (Giorno)(this.giorni.get(data));
    }

    public int getNumeroGiorni() {
        return this.giorni.size();
    }
    ...
```


Mappe

- Differenze rispetto alle liste
  - ⇨in una mappa non sono significative le posizioni degli elementi ma le chiavi
  - ⇨le ricerche sulla base della chiave sono enormemente facilitate (nella lista richiederebbero una scansione)
  - ⇨utilizzata tipicamente quando più che le scansioni sono importanti le ricerche

Mappe

- Attenzione però
  - ⇨ad ogni chiave può essere associato un unico oggetto
  - ⇨put successive con la stessa chiave sostituiscono i valori precedenti
  - ⇨non può essere usata quando possono esserci più valori per la stessa chiave
  - ⇨es: impegni per orario nella giornata

HashMap

- Un requisito fondamentale per le mappe
  - ⇨la rapidità di inserimento e cancellazione
- L'implementazione fondamentale
  - ⇨HashMap
  - ⇨di gran lunga la più veloce
- La tecnica sottostante
  - ⇨tecnica di hashing
  - ⇨ovvero basata su "funzioni di hashing"

HashMap

- Funzione di hash
  - ⇨funzione che trasforma un valore (“chiave”) di lunghezza variabile in uno di lunghezza fissa (“hash” della chiave)
- Esempio
  - ⇨nome dello studente → ultimi 3 bit
  - ⇨oggetto Java → indirizzo nello heap
- Caratteristica tipica di una funzione hash
  - ⇨l’hash deve essere calcolabile rapidamente

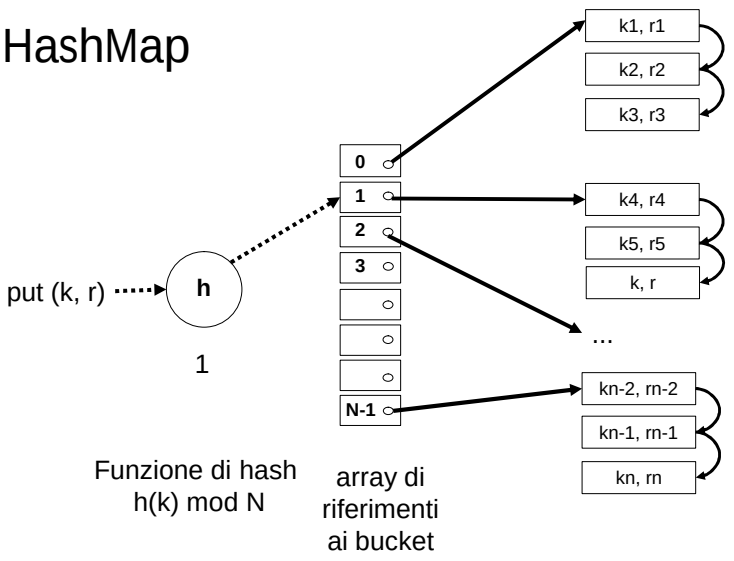
HashMap

- Classificazione delle funzioni hash
  - ⇨funzioni con collisioni o prive di collisioni
- Funzione priva di collisione
  - ⇨non ci sono due valori per cui l’hash è uguale
  - ⇨possibile solo se i valori sono finiti
- Funzione con collisione
  - ⇨più valori possono avere lo stesso valore di hash
  - ⇨caso tipico

HashMap

- Implementazione di put() nella HashMap
  - ⇨ la mappa mantiene gli oggetti in un array di N riferimenti a liste (dette “bucket”)
  - ⇨ ogni elemento della lista memorizza una coppia <chiave, riferimento>
  - ⇨ viene calcolato il valore della funzione di hash sulla chiave e poi viene applicato un operatore modulo per ridurlo ad un numero tra 0 e N – 1
  - ⇨ in questo modo si ottiene un indice nell’array; la coppia <chiave, riferimento> viene aggiunta in coda al bucket della posizione ottenuta

HashMap



HashMap

- Implementazione di get() in HashMap
  - ⇨viene calcolato il valore di hash della chiave per risalire al bucket (indice nell'array)
  - ⇨viene scandito il bucket e la chiave viene confrontata con ogni chiave
  - ⇨se viene trovata una chiave identica a quella cercata, viene restituito il riferimento
  - ⇨altrimenti viene restituito null

HashMap

- Due operazioni fondamentali
  - ⇨il calcolo della funzione di hash
  - ⇨il confronto tra le chiavi
- Calcolo della funzione di hash
  - ⇨viene usato il metodo hashCode() ereditato da Object
- Confronto tra le chiavi
  - ⇨viene utilizzato il metodo equals() ereditato da Object

HashMap

- Nota
  - ⇨le implementazioni standard sono basate sull'indirizzo in memoria
  - ⇨potrebbero non essere quelle adatte a fare hashing in alcuni casi
- Nelle classi principali della piattaforma
  - ⇨sono ridefinite opportunamente quando è necessario

HashMap

- Di conseguenza
  - ⇨è opportuno utilizzare come chiave per le mappe oggetti di classi note
  - ⇨es: String, Integer, ecc.
- Nel caso in cui questo non sia possibile
  - ⇨per la classe di oggetti da utilizzare come chiavi è necessario ridefinire opportunamente hashCode() ed equals()

Insiemi

- Altre classi significative delle collezioni
  - ⇨interface java.util.Set: rappresenta una collezione non ordinata e priva di duplicati
  - ⇨due principali implementazioni: HashSet e TreeSet
  - ⇨interface java.util.Collection: rappresenta una collezione generica (può essere un insieme oppure una lista)
  - ⇨tutte le collezioni sono scandibili con iteratori

Insiemi

- A cosa servono gli insiemi ?
  - ⇨possono essere quasi sempre sostituiti dalle liste
  - ⇨in alcuni casi sono più naturali
- Esempio: Gestione Appuntamenti
  - ⇨potrei utilizzare un oggetto di tipo java.util.Set per rappresentare l'insieme dei partecipanti ad una Riunione

Insiemi

- A questo punto
  - ⇨possiamo vedere come scandire una mappa
  - ⇨due metodi principali
- Primo metodo
  - ⇨ottengo l'insieme delle chiavi con Set keySet()
  - ⇨scandisco l'insieme con un iteratore e prelevo dalla mappa tutti gli elementi

Insiemi

- Secondo metodo
  - ⇨ottengo la collezione dei valori con il metodo Collection values()
  - ⇨ottengo un iteratore dalla collezione e scandisco tutti gli elementi
- Perchè Set e Collection ?
  - ⇨per rispettare la regola secondo cui si programma con le interfacce e non con le implementazioni

Insiemi

- Un esempio
  - ⇨il metodo toString() di Agenda
  - ⇨obiettivo: stampare tutti i dati dell'agenda
  - ⇨utente
  - ⇨mappa dei giorni con relative liste di impegni
- I versione
  - ⇨scandisce l'insieme dei valori nella mappa e li scandisce

```
// il metodo toString() di Agenda
// I versione

public String toString() {
        StringBuffer stringa = new StringBuffer();
        stringa.append("Utente: " + this.utente + "\n");
        java.util.Collection giorni = this.giorni.values();
        java.util.Iterator iterator = giorni.iterator();
        while (iterator.hasNext()) {
            Object giornoIesimo = iterator.next();
            stringa.append(giornoIesimo);
        }
        return stringa.toString();
}
```

Insiemi

- Il difetto di questa soluzione
  - ⇨i dati vengono prodotti in forma non ordinata per data
- Infatti
  - ⇨una HashMap è una collezione non ordinata
  - ⇨mantiene un insieme di chiavi e non una lista
- Obiettivo
  - ⇨produrre i dati in forma ordinata

Ordinamenti

- In Java
  - ⇨l'ordinamento delle collezioni è una funzionalità offerta dalla piattaforma
  - ⇨es: java.util.Collections.sort(java.util.List list)
  - ⇨ma bisogna rispettare delle regole
- In particolare
  - ⇨per ordinare una collezione di riferimenti, gli oggetti relativi devono essere confrontabili
  - ⇨rispetto ad una precisa relazione di ordine

Ordinamenti

- Esempio
  - la lista degli Impegni
  - un ordinamento possibile: per orario
  - un altro ordinamento: in ordine alfabetico rispetto alla descrizione
  - un altro ordinamento ancora: rispetto al luogo di svolgimento
  - per poter ordinare una lista di impegni bisogna decidere il criterio di ordinamento

Ordinamenti

- In Java
  - gli oggetti ordinabili devono implementare l'interfaccia java.lang.Comparable
- java.lang.Comparable
  - prevede un unico metodo
  - int compareTo(Object o)
  - risultato > 0 se this segue o nell'ordinamento
  - risultato < 0 se this precede o nell'ordinam.
  - risultato = 0 se this non precede nè segue o

Ordinamenti

- La strategia utilizzata
  - ⇨per ordinare gli impegni utilizziamo l'ordinamento degli orari
  - ⇨per ordinare gli orari sfruttiamo l'ordinamento basato sull'oggetto Date sottostante
  - ⇨Date implementa Comparable, per cui gli oggetti relativi sono già ordinati secondo l'ordinamento del calendario

Ordinamenti

```
package it.unibas.appuntamenti.modello;

public class Orario implements Comparable {

    private java.util.GregorianCalendar calendar;

    public int compareTo(Object orario) {
        java.util.Date questaData = this.calendar.getTime();
        java.util.Date altraData = ((Orario)orario).calendar.getTime();
        return questaData.compareTo(altraData);
    }
    ...
```

Ordinamenti

```
package it.unibas.appuntamenti.modello;
public interface Impegno extends Comparable { ... }

package it.unibas.appuntamenti.modello;
public abstract class ImpegnoAstratto implements Impegno {

    protected Orario orario;

    public int compareTo(Object o){
        return this.orario.compareTo(((Impegno)o).getOrario());
    }
```


```
package it.unibas.appuntamenti.modello;

public class Giorno implements Comparable {

    private java.util.List listaImpegni = new java.util.LinkedList();

    public String stringaImpegni() {
        ordinaImpegni();
        StringBuffer stringaImpegni = new StringBuffer();
        for (int i = 0; i < this.listaImpegni.size(); i++) {
            stringaImpegni.append(i + ". - " + this.listaImpegni.get(i).toString());
            stringaImpegni.append("\n");
            i++;
        }
        return stringaImpegni.toString();
    }

    public void ordinaImpegni() {
        java.util.Collections.sort(this.listaImpegni);
    }
    ...
```

Ordinamenti

- Analogamente
  - ⇨per ordinare la mappa dei giorni dell'agenda è possibile utilizzare l'ordinamento dei giorni
  - ⇨per ordinare i giorni è possibile utilizzare l'ordinamento della data sottostante
- Attenzione
  - ⇨stavolta non siamo di fronte ad una lista ma ad una mappa
  - ⇨l'operazione è leggermente più complessa

Ordinamenti

```
package it.unibas.appuntamenti.modello;

public class Giorno implements Comparable {

    private java.util.GregorianCalendar calendar;

    public int compareTo(Object giorno) {
        java.util.Date questaData = this.calendar.getTime();
        java.util.Date altraData = ((Giorno)giorno).calendar.getTime();
        return questaData.compareTo(altraData);
    }
```

```
package it.unibas.appuntamenti.modello;

public class Agenda {

    private String utente;
    private java.util.HashMap giorni = new java.util.HashMap();

     public String toString() {
        StringBuffer stringa = new StringBuffer();
        stringa.append("Utente: " + this.utente + "\n");
        java.util.List listaGiorni = ordinaGiorni()
        for (int i = 0; i < listaGiorni.size(); i++) {
            Object giornoIesimo = listaGiorni.get(i);
            stringa.append(giornoIesimo);
        }
        return stringa.toString();
    }

    private java.util.List ordinaGiorni() {
        java.util.Collection giorni = this.giorni.values();
        java.util.List lista = new java.util.ArrayList(coll);
        java.util.Collections.sort(lista);
        return lista;
    }
```

il metodo ordinaGiorni() mi restituisce una lista ordinata di giorni a partire dalla mappa

non è possibile ordinare qualsiasi Collection (potrebbe essere un Set) è necessario trasformarlo in lista
NOTA: non basta il cast

Ordinamenti

- Nota
  - ⇨l'utilizzo di Comparable prevede che sia possibile cambiare il codice delle classi che definiscono gli oggetti da ordinare
  - ⇨per implementare Comparable e definire compareTo()
  - ⇨se questo non è possibile, è possibile utilizzare un approccio alternativo
  - ⇨utilizzare un oggetto esterno comparatore

Ordinamenti

- L'interfaccia java.util.Comparator
  - ⇨unico metodo: int compare(Object o1, Object o2)
  - ⇨gli oggetti che implementano l'interfaccia sono quindi capaci di confrontare oggetti secondo algoritmi specifici
- Esempio
  - ⇨potrei implementare un comparatore per i Giorni
  - ⇨e poi un comparatore per gli Orari

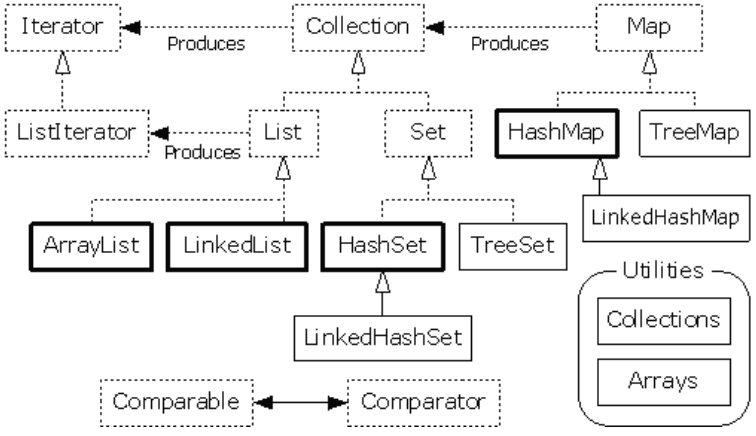
Ordinamenti

- Il metodo Collections.sort()
  - ⇨è sovraccarico
  - ⇨public static void sort(List list)
  - ⇨public static void sort(List list, Comparator c)
  - ⇨la seconda versione utilizza c per effettuare il confronto tra gli elementi di list

Java Collections Framework

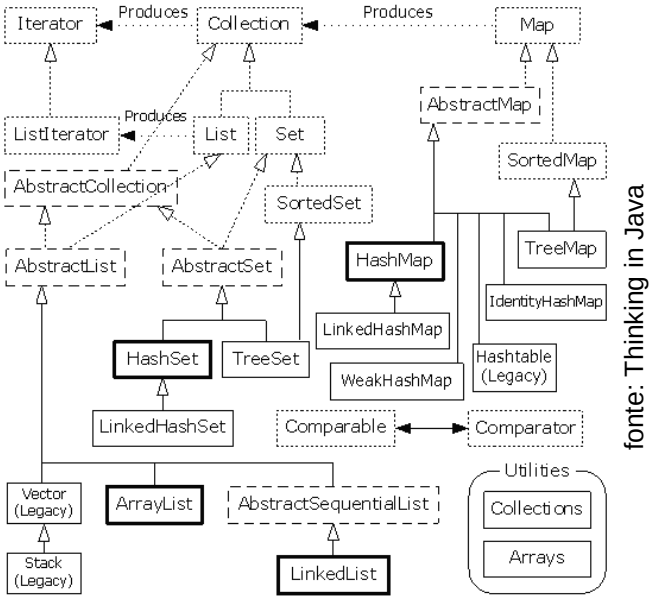
- Nel complesso
  - ⇨le classi corrispondenti alle collezioni compongono il cosiddetto Java Collections Framework
  - ⇨occupa quasi interamente il package java.util
- Nota
  - ⇨mancano alcune altre collezioni notevoli (es: code), ma ne esistono implementazioni in progetti separati (es: Jakarta Commons)

Java Collections Framework



fonte: Thinking in Java

Java Collections Framework
Versione Completa



fonte: Thinking in Java

Riassumendo

- Mappe
  - ⇨HashMap
- Insiemi
- Ordinamenti
- Java Collections Framework

Termini della Licenza