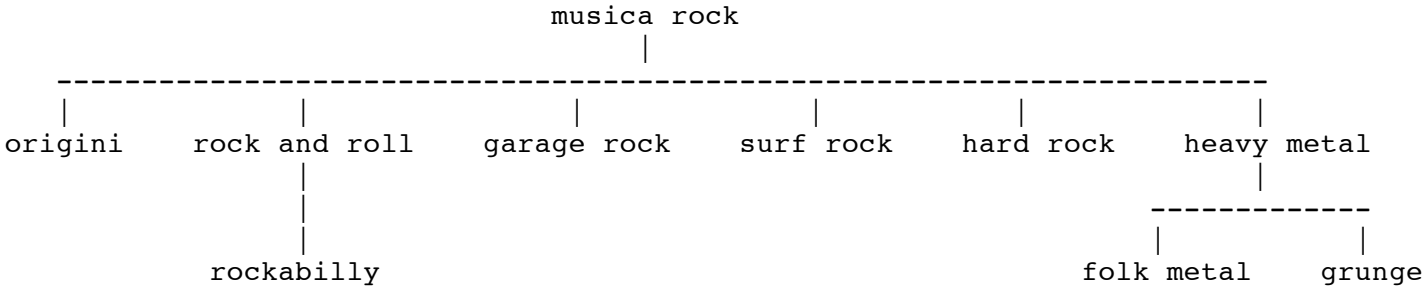
Metodologie di Programmazione a.a. 2008-2009 (canale E-O)

Progetto

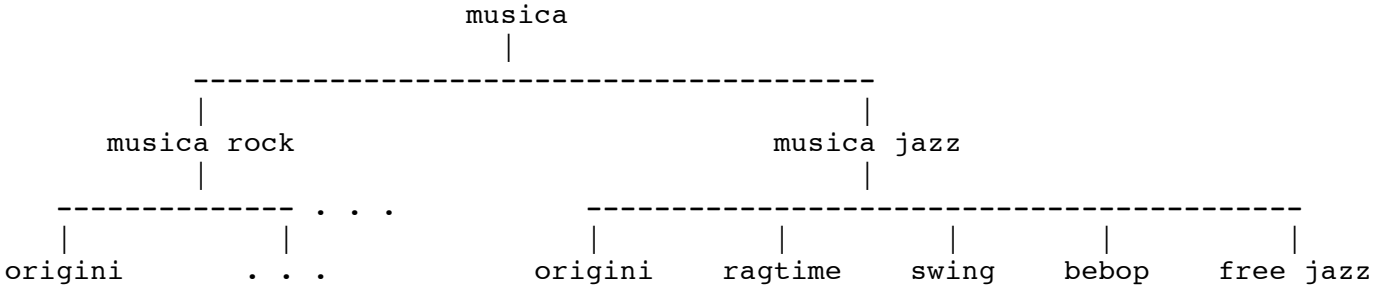
18 maggio 2009

Descrizione generale

L'obbiettivo del progetto è la realizzazione di una applicazione per l'organizzazione e il recupero di documenti tramite keywords gerarchiche. Una *keyword* è una parola, una frase o un termine che risulta utile per la classificazione e il successivo recupero di documenti di genere vario. Per *keywords gerarchiche* o *gerarchia di keywords* si intende una collezione di keywords organizzate in una gerarchia. Le keywords che si trovano ai livelli più alti della gerarchia rappresentano concetti più generali mentre quelle a livelli più bassi rappresentano concetti più specifici. Ad esempio, ecco una semplice gerarchia di keywords (non esaustiva) che riguarda la musica rock:



Si noti che, essendo le keywords organizzate in una gerarchia, il significato di una keyword non dipende solamente dal suo nome ma anche dalla posizione che occupa nella gerarchia. Ad esempio, la nostra gerarchia potrebbe comprendere anche la musica jazz:



La keyword `orgini` appare sia nella sottogerarchia della keyword `musica rock` che in quella della keyword `musica jazz`, ovviamente, con significati diversi. Nel primo caso significa le origini della musica rock e nel secondo caso significa le origini della musica jazz. Grazie alla gerarchia alcuni nomi di keywords possono essere abbreviati. Ad esempio, siccome sia `musica rock` che `musica jazz` sono nella gerarchia di `musica`, potrebbero essere denominate semplicemente `rock` e `jazz`. Il vantaggio principale apportato dall'organizzazione gerarchica delle keywords sta nel fatto che se a un documento *D* è associata, ad esempio, la keyword `grunge` e si effettua una ricerca relativamente a `heavy metal` allora sarà recuperato automaticamente anche il documento *D* (anche se il documento non ha specificatamente associata la keyword `heavy metal`). In generale, questo permette una classificazione più accurata e nel contempo una maggiore facilità e flessibilità nell'esprimere condizioni di ricerca tramite le keywords.

Introduciamo ora un po' di terminologia. Una keyword KB che si trova nella sottogerarchia di un'altra keyword KA è una *sub-keyword* di KA e KA è una *super-keyword* di KB. Ad esempio, `grunge` è una sub-keyword di `heavy metal` e anche di `rock` e `musica`, inoltre, `rock`, `musica` e `heavy metal` sono super-keywords di `grunge`. Se KA è una super-keyword di KB e non ci sono altre keywords tra KA e KB allora diremo che KA è una *super-keyword diretta* di KB e che KB è una *sub-keyword diretta* di KA. Ad esempio `heavy metal` è una super-keyword diretta di `grunge` e `grunge` è una sub-keyword diretta di `heavy metal`, ma `rock` è una super-keyword di `grunge` che non è una super-keyword diretta di `grunge`.

Assumeremo che la gerarchia di keyword contenga sempre una keyword speciale che è una super-keyword di tutte le keyword della gerarchia. Tale keyword è unica ed è denominata ANYTHING. Per semplicità assumeremo che la gerarchia sia pura, cioè, per ogni keyword K (diversa da ANYTHING) c'è esattamente una e una sola keyword che è una super-keyword diretta di K. In generale, per specificare una keyword non è sufficiente specificare solamente il nome è necessario specificare anche la sua posizione nella gerarchia. Ogni keyword può essere specificata senza ambiguità mediante la sequenza di tutti i nomi delle sue super-keywords, tali sequenze saranno chiamate *path-keywords*. Ad esempio, il path-keyword di `grunge` è `musica>rock>heavy metal>grunge`, i path-keywords delle due keywords di nome `orgini` sono `musica>rock>origini` e `musica>jazz>origini` (nei path-keywords la keyword speciale ANYTHING è sempre sottointesa). Si conviene di usare il carattere `'>'` per separare i nomi delle keywords.

L'applicazione dovrà quindi gestire una gerarchia di keywords, l'associazione di tali keywords a documenti e la ricerca dei documenti in base alle keywords. Per facilitare la gestione delle associazioni delle keywords ai documenti in combinazione con la gestione delle modifiche alla gerarchia delle keywords, l'applicazione dovrà assegnare ad ogni keyword un codice univoco, nel momento in cui la keyword viene aggiunta alla gerarchia. Così ad esempio le due keywords con lo stesso nome `origini` avranno codici differenti. Tali codici saranno del tutto trasparenti all'utente e serviranno solamente per la gestione interna delle keywords e delle loro associazioni ai documenti. L'associazione delle keywords è gestita tramite una *Classificazione*: una mappa che assegna ad ogni documento, appartenente ad un certo insieme di documenti, una lista (eventualmente vuota) di keywords appartenenti ad una certa gerarchia. L'applicazione gestirà una sola Classificazione alla volta, la Classificazione corrente. Parimenti, gestirà una sola gerarchia di keywords alla volta, la gerarchia corrente che generalmente sarà la gerarchia associata alla Classificazione corrente. Le funzionalità che l'applicazione dovrà realizzare sono specificate nel seguente elenco.

Impostazione directory corrente
: L'utente può specificare il pathname assoluto di una directory che diventerà la directory corrente. Così, ogni volta che l'utente dovrà specificare un file o una directory avrà la possibilità di specificare il pathname assoluto o quello relativo alla directory corrente.

Nuova gerarchia di keywords
: Creazione di un nuova gerarchia di keywords che inizialmente consisterà solamente nella keyword speciale ANYTHING. Sarà creato un file per mantenere la gerarchia il cui pathname è specificato dall'utente.

Apertura gerarchia di keywords
: Apertura di un file, il cui pathname è specificato dall'utente, che contiene una gerarchia di keywords. Predisposizione della gerarchia per le operazioni di visualizzazione e modifica appresso descritte.

Navigazione keywords
: L'utente può specificare una keyword e chiedere che siano visualizzati tutti i nomi delle sub-keywords dirette della keyword o che sia visualizzato il nome della super-keyword diretta. L'utente può specificare o il path-keyword o solamente il nome. In quest'ultimo caso, se nella gerarchia ci sono due o più keywords con quel nome, l'applicazione mostra i path-keywords di

queste e permette all'utente di selezionarne una. Se l'utente ha scelto di visualizzare le sub-keywords poi potrà selezionarne una e scegliere se visualizzarne le sub-keywords o la super-keyword. Parimenti se l'utente ha chiesto di visualizzare la super-keyword poi potrà chiedere che l'applicazione ne visualizzi o le sub-keywords o la super-keyword. E così via, permettendo all'utente di navigare nell'intera gerarchia di keywords.

Aggiunta nuova keyword

L'utente può aggiungere una keyword specificando una keyword esistente e il nome della keyword che vuole aggiungere. La nuova keyword sarà aggiunta come sub-keyword diretta della keyword specificata, purché non ci sia già una sub-keyword diretta con lo stesso nome. L'applicazione assegnerà (internamente) un codice alla nuova keyword.

Cambio nome keyword

Specificando una keyword esistente l'utente potrà chiedere che sia cambiato il nome della keyword con un altro nome sempre specificato dall'utente. Il cambiamento è ammissibile purché il nuovo nome sia diverso da quello delle eventuali altre keywords che sono sub-keywords dirette della super-keyword della keyword che l'utente vuole cambiare. Il codice della keyword rimane invariato.

Nuova Classificazione

Per creare una nuova Classificazione l'utente deve specificare una gerarchia di keywords, il pathname di una directory e il nome da dare alla nuova Classificazione. Le keywords che si potranno associare ai documenti della nuova Classificazione potranno essere solamente quelle della gerarchia specificata. Inoltre, i files che mantengono la nuova Classificazione saranno creati nella directory specificata e avranno nomi che derivano dal nome della Classificazione. Dopo questa operazione la gerarchia di keywords corrente sarà quella della Classificazione.

Apertura Classificazione

L'utente può aprire una Classificazione esistente specificando il pathname di una directory, il nome di una Classificazione e una gerarchia di keywords. Se i files della Classificazione con il nome specificato risiedono nella directory specificata la Classificazione sarà aperta. Inoltre, la gerarchia di keywords della Classificazione diventerà la gerarchia corrente.

Navigazione Classificazione

L'utente può, specificando un documento (cioè il pathname del file), vedere la lista delle keywords associate a quel documento. Purché il documento faccia parte della classificazione. Inoltre, l'utente può chiedere che siano mostrati i pathnames di tutti i documenti della Classificazione (se sono molti una pagina alla volta) ed avere la possibilità di selezionarne uno per vederne la lista delle keywords.

Aggiunta documento

L'utente può aggiungere un documento alla Classificazione corrente specificando il pathname del file.

Associazione keyword

L'utente può specificare un documento della Classificazione corrente o tramite il pathname o tramite il solo nome. Se ci sono più documenti con lo stesso nome l'applicazione li mostrerà tutti lasciando all'utente la possibilità di selezionarne uno. Una volta che un documento è stato individuato saranno mostrate le keywords associate al documento. Poi l'utente avrà la possibilità di rimuovere una delle keywords oppure di aggiungere una keyword. La keyword aggiunta deve appartenere alla gerarchia associata alla Classificazione. Ciò che è associato al documento è in realtà il codice della keyword.

Ricerca

È possibile ricercare documenti della Classificazione corrente tramite espressioni booleane di keywords. Una *Espressione Booleana di Keywords*, in breve EBK, è definita induttivamente come segue:

- per ogni keyword `K`, sia `K` che `!K` sono EBK, più precisamente, sono EBK *atomiche*;
- se *E* e *F* sono EBK allora anche (*E* & *F*) e (*E* | *F*) sono EBK.

Come è facile intuire il carattere & denota l'AND, il carattere | denota l'OR e ! denota il NOT. Si noti che il NOT è applicabile solamente alle keywords. Un insieme di keywords *S soddisfa* una EBK *E* se l'espressione booleana che si ottiene da *E* sostituendo ogni occorrenza di EBK atomica *A* con un valore booleano (*true*, *false*), secondo le regole di seguito specificate, ha valore *true*:

- se *A* = !`K` (la keyword è negata) allora è sostituita con *false* se e solo se o `K` appartiene all'insieme *S* o c'è una keyword `K'` in *S* tale che `K'` è una sub-keyword di `K`;
- se *A* = `K` allora è sostituita con *true* se e solo se o `K` appartiene all'insieme *S* o c'è una keyword `K'` in *S* tale che `K'` è o una super-keyword di `K` o una sub-keyword di `K`.

Quindi un documento della Classificazione è *selezionato* da una EBK *E* se l'insieme delle keywords associato al documento soddisfa l'espressione *E*. L'utente può specificare una EBK e l'applicazione, se l'espressione è sintatticamente corretta, visualizza l'insieme dei documenti che sono selezionati dall'espressione. Nella specifica dell'EBK l'utente può indicare una keyword `K` o con una path-keyword o semplicemente con il nome della keyword. In quest'ultimo caso, ci possono essere più keywords che hanno lo stesso nome allora se `K` non è negata il valore booleano corrispondente sarà *true* se è *true* rispetto ad una delle keywords che hanno quel nome, se invece `K` è negata allora il valore booleano corrispondente sarà *false* se è *false* rispetto a una delle keywords che hanno quel nome. Consideriamo, ad esempio, le seguenti EBK:

```
E = ((origini & (musica>jazz>ragtime | rock and roll)) & !heavy metal)
F = ((heavy metal | !bebop) & !origini)
G = (!musica>jazz>origini & heavy metal)
```

Supponiamo che i documenti nella Classificazione abbiano le seguenti keywords:

```
D1    musica>jazz
D2    musica>rock
D3    musica>rock>rock and roll>rockabilly,  musica>jazz>origini
D4    musica>rock>origini,  musica>rock>heavy metal>folk metal
D5    musica>jazz,  musica>rock>heavy metal>grunge
```

Allora *E* seleziona i documenti D_1, D_2 e D_3, l'espressione *F* seleziona i documenti D_1, D_2 e D_5 e l'espressione *G* seleziona D_2, D_4 e D_5.

Ogniqualvolta una Classificazione o una gerarchia di keyword è chiusa (quando si passa ad un'altra Classificazione o gerarchia o quando l'applicazione viene teminata), l'applicazione deve aggiornare i corrispondenti files con le modifiche eventualmente apportate.

Maggiori dettagli sulle funzionalità sono forniti nelle prossime sezioni che specificano i principali moduli in cui l'implementazione dell'applicazione deve essere decomposta. I moduli sono stati pensati per migliorare sia la riusabilità che la leggibilità del codice e per facilitare eventuali estensioni dell'applicazione.

Interfaccia utente L'interfaccia utente può essere realizzata, a scelta, sia in forma testuale (cioè, accessibile tramite la linea di comando di un semplice terminale) sia con l'ausilio di componenti grafiche (`AWT` e `Swing`). Nel primo caso bisognerà realizzare opportuni menu e sotto-menu testuali.

Gestione delle keywords

La gestione delle gerarchie di keywords si deve basare su una classe `KeywordHierarchy` che rispetta il contratto qui di seguito specificato. L'intera implementazione della classe deve essere contenuta nel

package `metodologie.progetto.kh`. Nell'implementazione della classe non si possono aggiungere o rimuovere membri pubblici (campi, metodi e costruttori). Si possono invece aggiungere membri privati o con accessibilità ristretta al package.

metodologie.progetto.kh

Class KeywordHierarchy

```
public class KeywordHierarchy
extends Object
```

Questa classe gestisce gerarchie di keywords mantenute su files che obbediscono ad uno specifico formato.

I nomi delle keywords sono stringhe non vuote di qualsiasi lunghezza e composte da caratteri appartenenti all'insieme `{'a','b',...,'z','A','B',...'Z','0','1',...,'9','-','_',' '}` (cioè, caratteri alfabetici minuscoli e maiuscoli, cifre, trattino, underscore e spazio). I nomi delle keywords non sono sensibili alla capitalizzazione e non possono iniziare né terminare con un carattere spazio. Ad ogni keyword è assegnato un codice univoco di `CODELENGTH` caratteri appartenenti all'insieme `{'a','b',...,'z','A','B',...'Z','0','1',...,'9'}` (se `CODELENGTH` = 6 ci sono più di 56 miliardi di codici possibili). La keyword speciale `ANYTHING` ha sempre il codice `00...0`.

Una gerarchia di keywords è mantenuta in un file di testo (gestito quindi tramite accesso sequenziale) che ha la seguente struttura. Il nome del file deve essere *`name`*`.KWH` dove *`name`* è il nome della gerarchia. La prima linea del file contiene l'identificatore della gerarchia di keywords che è la stringa `KEYWORD_HIERARCHY:`*`name`*`:`*`N`* dove *`name`* è il nome della gerarchia e *`N`* è il numero di nanosecondi ritornato dal metodo `System.nanoTime()` nel momento in cui il file della gerarchia è creato. La seconda linea contiene *`lastcode`* che è l'ultimo codice di keyword usato nella gerarchia. Dopo le prime due linee, per ogni keyword (eccetto la keyword speciale `ANYTHING`) c'è una linea con il seguente formato: *`code`*`;`*`K`*`;`*`supercode`* dove *`code`* è il codice della keyword, *`K`* è il nome della keyword e *`supercode`* è il codice della super-keyword diretta. Queste informazioni sono sufficienti per ricostruire l'intera gerarchia.

Field Summary

static int	**CODELENGTH** La lunghezza dei codici delle keywords.
static char	**PATHSEP** Carattere usato come separatore nei path-keywords.

Constructor Summary

KeywordHierarchy(String dir, String name) Apre una gerarchia di keywords o ne crea una nuova.

Method Summary

String	**add**(String supercode, String kName) Aggiunge una nuova keyword alla gerarchia.

void	**close**() Chiude la gerarchia di keywords.
String[]	**codeFromName**(String kName) Ritorna in un array i codici delle keywords che hanno nome kName.
String	**codeFromPath**(String pathK) Ritorna il codice della keyword specificata dal path-keyword pathK.
String	**getID**() Ritorna l'identficatore di questa gerarchia di keywords.
boolean	**isSub**(String subcode, String code) Ritorna **true** se la keyword di codice subcode è uguale o è una sub-keyword della keyword di codice code.
boolean	**isSuper**(String supercode, String code) Ritorna **true** se la keyword di codice supercode è uguale o è una super-keyword della keyword di codice code.
String	**name**(String code) Ritorna il nome della keyword che ha codice code.
String	**path**(String code) Ritorna il path-keyword della keyword che ha codice code.
boolean	**replaceName**(String code, String kName) Sostituisce il nome della keyword di codice code con kName e ritorna **true**.
String[]	**subK**(String code) Ritorna in un array i codici delle keywords che sono sub-keywords dirette della keyword con codice code.
String	**superK**(String code) Ritorna il codice della super-keyword diretta della keyword di codice code.

Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

CODELENGTH

public static final int **CODELENGTH**

La lunghezza dei codici delle keywords: valore di default 6.

PATHSEP

public static final char **PATHSEP**

Carattere usato come separatore nei path-keywords: valore di default '>'.

Constructor Detail

KeywordHierarchy

```
public KeywordHierarchy(String dir,
                        String name)
                 throws java.io.FileNotFoundException
```

Apre una gerarchia di keywords o ne crea una nuova. Il nome della gerarchia è `name` e il file si trova nella directory di pathname `dir`.

Parameters:

`dir` - il pathname di una directory
`name` - il nome di una gerarchia di keywords

Throws:

`FileNotFoundException` - se il file non è accessibile.
`IllegalArgumentException` - se il file non è compatibile con la struttura di un file che contiene una gerarchia di keywords.

Method Detail

getID

```
public String getID()
```

Ritorna l'identficatore di questa gerarchia di keywords. Ovvero una stringa contenente `KEYWORD_HIERARCHY:`*`name`*`:`*`N`* dove *`name`* è il nome della gerarchia e *`N`* è il numero di nanosecondi ritornato dal metodo `System.nanoTime()` nel momento in cui il file della gerarchia è creato. L'identificatore è registrato nel file della gerarchia.

Returns:

l'identificatore di questa gerarchia di keywords.

---

codeFromPath

```
public String (String pathK)
```

Ritorna il codice della keyword specificata dal path-keyword `pathK`. Se non è presente ritorna **`null`**.

Parameters:

`pathK` - un path-keyword.

Returns:

il codice della keyword o **`null`**.

---

codeFromName

```
public String[] codeFromName(String kName)
```

Ritorna in un array i codici delle keywords che hanno nome `kName`. Se non ci sono keyword con quel nome ritorna un array di lunghezza 0.

Parameters:
`kName` - un nome di keyword.
Returns:
l'array dei codici delle keyword con il nome dato.

---

path

```
public String path(String code)
```

Ritorna il path-keyword della keyword che ha codice `code`. Se non ci sono keyword con quel codice ritorna **`null`**.

Parameters:
`code` - il codice di una keyword.
Returns:
un path-keyword o **`null`**.

---

name

```
public String name(String code)
```

Ritorna il nome della keyword che ha codice `code`. Se non ci sono keywords con quel codice ritorna **`null`**.

Parameters:
`code` - il codice di una keyword
Returns:
il nome della keyword o **`null`**.

---

superK

```
public String superK(String code)
```

Ritorna il codice della super-keyword diretta della keyword di codice `code`. Se non ci sono keywords con quel codice ritorna **`null`**.

Parameters:
`code` - il codice di una keyword.
Returns:
il codice della super-keyword diretta o **`null`**.

---

subK

```
public String[] subK(String code)
```

Ritorna in un array i codici delle keywords che sono sub-keywords dirette della keyword con codice `code`. Se non ci sono sub-keywords, ritorna un array di lunghezza 0 e se non ci sono keywords con codice `code`, ritorna **`null`**.

Parameters:
`code` - il codice di una keyword.
Returns:
l'array dei codici delle sub-keywords dirette o **`null`**.

---

isSub

```
public boolean isSub(String subcode,
                     String code)
```

Ritorna **`true`** se la keyword di codice `subcode` è uguale o è una sub-keyword della keyword di codice `code`. Altrimenti ritorna **`false`**, anche quando uno dei codici è errato.

Parameters:
`subcode` - il codice di una keyword.
`code` - il codice di una keyword.
Returns:
`true` o **`false`**.

---

isSuper

```
public boolean isSuper(String supercode,
                       String code)
```

Ritorna **`true`** se la keyword di codice `supercode` è uguale o è una super-keyword della keyword di codice `code`. Altrimenti ritorna **`false`**, anche quando uno dei codici è errato.

Parameters:
`supercode` - il codice di una keyword.
`code` - il codice di una keyword.
Returns:
`true` o **`false`**.

---

add

```
public String add(String supercode,
                  String kName)
```

Aggiunge una nuova keyword alla gerarchia. Il nome della nuova keyword è `kName` ed è aggiunta come sub-keyword diretta della keyword di codice `supercode`. Ritorna il codice assegnato alla nuova keyword. Se l'operazione non può essere effettuata (perché `supercode` non esiste o `kName` non è un nome ammissibile o esiste già una sub-keyword diretta della keyword `supercode` con quello stesso nome), allora ritorna **`null`**.

Parameters:
`supercode` - il codice della super-keyword diretta.

`kName` - il nome della nuova keyword.

Returns:

il codice della nuova keyword o **`null`**.

---

replaceName

```
public boolean replaceName(String code,
                           String kName)
```

Sostituisce il nome della keyword di codice `code` con `kName` e ritorna **`true`**. Se l'operazione fallisce (perché o non esistono keyword con quel codice, o il nuovo nome non è ammissibile o il nuovo nome coincide con quello di un'altra keyword che è una sub-keyword diretta della stessa super-keyword diretta della keyword di codice `code`), allora ritorna **`false`**,

Parameters:

`code` - il codice di una keyword.

`kName` - il nuovo nome della keyword.

Returns:

`true` o, se fallisce, **`false`**.

---

close

```
public void close()
```

Chiude la gerarchia di keywords. Se sono state apportate delle modifiche sono salvate nel file. Tutte le risorse sono rilasciate. Dopo l'invocazione di questo metodo l'invocazione di un qualunque metodo della classe dovrebbe provocare il lancio di una eccezione di tipo `IllegalStateException`.

Gestione delle Classificazioni

La gestione delle Classificazioni deve basarsi su una classe `Classification` che rispetta il contratto qui di seguito specificato. L'intera implementazione della classe deve essere contenuta nel package `metodologie.progetto.classif`. Nell'implementazione della classe non si possono aggiungere o rimuovere membri pubblici (campi, metodi e costruttori). Si possono invece aggiungere membri privati o con accessibilità ristretta al package.

metodologie.progetto.classif

Class Classification

All Implemented Interfaces:

Iterable<String>

```
public class Classification
```

```
extends Object
implements Iterable<String>
```

Questa classe gestisce Classificazioni che sono matenute in files che obbediscono a un opportuno formato. La classe opera in congiunzione con la classe `metodologie.progetto.kh.KeywordHierarchy`. La natura degli elementi di una Classificazione non è specificata (possono essere files, siti web, oggetti fisici come libri o dvd, ecc.). Ogni elemento è identificato da una stringa chiamata appunto l'*identificatore* dell'elemento (se gli elementi sono files l'identificatore potrebbe essere il pathname, se sono siti web potrebbe essere l'URL, se sono libri l'identificatore potrebbe essere la collocazione).

Siccome gli identificatori possono essere stringhe piuttosto lunghe e in vista di possibili estensioni che permettano di gestire anche link tra elementi, conviene assegnare ad ogni elemento un codice, ovviamente univoco. Il codice è composto da `CODELENGTH` caratteri appartenenti all'insieme `{'a','b',...,'z','A','B',...'Z','0','1',...,'9'}`.

Una Classificazione è mantenuta in due files. Il file degli elementi con nome *`name`*`.ELM` e il file delle keywords con nome *`name`*`.KYW`. La stringa *`name`* deve essere comune ad entrambi i files e rappresenta il nome della Classificazione. Entrambi i files sono di tipo testo e quindi sono gestiti tramite accesso sequenziale. La struttura del file degli elementi è la seguente. La prima linea contiene `CLASSIFICATION:`*`name`*`:`*`N`*, dove *`name è il nome della Classificazione e N è il numero di nanosecondi ritornato dal metodo System.nanoTime() nel momento in cui la Classificazione è creata. La seconda linea contiene lastcode che è l'ultimo codice usato per gli elementi della Classificazione. Poi c'è una linea per ogni elemento contenente code;ID, dove code è il codice dell'elemento e ID è l'identificatore dell'elemento. La struttura del file delle keywords è la seguente. La prima linea contiene CLASSIFICATION_KEYWORDS:name:N, dove name è il nome della Classificazione N è il numero di nanosecondi mantenuto nel file degli elementi. La seconda linea contiene l'identificatore della gerarchia di keywords usata (cioè la stringa ritornata dal metodo getID() della classe metodologie.progetto.kh.KeywordHierarchy). Poi per ogni elemento che ha almeno una keyword associata c'è una linea contenente code;K1;K2;...KN, dove code è il codice dell'elemento e K1, K1,..., KN sono i codici delle keywords associate all'elemento.`*

Field Summary

static int	**CODELENGTH** La lunghezza dei codici degli elementi.

Constructor Summary

Classification(String dir, String name, KeywordHierarchy kh) Apre una Classificazione o ne crea una nuova.

Method Summary

String	**add**(String id) Aggiunge alla Classificazione un nuovo elemento.
boolean	**addKeyword**(String code, String kCode) Associa ad un elemento una nuova keyword.
void	**close**() Chiude la Classificazione.
String	**code**(String id)

	Ritorna il codice dell'elemento con identificatore id.
String	**id**(String code) Ritorna l'identificatore dell'elemento di codice code.
java.util.Iterator<String>	**iterator**() Ritorna un iteratore sui codici degli elementi.
String[]	**keywords**(String code) Ritorna in un array i codici delle keywords associate all'elemento di codice code.
boolean	**removeKeyword**(String code, String kCode) Rimuove la keyword di codice kCode dalle keywords associate all'elemento di codice code.

Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

CODELENGTH

public static final int ***CODELENGTH***

La lunghezza dei codici degli elementi: valore di default 6.

Constructor Detail

Classification

public ***Classification****(String dir,*
String name,
KeywordHierarchy kh)
throws java.io.FileNotFoundException

Apre una Classificazione o ne crea una nuova. Il nome della Classificazione è name e i files si trovano nella directory di pathname dir. La Classificazione usa la gerarchia di keyword kh.

Parameters:
dir - il pathname di una directory.
name - il nome di una Classificazione.
kh - una gerarchia di keyword.
Throws:
FileNotFoundException - se uno dei files coinvolti non è accessibile.
IllegalArgumentException - se kh è ***null*** *o uno dei files non è compatibile con la struttura di un file di una Classificazione o la gerarchia di keywords non è compatibile con la Classificazione (ad esempio l'identificatore della gerarchia è differente da quello riportato nel file delle keywords).*

Method Detail

code

*public String **code**(String id)*

*Ritorna il codice dell'elemento con identificatore id. Se non c'è nessun elemento con quell'identificatore ritorna **null**.*

Parameters:
id - l'identificatore di un elemento.
Returns:
*il codice dell'elemento o **null**.*

---

id

*public String **id**(String code)*

*Ritorna l'identificatore dell'elemento di codice code. Se non c'è nessun elemento con quel codice ritorna **null**.*

Parameters:
code - il codice di un elemento
Returns:
*l'identificatore dell'elemento o **null**.*

---

iterator

*public java.util.Iterator<String> **iterator**()*

Ritorna un iteratore sui codici degli elementi. L'oggetto di tipo Iterator<String> ritornato permette di scorrere i codici di tutti gli elementi della Classificazione (anche quelli che non hanno keywords associate).

Specified by:
iterator in interface Iterable<String>

Returns:
un iteratore sui codici degli elementi.

---

keywords

*public String[] **keywords**(String code)*

*Ritorna in un array i codici delle keywords associate all'elemento di codice code. Se L'elemento non ha keywords associate ritorna un array di lunghezza 0. Se l'elemento non esiste ritorna **null**.*

Parameters:
code - il codice di un elemento.
Returns:
*un array dei codici delle keywords associate all'elemento o **null**.*

---

add

*public String **add**(String id)*

*Aggiunge alla Classificazione un nuovo elemento. L'identificatore del nuovo elemento è id. Ritorna il codice assegnato all'elemento. Se c'è già un elemento con quel identificatore, allora non aggiunge l'elemento e ritorna **null**.*

Parameters:
id - l'identficatore di un nuovo elemento.
Returns:
*il codice del nuovo elemento o **null**.*

---

addKeyword

```
public boolean addKeyword(String code,
                          String kCode)
```

*Associa ad un elemento una nuova keyword. Il codice dell'elemento è code e il codice della keyword è kCode. Se l'operazione ha successo ritorna **true**. Se la keyword era già stata associata all'elemento o uno dei codici è errato ritorna **false**.*

Parameters:
code - il codice di un elemento.
kCode - il codice di una keyword.
Returns:
***true** se l'operazione ha successo, **false** altrimenti.*

---

removeKeyword

```
public boolean removeKeyword(String code,
                             String kCode)
```

*Rimuove la keyword di codice kCode dalle keywords associate all'elemento di codice code. Se l'operazione ha successo ritorna **true**. Se la keyword non è presente o uno dei codici è errato, ritorna **false**.*

Parameters:
code - il codice di un elemento.
kCode - il codice di una keyword.
Returns:
***true** se l'operazione ha successo, **false** altrimenti.*

---

close

```
public void close()
```

Chiude la Classificazione. Se sono state apportate delle modifiche sono salvate nei files. Tutte le risorse sono rilasciate. Dopo l'invocazione di questo metodo l'invocazione di uno qualunque dei metodi della classe dovrebbe provocare il lancio di una eccezione di tipo IllegalStateException.

Ricerca di documenti

Le ricerche di documenti effettuate mediante EBK deve basarsi sulla implementazione della classe StrBoolExpr e dell'interfaccia SBEAtom. L'intera implementazione della classe StrBoolExpr deve essere contenuta nel package metodologie.progetto.sbe. Per potenziare le possibilità di riuso ed estensione del codice la valutazione di una EBK è stata divisa in due parti: la parte che riguarda la valutazione delle EBK atomiche (interfaccia SBEAtom) e la parte che riguarda la valutazione dell'espressone booleana, dopo che le espressioni atomiche sono state valutate (classe StrBoolExpr). Inoltre, sia l'interfaccia che la classe sono state pensate per la valutazione di espressioni più generali delle EBK che

sono state chiamate SBE (String Boolean Expression). Le SBE sono definite nella documentazione della classe StrBoolExpr. La connessione tra EBK e SBE è spiegata nei dettagli alla fine di questa sezione.

metodologie.progetto.sbe

Interface SBEAtom

---

public interface ***SBEAtom***

Interfaccia per oggetti che implementano la valutazione delle SBE atomiche.

Method Summary	
boolean	**check**(String atom) Ritorna **true** se la stringa atom contiene una SBE atomica ammissibile.
boolean	**eval**(String atom) Ritorna il valore booleano della SBE atomica atom.

Method Detail

check

boolean ***check****(String atom)*

Ritorna ***true*** *se la stringa atom contiene una SBE atomica ammissibile. Altrimenti ritorna* ***true****.*

Parameters:
atom - una stringa.
Returns:
true *se è una SBE atomica ammissibile.*

---

eval

boolean ***eval****(String atom)*

Ritorna il valore booleano della SBE atomica atom. Se non è una SBE atomica ammissibile lancia l'eccezione IllegalArgumentException.

Parameters:
atom - una stringa.
Returns:
il valore della SBE atomica.
Throws:
IllegalArgumentException - se non è una SBE atomica ammissibile.

---

metodologie.progetto.sbe

Class StrBoolExpr

---

public class ***StrBoolExpr***

extends Object

Questa classe permette di valutare String Boolean Expressions, in breve SBE.

Le SBE possono essere definite induttivamente:

- *una qualsiasi stringa non vuota che non contiene i caratteri '(', ')', '&', '|' e che non inizia né termina con il carattere spazio è una SBE atomica;*
- *se E e F sono SBE allora anche (E & F) e (E | F) sono SBE.*

Una volta che i valori booleani delle SBE atomiche che occorrono in una SBE sono dati, il valore booleano dell'intera espressione è determinato.

Constructor Summary

StrBoolExpr(String expr, SBEAtom atom) Crea un oggetto relativo alla (presunta) SBE nella stringa expr e il valutatore di SBE atomiche atom.

Method Summary

int	**check**() Fa un controllo sintattico della espressione dell'oggetto: parentesi, operatori, espressioni atomiche.
boolean	**eval**() Ritorna il valore della SBE.

Methods inherited from class Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

StrBoolExpr

*public **StrBoolExpr**(String expr,*
SBEAtom atom)

Crea un oggetto relativo alla (presunta) SBE nella stringa expr e il valutatore di SBE atomiche atom.

Parameters:
expr - una stringa contenente una (presunta) SBE.
atom - un valutatore di SBE atomiche.

Method Detail

check

*public int **check**()*

Fa un controllo sintattico della espressione dell'oggetto: parentesi, operatori, espressioni atomiche. Se il controllo ha successo ritorna 0, altrimenti ritorna 1 + j, dove j è l'indice del primo carattere della stringa che provoca un errore sintattico.

Returns:
0 se il controllo ha successo e un valore positivo altrimenti.

---

eval

*public boolean **eval()***

Ritorna il valore della SBE.

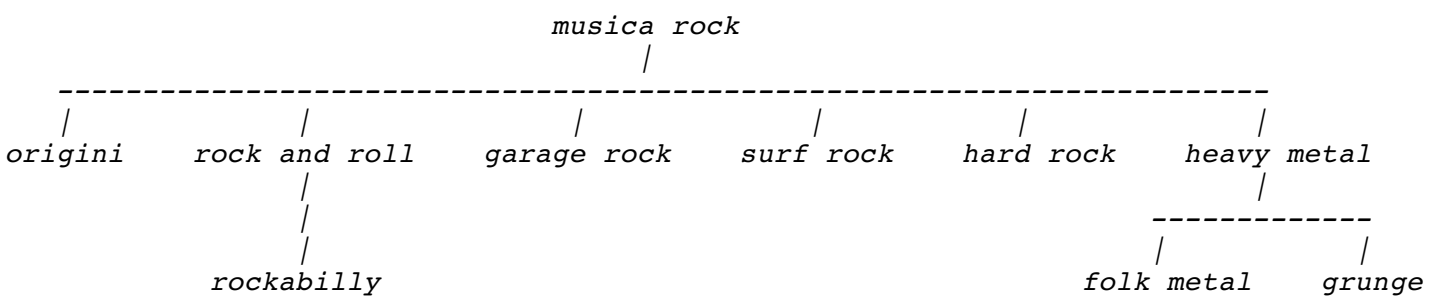
Returns:
il valore della SBE.
Throws:
IllegalStateException - se l'SBE non è corretta.

Nel caso delle EBK le SBE atomiche sono esattamente le EBK atomiche. Tramite una opportuna classe che implementa l'interfaccia SBEAtom è possibile realizzare un valutatore di EBK atomiche che in congiunzione con una implementazione della classe StrBoolExpr permetterà di valutare una qualsiasi EBK. Si noti che l'implementazione della classe StrBoolExpr non dipende dal particolare tipo di SBE (EBK, variazioni di EBK o altro). Solamente la classe che realizza un valutatore di SBE atomiche dipende dalle specificità di tali espressioni.

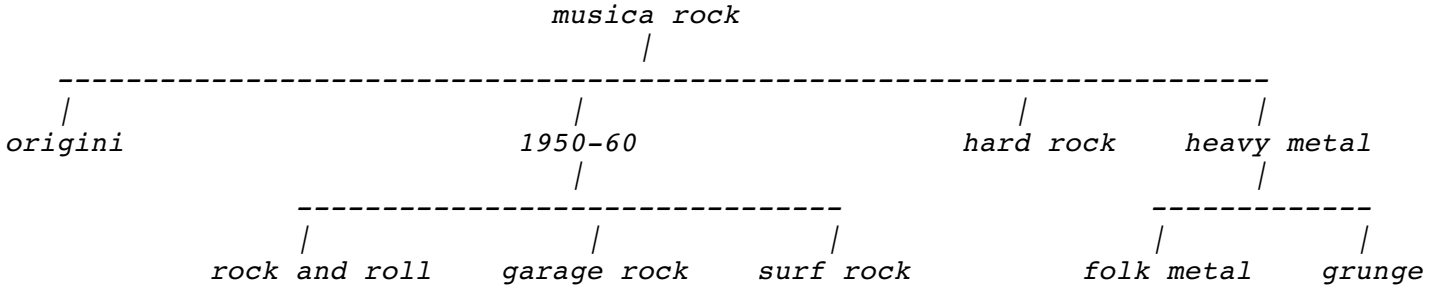
Gruppi con due o tre studenti

Se il progetto è realizzato da un gruppo con due o tre studenti allora, oltre ad osservare le specifiche descritte precedentemente, l'applicazione dovrà anche implementare alcune ulteriori funzionalità.

Gruppi con due studenti *L'applicazione deve permettere all'utente di aggiungere ad una gerarchia keywords interne, cioè, keywords che si situano tra keywords già esistenti. Consideriamo, ad esempio, la seguente gerarchia:*



L'utente potrebbe voler aggiungere una keyword 1950-60 per classificare anche in termini cronologici i vari stili della musica rock. Allora l'applicazione deve permettere di fare ciò, trasformando la gerarchia precedente in quella qui sotto:



Quindi la nuova keyword è stata inserita come sub-keyword diretta di musica rock e alcune sub-keywords dirette di quest'ultima sono ora diventate sub-keyword dirette della nuova keyword.

L'implementazione di questa funzionalità deve basarsi su un metodo pubblico che deve essere aggiunto alla classe KeywordHierarchy:

insert

```
public String insert(String supercode,
                     String kName
                     String...subs)
```

*Inserisce una nuova keyword all'interno della gerarchia. Il nome della nuova keyword è kName, è inserita come sub-keyword diretta della keyword di codice supercode e le keywords i cui codici sono elencati in subs diventano sub-keywords dirette della nuova keyword. Ritorna il codice assegnato alla nuova keyword. Se l'operazione non può essere effettuata, ritorna **null**. Le ragioni che possono causare il fallimento dell'operazioni sono le seguenti:*

- *supercode non esiste;*
- *kName non è un nome ammissibile;*
- *esiste già una sub-keyword diretta della keyword supercode che ha nome kName;*
- *i codici delle keywords elencati in subs non sono codici di sub-keyword dirette della keyword supercode.*

I codici di tutte le keywords già esistenti rimangono invariati.

Parameters:
supercode - il codice della super-keyword diretta.
kName - il nome della nuova keyword.
subs - l'elenco dei codici delle sub-keywords dirette.
Returns:
*il codice della nuova keyword o **null**.*

Gruppi con tre studenti

I gruppi con tre studenti oltre a realizzare quanto richiesto per gruppi con due studenti devono implementare una ulteriore funzionalità. La possibilità di creare una nuova Classificazione mediante la "fusione" di due Classificazioni esistenti che usano la stessa gerarchia di keywords. La realizzazione di tale funzionalità deve basarsi su un metodo statico che deve essere aggiunto alla classe Classification:

merge

```
public static Classification merge(String dir,
                                   String name,
                                   Classification c1,
                                   Classification c2)
```

*Crea una nuova Classificazione tramite la "fusione" delle Classificazioni c1 e c2. Il nome della nuova Classificazione è name e i relativi files sono creati nella directory di pathname dir. Ritorna il riferimento alla nuova Classificazione. Le due Classificazioni devono usare la stessa gerarchia di keywords che sarà anche la gerarchia usata dalla nuova Classificazione. Nella nuova Classificazione sono inseriti tutti gli elementi appartenenti alle due Classificazioni e le relative associazioni di keywords. Se ci sono due elementi uguali (cioè, con lo stesso identificatore) nelle due Classificazioni allora la lista delle keywords associata all'elemento nella nuova Classificazione è l'unione delle due liste (ovviamente, senza ripetizioni). Se l'operazione fallisce ritorna **null**.*

Parameters:
dir - il pathname di una directory.
name - il nome della nuova Classificazione.

c1 - una Classificazione.
c2 - un'altra Classificazione.
Returns:
*la nuova Classificazione o **null**.*

Chiaramente, l'applicazione deve realizzare una opportuna interfaccia per dare all'utente la possibilità di usufruire di questa operazione.

Valutazione del progetto

Il progetto consegnato sarà valutato relativamente a diversi aspetti. Gli aspetti che saranno considerati nella valutazione, ordinati per importanza decrescente, sono i seguenti:

1. *rispetto delle specifiche e correttezza del codice;*
2. *qualità della progettazione;*
3. *documentazione del codice tramite javadoc;*
4. *leggibilità del codice e stile di programmazione;*
5. *qualità dell'interfaccia utente;*
6. *efficienza.*

In particolare, relativamente ai primi due aspetti, sarà valutato come sono state analizzate ed affrontate le situazioni che possono portare ad errori e che non sono state menzionate nelle specifiche.

Per ulteriori informazioni e chiarimenti circa il progetto rivolgersi al docente. Le modalità per la consegna saranno pubblicate sulle pagine del corso.