Università di Roma "La Sapienza"
Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE I"
(Corso di Laurea in Ingegneria Informatica)
Proff. Giuseppe De Giacomo e Marco Cadoli
Canali A-L & M-Z
A.A. 2004-05

Compito d'esame del 1° aprile 2005

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda la gestione di *playlist* musicali su dispositivi informatici portatili. Una playlist è formata da un insieme di file musicali da suonare in un certo ordine. Essa ha associato un nome ed una durata (calcolata come la somma delle durate in secondi dei file musicali in essa presenti). Ciascun file musicale è caratterizzato da un nome, da una dimensione in kilobyte, da una durata in secondi e può essere solo in due formati: *traccia-cd* e *mp3*. Dei file in formato traccia-cd viene memorizzato il numero della traccia sul cd di cui originariamente faceva parte. I file in formato mp3 hanno associato un *tag* contenente il nome del brano musicale, il nome dell'artista e l'anno in cui il brano è stato scritto.

Una playlist, una volta preparata, può essere mandata in esecuzione. Una volta in esecuzione, la playlist viene ciclicamente ripetuta, a meno che non venga messa in pausa o in stop. Quando è in pausa o in stop può essere rimessa in esecuzione. La playlist può essere modificata solo quando è in stop.

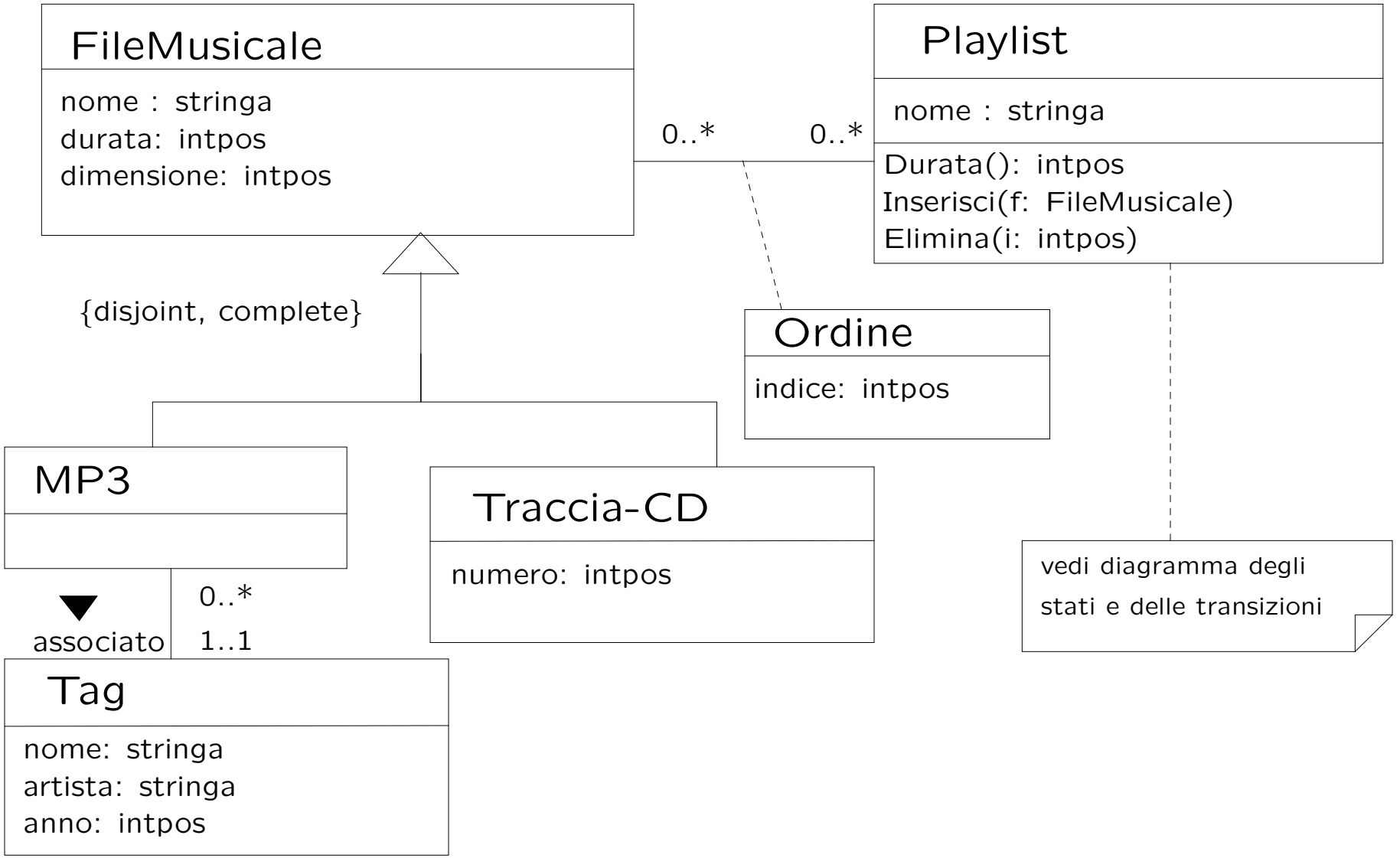
Requisiti (cont.)

L'utente del dispositivo musicale portatile è interessato ad effettuare i seguenti controlli:

- data una playlist p ed un numero intero n, verificare se la dimensione complessiva dei file che compongono p è minore di n kilobyte.
- data una playlist p, calcolare la playlist formata dai soli file mp3 presenti in p mantenendo l'ordine relativo degli stessi.
- dato un file musicale m, restituire il numero di playlist in cui esso è presente.

Fase di analisi

Diagramma delle classi



Commento sul diagramma delle classi

La possibilità di modificare l'insieme di file musicali associati ad una playlist si evince dai requisiti.

Abbiamo assunto che le uniche operazioni di modifica di una playlist fossero l'inserimento di un file musicale nella prima posizione disponibile e la cancellazione di un file data la posizione.

Altre soluzioni erano ovviamente possibili, ad esempio l'inserimento di un file data la sua posizione desiderata. In tale caso tuttavia sarebbe più oneroso il controllo che l'ordine sia ben fatto (senza duplicati e senza "buchi").

Per semplicità, per la classe *FileMusicale* non sono state previste operazioni di inserimento/cancellazione in/da una playlist.

Diagramma degli stati e delle transizioni classe Playlist

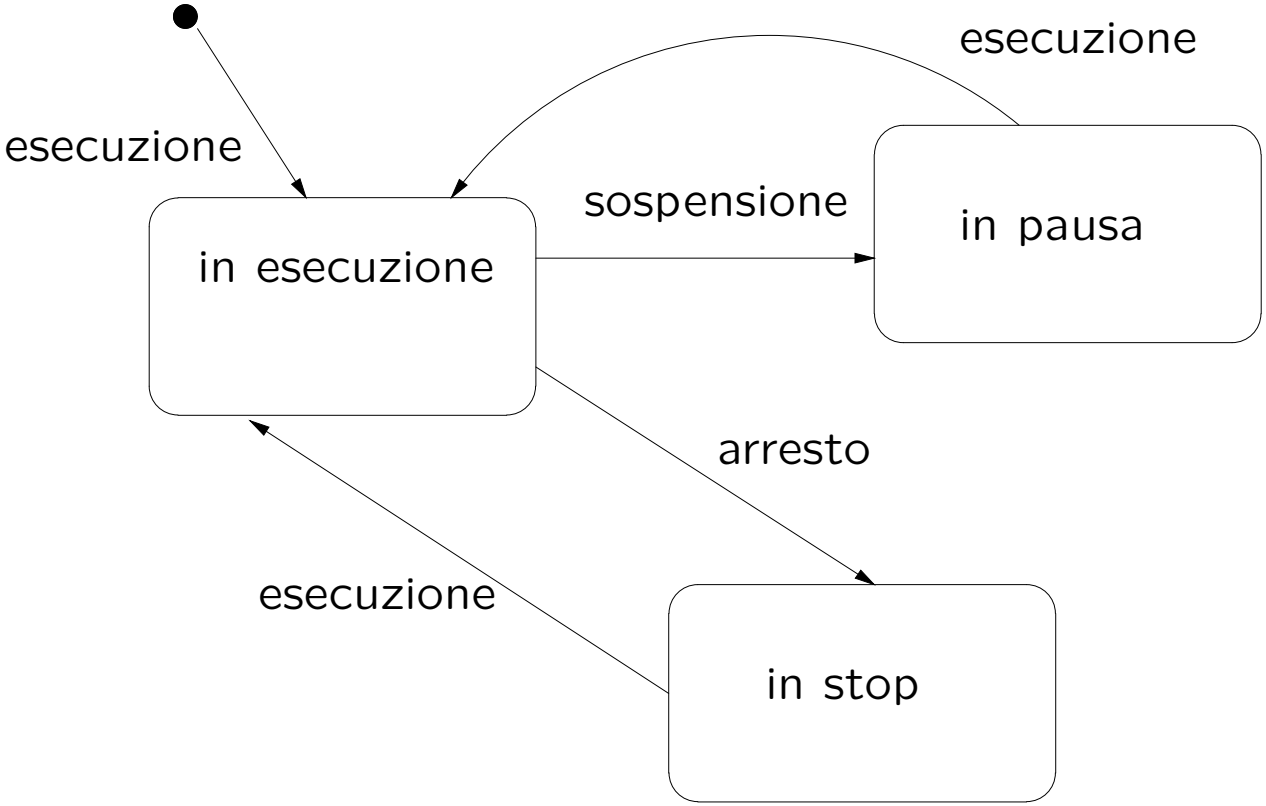
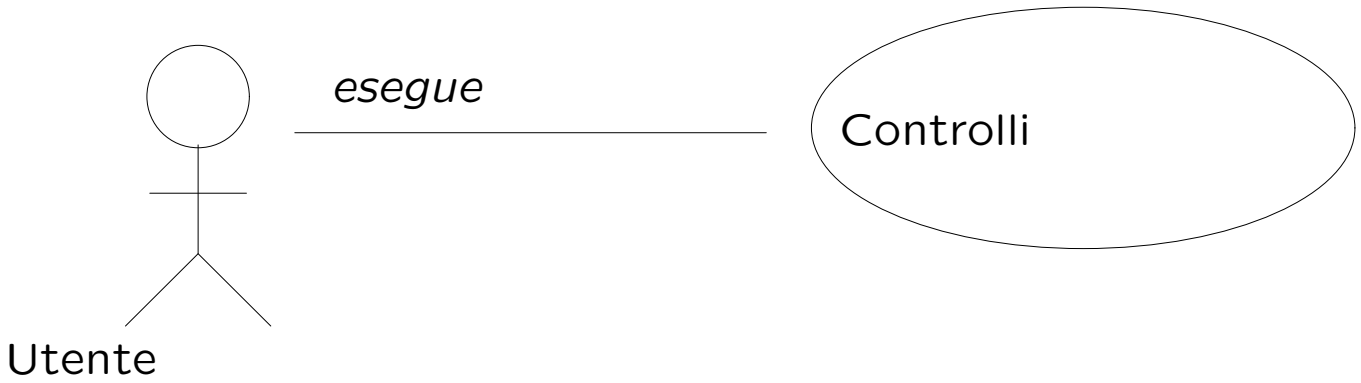


Diagramma degli use case



Specifica della classe Playlist

InizioSpecificaClasse Playlist

Durata *(): intpos*

pre: nessuna

post: *result* è pari a $\sum_{l \in L} l.FileMusicale.durata$, dove L è l'insieme di link di tipo *Ordine* legati all'oggetto *this*.

Inserisci *(f: FileMusicale)*

pre: *this.IsInStop()* = true. Inoltre, nell'insieme L di link di tipo *Ordine* legati all'oggetto *this* non ve n'è alcuno il cui oggetto *FileMusicale* sia *f*

post: *this* è legato ad un nuovo link di tipo *Ordine*, il cui oggetto *FileMusicale* è *f*, e il cui attributo *indice* è pari a $|pre(L)| + 1$.

...

Specifica della classe Playlist (cont.)

...

Elimina *(i: intpos)*

pre: *this.IsInStop()* true. Inoltre, $i \leq |L|$, dove L è l'insieme di link di tipo *Ordine* legati all'oggetto *this*

post: l'insieme di link di tipo *Ordine* legati all'oggetto *this* è l'unione dei seguenti insiemi:

- $\{l \mid pre(l) \in L \ \wedge \ pre(l).indice \leq i-1 \ \wedge$
  $l.FileMusicale = pre(l).FileMusicale \ \wedge \ l.indice = pre(l).indice\}$

- $\{l \mid pre(l) \in L \ \wedge \ pre(l).indice > i \ \wedge$
  $l.FileMusicale = pre(l).FileMusicale \ \wedge \ l.indice = pre(l).indice - 1\}$

FineSpecifica

Specifica dello use case

InizioSpecificaUseCase Controlli

VerificaDimensione *(p: Playlist, n: intpos): booleano*

pre: nessuna

post: *result* è pari a *true* se $\sum_{l \in L} l.FileMusicale.dimensione < n$, dove L è l'insieme di link di tipo *Ordine* legati all'oggetto *p*

SoliMP3 *(p: Playlist): Playlist*

pre: nessuna

post: *result* è una nuova playlist tale che, per ogni link l di tipo *Ordine* legato all'oggetto *p* tale che $l.IsMP3 = true$, esiste un link s di tipo *Ordine* legato all'oggetto *result*. Inoltre i valori $s.indice$ hanno lo stesso ordinamento (senza buchi) dei valori $l.indice$

...

Specifica dello use case (cont.)

...

QuanteListe *(f: FileMusicale): intpos*

pre: nessuna

post: *result* è pari a $|L|$, dove L è l'insieme di link di tipo *Ordine* legati all'oggetto *f*

FineSpecifica

Fase di progetto

Algoritmi per le operazioni delle classi

Adottiamo i seguenti algoritmi:

- Per l'operazione **Durata** della classe *Playlist*:

```
int result = 0;
per ogni link l di tipo Ordine in cui this è coinvolto
        result += l.FileMusicale.durata;
return result;
```

- Per l'operazione **Inserisci** della classe *Playlist*:

```
int max = numero di link di tipo Ordine in cui this è coinvolto;

crea un nuovo link s di tipo Ordine;
s.indice = max + 1;
s.Playlist = this;
s.FileMusicale = f;
```

Algoritmi per le operazioni delle classi (cont.)

- Per l'operazione **Elimina** della classe *Playlist*:

```
Insieme(Ordine) L = insieme di link di tipo Ordine in cui this è coinvolto;
per ogni link l di L
        se (l.indice == i) allora
              elimina l da L;
        se (l.indice > i) allora
              l.indice--;
```

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **VerificaDimensione**:

```
int sum = 0;
per ogni link l di tipo Ordine in cui p è coinvolto
        sum += p.FileMusicale.dimensione;
return sum < n;
```

- Per l'operazione **SoliMP3**:

```
Playlist result = copia di p, compresi i link di tipo Ordine;
per ogni link l di tipo Ordine in cui result è coinvolto
        se (l.FileMusicale.IsMP3() == true) allora
                int index = l.indice;
                result.Elimina(i);
return result;
```

Algoritmi per le operazioni degli use case (cont.)

- Per l'operazione **QuanteListe**:

```
Insieme(Ordine) L = insieme di link di tipo Ordine in cui f è coinvolto;
return f.size();
```

Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

1. i requisiti,

2. la specifica degli algoritmi per le operazioni di classe e use-case,

3. i vincoli di molteplicità nel diagramma delle classi.

Associazione	Classe	ha resp.
Ordine	*FileMusicale*	SÌ[1,2]
	PlayList	SÌ[1,2]
associato	*MP3*	SÌ[3]
	Tag	NO

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, a causa:

- dei vincoli di molteplicità 0..* delle associazioni,
- delle variabili locali necessarie per vari algoritmi.

Per fare ciò, utilizzeremo la classe Java `InsiemeListaOmogeneo`.

API per le strutture di dati

```
// File insiemelista/InsiemeListaOmogeneo.java

package insiemelista;

public class InsiemeListaOmogeneo extends InsiemeLista {
  public InsiemeListaOmogeneo(Class cl)
  public InsiemeListaOmogeneo()
  public int size()
  public boolean isEmpty()
  public boolean contains(Object e)
  public boolean add(Object e)
  public boolean remove(Object e)
  public Iterator iterator()
  public boolean containsAll(Collection c)
  public Object[] toArray()
  public Object[] toArray(Object[] a)
  public boolean equals(Object o)
  public Object clone()
  public String toString()
}
```

Corrispondenza fra tipi UML e Java

Riassumiamo le nostre scelte nella seguente tabella di corrispondenza dei tipi UML.

Tipo UML	Rappresentazione in Java
booleano	`boolean`
intero	`int`
intpos	`int`
stringa	`String`
Insieme	`InsiemeListaOmogeneo`

Per tenere conto del fatto che, nei casi "intpos" e "intero" il tipo Java è semanticamente più esteso del corrispondente tipo UML, prevediamo una verifica delle condizioni di ammissibilità sul lato server, perché è una soluzione di migliore qualità.

Tabelle di gestione delle proprietà di classi UML

Riassumiamo le nostre scelte differenti da quelle di default mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita*.

Classe UML	**Proprietà immutabile**
FileMusicale	*nome*
	durata
	dimensione
	associato
Tag	*nome*
	artista
	anno
Traccia-CD	*numero*

Classe UML	**Proprietà**	
	nota alla nascita	**non nota alla nascita**

Altre considerazioni

Sequenza di nascita degli oggetti: Poiché le responsabilità su *associato* è singola, e la molteplicità è (nel verso della responsabilità) 1..1, è ragionevole assumere che quando nasce un oggetto Java corrispondente ad MP3 sia nota il tag associato.

Valori alla nascita: Non sembra ragionevole assumere che per qualche proprietà esistano valori di default validi per tutti gli oggetti.

Rappresentazione degli stati in Java

Per la classe UML *Playlist*, ci dobbiamo occupare della rappresentazione in Java del diagramma degli stati e delle transizioni.

Scegliamo di rappresentare gli stati mediante una variabile `int`, secondo la seguente tabella.

Stato	Rappresentazione in Java	
	tipo var.	`int`
	nome var.	`stato`
in esecuzione	valore	1
in pausa	valore	2
in stop	valore	3

API delle classi Java progettate

A titolo di esempio, viene fornita la API della classe `FileMusicale`:

```
public abstract class FileMusicale {
// COSTRUTTORE
    public FileMusicale(String no, int du, int di)
// GESTIONE ATTRIBUTI
    public String getNome()
    public int getDurata()
    public int  getDimensione()
// GESTIONE ASSOCIAZIONI
//   - Ordine
    public void inserisciLinkOrdine(AssociazioneOrdine a)
    public void eliminaLinkOrdine(AssociazioneOrdine a)
    public InsiemeListaOmogeneo getLinkOrdine()
// STAMPA
    public String toString()
}
```

Fase di realizzazione

Considerazioni iniziali

Dalle fasi precedenti traiamo come conseguenza che dobbiamo realizzare:

1. cinque classi UML, di cui una ha associato un diagramma degli stati e delle transizioni;

2. una associazione a responsabilità singola e con vincoli di molteplicità 0..* e 1..1 (molteplicità minima diversa da zero, ma immutabile e nota alla nascita);

   una associazione con attributi e a responsabilità doppia e con vincoli di molteplicità 0..* e 0..*;

3. uno use case.

Per facilitare la fase di test e debugging, prevediamo l'overriding della funzione `toString()` per ognuna delle classi Java di cui al punto 1.

Sommario classi Java relative a classi UML

`Playlist:` responsabilità su *Ordine* (doppia); ha diagramma stati e transizioni.

`FileMusicale:` astratta; responsabilità su *Ordine* (doppia).

`TracciaCD:` derivata; nessuna responsabilità.

`MP3:` derivata; responsabilità su *associato* (singola); cardinalità minima maggiore di zero su *associato*.

`Tag:` nessuna responsabilità.

Struttura dei file e dei package

```
+---insiemelista
|       InsiemeListaOmogeneo.java
|       InsiemeLista.java
|       IteratorInsiemeLista.java
|
\---AppPlayer
    |   TestPlayer.java
    |   Controlli.java
    |   AssociazioneOrdine.java
    |   TipoLinkOrdine.java
    |   EccezionePrecondizioni.java
    |   Playlist.java
    |   Tag.java
    |
    +---TracciaCD
    |       TracciaCD.java
    |
    +---FileMusicale
    |       FileMusicale.java
    |
    \---MP3
            MP3.java
```

La classe Java FileMusicale

```
// File AppPlayer/FileMusicale/FileMusicale.java
package AppPlayer.FileMusicale;
import insiemelista.*;
import AppPlayer.*;
import java.util.*;

public abstract class FileMusicale {
    protected final String nome;
    protected final int durata, dimensione;
    protected InsiemeListaOmogeneo ordine;
    public FileMusicale(String no, int du, int di)
        throws EccezionePrecondizioni {
        if (du <= 0)  // CONTROLLO PRECONDIZIONI
            throw new
                EccezionePrecondizioni
                ("La durata del file musicale deve essere positiva");
        if (di <= 0)  // CONTROLLO PRECONDIZIONI
            throw new
                EccezionePrecondizioni
                ("La dimensione del file musicale deve essere positiva");
        nome = no;
        durata = du;
        dimensione = di;
        ordine = new InsiemeListaOmogeneo(TipoLinkOrdine.class);
```

```
    }
    public String getNome() { return nome; }
    public int getDurata() { return durata; }
    public int  getDimensione() { return dimensione; }
    public void inserisciLinkOrdine(AssociazioneOrdine a) {
        if (a != null) ordine.add(a.getLink());
    }
    public void eliminaLinkOrdine(AssociazioneOrdine a) {
        if (a != null) ordine.remove(a.getLink());
    }
    public InsiemeListaOmogeneo getLinkOrdine() {
        return (InsiemeListaOmogeneo)ordine.clone();
    }
    public String toString() {
        return nome + ", " + durata + " sec, " + dimensione + " kB";
    }
}
```

La classe Java MP3

```
// File AppPlayer/MP3/MP3.java
package AppPlayer.MP3;
import AppPlayer.*;
import AppPlayer.FileMusicale.*;

public class MP3 extends FileMusicale {
    protected final Tag associato;
    public MP3(String no, int du, int di, Tag as)
        throws EccezionePrecondizioni {
        super(no,du,di);
        associato = as;
    }
    public Tag getAssociato() {
        return associato;
    }
    public String toString() {
        return super.toString() + " . MP3, " + associato;
    };
}
```

La classe Java TracciaCD

```
// File AppPlayer/TracciaCD/TracciaCD.java
package AppPlayer.TracciaCD;
import AppPlayer.*;
import AppPlayer.FileMusicale.*;

public class TracciaCD extends FileMusicale {
    protected final int numero;
    public TracciaCD(String no, int du, int di, int nu)
        throws EccezionePrecondizioni {
        super(no,du,di);
        if (nu <= 0)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Il numero della traccia deve essere positivo");
        numero = nu;
    }
    public int getNumero() {
        return numero;
    }
    public String toString() {
        return super.toString()  + " . Numero traccia CD: " + numero;
    };
}
```

La classe Java Playlist

```
// File AppPlayer/Playlist.java
package AppPlayer;
import AppPlayer.FileMusicale.*;
import insiemelista.*;
import java.util.*;

public class Playlist {
    private final String nome;
    private InsiemeListaOmogeneo ordine;
    protected static final int in_esecuzione = 1,
        in_pausa = 2, in_stop = 3;
    protected int stato_corrente = in_esecuzione;
    public Playlist(String no) {
        nome = no;
        ordine = new InsiemeListaOmogeneo(TipoLinkOrdine.class);
    }
    public String getNome() { return nome; }
    public int durata() {
        int result = 0;
        Iterator it = ordine.iterator();
        while(it.hasNext()) {
            TipoLinkOrdine t = (TipoLinkOrdine)it.next();
            result+= t.getFileMusicale().getDurata();
        }
```

```
    return result;
}
public void inserisciLinkOrdine(AssociazioneOrdine a) {
    if (a != null) ordine.add(a.getLink());
}
public void eliminaLinkOrdine(AssociazioneOrdine a) {
    if (a != null) ordine.remove(a.getLink());
}
public InsiemeListaOmogeneo getLinkOrdine() {
    return (InsiemeListaOmogeneo)ordine.clone();
}
public boolean InStop() {
    return stato_corrente == in_stop;
};
public void esecuzione() {
    if (stato_corrente == in_pausa ||
        stato_corrente == in_stop)
        stato_corrente = in_esecuzione;
};
public void sospensione() {
    if (stato_corrente == in_esecuzione)
        stato_corrente = in_pausa;
};
public void arresto() {
    if (stato_corrente == in_esecuzione)
        stato_corrente = in_stop;
```

```
};
public void inserisci(FileMusicale f)
    throws EccezionePrecondizioni {
    // CONTROLLO PRIMA PRECONDIZIONE
    if (stato_corrente != in_stop)
        throw new EccezionePrecondizioni
            ("La lista deve essere nello stato di stop");
    int max = ordine.size();
    Iterator it = ordine.iterator();
    while(it.hasNext()) {
        TipoLinkOrdine t = (TipoLinkOrdine)it.next();
        if (t.getFileMusicale() == f)
            // CONTROLLO SECONDA PRECONDIZIONE
        throw new EccezionePrecondizioni
            ("File gia' presente in lista");
    }
    TipoLinkOrdine t = null;
    try {
        t = new TipoLinkOrdine(f,this,max + 1);
    }
    catch (EccezionePrecondizioni e) {
        System.out.println(e);
    }
    AssociazioneOrdine.inserisci(t);
}
public void elimina(int i)
```

```
throws EccezionePrecondizioni {
// CONTROLLO PRIMA PRECONDIZIONE
if (stato_corrente != in_stop)
    throw new EccezionePrecondizioni
        ("La lista deve essere nello stato di stop");
// CONTROLLO SECONDA PRECONDIZIONE
if (ordine.size() < i)
    throw new EccezionePrecondizioni
        ("Indice troppo elevato");
Iterator it = ordine.iterator();
while(it.hasNext()) {
    TipoLinkOrdine t = (TipoLinkOrdine)it.next();
    if (t.getIndice() == i)
        AssociazioneOrdine.elimina(t);
    if (t.getIndice() > i) {
        FileMusicale f = t.getFileMusicale();
        TipoLinkOrdine t2 = null;
        try {
            t2 = new TipoLinkOrdine(f,this,t.getIndice() - 1);
        }
        catch (EccezionePrecondizioni e) {
            System.out.println(e);
        }
        AssociazioneOrdine.elimina(t);
        AssociazioneOrdine.inserisci(t2);
    }
```

```
        }
    }
    public String toString() {
        Iterator it = ordine.iterator();
        String result;
        result = "Playlist: " +  nome + ", durata " + durata() +
            " secondi\nFile musicali contenuti:\n";
        while(it.hasNext()) {
            TipoLinkOrdine t = (TipoLinkOrdine)it.next();
            result+= " " + t.getIndice() + ": " +
                t.getFileMusicale().toString() + "\n";
        }
        return result;
    }
}
```

La classe Java Tag

```
// File AppPlayer/Tag.java
package AppPlayer;

public class Tag {
    private final String nome, artista;
    private final int anno;
    public Tag(String no, String ar, int an)
        throws EccezionePrecondizioni {
        if (an <= 0)  // CONTROLLO PRECONDIZIONI
            throw new
                EccezionePrecondizioni
                ("L'anno del tag deve essere positivo");
        nome = no;
        artista = ar ;
        anno = an;
    }
    public String getNome() { return nome; }
    public String getArtista() { return artista; }
    public int getAnno() { return anno; }
    public String toString() {
        return "Tag: " +  nome + ", " +
            anno + ", artista " + artista;
    }
}
```

La classe Java TipoLinkOrdine

```
// File AppPlayer/TipoLinkOrdine.java
package AppPlayer;
import AppPlayer.FileMusicale.*;
import AppPlayer.Playlist.*;

public class TipoLinkOrdine {
    private final FileMusicale ilFileMusicale;
    private final Playlist laPlaylist;
    private final int indice;
    public TipoLinkOrdine(FileMusicale f, Playlist p, int c)
        throws EccezionePrecondizioni {
        if (f == null || p == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        ilFileMusicale = f; laPlaylist = p; indice = c;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkOrdine b = (TipoLinkOrdine)o;
            return b.laPlaylist == laPlaylist &&
                b.ilFileMusicale == ilFileMusicale;
        }
        else return false;
    }
```

```
    public FileMusicale getFileMusicale() { return ilFileMusicale; }
    public Playlist getPlaylist() { return laPlaylist; }
    public int getIndice() { return indice; }
    public String toString() {
        return ilFileMusicale.getNome() + " " +
            laPlaylist.getNome() + " " + indice;
    }
}
```

La classe Java AssociazioneOrdine

```
// File AppPlayer/AssociazioneOrdine.java
package AppPlayer;

public class AssociazioneOrdine {
  private AssociazioneOrdine(TipoLinkOrdine x) { link = x; }
  private TipoLinkOrdine link;
  public TipoLinkOrdine getLink() { return link; }
  public static void inserisci(TipoLinkOrdine y) {
    if (y != null) {
      AssociazioneOrdine k = new AssociazioneOrdine(y);
      k.link.getFileMusicale().inserisciLinkOrdine(k);
      k.link.getPlaylist().inserisciLinkOrdine(k);
    }
  }
  public static void elimina(TipoLinkOrdine y) {
    if (y != null) {
      AssociazioneOrdine k = new AssociazioneOrdine(y);
      k.link.getFileMusicale().eliminaLinkOrdine(k);
      k.link.getPlaylist().eliminaLinkOrdine(k);
    }
  }
}
```

Realizzazione in Java dello use case

```
// File AppPlayer/Controlli.java
package AppPlayer;
import java.util.*;
import insiemelista.*;
import AppPlayer.MP3.*;
import AppPlayer.TracciaCD.*;

public final class Controlli {
    private Controlli() { }
    public static boolean VerificaDimensione(Playlist p, int n)
        throws EccezionePrecondizioni {
        if (n <= 0)  // CONTROLLO PRECONDIZIONI
            throw new
                EccezionePrecondizioni
                ("La dimensione da controllare deve essere positiva");
        int sum = 0;
        InsiemeListaOmogeneo ins = p.getLinkOrdine();
        Iterator it = ins.iterator();
        while(it.hasNext()) {
            TipoLinkOrdine t = (TipoLinkOrdine)it.next();
            sum+= t.getFileMusicale().getDimensione();

        }
        return sum < n;
```

```
    }
}
```