Università di Roma "La Sapienza"
Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE I"
(Corso di Laurea in Ingegneria Informatica)
Prof. Giuseppe De Giacomo
Canali A-L & M-Z
A.A. 2006-07

Compito d'esame del 26 marzo 2007

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda la gestione di librerie di contenuti multimediali. Ogni libreria ha un nome (una stringa) ed un insieme arbitrario di elementi. Ogni elemento è caratterizzato da un nome di un file (una stringa). Gli elementi sono di diversi tipi, tra questi abbiamo i video e i brani musicali. I video sono a loro volta suddivisi in film e video musicali. Dei film interessa il titolo (una stringa) e il regista (una stringa). Di ciascun video musicale interessa il brano musicale a cui è associato. Dei brani musicali interessa il nome (una stringa), l'artista (una stringa) e opzionalmente il nome dell'album in cui il brano è originariamente apparso (una stringa).

Un elemento può essere disponibile, in accesso, o in aggiornamento. Quando è disponibile può essere messo in accesso o in aggiornamento. Sia quando è in accesso che in aggiornamento può essere reso nuovamente disponibile. L'elemento può essere modificato solo quando questo è in aggiornamento.

Requisiti (cont.)

Il fruitore della applicazione è interessato ad effettuare diverse operazioni, in particolare:

- data una libreria ℓ, verificare se essa è formata solo da brani musicali e video musicali;

- dato un brano b, restituire una nuova libreria ℓ avente come nome il nome di b e contenente b e tutti i video musicali associati a b.

Requisiti (cont.)

Domanda 1. Basandosi sui requisiti riportati sopra, effettuare la fase di analisi producendo lo schema concettuale in UML per l'applicazione e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

Domanda 2. Effettuare la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

È obbligatorio solo progettare gli algoritmi e definire le responsabilità sulle associazioni.

Domanda 3. Effettuare la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

Requisiti (cont.)

È obbligatorio realizzare in Java solo i seguenti aspetti dello schema concettuale:

- la gerarchia di generalizzazione che parte dalla classe `Elemento` e le eventuali associazioni tra sottoclassi di questa gerarchia.

Fase di analisi

Diagramma delle classi

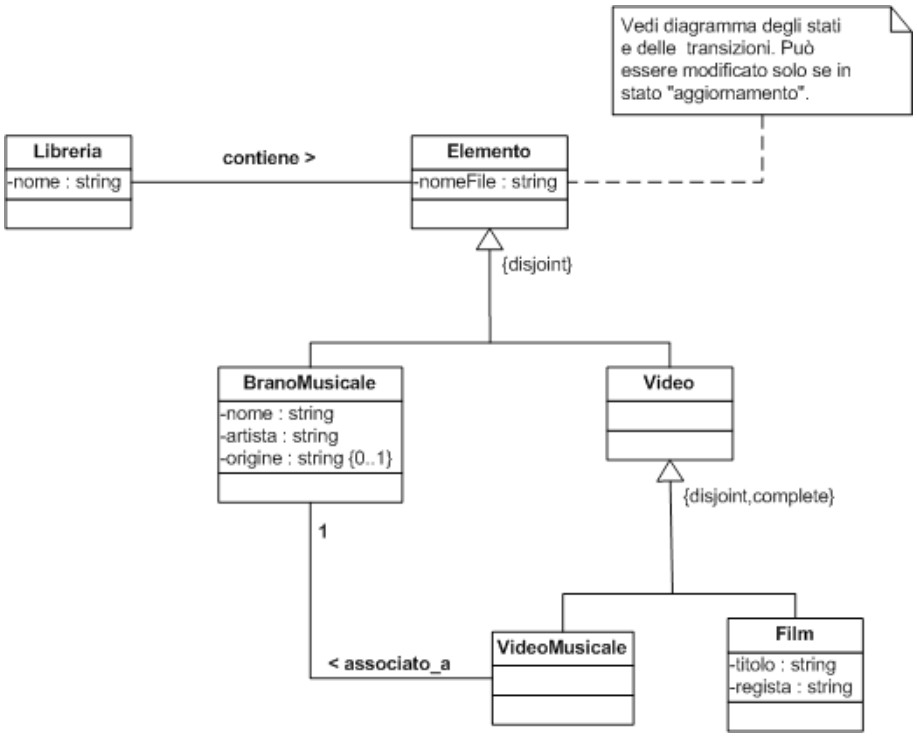
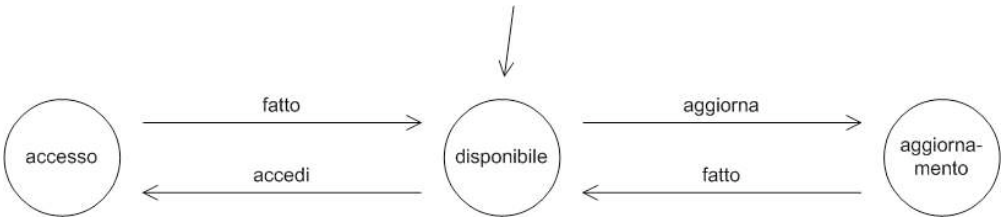
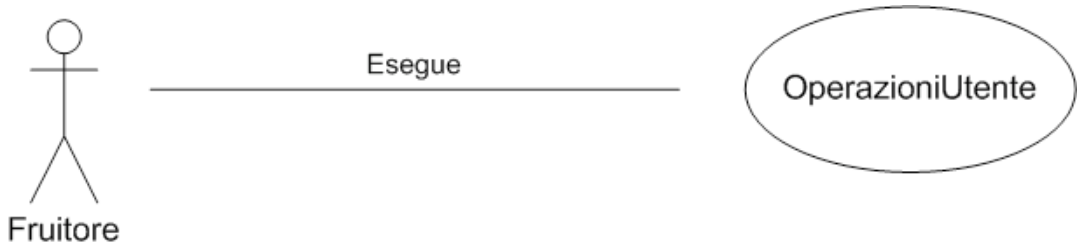


Diagramma degli stati e delle transizioni della classe Elemento



NOTA: Lo stato iniziale può essere scelto arbitrariamente. In questa soluzione si assume che un elemento, alla nascita, sia disponibile.

Diagramma degli use case



Specifica dello use case

InizioSpecificaUseCase OperazioniUtente

tuttoMusicale *(l: Libreria): booleano*

pre: `true`

post: *result* = `true` se e solo se

$$\forall e \in \{e \mid e \in Elemento \wedge \langle l, e\rangle \in contiene\}$$
si ha $(e \in BranoMusicale \vee e \in VideoMusicale)$

...

Specifica dello use case (cont.)

...

libreriaDaBrano *(b: BranoMusicale): Libreria*

pre: `true`

post: $result \in Libreria$ è tale che:

- $result.nome = b.nome$
- $\langle b, result \rangle \in contiene$ e

$$\forall v \in \{v \mid v \in VideoMusicale \wedge \langle v, b \rangle \in associato\_a\}$$
si ha $\langle v, result \rangle \in contiene$

e non ci sono altre tuple della forma $\langle v, result \rangle$ in $contiene$.

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **tuttoMusicale**(*l: Libreria*): *booleano*

```
per ogni link x di tipo contiene in cui l è coinvolto
     se (x.Elemento.class != BranoMusicale && x.Elemento.class != VideoMusicale)
          return false;
return true;
```

- Per l'operazione **libreriaDaBrano**(*b: BranoMusicale*): *Libreria*

```
result = new Libreria;
result.nome = b.nome;
contiene = contiene union {<result,b>};
per ogni link l di tipo associato_a in cui b è coinvolto {
     contiene = contiene union {<result, l.VideoMusicale>};
}
```


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.
contiene	*Libreria*	SÌ[2]
	Elemento	NO
associato_a	*VideoMusicale*	SÌ[3]
	BranoMusicale	SÌ[2]

Strutture di dati

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

- dei vincoli di molteplicità 0..* delle associazioni,

Per fare ciò, utilizzeremo le classi del collection framework di Java 1.5: `Set`, `HashSet`.

Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
stringa	`String`
booleano	`boolean`
Insieme	`HashSet`

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**
Libreria	*nome*
Elemento	*nomeFile*
BranoMusicale	*nome*
	artista
Film	*titolo*
	regista

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

Altre considerazioni

Sequenza di nascita degli oggetti: Non dobbiamo assumere una particolare sequenza di nascita degli oggetti.

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 *Elemento*, 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`
disponibile	valore	1
accesso	valore	2
aggiornamento	valore	3

API delle classi Java progettate

Omesse per brevità (si faccia riferimento al codice Java).

Fase di realizzazione

Considerazioni iniziali

La traccia ci richiede di realizzare:

1. La classe Elemento e tutte le sue specializzazioni, ovvero le classi: *Elemento*, *BranoMusicale*, *Video*, *VideoMusicale*, *Film*.

2. l'associazione UML *associato_a* con responsabilità doppia e con vincoli di molteplicità 1..1 (molteplicità massima finita e molteplicità minima diversa da zero) e 0..*;

Nel seguito verranno realizzate tutte le classi e gli use case individuati in fase di analisi.

Struttura dei file e dei package

```
+---AppLibrerie
   |   Libreria.java
   |   AssociazioneAssociatoA.java
   |   TipoLinkAssociatoA.java
   |   OperazioniUtente.java
   |   EccezionePrecondizioni.java
   |   EccezioneCardMin.java
   |
   +---Elemento
   |       Elemento.java
   |
   +---Video
   |       Video.java
   |
   +---BranoMusicale
   |       BranoMusicale.java
   |
   +---VideoMusicale
   |       VideoMusicale.java
   |
   \---Film
           Film.java
```


La classe Java Libreria

```
// File AppLibrerie/Libreria.java
package AppLibrerie;

import AppLibrerie.Elemento.*;
import java.util.*;

public final class Libreria {
  private String nome;
  private HashSet<Elemento> contiene;

  public Libreria(String nome){
    this.nome = nome;
    contiene = new HashSet<Elemento>();
  }

  public String getNome(){
    return nome;
  }

  public void inserisciLinkContiene(Elemento e){
        if (e != null)
          contiene.add(e);
  }
```

```
  public void eliminaLinkContiene(Elemento e){
        if (e != null) contiene.remove(e);
  }

  public Set<Elemento> getLinkContiene(){
            return (HashSet<Elemento>)contiene.clone();
  }
}
```

La classe Java Elemento

```
// File AppLibrerie/Elemento/Elemento.java
package AppLibrerie.Elemento;

import java.util.*;

public class Elemento {
  protected String nomeFile;
  private final static int disponibile = 1;
  private final static int accesso = 2;
  private final static int aggiornamento = 3;
  private int corrente = disponibile;

  public Elemento(String nomeFile) {
    this.nomeFile = nomeFile;
  }

  public String getNomeFile(){
    return nomeFile;
  }

  public void accedi(){
    if (corrente == disponibile)
      corrente = accesso;
  }
```

```
  public void aggiorna(){
    if (corrente == disponibile)
      corrente = aggiornamento;
  }

  public void fatto(){
    if (corrente == accesso || corrente == aggiornamento)
      corrente = disponibile;
  }

  public boolean estInAggiornamento(){
    return corrente == aggiornamento;
  }
}
```

La classe Java BranoMusicale

```
// File AppLibrerie/BranoMusicale/BranoMusicale.java
package AppLibrerie.BranoMusicale;

import AppLibrerie.*;
import AppLibrerie.Elemento.*;

import java.util.*;

public final class BranoMusicale extends Elemento{
  private String nome;
  private String artista;
  private String origine;
  private HashSet<TipoLinkAssociatoA> associatoA;

  public BranoMusicale(String nomeFile, String nome, String artista) {
    super(nomeFile);
    this.nome = nome;
    this.artista = artista;
    this.origine = null;
  }

  public String getNome() {
    return nome;
  }
```

```
  public String getArtista() {
    return artista;
  }

  public String getOrigine() {
    return origine;
  }

  public void setOrigine(String origine) {
   if (!estInAggiornamento()) // Verifica lo stato corrente
      throw
        new EccezionePrecondizioni("Elemento modificabile solo se in aggiornamento");
     this.origine=origine;
  }

  public void inserisciLinkAssociatoA(AssociazioneAssociatoA a)
    if (a != null){
      associatoA.add(a.getLink());
    }
  }

  public void eliminaLinkAssociatoA(AssociazioneAssociatoA a)
    if (a != null){
      associatoA.remove(a.getLink());
    }









  }

  public Set<TipoLinkAssociatoA> getLinkAssociatoA() {
    return (HashSet<TipoLinkAssociatoA>) associatoA.clone();
  }
}
```

La classe Java Video

```
// File AppLibrerie/Video/Video.java
package AppLibrerie.Video;

import AppLibrerie.Elemento.*;

import java.util.*;

public abstract class Video extends Elemento{
  protected Video(String nomeFile){
    super(nomeFile);
  }
}

//Nota: il costruttore può anche essere dichiarato public
```


La classe Java VideoMusicale

```
// File AppLibrerie/VideoMusicale/VideoMusicale.java
package AppLibrerie.VideoMusicale;

import AppLibrerie.EccezioneCardMin;
import AppLibrerie.EccezionePrecondizioni;
import AppLibrerie.Video.*;
import AppLibrerie.BranoMusicale.*;
import AppLibrerie.TipoLinkAssociatoA;
import AppLibrerie.AssociazioneAssociatoA;

import java.util.*;

public final class VideoMusicale extends Video {
  private TipoLinkAssociatoA linkAssociatoA;

  public VideoMusicale(String nomeFile, BranoMusicale b) {
    super(nomeFile);
    linkAssociatoA = null;
  }

  public int quantiAssociatoA() {
    if (linkAssociatoA == null)
      return 0;
    else return 1;
```

```
  }

  public void inserisciLinkAssociatoA(AssociazioneAssociatoA a)
    if (a != null){
      linkAssociatoA = a.getLink();
    }
  }

  public void eliminaLinkAssociatoA(AssociazioneAssociatoA a)
    if (a != null){
      linkAssociatoA = null;
    }
  }

  public TipoLinkAssociatoA getLinkAssociatoA() throws EccezioneCardMin{
    if (linkAssociatoA == null)
      throw new EccezioneCardMin("Cardinalità minima violata");
    return linkAssociatoA;
  }
}
```

La classe Java Film

```
// File AppLibrerie/Film/Film.java
package AppLibrerie.Film;

import AppLibrerie.Video.*;

import java.util.*;

public final class Film extends Video {
  private String titolo;
  private String regista;

  public Film(String nomeFile, String titolo, String regista){
    super(nomeFile);
    this.titolo = titolo;
    this.regista = regista;
  }
}
```

La classe Java TipoLinkAssociatoA

```
// File AppLibrerie/TipoLinkAssociatoA.java
package AppLibrerie;

import AppLibrerie.BranoMusicale.*;
import AppLibrerie.VideoMusicale.*;
import java.util.*;

public class TipoLinkAssociatoA {
    private final VideoMusicale ilVideoMusicale;
    private final BranoMusicale ilBranoMusicale;

    public TipoLinkAssociatoA(VideoMusicale v, BranoMusicale b)
        throws EccezionePrecondizioni {
        if (v == null || b == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        ilVideoMusicale = v;
        ilBranoMusicale = b;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkAssociatoA b = (TipoLinkAssociatoA) o;
            return b.ilVideoMusicale == ilVideoMusicale &&
```



```
                b.ilBranoMusicale == ilBranoMusicale;
        }
        else return false;
    }

    public int hashCode() {
      return  ilVideoMusicale.hashCode() + ilBranoMusicale.hashCode();
    }

    public VideoMusicale getVideoMusicale(){
      return ilVideoMusicale;
    }

    public BranoMusicale getBranoMusicale(){
      return ilBranoMusicale;
    }

    public String toString() {
        return "<" + ilVideoMusicale + ", " + ilBranoMusicale + ">";
    }
}
```

La classe Java AssociazioneAssociatoA

```
// File AppLibrerie/AssociazioneAssociatoA.java
package AppLibrerie;

public final class AssociazioneAssociatoA {
  private TipoLinkAssociatoA link;

  private AssociazioneAssociatoA(TipoLinkAssociatoA x){
    link = x;
  }

  public TipoLinkAssociatoA getLink(){
    return link;
  }

  public static void inserisci(TipoLinkAssociatoA y) {
    if (y != null &&                                  //il link e' significativo
        !y.getBranoMusicale().estInAggiornamento() && //il brano non e' in aggiornamento
        !y.getVideoMusicale().estInAggiornamento() && //il video non e' in aggiornamento
        y.getVideoMusicale().quantiAssociatoA == 0)   //il video non e' gia'
    {                                                 //associato a un brano
      AssociazioneAssociatoA k = new AssociazioneAssociatoA(y);
      y.getBranoMusicale().inserisciLinkAssociatoA(k);
      y.getVideoMusicale().inserisciLinkAssociatoA(k);
    }
```



```
  }

  public static void elimina(TipoLinkAssociatoA y) {
    if (y != null &&
        !y.getBranoMusicale().estInAggiornamento() &&
        !y.getVideoMusicale().estInAggiornamento()) {
      AssociazioneAssociatoA k = new AssociazioneAssociatoA(y);
      y.getBranoMusicale().eliminaLinkAssociatoA(k);
      y.getVideoMusicale().eliminaLinkAssociatoA(k);
    }
  }
}
```

La classe Java OperazioniUtente

```
// File AppLibrerie/OperazioniUtente.java
package AppLibrerie;

import AppLibrerie.BranoMusicale.*;
import AppLibrerie.Elemento.*;
import AppLibrerie.Video.*;

import java.util.*;

public final class OperazioniUtente{
  private OperazioniUtente(){}

  public static boolean tuttoMusicale(Libreria l){
    Set<Elemento> elementi = l.getLinkContiene();
    Iterator<Elemento> ie = elementi.iterator();
    while (ie.hasNext()){
      Elemento curr = ie.next();
      if (!(curr instanceof BranoMusicale) && !(curr instanceof VideoMusicale))
        return false;
    }
      return true;
  }

  public static Libreria libreriaDaBrano(BranoMusicale b){
```



```
    Libreria result = new Libreria(b.getNome());
    result.inserisciLinkContiene(b);
    Set<TipoLinkAssociatoA> video = b.getLinkAssociatoA();
    Iterator<TipoLinkAssociatoA> iv = video.iterator();
    while (iv.hasNext())
      result.inserisciLinkContiene(iv.next().getVideoMusicale());
    return result;
  }
}
```

Realizzazione in Java delle classi per eccezioni

```
// File AppLibrerie/EccezioneCardMin.java
package AppLibrerie;

public class EccezioneCardMin extends Exception {
    private String messaggio;
    public EccezioneCardMin(String m) {
        messaggio = m;
    }
    public String toString() {
        return messaggio;
    }
}
```

```
// File AppLibrerie/EccezionePrecondizioni.java
package AppLibrerie;

public class EccezionePrecondizioni extends RuntimeException {
  private String messaggio;
  public EccezionePrecondizioni(String m) {
    messaggio = m;
  }
  public EccezionePrecondizioni() {
    messaggio = "Si e' verificata una violazione delle precondizioni";
  }
```



```
  public String toString() {
    return messaggio;
  }
}
```