Università di Roma "La Sapienza"
Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE I"
(Corso di Laurea in Ingegneria Informatica)
Proff. Marco Cadoli e Giuseppe De Giacomo
Canali A-L & M-Z
A.A. 2005-06

Compito d'esame del 7 luglio 2006

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda la gestione di "slideshow" di fotografie in formato elettronico. Uno slideshow è caratterizzato dal nome (una stringa che rappresenta il nome del file nel quale lo slideshow è memorizzato), da un insieme non vuoto e ordinato di fotografie, e da un brano musicale (che viene usato come colonna sonora durante l'esecuzione dello slideshow). Una fotografia è caratterizzata da un nome (una stringa che rappresenta il nome del file nel quale è memorizzata l'immagine) e dal tipo dell'immagine stessa (una stringa, per esempio "panorama", "primo piano", "piano americano", ecc.). Alcuni slideshow sono "speciali" in quanto possono associare ad alcune fotografie degli effetti di visualizzazione speciali. Gli effetti stessi sono identificati da un codice numerico (un intero). Gli slideshow speciali hanno un titolo (una stringa) che viene visualizzato quando vanno in esecuzione.

Requisiti (cont.)

Uno slide show può essere in tre stati "non-attivo", "in-esecuzione", "in-pausa". Quando non attivo può essere messo in esecuzione con il comando "play"; quando in esecuzione può essere messo in pausa con il comando "pause" o nello stato non-attivo con il comando "stop"; quando in pausa può essere rimesso in esecuzione con il comando "play". Lo slideshow può essere modificato (aggiungendo o eliminando fotografie, ecc.) solo quando nello stato "non-attivo".

L'utente dell'applicazione è interessato ad effettuare diverse operazioni tra cui:

- dato uno slideshow s ed un tipo di fotografia t, generare in modo automatico un nuovo slideshow s' contenente la stessa colonna sonora di s e solo le fotografie del tipo t presenti in s (in un ordine qualsiasi);

- data una fotografia f restituire il numero di slideshow speciali in cui essa è presente.

Fase di analisi

Diagramma delle classi

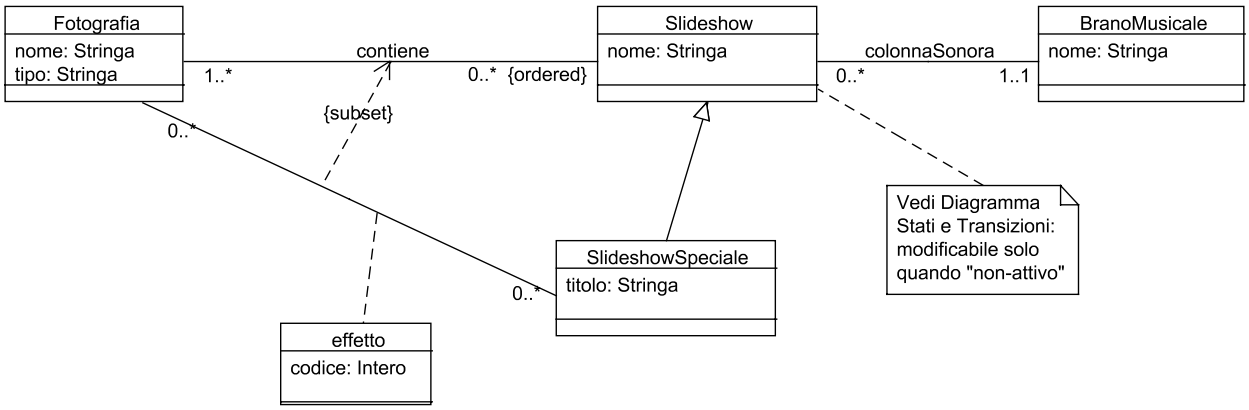


Diagramma degli stati e delle transizioni classe SlideShow

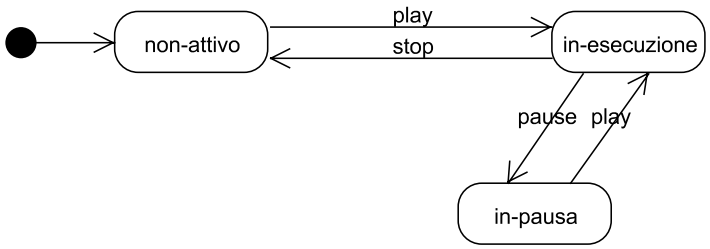
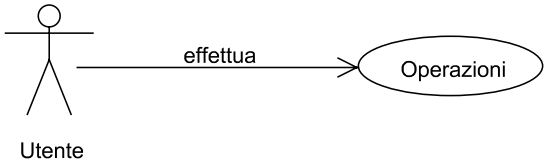


Diagramma degli use case



Specifica dello use case

InizioSpecificaUseCase Operazioni

EstraiSlideshowTipo *(s: Slideshow, t: Stringa): Slideshow*

pre: nessuna

post: *result* = $new\ Slideshow$ tale che

- $\langle result, f\rangle \in contiene$ sse $\langle s, f\rangle \in contiene$ e $f.tipo = t$
- $\langle result, m\rangle \in colonnaSonora$ sse $\langle s', m\rangle \in colonnaSonora$

numeroSlideshowSpeciali *(f: Fotografia): intero*

pre: nessuna

post: *result* = $|\{s \mid s \in SlideshowSpeciale \wedge \langle s, f\rangle \in contiene\}|$

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **estraiSlideshowTipo**:

```
int result = new Slideshow;
per ogni link l di tipo contiene in cui s è coinvolto
        se (l.fotografia.tipo = t)
            aggiungi new contiene(result,l.fotografia);
sia ll il link di tipo colonnaSonora in cui s e' coninvolto
aggiungi new colonnaSonora(result,ll.branoMusicale)
return result;
```

- Per l'operazione **numeroSlideshowSpeciali**:

```
result = 0;
per ogni link l di tipo contiene in cui f è coinvolto
  se (l.slideshow e' istanza di SlideshowSpeciale)
    result = result + 1;
return result;
```

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	*Slideshow*	SÌ[3]
	Fotografia	SÌ[2]
colonnaSonora	*Slideshow*	SÌ[3]
	BranoMusicale	NO
effetto	*Slideshow*	SÌ[1]
	Fotografia	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 le classi del collection framework di Java 1.5: `Set`, `HashSet` (per le associazioni non ordinate) e `Link`, `LinkedList` (per le associazioni ordinate).

Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
intero	`int`
Stringa	`String`
Insieme	`HashSet`
Lista	`LinkedList`

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**

	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 *Slideshow*, 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`
non-attivo Pausa	valore	1
in-esecuzione	valore	2
in-pausa	valore	3

API delle classi Java progettate

Omesse per brevità.

Fase di realizzazione

Considerazioni iniziali

Il compito richiedeva di realizzare solo quanto segue:

1. la classe UML *Slideshow*, che ha associato un diagramma degli stati e delle transizioni;
2. la classe UML *Fotografia*
3. l'associazione UML *contiene* con responsabilità doppia avente un attributo e con vincoli di molteplicità 0..* e 1..* (molteplicità minima diversa da zero);
4. il primo use case.

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

Decidiamo di realizzare le classi (funzioni get) senza condivisione di memoria (facendo uso di clone() quando necessario).

Struttura dei file e dei package

```
\---AppSlideshow
    |   BranoMusicale.java
    |   Fotografia.java
    |   AssociazioneContiene.java
    |   TipoLinkContiene.java
    |   TipoLinkEffetto.java
    |   EccezionePrecondizioni.java
    |   EccezioneMolteplicita.java
    |   EccezioneSubset.java
    |   Operazioni.java
    |
    +---Slideshow
    |       Slideshow.java
    |
    \---SlideshowSpeciale
            SlideshowSpeciale.java
```

La classe Java Slideshow

```
// File AppSlideshow/Slideshow/Slideshow.java
package AppSlideshow.Slideshow;
import AppSlideshow.*;
import java.util.*;

public class Slideshow {
  private String nome;
  private LinkedList<TipoLinkContiene> contiene;
  private BranoMusicale colonnaSonora;
  private static final int MOLT_MIN = 1;
  private static final int NON_ATTIVO = 1,
    IN_ESECUZIONE = 2, IN_PAUSA = 3;
  private int stato_corrente;
  public Slideshow(String n) {
    nome = n;
    colonnaSonora = null;
    contiene = new LinkedList<TipoLinkContiene>();
    stato_corrente = NON_ATTIVO;
  }
  public String getNome() { return nome; }
  public void setNome(String n) { nome = n; }
  public BranoMusicale getColonnaSonora() throws
    EccezioneMolteplicita {
    if (colonnaSonora == null)
```



```
      throw new EccezioneMolteplicita("Molteplicità minima violata");
    return colonnaSonora;
  }
  public void setColonnaSonora(BranoMusicale c) { colonnaSonora = c; }
  public boolean colonnaSonoraPresente() {
    return colonnaSonora == null;
  }
  public int quantiLinkContiene() {
    return contiene.size();
  }
  public void inserisciLinkContiene(AssociazioneContiene a) {
    if (a != null &&
        !contiene.contains(a.getLink())) contiene.add(a.getLink());
  }
  public void eliminaLinkContiene(AssociazioneContiene a) {
    if (a != null) contiene.remove(a.getLink());
  }
  public List<TipoLinkContiene> getLinkContiene() throws
    EccezioneMolteplicita {
    if (contiene.size() < MOLT_MIN)
      throw new EccezioneMolteplicita("Molteplicità minima violata");
    return (LinkedList<TipoLinkContiene>)contiene.clone();
  }
  public boolean estNonAttivo() {
    return stato_corrente == NON_ATTIVO;
  }
```

```
  public void play() {
    if (stato_corrente == NON_ATTIVO || stato_corrente == IN_PAUSA)
      stato_corrente = IN_ESECUZIONE;
  }
  public void stop() {
    if (stato_corrente == IN_ESECUZIONE)
      stato_corrente = NON_ATTIVO;
  }
  public void pause() {
    if (stato_corrente == IN_ESECUZIONE)
      stato_corrente = IN_PAUSA;
  }
  public String toString() {
    return nome;
  }
}
```

La classe Java Fotografia

```
// File AppSlideshow/Fotografia.java
package AppSlideshow;
import AppSlideshow.Slideshow.*;
import java.util.*;

public class Fotografia {
  private String nome;
  private String tipo;
  private HashSet<TipoLinkContiene> contiene;
  public Fotografia(String n, String t) {
    nome = n;
    tipo = t;
    contiene = new HashSet<TipoLinkContiene>();
  }
  public String getNome() { return nome; }
  public void setNome(String n) { nome = n; }
  public String getTipo() { return tipo; }
  public void setTipo(String t) { tipo = t; }
  public void inserisciLinkContiene(AssociazioneContiene a) {
    if (a != null) contiene.add(a.getLink());
  }
  public void eliminaLinkContiene(AssociazioneContiene a) {
    if (a != null) contiene.remove(a.getLink());
  }
```

```
  public Set<TipoLinkContiene> getLinkContiene() {
    return (HashSet<TipoLinkContiene>)contiene.clone();
  }
  public String toString() {
    return "<" + nome + ", " + tipo + ">";
  }
}
```

La classe Java AssociazioneContiene

```
// File AppSlideshow/AssociazioneContiene.java
package AppSlideshow;

public final class AssociazioneContiene {
  private AssociazioneContiene(TipoLinkContiene x) { link = x; }
  private TipoLinkContiene link;
  public TipoLinkContiene getLink() { return link; }
  public static void inserisci(TipoLinkContiene y) {
    if (y != null &&
        y.getSlideshow().estNonAttivo()) {
      AssociazioneContiene k = new AssociazioneContiene(y);
      k.link.getSlideshow().inserisciLinkContiene(k);
      k.link.getFotografia().inserisciLinkContiene(k);
    }
  }
  public static void elimina(TipoLinkContiene y) {
    if (y != null &&
        y.getSlideshow().estNonAttivo()) {
      AssociazioneContiene k = new AssociazioneContiene(y);
      k.link.getSlideshow().eliminaLinkContiene(k);
      k.link.getFotografia().eliminaLinkContiene(k);
    }
  }
}
```

La classe Java TipoLinkContiene

```
// File AppSlideshow/TipoLinkContiene.java
package AppSlideshow;
import AppSlideshow.Slideshow.*;

public class TipoLinkContiene {
  private final Slideshow loSlideshow;
  private final Fotografia laFotografia;
  public TipoLinkContiene(Slideshow q, Fotografia p)
    throws EccezionePrecondizioni {
    if (q == null || p == null)  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni
        ("Gli oggetti devono essere inizializzati");
    loSlideshow = q; laFotografia = p;
  }
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkContiene b = (TipoLinkContiene)o;
      return b.laFotografia == laFotografia &&
        b.loSlideshow == loSlideshow;
    }
    else return false;
  }
  public int hashCode() {
    return loSlideshow.hashCode() + laFotografia.hashCode();
```



```
  }
  public Slideshow getSlideshow() { return loSlideshow; }
  public Fotografia getFotografia() { return laFotografia; }
  public String toString() {
    return "<" + loSlideshow + ", " + laFotografia + ">";
  }
}
```

La classe Java SlideshowSpeciale

```
// File AppSlideshow/Speciale/SlideshowSpeciale.java
package AppSlideshow.SlideshowSpeciale;
import AppSlideshow.Slideshow.*;
import AppSlideshow.*;
import java.util.*;

public class SlideshowSpeciale extends Slideshow {
  private String titolo;
  private HashSet<TipoLinkEffetto> effetti;
  public SlideshowSpeciale(String n, String t) {
    super(n);
    titolo = t;
    effetti = new HashSet<TipoLinkEffetto>();
  }
  public String getTitolo() { return titolo; }
  public void setTitolo(String t) { titolo = t; }
  public void inserisciLinkEffetto(TipoLinkEffetto t) {
    if (t != null) effetti.add(t);
  }
  public void eliminaLinkEffetto(TipoLinkEffetto t) {
    if (t != null) effetti.remove(t);
  }
  public Set<TipoLinkEffetto> getLinkEffetto()
    throws EccezioneMolteplicita, EccezioneSubset {
```



```
    List<TipoLinkContiene> c = this.getLinkContiene(); //throws EccezioneMolteplicita
    Iterator<TipoLinkEffetto> it = effetti.iterator();
    while (it.hasNext()) {
      Fotografia f = it.next().getFotografia();
      if (!c.contains(new TipoLinkContiene(this,f)))
        throw new EccezioneSubset("effetti non è un subset di contiene");
    }
    return (HashSet<TipoLinkEffetto>)effetti.clone();
  }
}
```

La classe Java TipoLinkEffetto

```
// File AppSlideshow/TipoLinkEffetto.java
package AppSlideshow;
import AppSlideshow.Slideshow.*;

public class TipoLinkEffetto {
  private final Slideshow loSlideshow;
  private final Fotografia laFotografia;
  private final int codiceEffetto;
  public TipoLinkEffetto(Slideshow s, Fotografia f, int e)
    throws EccezionePrecondizioni {
    if (s == null || f == null)  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni
        ("Gli oggetti devono essere inizializzati");
    loSlideshow = s; laFotografia = f; codiceEffetto = e;
  }
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkEffetto b = (TipoLinkEffetto)o;
      return b.laFotografia == laFotografia &&
        b.loSlideshow == loSlideshow;
    }
    else return false;
  }
  public int hashCode() {
```



```
    return loSlideshow.hashCode() + laFotografia.hashCode();
  }
  public Slideshow getSlideshow() { return loSlideshow; }
  public Fotografia getFotografia() { return laFotografia; }
  public String toString() {
    return "<" + loSlideshow + ", " + laFotografia + ", " + codiceEffetto + ">";
  }
}
```

La classe Java BranoMusicale

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

public class BranoMusicale  {
  private String nome;
  public BranoMusicale(String n) {
    nome = n;
  }
  public String getNome() { return nome; }
  public void setNome(String n) { nome = n; }
  public String toString() {
    return nome;
  }
}
```


Realizzazione in Java degli use case

```
// File AppSlideshow/Operazioni.java
package AppSlideshow;
import AppSlideshow.Slideshow.*;
import AppSlideshow.SlideshowSpeciale.*;
import java.util.*;

public final class Operazioni {
  private Operazioni() { }
  public static Slideshow estraiSlideshowTipo(Slideshow s, String t) {
    Slideshow ss = new Slideshow(s.getNome());
    ss.setColonnaSonora(s.getColonnaSonora());
    Iterator<TipoLinkContiene> it = s.getLinkContiene().iterator();
    while(it.hasNext()) {
      TipoLinkContiene tlc = it.next();
      Fotografia f = tlc.getFotografia();
      if (f.getTipo() == t)
        AssociazioneContiene.inserisci(new TipoLinkContiene(ss,f));
    }
    return ss;
  }
  public static int numeroSlideshowSpeciali(Fotografia f) {
    int result = 0;
    Set<TipoLinkContiene> ins = f.getLinkContiene();
    Iterator<TipoLinkContiene> it = ins.iterator();
```

```
    while(it.hasNext()) {
      TipoLinkContiene t = it.next();
      if (t.getSlideshow() instanceof SlideshowSpeciale)
        result++;
    }
    return result;
  }
}
```