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 28 marzo 2006

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda la gestione di partite all' interno di un videogioco. Una partita è caratterizzata da un codice (una stringa), da un insieme non vuoto e ordinato di quadri giocati, e dai punti (un intero) guadagnati durante la partita in ciascun quadro. Un quadro è caratterizzato dal nome del file che contiene l'ambientazione. In un quadro, inoltre, possono essere presenti dei personaggi (dato un quadro non è di interesse conoscere quali personaggi sono presenti in esso). Ogni personaggio è caratterizzato dal nome del file che contiene la sua immagine. Alcuni quadri sono *dedicati* ad un particolare personaggio presente nel quadro stesso e sono caratterizzati dal nome di un file contenente un filmato.

Una partita, quando viene creata è inizialmente in pausa. Una partita in pausa può essere messa in gioco oppure può essere terminata. Quando è in gioco può essere messa di nuovo in pausa. La partita può essere modificata solo quando è in gioco.

Requisiti (cont.)

L'utente del gioco è interessato ad effettuare i seguenti controlli:

- dato un quadro q ed un intero n restituire il numero di partite in cui sono stati guadagnati più di n punti in q;
- dato un personaggio p restituire l'insieme delle partite che includono un quadro a lui dedicato.

Fase di analisi

Diagramma delle classi

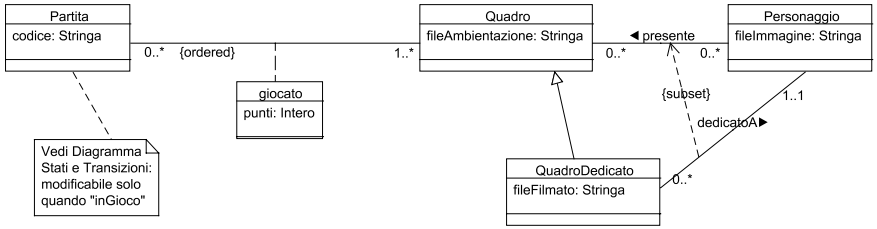


Diagramma degli stati e delle transizioni classe Partita

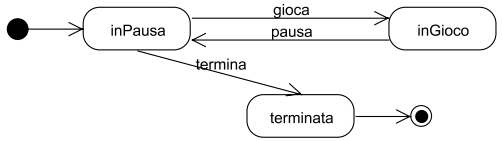
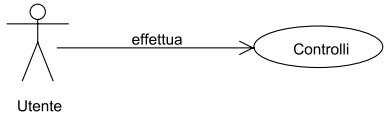


Diagramma degli use case



Specifica dello use case

InizioSpecificaUseCase Controlli

QuantePartite *(q: Quadro, n: intero): intero*
pre: nessuna
post: $result = |\{p \mid \langle p, q\rangle \in giocato \wedge giocato.punti(\langle p, q\rangle) > n\}|$

PartiteConPersonaggio *(p: Personaggio): Insieme(Partite)*
pre: nessuna
post: $result = \{r \mid \langle r, q\rangle \in giocato \wedge \langle q, p\rangle \in dedicato\}$

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **QuantePartite**:

```
int result = 0;
per ogni link l di tipo Giocato in cui q è coinvolto
        se (l.punti > n)
            result++;
return result;
```

- Per l'operazione **PartiteConPersonaggio**:

```
Insieme(Partite) result = new Insieme(Partite);  //restituisce un ins. vuoto
per ogni link l di tipo Dedicato in cui p è coinvolto
     per ogni link ll di tipo Giocato in cui ll.Quadro = l.QuadroDedicato
            result = result $\cup$ {ll.Partita}
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.
giocato	*Partita*	SÌ[3]
	Quadro	SÌ[2]
presente	*Personaggio*	SÌ[1]
	Quadro	NO
dedicatoA	*QuadroDedicato*	SÌ[3]
	Personaggio	SÌ[2]

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**
Partita	*codice*

	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 *Partita*, 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.

	Rappresentazione in Java	
	tipo var.	`int`
Stato	nome var.	`stato`
inPausa	valore	1
inGioco	valore	2
terminata	valore	3

API delle classi Java progettate

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

Fase di realizzazione

Considerazioni iniziali

Dalle fasi precedenti traiamo come conseguenza che dobbiamo realizzare:

1. la classe UML *Partita*, che ha associato un diagramma degli stati e delle transizioni;
2. la classe UML *Quadro*
3. l'associazione UML *giocato* con responsabilità doppia avente un attributo e con vincoli di molteplicità 0..* e 1..* (molteplicità minima diversa da zero);
4. uno 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

```
\---AppPlayer
    |   Partita.java
    |   Personaggio.java
    |   AssociazioneGiocato.java
    |   AssociazioneDedicatoA.java
    |   TipoLinkGiocato.java
    |   TipoLinkDedicatoA.java
    |   EccezionePrecondizioni.java
    |   EccezioneMolteplicita.java
    |   EccezioneSubset.java
    |   Controlli.java
    |
    +---Quadro
    |       Quadro.java
    |
    \---QuadroDedicato
            QuadroDedicato.java
```


La classe Java Partita

```
// File AppGioco/Partita.java
package AppGioco;
import AppGioco.Quadro.*;
import java.util.*;

public class Partita {
  private final String codice;
  private LinkedList<TipoLinkGiocato> giocato;
  public static final int MOLT_MIN = 1;
  private static final int IN_PAUSA = 1,
    IN_GIOCO = 2, TERMINATA = 3;
  private int stato_corrente;
  public Partita(String c) {
    codice = c;
    giocato = new LinkedList<TipoLinkGiocato>();
    stato_corrente = IN_PAUSA;
  }
  public String getCodice() { return codice; }
  public int quantiLinkGiocato() {
    return giocato.size();
  }
  public void inserisciLinkGiocato(AssociazioneGiocato a) {
    if (a != null &&
        !giocato.contains(a.getLink())) giocato.add(a.getLink());
```



```
  }
  public void eliminaLinkGiocato(AssociazioneGiocato a) {
    if (a != null) giocato.remove(a.getLink());
  }
  public List<TipoLinkGiocato> getLinkGiocato() throws
    EccezioneMolteplicita {
    if (giocato.size() < MOLT_MIN)
      throw new EccezioneMolteplicita("Molteplicità minima violata");
    return (LinkedList<TipoLinkGiocato>)giocato.clone();
  }
  public boolean estInGioco() {
    return stato_corrente == IN_GIOCO;
  }
  public void gioca() {
    if (stato_corrente == IN_PAUSA)
      stato_corrente = IN_GIOCO;
  }
  public void pausa() {
    if (stato_corrente == IN_GIOCO)
      stato_corrente = IN_PAUSA;
  }
  public void termina() {
    if (stato_corrente == IN_PAUSA)
      stato_corrente = TERMINATA;
  }
  public String toString() {
```

```
    return codice;
  }
}
```

La classe Java Quadro

```
// File AppGioco/Quadro/Quadro.java
package AppGioco.Quadro;
import AppGioco.*;
import java.util.*;

public class Quadro {
  protected String fileAmbientazione;
  protected HashSet<TipoLinkGiocato> giocato;
  public Quadro(String fa) {
    fileAmbientazione = fa;
    giocato = new HashSet<TipoLinkGiocato>();
  }
  public String getFileAmbientazione() { return fileAmbientazione; }
  public void setFileAmbientazione(String fa) { fileAmbientazione = fa; }
  public void inserisciLinkGiocato(AssociazioneGiocato a) {
    if (a != null) giocato.add(a.getLink());
  }
  public void eliminaLinkGiocato(AssociazioneGiocato a) {
    if (a != null) giocato.remove(a.getLink());
  }
  public Set<TipoLinkGiocato> getLinkGiocato() {
    return (HashSet<TipoLinkGiocato>)giocato.clone();
  }
}
```


La classe Java AssociazioneGiocato

```
// File AppGioco/AssociazioneGiocato.java
package AppGioco;

public final class AssociazioneGiocato {
  private AssociazioneGiocato(TipoLinkGiocato x) { link = x; }
  private TipoLinkGiocato link;
  public TipoLinkGiocato getLink() { return link; }
  public static void inserisci(TipoLinkGiocato y) {
    if (y != null &&
        y.getPartita().estInGioco()) { //NOTA!
      AssociazioneGiocato k = new AssociazioneGiocato(y);
      k.link.getQuadro().inserisciLinkGiocato(k);
      k.link.getPartita().inserisciLinkGiocato(k);
    }
  }
  public static void elimina(TipoLinkGiocato y) {
    if (y != null &&
        y.getPartita().estInGioco()) { //NOTA!
      AssociazioneGiocato k = new AssociazioneGiocato(y);
      k.link.getQuadro().eliminaLinkGiocato(k);
      k.link.getPartita().eliminaLinkGiocato(k);
    }
  }
}
```


La classe Java TipoLinkGiocato

```
// File AppGioco/TipoLinkGiocato.java
package AppGioco;
import AppGioco.Quadro.*;

public class TipoLinkGiocato {
  private final Quadro ilQuadro;
  private final Partita laPartita;
  private final int punti;
  public TipoLinkGiocato(Quadro q, Partita p, int n)
    throws EccezionePrecondizioni {
    if (q == null || p == null)  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni
        ("Gli oggetti devono essere inizializzati");
    ilQuadro = q; laPartita = p; punti =n;
  }
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkGiocato b = (TipoLinkGiocato)o;
      return b.laPartita == laPartita &&
        b.ilQuadro == ilQuadro;
    }
    else return false;
  }
  public int hashCode() {
```



```
    return ilQuadro.hashCode() + laPartita.hashCode();
  }
  public Quadro getQuadro() { return ilQuadro; }
  public Partita getPartita() { return laPartita; }
  public int getPunti() { return punti; }
  public String toString() {
    return "(<" + ilQuadro + ", " + laPartita + ">, " + punti + ")";
  }
}
```

La classe Java QuadroDedicato

```
// File AppGioco/QuadroDedicato/QuadroDedicato.java
package AppGioco.QuadroDedicato;
import AppGioco.Quadro.*;
import AppGioco.*;
import java.util.*;

public class QuadroDedicato extends Quadro {
  protected String fileFilmato;
  protected TipoLinkDedicatoA dedicato;
  public QuadroDedicato(String fa, String ff) {
    super(fa);
    fileFilmato = ff;
    dedicato = null;
  }
  public String getFileFilmato() { return fileFilmato; }
  public void setFileFilmato(String ff) { fileFilmato = ff; }
  public boolean estPresenteLinkDedicatoA() {
    return dedicato != null;
  }
  public void inserisciLinkDedicatoA(AssociazioneDedicatoA a) {
    if (a != null) dedicato = a.getLink();
  }
  public void eliminaLinkDedicatoA(AssociazioneDedicatoA a) {
    if (a != null) dedicato = null;
```



```
  }
  public TipoLinkDedicatoA getLinkDedicatoA()
    throws EccezioneMolteplicita, EccezioneSubset {
    if (!estPresenteLinkDedicatoA())
      throw new EccezioneMolteplicita("Violata molteplicità minima");
    if (!dedicato.getPersonaggio().getLinkPresente().contains(this))
      throw new EccezioneSubset("dedicatoA non è un subset di presente");
    return dedicato;
  }
}
```

La classe Java Personaggio

```
// File AppGioco/Personaggio/Personaggio.java
package AppGioco;
import AppGioco.Quadro.*;
import AppGioco.QuadroDedicato.*;
import java.util.*;

public class Personaggio  {
  private String fileImmagine;
  private HashSet<Quadro> presente;
  private HashSet<TipoLinkDedicatoA> dedicato;
  public Personaggio(String fi) {
    fileImmagine = fi;
    presente = new HashSet<Quadro>();
    dedicato = new HashSet<TipoLinkDedicatoA>();
  }
  public String getFileImmagine() { return fileImmagine; }
  public void setFileImmagine(String fi) { fileImmagine = fi; }
  public void inserisciLinkPresente(Quadro q) {
    if (q != null) presente.add(q);
  }
  public void eliminaLinkPresente(Quadro q) {
    if (q != null) presente.remove(q);
  }
  public Set<Quadro> getLinkPresente() {
```



```
    return (HashSet<Quadro>)presente.clone();
  }
  public void inserisciLinkDedicatoA(AssociazioneDedicatoA a) {
    if (a != null) dedicato.add(a.getLink());
  }
  public void eliminaLinkDedicatoA(AssociazioneDedicatoA a) {
    if (a != null) dedicato.remove(a.getLink());
  }
  public Set<TipoLinkDedicatoA> getLinkDedicatoA() throws EccezioneSubset {
    Iterator<TipoLinkDedicatoA> it = dedicato.iterator();
    while(it.hasNext()) {
      QuadroDedicato q = it.next().getQuadroDedicato();
      if (!presente.contains(q))
        throw new EccezioneSubset("dedicatoA non è un subset di presente");
    }
    return (HashSet<TipoLinkDedicatoA>)dedicato.clone();
  }
}
```

La classe Java TipoLinkDedicatoA

```
// File AppGioco/TipoLinkDedicatoA.java
package AppGioco;
import AppGioco.QuadroDedicato.*;

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



```
  }
  public QuadroDedicato getQuadroDedicato() { return ilQuadroDedicato; }
  public Personaggio getPersonaggio() { return ilPersonaggio; }
  public String toString() {
    return "<" + ilQuadroDedicato + ", " + ilPersonaggio + ">";
  }
}
```

La classe Java AssociazioneDedicatoA

```
// File AppGioco/AssociazioneDedicatoA.java
package AppGioco;

public final class AssociazioneDedicatoA {
  private AssociazioneDedicatoA(TipoLinkDedicatoA x) { link = x; }
  private TipoLinkDedicatoA link;
  public TipoLinkDedicatoA getLink() { return link; }
  public static void inserisci(TipoLinkDedicatoA y) {
    if (y != null &&
        !y.getQuadroDedicato().estPresenteLinkDedicatoA()) {
      AssociazioneDedicatoA k = new AssociazioneDedicatoA(y);
      k.link.getQuadroDedicato().inserisciLinkDedicatoA(k);
      k.link.getPersonaggio().inserisciLinkDedicatoA(k);
    }
  }
  public static void elimina(TipoLinkDedicatoA y) {
    if (y != null) {
      AssociazioneDedicatoA k = new AssociazioneDedicatoA(y);
      k.link.getQuadroDedicato().eliminaLinkDedicatoA(k);
      k.link.getPersonaggio().eliminaLinkDedicatoA(k);
    }
  }
}
```


Realizzazione in Java degli use case

```
// File AppGioco/Controlli.java
package AppGioco;
import java.util.*;
import AppGioco.Quadro.*;
import AppGioco.QuadroDedicato.*;

public final class Controlli {
  private Controlli() { }
  public static int quantePartite(Quadro q, int n) {
    int result = 0;
    Set<TipoLinkGiocato> ins = q.getLinkGiocato();
    Iterator<TipoLinkGiocato> it = ins.iterator();
    while(it.hasNext()) {
      TipoLinkGiocato t = it.next();
      if (t.getPunti() > n)
        result++;
    }
    return result;
  }
  public static Set<Partita> partiteConPersonaggio(Personaggio p) {
    HashSet<Partita> result = new HashSet<Partita>();
    Iterator<TipoLinkDedicatoA> it = p.getLinkDedicatoA().iterator();
    while(it.hasNext()) {
      TipoLinkDedicatoA t = it.next();
```

```
      Iterator<TipoLinkGiocato> itt = t.getQuadroDedicato().getLinkGiocato().iterator();
      while (itt.hasNext())
        result.add(itt.next().getPartita());
    }
    return result;
  }
}
```