SAPIENZA Università di Roma
Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE I"
(Laurea in Ingegneria Informatica)
Proff. Giuseppe De Giacomo e Domenico Lembo
Canali A-L & M-Z
A.A. 2007-08

Compito d'esame del 3 aprile 2008

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda una parte del sistema di gestione di un asilo per il corrente anno di iscrizione. Ogni classe è caratterizzata da un nome (una stringa), dai bambini ad essa assegnati e dalle maestre che vi insegnano. In una classe insegnano 1 o 2 maestre. Ogni bambino ha un nome e un'età (compresa tra 0 e 5 anni) ed è assegnato ad esattamente una classe. Ogni maestra ha un nome ed una anzianità di servizio (un intero). Alcune classi sono classi di scolarizzazione e ad esse vengono assegnati almeno 5 bambini non-scolarizzati. Dei bambini non-scolarizzati interessa sapere se portano ancora il pannolino (un booleano). Alle classi di scolarizzazione sono assegnate esattamente 2 maestre.

Requisiti (cont.)

Un bambino è inizialmente accolto dalla struttura, dopo qualche giorno passa alla fase di inserimento. Completata questa fase il bambino viene considerato inserito. Se però si assenta per un periodo lungo allora torna alla fase di inserimento.

Il coordinatore didattico è interessato ad effettuare diversi controlli sulle classi, in particolare:

- dato un insieme di classi s, restituire il sottoinsieme formato dalle classi problematiche di s: dove una classe è problematica se è una classe di scolarizzazione tale che tutti i bambini assegnati ad essa sono non-scolarizzati;
- data una classe c, restituire l'età media dei bambini ad essa assegnati.

Requisiti (cont.)

Domanda 1. Basandosi sui requisiti riportati sopra, svolgere 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. Svolgere 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. Svolgere la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate. È obbligatorio realizzare in Java solo i seguenti aspetti dello schema concettuale:

- le classe `Classe` e `Bambino` e tutte le associazioni che le legano;
- il primo use case.

Fase di analisi

Diagramma delle classi

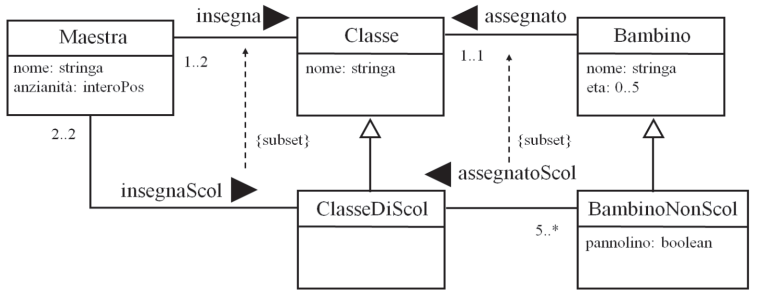

Diagramma degli stati e delle transizioni classe Bambino

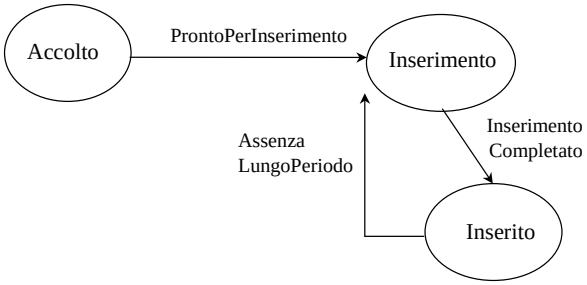

Diagramma degli use case

Specifica dello use case

InizioSpecificaUseCase **Controlli**

ClassiProblematiche *(I: Insieme(Classe)): Insieme(Classe)*

pre: *true*

post: Definiamo l'insieme:

$$C \doteq \{c \in I \mid c \in ClasseDiScol \wedge \nexists b \in Bambino \text{ tale che } b \notin BambinoNonScol \wedge (b,c) \in assegnato\}$$

result $= C$

...

Specifica dello use case (cont.)

...

EtaMedia *(c: Classe): reale*

Definiamo l'insieme

$$C \doteq \{b \in Bambino \mid (b,c) \in assegnato\}$$

pre: $C \neq \emptyset$

post: *result* $= (\sum_{b \in C} b.eta \backslash card(C)) * 100$

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **ClassiProblematiche**:

```
Boolean classProb;
Insieme(Classe) result = ∅
per ogni classe c appartenente ad I
    classProb = true;
    per ogni link l di tipo assegnato in cui c è coinvolta
        se !(l.Bambino.class = BambinoNonScol)
            classeProb = false;
    se (classeProb = false)
       result = result ∪{c};
return result;
```

- Per l'operazione **EtaMedia**:

```
int sum = 0;
int card = 0;
per ogni link l di tipo assegnato in cui c è coinvolta
    sum = sum + l.Bambino.eta;
    card++;
return (sum\card)*100;
```

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.
insegna	*Maestra*	NO
	Classe	SÌ[3]
insegnaScol	*Maestra*	NO
	ClasseDiScol	SI[3]
assegnato	*Bambino*	SÌ[3]
	Classe	SÌ[2]
assegnatoScol	*BambinoNonScol*	NO
	ClasseScol	SÌ[3]

Strutture di dati

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

- dei vincoli di molteplicità 0..* delle associazioni,
- dei paramentri in input alle operazioni e delle variabili locali necessarie per i vari algoritmi.

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
interoPos	`int`
stringa	`String`
0..5	`int`
Boolean	`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**
Maestra	*nome*
Bambino	*nome*

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

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 *Bambino*, 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`
Accolto	valore	1
Inserimento	valore	2
Inserito	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 UML *Classe*;
2. la classe UML *Bambino*
3. l'associazione UML *assegnato* con responsabilità doppia e con vincoli di molteplicità 1..1 (molteplicità massima finita e molteplicità minima diversa da zero) e 0..*;
4. il primo use case.

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

Struttura dei file e dei package

```
AppAsilo
  |   ManagerAssegnato.java
  |   Controlli.java
  |   Maestra.java
  |   EccezioneCardMax.java
  |   EccezioneCardMin.java
  |   EccezionePrecondizioni.java
  |   TestAsilo.java
  |   TipoLinkAssegnato.java
  |
  +---Bambino
  |       Bambino.java
  |
  +---BambinoNonScol
  |       BambinoNonScol.java
  |
  +---Classe
  |       Classe.java
  |
  \---ClasseDiScol
        ClasseDiScol.java
```


La classe Java Classe

```
// File AppAsilo/Classe/Classe.java
package AppAsilo.Classe;

import AppAsilo.Classe.*;
import AppAsilo.*;
import AppAsilo.ClasseDiScol.*;
import java.util.*;

public class Classe  {
    protected String nome;
    protected HashSet<Maestra> insieme_link_m;
    protected HashSet<TipoLinkAssegnato> insieme_link_b;
    private static final int MAX_LINK_INSEGNA = 2;
    private static final int MIN_LINK_INSEGNA = 1;
    public Classe(String n) {
        nome = n;
        insieme_link_m = new HashSet<Maestra>();
        insieme_link_b = new HashSet<TipoLinkAssegnato>();
    }
    public String getNome() { return nome; }

    public void inserisciLinkInsegna(Maestra m) {
      if (m != null) insieme_link_m.add(m);
    }
```



```
    public void eliminaLinkInsegna(Maestra m) {
      if (m != null) insieme_link_m.remove(m);
    }
    public Set<Maestra> getLinkInsegna() throws EccezioneMoltMin,EccezioneMoltMax {
      if (insieme_link_m.size() > MAX_LINK_INSEGNA)
          throw new EccezioneMoltMax("Molteplicita' massima violata");
      if (insieme_link_m.size() < MIN_LINK_INSEGNA)
          throw new EccezioneMoltMin("Molteplicita' minima violata");
      return (HashSet<Maestra>)insieme_link_m.clone();
    }
    public int quantiInsegna() {
        return insieme_link_m.size();
    }
    public void inserisciLinkAssegnato(TipoLinkAssegnato t) {
      if (t != null && t.getClasse()==this)
          ManagerAssegnato.inserisci(t);
    }
    public void eliminaLinkAssegnato(TipoLinkAssegnato t) {
     if (t != null && t.getClasse()==this)
          ManagerAssegnato.elimina(t);
    }
    public Set<TipoLinkAssegnato> getLinkAssegnato() {
      return (HashSet<TipoLinkAssegnato>)insieme_link_b.clone();
    }
    public void inserisciPerManagerAssegnato(ManagerAssegnato a) {
        if (a != null) insieme_link_b.add(a.getLink());
```

```
    }
    public void eliminaPerManagerAssegnato(ManagerAssegnato a) {
        if (a != null) insieme_link_b.remove(a.getLink());
    }
}
```

La classe Java ClasseDiScol

```
// File AppAsilo/ClasseDiScol/ClasseDiScol.java
package AppAsilo.ClasseDiScol;

import AppAsilo.*;
import AppAsilo.Classe.*;
import AppAsilo.Bambino.*;
import AppAsilo.BambinoNonScol.*;
import java.util.*;

public class ClasseDiScol extends Classe {
    //Utiliziamo questa classe per vedere due strategie per gestire
    //i vincoli di subset.
    //
    //La prima strategia consiste nel non permettere l'inserimento di un
    //link nella associazione contenuta se un analogo link non e' gia' presente
    //nella associazione contenente.
    //Illustriamo questa startegia sulla associazione AssegnatoNonScol
    //
    //La seconda strategia consiste nel verificare il contenimento solo
    //al momento di restituire i link della associazione contenuta.
    //Illustriamo questa strategia sulla associazione InsegnaScol
    //
    protected HashSet<BambinoNonScol> insieme_link_a;
    protected HashSet<Maestra> insieme_link_b;
```



```
    private static final int MAX_LINK_INSEGNA_SCOL = 2;
    private static final int MIN_LINK_INSEGNA_SCOL = 2;
    private static final int MIN_LINK_ASSEGNATO_SCOL = 5;
    public ClasseDiScol(String n) {
        super(n);
        insieme_link_a = new HashSet<BambinoNonScol>();
        insieme_link_b = new HashSet<Maestra>();
    }
    public int quantiBambiniNonScol() {
        return insieme_link_a.size();
    }
    public void inserisciLinkAssegnatoNonScol(BambinoNonScol b) throws EccezioneSubset{
      //Prima strategia gestione vincoli di subset
      if (b != null)
        if (!getLinkAssegnato().contains(new TipoLinkAssegnato(b,this))) //Nota!
        throw new EccezioneSubset("Il link non e' presente in Assegnato:
                                   inserimento impossibile" );
      insieme_link_a.add(b);
    }
    public void eliminaLinkAssegnatoNonScol(BambinoNonScol b) {
      if (b != null) insieme_link_a.remove(b);
    }
    public Set<BambinoNonScol> getLinkAssegnatoScol() throws EccezioneMoltMin{
      if (insieme_link_a.size() < MIN_LINK_ASSEGNATO_SCOL)
          throw new EccezioneMoltMin("Molteplicita' minima violata");
      return (HashSet<BambinoNonScol>)insieme_link_a.clone();
```

```
    }
    public void inserisciLinkInsegnaScol(Maestra m) {
      if (m != null) insieme_link_b.add(m);
    }
    public void eliminaLinkInsegnaScol(Maestra m) {
      if (m != null) insieme_link_b.remove(m);
    }
    public Set<Maestra> getLinkInsegnaScol() throws
           EccezioneMoltMin,EccezioneMoltMax,EccezioneSubset{
      //Seconda strategia gestione vincoli di subset
      if (insieme_link_b.size() > MAX_LINK_INSEGNA_SCOL)
          throw new EccezioneMoltMax("Molteplicita' massima violata");
      if (insieme_link_b.size() < MIN_LINK_INSEGNA_SCOL)
          throw new EccezioneMoltMin("Molteplicita' minima violata");
      if (!getLinkInsegna().containsAll(insieme_link_b)) //Nota!
      throw new EccezioneSubset("Vincolo di subset violato");
      return (HashSet<Maestra>)insieme_link_b.clone();
    }
    public int quantiInsegnaScol() {
      return insieme_link_b.size();
    }


}
```

La classe Java Bambino

```
// File AppAsilo/Bambino/Bambino.java
package AppAsilo.Bambino;

import AppAsilo.Bambino.*;
import AppAsilo.*;
import AppAsilo.BambinoNonScol.*;
import java.util.*;

public class Bambino  {
    private final static int accolto = 1;
    private final static int inserimento = 2;
    private final static int inserito = 3;
    private int corrente = 1;
    private static final int MIN_LINK_ASSEGNATO = 1;
    protected String nome;
    protected int eta;
    protected TipoLinkAssegnato link;
    public Bambino(String n, int e) throws EccezionePrecondizioni {
        if ((e < 0) || (e > 5))
            throw new EccezionePrecondizioni("Eta' non corretta");
        eta = e;
        nome = n;
        link = null;
    }
```



```
    public String getNome() { return nome; }
    public int getEta() { return eta; }
    public void setEta(int e) throws EccezionePrecondizioni {
        if ((e < 0) || (e > 5))
            throw new EccezionePrecondizioni("Eta' non corretta");
        eta = e;
    }
    public void prontoInserimento() {
        if (corrente == accolto)
            corrente = inserimento;
    }
    public void completatoInserimento() {
        if (corrente == inserimento)
            corrente = inserito;
    }
    public void assenzaLungoPeriodo() {
        if (corrente == inserito)
            corrente = inserimento;
    }
    public int quantiAssegnato() {
        if (link == null)
            return 0;
        else return 1;
    }
    public void inserisciLinkAssegnato(TipoLinkAssegnato t) {
        if (t != null && t.getBambino()==this)
```

```
            ManagerAssegnato.inserisci(t);
    }
    public void eliminaLinkAssegnato(TipoLinkAssegnato t) {
        if (t != null && t.getBambino()==this)
            ManagerAssegnato.elimina(t);

    }
    public TipoLinkAssegnato getLinkAssegnato() throws EccezioneMoltMin {
        if (link == null)
            throw new EccezioneMoltMin("Molteplicita' minima violata");
        else
            return link;
    }
    public void inserisciPerManagerAssegnato(ManagerAssegnato a) {
        if (a != null) link = a.getLink();
    }
    public void eliminaPerManagerAssegnato(ManagerAssegnato a) {
        if (a != null) link = null;
    }
}
```

La classe Java BambinoNonScol

```
// File AppAsilo/BambinoNonScol/BambinoNonScol.java
package AppAsilo.BambinoNonScol;
import AppAsilo.Bambino.*;
import AppAsilo.*;
import java.util.*;

public class BambinoNonScol extends Bambino {
    protected boolean pannolino;
    public BambinoNonScol(String n, int e, boolean p) {
        super(n,e);
        pannolino = p;
    }
    public boolean getPannolino() { return pannolino; }
    public void setPannolino(boolean b) { pannolino = b; }
}
```

La classe Java Maestra

```
// File AppAsilo/Maestra.java
package AppAsilo;

import AppAsilo.Classe.*;
import AppAsilo.*;
import AppAsilo.ClasseDiScol.*;
import java.util.*;

public class Maestra  {
    private String nome;
    private int anzianita;
    public Maestra(String n, int a) {
        nome = n;
        anzianita = a;
    }
    public String getNome() { return nome; }
    public int getAnzianita() { return anzianita; }
    public void setAnzianita(int a) throws EccezionePrecondizioni {
        if (a<0)
            throw new EccezionePrecondizioni("Anzianita' non corretta");
        anzianita = a;
    }
}
```


La classe Java ManagerAssegnato

```
// File AppAsilo/ManagerAssegnato.java
package AppAsilo;

public final class ManagerAssegnato {
    private ManagerAssegnato(TipoLinkAssegnato x) { link = x; }
    private TipoLinkAssegnato link;
    public TipoLinkAssegnato getLink() { return link; }
    public static void inserisci(TipoLinkAssegnato y) {
        if (y != null &&
            y.getBambino().quantiAssegnato() == 0) {
            ManagerAssegnato k = new ManagerAssegnato(y);
            y.getBambino().inserisciPerManagerAssegnato(k);
            y.getClasse().inserisciPerManagerAssegnato(k);
        }
    }
    public static void elimina(TipoLinkAssegnato y) throws EccezioneMoltMin{
        if (y != null && y.getBambino().getLinkAssegnato().equals(y)) {
            ManagerAssegnato k = new ManagerAssegnato(y);
            y.getBambino().eliminaPerManagerAssegnato(k);
            y.getClasse().eliminaPerManagerAssegnato(k);
        }
    }
}
```


La classe Java TipoLinkAssegnato

```
// File AppAsilo/TipoLinkAssegnato.java
package AppAsilo;

import AppAsilo.Classe.*;
import AppAsilo.Bambino.*;
import AppAsilo.*;

public class TipoLinkAssegnato {
    private final Bambino ilBambino;
    private final Classe laClasse;
    public TipoLinkAssegnato(Bambino b, Classe c)
        throws EccezionePrecondizioni {
        if (b == null || c == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        ilBambino = b;
        laClasse = c;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkAssegnato a = (TipoLinkAssegnato) o;
            return a.ilBambino == ilBambino &&
                a.laClasse == laClasse;
        }
```



```
        else return false;
    }
    public int hashCode() {
        return ilBambino.hashCode() + laClasse.hashCode();
    }
    public Bambino getBambino() { return ilBambino; }
    public Classe getClasse() { return laClasse; }
    public String toString() {
        return "<" + ilBambino + ", " + laClasse + ">";
    }
}
```

Realizzazione in Java dello use case

```
// File AppAsilo/Controlli.java
package AppAsilo;
import java.util.*;
import AppAsilo.Classe.*;
import AppAsilo.Bambino.*;
import AppAsilo.ClasseDiScol.*;
import AppAsilo.BambinoNonScol.*;

public final class Controlli {
    private Controlli() { }
    public static Set<ClasseDiScol> ClassiProblematiche(HashSet<Classe> I)
           throws EccezioneMoltMin, EccezioneMoltMax {
        boolean classeProb;
        HashSet<ClasseDiScol> res = new HashSet<ClasseDiScol>();
        Iterator<Classe> it = I.iterator();
        while (it.hasNext()) {
           Classe c = it.next();
           if (c.getClass().equals(ClasseDiScol.class)){
             Set<TipoLinkAssegnato> r = c.getLinkAssegnato();
             classeProb = true;
             Iterator<TipoLinkAssegnato> it2 = r.iterator();
             while (it2.hasNext() & classeProb)
               if (!(it2.next().getBambino().getClass().equals(BambinoNonScol.class)))
                  classeProb = false;
```



```
             if (classeProb) res.add((ClasseDiScol) c);
           }
        }
        return res;
    }
    //Il secondo use case e' lasciato per esercizio.
}
```

Realizzazione in Java delle classi per eccezioni

```
// File AppAsilo/EccezioneMoltMin.java
package AppAsilo;

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

// File AppAsilo/EccezioneMoltMax.java
package AppAsilo;
public class EccezioneMoltMax extends Exception {
    private String messaggio;
    public EccezioneMoltMax(String m) {
        messaggio = m;
    }
    public String toString() {
        return messaggio;
    }
}
```



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

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;
  }
}

// File AppAsilo/EccezioneSubset.java
package AppAsilo;

public class EccezioneSubset extends RuntimeException {
  private String messaggio;
  public EccezioneSubset(String m) {
    messaggio = m;
  }
  public EccezioneSubset() {
    messaggio = "Si e' verificata una violazione del vincolo di subset";
```

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