Sapienza - Università di Roma
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 13 settembre 2007

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda una parte di un sistema di gestione di informazioni su aziende. Ogni azienda è caratterizzata da un nome (una stringa), da una descrizione testuale (una stringa) e dalle sedi in cui essa è presente con nome della località (una stringa) e indirizzo (una stringa). Le aziende sono suddivise in aziende pubbliche e aziende private. Delle prime interessa l'ente che le gestisce, con codice (una stringa) e nome (una stringa). Delle seconde, invece, interessa il capitale sociale (un reale). Di ogni azienda (sia essa pubblica o privata) interessa inoltre l'eventuale azienda controllante.

Requisiti (cont.)

Il fruitore dell'applicazione è interessato ad effettuare diversi controlli, in particolare:

- data una azienda a, verificare se essa è controllata direttamente o indirettamente (cioè attraverso altre aziende) da se stessa.
- data una azienda a, restituire l'insieme delle aziende private che essa controlla.

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 classe `Azienda` e tutte le associazioni in cui essa è coinvolta;
- il primo use case.

Fase di analisi

Diagramma delle classi

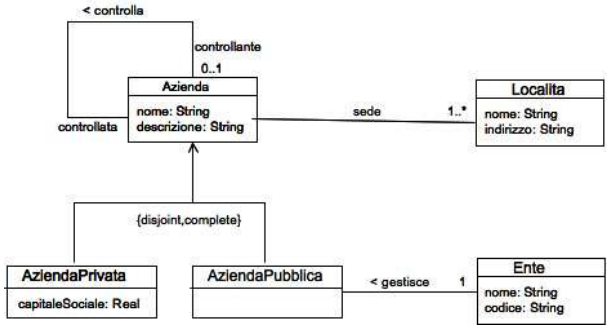
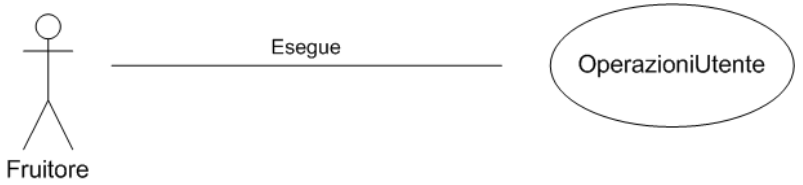


Diagramma degli use case



Specifica dello use case

InizioSpecificaUseCase **OperazioniUtente**

controllaSeStessa *(a: Azienda): boolean*

pre: `true`

post:

Assumiamo che la prima componente della relazione $controlla$ rivesta il ruolo di *controllante*.

Definiamo ricorsivamente l'insieme $A \subseteq Azienda$ delle aziende che controllano a, direttamente o indirettamente:

1. (Passo base) se $\langle a_1, a\rangle \in controlla$ allora $a_1 \in A$;
2. (Passo ricorsivo) se a_1 è t.c. $\exists a_2 \in A \wedge \langle a_1, a_2\rangle \in controlla$ allora $a_1 \in A$;

result=$true$ se e solo se $a \in A$.

...

Specifica dello use case (cont.)

...

azPrivateControllateDa *(a: Azienda): Insieme<AziendaPrivata>*

pre: `true`

post: *result* $= \{ap \mid \langle a, ap\rangle \in controlla \wedge ap \in AziendaPrivata\}$

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **controllaSeStessa***(a: Azienda): boolean*

```
//Usiamo while. In alternativa potevamo usare la ricorsione.

Azienda azCorrente = a.controllante;
while(azCorrente <> null){
      if (azCorrente == a)
            return true;
      azCorrente = azCorrente.controllante;
}
return false;
```

- ...

- ...

- Per l'operazione
  azPrivateControllateDa(*a: Azienda*): *Insieme<AziendaPrivata>*

```
result = new Insieme<AziendaPrivata>;
per ogni link l di tipo controlla in cui a  e' controllante{
     if (l.controllata instanceof AziendaPrivata)
          result.add(l.controllata);
}
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.
controlla	*controllata*	Sì[2,3]
	controllante	Sì[2]
sede	*Azienda*	Sì[3]
	Localita	NO
gestisce	*AziendaPubblica*	Sì[3]
	Ente	NO

Strutture di dati

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

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

- delle variabili necessarie per 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
Real	`float`
String	`String`
boolean	`boolean`
Insieme	`HashSet`

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.

Fase di realizzazione

Considerazioni iniziali

La traccia ci richiede di realizzare:

1. La classe Azienda.
2. l'associazione UML *controlla* con responsabilità doppia e con vincoli di molteplicità 0..1 (molteplicità massima diversa da zero) e 0..*;
3. l'associazione UML *sede* con responsabilità singola e con vincoli di molteplicità 1..* (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

```
+---AppAzienda
   |   TipoLinkControlla.java
   |   AssociazioneControlla.java
   |   OperazioniUtente.java
   |   EccezioneMolteplicita.java
   |   EccezionePrecondizioni.java
   |
   +---Azienda
   |       Azienda.java
   |
   +---AziendaPubblica
   |       AziendaPubblica.java
   |
   +---AziendaPrivata
   |       AziendaPrivata.java
   |
   +---Localita
   |       Localita.java
   |
   \---Ente
           Ente.java
```

La classe Java Azienda

```
// File AppAzienda/Azienda/Azienda.java
package AppAzienda.Azienda;
import AppAzienda.*;
import AppAzienda.Localita.*;
import java.util.*;

public abstract class Azienda{
  private final int MOLT_MIN_SEDI=1;

  private String nome;
  private String descrizione;
  private int stato;

  private HashSet<Localita> sedi;
  // Il nome del campo corrisponde al ruolo rivestito dalla classe nel link:
  private HashSet<TipoLinkControlla> controllante;
  private TipoLinkControlla controllata;

  protected Azienda(String nome, String descrizione){
    this.nome = nome;
    this.descrizione = descrizione;
    controllante = new HashSet<TipoLinkControlla>();
    controllata = null;
    sedi = new HashSet<Localita>();
```



```
  }

  public String getNome(){
    return nome;
  }

  public String getDescrizione(){
    return descrizione;
  }

  public int quanteSedi() {
    return sedi.size();
  }

  public Set<Localita> getSedi() throws EccezioneMolteplicita{
    if (sedi.size() < MOLT_MIN_SEDI)
        throw new EccezioneMolteplicita("Molteplicita' minima violata");
    return (HashSet<Localita>) sedi.clone();
  }

  public void inserisciSede(Localita l){
    if (l != null)
      sedi.add(l);
  }

  public void eliminaSede(Localita l){
    if (l != null)
      sedi.remove(l);
  }

  public TipoLinkControlla getLinkControllata(){
    return controllata;
  }

  public void inserisciLinkControllata(AssociazioneControlla a){
    if (a != null)
      controllata = a.getLink();
  }

 public void eliminaLinkControllata(AssociazioneControlla a){
    // l'argomento a e' dovuto alla doppia responsabilita'!
    if( a!= null)
      controllata = null;
  }

  public Set<TipoLinkControlla> getLinkControllante(){
    return (HashSet<TipoLinkControlla>) controllante.clone();
  }

  public void inserisciLinkControllante(AssociazioneControlla a){
    if (a != null){
      controllante.add(a.getLink());
    }
  }

  public void eliminaLinkControllante(AssociazioneControlla a){
    if( a!= null)
  controllante.remove(a.getLink());
  }
}
```

La classe Java AziendaPrivata

```
// File AppAzienda/AziendaPrivata/AziendaPrivata.java
package AppAzienda.AziendaPrivata;
import AppAzienda.*;
import AppAzienda.Azienda.*;
import java.util.*;

public class AziendaPrivata extends Azienda{
  private double capitaleSociale;

  public AziendaPrivata(String nome, String descrizione, double capitaleSociale){
    super(nome, descrizione);
    this.capitaleSociale = capitaleSociale;
  }

  public double getCapitaleSociale() {
    return capitaleSociale;
  }
}
```


La classe Java AziendaPubblica

```
// File AppAzienda/AziendaPubblica/AziendaPubblica.java
package AppAzienda.AziendaPubblica;
import AppAzienda.*;
import java.util.*;

public class AziendaPubblica extends Azienda{
  private Ente gestore;

  public AziendaPubblica(String nome, String descrizione){
    super(nome, descrizione):
  }

  public void setGestore(Ente e){
    if (e != null)
      gestore = e;
  }

  public Ente getEnte() throws EccezioneMolteplicita{
    if (gestore == null)
      throw new EccezioneMolteplicita("Molteplicita min/max violata");
    return gestore;
  }
}
```


La classe Java Localita

```
// File AppAzienda/Localita/Localita.java
package AppAzienda.Localita;
import java.util.*;

public class Localita {
  private String nome;
  private String indirizzo;

  public Localita(String nome, String indirizzo){
    this.nome = nome;
    this.indirizzo=indirizzo;
  }

  public String getNome(){
    return nome;
  }

  public String getIndirizzo(){
    return indirizzo;
  }
}
```


La classe Java Ente

```
// File AppAzienda/Ente/Ente.java
package AppAzienda.Ente;
import java.util.*;

public class Ente {
  private String nome;
  private String codice;

  public Ente(String nome, String codice){
    this.nome = nome;
    this.codice=codice;
  }

  public String getNome(){
    return nome;
  }

  public String getCodice(){
    return codice;
  }
}
```

La classe Java TipoLinkControlla

```
// File AppAzienda/TipoLinkControlla.java
package AppAzienda;
import AppAzienda.Azienda.*;
import java.util.*;

public class TipoLinkControlla{
  private final Azienda controllante;
  private final Azienda controllata;

  public TipoLinkControlla(Azienda controllante, Azienda controllata)
                                      throws EccezionePrecondizioni {
    if (controllante == null || controllata == null)  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni("Gli oggetti devono essere inizializzati");
      this.controllante = controllante;
      this.controllata = controllata;
  }

  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkControlla l = (TipoLinkControlla) o;
      return l.controllante == controllante && l.controllata == controllata;
    }
    else
      return false;
```



```
  }

  public int hashCode() {
    return  controllante.hashCode() + controllata.hashCode();
  }

  public Azienda getControllante(){
    return controllante;
  }

  public Azienda getControllata(){
    return controllata;
  }

  public String toString() {
    return "<" + controllante + ", " + controllata + ">";
  }
}
```

La classe Java AssociazioneControlla

```
// File AppAzienda/AssociazioneControlla.java
package AppAzienda;

public final class AssociazioneControlla{
  private TipoLinkControlla link;

  private AssociazioneControlla(TipoLinkControlla link){
    this.link = link;
  }

  public TipoLinkControlla getLink(){
    return link;
  }

  public static void inserisci(TipoLinkControlla y) {
    if (y != null) {
      AssociazioneControlla k = new AssociazioneControlla(y);
      y.getControllante().inserisciLinkControllante(k);
      y.getControllata().inserisciLinkControllata(k);
    }
  }

  public static void elimina(TipoLinkControlla y) {
    if (y != null) {
```



```
      AssociazioneControlla k = new AssociazioneControlla(y);
      y.getControllante().eliminaLinkControllante(k);
      y.getControllata().eliminaLinkControllata(k);
    }
  }
}
```

La classe Java OperazioniUtente

```
// File AppAzienda/OperazioniUtente.java
package AppAzienda;
import AppAzienda.*;
import AppAzienda.Azienda.*;
import AppAzienda.AziendaPrivata.*;
import java.util.*;

public final class OperazioniUtente{
private OperazioniUtente() {}

    public static boolean controllaSeStessa(Azienda a){
      Azienda azCorrente = a.getLinkControllata().getControllante();
      while(azCorrente!= null){
        if (azCorrente.equals(a))
          return true;
        azCorrente = azCorrente.getLinkControllata().getControllante();
      }
      return false;
    }

    public static Set<AziendaPrivata> azPrivateControllateDa(Azienda a){
      HashSet <AziendaPrivata> result = new HashSet<AziendaPrivata>();
      Set <TipoLinkControlla> linkControlla = a.getLinkControllante();
      Iterator <TipoLinkControlla> it = linkControlla.iterator();
```



```
      while(it.hasNext()){
        TipoLinkControlla l = it.next();
        if (l.getControllata() instanceof AziendaPrivata)
        result.add((AziendaPrivata)l.getControllata());
      }
      return result;
    }
}
```

Realizzazione in Java delle classi per eccezioni

```
// File AppAzienda/EccezioneMolteplicita.java
package AppAzienda;

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

// File AppAzienda/EccezionePrecondizioni.java
package AppAzienda;

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;
  }
}
```