Università di Roma "La Sapienza", Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE I" (Ing. Informatica)
Proff. Marco Cadoli e Maurizio Lenzerini, Canali A-L & M-Z
A.A. 2003-04

Compito d'esame del 30 giugno 2004

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda le informazioni su facoltà, docenti e corsi di una certa università. Di ogni docente interessa il nome ed il cognome, la facoltà di cui è dipendente e da che anno, i corsi che insegna e da che anno. Di ogni corso interessa il nome. Di ogni facoltà interessa il nome e il preside, che è un docente della stessa università, ma non necessariamente dipendente della stessa facoltà. Un docente può essere preside di più facoltà. Una facoltà può essere scientifica o umanistica (ma non entrambe) o non appartenere a nessuna di queste categorie. Delle prime facoltà interessa il numero di laboratori, delle seconde il numero di biblioteche.

Ogni facoltà ha almeno otto docenti fra i suoi dipendenti. Ogni docente insegna fra uno e tre corsi, mentre ogni corso è insegnato da almeno un docente.

Requisiti (cont.)

Ogni docente può essere in servizio oppure assente, in particolare per ferie, malattia o anno sabbatico. Dall'anno sabbatico si può tornare solamente in servizio, mentre se se si è in ferie si può anche andare in malattia e se si è in malattia si può anche andare in anno sabbatico.

Il Ministero dell'Istruzione deve poter effettuare, come cliente della nostra applicazione, dei controlli sulla gestione delle facoltà. A questo scopo, si faccia riferimento ad uno use case che prevede le seguenti operazioni:

- dato un insieme di facoltà, calcolare la percentuale di quelle il cui preside non è loro dipendente;

- dato un insieme di facoltà, calcolare se il numero medio di corsi insegnati da ciascun docente è più alto per quelle scientifiche o per quelle umanistiche.

Fase di analisi

Diagramma delle classi

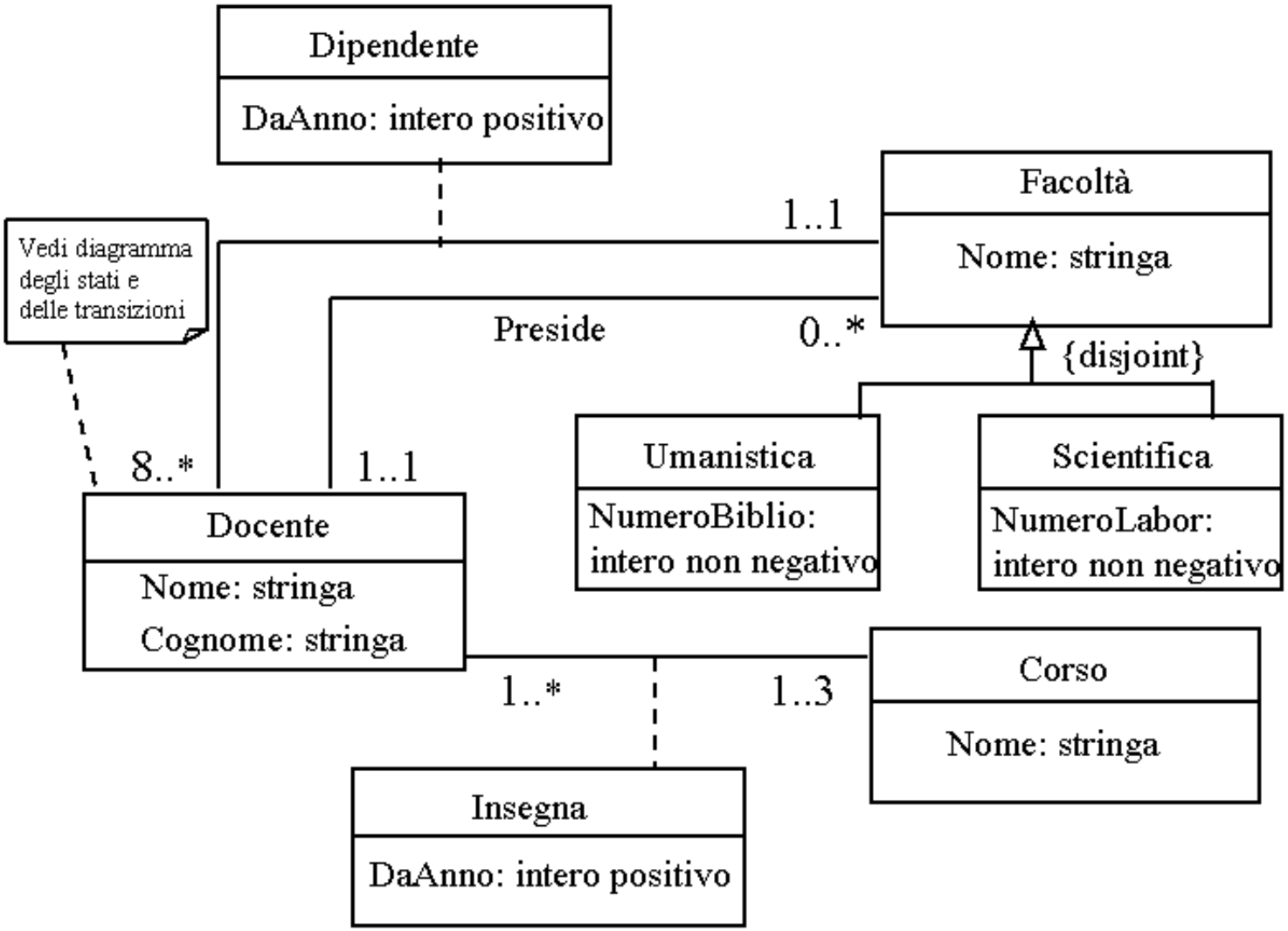


Diagramma degli stati e delle transizioni classe Docente

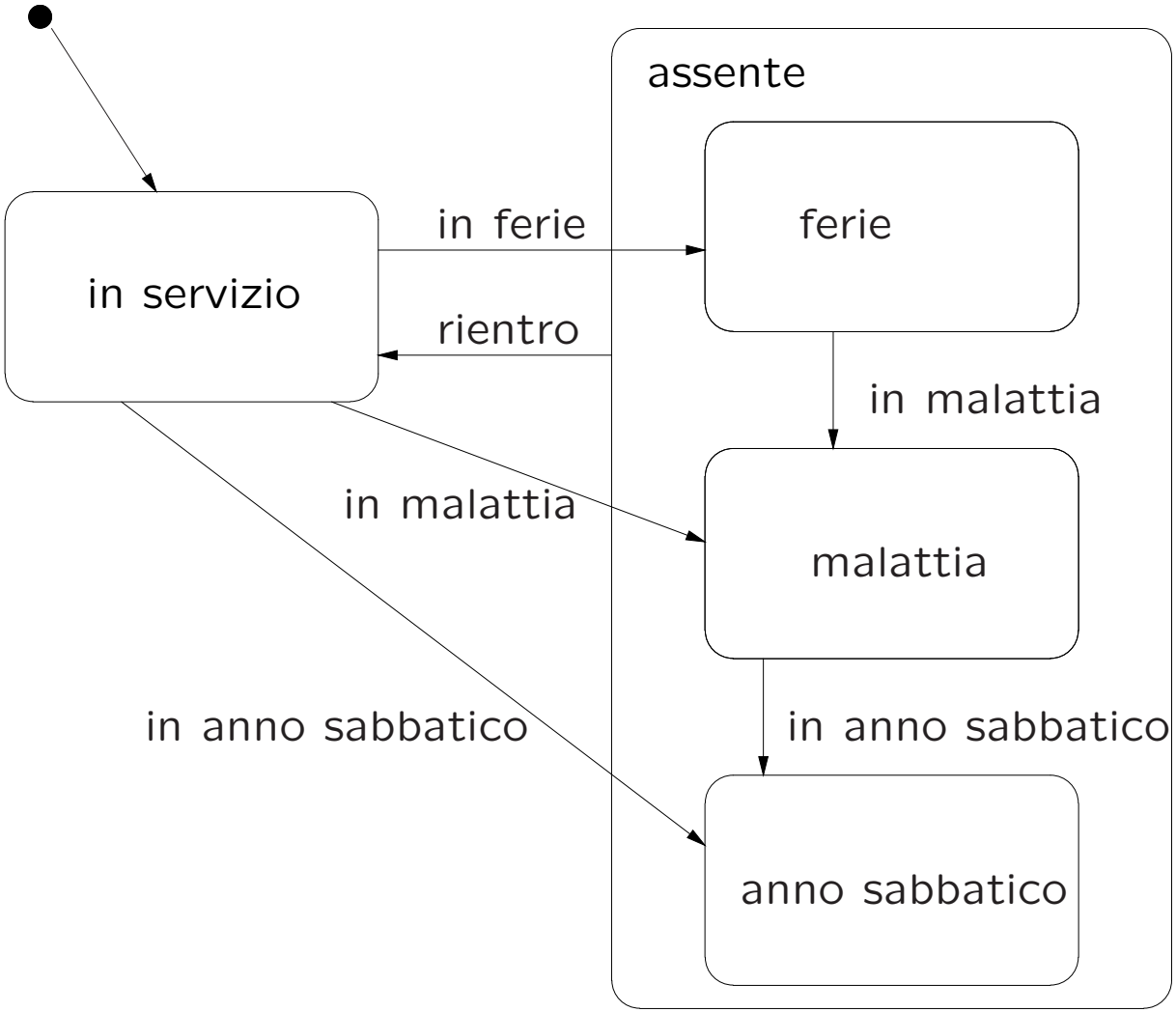
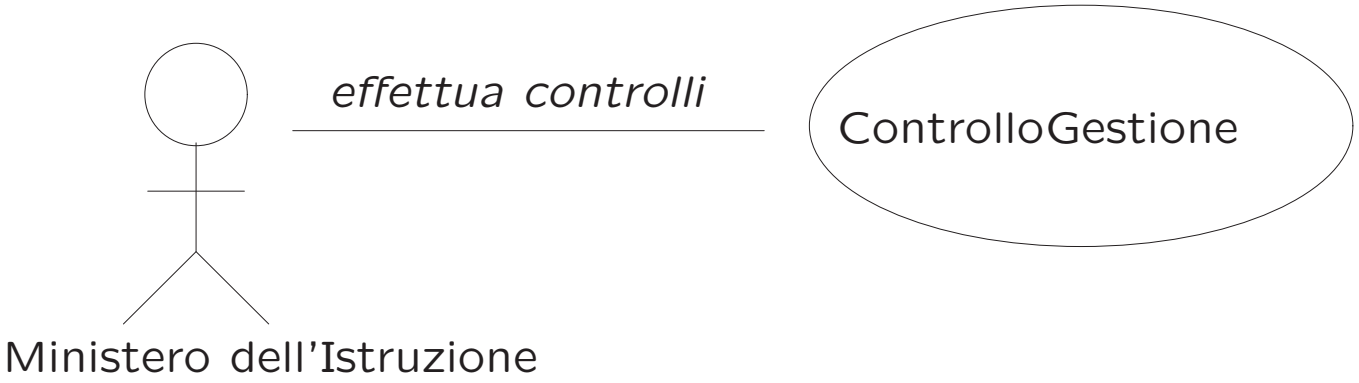


Diagramma degli use case



Specifica delle classi

Non è necessaria la specifica di alcuna delle classi del diagramma delle classi.

Specifica dello use case

InizioSpecificaUseCase ControlloGestione

PercentualePresidiEsterni *(i: Insieme(Facoltà)): reale*

pre: *i* non è vuoto

post: *result* è il valore $|s|/|i| \cdot 100$, dove s è il sottoinsieme di i costituito dalle facoltà f tali che $f.preside \notin f.dipendente$

FacoltàPiùOberate *(i: Insieme(Facoltà)): {0,1,2}*

pre: *i* ha almeno una facoltà umanistica e una scientifica

post: siano u ed s i sottoinsiemi di i costituiti dalle facoltà umanistiche e scientifiche, rispettivamente; siano

$$r_u \doteq \frac{\sum_{f \in u} \sum_{d \in f.dipendente} |d.insegna|}{\sum_{f \in u} |f.dipendente|},$$

$$r_s \doteq \frac{\sum_{f \in s} \sum_{d \in f.dipendente} |d.insegna|}{\sum_{f \in s} |f.dipendente|}$$

result è pari a 0, 1, o 2 se $r_u = r_s$, $r_s < r_u$, o $r_s > r_u$, rispettivamente

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Per l'operazione *PercentualePresidiEsterni()* adottiamo il seguente algoritmo:

```
intero non_dipendenti = 0;
per ogni facoltà f di i
    se (!f.dipendente.appartiene(f.preside)) allora
        non_dipendenti++;
return 100 * non_dipendenti / i.cardinalità;
```

Algoritmi per le operazioni dello use-case (cont.)

Per l'operazione *FacoltàPiùOberate()* adottiamo il seguente algoritmo:

```
intero docenti_umanistiche = 0;
intero docenti_scientifiche = 0;
intero corsi_umanistiche = 0;
intero corsi_scientifiche = 0;
per ogni facoltà f di i
    se (f è umanistica) allora
        docenti_umanistiche += f.dipendente.cardinalità;
        per ogni docente d di f.dipendente
            corsi_umanistiche += d.insegna.cardinalità;
    se (f è scientifica) allora
        docenti_scientifiche += f.dipendente.cardinalità;
        per ogni docente d di f.dipendente
            corsi_scientifiche += d.insegna.cardinalità;
reale r_u = corsi_umanistiche/docenti_umanistiche;
reale r_s = corsi_scientifiche/docenti_scientifiche;
se (r_s == r_u) allora return 0;
se (r_s <  r_u) allora return 1;
se (r_s >  r_u) allora return 2;
```

Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

1. i requisiti,
2. i vincoli di molteplicità nel diagramma delle classi,
3. la specifica degli algoritmi per le operazioni di use-case.

Associazione	Classe 1	ha resp.	Classe 2	ha resp.
dipendente	*Docente*	SI[1,2]	*Facoltà*	SI[2,3]
preside	*Docente*	NO	*Facoltà*	SI[1,2,3]
insegna	*Docente*	SI[1,2,3]	*Corso*	SI[2]

Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
intero positivo	`int`
intero non negativo	`int`
$\{0, 1, 2\}$	`int`
stringa	`String`
reale	`float`
Insieme	`InsiemeListaOmogeneo`

Si noti come nei casi "intero positivo", "intero non negativo" e "$\{0, 1, 2\}$" il tipo Java è semanticamente più esteso del corrispondente tipo UML. Per pure necessità di compattezza, scegliamo di effettuare le relative verifiche delle condizioni di ammissibilità sul lato client.

Strutture di dati

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

- dei vincoli di molteplicità diversi da 0..1 e 1..1 delle associazioni,
- dei parametri delle operazioni dello use-case.

Per fare ciò, utilizzeremo la classe Java `InsiemeListaOmogeneo`.

API per le strutture di dati

```
// File insiemelista/InsiemeListaOmogeneo.java

package insiemelista;

public class InsiemeListaOmogeneo extends InsiemeLista {
  public InsiemeListaOmogeneo(Class cl)
  public InsiemeListaOmogeneo()
  public int size()
  public boolean isEmpty()
  public boolean contains(Object e)
  public boolean add(Object e)
  public boolean remove(Object e)
  public Iterator iterator()
  public boolean containsAll(Collection c)
  public Object[] toArray()
  public Object[] toArray(Object[] a)
  public boolean equals(Object o)
  public Object clone()
  public String toString()
}
```

Proprietà immutabili di classi UML

Riassumiamo le nostre scelte nella seguente tabella delle proprietà immutabili.

Classe UML	Proprietà immutabile
Docente	*nome*
	cognome
Facoltà	*nome*
Corso	*nome*

Sequenza di nascita degli oggetti

Poiché la responsabilità su *preside* è singola, e la molteplicità è (nel verso della responsabilità) 1..1, è ragionevole assumere che, quando nasce un oggetto Java corrispondente ad una facoltà sia noto il docente che ne è preside.

Rappresentazione degli stati in Java

Per la classe UML *Docente*, 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`
in servizio	valore	1
ferie	valore	2
malattia	valore	3
anno sabbatico	valore	4

Ulteriori considerazioni

Non sembrano essere rilevanti altre considerazioni.

API delle classi Java progettate

A titolo di esempio, viene fornita la API della classe `Docente`:

```
public class Docente {
// COSTANTI
    public static final int MIN_LINK_INSEGNA = 1;
    public static final int MAX_LINK_INSEGNA = 3;
// COSTRUTTORE
    public Docente(String n, String c);
// GESTIONE ATTRIBUTI
    public String getNome();
    public String getCognome();
// GESTIONE ASSOCIAZIONI
//   - dipendente
    public int quantiDipendente();
    public void inserisciLinkDipendente(AssociazioneDipendente a);
    public void eliminaLinkDipendente(AssociazioneDipendente a);
    public TipoLinkDipendente getLinkDipendente();
//   - insegna
    public int quantiInsegna();
    public void inserisciLinkInsegna(AssociazioneInsegna a);
    public void eliminaLinkInsegna(AssociazioneInsegna a);
    public InsiemeListaOmogeneo getLinkInsegna();
// GESTIONE EVENTI
    public void in_ferie();
    public void rientro();
    public void in_malattia();
    public void in_anno_sabbatico();
}
```

Struttura dei file e dei package

```
+---AppUniversita
|   |   ControlloGestione.java
|   |   AssociazioneDipendente.java
|   |   AssociazioneInsegna.java
|   |   Corso.java
|   |   Docente.java
|   |   EccezioneCardMax.java
|   |   EccezioneCardMin.java
|   |   EccezionePrecondizioni.java
|   |   Main.java
|   |   TipoLinkDipendente.java
|   |   TipoLinkInsegna.java
|   |
|   +---Facolta
|   |       Facolta.java
|   |
|   +---Scientifica
|   |       Scientifica.java
|   |
|   \---Umanistica
|           Umanistica.java
|
\---insiemelista
        InsiemeListaOmogeneo.java
        InsiemeLista.java
        IteratorInsiemeLista.java
```

Fase di realizzazione

Considerazioni iniziali

Dalle fasi precedenti traiamo come conseguenza che dobbiamo realizzare:

1. cinque classi UML, di cui una ha associato un diagramma degli stati e delle transizioni;

2. due associazioni a responsabilità doppia, di cui una con attributi, e in alcuni casi con vincoli di molteplicità minima diversa da zero e massima finita;

   una associazione senza attributi e a responsabilità singola e con vincoli di molteplicità 1..1 (molteplicità minima diversa da zero);

3. uno use case.

Per facilitare la fase di test e debugging, prevediamo l'overriding della funzione `toString()` per ognuna delle classi Java di cui al punto 1.

Considerazioni iniziali (cont.)

Per quanto riguarda le strutture di dati per la rappresentazione delle associazioni:

- le classi Java `Docente`, `Facolta` e `Corso` avranno un campo di tipo `Insieme-ListaOmogeneo`, in quanto hanno tutte responsabilità in una associazione con molteplicità massima maggiore di uno;
- la classe Java `Docente` avrà inoltre un campo di tipo `TipoLinkDipendente`;
- la classe Java `Facolta` avrà inoltre un campo di tipo `Docente`.

Le classi Java `Docente`, `Facolta` e `Corso` avranno, per ogni associazione `ASSOC` in cui hanno responsabilità:

- una funzione `quantiASSOC()`;
- la funzione `getLinkASSOC()`; tale funzione lancerà una eccezione di tipo `EccezioneCardMin` e/o `EccezioneCardMax`, a seconda dei vincoli.

Considerazioni iniziali (cont.)

La classe Java `AssociazioneDipendente` (coinvolta in un vincolo di molteplicità massima uguale a uno) deve effettuare i controlli del caso, in particolare:

- che l'inserimento avvenga solo se il docente non è già dipendente (sfruttando la funzione `quantiDipendente()` di `Docente`),
- che la cancellazione avvenga solo per link esistenti.

La maniera di realizzare le seguenti classi Java è quella usuale:

- `AssociazioneInsegna`,
- `TipoLinkDipendente`,
- `TipoLinkInsegna`,
- `EccezionePrecondizioni`,
- `EccezioneCardMin`,
- `EccezioneCardMax`.

Sommario classi Java **relative a classi** UML

`Docente:` base, non astratta; responsabilità su *dipendente* e *insegna* (doppie); gestione cardinalità minima maggiore di zero su entrambe le associazioni e massima finita sulla seconda; ha diagramma stati e transizioni.

`Umanistica, Scientifica:` derivate; nessuna responsabilità.

`Facolta:` responsabilità su *dipendente* (doppia) e *preside* (singola); gestione cardinalità minima maggiore di zero su entrambe.

`Corso:` responsabilità su *insegna* (doppia); gestione cardinalità minima maggiore di zero su *insegna*.

Classe Java Facolta

```
// File AppUniversita/Facolta/Facolta.java
package AppUniversita.Facolta;
import AppUniversita.*;
import insiemelista.*;
public class Facolta {
    protected final String nome;
    protected Docente link_preside;
    protected InsiemeListaOmogeneo insieme_link_insegna;
    public static final int MIN_LINK_DIPENDENTE = 8;
    public Facolta(String n, Docente p) {
        nome = n;
        link_preside = p;
        insieme_link_insegna = new
            InsiemeListaOmogeneo(TipoLinkDipendente.class);
    }
    public String getNome() { return nome; }
    public int quantiPreside() {
        if (link_preside == null)
            return 0;
        else return 1;
    }
    public Docente getPreside() throws EccezioneCardMin {
        if (link_preside == null)
            throw new EccezioneCardMin("Cardinalita' minima violata");
```

```
        else
            return link_preside;
    }
    public void setPreside(Docente p) {
        link_preside = p;
    }
    public int quantiDipendente() { return insieme_link_insegna.size(); }
    public void inserisciLinkDipendente(AssociazioneDipendente a) {
        if (a != null) insieme_link_insegna.add(a.getLink());
    }
    public void eliminaLinkDipendente(AssociazioneDipendente a) {
        if (a != null) insieme_link_insegna.remove(a.getLink());
    }
    public InsiemeListaOmogeneo getLinkDipendente() throws EccezioneCardMin {
        if (quantiDipendente() < MIN_LINK_DIPENDENTE)
            throw new EccezioneCardMin("Cardinalita' minima violata");
        else return (InsiemeListaOmogeneo)insieme_link_insegna.clone();
    }
}
```

Classe Java Scientifica

```
// File AppUniversita/Scientifica/Scientifica.java

package AppUniversita.Scientifica;

import insiemelista.*;
import AppUniversita.*;
import AppUniversita.Facolta.*;

public class Scientifica extends Facolta {
    protected int numeroLabor;
    public Scientifica(String n, Docente d) {
        super(n,d);
        numeroLabor = 0;
    }
    public int getNumeroLabor() { return numeroLabor; }
    public void setNumeroLabor(int n) { numeroLabor = n; }

}
```

Classe Java Umanistica

```
// File AppUniversita/Umanistica/Umanistica.java

package AppUniversita.Umanistica;

import insiemelista.*;
import AppUniversita.*;
import AppUniversita.Facolta.*;

public class Umanistica extends Facolta {
    protected int numeroBiblio;
    public Umanistica(String n, Docente d) {
        super(n,d);
        numeroBiblio = 0;
    }
    public int getNumeroBiblio() { return numeroBiblio; }
    public void setNumeroBiblio(int n) { numeroBiblio = n; }

}
```

Classe Java Corso

```
// File AppUniversita/Corso.java
package AppUniversita;
import insiemelista.*;
import AppUniversita.Facolta.*;
public class Corso {
    private final String nome;
    private InsiemeListaOmogeneo insieme_link;
    public static final int MIN_LINK_INSEGNA = 1;
    public Corso(String n) {
        nome = n;
        insieme_link = new InsiemeListaOmogeneo(TipoLinkInsegna.class);
    }
    public String getNome() { return nome; }
    public int quantiInsegna() { return insieme_link.size(); }
    public void inserisciLinkInsegna(AssociazioneInsegna a) {
        if (a != null) insieme_link.add(a.getLink());
    }
    public void eliminaLinkInsegna(AssociazioneInsegna a) {
        if (a != null) insieme_link.remove(a.getLink());
    }
    public InsiemeListaOmogeneo getLinkInsegna() throws EccezioneCardMin {
        if (quantiInsegna() < MIN_LINK_INSEGNA)
            throw new EccezioneCardMin("Cardinalita' minima violata");
        else return (InsiemeListaOmogeneo)insieme_link.clone();
```

```
    }
}
```

Classe Java Docente

```
// File AppUniversita/Docente.java
package AppUniversita;
import insiemelista.*;
public class Docente {
    private final static int in_servizio = 1;
    private final static int ferie = 2;
    private final static int malattia = 3;
    private final static int anno_sabbatico = 4;
    private int corrente = in_servizio;
    private final String nome;
    private final String cognome;
    private TipoLinkDipendente link_dipendente;
    private InsiemeListaOmogeneo insieme_link_insegna;
    public static final int MIN_LINK_INSEGNA = 1;
    public static final int MAX_LINK_INSEGNA = 3;
    public Docente(String n, String c) {
        nome = n;
        cognome = c;
        insieme_link_insegna = new
            InsiemeListaOmogeneo(TipoLinkInsegna.class);
    }
    public String getNome() { return nome; }
    public String getCognome() { return cognome; }
    public int quantiDipendente() {
```

```
    if (link_dipendente == null)
        return 0;
    else return 1;
}
public int quantiInsegna() {
    return insieme_link_insegna.size();
}
public void inserisciLinkDipendente(AssociazioneDipendente a) {
    if (a != null) link_dipendente = a.getLink();
}
public void eliminaLinkDipendente(AssociazioneDipendente a) {
    if (a != null) link_dipendente = null;
}
public TipoLinkDipendente getLinkDipendente() throws EccezioneCardMin {
    if (link_dipendente == null)
        throw new EccezioneCardMin("Cardinalita' minima violata");
    else
        return link_dipendente;
}
public void inserisciLinkInsegna(AssociazioneInsegna a) {
    if (a != null) insieme_link_insegna.add(a.getLink());
}
public void eliminaLinkInsegna(AssociazioneInsegna a) {
    if (a != null) insieme_link_insegna.remove(a.getLink());
}
public InsiemeListaOmogeneo getLinkInsegna() throws EccezioneCardMin,
```

```
                                                    EccezioneCardMax {
        if (quantiInsegna() < MIN_LINK_INSEGNA)
            throw new EccezioneCardMin("Cardinalita' minima violata");
        if (quantiInsegna() > MAX_LINK_INSEGNA)
            throw new EccezioneCardMax("Cardinalita' massima violata");
        else return (InsiemeListaOmogeneo)insieme_link_insegna.clone();
    }
    public void in_ferie() {
        if (corrente == in_servizio)
            corrente = ferie;
    }
    public void rientro() {
        if (corrente == ferie || corrente == malattia ||
            corrente == anno_sabbatico)
            corrente = in_servizio;
    }
    public void in_malattia() {
        if (corrente == ferie || corrente == in_servizio)
            corrente = malattia;
    }
    public void in_anno_sabbatico() {
        if (corrente == in_servizio || corrente == malattia)
            corrente = anno_sabbatico;
    }
}
```

Classe AssociazioneInsegna

```
// File AppUniversita/AssociazioneInsegna.java
package AppUniversita;
import AppUniversita.Facolta.*;

public class AssociazioneInsegna {
    private AssociazioneInsegna(TipoLinkInsegna x) { link = x; }
    private TipoLinkInsegna link;
    public TipoLinkInsegna getLink() { return link; }
    public static void inserisci(TipoLinkInsegna y) {
        if (y != null) {
            AssociazioneInsegna k = new AssociazioneInsegna(y);
            k.link.getCorso().inserisciLinkInsegna(k);
            k.link.getDocente().inserisciLinkInsegna(k);
        }
    }
    public static void elimina(TipoLinkInsegna y) {
        if (y != null) {
            AssociazioneInsegna k = new AssociazioneInsegna(y);
            k.link.getCorso().eliminaLinkInsegna(k);
            k.link.getDocente().eliminaLinkInsegna(k);
        }
    }
}
```

Classe AssociazioneDipendente

```
// File AppUniversita/AssociazioneDipendente.java
package AppUniversita;
import AppUniversita.Facolta.*;

public class AssociazioneDipendente {
    private AssociazioneDipendente(TipoLinkDipendente x) { link = x; }
    private TipoLinkDipendente link;
    public TipoLinkDipendente getLink() { return link; }
    public static void inserisci(TipoLinkDipendente y) {
        if (y != null && y.getDocente().quantiDipendente() == 0) {
            AssociazioneDipendente k = new AssociazioneDipendente(y);
            k.link.getFacolta().inserisciLinkDipendente(k);
            k.link.getDocente().inserisciLinkDipendente(k);
        }
    }
    public static void elimina(TipoLinkDipendente y) {
        try {
            if (y != null && y.getDocente().getLinkDipendente().equals(y) ) {
                AssociazioneDipendente k = new AssociazioneDipendente(y);
                k.link.getFacolta().eliminaLinkDipendente(k);
                k.link.getDocente().eliminaLinkDipendente(k);
            }
        }
        catch (EccezioneCardMin e) {
```

```
            System.out.println(e);
        }
    }
}
```

Classe TipoLinkInsegna

```
// File AppUniversita/TipoLinkInsegna.java
package AppUniversita;
import AppUniversita.Facolta.*;

public class TipoLinkInsegna {
    private final Corso ilCorso;
    private final Docente ilDocente;
    private final int daAnno;
    public TipoLinkInsegna(Corso x, Docente y, int a)
        throws EccezionePrecondizioni {
        if (x == null || y == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        ilCorso = x; ilDocente = y; daAnno = a;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkInsegna b = (TipoLinkInsegna)o;
            return b.ilCorso == ilCorso &&
                b.ilDocente == ilDocente;
        }
        else return false;
    }
    public Corso getCorso() { return ilCorso; }
```

```
    public Docente getDocente() { return ilDocente; }
    public int getDaAnno() { return daAnno; }
}
```

Classe TipoLinkDipendente

```
// File AppUniversita/TipoLinkDipendente.java
package AppUniversita;
import AppUniversita.Facolta.*;

public class TipoLinkDipendente {
    private final Facolta laFacolta;
    private final Docente ilDocente;
    private final int daAnno;
    public TipoLinkDipendente(Facolta x, Docente y, int a)
        throws EccezionePrecondizioni {
        if (x == null || y == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laFacolta = x; ilDocente = y; daAnno = a;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkDipendente b = (TipoLinkDipendente)o;
            return b.laFacolta == laFacolta &&
                b.ilDocente == ilDocente;
        }
        else return false;
    }
    public Facolta getFacolta() { return laFacolta; }
```

```
    public Docente getDocente() { return ilDocente; }
    public int getDaAnno() { return daAnno; }
}
```

Classe EccezionePrecondizioni

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

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

Classe EccezioneCardMin

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

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

Classe EccezioneCardMax

```
// File AppUniversita/EccezioneCardMax.java
package AppUniversita;

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

Realizzazione in Java dello use case

```
// File AppUniversita/ControlloGestione.java
package AppUniversita;
import insiemelista.*;
import java.util.*;
import AppUniversita.Facolta.*;
import AppUniversita.Umanistica.*;
import AppUniversita.Scientifica.*;

public class ControlloGestione {
    public static double percentualePresidiEsterni(InsiemeListaOmogeneo i)
        throws EccezioneCardMin {
        int non_dipendenti = 0;
        Iterator it = i.iterator();
        while(it.hasNext()) {
            Facolta f = (Facolta)it.next();
            Docente preside = f.getPreside();
            boolean presideEsterno = true;
            InsiemeListaOmogeneo dipendenti = f.getLinkDipendente();
            Iterator it2 = dipendenti.iterator();
            while(it2.hasNext() && presideEsterno) {
                TipoLinkDipendente tld = (TipoLinkDipendente) it2.next();
                if (tld.getDocente() == preside)
                    presideEsterno = false;
            }
```

```
        if (presideEsterno)
            non_dipendenti++;
    }
    return 100.0 * non_dipendenti / i.size();
}
public static int facoltaPiuOberate(InsiemeListaOmogeneo i)
    throws EccezioneCardMin, EccezioneCardMax {
    int docenti_umanistiche = 0;
    int docenti_scientifiche = 0;
    int corsi_umanistiche = 0;
    int corsi_scientifiche = 0;
    Iterator it = i.iterator();
    while(it.hasNext()) {
        Facolta f = (Facolta)it.next();
        if (f.getClass().equals(Umanistica.class)) {
            InsiemeListaOmogeneo dipendenti = f.getLinkDipendente();
            docenti_umanistiche += dipendenti.size();
            Iterator it2 = dipendenti.iterator();
            while(it2.hasNext()) {
                TipoLinkDipendente tld = (TipoLinkDipendente) it2.next();
                corsi_umanistiche +=
                    tld.getDocente().getLinkInsegna().size();
            }
        }
        else if (f.getClass().equals(Scientifica.class)) {
            InsiemeListaOmogeneo dipendenti = f.getLinkDipendente();
```

```
                docenti_scientifiche += dipendenti.size();
                Iterator it2 = dipendenti.iterator();
                while(it2.hasNext()) {
                    TipoLinkDipendente tld = (TipoLinkDipendente) it2.next();
                    corsi_scientifiche +=
                        tld.getDocente().getLinkInsegna().size();
                }
            }
        }
        double r_u = ((double)corsi_umanistiche)/docenti_umanistiche;
        double r_s = ((double)corsi_scientifiche)/docenti_scientifiche;
        if (r_s == r_u)
            return 0;
        else if (r_s <  r_u)
            return 1;
        return 2;
    }
    private ControlloGestione() { }
}
```