

Università di Roma “La Sapienza”, Facoltà di Ingegneria

Corso di

“PROGETTAZIONE DEL SOFTWARE I” (Ing. Informatica)

Prof. Maurizio Lenzerini, Canale A-L, A.A. 2002-03

Esercitazione su compito di esame

Docenti universitari: requisiti

L'applicazione da progettare riguarda le informazioni su docenti di una certa università.

1. Dei docenti interessa il nome, il cognome, la data della loro presa di servizio, l'ammontare del loro stipendio, ed il loro dipartimento di appartenenza (ogni docente appartiene ad uno ed un solo dipartimento)..
2. I docenti possono insegnare corsi universitari di varie facoltà.
3. Di ogni corso interessa la facoltà dove è insegnato ed il nome.
4. Esistono tre categorie di docenti distinte fra loro: ordinari, associati e ricercatori.

Docenti universitari: requisiti (cont.)

5. Un ordinario, quando viene assunto, si chiama “straordinario”. Successivamente, può cambiare il suo stato, diventando “ordinario”.

Similmente, associati e ricercatori vengono assunti come “non confermati”, e successivamente possono cambiare il loro stato, diventando “confermati”.

Docenti universitari: requisiti (cont.)

6. La retribuzione dipende dal tipo di docente, dall'anzianità di servizio e dallo stato (confermato o no, straordinario o no).

La formula per gli ordinari è:

$$ord \cdot (1000 + 10 \cdot mesi_di_anzianita)$$

dove *ord* vale 1 per gli straordinari e 1.1 per gli ordinari.

Similmente, per gli associati la formula è:

$$conf \cdot (700 + 7 \cdot mesi_di_anzianita)$$

e per i ricercatori è:

$$conf \cdot (500 + 5 \cdot mesi_di_anzianita)$$

dove *conf* vale 1 per i non confermati e 1.1 per i confermati

L'anzianità si intende in mesi a partire dalla presa di servizio.

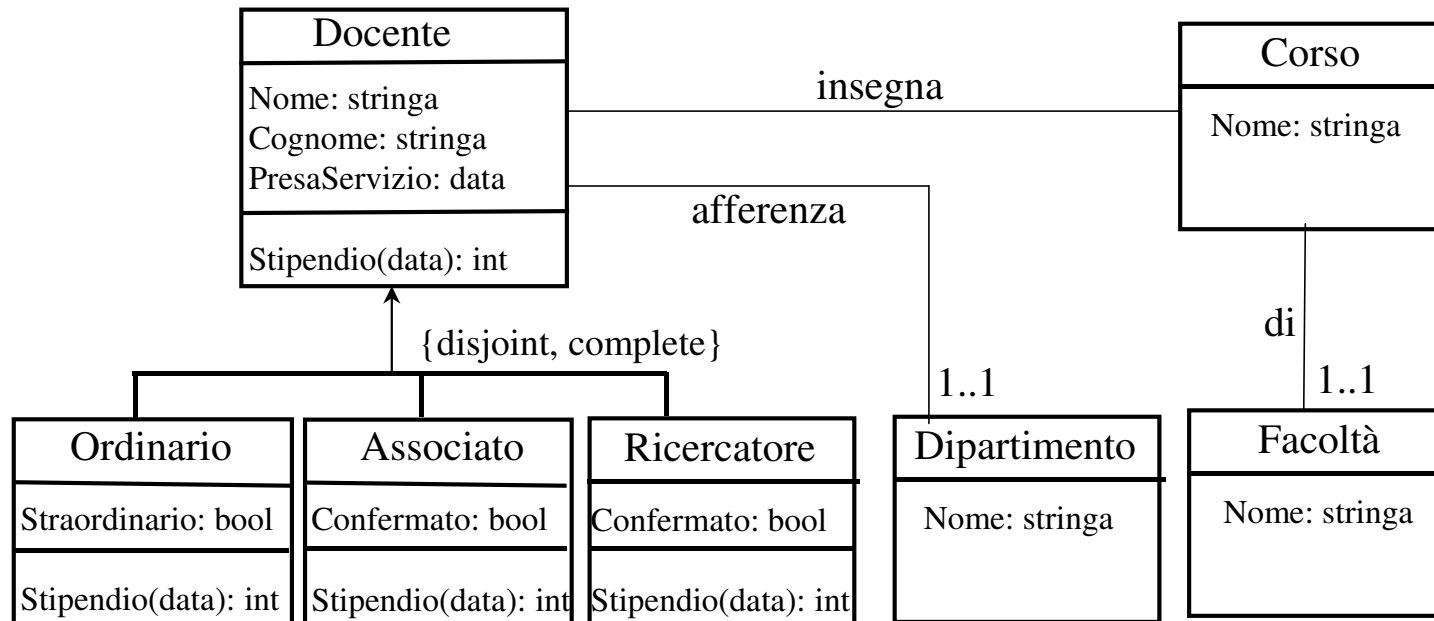
Docenti universitari: requisiti (cont.)

7. La Segreteria Amministrativa deve poter effettuare, come cliente della nostra applicazione, delle elaborazioni. A questo scopo, si faccia riferimento ad uno use case che prevede le seguenti operazioni:
- (a) Stampa del Cedolino relativo ad un docente d , che effettua la stampa delle informazioni principali di d :
- nome e cognome e dipartimento di appartenenza
 - anzianità di servizio in mesi
 - status (tipo di docente, confermato o no, straordinario o no)
 - retribuzione
- (b) Dato un insieme di docenti ins , una facoltà f ed un intero n , stampa dei nomi dei dipartimenti di appartenenza dei soli ordinari in ins che insegnano almeno n corsi in f .

Soluzione: passi necessari

- Analisi
 1. Diagramma delle classi
 2. Specifica delle classi
 3. Diagramma degli use case
 4. Specifica degli use case
- Realizzazione
 1. Considerazioni iniziali (responsabilità sulle associazioni, organizzazione in package, API delle classi ausiliarie)
 2. Realizzazione delle classi del diagramma
 3. Realizzazione delle classi per gli use case

Diagramma delle classi UML



Specifica delle classi UML

InizioSpecificaClasse *Docente*

stipendio(data d): intero

Pre: nessuna

Post: result è lo stipendio mensile del docente this. Lo stipendio dipende dalla sottoclasse di Docente alla quale appartiene this e dalla anzianità di this alla data d.

FineSpecificaClasse

InizioSpecificaClasse *Ordinario*

stipendio(data d): intero

Pre: nessuna

Post: result è pari a $ord \cdot (1000 + 10 \cdot mesi_di_anzianita)$, dove *ord* vale 1 se *this.ordinario = false* e 1.1 se *this.ordinario = true*.

FineSpecificaClasse

Specifica delle classi UML (cont.)

InizioSpecificaClasse *Associato*

stipendio(data d): intero

Pre: nessuna

Post: result è pari a $conf \cdot (700 + 7 \cdot mesi_di_anzianita)$, dove *conf* vale 1 se *this.confermato = false* e 1.1 se *this.confermato = true*.

FineSpecificaClasse

InizioSpecificaClasse *Ricercatore*

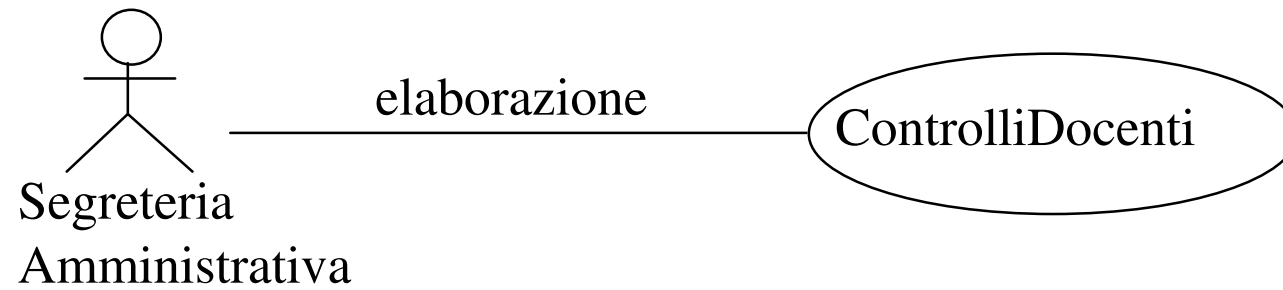
stipendio(data d): intero

Pre: nessuna

Post: result è pari a $conf \cdot (500 + 5 \cdot mesi_di_anzianita)$, dove *conf* vale 1 se *this.confermato = false* e 1.1 se *this.confermato = true*.

FineSpecificaClasse

Diagramma degli use case



Specifica degli use case

InizioSpecificaUseCase ControlliDocenti

Cedolino (*do: Docente, da: data*): *stringa*

pre: nessuna

post: *result* è una stringa che contiene il nome, il cognome ed il dipartimento di appartenenza di *do*, la sua anzianità di servizio in mesi alla data *da*, il suo status (tipo di docente, confermato o no, straordinario o no) e la sua retribuzione alla data *da*.

DipartimentiOrdinari (*ins: Insieme(Docente), f: Facoltà, n: intero*):

Insieme(Dipartimento)

pre: $n \geq 0$

post: *result* è un insieme che contiene i dipartimenti di appartenenza dei soli ordinari in *ins* che insegnano almeno *n* corsi in *f*.

FineSpecifica

Responsabilità sulle associazioni

- Dal requisito 7(a) evince che, dato un docente, è di interesse risalire al suo dipartimento di afferenza.

Di conseguenza, *Docente* ha responsabilità su *afferenza*.

- Dal requisito 7(b) evince che, dato un docente, è di interesse risalire ai corsi che insegna e che, dato un corso, è di interesse risalire alla facoltà in cui è insegnato.

Di conseguenza, *Docente* ha responsabilità su *insegna*, e che *Corso* ha responsabilità su *di*.

- **Nota:** alcune di tali responsabilità sono anche implicate dalla presenza di vincoli di molteplicità minima diversa da zero.

API della classe Java Data

Per la rappresentazione del tipo UML *data*, si suppone essere disponibile una classe Java Data con la seguente interfaccia:

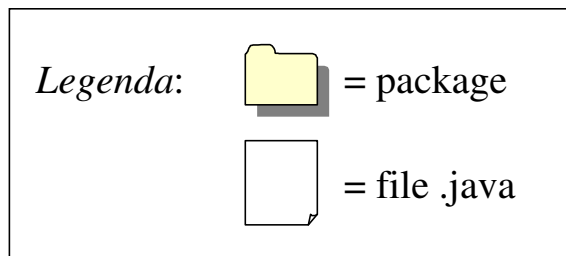
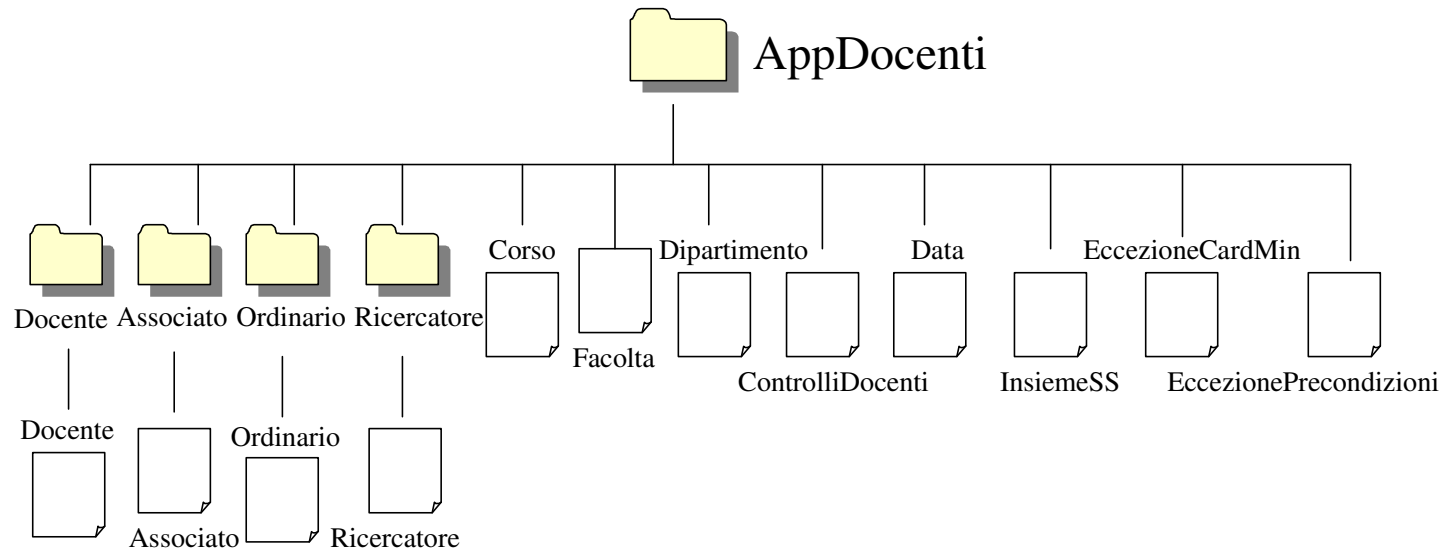
```
// File Data.java
public class Data implements Cloneable {
    public Data();
    public Data(int a, int me, int g);
    public int giorno();
    public int mese();
    public int anno();
    public boolean prima(Data d);
    public String toString();
    public Object clone();
    public boolean equals(Object o);
}
```

API della classe Java InsiemeSS

Per la rappresentazione del tipo UML *Insieme*, si suppone essere disponibile una classe Java `InsiemeSS` con la seguente interfaccia:

```
public class InsiemeSS implements Cloneable {
    public InsiemeSS(Class cl)
    public boolean estVuoto()
    public boolean membro(Object e)
    public void inserisci(Object e)
    public void elimina(Object e)
    public Object scegli()
    public int cardinalita()
    public boolean equals(Object o)
    public Object clone()
    public String toString()
}
```

Organizzazione in package



Classe Java EccezioneCardMin

```
// File AppDocenti/EccezioneCardMin.java

package AppDocenti;

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

Classe Java EccezionePrecondizioni

```
// File AppDocenti/EccezionePrecondizioni.java

package AppDocenti;

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 Java Docente

```
// File AppDocenti/Docente/Docente.java
package AppDocenti.Docente;
import AppDocenti.*;
public abstract class Docente {
    protected final Data presa_di_servizio; // attributo UML
    protected final String nome, cognome; // attributi UML
    protected InsiemeSS corsiInsegnati; // associazione
    protected Dipartimento afferenza; // associazione
    protected Docente(String n, String c, Data p) {
        nome = n;
        cognome = c;
        presa_di_servizio = p;
        corsiInsegnati = new InsiemeSS(Corso.class);
    }
    public String getNome() { return nome; }
    public String getCognome() { return cognome; }
```

```
public int anzianita(Data d) {
    // restituisce i mesi di anzianita' alla data d
    int mesi = d.mese() - presa_di_servizio.mese() +
                12*(d.anno() - presa_di_servizio.anno());
    if (d.giorno() < presa_di_servizio.giorno()) mesi--;
    return mesi;
}

public abstract double stipendio(Data d);
public InsiemeSS getCorsiInsegnati() {
    return (InsiemeSS)corsiInsegnati.clone();
}

public void inserisciLinkInsegna(Corso c) {
    corsiInsegnati.inserisci(c);
}

public void eliminaLinkInsegna(Corso c) {
    corsiInsegnati.elimina(c);
}

public Dipartimento getAfferenza() throws EccezioneCardMin {
```

```
        if (afferenza == null)
            throw new EccezioneCardMin("Cardinalita minima violata");
        return afferenza;
    }
    public void setAfferenza(Dipartimento d) { afferenza = d; }
    public String toString() {
        return nome + " " + cognome + " ";
    }
}
```

Classe Java Ordinario

```
// File AppDocenti/Ordinario/Ordinario.java
package AppDocenti.Ordinario;
import AppDocenti.Docente.*;
import AppDocenti.*;

public class Ordinario extends Docente {
    public Ordinario(String n, String c, Data p) {
        super(n,c,p);
        straordinario = false;
    }
    public double stipendio(Data d) {
        double retr = 1000 + 10 * anzianita(d);
        if (ordinario)
            retr *= 1.1; // aumento del 10%
        return retr;
    }
}
```

```
public void diventaOrdinario() {
    straordinario = false;
}
public String toString() {
    return super.toString() +
        (straordinario?" Professore straordinario":" Professore ordinario");
}
protected boolean straordinario;
}
```

Classe Java Associato

```
// File AppDocenti/Associato/Associato.java
package AppDocenti.Associato;
import AppDocenti.Docente.*;
import AppDocenti.*;

public class Associato extends Docente {
    public Associato(String n, String c, Data p) {
        super(n,c,p);
        confermato = false;
    }
    public double stipendio(Data d) {
        double retr = 700 + 7 * anzianita(d);
        if (confermato)
            retr *= 1.1; // aumento del 10%
        return retr;
    }
}
```

```
public void diventaConfermato() {
    confermato = true;
}
public String toString() {
    return super.toString() + " Professore associato " +
        (!confermato?"non ":"") + "confermato";
}
protected boolean confermato;
}
```

Classe Java Ricercatore

```
// File AppDocenti/Ricercatore/Ricercatore.java
package AppDocenti.Ricercatore;
import AppDocenti.Docente.*;
import AppDocenti.*;

public class Ricercatore extends Docente {
    public Ricercatore(String n, String c, Data p) {
        super(n,c,p);
        confermato = false;
    }
    public double stipendio(Data d) {
        double retr = 500 + 5 * anzianita(d);
        if (confermato)
            retr *= 1.1; // aumento del 10%
        return retr;
    }
}
```

```
public void diventaConfermato() {
    confermato = true;
}
public String toString() {
    return super.toString() + " Ricercatore " +
        (!confermato?"non ":"") + "confermato";
}
protected boolean confermato;
}
```

Classe Java Corso

```
// File AppDocenti/Corso.java

package AppDocenti;

public class Corso {
    public Corso(String s, Facolta f) {
        nome = s;
        facolta = f;
    }
    public String toString() { return nome; }
    public Facolta getFacolta() throws EccezioneCardMin {
        if (facolta == null)
            throw new EccezioneCardMin("Cardinalita minima violata");
        return facolta;
    }
}
```

```
private final String nome;  
private final Facolta facolta;  
}
```

Classe Java Facoltà

```
// File AppDocenti/Facolta.java

package AppDocenti;

public class Facolta {
    public Facolta(String s) {
        nome = s;
    }
    public String toString() { return nome; }
    private final String nome;
}
```

Classe Java Dipartimento

```
// File AppDocenti/Dipartimento.java

package AppDocenti;

public class Dipartimento {
    public Dipartimento(String s) {
        nome = s;
    }
    public String toString() { return nome; }
    private final String nome;
}
```

Implementazione use case

```
// File AppDocenti/ControlliDocenti.java

package AppDocenti;
import AppDocenti.Docente.*;
import AppDocenti.Ordinario.*;

public class ControlliDocenti {

    public static String Cedolino(Docente doc, Data d) {
        String result = new String();
        result += "===== UNIVERSITA' DI RIO ===== \n";
        result += "***** \n";
        result += "Rio, " + d + "\n";
        result += "CEDOLINO DI " + doc + "\n";
        result += "ANZIANITA' DI SERVIZIO (MESI) = "+
            doc.anzianita(d) + "\n";
    }
}
```

```
    result += "Retribuzione = " + doc.stipendio(d) + "\n";
    result += "----- \n";
    return result;
}
```

```
public static InsiemeSS DipartimentiOrdinari(InsiemeSS docenti,
                                             int minCorsi, Facolta fac)
    throws EccezioneCardMin, EccezionePrecondizioni {
    if (minCorsi < 0) throw new EccezionePrecondizioni();
    InsiemeSS result = new InsiemeSS(Dipartimento.class);
    if (docenti == null) return result;
    while (!docenti.estVuoto()) {
        Docente doc = (Docente)docenti.scegli();
        docenti.elimina(doc);
        if (Ordinario.class.isInstance(doc)) {
            InsiemeSS corsiInsegnati = doc.getCorsiInsegnati();
            int quanti = 0;
            while (!corsiInsegnati.estVuoto()) {
```

```
        Corso c = (Corso)corsiInsegnati.scegli();
        corsiInsegnati.elimina(c);
        if (c.getFacolta() == fac) {
            quanti++;
            if (quanti >= minCorsi) {
                result.inserisci(doc.getAfferenza());
                break;
            }
        }
    }
}
return result;
}
}
```

Esercizio

Riprogettare l'applicazione prendendo in considerazione anche i seguenti requisiti:

- Di ogni docente interessa anche la facoltà di cui è dipendente e l'anno in cui lo è diventato.
- Alla Segreteria Amministrativa interessa anche sapere quali sono, fra i dipendenti di una facoltà, quei docenti che insegnano corsi in altre facoltà.