Università di Roma "La Sapienza"

A.A. 2006-2007

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Corso di "PROGETTAZIONE DEL SOFTWARE I"
(Canale A-L & M-Z)

Esercitazione numero 8

LA FASE DI REALIZZAZIONE

Realizzazione di diagrammi degli stati e delle transizioni

(SOLUZIONE)

Fase di analisi

Diagramma delle classi e degli use case

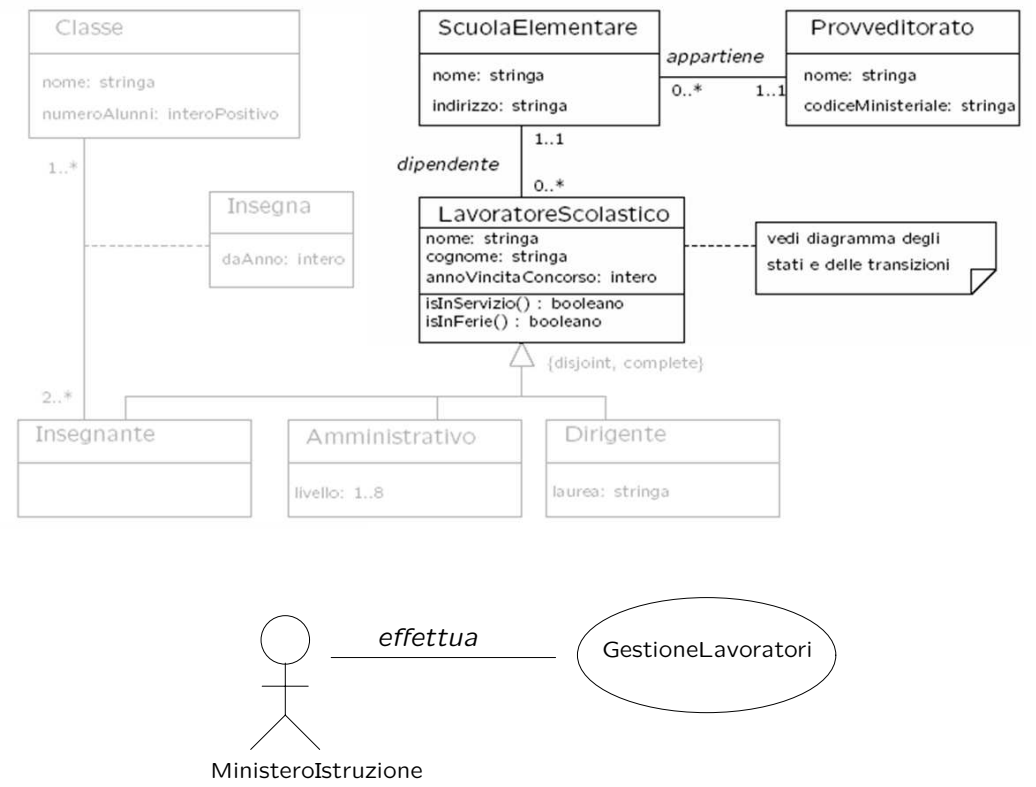
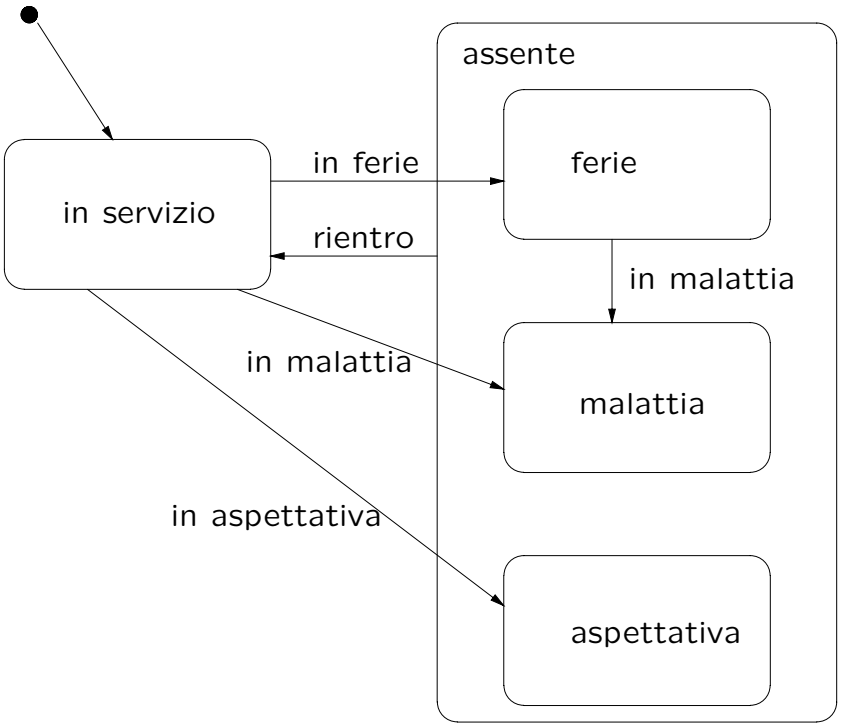


Diagramma degli stati e delle transizioni classe LavoratoreScolastico



Specifica classe LavoratoreScolastico

InizioSpecificaClasse **LavoratoreScolastico**

isInServizio *(): booleano*

pre: nessuna

post: *result* è pari a true se e solo se *this* si trova nello stato "in_servizio" (cf. diagramma degli stati e transizioni).

isInFerie *(): booleano*

pre: nessuna

post: *result* è pari a true se e solo se *this* si trova nello stato "ferie" (cf. diagramma degli stati e transizioni).

FineSpecifica

Specifica dello use-case

InizioSpecificaUseCase **GestioneLavoratori**

rientroForzatoDaFerie *(S: Insieme(LavoratoreScolastico)): boolean*

pre: nessuna

post: Definiamo preliminarmente alcuni insiemi.

$Scuole \doteq \{x \mid x \in Scuola \wedge (\exists lav\ lav \in S \wedge\ lav.dipendente = x)\}$

$ScuoleNonOK \doteq \{s \mid s \in\ Scuole\ \wedge$
$(\forall lav\ (lav \in S \wedge\ lav.dipendente = s) \rightarrow\ lav.isInServizio = false)\}$

Per ogni $s \in ScuoleNonOK$ definiamo:
$InFerie_s \doteq \{l \mid l \in S \wedge l.isInFerie = true \wedge l.dipendente = s)\}$

- Se $\exists s\ (s \in ScuoleNonOK \wedge InFerie_s = \emptyset)$ allora *result* è *false*.
- Altrimenti, per ogni $s \in ScuoleNonOK$ sia l un elemento arbitrario di $InFerie_s$; va generato l'evento *l.rientro()*. Inoltre, *result* vale *true*.

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use case

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

```
Insieme(LavoratoreScolastico) daFarRientrare = insieme vuoto;
Insieme(ScuolaElementare) scuoleNonOk = insieme vuoto;
Insieme(ScuolaElementare) scuoleOk = insieme vuoto;

per ogni LavoratoreScolastico lav in S {
  ScuolaElementare scuola = lav.dipende;
  se lav.isInServizio() == true {
    aggiungi scuola a scuoleOk;
    rimuovi scuola da scuoleNonOk;
  };
  altrimenti se scuola non è in scuoleOk
    aggiungi scuola a scuoleNonOk;
}
... CONTINUA
```

Algoritmi per le operazioni dello use case (cont.)

```
...
per ogni LavoratoreScolastico lav in S {
  ScuolaElementare scuola = lav.dipende;
  se scuola è in scuoleNonOk
    se lav.isInFerie() == true {
      aggiungi lav nell'insieme daFarRientrare;
      elimina scuola da scuoleNonOk;
    }
}
se scuoleNonOk non è l'insieme vuoto {
  result = false;
  return result;
}
altrimenti {
  per ogni lavoratore lav in daFarRientrare
    genera l'evento rientro per lav
  result = true;
  return result;
}
```


Responsabilità sulle associazioni

Riportiamo la tabella delle responsabilità.

Associazione	Classe	ha resp.
insegna	*Classe*	SÌ[1,2,3]
	Insegnante	SÌ[1,2,3]
dipendente	*ScuolaElementare*	NO
	LavoratoreScolastico	SÌ[1,3]
appartiene	*ScuolaElementare*	SÌ[1,3]
	Provveditorato	NO

1. dai requisiti
2. dagli algoritmi
3. dai vincoli di molteplicità

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, a causa dei parametri dell'operazione dello use-case e delle variabili locali necessarie per l'algoritmo. Per fare ciò, utilizzeremo la classe Java `HashSet`.

Corrispondenza fra tipi UML e Java

Possiamo riassumere il risultato delle nostre scelte nella seguente *tabella di corrispondenza dei tipi* UML.

Tipo UML	Rappresentazione in Java
intero	`int`
interoPositivo	`int`
1..8	`int`
stringa	`String`
Insieme	`HashSet`

Tabelle di gestione delle proprietà di classi UML

Riassumiamo tutte le nostre scelte **differenti da quelle di default** mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita* (v. lucidi della terza parte del corso).

Classe UML	Proprietà immutabile
Provveditorato	*nome*
Classe	*nome*
LavoratoreScolastico	*nome*
	cognome
	anno vincita concorso
Dirigente	*laurea*
ScuolaElementare	*appartiene*

Classe UML	**Proprietà**	
	nota alla nascita	**non nota alla nascita**
LavoratoreScolastico	–	*dipendente*

Sequenza di nascita degli oggetti

Poiché le responsabilità su *dipendente* ed *appartiene* sono *singole*, e la molteplicità è per entrambe 1..1, è ragionevole assumere che:

- quando nasce un oggetto Java corrispondente ad un lavoratore scolastico sia nota la scuola elementare di cui è dipendente;
- quando nasce un oggetto Java corrispondente ad una scuola elementare sia noto il suo provveditorato di appartenenza.

Rappresentazione degli stati in Java

Classe UML *LavoratoreScolastico*.

Rappresentazione in Java del diagramma degli stati e delle transizioni.

Tabella di codifica degli stati mediante **una variabile** `int`.

Stato	Rappresentazione in Java	
	tipo var.	`int`
	nome var.	`stato`
in servizio	valore	1
ferie	valore	2
malattia	valore	3
aspettativa	valore	4

Fase di realizzazione

Considerazioni

Dalle fasi precedenti traiamo come conseguenza che dobbiamo realizzare:

- tre classi UML, di cui una ha associato un diagramma degli stati e delle transizioni,
- due associazioni con attributi e a responsabilità singola, entrambe con vincolo di molteplicità 1..1,
- uno use case.

Struttura dei file e dei package

```
AppScuole
|   LavoratoreScolastico.java
|   Main.java
|   Provveditorato.java
|   GestioneLavoratori.java
|   ScuolaElementare.java
```

Classe Provveditorato

```
// File Provveditorato.java

public class Provveditorato {
    private final String nome;
    private String codMin;
    public Provveditorato(String no, String cod) {
        nome = no;
        codMin = cod;
    };
    public String getNome() {
        return nome;
    };
    public String getCodMin() {
        return codMin;
    };
    public void setCodMin(String c) {
        codMin = c;
    };
    public String toString() {
        return nome + " (" + codMin + ")";
    };
};
```


Classe ScuolaElementare

```
// File ScuolaElementare.java

public class ScuolaElementare {
    private String nome;
    private String indirizzo;
    private final Provveditorato provv;

    public ScuolaElementare(String no, String ind, Provveditorato p) {
        nome = no; indirizzo = ind;
        provv = p;
    }
    public String getNome()               { return nome; };
    public void setNome(String no)        { nome = no; };
    public String getIndirizzo()          { return indirizzo; };
    public void setIndirizzo(String ind) { indirizzo = ind; };
    public String toString() {
        return nome + " (" + indirizzo + ", " + provv + ")";
    };
};
```

Classe LavoratoreScolastico

```
// File LavoratoreScolastico.java

public class LavoratoreScolastico {
    private final String nome, cogn;
    private final int annoVinc;
    private ScuolaElementare dipendente;

    private static final int in_servizio = 1, ferie = 2, malattia = 3, aspettativa = 4;

    private int stato_corrente = in_servizio;

    public LavoratoreScolastico(String n, String cog, int a, ScuolaElementare dip) {
        nome = n;    cogn = cog;    annoVinc = a;    dipendente = dip;
    };
    public String getNome() { return nome; }
    public String getCognome() { return cogn; }
    public ScuolaElementare getScuolaElementare() { return dipendente; }
    public void setScuolaElementare(ScuolaElementare s) { dipendente = s; };
    public boolean isInFerie() { return stato_corrente == ferie; };
    public boolean isInServizio() { return stato_corrente == in_servizio; };


    // Gestione eventi, stati e transizioni
    public void in_ferie() {
```

```
        if (stato_corrente == in_servizio) stato_corrente = ferie;
    };
    public void in_malattia() {
        if (stato_corrente==in_servizio || stato_corrente==ferie) stato_corrente=malattia;
    };
    public void in_aspettativa() {
        if (stato_corrente == in_servizio) stato_corrente = aspettativa;
    };
    public void rientro() {
        if (stato_corrente == ferie || stato_corrente == malattia ||
            stato_corrente == aspettativa)    stato_corrente = in_servizio;
    };
    public String toString() {
        return nome + " " + cogn + "(" + dipendente + ")";
    };
};
```

Classe GestioneLavoratori

```
// File GestioneLavoratori.java

import java.util.*;

public final class GestioneLavoratori {

    private GestioneLavoratori() {};

    public static boolean rientroForzatoDaFerie(Set<LavoratoreScolastico> s) {
        HashSet<LavoratoreScolastico> daFarRientrare =
            new HashSet<LavoratoreScolastico>();
        HashSet<ScuolaElementare> scuoleNonOk = new HashSet<ScuolaElementare>();
        HashSet<ScuolaElementare> scuoleOk = new HashSet<ScuolaElementare>();
        Iterator<LavoratoreScolastico> it = s.iterator();
        while (it.hasNext()) {
            LavoratoreScolastico lav = it.next();
            ScuolaElementare scuola = lav.getScuolaElementare();
            if (lav.isInServizio()) {
                scuoleOk.add(scuola);
                scuoleNonOk.remove(scuola);
            }
            else if (!scuoleOk.contains(scuola))
                scuoleNonOk.add(scuola);
        };
```

```
        it = s.iterator();
        while(it.hasNext()) {
            LavoratoreScolastico lav = it.next();
            ScuolaElementare scuola = lav.getScuolaElementare();
            if (!scuoleNonOk.contains(scuola)) continue;

            if (lav.isInFerie()) {
                daFarRientrare.add(lav);
                scuoleNonOk.remove(scuola);
            };
        };
        if (scuoleNonOk.size() != 0)
            // Non tutte le scuole hanno un lavoratore che può rientrare
            return false;
        it = daFarRientrare.iterator();
        while (it.hasNext()) {
            LavoratoreScolastico lav = it.next();
            lav.rientro();
        };
        return true;
    };
}
```