Università di Roma "La Sapienza"
Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE I"
(Corso di Laurea in Ingegneria Informatica)
Proff. Giuseppe De Giacomo e Marco Cadoli
Canali A-L & M-Z
A.A. 2004-05

Compito d'esame del 22 giugno 2005

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda le informazioni sulle visite mediche effettuate in una certa regione. Di ogni visita interessa l'assistito a cui è stata effettuata, il medico che l'ha fatta, la data di effettuazione e le eventuali prescrizioni indicate. Di ogni persona (medico o assistito) interessa il codice regionale, il nome, il cognome e la data di nascita, le visite che ha ricevuto come assistito. Si noti che una stessa persona può essere medico o assistito, a seconda delle circostanze. Di ogni prescrizione interessa il codice e la visita in cui è stata indicata. Esistono solamente due categorie di prescrizioni, che sono fra loro disgiunte: prescrizione di farmaco e richiesta di ricovero. Delle prime interessa la quantità del farmaco. Delle seconde interessa il reparto previsto.

Requisiti (cont.)

Una visita è inizialmente prenotata; successivamente può essere disdetta o effettuata. Solamente le visite effettuate possono avere l'indicazione di prescrizioni.

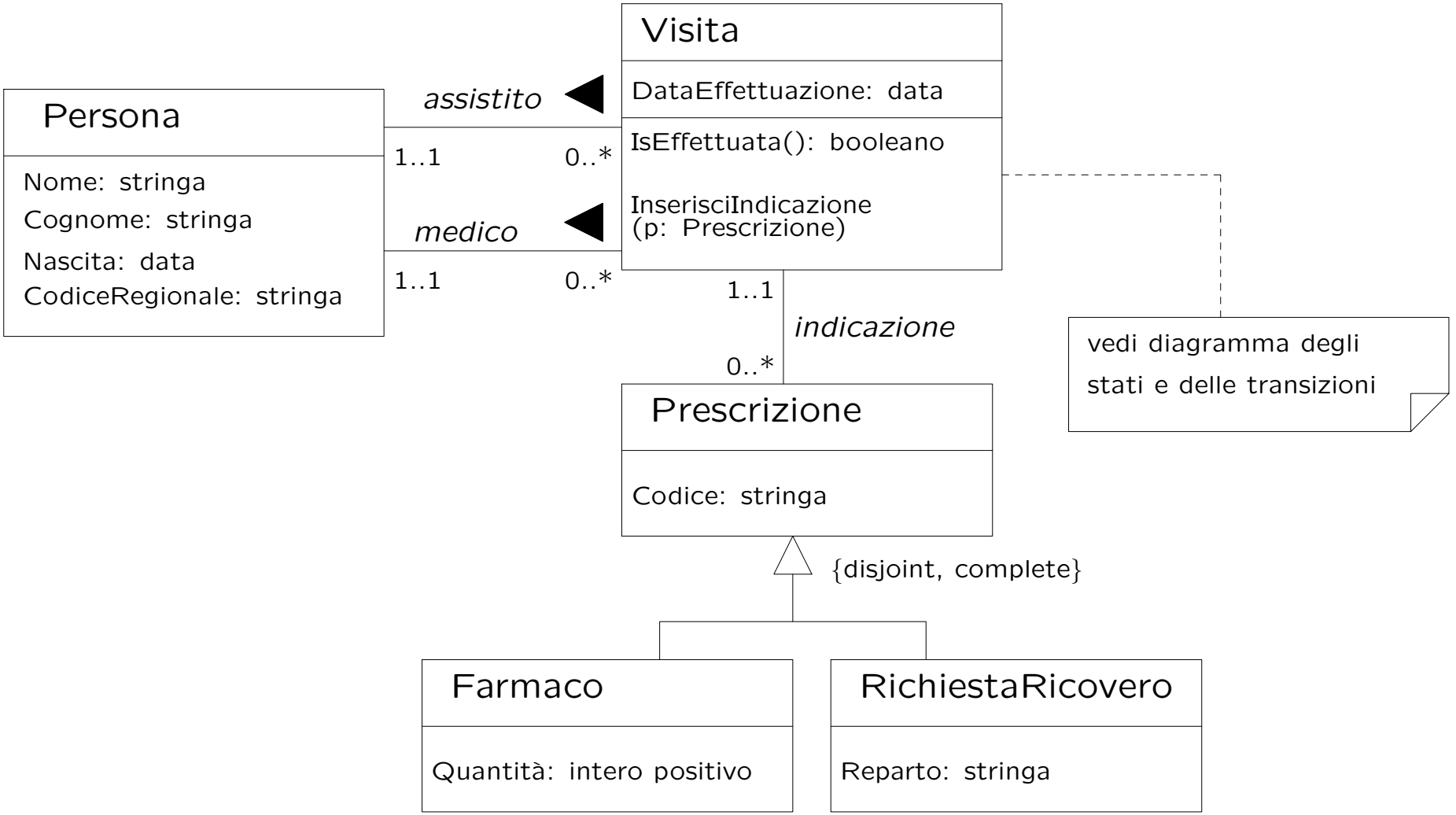
La ASL vuole effettuare, come cliente della nostra applicazione, dei controlli sulle visite. A questo scopo, si faccia riferimento ad uno use case che prevede che si possa sapere:

- dato un insieme di visite, la percentuale di visite in cui è stato prescritto almeno un farmaco,

- data una persona p e un anno a, la quantità totale di farmaci che sono stati prescritti a p durante a.

Fase di analisi

Diagramma delle classi



Commento sul diagramma delle classi

Una soluzione alternativa prevede che la classe *Persona* abbia due sottoclassi *Medico* e *Assistito*, che formano una generalizzazione **non disgiunta**, in quanto i requisiti prevedono che una stessa persona possa essere medico o assistito, a seconda delle circostanze.

È stata scelta la soluzione con un'unica classe a causa della difficoltà di realizzare in Java generalizzazioni non disgiunte.

Diagramma degli stati e delle transizioni classe Visita

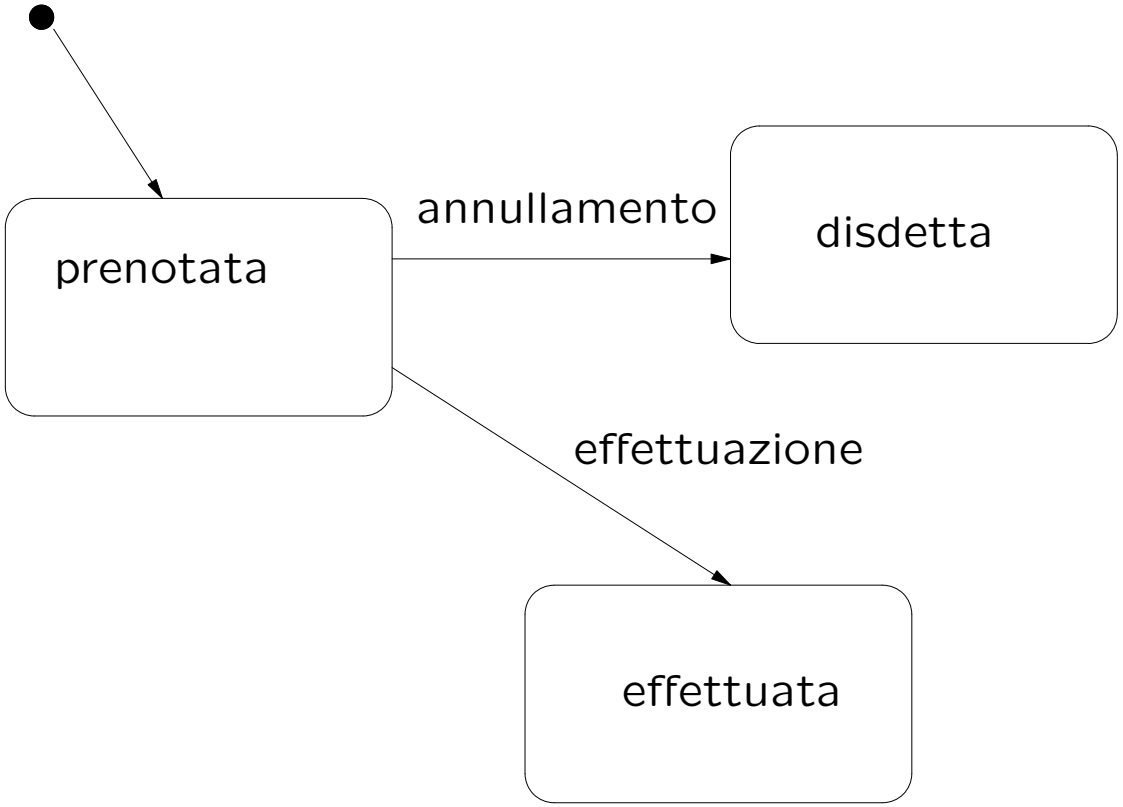
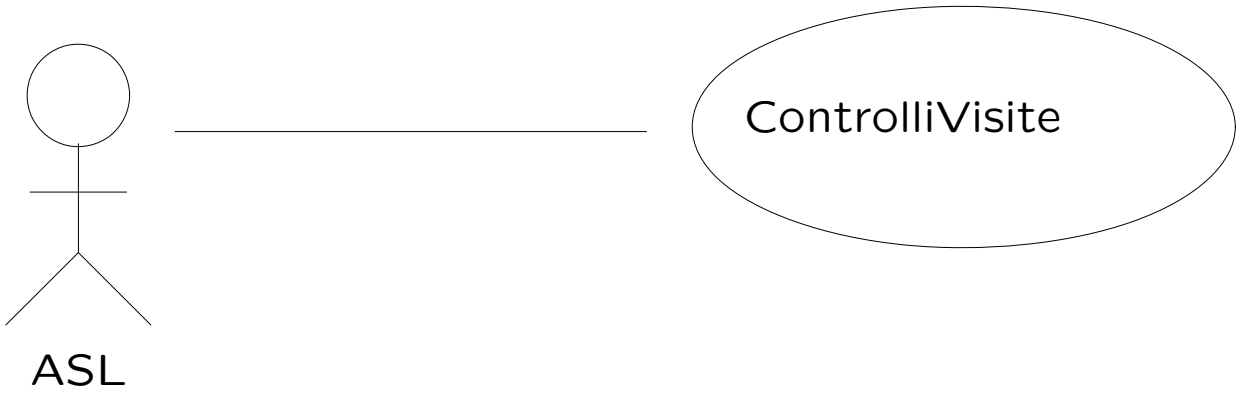


Diagramma degli use case



Specifica della classe Visita

InizioSpecificaClasse Visita

InserisciIndicazione *(p: Prescrizione)*

pre: *this.IsEffettuata()* = true. Inoltre, nell'insieme L di link di tipo *indicazione* legati all'oggetto *this* non ve n'è alcuno il cui oggetto *Prescrizione* sia *p*

post: *this* è legato ad un nuovo link di tipo *indicazione*, il cui oggetto *Prescrizione* è *p*.

IsEffettuata() *: booleano*

pre: nessuna

post: *result* vale *true* se e solo se lo stato di *this* è *effettuata*.

FineSpecifica

Specifica dello use case

InizioSpecificaUseCase **ControlliVisite**

PercentualeVisiteConFarmaci *(iv: Insieme(Visita)): reale*

pre: $iv \neq \emptyset$

post: sia $iv' \doteq \{v \in iv \wedge (\exists i \ i \in v.indicazione \wedge i.class = Farmaco)\}$.
result vale $|iv'| \cdot 100/|iv|$.

QuantiFarmaci *(p: Persona, a: int): intero positivo*

pre: $a > 0$

post: *result* vale

$$\sum_{\substack{v \in p.assistito \ \wedge \\ v.DataEffettuazione.anno = a}} \ \sum_{\substack{pr \in v.indicazione \ \wedge \\ pr.class = Farmaco}} pr.Quantità$$

FineSpecifica

Fase di progetto

Algoritmi per le operazioni delle classi

Adottiamo i seguenti algoritmi:

- Per l'operazione **InserisciIndicazione** della classe *Visita*:

```
TipoLinkIndicazione t = nuovo oggetto con contenuti <this,p>
this.indicazione.aggiungi(t);
```

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **PercentualeVisiteConFarmaci**:

```
int quante_visite = iv.size();
int quante_visite_con_farmaci = 0;
per ogni visita v di iv
    Insieme(Prescrizione) p = v.indicazione;
    booleano visita_con_farmaco = false;
    per ogni prescrizione i di p
        se i.class == Farmaco AND NOT visita_con_farmaco
            visita_con_farmaco = true
            quante_visite_con_farmaci++;
return (float)quante_visite_con_farmaci * 100 / quante_visite;
```

Algoritmi per le operazioni degli use case (cont.)

- Per l'operazione **QuantiFarmaci**:

```
int result = 0;
Insieme(Visita) iv = p.assistito;
per ogni visita v di iv
    se v.DataEffettuazione.anno == a
        Insieme(Prescrizione) ip = v.indicazione;
        per ogni prescrizione pr di ip
            se pr.class == Farmaco
                result += pr.Quantità
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.
assistito	*Persona*	SÌ[1,2]
	Visita	SÌ[1,3]
medico	*Persona*	NO
	Visita	SÌ[1,3]
indicazione	*Visita*	SÌ[1,2]
	Prescrizione	SÌ[1,3]

Strutture di dati

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

- dei vincoli di molteplicità 0..* delle associazioni,
- delle variabili locali necessarie per vari algoritmi.

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

Abbiamo anche la necessità di rappresentare date. A tale scopo utilizzeremo la classe Java `Data`.

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()
}
```

API per le strutture di dati (cont.)

```
// File Data/Data.java
package Data;

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 void avanzaUnGiorno();
  public String toString();
  public Object clone();
  public boolean equals(Object o);
}
```

Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
booleano	`boolean`
intero	`int`
intero positivo	`int`
stringa	`String`
Insieme	`InsiemeListaOmogeneo`
data	`Data`

Per tenere conto del fatto che, nel caso "intero positivo", il tipo Java è semanticamente più esteso del corrispondente tipo UML, prevediamo una verifica delle condizioni di ammissibilità sul lato server, perché è una soluzione di migliore qualità.

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**
Persona	*Nome*
	Cognome
	Nascita
	CodiceRegionale
Visita	*DataEffettuazione*
	assistito
	medico
Prescrizione	*Codice*
	indicazione
Farmaco	*Quantità*
RichiestaRicovero	*Reparto*

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

Altre considerazioni

Sequenza di nascita degli oggetti: Poiché le responsabilità su *medico* è singola, e la molteplicità è (nel verso della responsabilità) 1..1, è ragionevole assumere che quando nasce un oggetto Java corrispondente ad una visita sia nota la persona (che svolge il ruolo di medico) associata.

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 *Visita*, 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`
prenotata	valore	1
disdetta	valore	2
effettuata	valore	3

API delle classi Java progettate

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

```
public class Visita {
    // RAPPRESENTAZIONE VINCOLI MOLTEPLICITÀ MINIMA MAGGIORE DI ZERO
    public static final int MIN_LINK_ASSISTITO = 1;
    // COSTRUTTORE
    public Visita(Data d, Persona m)
    // GESTIONE ATTRIBUTI
    public Persona getMedico()
    public Persona getAssistito()
    public Data getData()
    // GESTIONE STATO
    public void annullamento()
    public void effettuazione()
    public boolean isEffettuata()
    // GESTIONE ASSOCIAZIONI
    // - ASSISTITO
    public void inserisciLinkAssistito(AssociazioneAssistito a)
    public void eliminaLinkAssistito(AssociazioneAssistito a)
    public int quantiAssistiti()
    public TipoLinkAssistito getLinkAssistito() throws EccezioneCardMin
    // - INDICAZIONE
    public void inserisciLinkIndicazione(AssociazioneIndicazione a)
    public void eliminaLinkIndicazione(AssociazioneIndicazione a)
    public InsiemeListaOmogeneo getLinkIndicazione()
    // STAMPA
    public String toString()
}
```

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. una associazione a responsabilità singola e con vincoli di molteplicità 0..* e 1..1 (molteplicità minima diversa da zero, ma immutabile e nota alla nascita);

   due associazioni a responsabilità doppia e con vincoli di molteplicità 0..* e 1..1;

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.

Sommario classi Java relative a classi UML

`Visita:` responsabilità su *assistito* (doppia, cardinalità minima maggiore di zero, cardinalità massima finita), *indicazione* (doppia), *medico* (singola, cardinalità minima maggiore di zero, cardinalità massima finita); ha diagramma stati e transizioni.

`Prescrizione:` astratta; responsabilità su *indicazione* (doppia, cardinalità minima maggiore di zero, cardinalità massima finita).

`Farmaco:` derivata; nessuna associazione.

`RichiestaRicovero:` derivata; nessuna associazione.

`Persona:` responsabilità su *assistito* (doppia).

Struttura dei file e dei package

```
+---insiemelista
|       InsiemeListaOmogeneo.java
|       InsiemeLista.java
|       IteratorInsiemeLista.java
|
+---AppVisiteMediche
|   |   AssociazioneAssistito.java
|   |   AssociazioneIndicazione.java
|   |   ControlliVisite.java
|   |   EccezioneCardMin.java
|   |   EccezionePrecondizioni.java
|   |   Persona.java
|   |   TipoLinkAssistito.java
|   |   TipoLinkIndicazione.java
|   |   Visita.java
|   |
|   +---Farmaco
|   |       Farmaco.java
|   |
|   +---Prescrizione
|   |       Prescrizione.java
|   |
|   \---RichiestaRicovero
|           RichiestaRicovero.java
|
\---Data
        Data.java
```

La classe Java Prescrizione

```
// File AppVisiteMediche/Prescrizione/Prescrizione.java
package AppVisiteMediche.Prescrizione;
import AppVisiteMediche.*;

public abstract class Prescrizione {
    private final String codice;
    private TipoLinkIndicazione indicazione;
    public static final int MIN_LINK_INDICAZIONE = 1;
    public Prescrizione(String c) {
        codice = c;
    }
    public String getCodice() { return codice; }
    public void inserisciLinkIndicazione(AssociazioneIndicazione a) {
        if (a != null)
            indicazione = a.getLink();
    }
    public int quantiIndicazione() {
        if (indicazione == null)
            return 0;
        else return 1;
    }
    public void eliminaLinkIndicazione(AssociazioneIndicazione a) {
        if (a != null) indicazione = null;
    }
```

```
    public TipoLinkIndicazione getLinkIndicazione() throws EccezioneCardMin {
        if (indicazione == null)
            throw new EccezioneCardMin("Cardinalita' minima violata");
        else
            return indicazione;
    }
    public String toString() {
        return codice;
    }
}
```

La classe Java Farmaco

```
// File AppVisiteMediche/Farmaco/Farmaco.java
package AppVisiteMediche.Farmaco;
import AppVisiteMediche.Prescrizione.*;
import AppVisiteMediche.*;

public class Farmaco extends Prescrizione  {
    private final int quantita;
    public Farmaco(String c, int q) throws EccezionePrecondizioni {
        super(c);
        if (q < 0)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("La quantita' deve essere maggiore di 0");
        quantita = q;
    }
    public int getQuantita() { return quantita; }
    public String toString() {
        return super.toString() + " quantita' farmaco: " + quantita;
    }
}
```

La classe Java RichiestaRicovero

```
// File AppVisiteMediche/RichiestaRicovero/RichiestaRicovero.java
package AppVisiteMediche.RichiestaRicovero;
import AppVisiteMediche.Prescrizione.*;

public class RichiestaRicovero extends Prescrizione {
    private final String reparto;
    public RichiestaRicovero(String c, String r) {
        super(c);
        reparto = r;
    }
    public String getReparto() { return reparto; }
    public String toString() {
        return super.toString() + " reparto: " + reparto;
    }
}
```

La classe Java Visita

```
// File AppVisiteMediche/Visita.java
package AppVisiteMediche;
import Data.*;
import AppVisiteMediche.Prescrizione.*;
import insiemelista.*;
import java.util.*;

public class Visita {
    // CAMPI DATI
    // - RAPPRESENTAZIONE PROPRIETÀ
    private final Data data;
    private final Persona medico;
    private InsiemeListaOmogeneo indicazione;
    private TipoLinkAssistito assistito;
    // - RAPPRESENTAZIONE STATO
    private static final int prenotata = 1, disdetta = 2, effettuata = 3;
    private int stato_corrente = prenotata;
    // - RAPPRESENTAZIONE VINCOLI MOLTEPLICITÀ MINIMA MAGGIORE DI ZERO
    public static final int MIN_LINK_ASSISTITO = 1;
    // COSTRUTTORE
    public Visita(Data d, Persona m) {
        data = d;
        medico = m;
        indicazione = new InsiemeListaOmogeneo(TipoLinkIndicazione.class);
```

```
}
// GESTIONE ATTRIBUTI
public Persona getMedico() { return medico; }
public Persona getAssistito() { return assistito.getPersona(); }
public Data getData() { return data; }
// GESTIONE STATO
public void annullamento() {
    if (stato_corrente == prenotata)
        stato_corrente = disdetta;
}
public void effettuazione() {
    if (stato_corrente == prenotata)
        stato_corrente = effettuata;
}
public boolean isEffettuata() { return stato_corrente == effettuata; }
// GESTIONE ASSOCIAZIONI
// - ASSISTITO
public void inserisciLinkAssistito(AssociazioneAssistito a) {
    if (a != null) assistito = a.getLink();
}
public void eliminaLinkAssistito(AssociazioneAssistito a) {
    if (a != null) assistito = null;
}
public int quantiAssistiti() {
    if (assistito == null)
        return 0;
```

```
    else return 1;
}
public TipoLinkAssistito getLinkAssistito() throws EccezioneCardMin {
    if (assistito == null)
        throw new EccezioneCardMin("Cardinalita' minima violata");
    else
        return assistito;
}
// - INDICAZIONE
public void inserisciLinkIndicazione(AssociazioneIndicazione a) {
    if (a != null) indicazione.add(a.getLink());
}
public void eliminaLinkIndicazione(AssociazioneIndicazione a) {
    if (a != null) indicazione.remove(a.getLink());
}
public InsiemeListaOmogeneo getLinkIndicazione() {
    return (InsiemeListaOmogeneo)indicazione.clone();
}
// STAMPA
public String toString() {
    String result = "VISITA Data: " + data + ". Assistito: " +
        (assistito == null?" non noto":assistito.getPersona().toString()) +
        ". Medico: " + medico + ". Prescrizioni: ";
    Iterator it_prescrizioni = indicazione.iterator();
    while(it_prescrizioni.hasNext())
        result += ((TipoLinkIndicazione)it_prescrizioni.next()).
```

```
                getPrescrizione().toString() + " ";
        return result;
    }
}
```

La classe Java Persona

```
// File AppVisiteMediche/Persona.java
package AppVisiteMediche;
import Data.*;
import insiemelista.*;

public class Persona {
    private final String nome, cognome, codiceRegionale;
    private final Data nascita;
    private InsiemeListaOmogeneo assistito;
    public Persona(String no, String co, String cr, Data na) {
        nome = no;
        cognome = co;
        nascita = na;
        codiceRegionale = cr;
        assistito = new InsiemeListaOmogeneo(TipoLinkAssistito.class);
    }
    public String getCodiceRegionale() { return codiceRegionale; }
    public String getNome() { return nome; }
    public String getCognome() { return cognome; }
    public Data getNascita() { return nascita; }
    public void inserisciLinkAssistito(AssociazioneAssistito a) {
        if (a != null) assistito.add(a.getLink());
    }
    public void eliminaLinkAssistito(AssociazioneAssistito a) {
```

```
        if (a != null) assistito.remove(a.getLink());
    }
    public InsiemeListaOmogeneo getLinkAssistito() {
        return (InsiemeListaOmogeneo)assistito.clone();
    }
    public String toString() {
        return codiceRegionale + " " + cognome + " " + nome + " " + nascita;
    }

}
```

La classe Java TipoLinkAssistito

```
// File AppVisiteMediche/TipoLinkAssistito.java
package AppVisiteMediche;

public class TipoLinkAssistito {
    private final Persona laPersona;
    private final Visita laVisita;
    public TipoLinkAssistito(Persona p, Visita v)
        throws EccezionePrecondizioni {
        if (v == null || p == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laPersona = p; laVisita = v;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkAssistito b = (TipoLinkAssistito)o;
            return b.laPersona == laPersona &&
                b.laVisita == laVisita;
        }
        else return false;
    }
    public Visita getVisita() { return laVisita; }
    public Persona getPersona() { return laPersona; }
    public String toString() {
```

```
        return laVisita + " " + laPersona;
    }
}
```

La classe Java TipoLinkIndicazione

```
// File AppVisiteMediche/TipoLinkIndicazione.java
package AppVisiteMediche;
import AppVisiteMediche.Prescrizione.*;

public class TipoLinkIndicazione {
    private final Visita laVisita;
    private final Prescrizione laPrescrizione;
    public TipoLinkIndicazione(Visita v, Prescrizione p)
        throws EccezionePrecondizioni {
        if (v == null || p == null)  // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laVisita = v; laPrescrizione = p;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkIndicazione b = (TipoLinkIndicazione)o;
            return b.laPrescrizione == laPrescrizione &&
                b.laVisita == laVisita;
        }
        else return false;
    }
    public Visita getVisita() { return laVisita; }
    public Prescrizione getPrescrizione() { return laPrescrizione; }
```

```
    public String toString() {
        return laVisita.toString() + " " + laPrescrizione.toString();
    }
}
```

La classe Java AssociazioneAssistito

```
// File AppVisiteMediche/AssociazioneAssistito.java
package AppVisiteMediche;

public class AssociazioneAssistito {
    private AssociazioneAssistito(TipoLinkAssistito x) { link = x; }
    private TipoLinkAssistito link;
    public TipoLinkAssistito getLink() { return link; }
    public static void inserisci(TipoLinkAssistito y) {
        if (y != null && y.getVisita().quantiAssistiti() == 0) {
            AssociazioneAssistito k = new AssociazioneAssistito(y);
            y.getVisita().inserisciLinkAssistito(k);
            y.getPersona().inserisciLinkAssistito(k);
        }
    }
    public static void elimina(TipoLinkAssistito y) {
        try {
            if (y != null && y.getVisita().getLinkAssistito().equals(y)) {
                AssociazioneAssistito k = new AssociazioneAssistito(y);
                y.getVisita().eliminaLinkAssistito(k);
                y.getPersona().eliminaLinkAssistito(k);
            }
        }
        catch (EccezioneCardMin e) {
            System.out.println(e);
```

```
            }
        }
    }
```

La classe Java AssociazioneIndicazione

```
// File AppVisiteMediche/AssociazioneIndicazione.java
package AppVisiteMediche;
import AppVisiteMediche.Prescrizione.*;

public class AssociazioneIndicazione {
    private AssociazioneIndicazione(TipoLinkIndicazione x) { link = x; }
    private TipoLinkIndicazione link;
    public TipoLinkIndicazione getLink() { return link; }
    public static void inserisci(TipoLinkIndicazione y) {
        if (y != null && y.getPrescrizione().quantiIndicazione() == 0
            // CONTROLLO PRECONDIZIONE Visita.InserisciIndicazione
            && y.getVisita().isEffettuata()) {
            AssociazioneIndicazione k = new AssociazioneIndicazione(y);
            y.getVisita().inserisciLinkIndicazione(k);
            y.getPrescrizione().inserisciLinkIndicazione(k);
        }
    }
    public static void elimina(TipoLinkIndicazione y) {
        try {
            if (y != null && y.getPrescrizione().getLinkIndicazione().
                equals(y)) {
                AssociazioneIndicazione k = new AssociazioneIndicazione(y);
                y.getVisita().eliminaLinkIndicazione(k);
                y.getPrescrizione().eliminaLinkIndicazione(k);
```

```
            }
        }
        catch (EccezioneCardMin e) {
            System.out.println(e);
        }
    }
}
```

La classe Java EccezioneCardMin

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

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

La classe Java EccezionePrecondizioni

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

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;
    }
}
```

Realizzazione in Java dello use case

```
// File AppVisiteMediche/ControlliVisite.java
package AppVisiteMediche;
import insiemelista.*;
import java.util.*;
import AppVisiteMediche.Prescrizione.*;
import AppVisiteMediche.Farmaco.*;

public class ControlliVisite {
    public static float PercentualeVisiteConFarmaci(InsiemeListaOmogeneo iv) {
        int quante_visite = iv.size();
        int quante_visite_con_farmaci = 0;
        Iterator it_visite = iv.iterator();
        while(it_visite.hasNext()) {
            Visita v = (Visita)it_visite.next();
            InsiemeListaOmogeneo prescrizioni = v.getLinkIndicazione();
            Iterator it_prescrizioni = prescrizioni.iterator();
            boolean visita_con_farmaco = false;
            while(it_prescrizioni.hasNext() && !visita_con_farmaco) {
                Prescrizione p = ((TipoLinkIndicazione)it_prescrizioni.next()).
                    getPrescrizione();
                if (p.getClass().equals(Farmaco.class))
                    visita_con_farmaco = true;
            }
            if (visita_con_farmaco)
```

```
                quante_visite_con_farmaci++;
            }
            return (float)quante_visite_con_farmaci * 100 / quante_visite;
        }
        public static float QuantiFarmaci(Persona p, int a) {
            int result = 0;
            Iterator it_visite = p.getLinkAssistito().iterator();
            while(it_visite.hasNext()) {
                Visita v = ((TipoLinkAssistito)it_visite.next()).getVisita();
                if (v.getData().anno() == a) {
                    InsiemeListaOmogeneo prescrizioni = v.getLinkIndicazione();
                    Iterator it_prescrizioni = prescrizioni.iterator();
                    while(it_prescrizioni.hasNext()) {
                        Prescrizione pr =
                            ((TipoLinkIndicazione)it_prescrizioni.next()).
                            getPrescrizione();
                        if (pr.getClass().equals(Farmaco.class))
                            result += ((Farmaco)pr).getQuantita();
                    }
                }
            }
            return result;
        }
}
```