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 2 aprile 2004

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda le informazioni sui neonati, le loro madri, i medici che hanno prestato assistenza durante il parto, e le strutture sanitarie presso cui questi ultimi lavorano. Di ogni persona interessa la data di nascita, il nome ed il cognome. Esistono solamente tre categorie di persone: donne, neonati e medici. Ai fini di questa applicazione, queste categorie possono essere considerate disgiunte. Delle prime interessa il numero di gravidanze, dei secondi se vengono allattati artificialmente, dei terzi i titoli di studio di cui sono in possesso e l'anno in cui hanno conseguito ciascuno di essi. Dei medici interessa inoltre la struttura sanitaria presso cui lavorano. Delle strutture sanitarie interessano il codice regionale, l'indirizzo e l'anno in cui sono state autorizzate dalla Regione.

Di ogni neonato interessa la madre, il medico che ha prestato assistenza durante il parto, e se quest'ultimo è stato cesareo oppure no.

Requisiti (cont.)

Dopo la nascita un neonato può essere registrato all'anagrafe, dopo di che gli può essere attribuito un codice fiscale e successivamente può essere registrato presso la ASL.

Il Ministero della Sanità deve poter effettuare, come cliente della nostra applicazione, dei controlli sulla produttività delle strutture sanitarie. A questo scopo, si faccia riferimento ad uno use case che prevede le seguenti operazioni:

- dato un insieme di medici, calcolare il rapporto fra il numero di parti a cui hanno assistito e il numero di strutture sanitarie distinte in cui lavorano;

- dato un insieme di neonati, calcolare la percentuale di quelli registrati all'anagrafe.

Fase di analisi

Diagramma delle classi

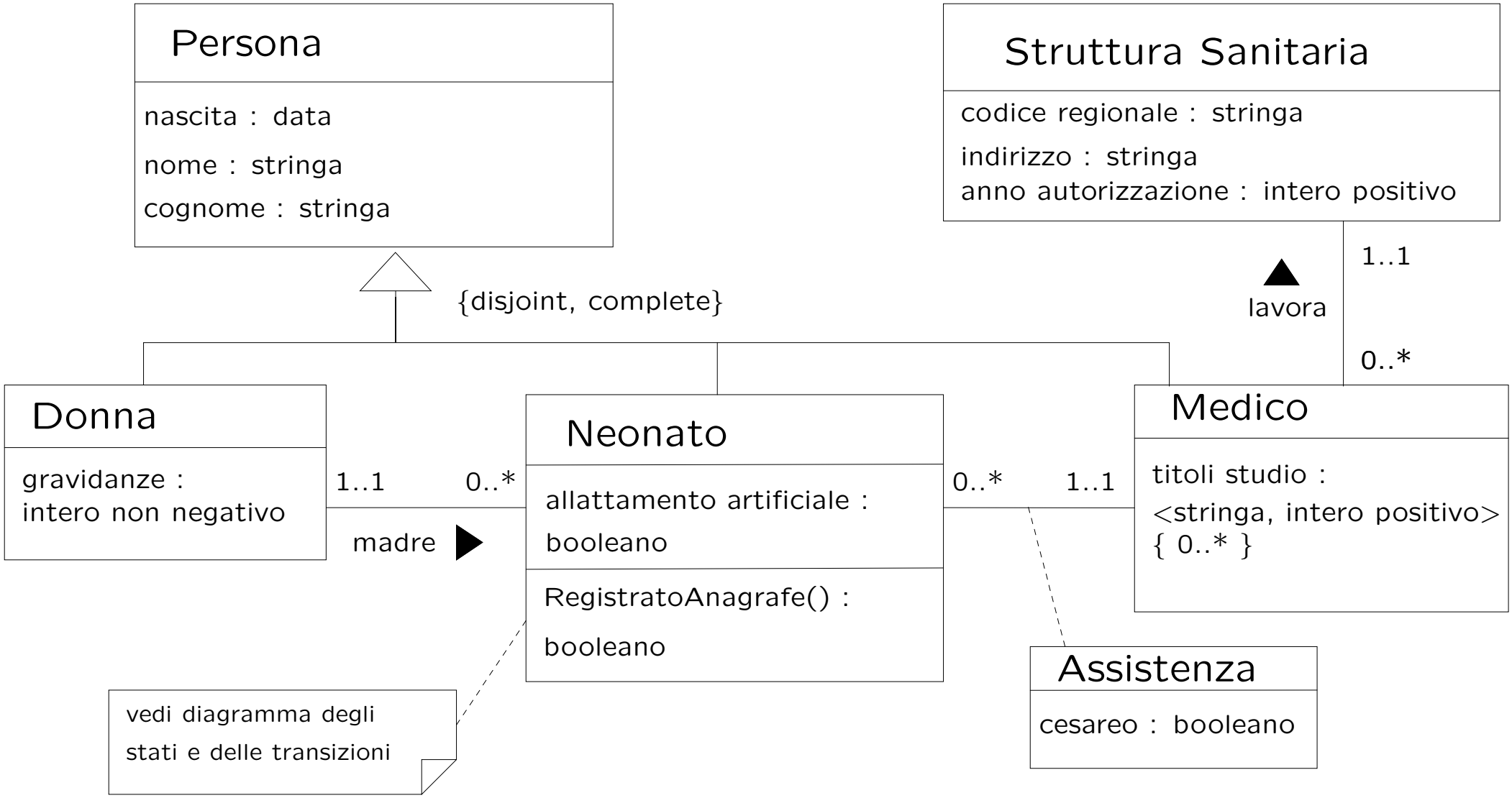
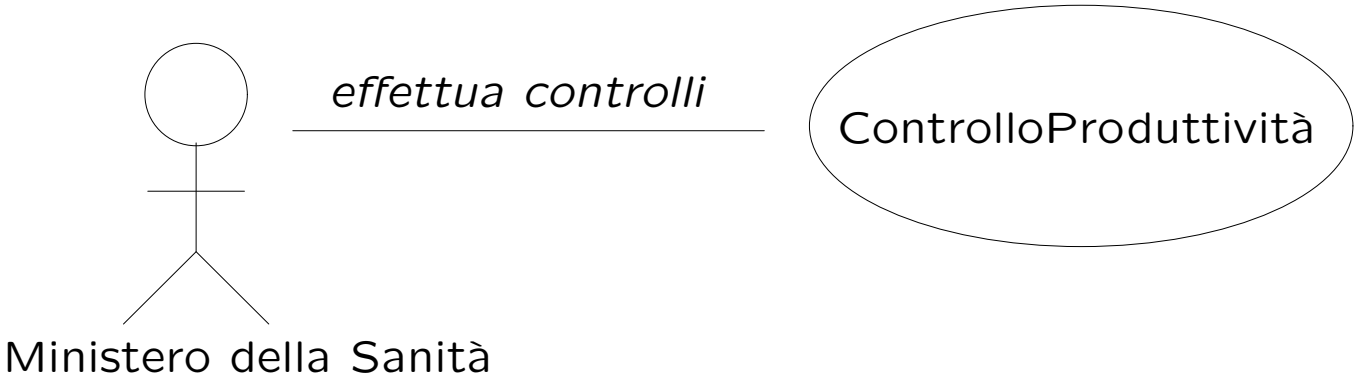


Diagramma degli stati e delle transizioni classe Neonato

Diagramma degli use case



Specifica della classe Neonato

InizioSpecificaClasse **Neonato**

RegistratoAnagrafe *(): booleano*

pre: nessuna

post: *result* è *true* se *this* è stato registrato all'anagrafe, è *false*, altrimenti

FineSpecifica

Specifica dello use case

InizioSpecificaUseCase **ControlloProduttività**

MediaPartiPerStruttura *(i: Insieme(Medico)): reale*

pre: *i* non è vuoto

post: *result* è il numero medio, per struttura sanitaria, di parti a cui i medici di *i* hanno assistito

PercentualeRegistratiAnagrafe *(i: Insieme(Neonato)): reale*

pre: *i* non è vuoto

post: *result* è la percentuale di neonati di *i* che sono stati registrati all'anagrafe

FineSpecifica

Fase di progetto

Algoritmi per le operazioni delle classi

Per l'operazione *Neonato.RegistratoAnagrafe()* adottiamo il seguente algoritmo:

```
return (this si trova in uno degli stati ‘‘registrato anagrafe’’,
                                         ‘‘ha codice fiscale’’, o
                                         ‘‘registrato ASL’’)
```

Algoritmi per le operazioni dello use-case

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

```
int parti = 0;
Insieme(StrutturaSanitaria) ins = insieme vuoto;
per ogni medico m di i
    parti += cardinalità(m.assistenza);
    ins.inserisci(m.lavora);
return parti / cardinalità(ins);
```

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

```
int registrati = 0;
per ogni neonato n di i
    se (n.RegistratoAnagrafe()) allora
        registrati++;
return registrati * 100 / cardinalità(i);
```

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 classe e use-case.

Associazione	Classe 1	ha resp.	Classe 2	ha resp.
madre	*Neonato*	SI[1,2]	*Donna*	NO
assistenza	*Neonato*	SI[1,2]	*Medico*	SI[3]
lavora	*Struttura Sanitaria*	NO	*Medico*	SI[1,2]

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`
intero non negativo	`int`
stringa	`String`
data	`Data`
reale	`float`
Insieme	`InsiemeListaOmogeneo`
<stringa, intero positivo>	`StringInt`

Si noti come nei casi "intero positivo" e "intero non negativo" 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 0..* delle associazioni,
- dei parametri dell'operazione dello use-case,
- delle variabili locali necessarie per l'algoritmo.

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

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

Per la rappresentazione del tipo <stringa, intero positivo> (coppia composta da una stringa e da un intero positivo), dobbiamo realizzare una apposita classe Java, `StringInt`.

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.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);
}
```

API per le strutture di dati (cont.)

```
// File AppNascite/Medico/StringInt.java
package Medico;

public class StringInt {
    public StringInt(String s, int i);
    public boolean equals(Object o);
    public String getString();
    public int getInt();
}
```

Notiamo che nella classe occorre effettuare l'overriding della funzione `equals()`, affinché effettui correttamente il test di uguaglianza. Ricordiamo infatti che tale funzione viene usata dai metodi della classe `InsiemeLista`. In particolare, affinché due valori del tipo <stringa, intero positivo> diano uguali, devono essere uguali in maniera profonda sia le stringhe sia gli interi.

Proprietà immutabili di classi UML

Riassumiamo le nostre scelte nella seguente tabella delle proprietà immutabili.

Classe UML	Proprietà immutabile
Persona	*nome*
	cognome
	nascita
Struttura Sanitaria	*codice regionale*
Neonato	*madre*
	assistenza

Sequenza di nascita degli oggetti

Poiché le responsabilità su *madre* e *lavora* sono singole, e la molteplicità è (nel verso della responsabilità) per entrambe 1..1, è ragionevole assumere che:

- quando nasce un oggetto Java corrispondente ad un neonato sia nota la madre;

- quando nasce un oggetto Java corrispondente ad un medico sia nota la struttura sanitaria in cui lavora.

Rappresentazione degli stati in Java

Per la classe UML *Neonato*, 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`
non registrato anagrafe	valore	1
registrato anagrafe	valore	2
ha codice fiscale	valore	3
registrato ASL	valore	4

Ulteriori considerazioni

- Il numero di gravidanze di una donna può solamente crescere.
- L'insieme dei titoli di studio di un medico può subire inserimenti, ma non cancellazioni.
- L'insieme dei link di tipo *assistenza* può subire inserimenti, ma non cancellazioni.
- Poiché l'associazione *Neonato.madre* è immutabile, e la madre è nota alla nascita del neonato, possiamo ignorare il fatto che la molteplicità minima sia maggiore di zero.

API delle classi Java progettate

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

```
public class Neonato extends Persona;
// COSTRUTTORE
    public Neonato(String no, String co, Data na, Donna ma);
// GESTIONE ATTRIBUTI
    public boolean getAllattamentoArtificiale();
    public void setAllattamentoArtificiale(boolean a);
// GESTIONE ASSOCIAZIONI
//   - assistenza
    public int quantiAssistenza();
    public TipoLinkAssistenza getAssistenza() throws EccezioneCardMin;
    public void inserisciLinkAssistenza(AssociazioneAssistenza a);
//   - madre
    public Donna getMadre();
// OPERAZIONI DI CLASSE
    public boolean RegistratoAnagrafe();
// GESTIONE EVENTI
    public void registrazione_anagrafe();
    public void attribuzione_codice_fiscale();
    public void registrazione_ASL();
// STAMPA
    public String toString();
}
```

Struttura dei file e dei package

```
+---AppNascite
|   |   StrutturaSanitaria.java
|   |   MainNascite.java
|   |   AssociazioneAssistenza.java
|   |   TipoLinkAssistenza.java
|   |   ControlloProduttivita.java
|   |   EccezioneCardMin.java
|   |
|   +---Medico
|   |       StringInt.java
|   |       Medico.java
|   |
|   +---Neonato
|   |       Neonato.java
|   |
|   +---Donna
|   |       Donna.java
|   |
|   \---Persona
|           Persona.java
|
+---Data
|       Data.java
|
\---insiemelista
        InsiemeLista.java
        InsiemeListaOmogeneo.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 singola e con vincoli di molteplicità 0..* e 1..1 (molteplicità minima diversa da zero);

   una associazione con attributi e a responsabilità doppia e con vincoli di molteplicità 0..* e 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.

Sommario classi Java relative a classi UML

`Persona:` astratta; nessuna responsabilità.

`Donna:` derivata; nessuna responsabilità.

`Neonato:` derivata; responsabilità su *madre* (singola) e su *assistenza* (doppia); ha diagramma stati e transizioni; gestione cardinalità minima maggiore di zero su *assistenza*.

`Medico:` derivata; responsabilità su *lavora* (singola) e su *assistenza* (doppia); gestione cardinalità minima maggiore di zero su *lavora*; gestione attributo con molteplicità 0..*.

`StrutturaSanitaria:` nessuna responsabilità.

La classe Java Persona

```
// File AppNascite/Persona/Persona.java
package AppNascite.Persona;
import Data.*;

public abstract class Persona {
    protected final String nome, cognome;
    protected final Data nascita;
    public Persona(String no, String co, Data na) {
        nome = no;
        cognome = co;
        nascita = (Data)na.clone();
    }
    public String getNome() { return nome; }
    public String getCognome() { return cognome; }
    public Data getNascita() { return (Data)nascita.clone(); }
    public String toString() {
        return nome + " " + cognome + ", nata/o il " + nascita;
    }
}
```

La classe Java Donna

```
// File AppNascite/Donna/Donna.java
package AppNascite.Donna;
import Data.*;
import AppNascite.Persona.*;

public class Donna extends Persona {
    protected int gravidanze;
    public Donna(String no, String co, Data na) { super(no,co,na); }
    public int getGravidanze() { return gravidanze; }
    public void incrementaGravidanze() { gravidanze++; }
    public String toString() {
        return super.toString() + ", " + gravidanze + " gravidanze";
    };

}
```

La classe Java Neonato

```
// File AppNascite/Neonato/Neonato.java
package AppNascite.Neonato;
import AppNascite.*;
import AppNascite.Persona.*;
import AppNascite.Donna.*;
import AppNascite.Medico.*;
import Data.*;

public class Neonato extends Persona {
    protected boolean allattamento_artificiale;
    protected final Donna madre;
    protected TipoLinkAssistenza assistenza;
    public static final int MIN_LINK_ASSISTENZA = 1;
    protected static final int non_registrato_anagrafe = 1,
        registrato_anagrafe = 2, ha_codice_fiscale = 3,
        registrato_ASL = 4;
    protected int stato_corrente = non_registrato_anagrafe;
    public Neonato(String no, String co, Data na, Donna ma) {
        super(no,co,na);
        madre = ma;
    }
    public int quantiAssistenza() {
        return assistenza==null?0:1;
    }
```

```
public TipoLinkAssistenza getAssistenza() throws EccezioneCardMin {
    if (quantiAssistenza() < MIN_LINK_ASSISTENZA)
        throw new EccezioneCardMin("Cardinalita' minima violata");
    else return assistenza;
}
public void inserisciLinkAssistenza(AssociazioneAssistenza a) {
    if (assistenza != null) return; // è immutabile
    if (a != null && a.getLink().getMedico() != null &&
        a.getLink().getNeonato() == this)
        assistenza = a.getLink();
}
public Donna getMadre() {
    return madre; }
public boolean getAllattamentoArtificiale() {
    return allattamento_artificiale; }
public void setAllattamentoArtificiale(boolean a) {
    allattamento_artificiale = a; }
public boolean RegistratoAnagrafe() {
    return (stato_corrente == registrato_anagrafe) ||
        (stato_corrente == ha_codice_fiscale) ||
        (stato_corrente == registrato_ASL);
};
public void registrazione_anagrafe() {
    if (stato_corrente == non_registrato_anagrafe)
        stato_corrente = registrato_anagrafe;
};
```

```
    public void attribuzione_codice_fiscale() {
        if (stato_corrente == registrato_anagrafe)
            stato_corrente = ha_codice_fiscale;
    };
    public void registrazione_ASL() {
        if (stato_corrente == ha_codice_fiscale)
            stato_corrente = registrato_ASL;
    };
    public String toString() {
        return super.toString() + ", stato corrente = " + stato_corrente +
            ", " + (allattamento_artificiale?"":"non ") +
            "allattata/o artificialmente" + ", figlio di " + madre;
    };
}
```

La classe Java EccezioneCardMin

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

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

La classe Java Medico

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

public class Medico extends Persona {
    protected StrutturaSanitaria lavora;
    protected InsiemeListaOmogeneo assistenza;
    protected InsiemeListaOmogeneo titoli_studio;
    public static final int MIN_LINK_LAVORA = 1;
    public Medico(String no, String co, Data na, StrutturaSanitaria ss) {
        super(no,co,na);
        lavora = ss;
        assistenza = new InsiemeListaOmogeneo(TipoLinkAssistenza.class);
        titoli_studio = new InsiemeListaOmogeneo(StringInt.class);
    }
    public void aggiungiTitoloDiStudio(StringInt s) {
        if (s != null) titoli_studio.add(s);
    }
    public InsiemeListaOmogeneo getTitoliDiStudio() {
        return (InsiemeListaOmogeneo)titoli_studio.clone();
    }
```

```
    public int quantiLavora() {
        return lavora==null?0:1;
    }
    public StrutturaSanitaria getLavora() throws EccezioneCardMin {
        if (quantiLavora() < MIN_LINK_LAVORA)
            throw new EccezioneCardMin("Cardinalita' minima violata");
        else return lavora;
    }
    public void setLavora(StrutturaSanitaria s) {
        lavora = s;
    }
    public void inserisciLinkAssistenza(AssociazioneAssistenza a) {
        if (a != null  && a.getLink().getNeonato() != null &&
            a.getLink().getMedico() == this)
            assistenza.add(a.getLink());
    }
    public InsiemeListaOmogeneo getLinkAssistenza() {
        return (InsiemeListaOmogeneo)assistenza.clone();
    }
    public String toString() {
        return super.toString() + lavora + titoli_studio + assistenza;
    };
}
```

La classe Java StringInt

```
// File AppNascite/Medico/StringInt.java
package AppNascite.Medico;

public class StringInt {
    private final String laString;
    private final int loInt;
    public StringInt(String s, int i) {
        laString = s; loInt = i;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            StringInt b = (StringInt)o;
            return b.laString != null &&
                b.laString.equals(laString) && b.loInt == loInt;
        }
        else return false;
    }
    public String getString() { return laString; }
    public int getInt() { return loInt; }
    public String toString() {
        return laString + ": " + loInt;
    };

}
```

La classe Java StrutturaSanitaria

```
// File AppNascite/StrutturaSanitaria.java
package AppNascite;

public class StrutturaSanitaria {
    private final String codice_regionale;
    private String indirizzo;
    private int anno_autorizzazione;
    public StrutturaSanitaria(String c, String i, int a) {
        codice_regionale = c;
        indirizzo = i ;
        anno_autorizzazione = a;
    }
    public String getCodiceRegionale() { return codice_regionale; }
    public String getIndirizzo() { return indirizzo; }
    public void setIndirizzo(String i) { indirizzo = i; }
    public int getAnnoAutorizzazione() { return anno_autorizzazione; }
    public void setAnnoAutorizzazione(int a) { anno_autorizzazione = a; }
    public String toString() {
        return "Struttura sanitaria: " +  codice_regionale + ", " +
            indirizzo + ", autorizzata nel " + anno_autorizzazione;
    }
}
```

La classe Java TipoLinkAssistenza

```
// File AppNascite/TipoLinkAssistenza.java
package AppNascite;
import AppNascite.Medico.*;
import AppNascite.Neonato.*;

public class TipoLinkAssistenza {
    private final Medico ilMedico;
    private final Neonato ilNeonato;
    private final boolean cesareo;
    public TipoLinkAssistenza(Medico m, Neonato n, boolean c) {
        ilMedico = m; ilNeonato = n; cesareo = c;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkAssistenza b = (TipoLinkAssistenza)o;
            return b.ilNeonato != null && b.ilMedico != null &&
                b.ilNeonato == ilNeonato && b.ilMedico == ilMedico;
        }
        else return false;
    }
    public Medico getMedico() { return ilMedico; }
    public Neonato getNeonato() { return ilNeonato; }
    public boolean getCesareo() { return cesareo; }
    public String toString() {
```

```
        return ilMedico.getNome() + ilMedico.getCognome() +
            ilNeonato.getNome() + ilNeonato.getCognome();
    }
}
```

La classe Java AssociazioneAssistenza

```
// File AppNascite/AssociazioneAssistenza.java
package AppNascite;

public class AssociazioneAssistenza {
    private AssociazioneAssistenza(TipoLinkAssistenza x) { link = x; }
    private TipoLinkAssistenza link;
    public TipoLinkAssistenza getLink() { return link; }
    public static void inserisci(TipoLinkAssistenza y) {
        if (y != null && y.getNeonato() != null && y.getMedico() != null &&
            y.getNeonato().quantiAssistenza() == 0) {
            AssociazioneAssistenza k = new AssociazioneAssistenza(y);
            y.getMedico().inserisciLinkAssistenza(k);
            y.getNeonato().inserisciLinkAssistenza(k);
        }
    }
}
```

Realizzazione in Java dello use case

```
// File AppNascite/ControlloProduttivita.java
package AppNascite;
import java.util.*;
import insiemelista.*;
import AppNascite.Medico.*;
import AppNascite.Neonato.*;

public final class ControlloProduttivita {
    private ControlloProduttivita() { }
    public static float MediaPartiPerStruttura(InsiemeListaOmogeneo i) {
        int parti = 0;
        InsiemeListaOmogeneo ins =
            new InsiemeListaOmogeneo(StrutturaSanitaria.class);
        Iterator it = i.iterator();
        while(it.hasNext()) {
            Medico m = (Medico)it.next();
            InsiemeListaOmogeneo assist =
                (InsiemeListaOmogeneo)m.getLinkAssistenza();
            parti+= assist.size();
            try {
                ins.add(m.getLavora());
            }
            catch (EccezioneCardMin e) {
                System.out.println(e);
```

```
            }
        }
        return (float)parti / ins.size();
    }
    public static float PercentualeRegistratiAnagrafe(InsiemeListaOmogeneo i) {
        int registrati = 0;
        Iterator it = i.iterator();
        while(it.hasNext()) {
            Neonato n = (Neonato)it.next();
            if(n.RegistratoAnagrafe())
                registrati++;
        }
        return (float)registrati * 100 / i.size();
    }
}
```