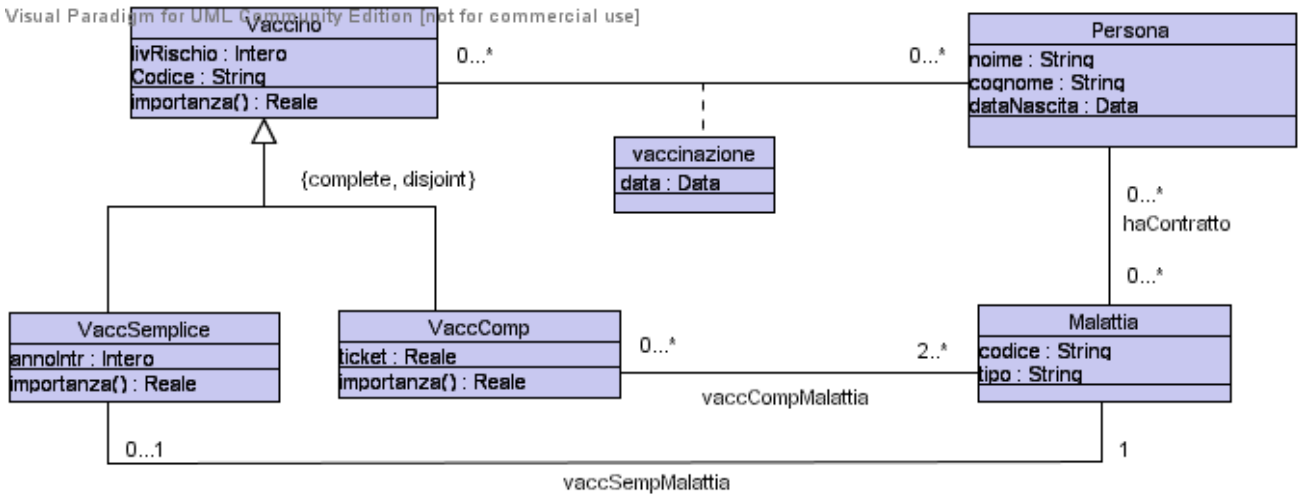
Università di Roma "La Sapienza", Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE" (Ing. Gestionale)
Giuseppe De Giacomo e Monica Scannapieco, A.A. 2003-04

SOLUZIONE ESAME DEL 18/07/2003

Roma, 11 marzo 2004

Diagramma delle classi UML



Specifica delel classi del diagramma

```
InizioSpecificaClasse Vaccino
  importanza(): Reale
    pre: nessuna
    post: result dipende dalla particolare sottoclasse di this.
FineSpecificaClasse


InizioSpecificaClasse VaccinoSemplice
  importanza(): Reale
    pre: nessuna
    post: result e' pari al numero di Persone che hanno un link
      HaContratto con l'oggetto Malattia legato a this da un link di
      tipo vaccinoSempliceMalattia.
FineSpecificaClasse


InizioSpecificaClasse VaccinoComposto
  importanza(): Reale
    pre: nessuna
    post: result e' pari alla somma del numero di Persone che hanno un link
      HaContratto con almeno un oggetto Malattia legato a this da un link di
      tipo vaccinoCompostoMalattia, diviso per il numero di tali link.
FineSpecificaClasse
```


Diagramma e specifica degli use case

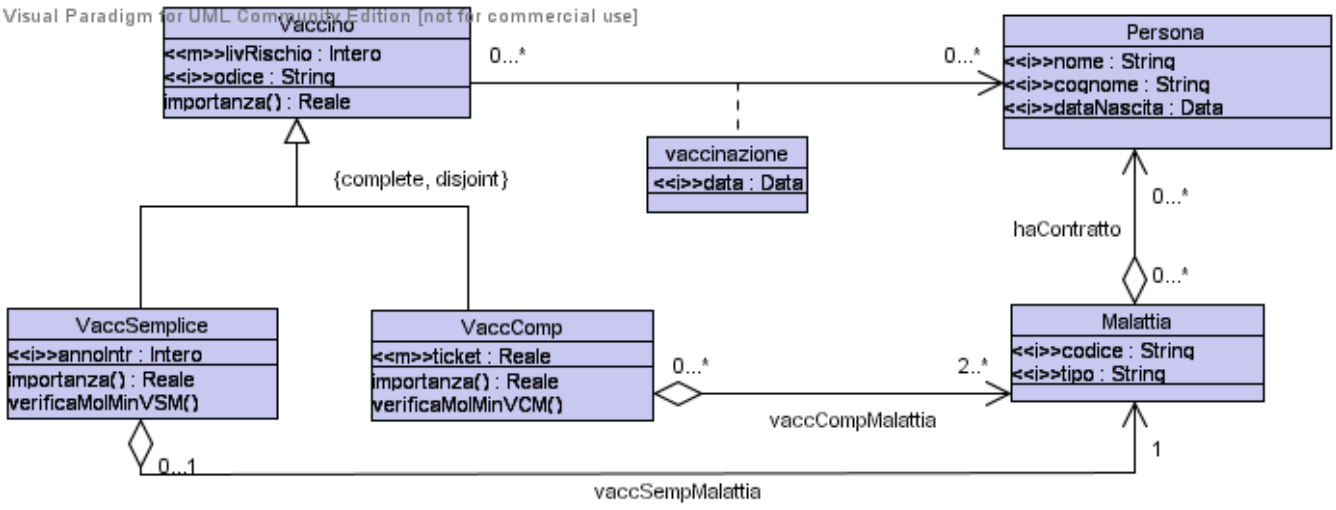
```
//UML-usecase-diagram.txt

        _
       |_|              /--------------\
       -|-  ------------| Controllo    |
       / \              | Vaccini      |
   MinisteroSalute      \--------------/


InizioSpecificaUseCase ControlloVaccini

  controllo(d: Data, n: Intero, s: Insieme(Vaccino)): Insieme(Persona)
     pre:  nessuna

     post: result e' l'insieme delle persone p che sono legate da un
           link l di tipo Vaccinazione ad un oggetto vp nell'insieme s
           di classe VaccinoSemplice per cui vale l.data < d e
           vp.importanza() > n.

FineSpecifica
```

Responsabilità sulle associazioni

Dalla specifica dello use case, delle operazioni e da alcune note nei requisiti emerge che:

- *Vaccino* ha responsabilità su *vaccinazione*
- *VaccSemplice* ha responsabilità su *vaccinoSempliceMalattia*
- *VaccComposto* ha responsabilità su *vaccinoCompostoMalattia*
- *Malattia* ha responsabilità su *haContratto*

Diagramma delle classi realizzativo



La classe Java TipoLinkVaccinazione

```
// File TipoLinkVaccinazione.java

public class TipoLinkVaccinazione {

  //Rappr. componenti della tupla
  private final Vaccino vaccino;
  private final Persona persona;
  private final Date data;

  //Costruttore
  public TipoLinkVaccinazione(Vaccino x, Persona y,Date d) {
    vaccino = x;
    persona = y;
    data = d;
  }

  //Accesso alle componenti
  public Vaccino getVaccino() { return vaccino; }
  public Persona getPersona() { return persona; }

  //Accesso agli attributi
  public Date getData() { return data; }

  //Overriding di funzioni speciali ereditate da Object
  //Nota TipoLinkRelativoA e' un tipo non una classe:
  //va rdefinito equals per effettuare test di uguaglianza profonda
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkRelativoA b = (TipoLinkRelativoA)o;
```



```
      return b.vaccino != null && b.persona != null &&
        b.vaccino.equals(vaccino) &&
        b.persona.equals(persona);
    }
    else return false;
  }
}
```

La classe Java Vaccino

```
// File Vaccino.java
import java.util.*;

abstract public class Vaccino {
  //Rappr. attributi
  private int livello;
  private final String codice;

  //Rappr. associazione "vaccinazione"
  private Set insieme_link;

  //Costruttore
  public Vaccino(Livello l, String c) {
    livello = c;
    codice = c;
    insieme_link = new InsiemeArrayOmogeneo(TipoLinkVaccinazione.class);
  }

  //Accesso agli attributi
  public String getLivelloRischio() { return livello; }
  public void setLivelloRischio(int l) { livello  = l; }
  public String getCodice() { return codice; }

  //Accesso alla associazione "vaccinazione"
  public Iterator getLinkVaccinazione() {
    return insieme_link.iterator();
  }

  public void inserisciLinkVaccinazione(TipoLinkVaccinazione t) {
```



```
    if (t != null  && t.getVaccino() == this &&
        t.getPersona() != null  && t.getData() != null)
      insieme_link.add(t);
    //else throw new RuntimeException("Inserimento impossibile!");
  }
  public void eliminaLinkVaccinazione(TipoLinkVaccinazione t) {
    if (t != null)
      insieme_link.remove(t);
    //else throw new RuntimeException("Eliminazione impossibile!");
  }

  // operazioni
  public double importanza();

}
```

La classe Java VaccinoSemplice

```
// File VaccinoSemplice.java

public class VaccinoSemplice extends Vaccino {
  private final int annoIntr;

  private Malattia mal;

  public VaccinoSemplice(int l, String c, int a) {
    super(l,c);
    annoIntr = a;
  }

  public int getAnnoIntroduzione() { return annoIntr; }

  public boolean verificaMoltMinMalattia() {
    return mal != null;
  }

  public Malattia getMalattia() {
    if (verificaMoltMinMalattia() == false)
      throw new RuntimeException( "VaccinoSemplice: partecipazione obbligatoria in vaccinoSempMalattia violata
      return mal;
  }

  public void setMalattia(Malattia m) {
    mal = m;
  }

  public double importanza() {
```



```
    double malati = 0;
    Iterator it = getMalattia().getPersone();
    while (it.hasNext()) {
      it.next();  //prendi il prossimo (e non farci nulla)
      malati = malati + 1;
    }
    return malati;
  }

}
}
```

La classe Java VaccinoComposto

```
// File VaccinoComposto.java

public class VaccinoComposto extends Vaccino {
  private double ticket;

  private Set malattie;

  public VaccinoComposto(int l, String c, double t) {
    super(l,c);
    ticket = t;
    malattie = new InsiemeArrayOmogeneo(Malattia.class);
  }

  public int getTicket() { return ticket; }
  public void setTicket(double t) { ticket = t; }

  public boolean verificaMoltMinMalattia() {
    return malattie.size() >= 2;
  }

  public Iterator getMalattie() {
    if (verificaMoltMinMalattia() == false)
      throw new RuntimeException(
        "VaccinoComposto: partecipazione obbligatoria in vaccinoCompMalattia violata");
      return malattie.itarator();
  }

  public void inserisciMalattia(Malattia m) {
    if (m != null) malattie.add(m);
```



```
  }
  public void eliminaMalattia(Malattia m) {
    if (m != null) malattie.remove(m);
  }

  public double importanza() {
    double malati = 0;
    int malattie = 0
    Iterator it = getMalattie();
    while(it.hasNext()) {
      Malattia m = (Malattia)it.next();
      malattie = malattie + 1;
      Iterator it2 = m.getPersone();
      while (it2.hasNext()) {
        it2.next();
        malati = malati + 1;
      }
    }
    return malati/malattie;
  }
}
```

La classe Java Malattia

```
// File Malattia.java

public class Malattia {
  private final String codice;
  private final String tipo;

  private Set persone;

  public Malattia(String c, double t) {
    codice = c
    tipo = t;
    persone = new InsiemeArrayOmogeneo(Persone.class);
  }

  public int getCodice() { return codice; }
  public int getTipo() { return tipo; }

  public Iterator getPersone() {
      return malattie.itarator();
  }

  public void inserisciPersona(Persona p) {
    if (p != null) persoan.add(p);
  }

  public void eliminaPersona(Persona p) {
    if (p != null) persona.remove(p);
  }
}
```


La classe Java Persona

```
// File Persona.java

public class Persona {
  //Rappr. attributi
  private final String nome;
  private final String cognome;
  private final Date nascita;

  //Costruttore
  public Persona(String n, String c, Date d) {
    nome = n;
    cognome = c;
    data = d;
  }

    //Accesso agli attributi
  public String getNome() { return nome; }
  public String getCognome() { return cognome; }
  public Date getDataNascita() { return data; }

}
```

Realizzazione in Java dello use case

```
// File ControlliVaccini.java
import java.util.*;

public class ControlliVaccini {

  public static Set controllo(Date oggi, int n, Set insVaccini) {
    Set result  = new InsiemeArrayOmogeno(Persona.class);
    Iterator it = insVaccini.iterator();
    while(it.hasNext()) {
      Vaccino v = (Vaccino)it.next();
      if (v istanceof VaccinoSemplice && v.importanza() > n) {
        Iterator it2 = v.getLinkVaccinazioni();
      while(it2.hasNext()) {
        TipoLinkVaccinazione t = (TipoLinkVaccinazione)it2.next();
        if (t.getData().before(oggi)) result.add(t.getPersona());
      }
    }
    return result;
  }

}
```


InsiemeArray

```
//Realizzazione dell'interfaccia Set con un array invece che con una lista

import java.util.*;
import java.lang.reflect.Array;

public class InsiemeArray implements Set, Cloneable {


  // campi dati
  protected Object[] array;
  protected static final int dimInit = 10; //dim. iniz. array

  protected int cardinalita;

  // costruttori
  public InsiemeArray() {
    array = new Object[dimInit];
    cardinalita = 0;
  }

  // funzioni proprie della classe
  // (realizzazione delle funzioni di Set)

  // basic operations
  public int size() {
    return cardinalita;
  }

  public boolean isEmpty() {
```

```
  return cardinalita == 0;
}

public boolean contains(Object e) {
  return appartiene(e);
}

public boolean add(Object e) {
  if (appartiene(e)) return false;
  else {
    if (cardinalita == array.length) {// raddoppia array
      Object[] aux = new Object[array.length*2];
      for(int i = 0; i < array.length; i++)
        aux[i] = array[i];
      array = aux;
    }
    array[cardinalita] = e;
    cardinalita++;
    return true;
  }
}

public boolean remove(Object e) {
  if (!appartiene(e)) return false;
  else {
    int k = 0;                  // trova l'elemento
    while (!array[k].equals(e))
      k++;
    for(int i = k; i < cardinalita-1; i++) // sposta di una poss
       array[i] = array[i+1];              // verso il basso gli
                                           // elementi dell'array
```

```
    cardinalita--;

    // rimpicciolisci l'array se e' il caso
    if (cardinalita > dimInit && cardinalita < array.length/3) {
      Object[] aux = new Object[array.length/2];
      for(int i = 0; i < cardinalita; i++)
        aux[i]=array[i];
      array = aux;
    }
    return true;
  }
}

public Iterator iterator() {
  return new IteratorInsiemeArray(this);
}


// bulk operations
public boolean containsAll(Collection c) {
  Iterator it = c.iterator();
  while (it.hasNext()) {
    Object e = it.next();
    if (!contains(e)) return false;
  }
  return true;
}

public boolean addAll(Collection c){
  throw new UnsupportedOperationException("addlAll() non e' supportata");
}
```

```
public boolean removeAll(Collection c) {
  throw new UnsupportedOperationException("removeAll() non e' supportata");
}
public boolean retainAll(Collection c) {
  throw new UnsupportedOperationException("retainAll() non e' supportata");
}
public void clear() {
  throw new UnsupportedOperationException("clear() non e' supportata");
}

// array operations
public Object[] toArray() {
  Object[] a = new Object[size()];
  int i = 0;
  Iterator it = iterator();
  while (it.hasNext()) {
    a[i] = it.next();
    i++;
  }
  return a;
}

public Object[] toArray(Object[] a) {
  if (a.length < size()) {
    //prendi il tipo degli elementi di a
    Class elemClass = a.getClass().getComponentType();
    //costruisci un array il cui tipo degli elementi e' quello in a
    a = (Object[])Array.newInstance(elemClass,size());
  }
  //riempi l'array con gli elementi della collezione
  int i = 0;
```

```
  Iterator it = iterator();
  while (it.hasNext()) {
    a[i] = it.next();
    i++;
  }
  for (; i < a.length; i++)
    a[i] = null;
  return a;
}


// funzioni speciali ereditate da Object
public boolean equals(Object o) {
  if (o != null && getClass().equals(o.getClass())) {
    InsiemeArray ins = (InsiemeArray)o;
    if (cardinalita != ins.cardinalita) return false;
    // ins non ha la cardinalita' giusta
    else {
      // verifica che gli elementi nella lista siano gli stessi
      for(int i = 0; i < ins.cardinalita; i++)
        if (!appartiene(ins.array[i])) return false;
      return true;
    }
  }
  return false;
}

public Object clone() {
  try {
    InsiemeArray ins = (InsiemeArray) super.clone();
    ins.array = new Object[array.length];
```

```
      for(int i = 0; i < cardinalita; i++)
        ins.array[i]=array[i];
      return ins;
    } catch(CloneNotSupportedException e) {
      throw new InternalError(e.toString());
    }
  }
  public String toString() {
    String s = "{ ";
    for(int i = 0; i < cardinalita; i++)
      s = s + array[i] + " ";
    s = s + "}";
    return s;
  }

  // funzioni ausiliarie

  protected boolean appartiene(Object e) {
    for(int i = 0; i < cardinalita; i++)
      if (array[i].equals(e)) return true;
    return false;
  }
}
```

InsiemeArrayOmogeneo

```
public class InsiemeArrayOmogeneo extends InsiemeArray {

  // campi dati
  protected Class elemClass;

  // costruttori
  public InsiemeArrayOmogeneo(Class cl) {
    super();
    elemClass = cl;
  }
  public InsiemeArrayOmogeneo() {
    this(Object.class);
  }

  // overriding delle funzioni proprie della classe
  // (realizzazione delle funzioni di Set)

  // overriding delle basic operations

  public boolean contains(Object e) {
    if (!elemClass.isInstance(e)) return false;
    else return super.contains(e);
  }

  public boolean add(Object e) {
    if (!elemClass.isInstance(e)) return false;
    else return super.add(e);
  }
```

```
  public boolean remove(Object e) {
    if (!elemClass.isInstance(e)) return false;
    else return super.remove(e);
  }

  // overridinig delle bulk operations

  // overriding delle array operations

  // overriding delle funzioni speciali ereditate da Object

  public boolean equals(Object o) {
    if (super.equals(o)) {
      InsiemeArrayOmogeneo ins = (InsiemeArrayOmogeneo)o;
      return elemClass.equals(ins.elemClass);
    }
    return false;
  }

  public Object clone() {
    InsiemeArrayOmogeneo ins = (InsiemeArrayOmogeneo) super.clone();
    return ins;
    //oppure semplicemente: return super.clone();
  }

  // funzioni ausiliarie

}
```

IteratorInsiemeArray

```
import java.util.*;

class IteratorInsiemeArray implements Iterator {
// nota non e' pubblica, cioe' ne impediamo l'uso diretto da parte
// dei clienti (che quindi possono usare solo l'interfaccia Iterator)

  private InsiemeArray insiemeArray;
  private int indice;

  public IteratorInsiemeArray(InsiemeArray ia) {
    insiemeArray = ia;
    indice = 0;
  }

  // Realizzazione funzioni di Itarator
  public boolean hasNext() {
    return indice < insiemeArray.cardinalita;
  }

  public Object next() {
    Object e = insiemeArray.array[indice];
    indice++;
    return e;
  }

  public void remove() {
    throw new UnsupportedOperationException("remove() non e' supportata");
  }
}
```