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 15 aprile 2005

SOLUZIONE

Requisiti

L'applicazione da progettare riguarda la gestione degli articoli sottomessi ad una conferenza scientifica. Le persone, di cui interessa nome, cognome ed indirizzo di posta elettronica, possono essere autori o revisori, ma non entrambi. Gli articoli, di cui interessa il titolo e la dimensione in kiloByte del file, sono scritti da almeno un autore. Ad ogni articolo viene assegnato un revisore *senior* e almeno due revisori *junior*. Un revisore, di cui interessa la nazionalità, non può essere contemporaneamente *senior* e *junior*. Su ciascun articolo assegnato loro, i primi esprimono un giudizio positivo o negativo, mentre i secondi assegnano un voto compreso fra 0 e 9.

Un articolo, una volta sottomesso, si trova sotto esame e può essere candidato all'accettazione o al rifiuto. Nel primo caso può essere definitivamente accettato, oppure tornare sotto esame. Il secondo caso è analogo. Solo ad articoli sotto esame possono essere assegnati revisori.

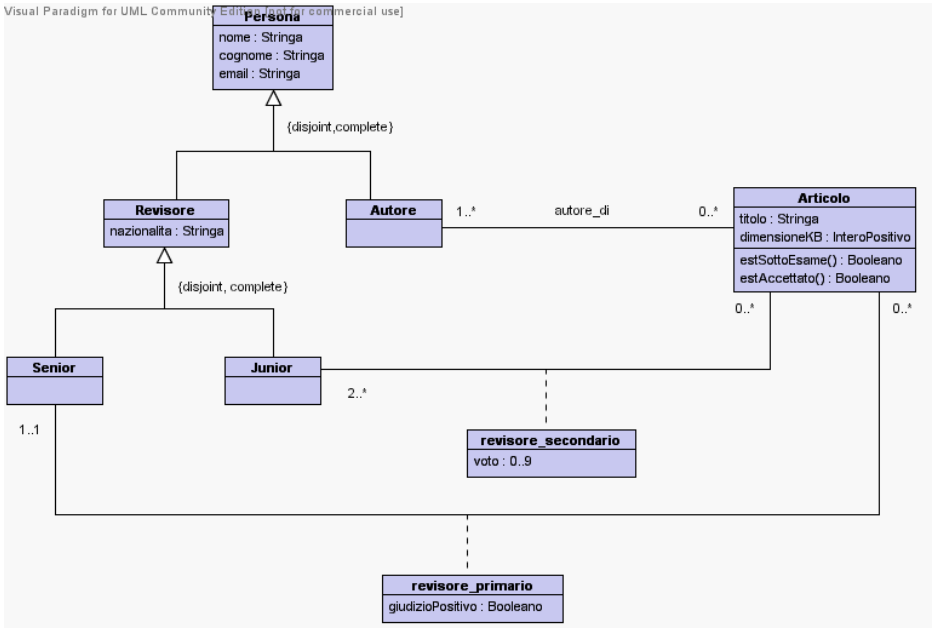
Requisiti (cont.)

Il comitato di indirizzo della conferenza è interessato, come cliente della nostra applicazione, ad effettuare i seguenti controlli:

- data una persona, sapere se è autore di almeno un articolo con giudizio negativo o con media dei voti inferiore a 4, che è stato accettato;
- dato un articolo, sapere se è stato assegnato ad almeno due revisori della stessa nazionalità.

Fase di analisi

Diagramma delle classi



Commento sul diagramma delle classi

La necessità di disporre nella classe *Articolo* di metodi per conoscere se un articolo è sotto esame e se è stato accettato si evince, rispettivamente, dal vincolo che solo articoli sotto esame possono essere assegnati a revisori e dal fatto che la prima funzionalità dello use case richiede verificare se un articolo è stato accettato o meno.

Diagramma degli stati e delle transizioni classe Articolo

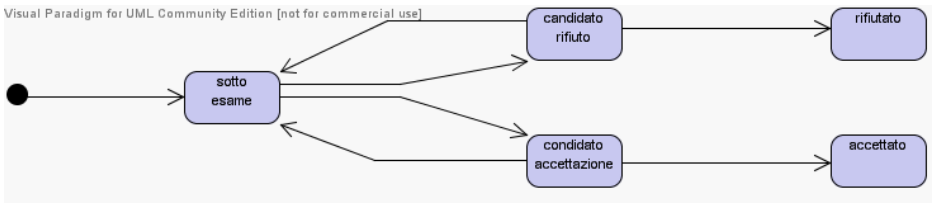


Diagramma degli use case

Specifica della classe Articolo

InizioSpecificaClasse Articolo

estSottoEsame *(): Booleano*

pre: nessuna

post: *result* è *true* se lo stato dell'articolo è "sotto esame", *false* altrimenti.

estAccettato *(): Booleano*

pre: *this.estSottoEsame() = false*

post: *result* è *true* se o stato dell'articolo è "accettato", *false* altrimenti.

FineSpecifica

Specifica dello use case

InizioSpecificaUseCase Controlli

autoreConRevisioneProblematica *(p: Persona): Booleano*

pre: nessuna

post: *result* è pari a *true* se *p* è un autore di un articolo con giudizio negativo o con media dei voti inferiore a 4, che è stato accettato; *false* altrimenti.

articoloRevisionatoStessaNazionalita *(a: Articolo): Booleano*

pre: nessuna

post: *result* è *true* se è stato assegnata ad almeno due revisori della stessa nazionalità; *false* altrimenti.

FineSpecifica

Fase di progetto

Algoritmi per le operazioni delle classi

Adottiamo i seguenti algoritmi:

- Per l'operazione **estSottoEsame** della classe *Articolo*:

```
se (stato corrente = ‘‘sotto esame’’) return true;
altrimenti return false;
```

- Per l'operazione **estAccettato** della classe *Articolo*:

```
se (stato corrente = ‘‘accettato’’) return true;
altrimenti return false;
```


Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **autoreConRevisioneProblematica**:

```
se (p non è istanza di Autore) return false;
altrimenti {
  per ogni link l di tipo autore_di in cui p è coinvolto {
     a = l.Articolo;
     se (a.estAccettato()
         &&
         (!ll.giudizioPositivo, dove ll è il link di tipo
                revisore_primario in cui a è coinvolto ||
         votomedio(a) < 4)) return true;
  }
  return false
}

dove votomedio(a) è calcolato come segue:

Reale sum = 0;
Intero cont = 0;
```

```
per ogni link l di tipo revisore_secondaro in cui a è coinvolto {
  sum = sum + l.voto;
  cont++;
}
return sum/cont;
```

Algoritmi per le operazioni degli use case (cont.)

- Per l'operazione **articoloRevisionatoStessaNazionalita**:

```
sia l il link di tipo revisore_primario a cui partecipa a;
Insieme naz  = {l.Senior.nazionalita};
per ogni ll di tipo revisore_secondario a cui partecipa a {
   Stringa n = ll.Junior.nazionalita;
   se (n appartiene naz) return true;
   altrimenti naz = naz + {n};
}
return false;
```

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.
autore_di	*Autore*	SÌ[2]
	Articolo	SÌ[3]
revisore_primario	*Articolo*	SÌ[2,3]
	Senior	NO
revisore_secondario	*Articolo*	SÌ[2,3]
	Junior	NO

Strutture di dati

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

- dei vincoli di molteplicità x..* delle associazioni,

- delle variabili locali necessarie per vari algoritmi.

Per fare ciò, utilizzeremo `Set` e `HashSet` di Java 1.5.

Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
Stringa	`String`
Booleano	`boolean`
InteroPositivo	`int`
0..9	`int`
Insieme	`HashSet`

Per tenere conto del fatto che, nei casi "InteroPositivo" e "0..9" 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
Revisore	*nazionalita*
Articolo	*titolo*

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

Altre considerazioni

Sequenza di nascita degli oggetti: Non abbiamo vincoli particolare se non quelli dettati dalle molteplicità: in particolare possiamo assumere che i revisori, (senior e junior) e gli autori siano già stati creati quando nascono gli oggetti articolo.

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 *Articolo*, 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`
sotto esecuzione	valore	1
candidato rifiuto	valore	2
candidato accettazione	valore	3
rifiutato	valore	4
accettato	valore	5

API delle classi Java progettate

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

```
public abstract class Persona {
// COSTRUTTORE
    public Persona(String nome, String cognome, String email)
// GESTIONE ATTRIBUTI
    public String getNome()
    public String getCognome()
    public String getEmail()
    public void setEmail(String e)
// STAMPA
    public String toString()
}
```


Fase di realizzazione

Considerazioni

La realizzazione in questo caso è assolutamente standard e non richiede particolari accorgimenti.

La classe Java Persona

```
// File Sottomissioni/Persona/Persona.java
package Sottomissioni.Persona;
import Sottomissioni.*;
import java.util.*;

public abstract class Persona {
  private final String nome;
  private final String  cognome;
  private String email;
  public Persona(String n, String c, String e) {
    nome = n;
    cognome = c;
    email = e;
  }
  public String getNome(){ return nome; }
  public String getCognome(){ return cognome; }
  public String getEmail(){ return email; }
  public void setEmail(String e){ email = e; }

  public String toString() {
    return  nome + " " + cognome + " " + email;
  }
}
```

La classe Java Revisore

```
// File Sottomissioni/Revisore/Revisore.java
package Sottomissioni.Revisore;
import Sottomissioni.Persona.*;

public abstract class Revisore extends Persona {
  private final String nazionalita;
  public Revisore(String n, String c, String e, String naz) {
    super(n,c,e);
    nazionalita = naz;
  }
  public String getNazionalita(){ return nazionalita; }
  public String toString() {
    return  super.toString() + " -- revisore";
  }
}
```


La classe Java Senior

```
// File Sottomissioni/Senior/Senior.java
package Sottomissioni.Senior;
import Sottomissioni.Revisore.*;

public class Senior extends Revisore {
  public Senior(String n, String c, String e,String naz) {
    super(n,c,e,naz);
  }
  public String toString() {
    return super.toString() + " senior";
  }
}
```

La classe Java Junior

```
// File Sottomissioni/Junior/Junior.java
package Sottomissioni.Junior;
import Sottomissioni.Revisore.*;

public class Junior extends Revisore {
  public Junior(String n, String c, String e, String naz) {
    super(n,c,e,naz);
  }
  public String toString() {
    return super.toString() + " junior";
  }
}
```


La classe Java Autore

```
// File Sottomissioni/Autore/Autore.java
package Sottomissioni.Autore;
import Sottomissioni.*;
import Sottomissioni.Persona.*;
import java.util.*;

public class Autore extends Persona {
  private HashSet<TipoLinkAutoreDi> autoreDi;
  public Autore(String n, String c, String e) {
    super(n,c,e);
    autoreDi = new HashSet<TipoLinkAutoreDi>();
  }
  public void inserisciLinkAutoreDi(AssociazioneAutoreDi a) {
    if (a != null) autoreDi.add(a.getLink());
  }
  public void eliminaLinkAutoreDi(AssociazioneAutoreDi a) {
    if (a != null) autoreDi.remove(a.getLink());
  }
  public Set<TipoLinkAutoreDi> getLinkAutoreDi() {
    return (HashSet<TipoLinkAutoreDi>)autoreDi.clone();
  }
  public String toString() {
    return  super.toString() + " -- autore";
  }
}
```

La classe Java Articolo

```
// File Sottomissioni/Articolo.java
package Sottomissioni;
import java.util.*;

public class Articolo {
  private final String titolo;
  private int dimensioneKB;

  private TipoLinkRevisorePrimario revisorePrimario;
  private HashSet<TipoLinkRevisoreSecondario> revisoriSecondari;
  private HashSet<TipoLinkAutoreDi> autoreDi;

  private static final int MIN_REVISORI_SECONDARI = 2;

  private static final int SOTTO_ESAME = 1,
    CANDIDATO_RIFIUTO = 2, CANDIDATO_ACCETTAZIONE = 3,
    RIFIUTATO = 4, ACCETTATO = 5;

  protected int stato_corrente;

  public Articolo(String t, int d) throws EccezionePrecondizioni {
    if (d < 0)
      throw new EccezionePrecondizioni("E' richiesto un intero positivo");
    titolo = t;
```



```
    dimensioneKB = d;
    autoreDi = new HashSet<TipoLinkAutoreDi>();
    revisorePrimario = null;
    revisoriSecondari = new HashSet<TipoLinkRevisoreSecondario>();
    stato_corrente = SOTTO_ESAME;
  }
  public String getTitolo() { return titolo; }
  public int getDimensioneKB() { return dimensioneKB; }
  public void setDimensioneKB(int d) { return dimensioneKB = d; }

  public void inserisciLinkAutoreDi(AssociazioneAutoreDi a) {
    if (a != null) autoreDi.add(a.getLink());
  }
  public void eliminaLinkAutoreDi(AssociazioneAutoreDi a) {
    if (a != null) autoreDi.remove(a.getLink());
  }
  public Set<TipoLinkAutoreDi> getLinkAutoreDi() {
    return (HashSet<TipoLinkAutoreDi>)autoreDi.clone();
  }

  public boolean presenteRevisorePrimario() {
    return revisorePrimario != null;
  }
  public void inserisciLinkRevisorePrimario(TipoLinkRevisorePrimario t) {
    if (t != null && t.getArticolo()==this &&
        !presenteRevisorePrimario()) revisorePrimario = t;
```

```
}
public void eliminaLinkRevisorePrimario() {
  revisorePrimario = null;
}
public TipoLinkRevisorePrimario getLinkRevisorePrimario()
  throws EccezioneMolteplicita {
  if (!presenteRevisorePrimario())
    throw new EccezioneMolteplicita("Molteplicita' minina non rispettata");
  return revisorePrimario;
}

public int quantiRevisoriSecondari() {
  return revisoriSecondari.size();
}
public void inserisciLinkAutoreDi(TipoLinkRevisoreSecondario t) {
  if (t != null && estSottoEsame()) revisoriSecondari.add(t);
}
public void eliminaLinkAutoreDi(TipoLinkRevisoreSecondario t) {
  if (t != null && estSottoEsame()) revisoriSecondari.remove(t);
}
public Set<TipoLinkRevisoreSecondario> getLinkRevisoriSecondari()
  throws EccezioneMolteplicita {
  if (quantiRevisoriSecondari() < MIN_REVISORI_SECONDARI)
    throw new EccezioneMolteplicita("Molteplicita' minina non rispettata");
  return (HashSet<TipoLinkRevisoreSecondario>)revisoriSecondari.clone();
}
```

```
public boolean estSottoEsame() {
  return stato_corrente == SOTTO_ESAME;
}
public boolean estAccettato() {
  return stato_corrente == ACCETTATO;
}
public void rifiutoProvvisorio() {
  if (stato_corrente == SOTTO_ESAME)
    stato_corrente = CANDIDATO_RIFIUTO;
}
public void accettazioneProvvisoria() {
  if (stato_corrente == SOTTO_ESAME)
    stato_corrente = CANDIDATO_ACCETTAZIONE;
}
public void rifiuto() {
  if (stato_corrente == CANDIDATO_RIFIUTO)
    stato_corrente = RIFIUTATO;
}
public void accettazione() {
  if (stato_corrente == CANDIDATO_ACCETTAZIONE)
    stato_corrente = ACCETTATO;
}
public void riesaminare() {
  if (stato_corrente == CANDIDATO_RIFIUTO ||
      stato_corrente == CANDIDATO_ACCETTAZIONE)
```

```
      stato_corrente = SOTTO_ESAME;
  }

  public String toString() {
    Iterator<TipoLinkAutoreDi> it = autoreDi.iterator();
    String result = "Articolo: " +  titolo +
      ", dimensioneKB " + dimensioneKB + " autori:\n";
    while(it.hasNext()) {
      TipoLinkAutoreDi t = it.next();
      result += t.getAutore().toString() + "\n";
    }
    return result;
  }
}
```

La classe Java AssociazioneAutoreDi

```
// File Sottomissioni/AssociazioneAutoreDi.java
package Sottomissioni;

public final class AssociazioneAutoreDi {
  private AssociazioneAutoreDi(TipoLinkAutoreDi x) { link = x; }
  private TipoLinkAutoreDi link;
  public TipoLinkAutoreDi getLink() { return link; }
  public static void inserisci(TipoLinkAutoreDi y) {
    if (y != null &&
        y.getArticolo().estSottoEsame()) { //nota non richiesto dalla traccia!
      AssociazioneAutoreDi k = new AssociazioneAutoreDi(y);
      k.link.getAutore().inserisciLinkAutoreDi(k);
      k.link.getArticolo().inserisciLinkAutoreDi(k);
    }
  }
  public static void elimina(TipoLinkAutoreDi y) {
    if (y != null &&
        y.getArticolo().estSottoEsame()) { //nota non richiesto dalla traccia!
      AssociazioneAutoreDi k = new AssociazioneAutoreDi(y);
      k.link.getAutore().eliminaLinkAutoreDi(k);
      k.link.getArticolo().eliminaLinkAutoreDi(k);
    }
  }
}
```

La classe Java TipoLinkAutoreDi

```
// File Sottomissioni/TipoLinkAutoreDi.java
package Sottomissioni;
import Sottomissioni.Autore.*;

public class TipoLinkAutoreDi {
  private final Autore lAutore;
  private final Articolo lArticolo;
  public TipoLinkAutoreDi(Autore a, Articolo p)
    throws EccezionePrecondizioni {
    if (a == null || p == null)  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni
        ("Gli oggetti devono essere inizializzati");
    lAutore = a; lArticolo = p;
  }
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkAutoreDi t = (TipoLinkAutoreDi)o;
      return t.lArticolo == lArticolo &&
        t.lAutore == lAutore;
    }
    else return false;
  }
  public int hashCode() {
    return lAutore.hashCode() + lArticolo.hashCode();
```



```
  }
  public Autore getAutore() { return lAutore; }
  public Articolo getArticolo() { return lArticolo; }
  public String toString() {
    return lAutore.getNome() + " " +
      lArticolo.getTitolo();
  }
}
```

La classe Java TipoLinkRevisorePrimario

```
// File Sottomissioni/TipoLinkRevisorePrimario.java
package Sottomissioni;
import Sottomissioni.Senior.*;

public class TipoLinkRevisorePrimario {
  private final Senior ilRevisore;
  private final Articolo lArticolo;
  private final boolean giudizioPositivo;
  public TipoLinkRevisorePrimario(Senior r, Articolo a, boolean g)
    throws EccezionePrecondizioni {
    if (r == null || a == null)  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni
        ("Gli oggetti devono essere inizializzati");
    ilRevisore = r; lArticolo = a; giudizioPositivo = g;
  }
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkRevisorePrimario t = (TipoLinkRevisorePrimario)o;
      return t.lArticolo == lArticolo &&
        t.ilRevisore == ilRevisore;
    }
    else return false;
  }
  public int hashCode() {
```



```
    return ilRevisore.hashCode() + lArticolo.hashCode();
  }
  public Senior getSenior() { return ilRevisore; }
  public Articolo getArticolo() { return lArticolo; }
  public boolean getGiudizioPositivo() { return giudizioPositivo; }
  public String toString() {
    return ilRevisore.getNome() + " " +
      lArticolo.getTitolo() + " " + giudizioPositivo;
  }
}
```

La classe Java TipoLinkRevisoreSecondario

```
// File Sottomissioni/TipoLinkRevisoreSecondario.java
package Sottomissioni;
import Sottomissioni.Junior.*;

public class TipoLinkRevisoreSecondario {
  private final Junior ilRevisore;
  private final Articolo lArticolo;
  private final int voto;
  public TipoLinkRevisoreSecondario(Junior r, Articolo a, int v)
    throws EccezionePrecondizioni {
    if (r == null || a == null )  // CONTROLLO PRECONDIZIONI
      throw new EccezionePrecondizioni
        ("Gli oggetti devono essere inizializzati");
    if (v < 0 || v > 9 )  // CONTROLLO VOTO SIGNIFICATIVO
      throw new EccezionePrecondizioni
        ("Il voto deve essere compreso tra 0 e 9");
    ilRevisore = r; lArticolo = a; voto = v;
  }
  public boolean equals(Object o) {
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkRevisoreSecondario t = (TipoLinkRevisoreSecondario)o;
      return t.lArticolo == lArticolo &&
        t.ilRevisore == ilRevisore;
    }
```



```
    else return false;
  }
  public int hashCode() {
    return ilRevisore.hashCode() + lArticolo.hashCode();
  }
  public Junior getJunior() { return ilRevisore; }
  public Articolo getArticolo() { return lArticolo; }
  public int getVoto() { return voto; }
  public String toString() {
    return ilRevisore.getNome() + " " +
      lArticolo.getTitolo() + " " + voto;
  }
}
```

La classe Java EccezioneMolteplicita

```
// File Sottomissioni/EccezioneMolteplicita.java
package Sottomissioni;

public class EccezioneMolteplicita extends RuntimeException {
  private String messaggio;
  public EccezioneMolteplicita(String m) {
    messaggio = m;
  }
  public EccezioneMolteplicita() {
    messaggio = "Si e' verificata una violazione delle molteplicita' ";
  }
  public String toString() {
    return messaggio;
  }
}
```


La classe Java EccezionePrecondizioni

Realizzazione in Java dello use case

```
// File Sottomissioni/Controlli.java
package Sottomissioni;
import java.util.*;
import Sottomissioni.Persona.*;
import Sottomissioni.Autore.*;
import Sottomissioni.Senior.*;
import Sottomissioni.Junior.*;

public final class Controlli {
  private Controlli() { }
  public static boolean autoreConRevisioneProblematica(Persona p) {
    if (!(p instanceof Autore)) return false;
    Set<TipoLinkAutoreDi> ins = ((Autore)p).getLinkAutoreDi();
    Iterator<TipoLinkAutoreDi> it = ins.iterator();
    while(it.hasNext()) {
      TipoLinkAutoreDi t = it.next();
      Articolo a = t.getArticolo();
      if (a.estAccettato() &&
          (!a.getLinkRevisorePrimario().getGiudizioPositivo() ||
           votoMedio(a) < 4))
        return true;
    }
    return false;
  }
```



```
  private static double votoMedio(Articolo a) {
    Set<TipoLinkRevisoreSecondario> ins = a.getLinkRevisoriSecondari();
    double sum = 0;
    Iterator<TipoLinkRevisoreSecondario> it = ins.iterator();
    while(it.hasNext()) {
      TipoLinkRevisoreSecondario t = it.next();
      sum +=  t.getVoto();
    }
    return sum/ins.size();
  }


  public static boolean articoloRevisionatoStessaNazionalita(Articolo a) {
    HashSet<String> naz = new HashSet<String>();
    naz.add(a.getLinkRevisorePrimario().getSenior().getNazionalita());
    Set<TipoLinkRevisoreSecondario> ins = a.getLinkRevisoriSecondari();
    Iterator<TipoLinkRevisoreSecondario> it = ins.iterator();
    while(it.hasNext()) {
      TipoLinkRevisoreSecondario t = it.next();
      String n = t.getJunior().getNazionalita();
      if(naz.contains(n)) return true;
      else naz.add(n);
    }
    return false;
  }
}
```