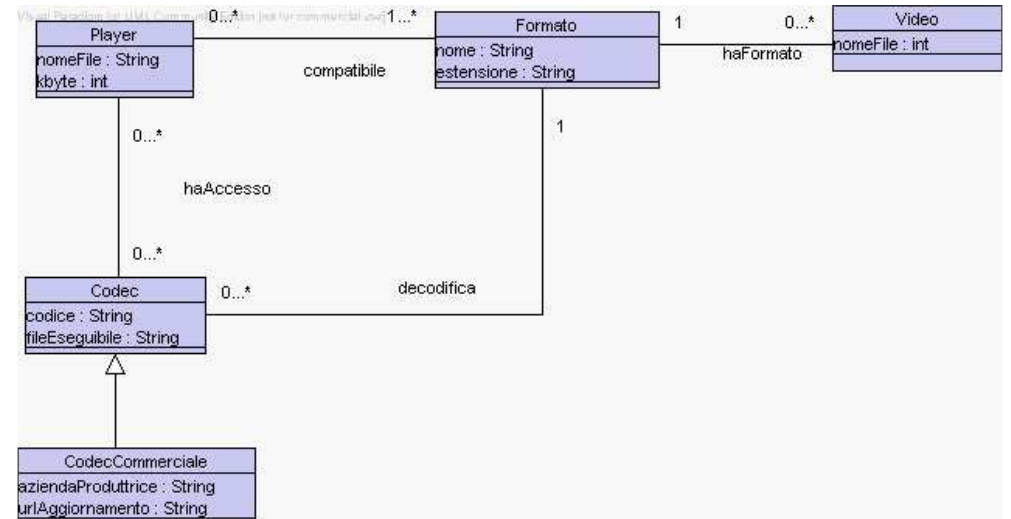


**SOLUZIONE ESAME DEL 22/07/2003**

Roma, 22 luglio 2003

1

## Diagramma delle classi UML



2

## Diagramma e specifica degli use case

//UML-usecase-diagram.txt

```

_
|_| /-----\
-|- -----| Verifiche |
/\         | video      |
sist. oper. \-----/
```

InizioSpecificaUseCase VerificheVideo

```
eseguibileDa(f: Formato): insieme(Player)
pre: nessuna
post: result e' l'insieme dei player che sono compatibili con
      il formato f e che hanno accesso ad un codec che decodifica f
```

```
aziendaProduttrice(c: Codec): stringa
pre: nessuna
post: result e' la stringa che identifica la l'azienda produttrice
      del codec c se esso e' un codec commerciale;
      altrimenti result e' la stringa "codec non commerciale"
```

```
esisteCodec(f: Formato): boolean
pre: nessuna
post: result e' true se esiste un codec che decodifica il formato f;
      false altrimenti
```

FineSpecifica

3

## Responsabilità sulle associazioni

Dalla specifica dello use case e delle molteplicità minime nel diagramma delle classi emerge che:

- *Player* ha responsabilità su *compatibile* e su *haAccesso*
- *Codec* ha responsabilità su *decodifica*
- *Formato* ha responsabilità su *decodifica* e su *compatibile*
- *Video* ha responsabilità su *haFormato*
- le altre classi non hanno responsabilità sulle associazioni.

4

## La classe Java TipoLinkCompatibile

```
// File TipoLinkCompatibile.java

public class TipoLinkCompatibile {

    //Rappr. componenti della tupla
    private final Player ilPlayer;
    private final Formato ilFormato;

    //Costruttore
    public TipoLinkCompatibile(Player x, Formato y) {
        ilPlayer = x;
        ilFormato = y;
    }

    //Accesso alle componenti
    public Player getPlayer() { return ilPlayer; }
    public Formato getFormato() { return ilFormato; }

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

5

```
}
}
```

## La classe Java TipoLinkDecodifica

```
// File TipoLinkDecodifica.java

public class TipoLinkDecodifica {

    //Rappr. componenti della tupla
    private final Codec ilCodec;
    private final Formato ilFormato;

    //Costruttore
    public TipoLinkDecodifica(Codec x, Formato y) {
        ilCodec = x;
        ilFormato = y;
    }

    //Accesso alle componenti
    public Codec getCodec() { return ilCodec; }
    public Formato getFormato() { return ilFormato; }

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

6

```
}
}
```

## La classe Java AssociazioneCompatibile

```
// File AssociazioneCompatibile.java

public class AssociazioneCompatibile {
    private AssociazioneCompatibile(TipoLinkCompatibile x) { link = x; }
    private TipoLinkCompatibile link;
    public TipoLinkCompatibile getLink() { return link; }
    public static void inserisci(TipoLinkCompatibile y) {
        if (y != null && y.getPlayer() != null && y.getFormato() != null) {
            AssociazioneCompatibile k = new AssociazioneCompatibile(y);
            y.getPlayer().inserisciLinkCompatibile(k);
            y.getFormato().inserisciLinkCompatibile(k);
        }
    }
    public static void elimina(TipoLinkCompatibile y) {
        if (y != null && y.getPlayer() != null && y.getFormato() != null) {
            AssociazioneCompatibile k = new AssociazioneCompatibile(y);
            y.getPlayer().eliminaLinkCompatibile(k);
            y.getFormato().eliminaLinkCompatibile(k);
        }
    }
}
```

7

## La classe Java AssociazioneDecodifica

```
// File AssociazioneDecodifica.java

public class AssociazioneDecodifica {
    private AssociazioneDecodifica(TipoLinkDecodifica x) { link = x; }
    private TipoLinkDecodifica link;
    public TipoLinkDecodifica getLink() { return link; }
    public static void inserisci(TipoLinkDecodifica y) {
        if (y != null && y.getCodec() != null && y.getFormato() != null) {
            AssociazioneDecodifica k = new AssociazioneDecodifica(y);
            y.getCodec().inserisciLinkDecodifica(k);
            y.getFormato().inserisciLinkDecodifica(k);
        }
    }
    public static void elimina(TipoLinkDecodifica y) {
        if (y != null && y.getCodec() != null && y.getFormato() != null) {
            AssociazioneDecodifica k = new AssociazioneDecodifica(y);
            y.getCodec().eliminaLinkDecodifica(k);
            y.getFormato().eliminaLinkDecodifica(k);
        }
    }
}
```

8

## La classe Java Formato

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

public class Formato {
    //Rappr. attributi
    private final String nome;
    private final String estensione;

    //Rappr. associazione "compatibile"
    private InsiemeArray links_compatibile;

    //Rappr. associazione "decodifica"
    private InsiemeArray links_decodifica;

    //Costruttore
    private Formato(String n, String e) {
        nome = n;
        estensione = e;
        links_compatibile = new InsiemeArray(TipoLinkCompatibile.class);
        links_decodifica = new InsiemeArray(TipoLinkDecodifica.class);
    }

    //Accesso agli attributi
    public String getNome() { return nome; }
    public String getEstensione() { return estensione; }

    //Accesso alla associazione "compatibile"

    public Set getLinkCompatibile() {
```

9

```
        return (InsiemeArray)links_compatibile.clone();
    }

    public void inserisciLinkCompatibile(AssociazioneCompatibile a) {
        if (a != null && a.getLink().getFormato() == this &&
            a.getLink().getPlayer() != null)
            links_compatibile.add(a.getLink());
    }
    public void eliminaLinkCompatibile(AssociazioneCompatibile a) {
        if (a != null && a.getLink().getFormato() == this)
            links_compatibile.remove(a.getLink());
    }

    //Accesso alla associazione "decodifica"

    public Set getLinkDecodifica() {
        return (InsiemeArray)links_decodifica.clone();
    }

    public void inserisciLinkDecodifica(AssociazioneDecodifica a) {
        if (a != null && a.getLink().getFormato() == this &&
            a.getLink().getCodec() != null)
            links_decodifica.add(a.getLink());
    }
    public void eliminaLinkDecodifica(AssociazioneDecodifica a) {
        if (a != null && a.getLink().getFormato() == this)
            links_decodifica.remove(a.getLink());
    }

    //Overriding delle funzione speciali (cioe' solo toString)
    public String toString() {
```

```

    return "Formato " + nome + ", " + estensione;
}
}

```

## La classe Java Player

```

// File Player.java
import java.util.*;

public class Player {
    //Rappr. attributi
    private final String nomeFile;
    private final int kbyte;

    //Rappr. associazione "compatibile"
    private InsiemeArray links_compatibile;

    //Rappr. associazione "decodifica"
    private InsiemeArray haAccesso;

    //Costruttore
    private Player(String n, int k) {
        nomeFile = n;
        kbyte = k;
        links_compatibile = new InsiemeArray(TipoLinkCompatibile.class);
        haAccesso = new InsiemeArray(Codec.class);
    }

    //Accesso agli attributi
    public String getNomeFile() { return nomeFile; }
    public int getKbyte() { return kbyte; }

    //Accesso alla associazione "compatibile"

    public int quantiCompatibile() {

```

10

```

    return links_compatibile.size();
}

public Set getLinkCompatibile() {
    if (quantiCompatibile() < 1)
        throw new RuntimeException(
            "Formato: partecipazione obbligatoria in compatibile violata");
    return (InsiemeArray)links_compatibile.clone();
}

public void inserisciLinkCompatibile(AssociazioneCompatibile a) {
    if (a != null && a.getLink().getPlayer() == this &&
        a.getLink().getPlayer() != null)
        links_compatibile.add(a.getLink());
}

public void eliminaLinkCompatibile(AssociazioneCompatibile a) {
    if (a != null && a.getLink().getPlayer() == this)
        links_compatibile.remove(a.getLink());
}

//Accesso alla associazione "haAccesso"

public Set getCodecs() {
    return (InsiemeArray)haAccesso.clone();
}

public void inserisciCodec(Codec c) {
    if (c != null)
        haAccesso.add(c);
}

public void eliminaCodec(Codec c) {

```

```

        if (c != null)
            haAccesso.remove(c);
    }

    //Overriding delle funzione speciali (cioe' solo toString)
    public String toString() {
        return "Player " + nomeFile + ", " + kbyte;
    }
}

```

## La classe Java Codec

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

public class Codec {
    //Rappr. attributi
    private final String codice;
    private final String fileEseguibile;

    //Rappr. associazione "decodifica"
    private TipoLinkDecodifica link_decodifica;

    //Costruttore
    public Codec(String c, String f) {
        codice = c;
        fileEseguibile = f;
    }

    //Accesso agli attributi
    public String getCodice() { return codice; }
    public String getFileEseguibile() { return fileEseguibile; }

    //Accesso alla associazione "decodifica"

    public boolean estSigLinkDecodifica() {
        return link_decodifica != null;
    }

    public TipoLinkDecodifica getLinkDecodifica() {
        if (!estSigLinkDecodifica())
```

11

```
        throw new RuntimeException(
            "Codec: partecipazione obbligatoria in decodifica violata");
        return link_decodifica;
    }

    public void inserisciLinkDecodifica(AssociazioneDecodifica a) {
        if (a != null && a.getLink().getCodec() == this &&
            a.getLink().getCodec() != null)
            link_decodifica = a.getLink();
    }

    public void eliminaLinkDecodifica(AssociazioneDecodifica a) {
        if (a != null && link_decodifica.equals(a.getLink()))
            link_decodifica = null;
    }

    //Overriding delle funzione speciali (cioe' solo toString)
    public String toString() {
        return "Codec " + codice + ", " + fileEseguibile;
    }
}
```

## La classe Java CodecCommerciale

```
// File CodecCommerciale.java

public class CodecCommerciale extends Codec {
    private final String nomeAzienda;
    private final String urlAggiornamento;
    public CodecCommerciale(String n, String f, String a, String u) {
        super(n,f);
        nomeAzienda = a;
        urlAggiornamento = u;
    }
    public String getNomeAziendaProduttrice() { return nomeAzienda; }
    public String getUrlAggiornamento() { return urlAggiornamento; }
}
```

12

## La classe Java Video

```
// File Video.java

public class Video {
    //Rappr. attributi
    private final String nomeFile;

    //Costruttore
    public Video(String n) {
        nomeFile = n;
    }

    //Accesso agli attributi
    public String getNomeFile() { return nomeFile; }
}
```

13

## Realizzazione in Java dello use case

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

public class VerificheVideo {

    public static Set eseguibileDa(Formato f) {

        Set ris = new InsiemeArray(Player.class);

        Set tuple = f.getLinkCompatibile();
        Iterator it = tuple.iterator();
        while(it.hasNext()) {
            TipoLinkCompatibile t = (TipoLinkCompatibile)it.next();

            Player p = t.getPlayer();
            Set cs = p.getCodecs();
            Iterator itc = cs.iterator();
            while(itc.hasNext()) {
                Codec c = (Codec)itc.next();
                if (c.getLinkDecodifica().getFormato().equals(f))
                    ris.add(p);
            }
        }
        return ris;
    }
}
```

14

```
public static String aziendaProduttrice(Codec c) {
    if (CodecCommerciale.class.isInstance(c))
        return ((CodecCommerciale)c).getNomeAziendaProduttrice();
    else return "codec non commerciale";
}
}
```

## InsiemeArray

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

```
import java.util.*;

public class InsiemeArray implements Set, Cloneable {

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

    protected int cardinalita;
    protected Class elemClass;

    // costruttori
    public InsiemeArray(Class cl) {
        array = new Object[dimIniz];
        cardinalita = 0;
        elemClass = cl;
    }

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

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

15

```
public boolean isEmpty() {
    return cardinalita == 0;
}

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

public boolean add(Object e) {
    if (!elemClass.isInstance(e)) return false;
    else 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 (!elemClass.isInstance(e)) return false;
    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("addAll() 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())
        a = new Object[size()];
    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 (!elemClass.equals(ins.elemClass)) return false;
        // ins non e' un insieme del tipo voluto
        else 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;
}
}

```

## IteratorInsiemeArray

```
// Quanto segue deve stare nello stesso package di InsiemeArray

import java.util.*;

public class IteratorInsiemeArray implements Iterator {

    private InsiemeArray insiemeArray;
    private int indice;

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

    // Realizzazione funzioni di Iterator
    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");
    }
}
```