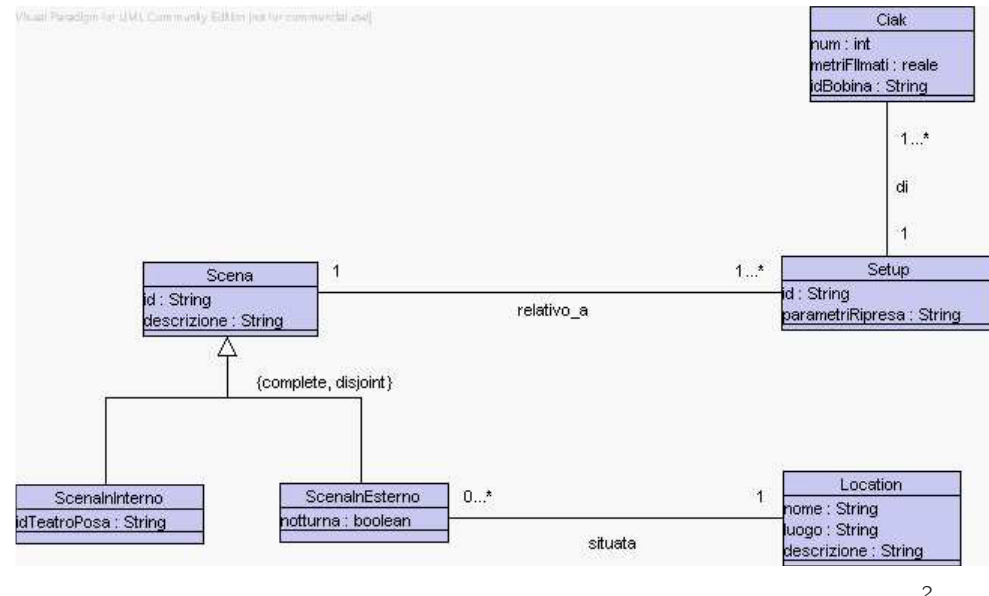


SOLUZIONE ESAME DEL 11/07/2003

Roma, 11 luglio 2003

1

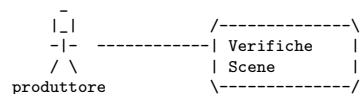
Diagramma delle classi UML



2

Diagramma e specifica degli use case

//UML-usecase-diagram.txt



InizioSpecificaUseCase VerificheScene

```
metriPellicola(s: Scena): reale
pre: nessuna
post: result e' la somma dei metri di pellicola utilizzati
    complessivamente in tutti i ciak di tutti i setup della scena s
```

```
doveSiGira(s: Scena): stringa
pre: nessuna
post: result e' la stringa che identifica la location se s
    e' una scena in esterno o il teatro di posa se s e' una
    scena in interno.
```

FineSpecifica

3

Responsabilità sulle associazioni

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

- *Scena* ha responsabilità su *relativo_a*
- *Setup* ha responsabilità su *relativo_a* e su *di*
- *ScenaInEsterno* ha responsabilità su *situata*
- le altre classi non hanno responsabilità sulle associazioni.

4

La classe Java TipoLinkRelativoA

```
// File TipoLinkRelativoA.java

public class TipoLinkRelativoA {

    //Rappr. componenti della tupla
    private final Scena laScena;
    private final Setup ilSetup;

    //Costruttore
    public TipoLinkRelativoA(Scena x, Setup y) {
        laScena = x;
        ilSetup = y;
    }

    //Accesso alle componenti
    public Scena getScena() { return laScena; }
    public Setup getSetup() { return ilSetup; }

    //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.laScena != null && b.ilSetup != null &&
                b.laScena == laScena &&
                b.ilSetup == ilSetup;
        }
        else return false;
    }
}
```

5

```
}
}
```

La classe Java AssociazioneRelativoA

```
// File AssociazioneRelativoA.java

public class AssociazioneRelativoA {
    private AssociazioneRelativoA(TipoLinkRelativoA x) { link = x; }
    private TipoLinkRelativoA link;
    public TipoLinkRelativoA getLink() { return link; }
    public static void inserisci(TipoLinkRelativoA y) {
        if (y.getScena() != null && y.getSetup() != null) {
            AssociazioneRelativoA k = new AssociazioneRelativoA(y);
            y.getScena().inserisciLinkRelativoA(k);
            y.getSetup().inserisciLinkRelativoA(k);
        }
    }
    public static void elimina(TipoLinkRelativoA y) {
        if (y.getScena() != null && y.getSetup() != null) {
            AssociazioneRelativoA k = new AssociazioneRelativoA(y);
            y.getScena().eliminaLinkRelativoA(k);
            y.getSetup().eliminaLinkRelativoA(k);
        }
    }
}
}
```

6

La classe Java Scena

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

abstract public class Scena {
    //Rappr. attributi
    private final String id;
    private final String descrizione;

    //Rappr. associazione "relativo_a"
    private InsiemeArray insieme_link;

    //Costruttore
    protected Scena(String i, String d) {
        id = i;
        descrizione = d;
        insieme_link = new InsiemeArray(TipoLinkRelativoA.class);
    }

    //Accesso agli attributi
    public String getId() { return id; }
    public String getDescrizione() { return descrizione; }

    //Accesso alla associazione "relativo_a"

    public int quantiSetup() {
        return insieme_link.size();
    }

    public Set getLinkRelativoA() {
```

7

La classe Java Setup

```
    if (quantiSetup() < 1)
        throw new RuntimeException(
            "Scena: partecipazione obbligatoria in relativo_a violata");
    return (InsiemeArray)insieme_link.clone();
}

public void inserisciLinkRelativoA(AssociazioneRelativoA a) {
    if (a != null && a.getLink().getScena() == this &&
        a.getLink().getSetup() != null)
        insieme_link.add(a.getLink());
}

public void eliminaLinkRelativoA(AssociazioneRelativoA a) {
    if (a.getLink() != null && a.getLink().getScena() == this)
        insieme_link.remove(a.getLink());
}

//Overriding delle funzione speciali (cioe' solo toString)
public String toString() {
    return "Scena " + id + ", " + descrizione;
}
}
```

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

public class Setup {
    //Rappr. attributi
    private final String id;
    private final String parametri_ripresa;

    //Rappr. associazione "relativo_a"
    private TipoLinkRelativoA relativo_a;

    //Rappr. associazione "di"
    private InsiemeArray insieme_link_di;

    //Costruttore
    public Setup(String i, String pr) {
        id = i;
        parametri_ripresa = pr;
        insieme_link_di = new InsiemeArray(Ciak.class);
    }

    //Accesso agli attributi
    public String getId() { return id; }
    public String getParametriRipresa() { return parametri_ripresa; }

    //Accesso alla associazione "relativo_a"

    public boolean estSigLinkRelativoA() {
        return parametri_ripresa != null;
    }
}
```

8

```
    }

    public TipoLinkRelativoA getLinkRelativoA() {
        if (!estSigLinkRelativoA())
            throw new RuntimeException(
                "Setup: partecipazione obbligatoria in relativo_a violata");
        return relativo_a;
    }

    public void inserisciLinkRelativoA(AssociazioneRelativoA a) {
        if (a != null && a.getLink().getSetup() == this &&
            a.getLink().getSetup() != null)
            relativo_a = a.getLink();
    }

    public void eliminaLinkRelativoA(AssociazioneRelativoA a) {
        if (a != null && relativo_a.equals(a.getLink()))
            relativo_a = null;
    }

    //Accesso alla associazione "di"

    public int quantiCiak() {
        return insieme_link_di.size();
    }

    public Set getCiak() {
        if (quantiCiak() < 1)
            throw new RuntimeException(
                "Scena: partecipazione obbligatoria in relativo_a violata");
        return (InsiemeArray)insieme_link_di.clone();
    }
}
```

```
    }

    public void inserisciCiak(Ciak c) {
        if (c != null)
            insieme_link_di.add(c);
    }

    public void eliminaCiak(Ciak c) {
        if (c != null)
            insieme_link_di.remove(c);
    }

    //Overriding delle funzione speciali (cioe' solo toString)
    public String toString() {
        return "Setup " + id + ", " + parametri_ripresa;
    }
}
```

La classe Java Ciak

```
// File Ciak.java

public class Ciak {
    //Rappr. attributi
    private final int num;
    private final double metriPellicola;
    private final String bobina;

    //Costruttore
    public Ciak(int n, double m, String b) {
        num = n;
        metriPellicola = m;
        bobina = b;
    }

    //Accesso agli attributi
    public double getNum() { return num; }
    public double getMetriPellicola() { return metriPellicola; }
    public String getIdBobina() { return bobina; }
}
```

9

La classe Java ScenaInInterno

```
// File ScenaInInterno.java

public class ScenaInInterno extends Scena {
    private final String idTeatroPosa;
    public ScenaInInterno(String i, String d, String t) {
        super(i,d);
        idTeatroPosa = t;
    }
    public String getIdTeatroPosa() { return idTeatroPosa; }
}
```

10

La classe Java ScenaInEsterno

```
// File ScenaInEsterno.java

public class ScenaInEsterno extends Scena {
    private final boolean notturno;

    private Location loc;

    public ScenaInEsterno(String i, String d, boolean b) {
        super(i,d);
        notturno = b;
    }

    public boolean getEstNotturno() { return notturno; }

    public boolean estSigLocation() {
        return loc != null;
    }

    public Location getLocation() {
        if (!estSigLocation())
            throw new RuntimeException(
                "ScenaInEsterno: partecipazione obbligatoria in situata violata");
        return loc;
    }

    public void setLocation(Location l) {
        loc = l;
    }
}
```

11

La classe Java Location

```
// File Location.java

public class Location {
    //Rappr. attributi
    private final String nome;
    private final String luogo;
    private final String descrizione;

    //Costruttore
    public Location(String n, String l, String d) {
        nome = n;
        luogo = l;
        descrizione = d;
    }

    //Accesso agli attributi
    public String getNome() { return nome; }
    public String getLuogo() { return luogo; }
    public String getDescrizione() { return descrizione; }
}
```

12

Realizzazione in Java dello use case

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

public class VerificheScene {

    public static double metriPellicola(Scena s) {

        double ris = 0;
        Set tuple = s.getLinkRelativoA();
        Iterator it = tuple.iterator();
        while(it.hasNext()) {
            TipoLinkRelativoA t = (TipoLinkRelativoA)it.next();
            Set ciak = t.getSetup().getCiak();
            Iterator it2 = ciak.iterator();
            while(it2.hasNext()) {
                Ciak c = (Ciak)it2.next();
                ris = ris + c.getMetriPellicola();
            }
        }
        return ris;
    }

    public static String doveSiGira(Scena s) {
        if (ScenaInInterno.class.isInstance(s))
            return ((ScenaInInterno)s).getIdTeatroPosa();
        else return ((ScenaInEsterno)s).getLocation().getLuogo();
    }
}
```

13

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 dimInit = 10; //dim. iniz. array

    protected int cardinalita;
    protected Class elemClass;

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

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

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

14

```
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("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())
        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");
    }
}

```