Università di Roma "La Sapienza", Facoltà di Ingegneria

Corso di
"PROGETTAZIONE DEL SOFTWARE" (Ing. Gestionale)
Prof. Giuseppe De Giacomo, A.A. 2001-02

SOLUZIONE ESAME DEL 19/07/2002

Roma, 19 luglio 2002

Diagramma delle classi UML

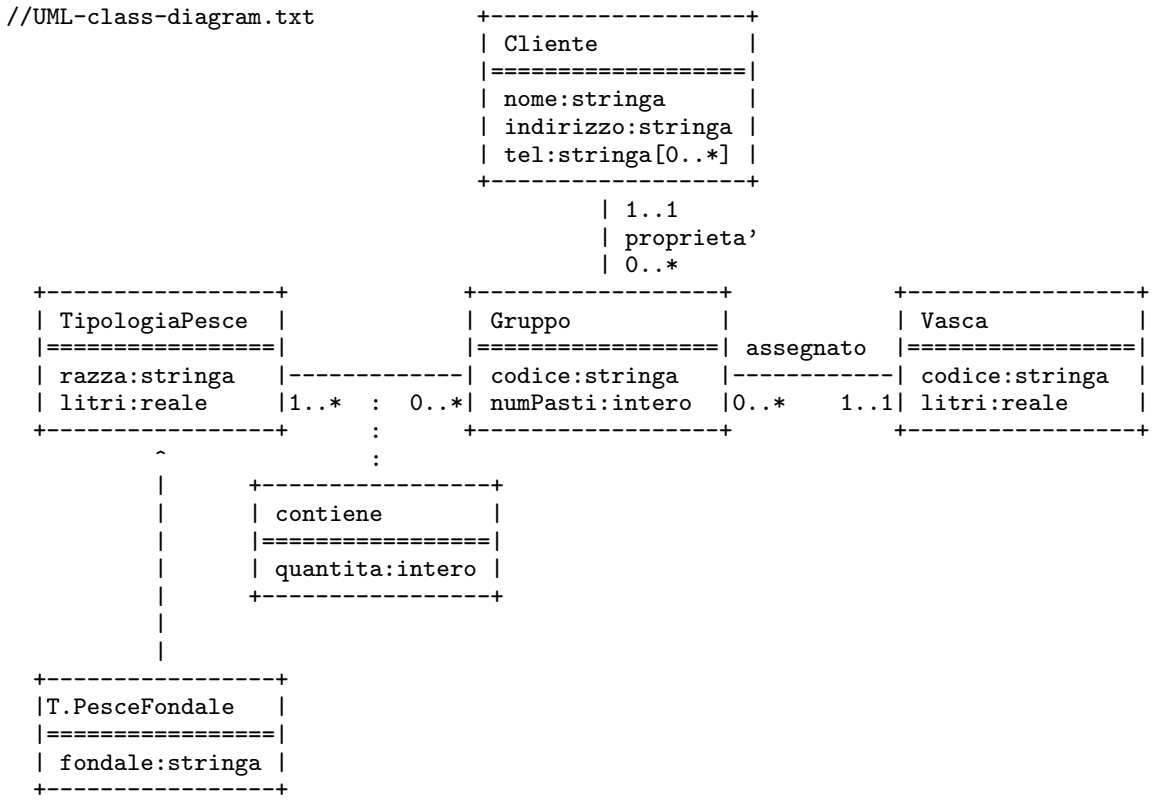

Diagramma e specifica degli use case

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


        _
       |_|              /--------------\
       -|-  ------------| Verifiche    |
       / \              | fish-sitting |
     Gestore            \--------------/
  servizio fish-sittig


InizioSpecificaUseCase VerificheFishSitting

  estOmogeneo(g: Gruppo): booleano
     pre:  nessuna
     post: result e' true se i pesci che formano il gruppo g sono tutti
           dello stessa razza, false altrimenti

  estAdatta(v: Vasca, g: Gruppo): booleano
     pre:  nessuna
     post: result e' true se i litri contenuti nella vasca v sono
           maggiori o uguali della somma dei litri necessari per i
           singoli pesci che costituiscono il gruppo g

FineSpecifica
```


Responsabilità sulle associazioni

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

- *Gruppo* ha responsabilità su *contiene*, *assegnato* e *proprietà*
- le altre classi non hanno responsabilità sulle associazioni.

La classe Java Gruppo

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

public class Gruppo {
  //Rappr. attributi
  private final String codice;
  private int numPasti;

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


  //Rappr. associazione "assegnato"
  private Vasca vasca;


  //Rappr. associazione "proprieta"
  private Cliente proprietario;

  //Costruttore
  public Gruppo(String c, int n) {
    codice = c;
    numPasti = n;
    insieme_link = new InsiemeArray(TipoLinkContiene.class);
    //Nota: non setta il proprietario e la vasca.
    //Sara' cura del cliente assicurarsi che ci sia sempre
    //un proprietario ed una vasca
  }
```



```
  //Accesso agli attributi
  public String getCodice() { return codice; }
  public int getNumPasti() { return numPasti; }
  public void setNumPasti(int n) { numPasti = n; }

  //Accesso alla associazione "contiene"

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

  public Set getLinksContiene() {
    if (quanteTipologieContiene() < 1)
      throw new RuntimeException(
        "Gruppo: partecipazione obbligatoria in contiene violata");
    return (InsiemeArray)insieme_link.clone();
  }

  public void inserisciLinkContiene(TipoLinkContiene l) {
    if (l != null  && l.getGruppo() == this &&
        l.getTipologiaPesce() != null)
      insieme_link.add(l);
  }
  public void eliminaLinkContiene(TipoLinkContiene l) {
    if (l != null  && l.getGruppo() == this)
      insieme_link.remove(l);
  }

  //Accesso alla associazione "proprieta'"
  public boolean haProprietario() { return proprietario != null; }
  public Cliente getProprietario() {
```

```
    if (!haProprietario())
      throw new RuntimeException(
        "Gruppo: partecipazione obbligatoria in proprieta' violata");
    return proprietario;
  }
  public void inserisciProprietaro(Cliente c) {
    if (c != null) proprietario = c;
  }
  public void eliminaProprietaro() { proprietario = null; }

  //Accesso alla associazione "assegnato"
  public boolean haVasca() { return vasca != null; }
  public Vasca getVasca() {
    if (!haVasca())
      throw new RuntimeException(
        "Gruppo: partecipazione obbligatoria in assegnato violata");
      return vasca;
  }
  public void inserisciVasca(Vasca v) {
    if( v!=null) vasca = v;
  }
  public void eliminaVasca() { proprietario = null; }


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

La classe Java TipoLinkContiene

```
// File TipoLinkContiene.java

public class TipoLinkContiene {

  //Rappr. componenti della tupla
  private final Gruppo ilGruppo;
  private final TipologiaPesce laTipologiaPesce;

  //Rappr. attributi della tupla
  private final int quantita;

  //Costruttore
  public TipoLinkContiene(Gruppo x, TipologiaPesce y, int q) {
    ilGruppo = x;
    laTipologiaPesce = y;
    quantita = q;
  }

  //Accesso alle componenti
  public Gruppo getGruppo() { return ilGruppo; }
  public TipologiaPesce getTipologiaPesce() { return laTipologiaPesce; }

  //Accesso agli attributi
  public int getQuantita() { return quantita; }

  //Overriding di funzioni speciali ereditate da Object
  //Nota TipoLinkContiene 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())) {
      TipoLinkContiene b = (TipoLinkContiene)o;
      return b.ilGruppo != null && b.laTipologiaPesce != null &&
        b.ilGruppo == ilGruppo &&
        b.laTipologiaPesce == laTipologiaPesce;
    }
    else return false;
  }
}
```

La classe Java TipologiaPesce

```
// File TipologiaPesce.java

public class TipologiaPesce {
  //Rappr. attributi
  private final String razza;
  private final double litri;

  //Costruttore
  protected TipologiaPesce(String r, double l) {
    razza = r;
    litri = l;
  }

    //Accesso agli attributi
  public String getRazza() { return razza; }
  public double getLitri() { return litri; }

}
```

La classe Java TipologiaPesceFondale

```
// File TipologiaPesceFondale.java

public class TipologiaPesceFondale extends TipologiaPesce {
  private final String fondale;
  public TipologiaPesceFondale(String r, double l, String f) {
    super(r,l);
    fondale = f;
  }
  public String getFondale() { return fondale; }
}
```


La classe Java Cliente

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

public class Cliente {

  //Rappr. attributi
  private final String nome;
  private String indirizzo;
  private InsiemeArray tels;

  //Costruttore
  public Cliente(String n, String i) {
    nome = n;
    indirizzo = i;
    tels = new InsiemeArray(String.class);
  }

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

  public String getIndirizzo() { return indirizzo; }
  public void setIndirizzo(String i) { indirizzo = i; }

  public Set getTels() { return (InsiemeArray)tels.clone(); }
  public void inserisciTel(String t) { tels.add(t); }
  public void eliminaTel(String t) { tels.remove(t); }

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

```
    return "Cliente " + nome;
  }
}
```

La classe Java Vasca

```
// File Vasca.java

public class Vasca {
  //Rappr. attributi
  private final String codice;
  private final double litri;

  //Costruttore
  public Vasca(String c, double l) {
    codice = c;
    litri = l;
  }

  //Accesso agli attributi
  public String getCodice() { return codice; }
  public double getLitri() { return litri; }

  //Overriding delle funzione speciali (cioe' solo toString): come prima
  public String toString() {
    return "Vasca: " + codice + ", " + litri;
  }
}
```

Realizzazione in Java dello use case

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

public class VerificheFishSitting {


  public static boolean estOmogeneo(Gruppo g) {
    Set tuple = g.getLinksContiene();
    if (tuple.isEmpty()) return true;
    else {
      Iterator it = tuple.iterator();
      TipoLinkContiene t1 = (TipoLinkContiene)it.next();
      String razza = t1.getTipologiaPesce().getRazza();
      while(it.hasNext()) {
        TipoLinkContiene t = (TipoLinkContiene)it.next();
        if (!t.getTipologiaPesce().getRazza().equals(razza))
          return false;
      }
      return true;
    }
  }

  public static boolean estAdatta(Vasca v, Gruppo g) {
    Set tuple = g.getLinksContiene();
    double somma = 0.0;
    Iterator it = tuple.iterator();
    while(it.hasNext()) {
      TipoLinkContiene t = (TipoLinkContiene)it.next();
      somma = somma + t.getTipologiaPesce().getLitri() * t.getQuantita();
```



```
    }
    return v.getLitri() >= somma;
  }
}
```

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;
  }
```



```
  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 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");
  }
}
```