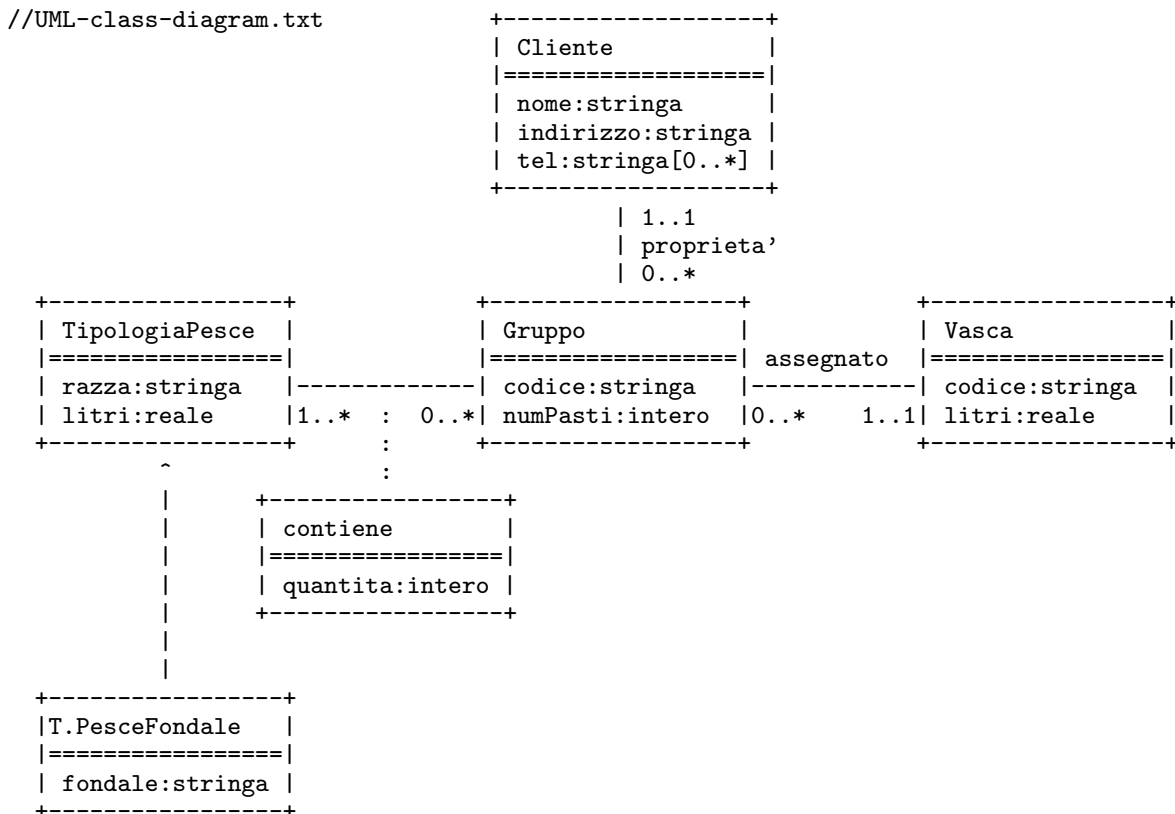


## SOLUZIONE ESAME DEL 19/07/2002

Roma, 19 luglio 2002

1

### Diagramma delle classi UML



2

# 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

3

## 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.

4

# La classe Java Gruppo

```
// File Gruppo.java

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

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

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

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

    //Costruttore
    public Gruppo(String c, int n) {
        codice = c;
        numPasti = n;
        insieme_link = new InsiemeSS(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 InsiemeSS getLinksContiene() {
        return (InsiemeSS)insieme_link.clone();
    }

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

    public void eliminaLinkContiene(TipoLinkContiene l) {
        if (l != null && l.getGruppo() == this)
            insieme_link.elimina(l);
    }

    //Accesso alla associazione "proprietaria"
    public Cliente getProprietario() { return proprietario; }
    public void inserisciProprietario(Cliente c) {
        if (c != null) proprietario = c;
    }
    public void eliminaProprietario() { proprietario = null; }

    //Accesso alla associazione "assegnato"
    public Vasca getVasca() { return vasca; }
    public void inserisciVasca(Vasca v) {
        if (v != null) vasca = v;
    }
}
```

```

public void eliminaVasca() { proprietario = null; }

//NOTA: come visto a lezione, non vi e' alcun trattamento
//particolare per le moteplicita' minime. Queste vanno gestite
//opportunamente dai clienti della classe. I cliente possono
//verificare se le moteplicita minime sono rispettate verificando
//che getProprietario() (getVasca()) non restituisca 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; }
}
```

8

## La classe Java Cliente

```
// File Cliente.java

public class Cliente {

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

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

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

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

public InsiemeSS getTels() { return (InsiemeSS)tels.clone(); }
public void inserisciTel(String t) { tels.inserisci(t); }
public void eliminaTel(String t) { tels.elimina(t); }

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

9

```
}  
}
```

## 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

public class VerificheFishSitting {

    public static boolean estOmogeneo(Gruppo g) {
        InsiemeSS tuple = g.getLinksContiene();
        if (tuple.estVuoto()) return true;
    else {
        TipoLinkContiene t1 = (TipoLinkContiene)tuple.scegli();
        String razza = t1.getTipologiaPesce().getRazza();
        tuple.elimina(t1);
        while(!tuple.estVuoto()) {
            TipoLinkContiene t = (TipoLinkContiene)tuple.scegli();
            if (!t.getTipologiaPesce().getRazza().equals(razza))
                return false;
            tuple.elimina(t);
        }
        return true;
    }

    public static boolean estAdatta(Vasca v, Gruppo g) {
        InsiemeSS tuple = g.getLinksContiene();
        double somma = 0.0;
        while(!tuple.estVuoto()) {
            TipoLinkContiene t = (TipoLinkContiene)tuple.scegli();
            somma = somma + t.getTipologiaPesce().getLitri() * t.getQuantita();
            tuple.elimina(t);
        }
        return v.getLitri() >= somma;
    }
}
```



# InsiemeSS

```
// File InsiemeSS.java

public class InsiemeSS implements Cloneable {
    // funzioni proprie del tipo astratto
    public InsiemeSS(Class cl) {
        // costruttore, realizza la funzione InsVuoto del tipo astratto Insieme
        inizio = null;
        cardinalita = 0;
        elemClass = cl;
    }
    public boolean estVuoto() {
        return inizio == null;
    }
    public boolean membro(Object e) {
        if (!elemClass.isInstance(e)) return false;
        else return appartiene(e,inizio);
    }
    public void inserisci(Object e) {
        if (!elemClass.isInstance(e)) return;
        else if (appartiene(e,inizio)) return;
        else {
            Lista l = new Lista();
            l.info = e;
            l.next = inizio;
            inizio = l;
            cardinalita = cardinalita + 1;
        }
    }
    public void elimina(Object e) {
        if (!appartiene(e,inizio)) return;
        else {
            inizio = cancella(e,inizio);
            cardinalita = cardinalita - 1;
        }
    }
    public Object scegli() {
        if (inizio == null) return null;
        else return inizio.info;
    }
    public int cardinalita() {
        return cardinalita;
    }
    // funzioni speciali ereditate da Object
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            InsiemeSS ins = (InsiemeSS)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
                Lista l = inizio;
                while (l != null) {
                    if (!appartiene(l.info,ins.inizio))
                        return false;
                    l = l.next;
                }
                return true;
            }
        }
    }
}
```

```

    }
    else return false;
}
public Object clone() {
    try {
        InsiemeSS ins = (InsiemeSS) super.clone();
        // chiamata a clone() di Object che esegue la copia campo a campo;
        // questa copia e' sufficiente per i campi cardinalita e elemClass
        // ma non per il campo inizio del quale va fatta una copia profonda
        ins.inizio = copia(inizio);
        return ins;
    } catch(CloneNotSupportedException e) {
        // non puo' accadere perche' implementiamo l'interfaccia cloneable,
        // ma va comunque gestita
        throw new InternalError(e.toString());
    }
}
public String toString() {
    String s = "{";
    Lista l = inizio;
    while (l != null) {
        s = s + l.info + " ";
        l = l.next;
    }
    s = s + "}";
    return s;
}

// campi dati
protected static class Lista {
    Object info;

    Lista next;
}
protected Lista inizio;
protected int cardinalita;
protected Class elemClass;

// funzioni ausiliarie
protected static boolean appartiene(Object e, Lista l){
    return (l != null) && (l.info.equals(e) || appartiene(e,l.next));
}
protected static Lista copia (Lista l) {
    if (l == null) return null;
    else {
        Lista ll = new Lista();
        ll.info = l.info;
        ll.next = copia(l.next);
        return ll;
    }
}
protected static Lista cancella(Object e, Lista l) {
    if (l == null) return null;
    else if (l.info.equals(e)) return l.next;
    else {
        l.next = cancella(e,l.next);
        return l;
    }
}
}
}

```