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 25/06/2002

Roma, 23 giugno 2003

Diagramma delle classi UML

//UML-class-diagram.txt

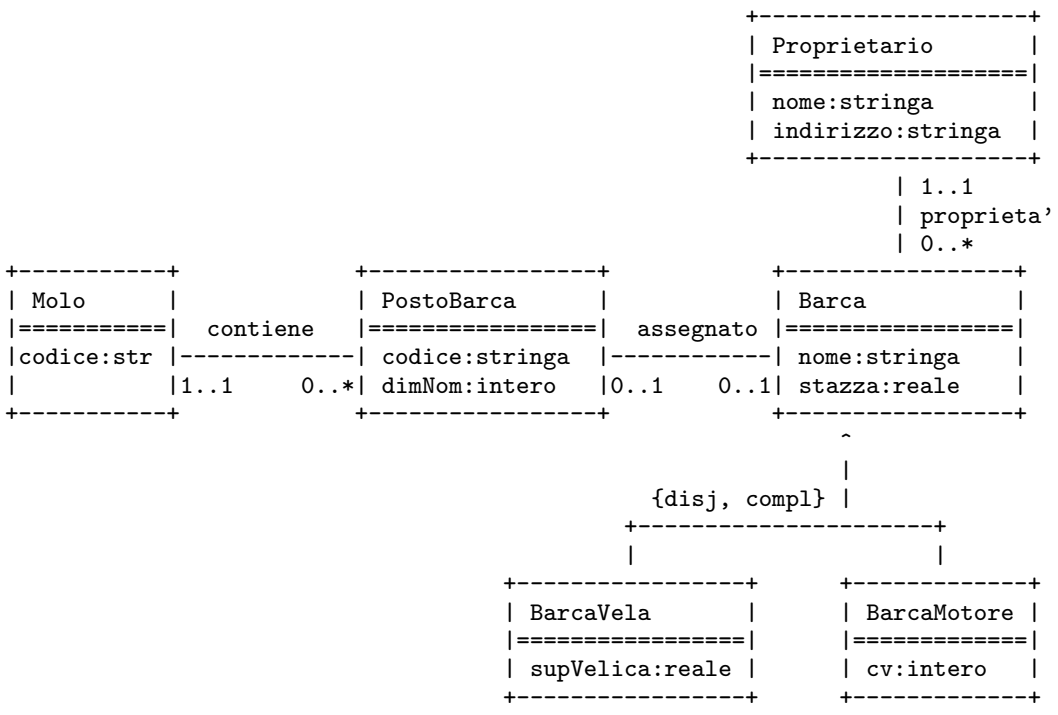

Diagramma e specifica degli use case

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


        _
       |_|              /------------------\
       -|-  ------------| Verifiche        |
       / \              | Porto Turistico  |
     Gestore            \------------------/
  Porto Turistico


InizioSpecificaUseCase VerifichePortoTuristico

  quantiPostiBarca(m: Molo): intero
     pre: nessuna
     post: result e' il numero di posti barca presenti nel molo

  quanteBarcheVela18(m: Molo): intero
     pre: nessuna
     post: result e' il numero di barche a vela con superfice velica nominale
           superiore a 18.0 presenti nei posti barca del molo

FineSpecifica
```


Responsabilità sulle associazioni

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

- entrambe *Molo* e PostoBarca hanno responsabilità su *contiene*
- solo *PostoBarca* ha responsabilità su *assegnato*, e
- solo *Barca* ha responsabilità su *proprieta'*.

La classe Java Molo

```
// File Molo.java

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

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

    //Costruttore
    public Molo(String c) {
        codice = c;
        insieme_link = new InsiemeSS(TipoLinkContiene.class);
    }

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

    //Accesso alla associazione "contiene"
    public InsiemeSS getLinksContiene() {
        return (InsiemeSS)insieme_link.clone();
    }

    //funzioni ausiliarie per l'inserimento e la cancellazione
    //che devono essere richamate dalle funzioni d'inserimento ed eliminazione
    //della classe AssociazioneContiene
    public void inserisciLinkContiene(AssociazioneContiene a) {
        if (a != null  && a.getLink().getMolo() == this)
            insieme_link.inserisci(a.getLink());
```



```
    }
    public void eliminaLinkContiene(AssociazioneContiene a) {
        if (a != null  && a.getLink().getMolo() == this)
            insieme_link.elimina(a.getLink());
    }

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

La classe Java TipoLinkContiene

```
// File TipoLinkContiene.java

public class TipoLinkContiene {
    private final Molo ilMolo;
    private final PostoBarca ilPostoBarca;
    public TipoLinkContiene(Molo x, PostoBarca y) {
        ilMolo = x;
        ilPostoBarca = y;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkContiene b = (TipoLinkContiene)o;
            return b.ilMolo != null && b.ilPostoBarca != null &&
                b.ilMolo == ilMolo &&
                b.ilPostoBarca == ilPostoBarca;
        }
        else return false;
    }
    public Molo getMolo() { return ilMolo; }
    public PostoBarca getPostoBarca() { return ilPostoBarca; }
}
```


La classe Java AssociazioneContiene

```
// File AssociazioneContiene.java

public class AssociazioneContiene {
    private AssociazioneContiene(TipoLinkContiene x) { link = x; }
    private TipoLinkContiene link;
    public TipoLinkContiene getLink() { return link; }
    public static void inserisci(TipoLinkContiene y) {
        if (y.getMolo() != null && y.getPostoBarca() != null) {
            AssociazioneContiene k = new AssociazioneContiene(y);
            y.getMolo().inserisciLinkContiene(k);
            y.getPostoBarca().inserisciLinkContiene(k);
        }
    }
    public static void elimina(TipoLinkContiene y) {
        if (y.getMolo() != null && y.getPostoBarca() != null) {
            AssociazioneContiene k = new AssociazioneContiene(y);
            y.getMolo().eliminaLinkContiene(k);
            y.getPostoBarca().eliminaLinkContiene(k);
        }
    }
}
```

La classe Java PostoBarca

```
// File PostoBarca.java

public class PostoBarca {
  //Rappr. attributi
  private final String codice;
  private final int dimNom;

  //Rappr. associazione "contiene"
  private TipoLinkContiene link;

  //Rappr. associazione "assegnata"
  private Barca barca;

  //Costruttore
  public PostoBarca(String c, int d) {
    codice = c;
    dimNom = d;
  }

  //Accesso agli attributi
  public String getCodice() { return codice; }
  public int getDimNom() { return dimNom; }


  //Accesso alla associazione "contine"
  public boolean estSigLinkContiene() {
    return link != null;
  }
  public TipoLinkContiene getLinkContiene() {
```



```
    if (!estSigLinkContiene())
      throw new RuntimeException("PostoBarca: molteplicita contiene violata");
    return link;
  }

  //funzioni ausiliarie per l'inserimento e la cancellazione
  //che devono essere richamate dalle funzioni d'inserimento ed eliminazione
  //della classe AssociazioneContiene
  public void inserisciLinkContiene(AssociazioneContiene a) {
    if (link == null && a != null  &&
        a.getLink().getPostoBarca() == this)
      link = a.getLink();
  }
  public void eliminaLinkContiene(AssociazioneContiene a) {
    if (a != null  && a.getLink().getPostoBarca() == this &&
        a.getLink().getMolo() == link.getMolo())
      link = null;
  }

  //Accesso associazione "assegnato"
  public Barca getBarca(){ return barca; }
  public void setBarca(Barca b) { barca = b; }

  //Overriding delle funzione speciali (cioe' solo toString): come prima
  public String toString() {
    return "PostoBarca: " + codice + ", " + dimNom + ", " + link.getMolo();
  }
}
```

La classe Java Barca

```
// File Barca.java

public abstract class Barca {
  //Rappr. attributi
  private final String nome;
  private final double stazza;

  //Rappr. associazione "proprieta"
  private Proprietario prop;

  //Costruttore
  protected Barca(String n, double s) {
    nome = n;
    stazza = s;
    //Nota: non setta il proprietario.
    //Sara' cura del cliente assicurarsi che ci sia sempre un proprietario
  }

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


  //Accesso alla associazione "proprieta"

  public boolean estSigProprietario() {
    return prop != null;
  }
  public Proprietario getProprietario() {
```



```
    if (!estSigProprietario())
      throw new RuntimeException("Barca: molteplicita proprietario violata");
    return prop;
  }

  public void setProprietario(Proprietario p) { prop = p; }


  //Overriding delle funzione speciali (cioe' solo toString)
  public String toString() {
    return "Barca: " + nome + ", " + stazza + ", " + prop;
  }
}
```

La classe Java BarcaVela

```
// File BarcaVela.java

public class BarcaVela extends Barca {
  private final double superficieVelica;
  public BarcaVela(String n, double s, double sv) {
    super(n,s);
    superficieVelica = sv;
  }
  public double getSuperficieVelica() { return superficieVelica; }
  public String toString() {
    return super.toString() + " BarcaVela, supefice velica " +
                              superficieVelica;
  }
}
```


La classe Java BarcaMotore

```
// File BarcaMotore.java
public class BarcaMotore extends Barca {
    protected final int cvNominali;
    public BarcaMotore(String n, double s, int cv) {
        super(n,s);
        cvNominali = cv;
    }
    public int getPotenza() { return cvNominali; }
    public String toString() {
        return super.toString() + " BarcaMotore, cv nominali " +
            cvNominali;
    }
}
```

La classe Java Proprietario

```
// File Proprietario.java

public class Proprietario {
    private final String nome;
    private String indirizzo;
    public Proprietario(String n, String i) {
        nome = n;
        indirizzo = i;
    }
    public String getNome() { return nome; }
    public String getIndirizzo() { return indirizzo; }
    public void setIndirizzo(String i) { indirizzo = i; }
}
```


Realizzazione in Java dello use case

```
// File VerifichePortoTuristico.java

public class VerifichePortoTuristico {


    public static int quantiPostiBarca(Molo m) {
        return m.getLinksContiene().cardinalita();
    }

 /*
    // Oppure se non si vuole usare cardinalita()
    public static int quantiPostiBarca(Molo m) {
        InsiemeSS tuple = m.getLinksContiene();
        int cont = 0;
        while(!tuple.estVuoto()) {
            TipoLinkContiene t = (TipoLinkContiene)tuple.scegli();
            cont++;
            tuple.elimina(t);
        }
        return cont;
    }
 */

    public static int quanteBarcheVela18(Molo m) {
        InsiemeSS tuple = m.getLinksContiene();
        int cont = 0;
        while(!tuple.estVuoto()) {
            TipoLinkContiene t = (TipoLinkContiene)tuple.scegli();
            Barca b = t.getPostoBarca().getBarca();
```

```
            if ((BarcaVela.class).isInstance(b) &&  //Nota: isInstance() tratta bene anche il caso b==null
                 //oppure: b!=null && b.getClass().equals(BarcaVela.class)&&
                ((BarcaVela)b).getSuperficieVelica() > 18.0) // si noti il casting!
                cont++;
            tuple.elimina(t);
        }
        return cont;
    }
}
```

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;
        }
    }
}
```