

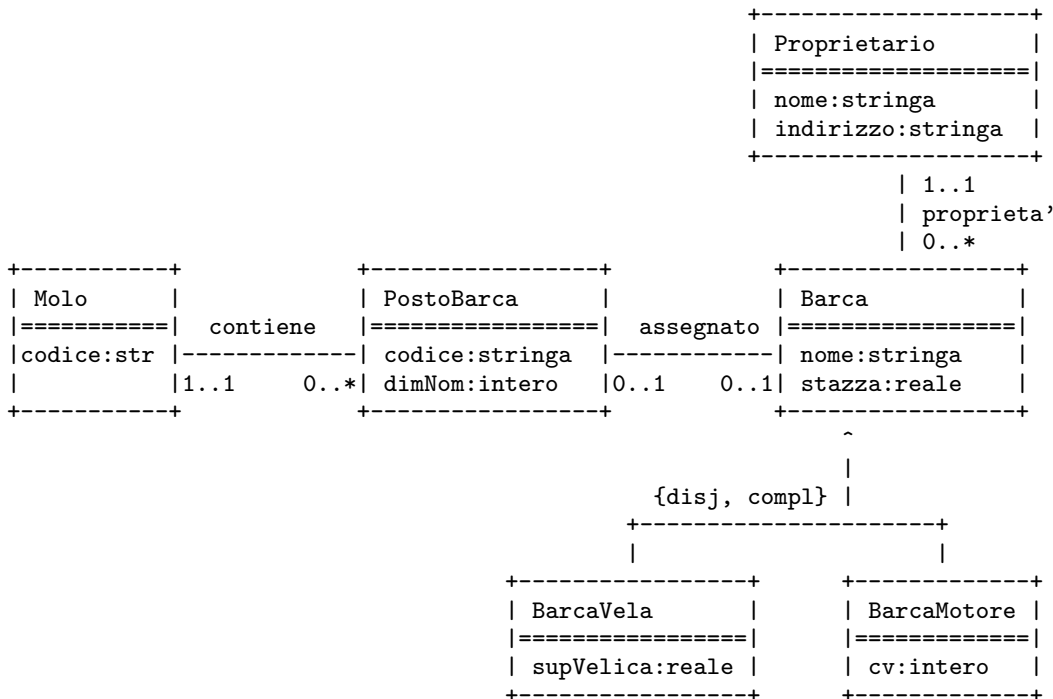
**SOLUZIONE ESAME DEL 25/06/2002**

Roma, 23 giugno 2003

1

**Diagramma delle classi UML**

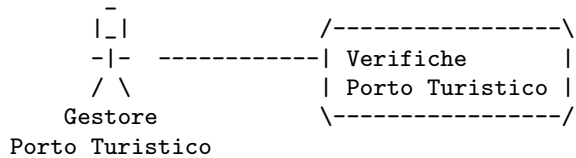
//UML-class-diagram.txt



2

# Diagramma e specifica degli use case

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



```
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
```

3

## 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 *proprietà*'.

4

# La classe Java Molo

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

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

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

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

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

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

    //funzioni ausiliarie per l'inserimento e la cancellazione
    //che devono essere richiamate dalle funzioni d'inserimento ed eliminazione
    //della classe AssociazioneContiene
    public void inserisciLinkContiene(AssociazioneContiene a) {
```

5

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

6

## 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);
        }
    }
}
```

7

# 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 "contiene"
    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 richiamate 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() {
```

9

```
        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;
    }
}
```

10

## 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;
    }
}
```

11

# 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; }
}
```

12

# Realizzazione in Java dello use case

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

public class VerifichePortoTuristico {

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

    public static int quanteBarcheVela18(Molo m) {
        Set tuple = m.getLinksContiene();
        Iterator it = tuple.iterator();
        int cont = 0;
        while(it.hasNext()) {
            TipoLinkContiene t = (TipoLinkContiene)it.next();
            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++;
        }
        return cont;
    }
}
```

13



# InsiemeLista

```
import java.util.*;

class Lista {
    Object info;
    Lista next;
}

public class InsiemeLista implements Set, Cloneable {

    // campi dati
    protected Lista inizio;
    protected int cardinalita;
    protected Class elemClass;

    // costruttori
    public InsiemeLista(Class cl) {
        inizio = null;
        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 inizio == null;
}

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

public boolean add(Object e) {
    if (!elemClass.isInstance(e)) return false;
    else if (appartiene(e,inizio)) return false;
    else {
        Lista l = new Lista();
        l.info = e;
        l.next = inizio;
        inizio = l;
        cardinalita = cardinalita + 1;
        return true;
    }
}

public boolean remove(Object e) {
    if (!elemClass.isInstance(e)) return false;
    if (!appartiene(e,inizio)) return false;
    else {
        inizio = cancella(e,inizio);
        cardinalita = cardinalita - 1;
        return true;
    }
}
```

```

public Iterator iterator() {
    return new IteratorInsiemeLista(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())) {
        InsiemeLista ins = (InsiemeLista)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 {
        InsiemeLista ins = (InsiemeLista) 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;
}

// 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;
    }
}

```

```
}  
}  
}
```

## IteratorInsiemeLista

```
// Quanto segue deve stare nello stesso package di InsiemeLista  
  
import java.util.*;  
  
public class IteratorInsiemeLista implements Iterator {  
    private Lista rif;  
  
    public IteratorInsiemeLista(InsiemeLista ins) {  
        rif = ins.inizio; //nota inizio e' accessibile perche'  
                        //InsiemeLista e' nello stesso package!!!  
    }  
  
    // Realizzazione funzioni di Iterator  
    public boolean hasNext() {  
        return rif != null;  
    }  
  
    public Object next() {  
        Object e = rif.info;  
        rif = rif.next;  
        return e;  
    }  
  
    public void remove() {  
        throw new UnsupportedOperationException("remove() non e' supportata");  
    }  
}
```