

Realizzare Classi Reattive

**CLASSE A
E AFIRED**

RESP	ATRIB. ASS	MOLTEPLICITÀ	TIPO ASS	REALIZZAZIONE CLASSE , GESTIONE ATTRIBUTI
SING	NO	0,...,1 0,...,*	BIN	<pre>private B as; public B getAs() { return as; } public void setAs(B b) { as = b; }</pre>
SING	NO	0,...,* 0,...,*	BIN	<pre>private HashSet insiemeLinkAs; public A(...){} ... insiemeLinkAs = new HashSet(); public void inserisciLinkAs(B b){ if (b != null) insiemeLinkAs.add(b); } public void eliminaLinkAs(B b){ if (b != null) insiemeLinkAs.remove(b); } public Set getLinkAs() { return (HashSet) insiemeLinkAs.clone(); }</pre>
SING	SI	0...1, 0...*	BIN	<p>CLASSE AUSILIARIA TIPOLINKAS</p> <pre>public class TipolinkAs { private final A a; private final B b; private final int n; public TipolinkAs(A a, B b, int n) throws EccezionePrecondizioni { if (x == null y == null) throw new EccezionePrecondizioni(); this.a = a; this.b = b; this.n = n; } public boolean equals(Object o) { if (o != null && getClass().equals(o.getClass())){ TipolinkAs t = (TipolinkAs)o; return t.getA() == a && t.getB() == b; } else return false; } public int hashCode() { return a.hashCode() + b.hashCode(); } public A getA() { return a; } public B getB() { return b; } public int getN() { return n; } } FINE CLASSE AUSILIARIA private TipolinkAs linkAs; public void inserisciLinkAs(TipolinkAs l) { if (l == null && l.getA() == a && linkAs == null) linkAs = l; } public void eliminaLinkAs() { linkAs = null; }</pre>

				public TipolinkAs getLinkAs() { return linkAs; }	2
SING	SI	0...*, 0...*	BIN	<pre> CLASSE AUSILIARIA TIPOLINKAS COME LA PRECEDENTE private HashSet<TipolinkAs> insiemeLinkAs; public A(...){ ... insiemeLinkAs = new HashSet<TipolinkAs>(); public void inserisciLinkAs(TipoLinkAs l) { if(l != null && l.getA() == this) insiemeLinkAs.add(l); } public void eliminaLinkAs(TipoLinkAs l) { if(l != null && l.getA() == this) insiemeLinkAs.remove(l); } public Set<TipoLinkAs> getLinkAs() { return (HashSet<TipolinkAs>) insiemeLinkAs.clone(); } } </pre>	
DOPPIA	SI	0...1, 0...1		<pre> CLASSE AUSILIARIA TIPOLINKAS COME LA PRECEDENTE CLASSE AUSILIARIA MANAGERAS public final class ManagerAs { private TipolinkAs link; private ManagerAs(TipoLinkAs l) { link = l; } public static void inserisci(TipoLinkAs l) { if(l != null && l.getA().getLinkAs() == null && l.getB().getLinkAs() == null) { ManagerAs k = new ManagerAs(l); l.getA().inserisciPerManagerAs(k); l.getB().inserisciPerManagerAs(k); } } public static void elimina(TipoLinkAs l) { if(l != null && l.getA().getLinkAs() != null && l.equals(l.getA().getLinkAs())) { ManagerAs k = new ManagerAs(l); l.getA().eliminaPerManagerAs(k); l.getB().eliminaPerManagerAs(k); } } public TipolinkAs getLink() { return link; } } FINE CLASSE AUSILIARIA MANAGER AS private TipolinkAs linkAs; public void inserisciLinkAs(TipoLinkAs l) { if(l != null && l.getA() == this) ManagerAs.inserisci(l); } public void eliminaLinkAs(TipoLinkAs l) { if(l != null && l.getA() == this) ManagerAs.elimina(l); } public TipolinkAs getLinkAs() { return linkAs; } </pre>	

				<pre> public void inserisciPerManagerAs(ManagerAs k){ if(k!= null) linkAs = k.getLink();} public void eliminaPerManagerAs(ManagerAs k){ if(k!= null) linkAs = null;} </pre>
DOPPIA	SI	0...*, 0...1	BIN	<p>CLASSE AUSILIARIA TIPOLINKAS COME LA PRECEDENTE CLASSE AUSILIARIA MANAGERAS</p> <p>La classe cambia solo per le condizioni:</p> <ul style="list-style-type: none"> • nel metodo inserisci : if(l!= null && l.getBC().getLink() == null) • nel metodo elimina : if(l!= null && l.equals(l.getBC().getLinkAs())) <p>FINE CLASSE AUSILIARIA MANAGER AS</p> <pre> private HashSet<TipolinkAs> insiemeLinkAs; public A(...){ ... insiemeLinkAs = new HashSet<TipolinkAs>();} public void inserisciLinkAs(TipolinkAs l){ if(l!= null && l.getAC() == this) ManagerAs.inserisci(l);} public void eliminaLinkAs(TipolinkAs l){ if(l!= null && l.getAC() == this) ManagerAs.elimina(l);} public Set<TipolinkAs> getLinkAs(){ return(HashSet<TipolinkAs>) insiemeLinkAs.clone();} public void inserisciPerManagerAs(ManagerAs k){ if(k!= null) insiemeLinkAs.add(k.getLink());} public void eliminaPerManagerAs(ManagerAs k){ if(k!= null) insiemeLinkAs.remove(k.getLink());} </pre>
DOPPIA	SI	0...*, 0...*	BIN	<p>CLASSE AUSILIARIA TIPOLINKAS COME LA PRECEDENTE CLASSE AUSILIARIA MANAGERAS</p> <p>La classe cambia solo per le condizioni:</p> <ul style="list-style-type: none"> • nel metodo inserisci : if(l!= null if(l.getBC().getLink() == null)) • nel metodo rimovi: if(l!= null) <p>FINE CLASSE AUSILIARIA MANAGERAS</p> <p>LA CLASSE A È IMPLEMENTATA COME LA PRECEDENTE</p>
DOPPIA	SI	n...m 0...* con n ≠ 0 & m ≠ 1	BIN	<p>CLASSE AUSILIARIA TIPOLINKAS COME LA PRECEDENTE CLASSE MANAGERAS COME LA PRECEDENTE</p> <pre> public class A { private HashSet<TipolinkAs> insiemeLinkAs; public static final int MIN_LINK_AS = n; public static final int MAX_LINK_AS = m; public A(...){ ... insiemeLinkAs = new HashSet<TipolinkAs>();} public int quantiAs(){ return insiemeLinkAs.size();} public void inserisciLinkAs(TipolinkAs l){} } </pre>

```

        if (l != null && l.getA() == this) ManagerAs.inserisci(k);}
    public void eliminalinkAs(TipoLinkAs l){
        if (l != null && l.getA() == this) ManagerAs.elimina(l);}
    public Set<TipoLinkAs> getLinkAs() throws EccezioneCardMin, EccezioneCardMax {
        if (quantiAs() < MINLINK_AS) throw new EccezioneCardMin();
        if (quantiAs() > MAXLINK_AS) throw new EccezioneCardMax();
        return (HashSet<TipoLinkAs>) insiemeLinkAs.clone();}
    public void inserisciPerManagerAs(ManagerAs k){
        if (k != null) insiemeLinkAs.add(k.getLink());}
    public void eliminaPerManagerAs(ManagerAs k) eliminaPerManagerAs(ManagerAs k){
        if (k != null) insiemeLinkAs.remove(k.getLink());}

```

DOPPIA	SI	1...1 0...*	BIN	<p>CLASSE AUSILIARIA TIPOLINKAS COME LA PRECEDENTE</p> <p>CLASSE AUSILIARIA MANAGERAS</p> <ul style="list-style-type: none"> la classe cambia per le condizioni nel metodo inserisci : if (l != null && l.get(A).quantiAs() == 0) nel metodo elimina : if (l != null && l.get(A) == l.equals(l.getA().getLinkAs())) <p>Il metodo elimina è inoltre inserito in un try{...} catch(EccezioneCardMin e){System.out.println(e);}</p> <p>FINE CLASSE AUSILIARIA MANAGERAS</p> <p>La classe A è uguale per la precedente ad eccezione di:</p> <ul style="list-style-type: none"> public static final MAXLINK_ISCRITO non presente public static final MINLINK_ISCRITO = 1; private private TipoLinkAs linkAs; //as non tituisce insieme LinkAs public int quantiAs(){ if (linkAs == null) return 0; else return 1;} public TipoLinkAs getLinkAs() throws EccezioneCardMin { if (linkAs == null) throw EccezioneCardMin(); if (quantiAs < MINLINK_ISCRITO) throw new EccezioneCardMin(); else return linkAs;}
DOPPIA / SINGOLA	SI / NO	(0...*) n volte	NARIA	<p>Si genera una classe TIPOLINKAS che ha n classi invece che 2.</p> <p>In caso di resp SINGOLA si genera un HashSet<TipoLinkAs> insiemeLinkAs.</p> <p>In caso di resp doppia l'HashSet sarà gestito da un ManagerAs.</p>
DOPPIA / SINGOLA	SI / NO	(0...*) <ordered> (0...*)		<p>la struttura dati che contiene i link di A è una LinkedList che può essere</p> <ul style="list-style-type: none"> i) <code>LinkedList</code> (resp sing e no attrib.) ii) <code>LinkedList<TipoLinkAs></code> resp doppia o attributi <p>il metodo inserisci deve avere controlli con contains se non si vogliono link duplicati (resp singola)</p> <p>il metodo inserisci per Manager deve avere ... se non si vogliono link duplicati (resp doppia)</p> <p>Qualora si vogliano link duplicati si dovrà usare il metodo add() per inserire il primo link a L.N().add(...);</p>

Gestione degli stati e delle transizioni di Classi

```
public static enum Stato { STATO1, STATO2, STATO3 };
Stato statoCorrente = Stato.STATO1;
    public Stato getStato() { return statoCorrente; }
    Tipe1 var1;  $\leftarrow$  variabili di stato
    Tipe2 var2;  $\leftarrow$ 
public void fired(Evento e) { TaskExecutor.getInstance().perform(new AFired(this, e)) }
```

Realizzazione di generalizzazioni e is-a

Le classi che sono superclassi o subclasses devono avere un altro package.

Se A è una superclasse completamente generalizzata allora deve essere implementata come abstract.

I parametri (attributi) di queste classi devono essere modificatori protected.

package a; ... import a; Le classi che estendono a devono importare il suo package.

Se necessario le variabili di stato possono essere definite protected.

Gestore delle Transizioni (concorrente), Realizzazione classe AFired

```

package A; // stesso package di A
class AFired implements Task {
    private boolean eseguita = false;
    private A a;
    private Evento e;
    public AFired(A a, Evento e){ this.a = a; this.e = e; }
    public synchronized void esegui(){
        if( eseguita || (e.getDestinatario() != null && e.getDestinatario() == null) )
            return;
        eseguita = true;
        switch(a.getStato()){
            case STATO1:
                if( e.getClass() == EventoSpecifico.class ){
                    // esegui operazioni, in particolare
                    a.statoCorrente = Stato.STATO2; // cambio stato
                    a.var1 = ...; // cambio var di stato
                    Environment.aggiungiEvento( new EventoSpecifico2(mitt, dest, payload) ); // genera un nuovo evento
                    TipoPayload1 x = ((EventoSpecifico) e).getPayload().getElementi(); // accedere al payload dell'evento
                }
                else if( ... ){ ... }
                break;
            Case STATO2:
                ...
                break;
            default:
                throw new RuntimeException("Stato corrente non riconosciuto");
        }
    }
    public synchronized boolean eseguita(){ return eseguita; }
    // è possibile aggiungere funzioni helper - private e (quando necessario) synchronized
}

```