SAPIENZA Università di Roma
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Ingegneria Informatica ed Automatica
Corso di Progettazione del Software
Esame del **26 giugno 2020**
Tempo a disposizione: 3 ore

Requisiti. L'applicazione da progettare riguarda un sistema di sviluppo software cooperativo.

Gli elementi centrali del sistema sono gli utenti e i progetti software che sviluppano. Degli utenti interessa il nome e l'email. Dei progetti interessa il nome e la data di creazione. Ogni utente può creare un numero arbitrario di progetti (anche nessuno). Ogni progetto ha un unico creatore ma può avere un arbitrario di altri manutentori. Il creatore è egli stesso un manutentore. I manutentori di un progetto sono al massimo venti. Un progetto contiene dei file (inizialmente nessuno); ogni file appartiene a un solo progetto. A un progetto possono venire fatte delle richieste di modifica. Ognuna di queste richieste è costituita da un codice (un intero) e una descrizione (una stringa). Una richiesta viene creata da un utente qualsiasi (che può o meno essere un manutentore del progetto), ed è associata a uno o più file; dato un file, non è di interesse sapere in quali richieste è coinvolto. Le richieste possono essere bloccanti o meno. Le richieste bloccanti hanno una priorità (un intero).

Una parte specifica dell'applicazione riguarda il meccanismo delle richieste. Un utente crea la richiesta, che si trova inizialmente nello stato di *creata*. Lo stesso utente può decidere di chiuderla oppure di inviarla. Nel primo caso va nello stato *chiusa*, altrimenti *inviata*. Quando è nello stato *inviata*, un manutentore del progetto può decidere di chiuderla (non interessa sapere se viene accettata o meno), oppure può mandare un suggerimento di modifica all'autore della richiesta. In questo caso, la richiesta passa nello stato *attesa*. L'autore della richiesta può decidere di chiuderla oppure di effettuare la modifica e far tornare la richiesta nello stato di *inviata*; in entrambi i casi, viene informato il manutentore che ha mandato il suggerimento.

Siamo interessati all'attività di rilascio di una versione del progetto software. In questa attività viene prima verificato che non ci siano richieste bloccanti. Se questo controllo fallisce viene stampato un messaggio di errore e l'attività termina. Se invece il controllo riesce vengono eseguite in parallelo due sottoattività; nella prima, il progetto viene compilato e poi testato; nel secondo, il progetto viene assemblato in un singolo archivio. Quando entrambe le sottoattività sono concluse, se il test è andato a buon fine il pacchetto viene rilasciato e viene stampato un messaggio di conferma. In caso contrario viene stampato un messaggio di errore.

Domanda 1. Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe *Richiesta*; diagramma delle attività di rilascio; specifica del diagramma stati e transizioni; specifica completa dell'attività di rilascio; sottoattività non atomiche; atomiche e segnali di input/output. Motivare, qualora ce ne fosse bisogno, le scelte di progetto.

Domanda 2. Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare definire SOLO le responsabilità sulle associazioni del diagramma delle classi (nella tabella, inserire anche il motivo di ognuna delle responsabilità).

Domanda 3. Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe `Richiesta` con la classe `RichiestaFired`, le classi JAVA per rappresentare le *associazioni* di cui la classe `Richiesta` ha responsabilità.
- L'*attività di rilascio* e le sue eventuali sottoattività NON atomiche.

DOMANDA 1

DIAGRAMMA DELLE CLASSI

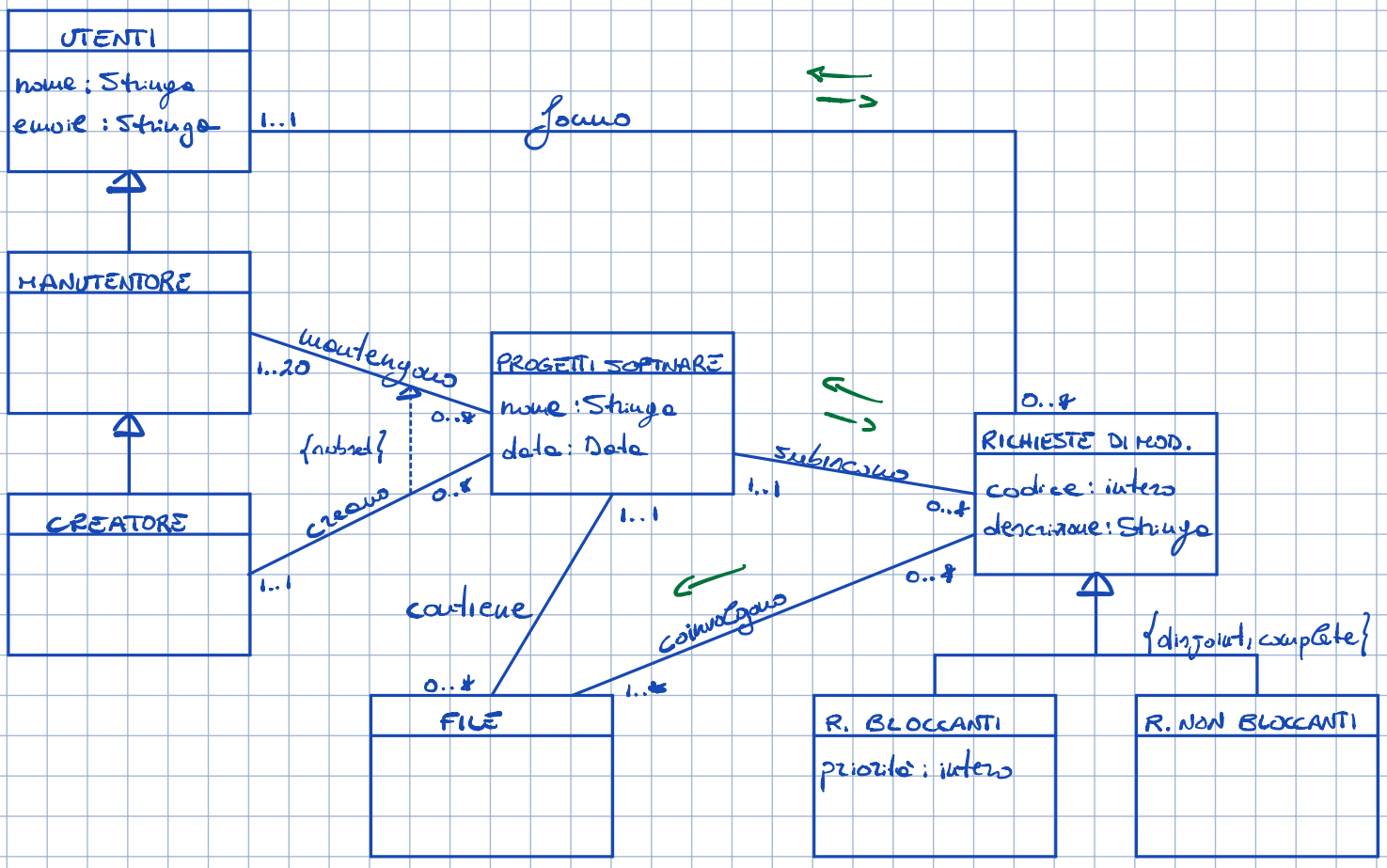
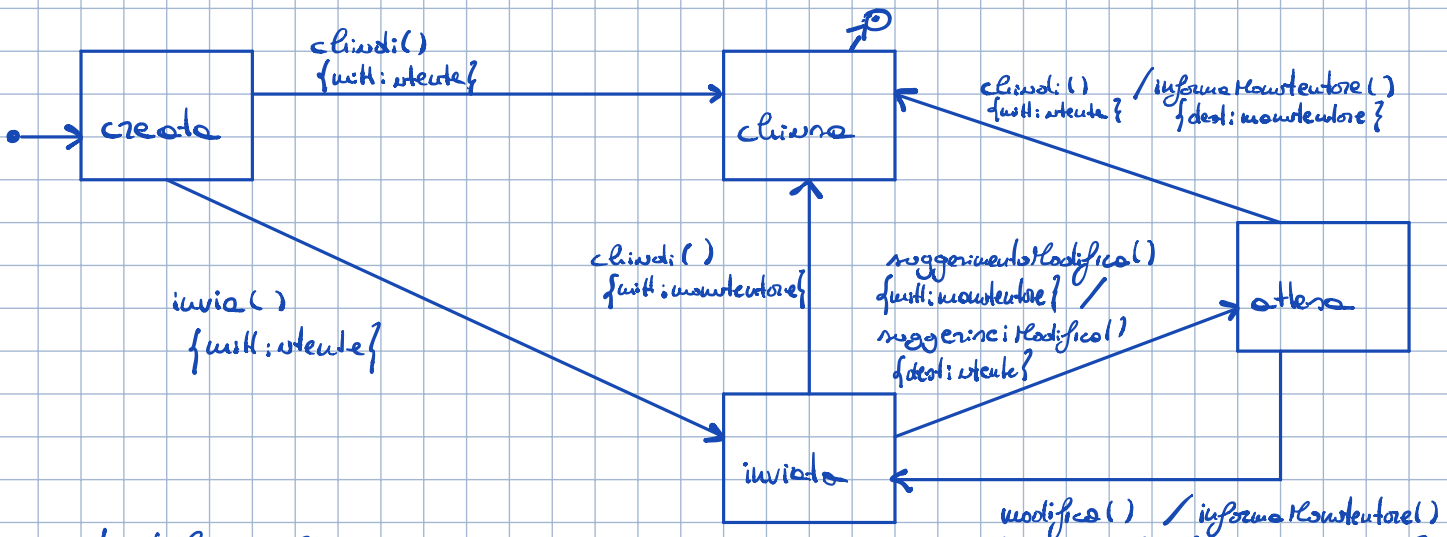


DIAGRAMMA STATI - TRANSIZIONI

Una parte specifica dell'applicazione riguarda il meccanismo delle richieste. Un utente crea la richiesta, che si trova inizialmente nello stato di *creata*. Lo stesso utente può decidere di chiuderla oppure di inviarla. Nel primo caso va nello stato *chiusa*, altrimenti *inviata*. Quando è nello stato *inviata*, un manutentore del progetto può decidere di chiuderla (non interessa sapere se viene accettata o meno), oppure può mandare un suggerimento di modifica all'autore della richiesta. In questo caso, la richiesta passa nello stato *attesa*. L'autore della richiesta può decidere di chiuderla oppure di effettuare la modifica e far tornare la richiesta nello stato di *inviata*; in entrambi i casi, viene informato il manutentore che ha mandato il suggerimento.

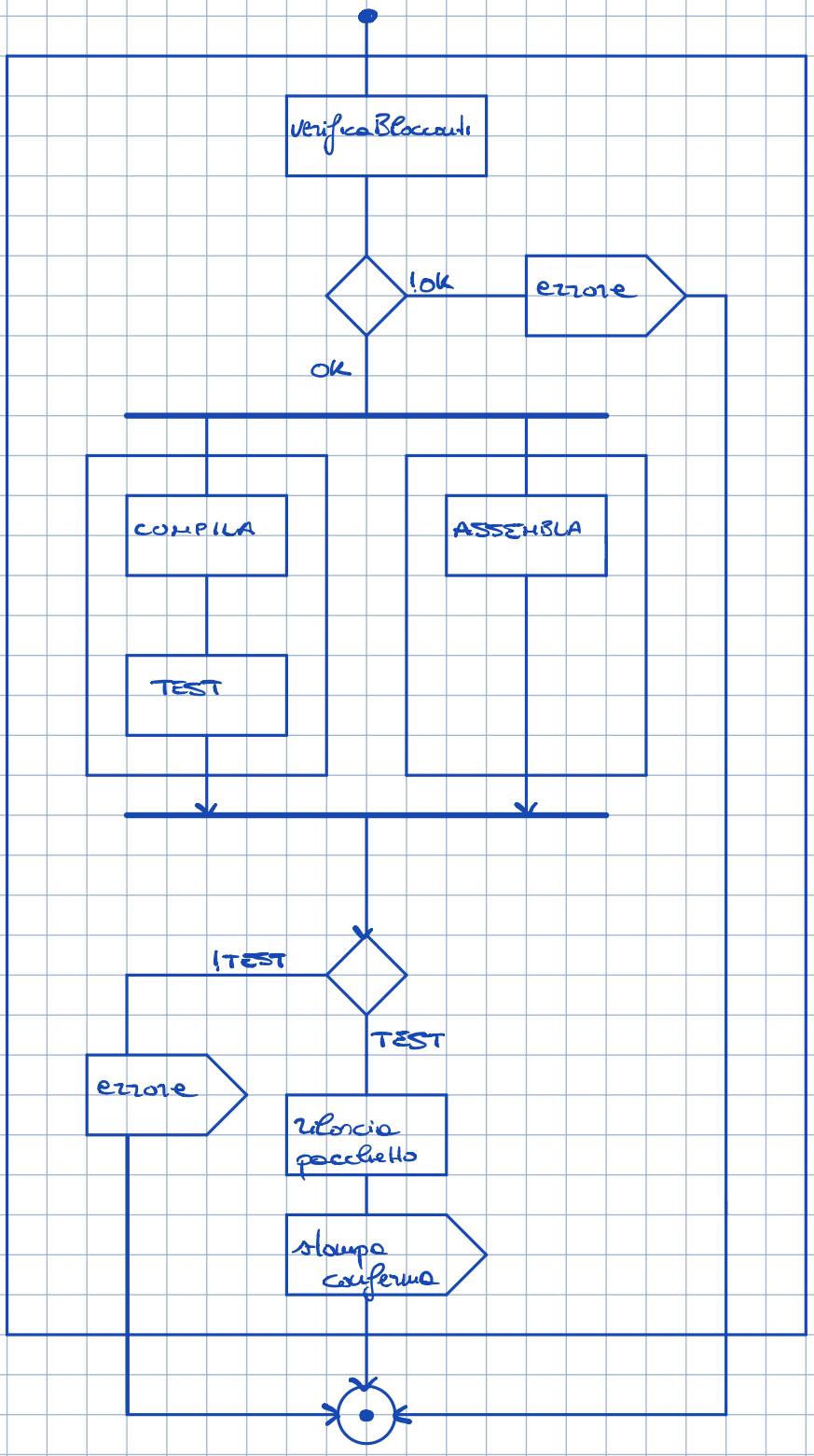


Variabili ausiliarie:

manutentore: manutentore

DIAGRAMMA ATTIVITÀ:

Siamo interessati all'attività di rilascio di una versione del progetto software. In questa attività viene prima verificato che non ci siano richieste bloccanti. Se questo controllo fallisce viene stampato un messaggio di errore e l'attività termina. Se invece il controllo riesce vengono eseguite in parallelo due sottoattività; nella prima, il progetto viene compilato e poi testato; nel secondo, il progetto viene assemblato in un singolo archivio. Quando entrambe le sottoattività sono concluse, se il test è andato a buon fine il pacchetto viene rilasciato e viene stampato un messaggio di conferma. In caso contrario viene stampato un messaggio di errore.



DOMANDA 2

TABELLA RESPONSABILITÀ

fanno	richiede	SI 1
	utenti	SI 2
subincaso	richiede	SI 12
	progetti	SI 2
coinvolgono	file	NO
	richiede	SI 1
contiene	file	SI 1
	progetti	NO
creano	creatore	NO
	progetti	SI 1
mantengono	manutentore	SI 2
	progetti	SI 12

1. molteplicità
2. operazioni
3. requisiti

DOMANDA 3:

✓Richiede, RichiedeBloccanti, RichiesteNonBloccanti, ✓RichiedeFixed

✓TipoLinkFanno, ✓RuoliFanno

✓TipoLinkSubincaso, ✓RuoliSubincaso

✓AttivitàRiferita, ✓SottoAttività1, ✓SottoAttività2

Specifiche diagramma stati-transizione

InizioSpecificaDiStatoClasse Richiesta

Stato: {CREATA, CHIUSA, INVIATA, ATTESA}
Variabili di stato ausiliarie:
Manutentore: manutentore;
Stato inziale:
statoCorrente = CREATA;
manutentore = null;

FineSpecifica;

InizioSpecificaTransizioneClasse Richiesta

Transizione CREATA -> CHIUSA
Chiudi() {mitt: utente, dest: this}
Evento: chiudi() {mitt: utente, dest: this}
Condizione: utente.equals(this.getUtente());
Azione:
pre: --
post: this.statoCorrente = Stato.CHIUSA

Transizione CREATA -> INVIATA
Invia() {mitt: utente, dest: this}
Evento: invia() {mitt:utente, dest:this}
Condizione: utente.equals(this.getUtente());
Azione:
pre:--
post: this.statoCorrente = Stato.INVIATA

Transizione INVIATA -> CHIUSA
chiudi() {mitt: manutentore, dest: this}
Evento: chiudi() {mitt:utente, dest:this}
Condizione:
this.getTipoLinkSubiscono().getProgetto().getManutentore.contains(manutentore)
Azione:
pre:--
post: this.statoCorrente = Stato.CHIUSA

Transizione INVIATA -> ATTESA
chiudi() {mitt: manutentore, dest: this}
Evento: suggerimentoModifica() {mitt:utente, dest:this}
Condizione:
this.getTipoLinkSubiscono().getProgetto().getManutentore.contains(manutentore)
Azione:
pre:--
post: nuovoEvento = suggerisciModifica() {mitt:this, dest:utente} &&
this.statoCorrente = Stato.ATTESA;

Transizione ATTESA -> CHIUSA

chiudi() {mitt: utente, dest: this} }/informaManutentore(){dest:manutentore}

Evento: chiudi() {mitt:utente, dest:this}

Condizione: utente.equals(this.getUtente()) &&
this.getTipoLinkSubiscono().getProgetto().getManutentore.contains(manutentore);

Azione:

pre:

post: nuovoEvento = informaManutentore() {mitt: this, dest: manutentore} &&
this.statoCorrente = Stato.CHIUSA;

Transizione ATTESA -> INVIATA

modifica() {mitt: utente}/informaManutentore(){dest:manutentore}

Evento: modfica() {mitt:utente}

Condizione: utente.equals(this.getUtente()) &&
this.getTipoLinkSubiscono().getProgetto().getManutentore.contains(manutentore)

Azione:

pre: --

post: nuovoEvento = informaManutentore(){dest:manutentore} &&
this.statoCorrente = Stato.INVIATA

FIneSpecifica

Specifica attività

InizioSpecificaAttivitàPrincipale Rilascio

```
Rilascio();

VariabiliProcesso
        Progetto p;
        boolean ok;
        boolean test;

InizioProcesso
        Verifica(p) : (ok);
        if(ok == false) {SegnaliIO.errore();}
        else{
                fork{
                        thread t1 : {
                                sottoAttività1(Progetto p):(boolean test)
                        }
                        Thread t2: {
                                sottoAttività2(Progetto p);()
                        }
                }
                Join t1, t2;
                If(test == false){
                        SegnaliIO.errore();
                }
                else{
                        rilasciaPacchetto():()
                        SegnaliIO.stampaConferma;
                }
        }
FinoProcesso
```

FineSpecifica

InizioSpecificaAttivitàAtomica errore1()

```
Errore1()
Pre:--
Post: stampa messaggio di errore
```

FineSpecifica

InizioSpecificaAttivitàAtomica errore2()

```
Errore2()
Pre:--
Post: stampa messaggio di errore
```

FineSpecifica

InizioSpecificaAttivitàAtomica verifica1()
verifica1(Progetto: p):(boolean ok)
Pre:--
Post: if(ok)prosegui, else errore1();
FineSpecifica

InizioSpecificaAttivitàAtomica verifica2()
verifica2(Progetto p):(boolean test)
Pre:--
Post: if(test)prosegui, else errore2;
FineSpecifica

InizioSpecificaAttivitàComplessa sottoAttività1
sottoAttivià1(Progetto p):(Boolean test)
variabiliProcesso:
Progetto p;
boolean test;

InizioProcesso
Compila(Progetto p):();
test(Progetto p):(boolean test)
FineProcesso
FineSpecifica

InizioSpecificaAttivitàComplessa sottoAttività2
sottoAttività2(Progetto p):()
variabiliProcesso:
Progetto p;
InizioProcesso
Assembla(Progetto p):();
FineProcesso
FineSpecifica

InizioSpecificaAttivitaAtomica rilasciaPacchetto()
rilasciaPacchetto(boolean test, Progetto p):()
pre: test == true;
post:--

Realizzazione

```
public abstact class Richiesta implements Task{
        Private int codice;
        Private String descrizione;
        private TipoLinkSubiscono subiscono;        //1..1
        private TipoLinkFanno fanno;                //1..1
        private HashSet<File> coinvolgono;          //1..*

        public Richiesta(int c, String d){
                codice = c;
                descrizione = d;
                file = new HashSet<File>();
        }

        public int getCodice() { return codice; }
        public String getDescrizione() {return descrizione;}
        public void setCodice(int c) { codice = c ;}
        public void setDescrizione (String d) {descrizione = d;}

        //gestione associazione subiscono

        public void getTipoLinkSubiscono(){
                if(subiscono == null){
                        throw new EccezioneMolteplicita("violata molteplicità");
                }
                return subiscono;
        }

        public int quantiSubiscono(){
                if(subiscono == null) {
                        throw new EccezioneMolteplicita("violata molteplicità");
                }
                else return 1;
        }

        public void inserisciTipoLinkSubiscono(TipoLinkSubiscono t){
                if(t!=null && t.getRichieste().equals(this)){
                        ManagerSubiscono.inserisci(t);
                }
        }

        public void eliminaTipoLinkSubiscono(TipoLinkSubiscono t){
                if(t!=null && t.getRichieste().equals(this)){
                        ManagerSubiscono.elimina(t);
                }
        }
```

```
public void inserisciPerManagerSubiscono(ManagerSubiscono m){
        if(m!=null){
                subiscono = m.getLink();
        }
}

public void eliminaPerManagerSubiscono(ManagerSubiscono m){
        if(m!=null){
                subiscono = null
        }
}
```

//gestione associazione fanno

```
public void getTipoLinkFanno(){
        if(fanno == null){
                throw new EccezioneMolteplicita("violata molteplicità");
        }
        return fanno;
}

public int quantiFanno(){
        if(fanno == null) {
                throw new EccezioneMolteplicita("violata molteplicità");
        }
        else return 1;
}

public void inserisciTipoLinkFanno(TipoLinkFanno t){
        if(t!=null && t.getRichieste().equals(this)){
                ManagerFanno.inserisci(t);
        }
}

public void eliminaTipoLinkFanno(TipoLinkFanno t){
        if(t!=null && t.getRichieste().equals(this)){
                ManagerFanno.elimina(t);
        }
}


public void inserisciPerManagerFanno(ManagerFanno m){
        if(m!=null){
                fanno = m.getLink();
        }
}
```

```
public void eliminaPerManagerFanno(ManagerSubiscono m){
        if(m!=null){
                fanno = null
        }
}

//gestione associazione subiscono

public void getTipoLinkCoinvolgono(){
        if(coinvolgono.size()<1){
                throw new EccezioneMolteplicita("violata molteplicità");
        }
        return (Set<File>)coinvolgono.clone();
}

public int quantiCoinvolgono(){
        if(coinvolgono.size()<1){
                throw new EccezioneMolteplicita("violata molteplicità");
        }
        else return coinvolgono.size();
}

public void inserisciTipoLinkCoinvolgono(File F){
        if(f!=null){
                coinvolgono.add(f);
        }
}

public void eliminaTipoLinkCoinvolgono(File F){
        if(f!=null){
                coinvolgono.remove(f);
        }
}


//gestione oggetto reattivo

public static enum Stato = {CREATA, CHIUSA, INVIATA, ATTESA;}
Stato statoCorrente = Stato.CREATA;
Manutentore manutentore;
public Stato getStato() {return statoCorrente;}
public void fired(Evento e){
        TaskExecutor.getInstance().perform(new RichestaFired(this,e));
}

}
```

```
public class TipoLinkSubiscono {
        private Progetto p;
        private Richiesta r;
        private TipoLinkSubiscono(Progetto p, Richesta r){
                if(p==null ll r== null){
                        thrown new EccezionePrecondizioni(“precondizioni violate”); }
                else{
                        this.p = p;
                        this.r = r; }
        }

        public Progetto getProgetto() {return p;}
        public Richiesta getRichiesta() {return r;}
        public boolean equals(Object o){
                If(o!=null && getClass().equals(o.getClass())){
                        TipoLinkSubiscono t = (TipoLinkSubiscono o);
                        return t.getProgetto() == p && t.getRichesta() == r;
                }
                else return false;
        }

        public int hashCode(){ return p.hashCode(); r.hashCode(); }
}

Public class ManagerSubiscono {
        private final TipoLinkSubiscono t;
        private ManagerSubiscono (TipoLinkSubiscono t){
                        this.t = t;
        }

        public TipoLinkSubiscono getLink() { return t;)

        public void inserisci(TipoLinkSubiscono t){
                if(t!=null){
                        ManagerSubiscono m = new ManagerSubiscono(t);
                        t.getRichesta().inserisciPerManagerSubiscono(m);
                        t.getProgetto().inserisciPerManagerSubiscono(m);
                }
        }

        public void elimina(TipoLinkSubiscono t){
                if(t!=null){
                        ManagerSubiscono m = new ManagerSubiscono(t);
                        t.getRichesta().eliminaPerManagerSubiscono(m);
                        t.getProgetto().eliminaPerManagerSubiscono(m);
                }
        }
```

```
}
public class TipoLinkFanno {
        private Utente u;
        private Richiesta r;
        private TipoLinkSubiscono(Utente u, Richesta r){
                if(u==null ll r== null){
                        thrown new EccezionePrecondizioni(“precondizioni violate”); }
                else{
                        this.u = u;
                        this.r = r; }
        }

        public Utente getUtente() {return u;}
        public Richiesta getRichiesta() {return r;}
        public boolean equals(Object o){
                If(o!=null && getClass().equals(o.getClass())){
                        TipoLinkSFanno t = (TipoLinkFanno o);
                        return t.getUtente() == u && t.getRichesta() == r;
                }
                else return false;
        }

        public int hashCode(){ return u.hashCode(); r.hashCode(); }
}

Public class ManagerFanno {
        private final TipoLinkFanno t;
        private ManagerFanno (TipoLinkFanno t){
                        this.t = t;
        }

        public TipoLinkFanno getLink() { return t;)

        public void inserisci(TipoLinkFanno t){
                if(t!=null){
                        ManagerFanno m = new ManagerFanno (t);
                        t.getRichesta().inserisciPerManagerFanno(m);
                        t.getUtente().inserisciPerManagerFanno(m);
                }
        }

        public void elimina(TipoLinkFanno t){
                if(t!=null){
                        ManagerFanno m = new ManagerFanno(t);
                        t.getRichesta().eliminaPerManagerFanno(m);
                        t.getUtente().eliminaPerManagerFanno(m);
                }
        }
```

```
}
Public class AttivitaRilascio implements Runnable {
        Private boolean eseguita = false;
        private Progetto p;

        public attivitaRilascio(Progetto p){
                this.p = p;
        }

        Public synchronized void run(){
                if(eseguita) return;
                eseguita = true;
                while(eseguita){
                        Verifica v = new Verifica(p);
                        TaskExecutor.getInstance().perform(v);
                        boolean ok = v.getResult();
                        if(!ok){
                                SegnaliIO.errore();
                        }
                        else{
                                SottoAttività1 a1 =  new SottoAttivita1(p);
                                Thread t1 = new Thread(a1);
                                t1.start();
                                SottoAttività1 a2 =  new SottoAttivita2(p);
                                Thread t2 = new Thread(a2);
                                t2.start();
                                try{
                                        t1.join();
                                        t2.join();
                                }catch(InterruptedException e){
                                        e.printStackTrace();
                                }
                                boolean test = t1.getResult();
                                if(!test){
                                        SegnaliIO.errore();
                                }
                                else{
                                        RilasciaPacchetto r = new RilasciaPacchetto(p);
                                        TaskExecutor.getInstance().perform(r);
                                        SegnaliIO.stampaConferma();
                                }
                        }
                }
        }
        Public synchronized boolean estEseguita() { return eseguita; }
}
```

```
public class SottoAttivita1 implements Runnable{
	boolean eseguita = false;
	boolean test = false;
	Progetto p;
	public SottoAttivita1(Progetto p){
		this.p = p;
	}

	public synchronized void run(){
		if(eseguita) return;
		eseguita = true;
		Compila c = new Compila(p);
		TaskExecutor.getInstance().perform(c);
		Test t = new Test(p);
		TaskExecutor.getInstance().perform(t);
		test = t.getResult();
	}
	public synchronized boolean getResult() {return test;}
	public synchronized boolean estEseguita(){return eseguita;}

}

public class sottoAttivita2 implements Runnable{
	boolean eseguita = false;
	Progetto p;
	public SottoAttivita2(Progetto p){
		this.p = p;
	}

	public synchronized void run(){
		if(eseguita) return;
		eseguita = true;
		Assembla a = new Assembla(p);
		TaskExecutor.getInstance().perform(a);
	}
	public synchronized boolean estEseguita(){return eseguita;}
}
```

```
class RichiestaFired implements Task{

	private boolean eseguita = false;
	private Richiesta r;
	private Evento e;

	public RichiestaFired (Richiesta r, Evento e){
		this.r = r;
		this.e = e;
	}

	public synchronized void esegui(){
		if (eseguita ll (e.getDestinatario() != r && e.getDestinario() != null){return;}
		eseguita = true;
		switch(r.getStato()){

			case CREATA:
				if(e.getClass().equals(INVIA.class)){
					INVIA ii = (INVIA) e;
					if(r.getTipoLinkFatta().getUtente().equals(ii.getMittente()){
						r.statoCorrente = Stato.INVIATA;
					}
				}

				else if(e.getClass().equals(CHIUDI.class)) {
					CHIUDI ii = (CHIUDI)e;
					if(r.getTipoLinkFatta().getUtente().equals(ii.getMittente()){
						r.statoCorrente = Stato.CHIUSA;
					}
				}

				break;

			case CHIUSA:
				break;

			case ATTESA:
				if(e.getClass().equals(MODIFICA.class)){
					MODIFICA ii = (MODIFICA)e;
					if(r.getTipoLinkFatta().getUtente().equals(ii.getMittente()))
					&&r.getTipoLinkSubiscono().getProgetto().getManutentore.co
					ntains(r.manutentore));
						Envirioment.aggiungiEvento(new
						INFORMAMANUTENTORE());
					}
				}
```

```
                    else if(e.getClass.equals(CHIUDI.class)) {
                        CHIUDI ii = (CHIUDI) e ;
                        if(r.getTipoLinkFatta().getUtente == ii.getMittente() &&
                        r.getTipoLinkSubiscono().getProgetto().getManutentore.conta
                        ins(r.manutentore)){
                            Envirioment.aggiungiEvento(new
                            INFORMAMANUTENTORE());
                            r.statoCorrente = Stato.CHIUSA;
                    }

                    break;

                case INVIATA:
                    if(e.getClass().equals(SUGGERISCIMODIFICA.class)){
                        SUGGERISCIMODIFICA ii = (SUGGERISCIMODIFICA)e;
                        if(r.getTipoLinkSubiscono().getProgetto().getManutentore().c
                        ontains(ii.getMittente())){
                            if(r.getTipoLinkFatta().getUtente().equals(ii.getMittent
                            e()){
                                Envirioment.aggiungiEvento(new
                                SUGGERISCIMODIFICA());
                                r.statoCorrente = Stato.ATTESA;
                            }
                        }
                    }

                    else if(e.getClass().equals(CHIUDI.class)){
                        CHIUDI ii = (CHIUDI) e;
                        if(r.getTipoLinkSubiscono().getProgetto().getManutentore().c
                        ontains(ii.getMittente())){
                            r.statoCorrente = Stato.CHIUSA;
                        }
                    }

                    break;

                case default:
                    throw new RuntimeException("stato corrente non riconosciuto");
            }
        }
        public synchronized boolean estEseguita(return eseguita;)
}
```