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 **24 luglio 2020**
Tempo a disposizione: 3:30 ore

Requisiti. L'applicazione da progettare riguarda un sistema per gestire campagne pubblicitarie.

Delle campagne pubblicitarie interessano il nome, una descrizione (una stringa) e i creativi che ci lavorano. Di questi interessano il nome e l'email. Tra i creativi che lavorano ad una campagna pubblicitaria, uno è il responsabile della campagna. Ogni campagna pubblicitaria ha esattamente un responsabile ed un numero arbitrario di altri creativi che ci lavorano. Le campagne pubblicitarie sono costituite da diverse pubblicità. Ogni pubblicità è associata ad esattamente una campagna pubblicitaria. Ad una campagna pubblicitaria possono venire associate delle "idee", costituite da un titolo (una stringa) ed una descrizione (una stringa). Ogni idea è associata ad esattamente una campagna pubblicitaria ed è "creata" da esattamente un creativo, che può o meno lavorare alla campagna pubblicitaria stessa. Ogni creativo può creare al più 100 idee. Alcune idee sono dirompenti in quanto richiedono un cambiamento profondo della campagna pubblicitaria stessa ed in tal caso devono indicare una descrizione aggiuntiva che giustifichi il cambiamento proposto (una stringa).

Siamo interessati allo gestione delle idee. Una idea si trova inizialmente nello stato *creata*. Il suo creatore può decidere di chiuderla oppure di sostenerla. Nel primo caso va nello stato *chiusa*, altrimenti va nello stato *sostenuta*. Quando è nello stato *sostenuta*, un lavoratore della campagna pubblicitaria può decidere di chiuderla, oppure può mandare una richiesta di approfondimenti al suo creatore. In questo caso, l'idea passa nello stato *attesa*. Il creatore dell'idea puó decidere di chiuderla oppure di aggiornare la descrizione dell'idea stessa, in tal caso l'idea manda una notifica al lavoratore che lo ha richiesto e torna nello stato *sostenuta*.

Siamo interessati all'attività di rilascio di una campagna pubblicitaria. In questa attività viene prima verificato che non ci siano idee dirompenti. 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 (i cui dettagli non interessano); nella prima, la campagna pubblicitaria viene pubblicata per avere nuove idee per un certo periodo al termine del quale viene preparato un elenco delle idee proposte (una stringa); nella seconda, si inseriscono le nuove pubblicità associate ad essa, stilandone un elenco delle pubblicità inserite (una stringa). Quando entrambe le sottoattività sono concluse, viene stampato un rapporto finale con entrambi gli elenchi proposti.

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 *Idea*; diagramma delle attività di rilascio; specifica del diagramma stati e transizioni, riportando solo gli stati, e le variabili di stato ausiliarie, ma non la specifica delle transizioni; la segnatura delle attività complesse, delle attività atomiche e dei 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 `Idea` con la classe `IdeaFired`, le eventuali sottoclassi e le classi Java per rappresentare le *associazioni* di cui la classe `Idea` ha responsabilità.
- L'*attività di rilascio*, ma non le sue sotto-attività, e l'*attività atomica di verifica*.

DIAGRAMMA DELLE CLASSI

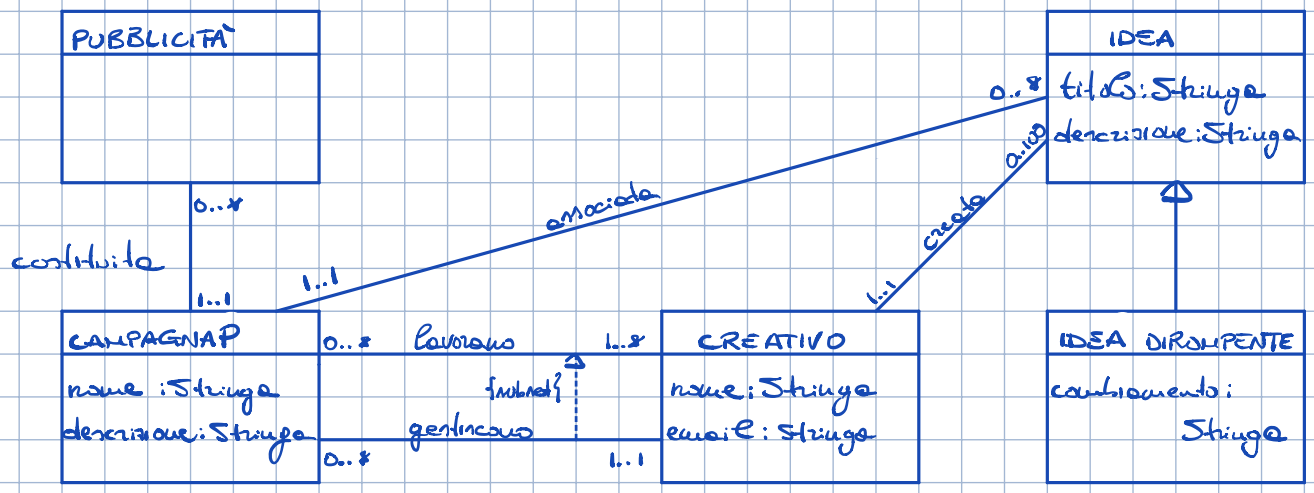
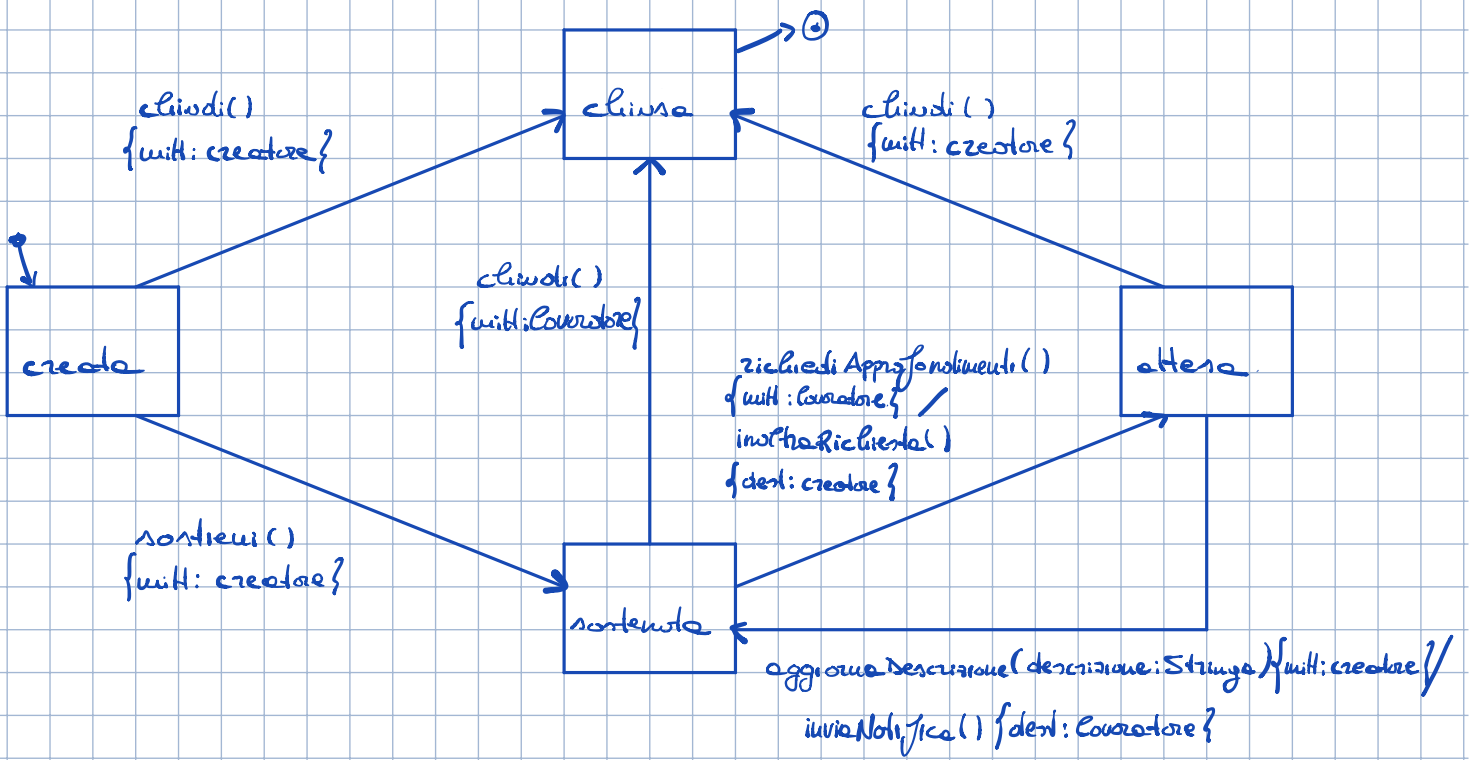


DIAGRAMMA STATI E TRANSIZIONI:



Var ausiliaria:

Lavoratore: Creativo

Nota: creatore: Creativo che può o meno appartenere all'insieme dei creativi che lavorano alla campagna.

Lavoratore: Creativo che appartiene all'insieme dei creativi che lavorano alla campagna.

DIAGRAMMA ATTIVITÀ

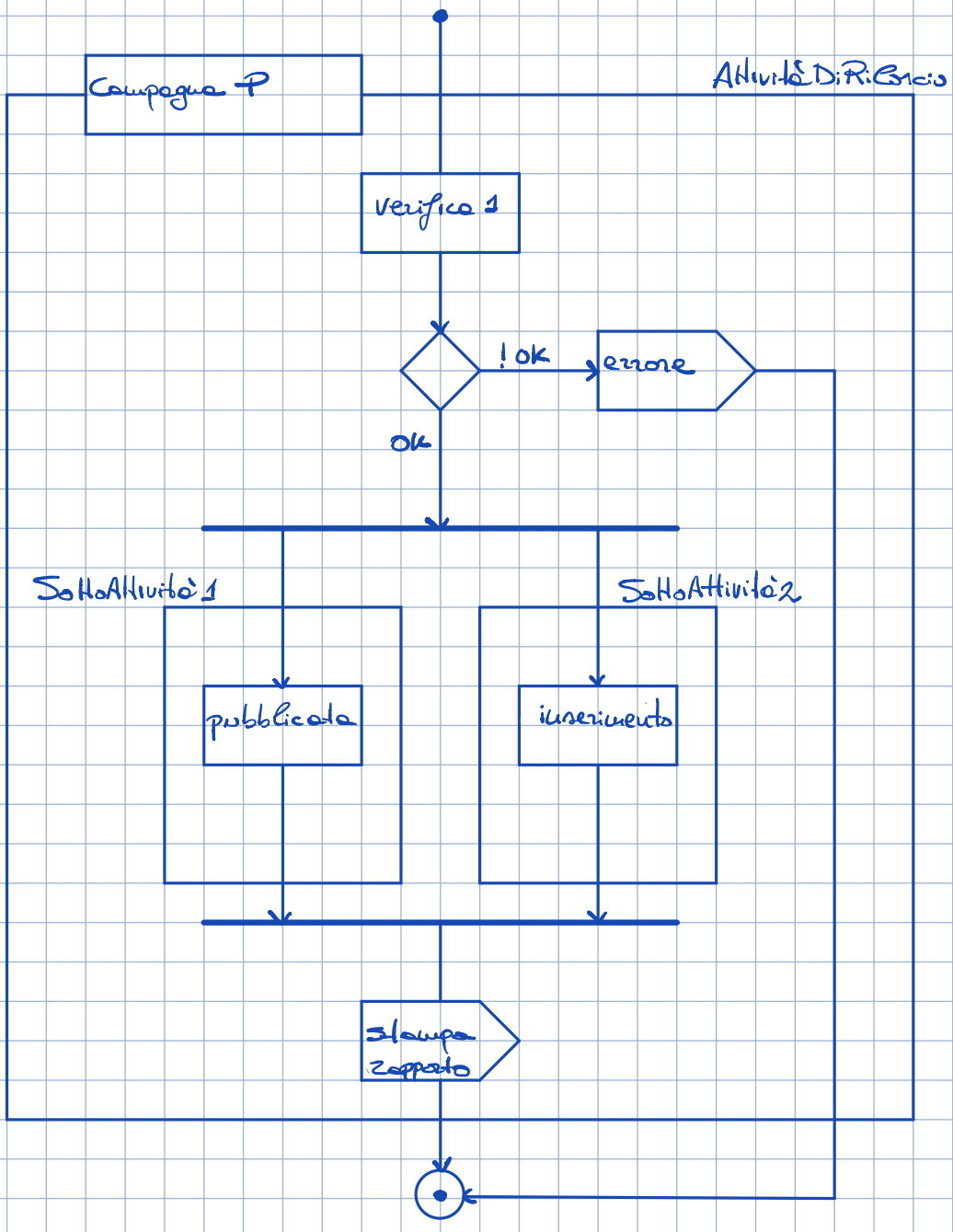
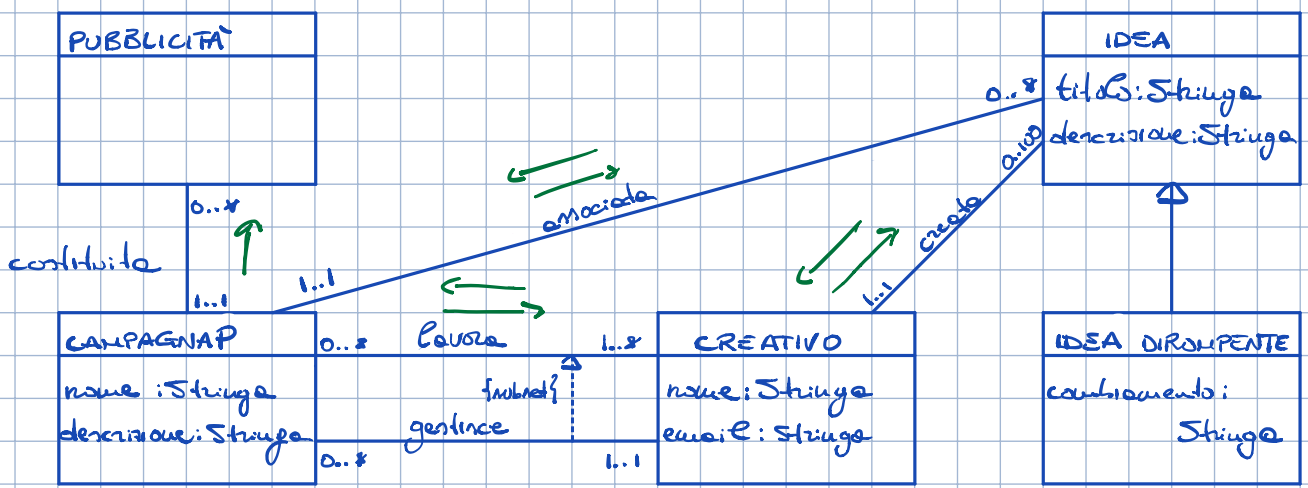


TABELLA RESPONSABILITA':

associata	idee	SI 1..2
	campagneP	SI 1..2
creata	idee	SI 1..2
	creativi	SI 1..2
lavora	campagneP	SI 1..2
	creativi	SI 1..2
gestisce	campagneP	SI 1
	creativo	NO
contenuta	campagneP	SI 1..2
	pubblicità	SI 1..*



Siamo interessati allo gestione delle idee. Una idea si trova inizialmente nello stato *creata*. Il suo creatore può decidere di chiuderla oppure di sostenerla. Nel primo caso va nello stato *chiusa*, altrimenti va nello stato *sostenuta*. Quando è nello stato *sostenuta*, un lavoratore della campagna pubblicitaria può decidere di chiuderla, oppure può mandare una richiesta di approfondimenti al suo creatore. In questo caso, l'idea passa nello stato *attesa*. Il creatore dell'idea puó decidere di chiuderla oppure di aggiornare la descrizione dell'idea stessa, in tal caso l'idea manda una notifica al lavoratore che lo ha richiesto e torna nello stato *sostenuta*.

Siamo interessati all'attività di rilascio di una campagna pubblicitaria. In questa attività viene prima verificato che non ci siano idee dirompenti. 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 (i cui dettagli non interessano); nella prima, la campagna pubblicitaria viene pubblicata per avere nuove idee per un certo periodo al termine del quale viene preparato un elenco delle idee proposte (una stringa); nella seconda, si inseriscono le nuove pubblicità associate ad essa, stilandone un elenco delle pubblicità inserite (una stringa). Quando entrambe le sottoattività sono concluse, viene stampato un rapporto finale con entrambi gli elenchi proposti.

Specifica diagramma stati-transizioni

InizioSpecificaStatiClasse Idea

Stato: {CREATA, SOSTENUTA, ATTESA, CHIUSA}

Variabili di stato ausiliarie:

lavoratore: Creativo;

StatoIniziale;

statoCorrente = CREATA;

FineSpecifica

Segnatura attività

AttivitàDiRilascio (c: CampagnaPubblicitaria) : ()	//attività complessa
SottoAttività1(c: CampagnaPubblicitaria) : (elenco1:Stringa)	//attività complessa
SottoAttività2 (c: CampagnaPubblicitaria) : (elenco2:Stringa)	//attività complessa
Verifica (c: CampagnaPubblicitaria) : (ok: Boolean)	//attività atomica
Pubblicata (c: CampagnaPubblicitaria) : (elenco1:Stringa)	//attività atomica
Inserimento (c: CampagnaPubblicitaria) : (elenco2:Stringa)	//attività atomica
Errore ():()	//segnaleIO
StampaRapporto (elenco1:Stringa, elenco2:Stringa):()	//segnaleIO

Realizzazione

Da realizzare:

- Idea
- IdeaFired
- TipoLinkAssociata, ManagerAssociata
- TipoLinkCreata, ManagerCreata
- AttivitaDiRilascio
- AttivitàDIVerifica

Classe Idea

```
Public class Idea implements Listener{
        private final String titolo;
        private String descrizione
        private TipoLinkAssociata associata;          //1..1
        private TipoLinkCreata creata;                //1..1
        public Idea(String t, String d){
                titolo = t;
                descrizione = d;
        }

        public String getTitolo(){return titolo;}
        public String getDescrizione(){return descrizione;}
        public void setDescrizione(String d){descrizione = d;}

        //gestione link associata

        public TipoLinkAssociata getTipoLinkAssociata(){
                if(associata == null){
                        throw new EccezioneMolteplicità("molteplicità violata");
                }
                else return associata;
        }

        public void inserisciTipoLinkAssociata(TipoLinkAssociata t){
                if(t!=null && t.getIdea().equals(this)){
                        ManagerAssociata.inserisci(t);
                }
        }

        public void eliminaTipoLinkAssociata(TipoLinkAssociata t){
                if(t!=null && t.getIdea().equals(this)){
                        ManagerAssociata.elimina(t);
                }
        }

        public void inserisciPerManagerAssociata(ManagerAssociata m){
                if(m!=null){
                        associata = m.getLink();
                }
        }

        public void eliminaPerManagerAssociata(ManagerCreata m){
                if(m!=null){
                        associata = null;
                }
        }
```

```
	//gestione link creata

	public TipoLinkCreata getTipoLinkCreata(){
		if(creata == null){
			throw new EccezioneMolteplicità(“molteplicità violata”);
		}
		else return creata;
	}

	public void inserisciTipoLinkCreata(TipoLinkCreata t){
		if(t!=null && t.getIdea().equals(this)){
			ManagerCreata.inserisci(t);
		}
	}

	public void eliminaTipoLinkCreata(TipoLinkCreata t){
		if(t!=null && t.getIdea().equals(this)){
			ManagerCreata.elimina(t);
		}
	}

	public void inserisciPerManagerCreata(ManagerCreata m){
		if(m!=null){
			creata = m.getLink();
		}
	}

	public void eliminaPerManagerCreata(ManagerCreata m){
		if(m!=null){
			creata = null;
		}
	}


	//gestione dell’oggetto reattivo
	public static enum Stato = {CREATA, SOSTENUTA, ATTESA, CHIUSA;}
	Creativo lavoratore;
	Stato statoCorrente = CREATA;
	public Stato getStato() {return statoCorrente;}
	public void fired(Evento e) {
		TaskExecutor.getInstance().perform(new IdeaFired(this, e));
	}


}
```

Classe ideaFired (Si trova nello stesso package di Idea)

```
class IdeaFired implements Task{
        private boolean eseguita = false;
        private Evento e;
        private Idea idea;
        public IdeaFired(Idea idea, Evento e){
                this.idea = i;
                this.e = e;
        }

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

                        case CREATA:
                                if(e.getClass().equals(Chiudi.class)){
                                        Chiudi ii = (Chiudi)e;
                                if(ii.getMittente().equals(idea.getTipoLinkCreata().getCreatore())){
                                        idea.statoCorrente = Stato.CHIUSA;
                                }
                                }
                                else if(e.getClass().equals(Sostieni.class)){
                                        Sostieni ii = (Sostieni)e;
                                if(ii.getMittente().equals(idea.getTipoLinkCreata().getCreatore())){
                                        idea.statoCorrente = Stato.SOSTENUTA;
                                }
                                }

                                break;
                                        If(idea.getLinkAssociata().getCampagnaP().getLinkLavora().contains(ii.getMittentte))
                        case SOSTENUTA:
                                if(e.getClass().equals(Chiudi.Class)){
                                        Chiudi ii = (Chiudi)e;
                                        idea.lavoratore = ii.getMittente();
                                        idea.statoCorrente = Stato.CHIUSA;
                                }
                                else if(e.getClass().equals(RichiediApprofondimenti.class)){
                                        RichediApprofondimenti ii = (RichediApprofondimenti)e;
                                        if(ii.getMittente().equals(idea.lavoratore)){
                                                Envirioment.aggiungiEvento(new InoltraRichiesta(idea,
                                                idea.getTipoLinkCreata().getCreatore()));
                                                idea.statoCorrente = Stato.ATTESA;
                                        }
                                }

                                break;
```

```
                    case ATTESA:
                        if(e.getClass().equals(Chiudi.class)){
                            Chiudi ii = (Chiudi)e;
                        if(ii.getMittente().equals(idea.getTipoLinkCreata().getCreatore())){
                            idea.StatoCorrente = Stato.CHIUSA;
                        }
                        }
                        else if(e.getClass().equals(AggiornaDescrizione.class)){
                            AggiornaDescrizione ii = (AggiornaDescrizione)e;
                        if(ii.getMittente().equals(idea.getTipoLinkCreata().getCreatore())){
                            String nuovaDescrizione = ii.getPayload();
                            idea.setDescrizione(nuovaDescrizione);
                            Envirioment.aggiungiEvento(new inviaNotifica(idea,
                            idea.lavoratore));
                            idea.StatoCorrente = Stato.SOSTENUTA;
                        }
                        }

                        break;

                    case CHIUSA:
                        break;

                    default:
                        throw new RuntimeException("Stato non riconosciuto");
                }
        }

        public synchronized boolean estEseguita(){return eseguita;}
}
```

Classe TipoLinkCreata

```
public class TipoLinkCreata{
        private Idea idea;
        private Creativo creativo;
        private TipoLinkCreata(Idea i, Creativo c){
                if(i==null ll c==null){
                        throw new EccezionePrecondizioni(“precondizioni violate”);
                }
                this.idea = i;
                this.creativo = c;
        }

        public Idea getIdea(){
                return idea;
        }

        public Creativo getCreativo(){
                return creativo;
        }

        public boolean equals(Object o){
                if(o!=null && getClass().equals(o.getClass())){
                        TipoLinkCreata t = (TipoLinkCreata)o;
                        return t.idea = idea && t.creativo = creativo;
                }
                else return false;
        }

        public int hashCode(){
                return idea.hashCode() + crativo.hashCode();
        }
}
```

Classe ManagerCreata

```
public class ManagerCreata{
	private final TipoLinkCreata t;
	private ManagerCreata(TipoLinkCreata t){
		this.t = t;
	}

	public inserisci(TipoLinkCreata t){
		if(t!=null){
			ManagerCreata m = new ManagerCreata(t);
			t.getIdea().inserisciPerManagerCreata(m);
			t.getCreativo().inserisciPerManagerCreata(m);
		}
	}

	public elimina(TipoLinkCreata t){
		if(t!=null){
			ManagerCreata m = new ManagerCreata(t);
			t.getIdea().eliminaPerManagerCreata(m);
			t.getCreativo().eliminaPerManagerCreata(m);
		}
	}

	public TipoLinkCreata getLink(){
		return t;
	}

}
```

Le classi TipoLinkAssociata e ManagerAssociata sono identiche a TipoLinkCreata e ManagerCreata;

Classe AttivitaDiRilascio

```
public class AttivitaDiRilascio implements Runnable{
	private CampagnaP c;
	private boolean eseguita = false;

	public AttivitòDiRilascio(CampagnaP c){
		this.c = c;
	}

	public synchronized void run(){
		if(eseguita) return;
		eseguita = true;
		while(eseguita){
			Verifica v = new Verifica(c);
			TaskExecutor.getInsance().perform(v);
			boolean ok = v.getResult();
			if(!ok){
				SegnaliIO.errore();
			}
			else{
				SottoAttivita1 a1 = new SottoAttivita1(c);
				SottoAttivita2 a2 = new SottoAttivita2(c);
				Thread t1 = new Thread(a1);
				Thread t2 = new Thread(a2);
				try{
					t1.join();
					t2.join();
				catch (InterruptedException e){
					e.printStackTrace();
				}
				SegnaliIO.stampaRapporto(a1.getResult(), a2.getResult());
			}
		}
	}

	public synchronized boolean estEseguita(){
		return eseguita;
	}

}
```

Classe Verifica

```
public class Verifica implements Task{
	private CampagnaP c;
	private boolean eseguita = false;
	private boolean ok;
	public Verifica(CampagnaP c){
		this.c = c;
	}

	public synchronized void esegui(){
		if(eseguita == false) return;
		eseguita = true;
		Iterator<TipoLinkAssociata> it = new Iterator<TipoLinkAssociata>();
		while(it.hasNext()){
			Idea idea = it.next().getIdea();
			if(idea.getClass().equals(IdeaDirompente.class)){
				ok = false;
			}
		}
		ok = true;;

	}

	public synchronized boolean getResult(){
		return ok;
	}

	public synchronized boolean estEseguita(){
		return eseguita;
	}

}
```