ESAME 26 GIUGNO 2020

1)

InizioSpecificaStati Richiesta:

Stati: {CREATA, CHIUSA, INVIATA, ATTESA}

Variabili di stato ausiliarie:
manutentore: Utente

Stato iniziale:
StatoCorrente = CREATA
manutentore = _

FineSpecifica

InizioSpecificaTransizioni Richiesta

Transizione: CREATA -> CHIUSA
Evento: chiusura(mitt = creatore, dest = this)
Condizione: creatore.equals(this.getCreatore())
Azione:
Pre: _
Post:_

Transizione: CREATA -> INVIATA
Evento: invio(mitt = creatore, dest = manutentori
Condizione: creatore.equals(this.getCreatore())
Azione:
Pre: _
Post:_

Transizione: INVIATA -> CHIUSA
Evento: chiusura(mitt = manutentore, dest = this)
Condizione: manutentore.estIn(this.getProgetto().getManutentori())
Azione:
Pre: _
Post: this.manutentore = null

Transizione: INVIATA -> ATTESA
Evento: suggerimento(mitt = manutentore, dest = this, payload = modifica)
Condizione: manutentore.estIn(this.getProgetto().getManutentori())
Azione:
Pre: _
Post: nuovoEvento = proponeModifica(mitt = manutentore, dest = creatore, payload = modifica), this.manutentore = manutetore

Transizione: ATTESA -> CHIUSA
Evento: chiusura(mitt = creatore, dest = this)
Condizione: creatore.equals(this.getCreatore())
Azione:
	Pre: _
	Post: nuovoEvento = notificaChiusura(mitt = creatore, dest = this.manutentore), this.manutentore = null

Transizione: ATTESA -> INVIATA
Evento: modifica(mitt = creatore, dest = this)
Condizione: creatore.equals(this.getCreatore())
Azione:
	Pre: _
	Post: nuovoEvento = notificaModifica(mitt = creatore, dest = this.manutentore)

FineSpecifica

InizioSpecificaAttività Rilascio

Rilascio(Progetto p):()

VariabiliProcesso
	p: Progetto
	okVerifica: boolean
	okTest: boolean

InizioProcesso
	verifica(p):(okVerifica) ;
	if(!okVerifica){
		errore1IO():() ;
		return ;
	}
	fork{
		thread t1:{
			primaSottoattività(p):(okTest) ;
		}
		thread t2{
			assemblaggio(p):() ;
		}
	join t1, t2;
	if(!okTest){
		errore2IO():() ;
		return ;
	}
	confermaIO():() ;
	return ;
FineProcesso

FineSpecifica

InizioSpecificaAttivitàAtomica Verifica

Verifica(p: Progetto):(okVerifica: boolean)
pre: _
post: boolean è true se il progetto non implica richieste bloccanti, false altrimenti

FineSpecifica

InizioSpecificaI/O Errore1

Errore1():()
pre: _
post: mando in output il messaggio “Errore1”

FineSpecifica

InizioSpecificaAttività PrimaSottoattività

PimaSottoattività(p: Progetto):(okTest: boolean)

Variabili di processo
    p: Progetto
    okTest: boolean

InizioProcesso
    compilazione(p):() ;
    test(p):(okTest) ;
FineProcesso

FineSpecifica

InizioSpecificaAttività Assemblaggio

Assemblaggio(p: Progetto):()

Variabili di processo
    p: Progetto

InizioProcesso
    assemblaggio(p):() ;
FineProcesso

FineSpecifica

InizioSpecificaAttivitàAtomica Compilazione

Compilazione(p: Progetto):()
pre: _
post: il progetto è compilato

FineSpecifica

InizioSpecificaAttivitàAtomica Test

Test(p: Progetto):(okTest: boolean)
pre: _
post: boolean è true se il test ha successo, false altrimenti

FineSpecifica

InizioSpecifical/O Errore2

Errore2():()
pre: _
post: manda in output il messaggio “Errore2”

FineSpecifica

InizioSpecifical/O Conferma

Conferma():()
pre: _
post: manda in output il messaggio “Conferma”

FineSpecifica

2)
Tabella responsabilità
1 molteplicità, 2 operazioni, 3 requisiti

Manutiene	Utente	2
	Progetto	1
Crea	Utente	no
	Progetto	1
Formula	Utente	2
	Richiesta	1, 2
Contiene	Progetto	no

	File	1
Associata	Richiesta	1
	File	no
Riguarda	Richiesta	1
	Progetto	3, 2

3)

Classe Richiesta

```
public abstract class Richiesta{ //astratta perché le sue sottoclassi sono complete, per permettere la derivazione i campi sono protected e non private
	protected int codice;
	protected String descrizione;
	protected HashSet<File> associata; //i file a cui è associata, almeno uno
	protected TipoLinkRiguarda riguarda; //il progetto che riguarda la richiesta
	protected TipoLinkFormula creatore; //l'utente che crea la richiesta

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

	//gestione degli attributi semplici
	public int getCodice(){
		return codice;
	}

	public String getDescrizione(){
		return descrizione;
	}

	public void setDescrizione(String d){
		descrizione = d;
	}

	//gestione di associata con responsabilità unilaterale
	protected int quantiAssociata(){ //per gestire la dimensione minima
		return associata.getSize();
	}
```

```
	public set getAssociata(){
		if(quantiAssociata()<1) throw new EccezioneMolteplicità;
		return associata.clone();
	}

	public void inserisciAssociata(File f){
		associata.add(f);
	}

	public void elimnaAssociata(File f){
		if(quantiAssociata()<2) throw new EccezioneMolteplicità;
		associata.remove(f);
	}

	//gestione di riguarda con responsabilità doppia (la gestione di creatore è identica e non viene riportata)
	public TipoLinkRiguarda getLinkRiguarda(){
		if(riguarda == null) throw new EccezioneMolteplicità;
		return riguarda; //oppure riguarda.clone()??
	}

	public void inserisciLinkRiguarda(TipoLinkRiguarda l){
		if(riguarda != null || l.getRichiesta !=this) throw new EccezionePrecodizioni;
		ManagerRigurda.inserisci(l);
	}

	public void eliminaLinkRiguarda(TipoLinkRiguarda l){ //non so se la parte di elimina va fatta visto che la molteplicità è 0..1
		if(riguarda == null || l.getRichiesta !=this) throw new EccezionePrecodizioni;
		ManagerRigurda.elimina(l);
	}

	public void inserisciPerManagerRigurda(ManagerRiguarda m){
		riguarda = m.getLink();
	}

	public void eliminaPerManagerRigurda(ManagerRiguarda m){
		riguarda = null;
	}

	//gestione dello stato

	public static enum Stato{CREATA, INVIATA, ATTESA, CHIUSA}

	statoCorrente = Stato.CREATA;
	manutentore = null;

	public Stato getStato(){
```

```
                return statoCorrente;
        }

        public void Fired(Evento e){
                TaskExecutor.getInstance().perform(new RichiestaFired(this,e));
        }
}
```

Classe TipoLinkRiguarda

```
public class TipoLinkRiguarda{
        public final Richiesta laRichiesta;
        public final Progetto ilProgetto;

        public TipoLinkRiguarda(Richiesta r, Progetto p){
                if(r == null || p == null) throw new EccezionePrecondizioni;
                laRichiesta = r;
                ilProgetto = p;
        }

        public Richiesta getRichiesta(){
                return laRichiesta;
        }

        public Richiesta getProgetto(){
                return ilProgetto;
        }

        public int hashCode(){
                return laRichiesta.hashCode() + ilProgetto.hashCode();
        }

        public boolean equals(Object o){
                if(o!=null && this.getClass().equals(o.getClass())){
                        TipoLinkRiguarda t = (TipoLinkRigurda)o;
                        return t.getRichiesta() == laRichiesta && t.getProgetto == progetto;
                }
                return false;
        }
}
```

Classe ManagerRigurda

```
public final class ManagerRiguarda{
        private final TipoLinkRiguarda link;

        private ManagerRiguarda(TipoLinkRiguarda l){
                link = l;
```

```
	}

	public TipoLinkRiguarda getLink(){
		return link;
	}

	public static void inserisci(TipoLinkRiguarda l){
		if(l!=null && l.getRichiesta().getRiguarda() == null){
			ManagerRiguarda m = new ManagerRiguarda(l);
			l.getRichiesta().inserisciPerManagerRiguarda(m);
			l.getProgetto().inserisicPerManagerRiguarda(m);
		}
	}

	public static void elimina(TipoLinkRiguarda l){
		if(l!=null){
			ManagerRiguarda m = new ManagerRiguarda(l);
			l.getRichiesta().eliminaPerManagerRiguarda(m);
			l.getProgetto().eliminaPerManagerRiguarda(m);
		}
	}
}
```

Classe RichiestaFired

```
//nello stesso package di Richiesta

class RichiestaFired implements Task{
	private Richiesta r;
	private Evento e;
	private boolean eseguita = false;

	public RichiestaFired(Richiesta ric, Evento ev){
		r = ric;
		ev = e;
	}

	public synchronized void esegui(){
		if(eseguita || (e.getDest() != r && e.getDest() != null)) return;
		eseguita = true;
		switch(r.getStato()){
			case CREATA:
				if(e.getClass() == chiusura.class){
					r.statoCorrente = Stato.CHIUSA;
				}
				if(e.getClass() == invio.class){
					Environmente.aggiungiEvento(new
inviata(r,r.getProgetto().getManutentori()));
```

```
                                        r.statoCorrente = Stato.INVIATA;
                                }
                                break;
                        case INVIATA:
                                if(e.getClass() == chiusura.clas){
                                        r.statoCorrente = Stato.CHIUSA;
                                        r.manutentore = null;
                                }
                                if(e.getClass() == suggerimento.class){
                                        r.Manutentore = e.getMitt();
                                        Environment.aggiungiEvento(new
suggiritaModifica(r.getCreatore(), e.getModifica());
                                        r.statoCorrente = Stato.ATTESA;
                                }
                                break;
                        case ATTESA:
                                if(e.getClass() == modifica.class){
                                        Environment.aggiungiEvento(new
ModificaEffettuata(r.manutentore()),r);
                                        r.statoCorrente = Stato.INVIATA;
                                }
                                if(e.getClass() == chiusura.class){
                                        Environment.aggiungiEvento(new
ModificaRifiutata(r.manutentore()),r);
                                        r.statoCorrente = Stato.CHIUSA;
                                        r.manutentore = null;
                                }
                                break;
                        default:
                                throw new RuntimeException(“Stato non riconosciuto”);
                }
        }

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

Attività di rilascio

```
public class Rilascio implements Runnable{
        private Progetto p;
        private boolean eseguita = false;
        private TaskExecutor executor = TaskExecutor.getInstance();

        public Rilascio(Progetto p){
                this.p = p;
        }
```

```
	public synchronized run(){
		if(eseguita) return;
		eseguita = true;
		Verifica v = new Verifica(p);
		executor.perform(v);
		boolean okVerifica = v.getRis();
		if(!okVerifica){
			system.out.println(“Verifica fallita”);
			return;
		}
		PrimaSottoattività p = new PrimaSottoattività(p);
		Assemblaggio a = new Assemblaggio(p);
		Thread t1 = new Thread(p);
		Thread t2 = new Thread(a);
		t1.start():
		t2.start();
		try{
			t1.join();
			t2.join();
		} catch (InterruptedException e){
			e.printStackTrace();
		}
		boolean okTest = PrimaSottoattività.getRis();
		if(!okTest){
			System.out.println(“Test fallito”);
			return;
		}
		System.out.println(“Attività terminata con successo”);
		return;
	}

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

Prima sottoattività

```
public class primaSottoattività implements Runnable{
	private Progetto p;
	private boolean eseguita = false;
	private boolean okTest = false;
	private TaskExecutor executor = TaskExecutor.getInstance();

	primaSottoattività(Progetto p){
		this.p = p;
	}
```

```
	public synchronized void run(){
		if(eseguita) return;
		eseguita = true;
		Compilazione c = new Compilazione(p);
		executor.perform(c);
		Test t = new Test(p);
		executor.perform(t);
		okTest = t.getRis();
	}

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

Assemblaggio

```
public class Assemblaggio implements Runnable{
	private Progetto p;
	private boolean eseguita = false;
	private TaskExecutor executor = TaskExecutor.getInstance();

	Assemblaggio(Progetto p){
		this.p = p;
	}

	public synchronized void run(){
		if(eseguita) return;
		eseguita = true;
		AssemblamentoAtomico a = new AssemblamentoAtomico(p);
	}

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