Programmazione 3
Appello d'esame – 26 gennaio 2015

Nome..... Cognome.....

Matricola.....

Non si possono consultare appunti e libri.

Esercizio 1

Si chiede di modellare un semplice sistema di *continuous integration* composto di 18 sviluppatori che concorrentemente apportano modifiche ad uno dei 4 moduli software contenuti in un repository centralizzato. In particolare, ogni sviluppatore modifica una copia del codice sorgente e la integra poi al repository solo dopo aver controllato che nel frattempo nessun altro abbia integrato altre modifiche allo stesso modulo software.
In particolare si chiede di:

- completare la definizione della classe `Repository` con la definizione dei metodi:
  - `Modulo getCopyM(int i)` che restituisce una **copia** del modulo `i`-esimo;
  - `boolean testM(int i, Modulo m)` che controlla se `m` è una copia del modulo `i`-esimo;
  - `void setM(int i, Modulo m)` che modifica il modulo `i`-esimo del repository assegnandogli il valore `m`;
  - `public void begin(int n)` che avvia un thread che chiama il metodo `build()` **dopo che** tutti gli `n` sviluppatori hanno terminato le loro modifiche.
- definire il metodo `run` della classe `Sviluppatore` in modo tale che effettui la seguente sequenza di passi:
  1. crea il riferimento `originale` inizializzato con una copia del modulo numero `nMod`,
  2. crea un'ulteriore copia di questo modulo, su cui invoca il metodo `modifica(''modificato da sviluppatore ''+nome)`,
  3. se il modulo numero `nMod` nel repository è ancora uguale al valore di `originale`, allora aggiorna il modulo del repository con il modulo ottenuto al punto 2, altrimenti ricomicia dal punto 1.
- completare la definizione della classe `CI` in modo tale che la chiamata del metodo `end` nell'ultima riga del metodo `main` provochi la stampa della stringa `fine main` **quando tutti gli sviluppatori hanno terminato**.
- **ATTENZIONE:** si chiede di indicare se l'esecuzione del programma ottenuto provoca sicuramente come ultima stampa la stringa `fine main`. **Giustificare brevemente la risposta.**
- È possibile aggiungere (pochi) campi e metodi ad ognuna delle classi del programma.

```
class Modulo{
     private static int n=0;
     private String contenuto;
     Modulo(){contenuto="modulo n."+n; n=n+1;}
     private Modulo(Modulo m){n=m.n; contenuto=m.contenuto;}
     public String toString(){return contenuto;}

     void modifica(String s){
         try{ contenuto=contenuto+s;
              Thread.sleep((int)(Math.random()*60));
         }catch(Exception e){}
     }
     Modulo copia(){return new Modulo(this);}
}

class Repository {
    private Modulo[] codiceSorgente;
    private int totS;
```

```
    Repository(){
        codiceSorgente=new Modulo[4];
        for(int i=0;i<codiceSorgente.length;i++) codiceSorgente[i]=new Modulo();
    }
    public String toString(){
        String s="Contenuto del repository:\n";
        for(Modulo m:codiceSorgente) s=s+m+"\n";
        return s;
    }
    private void build(){ System.out.println(this);}

   Modulo getCopyM(int i)
   boolean testM(int i, Modulo m)
   void setM(int i, Modulo m)
   public void begin(int n)
}

class Sviluppatore extends Thread{
    private Repository r;
    private String nome;
    private int nMod;
    Sviluppatore(Repository rep, String n){
      r=rep;nome=n; nMod=((int)(Math.random()*10)%4);//sceglie un modulo a caso da modificare
    }
    public void run()
}

public class CI{
    public static void main(String[] a){
        Repository r=new Repository();
        Sviluppatore[] elencoS=new Sviluppatore[18];
        for(int i=0;i<elencoS.length;i++) elencoS[i]=new Sviluppatore(r,"sviluppatore "+i);
        r.begin(elencoS.length);
        for(Sviluppatore s : elencoS) s.start();
        end(...);
    }
}
```

Esercizio 2

Si considerino le seguenti definizioni di classi, condivise da un programma client e un programma server.

```
public interface I extends Serializable{
    String get();
    void set(String s);
}
public class IImpl implements I{
    private String s;
    IImpl(String s){this.s=s;}
    public String get(){return s;}
    public void set(String s){ this.s=s;}
    public String toString(){ return s; }
}
public interface C extends Remote{
    void aggiungi(I i) throws RemoteException ;
    I get(int index) throws RemoteException;
    String stampa() throws RemoteException;
    Vector<I> getAll() throws RemoteException;
}
class T extends Thread {
      private C c;
      private String s;
      T(C cc, String ss){c=cc; s=ss;}
      public void run(){
          try{System.out.println(" in parallelo "); c.aggiungi(new IImpl(s));
          }catch(RemoteException e){}
      }
}
```

Si consideri inoltre la seguente definizione di classe che implementa un oggetto remoto di tipo `C`. Si ricorda inoltre che il tipo `Vector<E>` implementa l'interfaccia `Serializable`.

```
public class CImpl extends UnicastRemoteObject implements C{
    private Vector<I> v=new Vector<I>();
    private Vector<String> log=new Vector<String>()
    CImpl() throws RemoteException{ }

    public synchronized void aggiungi(I i) throws RemoteException {
      log.add("inizia aggiunta elemento "+i);
      v.add(i);
      log.add("aggiunto elemento "+i);
    }
    public synchronized I get(int index) throws RemoteException {
        log.add("get elemento "+index+"-esimo");
        return v.get(index);
    }
    public synchronized Vector<I> getAll() throws RemoteException{ return v; }

    public synchronized String stampa(){
      log.add("prepara la stringa totale ");
      String s=v.toString();
      log.add("pronta la stampa");
      return s;
    }
    public synchronized int size() { return v.size(); }
}
```

```
public class RemotoServer{
 public static void main(String[] a)
                          throws Exception{
  CImpl ci=new CImpl();
  Naming.rebind("pippo",ci);
  ci.aggiungi(new IImpl("A"));
  ci.aggiungi(new IImpl("B"));

  /***** PUNTO DI SINC. 1 ****/

  System.out.println("1:"+ci.stampa());




  /***** PUNTO DI SINC. 2 ****/

  T t1=new T(ci,"R");
  T t2=new T(ci,"S");
  System.out.println("2:"+ci.stampa());
  t2.start();
  t1.start();
  System.out.println("4: size="+ci.size());

  /***** PUNTO DI SINC. 3 ****/

  System.out.println("5:"+ci.stampa());




  /***** PUNTO DI SINC. 4 ****/

   I i=ci.get(0);
   i.set("P");
   System.out.println("2:ci="+ci.stampa());

   Vector<I> vv=ci.getAll();
   vv.add(i);
   System.out.println("6:"+vv.toString());
   System.out.println("5:"+ci.stampa());
 }
}
```

```
public class RemotoClient{

 public static void main(String[] a)
                     throws Exception{


   C c=(C) Naming.lookup("pippo");
   c.aggiungi(new IImpl("C"));

   /***** PUNTO DI SINC. 1 ****/

   System.out.println("1:"+c.stampa());
   T t=new T(c,"K");
   t.start();
   System.out.println("2:"+c.stampa());

   /***** PUNTO DI SINC. 2 ****/









   /***** PUNTO DI SINC. 3 ****/
   I i=c.get(0);
   i.set("Q");
   System.out.println("2:c="+c.stampa());

   Vector<I> vv=c.getAll();
   vv.add(i);
   System.out.println("4:"+vv.toString());
   System.out.println("5:"+c.stampa());

   /***** PUNTO DI SINC. 4 ****/









  }
}
```

Si assuma che l'esecuzione di ognuno dei due programmi non superi il "punto di sincronizzazione" fino a quando anche l'altro programma non abbia raggiunto il corrispondente "punto di sincronizzazione". Si assuma anche che i thread avviati terminino prima di oltrepassare il successivo punto di sincronizzazione. Usando uno schema simile al seguente, indicare tutte le possibili stampe prodotte dall'applicazione **assumendo che la prima stampa prodotta sia dal client che dal server sia sempre `A B C`**.

```
STAMPE DEL SERVER
      /***** PUNTO DI SINC 1 ****/
   1: A B C
      ..............................
      /***** PUNTO DI SINC 2 ****/
      ..............................
     /***** PUNTO DI SINC 3 ****/
      ..............................
      /***** PUNTO DI SINC 4 ****/
      ..............................
```

```
STAMPE DEL CLIENT
       /***** PUNTO DI SINC 1 ****/
   1: A B C
      ..............................
      /***** PUNTO DI SINC 2 ****/
      ..............................
      /***** PUNTO DI SINC 3 ****/
      ..............................
      /***** PUNTO DI SINC 4 ****/
      ..............................
```