Esame di Programmazione III, Programmazione in Rete e Laboratorio, Ist. di Programmazione in Rete e Laboratorio, a.a. 2022/23 - Appello IN ITINERE del 2 maggio 2023

Esercizio 1 (10 punti)
Si consideri il codice riportato sotto. Si valuti se i thread del programma possono interferire o meno l'uno con l'altro nell'accesso alle variabili condivise e **si spieghi dettagliatamente il perché**. **In caso positivo, modificare il codice** dell'applicazione per risolvere i problemi, preservando il massimo **parallelismo** nell'esecuzione delle operazioni ma **facendo in modo che pari.add();pari.hello() vengano eseguiti senza interruzioni** e analogamente per dispari.add();dispari.hello().

```
class MyThread extends Thread {
        private Contatore pari;
        private Contatore dispari;
        private Random r;

        public MyThread(Contatore o1, Contatore o2) {
                pari = o1; dispari = o2;
                r = new Random();
                start();
        }

        public void run() {
                while (true) {
                        int numero = r.nextInt(4);
                        if (numero%2==0) {
                                pari.add(numero); pari.hello(); }
                        else {
                                dispari.add(numero); dispari.hello(); }
                        System.out.println(Thread.currentThread().getName() +
                                                ": " + pari.getVal() + " - " + dispari.getVal());
                }
        }
}

class Contatore {
        private int count = 0;
        public void add(int val) { count = count+val; }
        public int getVal() { return count; }
        public void hello() { System.out.println("hello" + count); }
}
public class Es1 {
        public static void main(String[] args) {
                Contatore c1Pari = new Contatore();
                Contatore c2Dispari = new Contatore();
                MyThread t1 = new MyThread(c1Pari, c2Dispari);
                MyThread t2 = new MyThread(c1Pari, c2Dispari);
        }
}
```

Esercizio 2 (10 punti)
Si consideri il seguente codice. La classe **Point** rappresenta punti bidimensionali, con le loro coordinate. Si sviluppi il metodo **clone()** della classe **Point** (il metodo deve restituire una copia dell'oggetto su cui viene invocato) **facendo overriding** del metodo omonimo di **Object**.

```
public class Point {
   private int x;
   private int y;

   public Point(int x, int y) {
         this.x = x;
         this.y = y;
   }

   public int getX() { return x; }
   public int getY() { return y; }
}
```

Esercizio 3 (10 punti)

- Si descriva in dettaglio il pattern Observer-Observable specificando quali sono gli elementi architetturali di base, quali metodi offrono e come interagiscono tra di loro.
- Si faccia un semplice esempio di utilizzo di tale pattern in un'applicazione java. NB: NON è necessario sviluppare un'applicazione con interfaccia grafica.

POSSIBILI SOLUZIONI

Esercizio 1

Bisogna sincronizzare client side per mantenere la garanzia che le operazioni all'interno del condizionale vengano fatte senza interruzioni. NB ho reso atomica anche la stampa finale, anche se poteva essere garantita con due synchronized separate anzichè combinate (il testo non richiedeva che tale istruzione fosse eseguita senza interruzione).

```
public void run() {
    while (true) {
        int numero = r.nextInt( bound: 4);
        if (numero % 2 == 0) {
            synchronized(pari) {
                pari.add(numero); pari.hello();
            }
        } else {
            synchronized (dispari) {
                dispari.add(numero); dispari.hello();
            }
        }
        synchronized (pari) {
            synchronized (dispari) {
                System.out.println(Thread.currentThread().getName() +
                        ": " + pari.getVal() + " - " + pari.getVal());
            }
        }
    }
}
```

Esercizio 3

```
class Point {
 private int x; private int y;
 public Point(int x, int y) {
         this.x = x; this.y = y;
 }
 public int getX() { return x; }
 public int getY() { return y; }
 public String toString() { return x + "," + y;
   }
 public Object clone() {
          return new Point(x, y);
 }
}
```