ESERCITAZIONE TIGA: Thread, TimerTask, applicazione del pattern produttore-consumatore al controllo degli approvvigionamenti (compito di esame del 25/06/04)

Un insieme di merci è costituito da diversi tipi di merci e per ogni merce da un certo numero di articoli. La classe Java StockSet realizza un insieme ordinato di merci che ha i seguenti costruttori e metodi

- **public** StockSet(**int** numberOfStocks) che definisce un insieme costituito da un numero di merci pari a numberOfStocks. Il costruttore imposta a zero il numero di articoli di ciascun tipo di merce.
- **public** String toString() che converte uno StockSet in uno String.
- **public boolean** geq(StockSet other) **throws** ArrayIndexOutOfBoundsException che ritorna **true** se questo insieme di merci è maggiore o uguale dell'insieme di merci other specificato come argomento. Un insieme di merci è maggiore o uguale di un altro se, per ogni tipo di merce, il numero di articoli nel primo insieme è maggiore o uguale del numero di articoli nel secondo insieme. I due insiemi devono essere costituiti dallo stesso numero di merci; altrimenti viene sollevata l'eccezione ArrayIndexOutOfBoundsException.
- **public void** order(StockSet ordered) **throws** ArrayIndexOutOfBoundsException che sottrae l'insieme di merci ordered da questo insieme di merci. La sottrazione di un insieme di merci da un altro consiste nel sottrarre, per ciascuna merce, il numero degli articoli del primo da quello del secondo. I due insiemi devono essere costituiti dallo stesso numero di merci; altrimenti viene sollevata l'eccezione ArrayIndexOutOfBoundsException.
- **public void** supply(StockSet supplied) **throws** ArrayIndexOutOfBoundsException che somma l'insieme di merci supplied a questo insieme di merci. La somma di un insieme di merci ad un altro consiste nel sommare, per ciascuna merce, il numero degli articoli del primo a quello del secondo. I due insiemi devono essere costituiti dallo stesso numero di merci; altrimenti viene sollevata l'eccezione ArrayIndexOutOfBoundsException.
- **public void** initRandom(int maxItems) che imposta il numero di articoli di ciascuna merce di questo insieme ad un valore casuale compreso tra 0 e maxItems.
- **public void** numberOfStocks() che ritorna il numero di tipi di merci di questo insieme.

Un *magazzino* è uno StockSet in cui le operazioni di order e supply possono essere eseguite in modo concorrente in accordo alla seguente condizione di sincronizzazione: l'esecuzione di un ordine può essere completata se la merce in magazzino è maggiore di quella ordinata; altrimenti, l'ordine deve essere sospeso fino a quando il magazzino non sarà approvvigionato.

Esercizio n. I. Il candidato realizzi la classe Stockhouse che modella un magazzino

Esercizio n. II. Il candidato realizzi inoltre un timer task di nome Report che va in esecuzione periodicamente ogni 3 secondi e stampa lo stato del magazzino per mezzo del metodo print.

```
>java Simulazione 3
 Content:       [       0       0       0       ]
 Order  of t0: -[       7       8       3       ]
 Supply of t1: +[       7       2       0       ]
 Supply of t2: +[       4       7       2       ]
 Content:       [       11      9       2       ]
 Order  of t3: -[       3       1       1       ]
 Order  of t3   dispatched
 Content:       [       8       8       1       ]
 Supply of t4: +[       3       5       7       ]
 Order  of t0   dispatched
 Content:       [       4       5       5       ]
```

Fig.1 – Esempio di funzionamento

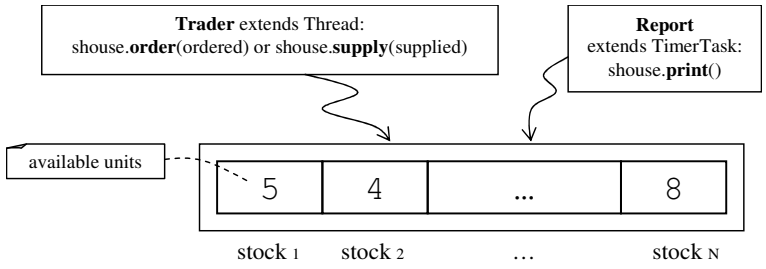

Fig.2 – Struttura dell'oggetto `shouse` di classe `Stockhouse`

TRACCIA PROPOSTA AL CANDIDATO:

```
// StockSet.java
public class StockSet {

    private int[]               stock;

    public StockSet(int numberOfStocks) {
        stock = new int[numberOfStocks];
        for (int i = 0; i < stock.length; i++)
            stock[i] = 0;
    }

    public String toString() {
        String result = "[\t";
        for (int i = 0; i < stock.length; i++)
            result += stock[i] + "\t";
        return result + "]";
    }

    public boolean geq(StockSet other) throws ArrayIndexOutOfBoundsException {
        if (stock.length!=other.numberOfStocks())
            throw new ArrayIndexOutOfBoundsException();

        for (int i = 0; i < stock.length; i++)
            if (stock[i] < other.stock[i])
                return false;
        return true;
    }

    public void order(StockSet ordered) throws ArrayIndexOutOfBoundsException {
        if (stock.length!=ordered.numberOfStocks())
            throw new ArrayIndexOutOfBoundsException();

        for (int i = 0; i < stock.length; i++)
            stock[i] -= ordered.stock[i];
    }

    public void supply(StockSet supplied) throws ArrayIndexOutOfBoundsException {
        if (stock.length!=supplied.numberOfStocks())
            throw new ArrayIndexOutOfBoundsException();

        for (int i = 0; i < stock.length; i++)
            stock[i] += supplied.stock[i];
    }

    public void initRandom(int maxItems) {
        for (int i = 0; i < stock.length; i++)
            stock[i] = (int)(Math.random()*maxItems);
    }

    public int numberOfStocks() {
        return stock.length;
    }
}

// Stockhouse.java
public class Stockhouse extends StockSet {

    // private part
    private static final int    DEFAULT_NUM_STOCKS = 5;

    public Stockhouse() {
        // …
    }

    public Stockhouse(int numberOfStocks) {
        // …
    }

    public synchronized void supply(StockSet supplied) {
        // …
    }

    public synchronized void order(StockSet ordered) {
        // …
    }

    public synchronized void print() {
        // …
    }

    public synchronized int numberOfStocks() {
```

```
        // …
    }
}


// Trader.java
public class Trader extends Thread {

    private Stockhouse house;

    public Trader(String name, Stockhouse house) {
        super(name);
        this.house = house;
    }

    public void run() {
        StockSet stock = new StockSet(house.numberOfStocks());
        stock.initRandom(10);

        if (Math.random()>= 0.5) {
            System.out.println (" Order  of " + getName() + ": -" + stock);
            house.order(stock);
            System.out.println (" Order  of " +  getName() + "   dispatched" );
        }
        else {
            System.out.println (" Supply of " + getName() + ": +" + stock);
            house.supply(stock);
        }
    }
}

// Report.java
import java.util.TimerTask;

public class Report extends TimerTask {
  // …
}

// Simulazione.java
import java.util.Timer;

public class Simulazione {

    public static void attesaRandom() {
        try {
            Thread.sleep(500 + (long)(Math.random()*2500));
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        final int N = Integer.parseInt(args[0]);

        Stockhouse house = new Stockhouse(N);

        // qui generare il TimerTask  Report e schedularlo

        for (int i=0; i<5; i++) {
            new Trader("t" + i, house).start();
            attesaRandom();
        }
   }
}
```