ESERCITAZIONE TIGA: Thread, pattern della sincronizzazione a barriera (compito di esame del 7/06/04)

Scrivere un programma Java che implementi il servizio di *sincronizzazione a barriera* per un insieme di thread che, dopo aver terminato una prima fase della propria attività, devono attendersi tra di loro prima di poter continuare (Fig.1).

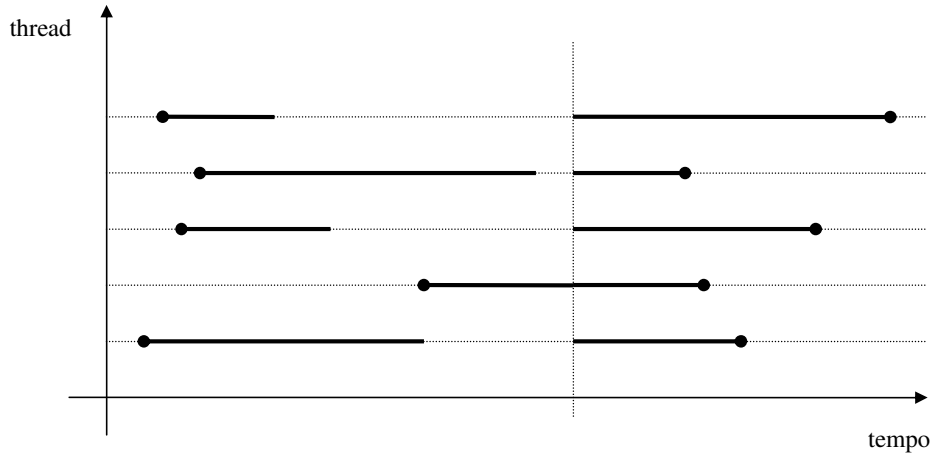

Fig.1 – Sincronizzazione a barriera

Ciascun thread (di classe `RandLifeThread`) attende per un tempo random e poi esegue sulla barriera (di classe `Barriera`) un metodo `waitAll()` che risulta bloccante sino a che è l'ultimo thread ad invocarlo, ed in tal caso esso risveglia tutti i thread.

ESEMPIO DI FUNZIONAMENTO:

```
> java Simulazione 5
 t0 partito...
 t0 sospeso...
 t1 partito...
 t2 partito...
 t3 partito...
 t1 sospeso...
 t4 partito...
 t3 sospeso...
 t2 sospeso...
 t4 sospeso...
 ---------------------
 t4 risvegliato...
 t1 risvegliato...
 t2 risvegliato...
 t0 risvegliato...
 t3 risvegliato...
 t1 terminato
 t4 terminato
 t2 terminato
 t0 terminato
 t3 terminato
```

TRACCIA PROPOSTA AL CANDIDATO:

```
// RandLifeThread.java
public class RandLifeThread extends Thread {
       //...
}

// Barriera.java
public class Barriera {
       //...
}

// Simulazione.java
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]);
        Barriera barriera = new Barriera(N);
        for (int i=0; i<N; i++) {
            new RandLifeThread("t" + i, barriera).start();
            attesaRandom();
        }
    }
}
```