Progetto Divisori

Cappellozza - Malovini - Spalenza

Overview

Lo scopo del programma è trovare il numero più piccolo avente un numero di divisori maggiore del precedente.

Con l’aumento del numero sotto processo, il calcolo diventa sempre più oneroso. Sfruttando una logica strettamente sequenziale, si nota un tempo di risposta sempre maggiore, inversamente proporzionale all’aumento del numero.

Per aumentare visibilmente le prestazioni e per mantenere una grafica responsiva abbiamo implementato una programmazione multithread.

Struttura generale

Di seguito vengono specificate le classi fondamentali del progetto dal punto di vista della gestione dei thread.

- Sequenza: rappresenta la sequenza di numeri trovati fino ad ora. Permette inoltre di aggiungere numeri alla lista e di inizializzarla.
- ProcessaNumeri: classe che sfrutta il multithreading per elaborare numeri in background. Permette l'avvio dei thread e la loro sincronizzazione. Rappresenta la logica pura del programma.
- ContaDivisori: rappresenta il corpo di ogni thread.

Implementazione

L’implementazione del multithreading è stata resa possibile dall’estensione della classe Thread.
I metodi che permettono l’utilizzo dei thread sono:
- run()
- start()

run() : contiene il codice del thread, invocato da start().
start() : permette di avviare il thread.

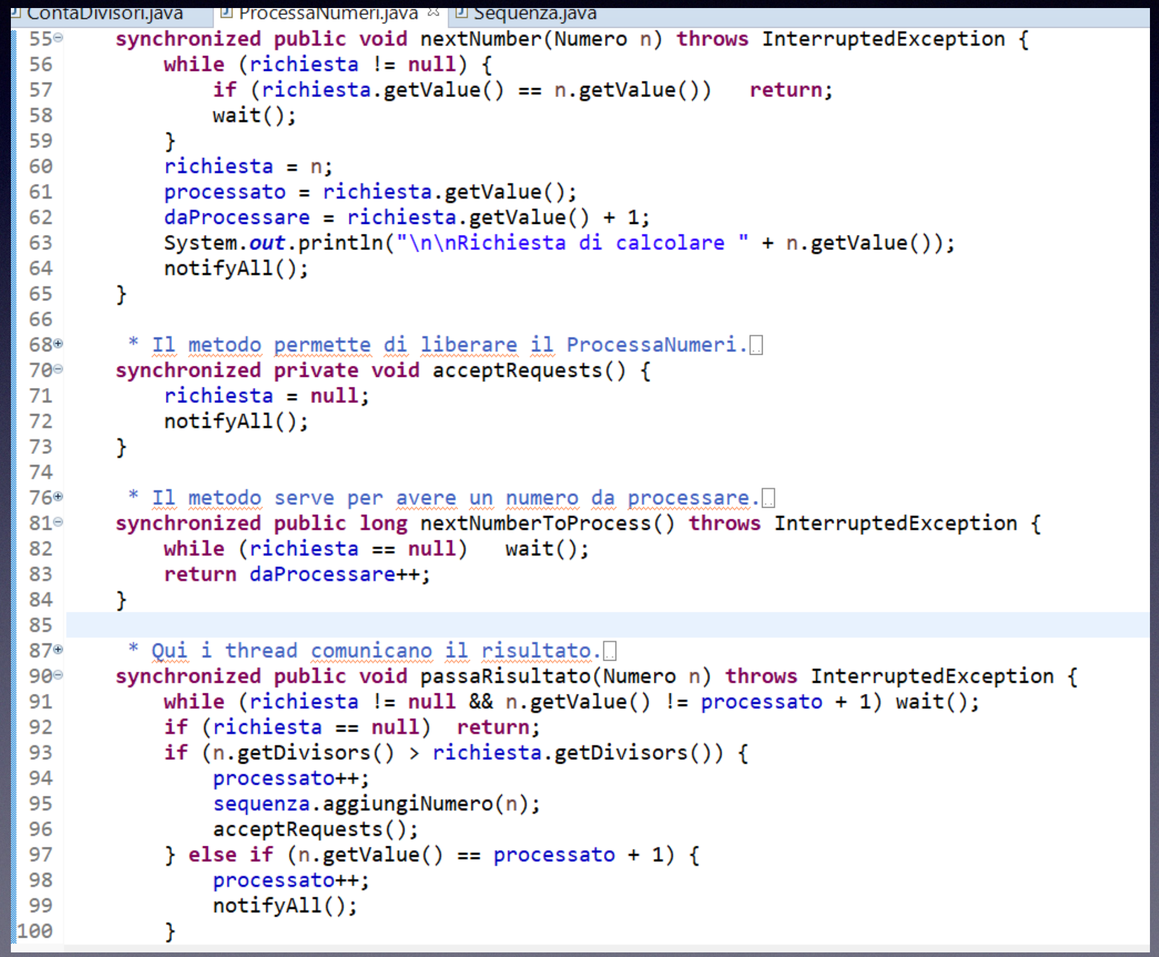
```
 */
public class ContaDivisori extends Thread {

    ProcessaNumeri numberProcessor;
    /**
    * Costruttore della classe ContaDivisori.
    * @param np
    *
    * */
    public ContaDivisori(ProcessaNumeri np) {
        numberProcessor = np;
    }
    /**
     * Azioni che esegue ogni thread.
     *
     */
    public void run() {
        while(true) {
            try {
                System.out.println(currentThread().getName()+ " richiede un nuovo numero da processare");
                long n = numberProcessor.nextNumberToProcess();
                System.out.println(currentThread().getName()+ " inizio calcolo "+ n);
                long d = contaDivisori(n);
                System.out.println(currentThread().getName() + " ha trovato " + n + " con " + d + " divisori");
                numberProcessor.passaRisultato(new Numero(n, d));
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }

    /**
```

```
/**
 * Inizia il calcolo con il numero di thread specificato.
 * @param nThread numero di thread
 */
public void startThreads(int nThread) {
    for (int i = 0; i < nThread; i++)
        new ContaDivisori(this).start();
}
```

Race condition

Nel multithreading, esiste la possibilità che si verifichino delle race condition; Per evitare questa condizione, abbiamo utilizzato i metodi synchronized che permettono di garantire la mutua esclusione sulle regioni critiche.

Non è stato necessario implementare blocchi synchronized poiché i metodi utilizzati sono sufficientemente brevi.


ContaDivisori.java | ProcessaNumeri.java | Sequenza.java

```
55      synchronized public void nextNumber(Numero n) throws InterruptedException {
56          while (richiesta != null) {
57              if (richiesta.getValue() == n.getValue())   return;
58              wait();
59          }
60          richiesta = n;
61          processato = richiesta.getValue();
62          daProcessare = richiesta.getValue() + 1;
63          System.out.println("\n\nRichiesta di calcolare " + n.getValue());
64          notifyAll();
65      }
66
68       * Il metodo permette di liberare il ProcessaNumeri.
70      synchronized private void acceptRequests() {
71          richiesta = null;
72          notifyAll();
73      }
74
76       * Il metodo serve per avere un numero da processare.
81      synchronized public long nextNumberToProcess() throws InterruptedException {
82          while (richiesta == null)   wait();
83          return daProcessare++;
84      }
85
87       * Qui i thread comunicano il risultato.
90      synchronized public void passaRisultato(Numero n) throws InterruptedException {
91          while (richiesta != null && n.getValue() != processato + 1) wait();
92          if (richiesta == null)  return;
93          if (n.getDivisors() > richiesta.getDivisors()) {
94              processato++;
95              sequenza.aggiungiNumero(n);
96              acceptRequests();
97          } else if (n.getValue() == processato + 1) {
98              processato++;
99              notifyAll();
100         }
```