Università degli Studi di Bologna
Facoltà di Ingegneria

Corso di
Reti di Calcolatori T

Esercitazione 0 (svolta)
Multithreading in Java

Luca Foschini

Anno accademico 2013/2014

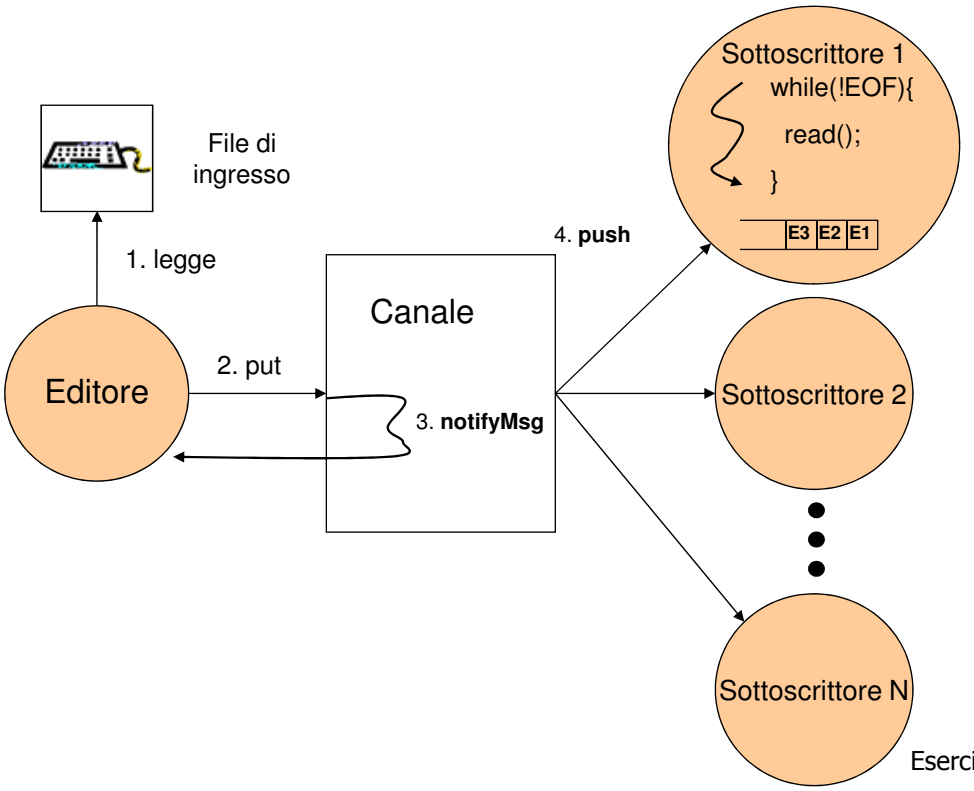
Modello ad eventi

L'architettura del **modello ad eventi** prevede l'interazione di **almeno due tipi di processi:**

- il **publisher** è il processo che produce gli eventi;
- il **subscriber** è il processo (o i processi, possono infatti esserci diversi subscriber interessati a registrarsi presso uno stesso publisher) che si registra presso il publisher per ricevere gli eventi.

Quando si presenta l'occorrenza di un evento il **publisher** notifica l'evento a **tutti i subscriber** registrati presso di lui e torna in ascolto di un nuovo evento.

Architettura di riferimento



Ulteriori dettagli

Nella realizzazione qui proposta oltre alle entità sopra definite (qui chiamate **Editore** e **Sottoscrittore**) viene introdotta una terza entità detta **Canale**.

Il Canale **non** è un **processo**, ma un **oggetto passivo** che mantiene la lista dei sottoscrittori e realizza i metodi per la notifica che vengono però eseguiti dal processo Editore, come evidenziato in figura.

Gli **eventi** sono, in questo caso, **stringhe lette** ciclicamente da console dal processo Editore. Un **piccolo programma test** verificherà il funzionamento delle entità progettate.

Java: Oggetti e Thread

Java fornisce due modalità per implementare thread:

- come **sottoclasse** della classe **`Thread`**
- come **classe** che implementa l'interfaccia **`Runnable`**

In ogni caso si noti bene che un singolo oggetto Java che implementi un thread ha una "doppia natura":

- di **oggetto passivo**: tutti i metodi ad esclusione del metodo **`run()`**
- di **oggetto attivo**: logica metodo **`run()`**

Specifica delle singole entità: Editore

L'**Editore** è un **processo** che legge ciclicamente da console (e quindi dal file di ingresso), **fino alla fine del file di ingresso**, la occorrenza di eventi rappresentati da stringhe inserite dall'utente, e li inserisce nel Canale.

Il processo deve comportarsi come **un filtro** che deve **consumare tutto il suo ingresso** fino a che non l'ha processato completamente, e ha segnalato altri processi dipendenti da questo

Specifica delle singole entità

I **Sottoscrittori** sono processi che ricevono gli eventi e li stampano a video, ed eseguono il seguente pseudo-codice:

1. elaborazione (attesa di un tempo casuale fra 2 e 5 sec.);
2. sottoscrizione al Canale;
3. esecuzione di un ciclo di ricezione di eventi fino a quando sono stati ricevuti e stampati a video **N eventi** (N numero intero casuale variabile, ad esempio, tra 2 e 6), oppure viene ricevuto un End Of File (**EOF**);
4. de-sottoscrizione dal Canale e terminazione.

I sottoscrittori sono **entità attive** con un **tempo di vita limitato** ed eseguono in modo ciclico il loro corpo fino alla terminazione

Specifica del Sottoscrittore

Ciascun **Sottoscrittore** internamente mantiene una coda (ad esempio un Vector) dove gli eventi inviati dal Canale vengono salvati, e implementa i seguenti metodi:

- `read`: metodo privato utilizzato per leggere il primo evento dalla coda degli eventi in arrivo. Il metodo è bloccante, cioè se la coda dei messaggi è vuota il Sottoscrittore viene bloccato → **NOTA:** come effetto la read consuma l'evento letto eliminandolo dalla coda degli eventi del Sottoscrittore;
- `push`: metodo pubblico utilizzato dal Canale per segnalare l'evento ad ogni Sottoscrittore, attraverso l'inserimento del messaggio relativo ed eventuale sblocco del Sottoscrittore in attesa

Suggerimento: per una migliore lettura dell'output (per identificare chi riceve il messaggio) si consiglia di aggiungere all'evento da stampare a video un campo intero, in modo da identificare l'i-esima istanza di Sottoscrittore. Tale campo sarà inizializzato al momento della costruzione del Sottoscrittore stesso.

Specifica delle singole entità: Canale

Il **Canale** (un oggetto passivo) deve mettere a disposizione i seguenti metodi:

- **`add/removeSubscriber`** per aggiungere e rimuovere i Sottoscrittori;
- **`put`**, per l'inserimento degli eventi da parte dell'Editore e notificare ordinatamente tutti i Sottoscrittori dell'arrivo dell'evento.

Si noti che se non ci sono Sottoscrittori registrati e l'Editore pubblica un evento, tale evento viene perso. Gli eventi sono cioè **non persistenti**.

La persistenza è gestita direttamente dalla scelta del mappaggio con le variabili Java

Codice: Editore

```
import java.io.*; /* Editore.java */

public class Editore extends Thread {
  private Canale canale;
  private final static String EOF = "-1";
  private BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

  public Editore(Canale c) { canale = c; } // Costruttore

  public void run() {
    String msg;
    System.out.println("\nEditore: inizio...");
    try {int i = 0;
       System.out.print("\n^D(Unix)/^Z(Win)+invio per uscire, invio per continuare: ");
       while ((msg = stdIn.readLine()) != null) { //while(!EOF)
        i++;
        System.out.print("E: messaggio " + i + " ? ");
        canale.put("" + msg);
        System.out.print("\n^D(Unix)/^Z(Win)+invio per uscire, invio per continuare: ");
      } //while, in uscita l'Editore ha letto EOF
    } catch (Exception e) { e.printStackTrace(); }
    System.out.println("E: termino...");
    canale.put(EOF);
  } //run()
} //Editore
```

Codice: Sottoscrittore 1/2

```
import java.util.Random; /* Sottoscrittore.java */
import java.util.Vector;

public class Sottoscrittore extends Thread {
 private Canale canale; private int identificatore = -1;
 private final static String EOF = "-1"; private Vector codaMessaggi = new Vector();
 // Costruttore
 public Sottoscrittore(Canale c, int id){ canale = c; identificatore = id; }

 public void run()
{try
{ Thread.sleep((int) (Math.random() * 3000)+2000);     //passo a.
  System.out.println("\nSottoscrittore(S"+identificatore+"): inizio e sottoscrizione...");
  canale.addSubscriber(this);                          //passo b.
  Random r = new Random(); int N = 2 + (r.nextInt(5));
  System.out.println("Il sottoscrittore S"+identificatore+" puo ricevere "+N+" messaggi");
  for(int i=0; i<N; i++){                              //passo c.
   String readMsg = read();
   if( readMsg.equals(EOF) ) break;                    //passo c.
   System.out.println("S"+identificatore+": "+readMsg);
  }
  Canale.removeSubscriber(this);                       //passo d.
  System.out.println("\nS"+identificatore: de-sottoscrizione e terminazione.");
 }
 catch(Exception e){ System.err.println("Errore: "+e); e.printStackTrace(System.err); }
 }  //run()
```


Codice: Sottoscrittore 2/2

```
 /**
   * Read: Chiamata sospensiva in caso la coda sia vuota.
   NOTA: la read consuma il messaggio dalla coda dei messaggi.
   *
   * @return il messaggio ricevuto, la stringa "-1" come EOF
     e null se ci sono problemi.
   */
  private synchronized String read() throws InterruptedException{
   if( codaMessaggi.size() == 0 ) wait();
   // Coda gestita con politica First In First Out (FIFO)
   String firstReceivedMessage = (String)
codaMessaggi.lastElement();
   codaMessaggi.remove(codaMessaggi.size()-1);
   return firstReceivedMessage;
  }  //read()

  public synchronized void push(String msg){
    codaMessaggi.insertElementAt(msg, 0);
    notify();
  }  //push()
```

Codice: Canale

```
import java.util.Vector;   /* Canale.java */
public class Canale {
 private Vector sottoscrittori = new Vector();
 /** Metodo per la pubblicazione di messaggi da parte degli Editori.
     Perché il metodo è synchronized??*/
public void synchronized put(String msg ) {
  int tot = sottoscrittori.size();
  if (sottoscrittori.isEmpty()) return;
  else /* recapito il msg a tutti i sottoscrittori */
  for (int i = 0; i < tot; i++)
  {Sottoscrittore sottoscrittoreIesimo = (Sottoscrittore)
   sottoscrittori.elementAt(i);
   sottoscrittoreIesimo.push(msg);
  }  //for
 }  //put()

public synchronized void addSubscriber(Sottoscrittore s)
  // Se il sottoscrittore non e' ancora stato registrato lo registro
{ if (!sottoscrittori.contains(s)) sottoscrittori.add(s);
 }  //addSubscriber()

public synchronized void removeSubscriber (Sottoscrittore s)
 //se il sottoscrittore esiste lo de-sottoscrivo
{ if (sottoscrittori.contains(s)) sottoscrittori.removeElement(s);
}  //removeSubscriber()
}//Canale
```


Codice: Main

Infine possiamo lanciare le entità implementate con un **programma di prova**:

```
/* testMain.java */
public class testMain {
 final static int NUM_SOTTOSCRITTORI = 5;

public static void main(String[] args)
{ Sottoscrittore sottoscrittore; Canale canale;  Editore editore;
  try
 { // Creo il canale e l'editore e faccio partire il thread
   canale = new Canale();
   editore = new Editore(canale);
   editore.start();
   for (int i = 0; i < NUM_SOTTOSCRITTORI; i++)
   { // Creo i Sottoscrittori e faccio partire i thread
    sottoscrittore = new Sottoscrittore(canale, i);
    sottoscrittore.start();
   }  //for
  } catch (Exception err)
   {System.err.println("Errore : "+err); System.exit(1);} //catch
}  //main

}  //testMain
```