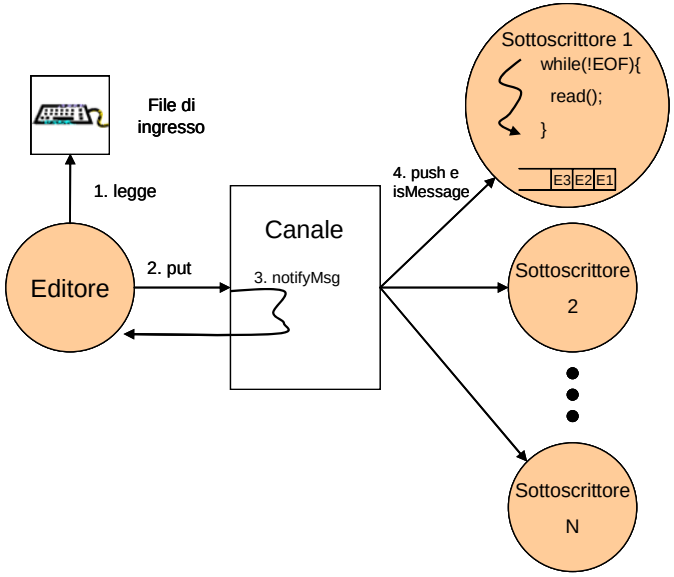
1° Esercitazione (svolta): Multithreading in Java

Modello ad eventi

L'architettura del **modello ad eventi** prevede l'interazione di **almeno due processi**, il publisher e il subscriber. Il **publisher**, è il processo che produce gli eventi, mentre 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.



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.

Specifica delle singole entità

- 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.
- I **Sottoscrittori** sono processi che ricevono gli eventi e li stampano a video, ed eseguono il seguente pseudo-codice:
  a. attesa di un tempo casuale fra 2 e 5 secondi;
  b. sottoscrizione al Canale;
  c. 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);
  d. de-sottoscrizione dal Canale e terminazione.

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;
- **`isMessage`**: metodo pubblico utilizzato dal Canale per notificare un Sottoscrittore, eventualmente bloccato su una read, dell'arrivo di un nuovo evento (sbloccandolo);
- **`push`**: metodo pubblico utilizzato dal Canale per fare la push di un evento al Sottoscrittore.

NOTA: come effetto collaterale la read consuma l'evento letto eliminandolo dalla coda degli eventi del Sottoscrittore.

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.

- 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;
  - **`notifyMsg`**, (privato) per copiare l'evento pubblicato dall'Editore nelle code degli eventi dei Sottoscrittori e notificare 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.

Nel seguito si riporta il codice che realizza l'esempio proposto.

```
/* Editore.java */

import java.io.*;

public class Editore extends Thread {
  private Canale canale;

  private final static String EOF = "-1";

  private BufferedReader stdIn =
    new BufferedReader(new InputStreamReader(System.in));

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

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

```
/* Sottoscrittore.java */

import java.util.Random;
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{
  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+" e' pronto a 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("Si e' verificato un errore, il "+
      "seguente: "+e);
  e.printStackTrace(System.err);
 }
 }  //run()
```



```
 /**
  * Chiamata sospensiva in caso la coda dei messaggi
  * sia vuota. NOTA: come effetto collaterale 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 void push(String msg){
  /**
   * push NON synchronized perche'
   * insertElementAt e' atomica.
   * Vector e' una struttura dati sincronizzata
   */
  codaMessaggi.insertElementAt(msg, 0);
 }  //push()

 public synchronized void isMessage(){
  notify();
 }  //isMessage()

}  //Sottoscrittore
```

```
/* Canale.java */

import java.util.Vector;

public class Canale {

 private Vector sottoscrittori = new Vector();


 /**
  * Metodo per la pubblicazione di messaggi da parte degli
  * Editori.
  *
  * Questo metodo va bene così o dovrebbe essere
  * synchronized? Perché?
  */
 public void put(String msg ) {
  if (sottoscrittori.isEmpty()) return;
  else notifyMsg(msg);
 }  //put()

 /**
  * Metodo per la notifica dei messaggi ricevuti ai
  * Sottoscrittori
  */
 private void notifyMsg(String msg) {
  /* recapito il msg a tutti i sottoscrittori */
  int tot = sottoscrittori.size();
  for (int i = 0; i < tot; i++) {
   Sottoscrittore sottoscrittoreIesimo =
      (Sottoscrittore) sottoscrittori.elementAt(i);
   sottoscrittoreIesimo.push(msg);
   sottoscrittoreIesimo.isMessage();
  }  //for
 }  //notifyMsg()

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
```

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 nel main, il seguente: "+
                                                 err);
  System.exit(1);
  }  //catch
 }  //main
}  //testMain
```