Decima Esercitazione

Accesso a risorse condivise tramite Monitor Java

Agenda

Esempio

L'album delle figurine: gestione di una risorsa condivisa da più thread, con politica prioritaria

Esercizio 1 – L'Anagrafe

Esercizio 2 – Il ponte

Esempio - La collezione di figurine (1/3)

Una casa editrice vuole realizzare un **sito web** dedicato ai collezionisti di figurine dell'album "***Campionato di calcio 2019-2020***".
L'album è composto da N=100 diverse figurine, ognuna individuata univocamente da un id intero [0, 99]; tra di esse:

- 30 sono classificate come **figurine rare** (id da 0 a 29),
- e le rimanenti 70 come **figurine normali** (id da 30 a 99).

Il sito offre un servizio che permette ad ogni utente collezionista di effettuare **scambi di figurine**.
A questo scopo il sistema gestisce un **deposito di figurine**, nel quale, **per ogni diversa figurina vi può essere più di un esemplare**.

Esempio - La collezione di figurine (2/3)

Il meccanismo di scambio, è regolamentato come segue:

- Si può scambiare solo **una figurina alla volta**, effettuando una **richiesta di scambio** con le seguenti regole:
  - ogni **utente U che desidera una figurina A può ottenerla, se a sua volta offre un'altra figurina B**;
  - in seguito a una richiesta di scambio, il **sistema aggiunge la figurina B** all'insieme delle figurine disponibili e **successivamente verifica se esiste almeno una figurina A** disponibile:
    - se **A è disponibile**, essa viene assegnata all'utente U, che può così continuare la propria attività;
    - se **A non è disponibile**, l'utente U viene messo in attesa.

Esempio - La collezione di figurine (3/3)

Si progetti la politica di gestione del servizio di scambio che tenga conto delle specifiche date e che inoltre soddisfi il seguente vincolo:

le richieste di **utenti che offrono figurine rare abbiano la precedenza sulle richieste di utenti che offrono figurine normali.**

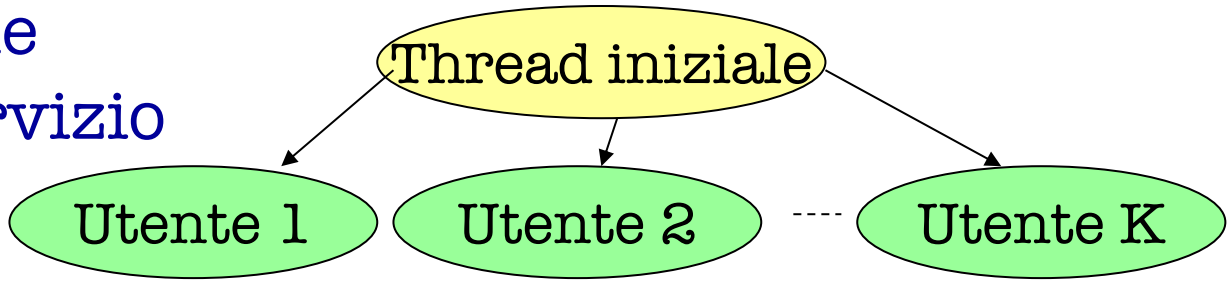
Ad esempio:

- Il thread TA chiede 7 offrendo 3[RARA] e 7 non disponibile: TA attende
- Il thread TB chiede 7 offrendo 50[NORM] e 7 non disponibile: TB attende
- 7 diventa disponibile => deve essere attivato TA(perchè offre una rara, quindi più prioritario).

Impostazione

Quali thread?

- il thread iniziale
- K utenti del servizio



Qual è la risorsa comune?

- **Deposito delle Figurine**
- associamo al Deposito un "**monitor**", che controlla gli accessi in base alla specifica politica di accesso. La sincronizzazione viene realizzata mediante **variabili condizione.**

Thread Collezionista

```
public class Collezionista extends Thread{
      private Monitor M;
      private int offerta, richiesta, N;

  public Collezionista(monitor m, int N){
            this.M=m;
            this.N=N;
  }
  public void run(){
      try { while (true){
            <definizione di offerta e richiesta>
            M.scambio(offerta, richiesta); //entry call
            Thread.sleep(…);
      }} catch (InterruptedException e) {}
  }
}
```

riferimento al monitor

numero di figurine diverse. Se ci atteniamo alle specifiche, N=100.

Monitor – Deposito figurine

Stato del Deposito:

Figurine disponibili: vettore di N=100 interi (uno per ogni figurina della collezione)

```
private int[] FIGURINE = new int[N];;
```

Dove: `FIGURINE[i]` è il numero di esemplari disponibili della figurina i.
(Hp: inizialmente 1 per ogni figurina**)**

Convenzione adottata:
Se i<30, si tratta di una **figurina rara;**
Se i>= 30, si tratta di una **figurina comune.**

Monitor – Deposito figurine

Lock per la mutua esclusione:

```
private Lock lock = new ReentrantLock();
```

Condition. Per la sospensione dei thread in attesa di una figurina, definiamo 2 condition (una per ogni livello di priorità):

```
private Condition rare= ….;
//thread sospesi che hanno offerto figurine rare
private Condition normali=…;
// thread sospesi che hanno offerto figurine
normali
```

Contatori dei thread sospesi in ogni coda:

```
private int[] sospRare = new int[N];
private int[] sospNormali = new int[N];
//devo sapere chi è sospeso in attesa di quella
specifica figurina dopo aver consegnato una
figurina rara o una normale
```

Monitor – Deposito figurine

```
public class Monitor {
  private final int N=100; //numero totale di figurine
  private final int maxrare=30;
  private int[] FIGURINE; //figurine disponibili
  private Lock lock = new ReentrantLock();
  private Condition rare = lock.newCondition();
  private Condition normali = lock.newCondition();
  private int[] sospRare;
  private int[] sospNormali;

  public Monitor() {…} //Costruttore
  public void scambio(int off,int rich)//metodo entry:
                    throws InterruptedException {…}
}
```

politica di sincronizzazione

Soluzione

```
import  java.util.Random;
public class Collezionista extends Thread{
        private Monitor M;
        private int offerta, richiesta, max;

        public Collezionista(Monitor m, int NF, String name){
                this.M=m; this.max=NF;
                this.setName(name);
        }

        public void run(){
        int op, cc, somma;
        try {   while (true)
                {       offerta= r.nextInt(max);
                        do {    richiesta= r.nextInt(max);
                        }while(richiesta==offerta);
                        M.scambio(offerta, richiesta);
                        Thread.sleep(250);
                }
        } catch (InterruptedException e) {    }
}}
```

Soluzione: monitor

```
import java.util.concurrent.locks.*;

public class Monitor{ //Dati:
        private  int N; //numero totale di figurine
        private final int maxrare;
        private int[] FIGURINE= new int[N];; //figurine disponibili

        private Lock lock= new ReentrantLock();
        private Condition rare= lock.newCondition();
        private Condition normali= lock.newCondition();
        private int[] sospRare= new int[N];
        private int[] sospNormali= new int[N];

        public Monitor(int N ) {
                int i;
                for(i=0; i<N; i++) {
                        FIGURINE[i]=1;//valore arbitrario
                        sospRare[i]=0;
                        sospNormali[i]=0;
                }
                maxrare=N/100*30; }
```

```
//metodi "entry":
public void scambio(int off, int rich) throws InterruptedException {
        lock.lock();
        try{    FIGURINE[off]++;
                if (sospRare[off]>0)
                        rare.signalAll();
                else if (sospNormali[off]>0)
                        normali.signalAll();
                if (off < maxrare) //ha offerto una figurina rara
                        while (FIGURINE[rich]==0){
                                sospRare[rich]++;
                                rare.await();
                                sospRare[rich]--; }
                else //ha offerto una normale
                        while (FIGURINE[rich]==0 || sospRare[rich]>0){
                                sospNormali[rich]++;
                                normali.await();
                                sospNormali[rich]--; }
                FIGURINE[rich]--; // tolgo la figurina scambiata
                } finally{ lock.unlock();}
                return;
        }
```

perchè signalAll?

Commenti finali

- La soluzione prevede solo 2 condition (rare, normali), ma le condizioni di sincronizzazione possibili sono 2*100: per ogni categoria (rare/normali), ogni thread si sospende in attesa di una particolare figurina.
- Per evitare le signalAll (poco efficienti) si potrebbero definire 2*100 condition:

```
private Condition []rare=new Condition[N];
//code thread che hanno offerto rare
private Condition []normali=new Condition[N];
//code thread hanno offerto normali
```

Esercizio 1 – l'Anagrafe

Si consideri l'ufficio dell'**Anagrafe** di un Comune.
L'ufficio è costituito da **N sportelli**, attraverso i quali vengono erogati al pubblico **2 diversi tipi di servizi**:

- Rilascio **Certificati** (CER)
- Rilascio **Carte d'Identità** (IDE)

Ogni sportello può eseguire un servizio alla volta (di qualunque tipo); per semplicità, si assuma che ogni utente richieda un solo servizio alla volta.

L'erogazione di un servizio a un utente presuppone **l'acquisizione di uno sportello libero** da parte dell'utente richiedente.
Si assuma inoltre che **la permanenza di un utente** allo sportello abbia una **durata non trascurabile**.

Per servire le richieste degli utenti, l'ufficio adotta la seguente politica, basata su **priorità dinamica**.

Siano **N_CER** e **N_IDE** i numeri delle richieste complessivamente servite (dall'avvio dell'applicazione) rispettivamente del tipo CER e del tipo IDE; la **precedenza** deve essere assegnata **agli utenti che richiedono il servizio con meno richieste complessivamente servite**.

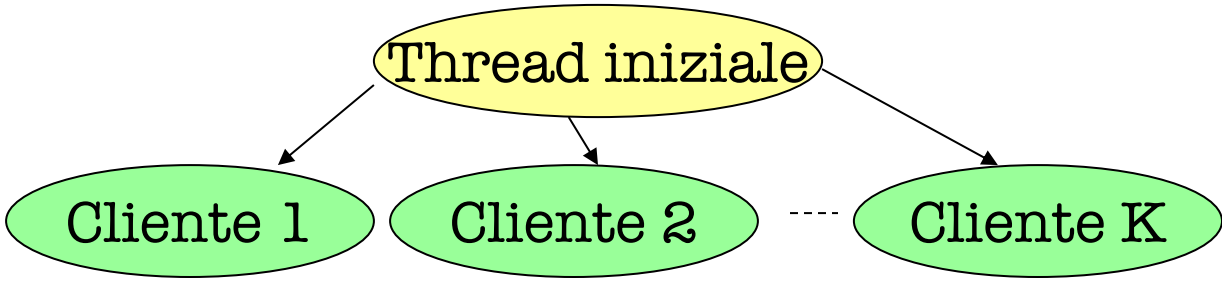
Quindi, se N_CER è maggiore di N_IDE, la precedenza va alle richieste di tipo IDE; se N_IDE è maggiore di N_CER, la precedenza va alle richieste di tipo CER.

Realizzare un'applicazione basata sul monitor che realizzi la gestione dell'ufficio in linguaggio Java, nella quale gli utenti siano rappresentati da thread concorrenti.

Impostazione

Quali thread?

- thread iniziale
- k utenti dell'ufficio



Quale risorsa comune?

- **l'ufficio (cioè, l'insieme degli N sportelli disponibili)**

- associamo all'ufficio un "***monitor***", che controlla gli accessi in base alla specifica politica di accesso. La sincronizzazione viene realizzata mediante **variabili condizione**

Struttura dei thread

```
public class Utente extends Thread{
      private Monitor M;
      private int tipo, max;
      public Cliente(...){ // costruttore..
      }
public void run(){
int op;
try {
      op=r.nextInt(2);
      if (op==0) {       //carta identità
            M.inizio_IDE(..);
            sleep(..);// occupa sport
            M.fine_IDE();
      } // continua…
```

Struttura dei thread -

```
    //…continua
    else{       //certificato
          M.inizio_CER(...);//entry call
          sleep(...);
          M.fine_CER(); //entry call
    }
    } catch (InterruptedException e) {     }
} //fine run()
```

Monitor: l'ufficio

Variabili di stato:

Stato Sportelli: vettore di N booleani (uno per sportello):

```
private bool[] libero;
//libero[i]=false se lo sportello è occupato da un utente

int occupati;
//occupati è numero degli sportelli occupati
```

Numero richieste servite:

```
int N_CER;
// N_CER è il numero di richieste CER complessivamente servite
int N_IDE;
// N_IDE numero di richieste IDE  complessivamente servite
```

Politica di Sincronizzazione

Un thread si sospende:

- se **non ci sono sportelli liberi**
- se ci sono thread **più prioritari** in attesa

Priorità:

- se N_CER è maggiore di N_IDE, la precedenza va alle richieste di tipo IDE;
- se N_IDE è maggiore di N_CER, la precedenza va alle richieste di tipo CER.

Ordine di priorità:

se N_CER >N_IDE

1. utenti IDE
2. utenti CER

se N_IDE<=N_CER

1. utenti CER
2. utenti IDE

Esercizio 2 – Il Ponte

Un piccolo ponte collega la riva Nord e la riva Sud di un fiume.

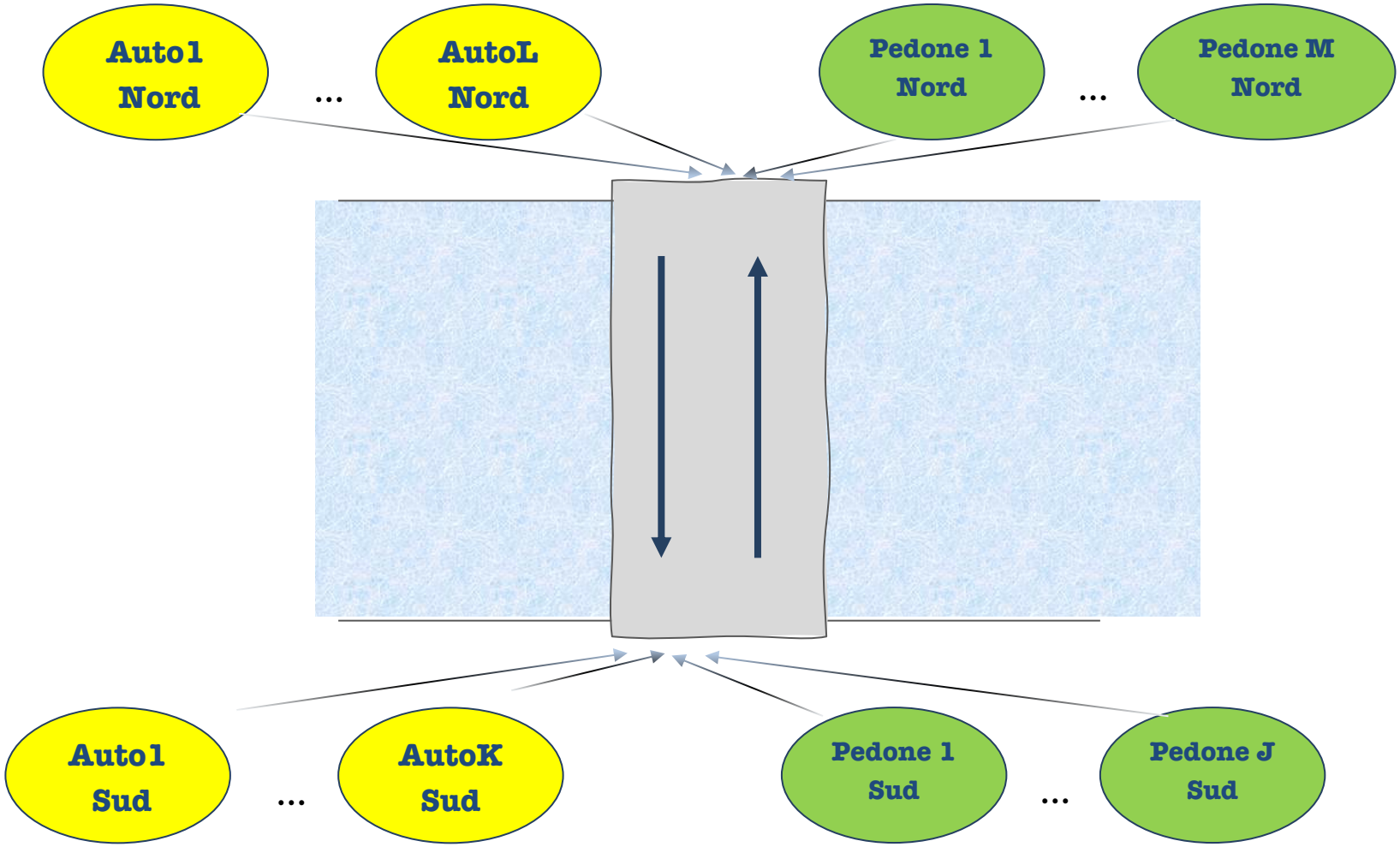
Il ponte può essere percorso sia da **automobili** che da **pedoni** in ognuna delle 2 direzioni.

Il ponte ha una **capacità massima MAX** che esprime il numero massimo di pedoni che possono contemporaneamente transitare sul ponte. A questo proposito si assuma che, ai soli fini della capacità, ogni auto equivalga a 10 pedoni.

Poichè **il ponte è stretto**, non è consentito il contemporaneo transito sul ponte di auto in entrambe le direzioni: in altre parole, la presenza di un'auto sul ponte in una data direzione impedisce ad altre auto di accedere al ponte in direzione opposta.

Si realizzi un programma Java nel quale auto e pedoni siano rappresentati da thread concorrenti e che, utilizzando il monitor e le variabili condizione, regoli gli accessi al ponte tenendo conto dei vincoli dati e, inoltre, del seguente vincolo di priorità: nell'accesso al ponte **i pedoni abbiano la priorità sulle automobili**.

Esercizio 2 - Il ponte



Impostazione

Quali thread?

- auto (con verso di percorrenza dir):

  <**entra_ponteA (dir)**>

  <attraversamento ponte>

  <**esci_ponteA (dir)**>

- pedone (con verso di percorrenza dir):

  <**entra_ponteP (dir)**>

  <attraversamento ponte>

  <**esci_ponteP (dir)**>

Impostazione

Quale risorsa comune?

Il **ponte**: associamo al ponte un **monitor**, che controlla gli accessi in base alla politica data.

- **Metodi entry**: **entrata** e **uscita** dal ponte.
- La sincronizzazione viene realizzata mediante **variabili condizione**:
  - I thread possono **sospendersi** solo eseguendo il metodo di **entrata** nel ponte.
  - La politica è **prioritaria: precedenza** ai pedoni
  - la condizione di sospensione considera **capacità, priorità o direzione**

Quante condition?