

Decima Esercitazione

Accesso a risorse condivise tramite
Monitor Java

Agenda

Esempio 1

La banca: gestione di una risorsa condivisa da più thread, con politica prioritaria

Esercizio 2 - da svolgere

Accesso a risorsa condivisa con sincronizzazioni complesse tramite Monitor

Esempio 1 – la banca

- Si consideri una piccola filiale di banca che rende disponibili alcuni servizi ai propri clienti tramite uno sportello.
 - I clienti della filiale possono essere intestatari di uno ed un solo conto corrente (**CC**).
 - Ogni conto corrente,
 - è individuato univocamente da un **codice** intero,
 - è caratterizzato da un **saldo**, che rappresenta l'importo attualmente depositato nel conto.
-

Esempio 1 - la banca

Per la gestione dei conti correnti, la filiale offre **2 servizi**:

- **Prelievo** dal proprio conto corrente: è l'operazione mediante la quale un cliente estrae dal proprio CC un **importo X** dato; si noti che tale operazione:
 - Può essere eseguita soltanto dall'intestatario del CC
 - Non può essere eseguita se il saldo del CC è minore di X; in tal caso il richiedente aspetta che nel conto vi sia un importo sufficiente.
- **Versamento** su un conto corrente: è l'operazione mediante la quale un cliente aggiunge ad un certo CC (**intestato a qualunque cliente**) un importo dato. Si noti che tale operazione può essere eseguita da qualunque cliente della filiale (eventualmente diverso dall'intestatario del CC).

Si supponga che ogni operazione richieda un **tempo non trascurabile**. **Ogni cliente che trova lo sportello occupato si mette in attesa.**

Esempio 1 – la banca

A tale scopo è **previsto 1 solo sportello**, mediante il quale i clienti possono richiedere sia **operazioni di prelievo che di versamento**.

I Clienti vengono classificati in due categorie:

- Clienti **Standard**, il cui saldo attuale è minore di 50.000 €;
- Clienti **Vip**, il cui saldo attuale è maggiore o uguale di 50.000 €.

Si noti che ogni cliente può **cambiare categoria dinamicamente** (in base, cioè, al saldo attuale).

Si progetti una politica di gestione dello sportello della filiale che tenga conto delle specifiche date e che, inoltre, soddisfi i seguenti vincoli:

- Le operazioni di versamento abbiano la priorità su quelle di prelievo;
 - Nell'ambito della stessa operazione (prelievo o versamento), i clienti Vip abbiano la priorità sui clienti Standard.
-

Esempio 1 - la banca

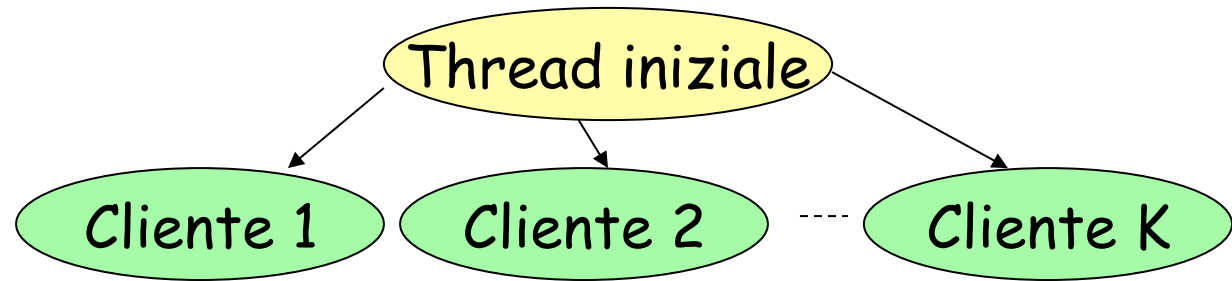
Si realizzi un'applicazione nel linguaggio Java che, utilizzando variabili condizione e monitor, implementi la politica di gestione e nella quale :

- i clienti siano rappresentati da thread concorrenti;
 - si supponga che ogni cliente richieda periodicamente alla filiale un'operazione (prelievo dal proprio CC o versamento in un CC casuale).
-

Impostazione

Quali thread?

- thread iniziale
- k clienti della banca



Quale risorsa comune?

- Sportello della filiale
 - associamo allo Sportello 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 Cliente extends Thread{  
    private Monitor M;  
    private int conto, max;  
    Random r = new Random();  
  
    public Cliente(int conto, Monitor m, int nCC) {  
        this.M=m;  
        this.conto=conto; // numero di conto  
        this.max=nCC;      //massimo numero di cc  
    }  
}
```

Struttura dei thread - versamento

```
public void run(){
    int op, cc, somma;
    try {
        while (true){
            op=r.nextInt(2) ;
            if (op==0) {        //versamento
                cc= r.nextInt(max) ;
                somma= r.nextInt(10)*10000; //[0-90000]
                M.versamento(cc, conto, somma) ;
                Thread.sleep(250) ;
                M.fine_operazione() ;
            } else{              //prelievo          continua...
```

Struttura dei thread - prelievo

```
//...
} else{                //prelievo
    somma= r.nextInt(10)*10000;
    M.prelievo(conto,somma);
    Thread.sleep(250);
    M.fine_operazione();
}
Thread.sleep(250);
} //fine while
} catch (InterruptedException e) { }
} //fine run()
```

Monitor: sportello della filiale

Variabili di stato:

Conti correnti: vettore di N interi (uno per conto corrente)

```
private int[] CC;
```

Dove: $CC[i]$ è il denaro disponibile nel conto del cliente i .

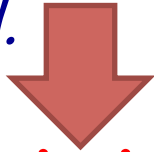
Politica di Sincronizzazione

Un thread si sospende:

- se lo sportello è **occupato**
- se deve prelevare e **non c'è disponibilità** sufficiente
- se ci sono thread **più prioritari** in attesa

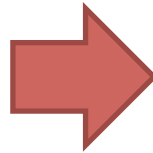
Vincoli di priorità:

- ✓ *Le operazioni di versamento hanno priorità sui prelievi;*
- ✓ *Nell'ambito della stessa operazione (prelievo o versamento), i clienti Vip abbiano la priorità sui clienti Standard.*



Ordine di priorità:

- Versamento vip
- Versamento standard
- Prelievo vip
- Prelievo standard



Per la sospensione dei thread definiamo **4 condition** (una per ogni livello di priorità)

Monitor

Lock per la mutua esclusione:

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

Condition. Per la sospensione dei thread in attesa definiamo 4 condition (una per ogni livello di priorità):

```
private Condition VV=lock.newCondition();  
    //coda dei Vip che vogliono versare  
private Condition VS= lock.newCondition();  
    //coda Standard che vogliono versare  
private Condition PV= lock.newCondition();  
    //coda Vip che vogliono prelevare  
private Condition PS= lock.newCondition();  
    //coda Standard che vogliono prelevare
```

Contatori dei thread sospesi in ogni coda:

```
private int  sospVV,  sospVS;  
private int  sospPV,  sospPS;
```

Monitor

```
public class Monitor{  
    private final int VIP=0; //TIPO VIP  
    private final int STA=1; //TIPO CLIENTE STANDARD  
    private final int soglia=50000;  
    private int N; //massimo numero di CC  
    private int[] CC; //CONTI CORRENTI  
  
    private Lock lock= new ReentrantLock();  
    private Condition VV= lock.newCondition();  
    private Condition VS= lock.newCondition();  
    private Condition PV= lock.newCondition();  
    private Condition PS= lock.newCondition();  
  
    private int sospVV, sospVS, sospPV, sospPS;  
    boolean occupato; //presenza di 1cliente allo sportello
```

Monitor

```
public Monitor( int NC) { //Costruttore:
```

```
    int i;
```

```
    N=NC;
```

```
    CC=new int[N];
```

```
    for(i=0; i<N; i++)
```

```
        CC[i]=0;
```

```
    occupato=false;
```

```
    sospPV=0; sospPS=0;
```

```
    sospVV=0; sospVS=0;
```

```
}
```

```
private int tipo(int conto){ //determina tipo di conto
```

```
    if (CC[conto] < soglia) return STA;
```

```
    else return VIP;
```

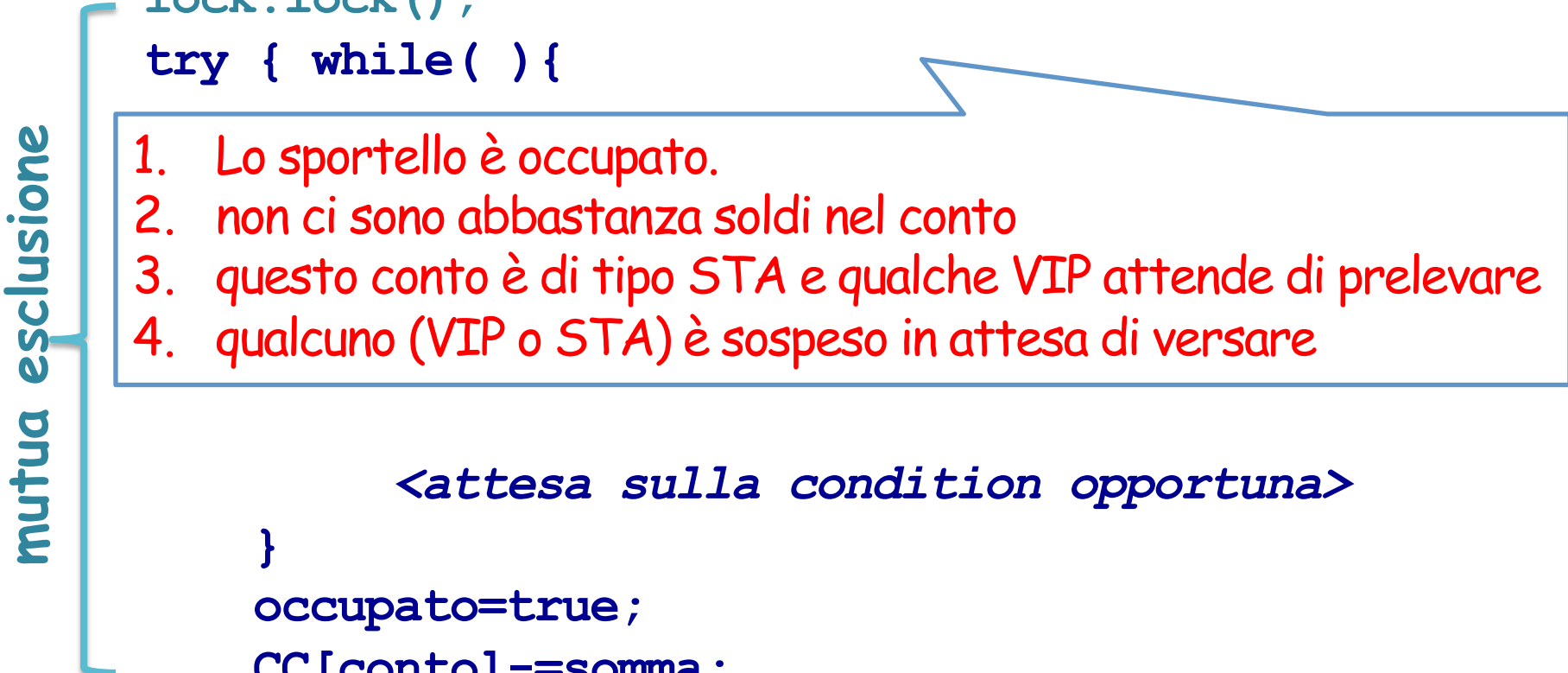
```
}
```

Inizialmente i conti sono tutti vuoti, lo sportello è libero e nessuno è sospeso in cosa

Monitor - prelievo

```
public void prelievo(int conto, int somma)  
    throws InterruptedException {
```

```
    lock.lock();  
    try { while( ){
```

- 
1. Lo sportello è occupato.
 2. non ci sono abbastanza soldi nel conto
 3. questo conto è di tipo STA e qualche VIP attende di prelevare
 4. qualcuno (VIP o STA) è sospeso in attesa di versare

<attesa sulla condition opportuna>

```
    }  
    occupato=true;
```

```
    CC[conto]-=somma;
```

```
    } finally { lock.unlock(); }
```

```
}
```

mutua esclusione

Monitor - prelievo

```
public void prelievo(int conto, int somma)
    throws InterruptedException {
    lock.lock();
    try { while( occupato || CC[conto]<somma ||
        ((sospPV > 0) && (tipo(conto)==STA)) ||
        sospVV>0 || sospVS>0 ){
        if (tipo(conto) == VIP){
            sospPV++; PV.await(); sospPV--;
        }else{
            sospPS++; PS.await(); sospPS--;
        }
    }
    occupato=true;
    CC[conto]-=somma;
    } finally { lock.unlock(); }
}
```

mutua esclusione

Monitor - versamento

```
public void versamento(int conto,int mioconto,int somma)  
    throws InterruptedException{
```

```
    lock.lock();  
    try{ while( ){
```

- 
1. Lo sportello è occupato
 2. questo conto è di tipo STA e qualche VIP attende di versare

<attesa sulla condition opportuna>

```
    }  
    occupato=true;  
    CC[conto]+=somma;
```

```
    } finally{ lock.unlock(); }
```

```
}
```

mutua esclusione

Monitor - versamento

```
public void versamento(int conto,int mioconto,int somma)
    throws InterruptedException{

lock.lock();
try{ while( occupato||
    ((sospVV > 0) && (tipo(mioconto)==STA))){
    if(tipo(mioconto) == VIP){
        sospVV++; VV.await(); sospVV--;
    } else {
        sospVS++; VS.await(); sospVS--;
    }
}
occupato=true;
CC[conto]+=somma;
} finally{ lock.unlock();}
}
```

mutua esclusione

```
public void fine_operazione() {  
    lock.lock();  
    try{  
        occupato=false;  
        //signal in ordine di priorità:  
        if (sospVV>0) VV.signalAll();  
        else if (sospVS>0) VS.signalAll();  
        else if (sospPV>0) PV.signalAll();  
        else if (sospPS>0) PS.signalAll();  
    } finally{ lock.unlock();}  
}
```

NB: Questa soluzione garantisce che la priorità venga rispettata, ma sveglia più processi di quelli che possono effettivamente accedere allo sportello.

Programma di test

```
public class Banca {  
    public static void main (String args[]) {  
        final int NC=20;  
        int i;  
        Cliente []clienti= new Cliente[NC];  
        Monitor M = new Monitor(NC);  
        for (i = 0; i < NC; i++) {  
            clienti[i] = new Cliente(i,M,NC);  
            clienti[i].start();  
        }  
    }  
}
```

NB: Che succede se tutti i clienti partono cercando di prelevare? e se un PS ha sufficienti soldi per prelevare, ma c'è un PV che ha richiesto di prelevare più di quel che ha?
In questo esercizio per costruzione possono verificarsi deadlock.

Esercizio 2 - La collezione di figurine

- Una nota casa editrice vuole realizzare un sito web dedicato ai collezionisti di figurine dell'album "*Campionato di calcio 2014-2015*".
 - L'album è composto da 100 diverse figurine, ognuna individuata univocamente da un intero; tra di esse,
 - 30 sono classificate come figurine rare,
 - e le rimanenti 70 come figurine normali.
 - 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.
-

Esercizio 2 - regole

Il meccanismo di scambio, è regolamentato come segue:

- Si può scambiare solo **una figurina alla volta**;
 - **Richiesta di scambio**:
 - 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.
-

Esercizio 2 - regole

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**;

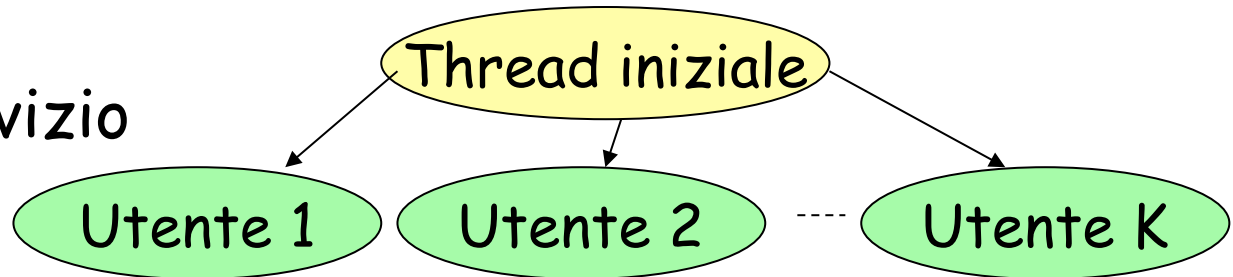
ad esempio:

- Colleza chiede 7 offrendo 3[RARA] ... 7 non disponibile
 - CollezB chiede 7 offrendo 50[NORM] ... 7 non disponibile
 - 7 diventa disponibile => signalAll() ...e la CPU schedula B
 - B ha offerto una figurina NORM. C'è qualcuno che ha offerto una RARA in attesa della stessa figurina?? Sì => B attende
-

Suggerimenti

Quali thread?

- 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**.
-

Suggerimenti

```
public class Collezionista extends Thread{  
    private Monitor M;  
    private int offerta, richiesta, max;  
  
    public Collezionista(monitor m, int NF) {  
        this.M=m;  
        this.max=NF;  
    }  
  
    public void run() {  
        try { while (true) {  
            <definizione di offerta e richiesta>  
            M.scambio(offerta, richiesta);  
            Thread.sleep(...);  
        } catch (InterruptedException e) { }  
    }  
}
```

riferimento
al monitor

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

Monitor - Deposito figurine

Stato del Deposito:

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

```
private int[] FIGURINE;
```

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

Convenzione adottata:

Se $i < 30$, si tratta di una figurina rara;

Se $i \geq 30$, 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= ...;  
//coda thread che hanno offerto figurine rare  
private Condition normali= ...;  
//coda thread hanno offerto figurine normali
```

Contatori dei thread sospesi in ogni coda:

```
private int[] sospRare;  
private int[] sospNormali;  
//devo sapere chi è sospeso in attesa di  
quella specifica figurina => array
```

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

NB: è plausibile che prima o poi il deposito vada in **deadlock**. Quando accadrà?
