Soluzioni esercizi

Esercitazioni.2

Esercizio 1

Versione semplice del produttore-consumatore

Un thread (Producer) produce dati (dallo 0 al 9, ciclicamente) e li memorizza in una variabile di scambio. Un thread (Consumer) legge i dati dalla variabile scambio e li visualizza.

Il consumatore deve leggere una sola volta il dato della variabile scambio. Produttore e consumatore impiegano un tempo casuale sleep((int)(Math.random()*100)); per generare e consumare il dato.

La variabile condivisa sia un oggetto di una classe, cui si accede solo tramite due metodi: get() e put().

soluzione Esercizio 1
SENZA SINCRONIZZAZIONE

```
public class SharedVariable {
    private int contents;
    public int get( ) {
        return contents;
    }
    public void put (int value) {
        contents = value;
    }
}
```

```
public class Producer extends Thread {
    private SharedVariable sharedVariable;
    private int number;
    public Producer(SharedVariable sv, int number) {
        sharedVariable = sv;
        this.number = number;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            sharedVariable.put(i);
            System.out.println("Prod #" + this.number + " put: " + i);
            try { sleep((int)(Math.random( ) * 100)); }
            catch (InterruptedException e) { }
        }
    }
}
```

```
public class Consumer extends Thread {
    private SharedVariable sharedVariable;
    private int number;
    public Consumer(SharedVariable sv, int number) {
        sharedVariable = sv;
        this.number = number;
    }
    public void run( ) {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = sharedVariable.get( );
            System.out.println("Cons #" + this.number + " got: " + value);
            try { sleep((int)(Math.random( ) * 100)); }
            catch (InterruptedException e) { }
        }
    }
}
```

```
public class ProducerConsumerTest {
    public static void main(String[ ] args) {
        SharedVariable sv = new SharedVariable( );
        Producer p1 = new Producer(sv, 1);
        Consumer c1 = new Consumer(sv, 1);
        p1.start( );
        c1.start( );
    }
}
```

soluzione Esercizio 1
CON SINCRONIZZAZIONE

```
class SharedVariable {
    private int contents;
    private boolean available = false;
    public synchronized int get( ) {
        while (available == false) {
            try { wait( );
            } catch (InterruptedException e) { }
        }
        available = false;
        notifyAll( );
        return contents;
    }
    …
}
```

```
class SharedVariable {
    …..
    public synchronized void put(int value) {
        while (available == true) {
            try { wait( );
            } catch (InterruptedException e { }
        }
        contents = value;
        available = true;
        notifyAll();
    }
}
```

ESERCIZI PROPOSTI

Possibili variazioni:

- aumentare il numero di produttori
- aumentare il numero di consumatori

Esercizio 2/A

Accesso ad una banca

Un oggetto della classe Banca è caratterizzato da un vettore di conti correnti, inizializzati a diversi valori, e possiede un metodo transfer() per spostare denaro da un conto ad un altro.

Il metodo transfer() deve essere in grado di soddisfare più richieste contemporaneamente, in maniera sicura

soluzione Esercizio 2/A - ERRATA

```
class Bank {
    private int[ ] accounts;
    public Bank(int n, int initialBalance) {
        accounts = new int[n];
        for (int i = 0; i < accounts.length; i++)
            accounts[i] = initialBalance;
    }
    public void transfer(int from, int to, int amount) {
        if (accounts[from] < amount)
            return;
        accounts[from] -= amount;
        accounts[to] += amount;
    }
}
```

soluzione Esercizio 2/A

```
class Bank {
    private int[ ] accounts;
    public Bank(int n, int initialBalance) {
        accounts = new int[n];
        for (int i = 0; i < accounts.length; i++)
            accounts[i] = initialBalance;
    }
    public synchronized void transfer(int from, int to, int amount) {
        if (accounts[from] < amount)
            return;
        accounts[from] -= amount;
        accounts[to] += amount;
    }
}
```

Esercizio 2/B

Accesso ad una banca – conto in rosso

Nell'esercizio precedente, se l'importo posseduto dal conto prima del trasferimento è inferiore, il trasferimento viene annullato. Modificare il metodo in modo che gestisca il conto in rosso: il trasferimento di prelievo non soddisfabile viene momentaneamente sospeso e riprende solo dopo che è arrivato un nuovo trasferimento di versamento su quel conto.

soluzione Esercizio 2/B

```
public synchronized void transfer(int from, int to, int amount) {
    try {
        while (accounts[from] < amount)
            wait( );
        accounts[from] -= amount;
        accounts[to] += amount;
        notifyAll();
    } catch(InterruptedException e) { }
}
```

Cosa succede se si utilizza il metodo notify() *invece di* notifyAll()?

Probabile deadlock! Infatti chi mi garantisce che il thread che sblocco può avanzare?