# Esercitazione #2 -- Corso di Sistemi Operativi

## *Sincronizzazione (indiretta-diretta) in Java*

*Luca Gherardi e Patrizia Scandurra – a.a. 2012-13*

1. **BankAccount**. Si consideri la classe BankAccount riportata sotto. Un oggetto di tale classe si comporta come una macchina ATM che effettua transazioni per depositare (il metodo deposit), prelevare (il metdo withdraw) e trasferire (il metodo transfer) denaro per i clienti. Si consideri una situazione di concorrenza in cui si effettuano diverse operazioni di trasferimento di denaro simultaneamente sugli stessi conti corrente.

   Si usi solo la sincronizzazione *indiretta* . Si modifichi, pertanto, la classe sfruttando opportunamente il modificatore *synchronized* per garantire la mutua esclusione sugli stessi conti corrente. Aggiungere poi una classe Transfer che definisce il comportamento di un tipo di thread che si occupa solo di trasferire da un conto a ad un conto b un certo ammontare amount di denaro. Testare, infine, il comportamento del programma definendo la classe principale con il metodo main suggerito nel secondo riquadro.

```
public class BankAccount { //Classe dell'oggetto condiviso
  private double balance;
  public BankAccount(double balance) {
  this.balance = balance;
}
public double getBalance() {
 return balance;
}
public void deposit(double amount) {
 balance += amount;
}
public void withdraw(double amount) throws RuntimeException {
  if (amount > balance) {
  throw new RuntimeException("Overdraft");
  }
  balance -= amount;
}
public void transfer(double amount, BankAccount destination) {
  this.withdraw(amount);
  destination.deposit(amount);
 }
}
```

```
public static void main(String[] args) {

     BankAccount a = new BankAccount(100.0);
     BankAccount b = new BankAccount(100.0);

     Transfer x = new Transfer(a, b, 50.0);
     Thread t = new Thread(x);
     Transfer y = new Transfer(b, a, 10.0);
     Thread t2 = new Thread(y);
     t.start();
     t2.start();

     System.out.println("Account a has $"+x.a.getBalance());
     System.out.println("Account b has $"+x.b.getBalance());
   }
```

2. **Contatore sincronizzato**. Si progetti una applicazione Java che permetta a più thread di operare contemporaneamente su un oggetto condiviso di classe Counter, sfruttando opportunamente il modificatore *synchronized* per evitare inconsistenze sull'oggetto condiviso. In particolare:
   - Un contatore (oggetto della classe Counter) è inizialmente a zero ed è in grado di contare fino a 10;
   - thread differenti devono poter incrementare/decrementare di una unità dallo stesso "contatore" (stesso oggetto di classe Counter) invocando il metodo increment()/decrement() della classe Counter*;* le operazioni di incremento/decremento devono essere possibili solo se mantengono rispettivamente il conteggio non superiore a 10/non negativo.

   Per testare il corretto funzionamento dell'oggetto di classe Counter, si preveda un programma principale che istanzi un certo numero di thread figli: alcuni thread con ruolo "TaskI" che all'interno di un ciclo infinito si alternano tra dormire un pò (un certo num. random di ms) e incrementare il contatore; e altri thread con ruolo "TaskD" che all'interno di un ciclo infinito si alternano tra dormire un pò (un certo num. random di ms) e decrementare il contatore.
   Cosa succede se si permette erroneamente l'accesso *non mutuamente esclusivo* (omettendo l'utilizzo del modificatore *synchronized*) all'oggetto contatore condiviso?
   *[**Suggerimento**:* Utilizzare solo la sincronizzazione *indiretta*.]

3. **Conto**. Si progetti una applicazione Java che permetta a più thread con ruoli diversi di operare contemporaneamente su un oggetto condiviso di classe Conto, sfruttando opportunamente il modificatore *synchronized* per evitare inconsistenze sull'oggetto condiviso. In particolare:
   - un thread padre deve poter aprire un "conto" (un oggetto di classe Conto) e condividerlo con altri n thread (considerati figli del thread padre);
   - il saldo iniziale del conto è pari a 10 000 euro;
   - thread differenti devono poter prelevare denaro, con importi diversi, dallo stesso "conto" (stesso oggetto di classe Conto) invocando il metodo prelievo(...) della classe Conto*;* le operazioni di prelievo devono essere possibili solo se mantengono il saldo del conto in positivo;
   - il thread padre deve poter aggiungere denaro al conto tutte le volte in cui il conto ha saldo minore o uguale a zero.

   Per testare il corretto funzionamento dell'oggetto di classe Conto, si preveda un programma principale che istanzi n thread figli che tentano di prelevare denaro continuamente (in un ciclo iterativo) dal conto.

   Cosa succede se si utilizza *notify()* anzicchè *notifyAll()* per il risveglio dei thread in attesa di cambiamento dell'ammontare del conto?
   *[**Suggerimento**:* Definire la classe Conto dell'oggetto condiviso ed utilizzare la sincronizzazione *diretta* (modificatore synchronized + i metodi wait-notify) per la mutua esclusione e la comunicazione tra thread. Si modelli poi il thread padre ed i thread figli come classi Java diverse. Introdurre, infine, una classe con il metodo main per creare ed avviare i vari thread, e testare l'applicazione.*]*

4. **ProducerConsumer**. Provare ad eseguire il programma multithread *Factory.java* nella cartella *ProducerConsumer* che implementa in Java una soluzione al problema del produttore-consumatore tramite *scambio di messaggi.* Cosa succede? Funziona correttamente? Perché non richiede sincronizzazione dei thread?
   *[**Suggerimento**:* Vedere dalla documentazione Java, la definizione della classe Vector usata per implementare il canale di comunicazione (oggetto condiviso).]