* java.util.concurrent
* java.util.concurrent.atomics
* java.util.concurrent.locks

Multithreading

Come lavorare alla mutua esclusione

•Lock: (→ new ReentrantLock)
- Si definiscono il lock e le condition variables. → Attenzione all'istanziazione del lock
- Si assegnano le CV al lock (lock.newCondition();)
- Come si lavora alla Sezione Critica:
  + Si acquisisce il lock (lock.lock();)
  + Si usa il while(¬ condizione) { condizione.await(); } (non dimenticare il blocco try)
  + Fuori dal while lavoriamo alle operazioni
  + Nel finally si fa l'unlock

•Semaphore:
- Si definiscono ed inizializzano i semafori (uno per ogni condizione)
- Come si lavora alla Sezione Critica:
  + Nel try-catch si fa l'acquire del semaforo
  + Si apre un try finally che conterrà un blocco synchronized(risorsa crit){}
  + In tal bbeeo si compiono le operazioni
  + Nel finally si farà la release del semaforo

•Synchronized:
- Si lavora nel blocco synchronized applicando wait() e notify() direttamente alla risorsa critica
- Se si associa al blocco l'istanza di un oggetto e si pongono all'interno dei metodi per esso, parleremo di sincronizzazione implicita.

Focus: Stampa log in un File:

1. Creare il File:
   FileOutputStream fileOut = new FileOutputStream("./nomeFile.txt");
2. Crea oggetto "Stampante":
   PrintStream printStream = new printStream(fileOut);
3. Metodo di utilizzo oggetto stampante:
   printStream.println("..." + var + "...");

Con Writers e Buffer:

1. Istanzia un oggetto FileWriter a cui passare il nome del file:
   FileWriter fw = new FileWriter("nome.txt", true);
2. Istanzia un oggetto BufferWriter a cui passare l'istanza fw.
3. Istanzia un oggetto PrintWriter a cui passare l'istanza bw.
   3.1 Usa i metodi println e flush del pw per scrivere sul file.
4. Chiudi i filter nell'ordine inverso di apertura.

Come si implementa un thread:

+ Per Derivazione da Thread

Prima soluzione:

```
public class MyThread extends Thread{
    public void run(){
      doWork();    //codice del thread
    }
}

Thread t= new MyThread();
t.start();
```

+ Con implemento della classe Runnable.

Seconda soluzione:

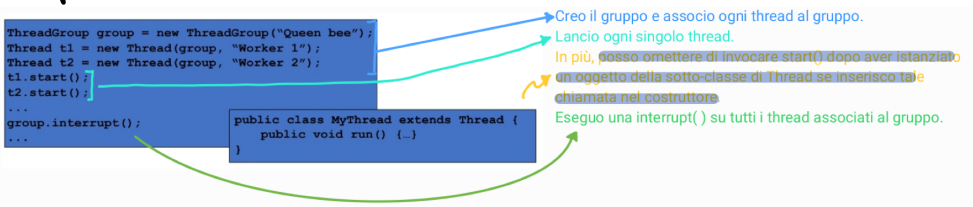
```
public class MyRunnable implements Runnable{

    public void run(){
      doWork();    //codice del thread
    }
}

…
Runnable r= new MyRunnable();
Thread t=new Thread(r);
t.start();
```

Metodi utili:

- Start: allocazione thread nella JVM
- Run: cosa fa il thread
- Sleep: sospende l'esecuzione
- Yields: sospende il thread in favore di un altro.
- Interrupt: solleva un'interruzione su di un oggetto thread
- Join: permette al thread padre di attendere l'interruzione dei thread figli

· Group e Pool:

Possiamo unire diversi thread in un unico group o pool per gestirli semplicemente all'unisono



Invece per avere un pool di thread che esistono indipendentemente da quella Runnable, useremo tali interfacce:

+ Executor
  - ExecutorService
  - SchedulerExecutorService

→ Forniscono dei metodi Factory per la creazione delle pool.

Recap di un monitor:

- Costrutto sintattico che associa un insieme di operazioni ad una risorsa critica.
  + Un thread per volta accede al Monitor
  + Esso possiede delle code per gestire i processi sospesi
  + Sfrutta le condition variables per attivare/bloccare i thread in Competizione

- In Java un monitor non è altro che una synchronized class
  → Qui si adotta un uso della notify tramite Signal and Continue ; quindi il processo "notificatore" continua ad occupare il monitor

CyclicBarrier:

Classe adoperata per la creazione di un punto di stop per tutti i thread.
+ Ogni thread raggiunge la barriera
+ Raggiunta dall'ultimo viene effettuata un'azione sui thread
+ I thread ripartono dal punto comune.