UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Interfacce Grafiche e Programmazione ad Eventi

Carmine Dodaro

Anno Accademico 2018/2019

Programmazione concorrente

Cos'è

La programmazione concorrente indica la possibilità di scrivere programmi che possono eseguire più operazioni in parallelo.

Perché usare la programmazione concorrente?

Usare la programmazione concorrente garantisce l'esecuzione di compiti potenzialmente pesanti in background in modo che l'utente non debba attendere l'esito dell'operazione prima di eseguirne altre.

Processi e thread

Cosa sono?

Un processo è un ambiente di esecuzione in cui gira il programma che l'ha creato. Ogni processo:

- afferisce ad un singolo programma
- è composto da un insieme di thread che condividono la stessa memoria
- ha sempre un thread di esecuzione rappresentato da sé stesso
- non vede le strutture dati degli altri processi

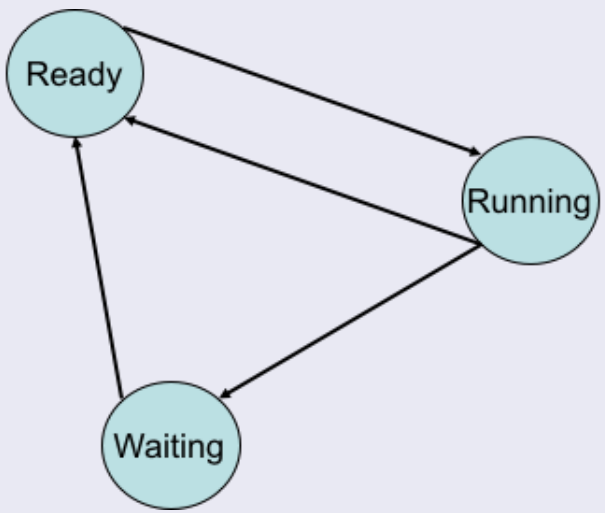
Un thread è un'unità di eleborazione in cui il processo può essere suddiviso. Ogni thread:

- ce ne sono più di uno per processo
- condividono le stesse risorse e strutture dati

Stati di un thread

Durante il suo ciclo di esecuzione, un thread può assumere diversi stati:

- ready: il thread è pronto ad essere eseguito
- running: il thread è avviato e sta eseguendo il codice
- waiting: il thread è in attesa di un evento esterno, ad esempio che qualche altro thread completi le sue operazioni



Passaggi di stato:

- Da Ready a Running
- Da Running a Ready
- Da Running a Waiting
- Da Waiting a Ready

I thread in java

La classe Thread

In Java i thread sono rappresentati da oggetti della classe Thread. La classe Thread mette a disposizione un metodo run in cui inserire il codice per effettuare le operazioni specifiche del thread.

```
public class ClasseThread1 extends Thread {
  @Override
  public void run() {
    super.run();
    while(true) {
      System.out.println("Sono nel thread 1 e stampo");
    }
  }
}

... nel main:

ClasseThread1 t1 = new ClasseThread1();
t1.start();
//Le operazioni scritte qui sono eseguite nonostante il codice
    di run contenga un while(true)!
```

La classe Thread

La classe Thread permette anche di interrompere l'esecuzione del thread per un certo periodo di tempo, quindi il thread passa da uno stato di running ad uno di waiting. Il metodo da utilizzare è sleep, che riceve come parametro un intero rappresentante il numero di millisecondi di pausa.

```
@Override
public void run() {
  super.run();
  while(true) {
    try {
      System.out.println("Sono nel thread 1 e stampo");
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      return;
    }
  }
}
```

L'interfaccia Runnable

Un'alternativa all'uso della classe Thread è l'uso dell'interfaccia Runnable.

```
public class ClasseThread2 implements Runnable {
  @Override
  public void run() {
    while(true) {
      try {
        System.out.println("Sono nel thread 2 e stampo");
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        return;
      }
    }
  }
}

... nel main:

ClasseThread2 t2 = new ClasseThread2();
Thread t = new Thread(t2);
t.start();
```

Gestire la sincronizzazione

In un programma con più thread è opportuno gestire la sincronizzazione tra tutti i thread in modo da garantire che non ci siano accessi simultanei alla stessa risorsa.

La sincronizzazione è attuata attraverso operazioni di lock, block e release, attraverso cui si garantisce che un solo thread alla volta possa avere accesso a determinati dati in un preciso momento (mutua esclusione).

In Java la sincronizzazione tra i vari thread è gestita attraverso un meccanismo di controllo chiamato monitor lock. Ad ogni oggetto e ad ogni classe è associato un monitor che serve a proteggere le loro variabili. Nell'ambito del programma possiamo decidere quali blocchi di codice devono essere protetti, cioè porzioni di codice che devono essere eseguite in modo esclusivo e sincronizzato. da un solo thread per volta.

Sincronizzazione nel codice

```
public synchronized void metodoSincronizzato() {
  //qui il codice sincronizzato
}
```

Ogni thread che proverà ad usare il metodoSincronizzato ha la garanzia che nessun altro thread lo potrà eseguire nello stesso momento (lock). I thread che proveranno ad usare quel metodo andranno in uno stato di blocco (block). Quando l'esecuzione del metodo termina per il thread che ha richiesto il lock, allora il metodo è reso disponibile anche gli altri thread (release).
Si può anche sincronizzare solo una porzione del metodo:

```
public void metodo()
  //qui codice non sincronizzato
  synchronized(this) {
    //qui codice sincronizzato
  }
  //qui codice non sincronizzato
}
```

Condizioni di blocco

Blocchi

Un programma concorrente può bloccarsi a causa di tre differenti cause:

- Deadlock: due thread sono bloccati perché aspettano tra di loro che ciascuno liberi una risorsa.
- Starvation: un thread ottiene un lock su una risorsa e compie delle operazioni che non consentono mai di rilasciarlo.
- Livelock: è simile in linea di principio al deadlock, solo che in questo caso i due thread non sono bloccati, ma compiono continuamente delle operazioni, l'uno rispetto all'altro, che non permettono di terminare la propria esecuzione.

Alcuni metodi utili sui thread

Alcuni metodi

- wait(); permette di mandare in uno stato di wait un thread per un tempo indefinito.
- notify(); notifica ad un thread su cui era stato chiamato il metodo wait() che può riprendere le operazioni.
- notifyAll(); notifica a tutti i thread su cui era stato chiamato il metodo wait() che possono riprendere le operazioni.
- join(); pone in attesa un thread fino a che non cessa di esistere.
- yield(); il thread cede spontaneamente il proprio tempo ad altri thread.
- setPriority(); riceve come parametro un intero
  - Thread.MAX_PRIORITY
  - Thread.NORM_PRIORITY
  - Thread.MIN_PRIORITY

Concorrenza con le API ad alto livello

Concurrences utilities

Java offre delle API ad alto livello per la gestione ad alto livello delle concorrenza. Queste API consentono:

- di aumentare l'affidabilità del codice perché sono state scritte da esperti e testate per evitare i problemi di deadlock, starvation, ecc;
- di migliorare la velocità del codice, in quanto il codice è ottimizzato per le performance;
- di garantire una manutenubilità del codice in quanto il singolo programmatore non si deve occupare della gestione della sincronizzazione.

API ad alto livello

Il package java.util.concurrent

Il package mette a disposizione diversi componenti:

- l'executor framework
- un fork/join framework
- le concurrent collections
- i syncronizers

Executor Framework

Cos'è

L'Executor Framework, che permette di astrarre la creazione e la gestione dei thread attraverso l'uso di altre classi, chiamate executor, e di classi che implementano il concetto di thread pool. Nell'ambito di questo framework si trovano tre interfacce:

- Executor, che consente di creare ed eseguire un nuovo thread rappresentato da un oggetto che estende Runnable;
- ExecutorService, che permette di gestire i thread in modo più versatile rispetto ad Executor;
- ScheduledExecutorService, che permette l'esecuzione dei thread dopo un certo intervallo di tempo, oppure in modo ciclico.

Come si usa

Uso di Executor Framework

- Executors.newCachedThreadPool(), si occupa di creare un nuovo thread quando serve, ed usa la cache nel caso in cui sia necessario riutlizzarli per velocizzare
- Executors.newFixedThreadPool(int NUM), si usano al massimo NUM thread
- Executors.newSingleThreadExecutor(), viene creato un unico thread che esegue le operazioni

```
ExecutorService es = Executors.newFixedThreadPool(3);
es.submit(new OggettoCheImplementaRunnable(i));
es.shutdown();
```

Fork/Join Framework

Cos'è

Consente di sviluppare dei programmi che agiscono in modo parallelo (fork), i cui risultati vengono poi uniti (join). La gestione dei programmi è fatta in modo leggero ed efficiente.

Concurrent Collections

Cosa sono

Sono collezioni che sono particolarmente adatte al contesto della programmazione concorrente. Un esempio è la classe ConcurrentHashMap.

I synchronizers

Cosa sono

Sono classi che implementano in un modo efficiente e pulito la gestione della sincronizzazione tra i vari thread.

Semafori

Un semaforo è usato per controllare il numero di thread che può avere accesso ad una determinata risorsa. Come dice il nome, un semaforo dà accesso ad alcuni thread bloccandone altri.