Laboratorio di programmazione di rete
Java Thread

Corso B

Anno accademico 2007-2008

Prof. Marco Danelutto

Assistente Dr.ssa Sonia Campa

Introduzione: i thread

- Concetto di thread
  (richiami di laboratorio di programmazione di sistema)
  flusso indipendente di controllo
  nello stesso spazio di indirizzamento del processo che lo contiene
  schedulato in modo indipendente o da uno schedulatore di sistema (come se fosse un processo) o da uno schedulatore nello spazio utente (system threads vs. user threads)

- Java Threads
  Flusso di controllo autonomo
  dotato di una propria priorità e schedulato conseguentemente insieme agli altri
  che di solito va avanti fino al proprio completamento
  con ridotte possibilità di interazione con gli altri thread
  che non siano quelle basate sulla condivisione di oggetti
  oppure demon thread
  che vanno avanti fino alla terminazione del programma

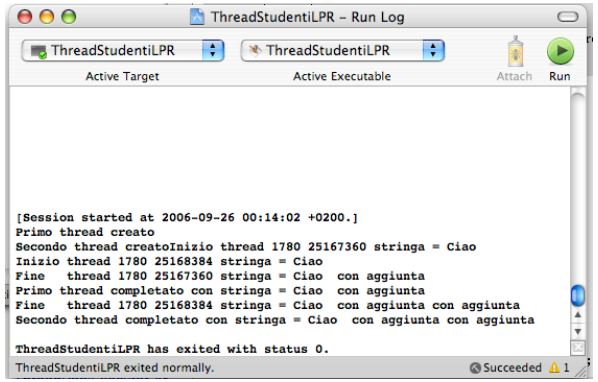
Esempio di thread C (dato per scontato)

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <stdlib.h>

#define SLEN 128

char * stringa = NULL;

void * corpo(void * param) {
 //extern char * stringa;
 int ritardo = (*(int *)(param));
 printf("Inizio thread %d %ld stringa = %s\n",getpid(),pthread_self(),stringa);
 sleep(ritardo);
 strcat(stringa," con aggiunta");
 printf("Fine   thread %d %ld stringa = %s\n",getpid(),pthread_self(),stringa);
 pthread_exit(stringa);
}

int main (int argc, char *  argv[]) {
 pthread_t tid1, tid2;
 int param1,param2;
 char  *retval1, *retval2;
 param1 = 1;
 param2 = 3;

 stringa = (char *) calloc(sizeof(char),SLEN);
 strcat(stringa,"Ciao ");
```

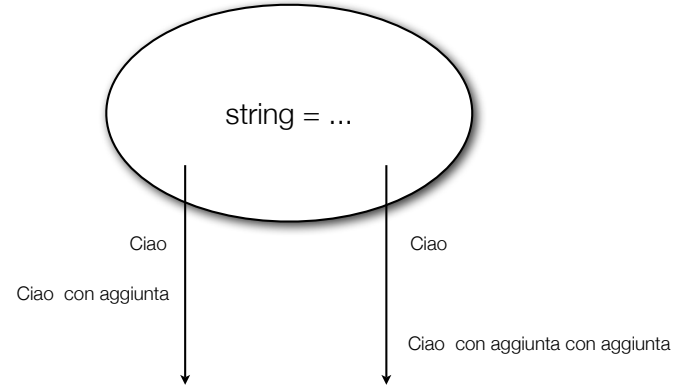

Esempio di thread C (dato per scontato)

```c
stringa = (char *) calloc(sizeof(char),SLEN);
strcat(stringa,"Ciao ");

if(pthread_create(&tid1,NULL,corpo,&param1)!=0) {
    printf("errore nella creazione del primo thread\n");
    return(-1);
}
printf("Primo thread creato\n");

if(pthread_create(&tid2,NULL,corpo,&param2)!=0) {
    printf("errore nella creazione del secondo thread\n");
    return(-1);
}
printf("Secondo thread creato");


if(pthread_join(tid1,(void **)&retval1)!=0) {
    printf("errore nella join del primo thread\n");
    exit(-1);
}
printf("Primo thread completato con stringa = %s\n",retval1);
if(pthread_join(tid2,(void **)&retval2)!=0) {
    printf("errore nella join del secondo thread\n");
    exit(-1);
}
printf("Secondo thread completato con stringa = %s\n",retval1);
   return 0;
}
```

Risultato della computazione



Computazione



Terminazione di un programma con thread

- un programma con thread termina
  - quando tutti i thread che ha lanciato sono terminati
  - quando se ne richiede esplicitamente l'uscita (return nel main) e non esistono più thread non demoni in esecuzione
  - quando si invoca una **exit**
- un thread bloccato in attesa di un evento non è terminato
  - quindi concorre alla non terminazione del programma principale

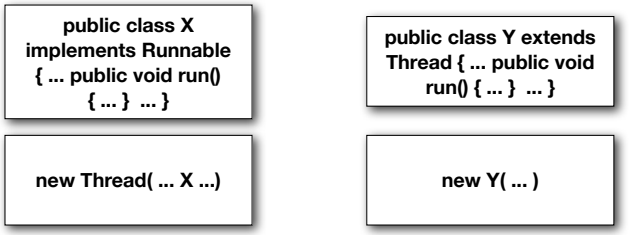
Proprietà dei thread

- Priorità
  - serve per implementare la schedulazione corretta dei thread
  - thread con priorità più alta schedulati per primi (int static accessibili come Thread.MAX_PRIORITY, Thread.MIN_PRIORITY, Thread.NORM_PRIORITY)
- Gruppo
  - concetto di pool di thread logicamente correlati che possono essere riferiti/ interrotti collettivamente

Modalità per la creazione di un thread

- Due modalità
  - sostanzialmente perchè Java non permette ereditarietà multipla
- implementazione dell'interfaccia Runnable e conseguente istanziazione di un oggetto Thread con la classe che implementa l'interfaccia come parametro
  Thread t = new OggettoExtendsThread(parametri);
  OggettoExtendsThread t = new OggettoExtendsThread(parametri);
- specializzazione della classe Thread (qualora non sia necessario specializzare altre classi)
  OggettoImplementaRunnnable o =
  new OggettoImplementaRunnable(param);
  Thread t = new Thread(o);

Due metodi



Creazione: Java vs. pthread

- Creazione con pthread
  - int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);
  - è di fatto sia la creazione che l'attivazione del thread, mentre in Java le due cose sono separate
  - controllo degli attributi del thread mediante parametri della creazione mentre in Java si fa con metodi accessori
- Attesa della terminazione simile a quella Java (ma restituisce exit value in più)
  - pthread_join(pthread_t thread, void **value_ptr);

Thread in un programma Java

- Alla partenza esiste un solo thread (non demone) che corrisponde all'esecuzione del **main**
- alla creazione di un thread il thread eredita priorità e modalità (demon, non demon) dal thread che lo crea
  - variazione della priorità di un thread
    public final void setPriority(int newPriority);
  - variazione della modalità di esecuzione
    public final void setDaemon(boolean on);

Concetto di thread dal punto di vista implementativo: il thread

- definizione di un metodo **void run(void)** che esprime il corpo del thread
- passaggio di parametri “propri” al thread mediante il costruttore
  - servono a inizializzare el variabili d’istanza che successivamente specializzeranno l’esecuzione del metodo run/corpo del thread nelle diverse istanze di thread della stessa classe
- implementazione di metodi ispettori per reperire il valore delle variabili d’istanza che di utilizzano **dopo la terminazione del thread** per evitare problemi con gli accessi concorrenti

Concetto di thread dal punto di vista implementativo: il lifecycle

- attivazione di un thread
  - evento separato dalla creazione
    creazione -> invocazione del costruttore con i parametri, descrittore di thread non ancora nella lista dei pronti
    attivazione -> descrittore del thread nella lista dei pronti, schedulato appena possibile secondo lo schedulatore della macchina virtual Java
- attesa della terminazione di un thread
  - evento bloccante richiesto sull’oggetto thread
    bloccato fino a quando il thread non termina spontaneamente o per un’interruzione generata dall’esterno (deprecato in questo corso)
    permette comunque la chiamata successiva dei metodi ispettori per il reperimento dei valori delle variabili d’istanza

Concetto di thread dal punto di vista implementativo: il lifecycle (2)

- possibilità di interagire con il thread mediante interruzioni
  - methodo interrupt()
  - interrupt viene recepito solo in certi punti (descheduling points)
  - l’interrupt può essere “sentito” dal thread oppure ne può semplicemente determinare la terminazione dell’operazione “di attesa” in atto

interrupt

- puo’ essere invocata da un altro thread
  tid.interrupt()
- ha effetto sul flusso di controllo e sullo stato di interrotto (boolean) associato al thread
- se il thread soggetto dell’interrupt
  sta facendo un wait, una sleep o una join (**richiami sulla wait**) riceve una **InterruptedException** e lo stato di interrotto viene resettato
  sta eseguendo una operazione di ingresso/uscita bloccata riceve una **ClosedByInterruptException** (solo sui canali interrompibili, non interessa ai fini di questo corso, così come il caso della **select** (NIO))
  in tutti gli altri casi viene settato lo stato di interrotto
  boolean interrupted() restituisce lo stato e lo resetta
  boolean isInterrupted() restituisce lo stato senza resettarlo

interrupt (2)

- da utilizzare principalmente per
  - segnalare la necessità di terminare il thread, appena questo sia in uno stato adatto alla terminazione
  - interrompere un thread in attesa di un evento che non potrà mai verificarsi (lo si sa in un ambiente esterno al thread, interrupt() e poi però try catch InterruptedException opportuna !!!)
- comunque da non considerare una primitiva che stoppa e/o termina arbitrariamente il thread

Attesa per la terminazione di un thread

- metodo
  void join()
  si blocca fino a quando il thread in oggetto non termina
- variante con timeout
  void join(int millisecondiTimeout);
  si blocca fino a che il thread non termina o scatta il timeout
  in questo caso si può utilizzare
  boolean isAlive()
  per testare per quale ragione è terminata l'esecuzione della join

yield

- thread che eseguono cicli di azioni che non prevedono deschedulazione
  per esempio cicli compute intensive, senza sincronizzazioni nè operazioni di ingresso uscita
  possono dare la possibilità di girare ad altri thread
- yield

static void	**yield**() Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Gruppi di thread

- usati per modellare pool di attività concorrenti soggette ad un qualche tipo di controllo comune (per esempio da interrompere collettivamente)
- Creazione di un gruppo di thread

ThreadGroup(String name) Constructs a new thread group.
ThreadGroup(ThreadGroup parent, String name) Creates a new thread group.

- Creazione del thread con nomina esplicita del gruppo cui appartiene

Thread(ThreadGroup group, Runnable target) Allocates a new Thread object.
Thread(ThreadGroup group, Runnable target, String name) Allocates a new Thread object so that it has target as its run object, has the specified name as its name, and belongs to the thread group referred to by group.
Thread(ThreadGroup group, Runnable target, String name, long stackSize) Allocates a new Thread object so that it has target as its run object, has the specified name as its name, belongs to the thread group referred to by group, and has the specified *stack size*.
Thread(ThreadGroup group, String name) Allocates a new Thread object.

Gruppi di thread (2)

- quanti sono i thread ancora attivi nel gruppo

int | **activeCount**()
Returns an estimate of the number of active threads in this thread group.

- interruzione di tutti i thread nel gruppo

void | **interrupt**()
Interrupts all threads in this thread group.

- distruzione del gruppo

void | **destroy**()
Destroys this thread group and all of its subgroups.

- controllo sulle modalità di esecuzione

boolean | **isDaemon**()
Tests if this thread group is a daemon thread group.

Esempi di codice: Creazione dei thread

```
public class UnThread extends Thread {
  ...
  int x;

  public UnThread(int i) { x = }

  public void run() {
     ...
  }

  public int getX() { return x; }
}
...
UnThread t = new UnThread(5);    // creazione
t.start();                       // attivazione
...
t.join();                        // attesa della terminazione
int x = t.getX();                // metodo ispettore
```


Esempi di codice: Creazione dei thread (2)

```
public class UnThread implements Runnable {
  ...
  int x;
  public UnThread(int i) { x = }
  public void run() {
     ...
  }
  public int getX() { return x; }
}
...
UnThread ut = new UnThread(5);  // creazione istanza oggetto
Thread t = new Thread(ut);      // creazione thread
t.start();                      // attivazione
...
t.join();                       // attesa della terminazione
int x = t.getX();               // metodo ispettore
```


A proposito di thread

- Per cosa si utilizzano i thread?
  - per tutte quelle attività che sono (devono essere) concorrenti (fra di loro **e rispetto al flusso di controllo principale del programma**)
- non voglio vedere sequenze di istruzioni
  - creazione thread, start, join
  - che di fatto rappresentano una chiamata di metodo
    - equivalente a creazione oggetto thread + invocazione metodo run

Esempi di codice sbagliato: Creazione dei thread

```
public class UnThread extends Thread {
  ...
  int x;
  public UnThread(int i) { x = }
  public void run() {
     ...
  }
  public int getX() { return x; }
}
...
Thread t = new Thread(5);       // creazione thread
t.run();                        // attivazione errata !!!!
...                             // è una chiamata di metodo !!!
t.join();                       // attesa della terminazione
int x = t.getX();               // metodo ispettore
```


Esempi di codice sbagliato: Codice equivalente a t.run()

```
public class UnThread extends Thread {
  ...
  int x;
  public UnThread(int i) { x = }
  public void run() {
     ...
  }
  public int getX() { return x; }
}
...
Thread t = new Thread(5);       // creazione thread
t.start();
t.join();                       // attesa della terminazione
int x = t.getX();               // metodo ispettore
```

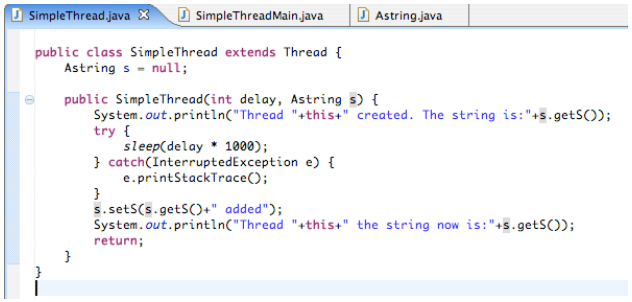

Sincronizzazione di thread mediante oggetti

- Caso tipico: produttore consumatore
- creazione di un oggetto buffer con metodi sincronizzati per contenere gli oggetti della produzione e del consumo
- thread produttore genera oggetto, invoca una insert sull'oggetto buffer
  L'esecuzione della insert eventualmente esegue una notify (notifyAll)
- thread consumatore invoca una remove sull'oggetto buffer
  l'esecuzione della remove può incorrere in una wait() nel caso il buffer sia vuoto
  in questo caso il thread si blocca
  si sblocca se e quando avviene una insert
- **(approfondimento sincronizzazione in caso)**

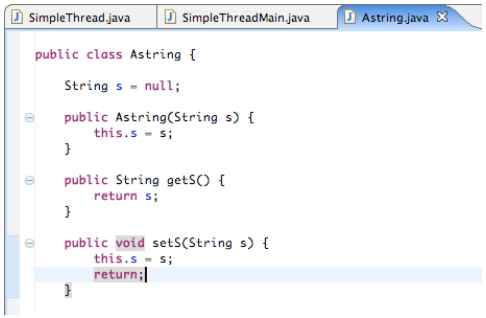
Sincronizzazione di thread mediante oggetti (2)

- Creazione dell'oggetto buffer deve precedere la creazione dei thread che lo utilizzano
- oggetto buffer passato come parametro del costruttore
- memorizzato in una variabile creata prima di creare i thread e passata ai thread mediante parametro del costruttore
- ATTENZIONE: i thread lavorano in un ambiente di memoria condivisa. Eventuali cambiamenti su oggetti passati (per riferimento) si propagano a tutti i thread che condividono l'oggetto (il riferimento all'oggetto)

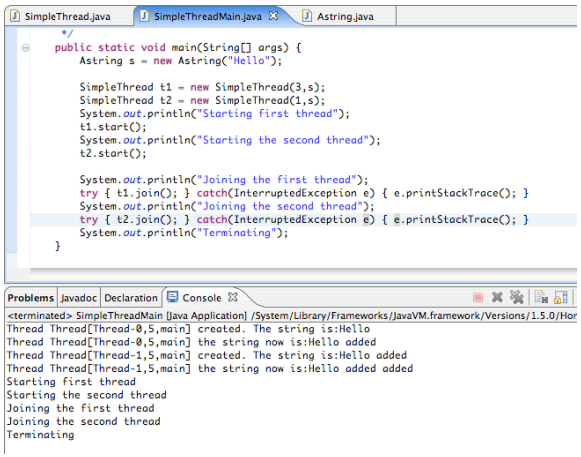
Esempio (ex C)



Esempio (ex C)



Esempio (ex C)



Esempio (ex C)

- Schedulazione Thread Java 1.5
  - a completamento
  - il thread 1 completa
  - poi va in esecuzione il thread 2
  - quindi:
    - Hello -> Hello added -> Hello added -> Hello added added

Concetto di thread pool

- costo nella creazione di un thread
- in caso occorra gestire in modo più efficiente la concorrenza di ricorre alla preallocazione di un certo numero di thread
- i thread si bloccano in attesa della richiesta di “attivazione logica” tipicamente aspettano un parametro/task
- quando il parametro di input è disponibile calcolano quanto debbono calcolare e poi si rimettono in attesa

Thread e Java 1.5

- Nuove facilities nella classe
  - java.util.concurrent
- le vedremo più avanti nel corso