
# Programmazione Java
# Corso Pratico
# 08 - Introduzione ai threads

Maurizio Franco

# COS'E' UN THREAD?

# Introduzione al multi-threading

Un thread è un singolo flusso sequenziale di istruzioni all'interno di un programma.

Si possono utilizzare più threads all'interno di un singolo programma;
ognuno di essi esegue un task differente.

Il thread utilizza le risorse allocate per il programma padre.

Esso deve ritagliarsi alcune delle sue risorse all'interno del programma in esecuzione.

In Java i threads sono degli oggetti.

# PERCHE' USIAMO IL THREAD?

Miglioramento delle performance dei programmi, specie in applicazioni grafiche.

E per applicazioni che prevedono eventi temporizzati.

# SVANTAGGI NELL'USO DEL THREAD

Difficoltà d'uso

Difficoltà nel debugging

Rischio di deadlock

Ci sono 2 modi per creare un thread:

- Estendere la classe java.lang.Thread (che implementa Runnable)

- Implementare l'interfaccia Runnable

# Esempio di estensione classe Thread

```
public class ThreadExample extends Thread {
    public ThreadExample(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((int)(Math.random()*1000));
            } catch (InterruptedException e){}
        }
        System.out.println(getName() + " end his job...");
    }
}
```

# Esempio di estensione classe Thread pt.2

```
public class ThreadsTest {
    public static void main (String[] args) {
        new ThreadExample("Pippo").start();
        new ThreadExample("Pluto").start();
    }
}
```

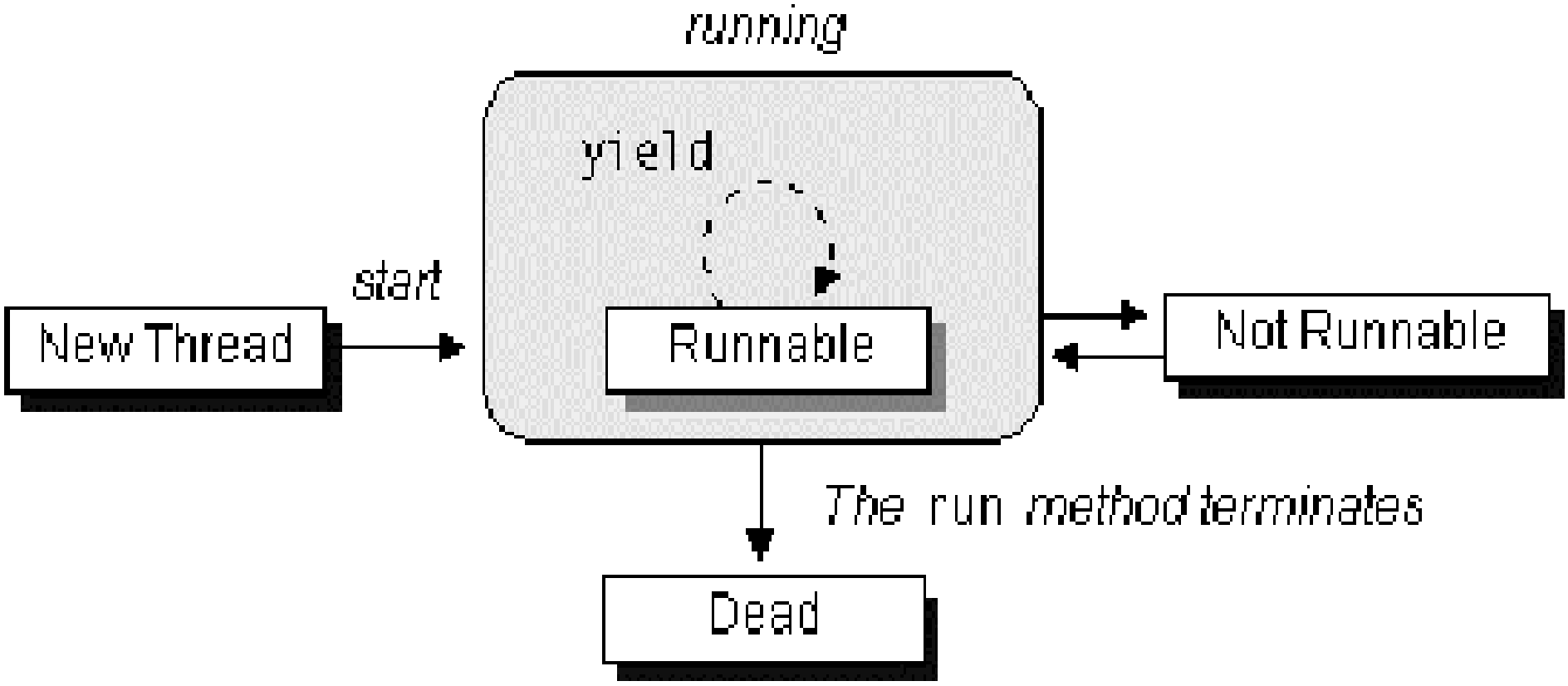
# Esempio di implementazione interfaccia Runnable

```
public class RunnableExample extends AltraMiaClasse
   implements Runnable {
    public void run() {
        for (int i=1; i<=10; i++)
            System.out.println(i + " " + i*i);
    }
}
```

# Esempio di implementazione interfaccia Runnable pt.2

```
public class TestRunnable  {
    public static void main(String args[]){
        RunnableExample e = new RunnableExample();
        Thread t = new Thread (e);
        t.start();
    }
}
```

# CICLO DI VITA DI UN THREAD



# STATI DEL THREAD pt.1

CREATO – E' stato istanziato l'oggetto mediante l'istruzione new, le variabili sono state locate ed inizializzate.

RUNNABLE - è in esecuzione, o in coda d’attesa per ottenere l’utilizzo della CPU

# STATI DEL THREAD pt.2

NOT RUNNABLE – il thread non può essere messo in esecuzione dallo scheduler. Il thread si trova in questo stato, quando sulla sua istanza viene invocato uno dei seguenti metodi: suspend(), wait(), sleep().

# STATI DEL THREAD pt.3

DEAD – quando il thread termina l'esecuzione delle istruzioni in esso contenute, o quando durante la sua esecuzione invochiamo il metodo stop()

# METODI PER IL CONTROLLO DEL THREAD

start() - fa partire l'esecuzione del thread.
La JVM invoca il metodo run() del thread appena creato.

stop() - forza la terminazione dell’esecuzione di un thread. Tutte le risorse utilizzate dal thread vengono immediatamente liberate (lock inclusi), come effetto della propagazione dell'eccezione ThreadDeath

suspend() - blocca l'esecuzione di un thread in attesa di una successiva operazione di resume. Non libera le risorse impegnate da un thread(possibilità di deadlock)

resume() - riprende l'esecuzione di un thread precedentemente sospeso. Se il thread riattivato ha una priorità maggiore di quello correntemente in esecuzione, avrà subito accesso alla CPU, altrimenti andrà in coda d'attesa.

sleep(long time) - blocca per un tempo specificato (time) l'esecuzione di un thread.

# INTRODUZIONE ALLA SINCRONIZZAZIONE DEI THREADS

Quando due o più threads eseguono concorrentemente delle operazioni è impossibile prevedere l'ordine con cui accederanno a delle funzionalità o risorse a disposizione.
Il risultato di questa situazione è la possibilità di generare inconsistenze.

Mediante la sincronizzazione dei threads è possibile ovviare a certe situazioni inconsistenti.
In pratica.... è previsto l'utilizzo dell'istruzione synchronized su un metodo o su di un blocco di istruzioni.

Succede cioè questo....:

1) a ogni oggetto è automaticamente associato un lock.

2) per accedere a un metodo o una sezione synchronized, un thread deve prima acquisire il lock dell'oggetto

3) Il lock è automaticamente rilasciato quando il thread esce dalla sezione synchronized, o se viene interrotto da una eccezione.

4) Un thread che non riesce ad acquisire un lock rimane sospeso sulla richiesta di una risorsa, fino a che il lock non è disponibile.

Ma, sincronizzare l'accesso ad una risorsa non è solo limitare l'accesso ad una risorsa/funzionalità per un thread/chiamante per volta. C'è bisogno anche che ci sia un certo coordinamento fra I vari “chiamanti”.

Questo coordinamento si ottiene mediante l'utilizzo di appositi metodi: wait(), notify(), notifyAll()
Questi metodi consentono ai threads di aspettare una condizione e notificare ad altri threads che la condizione è cambiata.