Java
thread, concorrenza

laboratorio 1

A.Memo – febbraio 2004

bibliografia: Java, Patrick Naughton e Herbert Schildt

il thread principale

- in Java ogni programma in esecuzione è un thread
- il metodo *main()* è associato al *main thread*
- per poter accedere alle proprietà del *main thread* è necessario ottenerne un riferimento tramite il metodo *currentThread()*

```
class ThreadMain {              // controllo del thread principale (main)
    public static void main(String args[ ]) {
        Thread t = Thread.currentThread();
        System.out.println("Thread corrente: " + t);
        t.setName("Mio Thread");
        System.out.println("Dopo aver cambiato il nome: " + t);
        try {
            for (int n=5; n>0;n--) {
                System.out.println(n);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) { };
    }
}
```

```
Thread corrente: Thread[main,5,main]
Dopo aver cambiato il nome: Thread[Mio Thread,5,main]
5
4
3
2
1
```

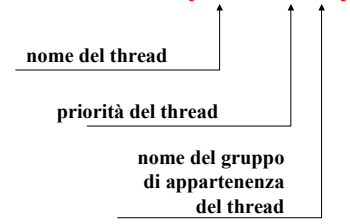
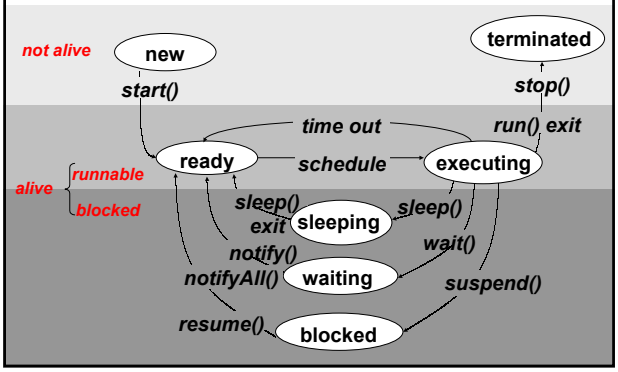


diagramma degli stati dei thread in Java



Creazione di un nuovo thread (1)

Ci sono due modi per creare un nuovo thread: questo il primo:

- dichiarare una classe che implementa l'interfaccia `Runnable`, che deve implementare il metodo `run`
- per crearla, la classe va istanziata, passata come argomento chiamando la classe `Thread`, e poi fatta partire con `start()`

Creazione di un nuovo thread (1)

```
class MioThread implements Runnable {
   MioThread(…) {
      // codice del costruttore
   }
   public void run() {
      // codice eseguito dal nuovo thread
   }
}
```

```
MioThread p = new MioThread(…);
new Thread(p).start();
```

```
class NuovoThreadRun implements Runnable {
    Thread t;
    NuovoThreadRun() {
        t = new Thread(this,"Thread secondario");
        System.out.println("\t\tThread figlio: " + t);
        t.start();                    // avvio il nuovo thread
    }
    public void run() {               // corpo del nuovo thread
        try {
            for (int n=5; n>0;n--) {
                System.out.println("\t\tThread figlio: " + n);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) { };
        System.out.println("\t\tTermine del thread figlio");
    }
}
```

```
class ThreadPrincipaleRun {
    public static void main(String args[]) {
        System.out.println("Thread padre: " + Thread.currentThread());
        new NuovoThreadRun();     // creo un nuovo thread secondario
        try {
            for (int n=5; n>0;n--) {
                System.out.println("Thread padre: " + n);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) { };
        System.out.println("Termine del thread padre");
    }
}
```

```
Thread padre: Thread[main,5,main]
        Thread figlio: Thread[Thread secondario,5,main]
Thread padre: 5
        Thread figlio: 5
        Thread figlio: 4
Thread padre: 4
        Thread figlio: 3
        Thread figlio: 2
Thread padre: 3
        Thread figlio: 1
        Termine del thread figlio
Thread padre: 2
Thread padre: 1
Termine del thread padre
```

Creazione di un nuovo thread (2)

Questo l'altro modo per creare un nuovo thread:

- dichiarare una classe derivata da `Thread`, che deve sovrascrivere il metodo `run`

```
class MioThread extends Thread {
   MioThread(…) {
      // codice del costruttore
   }
   public void run() {
      // codice eseguito dal nuovo thread
   }
}
```

```
MioThread p = new MioThread(…);
p.start();
```

```
class NuovoThreadThr extends Thread {
    NuovoThreadThr() {     // creo un nuovo thread con il costruttore
        super("Thread secondario"); // chiamo il costruttore della super
        System.out.println("\t\tThread figlio: " + this);
        start();       // avvio il nuovo thread
    }
    public void run() {  // corpo del nuovo thread
        try {
            for (int n=5; n>0;n--) {
                System.out.println("\t\tThread figlio: " + n);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) { };
        System.out.println("\t\tTermine del thread figlio");
    }
}
```

```
class ThreadPrincipaleThr {
    public static void main(String args[]) {
        System.out.println("Thread padre: " + Thread.currentThread());
        new NuovoThreadThr();        // creo un nuovo thread secondario
        try {
            for (int n=5; n>0;n--) {
                System.out.println("Thread padre: " + n);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) { };
        System.out.println("Termine del thread padre");
    }
}
```

```
Thread padre: Thread[main,5,main]
        Thread figlio: Thread[Thread secondario,5,main]
Thread padre: 5
        Thread figlio: 5
        Thread figlio: 4
Thread padre: 4
        Thread figlio: 3
        Thread figlio: 2
Thread padre: 3
        Thread figlio: 1
        Termine del thread figlio
Thread padre: 2
Thread padre: 1
Termine del thread padre
```

versione a più thread

```
class MultiThread implements Runnable {
    String nome;
    Thread t;
    MultiThread(String nomeThread) { // creo un nuovo thread
        nome = nomeThread;
        t = new Thread(this,nome);
        System.out.println("\t\tNuovo Thread: " + t);
        t.start();       // avvio il nuovo thread
    }
    public void run() {  // corpo del nuovo thread
        try {
            for (int n=5; n>0;n--) {
                System.out.println("\t\t"+nome+": " + n);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) { };
        System.out.println("\t\tTermine del thread "+nome);
    }
}
```

```
class ThreadMultipliDemo {
    public static void main(String args[]) {
        System.out.println("Thread padre: " + Thread.currentThread());
        new MultiThread("Uno");     // creo un nuovo thread secondario
        new MultiThread("Due");
        new MultiThread("Tre");
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) { };
        System.out.println("Termine del thread padre");
    }
}
```

```
Thread padre: Thread[main,5,main]
Nuovo Thread: Thread[Uno,5,main]
Nuovo Thread: Thread[Due,5,main]
Uno: 5
Nuovo Thread: Thread[Tre,5,main]
Tre: 5
Due: 5
Uno: 4
Tre: 4
Due: 4
Uno: 3
Tre: 3
Due: 3
Uno: 2
Due: 2
Tre: 2
Uno: 1
Due: 1
Tre: 1
Termine del thread Uno
Termine del thread Due
Termine del thread Tre
Termine del thread padre
```

```
class ThreadMultipliDemoOK {
    public static void main(String args[]) {
        System.out.println("Thread padre: " + Thread.currentThread());
        MultiThread th1 = new MultiThread("Uno");
        MultiThread th2 = new MultiThread("Due");
        MultiThread th3 = new MultiThread("Tre");
        try {
            th1.t.join();
            th2.t.join();
            th3.t.join();
        } catch (InterruptedException e) { };
        System.out.println("Termine del thread padre");
    }
}
```

Esercizio

Realizzare un programma in Java che attivi tre contatori indipendenti (tre thread concorrenti) e ne visualizzi i valori (totali o parziali) in un intervallo massimo di 5 secondi. Il contatore deve incrementare una proprietà interna (pubblica o privata) ed effettuare una visualizzazione ad ogni ciclo.

la priorità dei thread

- Java fissa una scala di valori, che è diversa nelle varie implementazione della JVM
- questa scala è garantita solo all'interno del linguaggio
- poi viene filtrata e adeguata dal S.O. e dall'ambiente hardware

```
class Contatore implements Runnable { // creo nuovo thread [Runnable]
    int conta = 0;
    Thread t;
    private boolean running = true;
    public Contatore(int p, String nomeThread) { // creo un nuovo thread
        t = new Thread(this,nomeThread);
        t.setPriority(p);
        System.out.println("Thread figlio: " + t);
    }
    public void run() {  // corpo del nuovo thread
        while (running) {
            conta++;
            System.out.print("\r");
        }
    }
    public void start() {
        t.start();
    }
    public void stop() {
        running = false;
    }
}
```

```
class ContatoreDemo {
    public static void main(String args[]) {
        Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
        System.out.println("Thread padre: " + Thread.currentThread());
        Contatore th1 = new Contatore(Thread.NORM_PRIORITY + 2,"Uno");
        Contatore th2 = new Contatore(Thread.NORM_PRIORITY - 2,"Due");
        th1.start();
        th2.start();
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) { };
        th1.stop();
        th2.stop();
        try {
            th1.t.join();
            th2.t.join();
        } catch (InterruptedException e) { };
        System.out.println("Thread ad alta priorita': "+th1.conta);
        System.out.println("Thread a bassa priorita': "+th2.conta);
    }
}
```

Thread padre: Thread[main,10,main]
Thread figlio: Thread[Uno,7,main]
Thread figlio: Thread[Due,3,main]
Thread ad alta priorita': 720371
Thread a bassa priorita': 3947

Esercizio

Realizzare un programma in Java che attivi tre contatori indipendenti (tre thread concorrenti) e ne visualizzi i valori (totali o parziali) in un intervallo massimo di 5 secondi. Ad ogni contatore venga assegnata una priorità iniziale diversa. Si analizzino i risultati proponendo le relative considerazioni.

relazione tra thread in Java e S.O.

l'applicazione multithread in Java viene vista dal S.O. come:

- come un unico thread o processo, da gestire in blocco (concorrenza a carico del linguaggio)
- l'applicazione è associata ad un solo processo, ed ogni thread del linguaggio viene associato ad un numero uguale o inferiore di thread del S.O. (Win NT/2000)
- ad ogni thread dell'applicazione un thread di S.O.

Method Summary (estratto)	
static Thread	currentThread() Returns a reference to the currently executing thread object.
String	getName() Returns this thread's name.
static Thread	currentThread() Returns a reference to the currently executing thread object.
int	getPriority() Returns this thread's priority.
boolean	isAlive() Tests if this thread is alive.
void	join() Waits for this thread to die.
void	run() If this thread was constructed using a separate `Runnable` run object, then that `Runnable` object's `run` method is called; otherwise, this method does nothing and returns.
void	setName(String name) Changes the name of this thread to be equal to the argument `name`.
void	setPriority(int newPriority) Changes the priority of this thread.
static void	sleep(long millis) Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
void	start() Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.