

Java Threads

a cura di Michele Franzin



Java User Group Padova



Quest'opera è protetta dalla licenza Creative Commons Attribution-ShareAlike 2.5; per vedere una copia di questa licenza, consultare:

<http://creativecommons.org/licenses/by-sa/2.5/deed.it> oppure inviare una lettera a: Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

This work is licensed under the Creative Commons Attribution-ShareAlike 2.5 License; to view a copy of this license, visit:

<http://creativecommons.org/licenses/by-sa/2.5/deed.en> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.



Multithreading Tips & Tricks

(aka nozioni avanzate sul locking ed altre oscure amenità del multithreading)



- Deadlock
- Race Condition
- Starvation
- Locking
- Operazioni [non] atomiche
- Mr. Singleton
- Memory model
- Sincronizzazione
- Soluzioni



29-30 Settembre e 1 Ottobre 2006



- La morte nera...

```
final Object resource1 = "resource1";
final Object resource2 = "resource2";

Thread t1 = new Thread() {
    public void run() {
        synchronized(resource1) {
            System.out.println(
                "Thread 1: locked resource 1");
            try { Thread.sleep(50); }
            catch (InterruptedException e) {}
            synchronized(resource2) {
                System.out.println(
                    "Thread 1: locked resource 2");
            }
        }
    }
};
```





```
Thread t2 = new Thread() {
    public void run() {
        synchronized(resource2) {
            System.out.println(
                "Thread 2: locked resource 2");
            try { Thread.sleep(50); }
            catch (InterruptedException e) {}
            synchronized(resource1) {
                System.out.println(
                    "Thread 2: locked resource 1");
            }
        }
    }
};

t1.start(); t2.start();
```

A screenshot of an IDE's console window. The window title is 'New_configuration [Java Application] /opt/jdk1.5.0/bin/java (23-ott-2004 8.22.07)'. The console output shows two lines of text: 'Thread 1: locked resource 1' and 'Thread 2: locked resource 2'. The IDE interface includes tabs for 'Hibernate', 'Console', 'Problems', 'Ant', 'Declaration', and 'Javadoc'. The 'Problems' tab is active, and the console output is visible below it.

```
Hibernate Console Problems Ant Declaration Javadoc
New_configuration [Java Application] /opt/jdk1.5.0/bin/java (23-ott-2004 8.22.07)
Thread 1: locked resource 1
Thread 2: locked resource 2
```

- Il programma **non ha un esito scontato**, ma dipende dall'esito della “competizione” per le risorse

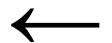
```
public class Race {  
  
    class Runner extends Thread {  
        public Object server;  
  
        public Runner(String name) {  
            super(name);  
        }  
  
        public void run() {  
            System.out.println(getName() +  
                " ha vinto la risorsa!");  
        }  
    }  
}
```



```
public static void main (String args[]) {
    Race race = new Race();
    Runner gianni = race.new Runner("Gianni");
    Runner alice = race.new Runner("Alice");

    gianni.server = new Object();
    alice.server = new Object();

    if (Math.random() > .5) {
        gianni.start();
        alice.start();
    } else {
        alice.start();
        gianni.start();
    }
}
```





```
$ java Race
Alice ha vinto la risorsa!      Gianni ha vinto la risorsa!
$ java Race
Gianni ha vinto la risorsa!    Alice ha vinto la risorsa!
$ java Race
Alice ha vinto la risorsa!    Gianni ha vinto la risorsa!
$ java Race
Gianni ha vinto la risorsa!    Alice ha vinto la risorsa!
$ java Race
Gianni ha vinto la risorsa!    Alice ha vinto la risorsa!
$ java Race
Alice ha vinto la risorsa!    Gianni ha vinto la risorsa!
$ java Race
Alice ha vinto la risorsa!    Gianni ha vinto la risorsa!
$ java Race
Gianni ha vinto la risorsa!    Alice ha vinto la risorsa!
$ java Race
Alice ha vinto la risorsa!    Gianni ha vinto la risorsa!
```



- Le specifiche della JVM non indicano come i thread Java debbano essere "mappati" sulle funzioni multi-tasking del S.O. sottostante

Diverse piattaforme → *Diverse implementazioni*

- La gestione dei thread in genere dipende dalla piattaforma sulla quale Java è in esecuzione
 - ✓ il più delle volte è di tipo preemptive
 - ✗ nelle altre piattaforme può accadere che un solo thread acquisisca il controllo del processore, impedendo ad altri di essere eseguiti

- Lo starvation accade quando un thread a bassa priorità non ha mai l'occasione di girare a causa di quelli a priorità più alta

```
public class Starvation {  
  
    class Runner extends Thread {  
        int count = 100;  
  
        public void run() {  
            for (int i=0; i<count; i++)  
                System.out.println(getName()  
                    + " sta elaborando " + i);  
        }  
    }  
}
```



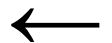


```
public static void main(String args[]) {
    Starvation starvation = new Starvation();

    for (int i = 1; i < 10; i++) {
        Runner r = starvation.new Runner();
        r.setPriority(Thread.MAX_PRIORITY);

        if (i == 1) {
            r.setPriority(Thread.MIN_PRIORITY);
            r.setName("Starvation Thread");
        }
        r.start();
    }

    System.exit(0);
}
```

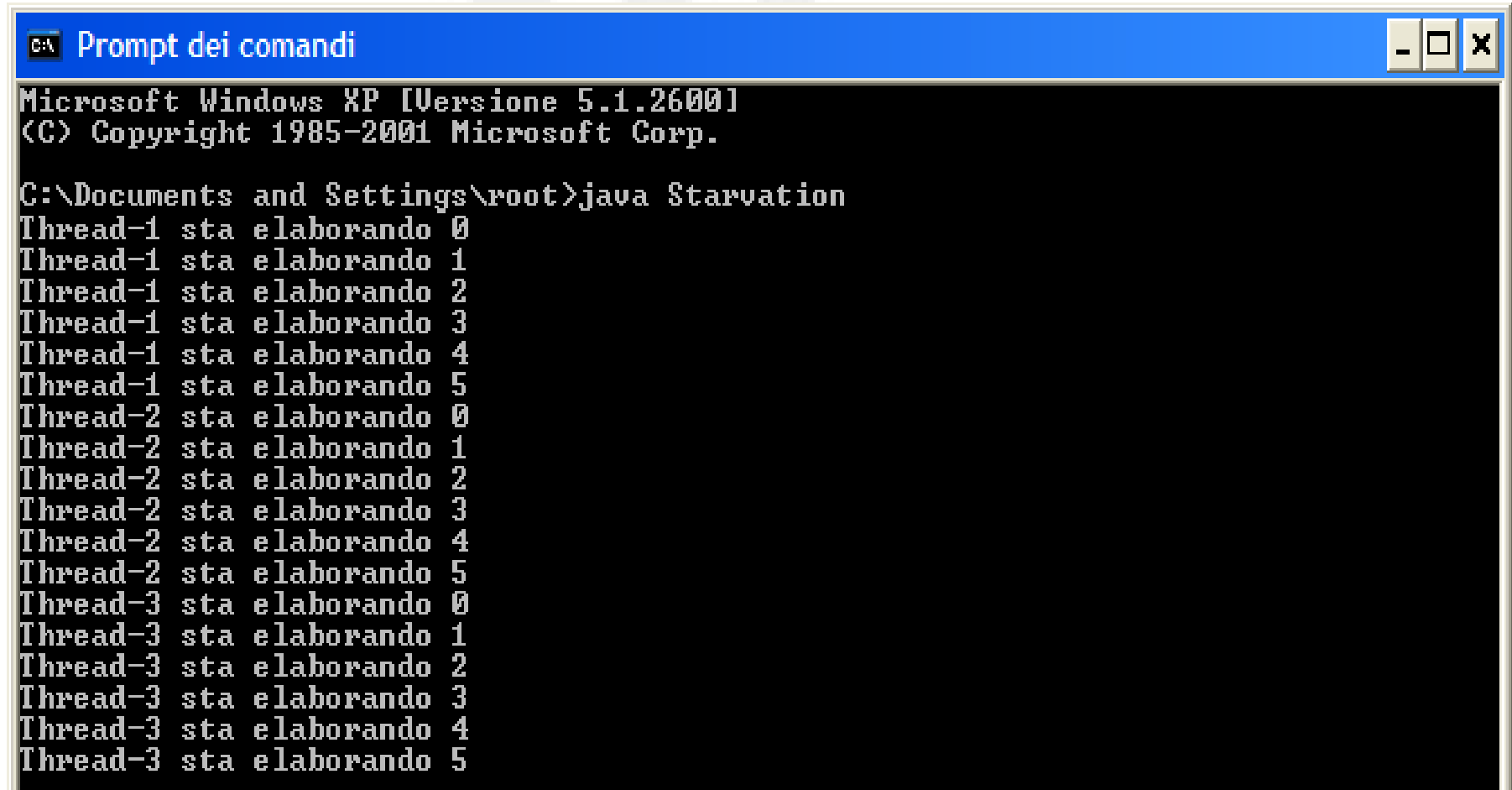




- 8 Processi
- 100 Elaborazioni

```
$ java Starvation
Thread-1 sta elaborando 0-26
Thread-5 sta elaborando 0-99
Thread-6 sta elaborando 0-99
Thread-1 sta elaborando 27-99
Thread-8 sta elaborando 0-99
Thread-7 sta elaborando 0
Thread-4 sta elaborando 0-70
Thread-3 sta elaborando 0
Thread-2 sta elaborando 0-99
Thread-3 sta elaborando 1-32
```

- 4 Processi
- 6 Elaborazioni

A screenshot of a Windows command prompt window titled 'Prompt dei comandi'. The window shows the output of running the command 'java Starvation'. The output consists of 18 lines of text, grouped by thread. Each thread (Thread-1, Thread-2, and Thread-3) prints 'sta elaborando' followed by a number from 0 to 5. The threads appear to be running in parallel, with Thread-1 starting first, followed by Thread-2, and then Thread-3.

```
Prompt dei comandi
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\root>java Starvation
Thread-1 sta elaborando 0
Thread-1 sta elaborando 1
Thread-1 sta elaborando 2
Thread-1 sta elaborando 3
Thread-1 sta elaborando 4
Thread-1 sta elaborando 5
Thread-2 sta elaborando 0
Thread-2 sta elaborando 1
Thread-2 sta elaborando 2
Thread-2 sta elaborando 3
Thread-2 sta elaborando 4
Thread-2 sta elaborando 5
Thread-3 sta elaborando 0
Thread-3 sta elaborando 1
Thread-3 sta elaborando 2
Thread-3 sta elaborando 3
Thread-3 sta elaborando 4
Thread-3 sta elaborando 5
```

- Accade sovente in sistemi che adottano scheduling di tipo preemptive (Unix)
- È di difficile dimostrazione su sistemi che utilizzano che usano meccanismi di time-slicing (Windows)
- ✓ È possibile modificare le priorità dei thread per non incappare nella starvation ...
- ✗ ... ma è una sistema di “cura”, non la prevenzione del problema

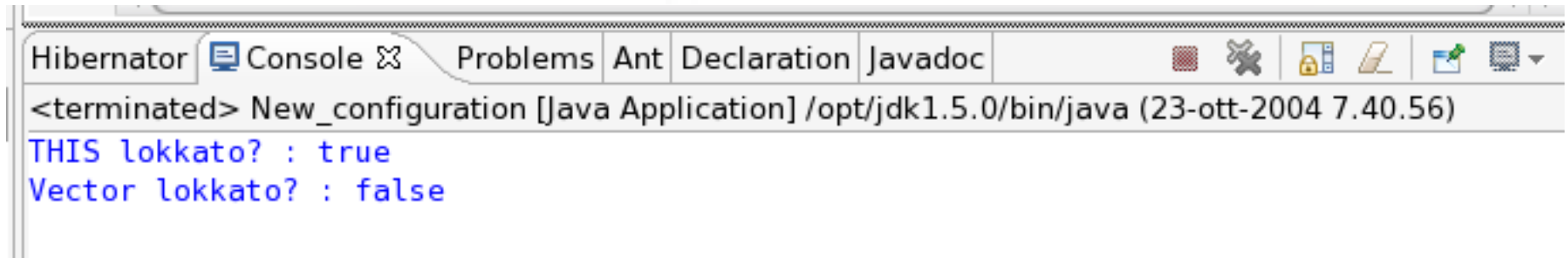


```
public class Test {
    private Vector vector = new Vector();

    public synchronized void doTest() {
        System.out.println("THIS lokkato? : " +
            Thread.currentThread().holdsLock(this));
        System.out.println("Vector lokkato? : " +
            Thread.currentThread().holdsLock
                (vector));
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.doTest();
    }
}
```

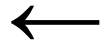

- Acquisire il lock su un'istanza implica acquisire il lock sui membri di quell'istanza ?



```

Hibernator Console Problems Ant Declaration Javadoc
<terminated> New_configuration [Java Application] /opt/jdk1.5.0/bin/java (23-ott-2004 7.40.56)
THIS lokkato? : true
Vector lokkato? : false
  
```

NO !



```
public class Test2 {

    public static synchronized void lockTest() {
        Test2 test = new Test2();
        System.out.println("Classe lokkata? : " +
            Thread.currentThread().holdsLock
                (test.getClass()));
        System.out.println("Istanza lokkata? : " +
            Thread.currentThread().holdsLock(test));
    }

    public static void main(String args[]) {
        //
        lockTest();
    }
}
```

- Acquisire il lock su un'istanza implica acquisire il lock sui membri di quell'istanza ?

```

Hibernator Console Problems Ant Declaration Javadoc
<terminated> New_configuration [Java Application] /opt/jdk1.5.0/bin/java (23-ott-2004 7.48.06)
Classe lокkata? : true
Istanza lокkata? : false
  
```

NO !

- Le operazioni atomiche sono indivisibili, sia in lettura che in scrittura
- Un thread non viene mai interrotto nel mezzo di un'operazione atomica
- L'assegnazione è un'operazione atomica **solo** se coinvolge interi!

```
x = 7;
```

```
y = x++;
```

in realtà è:

```
x = 7;
```

```
x = x + 1;
```

```
y = x;
```



PERICOLO !

```
Singleton singleton = Singleton.getInstance();
```

```
public class Singleton {
    private static Singleton uniqueInstance = null;
    private Singleton() {}
    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
}
```

```
Singleton singleton = Singleton.getInstance();
```

```
public class Singleton {
    private static Singleton uniqueInstance = null;
    private Singleton() {}
    public static synchronized Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
}
```

```
Singleton singleton = Singleton.getInstance();
```

```
public class Singleton {
    private static Singleton uniqueInstance = null;
    private Singleton() {}
    public static synchronized Singleton getInstance() {
        if (uniqueInstance == null) {
            synchronized (uniqueInstance) {
                uniqueInstance = new Singleton();
            }
        }
        return uniqueInstance;
    }
}
```

```
Singleton singleton = Singleton.getInstance();
```

```
public class Singleton {
    private static Singleton uniqueInstance = null;
    private Singleton() {}
    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            synchronized (Singleton.class) {
                if (uniqueInstance == null) {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}
```



```
Singleton singleton = Singleton.getInstance();
```

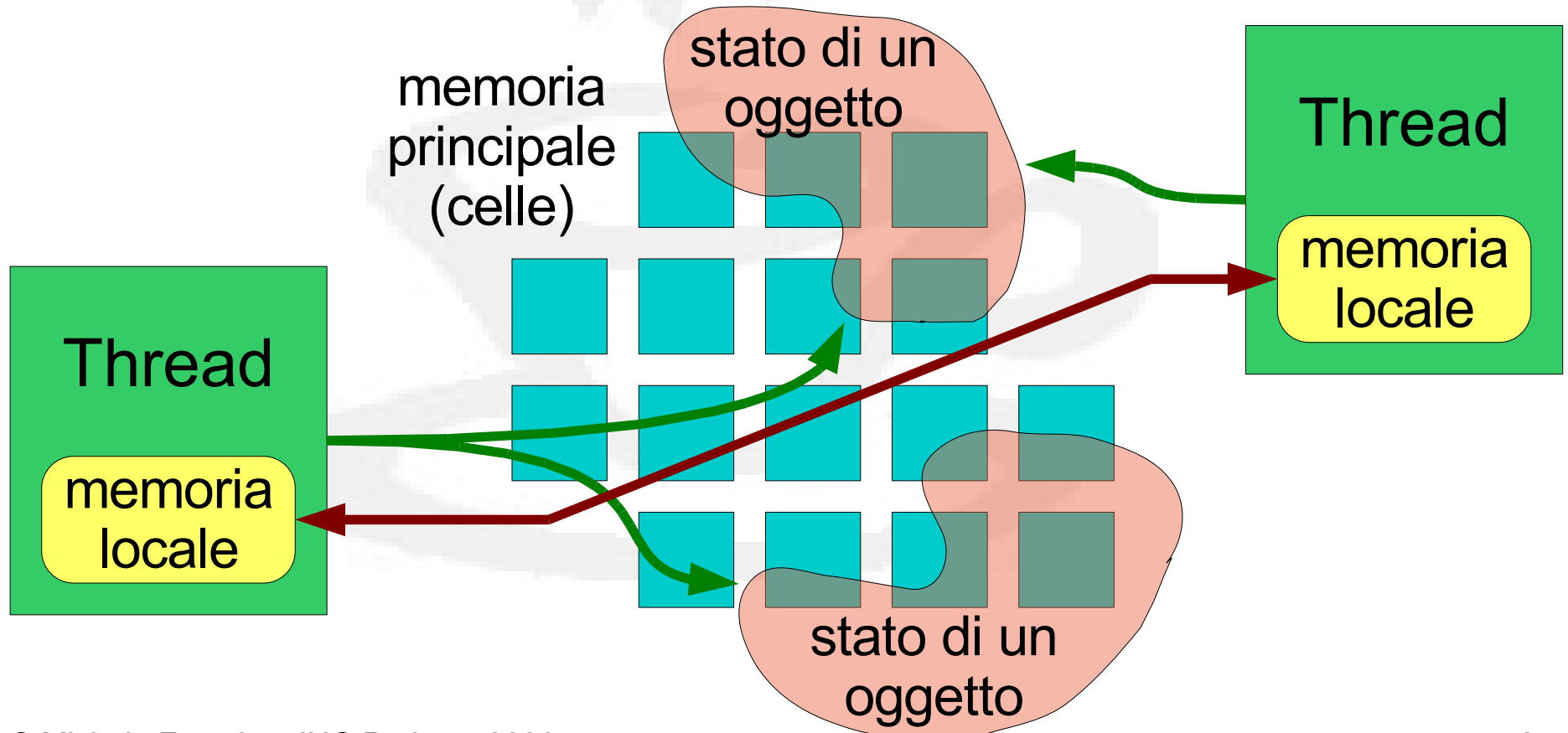
```
public class Singleton {
    private volatile static Singleton uniqueInstance;
    private Singleton() {}

    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            synchronized (Singleton.class) {
                if (uniqueInstance == null) {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
} //Singleton not guaranteed to work prior to Java 5
```



- Stabilisce la relazione tra le variabili di un programma e i dettagli per lettura e scrittura (a basso livello) nella memoria del PC
- Compilatore, JVM, processore e cache possono liberamente decidere **se e quando** muovere un valore dalla/nella cella di memoria assegnatagli
- Ambizioso = Confuso + portabile
 - ✗ Comprensione non intuitiva
 - ✓ Indipendenza dalla piattaforma fisica
- Atomicità, Visibilità, Ordinamento

- Memoria principale **condivisa** da tutti i thread
- Memoria locale (cache) **esclusiva** di ogni thread





- Anche se le operazioni (eccetto `long` e `double`) sono eseguite atomicamente ...
- ... le specifiche del linguaggio permettono ai thread l'uso di copie di lavoro (memoria locale) delle variabili condivise, permettendo un modello di multi-threading più efficiente
 - velocità d'accesso
 - registri CPU
 - cache del processore
- ✓ La sincronizzazione non fa uso di meccanismi di caching, ma ...
- ✗ ... l'uso eccessivo abbassa le prestazioni!

```

public class Termina {
    public static void main(String[] args) {
        Arbitro a = new Arbitro();
        Sommatore s = new Sommatore(a);
        s.start();
        a.start();
    }
}

class Sommatore extends Thread {
    private int somma = 0;
    public boolean termina = false;

    public void run() {
        while(!termina) {
            System.out.println("Somma=" + somma++);
            Thread.yield();
        }
    }
}

```

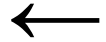


```
class Arbitro extends Thread {  
    private Sommatore s = null;  
  
    public Arbitro(Sommatore sommatore) {  
        s = sommatore;  
    }  
  
    public void run() {  
        try {  
            Thread.sleep(5000);  
        } catch (InterruptedException e) {  
            // coding style example...  
        }  
        s.termina = true;  
    }  
}
```





```
class Sommatore extends Thread {  
    private int somma = 0;  
    public volatile boolean termina = false;  
  
    public void run() {  
        while(!termina) {  
            System.out.println("Somma=" + somma++);  
            Thread.yield();  
        }  
    }  
}
```



- Impedisce alla JVM di utilizzare la memoria locale
- Garantisce che ogni thread acceda sempre al valore `termina` dalla memoria principale (unico per tutti i thread)

liscio

L'utilizzo della memoria locale è gestito dalla JVM con criteri dipendenti dall'implementazione: se un thread modifica una variabile locale, altri possono non “vedere” i cambiamenti

synchronized

Utilizza la memoria locale solamente nella sezione critica

“allinea” i valori della memoria principale solamente in:

- entrata nella sezione critica (acquisizione del lock)
- uscita dalla sezione critica (rilascio del lock)

volatile

I valori vengono letti/scritti nella memoria principale ad ogni utilizzo, nell'esatto ordine richiesto dal thread

- I threads bloccati non consumano cicli CPU e sono in uno stato di “animazione sospesa”
- Non attendono che un certo tempo sia passato, ma il verificarsi di un evento
- Usando un runtime antecedente alla release 1.4 il **locking** può spesso **sfuggire al controllo** del programmatore

Non a voi? Sicuri !?!? ...volete un esempio?

l'apertura di una connessione di rete ;-)

```
import java.net.*;

public class EsempioBlocco extends Thread {
    Object blocco = new Object();

    public static void main(String args[]) throws
Exception {
        EsempioBlocco e = new EsempioBlocco();
        e.start();
        Thread.currentThread().sleep(10);
        e.esegui();
    }

    public void esegui() {
        synchronized (blocco) {
            System.out.println("eseguo");
        }
    }
}
```





```
public void run() {  
    blocca();  
}  
  
public void blocca() {  
    synchronized (blocco) {  
        System.out.println("sono qui");  
        try {  
            ServerSocket ss = new ServerSocket(8080);  
        } catch (Exception e) {}  
    }  
}
```

```
$ java EsempioBlocco  
sono qui'
```



- Un thread che non è più running **mantiene** il blocco sulle risorse che utilizza!
- Non ricevendo l'evento che attende, non è in grado di rilasciare il blocco nè di destarsi
- Cosa posso fare ?
 - Usare J2SE \geq 1.4
 - Gestire correttamente `InterruptedException`
 - Scegliere operazioni di I/O non bloccanti (`java.nio.channels`)



- JavaDoc!
 - Responsabilità tra classe e utente
 - Caratteristiche di comportamento
 - E' sempre troppo tardi...
 - `SimpleDateFormat` per `JDK < 1.4`

...il codice necessita di sincronizzazione esterna?

- Operazioni singole: singola chiamata a metodo
- Operazioni multiple: più chiamate a metodi in una singola "unità" (esempio `Vector.Iterator()`)

Consistenza dello “stato” interno

- Non condivisa
 - thread-immutable: classe `String`
 - thread-compatible: utente `ArrayList`
 - thread-hostile: raro `System.setOut()`
- Condivisa tra utente e classe
 - thread-safe: operazioni multiple
 - conditional thread-safe: operazioni singole

- ✓ “Java Developer Exam with J2SE 1.4”. Mehran Habibi, Jeremy Patterson, Terry Camerlengo – Apress
- ✓ “Java Examples in a Nutshell”. David Flanagan – O'Reilly
- ✓ “Effective Java”. Joshua Bloch – Addison Wesley



- ✓ “The ABC of Synchronization 1/2”. Jeff Friesen

<http://today.java.net/>

- ✓ “Concurrent Programming in Java - Design principles and patterns – Online Supplement”. Doug Lea

<http://gee.cs.oswego.edu/dl/cpj/>

- ✓ “Java Theory and practice : Fixing the Java Memory Model”. Brian Goetz

<http://www.ibm-106.com/>

- ✓ **“Characterizing thread safety”. Brian Goetz**

<http://www.ibm-106.com/>

- ✓ **Concurrency Utilities Overview**

<http://java.sun.com/j2se/1.5.0/docs/guide/concurrency/overview.html>

- ✓ **“JSR 133: Java Memory Model and Thread Specification Revision”. Java Community Process**

<http://www.jcp.org/>



Sito Web

<http://www.jugpadova.it>



Mailing List

http://groups.yahoo.com/group/JUG_Padova/

Persone di riferimento



Dario Santamaria (dario.santamaria@jugpadova.it)

Lucio Benfante (lucio.benfante@jugpadova.it)

Paolo Donà (paolo.dona@jugpadova.it)