



05:23

Richiedi controllo



Abbandona

Programmazione Concorrente: Thread in Java - Parte 3

Programmazione Distribuita - A.A. 2020/2021



Biagio Cosenza
Dipartimento di Informatica
Università di Salerno
<http://cosenza.eu/>
bcosenza@unisa.it



Biagio COSENZA

SD

SALVATORE DANESE

MR

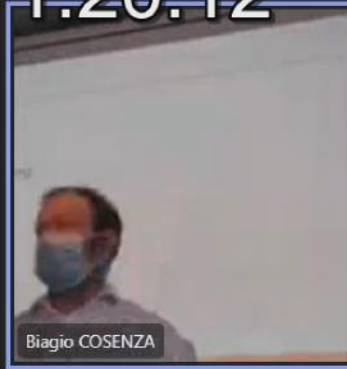
MARIAROS...

+50



Organizzazione della Lezione

- Esempi di race condition
- Efficienza dei thread
- Singleton ed il double-checked locking
- Conclusioni



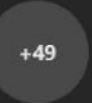
Biagio COSENZA



SALVATORE DANESE

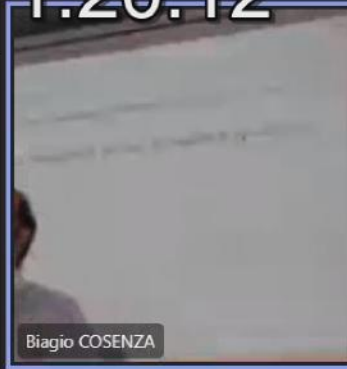


MARIAROS...



Esempi di Race Condition

- È necessario comprendere bene in che maniera i metodi `synchronized` sono eseguiti in mutua esclusione
- Vediamo un esempio semplice che, al variare di alcuni semplici keyword, si comporta in maniera diversa



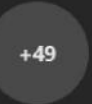
Biagio COSENZA



SALVATORE DANESE



MARIAROS...

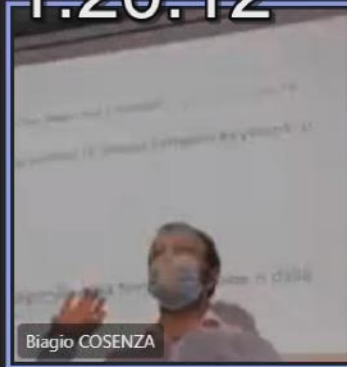


+49



Esempi di Race Condition

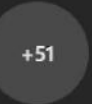
- È necessario comprendere bene in che maniera i metodi `synchronized` sono eseguiti in mutua esclusione
- Vediamo un esempio semplice che, al variare di alcuni semplici keyword, si comporta in maniera diversa
- Ricordiamo: **race condition**
 - il risultato finale dell'esecuzione dei processi dipende dalla temporizzazione o dalla sequenza con cui vengono eseguiti



SALVATORE DANESE



MARIAROS...



Esempio (1 di 2)

1. Uso dei logger
 - un logger di chiamato Example
2. **L'oggetto** su cui invocare i metodi
3. Creazione di due thread
 - e loro lancio
4. Stampiamo chi entra
 - e attende
5. **catch** della eccezione che può lanciare la `sleep()` se interrotta
 - (**SimpleThread**, a seguire)

```
import java.util.logging.Logger;
public class Example {

    private static Logger log =
        Logger.getLogger("Example");

    public static void main(String[] args) {
        Example ob = new Example();
        log.info("Creating threads");
        SimpleThread one = new SimpleThread("one", ob, 2);
        SimpleThread two = new SimpleThread("two", ob, 2);
        one.start();
        two.start();
    }

    public synchronized void firstWaitingRoom(int sec, String name) {
        log.info(name+"..entering.");
        try {
            Thread.sleep(sec*1000);
        } catch (InterruptedException e) { e.printStackTrace(); }
        log.info(name+"..exiting.");
    }

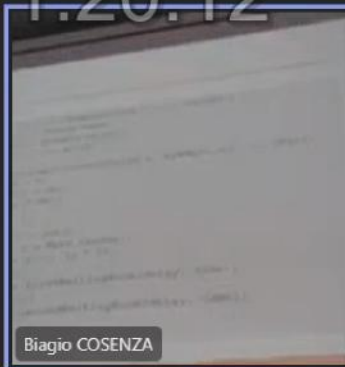
    public synchronized void secondWaitingRoom(int sec, String name) {
        log.info(name+"..entering.");
        try {
            Thread.sleep(sec*1000);
        } catch (InterruptedException e) { e.printStackTrace(); }
        log.info(name+"..exiting.");
    }
}
```



Esempio (2 di 2) : SimpleThread

1. Un thread
2. Variabili istanza
3. Costruttore
4. Metodo run da eseguire
5. Un double a caso tra 0 e <1
6. Arrotondamento ad un intero
 - risultato è 0 oppure 1
7. A seconda del valore si entra nella prima
 - o nella seconda sala di attesa

```
public class SimpleThread extends Thread {  
    private String name;  
    private Example object;  
    private int delay;  
  
    public SimpleThread(String n, Example obj, int del){  
        name = n;  
        object = obj;  
        delay = del;  
    }  
  
    public void run() {  
        double y = Math.random();  
        int x = (int) (y * 2);  
        if(x==0)  
            object.firstWaitingRoom(delay, name);  
        else //x==1  
            object.secondWaitingRoom(delay, name);  
    }  
}
```



Biagio COSENZA



SALVATORE DANESE 🗣️



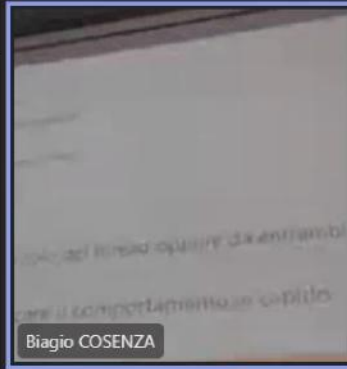
MARIAROS... 🗣️





Esempio: Casi possibili

- Ci sono due thread che cercano di accedere
 - a due metodi
 - a volte allo stesso metodo, contemporaneamente
- I metodi possono essere
 - di istanza oppure statici
 - sincronizzati oppure non sincronizzati
- Un metodo può essere invocato da uno solo dei thread oppure da entrambi i thread
- Per ogni combinazione si dovrebbe verificare il comportamento, e capirlo completamente



SALVATORE DANESE



MARIAROS...



Esempio: Caso 1: metodi di istanza sincronizzati

- Supponiamo che debbano accedere alla **stessa** sala d'attesa
- Entrambi i metodi sono sincronizzati

```
public synchronized void firstWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}  
  
public synchronized void secondWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try{  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}
```

Biagio COSENZA

SD

SALVATORE DANESE

MR

MARIAROS...

+54





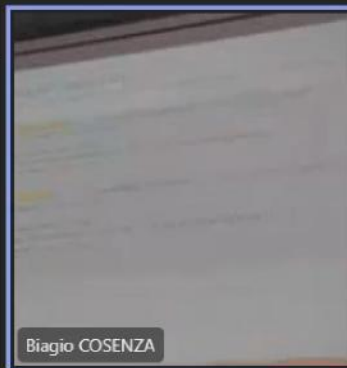
Esempio: Caso 1: metodi di istanza sincronizzati

- Supponiamo che debbano accedere alla **stessa** sala d'attesa
- Entrambi i metodi sono sincronizzati

```
public synchronized void firstWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}  
  
public synchronized void secondWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}
```

Problems @ Javadoc Declaration Console

```
<terminated> Example [Java Application] /System/Library/Java/Java  
6-ott-2013 17.14.51 Example main  
INFO: Creating threads  
6-ott-2013 17.14.51 Example secondWaitingRoom  
INFO: one.. entering.  
6-ott-2013 17.14.53 Example secondWaitingRoom  
INFO: one.. exiting.  
6-ott-2013 17.14.53 Example secondWaitingRoom  
INFO: two.. entering.  
6-ott-2013 17.14.55 Example secondWaitingRoom  
INFO: two.. exiting.
```



Biagio COSENZA



SALVATORE DANESE



MARIAROS...



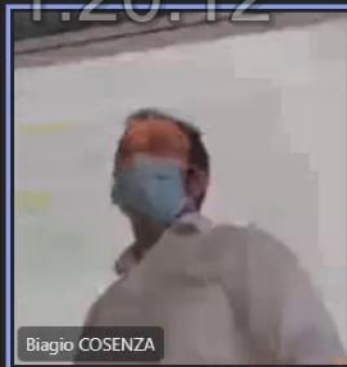
+54



Esempio: Caso 2: metodi di istanza sincronizzati

- Supponiamo che debbano accedere a sale di attesa **diverse**
- Entrambi i metodi sono sincronizzati

```
public synchronized void firstWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}  
  
public synchronized void secondWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try{  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}
```



Biagio COSENZA



SALVATORE DANESE



MARIAROS...



Esempio: Caso 2: metodi di istanza sincronizzati

- Supponiamo che debbano accedere a sale di attesa **diverse**
- Entrambi i metodi sono sincronizzati

```
public synchronized void firstWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}  
  
public synchronized void secondWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}
```

Problems @ Javadoc Declaration Console

<terminated> Example [Java Application] /System/Library/Java/J

6-ott-2013 17.15.08 Example main
INFO: Creating threads
6-ott-2013 17.15.08 Example firstWaitingRoom
INFO: one.. entering.
6-ott-2013 17.15.10 Example firstWaitingRoom
INFO: one.. exiting.
6-ott-2013 17.15.10 Example secondWaitingRoom
INFO: two.. entering.
6-ott-2013 17.15.12 Example secondWaitingRoom
INFO: two.. exiting.

Biagio COSENZA

SD

SALVATORE DANESE

MR

+53

MARIAROS...



Esempio: Caso 3: un solo metodo di istanza sincronizzati

- Primo metodo sincronizzato
- Secondo metodo no
- Si accede a due sale di attesa diverse
- No mutua esclusione

```
public synchronized void firstWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try {  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}  
  
public void secondWaitingRoom(int sec, String name) {  
    log.info(name+"..entering.");  
    try{  
        Thread.sleep(sec*1000);  
    } catch (InterruptedException e) { e.printStackTrace(); }  
    log.info(name+"..exiting.");  
}
```

Problems Javadoc Declaration Console

```
<terminated> Example [Java Application] /System/Library/Java  
6-ott-2013 17.14.04 Example main  
INFO: Creating threads  
6-ott-2013 17.14.04 Example secondWaitingRoom  
INFO: one.. entering.  
6-ott-2013 17.14.04 Example firstWaitingRoom  
INFO: two.. entering.  
6-ott-2013 17.14.06 Example firstWaitingRoom  
INFO: two.. exiting.  
6-ott-2013 17.14.06 Example secondWaitingRoom  
INFO: one.. exiting.
```



Biagio COSENZA

SD

SALVATORE DANESE

MR

MARIAROS...

+53



Efficienza dei Thread

- Esempio: Inizializziamo un array di interi
 - l'array è "grande" (10 milioni di elementi)
 - e ogni inizializzazione viene ripetuta 10000 volte



Biagio COSENZA



SALVATORE DANESE

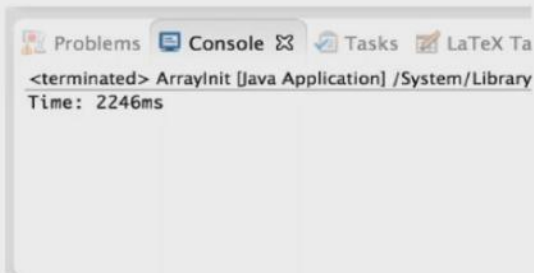


MARIAROS...



Soluzione semplice

1. 10 milioni di elementi
2. si prende il clock
3. per ogni elemento
 - ripetiamo 10000 volte assegnazione
4. fermiamo il clock



```
<terminated> ArrayInit [Java Application] /System/Library
Time: 2246ms
```

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 100000000;

    public static void main(String[] args){
        data = new int[SIZE];
        long begin, end;
        int j;
        begin = System.currentTimeMillis();
        for(int i = 0; i < SIZE; i++){
            for(j=0; j < 10000; j++)
                data[i] = i;
        }
        end = System.currentTimeMillis();
        System.out.println("Time:" + (end - begin) + "ms");
    } //end main
} //end class
```



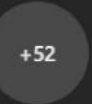
Biagio COSENZA



SALVATORE DANESE



MARIAROS...

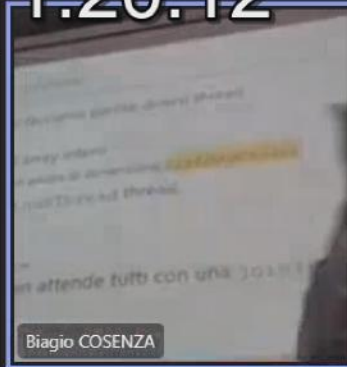


+52



Soluzione con inizializzazione concorrente

- L'idea: dividiamo l'array in blocchi e facciamo partire diversi thread
 - e ciascuno inizializza il proprio blocco
- Quindi, se `SIZE` è la dimensione dell'array intero
 - ognuno dei `numThread` thread prende un pezzo di dimensione `SIZE/numThread`
- Quindi, il thread main istanzia e lancia `numThread` thread
 - Al costruttore dei thread si passano:
 - la posizione di partenza nell'array: `start`
 - quanti elementi devono essere inizializzati: `dim`
- Dopo aver lanciato i thread, il thread main attende tutti con una `join()`
- Ripetiamo questo per `1, 2, \dots, MAX_THR`



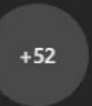
Biagio COSENZA



SALVATORE DANESE



MARIAROS...



Soluzione efficiente (1 di 2)

1. Classe che estende Thread
2. variabili private del thread
 - un array di interi, ...
3. Un array di thread
 - 8 thread totali
4. Codice di timing (prende il clock)
5. Prima posizione
6. Per ogni thread
 - parametri al costruttore di EffInit
 - creazione e lancio
7. Il main thread fa join su ogni thread
 - aspetta che termini

```
public class EffInit extends Thread {
    private int start;
    private int dim;
    public static int[] data;
    public static final int SIZE = 10000000;
    public static final int MAX_THR = 8;

    public static void main(String[] args) {
        data = new int[SIZE];
        long begin, end;
        int start, j;
        EffInit[] threads;
        for(int numThread = 1; numThread <= MAX_THR; numThread++){
            begin = System.currentTimeMillis();
            start = 0;
            threads = new EffInit[numThread];
            for(j = 0; j < numThread; j++){
                threads[j] = new EffInit(start, SIZE/numThread);
                threads[j].start();
                start += SIZE / numThread;
            }
            for(j = 0; j < numThread; j++){
                try {
                    threads[j].join();
                } catch (InterruptedException e) { e.printStackTrace(); }
            }
            end = System.currentTimeMillis();
            System.out.println(numThread + "Thread(s):" + (end - begin) + "ms");
        }
    }
}
```

Biagio COSENZA

SD

SALVATORE DANESE

MR

+52

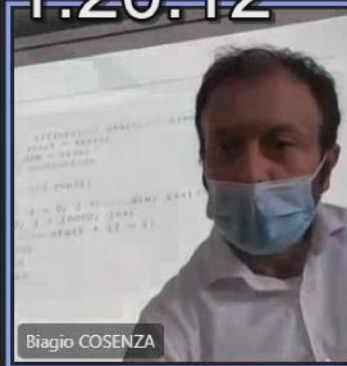
MARIAROS...



Soluzione efficiente (2 di 2)

1. Costruttore con due parametri
 - inizio e dimensione del blocco
2. Metodo run ()
3. Per tutti gli elementi del blocco
 - inizializzazione
 - ripetuta 10000

```
//...  
  
public EffInit(int start,int size){  
    this.start = start;  
    this.dim = size;  
} //end costruttore  
  
public void run(){  
    int j;  
    for(int i = 0; i < this.dim; i++){  
        for(j=0; j < 10000; j++){  
            data[this.start + i] = i;  
        } //end for  
    } //end run  
} //end class
```



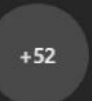
Biagio COSENZA



SALVATORE DANESE



MARIAROS...



+52

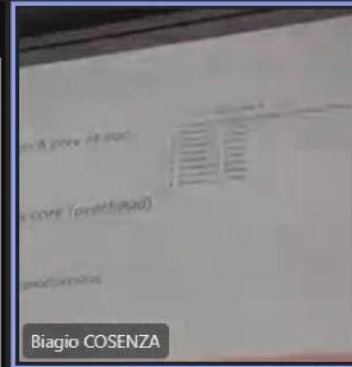




Soluzione efficiente:

- Verifichiamo l'efficienza
- Una esecuzione su una macchina con 8 core (4 con hyperthreading)
- Alcuni commenti:
- Miglioramenti non lineari al numero di core (*overhead*)
 - Motivi:
 - codice molto semplice, poca computazione
 - overhead di creazione dei threads rilevante in proporzione

```
Problems Console Tasks LaTeX T
<terminated> Effinit [Java Application] /System/Library/J
1 Thread(s): 2259ms
2 Thread(s): 1158ms
3 Thread(s): 812ms
4 Thread(s): 741ms
5 Thread(s): 676ms
6 Thread(s): 615ms
7 Thread(s): 566ms
8 Thread(s): 584ms
```



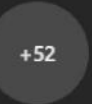
Biagio COSENZA



SALVATORE DANESE

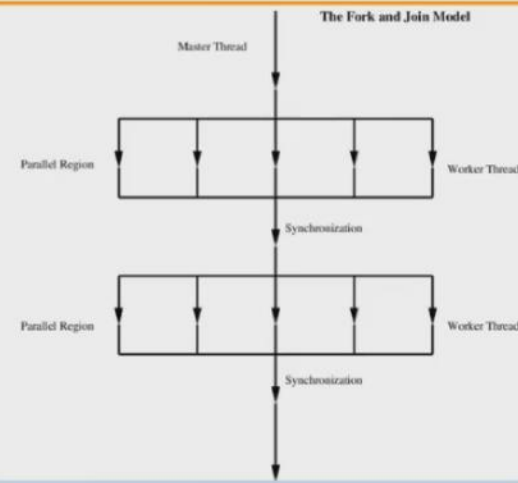


MARIAROS...

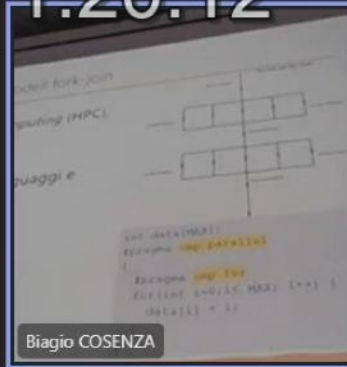


Considerazioni su l'Efficienza: modell fork-join

- In ambito di *high-performance computing* (HPC), l'efficienza e' critica
 - fork-join parallelism
- Supporto fork-join all'interno dei linguaggi e compilatori
- Esempi:
 - OpenMP (C/C++)
 - SYCL (C++)



```
int data[MAX];
#pragma omp parallel
{
    #pragma omp for
    for(int i=0; i< MAX; i++) {
        data[i] = i;
    }
}
```



SALVATORE DANESE



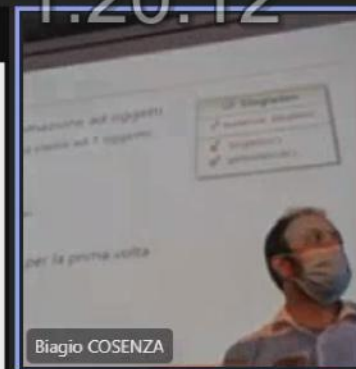
MARIAROS...

+52



Singleton

- Noto design pattern della programmazione ad oggetti
 - restringe la istanziiazione da parte di una classe ad 1 oggetto
- Usi
 - memorizzazione di stato
 - ad esempio, Printer, File, Resource Manager, . . .
- Lazy allocation
 - L'allocazione avviene solo quando utilizzato per la prima volta



SALVATORE DANESE



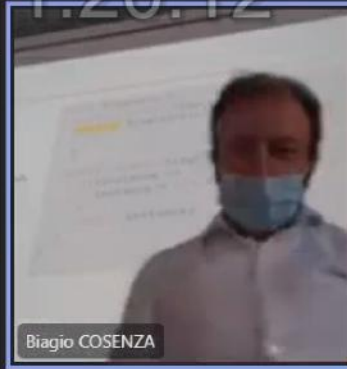
MARIAROS...



Singleton in Java

- Classico codice per singleton
 - variabile di classe che mantiene l'istanza
 - costruttore **privato**
 - metodo di classe che restituisce un Singleton
 - se non è stato ancora creata una istanza
 - allora creala
 - alla fine, restituisci la nuova istanza a disposizione

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



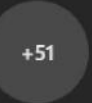
Biagio COSENZA



SALVATORE DANESE



MARIAROS...



+51



Singleton in Java

- Classico codice per singleton
 - variabile di classe che mantiene l'istanza
 - costruttore **privato**
 - metodo di classe che restituisce un Singleton
 - se non è stato ancora creata una istanza
 - allora creala
 - alla fine, restituisci la nuova istanza a disposizione
- Come si può implementare il Singleton pattern in un contesto multi-threading?
 - interleaving dei thread puo creare errore

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



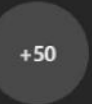
Biagio COSENZA



SALVATORE DANESE



MARIAROS...





Singleton in Java con Sincronizzazione

- Variabile di classe che mantiene l'istanza
 - costruttore privato
 - metodo sincronizzato
 - se non è stato ancora creata una istanza
 - ... allora creala
 - restituisci la nuova istanza
- Problemi?
- Efficienza!

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



Biagio COSENZA



SALVATORE DANESE



MARIAROS...



+50



Singleton in Java con Sincronizzazione Alternativa

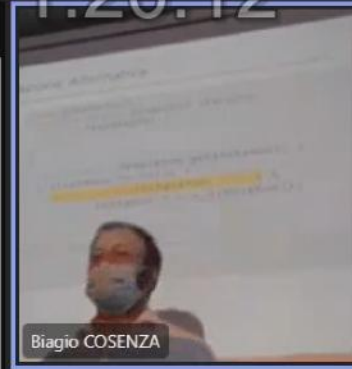
Versione alternativa

- variabile di classe che mantiene l'istanza
- costruttore privato
- metodo non sincronizzato
- se non è stato ancora creata una istanza
 - in maniera sincronizzata (ME)
- istanzia il singleton
- alla fine, restituisci la nuova istanza a disposizione

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

Funziona?

- **ATTENZIONE:** NON FUNZIONA! Perché?



Biagio COSENZA



SALVATORE DANESE



ANTONIO R...

+48



Singleton con Double-checked locking (Interblocco ricontrollato)

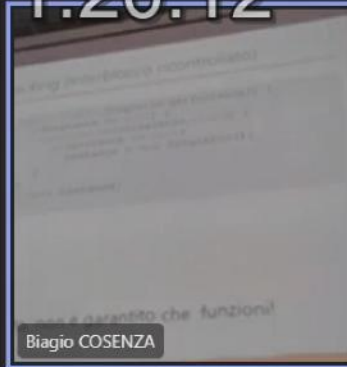
■ Versione con doppio controllo

- restituisce un Singleton
- se non esiste l'istanza
- entra in sezione critica
- controlla che nel waiting non sia cambiata la situazione
- se è il caso crea un nuovo Singleton
- esce dalla sezione critica

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

■ Funziona?

- Nonostante pubblicato e ampiamente utilizzato, non è garantito che funzioni!



SALVATORE DANESE 🗣️



ANTONIO R... 🗣️

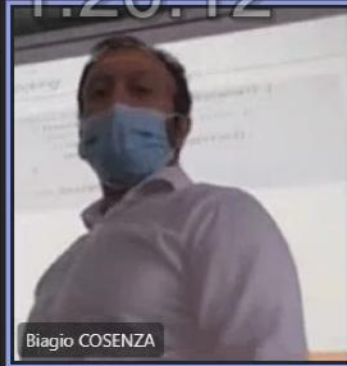
+46



Singleton con Double-checked locking

- Perché non funziona?
 - la chiamata al costruttore "sembra" essere prima dell'assegnazione di instance
 - ma questo non è detto che accada
 - un Singleton non inizializzato potrebbe essere assegnato a instance
 - e se un altro thread controlla il valore ...
 - poi lo può usare (in maniera scorretta)

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



Biagio COSENZA



SALVATORE DANESE



ANTONIO R...

+47

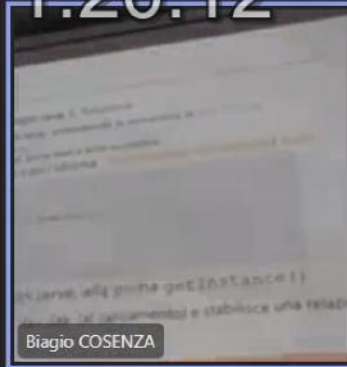


Soluzioni

- La variabile `instance` resa `volatile`, dopo Java 5, funziona
 - la JDK5 ha corretto il modello della memoria di Java, estendendo la semantica di `volatile`:
 - non permette il riordino di read e write precedenti
 - read su volatile non può essere riordinata rispetto a una read o write successiva
- Altrimenti: si possono usare le classi statiche con l'idioma *"Initialization-on-demand holder"*

```
public class Something {  
    private Something() {}  
    private static class LazyHolder {  
        private static final Something INSTANCE = new Something();  
    }  
    public static Something getInstance() {  
        return LazyHolder.INSTANCE;  
    }  
}
```

- LazyHolder è inizializzata dalla JVM solo quando serve, alla prima `getInstance()`
 - essendo un iniziatore statico, viene eseguito una sola volta (al caricamento) e stabilisce una relazione *happens-before* tutte le altre operazioni sulla classe



Biagio COSENZA



SALVATORE DANESE



ANTONIO R...

+47





1:19:24 / 1:20:12

Conclusioni

- Esempi di race condition
- Efficienza dei thread
- Singleton ed il double-checked locking
- Conclusioni



Biagio COSENZA



SALVATORE DANESE



ANTONIO R...

+46

