



Programmazione concorrente & Thread in Java (3)



Corso di Laurea in Informatica, Programmazione Distribuita
Delfina Malandrino, dmalandrino@unisa.it
<http://www.unisa.it/docenti/delfinamalandrino>

1

Organizzazione della lezione

2

- Un esempio
- L'efficienza dei thread
- Il Singleton ed il double-checked locking
- Conclusioni

2

1

Per comprendere la race condition

3

- È necessario comprendere bene in che maniera i metodi synchronized sono eseguiti in mutua esclusione
- Per questo motivo presentiamo un esempio, semplice, che al variare di alcuni semplici keyword si comporta in maniera diversa
- Uno strumento di “lavoro” per fare pratica. . .

Race Condition

Il risultato finale dell'esecuzione dei processi dipende dalla temporizzazione o dalla sequenza con cui vengono eseguiti.



3

Una classe di esempio - 1

4

```
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.");
    }
    //...
}
```

Per usare i logger

4

2

Una classe di esempio - 1

5

```
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.");
    }
}

//...
```

Per usare i logger

Un logger

5

Una classe di esempio - 1

6

```
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.");
    }
}

//...
```

Per usare i logger

Un logger

L'oggetto su cui invocare i metodi

6

Una classe di esempio - 1

7

```
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.");
    }
    //...
}
```

Per usare i logger

Un logger

L'oggetto su cui invocare i metodi

Creazione di due thread ...

7

Una classe di esempio - 1

8

```
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.");
    }
    //...
}
```

Per usare i logger

Un logger

L'oggetto su cui invocare i metodi

Creazione di due thread ...

...e loro lancio

8

Una classe di esempio - 1

9

```
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.");
    }
    //...
}
```

Per usare i logger

Un logger

L'oggetto su cui invocare i metodi

Creazione di due thread ...

... e loro lancio

Stampiamo chi entra

...

9

Una classe di esempio - 1

10

1

```
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.");
    }
    //...
}
```

Per usare i logger

Un logger

L'oggetto su cui invocare i metodi

Creazione di due thread ...

... e loro lancio

Stampiamo chi entra

...

...e attende

... stampa in uscita

10

5

Una classe di esempio - 2

11

```
//...

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.");
}
```

Stampiamo chi entra

...

11

Una classe di esempio - 2

12

```
//...

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.");
}
```

Stampiamo chi entra

...

...e aspetta

12

Una classe di esempio - 2

13

```
//...

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.");
}
}
```

Stampiamo chi entra

...

...e aspetta

Catch della eccezione che può
lanciare sleep se interrotta

... stampa in uscita

13

Il semplice thread

14

```
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);
    }
}
```

Un thread

14

Il semplice thread

15

```
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);
    }
}
```

Un thread

Variabili istanza

15

Il semplice thread

16

```
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);
    }
}
```

Un thread

Variabili istanza

Costruttore

16

Il semplice thread

17

```
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);
    }
}
```

Un thread

Variabili istanza

Costruttore

Metodo da eseguire

17

Il semplice thread

18

```
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);
    }
}
```

Un thread

Variabili istanza

Costruttore

Metodo da eseguire

Un double a caso tra 0 e <1

18

Il semplice thread

19

```
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);
    }
}
```

Un thread

Variabili istanza

Costruttore

Metodo da eseguire

Un double a caso tra 0 e <1

Arrotondamento ad un intero
che è 0 oppure 1

19

Il semplice thread

20

```
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);
    }
}
```

Un thread

Variabili istanza

Costruttore

Metodo da eseguire

Un double a caso tra 0 e <1

Arrotondamento ad un intero
che è 0 oppure 1

A seconda del valore si entra
nella prima ...

20

10

Il semplice thread

21

```
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);
    }
}
```

Un thread

Variabili istanza

Costruttore

Metodo da eseguire

Un double a caso tra 0 e <1

Arrotondamento ad un intero
che è 0 oppure 1

A seconda del valore si entra
nella prima ...

...o nella seconda sala di
attesa

21

Un esempio di lavoro

Vari possibili casi

22

- 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
- I metodi possono essere 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**

22

Esempio: due metodi di istanza sincronizzati

23

```
//...
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.");
}
//
```

Supponiamo che debbano accedere
alla stessa sala d'attesa

23

Esempio: due metodi di istanza sincronizzati

24

```
//...
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.");
}
//
```

Supponiamo che debbano accedere
alla stessa sala d'attesa

Entrambi i metodi sono
sincronizzati

24

Esempio: due metodi di istanza sincronizzati

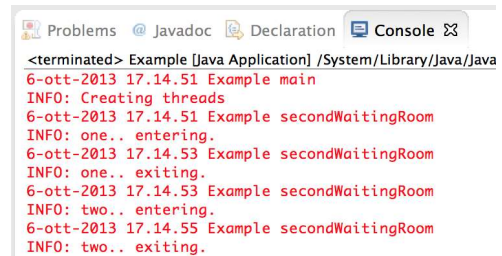
25

```
//...
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.");
}
//
```

Supponiamo che debbano accedere alla stessa sala d'attesa

Entrambi i metodi sono sincronizzati



```
<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.
```

25

Esempio: due metodi di istanza sincronizzati

26

```
//...
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.");
}
//
```

Supponiamo che debbano accedere a due sale di attesa diverse

26

Esempio: due metodi di istanza sincronizzati

27

```
//...
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.");
}
//
```

Supponiamo che debbano accedere a due sale di attesa diverse

Entrambi i metodi sono sincronizzati

27

Esempio: due metodi di istanza sincronizzati

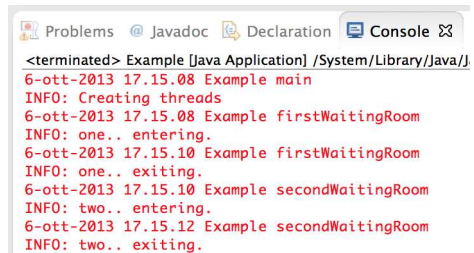
28

```
//...
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.");
}
//
```

Supponiamo che debbano accedere a due sale di attesa diverse

Entrambi i metodi sono sincronizzati



```
<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.
```

28

Esempio: un solo metodo sincronizzato

29

```
//...
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.");
}
//...
```

Primo metodo sincronizzato

29

Esempio: un solo metodo sincronizzato

30

```
//...
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.");
}
//...
```

Primo metodo sincronizzato

Secondo metodo no

30

Esempio: un solo metodo sincronizzato

31

```
//...
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.");
}
//...
```

Primo metodo sincronizzato

Secondo metodo no

Si debba accedere a due sale di attesa diverse

31

Esempio: un solo metodo sincronizzato

32

```
//...
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.");
}

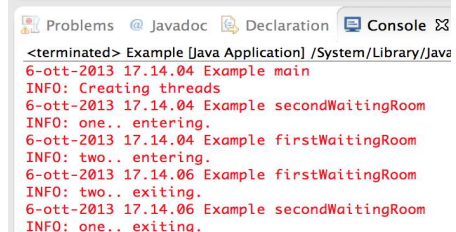
//...
public void secondWaitingRoom(
    int sec, String name) {
    log.info(name+"..entering.");
    try{
        Thread.sleep(sec*1000);
    }catch (InterruptedException e) {
        e.printStackTrace();
    }
    log.info(name+"..exiting.");
}
//...
```

Primo metodo sincronizzato

Secondo metodo no

Si debba accedere a due sale di attesa diverse

No mutua esclusione



```
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.
```

32

Organizzazione della lezione

33

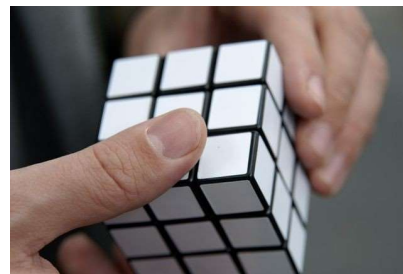
- Un esempio
- L'efficienza dei thread
- Il Singleton ed il double-checked locking
- Conclusioni

33

Inizializziamo un array di interi

34

- Beh, poco da spiegare
- L'array è "grande" (10 milioni di elementi)
- E ogni inizializzazione viene ripetuta 10000 volte!
- Semplice!



34

Una soluzione semplice

35

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 10000000;

    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
```

10 milioni di elementi

35

Una soluzione semplice

36

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 10000000;

    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
```

10 milioni di elementi

Si prende il clock...

36

Una soluzione semplice

37

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 10000000;

    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
```

10 milioni di elementi

Si prende il clock...

...per ogni elemento

37

Una soluzione semplice

38

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 10000000;

    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
```

10 milioni di elementi

Si prende il clock...

...per ogni elemento

...ripetiamo 10000 volte

38

Una soluzione semplice

39

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 10000000;

    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
```

10 milioni di elementi

Si prende il clock...

...per ogni elemento

...ripetiamo 10000 volte

...l'assegnazione

39

Una soluzione semplice

40

```
public class ArrayInit extends Thread{
    public static int[] data;
    public static final int SIZE = 10000000;

    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
```

10 milioni di elementi

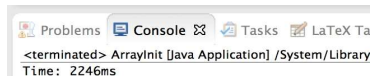
Si prende il clock...

...per ogni elemento

...ripetiamo 10000 volte

...l'assegnazione

Fermiamo il clock



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

40

20

Inizializzazione concorrente

41

- 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, . . . , **MAX_THR**

41

Una soluzione efficiente -1

42

```

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;
            }
        }
        //...
    }
}

```

Un thread

42

21

Una soluzione efficiente -1

43

```
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;
            }
        }
        //...
```

Un thread

Variabili private del thread

43

Una soluzione efficiente -1

44

```
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;
            }
        }
        //...
```

Un thread

Variabili private del thread

Un array di interi

44

Una soluzione efficiente -1

45

```
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;
            }
        }
        //...
```

Un thread

Variabili private del thread

Un array di interi

...graaaande

45

Una soluzione efficiente -1

46

```
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;
            }
        }
        //...
```

Un thread

Variabili private del thread

Un array di interi

...graaaande

8 thread

46

Una soluzione efficiente -1

47

```
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;
            }
        }
        //...
    }
}
```

Un thread

Variabili private del thread

Un array di interi

...graaaande 8

thread

Si dichiara l'array

47

Una soluzione efficiente -1

48

```
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;
            }
        }
        //...
    }
}
```

Un thread

Variabili private del thread

Un array di interi

...graaaande

8 thread

Si dichiara l'array

Un array di thread

48

Una soluzione efficiente -1

49

```
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;
            }
        }
        //...
    }
}
```

Un thread

Variabili private del thread

Un array di interi

...graaaande

8 thread

Si dichiara l'array

Un array di thread

Numero thread totali

49

Una soluzione efficiente -1

50

```
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;
            }
        }
        //...
    }
}
```

Un thread

Variabili private del thread

Un array di interi

...graaaande 8 thread

Si dichiara l'array

Un array di thread

Numero thread totali

Si prende il clock

50

Una soluzione efficiente - 2

51

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s) :
        " + (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

51

Una soluzione efficiente - 2

52

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s) :
        " + (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

52

Una soluzione efficiente - 2

53

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s) :
        " + (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

53

Una soluzione efficiente - 2

54

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s) :
        " + (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

...si crea

54

Una soluzione efficiente - 2

55

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s):
        "+ (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

...si crea

...si lancia

55

Una soluzione efficiente - 2

56

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s):
        "+ (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

...si crea

...si lancia

Si incrementa l'inizio del prossimo blocco

56

Una soluzione efficiente - 2

57

```
//...
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;
    }//end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        }catch(InterruptedException e) {
            e.printStackTrace();
        }
    }//end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s) :
        "+ (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

...si crea

...si lancia

Si incrementa l'inizio del prossimo blocco

Il main, per ogni thread,

57

Una soluzione efficiente - 2

58

```
//...
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;
    }//end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        }catch(InterruptedException e) {
            e.printStackTrace();
        }
    }//end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s) :
        "+ (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

...si crea

...si lancia

Si incrementa l'inizio del prossimo blocco

Il main, per ogni thread,

...aspetta che termini

58

Una soluzione efficiente - 2

59

```
//...
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;
    } //end for
    for(j = 0; j < numThread; j++){
        try{
            threads[j].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //end for
    end = System.currentTimeMillis();
    System.out.println(numThread + "Thread(s):
        "+ (end - begin) + "ms");
}
} //end main
//...
```

Prima posizione

Array di thread

Per ogni thread...

... si crea

... si lancia

Si incrementa l'inizio del prossimo blocco

Il main, per ogni thread,

... aspetta che termini

Si prende il clock

59

Una soluzione efficiente - 3

60

```
//...
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
```

Costruttore con due parametri

60

30

Una soluzione efficiente - 3

61

```
//...
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
```

Costruttore con due parametri

Inizio ...

61

Una soluzione efficiente - 3

62

```
//...
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
```

Costruttore con due parametri

Inizio ...

...e dimensione del blocco

62

Una soluzione efficiente - 3

63

```
//...
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
```

Costruttore con due parametri

Inizio ...

...e dimensione del blocco

Esecuzione del thread

63

Una soluzione efficiente - 3

64

```
//...
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
```

Costruttore con due parametri

Inizio ...

...e dimensione del blocco

Esecuzione del thread

Per tutti gli elementi del blocco

64

Una soluzione efficiente - 3

65

```
//...
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
```

Costruttore con due parametri

Inizio ...

...e dimensione del blocco

Esecuzione del thread

Per tutti gli elementi del blocco

inizializzato

65

Una soluzione efficiente - 3

66

```
//...
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
```

Costruttore con due parametri

Inizio ...

...e dimensione del blocco

Esecuzione del thread

Per tutti gli elementi del blocco

inizializzato

Ripetendolo 10000 volte!

66

67

OK, ABBIAMO FINITO. ...

67

68

OK, ABBIAMO FINITO. ...

...

Ok, dimenticavo l'efficienza

68

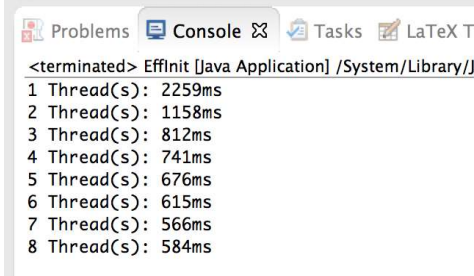
Inizializzazione concorrente

69

OK, ABBIAMO FINITO. ...

...

Ok, dimenticavo l'efficienza
Una esecuzione su una macchina con 8core (4 con
hyperthreading)



```

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
  
```

Alcuni commenti:

Miglioramenti non lineari nel numero di core (overhead) **Attenzione! La situazione è molto complessa (I/O bound, CPU bound, core reali, buffers) ⇒ "your mileage may vary"**

69

Organizzazione della lezione

70

- ☐ Un esempio
- ☐ L'efficienza dei thread
- ☒ Il Singleton ed il double-checked locking
- ☐ Conclusioni

70

Il Singleton

71

- 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: la allocazione avviene solo quando utilizzato per la prima volta



71

Singleton in Java - 1

72

```

class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

72

Singleton in Java - 1

73

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

variabile di classe che mantiene
l'istanza

costruttore privato

73

Singleton in Java - 1

74

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

variabile di classe che mantiene
l'istanza

costruttore privato

metodo di classe che restituisce un
Singleton

74

Singleton in Java - 1

75

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo di classe che restituisce un Singleton

se non è stato ancora creata una istanza

75

Singleton in Java - 1

76

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

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

76

Singleton in Java - 1

77

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

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

77

Singleton in Java - 1

78

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

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

E se ci sono più thread?

Il loro interleaving può creare errori: "più" singleton



78

Singleton in Java - 2

79

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

variabile di classe che mantiene
l'istanza

79

Singleton in Java - 2

80

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

variabile di classe che mantiene
l'istanza

costruttore privato

80

Singleton in Java - 2

81

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene
l'istanza

costruttore privato

metodo **sincronizzato**

81

Singleton in Java - 2

82

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene
l'istanza

costruttore privato

metodo **sincronizzato**

se non è stato ancora creata una
istanza

82

Singleton in Java - 2

83

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo **sincronizzato**

se non è stato ancora creata una istanza

... allora creala

83

Singleton in Java - 2

84

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo **sincronizzato**

se non è stato ancora creata una istanza

... allora creala

alla fine, restituisci la nuova istanza a disposizione

84

Singleton in Java - 2

85

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo **sincronizzato**

se non è stato ancora creata una istanza

... allora creala

alla fine, restituisci la nuova istanza a disposizione

Problemi?

85

Singleton in Java - 2

86

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

L'efficienza!

Le performance dell'applicazione potrebbero risentirne



variabile di classe che mantiene l'istanza

costruttore privato

metodo **sincronizzato**

se non è stato ancora creata una istanza

... allora creala

alla fine, restituisci la nuova istanza a disposizione

Problemi?

86

Singleton in Java - 3

87

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

variabile di classe che mantiene
l'istanza

87

Singleton in Java - 3

88

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

variabile di classe che mantiene
l'istanza

costruttore privato

88

Singleton in Java - 3

89

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo non sincronizzato

89

Singleton in Java - 3

90

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo non sincronizzato

se non è stato ancora creata una istanza

90

Singleton in Java - 3

91

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

variabile di classe che mantiene l'istanza

costruttore privato

metodo non sincronizzato

se non è stato ancora creata una istanza

in maniera sincronizzata (ME)

91

Singleton in Java - 3

92

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

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

92

Singleton in Java - 3

93

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

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

93

Singleton in Java - 3

94

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

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

Non funziona!



94

Perché non funziona...

95

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad eseguire “insieme” qui

95

Perché non funziona...

96

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad eseguire “insieme” qui

A entra in sezione critica

96

Perché non funziona...

97

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B*
arrivano ad eseguire “insieme” qui
A entra in sezione critica

crea una istanza

97

Perché non funziona...

98

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad
eseguire “insieme” qui

A entra in sezione critica

crea una istanza

esce dalla sezione critica

98

Perché non funziona...

99

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad eseguire “insieme” qui

A entra in sezione critica

crea una istanza

esce dalla sezione critica

e restituisce il “primo” singleton

99

Perché non funziona...

100

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad eseguire “insieme” qui

A entra in sezione critica

crea una istanza

esce dalla sezione critica

e restituisce il “primo” singleton

ma a questo punto *B* entra e crea un “nuovo” singleton

100

50

Perché non funziona...

101

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad eseguire “insieme” qui

A entra in sezione critica

crea una istanza

esce dalla sezione critica

e restituisce il “primo” singleton

ma a questo punto *B*

entra e crea un “nuovo” singleton

che sovrascrive il primo!

101

Perché non funziona...

102

```
class Singleton {
    private static Singleton instance;
    private Singleton() {
        //...
    }

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

se due thread *A* e *B* arrivano ad eseguire “insieme” qui

A entra in sezione critica

crea una istanza

esce dalla sezione critica

e restituisce il “primo” singleton

ma a questo punto *B*

entra e crea un “nuovo” singleton

che sovrascrive il primo!



102

Double-checked locking

103

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

restituisce un Singleton

103

Double-checked locking

104

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

restituisce un Singleton

se non esiste l'istanza

...

104

Double-checked locking

105

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

restituisce un Singleton
se non esiste l'istanza
...
si entra in CS

105

Double-checked locking

106

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

restituisce un Singleton
se non esiste l'istanza
...
si entra in CS
si controlla che nel *waiting* non
sia cambiata la situazione

106

Double-checked locking

107

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

restituisce un Singleton

se non esiste l'istanza

...

si entra in CS

si controlla che nel waiting non sia cambiata la situazione

se è il caso crea un nuovo Singleton

107

Double-checked locking

108

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

restituisce un Singleton

se non esiste l'istanza

...

si entra in CS

si controlla che nel waiting non sia cambiata la situazione

se è il caso crea un nuovo Singleton

esci dalla CS

108

Double-checked locking

109

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

Sorpresa!

Nonostante pubblicato e ampiamente utilizzato, non è garantito che funzioni.

restituisce un Singleton

se non esiste l'istanza

...

si entra in CS

si controlla che nel waiting non sia cambiata la situazione

se è il caso crea un nuovo Singleton

esci dalla CS

109

Perché non funziona

110

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

La chiamata al costruttore “sembra” essere prima dell’assegnazione di instance

110

Perché non funziona

111

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

La chiamata al costruttore “sembra” essere prima dell’assegnazione di instance

ma questo non è detto che accada

111

Perché non funziona

112

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

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

112

Perché non funziona

113

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

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 ...

113

Perché non funziona

114

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

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)

114

Perché non funziona

115

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



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)

115

Soluzioni?

116

- La variabile instance resa volatile, dopo Java 5, funziona
- 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;
    }
}
```

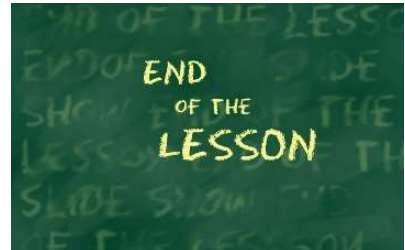
- LazyHolder è inizializzata dalla JVM solo quando serve (alla prima getInstance())
- Essendo un iniziatore statico, viene eseguito una sola volta (al caricamento) e stabilisce una relazione happens-before tutte le altre operazioni sulla classe

116

Conclusioni

117

- Un esempio
- L'efficienza dei thread
- Il Singleton ed il double-checked locking
- Conclusioni



Nelle prossime lezioni:

Java Method Invocation (RMI)

117