Corso di Laurea in Informatica
I Anno Magistrale, Indirizzo Cloud Computing
Reti Geografiche: Struttura, Analisi e Prestazioni

Programmazione concorrente & Thread in Java

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

1

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

2

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

3

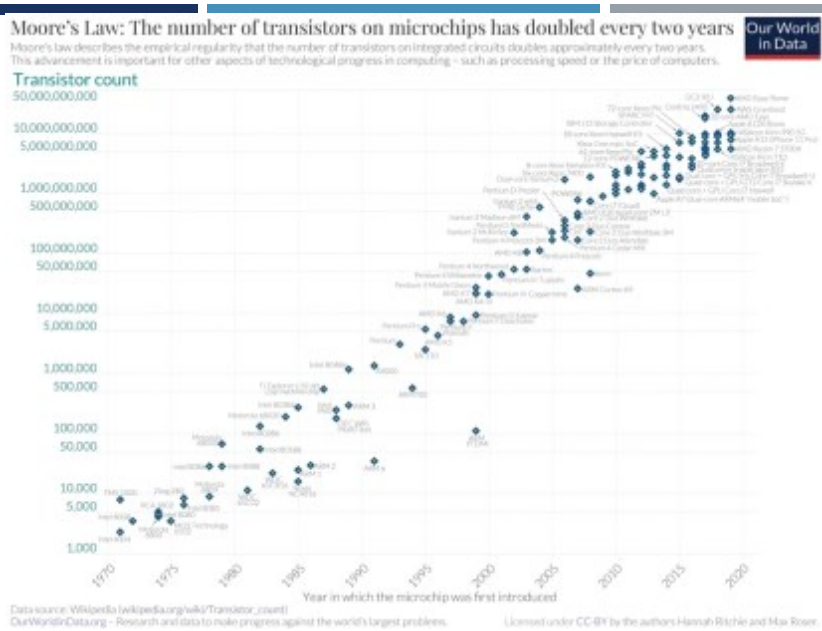
Legge di Moore

- Una delle leggi più citate dell'informatica:
- Il motore della crescita del nostro campo:
  - il nostro desktop costa poche centinaia di euro, ed è potente quanto calcolatori che ne costavano milioni una decina di anni fa

La legge di Moore

Il numero di transistor per chip raddoppia ogni 18 mesi

4



Source: https://ourworldindata.org/moores-law

5

The free performance lunch

- Non importa quanto veloci diventeranno i processori, i software troveranno nuovi modi di ***«mangiare»*** questa extra speed
- Il clock speed non è l'unica misura di performance ma è sicuramente una misura istruttiva
- Fino a quando continuerà questa crescita esponenziale?

Clock speed is the number of times a second that a circuit operates and is most associated with the central processing unit (CPU). It is measured in hertz, or cycles per second. The higher the clock speed, the more processing power -- all other things being equal. Clock speed is also known as *clock rate*, *core clock* or *clock frequency*.

6

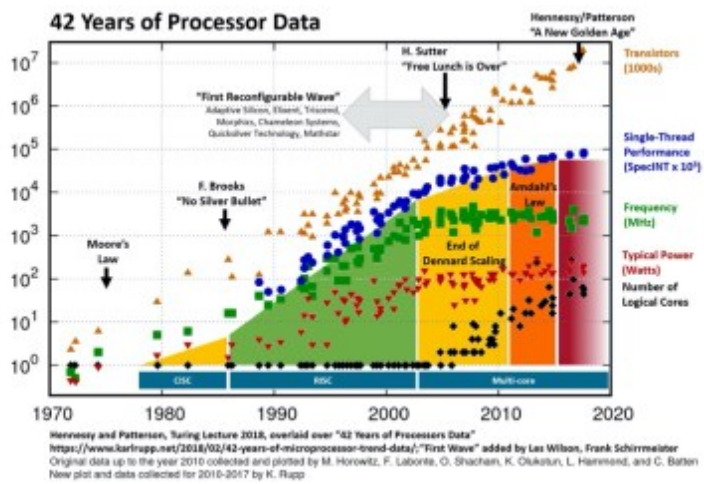
The free performance lunch

- L'obiettivo, comune a tutti i produttori, è chiaro:
  - ridurre lo spessore del package
  - incrementare le prestazioni
  - migliorare le caratteristiche termiche e le capacità di connessione (verso sensori o attuatori) dei chip

7

Tutto cresce (O No?)

- L'unico limite che si contrappone all'intuizione di Moore è l'impossibilità fisica di creare processori sempre più piccoli



8

Tutto cresce (O No?)

I minuscoli transistor dei circuiti integrati, infatti, non possono essere miniaturizzati all'infinito

- La corsa alla miniaturizzazione dei processori si scontra con i limiti della fisica
  - Questo limite è dato dall'incapacità di andare al di sotto dei 5 nanometri, corrispondenti alla lunghezza d'onda degli elettroni. Infatti, non è possibile fisicamente costruite il gate di un transistor in silicio che sia più piccolo di 5 nanometri. Gli elettroni che passano dal source al drain sono controllati dal gate, che si attiva e disattiva come un interruttore quando viene applicata una tensione esterna

… Da un lato i minuscoli transistor dei circuiti integrati non possono essere miniaturizzati all'infinito

… Dall'altro devono contenere al loro interno una carica elettrica, cioè un certo numero di elettroni, che come particelle occupano anch'esse dello spazio

9

Tutto cresce (O No?)

- Effetto delle termodinamica che disturba la crescita secondo la legge di Moore del numero di transistor

- Inizia ad avere effetto con la tecnologia al di sotto di 40 nm:
  - ci siamo: i transistor più moderni Core i7 e i9 hanno raggiunto dimensioni di 14 e 10 nanometri, rispettivamente

10

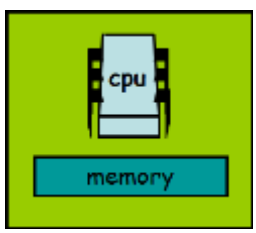
Tutto cresce (O No?)

In pratica: non si possono avere "tanti" transistor su un processore, che siano anche "facili da raffreddare" e che siano "veloci": si deve rinunciare ad una di queste caratteristiche

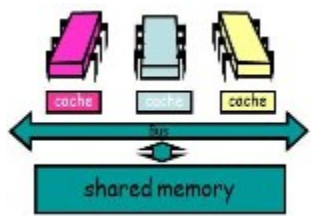
11

Il problema: thermal noise

Dal desktop con singolo processore . . .



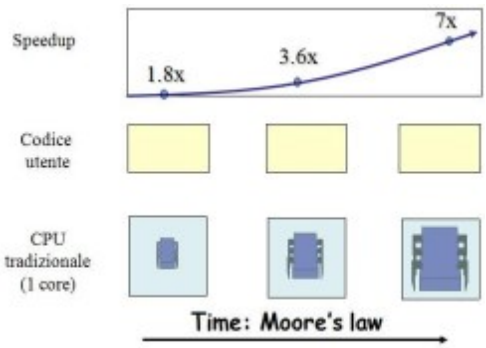
al desktop con più processori



12

Il problema: thermal noise

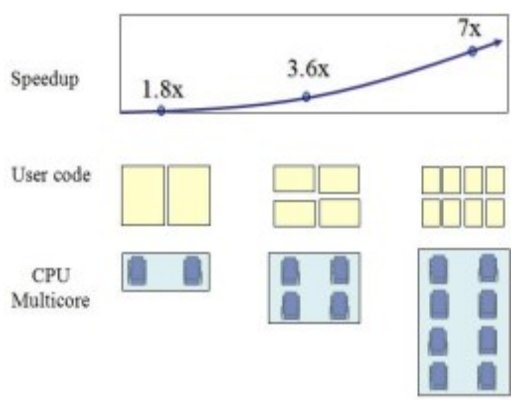
- Quello che accadeva con il singolo processore (core)



13

Il problema: thermal noise

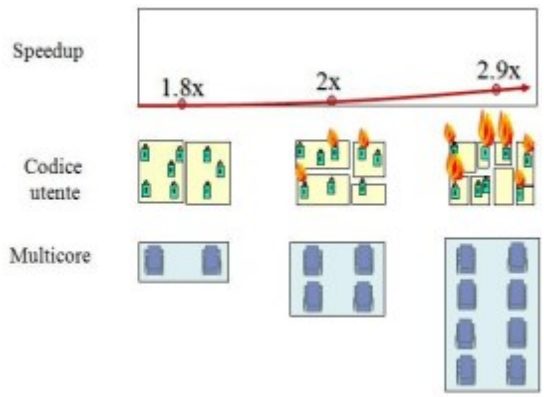
- Cosa ci piacerebbe che fosse vero anche per i multicore



14

Il problema: thermal noise

- La triste realtà: bilanciare carico è difficile e si creano hot-spots



15

"Free lunch is over"

- Fino a poco tempo fa, i miglioramenti della tecnologia comportavano un automatico miglioramento delle prestazioni software:
  - *CPU con clock maggiore eseguivano il codice con più velocità*
- Adesso: il miglioramento consiste in più transistor, ma organizzati in core multipli . . .
- . . . che per essere usati efficacemente hanno bisogno di software in grado di sfruttare il parallelismo delle applicazioni

16

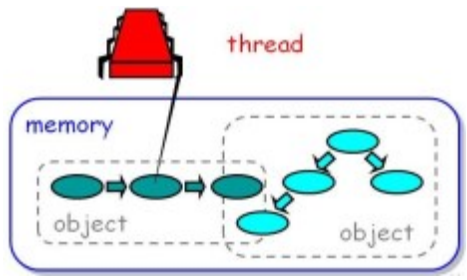
Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

17

L'accesso in memoria con single core

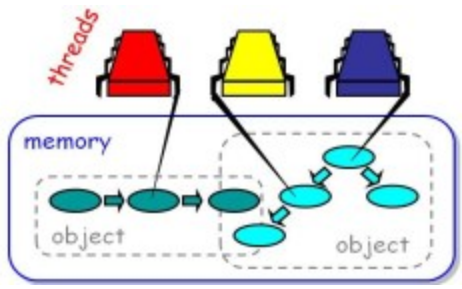
- Un singolo thread accede alla memoria (strutturata in oggetti)



18

L'accesso in memoria con multi core

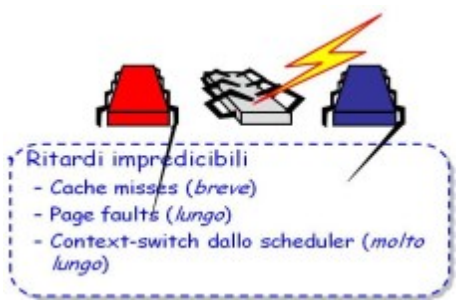
- La situazione si complica notevolmente . . .



19

L'accesso in memoria con multi core

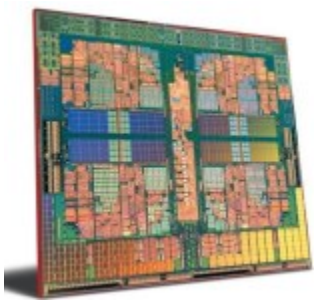
- Tenendo presente anche i problemi di asincronia!



20

Cosa ci riserva il futuro

- Il trend del presente/futuro (10 anni) è chiaramente in direzione multi-core . . .
- Siano essi omogenei (Intel, Sun) o eterogenei (AMD) o una "via di mezzo" (IBM) saranno numerosi



- Non è difficile immaginare migliaia di core sui server
  - e centinaia sui desktop
- Si ipotizza un corollario alla Legge di Moore: i core raddoppieranno ogni 18 mesi

21

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

22

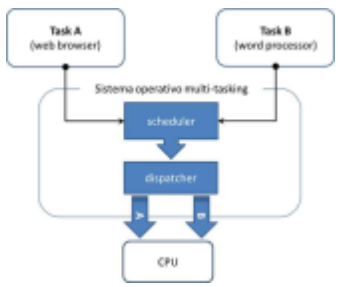
Programmazione distribuita e concorrente

- La programmazione distribuita implica la conoscenza (di base) della programmazione concorrente:
  - che coinvolge diversi processi che vengono eseguiti insieme
- Tre tipi di programmazione concorrente
  - programmazione **concorrente eseguita su calcolatori diversi**
  - processi concorrenti **sulla stessa macchina (multitasking)**
    - processo padre che genera processi figli per fork()
  - programmazione **concorrente nello stesso processo**
    - "processi lightweight" all'interno del processo: thread

23

Multitasking e multithread

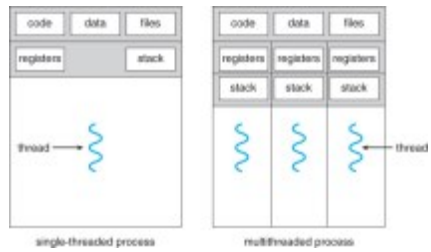
- S. O. Multitask: creano l'illusione (per l'utente) di una macchina completamente dedicata
  - ma durante l'interazione dell'utente con il proprio programma, il S. O. ha il tempo di eseguire più task contemporaneamente



24

Multitasking e multithread

- Il multithread è l'estensione del multitask riferita ad un singolo programma
  - in grado di eseguire più thread "contemporaneamente
- Thread: anche detti processi "light-weight"
  - **a differenza dei processi hanno a disposizione e condividono gli stessi dati (trovandosi all'interno dello stesso processo)**
- Meccanismo di comunicazione attraverso memoria condivisa:
  - strumento efficace per costruire programmi che necessitano di svolgere "in parallelo" più compiti, ma fonte di possibili problemi!

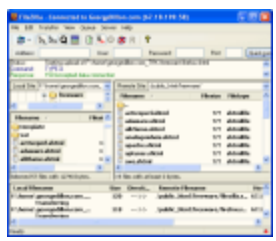


25

Tipiche applicazioni multithread - 1

- Un browser che, allo stesso tempo, deve poter
  - caricare dal server diverse immagini che sono nella stessa pagina
  - visualizzare la pagina così come arriva
  - reagire all'eventuale pulsante di stop premuto dall'utente

- Una applicazione di rete che, allo stesso tempo, deve
  - chiedere dati ad una altra applicazione
  - fornire dati a chi li richiede
  - tenere informato l'utente dell'andamento delle operazioni



26

Tipiche applicazioni multithread - 2

- Server che hanno bisogno di istanziare velocemente oggetti:
  - vengono istanziati tutti insieme come un pool di oggetti Thread
  - tenuti in stato “sospeso”
  - e riportati (velocemente) alla “vita” quando necessario

- Streaming audio application che deve poter:
  - leggere l’audio dalla rete
  - decomprimerlo
  - gestire l’output
  - aggiornare il display

27

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

28

Processi e thread

- Processo: ambiente di esecuzione con uno spazio di memoria privato
- La cooperazione tra processi avviene attraverso *InterProcess Communication* come pipe e socket

29

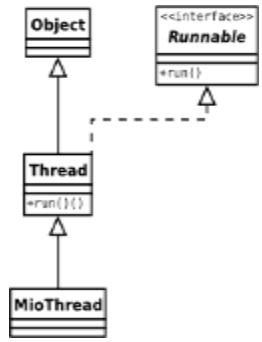
Processi e thread

- Thread ("*lightweight process*") esistono all'interno di un processo, condividendo tra loro memoria e file aperti
- In Java ogni applicazione ha almeno un thread utente ("*main thread*"), più alcuni thread di sistema che gestiscono la memoria e i segnali
- Il main thread può creare e far partire diversi altri thread

30

Processi e thread

- I thread in Java sono oggetti, istanze quindi di una **classe Thread**
- L'evoluzione di Java ha portato a due modalità di gestione dei thread
  - istanziare un oggetto **thread** ogni volta che serve un task asincrono (creazione e gestione a cura del programmatore)
  - astrarre la gestione, passando un task ad un executor
- Noi ci focalizziamo sulla prima modalità, di base



31

Usare i thread in Java

- Passi principali per scrivere un thread:
  1. Estendere la classe **java.lang.Thread**
  2. Riscrivere (ridefinire, override) il metodo **run()** nella sottoclasse di **Thread**
  3. Creare un'istanza di questa classe derivata
  4. Richiamare il metodo **start()** su questa istanza

32

Due modi per creare e lanciare i thread -I

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

Si deriva una classe da **`Thread`**

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

33

Due modi per creare e lanciare i thread -I

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

Si deriva una classe da **`Thread`**

Metodo che viene eseguito quando si lancia il thread

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

34

Due modi per creare e lanciare i thread -I

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si deriva una classe da **`Thread`**

Metodo che viene eseguito quando si lancia il thread

Metodo main

35

Due modi per creare e lanciare i thread -I

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si deriva una classe da **`Thread`**

Metodo che viene eseguito quando si lancia il thread

Metodo main

Si istanzia e si lancia con **`start()`**

36

Due modi per creare e lanciare i thread -I

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si deriva una classe da **`Thread`**

Metodo che viene eseguito quando si lancia il thread

Metodo main

Si istanzia e si lancia con **`start()`**

Semplice da realizzare ma con qualche limitazione

37

Due modi per creare e lanciare i thread -I

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si deriva una classe da **`Thread`**

Metodo che viene eseguito quando si lancia il thread

Metodo main

Si istanzia e si lancia con **`start()`**

Semplice da realizzare ma con qualche Limitazione

Se **`HelloThread`** deve estendere un'altra classe?

38

Due modi per creare e lanciare i thread - 2

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si implementa una interfaccia

39

Due modi per creare e lanciare i thread - 2

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si implementa una interfaccia

Metodo che viene eseguito quando si lancia il thread

40

Due modi per creare e lanciare i thread - 2

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si implementa una interfaccia

Metodo che viene eseguito quando si lancia il thread

Metodo main

41

Due modi per creare e lanciare i thread - 2

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si implementa una interfaccia

Metodo che viene eseguito quando si lancia il thread

Metodo main

L'oggetto istanziato è passato al costruttore di **`Thread`** e lanciato

42

Due modi per creare e lanciare i thread - 2

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a
        thread!");
    }

    public static void main(String args[])
        {  (new HelloThread()).start();
    }

}
```

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Si implementa una interfaccia

Metodo che viene eseguito quando si lancia il thread

Metodo main

L'oggetto istanziato è passato al costruttore di **`Thread`** e lanciato

Più generale utilizzo

43

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

44

Il metodo sleep()

```
public class SleepMessages {
  public static void main(String args[])
                    throws InterruptedException {

    String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
    };

    for(int i=0; i<importantInfo.length; i++) {

        Thread.sleep(4000);

        System.out.println(importantInfo[i]);

     }
  }
}
```

Stampa 4 stringhe con un ritardo

45

Il metodo sleep()

```
public class SleepMessages {
  public static void main(String args[])
                    throws InterruptedException {

    String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
    };

    for(int i=0; i<importantInfo.length; i++) {

        Thread.sleep(4000);

        System.out.println(importantInfo[i]);

     }
  }
}
```

Stampa 4 stringhe con un ritardo

Array stringhe

46

Il metodo sleep()

```
public class SleepMessages {
  public static void main(String args[])
                    throws InterruptedException {

    String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
    };

    for(int i=0; i<importantInfo.length; i++) {

        Thread.sleep(4000);

        System.out.println(importantInfo[i]);

     }
  }
}
```

Stampa 4 stringhe con un ritardo

Array stringhe

Per ogni stringa

47

Il metodo sleep()

```
public class SleepMessages {
  public static void main(String args[])
                    throws InterruptedException {

    String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
    };

    for(int i=0; i<importantInfo.length; i++) {

        Thread.sleep(4000);

        System.out.println(importantInfo[i]);

     }
  }
}
```

Stampa 4 stringhe con un ritardo

Array stringhe

Per ogni stringa

Sospendo il thread per 4 secondi

48

Il metodo sleep()

```
public class SleepMessages {
  public static void main(String args[])
                     throws InterruptedException {

    String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
    };

    for(int i=0; i<importantInfo.length; i++) {

        Thread.sleep(4000);

        System.out.println(importantInfo[i]);


     }
  }
}
```

Stampa 4 stringhe con un ritardo

Array stringhe

Per ogni stringa

Sospendo il thread per 4 secondi

Stampo

49

Il metodo sleep()

```
public class SleepMessages {
  public static void main(String args[])
                     throws InterruptedException {

    String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
    };

    for(int i=0; i<importantInfo.length; i++) {

        Thread.sleep(4000);

        System.out.println(importantInfo[i]);


     }
  }
}
```

Stampa 4 stringhe con un ritardo

Array stringhe

Per ogni stringa

Sospendo il thread per 4 secondi

Stampo

Eccezione lanciata dal thread corrente se interrotto mentre in sleep()

50

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

51

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro
- **Il programmatore decide cosa fare**

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

52

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro
- Il programmatore decide cosa fare
- **Nell'esempio precedente**

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

53

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro
- Il programmatore decide cosa fare
- Nell'esempio precedente

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

Nella sleep()

54

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro
- Il programmatore decide cosa fare
- Nell'esempio precedente

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

Nella sleep()

Si intercetta l'eccezione

55

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro
- Il programmatore decide cosa fare
- Nell'esempio precedente

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

Nella sleep()

Si intercetta l'eccezione

In caso si esce

56

Gli Interrupt - I

- Un interrupt è una indicazione che un thread dovrebbe fermare quello che sta facendo e fare qualcos'altro
- Il programmatore decide cosa fare
- Nell'esempio precedente

```
//...

for(int i = 0; i < importantInfo.length; i++)
  {  try{
       Thread.sleep(4000);
  }catch(InterruptedException e) {
       return;
  }
  System.out.println(importantInfo[i]);
}

//...
```

Nella sleep()

SI intercetta l'eccezione

In caso si esce

Altrimenti (a fine **sleep()**) si stampa

57

Il metodo join()

- A volte è necessario che un thread attenda il completamento di un altro thread
- Se **t** è un oggetto il cui thread è in esecuzione, allora:

```
//...
       t.join()
//...
```

- Mette il thread corrente in pausa fino a quando thread **t** non termina
- Possibile anche specificare un periodo di attesa come parametro
- **`join()`** risponde ad un interrupt generando **`InterruptedException`**

58

Un esempio: **SimpleThreads**

- Due thread
- Il primo thread è il main thread di un programma Java
  - . . . che crea un nuovo thread, da un oggetto MessageLoop
  - . . . aspetta il suo termine
- Se ci mette troppo, il main thread lo interrompe con il metodo interrupt()
  - . . . ed attende che termini

59

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
         "Mares eat oats",
         "Does eat oats",
         "Little lambs eat ivy",
         "A kid will eat ivy too"};
   try{
    for(int i = 0; i < impinf.length; i++) {
     Thread.sleep(4000);
     threadMessage(impinf[i]);
     }//end for
    }catch(InterruptedException e) {
       threadMessage("I wasn't done!");
   }//end catch
  }//end run()
 }//end class MessageLoop
 //...
```

Mostra un messaggio con il nome del thread

60

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
        "Mares eat oats",
        "Does eat oats",
        "Little lambs eat ivy",
        "A kid will eat ivy too"};
   try{
    for(int i = 0; i < impinf.length; i++) {
     Thread.sleep(4000);
     threadMessage(impinf[i]);
     }//end for
    }catch(InterruptedException e) {
       threadMessage("I wasn't done!");
   }//end catch
  }//end run()
 }//end class MessageLoop
 //...
```

Mostra un messaggio con il nome del thread

Stampa formattata

61

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
        "Mares eat oats",
        "Does eat oats",
        "Little lambs eat ivy",
        "A kid will eat ivy too"};
   try{
    for(int i = 0; i < impinf.length; i++) {
     Thread.sleep(4000);
     threadMessage(impinf[i]);
     }//end for
    }catch(InterruptedException e) {
       threadMessage("I wasn't done!");
   }//end catch
  }//end run()
 }//end class MessageLoop
 //...
```

Mostra un messaggio con il nome del thread

Stampa formattata

Interfaccia

62

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
      String impinf[] = {
      "Mares eat oats",
      "Does eat oats",
      "Little lambs eat ivy",
      "A kid will eat ivy too"};
 try{
  for(int i = 0; i < impinf.length; i++) {
   Thread.sleep(4000);
   threadMessage(impinf[i]);
   }//end for
  }catch(InterruptedException e) {
     threadMessage("I wasn't done!");
 }//end catch
}//end run()
}//end class MessageLoop
//...
```

Mostra un messaggio con il nome del thread

Stampa formattata

Interfaccia

Metodo che viene eseguito allo start del thread

63

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
      "Mares eat oats",
      "Does eat oats",
      "Little lambs eat ivy",
      "A kid will eat ivy too"};
    try{
     for(int i = 0; i < impinf.length; i++) {
      Thread.sleep(4000);
      threadMessage(impinf[i]);
      }//end for

    }catch(InterruptedException e) {
       threadMessage("I wasn't done!");
   }//end catch
  }//end run()
 }//end class MessageLoop
 //...
```

Mostra un messaggio con il nome del thread

Stampa formattata

Interfaccia

Metodo che viene eseguito allo start del thread

4 stringhe

64

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
       "Mares eat oats",
       "Does eat oats",
       "Little lambs eat ivy",
       "A kid will eat ivy too"};
    try{
     for(int i = 0; i < impinf.length; i++) {
      Thread.sleep(4000);
      threadMessage(impinf[i]);
      }//end for

     }catch(InterruptedException e) {
        threadMessage("I wasn't done!");
    }//end catch
   }//end run()
 }//end class MessageLoop
 //...
```

Mostra un messaggio con il nome del thread

Stampa formattata

Interfaccia

Metodo che viene eseguito allo start del thread

4 stringhe

blocco try...catch

65

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
       "Mares eat oats",
       "Does eat oats",
       "Little lambs eat ivy",
       "A kid will eat ivy too";
    try{
     for(int i = 0; i < impinf.length; i++) {
      Thread.sleep(4000);
      threadMessage(impinf[i]);
      }//end for
     }catch(InterruptedException e) {
        threadMessage("I wasn't done!");
    }//end catch
   }//end run()
 }//end class MessageLoop
 //...
```

Mostra un messaggio con il nome del thread

Stampa formattata

Interfaccia

Metodo che viene eseguito allo start del thread

4 stringhe

blocco try...catch

pausa di 4 secondi

66

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
       "Mares eat oats",
       "Does eat oats",
       "Little lambs eat ivy",
       "A kid will eat ivy too"};

   try{
    for(int i = 0; i < impinf.length; i++) {
     Thread.sleep(4000);
     threadMessage(impinf[i]);
     }//end for
    }catch(InterruptedException e) {
       threadMessage("I wasn't done!");
   }//end catch
  }//end run()
 }//end class MessageLoop
 //...
```

- Mostra un messaggio con il nome del thread
- Stampa formattata
- Interfaccia
- Metodo che viene eseguito allo start del thread
- 4 stringhe
- blocco try...catch
- pausa di 4 secondi
- poi stampa

67

Un esempio: SimpleThreads

```
public class SimpleThreads {
 static void threadMessage(String msg) {
   String tn = Thread.currentThread().getName();
   System.out.format("%s:%s%n", tn, msg);
 }

 private static class MessageLoop
                          implements Runnable {
  public void run() {
   String impinf[] = {
       "Mares eat oats",
       "Does eat oats",
       "Little lambs eat ivy",
       "A kid will eat ivy too"};

   try{
    for(int i = 0; i < impinf.length; i++) {
     Thread.sleep(4000);
     threadMessage(impinf[i]);
     }//end for
    }catch(InterruptedException e) {
       threadMessage("I wasn't done!");
   }//end catch
  }//end run()
 }//end class MessageLoop
 //...
```

- Mostra un messaggio con il nome del thread
- Stampa formattata
- Interfaccia
- Metodo che viene eseguito allo start del thread
- 4 stringhe
- blocco try...catch
- pausa di 4 secondi
- poi stampa
- se interrotta

68

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

69

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

Ritardo in millisecondi (default 1 ora)

70

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

Ritardo in millisecondi (default 1 ora)

Se c'è un argomento, è in secondi

71

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

Ritardo in millisecondi (default 1 ora)

Se c'è un argomento, è in secondi

Controllo formato

72

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

Ritardo in millisecondi (default 1 ora)

Se c'è un argomento, è in secondi

Controllo formato

Prende il tempo di inizio

73

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

Ritardo in millisecondi (default 1 ora)

Se c'è un argomento, è in secondi

Controllo formato

Crea un oggetto **`Thread`** da **`MessageLoop`**

74

Un esempio: SimpleThreads

```
//...
public static void main(String args[])
                    throws InterruptedException {
 long patience = 1000 * 60 * 60;

 if(args.length > 0) {
   try{
    patience = Long.parseLong(args[0]) * 1000;
   }catch(NumberFormatException e) {
   System.err.println("Argument must be an
         integer.");
   System.exit(1);
  }
 }

 threadMessage("Starting MessageLoop thread");
 long startTime = System.currentTimeMillis();
 Thread t = new Thread(new MessageLoop());
 t.start();

 //...
```

Lancia eccezione

Ritardo in millisecondi (default 1 ora)

Se c'è un argomento, è in secondi

Controllo formato

Crea un oggetto **`Thread`** da **`MessageLoop`**

E lo fa partire

75

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita ...

76

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita …

…aspetta 1 secondo al più

77

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita …

… aspetta 1 secondo al più

Se la pazienza è scaduta …

78

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita ...

... aspetta 1 secondo al più

Se la pazienza è scaduta ...

... e il thread è ancora vivo

79

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita ...

... aspetta 1 secondo al più

Se la pazienza è scaduta ...

... e il thread è ancora vivo

Lo si chiude

80

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita ...

... aspetta 1 secondo al più

Se la pazienza è scaduta ...

... e il thread è ancora vivo

Lo si chiude

E se ne attende la fine

81

Un esempio: SimpleThreads

```
//...  threadMessage("Waiting for
 MessageLoop to finish");

 while(t.isAlive()) {
  threadMessage("Still waiting...");

  t.join(1000);

  if(((System.currentTimeMillis() - startTime) >
        patience)
       && t.isAlive()) {

   threadMessage("Tired of waiting!");
   t.interrupt();

   t.join();
  }
 }
 threadMessage("Finally!");
}
}
```

Mentre `MessageLoop` è in vita ...

... aspetta 1 secondo al più

Se la pazienza è scaduta ...

... e il thread è ancora vivo

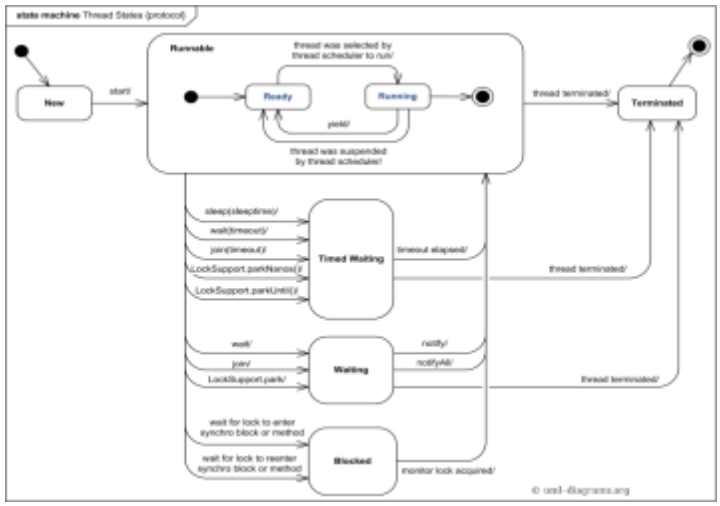
Lo si chiude

E se ne attende la fine

E si esce!

82

Ciclo di vita



83

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

84

Comunicazione fra thread

- I thread comunicano principalmente condividendo accesso a:
  - campi (tipi primitivi)
  - campi che contengono riferimenti a oggetti
- Comunicazione molto efficiente (rispetto a usare la rete)
- Possibili due tipi di errori:
  - interferenza di thread
  - inconsistenza della memoria
- Per risolvere questi problemi, necessaria la ***sincronizzazione***
  - che a sua volta genera problemi di contesa: quando più thread cercano di accedere alla stessa risorsa simultaneamente (deadlock e livelock)

85

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

86

Un semplice contatore

```
class Counter {
    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}
```

variabile privata

87

Un semplice contatore

```
class Counter {
    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}
```

variabile privata

metodo di incremento

88

Un semplice contatore

```
class Counter {
    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}
```

variabile privata

metodo di incremento

metodo di decremento

89

Un semplice contatore

```
class Counter {
    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}
```

variabile privata

metodo di incremento

metodo di decremento

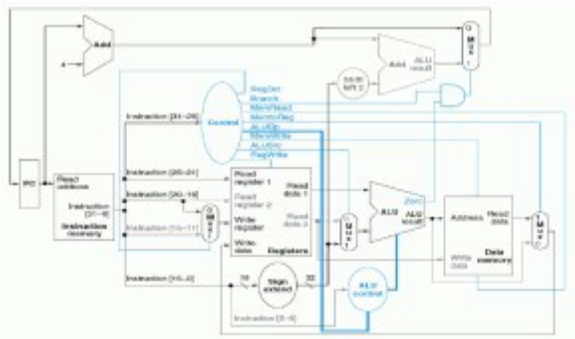
metodo di accesso

90

Interferenza

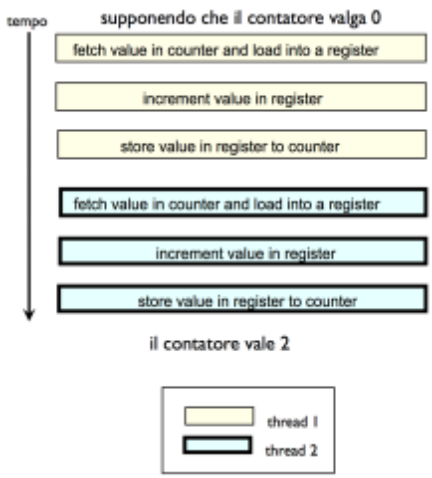
Le basi: come si esegue un incremento?

- L'operazione c++ è eseguita in tre passi
  - fetch operando dalla locazione di memoria di **c** nel registro
  - incremento del registro
  - memorizzazione in **c**



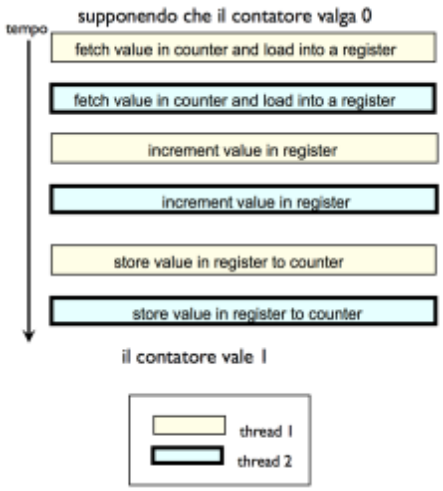
91

Interferenza
Interleaving dei Thread - 1



92

Interferenza
Interleaving dei Thread - 2



93

Interferenza
Il problema di accesso concorrente

- **Race condition**: quando il risultato di una operazione dipende dall'ordine di esecuzione di diversi thread
- Gli errori dovuti ad una race condition sono tipicamente transienti e difficili da riprodurre (debugging difficile!)
  - oltre alla transienza e irriproducibilità, l'uso del debugger può alterare l'ordine di esecuzione dei task
- Questi bug vengono anche detti Heisenbug che richiamano il principio di indeterminazione di Heisenberg in fisica quantistica

Principio di Heisenberg

Non è possibile misurare simultaneamente la posizione ed il momento (quantità di moto, cioè massa per velocità) di una particella"

94

Organizzazione della lezione

- Motivazioni alla programmazione concorrente
  - La tecnologia dei microprocessori
  - Le sfide
  - A cosa serve la programmazione concorrente
- I Thread in Java
  - Processi e Thread
  - Alcuni metodi utili
- I tipi di errori con la programmazione concorrente
  - Interferenza
  - Inconsistenza della memoria
- Conclusioni

95

Inconsistenza della memoria

- . . . quando thread diversi hanno visioni diverse dei dati
- Cause: protocolli di coerenza di cache, ottimizzazioni hardware/software, etc.
- La **happens-before** è una garanzia che la memoria scritta da un thread è visibile da un altro thread

96

Inconsistenza della memoria

- Un esempio: Il campo contatore è condiviso tra due thread, A e B

```
int counter = 0;
//...
counter++;
//...
System.out.println(counter);
```

- Supponiamo che A incrementi il contatore: counter++;
- Supponiamo che subito dopo B esegue la stampa: System.out.println(counter);
- . . . può capitare che la modifica di A non sia visibile a B (che stampa 0)
- Bisogna stabilire una relazione **happens-before**

97

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

98

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

99

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

Definizione metodo

100

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

Definizione metodo

Variabile final

101

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

Definizione metodo

Variabile final

Un thread che mette 1 in bar

102

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

Definizione metodo

Variabile final

Un thread che mette 1 in bar

Lancio del thread

103

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

Definizione metodo

Variabile final

Un thread che mette 1 in bar

Lancio del thread

Si attende 1 secondo…

104

Un esempio di memory (in)consistency

```
class Foo {
 int bar = 0;
 public static void main(String args[]) {
     (new Foo()).unsafeCall();
 }

 void unsafeCall () {
  final Foo thisObj = this;

  Runnable r = new Runnable () {
      public void run (){
              thisObj.bar = 1;
      }
  };

  Thread t = new Thread(r);
  t.start( );

  try{
      Thread.sleep(1000);
  }catch(InterruptedException e) {
      e.printStackTrace();
  }

  System.out.println("bar="+ bar );
 }
}
```

Dichiarazione variabile

Nel main si chiama un metodo

Definizione metodo

Variabile final

Un thread che mette 1 in bar

Lancio del thread

Si attende 1 secondo…

…e si stampa:
può essere 0 oppure 1!

105

Relazioni happens-before

Come si stabilisce una relazione happens-before?

106

Relazioni happens-before

1

- Una maniera è tramite la sincronizzazione

107

Relazioni happens-before

2

- Due operazioni, che abbiamo descritto, introducono una relazione happens-before
  - **Thread.start():** gli effetti del codice che ha condotto alla creazione sono visibili al nuovo thread
  - **Thread.join():** quando la terminazione di un thread A causa il return della join() di B, tutte le istruzioni di A sono in happens-before le istruzioni di B che seguono la join

108

Relazioni happens-before

3

- Un'altra maniera è rendere la variabile volatile: una scrittura a un campo volatile assicura la relazione happens-before per ogni successiva lettura della variabile (da parte di qualsiasi thread)

La keyword volatile è di solito associata ad una variabile il cui valore viene salvato e ricaricato in memoria ad ogni accesso senza utilizzare i meccanismi di caching.

109

Relazioni happens-before

- La keyword volatile è di solito associata ad una variabile il cui valore viene salvato e ricaricato in memoria ad ogni accesso senza utilizzare i meccanismi di caching
- Vediamo un esempio
  - Nell'esempio di seguito, non dichiarare la variabile volatile potrebbe portare il primo Thread a non terminare mai

110

Relazioni happens-before

Thread 1 parte ed incrementa un contatore

running = true

```
public class VolatileTest {
  volatile boolean running = true; // da notare la parola chiave volatile

   public void test() {
      // lancio un primo Thread

      new Thread(new Runnable() {
        public void run() {
         int counter = 0;
         while (running)
           counter++;
         System.out.println("Thread 1 concluso. Contatore = " + counter);
        }
      }).start();
```

111

Relazioni happens-before

Thread 2 parte ed esegue

running = false

```
// lancio il secondo Thread

      new Thread(new Runnable() {
        public void run() {
          try {
            Thread.sleep(100);
          // Questo sleep è necessario per dare al primo thread la possibilità di partire
          } catch (InterruptedException ignored) { }
          System.out.println("Thread 2 concluso");
          running = false;
        }
      }).start();
   }
   public static void main(String[] args) {
      new VolatileTest().test();
   }
}
```

112

Relazioni happens-before

- Cosa è successo?
- Il primo Thread potrebbe non terminare mai… perché?

113

Relazioni happens-before

- Il primo Thread carica in **cache** il valore della variabile booleana *running* (impostato a *true*) e non va più a leggere il valore effettivo quando il secondo Thread lo modifica a *false*
  - A causa di questo valore non aggiornato il primo Thread prosegue all'**infinito** senza mai terminare
- Dichiarando **volatile** la variabile *running* invece si costringe il Thread (o chi per esso) ad aggiornare di volta in volta il valore senza memorizzarlo in cache

114

Corso di Laurea in Informatica
I Anno Magistrale, Indirizzo Cloud Computing
Reti Geografiche: Struttura, Analisi e Prestazioni

Programmazione concorrente & Thread in Java

Delfina Malandrino
dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

115