Ottava esercitazione

File comandi Unix
Introduzione Java Thread

Stefano Monti
stefano.monti6@unibo.it

File Comandi Unix

Un esempio di ricorsione

- Scrivere un programma che, dato un direttorio in ingresso, stampi un elenco di file contenuti nel direttorio e in tutti i suoi sottodirettori (analogamente al comando ***ls -R***)

Schema di soluzione

`recurseDirectory <arg1>`

- caso base

  `<arg1>` **è un file** → stampo il nome

- caso generale espresso in termini ricorsivi

  `<arg1>` **è una directory** →

  - mi muovo nella directory `<arg1>`
  - per ogni file (normale o directory) **invoco nuovamente recurseDirectory**

Bozza di soluzione

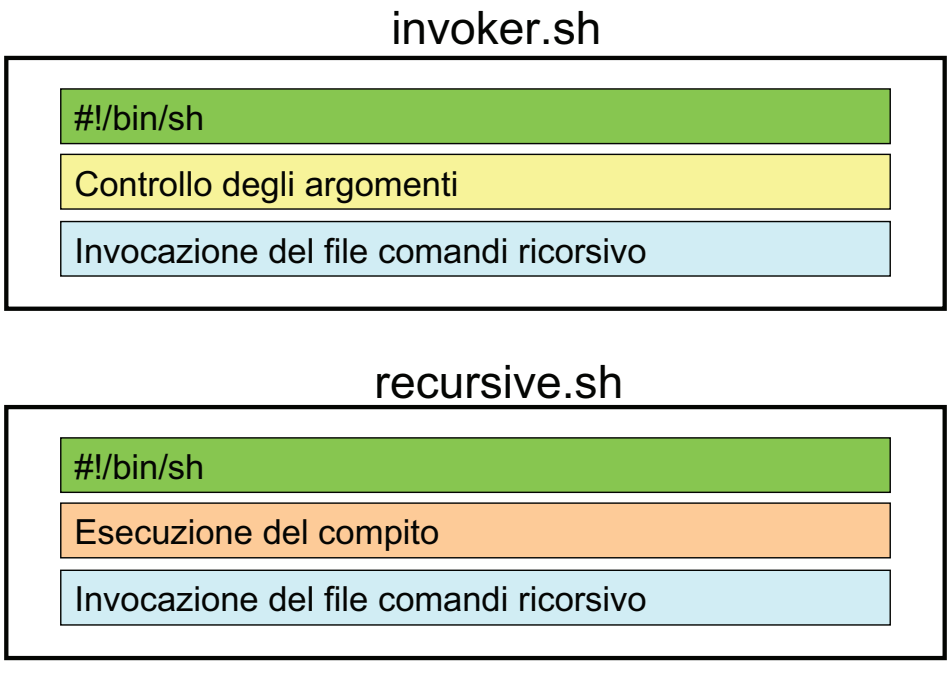
```
#!/bin/sh
if ! test -d $1
then
  echo `pwd`/$1
else
  cd $1
  for f in *
  do
   $0 "$f"
  done
fi
exit 0
```

Caso base

Caso generale

Ricorsione!

Struttura di un file comandi ricorsivo



Esercizio

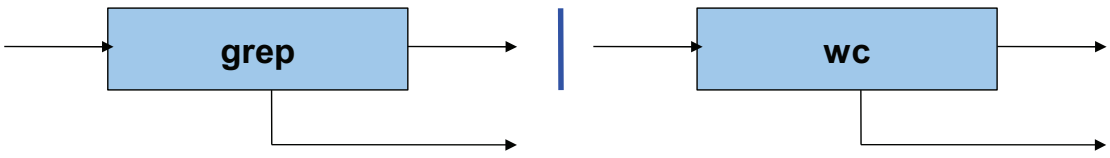
Si scriva un file comandi con interfaccia

```
contaOccorrenze <parola> <dir>
```

- **dir** è un nome **assoluto** di direttorio
- **parola** è una qualunque sequenza di caratteri
- compito del file comandi è quello di stampare a video il nome del file con il maggior numero di linee contenenti ***parola***, **per ogni sottodirettorio** di **dir**.

Suggerimenti...

- Per cercare occorrenze di stringhe all'interno di file → comando grep
  - a default, stampa in output le sole righe contenenti la stringa cercata
- Per contare righe, caratteri o parole di un file → comando wc
- Nel nostro caso, è molto comodo pensare che l'output di grep costituisca l'input di wc → piping



Soluzione – invoker.sh

```
#!/bin/sh
if test $# -ne 2
then
echo "usage:$0 <parola> <dir>"
   exit 1
fi
case $2 in
   /*) ;;
   *)  echo "$2 is not an absolute directory"
       exit 3;;
esac
if ! test -d "$2"
then
   echo "$2 is not a valid directory"
   exit 4
fi
oldpath=$PATH
PATH=$PATH:`pwd`
recursive.sh "$1" "$2"
PATH=$oldpath
```


Esercizio - recursive.sh

```
#!/bin/sh
cd "$2"
counter=0 #valore massimo corrente
for f in *
do
   if test -h "$f" #se il file corrente e' un link
   then
        continue
   elif test -d "$f"
   then
      $0 "$1" "$f"
   elif test -f "$f"
   then
    cur=`grep "$1" "$f" | wc -l `
    if test $cur -gt $counter
    then
        counter=$cur
    fi
   fi
done
echo massimo numero di righe in $2: $counter
exit 0
```

Se volessi il file con massimo num di righe **in tutti i sottodirettori**, potrei continuare ad utilizzare la variabile $counter?

Esercizio

Realizzare un file comandi (ricorsivo) che abbia la sintassi

```
search.sh <minSize> <maxSize> <dir1> <dir2>…<dirN>
```

Elenco (di lunghezza non nota a priori) di direttori

dove

- `<minSize> <maxSize>` sono due interi
- `<dir1>…<dirN>` sono un numero N qualsiasi, non noto a priori, di nomi di **direttori assoluti** che devono esistere nel file system.

Esercizio

Il compito del file comandi è quello di

- visitare (ricorsivamente) tutti i **sottoalberi** individuati da <dir1> … <dirN>
- per ogni file trovato verificare che la **dimensione in KB** sia compresa tra <minSize> e <maxSize>
  - suggerimento: vedere il comando **stat** per ottenere la dimensione del file
- scrivere il nome assoluto di ciascun file che soddisfi il precedente requisito in un file di report da inserire nella home dell'utente che ha invocato il comando

Java Thread

Thread

Un thread è un ***singolo flusso sequenziale*** di controllo all'interno di un processo (task)

Un thread (o processo leggero) è un'unità di esecuzione che ***condivide codice e dati*** con altri thread ad esso associati

Un ***thread***

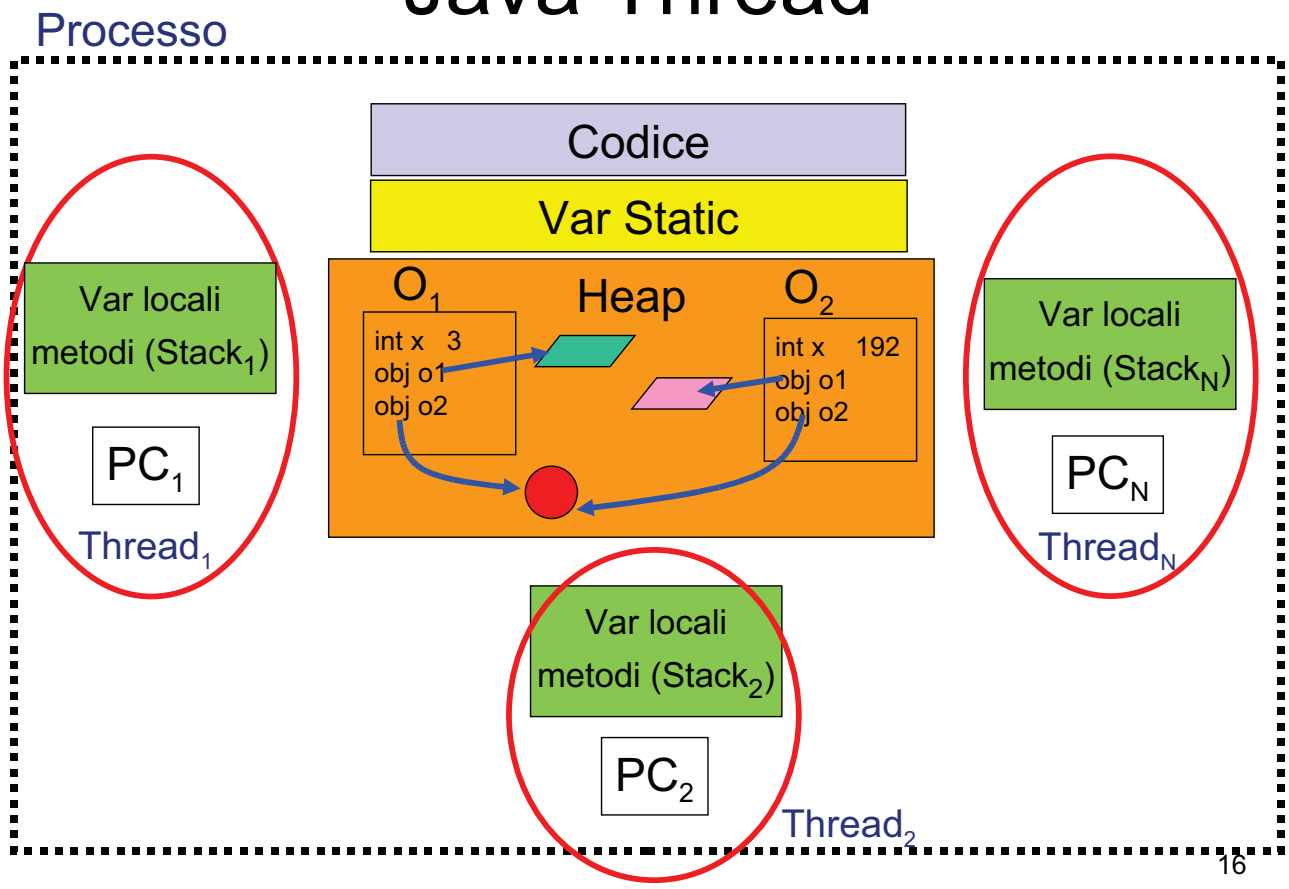
- ***NON*** ha spazio di memoria riservato per dati e heap: tutti i thread appartenenti allo stesso processo condividono lo ***stesso spazio di indirizzamento***
- ha ***stack*** e ***program counter privati***

I threads in Java

- All'esecuzione di ogni programma Java corrisponde un task che contiene almeno un *singolo thread*, corrispondente all'esecuzione del metodo *main()* sulla *JVM.*

- E' possibile creare dinamicamente nuovi thread *attivando concorrentemente* le loro esecuzioni all'interno del programma.

Java Thread



Java Thread: programmazione

Due modalità per implementare i thread in Java:

- ***estendendo la classe `Thread`***
- ***implementando l'interfaccia `Runnable`***

1. Thread come oggetti di sottoclassi della classe `Thread`

- I thread sono oggetti che derivano dalla **classe Thread** (fornita dal package **java.lang).**
- Il metodo **run()** della classe di libreria **Thread** definisce *l'insieme di istruzioni Java* che ogni thread (oggetto della classe) eseguirà (NB: nella classe **Thread** l'implementazione del metodo **run** è vuota).
- In ogni sottoclasse derivata da **Thread** il metodo **run** deve essere ridefinito (*override*) specificando all'interno di esso cosa far eseguire ai thread di quella classe.
- Per creare un thread, si deve creare un'istanza della classe che lo definisce tramite **new;** dopo la new il thread esiste, ma **non è ancora attivo**.
- Per attivare un thread si deve invocare il metodo **start()** che a sua volta invoca il metodo run()

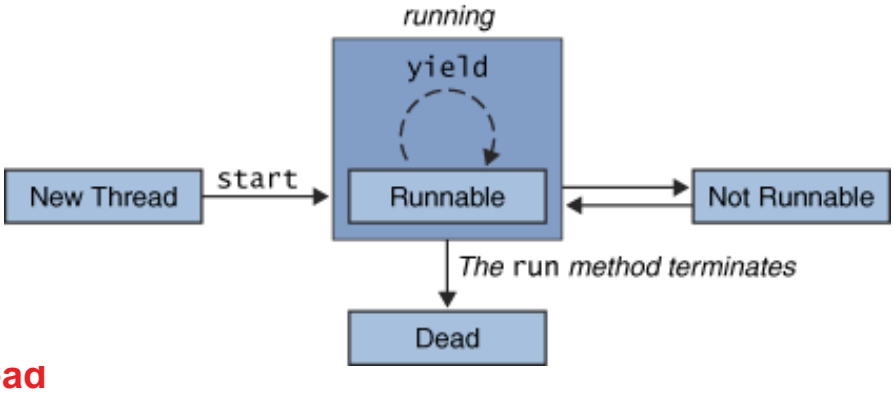
Possibile schema

```
class SimpleThread extends Thread {
  public void SimpleThread()
  { <costruttore> }

  public void run() {
    <corpo del programma eseguito>
    <da ogni thread di questa classe>
    }
}

public class EsempioConDueThreads
{  public static void main (string[] args)
   {  SimpleThread t1=new SimpleThread();
      t1.start();//attivazione del thread t1
      <resto del programma eseguito
      dal thread main>
   }
}
```


- La classe **SimpleThread** (estensione di Thread) implenta i nuovi thread *ridefinendo il metodo* **run**.
- La classe EsempioConDueThreads fornisce il **main** nel quale viene creato il thread **t1** come oggetto derivato dalla classe **Thread.**
- Per **attivare** il thread deve essere chiamato il metodo **start()** che invoca il metodo **run()** (il metodo **run()** non può essere chiamato direttamente, ma solo attraverso **start()**).

➔ Abbiamo creato *due thread concorrenti*: il thread principale associato al **main** ed il thread **t1.**

Ciclo di vita di un thread



New Thread

- Subito dopo l'istruzione **new**
- Il costruttore alloca e inizializza le variabili di istanza

Runnable

- Il thread è eseguibile ma potrebbe non essere in esecuzione

Ciclo di vita di un thread

Not Runnable

- Il thread non può essere messo in esecuzione
- Entra in questo stato quando è in attesa della terminazione di un'operazione di I/O, cerca di accedere ad un metodo "synchronized" di un oggetto bloccato, o dopo aver invocato uno dei seguenti metodi: sleep(), wait(), suspend()
- Esce da questo stato quando si verifica la condizione complementare

Dead

- Il thread giunge a questo stato per "morte naturale" o perché un altro thread ha invocato il suo metodo stop()

Esempio: primo metodo

```
public class SimpleThread extends Thread{

        public SimpleThread(String str)
    {super(str);}

    public void run() {
        for(int i=0; i<10; i++)
        { System.out.println(i+ " " +getName());
          try{
           sleep((int)Math.random()*1000);
           } catch (InterruptedException e){}
        }
          System.out.println("DONE! "+getName());
        }
    }


}
```


Java Thread

```
public class EsempioConDueThreads
{ public static void main(String[] args)
   {   SimpleThread st1 = new SimpleThread(”Pippo");
    st1.start();
   }
}
```

E se occorre definire thread che non siano necessariamente sottoclassi di Thread?

2. Thread come classi che implementano Runnable

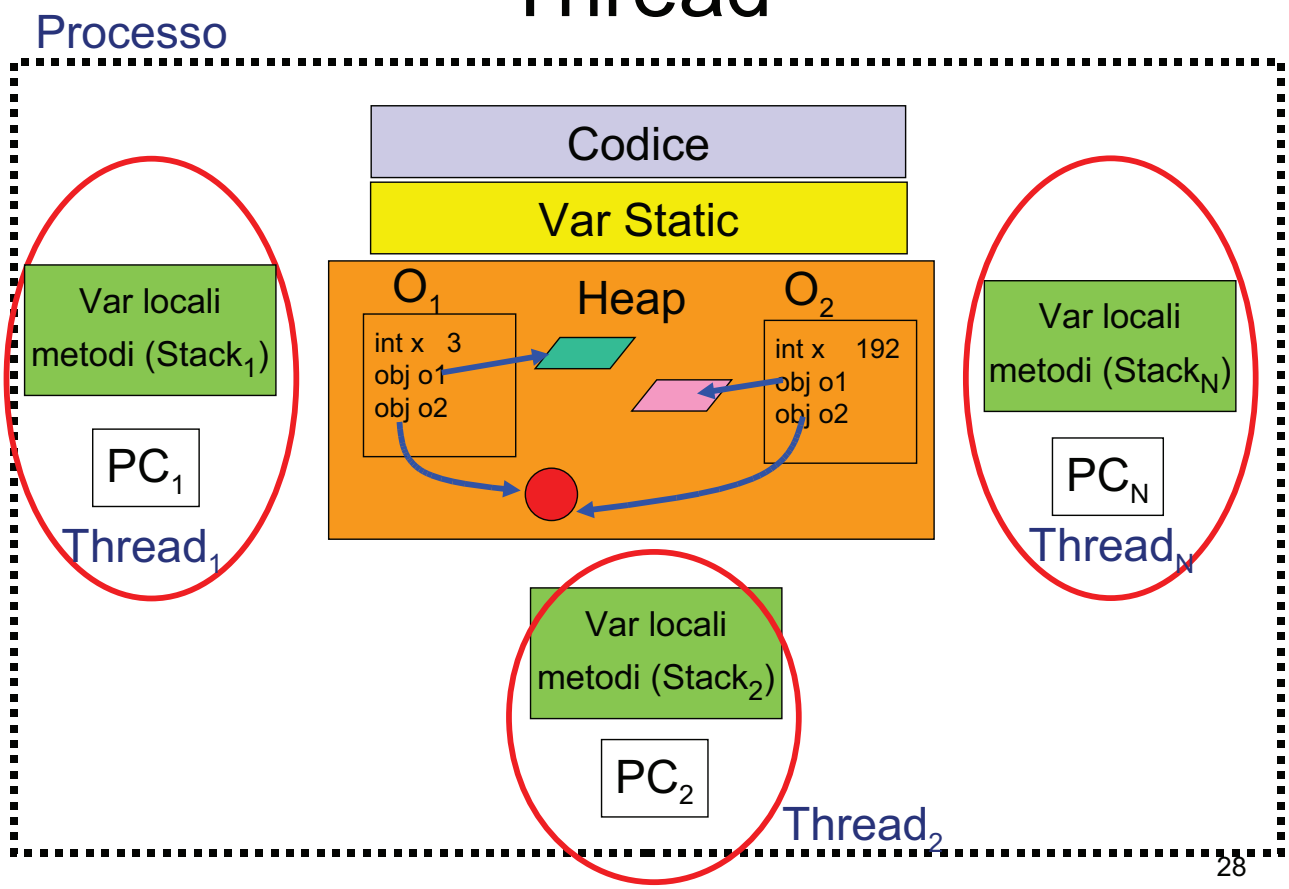
Definizione di thread come classe che implementa interfaccia **Runnable**

- la classe deve ***ridefinire il metodo*** `run()`
- si crea un'istanza di tale classe tramite **`new`**
- si crea ***un'istanza della classe Thread*** con **`new`**, passandole come ***parametro l'oggetto che implementa*** `Runnable`
- si esegue il thread invocando il metodo **`start()`** ***sull'oggetto con classe Thread creato***

Java Thread

```
class EsempioRunnable extends MiaClasse
            implements Runnable {
   public void run() {
         for (int i=1; i<=10; i++)
    System.out.println(i + " " + i*i);
   }
}

public class Esempio {
   public static void main(String args[]){
    EsempioRunnable e = new EsempioRunnable();
    Thread t = new Thread (e);
    t.start();
   }
}
```


Thread



Sincronizzazione di thread

Differenti thread ***condividono lo stesso spazio di memoria (heap)***

- è possibile che più thread accedano **contemporaneamente** a uno stesso oggetto, ***invocando un metodo che modifica lo stato dell'oggetto***
- stato **finale** dell'oggetto sarà ***funzione dell'ordine*** con cui i thread accedono ai dati

→Servono meccanismi di ***sincronizzazione***

Esercizio Java Thread

Si consideri un lavaggio automatico di veicoli a cui possono accedere auto e moto. Ciascun veicolo può entrare nel lavaggio auto e richiedere il lavaggio specifico per il tipo di veicolo.

Facendo riferimento al meccanismo di creazione dei thread in Java si realizzi un'applicazione che crei i thread, rappresentativi dei veicoli.

In particolare, l'applicazione dovrà creare thread rappresentativi delle moto e thread rappresentativi di auto piccole e auto grandi che ereditino le proprietà dalla classe auto.

L'applicazione permetterà a ciascun veicolo di scrivere a video il tipo di lavaggio richiesto.

Impostazione

Come creare i thread?

- **Automobili**: auto grandi e piccole come sottoclassi di auto:
  - È necessario usare l'interfaccia runnable
- **Moto**: non ci sono vincoli
  - È possibile usare entrambi i metodi

Quante classi?

- Lavaggio: contiene il main
- Auto: definisce le caratteristiche delle auto in generale (marca, modello, targa, cilindrata,….)
- Autograndi: eredita da Auto ed esibisce un comportamento specifico (messaggio da stampare)
- Autopiccole: eredita da Auto ed esibisce un comportamento specifico (messaggio da stampare)
- Moto