Ottava Esercitazione

Shell Scripting
Introduzione a Java Thread

Andrea Reale
andrea.reale@unibo.it

Modifica orario di ricevimento

- Mercoledì
  - ore 16 - 18

- Conferma via email (almeno la mattina stessa :-))
  - andrea.reale@unibo.it
  - Tel. (051 20) 93087

Agenda

- **Esercizio 1**
  - Shell scripting
- **Esercizio 2**
  - Concorrenza in Java: scrivere codice concorrente ed eseguirlo tramite Thread.

Esercizio 1

Shell Scripting

Esercizio 1 - Traccia (1/3)

Creare un file comandi Unix (script bash) avente la seguente interfaccia:

chperms.sh mode path

- **path**: è un path relativo o assoluto di una **directory esistente** nel file system
- **mode:** corrisponde ad un singolo carattere che può essere il carattere '+' o il carattere '-'

Esercizio 1 - Traccia (2/3)

Il comportamento dello script realizzato dovrà essere il seguente:

1. Il sottoalbero individuato da **path** dovrà essere esplorato ricorsivamente in cerca di **file regolari**
2. Per ogni **file regolare** incontrato durante l'esplorazione si dovrà controllare che il **proprietario** del file sia l'*utente che sta eseguendo lo script*
3. In caso affermativo:
   - Se il parametro **mode** passato allo script è '+', si dovranno **assegnare permessi di scrittura** al proprietario del file
   - Se il parametro **mode** è '-', si dovranno **revocare i permessi di scrittura** al proprietario del file

Esercizio 1 - Traccia (3/3)

Inoltre il programma deve produrre un'output che soddisfi le seguenti specifiche:

- Per ognifile **non appartenente all'utente che esegue lo script** incontrato, si dovrà produrre su stderr il messaggio
  - "Il file <X> non appartiene all'utente <Y>"
- Ogni volta che i permessi di un file sono impostati, si dovrà produrre su stdout il messaggio
  - "Permessi di scrittura modificati per il file <X>"

Note alla soluzione

- Come assunzione semplificativa, non è richiesto di controllare quali siano i permessi dei file incontrati prima di cambiarli
- Come si fa a stampare messaggi su stderr?
- Comandi utili (si guardino le rispettiva **man pages** per dettagli)
  - **stat -c %U <file>**
  - **whoami** (o variabile d'ambiente **$USER)**
  - **chmod [+|-]w**

Invocazione dello script ricorsivo: un'alternativa

- Uso dell'ereditarietà delle variabili d'ambiente

invoker.sh

```
#!/bin/bash

# controllo argomenti
...

# oldpath=$PATH
PATH="$PATH:`pwd`"

recurse.sh <params>

# PATH=$oldpath
```

recurse.sh

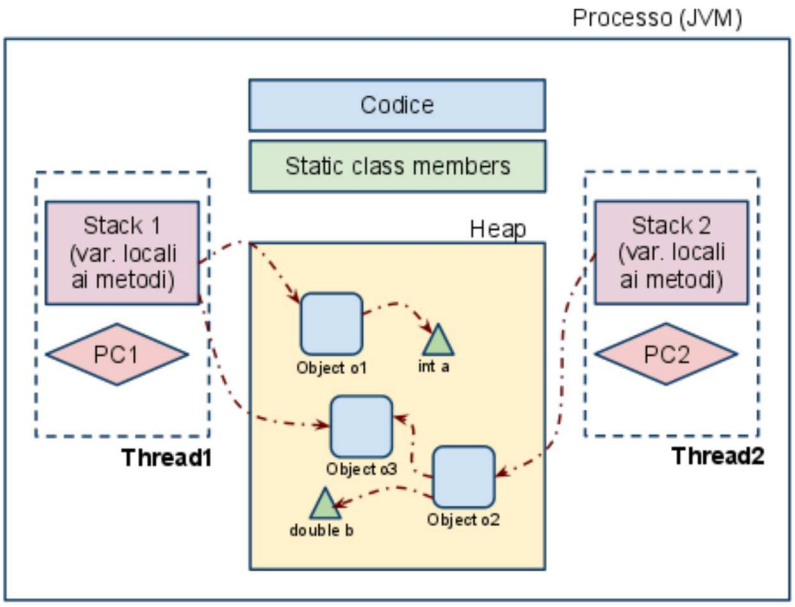
```
#!/bin/bash

# 1. check arguments
...
# 2. do program logic
...
# 3. If necesseray,
# recurse
"$0" <params>
```

Esercizio 2

Programmazione concorrente in Java: i Java Thread

Modello di Threading base in Java



- Un processo pesante che esegue la JVM (aka JRE)
- All'interno di una singola JVM possono agire più processi leggeri (thread)
- Spazio di indirizzamento comune tra Thread
  - Heap Condiviso
  - Stack non condiviso
  - Program Counter non condiviso

Il Thread main

- Ogni applicazione Java ha un entry point, che è il metodo statico main() della classe eseguita dalla JVM
- All'avvio dell'applicazione, la JVM crea un Thread che esegue il metodo main()
- Ulteriori Thread possono essere creati dinamicamente ed eseguiti in maniera concorrente
  - Definendo sottoclassi della classe Thread, istanziandole e facendole partire a runtime
  - Definendo oggetti che implementano l'interfaccia Runnable

Sottoclassi della classe Thread

```
public class SimpleThread extends Thread {
   // Class members
   // Costruttore
   public SimpleThread() { ... }

   public void run() {
      // Questo metodo è eseguito all'avvio
      // di tutti i thread di questa classe
      // (entry point)
   }

   public static void main(String args[]) {
      // 1. Instanzion un oggetto della classe SimpleThreads
      SimpleThread t = new SimpleThread();
      // 2. Avvio il thread con il metodo start()(non run()!)
      t.start();
      // Il resto del codice del main sarà eseguito dal Thread
      // main. Il thread t eseguirà il suo metodo run()
   }
}
```

Implementare l'interfaccia Runnable

```
public class SimpleThread implements Runnable {
   // Class members
   // Costruttore
   public SimpleThread() { ... }
   public void run() {
      // Questo metodo è eseguito all'avvio
      // di tutti i thread di questa classe
      // (entry point)
   }
   public static void main(String args[]) {
      // 1. Instanzion un oggetto della classe SimpleThreads
      SimpleThread st = new SimpleThread();
      // 2. Instanzio un oggetto Thread che dovrà eseguire
      //    il comportamento definito da SimpleThread
      Thread t = new Thread(st);
      // 3. Avvio il thread con il metodo start()(non run()!)
      t.start();
      // Il resto del codice del main sarà eseguito dal Thread
      // main. Il thread t eseguirà il suo metodo run()
   }
}
```

Confronto tra le soluzioni

- **Ereditare della classe Thread**
  - **Più semplice** (?) da usare in applicazioni semplice
  - **Come funziona l'ereditarietà in Java?** Quali limiti impone a questo approccio?
  - Che vantaggi, invece?

- **Implementare l'interfaccia Runnable** e "farsi eseguire" da un oggetto Thread
  - Approccio più generale e **flessibile**
  - Permette di **ereditare** comportamento da altre classi
  - **Separazione** del concetto di *task* da eseguire (implementazione del comportamento nell'oggetto Runnable) e di oggetto *esecutore* (oggetto Thread)

Esercizio 2 - Traccia (1/2)

Scrivere una applicazione Java che simuli un semplice autolavaggio.

- Nell'autolavaggio possono entrare sia automobili che moto.
  - Le automobili possono essere di **due tipi**, ossia auto grandi oppure auto piccole.
  - Le moto, invece, **sono di un unico tipo**.
- Tutti gli autoveicoli devono essere *oggetti attivi* (**ossia in grado di eseguire in maniera concorrente tramite thread**)
- In particolare, ciascun autoveicolo, quando eseguito, dovrà **stampare** su stdout un opportuno messaggio che descriva le sue caratteristiche.

Esercizio 2 - Traccia (2/2)

Nello specifico, il programma deve definire le seguenti classi:

- **Automobile**: definisice le caratteristiche comuni di un'automobile (marca, modello, targa, cilindrata, ...), un **metodo astratto** getType() ed un **metodo** concreto getMessage()che ritorna il messaggio da stampare e che richiami getType() per capire il tipo di automobile.
  - **AutoGrande**: eredita da Automobile e specializza il comportamento di getType() in modo che ritorni la stringa "auto grande"
  - **AutoPiccola**: eredita da Automobile e specializza il comportamento di getType() in modo che ritorni la stringa "auto piccola"
- **Moto:** definisce le caratteristiche della moto
- **Autolavaggio:** implementa il metodo main() che crea un numero (a vostra scelta) di veicoli di ciascun tipo e li mette in esecuzione tramite thread

Note

- Per risolvere l'esercitazione potete (come sempre) utilizzare i tool di vostra preferenza tra quelli disponibili in laboratorio
  - **Eclipse IDE**
  - editor di testo + javac + java + jdb

- **Scopo dell'esercitazione**
  - Scegliere opportunamente il modello di creazione dei thread più appropriato, capendone le differenze di uso e di capacità espressiva
  - Primi passi con l'esecuzione concorrente di Thread in Java

- Link utili:
  - Oracle **Java Doc** per Java 6 SE: http://download.oracle.com/javase/6/docs/api/
  - Buon tutorial Oracle sulla concorrenza in Java: http://download.oracle.com/javase/tutorial/essential/concurrency/