UNIVERSITÀ
DI PARMA

Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Corso di Laurea Triennale in Informatica

Consegna3 Esame LaTex

Submission3 LaTeX Exams

Candidato:
Dennis Turco

Relatore:
Prof. Luigi Marchi

Correlatori:
Prof. Marco Aurelio
Prof. Alessio Franchi

Anno Accademico 2020–2022

Indice

Introduzione **1**

1 Example01 **3**
1.1 Costruzione dei thread 4
1.1.1 Notifier 4
1.1.2 Waiter 5
1.2 Example01 Programma 5
1.2.1 Example01.java 5
1.2.2 Output Programma 7
1.3 Citazioni 7

Conclusione **8**

Bibliografia **9**

Elenco delle figure

1.1 Stati e Metodi del Thread 3
1.2 Output del Programma (Example01.java) 7

Elenco degli algoritmi

1 Example01.java 6

Elenco delle tabelle

1.1 Descrizione Metodi 4

Introduzione

Prima di tutto ci tengo a precisare che le informazioni trattate le ho recuperate dal corso di Ingegneria del Software dell'Università di Parma. Per comodità e siccome in si tratta di davvero una tesi di laurea tratterò solo un capitolo del corso.
In realtà, il corso partirebbe con la teoria a partire dalle **Tautologie** (che però non tratterò).
Tautologie = Nella logica formale classica, proposizione che, volendo definire qualche oggetto o concetto, non faccia altro che ripetere sul predicato quanto è già detto sul soggetto. Qui la Definizione
Le tautologie sono dette anche leggi logico-enunciative. Sono esempi di proposizione vere a prescindere dal valore di verità delle variabili enunciative (Wikipedia).

Esempio, Supponiamo:
"$x > y$ " is true.
"$\int f(x)\,dx = g(x) + C$ " è falso.
"Calvin ha i calzini viola" è vero.
Determinare il valore di verità.

$$(x > y \int f(x)\,dx = g(x) + C), \neg(\text{Calvin ha i calzini viola})$$

Per semplicità:
P = "$x > y$ ".
Q = "$\int f(x)\,dx = g(x) + C$ ".
R = "Calvin ha i calzini viola".
Voglio determinare il valore di verità di $(PQ), \neg R$. Poiché mi sono stati dati valori di verità specifici per P, Q e R, ho impostato una tabella di verità con

una singola riga utilizzando i valori dati per P, Q e R:

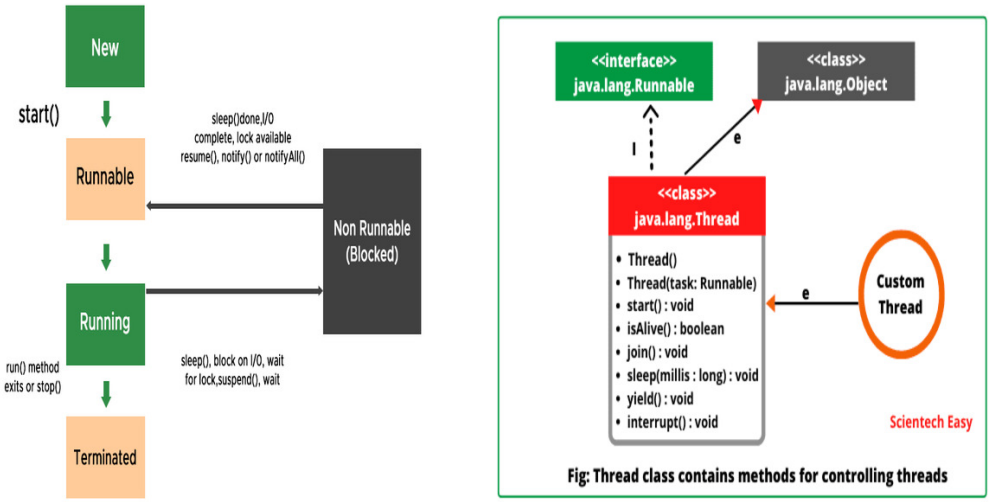
P	Q	R	PQ	$\neg R$	$(PQ), \neg R$
T	F	T	F	F	T

Capitolo 1

Example01

In questo primo capitolo tratterò del primo esempio che ci è stato mostrato nel corso di Ingegneria del software: scriveremo in una classe 2 metodi (Waiter, Notifier) attraverso una labda-expression, che gestiranno al loro interno dei Thread e svolgeranno operazioni rispettivamente di wait() e notify().

Innanzitutto è necessario fare una doverosa precisazione sullo stato e metodi dei Threads (Figura 1.1).



(a) Stato del Thread (b) Metodi del Thread

Figura 1.1: Stati e Metodi del Thread

Attraverso la seguente tabella è possibile per metodo, ottenere una breve descrizione.

Metodo	Significato
getName	*ottiene il nome del Thread*
getPriority	*ottiene la priorità del Thread*
isAlive	*determina se il Thread è ancora in esecuzione*
join	*attende che il Thread termini*
run	*mette in esecuzione il Thread*
sleep	*sospende il thread per un periodo di tempo*
start	*attiva il thread attraverso il metodo "run"*

Tabella 1.1: Descrizione Metodi

Abbiamo precedente visto la costruzione dei thread in JAVA (estendendo Thread o implementando Runnable) e abbiamo visto come costruire un progetto in Eclipse. Abbiamo visto come aggiungere una classe in un package, quindi usiamo quello appena imparato per crearne due:

- un thread ***Waiter*** rimane in attesa aspettando la notifica di un altro;
- un thread ***Notifier*** dopo un'attesa di tot secondi randomica, notifica con notifyAll().

1.1 Costruzione dei thread

1.1.1 Notifier

Al suo interno estendiamo la classe **Thread**, siccome uno dei due modi per creare thread è o questo, o implementare l'interfaccia Runnable. Siccome il metodo run() (seppure vuoto) esiste già all'interno di **Thread**, facciamo @Override per scriverne un'implementazione.

Inoltre dobbiamo catturare l'eccezione perché quando siamo in stato d'attesa qualcuno dall'esterno, può bloccare il **Thread** che lancerà, per ciascuno, *InterruptedException*. Ogni **Thread** verrà interrotto

Se il Waiter deve fare wait() su un oggetto, e il Notifier deve fare notifyAll() su un oggetto, quello sarà lo stesso oggetto, che è **Example01**. Notifier non vede tuttavia l'oggetto **Example01**. Ragionevolmente lo scopo di Notifier è solo quello di costruire un thread e fare notifyAll(), mentre quello del Waiter e' fare wait(). Un nuovo file separato dall'oggetto **Example01** non

ha molto senso, siccome entrambi Notifier e Waiter devono vedersi gli stati a vicenda. Spostiamo quindi il codice della Notifier dentro a **Example01**, cancellando il vecchio file e mentre lo facciamo, anonimizziamo la classe siccome dargli un nome non e' necessario, siccome la usiamo una volta soltanto.

1.1.2 Waiter

Si dovrà mettere in attesa e aspettare che il Notifier lo notifichi. Per crearlo, ci basta prima ridefinire l'implementazione di **Runnable**.
L'oggetto su cui facciamo operazioni e' uno di tipo **Object**, privato ad **Example01**, e lo scriviamo in cima, prima della go(). Chiamiamo per semplicita' questo oggetto: ***mutex***.

Abbiamo tuttavia un problema, non irrilevante, di sincronizzazione. Seguiamo questo ragionamento per identificarlo (vedi commenti in alto numerati):

1. facciamo start() di Notifier;
2. facciamo sleep(5000);
3. facciamo notifyAll();
4. facciamo start() del Waiter;
5. facciamo System.out.println("Waiter started");
6. facciamo wait();

Niente ci garantisce che la NotifyAll() venga fatta dopo la wait(). Potrebbe succedere, anche se poco probabile nel caso di 5 secondi di attesa, che i due thread partano ma senza l'ordine da noi voluto.

1.2 Example01 Programma

Figura 1.2 mostra l'output del programma.

1.2.1 Example01.java

Algoritmo 1 Example01.java

```
1   package it.unipr.informatica.example;
2   public class Example01 {
3       private Object mutex = new Object();
4       private boolean waitInProgress = false;
5
6        public void go() {
7           waitInProgress = false;
8           Thread notifier = new Thread(this::doNotify);
9           Thread notifier = new Thread(this::doWait);
10          notifier.start();
11          waiter.start();
12       }
13       private void doWait() {
14           System.out.println("Waiter started");
15           synchronized(mutex) {
16               waitInProgress = true;
17               mutex.notifyAll();
18               try {
19                   mutex.wait();
20               } catch(Throwable throwable) { //blank }
21           }
22           System.out.println("Waiter terminated");
23       }
24       private void doNotify() {
25           System.out.println("Notifier started");
26           synchronized(mutex) {
27               try {
28                   while (!waitInProgress)  mutex.wait();
29                   Thread.sleep(5000);
30                   mutex.notifyAll();
31               } catch (Throwable trhowable) { //blank }
32           }
33           System.out.println("Notifier terminated");
34       }
35       public static void main(String[] args) {
36           new Example01().go();
37       }
38  }
39
```

1.2.2 Output Programma

Example01.java

Figura 1.2: Output del Programma (Example01.java)

1.3 Citazioni

La scrittura di questo report accademico è stato possibile anche grazie a testi che ci sono stati assegnati durante il corso come: [Lewis et al., 2009, Robles, 2004]. Estremamante importante è stato anche l'articolo del professore Lars Bendix [Bendix, 2006], che oltretutto, è stato mio insegnante all'università per 2 settimane.

Conclusione

Questo esercizio consiste in un semplice programma con i Thread, in particolare è pensato come primo impatto alla progammazione multi Thread in cui si cerca di fornire un primo approccio con esso, attraverso i metodi classici: Notift(), NotifyAll() e Wait(). Si cerca, inoltre, di insegnare al lettore la metodologia corretta, in quanto lo stesso programma lo si poteva realizzare anche in altri modi, per esempio attraverso l'utilizzo di più classi, inserendo il metodo "doWait()" e "doNotify()" in 2 classi separate (approccio dal punto di vista metodologico scorretto). Infine, si cerca anche di fornire familiarità con l'utilizzo del blocco "syncronized { ... }", in cui vengono inserite in esso quelle risorse in mutua esclusione che non devono essere accedute/modificate da 2 o più Thread contemporaneamente.

Bibliografia

[Bendix, 2006] Bendix, L. (2006). A short introduction to software configuration management. *Open Source Wesler*.

[Lewis et al., 2009] Lewis, G. A., Poernomo, I., and Hofmeister, C. (2009). Component-based software engineering. In *12th International Symposium, CBSE*, pages 24–26.

[Robles, 2004] Robles, G. (2004). A software engineering approach to libre software. *Open Source Jahrbuch*, 2004.

Ringraziamenti

. . .
. . .

Grazie a tutti per la lettura.