Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi – Febbraio 2016

1. PER GLI STUDENTI DI SISTEMI OPERATIVI E RETI: è necessario sostenere e consegnare entrambi gli esercizi. Sarà attribuito un unico voto su tutta la prova.

2. PER GLI STUDENTI DI SISTEMI OPERATIVI: si può sostenere solo uno dei due esercizi se si è già superata in un appello precedente la corrispondente prova. Il tempo a disposizione in questo caso è di 2 ORE.

Troverete sul vostro Desktop una cartella chiamata "CognomeNomeMatricola" che contiene la traccia dell'elaborato ed eventuali altri file utili per lo svolgimento della prova. Ai fini del superamento della prova è indispensabile rinominare tale cartella sostituendo "Cognome" "Nome" e "Matricola" con i vostri dati personali. Ad esempio, uno studente che si chiama Alex Britti ed ha matricola 66052 dovrà rinominare la cartella "CognomeNomeMatricola" in "BrittiAlex66052".

Per il codice Java, **si consiglia di raggruppare tutto il proprio codice in un package dal nome "CognomeNomeMatricola",** secondo lo schema usato per rinominare la cartella "CognomeNomeMatricola".

Non saranno presi in considerazione file non chiaramente riconducibili al proprio autore. E' possibile caricare qualsiasi tipo di materiale didattico sul desktop nei primi 5 minuti della prova.

Per il codice Java, **si DEVE raggruppare tutto il proprio codice in un package dal nome "CognomeNomeMatricola",** secondo lo schema usato per rinominare la cartella "CognomeNomeMatricola".

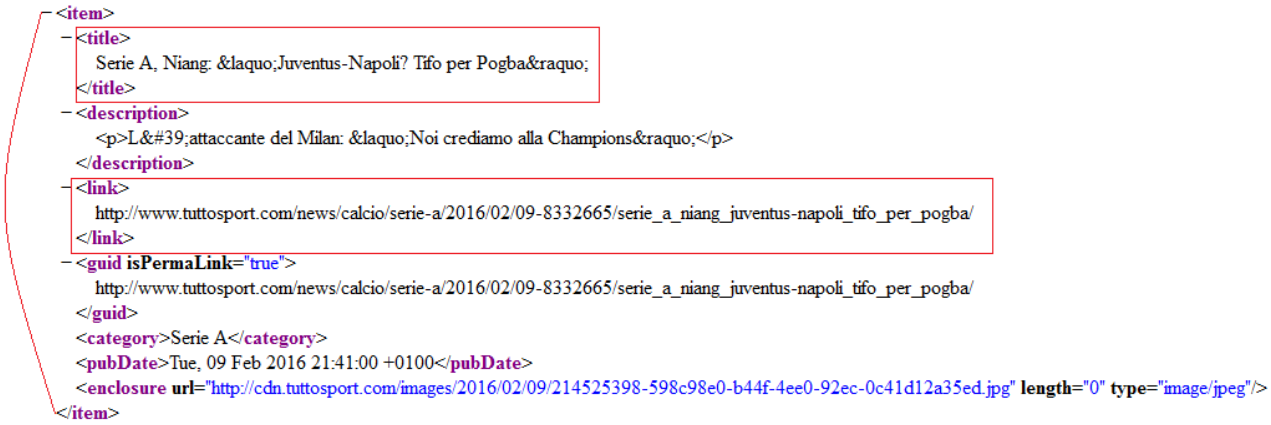
Non saranno valutate prove d'esame il cui codice è stato messo nel package di default.

Non saranno presi in considerazione file non chiaramente riconducibili al proprio autore. E' possibile caricare qualsiasi tipo di materiale didattico sul desktop nei primi 5 minuti della prova.

Si consiglia di salvare SPESSO il proprio lavoro.

ESERCIZIO 1 (Linguaggi di scripting. Punti 0-10)

La directory *rss* contiene dei feed RSS in formato xml provenienti dai siti di *gazzetta dello sport* e *corriere dello sport*. Come si vede dalla figura, un feed rss definisce diversi oggetti o *item*, racchiusi tra i tag xml `<item>…</item>`. Lo script da implementare deve essere in grado di estrarre le informazioni contenute nei tag `<title>` e `<link>` sulla base di un criterio di ricerca specificato dall'utente da linea di comando.



Più precisamente, si deve scrivere uno script perl capace di raccogliere alcune informazioni da questi file e stamparle su un nuovo file, *output.txt*. A tal scopo lo script:

- riceve da linea di comando una stringa, `stringa_da_cercare`, ad esempio "Sampdoria", "Milan", "Juve", "Serie A", che corrisponde all'informazione target rispetto alla quale aggregare le informazioni provenienti dai feed rss
- legge i file contenuti nella directory *rss* e per ciascun file:
    - individua se nel tag xml `<title>…</title>` è contenuta la stringa `stringa_da_cercare`
    - se presente, significa che si tratta di una informazione di interesse per l'utente, e quindi lo script dovrà stampare sul nuovo file il contenuto del tag `<title>` ed il contenuto del relativo tag `<link>` che indica il link in cui è reperibile il dettaglio sulla notizia
    - se non presente, vuol dire che l'informazione non è pertinente alla ricerca che l'utente intende effettuare e quindi va ignorata e non riportata nel file *output.txt*

Si assume che i tag `<title>`, `</title>`, `<link>` e `</link>` si trovino sempre su linee isolate.

Ad esempio, invocare lo script come:

perl script.pl "Serie A"

implica che sarà necessario trovare nei feed rss i tag <title> che contengano la stringa "Serie A" e stampare il contenuto del tag `<title>` e del tag `<link>` nel file *output.txt*. Perciò, al termine il file *output.txt* dovrebbe essere del tipo:

```
Serie A Chievo, Floro Flores: «Spero di segnare al Sassuolo»
http://www.corrieredellosport.it/news/calcio/serie-a/chievo/2016/02/09-
8331143/serie_a_chievo_floro_flores_spero_di_segnare_al_sassuolo_/


Serie A Palermo, a parte Maresca-Balogh
http://www.corrieredellosport.it/news/calcio/serie-a/palermo/2016/02/09-
8331121/serie_a_palermo_a_parte_maresca-balogh/

La Serie A non parla italiano Gli stranieri sono il 57,9%
http://www.gazzetta.it/Calcio/Serie-A/09-02-2016/serie-non-parla-italiano-solo-
inghilterra-belgio-hanno-piu-stranieri-140612700867.shtml
```

ESERCIZIO 2 (Programmazione multithread. Punti: 0-20)

Un ThreadPool è una struttura dati pensata per gestire un gruppo di Thread, ai quali possono essere assegnati determinati compiti da eseguire, detti Job. Un Job implementa l'interfaccia Java Runnable, dunque prevedendo l'esistenza del metodo `void run()`. Per "*eseguire un job J*" si intende invocare il metodo "J.run()".

Un Job può essere sottomesso a un ThreadPool per l'esecuzione. Il ThreadPool si deve occupare di eseguire il job il prima possibile e nell'ordine in cui è stato sottomesso, in accordo alla disponibilità di Thread liberi.

Lo studente deve implementare una classe ThreadPool che metta a disposizione i seguenti metodi pubblici, e che consenta a chi utilizzerà la classe, di sottomettere dei Job (o sottoclassi di questo) il cui metodo run() è arbitrariamente definito. Ogni metodo deve essere thread-safe.

`ThreadPool(int nThread)`. Crea un ThreadPool, con *nThread* thread pronti all'esecuzione di eventuali Job sottomessi.

`void submit(Job J)`. Accoda un Job J per l'esecuzione ed esce subito.

`void waitForStart(Job J)`. Il thread che chiama waitForStart si blocca fintanto che l'esecuzione di J non risulta effettivamente iniziata.

`void waitForFinish(Job J)`. Il thread che chiama waitFor si blocca fintantochè J non risulta eseguito e concluso.

`void pause()`. Sospende l'esecuzione di ulteriori job (i job correntemente in esecuzione non vengono interrotti, ed è possibile continuare a invocare il metodo submit).

`void waitForAll()`. Il thread che chiama waitForAll si blocca finchè tutti i job attualmente in esecuzione non terminano.

`void resume()`. Riprende l'esecuzione normale dei job.

`void HurryAndStop()`. Sospende l'utilizzabilità del metodo submit; esegue tutti i Job già sottomessi il prima possibile; infine mette in pausa il pool ed esce nel momento in cui tutti i job terminano. Se il metodo submit dovesse essere usato durante l'esecuzione di una HurryAndStop, è necessario sollevare un'eccezione.

BONUS: velocizzare il metodo HurryAndStop creando ulteriori thread (fino a uno per ogni Job in attesa) nel caso in cui il numero di thread predefiniti non dovessero bastare.

***E' parte integrante** di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo o estendendo tutte le strutture dati che si ritengano necessarie, e risolvendo eventuali ambiguità. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.*

Si può svolgere questo esercizio in un qualsiasi linguaggio di programmazione a scelta dotato di costrutti di supporto alla programmazione multi-threaded (esempio, C++ con libreria JTC, Java). E' consentito usare qualsiasi funzione di libreria di Java 6 o successivi, **AD ESCLUSIONE DELLE INTERFACCE EXECUTOR, THREADPOOL e LORO CLASSI DERIVATE**. Non è esplicitamente richiesto di scrivere un main() o di implementare esplicitamente del codice di prova, anche se lo si suggerisce per testare il proprio codice prima della consegna.