SCENARIO
Ci sono più “componenti software”, visti momentaneamente come file o interi package, ma in prospettiva Clever con plugin e progetti java separati, che vogliono partecipare al processo di creazione dinamica del file di configurazione di Log4J che gestirà i logger dichiarati al loro interno.

Ognuna di queste componenti prevede per default, in un predeterminato percorso, la presenza di un previsto numero di file xml, che da qui in poi chiamerò “frammenti”, che serviranno a comporre il mosaico finale rappresentato dal file di configurazione globale.

Questi frammenti xml devono essere opportunamente riempiti da colui che ha rilasciato il componenete software, seguendo sia le regole di sintassi previste da Log4J che le policy desiderate per il logging di quel componente.

PILOTA
In questa simulazione abbiamo la presenza dei seguenti “componenti software”:

LogClass.java rappresenta il **master** (futuro Clever Core).
LogClass1.java rappresenta lo **slave1** (futuro Agent 1 di Clever).
LogClass2.java rappresenta lo **slave2** (futuro Agent 2 di Clever).
Logclass.slave/LogClass3.java rappresenta lo **slave3** (futuro Agent 3 di Clever).

All'interno di ognuno di questi file sono previste delle politiche di logging.

Es: stralci di codice

```
private static org.apache.log4j.Logger log0 = Logger.getLogger("com.foo");
```

```
log0.trace("Trace Message! ");
log0.debug("Debug Message!");
log0.info("Info Message!");
log0.warn("Warn Message!");
log0.error("Error Message!");
log0.fatal("Fatal Message! ");
```

LogClass.java contiene la classe principale, gli altri 3 contengono ciascuno le classi che vengono richiamate all'interno della classe principale. In particolare LogClass3.java si trova all'interno del package *logclass.slave*.

La struttura del progetto, oltre alle componenti create di default, ha i seguenti elementi:

master/ rootLogger.xml
appender.xml
logger.xml

slave1/ rootLogger.xml
appender.xml
logger.xml

slave2/ rootLogger.xml
appender.xml
logger.xml

slave3/ rootLogger.xml
appender.xml
logger.xml

Ciascuna directory deve essere pensate come la directory residente su ciscun componente software preposta a contenete i frammenti. E' prevista la presenza di 3 frammenti, di cui più avanti verrà fornita descrizione.

Ipotesi

Ciascun componente software (master o slave N), per consentire un corretto svolgimento del processo di creazione dinamico del file di configurazione di Log4J, deve avere i 3 files elencati sopra.
La struttura interna ci ciascun file dovrà essere “curata” da chi ha progettato il componente.
Ciascun file è un “frammento” che sarà utilizzato per comporre dinamicamente, come una sorta di mosaico, il file di configurazione di Log4J del componente master.
Le regole di sintassi da seguire per costruire dei frammenti adeguati sono quelle della documentazione di Log4J.

Breve descrizione per ciascun file: (regole di Log4J da seguire)

Per ciascun frammento appender (appender.xml), ogni tag <*appender*> deve avere un name univoco all'interno del file, e in una visione più ampia, all'interno di più frammenti (cioè più file appender.xml) per più componenti software. (Questo significa che per ciascun appender dovrà essere un nome “ragionato”)
Il valore del campo value, all'interno del tag <*param*> di ciascun <*appender*> momentanemanete contiene il percorso interno al progetto Pilota dove risiede l'albero di directory atte a contenere i file di log.
(in questo caso /LOGS)

Es: stralci di codice

```
<appender name="FILE" class="org.apache.log4j.FileAppender">
  <param name="file" value="LOGS/ logclass1_output/SOLO_INFO.txt"/>
  <layout class="org.apache.log4j.PatternLayout" >
      <param name="ConversionPattern" value="%5p [%t] (%F:%L) - %m%n"/>
  </layout>
  <filter class="org.apache.log4j.varia.LevelMatchFilter">
    <param name="LevelToMatch" value="INFO" />
    <param name="AcceptOnMatch" value="true" />
   </filter>
   <filter class="org.apache.log4j.varia.DenyAllFilter" />
</appender>
.....................................
```

Bisogna assicurarsi che in ogni frammento logger.xml, tutti gli <*appender-ref*> presenti all'interno dei tag <*logger*> si riferiscano ad un valore di ref che corrisponde ad un appender **esistente** sul proprio frammento appender.xml di riferimento.

Es: stralci di codice

```
<logger name="pippo" additivity="false">
```

```
        <level value="debug" />
        <appender-ref ref="FILE" />
        ….....................................
        <appender-ref ref="FILEn" />
</logger>
….....................
```

Per ciascun <*appender-ref*> presente sul frammento rootLogger.xml, bisogna assicurarsi che vi sia come valore un appender **esistente** sul proprio frammento appender.xml.
L'elenco degli <*appender-ref*> deve contenere una entry per ogni appender presente sul proprio frammento appender.xml

Es: stralci di codice

```
<appender-ref ref="FILE"/>
….....................................
<appender-ref ref="FILEn"/>
```

Il processo di creazione del file di configurazione è contenuto nella routine **creaFileConfigurazioneLog()** che parte all'avvio del pilota contenuto in Logclass.java.
Questa routine è stata pensata per essere richiamata agilmente durante il run time.
Nel pilota viene richiamata una seconda volta, dopo la prima in fase di avvio, per aggiungere un nuovo componente software di cui gestire il logging in modo centralizzato e per simulare l'aggiunta di un futuro agente Agent per Clever.

Dentro questa routine, per prima cosa, dati in input un vettore di path, dove ogni path si riferisce ad una directory di un dato componente software che si vuole far partecipare al processo di composizione del file di configurazione di log4j, viene eseguito un controllo preliminare su ciascun path, in modo da verificare l'esistenza di tutti e i file necessari per il processo di composizione.
Da qui in poi chiamerò questa operazione “validazione”.

Non viene effettuato nessun controllo sul contenuto dei 3 file xml richiesti, la responsabilità sulla loro composizione come previsto nei prerequisiti è lasciata a ha chi ha scritto il componenete software.

**********OBSOLETO PILOTA 1
Ogni path il cui controllo abbia dato esito positivo viene posto all'interno di un nuovo vettore di path che verrà da qui in poi considerato come input di riferimento.
I path che non hanno dato esito positivo vengono di conseguenza esclusi e a video appare un messaggio di errore che segnala il path escluso e il/i file mancanti.

**********PILOTA 2
Viene eseguito un controllo sul vettore di path in input, per le componenti che non risultano “validate” (ossia che mancano di qualche frammento), è previsto un meccanismo che crea di default dei frammenti “generici” che consentono a tutti i componenti software di partecipare al processo di creazione del file di log. (in fase di sviluppo)

Una volta ottenuto un vettore di path regolari, questo viene usato in input per il processo di creazione vero e proprio.

Allinterno di **componiConfLog()** oltre a delle variabili che contengono dei frammenti per la “testa” e la “coda” del file di configurazione finale e la “testa” e la “coda” del frammento rootLogger, sono implementati in modo modulare i sengueti metodi:

String Appenders<- **componiAppConf**(vettore componenti software validati); //processo che unisce tutti gli appender.xml
String Loggers<- **componiLogConf**(vettore componenti software validati);//processo che unisce tutti i logger.xml
String rootLogger<- **componirootLogConf**(vettore componenti software validati);//processo che unisce tutti i rootLogger.xml

Ciascuno restituire una stringa che contiene una componente del file di configurazione finale.
A questo punto tutte le stringhe interessate vengono sommate opportunamente in modo da avere l'intero file di configurazione di log all'interno di una stringa.
Questa stringa viene salvata alla locazione che si è deciso deve essere occupata dal file di configurazionei in questione (log4jConfigFile).
Una volta creato il file e riempito dalla stringa completa, questo viene passato al metodo **DOMConfigurator.configure()** che si occupa di avviare il logging con Log4J.
Quando si aggiorna il file di configurazione a run time, bisogna resettare la configurazione attiva in Log4J col seguente comando **LogManager.resetConfiguration()** e poi ricaricare il nuovo file con la funzione **DOMConfigurator.configure()**.

Per esigenze operative, prima di uscire dal run time del Pilota, il file di configurazione viene cancellato (codice in coda al main).

Vengono creati i sengueti file di log:

/LOGS/logclass_output/DEBUG.txt
/SOLO_INFO.txt
/Threshold_ERROR_log.txt

logclass1_outputDEBUG.txt
/SOLO_INFO.txt
/Threshold_ERROR_log.txt

logclass2_output/DEBUG.txt
/SOLO_INFO.txt
/Threshold_ERROR_log.txt

logclass3_output/DEBUG.txt
/SOLO_INFO.txt
/Threshold_ERROR_log.txt

Le seguenti classi/componenti software scrivono sui file appena elencati seguendo il codice colore (comunque questo ha a che fare con il contenuto dei frammenti xml)
LogClass3
LogClass2
LogClass
LogClass1