Programmazione Distribuita II

A.A. 2011/12

Assignment n. 2

Tutto il materiale necessario per questa esercitazione si trova nel file .zip dal quale questo documento è stato estratto. Per lo svolgimento dell'esercitazione si consiglia di estrarre tutto l'archivio .zip in una cartella vuota, all'interno della quale si lavorerà.

Una volta estratto l'archivio nella cartella che chiameremo `[root]`, copiare il file `wfInfo.dtd` (quello sottomesso come soluzione dell'assignment 1) nella cartella `[root]/dtd` già predisposta.

L'assignment si compone di due parti:

1. Scrivere in *Java* un'applicazione per serializzare (cioè scrivere) i dati relativi ad un insieme di workflow in un file *XML* **valido** rispetto alla *DTD* progettata nell'assignment 1. L'applicazione deve essere sviluppata interamente nel package `it.polito.pd2.WF.sol2` e la classe principale deve chiamarsi `WFInfoSerializer`. Il file *XML* creato dall'applicazione deve contenere il riferimento ad una *DTD* esterna chiamata `wfInfo.dtd`, assumendo che tale file si trovi nella stessa cartella dove si trova il file *XML* generato. Il file *XML* generato deve contenere tutte le informazioni che è possibile estrarre tramite le interfacce definite nel package `it.polito.pd2.WF`. L'implementazione delle interfacce da usare come sorgente dei dati deve essere selezionata tramite il pattern "abstract factory": l'applicazione `WfInfoSerializer` deve creare la sorgente dei dati istanziando una `WorkflowMonitorFactory` tramite il metodo statico `newInstance()`. In pratica, il lavoro può essere svolto modificando l'applicazione *Java* `it.polito.pd2.WF.lab2.WFInfo` (fornita come codice sorgente), in modo che scriva i dati in un file *XML* valido rispetto alla propria *DTD* invece che scriverli su standard output (si noti che può anche essere necessario modificare l'ordine con cui `WFInfo` accede ai dati).
   L'applicazione `WFInfo` può essere lanciata dalla cartella `[root]` tramite il comando:

   ```
   $ ant WFInfo
   ```

   Lanciando l'applicazione in questo modo, viene usata una sorgente di dati pseudo casuale inclusa in un file jar fornito con il materiale dell'esercitazione, che genera un solo workflow ed un numero variabile di processi. E' possibile cambiare il comportamento del generatore pseudo casuale passando ulteriori parametri sulla linea di comando di ant, come nel seguente esempio:

   ```
   $ ant -Dseed=X –Dtestcase=Y WFInfo
   ```

   dove X è il seme del generatore pseudo casuale e Y è 0 (il default) oppure 1 (che significa generare 4 workflow e un numero variabile di processi) oppure 3 (0 workflow e 0 processi).

   L'applicazione `WFInfoSerializer` deve ricevere come parametro sulla linea di comando il nome del file *XML* da generare. Tutti i sorgenti dell'applicazione devono essere memorizzati nella cartella `[root]/src/it/polito/pd2/sol2/`. Lo script *ant* fornito con il materiale può essere usato anche per compilare e lanciare l'applicazione sviluppata. Per compilare, usare il comando

   ```
   $ ant build
   ```

   mentre per lanciare l'applicazione usare il comando

   ```
   $ ant -Doutput=file.xml WFInfoSerializer
   ```

   dove `file.xml` è il nome del file di uscita scelto. Anche qui si possono inserire i parametri `seed` e `testcase` per controllare il generatore pseudo casuale (impostare il seme è utile per garantire la ripetibilità degli esperimenti durante il debugging). Si noti che se questi comandi falliscono significa che probabilmente non sono state seguite correttamente le specifiche di questo documento.

Nota: Se si usa un *transformer XSLT* per scrivere il file *XML*, si noti che di default la dichiarazione *doctype* non viene scritta. Per scriverla, bisogna impostare la proprietà `OutputKeys.DOCTYPE_SYSTEM` del *transformer* al nome del file contenente la *DTD.*

2. Scrivere, usando l'interfaccia *DOM*, una libreria *Java* che possa essere usata per caricare e validare un file *XML* come quello generato dall'applicazione sviluppata nel primo punto di questo assignment. La libreria deve essere sufficientemente robusta per poter essere usata all'interno di un server di rete: deve considerare "non affidabile" il documento di input, e non deve mai lanciare eccezioni di runtime (come per esempio NullPointerException). La libreria deve implementare tutte le interfacce e le classi astratte definite nel package `it.polito.pd2.WF`, restituendo i dati caricati dal file. La libreria deve essere scritta interamente nel package `it.polito.pd2.WF.sol2` e tutti i suoi sorgenti devono essere memorizzati nella cartella `[root]/src/it/polito/pd2/WF/sol2/`. La libreria deve includere una classe factory con nome `WorkflowMonitorFactoryImpl`, che estenda la factory astratta `it.polito.pd2.WF.WorkflowMonitorFactory` e, tramite il metodo `newWorkflowMonitor()`, crei un'istanza della vostra classe che implementa l'interfaccia `it.polito.pd2.WF.WorkflowMonitor`. Il nome del file *XML* di input deve essere ottenuto leggendo la proprietà di sistema `it.polito.pd2.WF.sol2.WFInfo.file`.
   Per compilare la libreria si usi il comando:

   ```
   $ ant build
   ```

   Se questo fallisce significa che non sono state seguite correttamente le istruzioni.

Il serializzatore e la libreria devono essere portabili e interoperabili anche quando eseguiti in ambiente distribuito (per esempio, devono evitare dipendenze dalla "time zone" locale e dalle altre impostazioni locali della macchina).

Verifica finale della correttezza

Prima di sottomettere la soluzione dell'esercizio siete tenuti a verificarne il corretto funzionamento e l'aderenza alle specifiche date. Perché sia accettabile ai fini dell'esame, la soluzione deve includere entrambe le parti e deve superare almeno i test automatici obbligatori. Si noti che i test automatici controllano solo alcuni dei requisiti funzionali! In particolare essi controllano che:

- l'applicazione `WFInfoSerializer` generi file XML well-formed e validi (facenti riferimento alla DTD sottomessa per l'assignment 1).
- i dati memorizzati dall'applicazione `WFInfoSerializer` nel file *XML* di uscita vengano caricati dalle classi della libreria sviluppata al punto 2 senza errori.
- la catena serializzazione + rilettura dei dati da parte della libreria non alteri i dati stessi (i dati estratti dalla libreria alla quale è stato passato un file generato dal serializzatore devono essere gli stessi che erano stati forniti al serializzatore per la generazione del file; per date e tempo è richiesta la precisione del minuto).

Altri controlli e valutazioni sul codice verranno fatti in sede d'esame (quindi superare tutti i test non garantisce necessariamente che la valutazione sarà ottima).
Le verifiche di cui sopra vengono fatte usando dati pseudocasuali generati dal generatore incluso nei file jar..
Il file *.zip* fornito per questo assignment include un insieme di test come quelli che verranno fati girare sul sistema di sottomissione. I test sono stati scritti usando il sistema *Junit* per test di unità. I sorgenti dei test sono disponibili nel file .zip, nel package `it.polito.pd2.WF.lab2.tests`.

Sono disponibili 5 diversi test case. In ogni caso i dati vengono generati in modo casuale usando un seme. I primi 2 test case testano il normale funzionamento della soluzione e non effettuano controlli sulle date (il test case 0 genera un solo workflow e il test case 1 genera 4 workflow). Il test case 2 è come il test case 1 ma effettua anche i controlli sulle date. Gli ultimi 2 test case controllano il comportamento in alcune condizioni eccezionali (il test case 3 genera liste di workflow e processi vuote e il test case 4 chiama metodi con dati malformati). Sul server, il test case 0 viene fatto girare due volte, con due semi diversi, mentre gli altri test case vengono fatti girare una sola volta. Passare i due run del test case 0 è obbligatorio ai fini dell'esame, mentre passare gli altri test è opzionale.

Per far girare localmente sulla propria macchina uno dei primi 4 test case, si può usare il seguente comando:

```
$ ant –Dtestcase=X –Dseed=Y runFuncTest
```

dove `X` è il numero del test case (0, 1, 2 o 3) e `Y` è il seme per il generatore pseudo-casuale.

Per far girare invece l'ultimo test case si può usare il comando:

```
$ ant –Dseed=Y runExcTest
```

dove `Y` è nuovamente il seme per il generatore pseudo-casuale.

In entrambi i casi, se non viene specificato alcun seme, viene usato un seme scelto in base all'ora corrente.

Prima di provare i test finali è bene testare l'applicazione per conto proprio, usando il generatore di dati fornito (comando ant con target `WFInfoSerializer`) . Ricordate che i test automatici sono parziali e che all'esame verranno fatte ulteriori valutazioni sulle soluzioni presentate.

Formato del file da sottomettere

Occorre sottomettere un unico file *.zip*, contenente tutti i file prodotti. Il file *.zip* può essere prodotto con il seguente comando (dalla cartella `[root]`):

```
$ jar -cf lab2.zip src/it/polito/pd2/WF/sol2/*.java
```