Programmazione Orientata agli Oggetti

Il Framework ping
Conclusioni

versione 2.3

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Framework e Refactoring
- Configurazione Procedurale
- Annotazioni per il Binding
- Integrazione con l'Ambiente RAD

Framework e Refactoring

- Framework e refactoring
  - ⇨un rapporto tradizionalmente “difficile”
- Il problema
  - ⇨gli strumenti per il refactoring agiscono esclusivamente sul codice sorgente
  - ⇨i framework però tendono ad utilizzare file di configurazione (tipicamente in formato XML) in cui il refactoring browser non lavora

Framework e Refactoring

- Come conseguenza
  - ⇨effettuando refactoring nel codice (es: ridenominazioni delle classi o degli attributi), è necessario cambiare manualmente i valori nel file di configurazione
  - ⇨il che è frequentemente sorgente di errore
  - ⇨per queste ragioni, in vari contesti l’utilizzo di file di configurazione XML comincia ad essere messo in discussione

Framework e Refactoring

- Nel caso di ping
  - ⇨incoraggiare il refactoring è uno degli obiettivi dichiarati del framework
  - ⇨il framework offre modalità di lavoro che lo rendono pienamente compatibile con l'utilizzo di un refactoring browser
  - ⇨configurazione procedurale
  - ⇨utilizzo di annotazioni per il binding

Configurazione Procedurale

- Configurazione dell'applicazione
  - ⇨è possibile farla in due modi
  - ⇨attraverso il file /ping-config.xml
  - ⇨attraverso la classe conf.PingConfig
- In altri termini
  - ⇨all'avvio la classe Applicazione verifica se esiste una classe conf.PingConfig e se la trova la usa per la configurazione ignorando l'eventuale file ping-config.xml

Configurazione Procedurale

- La classe conf.PingConfig
  - ⇨viene usata per costruire l'oggetto che rappresenta la configurazione dell'applicaz.
  - ⇨deve estendere la classe Configurazione di ping
  - ⇨nel costruttore devono essere specificati i parametri di configurazione utilizzando l'API fornita dal framework

Configurazione Procedurale

- Quali parametri configurare
  - ⇨tutti i parametri relativi all'elemento pingConfig (nome applicazione, autori) e l'eventuale splash screen
  - ⇨tutti i parametri relativi all'elemento viste (vista principale, elenco delle sottoviste)
  - ⇨alcuni dei parametri relativi all'elemento azioni (azione iniziale, azione finale, registrazione degli osservatori)

Configurazione Procedurale

- L'interfaccia IConfigurazione

```
void   setNomeApplicazione(String nomeApplicazione)
void   setAutore(String autore)

void   setNomeFileSplashScreen(String nomeFileSplashScreen)

void   setVistaPrincipale(String vistaPrincipale)
void   addSottoVista(String idVista)

void   setAzioneFinale(String azioneFinale)
void   setAzioneIniziale(String azioneIniziale)

void   setRegistraOsservatori(boolean registraOsservatori)
```


Configurazione Procedurale

- Attenzione
  - ⇨non è necessario descrivere le singole azioni e i parametri per le azioni Swing
- Per questo
  - ⇨viene utilizzato un altro strumento
  - ⇨è possibile utilizzare una serie di annotazioni di Jdk 1.5 fornite dal framework per descrivere i parametri direttamente nel file di codice sorgente dell'azione ping

Configurazione Procedurale

- Le annotazioni di ping per le azioni
  - ⇨sono annotazioni della classe
  - ⇨hanno tutte un valore
  - ⇨@NomeSwing (valore di tipo String)
  - ⇨@DescrizioneSwing (valore di tipo String)
  - ⇨@AcceleratoreSwing (valore di tipo String)
  - ⇨@MnemonicoSwing (valore di tipo String)
  - ⇨@IconaSwing (valore di tipo String)
  - ⇨@DisabilitataAllAvvio (valore di tipo Boolean)

Configurazione Procedurale

>> volpiEConigli

- Un esempio
  - ⇨Volpi e Conigli
  - ⇨viene usato il meccanismo di configurazione procedurale
  - ⇨con le annotazioni per la configurazione delle azioni Swing

Configurazione Procedurale

- Vantaggi di questo approccio
  - tutto il contenuto dell'applicazione è descritto nei file di codice sorgente
  - eventuali operazioni di refactoring si ripercuotono sistematicamente su tutta l'applicazione
  - nella scrittura è possibile utilizzare il completamente automatico fornito dall'IDE
  - la classe di configurazione ha un contenuto ridotto e la descrizione delle azioni è concentrata in un unico file

Annotazioni per il Binding

- Nonostante la configurazione procedurale
  - restano alcuni aspetti del funzionamento del framework che sono fragili rispetto al refactoring
- In particolare
  - tutti i riferimenti basati su stringhe
  - non possono essere verificati a tempo di compilazione
  - e non vengono gestiti dal refactoring browser

Annotazioni per il Binding

- I principali riferimenti
  - ⇨riferimenti agli id delle azioni e delle viste
  - ⇨riferimenti ai nomi dei bean nel modello
  - ⇨riferimenti alle proprietà dei bean
  - ⇨riferimenti ai nomi dei componenti nella vista
  - ⇨riferimento ai nomi degli schermi della vista
- Id delle azioni e delle viste
  - ⇨il framework utilizza il nome della classe corrispondente e questo li rende robusti

Annotazioni per il Binding

- Nomi dei bean nel modello
  - ⇨il suggerimento è utilizzare costanti simboliche basate sul nome della classe
- Nota
  - ⇨non è sempre possibile utilizzare il nome della classe, nei casi in cui due bean dello stesso tipo devono stare contemporaneamente nel modello
  - ⇨in questo caso definire due costanti diverse

Annotazioni per il Binding

- Nomi delle proprietà dei bean
  - ⇨questo è l'aspetto più critico, visto che i nomi delle proprietà sono usate molto frequentemente per il binding e sono spesso oggetto di refactoring
- Per semplificare i riferimenti
  - ⇨ping consente di creare un livello di indirezione tra il nome della proprietà e i riferimenti utilizzati nel binding

Annotazioni per il Binding

- Annotazione per il binding
  - ⇨@BindingPing (valore di tipo String)
  - ⇨è un'annotazione di metodo, da utilizzare sul metodo get di una proprietà di un bean
  - ⇨consente di attribuire alla proprietà un nome utilizzabile nel binding indipendente dal nome reale della proprietà
  - ⇨in caso di ridenominazione della proprietà è sufficiente non cambiare il nome per il binding e il funzionamento dell'applicazione è preservato

Annotazioni per il Binding

- I passi per usare l'annotazione
  - annotare i metodi get delle proprietà per il binding con il nome di binding, usando costanti simboliche
  - nella definizione degli osservatori e dei collegatori, utilizzare il valore del nome di binding invece che il nome reale della proprietà
  - NOTA: in questo caso è necessario specificare nel costruttore anche il tipo del bean in cui cercare l'annotazione, passando il riferimento all'oggetto class
  - in seguito evitare di cambiare il nome di binding

Annotazioni per il Binding

- I costruttori di un osservatore

```
OsservatoreLabel()
OsservatoreLabel(JComponent componente, String nomeBean, String nomeProprieta)
OsservatoreLabel(JComponent componente, String nomeBean,
                 Class classeBean, String nomeBinding)
OsservatoreLabel(String nomeComponente, String nomeBean, String nomeProprieta)
OsservatoreLabel(String nomeComponente, String nomeBean,
                 String nomeProprieta, Format format)
OsservatoreLabel(String nomeComponente, String nomeBean,
                 Class classeBean, String nomeBinding)
OsservatoreLabel(String nomeComponente, String nomeBean,
                 Class classeBean, String nomeBinding, Format format)
```

Annotazioni per il Binding

>> appuntamenti

- Nota
  - ⇨in questo modo i nomi per il binding possono essere anche valori sintetici (es: numerici)
- Un esempio
  - ⇨appuntamenti
  - ⇨tutto il binding viene fatto utilizzando le annotazioni

Annotazioni per il Binding

- I riferimenti restanti
  - ⇨riferimenti ai nomi dei componenti
  - ⇨riferimenti ai nomi degli schermi
- Riferimenti ai nomi degli schermi
  - ⇨sarebbe possibile definire annotazioni anche per questi ma utilizzando il binding sono usati molto raramente
  - ⇨è sufficiente fare attenzione e, se il valore viene usato più di una volta nel codice, utilizzare costanti simboliche

Annotazioni per il Binding

- Riferimenti ai nomi dei componenti
  - ⇨sono definiti alla creazione del componente
  - ⇨e poi utilizzati nel codice della vista per modificarne il comportamento
  - ⇨l'uso è estremamente localizzato, per cui la manutenzione è semplificata
  - ⇨è opportuno utilizzare una convenzione (es: tipo di componente e descrizione, come "labelNome")
  - ⇨normalmente non è necessario definire costanti simboliche

Integrazione con l'Ambiente RAD

- Nonostante questi piccoli svantaggi
  - ⇨l'utilizzo sistematico dei nomi per il riferimento ai componenti ha il grande vantaggio di consentire di utilizzare facilmente ambienti RAD
- In generale
  - ⇨può essere utilizzato qualsiasi ambiente che consenta di definire pannelli

Integrazione con l'Ambiente RAD

- Idea
  - ⇨definire il pannello con l'ambiente RAD
  - ⇨creare l'oggetto di tipo JPanel in una vista di ping e aggiungerlo alla vista nel metodo di inizializzazione della vista
- Esempi di ambienti RAD
  - ⇨Abeille Forms Designer
  - ⇨NetBeans Matisse GUI Builder

Integrazione con l'Ambiente RAD

- Abeille Forms Designer
  - ⇨strumento completamente esterno all'IDE
  - ⇨è basato su un gestore di layout fisso: JGoodies Forms Layout
  - ⇨produce oggetti di tipo FormPanel (che estende JPanel) che forniscono un'API molto semplice per aquisire riferimenti ai componenti contenuti nel pannello

Integrazione con l'Ambiente RAD

- Per lavorare con Abeille
  - è necessario creare un progetto
  - specificando il riferimento alla cartella di progetto (viene creato un file di metadati con estensione .jpfr)
  - il riferimento alla cartella dei sorgenti
  - il riferimento alla cartella build\classes
  - nel caso in cui sia necessario caricare immagini, è necessario anche fornire il riferimento alla cartella a partire dalla quale sono specificati i riferimenti alle immagini (normalmente build\classes)

Integrazione con l'Ambiente RAD

- Successivamente
  - è possibile creare le form, ovvero i form panel
  - il contenuto del form panel viene salvato in un file binario con estensione .jfrm, normalmente nella cartella vista di src
  - oppure in un file .xml (sconsigliato perchè aumenta nettamente la dimensione del file e i tempi di caricamento)

Integrazione con l'Ambiente RAD

- Nota
  - ⇨Io strumento copia automaticamente tutti i file .jfrm nella cartella di build (build/classes) in modo che siano disponibili per il caricamento
  - ⇨per questo è necessario specificare il riferimento a build/classes
  - ⇨l'operazione però è inutile perchè in ogni caso viene effettuata dal file di build di ant (ping-template-build)

Integrazione con l'Ambiente RAD

>> mediaPesata

- Nell'applicazione
  - ⇨l'oggetto di tipo FormPanel viene costruito passando al costruttore il riferimento al file .jfrm
  - ⇨è necessario che nel classpath sia visibile formsrt.jar

Integrazione con l'Ambiente RAD

- Per usare abeille
  - è indispensabile imparare ad utilizzare il form layout
  - essenzialmente un gestore di layout che consente di costruire tabelle nidificate
  - con numerosi vincoli aggiuntivi
  - i dettagli del funzionamento del gestore sono descritti in un articolo disponibile sul sito di JGoodies
  - altre informazioni sono disponibili nell'Help di abeille

Riassumendo

- Framework e Refactoring
- Configurazione Procedurale
- Annotazioni per il Binding
- Integrazione con l'Ambiente RAD

Termini della Licenza

- This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

- Questo lavoro viene concesso in uso secondo i termini della licenza “Attribution-ShareAlike” di Creative Commons. Per ottenere una copia della licenza, è possibile visitare http://creativecommons.org/licenses/by-sa/1.0/ oppure inviare una lettera all’indirizzo Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.