Programmazione Orientata agli Oggetti

Il Framework ping
Binding
Parte b

versione 2.0

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

Sommario

- Creazione degli Osservatori
- Presentation Model
- Collezioni Monodimensionali
- Collezioni Bidimensionali
- Gestione dello Stato

Creazione degli Osservatori

- Creazione di un osservatore
  - ⇨è necessario creare l'osservatore
  - ⇨assegnare il riferimento al componente
  - ⇨assegnare il nome del bean e della proprietà
  - ⇨registrare l'osservatore
- Esempio: morraCinese
  - ⇨viene fatto un uso estensivo degli osservatori nel pannello dei dati della partita

Creazione degli Osservatori

- Gli osservatori di ping
  - ⇨come tutti i componenti del framework sono JavaBeans
  - ⇨e quindi possono essere creati e manipolati secondo lo stile dei JavaBeans
  - ⇨usando il costruttore no-arg per la creazione
  - ⇨e i metodi set per modificare le proprietà

Creazione degli Osservatori

>> OsservatoreLabel

- Ma...
  - ⇨uno degli obiettivi del framework è ridurre il numero di linee di codice
- A questo scopo
  - ⇨per quasi tutti i componenti esistono vari costruttori con argomenti che consentono di effettuare più rapidamente la configurazione

Creazione degli Osservatori

- Due versioni
  - ⇨la versione in stile JavaBean
  - ⇨la versione compatta (registra autom. l'oss.)

```
JLabel labelPG = new JLabel();

IOsservatore ossPuntG = new OsservatoreLabel();
ossPuntG.setComponente(labelPG);
ossPuntG.setNomeBean(Costanti.GIOCO);
ossPuntG.setNomeProprieta(Costanti.PARTITE_GIOCATORE);
ossPuntG.registra();

IOsservatore ossPG = new OsservatoreLabel(labelPG,
        Costanti.GIOCO, Costanti.PARTITE_GIOCATORE);
```

Creazione degli Osservatori

>> morraCinese - PannelloPartita

- Una ulteriore variante (preferibile)
  - ⇨consente di specificare il nome del componente e non il riferimento
  - ⇨facilita l'integrazione con ambienti RAD per lo sviluppo della vista perchè non richiede di acquisire i riferimenti ai componenti grafici

```
IOsservatore ossPG = new OsservatoreLabel("labelPGiocatore",
        Costanti.GIOCO, Costanti.PARTITE_GIOCATORE);
```


Presentation Model

- Vantaggi del meccanismo di binding
  - ⇨snellisce il codice delle azioni e degli schermi (evita la sincronizzazione per copia)
  - ⇨consente facilmente di accoppiare viste diverse agli stessi oggetti della logica applicativa
- Ma...
  - ⇨questi vantaggi hanno un piccolo prezzo

Presentation Model

- Svantaggio del meccanismo di binding
  - ⇨è necessario modificare lo strato del modello per consentire la notifica degli eventi (i bean dipendono dalla classe Modello)
  - ⇨questo aumenta l'accoppiamento tra vista e modello
  - ⇨ma più ancora tra modello e framework
  - ⇨in generale, il modello dovrebbe essere indipendente dal resto dell'applicazione

Presentation Model

- Una possibile soluzione
  - ⇨adottare il pattern "presentation model"
- Presentation Model
  - ⇨strato di componenti intermedio tra la vista e il modello
  - ⇨incapsula i componenti del modello, fornendo tutti i metodi per la visualizzazione di valori nella vista

Presentation Model

- La notifica di eventi agli osservatori
  - ⇨è un tipico compito dello strato di presentation model
- Altro compito tipico
  - ⇨contenere tutti i metodi che trasformano valori del modello in modo che siano visualizzabili nella vista
  - ⇨es: giocate (numeri interi) nell'URI dell'icona corrispondente da visualizzare

Presentation Model

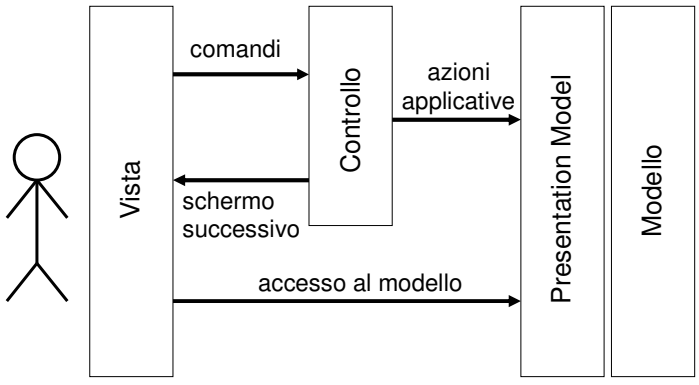
- Nell'applicazione
  - ⇨viene utilizzato uno strato di presentation model per evitare di modificare il modello rispetto all'applicazione console
- Componenti dello strato
  - ⇨GiocoPM
  - ⇨PartitaPM
  - ⇨ManoPM

Presentation Model

>> morraCinese

- Utilizzo dei value object
  - ⇨il binding viene fatto rispetto agli oggetti dello strato di presentation model, e non rispetto ai bean del modello
  - ⇨nelle azioni, vengono creati e modificati sempre gli oggetti di presentation model, e non i bean del modello
  - ⇨gli oggetti di presentation model utilizzano il modello per notificare gli eventi relativi ai cambiamenti delle proprietà

Presentation Model



Presentation Model

- Vantaggio di questo approccio
  - ⇨si riduce l'accoppiamento del modello con gli altri componenti dell'applicazione
- Svantaggio di questo approccio
  - ⇨cresce il numero di componenti e il numero di linee di codice dell'applicazione
- Il rapporto costo-benefici
  - ⇨è spesso sfavorevole

Presentation Model

- Nei progetti successivi
  - ⇨questo approccio non viene usato
  - ⇨il binding viene fatto direttamente sui componenti del modello
- Un esempio
  - ⇨mediaPesata
  - ⇨prima di discutere il codice vediamo alcuni aspetti interessanti

Presentation Model

- Un aspetto interessante dell'applicazione
  - ⇨viene utilizzata una "barra di stato"
- Barra di stato
  - ⇨componente dell'applicazione (normalmente in fondo al frame) che visualizza per ogni operazione un messaggio corrispondente all'esito dell'operazione effettuata
  - ⇨un JPanel con una JLabel che osserva un bean che incapsula una stringa

Presentation Model

- Supporto da parte del framework
  - ⇨il framework fornisce una classe MessaggioPing con una proprietà valore che può essere utilizzata per il binding dei messaggi
  - ⇨due costanti da usare per il nome del bean Controllo.MESSAGGIO_STATO, Controllo.MESSAGGIO_ERRORE
  - ⇨e una costante per il nome della proprietà: Controllo.VALORE_MESSAGGIO

Presentation Model

- Un altro aspetto interessante
  - ⇨nell'applicazione viene implementata un'azione applicativa di lunga durata (AzioneApriExcel)
  - ⇨utilizzando la classe PingThreadWorker
  - ⇨basata su SwingWorker
  - ⇨nel package it.unibas.ping.contrib

Presentation Model

>> mediaPesata – modello
>> mediaPesata – pannello princ.

- Un'ultima annotazione
  - ⇨viene fatta una distinzione tra azioni applicative e azioni sulla vista
- Azione sulla vista
  - ⇨azione che ha come unico effetto quello di modificare lo stato dei componenti della vista
  - ⇨può essere direttamente programmata nello strato della vista, senza implementare un'azione ping apposita

Collezioni Monodimensionali

- Binding di collezioni
  - ⇨il framework fornisce osservatori per il binding di collezioni
  - ⇨distinguendo tra collezioni monodimensionali e bidimensionali
- Binding di collezioni monodimensionali
  - ⇨binding tra proprietà multivalore (liste) nel modello e componenti JList o JTable

Collezioni Monodimensionali

- Binding per JList
  - ⇨molto simile a quello visto per le proprietà dei tipi di base
- OsservatoreLista
  - ⇨osservatore di una proprietà multivalore di un bean del modello
  - ⇨che aggiorna gli elementi di una JList
  - ⇨gestendone automaticamente il data model (crea un DefaultListModel)

Collezioni Monodimensionali

- OsservatoreTabellaMono
  - ⇨l'utilizzo è leggermente più complesso per via del fatto che è necessario specificare esplicitamente il TableModel
- ModelloTabellaPing
  - ⇨classe astratta che estende DefaultTableModel e può essere usata per costruire il modello di una tabella gestita con un osservatore

Collezioni Monodimensionali

- Di conseguenza
  - ⇨per effettuare binding su una tabella monodimensionale è necessario creare un table model, estendendo ModelloTabellaPing
  - ⇨a partire dal table model, è possibile creare l'osservatore

```
new OsservatoreTabellaMono(Costanti.JTABELLA_ESAMI,
    new ModelloTabellaEsami(), Costanti.STUDENTE, Costanti.LISTA_ESAMI);
```

Collezioni Monodimensionali

- Rispetto ad un table model ordinario
  - per rispettare il disaccoppiamento tra componente della vista e bean, il table model non mantiene un riferimento al bean
  - viceversa ne conosce il nome, e all'occorrenza acquisisce il riferimento eseguendo il metodo super.getBean();
  - è necessario però tenere in considerazione il fatto che i metodi del table model possono essere eseguiti anche quando il bean NON è presente nel modello, e quindi verificare sempre che bean != null

Collezioni Monodimensionali

>> mediaPesata

- La notifica degli eventi nel modello
  - si tratta di eventi di modifica collezione, per cui è necessario specificare l'intervallo di elementi coinvolti e la tipologia
  - Modello.AGGIUNTA, Modello.ELIMINAZIONE, Modello.MODIFICA
- Un indice particolare
  - Modello.ULTIMO_INDICE
  - rappresenta l'indice dell'ultimo elemento

Collezioni Bidimensionali

- Binding di collezioni bidimensionali
  - ⇨binding tra proprietà che sono array bidimensionali e componenti di tipo JTable
- OsservatoreTabellaBidim
  - ⇨molto simile al corrispondente monodimensionale
  - ⇨richiede anche in questo caso di sviluppare un table model

Collezioni Bidimensionali

>> volpiEConigli

- La notifica degli eventi nel modello
  - ⇨si tratta di eventi di modifica cella, per cui è necessario specificare le coordinate della cella
  - ⇨sono di un unico tipo (modifica) e non sono previste aggiunte o eliminazioni
- Una coordinata particolare
  - ⇨Modello.TUTTE_LE_CELLE
  - ⇨segnala all'oss. di aggiornare tutte le celle

Gestione dello Stato

- Gli osservatori
  - ⇨sono uno strumento portante del framework
  - ⇨che viene utilizzato anche per altri scopi
  - ⇨un esempio: la gestione dello stato
- Problema
  - ⇨frequentemente, nel codice di controllo, è necessario abilitare/disabilitare le azioni swing per abilitare/disabilitare componenti grafici

Gestione dello Stato

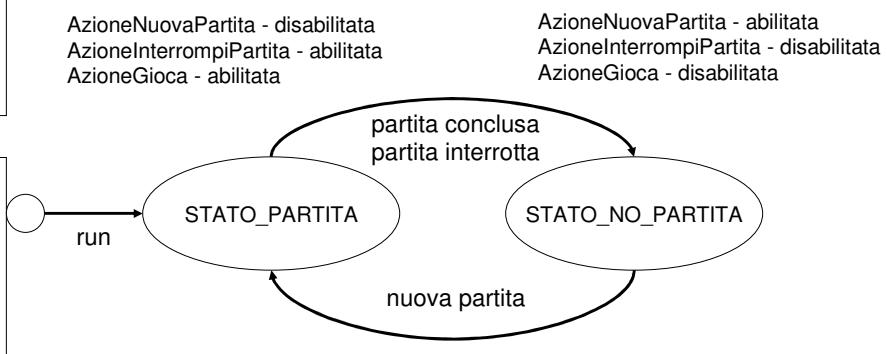
- Una possibile soluzione
  - ⇨rendere “dichiarativa” l’abilitazione e disabilitazione delle azioni
- Idea
  - ⇨ragionare sui possibili stati dell’applicazione
  - ⇨attribuire a ciascuno stato un id (es: intero)
  - ⇨in ciascuna azione, dichiarare in quali stati l’azione dovrebbe essere abilitata, e in quali disabilitata

Gestione dello Stato

- Esempio: la MorraCinese
  - ci sono due stati significativi nell'applicazione che influenzano lo stato dei componenti
  - lo stato in cui una partita è in corso, e quello in cui non ci sono partite in corso
- Nella classe Costanti identifico i due stati
  - public static int STATO_PARTITA = 0;
  - public static int STATO_NO_PARTITA = 1;

Gestione dello Stato

- Il diagramma degli stati



Gestione dello Stato

>> morraCinese - azioni

- Nell'applicazione
  - posso dichiarare in ciascuna azione di ping in quali stati l'azione deve essere abilitata e in quali disabilitata
  - attraverso i due metodi abilita e disabilita
  - NOTA: le implementazioni standard ereditate da AzionePingAstratta restituiscono false

```
public boolean abilita(Integer statusId);
public boolean disabilita(Integer statusId);
```


Gestione dello Stato

- Successivamente
  - vorrei poter segnalare i cambiamenti di stato dell'applicazione nelle azioni, lasciando che le azioni reagiscano autonomamente
- Il bean di stato di ping
  - è possibile utilizzare il bean predefinito it.unibas.ping.framework.StatoPing
  - il bean incapsula una proprietà intera che rappresenta l'id dello stato

Gestione dello Stato

- Per cambiare lo stato
  - ⇨nel codice delle azioni è sufficiente salvare nel modello una nuova istanza del bean di stato, utilizzando la chiave Controllo.STATO
- Esempio: in AzioneInterrompiPartita

```
public void esegui(EventObject object) {
   modello.putBean(Controllo.STATO, new StatoPing(Costanti.STATO_NO_PARTITA));
   modello.removeBean(Costanti.MANO);
}
```


Gestione dello Stato

- Il framework
  - ⇨utilizza il publish&subscribe per notificare alle azioni il cambiamento di stato (ovvero l'evento nel ciclo di vita del bean di stato)
  - ⇨per ciascuna azione il framework registra un osservatore del bean di stato
  - ⇨a seguito di variazioni del bean di stato, per gestire l'evento l'osservatore chiama i metodi abilita/disabilita dell'azione ping corrispondente e di conseguenza abilita o disabilita l'azione ping associata

Gestione dello Stato

>> morraCinese - azioni

- Un altro esempio: la media pesata
  - ci sono anche qui due stati
  - uno in cui sono aperti i dati di uno studente
  - uno in cui non c'è uno studente
- Nota
  - con lo stesso meccanismo avrei potuto gestire anche l'abilitazione delle azioni modifica e cancella sulla JList/JTable (che invece viene gestita come azione sulla vista con listener anonimi)

Gestione dello Stato

>> appuntamenti - azioni

- Esempio: appuntamenti
  - l'applicazione ha quattro stati
  - nessuna agenda aperta
  - agenda aperta (indipendentemente dalla selezione degli appuntamenti)
  - agenda aperta e appuntamento selezionato
  - agenda aperta ma nessun appuntamento selezionato
  - è istruttivo disegnare il diagramma di stato

Gestione dello Stato

- La gestione dello stato
  - è una caratteristica molto istruttiva del framework perchè incoraggia a pensare all'applicazione in termini di stati e transizioni tra stati
  - inoltre snellisce significativamente il codice delle azioni e riduce gli errori (es: abilitazione o disabilitazione dimenticata)

Gestione dello Stato

- Ma ha un prezzo
  - tende a far crescere il numero di osservatori e il numero di eventi nel modello
  - abilitando il logging nel framework è possibile facilmente vedere il numero di eventi generati
  - di conseguenza è possibile disabilitarla in sede di configurazione, utilizzando l'attributo registraOsservatori="false" dell'elemento azioni

Riassumendo

- Creazione degli Osservatori
- Presentation Model
- Collezioni Monodimensionali
- Collezioni Bidimensionali
- Gestione dello Stato

Termini della Licenza