Programmazione Orientata agli Oggetti

Il Framework ping
Form

versione 2.0

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

Sommario

- Form
- Convalidatori
- Collegatori
- Azione Commit e Azione Rollback
- Ciclo di Vita dell'Azione Commit
  - ⇨Creazione Automatica
  - ⇨Ciclo di Vita del Bean
- Wizard

Form

- Moltissimi framework
  - ⇨introducono un costrutto di form per gestire il binding bidirezionale
- Form di ping
  - ⇨oggetto di tipo it.unibas.ping.binding.IForm
  - ⇨ogni sottovista di ping ha una form associata che può essere usata per effettuare operazioni di immissione
  - ⇨sono composte di collegatori

Form

- Collegatore
  - ⇨oggetto di tipo ICollegatore
  - ⇨è associato ad un campo di immissione della vista, e ad una proprietà di un bean del modello
  - ⇨ha il compito di copiare il valore del campo di immissione nella proprietà del modello
  - ⇨può avere un convalidatore di valore associato

Form

- Convalidatore di Valore
  - ⇨oggetto di tipo IConvalidatoreValore
  - ⇨ha il compito di verificare il valore del campo di immissione prima della copia nel modello
  - ⇨se il valore è corretto, ha anche il compito di convertirlo prima di copiarlo nel modello
  - ⇨al termine della convalida restituisce una lista di stringhe (messaggi di errore), eventualmente vuota

Form

- Convalidatori di ping
  - ⇨ne esistono di due tipologie
  - ⇨i convalidatori di valore lavorano sul valore di un unico campo
- Convalidatore di form
  - ⇨oggetto di tipo IConvalidatoreForm
  - ⇨è associato all'intera form e non ad un singolo campo e consente di effettuare convalide su più valori contemporaneamente

Form

- Struttura della form
  - ⇨una collezione di collegatori, con gli eventuali convalidatori di valori
  - ⇨una collezione di convalidatori di form
  - ⇨un “bottone commit” che esegue i collegatori
  - ⇨un “bottone rollback” (opzionale) per annullare l’operazione

Form

- Semantica del commit
  - ⇨quando l’utente preme il bottone di commit, la form esegue uno per uno i convalidatori di valore associati ai collegatori
  - ⇨successivamente esegue gli eventuali convalidatori di form
  - ⇨se, al termine delle convalide, sono stati trovati errori, la form visualizza un “popup” per segnalare gli errori, e NON esegue i collegatori

Form

- Semantica del commit (continua)
  - ⇨se, viceversa, al termine delle convalide NON sono stati trovati errori, la form esegue i collegatori
- Quindi
  - ⇨la semantica della form è “atomica”
  - ⇨o vengono eseguiti tutti i collegatori o non ne viene eseguito nessuno

Convalidatori

>> ConvalidatoreNumeroIntero

- L’interfaccia IConvalidatoreValore
  - ⇨List convalida(Object valore)
  - ⇨Object converti(Object valore)
- Il metodo convalida()
  - ⇨effettua tutte le verifiche sul valore immesso dall’utente e prelevato dal campo di immissione
- Il metodo converti()
  - ⇨viene usato per effettuare l’eventuale conversione del valore prima di copiare il valore nel modello (NOTA: dopo aver effettuato la convalida)

Convalidatori

- I convalidatori forniti dal framework
  - ConvalidatoreNumeroIntero: consente di verificare valore minimo e massimo
  - ConvalidatoreNumeroReale
  - ConvalidatoreStringaNonNulla
- Ulteriori convalidatori
  - possono essere facilmente sviluppati dal programmatore

Convalidatori

- L'interfaccia IConvalidatoreForm
  - prevede un unico metodo
  - List convalida(IForm form)
- I convalidatori di form
  - devono essere sempre implementati dal programmatore perchè obbediscono a regole della logica applicativa

Collegatori

- I collegatori forniti dal framework
  - ⇨CollegatoreTextField (adeguato per ogni tipo di JTextComponent, incluso JPasswordField)
  - ⇨CollegatoreCheckBox
  - ⇨CollegatoreComboBox
  - ⇨CollegatoreButtonGroup (per i JRadioButton)
- Al solito
  - ⇨i collegatori sono JavaBeans
  - ⇨ma ci sono vari costruttori con argomenti

Collegatori

>> mediaPesata - FinestraStudente

- Creare un collegatore
  - ⇨il modo più semplice è crearlo a partire da un osservatore esistente e da una form
  - ⇨il costruttore acquisisce il riferimento al componente, il nome del bean e della proprietà dall'osservatore
  - ⇨e poi aggiunge sè stesso alla collezione di collegatori della form
  - ⇨è possibile poi aggiungere uno o più convalidatori al collegatore

Collegatori

>> mediaPesata - PannelloPrincipale

- In alternativa
  - ⇨è possibile creare il collegatore anche in assenza di un osservatore (caso di binding è monodirezionale dalla vista al modello)
  - ⇨in questo caso è necessario specificare le informazioni relative al binding nel costruttore del collegatore
  - ⇨nome del componente grafico, nome del bean, nome della proprietà

Azione Commit e Azione Rollback

- Il passo finale
  - ⇨assegnare alla form il bottone di commit e quello di rollback
- Per farlo
  - ⇨è possibile utilizzare i vari sovraccarichi del metodo setBottoneCommit() e setBottoneRollback()

Azione Commit e Azione Rollback

- I versione
  - ⇨specificano il nome del bottone e l'id dell'azione da eseguire
  - ⇨l'azione viene eseguita dopo i collegatori

  void setBottoneCommit(String nomeBottone, String idAzione)
  void setBottoneRollback(String nomeBottone, String idAzione)
- Ma...
  - ⇨in molti casi, dopo aver eseguito i collegatori non c'è nessun'altra operazione da eseguire

Azione Commit e Azione Rollback

>> mediaPesata – FinestraEsame
>> mediaPesata – PannelloPrincipale

- In questo caso
  - ⇨è possibile semplicemente specificare il riferimento al bottone
  - ⇨in questo caso la form si limita ad eseguire i collegatori e a nascondere la vista associata se si tratta di una finestra di dialogo

  void setBottoneCommit(String nomeBottone)
  void setBottoneRollback(String nomeBottone)

Azione Commit e Azione Rollback

- Ulteriori alternative
  - ⇨consentono di produrre un messaggio di stato dopo avere eseguito i collegatori
  - ⇨oppure di cambiare lo stato
  - ⇨oppure di effettuare tutte e due le operazioni
  - ⇨in questo caso è necessario specificare il messaggio da produrre e/o lo stato che viene raggiunto

Azione Commit e Azione Rollback

- API delle form per i bottoni

```
void setBottoneCommit(JButton bottone)
void setBottoneCommit(JButton bottone, MessaggioPing messaggio)
void setBottoneCommit(JButton bottone, StatoPing stato)
void setBottoneCommit(JButton bottone, StatoPing stato, MessaggioPing messaggio)
void setBottoneCommit(JButton bottone, String idAzione)
void setBottoneCommit(String nomeBott)
void setBottoneCommit(String nomeBott, MessaggioPing messaggio)
void setBottoneCommit(String nomeBott, StatoPing stato)
void setBottoneCommit(String nomeBott, StatoPing stato, MessaggioPing messaggio)
void setBottoneCommit(String nomeBottone, java.lang.String idAzione)
```

Azione Commit e Azione Rollback

>> Form

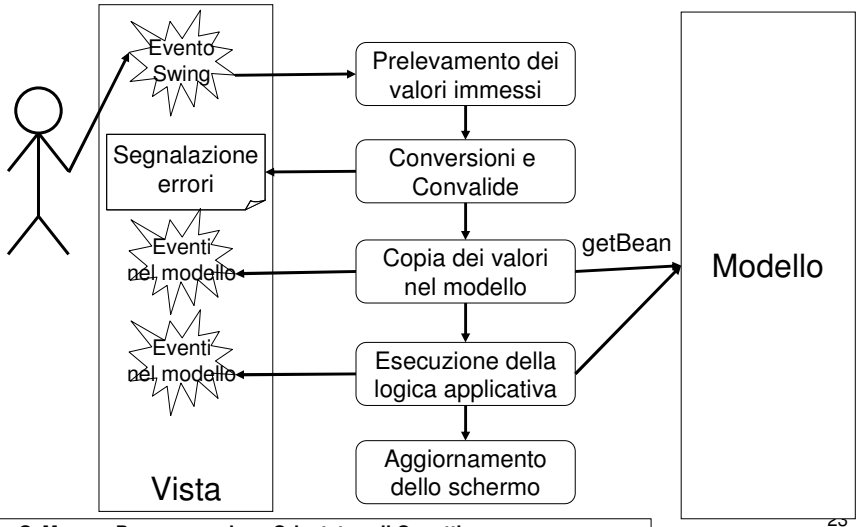
- API delle form per i bottoni (continua)

```
void setBottoneRollback(JButton bottone)
void setBottoneRollback(JButton bottone, MessaggioPing messaggio)
void setBottoneRollback(JButton bottone, StatoPing stato)
void setBottoneRollback(JButton bottone, StatoPing stato, MessaggioPing messaggio)
void setBottoneRollback(JButton bottone, String idAzione)
void setBottoneRollback(String nomeBott)
void setBottoneRollback(String nomeBott, MessaggioPing messaggio)
void setBottoneRollback(String nomeBott, StatoPing stato)
void setBottoneRollback(String nomeBott, StatoPing stato, MessaggioPing messaggio)
void setBottoneRollback(String nomeBottone, String idAzione)
```


Ciclo di Vita dell'Azione Commit

- A questo punto
  - ⇨possiamo schematizzare le varie fasi secondo le quali viene eseguita un'azione commit
  - ⇨ovvero il "ciclo di vita dell'azione commit"
  - ⇨si tratta di una schematizzazione utile perchè cicli di vita simili sono presenti in tutti i framework che forniscono supporto alle form

Ciclo di Vita dell'Azione Commit



Ciclo di Vita dell'Azione Commit

- Alcune considerazioni sul ciclo di vita
  - ⇨la prima considerazione è che perchè il commit funzioni, nel modello deve essere già disponibile una istanza del bean su cui effettuare il binding
  - ⇨seconda considerazione: c'è un'interazione tra collegatori e osservatori: i collegatori producono modifiche nelle proprietà del bean, che normalmente scatenano eventi gestiti dagli osservatori per aggiorn. la vista

Ciclo di Vita dell'Azione Commit

- Sulla base di queste considerazioni
  - è possibile interpretare due concetti importanti
  - la modalità di creazione automatica delle form di ping
  - il concetto di bean di richiesta e di bean di sessione

Creazione Automatica

- Prima di potere eseguire i collegatori
  - è necessario che nel modello sia disponibile un'istanza del bean su cui effettuare il binding
  - questo richiede normalmente di avere effettuato precedentemente un'azione che ha creato l'istanza del bean
  - per evitare questo fatto, il framework supporta una modalità di creazione diversa

Creazione Automatica

- Creazione automatica
  - subito prima di eseguire i collegatori, la form crea un'istanza del bean specificato e la salva nel modello
  - in questo modo è possibile che il bean non esista nel modello al commit della form, e ciononostante la form può eseguire il commit
  - al termine del commit la form ripulisce i componenti per riportarli allo stato iniziale
  - ci sono però delle sottigliezze

Creazione Automatica

- La sottigliezza principale
  - per via dell'interferenza tra collegatori e osservatori non è possibile utilizzare questo meccanismo se ai collegatori sono associati osservatori
  - ovvero se il binding della form è bidirezionale
  - infatti gli osservatori, a seguito della creazione automatica modificherebbero i valori forniti dall'utente

Creazione Automatica

- Anatomia del commit
  - ⇨dopo che l'utente ha immesso i valori (es: nome "Fisica") e sottomesso la form, la form crea l'istanza del bean e la salva nel modello
  - ⇨questo scatena una serie di eventi, a seguito dei quali gli osservatori cambiano il valore dei campi di immissione in modo che rispecchino i valori iniziali del bean (es: nome null oppure "")
  - ⇨quando i collegatori vengono eseguiti, prelevano dai campi di immissione i valori alterati e li copiano nel modello, di fatto effettuando un'operazione inutile

Creazione Automatica

- Di conseguenza
  - ⇨la creazione automatica può essere utilizzata solo nei casi in cui sia necessario il binding solo dalla vista al modello e non viceversa
- Caso tipico
  - ⇨una form con cui è necessario inserire nuovi elementi in una collezione ripulendo la form al termine di ogni immissione
  - ⇨es: immissione degli esami universitari

Creazione Automatica

- Per abilitare la creazione automatica
  - ⇨è possibile utilizzare il metodo setCreazioneAutomatica() di IForm
  - ⇨specificando il riferimento alla classe del bean da creare (oggetto class), e la chiave con cui deve essere salvato nel modello

>> mediaPesata - pannelloPrincipale

Ciclo di Vita dei Bean

- Un bean gestito in questo modo
  - ⇨è un bean dal ciclo di vita molto breve
  - ⇨che dura tanto quanto l'esecuzione dell'azione commit
  - ⇨infatti, durante l'esecuzione dell'azione, è possibile effettuare removeBean() invece che getBean() e questo non ha effetti sull'applicazione

Ciclo di Vita dei Bean

- Sulla base della durata del ciclo di vita
  - ⇨possiamo classificare i bean in tre categorie
- Bean di sessione
  - ⇨restano nel modello per tutta la durata dell'applicazione (ovvero della sessione di lavoro) perchè mantengono dati utili per dall'inizio alla fine
  - ⇨su questi bean viene eseguito putBean() nell'azione iniziale e non viene mai eseguito removeBean()
  - ⇨esempio: il bean Gioco nella Morra Cinese

Ciclo di Vita dei Bean

- Bean di richiesta
  - ⇨restano nel modello esclusivamente per l'esecuzione di una azione (tipicamente per il commit di una form con creazione automatica)
  - ⇨al commit successivo viene creata una nuova istanza che rimpiazza la precedente
  - ⇨su questi bean viene eseguito putBean() per effettuare il commit e removeBean() nell'azione associata
  - ⇨esempio: il bean Esame da inserire

Ciclo di Vita dei Bean

- Bean di caso d'uso
  - è una durata intermedia, la più frequente
  - restano nel modello per l'esecuzione di un caso d'uso che richiede varie azioni
  - su questi bean viene eseguito putBean() all'inizio del caso d'uso e removeBean() al termine del caso d'uso
  - esempio: il bean Partita di Morra Cinese, il bean Studente di Media Pesata

Ciclo di Vita dei Bean

ATTENZIONE
al ciclo di vita
dei bean

- In generale
  - è indispensabile ragionare sulla natura di ciascun componente del modello
  - e accertarsi di gestirne correttamente le fasi del ciclo di vita utilizzando getBean() e removeBean()

Wizard

- A questo punto
  - ⇨è facile interpretare la semantica dei wizard
- Wizard
  - ⇨sequenza di form concatenate
  - ⇨ciascuna form ha due bottoni aggiuntivi
  - ⇨un “bottone avanti” per muoversi alla maschera successiva
  - ⇨un “bottone indietro” per muoversi alla maschera precedente

Wizard

- Form che partecipano al wizard
  - ⇨sono sempre associate ad una finestra di dialogo (e non ad un pannello o al frame)
  - ⇨cambia leggermente la semantica del bottone commit e rollback
  - ⇨il bottone commit di una qualsiasi delle form del wizard conclude tutta la procedura
  - ⇨il bottone rollback di una qualsiasi delle form del wizard annulla tutta la procedura

Wizard

- Tipicamente
  - ⇨tutte le maschere hanno un bottone rollback per interrompere l'operazione
  - ⇨la prima maschera ha un bottone avanti oltre al bottone rollback
  - ⇨le maschere intermedie hanno un bottone avanti e uno indietro oltre al bottone rollback
  - ⇨la maschera finale ha un bottone indietro e un bottone commit oltre al bottone rollback

Wizard

- Di conseguenza
  - ⇨il wizard ha normalmente un unico bottone commit nell'ultima form
- Passi opzionali
  - ⇨nel caso in cui gli ultimi passi siano opzionali, le form finali corrispondenti possono inoltre avere un bottone commit che conclude la procedura senza aver raggiunto l'ultimo passo

Wizard

- Per impostare il bottone avanti
  - ⇨l'API del framework offre vari metodi
  - ⇨in ciascuno dei casi è necessario specificare il bottone e l'azione da eseguire
- Per specificare il bottone
  - ⇨è possibile utilizzare il riferimento
  - ⇨oppure il nome per facilitare l'utilizzo dell'ambiente RAD

Wizard

- Per specificare l'azione
  - ⇨è possibile utilizzare l'id
- In questo caso
  - ⇨alla pressione del tasto avanti viene eseguito il binding in modo ordinario
  - ⇨poi viene eseguita l'azione specificata
  - ⇨l'azione è responsabile di effettuare la navigazione alla form successiva usando il metodo vista.visualizzaSottovista(idSottoVista)

Wizard

- Azione avanti standard
  - ⇨nel caso in cui oltre ad eseguire il binding della form e a navigare alla vista successiva non ci siano ulteriori operazioni è possibile utilizzare l'azione avanti standard
  - ⇨in questo caso è necessario esclusivamente specificare l'id della vista a cui navigar
- Analogamente
  - ⇨per l'azione indietro

Wizard

- API delle form per i wizard

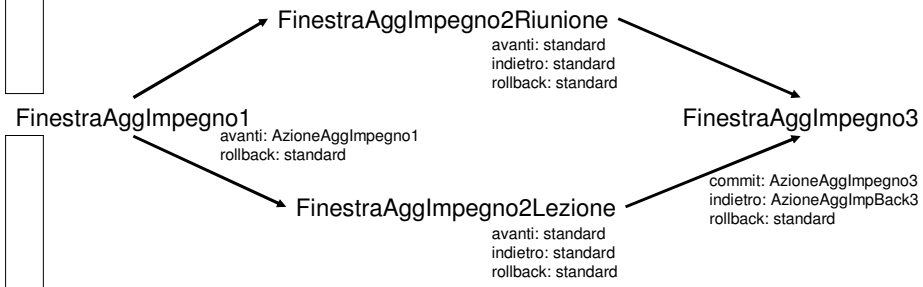
```
void setBottoneAvanti(JButton bottone, FinestraDiDialogoPing vista)
void setBottoneAvanti(JButton bottone, String idAzione)
void setBottoneAvanti(String nomeBottone, FinestraDiDialogoPing vista)
void setBottoneAvanti(String nomeBottone, String idAzione)

void setBottoneIndietro(JButton bottone, FinestraDiDialogoPing vista)
void setBottoneIndietro(JButton bottone, String idAzione)
void setBottoneIndietro(String nomeBottone, FinestraDiDialogoPing vista)
void setBottoneIndietro(String nomeBottone, String idAzione)
```

Wizard

- Un esempio di uso dei wizard
  - l'applicazione appuntamenti
  - il caso d'uso "Aggiungi appuntamento" richiede di sviluppare un wizard a tre passi
- Attenzione
  - il caso d'uso è reso particolarmente complesso dal fatto che lavora su una gerarchia
  - è necessario usare vari oggetti per effettuare il binding delle form
  - un oggetto ImpegnoAstratto, un oggetto Riunione e uno Lezione

Wizard

- Struttura del wizard



Wizard

>> appuntamenti

- Un aspetto interessante dell'applicazione
  - ⇨l'utilizzo della toolbar
- Alcuni accorgimenti
  - ⇨viene usato il set di icone di Eclipse
  - ⇨nella configurazione delle azioni viene specificato l'attributo icona e non quello nome; in questo modo le azioni sono pronte per l'utilizzo nella toolbar (il nome verrebbe visualizzato)
  - ⇨nel caso in cui le azioni vengano usate per bottoni e menu, viene specificato il nome del bottone o della voce di menu

Riassumendo

- Form
- Convalidatori
- Collegatori
- Azione Commit e Azione Rollback
- Ciclo di Vita dell'Azione Commit
  - ⇨Creazione Automatica
  - ⇨Ciclo di Vita del Bean
- Wizard

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.