Programmazione Orientata agli Oggetti in Linguaggio Java

Programmazione Grafica: Framework

versione 1.0

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

Sommario

- Problemi dell'Architettura Vista
- Verso un Framework per Java Swing
- Il Manuale di ping
  - ⇨Applicazione
  - ⇨File di Configurazione
  - ⇨Controllo e Azioni
  - ⇨Gerarchia di Viste
  - ⇨Modello e Binding
- Benefici di ping

Problemi dell'Architettura Vista

- Nelle unità precedenti
  - ⇨abbiamo introdotto un'architettura MVC per lo sviluppo con Java Swing
  - ⇨volta a standardizzare e semplificare l'organizzazione dei componenti
  - ⇨a separare gli strati applicativi
- Vantaggio
  - ⇨il codice è più organizzato, quindi più modulare e quindi più manutenibile

Problemi dell'Architettura Vista

- Ma...
  - ⇨restano alcuni problemi che sarebbe opportuno eliminare
- Di seguito
  - ⇨discutiamo i seguenti problemi per cercare di individuare in quali direzioni migliorare l'architettura vista

Problemi dell'Architettura Vista

- I problema: molti compiti ripetitivi
  - ⇨riprodurre l'architettura in ogni applicazione richiede di ripetere sempre le stesse operaz.
  - ⇨definire la classe Modello
  - ⇨definire la classe Controllo, inizializzare e salvare le azioni nella mappa
  - ⇨definire le viste, inizializzarle e salvarle opportunamente nelle varie mappe
  - ⇨ogni passo è una possibile fonte di errore

Problemi dell'Architettura Vista

- II problema: rapporto tra vista e controllo
  - ⇨i due strati sono fortemente accoppiati
  - ⇨le viste devono definire metodi pubblici per l'acquisizione dei dati forniti dall'utente
  - ⇨le azioni sono obbligate a "navigare" la gerarchia di viste (che può essere complessa e soggetta a modifiche) per acquisire i valori forniti dall'utente ed eseguire gli schermi
  - ⇨modifiche alla vista influiscono sulle azioni

Problemi dell'Architettura Vista

- III problema: binding
  - ⇨gestire la sincronizzazione tra componenti della vista attraverso il meccanismo di copia è noioso e soggetto ad errori
- IV problema: servizi di supporto
  - ⇨in tutte le applicazioni è necessario utilizzare gli stessi servizi (es: Logger, pool di connessioni >>)
  - ⇨questo richiede di utilizzare varie librerie

Verso un Framework per Java Swing

- Questi problemi
  - ⇨sarebbero risolvibili adottando un framework per lo sviluppo MVC con Java Swing
  - ⇨il framework dovrebbe fornire un'infrastruttura di classi che superi una volta per tutte questi problemi
- Di seguito
  - ⇨ragioniamo sui requisiti per un framework del genere

Verso un Framework per Java Swing

- I requisito
  - ⇨fornire classi già pronte per Modello e Controllo
  - ⇨fornire classi astratte da estendere per le viste
  - ⇨se possibile, automatizzare le operazioni di registrazione delle azioni e delle viste
  - ⇨sulla base di file di configurazione definiti dallo sviluppatore

Verso un Framework per Java Swing

- Il requisito
  - ⇨semplificare il rapporto tra vista e controllo disaccoppiandoli definitivamente
- Idealmente
  - ⇨le azioni dovrebbero poter chiedere alla vista “voglio il valore del componente X” oppure “esegui lo schermo Y” senza dover navigare
  - ⇨es: specificando X e Y attraverso stringhe che identifichino il componente o lo schermo

Verso un Framework per Java Swing

- III requisito
  - ⇨fornire un meccanismo di binding per automatizzare la sincronizzazione tra elementi della vista e bean
  - ⇨il componente grafico viene “collegato” ad una proprietà del modello
  - ⇨in modo che il componente grafico rifletta automaticamente le modifiche alla proprietà

Verso un Framework per Java Swing

- IV elemento
  - ⇨fornire tutti i servizi di supporto (logger ecc.) e la possibilità di configurarli senza dover includere altre librerie
  - ⇨fornire un unico componente che funga da “punto di ingresso” all’applicazione e consenta di acquisire tutti i componenti di servizio

Verso un Framework per Java Swing

- I framework disponibili per Swing
  - ⇨pochissimi
  - ⇨impostazioni diverse l'una dall'altra
  - ⇨i più sofisticati sono anche molto complessi
- Esempio: Scope MVC
  - ⇨basato sul concetto di MVC "gerarchico" (HMVC): gerarchia di triadi MVC nell'applicazione

Verso un Framework per Java Swing

- La nostra soluzione
  - ⇨utilizzeremo un nostro framework didattico
- it.unibas.ping

  - ⇨il primo framework didattico per lo sviluppo MVC con Java Swing
  - ⇨discende da un framework didattico gemello (it.unibas.pinco) pensato per lo sviluppo Web
  - ⇨ping = pinco per Swing
  - ⇨entrambi sono completamente open source

Verso un Framework per Java Swing

- Attenzione
  - ping è MOLTO più complesso di pinco
  - pinco: circa 800 lc in 30 classi e 4 package
  - ping: più di 2300 lc in 51 classi e 9 package
- Inoltre
  - la versione attuale è ancora ampiamente preliminare
  - richiede un'approfondita fase di test
  - avviamo il programma "Beta Testing" di ping!

Il Manuale di ping

- I componenti fondamentali di ping
  - la classe Applicazione
  - il file di configurazione ping-config.xml
  - la classe Controllo
  - la classe Modello
  - le classi della vista
  - tutte nel package it.unibas.ping.framework
  - le interfacce per le azioni
  - il framework per il binding dei componenti

Applicazione

>> ping - JavaDoc

- La classe Applicazione
  - componente unico con visibilità globale
  - funge da punto di accesso ai componenti
- Punto di accesso all'infrastruttura
  - consente l'accesso alle classi dell'infrastruttura MVC attraverso i metodi getControllo(), getModello(), getVistaPrincipale()
  - contiene il main e avvia l'applicazione

Applicazione

>> morracineseping – logging
>> logger.properties

- Punto di accesso ai servizi
  - fornisce l'accesso ai servizi integrati nel framework; in particolare il logger
  - Logger getPingLogger()
  - metodi statici per comodità richiamabili direttamente da Applicazione (logFiner(), logFine(), logInfo(), logSevere())
  - il logger è configurabile con il file di configurazione /risorse/logger.properties

Applicazione

- Inoltre
  - ⇨fornisce una propria mappa di riferimenti in cui è possibile salvare riferimenti ad altri componenti globali (es: servizi non previsti)
  - ⇨void putAttributo(String chiave, Object rif)
  - ⇨Object getAttributo(String chiave)
  - ⇨void removeAttributo(Object rif)

Il File di Configurazione

- ping-config.xml
  - ⇨file .xml che contiene una descrizione dichiarativa di molti aspetti dell'applicazione
  - ⇨valido rispetto ad un opportuno DTD ping.dtd
  - ⇨deve essere raggiungibile attraverso il classpath in una cartella /risorse
  - ⇨viene caricato all'avvio dell'applicazione

Il File di Configurazione

- Struttura del file
  - I sezione: descrizione della gerarchia di viste; ogni vista ha un nome, una classe, e una o più sottoviste
  - II sezione: descrizione delle azioni ping; ogni azione ping ha un nome chiamato “comando” e una classe
  - III sezione: descrizione dei listener swing

Il File di Configurazione

>> morraCinese: ping-config.xml

- Idea
  - molta parte del codice può essere evitato “descrivendo” alcuni aspetti dell’applicazione nel file di configurazione e lasciando il lavoro al framework
  - es: gerarchia delle viste, attributi delle azioni Swing
- Nel seguito
  - descriviamo le varie sezioni

Controllo e Azioni

- Una caratteristica fondamentale di ping
  - ⇨concetto generalizzato di azione applicativa
  - ⇨distinzione tra azione e listener
- Lo strato di controllo
  - ⇨è fatto di azioni “ping”
  - ⇨che vengono eseguite secondo la sequenza evento – listener – comando – azione

Controllo e Azioni

- Azioni di ping
  - ⇨le azioni sono oggetti che implementano l’interfaccia it.unibas.ping.azioni.AzionePing
  - ⇨public void esegui(java.util.EventObject evento) throws PingException;
  - ⇨ogni azione corrisponde ad un comando
- Comando
  - ⇨stringa attraverso la quale un listener chiede l’esecuzione di una azione

Controllo e Azioni

ATTENZIONE
alla differenza tra azioni ping e listener

- Attenzione alla differenza
  - ⇨le azioni di ping NON sono listener
  - ⇨non sono specifiche di un tipo di evento (es: ActionEvent), ma sono eseguibili in corrispondenza di ogni tipo di evento
- Naturalmente
  - ⇨per eseguire un'azione in corrispondenza di un evento, è necessario utilizzare un listener

Controllo e Azioni

- Listener di ping
  - ⇨oggetti che fungono esclusivamente da tramite tra i componenti grafici e il Controllo
  - ⇨ogni listener è associato ad un comando
  - ⇨l'unica funzione di un listener è chiedere al Controllo l'esecuzione dell'azione ping associata al comando

Controllo e Azioni

- Esempio: ActionListener
  - ⇨ogni ActionListener listener utilizzato in un'applicazione ping ha una proprietà che specifica il comando corrispondente
  - ⇨il metodo actionPerformed() è sempre uguale

```
private String comando;
public void actionPerformed(ActionEvent e) {
    Controllo.getInstance().eseguiAzione(this.comando, e);
}
```


Controllo e Azioni

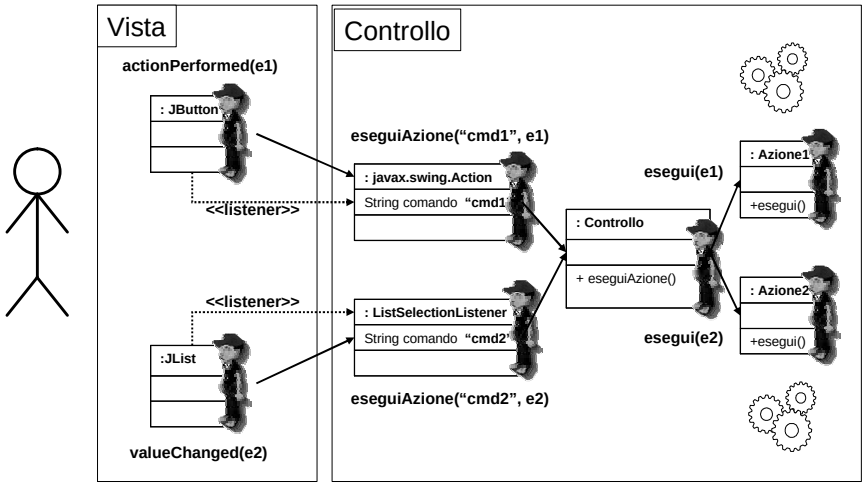
- Esempio: ListSelectionListener
  - ⇨un ListSelectionListener è un listener di basso livello utilizzato per gestire eventi di selezione (cambia la selezione in una lista o in una tabella)

```
private String comando;
public void valueChanged(ListSelectionEvent e) {
    Controllo.getInstance().eseguiAzione(this.comando, e);
}
```

Controllo e Azioni

- La classe Controllo
  - ⇨mantiene la mappa delle azioni
  - ⇨esegue le azioni associate ai comandi
  - ⇨eseguendo il metodo
    public void eseguiAzione(String comando,
    java.util.EventObject evento)
  - ⇨acquisisce l'azione ping corrispondente al comando dalla mappa ed esegue il metodo esegui(evento)

Controllo e Azioni



Controllo e Azioni

- Vantaggi di questo approccio
  - ⇨le azioni di ping sono gestori di eventi “generici”
  - ⇨totalmente disaccoppiati dalla vista
  - ⇨non sono direttamente sottoscrittori dei componenti grafici
  - ⇨consentono di gestire un’ampia gamma di eventi (potenzialmente anche eventi che non hanno origine nell’interfaccia grafica)

Controllo e Azioni

- Azioni di Swing
  - ⇨in questo contesto sono considerate particolari ActionListener
  - ⇨con proprietà aggiuntive (nome, descrizione, mnemonico, icona ecc.)
  - ⇨a ciascuna azione Swing è associato un comando
  - ⇨ciascuna azione Swing invoca l’esecuzione di un’azione ping

Controllo e Azioni

- Cosa sviluppa il programmatore
  - ⇨esclusivamente il codice delle Azioni ping
- Il resto
  - ⇨viene descritto al framework attraverso il file di configurazione ping-config.xml
  - ⇨in particolare, tutti gli oggetti listener vengono fabbricati dal controllo al caricamento del file di configurazione

Controllo e Azioni

- Il processo
  - ⇨l'oggetto Applicazione avvia l'esecuzione
  - ⇨viene caricato il file di configurazione
  - ⇨il Controllo fabbrica tutte le azioni ping e tutti i listener
  - ⇨in particolare le azioni Swing, inizializzandole con le proprietà specificate nel file di configurazione

Controllo e Azioni

- Il processo (continua)
  - ⇨da quel momento le azioni ping sono eseguibili attraverso il metodo void eseguiAzione() della classe Controllo
  - ⇨i listener sono disponibili per la vista attraverso metodi opportuni della classe Controllo
  - ⇨javax.swing.Action getAzioneSwing(String comando)
  - ⇨ChangeListener getChangeListener(String comando)

Controllo e Azioni

>> morraCinese: ping-config.xml
>> morraCinese: Vista
>> morraCinese: AzioneIniziaPartita

- Il processo (continua)
  - ⇨viene costruita la vista
  - ⇨nel codice della vista i listener vengono registrati nei componenti, acquisendoli dal Controllo
  - ⇨da quel momento in poi, ogni gesto dell'utente scatena un evento, che viene notificato ad un listener, che richiede al Controllo l'esecuzione di un'azione

Controllo e Azioni

- Due azioni particolari
  - ⇨ l'azione iniziale – oggetto di tipo it.unibas.ping.azioni.AzioneIniziale: eseguita alla costruzione del Controllo – utile per disabilitare i componenti nello schermo di avvio
  - ⇨ l'azione finale – oggetto di tipo it.unibas.ping.azioni.AzioneFinale: eseguita alla distruzione del Controllo (es: System.exit())
  - ⇨ sono entrambe opzionali

Controllo e Azioni

>> morraCinese: AzioneIniziale
>> morraCinese: uso delle azioni pred.

- Le azioni predefinite di ping
  - ⇨ ping fornisce una serie di azioni già pronte, che corrispondono a compiti standard e possono essere utilizzate senza doverle ridefinire
  - ⇨ it.unibas.ping.azioni.AzioneEsci: System.exit()
  - ⇨ it.unibas.ping.azioni.AzioneAboutApplicazione: finestra di dialogo che descrive l'applicazione
  - ⇨ it.unibas.ping.azioni.AzioneAboutPing: finestra di dialogo che descrive il framework

Controllo e Azioni

- Riassumendo
  - ⇨lo sviluppatore sceglie le azioni applicative e le associa ai comandi
  - ⇨sceglie il tipo di listener necessario per ciascun comando (di solito Azioni Swing)
  - ⇨sviluppa il file ping-config.xml
  - ⇨sviluppa il codice delle azioni di ping
  - ⇨sviluppa il codice dell'eventuale azione iniziale e finale

Gerarchia di Viste

- Le viste di ping
  - ⇨sono classi astratte che implementano it.unibas.ping.framework.VistaPing
  - ⇨lo sviluppatore deve estenderle e ridefinire il metodo inizializza() per disporre i componenti
  - ⇨ogni vista può avere sottoviste ed eredita automaticamente le funzionalità per la gestione della mappa delle sottoviste

Gerarchia di Viste

- Tre tipi di viste
  - it.unibas.ping.framework.VistaFramePrincipale: frame principale, estende JFrame
  - it.unibas.ping.framework.VistaPannelloSecondario: pannello, estende JPanel
  - it.unibas.ping.framework.VistaFinestraSecondaria: finestra secondaria, estende JDialog

Gerarchia di Viste

- Creazione della gerarchia di viste
  - la gerarchia viene descritta nel file ping-config.xml
  - ping inizializza l'albero dei riferimenti riempendo le mappe
- Una caratteristica importante di ping
  - non è richiesta la navigazione della gerarchia di viste

Gerarchia di Viste

- Esecuzione dello schermo successivo
  - ⇨la navigazione della gerarchia di viste viene effettuata da ping
- Schermo
  - ⇨ogni schermo è un metodo di una qualsiasi delle viste della gerarchia
  - ⇨con tipo di ritorno void
  - ⇨e senza parametri

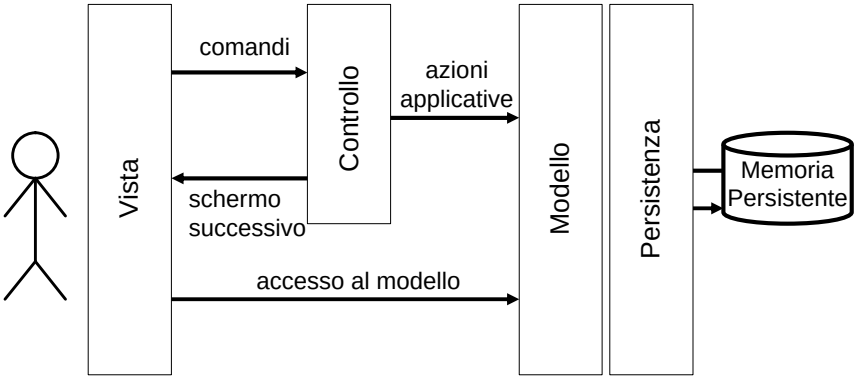
Gerarchia di Viste

- Esecuzione dello schermo
  - ⇨le azioni chiamano il metodo void eseguiSchermo(String nomeschermo) sulla vista principale specificando il nome dello schermo
  - ⇨il framework ispeziona la gerarchia delle viste e appena trova un metodo con il nome specificato lo esegue
  - ⇨è quindi indispensabile evitare che viste diverse abbiano schermi con nomi uguali

Gerarchia di Viste

- Schematizzando
  - ⇨vista e controllo sono del tutto disaccoppiati
  - ⇨la vista invoca le azioni nel controllo specificando comandi che sono stringhe
  - ⇨le azioni invocano gli schermi successivi nella vista specificando nomi che sono stringhe

Gerarchia di Viste



Gerarchia di Viste

- Acquisizione dei valori forniti dall'utente
  - ⇨analogamente, le azioni acquisiscono i valori dalla vista chiamando i metodi
    public Object trovaValore(String nomeComponente);
  - ⇨per farlo però, i componenti relativi devono essere stati identificati con una chiamata al metodo void setName(String nome)
  - ⇨naturalmente non possono esserci due componenti con lo stesso nome

Gerarchia di Viste

>> indovina: ping-config.xml
>> indovina: AzioneIniziaGioco
>> indovina: AzioneTentativo

- Componenti attualmente ispezionati
  - ⇨JTextField: restituisce un riferimento String, corrispondente al valore del testo
  - ⇨JCheckBox: restituisce un riferimento a Boolean, a seconda che sia selezionato o no
  - ⇨JRadioButton: restituisce un riferimento a Boolean, a seconda che sia seleizionato o no

Gerarchia di Viste

>> Convalidatore: JavaDoc

- Supporto alla convalida
  - ⇨la convalida dei dati è resa più snella utilizzando i metodi della classe Convalidatore
  - ⇨vari metodi che consentono di effettuare le verifiche più frequenti (numeri interi, numeri reali, stringhe nulle)
  - ⇨producendo messaggi di errore esplicativi
  - ⇨es: String verificaIntero(String valore, String nomeValore)

Gerarchia di Viste

>> mediaPesata: ping-config.xml
>> mediaPesata: AzioneAggiornaEsame (confronto con versione Swing)

- Selezione in una lista
  - ⇨funziona in modo analogo
  - ⇨metodo public int getIndiceSelezione(String nomeComponente);
  - ⇨cerca il componente con il nome specificato che abbia un modello di selezione (JList o JTable) e restituisce l'indice dell'elemento o della riga selezionata

Modello e Binding

- La classe Modello
  - ⇨fornisce i metodi per la gestione della sessione di lavoro
  - ⇨public void putBean(String nome, Object rif)
  - ⇨public Object getBean(String nome)
- Inoltre
  - ⇨ha un ruolo fondamentale nel sistema di binding tra componenti grafici e bean

Modello e Binding

- L'idea alla base del binding
  - ⇨utilizzare il meccanismo di publish&subscribe
  - ⇨rendere i componenti della vista gestori di eventi lanciati dai bean del modello
  - ⇨ogni volta che il bean viene aggiornato, notifica ai componenti della vista un evento di modifica
  - ⇨i componenti della vista gestiscono l'evento accedendo al bean e aggiornandosi

Modello e Binding

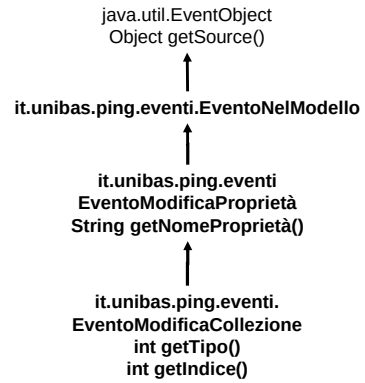
- Terminologia
  - si dice che i componenti della vista sono “osservatori” dei bean del modello
  - per similarità con il termine sensore/ascoltatore (“listener”) e assonanza con il termine “vista”
  - i bean del modello si dicono oggetti “osservabili” perchè accettano tra i loro sottoscrittori degli osservatori

Modello e Binding

- Gli strumenti di ping
  - una serie di classi per gli eventi nel modello
  - una serie di interfacce per gli osservatori degli eventi
  - una classe Osservabile che fornisce le funzionalità di pubblicazione degli eventi
  - i bean possono estenderla o lavora in associazione con un suo oggetto

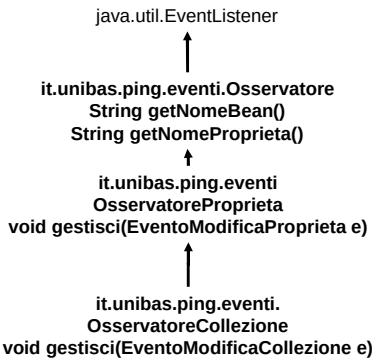
Modello e Binding

- La gerarchia degli eventi



Modello e Binding

- La gerarchia dei listener



Modello e Binding

>> Osservabile - JavaDoc

- La classe Osservabile
  - ⇨fornisce il supporto ai bean per il meccanismo di publish and subscribe
  - ⇨liste di osservatori (divise rispetto al tipo di osservatore)
  - ⇨metodi per la sottoscrizione e la rimozione degli osservatori
  - ⇨metodi per la notifica degli eventi agli osservatori

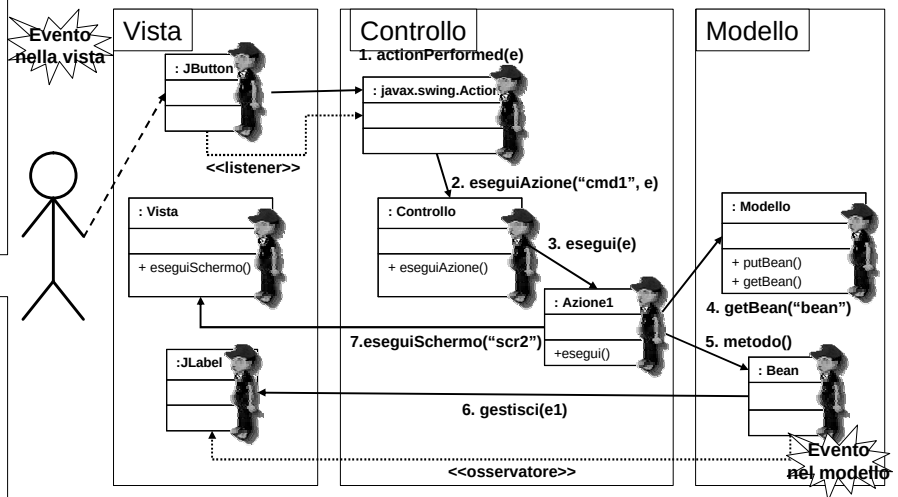
Modello e Binding

- I bean
  - ⇨devono estendere Osservabile o lavorare in associazione con un Osservabile se estendono un'altra classe
  - ⇨devono notificare eventi di modifica ogni volta che una delle proprietà viene modificata
  - ⇨creando un oggetto evento del tipo opportuno
  - ⇨e poi segnalandolo con i metodi di notifica

Modello e Binding

- Gli osservatori
  - ping fornisce una serie di componenti preconfezionati che sono osservatori
  - PLabel : osservatore proprietà
  - PTextField : osservatore proprietà
  - PCheckBox : osservatore proprietà
  - ModelloListaPing : osservatore collezione
  - ModelloTabellaPing: osservatore collezione

Controllo e Azioni



Modello e Binding

- La sottoscrizione
  - ⇨gli osservatori possono sottoscrivere direttamente gli eventi di un bean
  - ⇨oppure, più opportunamente, interagire con il Modello
- Infatti
  - ⇨i bean possono cambiare (es: new Partita())
  - ⇨il Modello è stabile

Modello e Binding

>> indovinaIlNumero: tentativo
>> mediaPesata: dati studente
+ modello lista/tabella

- Il ruolo del modello
  - ⇨il Modello funge da intermediario
  - ⇨ogni osservatore che si registra tra i sottoscrittori del Modello viene automaticamente registrato come gestore del bean opportuno
  - ⇨ogni volta che il bean cambia per via di putBean(), l'osservatore viene registrato tra i sottoscrittori del nuovo bean

Modello e Binding

- 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
- Nota
  - ⇨questi vantaggi hanno un prezzo

Modello e Binding

- Svantaggio del meccanismo di binding
  - ⇨è leggermente aumentato l'accoppiamento tra vista e modello a causa del meccanismo di publish & subscribe
  - ⇨infatti il modello è cambiato rispetto alla versione console
  - ⇨ma l'accoppiamento è controllabile perchè ha regole precise
  - ⇨in ogni caso il meccanismo è opzionale

Benefici di ping

no binding
no convalida
(solo infrastruttura)

maggiore “rigidità”
nello sviluppo
(AzioneGiocata)

- Alcune statistiche
  - ⇨sulle applicazioni di test
  - ⇨funzionalità comparabili (in alcuni casi maggiori nella versione ping)

	Versione Swing		Versione ping			
Applicaz.	L. di C.	Classi	L. di C.	Riduz.	Classi	Riduz.
Morra Cinese	760	14	689	-9%	14	-0%
Indovina il Numero	861	21	658	-24%	16	-23%
Media Pesata	1658	38	1323	-20%	32	-15%

Benefici di ping

- In sintesi
  - ⇨il vantaggio principale è legato alla riduzione della dimensione del codice
  - ⇨dovuto al riuso delle classi dell’infrastruttura
  - ⇨all’inizializzazione automatica delle mappe
  - ⇨alla navigazione automatica della gerarchia di viste
  - ⇨al supporto alla convalida

Benefici di ping

- Inoltre
  - ⇨c'è il vantaggio del meccanismo di binding
- Nota
  - ⇨esistono altri framework molto usati per il binding
  - ⇨alcuni integrati con framework per la convalida dei dati
  - ⇨es: JGoodies binding, JGoodies validation

Riassumendo

- Problemi dell'Architettura Vista
- Verso un Framework per Java Swing
- Il Manuale di ping
  - ⇨Applicazione
  - ⇨File di Configurazione
  - ⇨Controllo e Azioni
  - ⇨Gerarchia di Viste
  - ⇨Modello e Binding
- Benefici di ping

Termini della Licenza