Programmazione Orientata agli Oggetti in Linguaggio Java

Programmazione Grafica: Organizzazione del Codice Parte c

versione 1.0

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

Sommario

- MVC e Swing
  - ⇨Sviluppare un Componente Swing
  - ⇨Architettura dei Componenti Swing
  - ⇨Problemi di Questa Architettura
- Un Altro Esempio
  - ⇨Visualizzazione di Liste
  - ⇨Visualizzazione di Tabelle
- Il Problema del “Binding”

MVC e Swing

- Un equivoco frequente
  - ⇨l'utilizzo del termine "architettura MVC" collegato alla tecnologia Swing
  - ⇨molto frequente nella letteratura (libri, ed articoli tecnici), assente nello Swing tutorial
- Chiarimento importante
  - ⇨in questo caso il termine viene usato in un'accezione diversa rispetto a quella vista nelle unità precedenti

MVC e Swing

- Le due accezioni del termine MVC
  - ⇨in queste unità: linea guida per l'organizzazione dell'architettura (strati) dell'applicazione
  - ⇨ovvero infrastruttura a livello di applicazione
  - ⇨nella letteratura su Swing: descrizione dello schema secondo cui sono costruiti i componenti grafici di Swing

Sviluppare un Componente Swing

- Sviluppiamo un componente tipico
  - ⇨data una libreria per disegnare linee e punti
  - ⇨sviluppare un componente che si comporta come un tipico “bottone”
  - ⇨da includere in una libreria grafica
- I passo: analisi delle specifiche
  - ⇨ovvero analisi dei casi d’uso relativi all’utilizzo del bottone

Sviluppare un Componente Swing

- Esempio: “Utente schiaccia bottone”
  - ⇨l’utente preme il pulsante del mouse mentre il puntatore è sul bottone
  - ⇨il bottone entra nello stato “armato” e cambia visualizzazione per dare l’impressione di essere stato premuto
  - ⇨l’utente rilascia il pulsante del mouse mentre il puntatore è ancora sul bottone
  - ⇨il bottone entra nello stato “selezionato” e cambia visualizzazione (diventa il componente selezionato)
  - ⇨il bottone notifica l’evento ai gestori registrati

Sviluppare un Componente Swing

- Scenari alternativi
  - ⇨l'utente utilizza lo mnemonico del bottone e non il mouse (cambia il tipo di gesto)
  - ⇨l'utente schiaccia il bottone mentre non è abilitato
  - ⇨l'utente arma il bottone ma poi rilascia il tasto dopo aver spostato il puntatore in un'altra zona
  - ⇨cambia l'ambientazione ("look and feel") del bottone e quindi le modalità di visualizzazione dei vari stati

Sviluppare un Componente Swing

- Il passo: modello concettuale
  - ⇨il concetto principale è quello di bottone con le sue varie proprietà
  - ⇨vari altri concetti rilevanti evento, gestore ambientazione che per semplicità ignoriamo

Bottone
nome mnemonico tooltip abilitato (true/false) armato (true/false) premuto (true/false) selezionato (true/false)

Sviluppare un Componente Swing

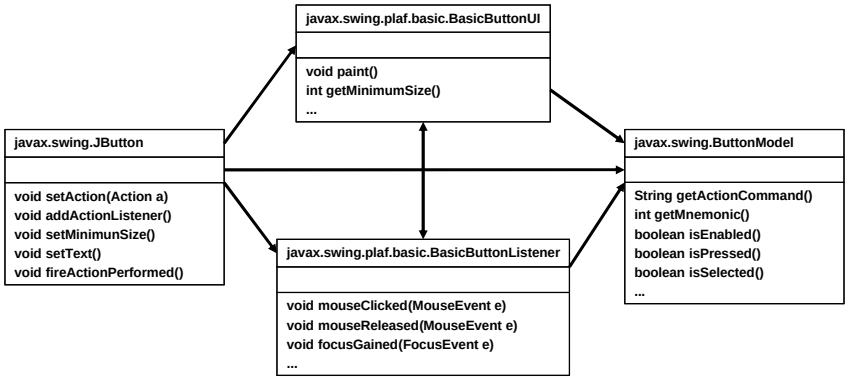
- III passo: sviluppo dei componenti
  - ⇨il concetto di bottone porta alla classe ButtonModel (modello del bottone) che ne mantiene le proprietà
  - ⇨poi servono i componenti di interf. e controllo
  - ⇨interfaccia: un oggetto che disegna il bottone e gestisce l'ambientazione
  - ⇨controllo: un oggetto che gestisce gli eventi relativi al bottone

Sviluppare un Componente Swing

- Nel package javax.swing
  - ⇨varie classi per il componente JButton
  - ⇨ButtonModel: modello (stato del bottone)
  - ⇨ButtonUI: componente per il disegno
  - ⇨ButtonListener: componente per il controllo
  - ⇨JButton: "facciata" per le tre classi precedenti
  - ⇨ne diamo una rappresentazione approssim. perchè le gerarchie sono molto complesse

Sviluppare un Componente Swing

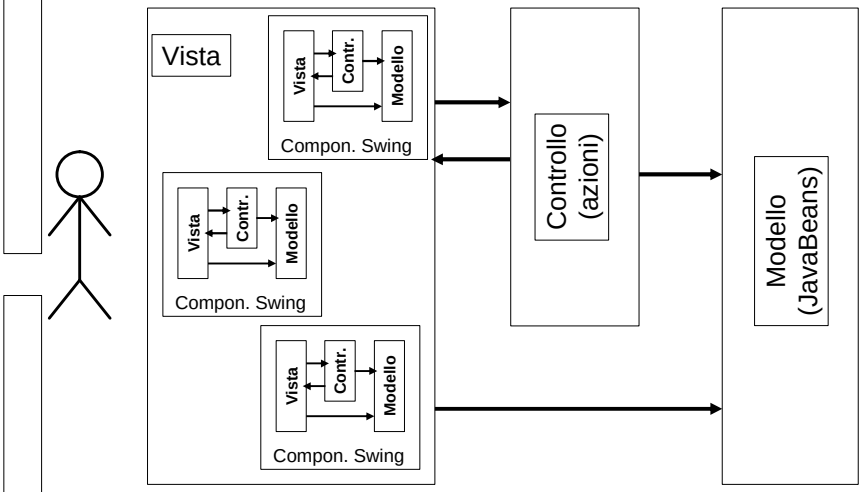
- Schema semplificato dei componenti



Architettura dei Componenti Swing

- Questa organizzazione
  - ⇨viene utilizzata (con varianti) per tutti i componenti di Swing
  - ⇨spesso viene detta architettura MVC
  - ⇨ma a volte anche archit. “separable model”
  - ⇨oppure architettura “UI delegate”
- Attenzione quindi
  - ⇨a non fare confusione tra questa terminologia e la terminologia a livello di strati

Architettura dei Componenti Swing



Architettura dei Componenti Swing

- Vantaggi dell'architettura di Swing
  - ⇨rende molto flessibili le operazioni di disegno dei componenti e di gestione degli eventi
- Un esempio
  - ⇨i componenti Swing sono visualizzabili con ambientazioni ("look&feel") diverse
  - ⇨selezionando l'ambientazione all'avvio
  - ⇨oppure dinamicamente durante il funzionamento dell'applicazione

Architettura dei Componenti Swing

- Selezionare l'ambientazione all'avvio
  - ⇨il metodo static void setLookAndFeel(String laf) della classe javax.swing.UIManager
  - ⇨l'argomento è una stringa che rappresenta il nome del gestore di ambientazione
  - ⇨per l'ambientazione Java: (Ocean) UIManager. getCrossPlatformLookAndFeelClassName()
  - ⇨per l'amb. di "sistema": il metodo UIManager. getSystemLookAndFeelClassName()

Architettura dei Componenti Swing

- Altre ambientazioni disponibili
  - ⇨ambientazione Java precedente a Ocean: "javax.swing.plaf.metal.MetalLookAndFeel"
  - ⇨ambientazione Windows XP: "com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
  - ⇨ambientazione GTK di Linux: "com.sun.java.swing.plaf.gtk.GTKLookAndFeel"
  - ⇨ambientazione Motif di Linux: "com.sun.java.swing.plaf.motif.MotifLookAndFeel"

Architettura dei Componenti Swing

- Selezionare il look & feel dinamicamente
  - ⇨dopo aver chiamato il metodo UIManager.setLookAndFeel(laf) è necessario chiamare due altri metodi
  - ⇨per aggiornare l'albero di componenti del SwingUtilities.updateComponentTreeUI(frame);
  - ⇨infine: frame.pack();

Architettura dei Componenti Swing

- In sintesi
  - ⇨questa architettura è il punto chiave per la flessibilità di Swing
  - ⇨nel caso di componenti semplici, resta "nascosta" dalla facciata del componente
- Viceversa, però...
  - ⇨emerge inevitabilmente nel caso di componenti complessi, rendendo più complessa la programmazione

Architettura dei Componenti Swing

- Esempi di componenti complessi
  - ⇨i componenti che visualizzano collezioni di oggetti
  - ⇨javax.swing.JList: visualizza una lista di voci singole
  - ⇨javax.swing.JTable: visualizza una lista di righe organizzate in colonne

Un Altro Esempio

- it.unibas.mediapesataswing
  - ⇨applicazione per calcolare la media pesata in 30mi e in 110mi di uno studente universitario
  - ⇨esempio di applicazione “orientata ai documenti”: crea, carica e salva un documento contenente i dati dello studente
  - ⇨caratteristica nuova: utilizza JList e JTable (intercambiabilmente) per visualizzare la lista degli esami dello studente

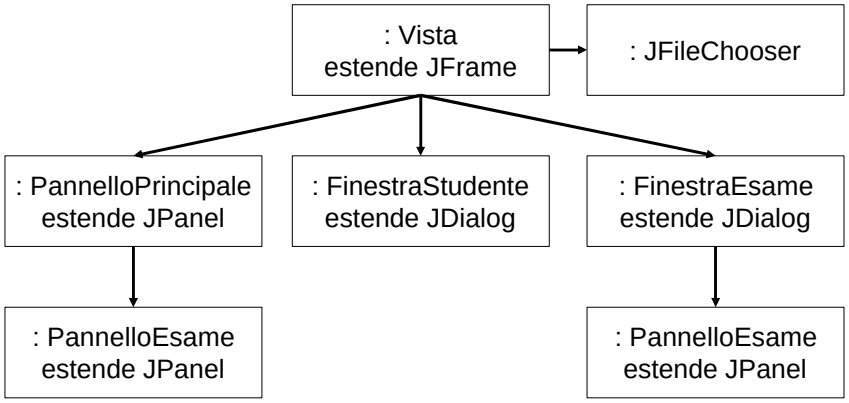
Un Altro Esempio

- Statistiche sul codice
  - ⇨versione console: 505 loc, 7 classi
  - ⇨versione Swing: 1658 loc, 38 classi (!)
  - ⇨organizzato secondo l'architettura MVC vista
  - ⇨al solito il codice è concentrato nello strato di interfaccia e in quello di controllo
  - ⇨16 azioni
  - ⇨gerarchia di viste complessa

Un Altro Esempio

- La gerarchia di viste
  - ⇨un componente principale di tipo JFrame
  - ⇨un componente secondario di tipo JPanel che rappresenta lo schermo principale
  - ⇨utilizza un sottopannello per i dati dell'esame
  - ⇨una finestra secondaria di tipo JDialog per l'inserimento dei dati dello studente
  - ⇨una finestra secondaria di tipo JDialog per aggiornare i dati di un esame
  - ⇨che utilizza un pannello per i dati dell'esame
  - ⇨infine un componente di tipo javax.swing.JFileChooser per aprire e salvare i file

Un Altro Esempio



Un Altro Esempio

- Utilizzo del JFileChooser
  - ⇨consente di selezionare un file dal disco
  - ⇨viene creato un oggetto JFileChooser
  - ⇨consente di aprire finestre diverse
- Finestra di caricamento
  - ⇨int showOpenDialog(Component parent)
- Finestra di salvataggio
  - ⇨int showSaveDialog(Component parent)

Un Altro Esempio

- Entrambi i metodi
  - restituiscono un intero che rappresenta l'esito dell'operazione
  - i possibili valori sono costanti intere di JFileChooser
  - APPROVE_OPTION: l'utente ha premuto il tasto ok per confermare l'operazione
  - CANCEL_OPTION: l'utente ha premuto il tasto annulla per annullare

Un Altro Esempio

- Al termine
  - è possibile conoscere il file selezionato dall'utente con File getSelectedFile()
- Processo tipico
  - l'azione visualizza la finestra di selezione
  - l'utente seleziona un file
  - il file viene passato ad un DAO per eseguire l'operazione di caricamento o salvataggio

Programmazione Grafica: Organizzazione del Codice >> Un Altro Esempio

```
public class AzioneApri extends javax.swing.AbstractAction {

   private Controllo controllo;

   public AzioneApri(Controllo controllo) {
      this.controllo = controllo;
      this.putValue(javax.swing.Action.NAME, "Apri");
      this.putValue(javax.swing.Action.SHORT_DESCRIPTION, "Carica i dati dello studente");
      this.putValue(javax.swing.Action.MNEMONIC_KEY,
                        new Integer(java.awt.event.KeyEvent.VK_A));
   }

   public void actionPerformed(java.awt.event.ActionEvent e) {
      String nomeFile = acquisisciFile();
      if (nomeFile != null) {
         try {
            caricaDatiStudente(nomeFile);
            abilitaAzioni();
         } catch (DAOException daoe) {
            Logger.logSevere("AzioneApri", "actionPerformed", "Impossibile caricare il file: " + daoe);
            this.controllo.getVista().finestraErrore("Impossibile caricare il file " + daoe);
         }
      }
   }
   // ... continua
```


Programmazione Grafica: Organizzazione del Codice >> Un Altro Esempio

```
   // continua
   private String acquisisciFile() {
      JFileChooser fileChooser = this.controllo.getVista().getFileChooser();
      int codice = fileChooser.showOpenDialog(this.controllo.getVista());
      if (codice == JFileChooser.APPROVE_OPTION) {
         File file = fileChooser.getSelectedFile();
         Logger.logInfo("AzioneApri", "acquisisciFile", "Caricamento: " + file.toString());
         return file.toString();
      } else {
         Logger.logInfo("AzioneApri", "acquisisciFile", "Comando apri annullato");
      }
      return null;
   }

   private void caricaDatiStudente(String nomeFile) throws DAOException {
      Studente studente = DAOStudente.carica(nomeFile);
      Modello modello = this.controllo.getModello();
      modello.putBean("studente", studente);
      Vista vista = this.controllo.getVista();
      PannelloPrincipale pannello =
                        (PannelloPrincipale)vista.getSottoVista("pannelloPrincipale");
      pannello.schermoStudente();
   }
```

```
public class AzioneSalva extends javax.swing.AbstractAction {

    private Controllo controllo;

    public AzioneSalva(Controllo controllo) {
        this.controllo = controllo;
        this.putValue(javax.swing.Action.NAME, "Salva");
        this.putValue(javax.swing.Action.SHORT_DESCRIPTION,
                      "Salva i dati dello studente sul disco");
        this.putValue(javax.swing.Action.MNEMONIC_KEY,
                      new Integer(java.awt.event.KeyEvent.VK_S));
    }

    public void actionPerformed(java.awt.event.ActionEvent e) {
        String nomeFile = acquisisciFile();
        if (nomeFile != null) {
            salvaDatiStudente(nomeFile);
        }
    }
    // ... continua
```


```
    // ... continua
    private String acquisisciFile() {
        JFileChooser fileChooser = this.controllo.getVista().getFileChooser();
        int codice = fileChooser.showSaveDialog(this.controllo.getVista());
        if (codice == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            Logger.logInfo("AzioneSalva", "acquisisciFile", "Salvataggio: " + file.toString());
            return file.toString();
        } else {
            Logger.logInfo("AzioneSalva", "acquisisciFile", "Comando salva annullato");
        }
        return null;
    }

    private void salvaDatiStudente(String nomeFile) {
        try {
            Modello modello = this.controllo.getModello();
            Studente studente = (Studente)modello.getBean("studente");
            DAOStudente.salva(studente, nomeFile);
        } catch (DAOException daoe) {
            Logger.logSevere("AzioneSalva", "salvaDatiStudente", "Impossibile caricare il file: " + daoe);
            this.controllo.getVista().finestraErrore("Impossibile salvare il file " + daoe);
        }
    }
```

Visualizzazione di Liste

- Visualizzazione degli esami
  - la prima soluzione consiste nell'utilizzare un componente di tipo javax.swing.JList
- javax.swing.JList
  - basato su due modelli diversi
  - un oggetto di tipo javax.swing.ListModel
  - un oggetto di tipo javax.swing.ListSelectionModel

Visualizzazione di Liste

- L'oggetto ListSelectionModel
  - rappresenta l'attuale selezione (nessuna riga, riga singola, intervallo di righe, intervalli multipli ecc.)
  - così come per i bottoni ed altri componenti semplici viene gestito dal framework
  - normalmente il programmatore non è obbligato ad interagire con l'oggetto

Visualizzazione di Liste

- L'oggetto ListModel
  - ⇨viceversa deve essere creato dal programmatore
  - ⇨per specificare come visualizzare gli oggetti del modello (la lista di esami dello Studente)
- In altri termini
  - ⇨si tratta di un "adattatore" che consente di utilizzare una lista qualsiasi secondo l'interfaccia prevista da JList

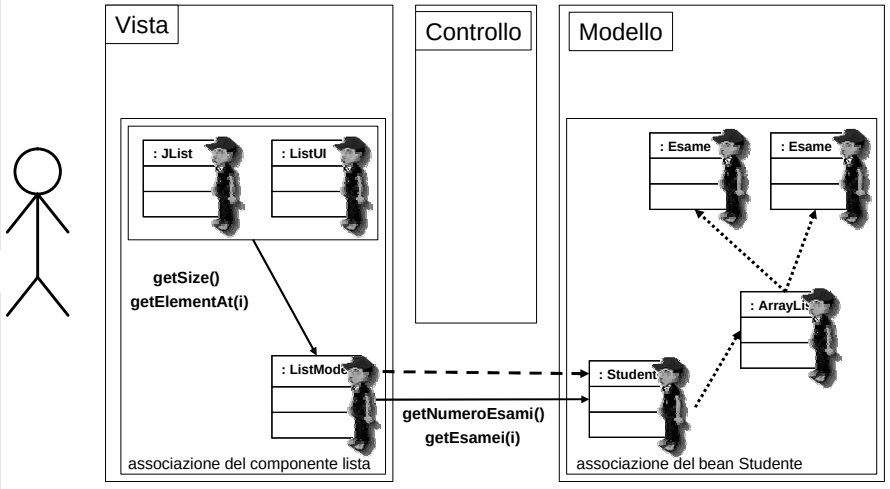
Visualizzazione di Liste

- L'interfaccia javax.swing.ListModel
  - ⇨vari metodi
- javax.swing.AbstractListModel
  - ⇨implementa tutti i metodi in modo standard tranne i metodi per l'acquisizione dei dati
  - ⇨int getSize()
  - ⇨Object getElementAt(int index)

Visualizzazione di Liste

- Come definire questi metodi ?
  - ⇨associando all'oggetto di tipo ListModel al bean Studente
  - ⇨e programmando i metodi getSize() e getElementAt() in modo da delegare l'esecuzione ai corrispondenti metodi del bean

Visualizzazione di Liste



```
package it.unibas.mediapesataswing.vista;

import it.unibas.mediapesataswing.modello.Studente;

public class ModelloLista extends javax.swing.AbstractListModel {

    private Studente studente;

    public ModelloLista(Studente studente) {
        this.studente = studente;
    }

    public int getSize() {
        return this.studente.getNumeroEsami();
    }

    public Object getElementAt(int i) {
        return this.studente.getEsame(i);
    }

}
```


Visualizzazione di Liste

- Per visualizzare la lista
  - ⇨è necessario creare l'oggetto JList
  - ⇨è necessario creare l'oggetto di tipo ModelloLista a partire da uno studente
  - ⇨impostare l'oggetto come modello della lista
  - ⇨è opportuno utilizzare un oggetto di tipo JScrollPane (pannello con barre di scorrim.)
  - ⇨aggiungere l'oggetto JList all'area visibile (viewPort) dello scrollPane

```
package it.unibas.mediapesataswing.vista;

public class PannelloPrincipale extends JPanel {

    private JList jListaEsami = new JList();
    private JScrollPane scrollPane = new JScrollPane();

    private void creaPannelloListaEsami() {
      JPanel pannelloListaEsami = new JPanel();
      Modello modello = this.controllo.getModello();
      Studente studente = (Studente)modello.getBean("studente");
      ModelloLista modelloListaEsami = new ModelloLista(studente);
      this.jListaEsami.setModel(modelloListaEsami);
      this.scrollPane.setViewportView(jListaEsami);
      pannelloListaEsami.add(scrollPane);
      pannelloListaEsami.setBorder(creaBordoTitolo("Lista Esami"));
      this.add(pannelloListaEsami);
    }
```


Visualizzazione di Liste

- A questo punto
  - ⇨la lista visualizza gli esami dello studente
  - ⇨per ogni esame viene visualizzato il risultato della chiamata al metodo toString()
- Ma...
  - ⇨in caso di modifica alla lista degli esami dello studente è necessario fare in modo che il modello della lista rifletta queste modifiche
  - ⇨problema complesso (>>)

Visualizzazione di Liste

- Una soluzione semplice
  - ⇨ma abbastanza inelegante
  - ⇨sostituire il modello di lista ogni volta che viene effettuata una modifica alla lista di esami (inserimenti, cancellazioni, modifiche)
  - ⇨ogni volta che il modello di lista cambia la lista riacquisisce gli elementi e rinfresca automaticamente la visualizzazione

Visualizzazione di Liste

>> AzioneModificaEsame
>> AzioneAggiornaEsame
>> PannelloPrincipale

- Un esempio interessante
  - ⇨la modifica dell'esame
  - ⇨l'utente seleziona un esame dalla lista con il mouse
  - ⇨poi schiaccia il tasto che richiede la modifica
  - ⇨per effettuare la modifica l'azione deve acquisire l'indice dell'esame selezionato
  - ⇨attraverso il metodo getSelectedIndex() di JList (che accede al ListSelectionModel)

Visualizzazione di Liste

- Un approccio alternativo al modello
  - ⇨esistono vari costruttori di JList che ricevono come argomento un array o un Vector
  - ⇨creano un oggetto di tipo DefaultListModel da utilizzare come modello per la vista
  - ⇨il DefaultListModel utilizza una propria collezione di tipo Vector interna invece di acquisire dati da un bean

Visualizzazione di Liste

- Vantaggio di questo approccio
  - ⇨dopo aver aggiornato la lista degli esami potrei chiamare i metodi opportuni sul DefaultListModel (set, add, remove) per fare gli aggiornamenti senza ricreare il modello
- Svantaggio di questo approccio
  - ⇨all'inizio devo "copiare" i riferimenti dalla lista degli esami nella collezione utilizzata dal modello

Visualizzazione di Tabelle

- In alternativa alla lista
  - ⇨gli esami possono essere visualizzati sotto forma di tabella
- javax.swing.JTable
  - ⇨componente molto sofisticato
  - ⇨consente un'interazione molto ampia (selezione di righe, intervalli, modifica di celle, ridimensionamento, ordinamento, spostamento di colonne)

Visualizzazione di Tabelle

- I modelli di JTable
  - ⇨un modello di tipo TableModel che mantiene i dati visualizzati nella tabella
  - ⇨un modello di tipo ListSelectionModel identico a quello usato per le liste per rappresentare la selezione delle righe
- Inoltre
  - ⇨oggetti di tipo CellRenderer per stabilire come rendere graficamente le singole celle

Visualizzazione di Tabelle

- javax.swing.table.TableModel
  - ⇨interfaccia che prevede molti metodi
- javax.swing.table.AbstractTableModel
  - ⇨tre metodi fondamentali da implementare
  - ⇨int getRowCount()
  - ⇨int getColumnCount()
  - ⇨Object getValueAt(int riga, int colonna)

Visualizzazione di Tabelle

- Altri metodi utili per la visualizzazione
  - ⇨String getColumnName(int indice): l'implementazione standard usa A, B, C...
  - ⇨Class getColumnClass(int indice): l'implementazione standard usa Object.class; alternative: Boolean, Image, Number...
  - ⇨boolean isCellEditable(int riga, int colonna): l'implementazione standard restituisce false per tutte le celle

```
public class ModelloTabella extends javax.swing.table.AbstractTableModel {

   private Studente studente;

   public ModelloTabella (Studente studente) {
      this.studente = studente;
   }

   public int getRowCount() {
      return studente.getNumeroEsami();
   }

   public int getColumnCount() { return 4; }

    public Class getColumnClass(int i) {
      if (i == 3) {
         return Boolean.class;
      }
      return super.getColumnClass(i); // Object.class
   }
  // continua
```


```
   // ... continua
   public String getColumnName(int i) {
      if (i == 0) { return "Insegnamento"; }
      else if (i == 1) { return "Crediti"; }
      else if (i == 2) { return "Voto"; }
      else if (i == 3) { return "Lode"; }
      return null;
   }

   public Object getValueAt(int i1, int i2) {
      if (i2 == 0) { return studente.getEsame(i1).getInsegnamento(); }
      else if (i2 == 1) { return studente.getEsame(i1).getCrediti(); }
      else if (i2 == 2) { return studente.getEsame(i1).getVoto(); }
      else if (i2 == 3) { return studente.getEsame(i1).isLode(); }
      return null;
   }
}
```

Visualizzazione di Tabelle

- L'aggiornamento della tabella
  - ⇨simile a quello della lista
  - ⇨un po' grossolanamente, ogni volta che la lista degli esami cambia, viene rigenerato il modello
- In questo caso
  - ⇨si vedono molto di più gli svantaggi di questo approccio (se l'utente ha modificato la visualizz. della tabella perde le modifiche)

Visualizzazione di Tabelle

- Anche in questo caso
  - ⇨sarebbe possibile creare un oggetto di tipo DefaultTableModel
  - ⇨che utilizza un Vector di riferimenti a Vector per rappresentare il contenuto della tabella
  - ⇨fornisce tutti i metodi per l'aggiornamento
  - ⇨è necessario copiare il contenuto della lista degli esami nel Vector di Vector

Il Problema del Binding

- Il problema di cui stiamo parlando
  - ⇨è un problema centrale della programmazione grafica
  - ⇨un problema di collegamento (“binding”)
- Binding
  - ⇨operazione con cui un valore mantenuto nel modello dell’applicazione (nei bean) viene collegato ad un componente grafico per la visualizzazione

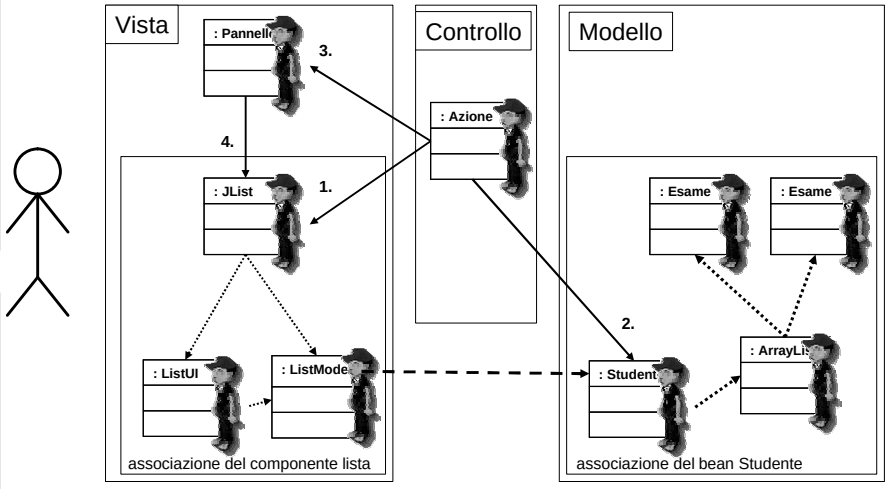
Il Problema del Binding

- In effetti
  - ⇨a ben guardare la maggior parte del codice della vista è dedicato alle operazioni di binding
- Esempi
  - ⇨le JLabel/JTextField che visualizzano stringhe dei bean
  - ⇨le JList/JTable che visualizzano collezioni del modello

Il Problema del Binding

- L'approccio che abbiamo usato
  - ⇨copia esplicita dei valori
- Copia esplicita dei valori
  - ⇨l'azione preleva dati dal componente (es: tentativo dell'utente)
  - ⇨aggiorna il modello
  - ⇨al termine la vista aggiorna i componenti (es: numero di tentativi)

Il Problema del Binding



Il Problema del Binding

- Vantaggio di questo approccio
  - ⇨semplice concettualmente
- Svantaggio di questo approccio
  - ⇨il codice della vista si allunga notevolmente per tenere “sincronizzati” i componenti
  - ⇨molto scomodo se più componenti visualizzino le stesse proprietà (es: barra di scorrimento per i tentativi) – copie multiple
  - ⇨nelle applicazioni complesse è un problema

Riassumendo

- MVC e Swing
  - ⇨Sviluppare un Componente Swing
  - ⇨Architettura dei Componenti Swing
  - ⇨Problemi di Questa Architettura
- Un Altro Esempio
  - ⇨Visualizzazione di Liste
  - ⇨Visualizzazione di Tabelle
- Il Problema del “Binding”

Termini della Licenza