13. Interfacce Utente Grafiche (GUI) con Swing

Questo capitolo introduce le **Graphical User Interfaces (GUI)** in Java, con un focus specifico sulla libreria **Swing**. Verranno esplorati i concetti fondamentali, i componenti principali, la gestione degli eventi e i pattern di progettazione consigliati per le applicazioni grafiche.

7.1. Introduzione alle Interfacce Utente Grafiche (GUI)

Le GUI sono interfacce grafiche progettate per l'interazione con l'utente, ritenute generalmente più semplici da utilizzare rispetto alle interfacce a riga di comando (CUI - Console User Interfaces). Sfruttano la capacità di disegnare pixel sullo schermo e interagire con dispositivi come mouse e tastiera, basandosi su astrazioni grafiche come pulsanti, icone e finestre.

- *Provenienza*: 17-swing_slides.pdf, Pagina 5

7.1.1. Gestione delle GUI in Java: AWT, Swing e Alternative

Storicamente, Java ha offerto diverse librerie per lo sviluppo di GUI:

- **Abstract Window Toolkit (AWT)**: Introdotta nelle prime versioni di Java (Java 1 e 2), è una libreria di basso livello che si basa sui *peer* nativi del sistema operativo. Questo significa che i componenti AWT sono disegnati e gestiti direttamente dal sistema operativo sottostante, il che può portare a un aspetto e un comportamento incoerenti su piattaforme diverse.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 5
- **Java Swing**: Introdotta in Java 2 (e stabilizzata da Java 5 in poi), è stata costruita sopra AWT. A differenza di AWT, Swing disegna i suoi componenti interamente in Java ("lightweight components"), il che garantisce un aspetto e un comportamento più coerente su tutte le piattaforme (look-and-feel pluggable). Swing è stata la libreria standard per le GUI desktop in Java per molti anni.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 5
- **Alternative Moderne**:
  - **JavaFX**: Consigliata per applicazioni moderne, offre un approccio più contemporaneo allo sviluppo GUI, con supporto per grafica avanzata, animazioni, FXML (linguaggio di markup per la GUI) e CSS per lo stile.
  - **SWT (Standard Widget Toolkit)**: Utilizzata da Eclipse, è un'altra libreria GUI che, come AWT, si basa sui widget nativi del sistema operativo per un aspetto più integrato.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 5

7.2. Architettura e Classi Principali di Swing

Swing è organizzato in una gerarchia di classi che estendono i componenti AWT, aggiungendo funzionalità e un look-and-feel indipendente dalla piattaforma.

7.2.1. I Package `java.awt` e `javax.swing`

- `java.awt`: Contiene le classi base per l'architettura delle GUI, come `Component` (il genitore di tutti i componenti grafici) e `Container` (un componente che può contenere altri componenti).
- `javax.swing`: Contiene le implementazioni "lightweight" di Swing, che disegnano i componenti "pixel per pixel" in Java. Le classi Swing hanno tipicamente il prefisso "J" (es. `JFrame`, `JButton`).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 7

7.2.2. Componenti Top-Level di Swing

I componenti top-level sono finestre indipendenti che non sono contenute in altri componenti. Sono i punti di ingresso per le applicazioni GUI.

- `JFrame`: La finestra principale di un'applicazione Swing. Include una "cornice" con barra del titolo, pulsanti di minimizzazione/massimizzazione/chiusura, e un'icona. È il contenitore più comune per i componenti Swing.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 7

- `JWindow`: Una finestra senza cornice o barra del titolo, utile per schermate di avvio (splash screen) o finestre pop-up.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 7
- `JDialog`: Una finestra di dialogo, tipicamente usata per interagire con l'utente e ottenere input specifico (es. messaggi di errore, conferme). Può essere modale (blocca l'interazione con altre finestre) o non modale.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 7
- `JApplet`: Una finestra per le applet Java, che erano applicazioni eseguite all'interno di un browser web. (Nota: le applet sono obsolete e non più supportate dai browser moderni).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 6 (diagramma UML)

7.2.3. Contenitori Intermedi e Componenti Atomici

- `JPanel`: Un contenitore leggero e generico, spesso usato per raggruppare altri componenti e organizzare il layout all'interno di un `JFrame` o `JDialog`.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 7
- `JComponent`: La classe base astratta per la maggior parte dei componenti Swing. Fornisce funzionalità comuni come il disegno, la gestione degli eventi, i bordi e i tooltip.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 7

Esempi di `JComponent` comuni:

- `JButton`: Un pulsante cliccabile.
- `JLabel`: Un'etichetta di testo o immagine non modificabile.
- `JTextField`: Un campo di testo a riga singola per l'input dell'utente.
- `JTextArea`: Un'area di testo multi-riga.
- `JList`: Un componente che visualizza una lista di elementi selezionabili.
- `JComboBox`: Un menu a discesa (combo box).
- `JCheckBox`: Una casella di controllo.
- `JRadioButton`: Un pulsante radio (spesso usato in gruppi con `ButtonGroup`).
- `JSlider`: Un cursore per selezionare un valore in un intervallo.
- `JProgressBar`: Una barra di progresso.
- `JMenuBar`, `JMenu`, `JMenuItem`: Componenti per creare menu.
- *Provenienza*: 17-swing_slides.pdf, Pagina 10, 12 (diagramma UML)

7.2.4. Esempio Base di un'Applicazione Swing

```
import javax.swing.*; // Importa tutte le classi Swing

public class TrySwing {
    public static void main(String[] args) {
        // 1. Creo il frame (la finestra principale)
        final JFrame frame = new JFrame();
        frame.setTitle("Prova di JFrame"); // Imposto il titolo della finestra

        // 2. Imposto l'operazione di default alla chiusura della finestra
        // DO_NOTHING_ON_CLOSE: non fa nulla (richiede gestione manuale)
        // HIDE_ON_CLOSE: nasconde la finestra
        // DISPOSE_ON_CLOSE: chiude la finestra e rilascia le risorse
        // EXIT_ON_CLOSE: termina l'applicazione Java (il più comune per app semplici)
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // 3. Imposto le dimensioni della finestra
        frame.setSize(320, 240);
```

```
        // 4. Creo un pannello (contenitore intermedio)
        final JPanel panel = new JPanel();

        // 5. Aggiungo il pannello al Content Pane del frame
        // Il Content Pane è il contenitore principale di un JFrame
        frame.getContentPane().add(panel);

        // 6. Aggiungo un pulsante al pannello
        panel.add(new JButton("Prova di pulsante"));

        // 7. Rendo visibile il JFrame
        frame.setVisible(true);
    }
}
```

- *Provenienza*: 17-swing_slides.pdf, Pagina 9

7.3. Gestione del Layout dei Componenti

Il layout di un pannello o di un frame definisce come i componenti sono posizionati e dimensionati al suo interno. Swing utilizza i **Layout Manager** per gestire la politica di dislocazione dei componenti in modo indipendente dalle dimensioni della finestra.

7.3.1. Il Pattern Strategy e `LayoutManager`

I Layout Manager in Swing implementano il **Pattern Strategy**. Un oggetto `LayoutManager` incapsula la strategia di inserimento e posizionamento dei componenti. Al pannello (o a qualsiasi `Container`) si associa un `LayoutManager`.

- **`Container.setLayout(LayoutManager lm)`**: Metodo per impostare il layout manager.
- **`Container.add(Component comp, Object constraints)`**: Il metodo `add()` può accettare un argomento aggiuntivo (es. una costante per `BorderLayout`) che il layout manager utilizza per posizionare il componente.
- *Provenienza*: 17-swing_slides.pdf, Pagina 15

7.3.2. Tipi Comuni di Layout Manager

- **`FlowLayout`**: Il layout manager di default per `JPanel`. Posiziona i componenti in una riga, da sinistra a destra, e va a capo quando non c'è più spazio. I componenti mantengono le loro dimensioni preferite.
  - **Caratteristiche**: I componenti sono disposti in un flusso, come il testo in un paragrafo.
  - **Costruttori**: `FlowLayout()`, `FlowLayout(int align)`, `FlowLayout(int align, int hgap, int vgap)`. `align` può essere `FlowLayout.LEFT`, `CENTER` (default), `RIGHT`. `hgap` e `vgap` definiscono lo spazio orizzontale e verticale tra i componenti.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 20
- **`BorderLayout`**: Il layout manager di default per `JFrame` e `JDialog`. Divide il contenitore in cinque regioni: `NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`. Ogni regione può contenere un solo componente.
  - **Caratteristiche**:
    - `NORTH` e `SOUTH` usano l'altezza preferita del componente e si estendono orizzontalmente.
    - `EAST` e `WEST` usano la larghezza preferita del componente e si estendono verticalmente.
    - `CENTER` occupa lo spazio rimanente.
  - **Costruttori**: `BorderLayout()`, `BorderLayout(int hgap, int vgap)`.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 17, 19
- **`GridLayout`**: Organizza i componenti in una griglia di dimensioni uguali. Ogni cella contiene un componente.
  - **Caratteristiche**: Tutti i componenti hanno la stessa dimensione.
  - **Costruttori**: `GridLayout()`, `GridLayout(int rows, int cols)`, `GridLayout(int rows, int cols, int hgap, int vgap)`.

- `GridBagLayout`: Il layout manager più potente e flessibile, ma anche il più complesso. Permette di posizionare i componenti in una griglia, ma con celle di dimensioni variabili e la possibilità di estendere i componenti su più celle. Richiede l'uso di `GridBagConstraints` per definire le proprietà di ogni componente (es. `gridx`, `gridy`, `gridwidth`, `gridheight`, `fill`, `insets`).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 23
- **Layout Nullo (`null` layout)**: Si può impostare il layout manager a `null` (`panel.setLayout(null)`). In questo caso, il programmatore è responsabile di impostare manualmente la posizione e le dimensioni di ogni componente (`setBounds(x, y, width, height)`). Questo approccio è **deprecabile** perché non è flessibile al ridimensionamento della finestra e rende il codice più rigido.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 16

7.3.3. Combinazione di Layout Manager

Spesso, per creare interfacce complesse, si combinano diversi layout manager. Questo si fa creando `JPanel` separati, ognuno con il proprio layout manager, e poi aggiungendo questi pannelli come componenti a un pannello o frame principale con un altro layout manager.

Esempio:
Un `JFrame` con `BorderLayout` può avere un `JPanel` con `FlowLayout` nella regione `NORTH` e un altro `JPanel` con `FlowLayout` nella regione `SOUTH`.

- *Provenienza*: 17-swing_slides.pdf, Pagina 22

7.4. Gestione degli Eventi nelle GUI (Programmazione ad Eventi)

Le interfacce grafiche sono "reattive" agli eventi generati dall'utente (click del mouse, digitazione da tastiera, ecc.). La gestione degli eventi in Swing si basa sul **Pattern Observer**.

7.4.1. Il Pattern Observer

- **Subject (Sorgente dell'Evento)**: Il componente GUI che genera l'evento (es. un `JButton`). La sorgente dell'evento mantiene una lista di "ascoltatori" (listeners) registrati.
- **Observer (Listener)**: Un oggetto che "ascolta" gli eventi generati dalla sorgente. Implementa un'interfaccia specifica del listener (es. `ActionListener`).
- **Registrazione**: L'Observer si registra con il Subject tramite un metodo `add<EventType>Listener()` (es. `addActionListener()`).
- **Notifica**: Quando un evento accade, il Subject invoca un metodo specifico su tutti i listener registrati, passando un oggetto `Event` che contiene i dettagli dell'evento.
- *Provenienza*: 17-swing_slides.pdf, Pagina 28, 29

7.4.2. Esempio: Click sui Pulsanti con `ActionListener`

Per gestire i click su un `JButton`, si usa l'interfaccia funzionale `ActionListener` e il metodo `actionPerformed(ActionEvent e)`.

- `JButton.addActionListener(ActionListener listener)`: Metodo per registrare un listener.
- `ActionEvent`: Oggetto che incapsula i dettagli dell'evento (es. `getActionCommand()` per ottenere un comando associato al pulsante).

Modi per implementare un `ActionListener`:

1. **Classe Esterna Separata**: Creare una classe separata che implementa `ActionListener`.
   - *Vantaggio*: Riusabilità se lo stesso listener è usato da più componenti.
   - *Svantaggio*: Può essere verboso per listener semplici.
   - *Provenienza*: 17-swing_slides.pdf, Pagina 31, 33
2. **Inner Class (Classe Interna)**: Definire il listener come una classe interna non statica all'interno della classe che gestisce la GUI.
   - *Vantaggio*: Ha accesso diretto ai membri (anche privati) della classe esterna (`OuterClass.this`).
   - *Svantaggio*: Ancora un po' verboso.
   - *Provenienza*: 17-swing_slides.pdf, Pagina 34
3. **Anonymous Class (Classe Anonima)**: Definire il listener "al volo" come una classe anonima.

- *Vantaggio*: Molto concisa per listener semplici usati una sola volta.
- *Svantaggio*: Può diventare illeggibile per logiche complesse.
- *Provenienza*: 17-swing_slides.pdf, Pagina 35

4. **Lambda Expression (Java 8+)**: Il modo più conciso per implementare interfacce funzionali come `ActionListener`.
   - *Vantaggio*: Massima concisione e leggibilità per logiche semplici.
   - *Provenienza*: 17-swing_slides.pdf, Pagina 36

Esempio con Lambda:

```
import java.awt.*;
import javax.swing.*;

public class UseButtonEvents3 {
   public static void main(String[] args) {
      final JButton b1 = new JButton("Say Hello");
      // Listener con lambda: quando il pulsante è cliccato, stampa "Hello!!"
      b1.addActionListener(e -> System.out.println("Hello!!"));

      final JButton b2 = new JButton("Quit");
      // Listener con lambda e blocco di codice: quando cliccato, stampa "Quitting.." ed esce
      b2.addActionListener(e -> {
         System.out.println("Quitting..");
         System.exit(0);
      });

      final JFrame frame = new JFrame("Events Example");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(300, 150);
      frame.setLayout(new FlowLayout()); // Imposto un FlowLayout
      frame.add(b1);
      frame.add(b2);
      frame.setVisible(true);
   }
}
```

7.4.3. Panoramica Eventi-Listeners Comuni

Swing offre un'ampia gamma di interfacce listener per diversi tipi di eventi:

- `ActionListener`: Per eventi di azione (es. click su pulsante, invio in campo di testo).
- `MouseListener`: Per eventi del mouse (click, pressione, rilascio, entrata/uscita).
- `MouseMotionListener`: Per eventi di movimento del mouse (trascinamento, movimento).
- `KeyListener`: Per eventi da tastiera (pressione, rilascio, digitazione di un carattere).
- `WindowListener`: Per eventi della finestra (apertura, chiusura, minimizzazione, ecc.).
- `ComponentListener`: Per eventi di cambiamento di dimensione, posizione, visibilità di un componente.
- `FocusListener`: Per eventi di focus (quando un componente ottiene o perde il focus).
- `ItemListener`: Per eventi di selezione di un elemento (es. `JCheckBox`, `JComboBox`).

Per ogni interfaccia listener, esiste spesso una classe "Adapter" (es. `MouseAdapter`, `WindowAdapter`) che fornisce implementazioni vuote di tutti i metodi dell'interfaccia. Questo è utile quando si vuole implementare solo uno o pochi metodi dell'interfaccia, evitando di dover implementare tutti i metodi inutilizzati.

- *Provenienza*: 17-swing_slides.pdf, Pagina 37

7.4.4. Il Thread di Dispatch degli Eventi (EDT)

In Swing, tutte le operazioni che modificano i componenti della GUI devono essere eseguite sul **Event Dispatch Thread (EDT)**. Questo è un singolo thread responsabile di elaborare tutti gli eventi GUI e di aggiornare lo schermo.

- **Problema**: Se un listener esegue un'operazione lunga o bloccante sull'EDT, l'interfaccia utente diventerà non responsiva ("freezerà") fino a quando l'operazione non sarà completata.
- **Soluzione**: Per operazioni lunghe, è necessario spostare l'esecuzione su un thread separato (es. usando `SwingWorker` o `ExecutorService`) e poi utilizzare `SwingUtilities.invokeLater()` per riportare eventuali aggiornamenti della GUI sull'EDT.
- *Provenienza*: 17-swing_slides.pdf, Pagina 38

7.5. Funzionalità Avanzate delle GUI Swing

7.5.1. GUI con I/O: Modificare l'Interfaccia

I listener possono interagire con i componenti della GUI per leggere input e aggiornare l'output.

Esempio: Moltiplicare un numero in un `JTextField`

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class UseIOGUI {
   public static void main(String[] args) {
      final JTextField tf = new JTextField(10); // Campo di testo per input
      final JLabel lb = new JLabel("Result: 0"); // Etichetta per output
      final JButton bt = new JButton("Multiply by 2"); // Pulsante per l'azione

      bt.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            try {
               String s = tf.getText(); // Leggo il testo dal campo
               int n = Integer.parseInt(s); // Converto in intero
               lb.setText("Result: " + n * 2); // Aggiorno l'etichetta con il risultato
            } catch (NumberFormatException ex) {
               JOptionPane.showMessageDialog(null, "Please enter an integer!", "Input Error", JOptionPane.ERROR_MES
SAGE);
            }
         }
      });

      final JFrame frame = new JFrame("I/O Example");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(300, 150);
      frame.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10)); // Layout con spaziature
      frame.add(tf);
      frame.add(lb);
      frame.add(bt);
      frame.setVisible(true);
   }
}
```

- *Provenienza*: 17-swing_slides.pdf, Pagina 40

7.5.2. GUI con Layout Dinamico

Quando si aggiungono o rimuovono componenti a un pannello dopo la sua visualizzazione iniziale, è necessario forzare il ricalcolo del layout.

- `Container.validate()`: Forzail ricalcolo del layout del contenitore e dei suoi figli.
- `Component.revalidate()`: Invalida il componente e i suoi genitori, forzando un ricalcolo del layout.

- `Component.repaint()`: Richiede che il componente venga ridisegnato.

Esempio: Aggiungere pulsanti dinamicamente

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class UseDynamicLayout {
   public static void main(String[] args) {
      final FlowLayout lay = new FlowLayout(FlowLayout.CENTER, 10, 10);
      final JFrame frame = new JFrame("Adding Buttons");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(400, 200);
      frame.setLayout(lay); // Imposto il FlowLayout direttamente sul frame
      final JPanel panel = (JPanel) frame.getContentPane(); // Ottengo il content pane come JPanel

      final JButton bt = new JButton("Add a Button");
      bt.addActionListener(new ActionListener() {
         int count = 0;
         public void actionPerformed(ActionEvent e) {
            panel.add(new JButton("Button" + count++)); // Aggiungo un nuovo pulsante
            panel.validate(); // Forza il ricalcolo del layout!
            // panel.repaint(); // Spesso necessario anche un repaint
         }
      });
      panel.add(bt);
      frame.setVisible(true);
   }
}
```

- *Provenienza*: 17-swing_slides.pdf, Pagina 41

7.5.3. Utilizzo di un Pannello come Canvas per il Disegno

È possibile specializzare un `JPanel` per disegnare grafici personalizzati, sovrascrivendo il metodo `paintComponent(Graphics g)`.

- `paintComponent(Graphics g)`: Questo metodo viene invocato automaticamente da Swing quando il componente deve essere disegnato. Il parametro `Graphics` fornisce metodi per disegnare forme, testo, immagini. È fondamentale chiamare `super.paintComponent(g)` all'inizio per garantire che lo sfondo e i bordi del componente siano disegnati correttamente.
- `repaint()`: Invocato per richiedere a Swing di ridisegnare il componente. Questo non invoca `paintComponent` direttamente, ma lo schedula sull'EDT.

Esempio: Disegnare cerchi casuali su un `JPanel`

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

// Classe DrawPanel (specializzazione di JPanel per il disegno)
class DrawPanel extends JPanel {
   private static final int RADIUS = 30;
   private static final Random RND = new Random();
   private final Map<Point, Color> circles = new HashMap<>();

   // Override del metodo di disegno
```

```
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g); // Chiama il metodo della superclasse
        for (Map.Entry<Point, Color> e : this.circles.entrySet()) {
            g.setColor(e.getValue()); // Imposta il colore
            g.fillOval(e.getKey().x, e.getKey().y, RADIUS, RADIUS); // Disegna un cerchio pieno
        }
    }

    // Metodo per aggiungere nuovi cerchi in posizione random
    public void addRandomPoint() {
        int x = RND.nextInt(this.getWidth());
        int y = RND.nextInt(this.getHeight());
        this.addPoint(x, y);
    }

    // Metodo per aggiungere nuovi cerchi in una posizione specifica
    public void addPoint(int x, int y) {
        final Color c = new Color(RND.nextInt(256), RND.nextInt(256), RND.nextInt(256));
        // Salvo il punto in alto a sinistra del cerchio
        this.circles.put(new Point(x - RADIUS / 2, y - RADIUS / 2), c);
    }
}

public class UseCanvas {
    public static void main(String[] args) {
        final DrawPanel pCenter = new DrawPanel();
        pCenter.setBorder(new TitledBorder("Circles here.."));

        // Intercetto i click del mouse sul pannello
        pCenter.addMouseListener(new MouseAdapter() { // Uso un MouseAdapter per implementare solo mouseClicked
            @Override
            public void mouseClicked(MouseEvent e) {
                pCenter.addPoint(e.getX(), e.getY()); // Aggiungo un cerchio nella posizione del click
                pCenter.repaint(); // Richiedo il ridisegno del pannello
            }
        });

        // Pannello per il pulsante "Draw"
        final JPanel pEast = new JPanel(new FlowLayout());
        final JButton bt = new JButton("Draw");
        bt.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                pCenter.addRandomPoint(); // Aggiungo un cerchio in posizione casuale
                pCenter.repaint(); // Richiedo il ridisegno
            }
        });
        pEast.add(bt);

        final JFrame frame = new JFrame("Canvas Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);
        frame.setLayout(new BorderLayout());
        frame.add(pCenter, BorderLayout.CENTER); // Pannello di disegno al centro
        frame.add(pEast, BorderLayout.EAST);     // Pulsante a destra
        frame.setResizable(true);
        frame.setVisible(true);
    }
```

```
}
```

- *Provenienza*: 17-swing_slides.pdf, Pagina 42-43

7.5.4. Uso delle Finestre di Dialogo (`JOptionPane`)

`JOptionPane` è una classe di utilità che semplifica la creazione di finestre di dialogo standard per messaggi, input e conferme.

Metodi comuni:

- **`showMessageDialog(Component parentComponent, Object message, String title, int messageType)`**: Visualizza un messaggio. `messageType` può essere `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `ERROR_MESSAGE`, `QUESTION_MESSAGE`, `PLAIN_MESSAGE`.
- **`showConfirmDialog(Component parentComponent, Object message, String title, int optionType)`**: Visualizza una domanda e offre opzioni (es. YES/NO, OK/CANCEL). Restituisce un intero che indica la scelta dell'utente.
- **`showInputDialog(Component parentComponent, Object message, String title, int messageType)`**: Richiede input testuale dall'utente.
- **`showOptionDialog(...)`**: Il metodo più flessibile per creare dialoghi personalizzati con un array di opzioni.

Esempio: Dialogo di conferma alla chiusura della finestra

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class UseDialogs {
   public static void main(String[] args) {
      final JFrame frame = new JFrame("Dialogs Example");
      frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE); // Non chiude automaticamente

      // Aggiungo un WindowListener per intercettare l'evento di chiusura
      frame.addWindowListener(new WindowAdapter() {
         @Override
         public void windowClosing(WindowEvent e) {
            int n = JOptionPane.showConfirmDialog(frame,
                  "Do you really want to quit?", "Quitting..", JOptionPane.YES_NO_OPTION);
            if (n == JOptionPane.YES_OPTION) {
               System.exit(0); // Termina l'applicazione se l'utente conferma
            }
         }
      });

      // ... (altri componenti e logica come nell'esempio UseIOGUI)
      final JTextField tf = new JTextField(10);
      final JLabel lb = new JLabel("Result: 0");
      final JButton bt = new JButton("Multiply by 2");
      bt.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            try {
               lb.setText("Result: " + Integer.parseInt(tf.getText()) * 2);
            } catch (Exception ex) {
               JOptionPane.showMessageDialog(frame, "An integer please..", "Input Error", JOptionPane.ERROR_MESSA
GE);
            }
         }
      });

      frame.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
      frame.add(tf);
      frame.add(lb);
```

```
        frame.add(bt);
        frame.pack(); // Adatta la dimensione del frame al contenuto
        frame.setVisible(true);
    }
}
```

- *Provenienza*: 17-swing_slides.pdf, Pagina 44

7.5.5. GUI Builders

I GUI Builders (o form designers) sono strumenti software che permettono di creare interfacce grafiche in modalità **WYSIWYG (What You See Is What You Get)**, trascinando e rilasciando i componenti su una tela. Generano automaticamente il codice sorgente della GUI.

- **Vantaggi**: Accelerano lo sviluppo di GUI complesse, permettono ai designer di contribuire al layout.
- **Svantaggi**: Il codice generato può essere difficile da leggere o modificare manualmente, e a volte può non essere ottimale. È importante comprendere e criticare il codice prodotto.
- **Esempi**: WindowBuilder (plugin per Eclipse), NetBeans GUI Builder.
- *Provenienza*: 17-swing_slides.pdf, Pagina 45-46

7.6. Organizzazione delle Applicazioni Grafiche con MVC

Il pattern architetturale **Model-View-Controller (MVC)** è un approccio strutturato e ampiamente raccomandato per progettare applicazioni interattive, specialmente quelle con GUI. Promuove il riuso, la modularità, l'estensibilità e la manutenibilità del software separando i "concern" (le preoccupazioni).

7.6.1. Descrizione del Pattern MVC

MVC divide l'applicazione in tre parti principali:

- **Model (Modello)**: Incapsula la logica di business e lo stato dell'applicazione. Non ha alcuna conoscenza dell'interfaccia utente. È indipendente dalla View e dal Controller.
  - *Metodi*: Metodi di "dominio", chiamati dal Controller (es. `reset()`, `attempt(int n)` in un gioco).
- **View (Vista)**: Gestisce la visualizzazione dei contenuti e le interazioni con l'utente (input e output). Si occupa di come i dati vengono presentati e di come l'utente interagisce con essi.
  - *Metodi*: Metodi (void) chiamati dal Controller per richiedere la visualizzazione (es. `result(DrawResult res)`, `numberIncorrect()`).
- **Controller (Controllore)**: Gestisce il coordinamento tra Model e View. Intercetta gli eventi dalla View, comanda le modifiche al Model e aggiorna di conseguenza la View.
  - *Metodi*: Metodi (void) chiamati dalla View, che esprimono "azioni utente" (es. `newAttempt(int n)`, `resetGame()`, `quit()`).
- *Provenienza*: 17-swing_slides.pdf, Pagina 48, 51

7.6.2. Interazioni tra Componenti MVC

Le interazioni seguono un flusso ben definito:

1. L'utente interagisce con la **View** (es. clicca un pulsante).
2. La **View** notifica il **Controller** dell'evento (tramite un'interfaccia `ViewObserver` o un listener).
3. Il **Controller** elabora l'evento, interagendo con il **Model** per aggiornare lo stato dell'applicazione.
4. Il **Model** (o il Controller, a seconda della variante MVC) notifica la **View** dei cambiamenti nello stato.
5. La **View** si aggiorna per riflettere il nuovo stato del Model.

Questo disaccoppiamento rende ogni parte più facile da testare, modificare e riutilizzare.

- *Provenienza*: 17-swing_slides.pdf, Pagina 49

7.6.3. Esempio di Applicazione MVC: "DrawNumber"

Il PDF presenta un esempio dettagliato di un'applicazione "DrawNumber" (indovina il numero) implementata con il pattern MVC.

- `DrawNumber` **(Model Interface)**: Definisce le operazioni di dominio (es. `reset()`, `attempt(int n)`).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 53
- `DrawNumberImpl` **(Model Implementation)**: Implementa la logica del gioco, mantenendo lo stato (numero da indovinare, tentativi rimanenti).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 54-55
- `DrawNumberView` **(View Interface)**: Definisce le operazioni per visualizzare informazioni e notificare il Controller (es. `setObserver()`, `start()`, `result()`, `numberIncorrect()`, `limitsReached()`).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 56
- `DrawNumberViewObserver` **(Controller Interface/Listener)**: Interfaccia implementata dal Controller per ricevere notifiche dalla View (es. `newAttempt(int n)`, `resetGame()`, `quit()`).
  - *Provenienza*: 17-swing_slides.pdf, Pagina 56
- `DrawNumberViewImpl` **(View Implementation)**: Implementa la GUI usando componenti Swing (JFrame, JPanel, JButton, JTextField, JLabel, JOptionPane). Contiene la logica per mostrare i messaggi e gestire gli input.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 57-59
- `DrawNumberApp` **(Controller Implementation)**: Implementa `DrawNumberViewObserver`. Collega il Model e la View, gestendo il flusso di controllo. Quando riceve un evento dalla View, chiama il Model e poi aggiorna la View in base al risultato.
  - *Provenienza*: 17-swing_slides.pdf, Pagina 61-62

Questo esempio dimostra come le responsabilità siano chiaramente separate: il Model non sa nulla della GUI, la View non sa nulla della logica del gioco, e il Controller fa da mediatore.

7.6.4. Linee Guida per MVC

- **Progettare le 3 Interfacce**: Definire chiaramente le interfacce per Model, View e Controller prima dell'implementazione.
- **Tecnologia GUI Interna alla View**: La tecnologia specifica della GUI (es. Swing) dovrebbe essere contenuta all'interno dell'implementazione della View e non dovrebbe essere menzionata altrove (nelle interfacce o nel Model/Controller).
- **Implementazione Separata e Test**: Implementare e testare Model, View e Controller separatamente, poi comporli.
- **Controller Esile**: Il Controller dovrebbe essere il più "esile" possibile, limitandosi a coordinare Model e View.
- *Provenienza*: 17-swing_slides.pdf, Pagina 63

7.6.5. Varianti del Pattern MVC

Esistono diverse varianti del pattern MVC, ognuna con leggere differenze nella distribuzione delle responsabilità:

- **Model-View-ViewModel (MVVM)**: Popolare in framework come WPF, Silverlight, Android (con Data Binding). Il ViewModel espone dati e comandi per la View, e la View si lega (bind) a queste proprietà.
- **Model-View-Presenter (MVP)**: Il Presenter agisce come "middle-man" tra Model e View. La View è passiva e il Presenter gestisce tutta la logica di presentazione.

L'approccio MVC proposto nelle slide è particolarmente indicato per la sua semplicità e per la sua efficacia nel disaccoppiamento dei componenti.

- *Provenienza*: 17-swing_slides.pdf, Pagina 63

Questo conclude il Capitolo 7, che ha esplorato le GUI con Java Swing. Ho cercato di spiegare ogni concetto in modo approfondito, fornendo esempi chiari e integrando le informazioni con dettagli tratti dalle ricerche sul web per offrire una visione completa.

Sono pronto per il prossimo capitolo quando lo sarai anche tu!