15
Graphical User Interfaces

Mirko Viroli
mirko.viroli@unibo.it

C.D.L. Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2022/2023

Outline

Goal della lezione

- Illustrare la libreria Java Swing
- Fornire pattern di progettazione per le GUI

Argomenti

- Organizzazione della libreria Swing
- Panoramica dei meccanismi principali
- La gestione degli eventi nelle GUI
- Elementi di programmazione ad eventi
- Organizzazione MVC delle GUI
- . . . in seguito, JavaFX

Outline

Graphical User Interfaces (GUI)

GUI

- Interfacce grafiche per l'interazione con l'utente
- Ritenute più semplici rispetto alle CUI (Console User Interfaces)
- Sfruttano la possibilità di disegnare più o meno arbitrariamente i pixel della matrice dello schermo
- Oltre allo schermo possono sfruttare altri dispositivi: mouse, tastiera,..
- Si appoggiano su astrazioni grafiche (pulsanti, icone, finestre)

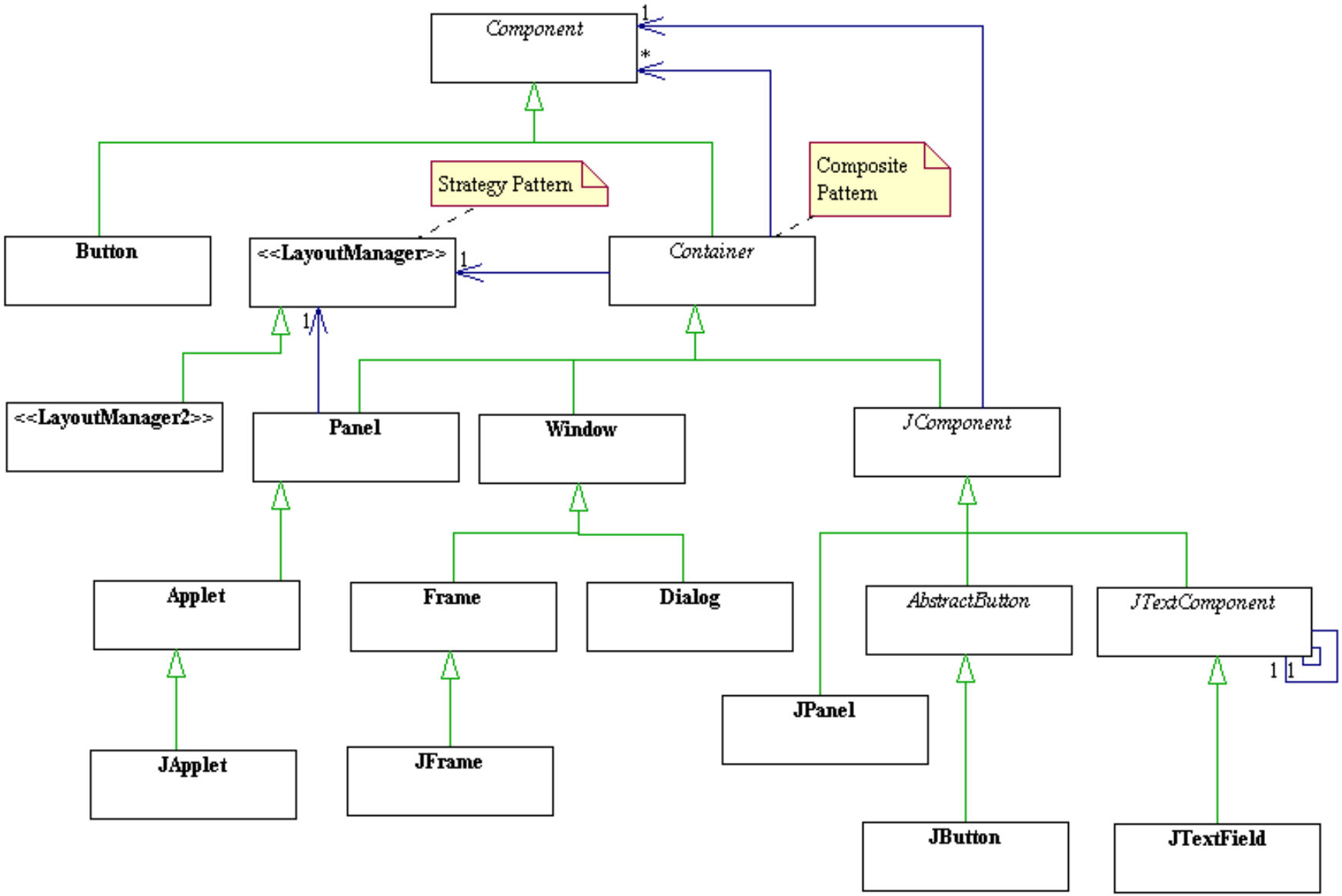
Gestione delle GUI in Java

- Abstract Window Toolkit (AWT) in Java 1 e 2 – basso livello
- Java Swing in Java 5,6,7,8
- Alternative: JavaFX (consigliata da Java 8), SWT (usato da Eclipse)

$\Rightarrow$ Vedremo Swing, che si appoggia su AWT

$\Rightarrow$ In laboratorio vedrete la più moderna JavaFX

AWT e Swing: UML



AWT, Swing e concetti principali

I due package

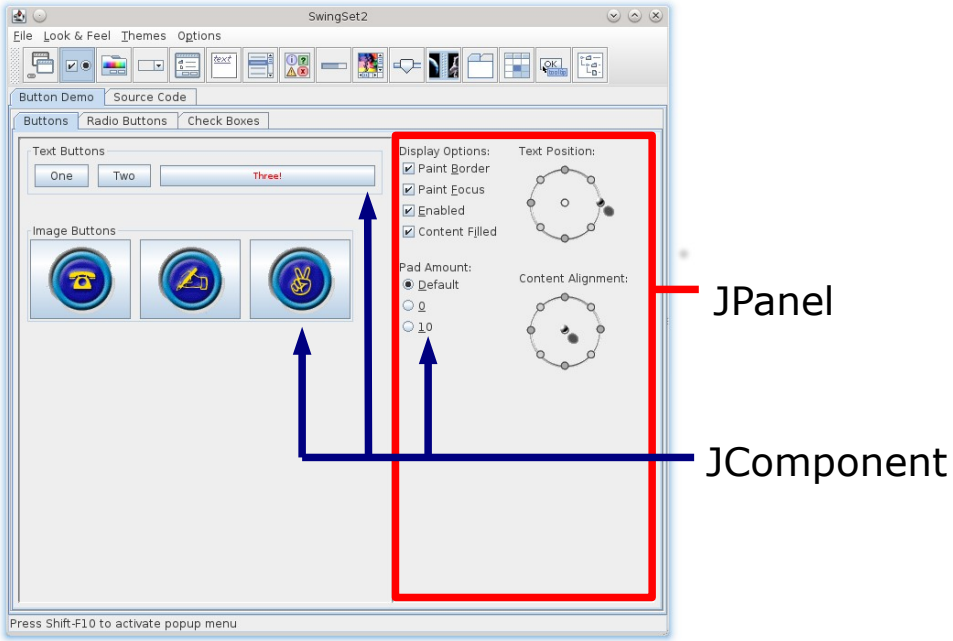
- `java.awt`: Classi base e implementazioni supportate dal S.O.
  - non molto utili da guardare in dettaglio
  - fornisce comunque l'architettura base
- `javax.swing`: implementazioni gestite "pixel per pixel"
  - le classi `J*` e quelle sottostanti

Alcune classi base di Swing

- `JFrame`: finestra con "cornice" (menù, barra, icone chiusura)
- `JPanel`: pannello di componenti inseribili in un `JFrame`
- `JComponent`: componente (pulsante, textfield, ..)
- `JDialog`: finestra di dialogo
- `JWindow`: componente piazzabile nel desktop (senza cornice)

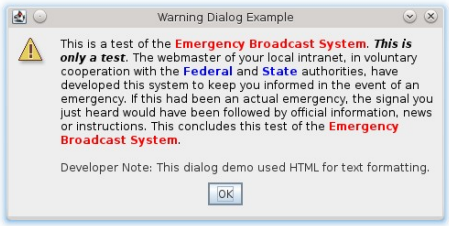
Concetti principali

JFrame



JWindow

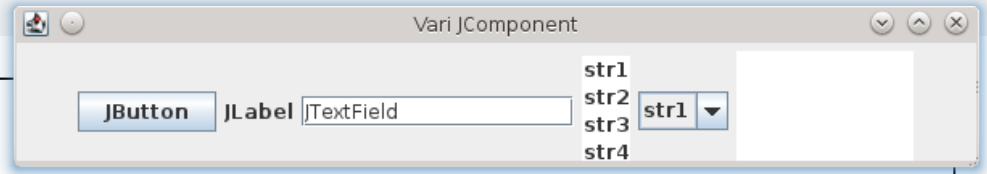
JDialog



Un primo esempio

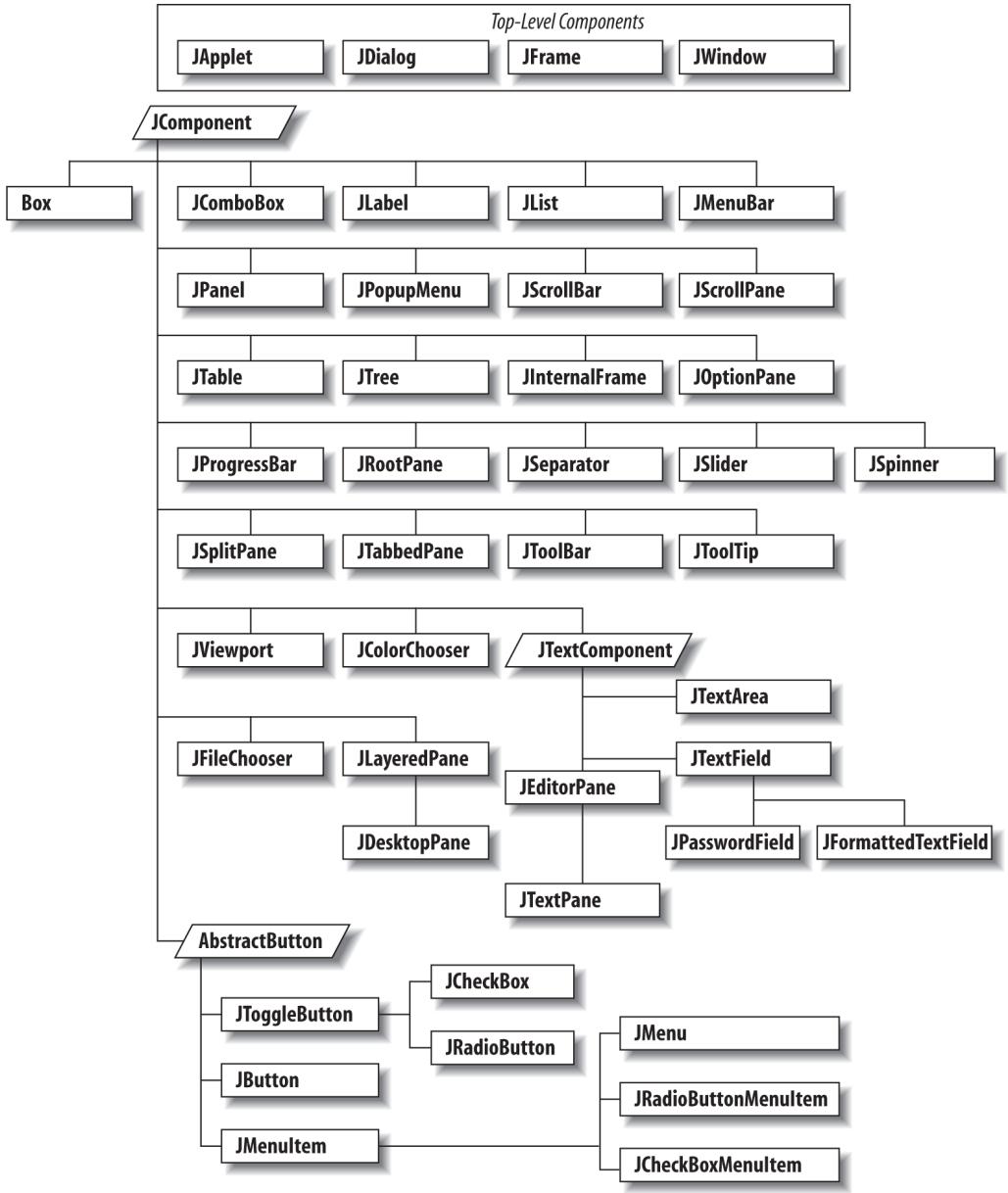

```
1  import javax.swing.*;
2
3  public class TrySwing {
4    public static void main(String[] args){
5      // Creo il frame e imposto titolo e altre proprietà
6      final JFrame frame = new JFrame();
7      frame.setTitle("Prova di JFrame");
8      frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
9      frame.setSize(320,240);
10
11     // Creo un pannello e gli imposto il bordino
12     final JPanel panel = new JPanel();
13     // Aggiungo il pannello ai 'contenuti' del frame
14     frame.getContentPane().add(panel);
15
16     // Aggiungo un pulsante al pannello
17     panel.add(new JButton("Prova di pulsante"));
18
19     // Alla fine rendo visible il JFrame
20     frame.setVisible(true);
21   }
22 }
```

Vari JComponent disponibili..




```
1  import javax.swing.*;
2
3  public class Components {
4    public static void main(String[] args){
5      // Creo il frame e imposto titolo e altre proprietà
6      final JFrame frame = new JFrame();
7      frame.setTitle("Vari JComponent");
8      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9      frame.setSize(600,100);
10
11     // Creo un pannello senza bordino e lo aggiungo al frame
12     final JPanel panel = new JPanel();
13     frame.getContentPane().add(panel);
14
15     final String[] strings = new String[]{"str1","str2","str3","str4"};
16     // Aggiungo vari componenti
17     panel.add(new JButton("JButton"));
18     panel.add(new JLabel("JLabel"));
19     panel.add(new JTextField("JTextField",15));
20     panel.add(new JList<String>(strings));
21     panel.add(new JComboBox<String>(strings));
22     panel.add(new JTextArea(5,10));
23
24     // Alla fine rendo visible il JFrame
25     frame.setVisible(true);
26   }
27 }
```

Classi di Swing



Materiale da consultare

Collezione di riferimenti utili

- JavaDoc delle librerie
- Tutorial ufficiali:
  - `http://docs.oracle.com/javase/tutorial/uiswing/`

Lezione di oggi

- Mostriamo le tecniche principali
- Occasionalmente mostreremo il funzionamento di vari componenti
- Costruire GUI efficaci (e avanzate) richiede però conoscenze ulteriori ottenibili all'occorrenza dai riferimenti di cui sopra

Outline

Il problema del Layout di un pannello

Problema

- Intervenire sulla politica di dislocazione dei componenti
- Scegliere politiche indipendenti dalle dimensioni della finestra
- Organizzare tali selezioni con una buona organizzazione OO

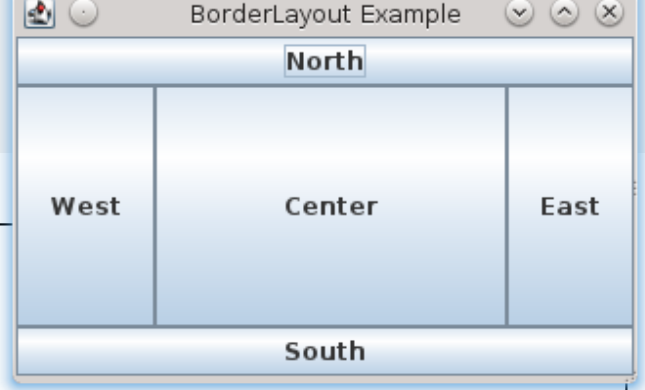
La classe `LayoutManager` e il pattern "Strategy"

- Al pannello si passa un oggetto di `LayoutManager`
- È lui che incapsula la strategia di inserimento dei componenti
- Vari casi: `FlowLayout` (default), `BorderLayout`, `GridBagLayout`,.. (tipicamente da comporre tra loro)
- Vedere: `http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html`
- Il metodo `add()` di `JPanel` accetta un ulteriore argomento (`Object` o `int`) usato dal Layout Manager

Senza Layout – deprecabile


```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class UseNoLayout {
5    public static void main(String[] args){
6      final JFrame frame = new JFrame();
7      frame.setTitle("AbsoluteLayout Example");
8      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9      frame.setResizable(false); // o true..
10     frame.setSize(320,200);
11
12     final JPanel panel = new JPanel();
13     panel.setLayout(null); // Nessun layout
14     frame.getContentPane().add(panel);
15
16     final JButton b1 = new JButton("Button 1");
17     final JButton b2 = new JButton("Button 2");
18     panel.add(b1);
19     panel.add(b2);
20
21     // Imposto dimensione e posizione
22     Dimension size = b1.getPreferredSize();
23     b1.setBounds(25, 5, size.width, size.height);
24     size = b2.getPreferredSize();
25     b2.setBounds(55, 40, size.width*3, size.height*3);
26
27     frame.setVisible(true);
28   }
```

BorderLayout

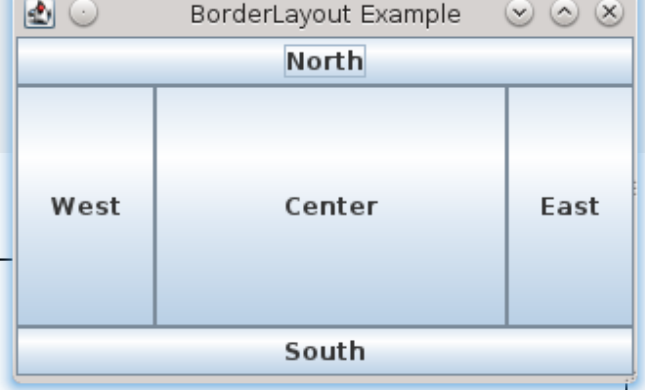



```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class UseBorderLayout {
5    public static void main(String[] args){
6      final JFrame frame = new JFrame();
7      frame.setTitle("BorderLayout Example");
8      frame.setSize(320,200);
9
10     final JPanel panel = new JPanel();
11     panel.setLayout(new BorderLayout()); // Imposto il layout
12     frame.getContentPane().add(panel);
13
14     // Nota l'argomento aggiuntivo di tipo int
15     // Nota che i pulsanti non usano la loro dim. preferita!
16     panel.add(new JButton("North"),BorderLayout.NORTH);
17     panel.add(new JButton("South"),BorderLayout.SOUTH);
18     panel.add(new JButton("Center"),BorderLayout.CENTER);
19     panel.add(new JButton("East"),BorderLayout.EAST);
20     panel.add(new JButton("West"),BorderLayout.WEST);
21
22     frame.setVisible(true);
23   }
24 }
```

Lavorare specializzando JFrame: MyFrame

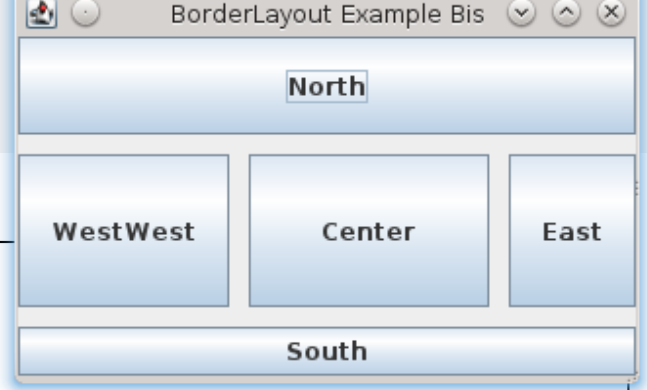
```java
import java.awt.*;
import javax.swing.*;

/* Specializzazione di JFrame:
 * - JFrame è Serializable!
 * - Il costruttore accetta titolo e layout-manager
 * - Si aggiunge il JPanel
 * - Un metodo getMainPanel() ci dà il pannello
 */
public class MyFrame extends JFrame{

  private final JPanel jp;

  public MyFrame(String title, LayoutManager lm){
    super(title);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setSize(320,200);
    // Il layout-manager può essere passato al costruttore di JPanel
    this.jp = new JPanel(lm);
    this.getContentPane().add(this.jp);
  }

  public JPanel getMainPanel(){
    return this.jp;
  }

}
```

Nuova versione UseBorderLayout2



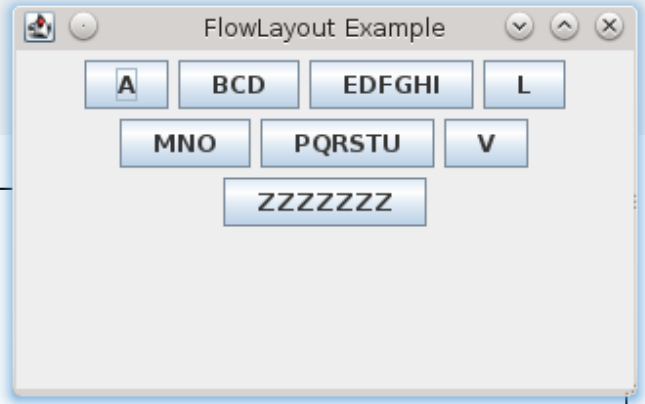

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class UseBorderLayout2{
5    public static void main(String[] args){
6      final MyFrame frame =
7          new MyFrame("BorderLayout Example",new BorderLayout());
8
9      frame.getMainPanel().add(new JButton("North"),BorderLayout.NORTH);
10     frame.getMainPanel().add(new JButton("South"),BorderLayout.SOUTH);
11     frame.getMainPanel().add(new JButton("Center"),BorderLayout.CENTER);
12     frame.getMainPanel().add(new JButton("East"),BorderLayout.EAST);
13     frame.getMainPanel().add(new JButton("West"),BorderLayout.WEST);
14
15     frame.setVisible(true);
16   }
17   /* Note sul BorderLayout:
18    * - In NORTH e SOUTH usa l'altezza preferita del componente
19    * - In EAST e WEST usa la larghezza preferita del componente
20    * - Altrove no..
21    */
22 }
```

Qualche modifica: UseBorderLayout3



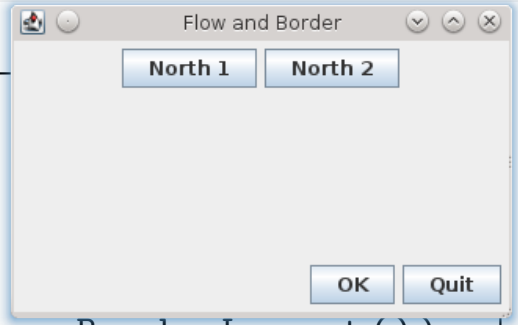

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class UseBorderLayout3{
5    public static void main(String[] args){
6      final BorderLayout b = new BorderLayout();
7      b.setHgap(10); // Parametri addizionali del lay-man
8      b.setVgap(10);
9      final MyFrame frame = new MyFrame("BorderLayout Example Bis",b);
10
11     final JButton button = new JButton("North");
12     final Dimension d = button.getPreferredSize(); // imposto le dim..
13     button.setPreferredSize(new Dimension(d.width,d.height*2));
14     frame.getMainPanel().add(button,BorderLayout.NORTH);
15     frame.getMainPanel().add(new JButton("South"),BorderLayout.SOUTH);
16     frame.getMainPanel().add(new JButton("Center"),BorderLayout.CENTER);
17     frame.getMainPanel().add(new JButton("East"),BorderLayout.EAST);
18     // nota l'effetto di una stringa più lunga qui
19     frame.getMainPanel().add(new JButton("WestWest"),BorderLayout.WEST);
20
21     frame.setVisible(true);
22   }
23 }
```

FlowLayout



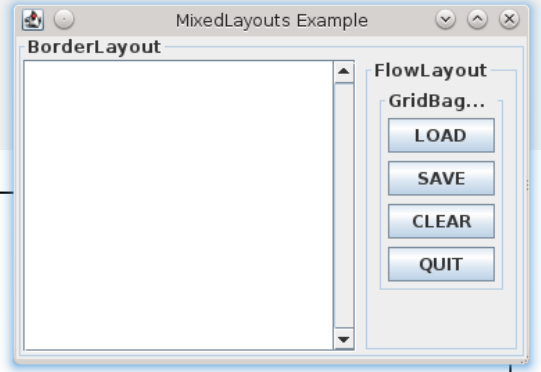

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class UseFlowLayout{
5    public static void main(String[] args){
6      final FlowLayout lay = new FlowLayout(FlowLayout.CENTER);
7      final MyFrame frame = new MyFrame("FlowLayout Example",lay);
8      frame.getMainPanel().add(new JButton("A"));
9      frame.getMainPanel().add(new JButton("BCD"));
10     frame.getMainPanel().add(new JButton("EDFGHI"));
11     frame.getMainPanel().add(new JButton("L"));
12     frame.getMainPanel().add(new JButton("MNO"));
13     frame.getMainPanel().add(new JButton("PQRSTU"));
14     frame.getMainPanel().add(new JButton("V"));
15     frame.getMainPanel().add(new JButton("ZZZZZZZ"));
16     //frame.pack(); ridimensiona la finestra: da provare!
17     frame.setVisible(true);
18   }
19   /* Note sul FlowLayout:
20    * - Di default mette i componenti da sx a dx, centrati
21    * - usa le loro dim preferite
22    * - va a capo quando necessario, partendo dall'alto
23    * .. tutti aspetti modificabili agendo sull'oggetto Layout
24    */
25 }
```

Un uso combinato di FlowLayout e BorderLayout




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

public class UseFlowBorder{
  public static void main(String[] args){
    final MyFrame frame = new MyFrame("Flow and Border",new BorderLayout());

    // Creo un sotto-pannello per la parte NORTH
    final JPanel pNorth = new JPanel(new FlowLayout());
    pNorth.add(new JButton("North 1"));
    pNorth.add(new JButton("North 2"));

    // Creo un sotto-pannello per la parte SOUTH
    final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    pSouth.add(new JButton("OK"));
    pSouth.add(new JButton("Quit"));

    // Grazie al polimorfismo, aggiungo pannelli
    frame.getMainPanel().add(pNorth,BorderLayout.NORTH);
    frame.getMainPanel().add(pSouth,BorderLayout.SOUTH);

    frame.setVisible(true);
  }
}
```

Altro uso combinato + GridBagLayout




```java
public class UseMixedLayouts{

  public static void main(String[] args){
    final JTextArea textArea = new JTextArea(); // Area di testo
    textArea.setLineWrap(true);
    final JScrollPane scroll = new JScrollPane(textArea); // Pannello con barra
    scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

    final JPanel pEastInternal = new JPanel(new GridBagLayout()); // Griglia flessibile
    pEastInternal.setBorder(new TitledBorder("GridBagLayout"));
    final GridBagConstraints cnst = new GridBagConstraints();
    cnst.gridy = 0;                  // 1-a riga
    cnst.insets = new Insets(3,3,3,3);        // spazio attorno al comp.
    cnst.fill = GridBagConstraints.HORIZONTAL;    // estensione in orizzont.
    pEastInternal.add(new JButton("LOAD"),cnst);
    cnst.gridy++;                  // prossima riga
    pEastInternal.add(new JButton("SAVE"),cnst);
    cnst.gridy++;
    pEastInternal.add(new JButton("CLEAR"),cnst);
    cnst.gridy++;
    pEastInternal.add(new JButton("QUIT"),cnst);

    final JPanel pEast = new JPanel(new FlowLayout());
    pEast.setBorder(new TitledBorder("FlowLayout"));
    pEast.add(pEastInternal);

    final MyFrame frame = new MyFrame("MixedLayouts Example",new BorderLayout(5,5));
    frame.getMainPanel().add(scroll,BorderLayout.CENTER);
    frame.getMainPanel().add(pEast,BorderLayout.EAST);
    frame.getMainPanel().setBorder(new TitledBorder("BorderLayout"));
    frame.setVisible(true);
  }
```

Lavorare incapsulando il frame dietro una interfaccia

L'indipendenza dalla tecnologia delle GUI

- in applicazioni reali sarà importante costruire GUI in modo che le scelte di basso livello dettate da Swing siano ben nascoste
- così da tenere tutto il resto del sistema indipendente da Swing, e quindi ben organizzato e ad alto livello

Tipica tecnica

- si disegni una `interface` pulita per la GUI
- i dettagli implementativi al solito siano delegati ad una classe che implementa

Lavorare incapsulando il frame dietro una interfaccia

```java
public interface UserInterface {

   void show();

   void setDimensions(int x, int y);
}
```

```java
public class UserInterfaceImpl implements UserInterface {

  private final MyFrame frame;

  public UserInterfaceImpl() {
    this.frame = new MyFrame("Flow and Border", new BorderLayout());
    final JPanel pNorth = new JPanel(new FlowLayout());
    pNorth.add(new JButton("North 1"));
    final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    pSouth.add(new JButton("OK"));
    this.frame.getMainPanel().add(pNorth, BorderLayout.NORTH);
    this.frame.getMainPanel().add(pSouth, BorderLayout.SOUTH);
  }

  public void show(){
    this.frame.setVisible(true);
  }

  public void setDimensions(int x, int y) {
    this.frame.setSize(x,y);
  }
}
```

Lavorare incapsulando il frame dentro una classe

```
1 public class UseEncapsulatedFrame {
2
3   public static void main(String[] args) {
4     final UserInterface ui = new UserInterfaceImpl();
5     ui.setDimensions(300, 300);
6     ui.show();
7   }
8
9 }
```

Outline

La gestione degli eventi

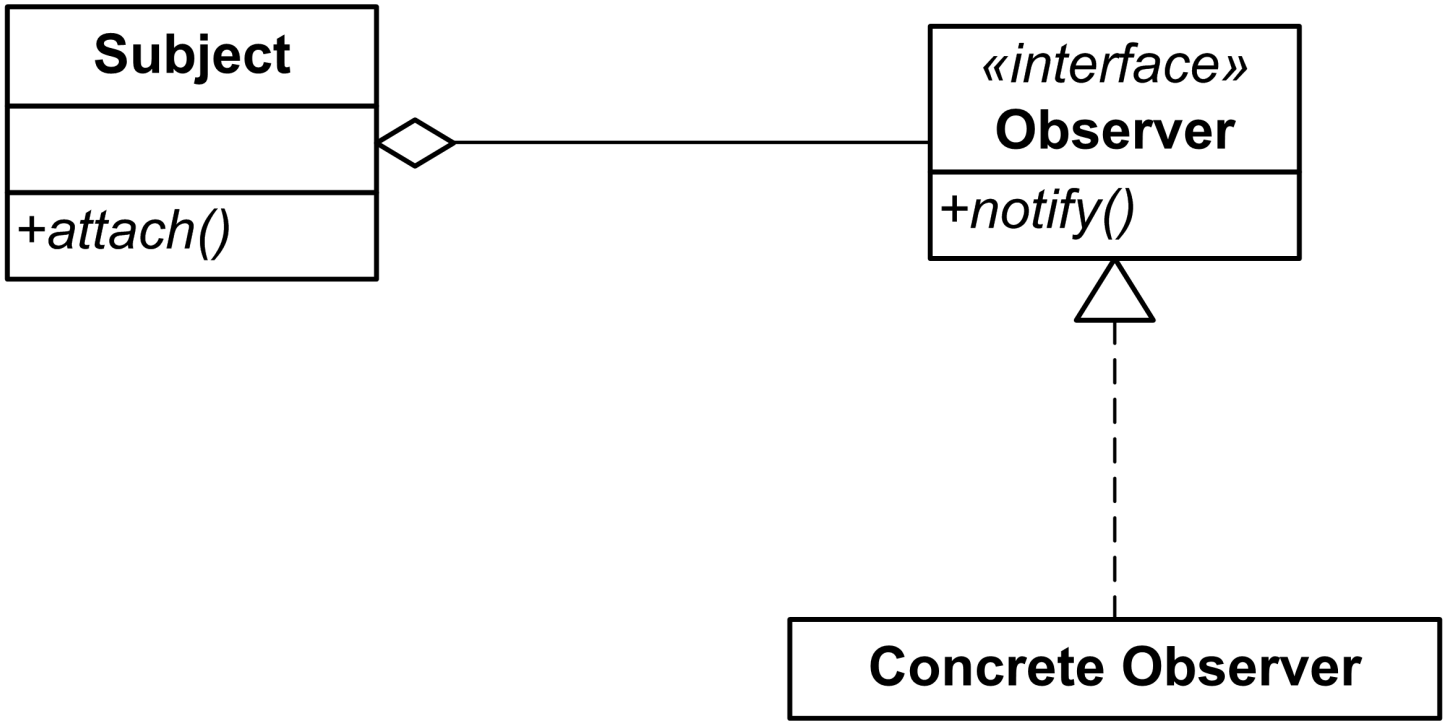
Come rendere le interfacce "vive"?

- come programmare la possibilità di intercettare le varie azioni che un utente potrebbe compiere sull'interfaccia, ossia la parte "input"?
- al solito, sarebbe necessario uno strumento altamente configurabile e ben organizzato

Il pattern Observer

- È possibile "registrare" nei componenti degli oggetti "ascoltatori" (listeners)
- Quando certi eventi accadono, il componente richiama un metodo dei listener registrati
- Tale metodo contiene il codice da eseguire in risposta all'evento
- Si assume (per ora) che tale codice arrivi "velocemente" a compimento

Il pattern Observer

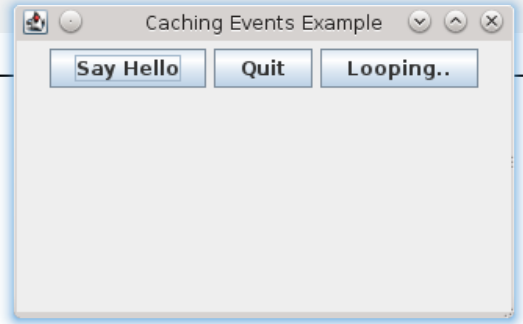


- `Subject` è la sorgente degli eventi
- `Observer` si registra con la `attach(o:Observer)`
- Quando accade l'evento, `Subject` chiama `notify(e:Event)`

Il caso dei click sui pulsanti: 3 classi

```
1  // E' il subject degli eventi
2  class JButton .. {
3      void setActionCommand(String s){..}
4      void addActionListener(ActionListener listener){..}
5      void removeActionListener(ActionListener listener){..}
6  }
7
8  // Interfaccia da implementare per ascoltare gli eventi
9  interface ActionListener .. {
10     void actionPerformed(ActionEvent e);
11 }
12
13 // Classe per rappresentare un evento
14 class ActionEvent .. {
15     String  getActionCommand(){..}
16     long  getWhen(){..}
17     ..
18 }
```

Catturare gli eventi dei pulsanti con ActionCommand




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class UseButtonEvents{
6    public static void main(String[] args){
7      final MyFrame frame = new MyFrame("Events Example",new FlowLayout());
8
9      final ActionListener listener = new MyActionListener();
10
11     final JButton b1 = new JButton("Say Hello");
12     b1.setActionCommand("hello");  // nome comando
13     b1.addActionListener(listener);// registro il listener
14
15     final JButton b2 = new JButton("Quit");
16     b2.setActionCommand("quit");
17     b2.addActionListener(listener);
18
19     final JButton b3 = new JButton("Looping..");
20     b3.setActionCommand("loop");
21     b3.addActionListener(listener);
22
23     frame.getMainPanel().add(b1);
24     frame.getMainPanel().add(b2);
25     frame.getMainPanel().add(b3);
26     frame.setVisible(true);
27   }
28 }
```

Corrispondente listener, come classe esterna

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

// Nota: si potrebbero usare delle inner class
// Nota: ActionCommand "abusa" delle stringhe..
public class MyActionListener implements ActionListener {

  public void actionPerformed(ActionEvent e){
    if (e.getActionCommand().equals("hello")){
      System.out.println("Hello!!");
    } else if (e.getActionCommand().equals("quit")){
      System.out.println("Quitting..");
      System.exit(0);
    } else if (e.getActionCommand().equals("loop")){
      System.out.println("Going stuck..");
      for(;true;){} // Nota l'effetto del loop sulla GUI
    }
  }
}
```

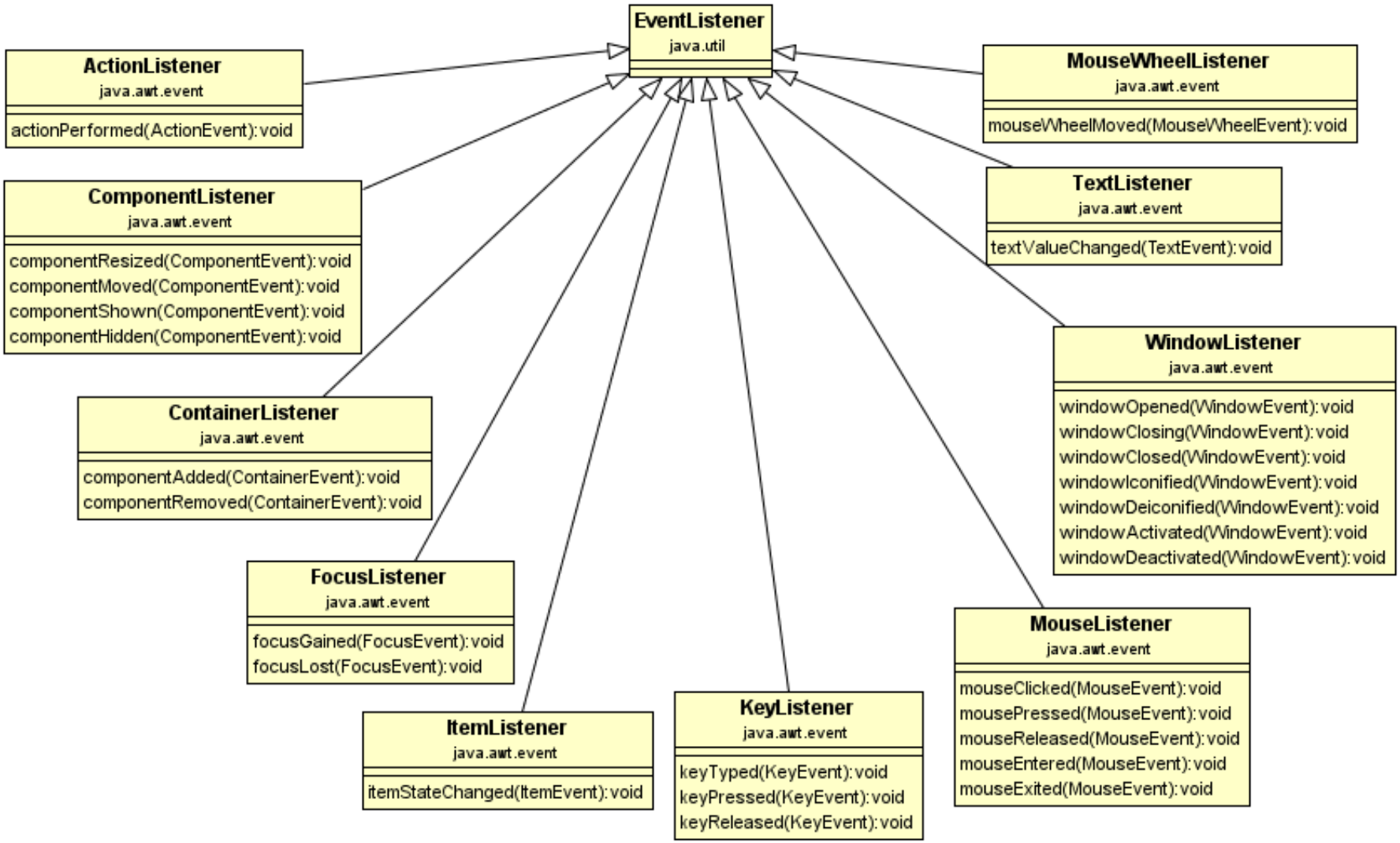
Versione incapsulata (inner listener + source eventi)

```
1  public class EventsFrame {
2
3    private final JButton b1 = new JButton("Say Hello");
4    private final JButton b2 = new JButton("Quit");
5    private final JButton b3 = new JButton("Looping..");
6
7    public EventsFrame() {
8      final MyFrame frame = new MyFrame("Events Example", new FlowLayout());
9      frame.getMainPanel().add(b1); // aggiungo i pulsanti
10     frame.getMainPanel().add(b2);
11     frame.getMainPanel().add(b3);
12     final ActionListener listener = new MyActionListener(); //listener unico
13     b1.addActionListener(listener);// registro il listener
14     b2.addActionListener(listener);// senza actionCommand!!
15     b3.addActionListener(listener);
16     frame.setVisible(true);
17   }
18
19   private class MyActionListener implements ActionListener {
20
21     public void actionPerformed(ActionEvent e) { //switch su getSource
22       if (e.getSource()==EventsFrame.this.b1) {
23         System.out.println("Hello!!");
24       } else if (e.getSource()==EventsFrame.this.b2) {
25         System.out.println("Quitting..");
26         System.exit(0);
27       } else if (e.getSource()==EventsFrame.this.b3) {
28         System.out.println("Going stuck..");
29         for (; true;) {
30         } // Nota l'effetto del loop sulla GUI
31       }
32     }
```

Listeners come classi anonime

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class UseButtonEvents2{
6    public static void main(String[] args){
7      final JButton b1 = new JButton("Say Hello");
8      b1.addActionListener(new ActionListener(){
9        public void actionPerformed(ActionEvent e) {
10         System.out.println("Hello!!");
11       }
12     }); // Uso una inner class anonima..
13
14     final JButton b2 = new JButton("Quit");
15     b2.addActionListener(new ActionListener(){
16       public void actionPerformed(ActionEvent e) {
17         System.out.println("Quitting..");
18         System.exit(0);
19       }
20     }); // Uso una inner class anonima..
21
22     final MyFrame frame = new MyFrame("Events Example",new FlowLayout());
23     frame.getMainPanel().add(b1);
24     frame.getMainPanel().add(b2);
25     frame.setVisible(true);
26   }
27 }
```

Panoramica eventi-listeners



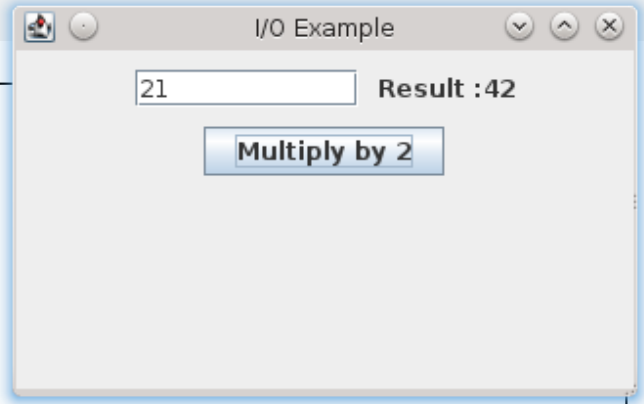
Sulla gestione degli eventi

Il flusso di controllo con le GUI di Swing

- Quando si crea un `JFrame`, la JVM crea l'`EventDispatchThread` (EDT)
- Quindi l'applicazione non termina quando il `main` completa
- Quando un evento si verifica la JVM fa eseguire il corrispondente codice all'EDT
- Ecco perché la GUI non risponde a nuovi eventi finché uno precedente non è stato gestito
- Per gestire con migliore flessibilità le GUI servono meccanismi di programmazione concorrente, che vedremo in futuro

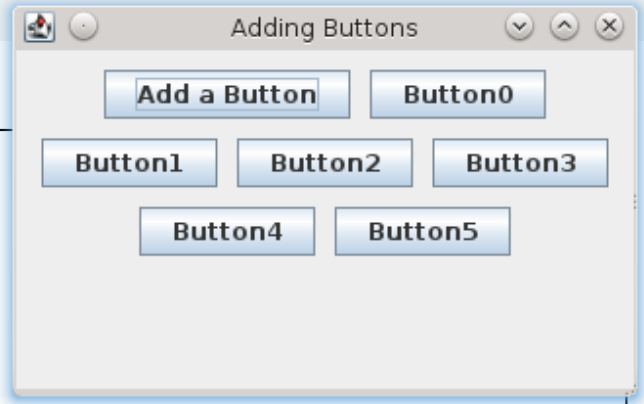
Outline

GUI con I/O: listeners che modificano l'interfaccia



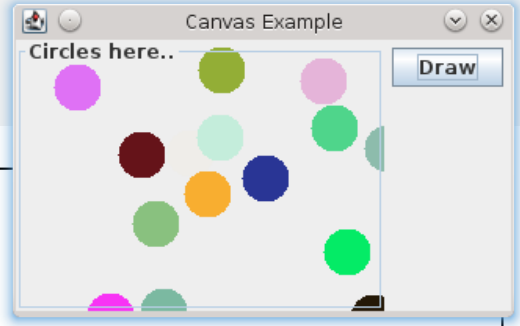

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class UseIOGUI{
6    public static void main(String[] args){
7      final JTextField tf = new JTextField(10);
8      final JLabel lb = new JLabel("Result: 0");
9      final JButton bt = new JButton("Multiply by 2");
10
11     bt.addActionListener(new ActionListener(){
12       public void actionPerformed(ActionEvent e) {
13         String s = tf.getText(); // "21"
14         int n = Integer.parseInt(s); // 21
15         lb.setText("Result :"+n*2);
16       }
17     });
18
19     final FlowLayout lay = new FlowLayout(FlowLayout.CENTER,10,10);
20     final MyFrame frame = new MyFrame("I/O Example",lay);
21     frame.getMainPanel().add(tf);
22     frame.getMainPanel().add(lb);
23     frame.getMainPanel().add(bt);
24     frame.setVisible(true);
25   }
26 }
```

GUI con Layout dinamico




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class UseDynamicLayout{
6    public static void main(String[] args){
7
8      final FlowLayout lay = new FlowLayout(FlowLayout.CENTER,10,10);
9      final MyFrame frame = new MyFrame("Adding Buttons",lay);
10     final JPanel panel = frame.getMainPanel();
11
12     final JButton bt = new JButton("Add a Button");
13     bt.addActionListener(new ActionListener(){
14       int count = 0;
15       public void actionPerformed(ActionEvent e) {
16         panel.add(new JButton("Button"+count++));
17         panel.validate(); // forza il ricalcolo del layout!
18       }
19     });
20     panel.add(bt);
21     frame.setVisible(true);
22   }
23 }
```

Uso di un pannello come Canvas

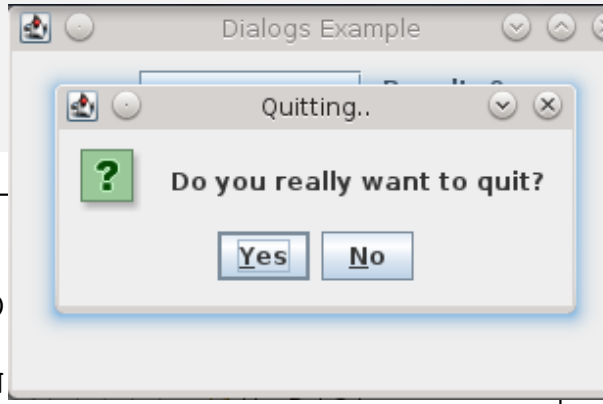



```
1  public class UseCanvas{
2
3    public static void main(String[] args){
4      final DrawPanel pCenter = new DrawPanel();
5      pCenter.setBorder(new TitledBorder("Circles here.."));
6      // Intercetto i click del mouse!
7      // L'Adapter già implementa banalmente i metodi, quindi basta fare un override
8      pCenter.addMouseListener(new MouseAdapter(){
9        public void mouseClicked(MouseEvent e) {
10         pCenter.addPoint(e.getX(), e.getY());
11         pCenter.repaint();
12       }
13     });
14     // Intercetto il click sul pulsante
15     final JPanel pEast = new JPanel(new FlowLayout());
16     final JButton bt = new JButton("Draw");
17     bt.addActionListener(new ActionListener(){
18       public void actionPerformed(ActionEvent e) {
19         pCenter.addRandomPoint();
20         pCenter.repaint();
21       }
22     });
23     pEast.add(bt);
24
25     final MyFrame frame = new MyFrame("Canvas Example",new BorderLayout());
26     frame.getMainPanel().add(pCenter,BorderLayout.CENTER);
27     frame.getMainPanel().add(pEast,BorderLayout.EAST);
28     frame.setResizable(false); // non estendibile!!
29     frame.setVisible(true);
30   }
31 }
```

La classe DrawPanel

```java
// Specializzazione ad-hoc per un JPanel
public class DrawPanel extends JPanel {

  private static final long serialVersionUID = 71140663470617018 32L;
  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
  protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    for (Map.Entry<Point,Color> e : this.circles.entrySet()) {
      g.setColor(e.getValue());
      g.fillOval(e.getKey().x, e.getKey().y, RADIUS, RADIUS);
    }
  }

  // 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
  public void addPoint(int x, int y){
    final Color c = new Color(RND.nextInt(256),RND.nextInt(256),RND.nextInt(256));
    this.circles.put(new Point(x-RADIUS/2,y-RADIUS/2),c);
  }
}
```

Uso delle finestre di dialogo



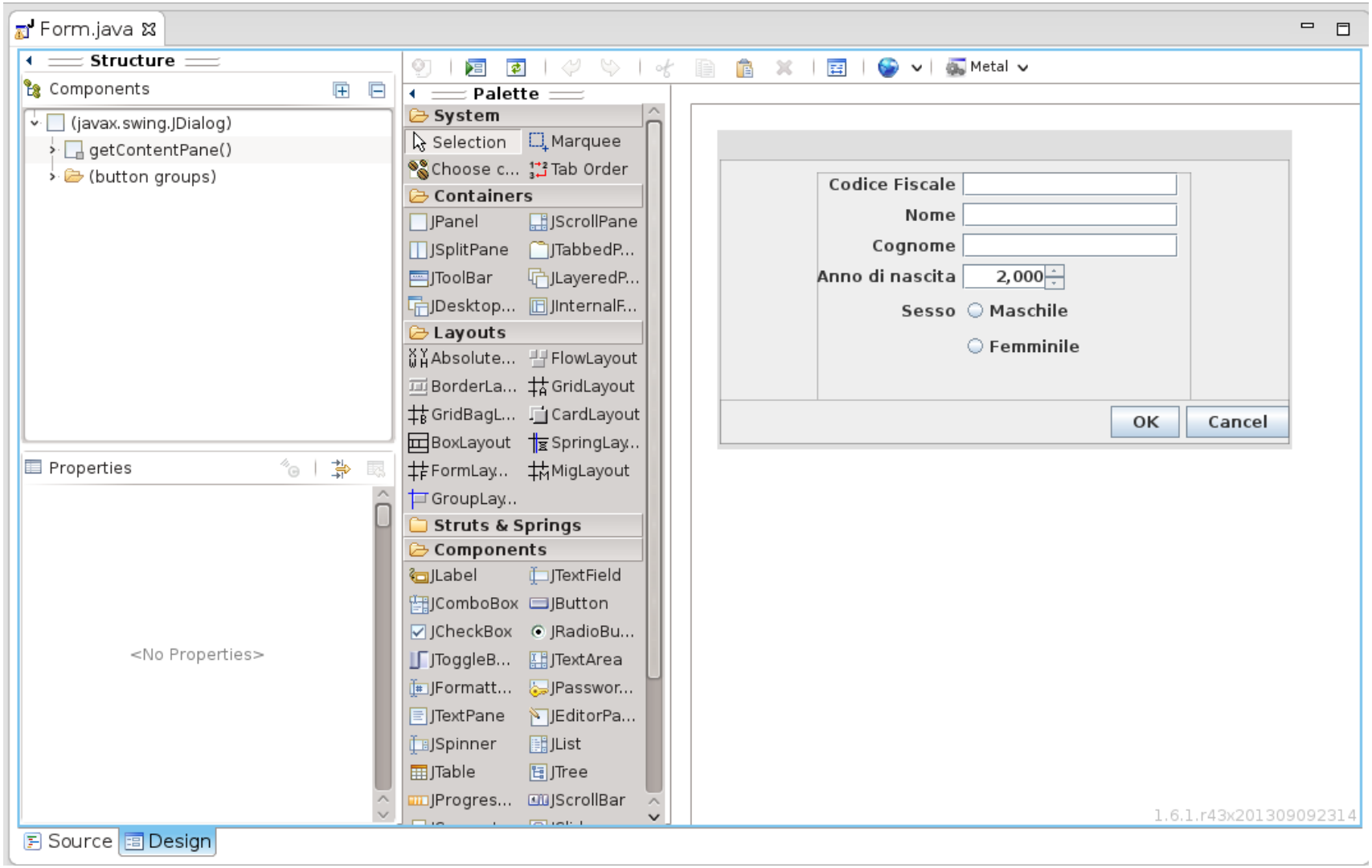

```
1  public class UseDialogs{
2    public static void main(String[] args){
3      final FlowLayout fl = new FlowLayout(FlowLayout.CENTER,10,10
4      final MyFrame frame = new MyFrame("Dialogs Example",fl);
5      frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON
6      frame.addWindowListener(new WindowAdapter(){
7        public void windowClosing(WindowEvent e) {
8          int n = JOptionPane.showConfirmDialog(frame,
9                "Do you really want to quit?",
10               "Quitting..", JOptionPane.YES_NO_OPTION);
11         if (n == JOptionPane.YES_OPTION){
12           System.exit(0);
13         }
14       }
15     });
16     final JTextField tf = new JTextField(10);
17     final JLabel lb = new JLabel("Result: 0");
18     final JButton bt = new JButton("Multiply by 2");
19     bt.addActionListener(new ActionListener(){
20       public void actionPerformed(ActionEvent e) {
21         try{
22           lb.setText("Result :"+Integer.parseInt(tf.getText())*2);
23         } catch (Exception ex){
24           JOptionPane.showMessageDialog(frame, "An integer please..");
25         }
26       }
27     });
28     frame.getMainPanel().add(tf);
29     frame.getMainPanel().add(lb);
30     frame.getMainPanel().add(bt);
31     frame.setVisible(true);
32   }
```

Le GUI builder

Cosa sono?

- Sono sistemi software usabili per creare il codice che genera le interfacce
- Permettono una descrizione WYSIWYG (What you see is what you get)
- Spesso non sono particolarmente semplici da usare
- Con un po' di esperienza e una buona conoscenza delle librerie sottostanti, possono essere usati con successo
- Se li si usasse, si deve però anche comprendere (e criticare) il codice che producono

WindowBuilder: un plugin per Eclipse



Outline

Quale architetturale complessiva?

Cos'è una architettura software?

- è la struttura del sistema software di interesse
- definisce:
  1. elementi software (o componenti, che poi diventeranno (gruppi di) classi/interfacce)
  2. relazioni fra elementi (tipicamente, dipendenze/usi)
  3. proprietà degli elementi software (ruoli/task) e delle relazioni (tipologia, dati/flussi coinvolti)

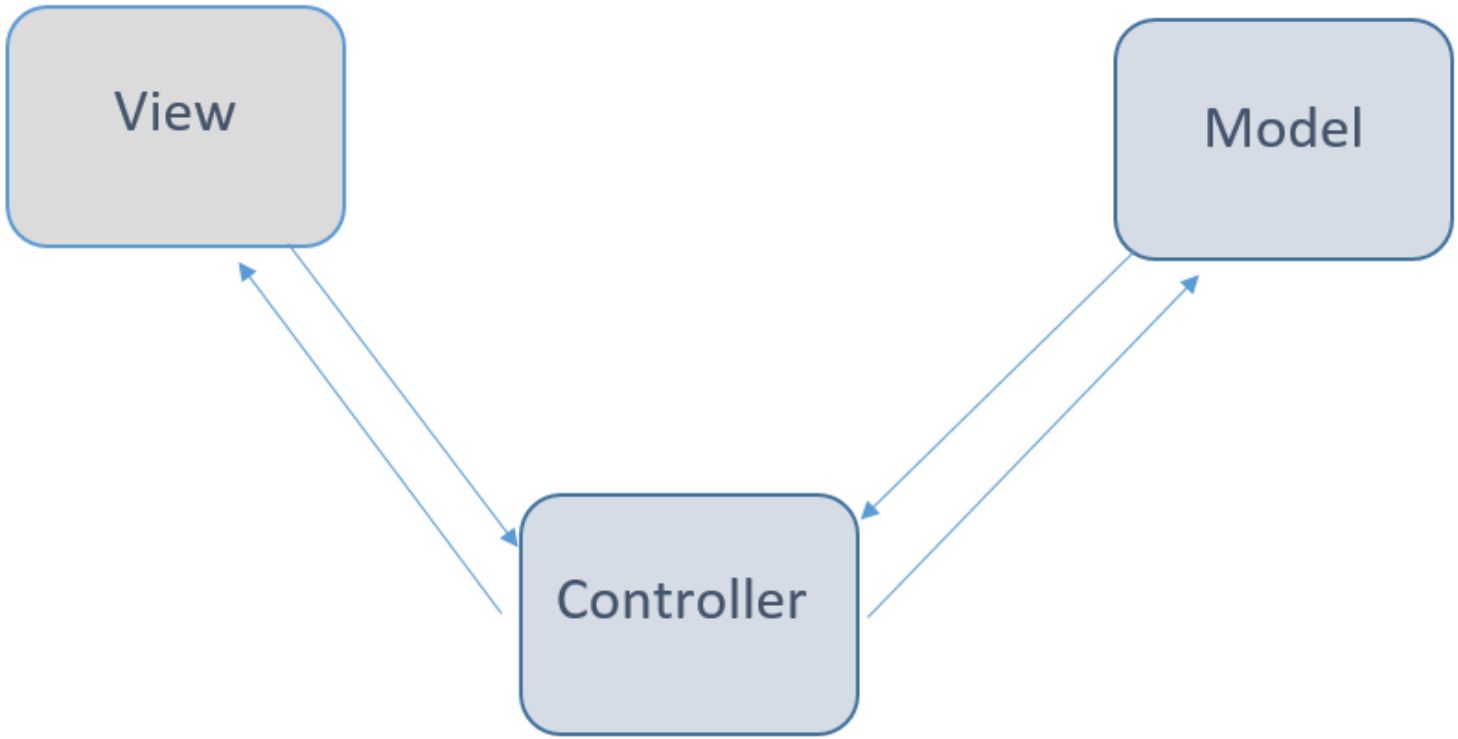
Pattern architetturale

- è una architettura software ritenuta ricorrente e con benefici, e quindi riusabile
- è descritta generalmente, e quindi è da calare nel contesto specificio del software da realizzare
- esempi: MVC, MVVM, ECB, Layers, P2P

Il pattern architetturale MVC

MVC – divide l'applicazione in 3 parti

- `Model`: modello OO del dominio applicativo del sistema
- `View`: gestisce le interazioni con l'utente (input e output)
- `Controller`: gestisce il coordinamento fra Model e View



Applicazione del MVC

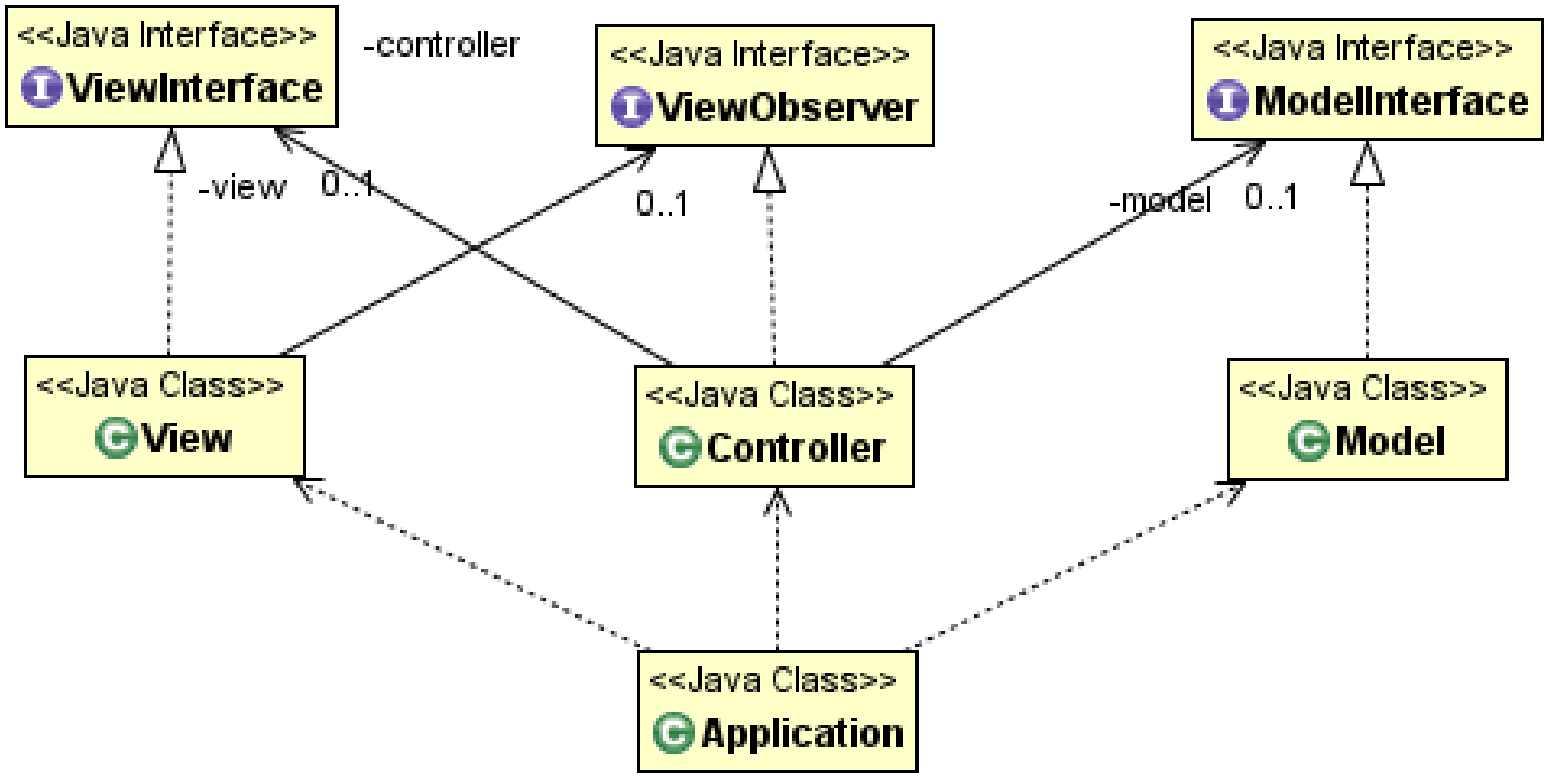
Sulla costruzione di applicazioni con GUI

- Specialmente se non esperti, possono essere alquanto laboriose
- Usare un approccio strutturato sembra richiedere più tempo nel complesso, ma in realtà porta a soluzione più facilmente modificabili e controllabili

Alcune linee guida

- Usare il pattern MVC per la struttura generale
- Identificate le varie "interazioni", e quindi costruire le interfacce dei 3 componenti bene fin dall'inizio
- Cercare massima indipendenza fra i vari componenti (interfacce con meno metodi possibile)
- La tecnologia di V non abbia dipendenze in C e M (e viceversa)
- Costruire e testare modello e GUI separatamente (M e V), poi collegare il tutto col controllore (C) che risulterà particolarmente esile

MVC con le GUI: un esempio di design



model, view e controller potrebbero poi delegare a varie classi aggiuntive. . .

Componenti e loro interazioni

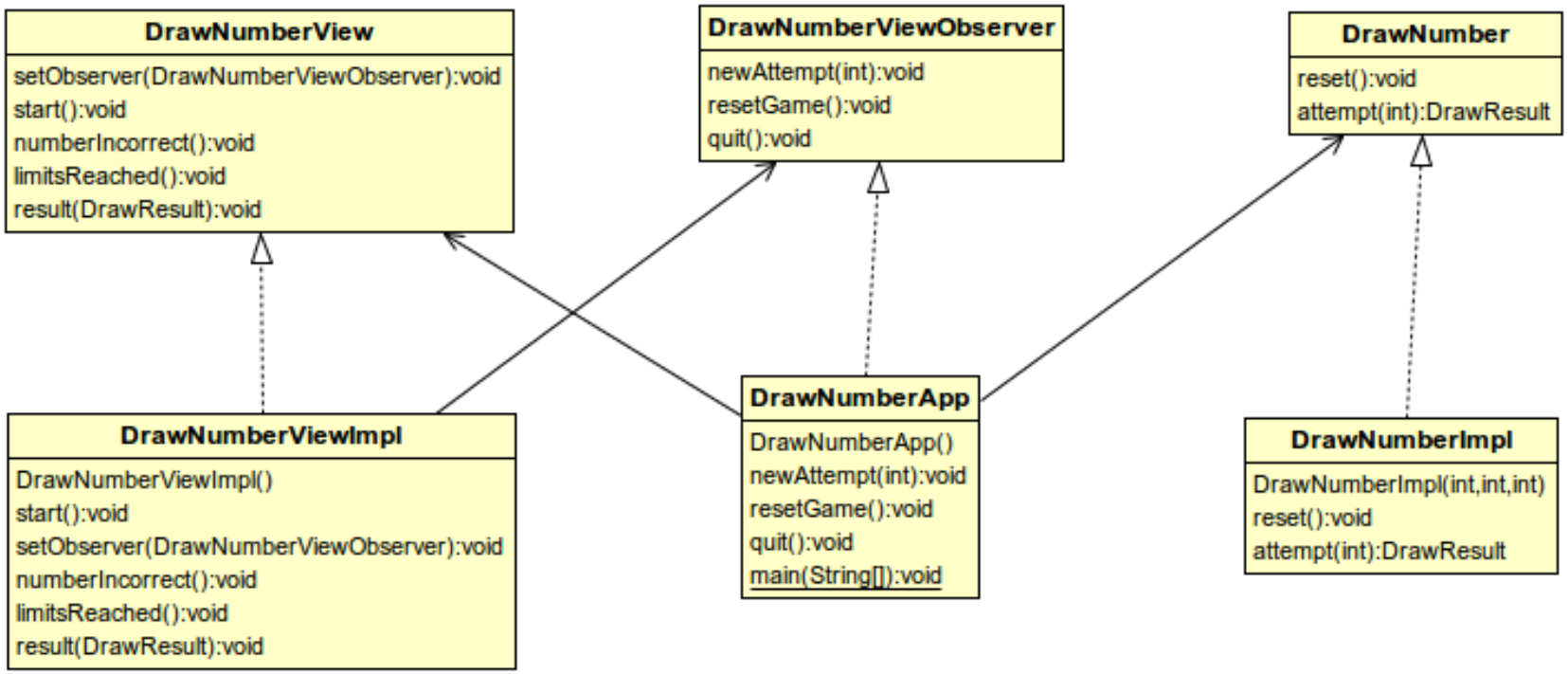
MVC

- `Model`: incapsula dati e logica relativi al dominio della applicazione
- `View`: incapsula la GUI, le sue sottoparti, e la logica di notifica
- `Controller`: intercetta gli eventi della View, comanda le modifiche al modello, cambia di conseguenza la View

Interfacce – nomi da modificare in una applicazione concreta

- `ModelInterface`: letture/modifiche da parte del Controller
- `ViewObserver`: comandi inviati dalla view al controller (`void`)
- `ViewInterface`: comandi per settare la view, notifiche a fronte dei comandi (errori..)

Un esempio di applicazione: `DrawNumber`



Interfaccia del model: DrawNumber

```java
public interface DrawNumber {

  void reset();

  DrawResult attempt(int n) throws AttemptsLimitReachedException;
}
```

```java
public enum DrawResult {
  YOURS_IS_LOWER, YOURS_IS_HIGHER, YOU_WON;
}
```

```java
public class AttemptsLimitReachedException extends Exception {

    public AttemptsLimitReachedException() {}
}
```

Implementazione del model: DrawNumberImpl (1/2)

```java
public class DrawNumberImpl implements DrawNumber {

  private int choice;
  private final int min;
  private final int max;
  private final int attempts;
  private int remainingAttempts;
  private final Random random = new Random();

  public DrawNumberImpl(final int min, final int max, final int attempts) {
    this.min = min;
    this.max = max;
    this.attempts = attempts;
    this.reset();
  }

  public void reset() {
    this.remainingAttempts = this.attempts;
    this.choice = this.min + random.nextInt(this.max-this.min+1);
  }
```

Implementazione del model: DrawNumberImpl (2/2)

```
1   public DrawResult attempt(int n) throws AttemptsLimitReachedException {
2     if (this.remainingAttempts == 0) {
3       throw new AttemptsLimitReachedException();
4     }
5     if (n < this.min || n > this.max){
6       throw new IllegalArgumentException();
7     }
8     if (n > this.choice){
9       return DrawResult.YOURS_IS_HIGHER;
10    }
11    if (n < this.choice){
12      return DrawResult.YOURS_IS_LOWER;
13    }
14    return DrawResult.YOU_WON;
15  }
16
17 }
```

Interfacce della view: DrawNumberView

```
1  public interface DrawNumberView {
2
3    void setObserver(DrawNumberViewObserver observer);
4
5    void start();
6
7    void numberIncorrect();
8
9    void limitsReached();
10
11   void result(DrawResult res);
12
13 }
```

```
1  public interface DrawNumberViewObserver {
2
3    void newAttempt(int n);
4
5    void resetGame();
6
7    void quit();
8  }
```

Implementazione della view: DrawNumberViewImpl (1/3)

```
1  public class DrawNumberViewImpl implements DrawNumberView {
2
3    private static final String FRAME_NAME = "Draw Number App";
4    private static final String QUIT = "Quit";
5    private static final String RESET = "Reset";
6    private static final String GO = "Go";
7    private static final Dimension WINDOW_DIMENSION = new Dimension(320,200);
8
9    private static final Map<DrawResult,String> messages = Map.of(
10       DrawResult.YOURS_IS_HIGHER,"Your number is too high",
11       DrawResult.YOURS_IS_LOWER,"Your number is too low",
12       DrawResult.YOU_WON,"You won the game!!");
13
14   private DrawNumberViewObserver observer;
15   private JFrame frame = new JFrame(FRAME_NAME);
16
17   public DrawNumberViewImpl() {
18     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19     frame.setSize(WINDOW_DIMENSION);
20     frame.getContentPane().add(new JPanel(new BorderLayout()));
21     final JPanel pNorth = new JPanel(new FlowLayout());
22     final JTextField tNumber = new JTextField(10);
23     final JButton bGo = new JButton(GO);
24     pNorth.add(tNumber);
25     pNorth.add(bGo);
26     final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
27     final JButton bReset = new JButton(RESET);
```

Implementazione della view: DrawNumberViewImpl (2/3)

```
    final JButton bQuit = new JButton(QUIT);
    pSouth.add(bReset);
    pSouth.add(bQuit);
    frame.getContentPane().add(pNorth,BorderLayout.NORTH);
    frame.getContentPane().add(pSouth,BorderLayout.SOUTH);
    bGo.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        try{
          observer.newAttempt(Integer.parseInt(tNumber.getText()));
        } catch (NumberFormatException exception){
          JOptionPane.showMessageDialog(frame, "An integer please..");
        }
      }});
    bQuit.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        if (confirmDialog("Confirm quitting?","Quit")){
          observer.quit();
        }
      }});
    bReset.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        if (confirmDialog("Confirm resetting?","Reset")){
          observer.resetGame();
        }
      }});
  }

  public void start(){
```

Implementazione della view: DrawNumberViewImpl (3/3)

```
1    }
2
3    private boolean confirmDialog(String question, String name){
4      return JOptionPane.showConfirmDialog(frame,question,name,JOptionPane.
       YES_NO_OPTION)
5          == JOptionPane.YES_OPTION;
6    }
7
8    public void setObserver(DrawNumberViewObserver observer){
9      this.observer = observer;
10   }
11
12   public void numberIncorrect() {
13     JOptionPane.showMessageDialog(frame, "Incorrect Number.. try again",
14         "Incorrect Number", JOptionPane.ERROR_MESSAGE);
15   }
16
17   public void limitsReached() {
18     JOptionPane.showMessageDialog(frame, "You lost.. a new game starts",
19         "Lost", JOptionPane.WARNING_MESSAGE);
20   }
21
22   public void result(DrawResult res) {
23       JOptionPane.showMessageDialog(frame, messages.get(res),
24                                     "Result",JOptionPane.PLAIN_MESSAGE);
25   }
26 }
```

Implementazione del controller: DrawNumberApp (1/2)

```
1  public class DrawNumberApp implements DrawNumberViewObserver {
2
3    private static final int MIN = 0;
4    private static final int MAX = 100;
5    private static final int ATTEMPTS = 10;
6    private final DrawNumber model;
7    private final DrawNumberView view;
8
9    public DrawNumberApp() {
10     this.model = new DrawNumberImpl(MIN,MAX,ATTEMPTS);
11     this.view = new DrawNumberViewImpl();
12     this.view.setObserver(this);
13     this.view.start();
14   }
15
16   public void newAttempt(int n) {
17     try {
18         final DrawResult result = this.model.attempt(n);
19       this.view.result(result);
20       if (result == DrawResult.YOU_WON){
21           this.quit();
22       }
23     } catch (IllegalArgumentException e) {
24       this.view.numberIncorrect();
25     } catch (AttemptsLimitReachedException e) {
26       this.view.limitsReached();
27     }
28   }
```

Implementazione del controller: DrawNumberApp (2/2)

```
1
2     public void resetGame() {
3       this.model.reset();
4     }
5
6     public void quit() {
7       System.exit(0);
8     }
9
10    public static void main(String[] args) {
11      new DrawNumberApp();
12    }
13
14  }
```

Linee guida generali consigliate su MVC

Metodologia proposta

- progettare le 3 interfacce
  - M: metodi di "dominio", chiamati da C
  - C: metodi (`void`) chiamati da V, esprimono "azioni utente"
  - V: metodi (`void`) chiamati da C, esprimono richieste di visualizzazione
- la tecnologia scelta per le GUI sia interna alla View, e mai menzionata altrove o nelle interfacce
- implementare separatamente M, V e C, poi comporre e testare
- tipicamente: M, V e C si compongono di varie sottoparti

Aspetti

- MVC è implementato in vari modi
- l'approccio proposto è particolarmente indicato per la sua semplicità
- si usino altri approcci se non peggiorativi