

# 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. 2024/2025

## 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

- 1 Introduzione
- 2 Il layout dei pannelli
- 3 La gestione degli eventi nelle GUI
- 4 Alcune funzionalità avanzate GUI
- 5 Organizzazione applicazioni grafiche con MVC

# Graphical User Interfaces (GUI)

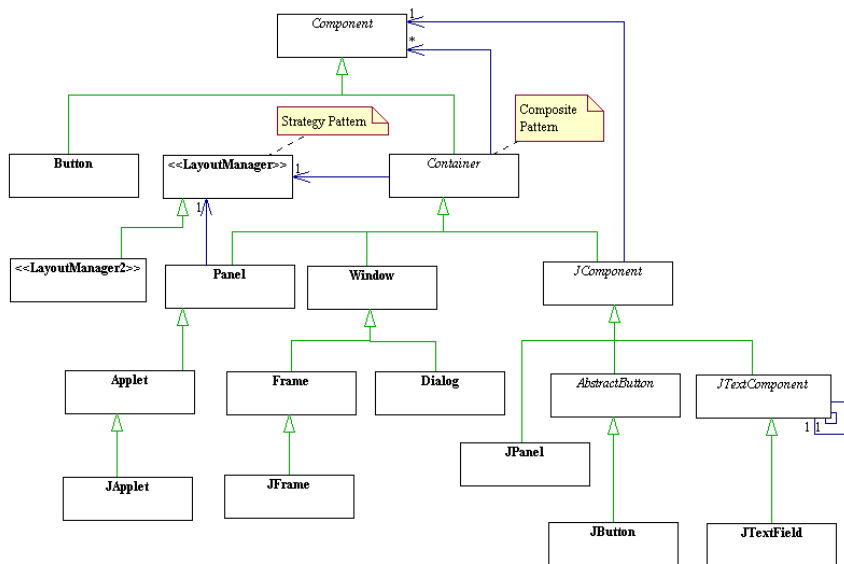
## GUI

- 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)
- ⇒ Vedremo Swing, che si appoggia su AWT
- ⇒ In laboratorio vedrete la più moderna JavaFX

# AWT e Swing: UML



# AWT, Swing e concetti principali

## I due package

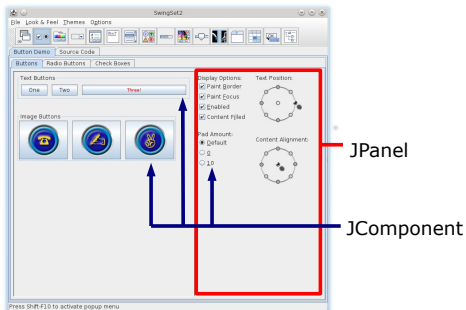
- `java.awt`: Classi base e implementazioni supportate dal S.O.
  - ▶ non molto utili da guardare in dettaglio
  - ▶ fornisce comunque l'architettura base
- `javax.swing`: implementazioni gestite “pixel per pixel”
  - ▶ le classi `J*` e quelle sottostanti

## 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

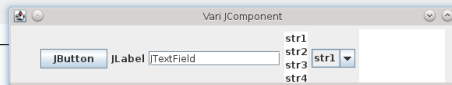


```
1 import javax.swing.*;
2
3 public class TrySwing {
4     public static void main(String[] args){
5         // Creo il frame e imposto titolo e altre proprietà
6         final JFrame frame = new JFrame();
7         frame.setTitle("Prova di JFrame");
8         frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
9         frame.setSize(320,240);
10
11        // Creo un pannello e gli imposto il bordo
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 visibile il JFrame
20        frame.setVisible(true);
21    }
22 }
```

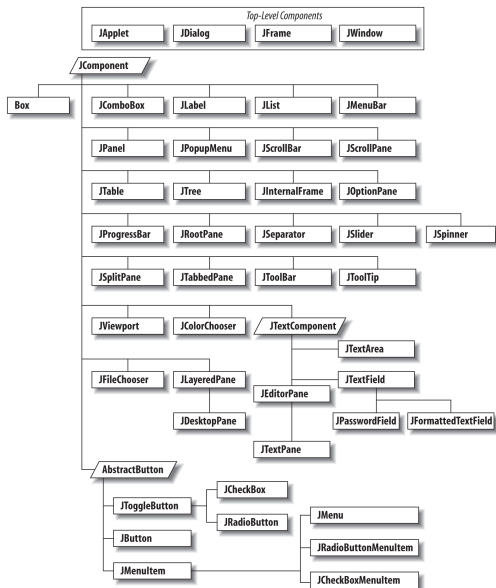


# Vari JComponent disponibili..

```
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 visibile il JFrame
25        frame.setVisible(true);
26    }
27 }
```



# Classi di Swing



## 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

- 1 Introduzione
- 2 Il layout dei pannelli**
- 3 La gestione degli eventi nelle GUI
- 4 Alcune funzionalità avanzate GUI
- 5 Organizzazione applicazioni grafiche con MVC

# Il problema del Layout di un pannello

## Problema

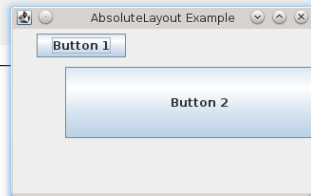
- Intervenire sulla politica di dislocazione dei componenti
- Scegliere politiche indipendenti dalle dimensioni della finestra
- Organizzare tali selezioni con una buona organizzazione OO

## 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

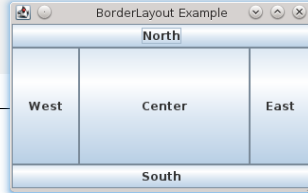
# Senza Layout – deprecabile

```
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 }
```

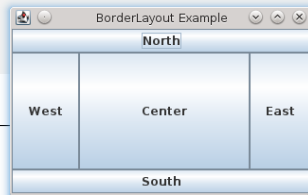


# Lavorare specializzando JFrame: MyFrame

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 /* Specializzazione di JFrame:
5  * - JFrame è Serializable!
6  * - Il costruttore accetta titolo e layout-manager
7  * - Si aggiunge il JPanel
8  * - Un metodo getMainPanel() ci dà il pannello
9  */
10 public class MyFrame extends JFrame{
11
12     private final JPanel jp;
13
14     public MyFrame(String title, LayoutManager lm){
15         super(title);
16         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17         this.setSize(320,200);
18         // Il layout-manager può essere passato al costruttore di JPanel
19         this.jp = new JPanel(lm);
20         this.getContentPane().add(this.jp);
21     }
22
23     public JPanel getMainPanel(){
24         return this.jp;
25     }
26
27 }
```



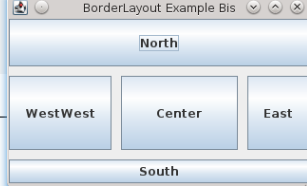
# Nuova versione BorderLayout2



```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class BorderLayout2{
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 }
```

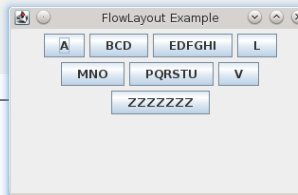
# Qualche modifica: UseBorderLayout3

```
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 aggiuntivi del lay-man
8         b.setVgap(10);
9         final JFrame frame = new JFrame("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.getContentPane().add(button,BorderLayout.NORTH);
15         frame.getContentPane().add(new JButton("South"),BorderLayout.SOUTH);
16         frame.getContentPane().add(new JButton("Center"),BorderLayout.CENTER);
17         frame.getContentPane().add(new JButton("East"),BorderLayout.EAST);
18         // nota l'effetto di una stringa più lunga qui
19         frame.getContentPane().add(new JButton("WestWest"),BorderLayout.WEST);
20
21         frame.setVisible(true);
22     }
23 }
```



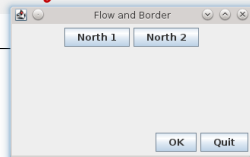
# FlowLayout

```
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 JFrame frame = new JFrame("FlowLayout Example",lay);
8         frame.getContentPane().add(new JButton("A"));
9         frame.getContentPane().add(new JButton("BCD"));
10        frame.getContentPane().add(new JButton("EDFGHI"));
11        frame.getContentPane().add(new JButton("L"));
12        frame.getContentPane().add(new JButton("MNO"));
13        frame.getContentPane().add(new JButton("PQRSTU"));
14        frame.getContentPane().add(new JButton("V"));
15        frame.getContentPane().add(new JButton("ZZZZZZ"));
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 }
```



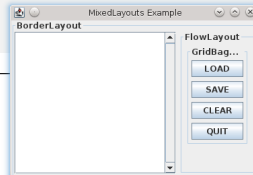
# Un uso combinato di FlowLayout e BorderLayout

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class UseFlowBorder{
5     public static void main(String[] args){
6         final JFrame frame = new JFrame("Flow and Border",new BorderLayout());
7
8         // Creo un sotto-pannello per la parte NORTH
9         final JPanel pNorth = new JPanel(new FlowLayout());
10        pNorth.add(new JButton("North 1"));
11        pNorth.add(new JButton("North 2"));
12
13        // Creo un sotto-pannello per la parte SOUTH
14        final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
15        pSouth.add(new JButton("OK"));
16        pSouth.add(new JButton("Quit"));
17
18        // Grazie al polimorfismo, aggiungo pannelli
19        frame.getContentPane().add(pNorth,BorderLayout.NORTH);
20        frame.getContentPane().add(pSouth,BorderLayout.SOUTH);
21
22        frame.setVisible(true);
23    }
24 }
```



# Altro uso combinato + GridBagLayout

```
1 public class UseMixedLayouts{
2
3     public static void main(String[] args){
4         final JTextArea textArea = new JTextArea(); // Area di testo
5         textArea.setLineWrap(true);
6         final JScrollPane scroll = new JScrollPane(textArea); // Pannello con barra
7         scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
8
9         final JPanel pEastInternal = new JPanel(new GridBagLayout()); // Griglia flessibile
10        pEastInternal.setBorder(new TitledBorder("GridBagLayout"));
11        final GridBagConstraints cnst = new GridBagConstraints();
12        cnst.gridy = 0; // 1-a riga
13        cnst.insets = new Insets(3,3,3,3); // spazio attorno al comp.
14        cnst.fill = GridBagConstraints.HORIZONTAL; // estensione in orizzont.
15        pEastInternal.add(new JButton("LOAD"),cnst);
16        cnst.gridy++; // prossima riga
17        pEastInternal.add(new JButton("SAVE"),cnst);
18        cnst.gridy++;
19        pEastInternal.add(new JButton("CLEAR"),cnst);
20        cnst.gridy++;
21        pEastInternal.add(new JButton("QUIT"),cnst);
22
23        final JPanel pEast = new JPanel(new FlowLayout());
24        pEast.setBorder(new TitledBorder("FlowLayout"));
25        pEast.add(pEastInternal);
26
27        final MyFrame frame = new MyFrame("MixedLayouts Example",new BorderLayout(5,5));
28        frame.getMainPanel().add(scroll,BorderLayout.CENTER);
29        frame.getMainPanel().add(pEast,BorderLayout.EAST);
30        frame.getMainPanel().setBorder(new TitledBorder("BorderLayout"));
31        frame.setVisible(true);
32    }
```



# 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** per la GUI che non menziona Swing
- i dettagli implementativi al solito siano delegati ad una classe che implementa, e che si compone di un JFrame

# Lavorare incapsulando il frame dietro una interfaccia

```
1 public interface UserInterface {
2
3     void show();
4
5     void setDimensions(int x, int y);
6 }
```

```
1 public class UserInterfaceImpl implements UserInterface {
2
3     private final MyFrame frame;
4
5     public UserInterfaceImpl() {
6         this.frame = new MyFrame("Flow and Border", new BorderLayout());
7         final JPanel pNorth = new JPanel(new FlowLayout());
8         pNorth.add(new JButton("North 1"));
9         final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
10        pSouth.add(new JButton("OK"));
11        this.frame.getMainPanel().add(pNorth, BorderLayout.NORTH);
12        this.frame.getMainPanel().add(pSouth, BorderLayout.SOUTH);
13    }
14
15    public void show(){
16        this.frame.setVisible(true);
17    }
18
19    public void setDimensions(int x, int y) {
20        this.frame.setSize(x,y);
21    }
22 }
```

# 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

- 1 Introduzione
- 2 Il layout dei pannelli
- 3 La gestione degli eventi nelle GUI**
- 4 Alcune funzionalità avanzate GUI
- 5 Organizzazione applicazioni grafiche con MVC

# La gestione degli eventi

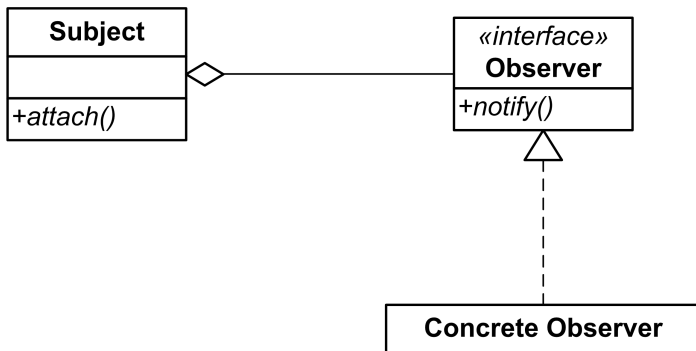
## Come rendere le interfacce “vive”?

- 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

```
1 import java.awt.event.*;
2
3 // Nota: si potrebbero usare delle inner class
4 // Nota: ActionCommand "abusa" delle stringhe..
5 public class MyActionListener implements ActionListener {
6
7     public void actionPerformed(ActionEvent e){
8         if (e.getActionCommand().equals("hello")){
9             System.out.println("Hello!!");
10        } else if (e.getActionCommand().equals("quit")){
11            System.out.println("Quitting..");
12            System.exit(0);
13        } else if (e.getActionCommand().equals("loop")){
14            System.out.println("Going stuck..");
15            for(;;){} // Nota l'effetto del loop sulla GUI
16        }
17    }
18 }
```

# Versione incapsulata (inner listener + source eventi)

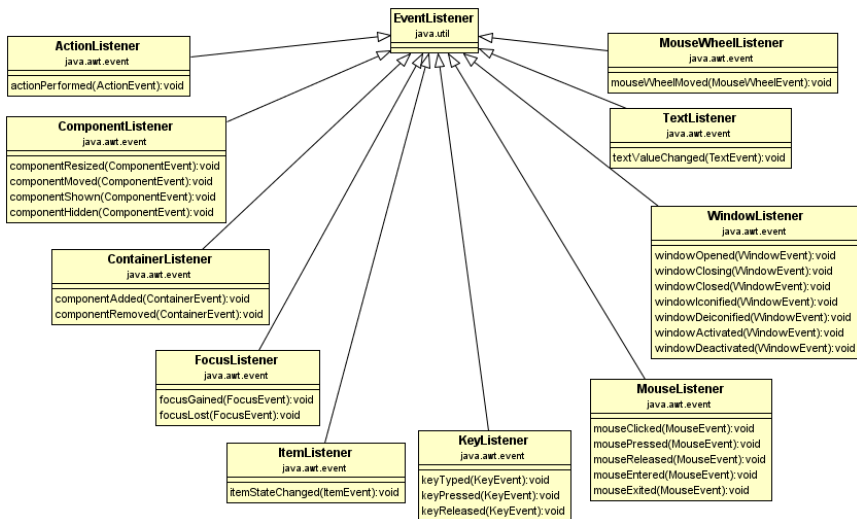
```
1 public class EventsFrame {
2
3     private final JButton b1 = new JButton("Say Hello");
4     private final JButton b2 = new JButton("Quit");
5     private final JButton b3 = new JButton("Looping..");
6
7     public EventsFrame() {
8         final MyFrame frame = new MyFrame("Events Example", new 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        }
33    }
```

# 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.getContentPane().add(b1);
24         frame.getContentPane().add(b2);
25         frame.setVisible(true);
26     }
27 }
```



# Panoramica eventi-listeners



## Il flusso di controllo con le GUI di Swing

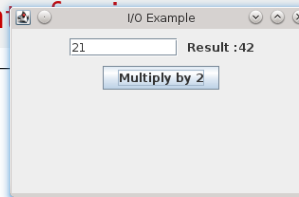
- Quando si crea un `JFrame`, la JVM crea l'`EventDispatchThread` (EDT)
- Quindi l'applicazione non termina quando il `main` completa
- Quando un evento si verifica la JVM fa eseguire il corrispondente codice all'EDT
- Ecco perché la GUI non risponde a nuovi eventi finché uno precedente non è stato gestito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

# Outline

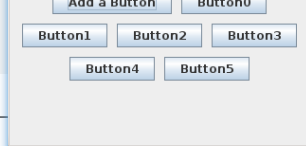
- 1 Introduzione
- 2 Il layout dei pannelli
- 3 La gestione degli eventi nelle GUI
- 4 Alcune funzionalità avanzate GUI**
- 5 Organizzazione applicazioni grafiche con MVC

# 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 }
```

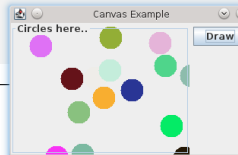


# GUI con Layout dinamico



```
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.getContentPane();
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



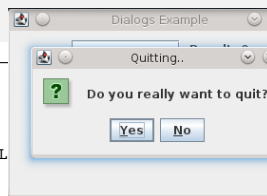
```
1 public class UseCanvas{
2
3     public static void main(String[] args){
4         final DrawPanel pCenter = new DrawPanel();
5         pCenter.setBorder(new TitledBorder("Circles here.."));
6         // Intercetto i click del mouse!
7         // L'Adapter già implementa banalmente i metodi, quindi 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

```
1 // Specializzazione ad-hoc per un JPanel
2 public class DrawPanel extends JPanel {
3
4     private static final long serialVersionUID = 7114066347061701832L;
5     private static final int RADIUS = 20;
6     private static final Random RND = new Random();
7     private final Map<Point,Color> circles = new HashMap<>();
8
9     // override del metodo di disegno
10    protected void paintComponent(Graphics g) {
11        super.paintComponent(g);
12        for (Map.Entry<Point,Color> e : this.circles.entrySet()) {
13            g.setColor(e.getValue());
14            g.fillOval(e.getKey().x, e.getKey().y, RADIUS * 2, RADIUS * 2);
15        }
16    }
17
18    // Metodo per aggiungere nuovi cerchi in posizione random
19    public void addRandomPoint(){
20        int x = RND.nextInt(this.getWidth());
21        int y = RND.nextInt(this.getHeight());
22        this.addPoint(x,y);
23    }
24
25    // Metodo per aggiungere nuovi cerchi
26    public void addPoint(int x, int y){
27        final Color c = new Color(RND.nextInt(256),RND.nextInt(256),RND.nextInt(256));
28        this.circles.put(new Point(x - RADIUS, y - RADIUS), c);
29    }
30 }
```

# 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_CLOSE);
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    }
```

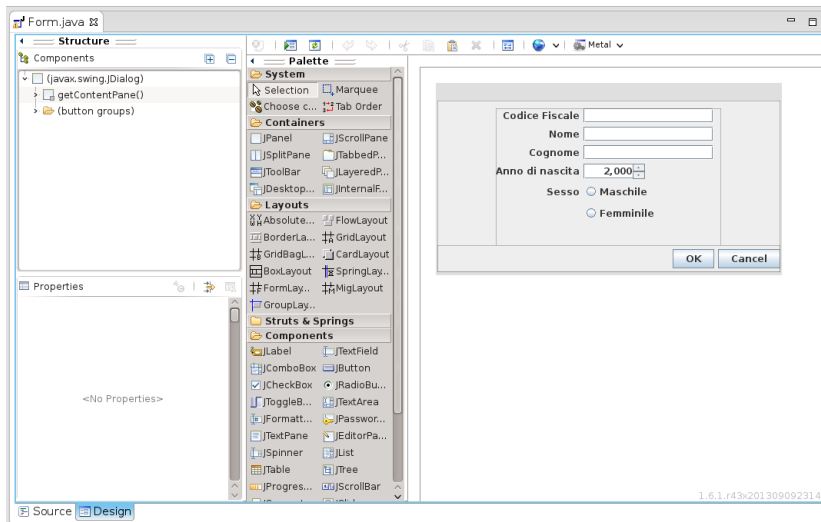




## Cosa sono?

- 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

- 1 Introduzione
- 2 Il layout dei pannelli
- 3 La gestione degli eventi nelle GUI
- 4 Alcune funzionalità avanzate GUI
- 5 Organizzazione applicazioni grafiche con MVC**

# 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)

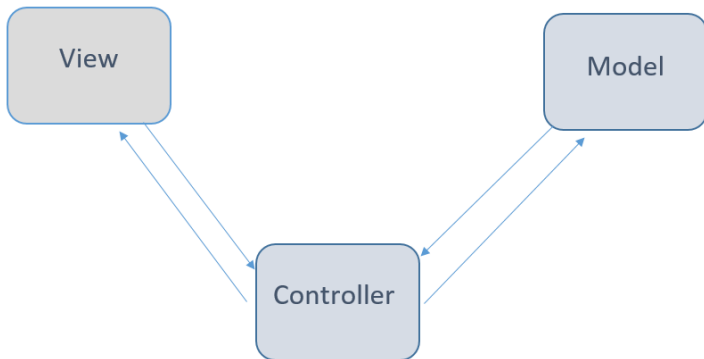
## Pattern architetturale

- è una architettura software ritenuta ricorrente e con benefici, e quindi riusabile
- è descritta generalmente, e quindi è da calare nel contesto specifico 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

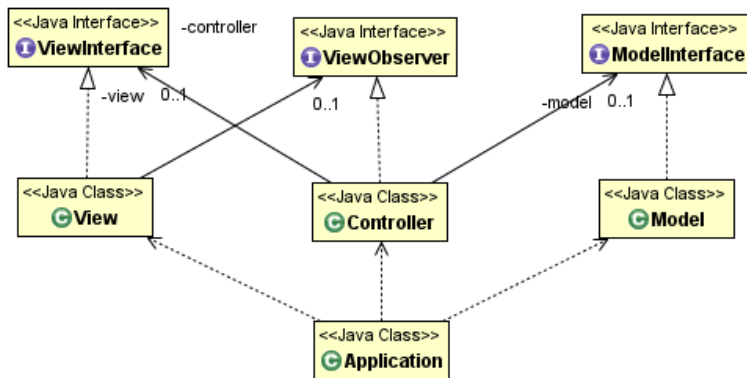
## Sulla costruzione di applicazioni con GUI

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

## Alcune linee guida

- Usare il pattern MVC per la struttura generale
- Identificare 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

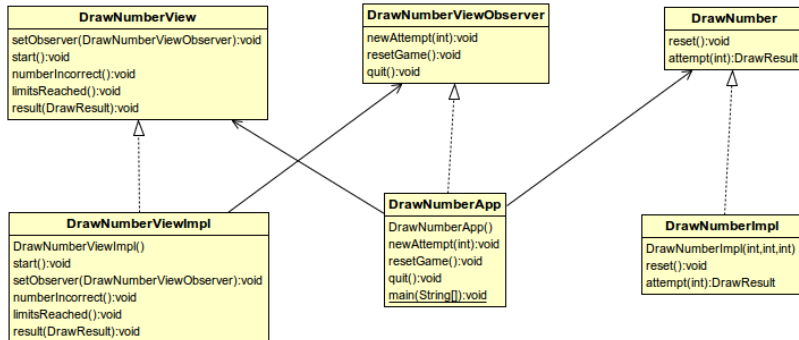
- Model: incapsula dati e logica relativi al **dominio** della applicazione
- View: incapsula la GUI, le sue sottoparti, e la logica di notifica
- Controller: realizza l'applicazione SW: intercetta gli eventi della View, comanda le modifiche al modello, cambia di conseguenza la View

## 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

```
1 public interface DrawNumber {  
2  
3     void reset();  
4  
5     DrawResult attempt(int n);  
6 }
```

```
1 public enum DrawResult {  
2     YOURS_IS_LOWER, YOURS_IS_HIGHER, YOU_WON, YOU_LOST;  
3 }
```

# Implementazione del model: DrawNumberImpl

```
1 public class DrawNumberImpl implements DrawNumber {
2     private final int min;
3     private final int max;
4     private final int attempts;
5     private int remainingAttempts;
6     private int choice;
7
8     public DrawNumberImpl(final int min, final int max, final int attempts) {
9         this.min = min;
10        this.max = max;
11        this.attempts = attempts;
12        this.reset();
13    }
14
15    public void reset() {
16        this.remainingAttempts = this.attempts;
17        this.choice = this.min + new Random().nextInt(this.max - this.min + 1);
18    }
19
20    public DrawResult attempt(int n) {
21        if (this.min > n || this.max < n) {
22            throw new IllegalArgumentException();
23        }
24        return this.remainingAttempts == 0 ? DrawResult.YOU_LOST :
25            n > this.choice ? DrawResult.YOURS_IS_HIGHER :
26            n < this.choice ? DrawResult.YOURS_IS_LOWER :
27            DrawResult.YOU_WON;
28    }
29 }
```

# 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 public class DrawNumberViewImpl implements DrawNumberView {
2
3     private static final Map<DrawResult,String> messages = Map.of(
4         DrawResult.YOURS_IS_HIGHER,"Your number is too high",
5         DrawResult.YOURS_IS_LOWER,"Your number is too low",
6         DrawResult.YOU_WON,"You won the game!!");
7
8     private DrawNumberFrame frame = new DrawNumberFrame();
9
10    public void start(){
11        this.frame.setVisible(true);
12    }
13
14    public void numberIncorrect() {
15        JOptionPane.showMessageDialog(frame, "Incorrect Number.. try again",
16            "Incorrect Number", JOptionPane.ERROR_MESSAGE);
17    }
18
19    public void limitsReached() {
20        JOptionPane.showMessageDialog(frame, "You lost.. a new game starts",
21            "Lost", JOptionPane.WARNING_MESSAGE);
22    }
23
24    public void result(DrawResult res) {
25        JOptionPane.showMessageDialog(frame, messages.get(res),
26            "Result",JOptionPane.PLAIN_MESSAGE);
27    }
28
29    public void setObserver(DrawNumberViewObserver observer) {
30        this.frame.setObserver(observer);
31    }
32 }
```

# Implementazione della view: DrawNumberFrame

```
1 class DrawNumberFrame extends JFrame {
2
3     private DrawNumberViewObserver observer;
4     private final ActionProcessor processor = new ActionProcessor();
5     private final JButton bGo = new JButton("Go");
6     private final JButton bReset = new JButton("Reset");
7     private final JButton bQuit = new JButton("Quit");
8     private final JTextField tNumber = new JTextField(10);
9
10    public DrawNumberFrame() {
11        this.setTitle("Draw Number App");
12        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13        this.setSize(new Dimension(640, 400));
14        this.getContentPane().add(new JPanel(new BorderLayout()));
15        final JPanel pNorth = new JPanel(new FlowLayout());
16        pNorth.add(tNumber);
17        pNorth.add(bGo);
18        final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
19        pSouth.add(bReset);
20        pSouth.add(bQuit);
21        this.getContentPane().add(pNorth, BorderLayout.NORTH);
22        this.getContentPane().add(pSouth, BorderLayout.SOUTH);
23        bGo.addActionListener(processor);
24        bQuit.addActionListener(processor);
25        bReset.addActionListener(processor);
26    }
27
28    public void setObserver(DrawNumberViewObserver observer) {
29        this.observer = observer;
30    }
31
32    // inner class ActionProcessor...
```

# Action Processor inner class

```
1 private class ActionProcessor implements ActionListener {
2     public void actionPerformed(ActionEvent e) {
3         if (e.getSource() == bGo) {
4             try {
5                 observer.newAttempt(Integer.parseInt(tNumber.getText()));
6             } catch (NumberFormatException exception) {
7                 JOptionPane.showMessageDialog(
8                     DrawNumberFrame.this, "An integer please..");
9             }
10        } else if (e.getSource() == bQuit) {
11            if (confirmDialog("Confirm quitting?", "Quit")) {
12                observer.quit();
13            }
14        } else {
15            if (confirmDialog("Confirm resetting?", "Reset")) {
16                observer.resetGame();
17            }
18        }
19    }
20
21    private boolean confirmDialog(String question, String name) {
22        return JOptionPane.showConfirmDialog(DrawNumberFrame.this, question, name,
23            JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION;
24    }
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             if (result == DrawResult.YOU_LOST){
20                 this.view.limitsReached();
21             }
22             this.view.result(result);
23             if (result == DrawResult.YOU_WON){
24                 this.quit();
25             }
26         } catch (IllegalArgumentException e) {
27             this.view.numberIncorrect();
28         }
29     }
30 }
```



## Implementazione del controller: DrawNumberApp (2/2)

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

# 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