

16

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. 2015/2016



Goal della lezione

- Illustrare la libreria Java Swing
- Introdurre qualche rudimento di programmazione ad eventi
- 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



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



Graphical User Interfaces (GUI)

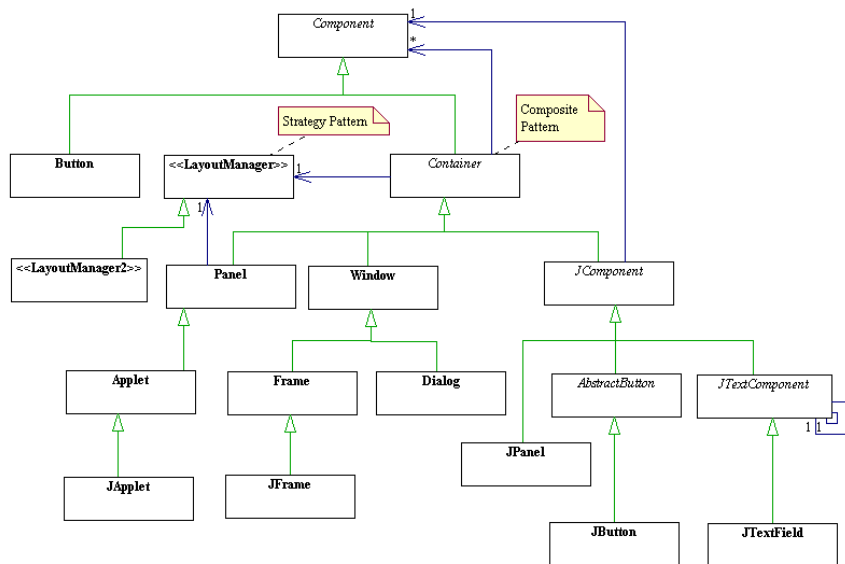
GUI

- Interfacce grafiche per l'interazione con l'utente
- Ritenute più semplice 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

AWT e Swing: UML



AWT, Swing e concetti principali

I due package

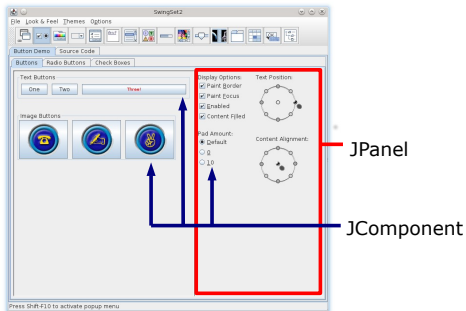
- `java.awt`: Classi base e implementazioni supportate dal S.O.
 - ▶ non molto utili da guardare in dettaglio
 - ▶ fornisce comunque l'architettura base
- `javax.swing`: implementazioni gestite “pixel per pixel”
 - ▶ le classi `J*` e quelle sottostanti

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

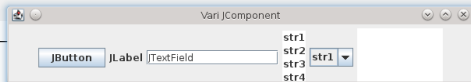


```
1 import javax.swing.*;
2
3 public class TrySwing {
4     public static void main(String[] args){
5         // Creo il frame e imposto titolo e altre proprietà
6         final JFrame frame = new JFrame();
7         frame.setTitle("Prova di JFrame");
8         frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
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' 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 }
```

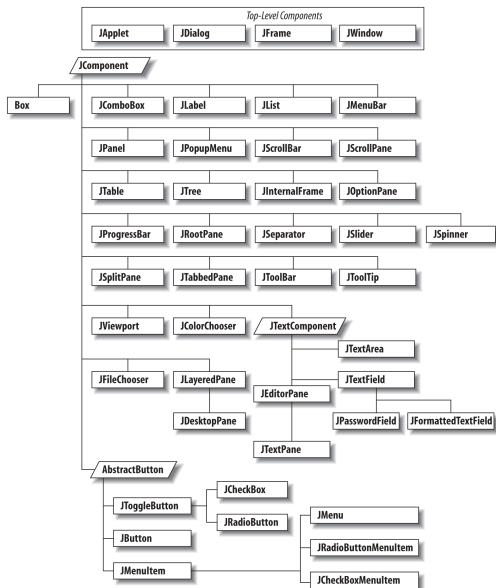


Vari JComponent disponibili..

```
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[] str = 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>(str));
21        panel.add(new JComboBox<String>(str));
22        panel.add(new JTextArea(5,10));
23
24        // Alla fine rendo visibile il JFrame
25        frame.setVisible(true);
26    }
27 }
```



Classi di Swing



Materiale da consultare

Collezione di riferimenti utili

- JavaDoc delle librerie:
 - ▶ <http://docs.oracle.com/javase/8/docs/api>
- Tutorial ufficiali:
 - ▶ <http://docs.oracle.com/javase/tutorial/uiswing/>
- Esempi d'uso di Java:
 - ▶ <http://download.oracle.com/otn-pub/java/jdk/8u25-b17-demos/jdk-8u25-linux-x64-demos.tar.gz>
 - ▶ e in particolare SwingSet2 nella cartella `demo/jfc/`

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 Elementi di programmazione ad eventi
- 5 Alcune funzionalità avanzate GUI
- 6 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”

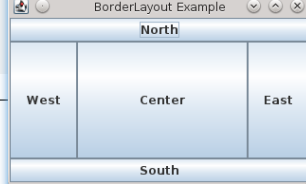
- Al pannello si passa un oggetto di `LayoutManager`
- È lui che incapsula la strategia di inserimento dei componenti
- Vari casi: `FlowLayout` (default), `BorderLayout`, `GridBagLayout`,... (tipicamente da comporre tra loro)
- Vedere: <http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
- 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(new JButton("Button 1"));
19        panel.add(new JButton("Button 2"));
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

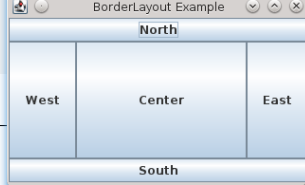


```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class UseBorderLayout {
5     public static void main(String[] args){
6         final JFrame frame = new JFrame();
7         frame.setTitle("BorderLayout Example");
8         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9         frame.setSize(320,200);
10
11         final JPanel panel = new JPanel();
12         panel.setLayout(new BorderLayout()); // Imposto il layout
13         frame.getContentPane().add(panel);
14
15         // Nota l'argomento aggiuntivo di tipo int
16         // Nota che i pulsanti non usano la loro dim. preferita!
17         panel.add(new JButton("North"),BorderLayout.NORTH);
18         panel.add(new JButton("South"),BorderLayout.SOUTH);
19         panel.add(new JButton("Center"),BorderLayout.CENTER);
20         panel.add(new JButton("East"),BorderLayout.EAST);
21         panel.add(new JButton("West"),BorderLayout.WEST);
22
23         frame.setVisible(true);
24     }
25 }
```

Una specializzazione di frame: MyFrame

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 /* Specializzazione di JFrame:
5  * - JFrame è Serializable!
6  * - Il costruttore accetta titolo e lay-man
7  * - Si aggiunge il JPanel
8  * - Un metodo getMainPanel() ci dà il pannello
9  */
10 public class MyFrame extends JFrame{
11
12     private static final long serialVersionUID = -1947466585058197323L;
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 lay-man può essere passato al costruttore di JPanel
19         this.getContentPane().add(new JPanel(lm));
20     }
21
22     public JPanel getMainPanel(){
23         // Libreria pre-generici
24         return (JPanel)this.getContentPane().getComponent(0);
25     }
26
27 }
```


Nuova versione UseBorderLayout2

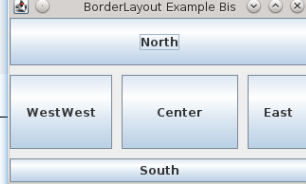


```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class UseBorderLayout2{
5     public static void main(String[] args){
6         final JFrame frame =
7             new JFrame("BorderLayout Example",new BorderLayout());
8
9         frame.getContentPane().add(new JButton("North"),BorderLayout.NORTH);
10        frame.getContentPane().add(new JButton("South"),BorderLayout.SOUTH);
11        frame.getContentPane().add(new JButton("Center"),BorderLayout.CENTER);
12        frame.getContentPane().add(new JButton("East"),BorderLayout.EAST);
13        frame.getContentPane().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 }
```

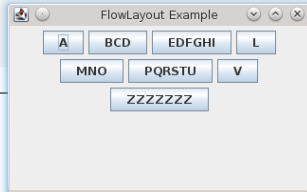


Qualche modifica: UseBorderLayout3

```
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.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 }
```



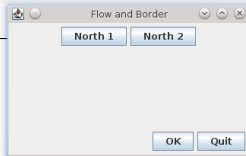
FlowLayout



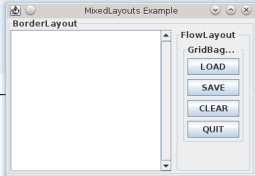
```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class UseFlowLayout{
5     public static void main(String[] args){
6         final FlowLayout lay = new FlowLayout(FlowLayout.CENTER);
7         final JFrame frame = new JFrame("FlowLayout Example",lay);
8         frame.getContentPane().add(new 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("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

```
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    }
```

Versione totalmente incapsulata

```
1 public class EncapsulatedFrame {
2
3     private final MyFrame frame;
4
5     public EncapsulatedFrame() {
6
7         this.frame = new MyFrame("Flow and Border", new BorderLayout());
8         final JPanel pNorth = new JPanel(new FlowLayout());
9         pNorth.add(new JButton("North 1"));
10        pNorth.add(new JButton("North 2"));
11        final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
12        pSouth.add(new JButton("OK"));
13        pSouth.add(new JButton("Quit"));
14        this.frame.getContentPane().add(pNorth, BorderLayout.NORTH);
15        this.frame.getContentPane().add(pSouth, BorderLayout.SOUTH);
16    }
17
18    public void show(){
19        this.frame.setVisible(true);
20    }
21
22    // altri metodi d'accesso al frame.. solo quelli che servono
23    // si nascondono tutti i dettagli di Swing
24    // Filosofia: 'favour encapsulation over inheritance'
25 }
```

```
1 public class UseEncapsulatedFrame {
2
3     public static void main(String[] args) {
4         new EncapsulatedFrame().show(); // Nota la minimalità!
5     }
6 }
```

Outline

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



La gestione degli eventi

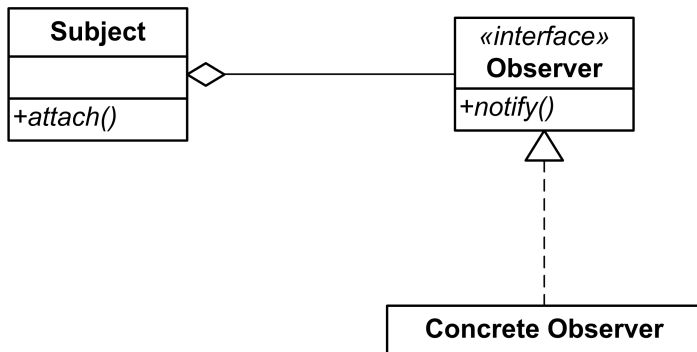
Come rendere le interfacce “vive”?

- 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)`
- All'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



```
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

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



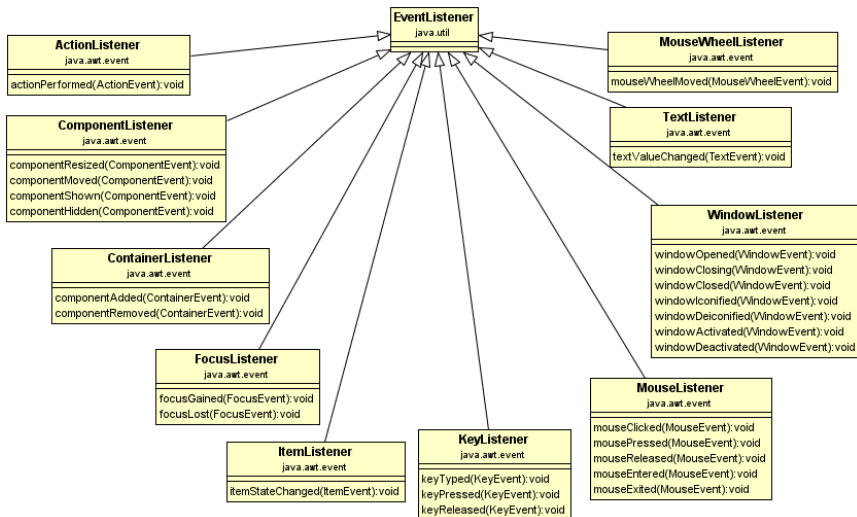
Versione incapsulata (inner listener + source eventi)

```
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.getMainPanel().add(b1);
24         frame.getMainPanel().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 Elementi di programmazione ad eventi**
- 5 Alcune funzionalità avanzate GUI
- 6 Organizzazione applicazioni grafiche con MVC

Programmazione sequenziale vs event-based

Programmazione sequenziale

- Un programma esegue passo-passo delle istruzioni, e controlla di quando in quando (bloccandosi o meno) se ha ricevuto input dall'esterno, e di quando in quando produce degli output
- È lo stile tradizionale della computazione “algoritmica”

Programmazione ad eventi

- All'arrivo di un evento, uno (o più) handler corrispondente viene eseguito fino alla sua terminazione, eventualmente (non necessariamente) producendo un output
- È lo stile della computazione “interattiva”, usata tipicamente con le GUI, i sistemi Web, i sistemi embedded, i sistemi robotici



Implementiamo una mini-libreria per il pattern observer

Classe ESource e interfaccia EObserver

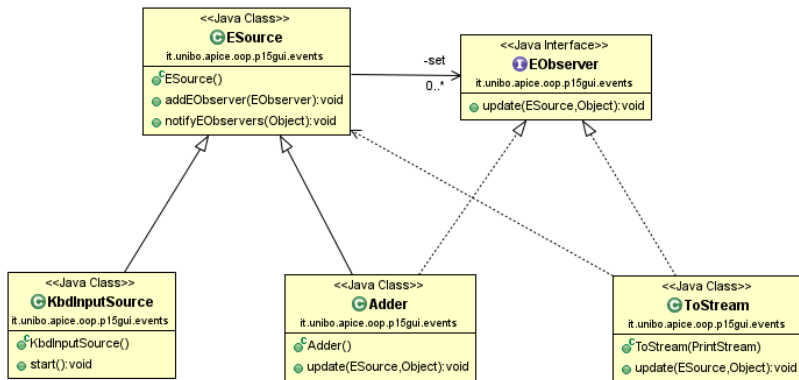
- Un ESource è un generatore di eventi
- Accetta registrazioni da uno o più EObserver
- EObserver ha un metodo update per ricevere gli eventi
- ESource fornisce un metodo per notificarli tutti

Esempio di applicazione ad eventi

- Leggiamo da tastiera un insieme di numeri positivi
- All'arrivo di un numero negativo vogliamo stampare la somma dei numeri ottenuti fin lì sia su tastiera che su file (in generale su un qualunque stream)
- Concepiamo il sistema ad "eventi"



Diagramma delle classi



Idea: 4 oggetti

- un `KbdInputSource` che genera un evento per ogni intero digitato
- un `Adder` che genera l'evento "somma" quando giunge un negativo
- due `ToStream` che ricevono quest'ultimo e lo stampano

Classe di partenza

```
1 import java.io.*;
2
3 public class Application {
4
5     private static final String FILE_NAME = "/home/mirko/aula/16/prova.txt";
6
7     public static void main(String[] args) throws IOException {
8         // Genero i 4 oggetti
9         try (final PrintStream ps = new PrintStream(
10             new FileOutputStream(FILE_NAME))) {
11             final KbdInputSource source = new KbdInputSource();
12             final Adder adder = new Adder();
13             final ToStream toConsole = new ToStream(System.out);
14             final ToStream toFile = new ToStream(ps);
15             // Li collego registrando i listener
16             adder.addObserver(toFile);
17             adder.addObserver(toConsole);
18             source.addObserver(adder);
19
20             // Faccio partire la applicazione
21             source.start();
22         }
23     }
24 }
25 }
```

EObserver e ESource

```
1 public interface EObserver {  
2  
3     public void update(ESource s, Object arg);  
4  
5 }
```

```
1 import java.util.*;  
2  
3 public class ESource {  
4  
5     private final Set<EObserver> set = new HashSet<>();  
6  
7     public void addEObserver(EObserver obs){  
8         this.set.add(obs);  
9     }  
10  
11     public void notifyEObservers(Object arg){  
12         for (final EObserver obs : this.set){  
13             obs.update(this, arg);  
14         }  
15     }  
16  
17 }
```

KbdInputSource

```
1 import java.io.*;
2
3 public class KbdInputSource extends ESource {
4
5     // Metodo di partenza
6     public void start() {
7         final BufferedReader br = new BufferedReader(new InputStreamReader(
8             System.in));
9         int i = 0;
10        do { // Con valori negativi esco..
11            try {
12                String s = br.readLine(); // leggo
13                i = Integer.parseInt(s); // interpreto
14                this.notifyEObservers(i); // notifico
15            } catch (IOException e) {
16                System.out.println("Not understood, try again..");
17            }
18        } while (i >= 0);
19    }
20 }
```



Adder

```
1 public class Adder extends ESource implements EObserver {
2
3     // Somma degli elementi giunti finora
4     private int sum = 0;
5
6     // Alla notifica..
7     public void update(ESource o, Object arg) {
8         if (((Integer) arg) == -1) {
9             // Se -1 notifico a mia volta
10            this.notifyEObservers(sum);
11        } else {
12            // Altrimenti sommo
13            sum += (Integer) arg;
14        }
15    }
16 }
```



ToStream

```
1 import java.io.*;
2
3 public class ToStream implements EObserver{
4
5     private final PrintStream ps;
6
7     public ToStream(final PrintStream ps){
8         this.ps = ps;
9     }
10
11     public void update(ESource o, Object arg) {
12         ps.println("The sum is "+arg);
13     }
14 }
```



Considerazioni sulla programmazione ad aventi

Aspetti positivi

- Molto utile quando si gestisce l'interazione col SISOP
- Consente altissimo disaccoppiamento fra i componenti
- Consente alta flessibilità nella composizione dinamica
- Se ben realizzata porta a buone performance

Aspetti negativi

- Più sofisticata e meno intuitiva
- Più difficile intercettare errori di programmazione
- Più laboriosa
- Potrebbe scomodare aspetti di multi-threading

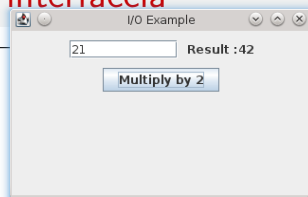


Outline

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

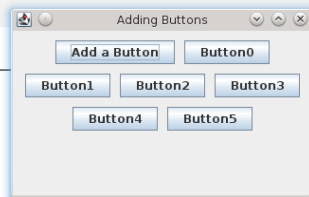
GUI con I/O: listeners che modificano l'interfaccia

```
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 }
```

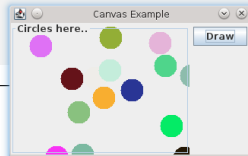


GUI con Layout dinamico

```
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



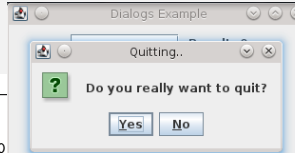
```
1 public class UseCanvas{
2
3     public static void main(String[] args){
4         final DrawPanel pCenter = new DrawPanel();
5         pCenter.setBorder(new TitledBorder("Circles here.."));
6         // Intercetto i click del mouse!
7         // L'Adapter già implementa banalmente i metodi, quindi 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 = 30;
6     private static final Random RND = new Random();
7     private final Map<Point,Color> circles = new HashMap<>();
8
9     @Override // 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, RADIUS);
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/2,y-RADIUS/2),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
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    }
```

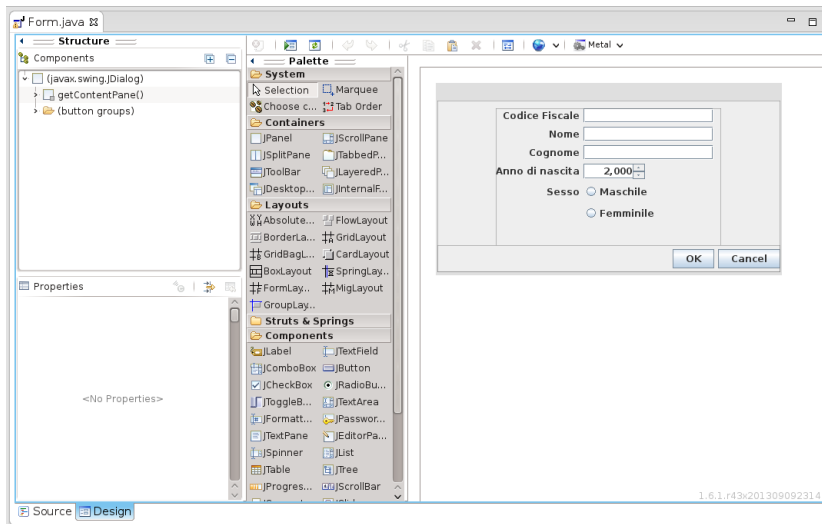


Cosa sono?

- 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 semplicemente usabili
- 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 Elementi di programmazione ad eventi
- 5 Alcune funzionalità avanzate GUI
- 6 Organizzazione applicazioni grafiche con MVC**



Applicazione del MVC

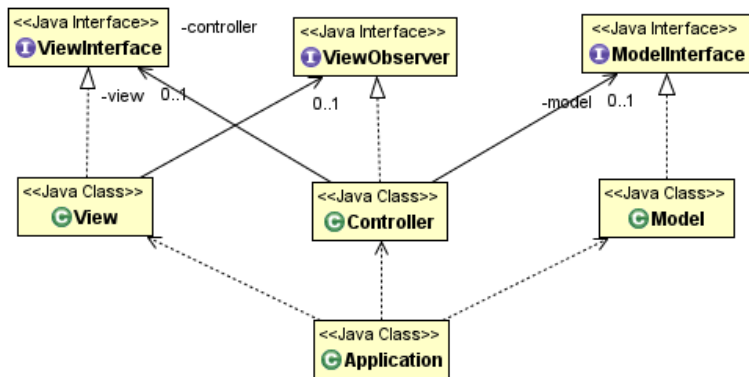
Sulla costruzione di applicazioni con GUI

- 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
- Identificare le varie “interazioni”, e quindi costruire le interfacce dei vari componenti bene fin dall’inizio
- Cercare massima indipendenza fra i vari componenti (interfacce con meno metodi possibile)
- Costruire e testare modello e GUI separatamente (M e V), poi collegare il tutto col controllore (C) che risulterà particolarmente esile
- Per la GUI eventualmente usare un GUI Builder

MVC con le GUI: un esempio di struttura



Componenti e loro interazioni

MVC

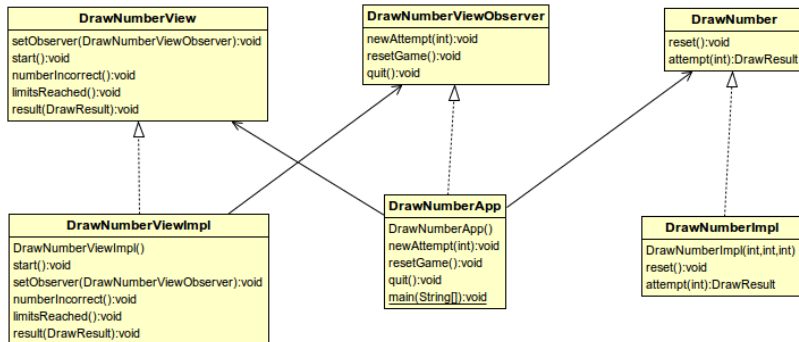
- Model: incapsula i dati che formano lo stato della applicazione
- View: incapsula la GUI, le sue sottoparti, e la logica di notifica
- Controller: intercetta gli eventi della View, comanda le modifiche al modello, cambia di conseguenza la View

Interfacce

- ModelInterface: letture/modifiche da parte del Controller
- ViewObserver: comandi inviati dalla view al controller
- 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) throws AttemptsLimitReachedException,  
6         IllegalArgumentException;  
}
```

```
1 public enum DrawResult {  
2     YOURS_LOW, YOURS_HIGH, YOU_WON;  
3 }
```

```
1 public class AttemptsLimitReachedException extends Exception {  
2  
3     private static final long serialVersionUID = 8886777512828593430L  
4     ;  
5  
6     public AttemptsLimitReachedException() {}  
}
```


Implementazione del model: DrawNumberImpl (1/2)

```
1 public class DrawNumberImpl implements DrawNumber {
2
3     private int choice;
4     private final int min;
5     private final int max;
6     private final int attempts;
7     private int remainingAttempts;
8     private final Random random = new Random();
9
10    public DrawNumberImpl(final int min, final int max, final int attempts) {
11        this.min = min;
12        this.max = max;
13        this.attempts = attempts;
14        this.reset();
15    }
16
17    @Override
18    public void reset() {
19        this.remainingAttempts = this.attempts;
20        this.choice = this.min + random.nextInt(this.max-this.min+1);
21    }
```



Implementazione del model: DrawNumberImpl (2/2)

```
1  @Override
2
3  public DrawResult attempt(int n) throws AttemptsLimitReachedException {
4      if (this.remainingAttempts == 0) {
5          throw new AttemptsLimitReachedException();
6      }
7      if (n < this.min || n > this.max){
8          throw new IllegalArgumentException();
9      }
10     if (n > this.choice){
11         return DrawResult.YOURS_HIGH;
12     }
13     if (n < this.choice){
14         return DrawResult.YOURS_LOW;
15     }
16     return DrawResult.YOU_WON;
17 }
18
19 }
```



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 (1/4)

```
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
8     private DrawNumberViewObserver observer;
9     private JFrame frame = new JFrame(FRAME_NAME);
10
11     public DrawNumberViewImpl() {
12         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         frame.setSize(320,200);
14         frame.getContentPane().add(new JPanel(new BorderLayout()));
15         final JPanel pNorth = new JPanel(new FlowLayout());
16         final JTextField tNumber = new JTextField(10);
17         final JButton bGo = new JButton(GO);
18         pNorth.add(tNumber);
19         pNorth.add(bGo);
20         final JPanel pSouth = new JPanel(new FlowLayout(FlowLayout.RIGHT));
21         final JButton bReset = new JButton(RESET);
22         final JButton bQuit = new JButton(QUIT);
23         pSouth.add(bReset);
24         pSouth.add(bQuit);
25         frame.getContentPane().add(pNorth,BorderLayout.NORTH);
26         frame.getContentPane().add(pSouth,BorderLayout.SOUTH);
```

Implementazione della view: DrawNumberViewImpl (2/4)

```
1  bGo.addActionListener(new ActionListener(){
2      public void actionPerformed(ActionEvent e) {
3          try{
4              observer.newAttempt(Integer.parseInt(tNumber.getText()));
5          } catch (NumberFormatException exception){
6              JOptionPane.showMessageDialog(frame, "An integer please..");
7          }
8      }
9  });
10 bQuit.addActionListener(new ActionListener(){
11     public void actionPerformed(ActionEvent e) {
12         if (confirmDialog("Confirm quitting?", "Quit")){
13             observer.quit();
14         }
15     }
16 });
17 bReset.addActionListener(new ActionListener(){
18     public void actionPerformed(ActionEvent e) {
19         if (confirmDialog("Confirm resetting?", "Reset")){
20             observer.resetGame();
21         }
22     }
23 });
24 }
```



Implementazione della view: DrawNumberViewImpl (3/4)

```
1 public void start(){
2     this.frame.setVisible(true);
3 }
4
5 private boolean confirmDialog(String question, String name){
6     return JOptionPane.showConfirmDialog(frame,question,name,JOptionPane.
7         YES_NO_OPTION)
8         == JOptionPane.YES_OPTION;
9 }
10
11 @Override
12 public void setObserver(DrawNumberViewObserver observer){
13     this.observer = observer;
14 }
15
16 @Override
17 public void numberIncorrect() {
18     JOptionPane.showMessageDialog(frame, "Incorrect Number.. try again",
19         "Incorrect Number", JOptionPane.ERROR_MESSAGE);
20 }
```



Implementazione della view: DrawNumberViewImpl (4/4)

```
1  @Override
2
3  public void limitsReached() {
4      JOptionPane.showMessageDialog(frame, "You lost.. a new game starts",
5          "Lost", JOptionPane.WARNING_MESSAGE);
6  }
7
8  @Override
9  public void result(DrawResult res) {
10     switch (res){
11     case YOURS_HIGH:
12         JOptionPane.showMessageDialog(frame, "Your number is too high",
13             "Result", JOptionPane.PLAIN_MESSAGE);
14         return;
15     case YOURS_LOW:
16         JOptionPane.showMessageDialog(frame, "Your number is too low",
17             "Result", JOptionPane.PLAIN_MESSAGE);
18         return;
19     case YOU_WON:
20         JOptionPane.showMessageDialog(frame, "You won the game!!",
21             "Result", JOptionPane.PLAIN_MESSAGE);
22         observer.resetGame();
23         return;
24     }
25 }
26 }
```

Implementazione del controller: DrawNumberApp (1/2)

```
1 import it.unibo.apice.oop.p16gui.mvc.model.*;
2 import it.unibo.apice.oop.p16gui.mvc.view.*;
3
4 public class DrawNumberApp implements DrawNumberViewObserver {
5
6     private final static int MIN = 0;
7     private final static int MAX = 100;
8     private final static int ATTEMPTS = 10;
9     private final DrawNumber model;
10    private final DrawNumberView view;
11
12    public DrawNumberApp() {
13        this.model = new DrawNumberImpl(MIN, MAX, ATTEMPTS);
14        this.view = new DrawNumberViewImpl();
15        this.view.setObserver(this);
16        this.view.start();
17    }
18
19    @Override
20    public void newAttempt(int n) {
21        try {
22            this.view.result(this.model.attempt(n));
23        } catch (IllegalArgumentException e) {
24            this.view.numberIncorrect();
25        } catch (AttemptsLimitReachedException e) {
26            this.view.limitsReached();
27        }
28    }
29 }
```


Implementazione del controller: DrawNumberApp (2/2)

```
1  @Override
2  public void resetGame() {
3      this.model.reset();
4  }
5
6  @Override
7  public void quit() {
8      System.exit(0);
9  }
10
11 public static void main(String[] args) {
12     new DrawNumberApp();
13 }
14
15 }
```

