INTERFACCE GRAFICHE CON SWING

I COMPONENTI SWING

- Finora le applicazioni create sono state eseguite a riga di comando e senza interfacce grafiche (User Interfaces)
- La libreria Java Swing permette di creare programmi con finestre, pulsanti, ecc.
- Tutti i controlli Swing ereditano dalla classe JComponent
- Per lavorare con Swing importare la libreria con import javax.swing.*
- javax si chiama in questo modo perché in origine si è trattato di estendere le libreria Java di default

CONTAINER / JFRAME

- Normalmente i controlli Swing vengono ospitati in *container*, ovvero in componenti che permettono di inserire al loro interno altri componenti (classe Container) (es. le finestre)

- Window è una sottoclasse di Container, ma normalmente si usa la sottoclasse di Window chiamata JFrame, che incorpora bordi, barre del titolo, ecc.

- La J iniziale è tipica di tutti i componenti Swing (JFrame, JLabel, JButton…)

JFRAME

- Per creare un'istanza di JFrame si possono usare i costruttori:
  - JFrame(): crea una finestra inizialmente invisibile e senza titolo
  - JFrame(String title): crea una finestra inizialmente invisibile con il titolo specificato

- Metodi di JFrame:
  - void setTitle(String title): imposta il titolo
  - void setSize(int width, int height): imposta larghezza e altezza in pixel
  - void setResizable(boolean resizable): imposta la possibilità di ridimensionare o meno la finestra
  - void setVisible(boolean visible): rende la finestra visible o invisibile

ESEMPIO DI CREAZIONE DI UN JFRAME

```
import javax.swing.*;

public class JFrame1 {
  public static void main(String[] args) {
    JFrame aFrame = new JFrame("First frame");
    aFrame.setSize(250, 100);
    aFrame.setVisible(true);
  }
}
```

CHIUSURA DEL JFRAME

- Nell'esempio precedente, alla chiusura della finestra, l'applicazione continua a funzionare. Per selezionare il comportamento in caso di chiusura di un JFrame si può utilizzare il metodo `setDefaultCloseOperation()` con uno dei seguenti parametri:
  - `WindowConstants.HIDE_ON_CLOSE`: (default) chiude la finestra ma l'applicazione continua a funzionare
  - `WindowConstants.DISPOSE_ON_CLOSE`: chiude la finestra, elimina l'oggetto, ma l'applicazione continua a funzionare
  - `WindowConstants.EXIT_ON_CLOSE`: esce dall'applicazione
  - `WindowConstants.DO_NOTHING_ON_CLOSE`: non effettua nessuna operazione (non consente la chiusura della finestra)

COMPONENTE JLABEL

- Uno dei componenti Swing più elementari è JLabel, che rappresenta un'etichetta testuale. I costruttori principali sono:
  - JLabel(): crea una label vuota
  - JLabel(String text): crea una label con contenuto specificato
  - … (esistono costruttori in cui includere icone da inserire nella label)

- Una volta creato un componente (ad es. una JLabel) esso deve essere aggiunto al contenitore che lo ospiterà (ad es. un JFrame) con il metodo add(Component c). Per eliminare un componente esiste invece remove(Component c)

- Il metodo setText(String text) modifica il testo della label dopo la sua creazione
- Il metodo getText() restituisce il contenuto corrente di una label

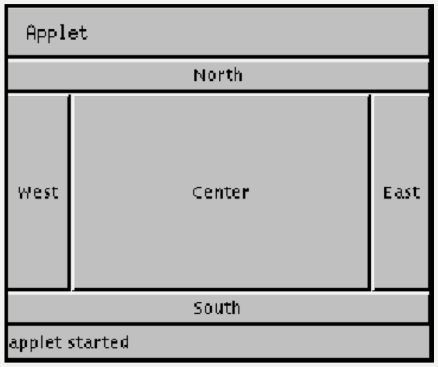
ESEMPIO DI CREAZIONE DI UN JFRAME

```
import javax.swing.*;

public class JFrame1 {
  public static void main(String[] args) {
    JFrame aFrame = new JFrame("First frame");
    aFrame.setSize(250, 100);
    aFrame.setDefaultCloseOperation(
        WindowConstants.EXIT_ON_CLOSE);
    JLabel label = new JLabel("Etichetta");
    aFrame.add(label);
    aFrame.setVisible(true);
  }
}
```

USARE UN LAYOUT MANAGER

- Se aggiungessimo un'altra JLabel all'esempio precedente la seconda verrebbe posizionata sopra la prima, oscurandola
- Per controllare il posizionamento dei componenti si usa un layout manager
- Il layout manager di default è BorderLayout, che divide il container in regioni. Se non viene specificata una regione in cui inserire i nuovi componenti essi vengono messi nella stessa regione, oscurandosi
- BorderLayout mappa il container nel seguente modo:



- As es. per inserire una JLabel a sinistra e una a destra:

```
aFrame.add(label1, BorderLayout.WEST);
aFrame.add(label2, BorderLayout.EAST);
```

FLOWLAYOUT

- Un altro layout manager disponibile (ne esistono molti) è FlowLayout, che dispone i componenti su righe: quando una riga si riempie, sposta i componenti nella successiva

- FlowLayout.LEFT, FlowLayout.CENTER (default) e FlowLayout.RIGHT sono tre costanti che impostano il modo di disporre i componenti sulle righe

- Esempio:
  FlowLayout flow = new FlowLayout(FlowLayout.CENTER);
  aFrame.setLayout(flow);
  …

- È opportuno, quando si usa un layout manager, prima di impostare a true la visibilità del frame, chiamare il metodo pack(), che si occupa di calcolare le dimensioni ottimali dei componenti e di disporli nel modo corretto (visivamente all'apertura della finestra tutto risulta già presente)

ESTENDERE LA CLASSE JFRAME

- Normalmente è utile estendere la classe di base JFrame per includere propri metodi, proprietà, ecc.

- Es.:

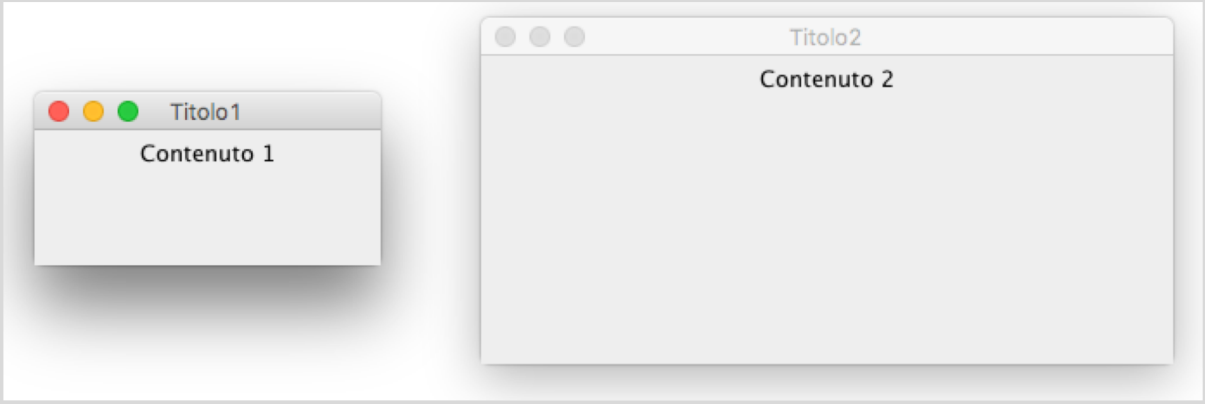
```
import javax.swing.*;
public class JMyFrame extends JFrame {
  final int WIDTH = 200;
  final int HEIGHT = 120;

  public JMyFrame() {
    super("My frame");
    setSize(WIDTH, HEIGHT);
    setVisible(true);
  }
}
```

ESERCIZIO

(JFrame1.java)

- Si crei una classe JFrame1 che eredita da JFrame e implementa un costruttore che riceve in input titolo, testo, larghezza, altezza
- Il testo dovrà essere mostrato in una JLabel centrata nella finestra
- JFrame1 dovrà terminare l'applicazione alla chiusura della finestra
- Si creino nel main 2 istanze di JFrame1 con parametri diversi
- Esempio di output:



JTEXTFIELD

- JTextField è il componente che rappresenta una casella di testo editabile

- Costruttori:
  - JTextField(): crea una casella di testo
  - JTextField(int columns): crea una casella di testo che può ospitare circa il numero specificato di caratteri
  - JTextField(String text): crea una casella di testo con il contenuto iniziale specificato
  - JTextField(String text, int columns): crea una casella di testo che può ospitare circa il numero specificato di caratteri con il contenuto iniziale specificato

- Metodi:
  - setText(String text): modifica il contenuto della casella di testo
  - String getText(): restituisce il contenuto della casella di testo
  - setEditable(boolean editable): se chiamato con false rende la casella di sola lettura

JBUTTON

- JButton rappresenta un pulsante

- Costruttori:
  - JButton(): crea un pulsante
  - JButton(String text): crea un pulsante con il testo specificato

- Metodi:
  - setText(String text): modifica il testo del pulsante
  - String getText(): restituisce il testo del pulsante

MODIFICARE IL FONT

- In tutti i componenti esiste un metodo per modificare il font usato per rappresentare i contenuti testuali:
  setFont(Font font)

- La classe Font ha il costruttore
  Font(String name, int style, int size)

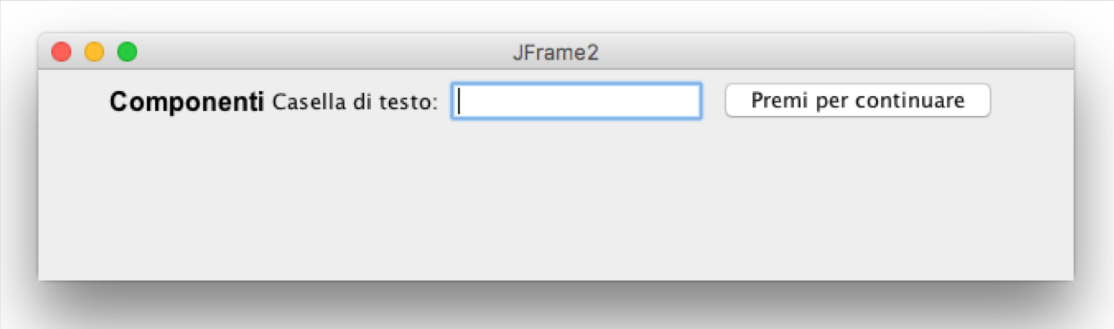
- name è il nome del tipo di font da utilizzare. Esso può essere sia il nome di un font fisico esistente (es. "Arial"), sia un font logico (es. "Monospaced", "Serif", o "SansSerif"). Se il font specificato non viene trovato sul sistema, viene effettuata una sostituzione
- style può essere una delle costanti Font.PLAIN, Font.BOLD, o Font.ITALIC
- size definisce la grandezza del font (il default è 13), che dipende da risoluzione, dpi, ecc.

ESERCIZIO

(JFrame2.java)

- Scrivere un software che crei una finestra come in figura

Originale



Ridimensionata

EVENTI

- Nelle slide precedenti è stato introdotto il componente JButton, la cui funzione più comune è rispondere al click con un'azione
- Il click su un componente è un esempio di *evento (event)* scatenato
- Il componente su cui si è verificato un evento viene chiamato *sorgente (source)*
- Un oggetto che si "interessa" degli eventi che avvengono viene detto *listener*
- Non tutti gli oggetti devono recepire eventi; per esempio: il click sull'area vuota di una finestra o su una label solitamente non viene gestito in nessun modo

INTERCETTARE EVENTI

- Per rispondere a un evento occorre:
  - Preparare la classe per accettare eventi
  - Dire alla classe quali eventi gestire
  - Dire alla classe come rispondere agli eventi quando accadono

INTERCETTARE EVENTI

- Preparare la classe per accettare eventi
  - Importare il package per gestire gli eventi java.awt.event.*;
  - Dichiarare che la classe implementarà l'interfaccia ActionListener
- Dire alla classe quali eventi gestire
  - Richiamare ad esempio il metodo addActionListener(ActionListener listener) su un pulsante di cui si vuole gestire il click
  - (Per ogni tipo di evento da intercettare si usa un metodo diverso)
- Dire alla classe come rispondere agli eventi quando accadono
  - Implementare il metodo actionPerformed(ActionEvent e) dell'interfaccia ActionListener

ESEMPIO

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

public class JFrame3 extends JFrame implements ActionListener {
  JLabel label;
  public JFrame3() {
    super("JFrame3");
    setSize(300, 100);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    label = new JLabel("Label da modificare");
    JButton button = new JButton("Modifica");
    setLayout(new FlowLayout());
    add(label);
    add(button);
    button.addActionListener(this);
  }

  public static void main(String[] args) {
    JFrame3 frame = new JFrame3();
    frame.setVisible(true);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    label.setText("Label modificata");
  }
}
```

GESTIRE PIÙ SORGENTI

- Nell'esempio precedente il listener è il JFrame, e lo è indipendentemente dai componenti che scatenano l'evento

- Ad es., se si aggiunge una casella di testo textField alla finestra e si invoca il metodo `textField.addActionListener(this)`, la classe risponderà anche all'evento di pressione di invio sulla casella di testo

- Se il gestore dell'evento deve compiere le stesse operazioni previste alla pressione del pulsante già presente non ci sono problemi, ma di solito occorre conoscere il componente che ha scatenato l'evento per intraprendere azioni diverse

GESTIRE PIÙ SORGENTI

- All'interno del metodo actionPerformed(ActionEvent e):
  - e.getSource() restituisce l'oggetto (Object) che ha scatenato l'evento

- Ci sono diversi modi per intraprendere azioni diverse:
  - `if (e.getSource() == component)`
    controlla se l'oggetto è precisamente il componente specificato
    (ad es.: `if (e.getSource() == textField)`)
  - `if (e.getSource() istanceof Component)`
    controlla se l'oggetto è un tipo di componente specificato
    (ad es.: `if (e.getSource() istanceof JButton)`)
  - Un altro modo è creare diversi listener per i diversi componenti su cui occorre intercettare gli eventi (vedi prossima slide)

- Nota: per disabilitare un componente si può utilizzare il metodo setEnabled(false); il componente disabilitato non scatenerà più gli eventi corrispondenti

LISTENER SPECIFICO PER COMPONENTE

- Invece di usare come listener il JFrame, si possono creare listener dedicati ai singoli eventi

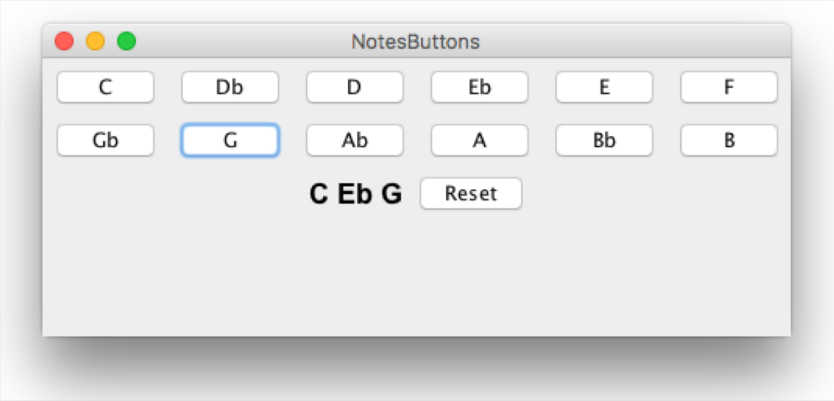
- Ad esempio, avendo un pulsante `button`:

```
button.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    …
  }
});
```

ESERCIZIO

(NotesButtons.java)

- Scrivere un software che crei una finestra come in figura



- Si utilizzi un'enumerazione per creare i pulsanti con i nomi delle note (cioè senza creare manualmente ogni pulsante singolo)

- La pressione di un pulsante aggiunge la nota corrispondente a un'etichetta di testo inizialmente vuota (nella figura l'etichetta contiene "C Eb G" dopo la pressione dei pulsanti corrispondenti)

- Il pulsante Reset cancella il contenuto dell'etichetta

ALTRI TIPI DI EVENTI

- Oltre agli eventi di tipo Action, che sono stati intercettati per gestire il click sui pulsanti, esistono altre tipologie di eventi, con diverse sorgenti e relativi listener

Listener	Componenti sorgente	Esempio
ActionListener	JButton, JCheckBox, JComboBox, JTextField, JRadioButton	Click su pulsante, invio in casella di testo, selezione di una checkbox
AdjustmentListener	JScrollBar	Riposizionamento di una barra di scorrimento
ChangeListener	JSlider, JCheckBox	Riposizionamento di uno slider, modifica di una checkbox
FocusListener	Tutti i componenti	Un componente riceve o perde il focus
ItemListener	JButton, JCheckBox, JComboBox, JRadioButton	Cambiamento di stato di una checkbox
KeyListener	Tutti i componenti	Inserimento di testo in una casella
MouseListener	Tutti i componenti	Click del mouse
MouseMotionListener	Tutti i componenti	Movimento del mouse
WindowListener	JWindow, JFrame	Chiusura della finestra

ALTRI TIPI DI EVENTI

- Così come visto in precedenza, ogni sorgente che può scatenare l'evento potrà dichiarare con il corrispondente metodo un listener
  - Ad es.: per gestire la modifica di stato di una checkbox:
    `checkbox.addItemListener(new ItemListener() { … });`

- Il listener dovrà implementare l'interfaccia relativa, con la creazione degli **event handler** corrispondenti
  - Ad es.: il corpo dell'ItemListener creato sopra dovrà includere:

```
@Override
public void itemStateChanged(ItemEvent e) {
  …
}
```

- Ogni interfaccia ha i propri metodi da implementare, anche più di uno: verranno visti più avanti

JCHECKBOX

- Il componente JCheckBox rappresenta la casella di spunta

- Costruttori:
  - JCheckBox()
  - JCheckBox(String text): crea una casella di spunta con il testo specificato
  - JCheckBox(String text, boolean selected): crea una casella di spunta con il testo specificato impostando o meno la spunta iniziale

- Metodi:
  - setText(String text): modifica il testo
  - String getText(): restituisce il testo
  - setSelected(boolean selected): imposta la spunta
  - boolean isSelected(): restituisce lo stato di selezione della spunta

JCHECKBOX – EVENTO DI SPUNTA

- Alla modifica della spunta di una checkbox viene generato l'evento ItemEvent. L'interfaccia ItemListener contiene il metodo seguente:

```
@Override
public void itemStateChanged(ItemEvent e) {
  Object source = e.getItem();
  if (source == checkBox) {
    int select = e.getStateChange();
    if(select == ItemEvent.SELECTED)
      // checkbox selezionata
    else
      // checkbox deselezionata
  }
}
```

JCOMBOBOX

- Il componente JComboBox rappresenta la casella con elenco a discesa

- Costruttori:
  - JComboBox(): crea una combo box vuota
  - JComboBox(Object[] items): crea una combo box con gli elementi specificati (il primo elemento è selezionato di default)
  - JComboBox<E>(): crea una combo box vuota che ospita elementi di classe E (generics)
  - JComboBox<E>(E[] items): crea una combo box vuota che ospita elementi di classe E con gli elementi specificati (il primo elemento è selezionato di default)

JCOMBOBOX

- Metodi (in caso di utilizzo di generics alcuni elementi qui specificati come Object sono della classe usata all'atto di creazione della combo box):
  - addItem(Object item): aggiunge un elemento alla lista
  - removeItem(Object item): rimuove dalla lista l'elemento specificato
  - removeAllItems(): svuota la lista
  - Object getItemAt(int index): restituisce l'elemento presente nella posizione specificata
  - int getItemCount(): restituisce il numero di elementi della lista
  - int getSelectedIndex(): restituisce la posizione dell'elemento correntemente selezionato
  - Object getSelectedItem(): restituisce l'elemento correntemente selezionato
  - setSelectedIndex(int index): seleziona l'elemento presente nella posizione specificata
  - setSelectedItem(Object item): seleziona l'elemento specificato
  - setEditable(boolean editable): se impostato a true consente di scrivere un valore diverso da quelli presenti in lista (attenzione: in questo caso alcuni metodi possono restituire valori non previsti: ad esempio getSelectedIndex() restituisce -1 se il testo digitato non è presente tra i valori della lista)

ESERCIZIO

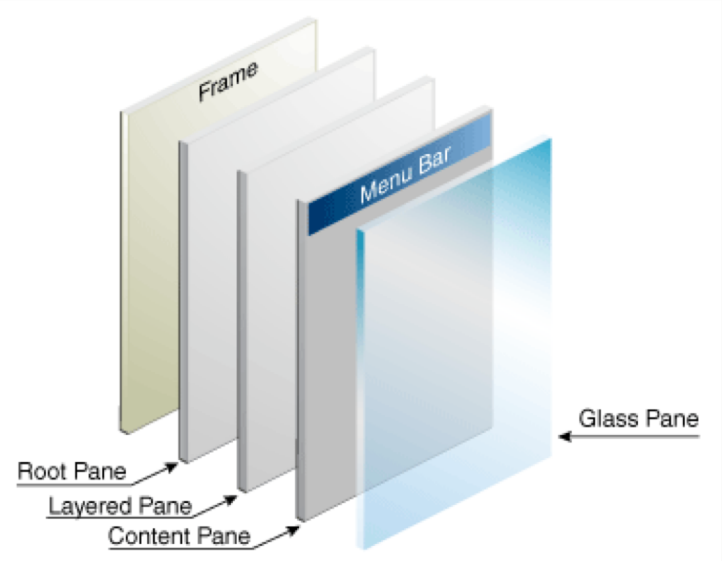
(NotesButtons2.java)

Modificare il software creato nell'esercizio precedente (NotesButtons.java) nel seguente modo:

- Aggiungere dopo il pulsante Reset una check box che abiliti/disabiliti il pulsante stesso
- Aggiungere una combobox che contenga la lista delle note selezionate con la pressione dei pulsanti, nell'ordine di selezione
- Aggiungere un pulsante (dopo la combo box) che rimuova dalla combobox – e di conseguenza dalla label contenente la lista di note inserite – la nota selezionata nella combobox

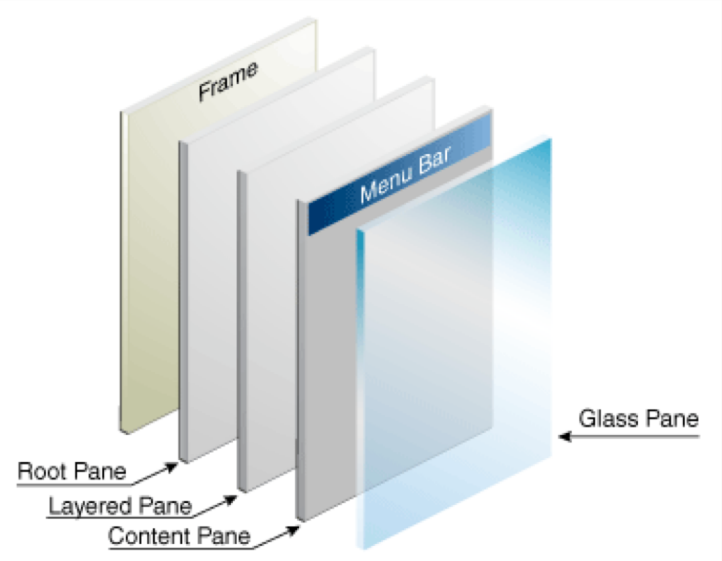
TOP-LEVEL CONTAINERS

- JFrame costituisce una finestra di un'applicazione. Finora sono stati aggiunti componenti ad essa utilizzando direttamente il metodo add(Component c)
- In realtà un JFrame, come gli altri *top-level containers*, è in realtà formato da un root pane, che ospita i livelli sottostanti:
  - Layered pane
  - Content pane
  - Glass pane



TOP-LEVEL CONTAINERS

- Root pane
  - Ospita la barra opzionale dei menu e il content pane
- Content pane
  - È in realtà il livello che ospita tutti i componenti visibili aggiunti al JFrame
  - Si recupera con il metodo di JFrame getContentPane()
  - Quando in precedenza è stato usato il metodo add in realtà il sistema ha tradotto in getContentPane().add
- Glass pane
  - È un livello solitamente invisibile che però può ospitare ad es. tool tips oppure può essere usato per disegnare al di sopra di tutti i controlli presenti nel content pane
- Layered pane
  - Solitamente non utilizzato in modo esplicito, ma solo internamente



COLORI

- È possibile modificare colori di primo piano e sfondo dei componenti usando la classe Color:
  - Occorre includere java.awt.Color
  - Esistono costanti già dichiarate per 13 colori (statiche della classe Color): BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW
  - In alternativa si può usare uno dei costruttori di Color:
    - Color(int r, int g, int b): con i valori RGB tra 0 e 255
    - Color(float r, float g, float b): con i valori RGB tra 0 e 1
- Per settare il colore di primo piano di un componente setForeground(Color c)
- Per settare il colore di sfondo di un componente setBackground(Color c)

RIEPILOGO LAYOUT MANAGER

- BorderLayout
  - Suddivisione in 5 parti dello spazio
  - All'aggiunta di un componente si può specificare la collocazione:
    - add(component, BorderLayout.NORTH)
    - add(component, BorderLayout.SOUTH)
    - add(component, BorderLayout.WEST)
    - add(component, BorderLayout.EAST)
    - add(component, BorderLayout.CENTER)
- FlowLayout
  - Suddivisione in righe dello spazio
  - Si imposta la collocazione dei componenti all'interno di ogni riga:
    - flowLayout.setAlignment(FlowLayout.LEFT)
    - flowLayout.setAlignment(FlowLayout.CENTER)
    - flowLayout.setAlignment(FlowLayout.RIGHT)

GRIDLAYOUT

- Nuovo layout manager: GridLayout
  - Suddivisione in righe e colonne dello spazio
  - Si imposta il numero di righe e colonne alla creazione del layout:
    - new GridLayout(int rows, int columns)
    - new GridLayout(int rows, int columns, int hgap, int vgap): imposta lo spazio orizzontale e verticale tra i componenti
  - L'inserimento dei componenti parte dalla casella in alto a sinistra e continua sulle altre colonne della stessa riga e poi sulle righe successive (non è possibile saltare una casella ma si possono inserire controlli vuoti)
  - Inserendo uno 0 come numero di righe o di colonne (non entrambi) si lascia al sistema il calcolo di quella dimensione

JPANEL

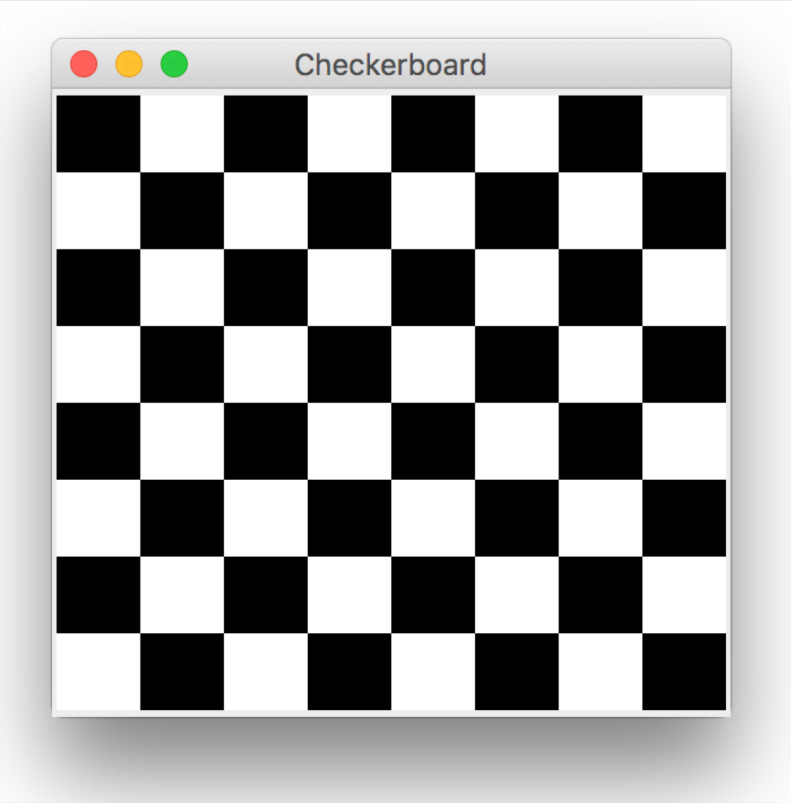
- Il componente rappresenta un pannello che può essere usato per disporre altri componenti al suo interno, così da ovviare alle limitazione di layout imposte dai manager visti
- Il layout manager di default per un JPanel è FlowLayout
- I componenti si possono aggiungere all'interno di un JPanel con il metodo add(Component c) già visto

- Costruttori
  - JPanel()
  - JPanel(LayoutManager layout): usa un layout manager diverso dal default

ESERCIZIO

(Checkerboard.java)

- Scrivere un software che crei una finestra rappresentante una scacchiera 8x8 con caselle bianche e nere come in figura



JSCROLLPANE

- Per utilizzare uno spazio maggiore rispetto a quello visibile si può usare il componente JScrollPane, che consente di controllare con barre di scroll le aree invisibili
- Può avere barre di scroll orizzontale e/o verticali
- Costruttori:
  - JScrollPane(): crea un nuovo JScrollPane con barre orizzontali e verticali visibili solo se necessario
  - JScrollPane(int vsb, int hsb): imposta i parametri per la visualizzazione delle barre orizzontali e verticali
  - JScrollPane(Component c): visualizza il componente specificato
  - JScrollPane(Component c, int vsb, int hsb): visualizza il componente specificato impostando i parametri per la visualizzazione delle barre orizzontali e verticali

JSCROLLPANE - PARAMETRI

- I parametri che controllano la visualizzazione delle barre di scorrimento orizzontale e verticale possono assumere i valori (dal significato intuitivo):
  - ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED
  - ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS
  - ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER
  - ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
  - ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS
  - ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER

ESEMPIO (JScrollDemo.java)

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

public class JScrollDemo extends JFrame {
  JPanel panel = new JPanel();
  JScrollPane scroll = new JScrollPane(panel,
      ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
      ScrollPaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
  JLabel label = new JLabel("Visualizzazione scrollbar");

  public JScrollDemo() {
    super("JScrollDemo");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    label.setFont(new Font(null, Font.PLAIN, 40));
    getContentPane.add(scroll);
    panel.add(label);
  }

  public static void main(String[] args) {
    final int WIDTH = 180;
    final int HEIGHT = 100;
    JScrollDemo aFrame = new JScrollDemo();
    aFrame.setSize(WIDTH, HEIGHT);
    aFrame.setVisible(true);
  }
}
```

TABELLA RIASSUNTIVA HANDLER

Event	Listener(s)	Handler(s)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
TextEvent	TextListener	textValueChanged(TextEvent)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
ComponentEvent	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
FocusEvent	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
MouseEvent	MouseListener MouseMotionListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent) mouseDragged(MouseEvent) mouseMoved(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent) keyTyped(KeyEvent) keyReleased(KeyEvent)
WindowEvent	WindowListener	windowActivated(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)
MouseWheelEvent	MouseWheelListener	mouseWheelMoved(MouseWheelEvent)

ADAPTER CLASS

- Quando si desidera implementare una classe listener si devono implementare tutti i metodi dell'interfaccia
- Per comodità, in caso di listener con più di un metodo da implementare, si possono usare le classi **adapter**
  - sono astratte
  - forniscono un'implementazione vuota delle corrispondenti interfacce
  - in questo modo occorre fare l'override dei soli metodi a cui si è interessati
- Es.:

```
class MyMouseAdapter extends MouseAdapter {
  @Override
  public void mousePressed(MouseEvent e) {
    ...
  }
}
… .addListener(new MyMouseAdapter());
```

ESEMPIO (JDemoKey.java)

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

public class JDemoKey extends JFrame {
  private JLabel prompt = new JLabel("Scrivere qualcosa:");
  private JLabel outputLabel = new JLabel();
  private JTextField textField = new JTextField(10);

  public JDemoKey() {
    setTitle("KeyTyped");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
    add(prompt, BorderLayout.NORTH);
    add(textField, BorderLayout.CENTER);
    add(outputLabel, BorderLayout.SOUTH);
    MyKeyAdapter adapter = new MyKeyAdapter();
    textField.addKeyListener(adapter);
  }

  public static void main(String[] args) {
    JDemoKey keyFrame = new JDemoKey();
    keyFrame.setSize(250, 100);
    keyFrame.setVisible(true);
  }

  class MyKeyAdapter extends KeyAdapter {
    @Override
    public void keyTyped(KeyEvent e) {
      char c = e.getKeyChar();
      outputLabel.setText("Ultimo tasto premuto: " + c);
    }
  }
}
```

EVENTI DEL MOUSE

- L'interfaccia MouseMotionListener contiene i metodi che vengono chiamati al movimento del mouse
  - void mouseDragged(MouseEvent e)
  - void mouseMoved(MouseEvent e)
- L'interfaccia MouseListener contiene i metodi che vengono chiamati al click del mouse (primi 3) e all'ingresso/uscita del puntatore su un componente (ultimi 2)
  - void mouseClicked(MouseEvent e)
  - void mousePressed(MouseEvent e)
  - void mouseReleased(MouseEvent e)
  - void mouseEntered(MouseEvent e)
  - void mouseExited(MouseEvent e)

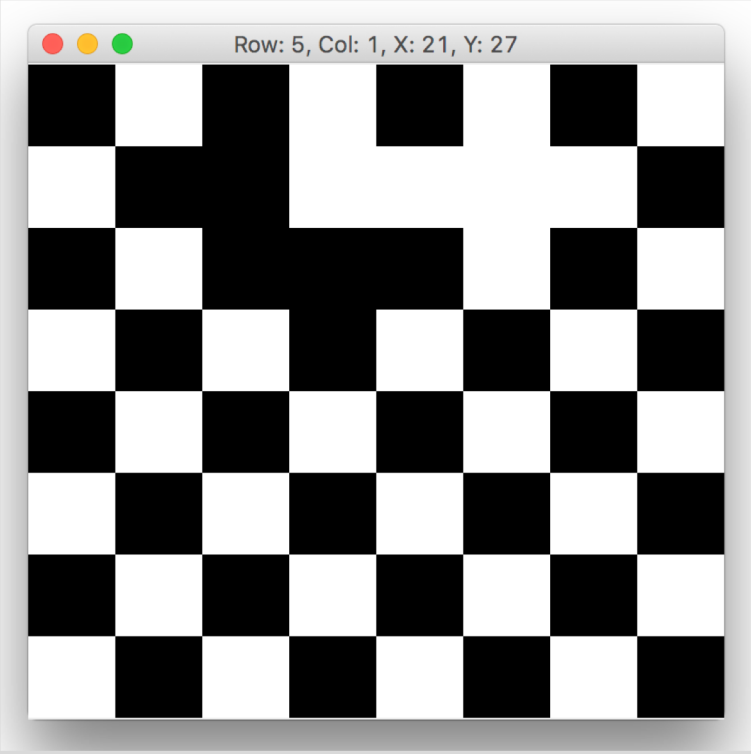
MOUSEEVENT

- Metodi della classe MouseEvent:
  - int getButton(): tasto premuto, valori:
    - java.awt.event.MouseEvent.NOBUTTON
    - java.awt.event.MouseEvent.BUTTON1 (tasto sx)
    - java.awt.event.MouseEvent.BUTTON2 (tasto centrale)
    - java.awt.event.MouseEvent.BUTTON3 (tasto dx)
  - int getClickCount(): numero di clic effettuati (singolo, doppio clic…)
  - int getX(), int getY(): coordinate del puntatore rispetto al componente
  - int getXOnScreen(), int getYOnScreen(): coordinate del puntatore rispetto allo schermo

ESERCIZIO

(CheckerboardMouse.java)

- Modificare l'esercizio CheckerBoard.java:
  - Al muoversi del mouse sulla scacchiera il titolo della finestra riporta i valori di riga e colonna della casella corrente e le coordinate del mouse rispetto ad essa (es.: "Row: 5, Col: 1, X: 21, Y: 27")
  - Al clic del mouse su una casella questa cambia colore, alternando tra bianco e nero



GRAFICA

- Ogni volta che un componente deve essere (ri)disegnato, viene invocato automaticamente dal sistema il metodo
  `public void paint(Graphics g)`

- L'oggetto passato di classe Graphics (java.awt.Graphics) può essere usato per controllare manualmente le operazioni di disegno di un componente, facendo l'override del metodo paint di un componente

- Quando si effettua l'override del metodo paint, il parametro Graphics è già impostato con le caratteristiche del componente stesso (il colore di primo piano/sfondo, l'area di disegno, ecc.)

- Il metodo paint solitamente non viene chiamato direttamente: se si ha necessità di forzare il ridisegno di un componente si richiama repaint()

- Attenzione: richiamando il repaint() di un container esso viene ridisegnato completamente, includendo il ridisegno di tutti i componenti inclusi

SETLOCATION

- I componenti inseriti in un container sono normalmente posizionati automaticamente dal layout manager adottato

- Per posizionare invece un certo componente alle coordinate (in pixel) x, y si può invocare il suo metodo setLocation(int x, int y)

- Esiste anche il metodo Point getLocation(), che restituisce la posizione corrente (Point è una classe con attributi x e y)

- Attenzione: setLocation deve essere richiamato quando il layout manager ha completato la collocazione dei componenti, altrimenti esso effettua il riposizionamento automatico (ad es. all'interno del costruttore di un container in cui sono aggiunti i componenti)

PAINT VS PAINTCOMPONENT

- Nei componenti che non siano Top Level Containers (JFrame, JDialog, JWindow…) è opportuno effettuare l'overload del metodo `public void paintComponent(Graphics g)`, che evita di occuparsi di aspetti a basso livello

- Differenza tra paint e paintComponent in caso di JPanel: in caso di componenti contenuti nel JPanel `paintComponent` disegna *sotto* i componenti, `paint` *sopra*

GETGRAPHICS

- Se occorre disegnare al di fuori del metodo paint è possibile invocare il metodo `Graphics getGraphics()` di un componente

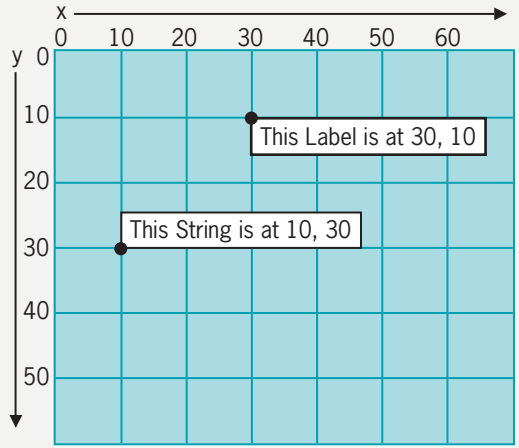
- Es. (al clic su un pulsante si disegna su di esso):

```
public void actionPerformed(ActionEvent e) {
  Graphics draw = getGraphics();
  draw.drawString("You clicked the button!", 50, 100);
}
```

- È in realtà sempre consigliabile effettuare l'override del metodo paintComponent, perché il ridisegno automatico effettuato dal sistema in certe circostanze può annullare il disegno effettuato con getGraphics. Ad es., se il pulsante dell'esempio precedente viene ridimensionato al ridimensionamento della finestra, la scritta disegnata scompare)

METODI DI GRAPHICS: DRAWSTRING

- Per disegnare una stringa di testo su un componente si usa il metodo `drawString(String s, int x, int y)`
- C'è differenza rispetto all'uso di setLocation (ad es. di una JLabel) e le coordinate di drawString:
  - In setLocation le coordinate dell'etichetta di testo si riferiscono all'angolo in alto a sinistra della stessa
  - In drawString le coordinate si riferiscono all'angolo in basso a sinistra



SETFONT E SETCOLOR

- Per controllare le caratteristiche delle stringhe disegnate con drawString si possono usare i metodi (di Graphics)
  - setFont(Font f)
  - setColor(Color c)

- Il metodo setColor influisce anche sui metodi esposti più avanti: dalla impostazione del colore in poi tutti i metodi di disegno useranno quel colore

ALTRI METODI PER IL DISEGNO

- Disegno di una linea:
  `drawLine(int x1, int y1, int x2, int y2)`

- Disegno di un rettangolo (solo bordo):
  `drawRect(int x, int y, int width, int height)`
- Disegno di un rettangolo (rettangolo pieno):
  `fillRect(int x, int y, int width, int height)`
- Disegno di un rettangolo (rettangolo pieno con colore di sfondo del componente):
  `clearRect(int x, int y, int width, int height)`

- Disegno di un rettangolo con angoli arrotondati (solo bordo):
  `drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`
- Disegno di un rettangolo con angoli arrotondati (rettangolo pieno):
  `fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`

ALTRI METODI PER IL DISEGNO

- Disegno di un ovale (solo bordo):
  `drawOval(int x1, int y1, int width, int height)`
- Disegno di un ovale (ovale pieno):
  `fillOval(int x, int y, int width, int height)`

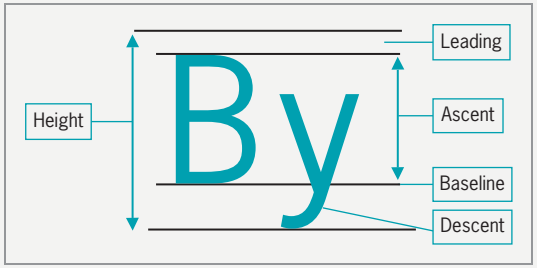
- Disegno di un arco (solo bordo):
  `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
- Disegno di un arco (arco pieno):
  `fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`

- Nel disegno degli archi, gli angoli sono espressi in gradi sessagesimali sulla circonferenza trigonometrica

INFORMAZIONI SUL FONT

- Spesso è utile ottenere informazioni sul font in uso per poter disegnare correttamente stringhe di testo

- Il metodo (di Graphics) getFontMetrics() restituisce un oggetto FontMetrics, che espone i metodi seguenti:
  - int getLeading()
  - int getAscent()
  - int getDescent()
  - int getHeight()



  - int stringWidth(String s): restituisce la larghezza in pixel della stringa passata

GRAPHICS2D

- La classe Graphics2D (java.awt.Graphics2D) offre alcune migliorie rispetto a Graphics, tra cui:
  - Impostazione del tratto
  - Antialiasing

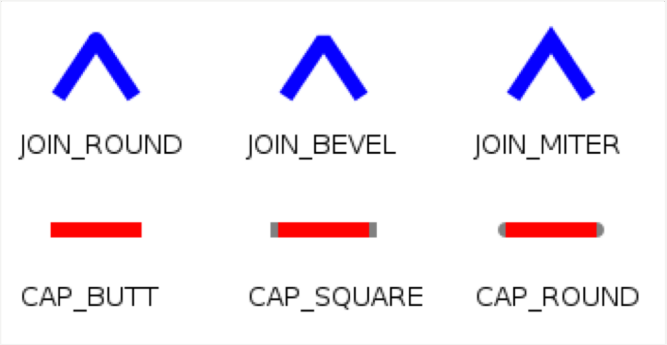
- Si può castare direttamente l'oggetto Graphics del metodo paint, paintComponent e getGraphics a Graphics2D:

```
Graphics2D g2d = (Graphics2D)g;
```

IMPOSTAZIONE DEL TRATTO

- Il metodo di Graphics2D setStroke(Stroke s) permette di selezionare il tipo di linea utilizzato per le operazioni di disegno successive

- Stroke è un'interfaccia, la classe che la implementa è BasicStroke, i cui costruttori sono:
  - BasicStroke()
  - BasicStroke(float width): specifica lo spessore del tratto
  - BasicStroke(float width, int cap, int join): specifica spessore, tipo di terminazione della linea e tipo di giunzione tra linee. Le costanti corrispondenti sono:
    - BasicStroke.CAP_BUTT
    - BasicStroke.CAP_SQUARE
    - BasicStroke.CAP_ROUND
    - BasicStroke.JOIN_ROUND
    - BasicStroke.JOIN_BEVEL
    - BasicStroke.JOIN_MITER



ANTIALIASING

- Per controllare la qualità del disegno di Graphics2D si può usare un oggetto di classe RenderingHints

- È possibile specificare diversi "suggerimenti" (poiché non tutte le piattaforme potrebbero supportare tutte le modalità, quindi si tratta di "suggerire" quello che si vorrebbe ottenere)

- Per impostare un generico antialiasing:

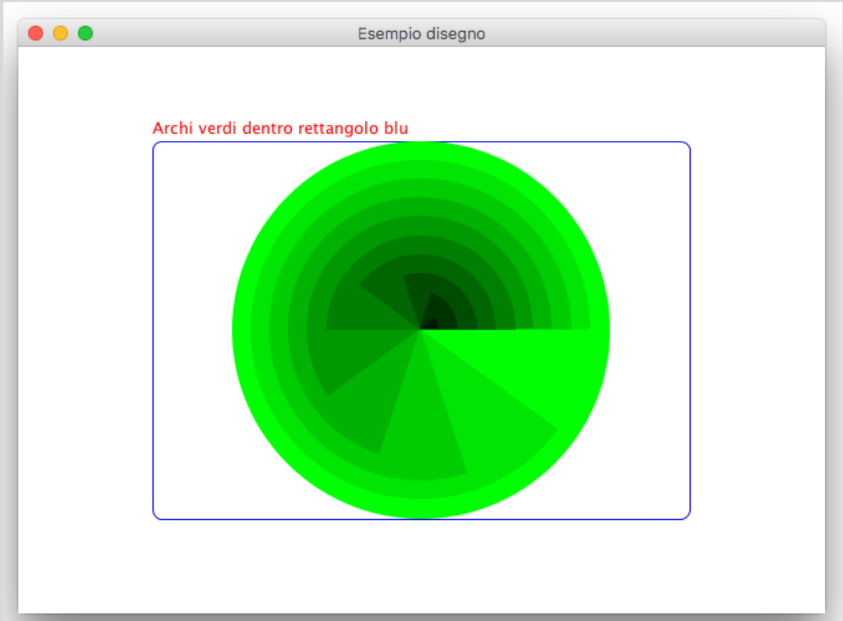
```
((Graphics2D)g).setRenderingHints(
    new RenderingHints(
         RenderingHints.KEY_ANTIALIASING,
          RenderingHints.VALUE_ANTIALIAS_ON))
```

ESERCIZIO

(Graphics1.java)

Creare una finestra come in figura:

- Il rettangolo (Color.BLUE) è centrato e occupa 2/3 della finestra, sia in orizzontale che in verticale (la dimensione del JFrame si ottiene con il metodo getSize())
- La scritta (Color.RED) è in alto a sinistra del rettangolo
- All'interno vengono disegnati 10 archi pieni:
  - Colore variabile tra RGB 0,0,0 e RGB 0,255,0
  - Dimensione variabile tra 0 e minima dimensione tra altezza e larghezza del rettangolo
  - Angolo variabile tra 0 e 360 gradi

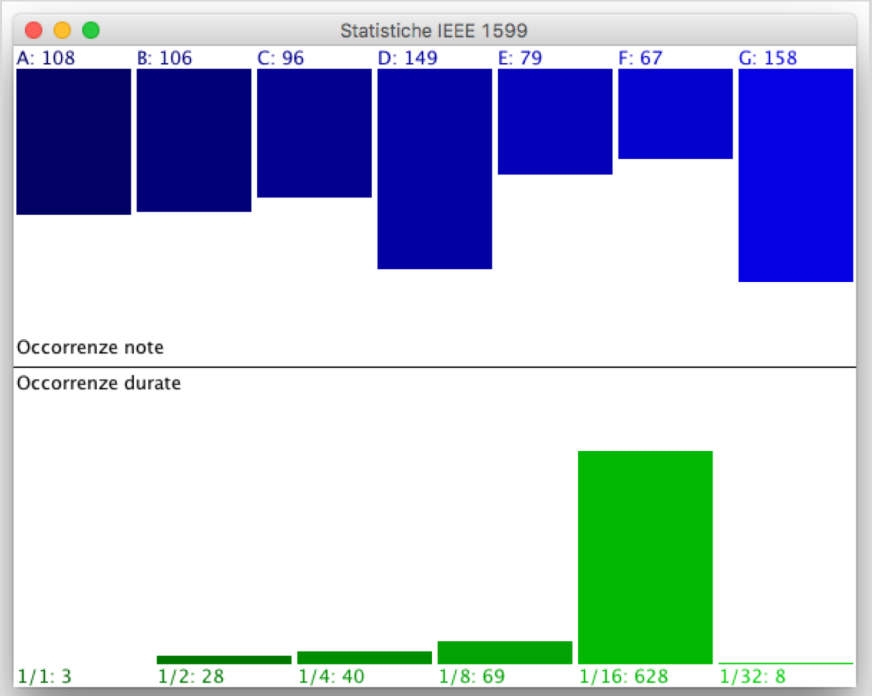


ESERCIZIO

(XmlStatsGraphics.java)

Riprendendo spunto dall'esercizio XmlStats, in cui venivano calcolati per un file IEEE 1599 le statistiche relative a numero di note di una certa altezza (a meno delle ottave) e numero di valori ritmici (di note e pause), disegnare due grafici a barre all'interno della finestra principale:

- Il grafico che occupa circa il terzo superiore è relativo alle altezze
- Il grafico che occupa circa il terzo inferiore è relativo alle durate



THREAD IN SWING

- Java supporta la programmazione multi-thread, e Swing offre una specifica classe per gestire le operazioni da svolgersi in background

- È importante che in un'applicazione che presenta una GUI non ci siano momenti di "blocco", ma che il software sia sempre pronto a rispondere alle interazioni dell'utente

- Per svolgere compiti gravosi si possono usare istanze di javax.swing.SwingWorker

EVENT DISPATCH THREAD

- Finora le finestre Swing sono state create negli esempi/esercizi direttamente nel main
- In realtà in Swing vengono coinvolte varie tipologie di thread, tra cui quello che contiene il main che viene chiamato *thread iniziale*
- Il thread che si occupa di gestire eventi e interazioni con l'utente è invece l'*event dispatch thread (EDT)*
- Per creare la finestra principale MyJFrame è opportuno usare:

```
java.awt.EventQueue.invokeLater(new Runnable() {
  @Override
  public void run() {
    new MyJFrame();
  }
});
```

- In questo modo la finestra viene gestita dall'EDT: anche se nella maggior parte dei casi la creazione diretta nel main funziona correttamente, alcuni componenti di Swing non sono thread-safe e quindi potrebbero generare problemi se creati e gestiti nel thread iniziale

SWINGWORKER

- SwingWorker è una classe astratta utilizzabile per operazioni in background: occorre definire una propria implementazione

- Caratteristiche salienti della classe:
  - Offre un metodo che viene invocato al completamento del task
  - Si può restituire un valore al termine
  - Si può interrompere l'esecuzione
  - Possono essere passati dal task in esecuzione dei valori intermedi

SWINGWORKER: ESEMPIO

```
SwingWorker worker = new SwingWorker<T, V>() {
  @Override
  public T doInBackground() {
    ...
  }
};
worker.execute();

(T result = worker.get())
```

- All'interno del metodo doInBackground vengono effettuate le operazioni del task
- La prima parte definisce lo SwingWorker, l'ultima riga lo esegue e passa subito alla riga successiva (se si desidera attendere la fine dell'esecuzione del task si può chiamare il metodo get())
- Il thread da cui si invoca l'execute è il *thread corrente*, che spesso corrisponde anche all'*Event Dispatch Thread*, cioè al thread che riceve le notifiche sull'esecuzione del worker

SWINGWORKER: APPROFONDIMENTI

- SwingWorker è una classe generic: i due parametri della dichiarazione/istanziazione sono:
  - T: il tipo di dato restituito al termine del task
  - V: il tipo di dato usato dal worker per informare sulla sua evoluzione durante l'esecuzione del task

- Il thread del worker può informare sull'esecuzione del task in diversi modi:
  - Usando il metodo setProgress(int progress): viene impostato un valore tra 0 e 100 indicante lo stato di evoluzione
  - Usando il metodo done(): viene richiamato quando il task è concluso
  - Usando il metodo publish(V… chunks)

SWINGWORKER: APPROFONDIMENTI

```
public class MyWorker extends SwingWorker<Integer, String> {
  @Override
  protected Integer doInBackground() throws Exception {
    // Start
    publish("Start");
    setProgress(1);

    // More work was done
    publish("More work was done");
    setProgress(10);

    // Complete
    publish("Complete");
    setProgress(100);
    return 1;
  }

  @Override protected void process(List<String> chunks) {
    // Messaggi ricevuti da doInBackground()
    // (all'invocazione del metodo publish())
  }
}
```

SWINGWORKER: APPROFONDIMENTI

- Il metodo publish potrebbe essere invocato troppo spesso all'interno di doInBackground, appesantendo il lavoro dell'EDT

- Per questo motivo, mentre publish viene invocato dal thread del worker, process viene invece invocato dall'EDT, e recupera la lista dei valori passati dai micro-task generati da publish accorpandoli nei parametri in ingresso

- Nel metodo process possono essere quindi aggiornate strutture dati esterne perché il thread è l'EDT

- Es.:
  `publish("a"); publish("b", "c"); publish("d");`
  potrebbe risultare in una sola chiamata
  `process("a", "b", "c", "d");`

SWINGWORKER: APPROFONDIMENTI

- Recuperare dall'esterno dello SwingWorker informazioni sul suo stato:

```
searchWorker.addPropertyChangeListener(new
PropertyChangeListener() {
  @Override
  public void propertyChange(PropertyChangeEvent event) {
    switch (event.getPropertyName()) {
      case "progress":
        // Valore del progress: (Integer)event.getNewValue()
        break;
      case "state":
        switch ((StateValue) event.getNewValue()) {
          case PENDING:  // Stato iniziale
            break;
          case STARTED: // Prima di eseguire doInBackground
            break;
          case DONE: // Task concluso
            break;
        }
        break;
    }
  }
});
```

SWINGWORKER: APPROFONDIMENTI

- Un buon metodo per essere certi di non compromettere le strutture dati non thread-safe è inserire nel blocco di gestione dello stato DONE la chiamata a get():

```
case DONE: // Task concluso
  try {
    Object o = worker.get();
    ...
  } catch (InterruptedException ex) {
    // Task interrotto
  } catch (ExecutionException ex) {
    // Errore nell'esecuzione del task
  }
  break;
```

SWINGWORKER: CANCELLARE UN TASK

- Per cancellare l'esecuzione di un task si può invocare il metodo `boolean cancel(boolean interrupt)`
  che, con parametro true, blocca il worker. Se per qualche motivo l'interruzione non può avere luogo (includendo anche il caso in cui sia già terminata normalmente), restituisce false

- Il metodo `boolean isDone()` di SwingWorker restituisce true se il task è stato completato (anche se è stato bloccato)

- Il metodo `boolean isCancelled()` di SwingWorker restituisce true se `cancel()` ha restituito true

SWINGWORKER: IL METODO DONE

- Con l'override del metodo
  `void done()`
  all'interno dello SwingWorker è possibile eseguire delle operazioni alla conclusione del doInBackground

- Perché non eseguire le istruzioni alla fine del corpo di doInBackground invece che in un metodo a parte?
  Perché done() viene richiamato dall'EDT, quindi non ci sono problemi di conflitti per strutture non thread-safe

ESEMPIO

(CountThread.java)

- Nell'esempio viene creata una finestra con una JTextArea (una casella di testo multilinea) e un pulsante
- Alla pressione del pulsante viene avviato uno SwingWorker che inizia a contare da 1 e ogni secondo incrementa il conteggio, aggiornando l'area di testo
- Note:
  - Viene implementato un PropertyChangeListener per intercettare l'avvio e la conclusione del task di conteggio
  - Viene aggiornata l'area di testo nel corpo dell'implementazione del metodo process di SwingWorker
  - Si noti cosa accade se nell'istruzione Thread.sleep si modifica il valore passato a 10 (l'incremento del contatore avviene ogni decimo di secondo): la chiamata a process accoda i passati conteggi aggiornati dalle chiamate di publish

ESERCIZIO

(XmlStatsGraphicsThread.java)

- Riprendendo spunto dall'esercizio XmlStatsGraphics, si modifichi il codice per far calcolare in background i grafici risultanti
- Si usi un FlowLayout per disporre un pannello con i grafici e un pulsante che avvii il calcolo dei valori
- All'avvio dell'applicazione il pannello mostra una stringa che invita a cliccare per effettuare il calcolo delle statistiche
- All'avvio del calcolo il pannello mostra una stringa "Calcolo in corso…"
- Al termine del normale calcolo vengono mostrati sul pannello i grafici
- Se l'utente blocca il calcolo vengono mostrati i grafici vuoti
- Al termine o al blocco del calcolo il pulsante viene disabilitato

JOPTIONPANE

- Per creare finestre di dialogo elementari modali che visualizzano un messaggio o chiedono all'utente un semplice input si può usare la classe JOptionPane

- Quasi sempre si usano i metodi statici seguenti:
  - showMessageDialog: dialogo con un messaggio informativo
  - showConfirmDialog: dialogo di conferma, con pulsanti come sì/no/annulla
  - showInputDialog: richiesta di input (mostra una combobox con i valori selezionabili)

JOPTIONPANE - SHOWMESSAGEDIALOG

(I pedici indicano i parametri presenti nelle varie implementazioni)

void showMessageDialog(
    $_{1,2,3}$Component parentComponent,
    $_{1,2,3}$Object message,
    $_{2,3}$String title,
    $_{2,3}$int messageType,
    $_{3}$Icon icon)

- parentComponent: frame nel quale il dialogo è presente (se null viene usato un frame di default)
- message: l'oggetto da visualizzare
- title: il titolo del dialogo
- messageType: un valore utilizzato per impostare l'icona di sistema legata al tipo di messaggio:
  - ERROR_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE
- icon: l'icona da mostrare (se specificata, viene usata al posto di quella del precedente parametro)

JOPTIONPANE - SHOWCONFIRMDIALOG

(I pedici indicano i parametri presenti nelle varie implementazioni)

int showConfirmDialog(
$_{1,2,3,4}$Component parentComponent,
$_{1,2,3,4}$Object message,
$_{2,3,4}$String title,
$_{2,3,4}$int optionType,
$_{3,4}$int messageType,
$_{4}$Icon icon)

- Valore restituito, che indica la selezione del pulsante corrispondente:
  - YES_OPTION
  - NO_OPTION
  - CANCEL_OPTION
  - OK_OPTION
  - CLOSED_OPTION (la finestra è stata chiusa senza selezionare nulla)
- parentComponent: frame nel quale il dialogo è presente (se null viene usato un frame di default)
- message: l'oggetto da visualizzare
- title: il titolo del dialogo
- optionType: un valore che indica i pulsanti presenti:
  - YES_NO_OPTION
  - YES_NO_CANCEL_OPTION
  - OK_CANCEL_OPTION
- messageType: un valore utilizzato per impostare l'icona di sistema legata al tipo di messaggio:
  - ERROR_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE
- icon: l'icona da mostrare (se specificata, viene usata al posto di quella del precedente parametro)

JOPTIONPANE - SHOWINPUTDIALOG

(Si veda la documentazione per le varie implementazioni)

```
String showInputDialog(
        Component parentComponent,
        Object message,
        String title,
        int messageType,
        Icon icon,
        Object[] selectionValues,
        Object initialSelectionValue)
```

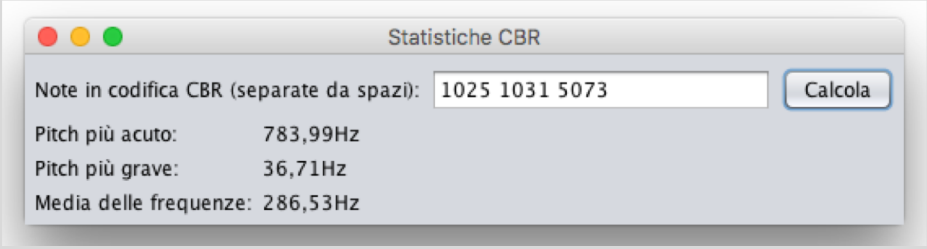
- Valore restituito: il valore selezionato dall'utente (null se l'utente ha annullato la selezione)
- parentComponent: frame nel quale il dialogo è presente (se null viene usato un frame di default)
- message: l'oggetto da visualizzare
- title: il titolo del dialogo
- messageType: un valore utilizzato per impostare l'icona di sistema legata al tipo di messaggio:
  - ERROR_MESSAGE
  - INFORMATION_MESSAGE
  - WARNING_MESSAGE
  - QUESTION_MESSAGE
  - PLAIN_MESSAGE
- icon: l'icona da mostrare (se specificata, viene usata al posto di quella del precedente parametro)
- selectionValues: i valori selezionabili come output
- initialSelectionValue: il valore selezionato all'avvio

ESERCIZIO

(CbrStatsSwing.java)

Riprendendo spunto dall'esercizio CbrStats, che calcolava – da una lista di valori CBR passati come parametri in ingresso – il pitch più alto, quello più basso e il pitch medio:

- Si crei un'applicazione Swing come in figura
- I parametri in ingresso sono presenti in un JTextField, separati da spazi
- I risultati vengono mostrati, alla pressione di un pulsante, in 3 JLabel
- I messaggi di errore originariamente mostrati su console vengono invece mostrati usando la classe JOptionPane



JFILECHOOSER

- La classe JFileChooser mostra la classica finestra di dialogo in cui è possibile scegliere in input/output un file o una directory

- Costruttori:
  - JFileChooser(): crea un dialog che si apre sulla directory di default dell'utente
  - JFileChooser(String currentDirectoryPath): crea un dialog che si apre sulla directory passata come parametro
  - JFileChooser(File currentDirectory): crea un dialog che si apre sulla directory passata come parametro

JFILECHOOSER – METODI

- File getSelectedFile(): restituisce il file selezionato
- File[] getSelectedFiles(): restituisce i file selezionati (se è abilitata la selezione multipla)
- setDialogTitle(String title): imposta il titolo della finestra di dialogo
- setFileFilter(FileFilter filter): imposta una visualizzazione dei file filtrata per estensione. Ad esempio:
  ```
  setFileFilter(new FileNameExtensionFilter("Immagini JPG & GIF", "jpg", "gif"));
  ```
  (il primo parametro è la descrizione della tipologia, i successivi le varie estensioni da filtrare)
- setFileSelectionMode(int mode): imposta il tipo di selezione:
  - JFileChooser.FILES_ONLY
  - JFileChooser.DIRECTORIES_ONLY
  - JFileChooser.FILES_AND_DIRECTORIES
- setMultiSelectionEnabled(boolean b): abilita o meno la selezione multipla

JFILECHOOSER – METODI

- int showOpenDialog(Component parent): apre una finestra per l'apertura di uno o più file, legandola al parent
- int showSaveDialog(Component parent): apre una finestra per il salvataggio di un file, legandola al parent
- int showDialog(Component parent, String approveButtonText): apre una finestra generica di selezione, impostando il testo del pulsante di conferma uguale al parametro passato

- I tre metodi precedenti possono restituire:
  - JFileChooser.CANCEL_OPTION: se l'azione è annullata
  - JFileChooser.APPROVE_OPTION: se l'azione è confermata
  - JFileChooser.ERROR_OPTION: se si è verificato un errore o la finestra è stata chiusa

JFILECHOOSER – ESEMPIO

Esempio di apertura di un file di tipo JPG o GIF

```
JFileChooser chooser = new JFileChooser();
FileNameExtensionFilter filter = new FileNameExtensionFilter(
    ”Immagini JPG & GIF", "jpg", "gif");
chooser.setFileFilter(filter);
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
  System.out.println("File aperto: "
      + chooser.getSelectedFile().getName());
}
```

JLIST

- Il componente JList rappresenta un elenco di contenuti (simile a JComboBox)

- Costruttori:
  - JList(): crea una lista vuota
  - JList(Object[] items): crea una lista con gli elementi specificati
  - JList<E>(): crea una lista vuota che ospita elementi di classe E (generics)
  - JList<E>(E[] items): crea una lista vuota che ospita elementi di classe E con gli elementi specificati

JLIST

- Per impostare il tipo di selezione supportata:
  - setSelectionMode(int selectionMode). Valori supportati:
    - ListSelectionModel.SINGLE_SELECTION: è possibile selezionare un solo elemento
    - ListSelectionModel.SINGLE_INTERVAL_SELECTION: è possibile selezionare solo elementi contigui
    - ListSelectionModel.MULTIPLE_INTERVAL_SELECTION: è possibile la seleziona multipla
- Gli elementi presenti in lista vengono memorizzati in un *modello* di contenuto: il più semplice è il DefaultListModel, che implementa l'interfaccia Vector, ma si può implementare l'interfaccia ListModel per modelli personalizzati
  - setModel(ListModel model): imposta il modello della lista
  - ListModel getModel(): restituisce il modello (se è stato impostato in precedenza è possibile effettuare un cast sul modello usato)

JLIST

- Esempio di creazione di una lista di stringhe:

```
JList<String> jList1 = new JList();
DefaultListModel<String> listModel = new DefaultListModel();
listModel.addElement("Jane Doe");
listModel.addElement("John Smith");
listModel.addElement("Kathy Green");
jList1.setModel(listModel);
...
DefaultListModel<String> listModelNew =
(DefaultListModel<String>)jList1.getModel();
```

DEFAULTLISTMODEL

- Costruttore: DefaultListModel()
- Metodi:
  - addElement(E element): accoda un elemento
  - add(int index, E element): inserisce un elemento nella posizione specificata
  - clear(): rimuove tutti gli elementi
  - E get(int index): restituisce l'elemento nella posizione specificata
  - int indexOf(Object element): restituisce l'indice dell'elemento specificato, -1 se non esiste
  - E remove(int index): rimuove l'elemento nella posizione specificata, restituendolo
  - E set(int index, E element): sostituisce l'elemento presente nella posizione specificata con l'elemento passato (restituisce il vecchio elemento)
  - int size(): restituisce il numero di elementi della lista

JLIST

- Metodi:
  - int getSelectedIndex(): restituisce la posizione dell'elemento correntemente selezionato (il primo in caso di selezione multipla)
  - int[] getSelectedIndices(): restituisce le posizioni degli elementi correntemente selezionati
  - E getSelectedValue(): restituisce l'elemento correntemente selezionato (il primo in caso di selezione multipla)
  - List<E> getSelectedValuesList(): restituisce la lista degli elementi correntemente selezionati (in ordine di indice)
  - boolean isSelectedIndex(int index): restituisce true se l'indice passato è selezionato
  - setSelectedIndex(int index): seleziona l'elemento nella posizione specificata
  - setSelectedIndices(int[] indices): seleziona gli elementi nelle posizioni specificate
  - setSelectedValue(Object item, boolean shouldScroll): seleziona l'elemento specificato, se esiste; il secondo parametro specifica se la lista deve scorrere fino a mostrare l'elemento
  - ensureIndexIsVisible(int index): scorre la visualizzazione della lista per rendere visibile l'elemento presente nella posizione specificata

JLIST: EVENTO DI SELEZIONE

- Per intercettare la modifica della selezione all'interno della lista si può implementare il gestore dell'evento ListSelectionEvent:

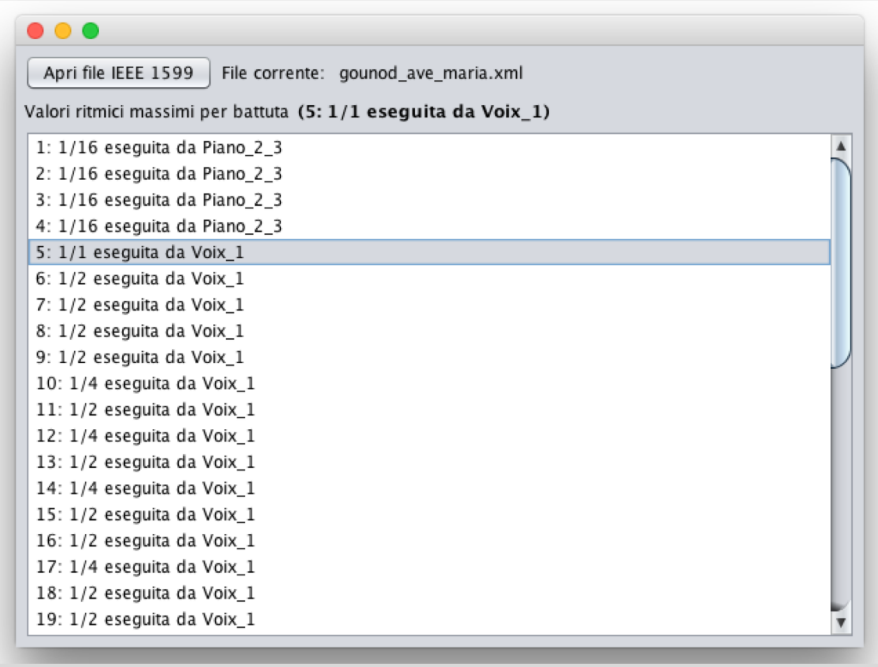
```
jList.addListSelectionListener(new ListSelectionListener() {
  @Override
  public void valueChanged(ListSelectionEvent e) {
    jLabel.setText(jListStats.getSelectedValue()); // ad es.
  }
});
```

ESERCIZIO

(XPathAnalysisSwing.java)

Riprendendo spunto dall'esercizio XPathAnalysis, che per ogni battuta presente in un file IEEE 1599 calcolava la nota più lunga presente:

- Si crei un'applicazione Swing come in figura
- Il pulsante consente di scegliere un file IEEE 1599 da aprire
- All'apertura del file:
  - viene aggiornata la label che indica il file corrente
  - vengono disabilitati: il pulsante e la lista
  - viene fatto partire uno SwingWorker che effettua i calcoli e aggiorna il contenuto della lista
- Al termine del calcolo:
  - vengono riabilitati tutti i componenti
  - la selezione di una riga della lista copia il valore selezionato nella label mostrata in grassetto



TIMER

- Per impostare nel futuro l'esecuzione di una certa operazione è possibile usare la classe java.util.Timer
- Ogni oggetto Timer ha associato un thread in background
- È possibile eseguire un task dopo un certo lasso di tempo o impostare la sua esecuzione a intervalli regolari
- I task che vengono schedulati dovrebbero venire completati velocemente: in caso contrario, potrebbe crearsi una *coda* di operazioni che aspettano la fine dell'esecuzione del task corrente
- La classe Timer è thread-safe: diversi thread possono lavorare sullo stesso oggetto Timer senza problemi
- La classe è scalabile: la schedulazione di migliaia di task non dovrebbe presentare problemi
- Costruttore più comune: Timer()

TIMERTASK

- Per specificare qual è l'operazione da eseguire attraverso l'uso di un timer si utilizza la classe astratta java.util.TimerTask
- Deve essere implementato il metodo
  `public abstract void run()`
- All'interno del corpo dell'implementazione viene specificato il task da eseguire

- Esempi:

```
public class MyTask extends TimerTask {
  @Override
  public void run(){
    System.out.println("Task eseguito");
  }
}

Timer timer = new Timer();
timer.scheduleAtFixedRate(new TimerTask() { // new MyTask()
  @Override
  public void run() {
    System.out.println("Task eseguito");
  }
}, 5000, 5000);
```

TIMER – METODI FIXED-DELAY

- void schedule(TimerTask task, long delay): esegue un task dopo *delay* millisecondi
- void schedule(TimerTask task, Date time): esegue un task ad un certo istante di tempo (se *time* è nel passato, esegue immediatamente)
- void schedule(TimerTask task, long delay, long period): esegue un task dopo *delay* millisecondi, e esecuzioni successive all'incirca ogni *period* millisecondi
- void schedule(TimerTask task, Date time, long period): esegue un task ad un certo istante di tempo (se *time* è nel passato, esegue immediatamente), e esecuzioni successive all'incirca ogni *period* millisecondi

- Nei metodi elencati l'esecuzione è detta a *fixed-delay*. Ogni esecuzione successiva è schedulata a partire dalla precedente. Se – per qualsiasi motivo – il task precedente viene eseguito con un certo ritardo, le successive esecuzioni vengono posticipate di conseguenza. Questo tipo di esecuzione è indicata quando è più importante mantenere costante l'intervallo di tempo tra i task, più che il numero di esecuzioni in un certo lasso grande di tempo (ad es.: il lampeggiamento del cursore)

TIMER – METODI FIXED-RATE

- void scheduleAtFixedRate(TimerTask task, long delay, long period): esegue un task dopo *delay* millisecondi, e esecuzioni successive all'incirca ogni *period* millisecondi
- void scheduleAtFixedRate(TimerTask task, Date time, long period): esegue un task ad un certo istante di tempo (se *time* è nel passato, esegue immediatamente), e esecuzioni successive all'incirca ogni *period* millisecondi

- Nei metodi elencati l'esecuzione è detta a *fixed-rate*. Ogni esecuzione successiva è schedulata a partire dalla prima esecuzione. Se – per qualsiasi motivo – il task precedente viene eseguito con un certo ritardo, le successive esecuzioni possono accadere in rapida successione per "recuperare il ritardo". Questo tipo di esecuzione è indicata quando è più importante il tempo assoluto di svolgimento dei task (ad es.: un segnale acustico che suona ogni minuto)

TIMER – CANCELLAZIONE

- Con il metodo `cancel()` è possibile terminare l'esecuzione del/i task programmati
- Il metodo termina l'esecuzione del timer, e quindi tutti i task in programma nel futuro non saranno eseguiti
- L'esecuzione del task corrente, se è attivo alla chiamata di cancel(), continua fino alla sua naturale conclusione
- Se cancel() viene chiamato all'interno del metodo run di un task creato dal timer, si è certi che l'attuale esecuzione sarà l'ultima eseguita dal timer
- Dopo la chiamata di cancel(), il timer non può essere usato per altri task

ESERCIZIO

(IEEE1599Timer.java)

Si crei un'applicazione che consente di aprire un file IEEE 1599 e:

- legge i valori dell'attributo *timing* degli elementi *event* dello spine (*ieee1599/logic/spine*) e li interpreta come millisecondi tra l'evento precedente e quello attuale
- mostri la visualizzazione temporizzata degli eventi che accadono alla pressione di un pulsante play