# Scorrendo la gerarchia

- Component
  - “A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface”
  - Component è quindi una classe molto generale e definisce un sacco di metodi
- Container
  - Un Component con la proprietà di essere abilitato a contenere altri Component
    - Prevede metodi di tipo “add” per aggiungere componenti
  - Prevede anche metodi per rappresentare il contenitore e i suoi contenuti e per aggiornarne l’immagine

# Scorrendo la gerarchia

- Window
  - È un particolare contenitore che può apparire sullo schermo come entità propria, ma non ha bordi, barre e controlli
  - Possiede metodi per mostrare la finestra, nascondere la finestra, posizionare la finestra e aggiustare l'ordine di comparizione relativamente ad altre finestre
- Frame
  - Si tratta di Window con bordi e barra del titolo, oltre alle caratteristiche solite di un'interfaccia (minimizzazione, iconizzazione, chiusura, resizing)
- JFrame
  - È un Frame AWT a cui SWING aggiunge una serie di metodi per ridefinire i dettagli grafici. Ad esempio
    - Il metodo setSize(int l, int h) permette di determinare le dimensioni del Frame
    - Il metodo setLocation(int x, int y) consente di definire le coordinate del pixel in alto a sinistra del frame nel riferimento dello schermo

# Cosa possiamo fare con un JFrame?

- Non possiamo disegnare, scrivere o aggiungere elementi direttamente al Frame
- Gli elementi diversi dal Menu debbono essere "aggiunti" ad un Container opportuno

- Ad esempio
  - Per scrivere del testo dentro un Frame, il testo dovrà essere scritto (aggiunto) al pannello contentPane del JFrame
  - Per convenienza il metodo add() di JFrame si occupa di tutto il lavoro

    frame.getContentPane().add(label) = frame.add(label)

# HelloWorldSwing


```
import javax.swing.*;

public class HelloWorldSwing {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("HelloWorldSwing");
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        createAndShowGUI();
    }
}
```

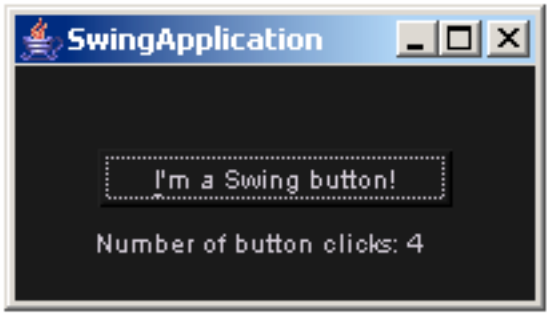
The pack method sizes the frame so that all its contents are at or above their preferred sizes. An alternative to pack is to establish a frame size explicitly by calling setSize or setBounds.

# HelloWorldSwing (seconda versione)

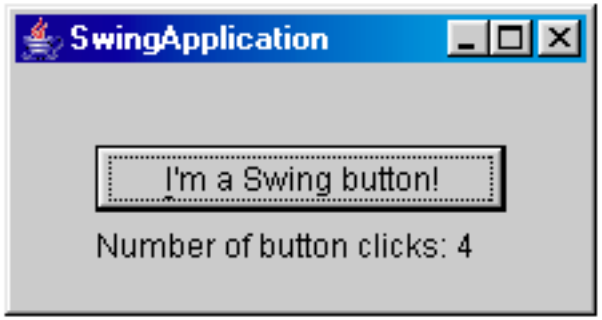

```
import javax.swing.*;

public class HelloWorldSwing {
    private static void createAndShowGUI() {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        createAndShowGUI();
    }
}
```

# Look&feel

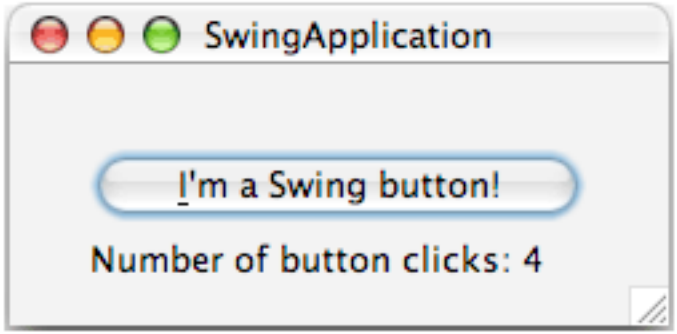

Java look and feel




GTK+ look and feel




Windows look and feel



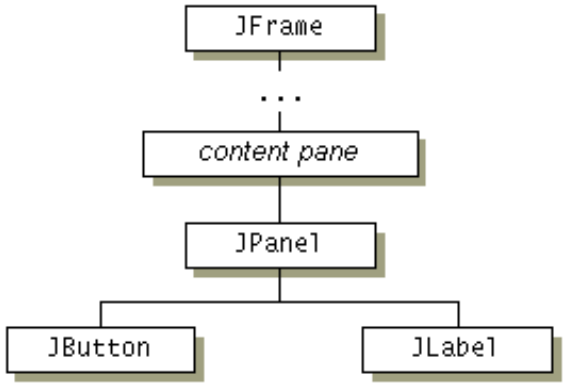

Mac OS look and feel

# Look&feel nativo

```
class WindowUtilities {
        public static void setNativeLookAndFeel() {
                try {
                        String lookAndFeel =
"com.sun.java.swing.plaf.motif.MotifLookAndFeel";
                        UIManager.setLookAndFeel(lookAndFeel);
                } catch (Exception e) {
                        System.out.println("Problema Look and feel nativo" + e);
                }
        }
}
```

# Aggiungiamo componenti

- Dobbiamo sapere:
  - Da dove prendere i componenti?
    - Una lista/descrizione delle librerie disponibili
  - Come costruirli e personalizzarli?
    - Costruttori e modificatori
  - Come usarli?
    - Quali eventi sono in grado di raccogliere e quali i listener necessari
  - Come disporli sui frame che costituiscono la nostra applicazione?
    - Layout manager



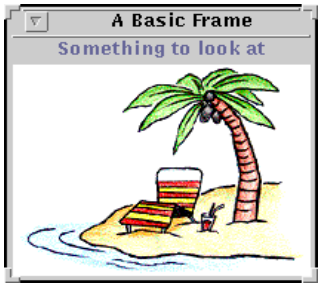
# Container

- Top-level container

**JApplet**

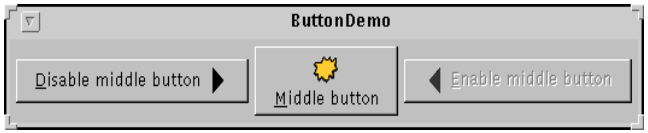

**JDialog**


**JFrame**



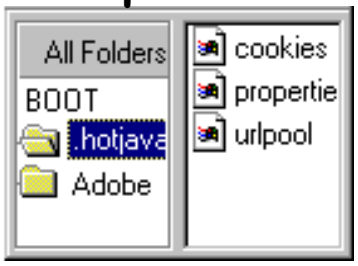

- General-purpose container

**JPanel**

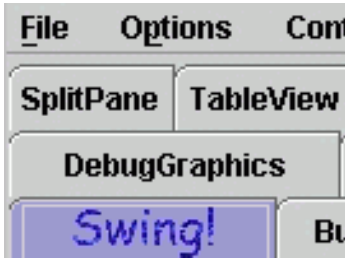



**JScrollPane**


**JSplitPane**



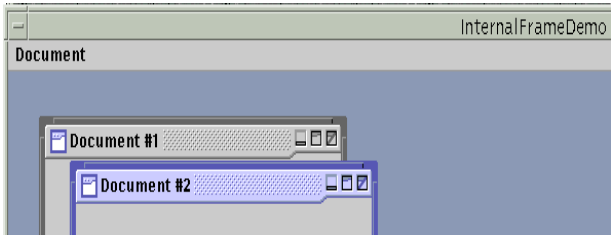

**JTabbedPane**



# Container

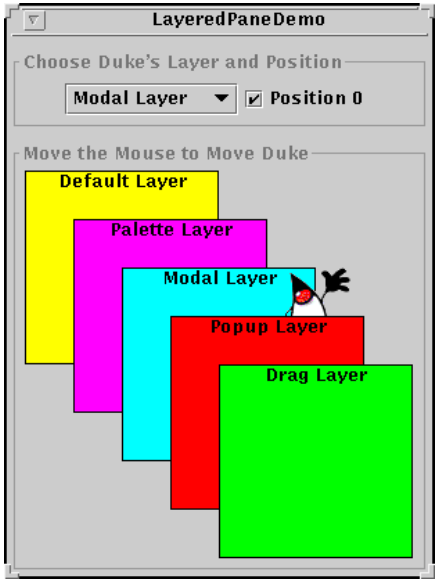
- Special-purpose container

## JInternalFrame




Permette di inserire frame dentro altri frame

## JLayeredPane



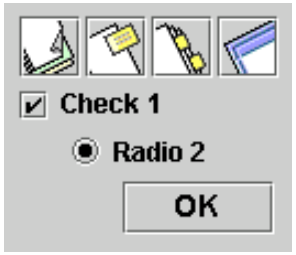

Permette di inserire componenti a vari livelli di profondità

## JToolBar

Permette di semplificare l'attivazione di determinate funzioni per mezzo di semplici pulsanti

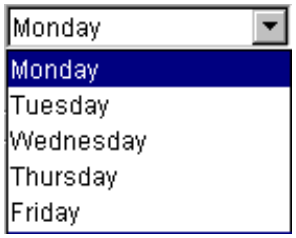
# Controlli di base

**JButtons**

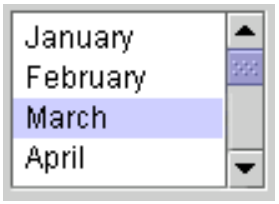



**Include buttons, radioButtons, checkbox, MenuItem, ToggleButton**

**JComboBox**

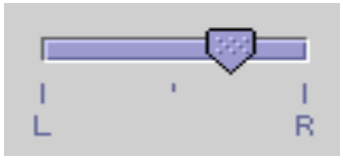



**JList**




**JMenu**


**JSlider**

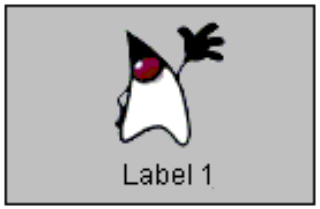



**JTextField**


**Include JPasswordField, JTextArea**

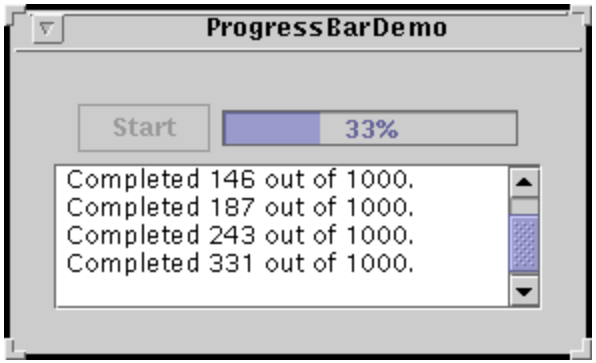
# Visualizzatori di informazioni non editabili

**JLabel**




**Può includere immagini e/o testo**

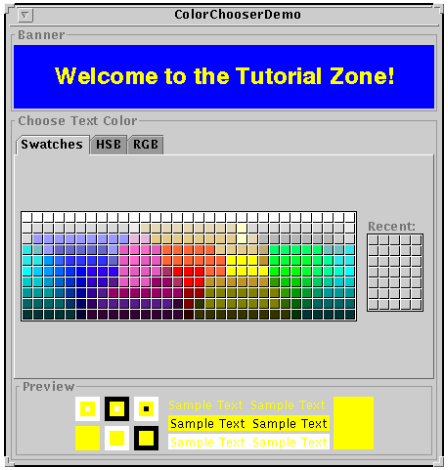
**JProgressBar**



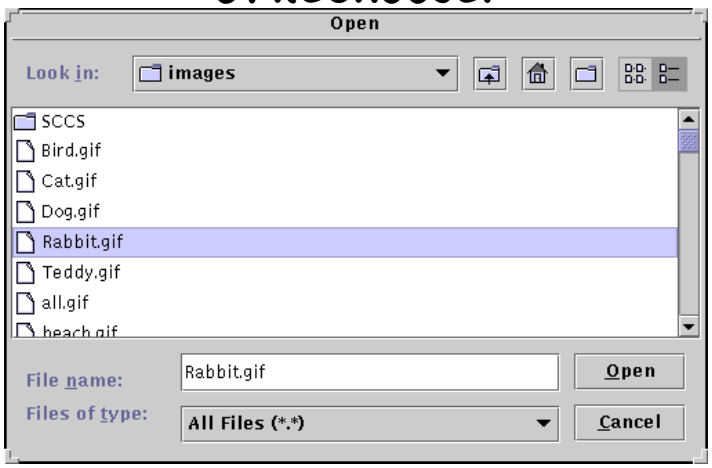

**Jcomponent.setToolTipText(String)**

# Visualizzatori di informazioni formattate editabili

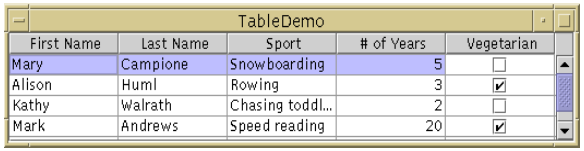
**JColorChooser**



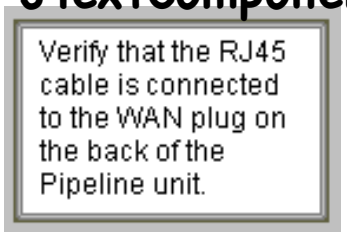

**JFileChooser**




**JTable**

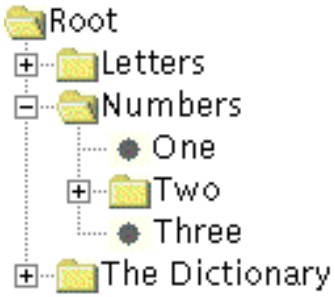


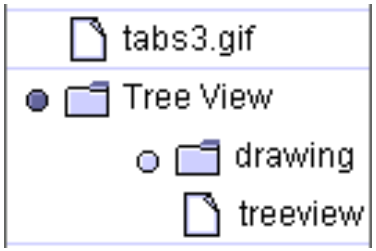

**JTextComponent**




**JTextField, JPasswordField, JTextArea, JEditorPane, JTextPane**

**JTree**



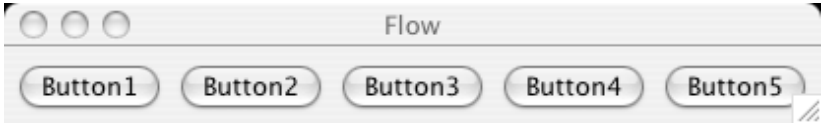
# Altri componenti grafici

- Label
- PushButton
- CheckBoxButton
- ScrollBar
- PullDownMenu
- PopupMenu
- …

# Layout

- Java gestisce la disposizione dei componenti dentro i Container mediante oggetti che si chiamano LayoutManager
  - Incapsulano gli algoritmi per il posizionamento delle componenti di una GUI
  - Il LayoutManager mantiene l'algoritmo separato in una classe a parte
- È un'interfaccia che descrive come un componente deve comunicare con il suo LayoutManager
- Esiste un'ampia collezione di LayoutManager, ma se si vuole si può creare il proprio
  - Noi vediamo solo i LayoutManager più comuni: FlowLayout, BorderLayout, GridLayout, CardLayout e GridBagLayout
- L'associazione avviene tramite il metodo setLayout() di cui è provvista la classe Container (e quindi tutte le sue sottoclassi)
  - p.setLayout(new BorderLayout());

# FlowLayout




```
private static void createAndShowGUI() {
    JFrame frame = new JFrame("Flow");
    frame.setLayout(new FlowLayout());
    frame.add(new JButton("Button1"));
    frame.add(new JButton("Button2"));
    frame.add(new JButton("Button3"));
    frame.add(new JButton("Button4"));
    frame.add(new JButton("Button5"));
    frame.pack()
    frame.setVisible(true);
}
```

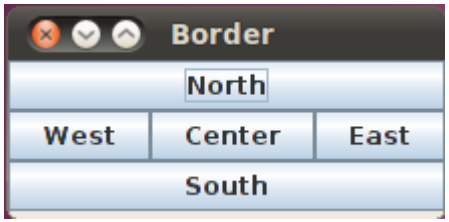
E' il più semplice. La sua strategia è:

- Rispettare la dimensione di tutti i componenti
- Disporre i componenti in orizzontale finché non viene riempita tutta una riga, altrimenti iniziare su una nuova riga
- Se non c'è spazio i componenti non vengono visualizzati

# BorderLayout

- Definisce 5 aree logiche: NORTH, SOUTH, CENTER, EAST e WEST
- Richiede la dimensione preferita del componente (altezza e larghezza)
- Se il componente è NORTH o SOUTH setta l'altezza al valore preferito e la larghezza in modo da occupare tutto lo spazio orizzontale
- Se il componente è EAST o WEST setta la larghezza al valore preferito e l'altezza in modo da occupare tutto lo spazio verticale restante
- Se il componente è CENTER setta l'altezza e la larghezza in modo da occupare tutto lo spazio centrale restante
- Quindi
  - Le posizioni NORTH e SOUTH servono quando vogliamo fissare l'altezza di un componente al valore preferito
  - Le posizioni EAST e WEST servono quando vogliamo fissare la larghezza di un componente al valore preferito
  - La parte CENTER è quella che si espande

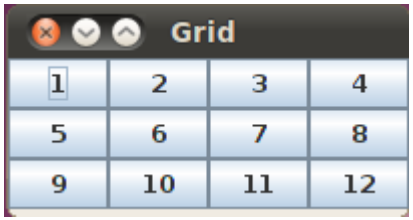
# Esempio




```
private static void createAndShowGUI() {
        JFrame frame = new JFrame("Border");
        frame.setLayout(new BorderLayout());
        frame.add(new JButton("North"), BorderLayout.NORTH);

        frame.add(new JButton("South"), BorderLayout.SOUTH);
        frame.add(new JButton("Center"), BorderLayout.CENTER);

        frame.add(new JButton("East"), BorderLayout.EAST);
        frame.add(new JButton("West"), BorderLayout.WEST);


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

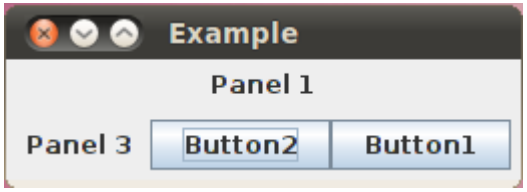
# GridLayout

- Dispone i componenti su una griglia




```
private static void createAndShowGUI() {
        JFrame frame = new JFrame("Grid");
        frame.setLayout(new GridLayout(3,4));

        for (int x=1; x<13; x++)
                frame.add(new JButton(""+x));


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

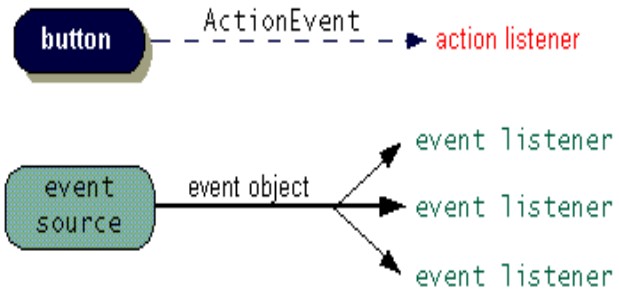
# Stratificazione




```
private static void createAndShowGUI() {
    JFrame f = new JFrame("Example");
    JPanel p1 = new JPanel();
    JPanel p2 = new JPanel();
    JPanel p3 = new JPanel();
    JPanel p4 = new JPanel();


    f.setLayout(new BorderLayout());
    p2.setLayout(new FlowLayout());
    p4.setLayout(new BorderLayout());

    p1.add(new JLabel("Panel 1"));
    p4.add(new JButton("Button1"), BorderLayout.EAST);
    p4.add(new JButton("Button2"), BorderLayout.WEST);
    p2.add(p3);
    p2.add(p4);
    p3.add(new JLabel("Panel 3"));
    f.add(p1, BorderLayout.NORTH);
    f.add(p2, BorderLayout.SOUTH);

    f.pack();
    f.setVisible(true);
}
```

# Eventi

- L'interazione tra gli elementi dell'interfaccia e la logica applicativa è gestita tramite eventi
- Gli EventListener sono interfacce definite per catturare e processare tipi di eventi particolari
- Un listener deve
  - Essere associato al componente
  - Essere informato quando il componente genera un evento del tipo richiesto
  - Rispondere facendo qualcosa di appropriato



# EventHandler

- Devono avere tre pezzi di codice
  - Dichiarazione
    - Estendono o implementano listener esistenti
    - public class MyClass implements ActionListener {
  - Associazione tra handler (ovvero listener) e istanza
    - someComponent.addActionListener(instanceOfMyClass);
  - Definizione del codice che implementa i metodi dell'interfaccia listener
    - public void actionPerformed(ActionEvent e) { ...

Demo 1 (vedi allegati)

# Eventi e Listener

| **Categoria** | **Evento** | **handler** |
|---|---|---|
| Mouse | MouseEvent | MouseListener, MouseMotionListener |
| Keyboard | KeyEvent | KeyListener |
| Selezione elemento | ItemEvent | ItemListener |
| Input di testo | TextEvent | TextListener |
| Scrollbar | AdjustmentEvent | AdjustmentListener |
| Bottoni, menu,... | ActionEvent | ActionListener |
| Cambiamenti nella finestra | WindowEvent | WindowListener |
| Focus | FocusEvent | FocusListener |

# GUI Swing - *multithreading*

- Quando si scrive una GUI Swing si assiste a un cambio di paradigma
  - Programma con flusso centralizzato -> programma reattivo basato su eventi

- Per questo motivo le Swing adottano un'architettura multi-thread
- Ciò permette di ottenere interfacce sempre reattive e che non si "bloccano"
  - Esempio: se ho un client di posta elettronica e qualcuno mi spedisce un messaggio di 10M voglio comunque potere: vedere l'intestazione della mail, vedere una barra che mi indica quanto manca al download, scrivere una nuova mail nel frattempo, ecc.

# 3 tipi di thread - *thread iniziale*

- Il thread iniziale instanzia la GUI e prosegue con le sue operazioni

```
import javax.swing.*;

public class HelloWorldSwing {
     private static void createAndShowGUI() {
          JFrame frame = new JFrame("HelloWorldSwing");
          JLabel label = new JLabel("Hello World");
          frame.getContentPane().add(label);
          frame.pack();
          frame.setVisible(true);
   }

   public static void main(String[] args) {
          createAndShowGUI();

          // …posso andare avanti a fare altro…
   }
}
```

# 3 tipi di thread - Event Dispatch Thread (EDT)

- Tutto il codice per la gestione dell'interfaccia grafica viene eseguito dentro l'EDT
- Esiste una coda di eventi di sistema che raccoglie i click del mouse, la pressione sui tasti della tastiera, ecc.
- L'EDT controlla ciclicamente la coda
  - Raccoglie un evento e decide cosa farne
  - Se è un click su un bottone chiama il metodo per la gestione del click del mouse su quel bottone
- La gestione mediante coda garantisce che venga rispettato un certo ordine nella gestione degli eventi

# Immaginiamo cosa sarebbe successo se…

- …non avessimo avuto l'EDT

- Ipotizziamo di avere due tasti: A e B
  - Tasto A
    - Legge il contenuto di un file e lo presenta all'utente all'interno di una textBox
  - Tasto B
    - Legge dallo stesso textBox e scrive i contenuti su un file
- Cosa succede se premo molto velocemente prima A e poi B?
  - Probabilmente avrei un comportamento anomalo!!!

- E' per questo che tutto ciò che lavora sull'interfaccia gira all'interno di un unico thread

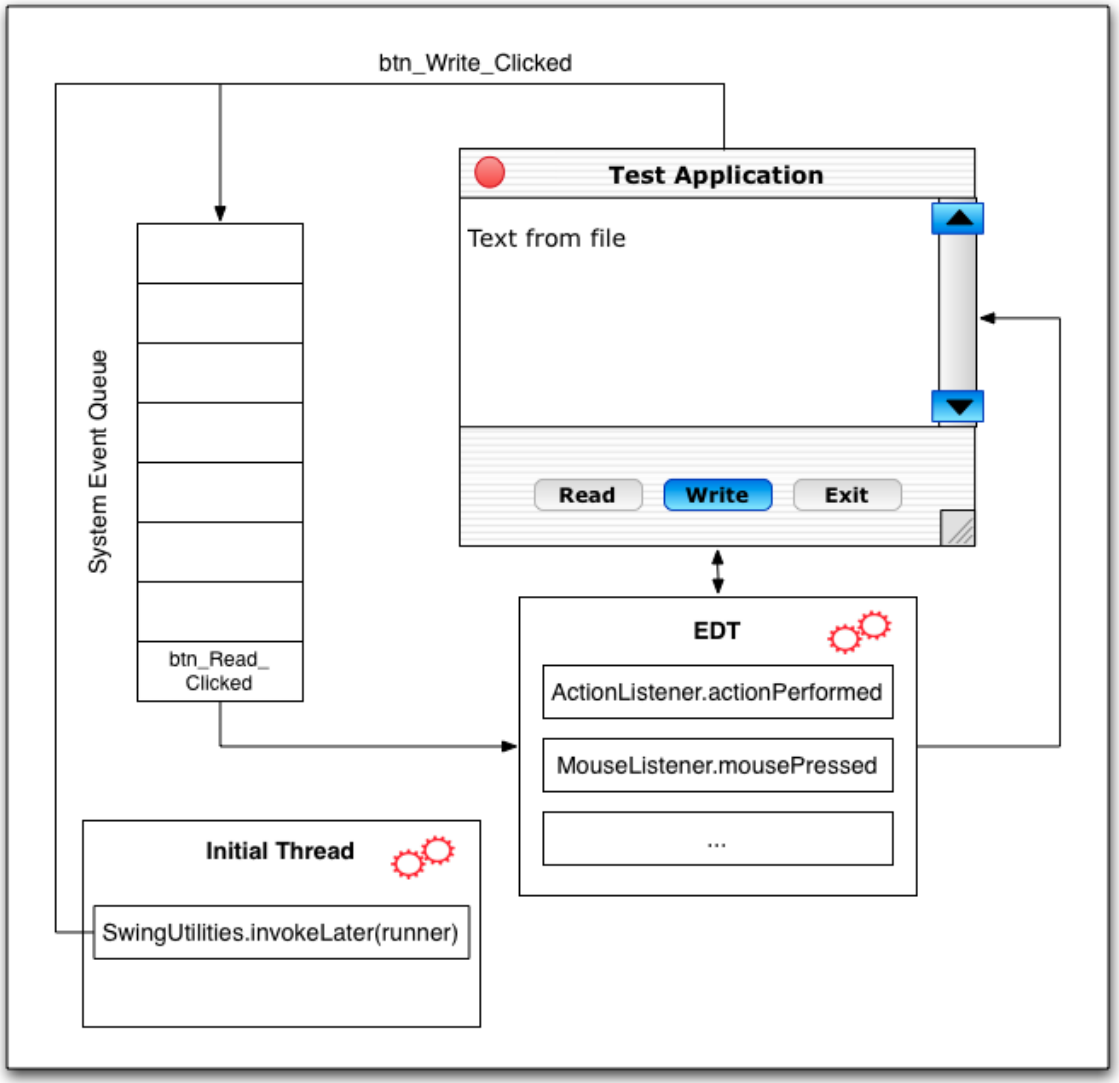
# SwingUtilities

- Il modo giusto di inizializzare una GUI quindi non è quello che abbiamo visto prima, ma…

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        createAndShowGUI();
    }
})
```

- In questo modo creo un oggetto Runnable (il quale si occupa di creare l'interfaccia) e chiedo che venga gestito dall'EDT

# La coda degli eventi di sistema



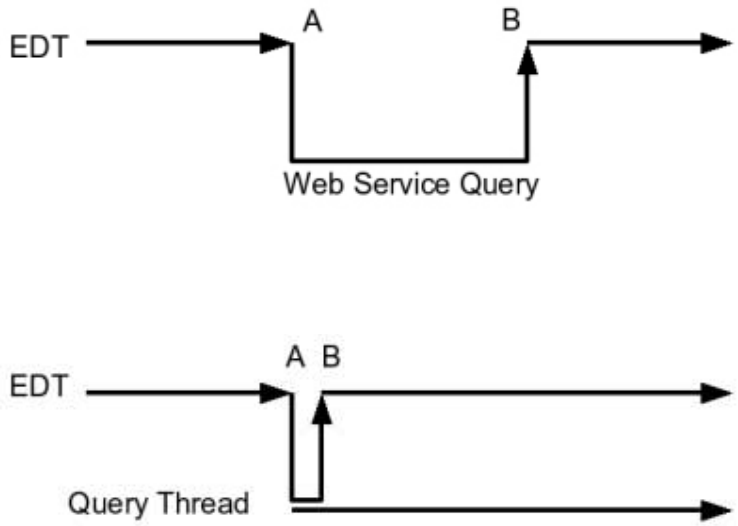
# Considerazioni per la progettazione

- Per avere una buona interfaccia grafica bisogna essere reattivi
  - Bisogna dare l'impressione all'utente che sia sempre sotto controllo
- Il segreto è cercare di scrivere EventListener che siano veloci
- In questo modo eviteremo situazioni in cui un click può portare a una lunga attesa prima di vedere un risultato

- Ma se devo fare qualcosa che richiede tempo? Come faccio?
  - Dobbiamo introdurre il terzo tipo di thread

# 3 tipi di thread - Worker Thread

- Il Worker thread
  - viene creato dal programmatore
  - per l'esecuzione di compiti gravosi in background



# Un indizio…

- Per capire a fondo dobbiamo introdurre (per davvero) il multi-threading e la concorrenza
  - Sono argomenti di corsi precedenti, li rivedremo anche in Java

- Per chi volesse approfondire
  - Il modo giusto è quello di usare un oggetto SwingWorker
    - Definisce un metodo che si occupa del lavoro in background, e
    - Un secondo metodo che invece verrà eseguito nell'EDT e che fa da ponte tra il background thread e la GUI

# Link utili

- http://java.sun.com/docs/books/tutorial/uiswing/
  - Il miglior punto di partenza.
- Java Swing, Second Edition (Marc Loy, Robert Eckstein, Dave Wood,James Elliott, et al.)
  - Ricco di esempi dettagliati e ben commentati.
- Thinking in Java 4^th Edition (Bruce Eckel)
  - Sempre attuale, contiene esempi eleganti.
- http://en.wikipedia.org/wiki/Event-driven_programming
  - Event driven programming in generale.
  - In particolare consiglio: http://eventdrivenpgm.sourceforge.net/