# Pulsanti animati.

## Introduzione.

Questo tutorial descrive un'idea – stantìa – per animare l'icona di un pulsante e presenta la sua traduzione in una classe Java.

## Dal risultato alla strumento.

Dato un AbstractButton (JButton, JMenuItem, JCheckBox eccetera) ed un'immagine contenente i frame dell'animazione, l'applicazione dell'animazione al pulsante avviene con l'invocazione:

```
Animaker.bindAnimation(P, W, H, T, B);
```

dove P è il percorso relativo dell'immagine, W ed H sono, rispettivamente, la larghezza e l'altezza di ciascun frame, T è il periodo dell'animazione e B l'AbstractButton a cui è applicata l'animazione.

## Predisposizione dei frame.

I frame dell'animazione devono avere dimensioni costanti e sono distribuiti all'interno di un'immagine secondo lo schema esemplificato nell'immagine seguente.


Figura 1 – Schema di distribuzione dei frame.

Il primo frame dell'animazione è anche l'icona del pulsante quando questi non sia animato. L'animazione è attivata quando il mouse entra nell'area del pulsante ed è disattivata quando il mouse è premuto o quando il mouse esce dall'area del controllo animato.

## AniMaker – Disegno del frame corrente.

AniMaker è un Icon ed usa sè stesso come icona del controllo animato, proiettando il frame corrente dell'animazione nel suo metodo

```
public void paintIcon(Component c, Graphics g, int x, int y)
```

La proiezione usa la posizione del frame corrente, in termini di indici di riga e di colonna e la dimensione di un frame per determinare lo spostamento e il clip necessari a proiettare il frame corrente usando l'intera immagine contenente i frame. Il metodo paintIcon di AniMaker ha la seguente forma.

```
public void paintIcon(Component c, Graphics g, int x, int y) {
        int offX = currentFrameColumn * FRAME_WIDTH;
        int offY = currentFrameRow * FRAME_HEIGHT;
        Shape previousClip = g.getClip();
        FRAME_CLIP_BUFFER.setRect(x, y, FRAME_WIDTH, FRAME_HEIGHT);
        g.setClip(FRAME_CLIP_BUFFER.createIntersection(g.getClipBounds()));
        g.translate(-offX, -offY);
        g.drawImage(FRAME_SET, x, y, c);
```

```
        g.translate(offX, offY);
        g.setClip(previousClip);
}
```

I primi due enunciati determinano il valore dello spostamento degli assi da applicare per far sì che l'origine del frame corrente nell'immagine coincida con l'origine della regione del contesto grafico in cui dovrà essere disegnata l'icona. Il terzo enunciato immagazzina un riferimento alla regione di spazio in cui è limitata l'applicazione di un effetto grafico (c.d. *clip area*) al momento dell'invocazione di paintIcon. La conservazione è necessaria poichè la regione di clipping sarà mutata negli enunciati successivi. Tale mutazione, utile ai fini della proiezione di un singolo frame a partire dall'immagine di tutti i frame, non deve propagarsi ai successivi utenti dello stesso contesto grafico. La conservazione della regione di clipping originale permette quindi di ripristinare lo stato di questo aspetto del contesto grafico una volta che siano terminate le particolari operazioni di disegno di AniMaker. La quarta operazione di paintIcon consiste nell'impostazione iniziale di un'area rettangolare avente le stesse dimensioni di un frame, collocata nel punto in cui dovrà essere disegnata l'icona. Le coordinate di questo punto sono fornite dall'API Swing come parametri del metodo paintIcon. Successivamente, tale regione assume il valore dell'intersezione tra sè e la regione di clipping attiva nel contesto grafico al momento dell'invocazione di paintIcon. L'intersezione fa sì che il disegno non si propaghi oltre le dimensioni del pulsante nel caso in cui queste siano insufficiente a proiettare completamente l'icona.
Dopo aver calcolato queste impostazioni iniziali, paintIcon procede al disegno vero e proprio. La prima operazione è lo spostamento dell'origine degli assi del contesto grafico – in realtà è una concatenazione di trasformazioni a determinare l'effettivo spostamento dell'origine ma, ai nostri fini, si tratta di un dettaglio non rilevante. Questa operazione può essere figuratamente intesa come uno scivolamento dell'immagine contente i frame che termina quando l'angolo in alto a sinistra del frame corrente coincida con il punto a partire dal quale avrà effetto il disegno. La seconda operazione è il disegno dell'intera immagine contenente tutti i frame. Il prodotto di questo disegno è influenzato dalle impostazioni correnti del contesto grafico tra cui la regione di clipping e la traslazione degli assi. Il risultato è l'apparizione sul controllo animato di una porzione del singolo frame non superiore all'area effettivamente disponibile all'icona del pulsante. Gli ultimi due passaggi ripristinano lo stato originale del contesto grafico annullando lo spostamento degli assi e la mutazione alla regione influenzata dalle operazioni di disegno.

# AniMaker – Passaggio al frame successivo.

Il frame corrente dell'animazione è identificato da un'indice di riga ed un indice di colonna e l'algoritmo che determina il passaggio da un frame al successivo è contenuto nel metodo

```
private void showNextFrame() {
        currentFrameColumn++;
        if(currentFrameColumn == COLUMN_COUNT) {
                currentFrameColumn = 0;
                currentFrameRow++;
                if(currentFrameRow == ROW_COUNT) {
                        currentFrameRow = 0;
                }
        }
        TARGET_BUTTON.repaint();
}
```

La succesione è realizzabile anche usando un solo indice ma ritengo che l'identificazione esplicita della riga e della colonna del frame corrente sia più chiara. Il metodo showNextFrame scorre gli indici di riga e di colonna da sinistra verso destra e dall'alto verso il basso. La successione è ciclica.

# AniMaker – Motore di animazione.

La sostituzione periodica ciclica del frame corrente è realizzata da un javax.swing.Timer. Questo tipo di Timer opera come capsula di un Thread che esegue periodicamente l'accodamento di un blocco di

istruzioni nella coda degli eventi AWT. Il risultato è un'animazione coordinata con l'esecuzione dell'AWT Event Dispatcher Thred conforme all'architettura a Thread singolo di Swing.

# AniMaker – Codice sorgente.

Segue il codice completo della classe AniMaker.

```
package it.tukano.anibutton;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.imageio.ImageIO;
import javax.swing.*;

public class AniMaker extends MouseAdapter implements Icon, ActionListener {
        /** Periodo dell'animazione */
        private final int ANIMATION_PERIOD;

        /** Larghezza di un frame */
        private final int FRAME_WIDTH;

        /** Altezza di un frame */
        private final int FRAME_HEIGHT;

        /** Numero di righe nell'immagine contenente i
        frame */
        private final int ROW_COUNT;

        /** Numero di colonne nell'immagine contenente i
        frame */
        private final int COLUMN_COUNT;

        /** Temporizzatore swing dell'animazione */
        private final javax.swing.Timer ANIMATION_TIMER;

        /** Immagine contenente tutti i frame dell'animazione */
        private final BufferedImage FRAME_SET;

        /** Pulsante bersaglio dell'animazione */
        private final AbstractButton TARGET_BUTTON;

        /** Area di clipping usata durante il disegno */
        private final Rectangle FRAME_CLIP_BUFFER = new Rectangle();

        /** Indice della riga in cui si trova il frame corrente
        dell'animazione */
        private int currentFrameRow;

        /** Indice della colonna in cui si trova il frame corrente
        dell'animazione */
        private int currentFrameColumn;

        /** Inizializza un AniMaker.
        @param frameSet immagine contente i frame dell'animazione
        @param frameWidth larghezza di un frame
        @param frameHeight altezza di un frame
        @param animationPeriod periodo dell'animazione
        @param button pulsante a cui è applicata l'animazione */
        public AniMaker(BufferedImage frameSet, int frameWidth, int frameHeight,
                int animationPeriod, AbstractButton button)
        {
                FRAME_WIDTH = frameWidth;
                FRAME_HEIGHT = frameHeight;
                ROW_COUNT = frameSet.getHeight() / frameHeight;
```

```
        COLUMN_COUNT = frameSet.getWidth() / frameWidth;
        FRAME_SET = frameSet;
        ANIMATION_PERIOD = animationPeriod;
        ANIMATION_TIMER = new javax.swing.Timer(ANIMATION_PERIOD, this);
        TARGET_BUTTON = button;
        TARGET_BUTTON.addMouseListener(this);
        TARGET_BUTTON.setIcon(this);
}

/** Salta al primo frame dell'animazione
e richiede un aggiornamento del pulsante
bersaglio */
private void jumpToFirstFrame() {
        currentFrameColumn = currentFrameRow = 0;
        TARGET_BUTTON.repaint();
}

/** Calcola gli indici di riga e colonna del
frame successivo e richiede un aggiornamento
grafico del pulsante bersaglio */
private void showNextFrame() {
        currentFrameColumn++;
        if(currentFrameColumn == COLUMN_COUNT) {
                currentFrameColumn = 0;
                currentFrameRow++;
                if(currentFrameRow == ROW_COUNT) {
                        currentFrameRow = 0;
                }
        }
        TARGET_BUTTON.repaint();
}

/** Avvia l'animazione */
public void startAnimation() {
        ANIMATION_TIMER.start();
}

/** Interrompe l'animazione. L'interruzione
riporta l'animazione al primo frame */
public void stopAnimation() {
        jumpToFirstFrame();
        ANIMATION_TIMER.stop();
}

/** @inheritDoc */
public int getIconWidth() {
        return FRAME_WIDTH;
}

/** @inheritDoc */
public int getIconHeight() {
        return FRAME_HEIGHT;
}

/** Disegna il frame corrente dell'animazione sul controllo c */
public void paintIcon(Component c, Graphics g, int x, int y) {
        int offX = currentFrameColumn * FRAME_WIDTH;
        int offY = currentFrameRow * FRAME_HEIGHT;
        Shape previousClip = g.getClip();
        FRAME_CLIP_BUFFER.setRect(x, y, FRAME_WIDTH, FRAME_HEIGHT);
        g.setClip(FRAME_CLIP_BUFFER.createIntersection(g.getClipBounds()));
        g.translate(-offX, -offY);
        g.drawImage(FRAME_SET, x, y, c);
        g.translate(offX, offY);
        g.setClip(previousClip);
}

/** L'ingresso del mouse nell'area del pulsante
animato causa avvio dell'animazione */
```

```
        public void mouseEntered(MouseEvent e) {
                if(e.getSource() == TARGET_BUTTON) {
                        startAnimation();
                }
        }

        /** L'uscita del mouse dall'area del pulsante
        animato causa interruzione dell'animazione */
        public void mouseExited(MouseEvent e) {
                if(e.getSource() == TARGET_BUTTON) {
                        stopAnimation();
                }
        }

        /** La pressione del mouse sul pulsante animato
        causa interruzione dell'animazione */
        public void mousePressed(MouseEvent e) {
                if(e.getSource() == TARGET_BUTTON) {
                        stopAnimation();
                }
        }

        /** Blocco ciclicamente eseguito dall'AWT Event
        Dispatcher tramite il javax.swing.Timer che controlla
        l'animazione */
        public void actionPerformed(ActionEvent e) {
                if(e.getSource() == ANIMATION_TIMER) {
                        showNextFrame();
                }
        }

        /** Semplificazione dell'uso di un AniMaker */
        public static AniMaker bindAnimation(String frameResourceAddress,
                int frameWidth, int frameHeight, int animationPeriod,
                AbstractButton button)
        {
                try {
                        java.net.URL address = AniMaker.class.getResource(
                                frameResourceAddress);
                        return new AniMaker(ImageIO.read(address), frameWidth,
                                frameHeight, animationPeriod, button);
                } catch(java.io.IOException ex) {
                        throw new RuntimeException(ex);
                }
        }
}
```

# Note.

I frame, che devono avere ognungo dimensioni uguali all'altro, possono avere un'altezza diversa dalla larghezza. Il formato dell'immagine può essere uno qualsiasi tra i formati supportati da ImageIO – sebbene png resti il formato principe della piattaforma Java. Da un punto di vista ergonomico, lo scopo di un'animazione su un controllo è quello di fornire una documentazione visiva dello scopo nè più nè meno di quanto fa un testo a scomparsa. AniMaker non considera la possibilità che l'animazione, una volta agganciata ad un pulsante, possa dover essere rimossa. In tal caso occorrerebbe aver cura di scollegare dal pulsante il MouseListener AniMaker così da garantire la possibilità che il garbage collector rimuova tanto l'istanza di AniMaker quanto, eventualmente, il pulsante animato dalla memoria. Frame di dimensione relativamente piccola non necessitano di particolari accorgimenti per garantire un'animazione fluida. Nel caso in cui si vogliano ottenere performance migliori l'immagine contente i frame dovrebbe essere convertita in un formato compatibile con il display che proietta il pulsante.