

Caratteristiche delle applet

Mentre le **applicazioni** Java sono programmi stand-alone a tutti gli effetti, le **applets** sono programmi pensati per animare e rendere più interessanti le pagine WWW.

Le applets Java eseguono all'interno dei browser WWW (HotJava, Netscape), oppure con gli appletviewer forniti nel JDK.

- All'interno di una pagina HTML si inserisce un puntatore alla applet, mediante uno specifico HTML tag.
- Quando il browser, leggendo la pagina HTML, trova il tag <APPLET>, scarica la applet dal server Web.
- La applet viene eseguita sul sistema locale, il client, dove il browser risiede.

Le applet, proprio perché eseguono in un ambiente grafico supportato dal browser possono sfruttare tutte le capacità di gestione della grafica, delle immagini, delle interfacce utente e dell'accesso in rete proprie dei browser.

Limiti delle applet

Le applet sono soggette a molte restrizioni sulle loro capacità per motivi di sicurezza, proprio perché eseguono sulla macchina locale.

- non possono leggere e scrivere sul file system della macchina locale, ad eccezione di alcuni direttori specificamente indicati dall'utente;
- non possono comunicare con altri server a parte quello da cui sono state originate;
- non possono eseguire programmi sul file system locale (ad esempio non possono fare fork di processi);
- non possono caricare programmi nativi della piattaforma locale, comprese le librerie DLL.

⇒ Il compilatore e l'interprete Java eseguono diversi controlli di consistenza e sicurezza.

Creazione di una applet

1. Per creare una applet, si crea una sottoclasse della classe `Applet`, appartenente al package `java.applet`.

```
public class myClass extends java.applet.Applet {  
    ....  
}
```

⇒ la classe principale della applet deve essere dichiarata `public`

2. Quando un browser incontra il tag `<APPLET>`, scarica il codice relativo alla applet, insieme a tutte le classi necessarie alla applet.
 3. Non esistendo il metodo `main()`, Java crea una istanza della classe associata alla applet ed attiva tutti i metodi base di sistema sull'istanza
- ⇒ applet diverse appartenenti alla stessa classe usano istanze diverse.

Metodi base di sistema

- Le applet “riconoscono” molte attività, che corrispondono ad eventi importanti per il sistema, quali l’inizializzazione, l’aggiornamento dello schermo, la pressione del pulsante del mouse oppure di un tasto sulla tastiera.
 - A ciascuna attività corrisponde un metodo che descrive il comportamento della applet a fronte dell’evento.
 - Il browser chiama il metodo opportuno a seconda dell’evento che si verifica.

⇒ La definizione dei metodi base è una delle parti più importanti nella definizione di una applet.

Metodi base di sistema

- metodo `init()`: inizializzazione della applet
 - si esegue quando la applet è caricata
 - eseguito una volta sola durante la vita della applet
- metodo `start()`: inizio delle attività della applet
 - può essere chiamato dopo l'inizializzazione o dopo una sospensione
- metodo `stop()`: sospensione delle attività della applet
 - l'esecuzione di una applet può essere sospesa quando l'utente cambia pagina
- metodo `destroy()`: distruzione della applet
 - chiamato al termine dell'esecuzione
 - permette di rilasciare le risorse utilizzate dalla applet
- metodo `paint()`: aggiornamento della parte di schermo cui la applet ha accesso
 - eseguito più volte durante la vita della applet

Spesso è sufficiente utilizzare direttamente i metodi ereditati dalla classe `Applet` di Java (che non eseguono alcuna attività); se devono essere modificati è necessario farne l'overriding.

Un esempio: la applet Hello Again

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;

public class HelloAgainApplet extends java.applet.Applet {
    Font f = new Font("TimesRoman", Font.BOLD, 36);

    public void paint(Graphics g) {
        g.setFont(f);
        g.setColor(Color.red);
        g.drawString("Hello again!", 5, 50);
    }
}
```

Inclusione della applet in una pagina HTML

Si utilizza il tag `<APPLET>`. I suoi parametri sono molto simili a quelli del tag ``.

- `WIDTH` e `HEIGHT` definiscono la zona dello schermo (in pixel) dedicata alla applet.
- Se la applet è piccola, può essere “inclusa” in una linea di testo. In questo caso, `ALIGN` definisce l’allineamento della applet nella linea rispetto agli altri elementi contenuti nella linea stessa. Può assumere i valori `LEFT`, `RIGHT`, `TOP`, `TEXTTOP`, `MIDDLE`, `ABSMIDDLE`, `BASELINE`, `BOTTOM` e `ABSBOTTOM`.
- `HSPACE` and `VSPACE` fissano lo spazio (in pixel) tra la applet ed il testo che la circonda.
- `CODE` and `CODEBASE` indicano rispettivamente il nome del file `.class` che contiene la applet e il direttorio in cui si trova il file.
- Il testo contenuto tra `<APPLET>` e `</APPLET>` è mostrato dai browser che non capiscono il tag `<APPLET>`.

In HTML 4.0 si utilizza il tag `<OBJECT>`. I suoi parametri sono molto simili a quelli del tag `<APPLET>`, ad eccezione dell’uso del parametro `CLASSID: "java:pippo.class"` al posto del parametro `CODE`.

Un esempio di applet

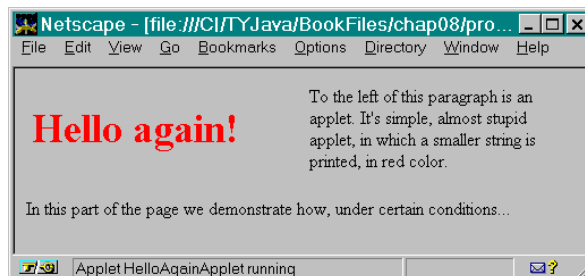
```
<P><APPLET CODE="HelloAgainApplet"
```

```
  WIDTH=300 HEIGHT=200 ALIGN=LEFT>Hello Again!</APPLET>
```

```
To the left of this paragraph is an applet. It's simple, almost stupid  
applet, in which a smaller string is printed, in red color.
```

```
<BR CLEAR=ALL>
```

```
<P>In this part of the page we demonstrate how, under certain conditions...
```



Passaggio di parametri ad una applet

Il tag PARAM permette di passare parametri alla applet tramite la pagina HTML.

- I suoi parametri sono NAME e VALUE.

```
<P><APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=100>
<PARAM NAME=font VALUE="TimesRoman">
<PARAM NAME=size VALUE="36">
A Java applet appears here (maybe....)</APPLET>
```

Quando il browser legge questa pagina HTML, passa alla applet MyApplet due parametri.

Passaggio di parametri ad una applet

È possibile leggere i parametri di una applet nel metodo `init()`, grazie al metodo `getParameter()`.

- `getParameter()` ha come argomento una stringa contenente il nome del parametro e ritorna in una stringa il valore associato a tale parametro.
- La stringa con il nome del parametro deve essere esattamente identica (comprese maiuscole e minuscole) a quella associata al parametro NAME nella pagina HTML per essere riconosciuta.
- Se il parametro non è stato specificato nella pagina HTML, `getParameter` ritorna `null`.

```
String theFontName = getParameter("font");
if (theFontName == null)
    theFontName = "Courier";

int theSize;
String s = getParameter("size");
if (s == null)
    theSize = 12;
else
    theSize = Integer.parseInt(s);
```

Un esempio completo

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;

public class MoreHelloApplet extends java.applet.Applet {

    Font f = new Font("TimesRoman",Font.BOLD,36);
    String name;

    public void init() {
        this.name = getParameter("name");
        if (this.name == null)
            this.name = "Laura";
        this.name = "Hello " + name + "!";
    }

    public void paint(Graphics g) {
        g.setFont(f);
        g.setColor(Color.red);
        g.drawString(this.name, 5, 50);
    }
}
```

Archivi: file jar

Per evitare di dovere aprire una connessione per ogni singolo file necessario all'esecuzione della applet (file .class, file audio, immagini, file di testo) è possibile creare un archivio, ovvero un file JAR.

- Un archivio Java è un insieme di classi e di altri file (eventualmente compressi) contenuti in un unico file.
- Il jdk fornisce un programma `jar` che permette di creare archivi.

```
jar cf Animazione.jar *.class *.gif // Crea archivio
```

- Si utilizza il parametro `archive=` nel tag `<APPLET>` per permettere al browser di trasferire l'archivio. È comunque necessario specificare mediante il parametro `CODE` il nome del file eseguibile.

Grafica: la classe Graphics

È la classe che supporta le capacità grafiche delle applet, quali disegnare linee, forme, caratteri e presentare immagini sullo schermo, mediante una serie di metodi.

- Non è necessario creare una istanza della classe `Graphics` per disegnare sullo schermo
⇒ Il metodo `paint()` fornisce un oggetto `Graphics` agendo sul quale si disegna sullo schermo.
- Il sistema di coordinate bidimensionali ha
 - l'origine, ovvero il punto $(0, 0)$ in alto a sinistra
 - i valori positivi della coordinata x si hanno spostandosi sulla destra
 - i valori positivi della coordinata y si hanno muovendosi in basso
- I punti `Point`, usati come riferimento per disegnare qualunque oggetto, esprimono le coordinate in pixels sullo schermo e sono valori interi.

Disegnare linee e rettangoli

Per disegnare una linea

```
public void paint(Graphics g) {  
    g.drawLine(25, 25, 75, 75);  
}
```

Per disegnare un rettangolo, si specificano la coordinata del punto in alto a sinistra, larghezza e lunghezza

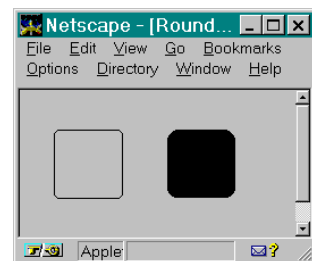
```
public void paint(Graphics g) {  
    g.drawRect(20, 20, 60, 60); // x0, y0, width, height  
    g.fillRect(120, 20, 60, 60);  
}
```

Per disegnare un rettangolo con gli angoli arrotondati

```
public void paint(Graphics g) {  
    g.drawRoundRect(20, 20, 60, 60, 10, 10);  
    g.fillRoundRect(120, 20, 60, 60, 20, 20);  
}
```

Per disegnare un rettangolo “tridimensionale” (in realtà si ottiene un effetto tridimensionale stile bottoni)

```
public void paint(Graphics g) {  
    g.draw3DRect(20, 20, 60, 60, true);  
    g.fill13DRect(120, 20, 60, 60, false);  
}
```

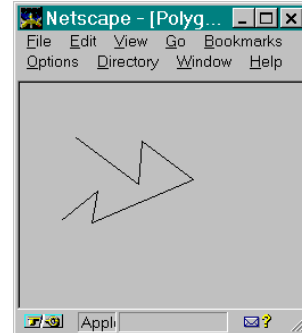


Disegnare poligoni

Per disegnare un poligono è necessario definire una serie di coordinate x e y, come array oppure come istanze della classe Polygon.

```
public void paint(Graphics g) {  
    int ics[] = {39, 94, 97, 142, 53, 58, 26};  
    int ipsilon[] = {33, 74, 36, 70, 108, 80, 106};  
    int punti = ics.length;  
    g.drawPolygon(ics, ipsilon, punti);  
}
```

```
public void paint(Graphics g) {  
    int ics[] = {39, 94, 97, 142, 53, 58, 26};  
    int ipsilon[] = {33, 74, 36, 70, 108, 80, 106};  
    int punti = ics.length;  
    Polygon poly = new Polygon(ics, ipsilon, punti);  
    g.fillPolygon(poly);  
}
```



Il poligono viene chiuso in modo automatico da Java. `drawPolyline()` permette di avere poligoni aperti.

È possibile aggiungere punti ad un oggetto Polygon

```
poly.addPoint(20, 35);
```

CREAZIONE DI APLET JAVA – 15

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Disegnare cerchi, ellissi e archi

Per disegnare cerchi oppure ellissi si usano gli ovali.

```
public void paint(Graphics g) {  
    g.drawOval(20, 20, 60, 60); // x0, y0, width, height  
    g.fillOval(120, 20, 100, 60);  
}
```

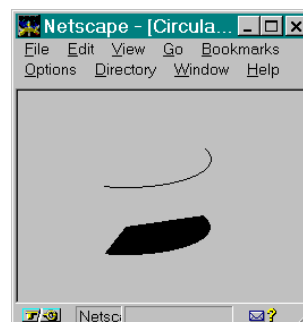
Gli archi sono definiti come pezzi di ellissi o cerchi. È piuttosto difficile visualizzarli; il modo migliore è provare e riprovare.

Si definiscono inizialmente le dimensioni del cerchio (ellisse) da cui si otterrà l'arco. Infine si devono fornire i punti di inizio e fine dell'arco mediante l'angolo di inizio e l'angolo sotteso all'arco. Angoli definiti positivi in senso orario (90° asse verticale).

```
public void paint(Graphics g) {  
    g.drawArc(20, 20, 60, 60, 90, 180);  
    g.fillArc(120, 20, 60, 60, 90, 180);  
}
```



```
public void paint(Graphics g) {  
    g.drawArc(10, 20, 150, 50, 25, -130);  
    g.fillArc(10, 80, 150, 50, 25, -130);  
}
```



CREAZIONE DI APLET JAVA – 16

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Un semplice esempio

```
import java.awt.*;
public class Lamp extends java.applet.Applet {

    public void init() {
        resize(300,300);
    }

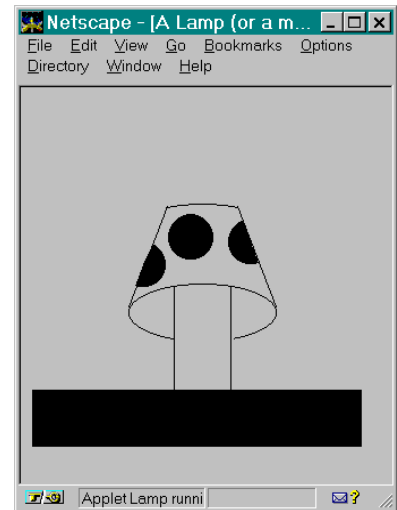
    public void paint(Graphics g) {
        g.fillRect(0,250,290,290);          // la base

        g.drawLine(125,250,125,160);        // lo stelo
        g.drawLine(175,250,175,160);

        g.drawArc(85,157,130,50,-65,312);
        g.drawArc(85,87,130,50,62,58);

        g.drawLine(85,177,119,89);
        g.drawLine(215,177,181,89);

        g.fillArc(78,120,40,40,63,-174); // disegni
        g.fillOval(120,96,40,40);
        g.fillArc(173,100,40,40,110,180);
    }
}
```



CREAZIONE DI APLET JAVA – 17

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Copiare e cancellare pezzi di schermo

- Per copiare un pezzo di schermo si usa il metodo `copyArea` che ha come parametri `x` e `y` dell'angolo in alto a sinistra del rettangolo da copiare, ampiezza ed altezza del rettangolo, distanza in `x` e `y` dell'area su cui copiare.

Esempio

```
g.copyArea (0, 0, 100, 100, 100, 0);
```

- Per cancellare un pezzo di schermo si usa il metodo `clearRect` che ha gli stessi parametri di `drawRect`. In questo modo si colora con il colore del background l'area specificata come parametro nella chiamata al metodo `clearRect()`.

Esempio: cancello l'intera area dedicata alla applet.

```
g.clearRect (0, 0, this.getSize().width, this.size().height):
```

Il metodo `getSize()` torna un oggetto di tipo `Dimension`.

CREAZIONE DI APLET JAVA – 18

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Stampa di testo e gestione dei font

Per scrivere sullo schermo si deve innanzitutto creare una istanza della classe `Font`.

Gli oggetti `Font` sono identificati da

- un nome: una stringa che rappresenta la famiglia del font: `TimesRoman`, `Courier`, `Helvetica`. Nella versione 1.2 si usano i nomi `.serif`, `monospaced`, `sanserif`.
- uno stile: una costante intera (`Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`)
⇒ le costanti possono essere sommate: `Font.BOLD + Font.ITALIC`
- la dimensione in punti: un numero

```
public void paint(Graphics g) {  
    Font f = new Font("TimesRoman", Font.BOLD, 72);  
    g.setFont(f);  
    g.drawString("Questo e' un font enorme.", 10, 100);  
}
```

Esiste anche il metodo `drawChars`, che richiede come parametri un array di caratteri, un intero che rappresenta il primo carattere da riprodurre sullo schermo, un intero per l'ultimo carattere da rappresentare, e le coordinate `x` ed `y`.

Gestione dei Font

I metodi più importanti per ottenere informazioni sui font correnti sono:

```
getFont(): ritorna l'oggetto Font corrente  
getName(): ritorna una stringa con il nome del font  
getSize(): ritorna la dimensione del font  
getStyle(): ritorna lo stile del font  
isPlain()  
isBold()  
isItalic()
```

Gestione dei Font

Per ottenere informazioni più specifiche sui singoli font si sfrutta la classe `FontMetrics`.

I metodi principali sono:

- `stringWidth()`: data una stringa, ne ritorna l'ampiezza in pixel
- `charWidth()`: dato un carattere ne ritorna l'ampiezza
- `getAscent()`: ritorna la distanza tra la base e l'estremo superiore del font
- `getDescent()`: ritorna la distanza tra la base e l'estremo inferiore del font
- `getLeading()`: ritorna lo spazio tra l'estremo inferiore di un carattere e l'estremo superiore di quello nella riga successiva
- `getHeight()`: ritorna l'altezza totale del font

Un esempio di gestione dei font

La seguente applet centra orizzontalmente e verticalmente la scritta rispetto allo spazio dedicato alla applet.

```
import java.awt.Font;
import java.awt.Graphics;
import java.awt.FontMetrics;

public class Centered extends java.applet.Applet {

    public void paint(Graphics g) {
        Font f = new Font("TimesRoman", Font.PLAIN, 36);
        FontMetrics fm = getFontMetrics(f);
        g.setFont(f);

        String s = "This is how the world ends.";
        int xstart = (this.getSize().width - fm.stringWidth(s)) / 2;
        int ystart = (this.getSize().height + fm.getHeight()) / 2;
        g.drawString(s, xstart, ystart);
    }
}
```

Gestione dei colori

- I metodi per gestire i colori sono contenuti nella classe `Color`.
- I colori sono codificati su 24 bit; ogni colore è costituito da una combinazione di componenti rosso, verde e blu.
- Ciascuna componente è rappresentata con un numero intero tra 0 e 255 (oppure con un `float` tra 0. e 1.).

Sono definite variabili di classe per i principali colori.

Colore	Codice RGB	Colore	Codice RGB
<code>Color.white</code>	255, 255, 255	<code>Color.blue</code>	0, 0, 255
<code>Color.black</code>	0, 0, 0	<code>Color.yellow</code>	255, 255, 0
<code>Color.lightGray</code>	192, 192, 192	<code>Color.magenta</code>	255, 0, 255
<code>Color.gray</code>	128, 128, 128	<code>Color.cyan</code>	0, 255, 255
<code>Color.darkGray</code>	64, 64, 64	<code>Color.pink</code>	255, 175, 175
<code>Color.red</code>	255, 0, 0	<code>Color.orange</code>	255, 200, 0
<code>Color.green</code>	0, 255, 0		

Gestione dei colori

I metodi più importanti per la gestione dei colori sono:

`setColor()`: imposta il colore corrente

`setBackground()`: imposta il colore corrente dello sfondo

`setForeground()`: modifica il colore degli oggetti dell'interfaccia utente (ad esempio i bottoni) ed agisce su componenti dell'interfaccia utente, non su un'istanza della classe `Graphics`

Per tutti i metodi esiste il corrispondente metodo `get . . . ()` che permette di leggere il colore corrente.

Un semplice esempio di uso dei colori

```
import java.awt.Graphics;
import java.awt.Color;

public class ColorBoxes extends java.applet.Applet {

    public void paint(Graphics g) {
        int rval, gval, bval;

        for (int j = 30; j < (this.size().height - 25); j += 30)
            for (int i = 5; i < (this.size().width - 25); i += 30) {
                rval = (int)Math.floor(Math.random() * 256);
                gval = (int)Math.floor(Math.random() * 256);
                bval = (int)Math.floor(Math.random() * 256);

                g.setColor(new Color(rval,gval,bval));
                g.fillRect(i,j,25,25);
                g.setColor(Color.black);
                g.drawRect(i-1,j-1,27,27);
            }
    }
}
```

CREAZIONE DI APPLLET JAVA – 25

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Animazione

Per eseguire l'animazione di un'immagine si devono eseguire due passi:

1. definizione dell'immagine da animare
2. ripetizione dell'aggiornamento dello schermo in modo da creare l'illusione del movimento

- Il metodo `paint()` è chiamato automaticamente da Java tutte le volte che risulta necessario aggiornare (rinfrescare) la zona del video dedicata alla applet: la prima volta che una applet viene attivata, ogni volta che si sposta la finestra del browser, ogni volta che un'altra finestra si sovrappone a quella del browser, ...).
- È possibile richiedere esplicitamente a Java l'esecuzione del metodo di aggiornamento dello schermo ogni volta che sia necessario.

⇒ Per modificare ciò che si vede sullo schermo, è sufficiente costruire un'immagine di ciò che si vuole disegnare, e chiedere a Java di rinfrescare lo schermo.

- Tutte le modifiche necessarie per creare le immagini sono fatte in un metodo specifico. Il metodo `paint()` si occupa solo di copiare sullo schermo l'immagine corrente.
- Al termine delle operazioni di preparazione dell'immagine, si chiama il metodo `repaint()` che a sua volta eseguirà la chiamata a `paint()`.

⇒ Eseguendo in modo ciclico e ad una certa velocità le operazioni di cui sopra si ottiene una semplice animazione.

CREAZIONE DI APPLLET JAVA – 26

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Animazione: i metodi `start()` e `stop()`

Per eseguire applet contenenti animazioni si devono utilizzare i metodi `start()` e `stop()`.

- Il metodo `start()` innesca l'esecuzione della applet.
- Il metodo `stop()` viene eseguito quando la applet sospende la sua esecuzione (cambio di pagina HTML da parte del browser) e permette di rilasciare le risorse di sistema utilizzate per l'esecuzione della applet.

⇒ è necessario eseguire la ridefinizione di `start()` e `stop()`

Un semplice esempio: l'orologio digitale

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;

public class DigitalClock extends java.applet.Applet {
    Font theFont = new Font("TimesRoman",Font.BOLD,24);
    Date theDate;

    public void start() {
        while (true) {
            theDate = new Date();
            repaint();
            try { Thread.sleep(1000); }
            catch (InterruptedException e) { }
        }
    }
    public void paint(Graphics g) {
        g.setFont(theFont);
        g.drawString(theDate.toString(),10,50);
    }
}
```

Multithreading

L'esempio precedente non funziona!

- Il ciclo infinito presente nel metodo `start()` monopolizza le risorse del sistema, impendendo anche al metodo `paint()` di effettuare il rinfresco dello schermo.
- Non è possibile sospendere la applet perchè non si può chiamare il metodo `stop()`.

⇒ La applet deve essere riscritta utilizzando un thread.

- Un thread è un processo.
- Multithreading: più thread di esecuzione diversi eseguono contemporaneamente all'interno dello stesso programma, in parallelo e senza interferenze.
- Ogni volta che è necessario eseguire una computazione di una certa lunghezza è bene creare un thread separato che esegua tale operazione.

È **sempre** una buona norma utilizzare la programmazione con threads quando si scrive una applet.

Creazione di una applet con thread

Per creare una applet che sfrutti i thread si deve:

- modificare la segnatura della applet in modo che contenga le parole `implements Runnable`
- definire una variabile di istanza che contenga il thread della applet
- modificare il metodo `start()` in modo che si limiti a creare un thread eseguibile
- definire un metodo `run()` che contenga il codice che descrive il funzionamento della applet
- definire un metodo `stop()` che sospenda l'esecuzione del thread quando l'utente cambia pagina HTML

L'orologio digitale corretto (1/2)

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;

public class DigitalThreads extends java.applet.Applet
    implements Runnable {

    Font theFont = new Font("TimesRoman",Font.BOLD,24);
    Date theDate;
    Thread runner;

    public void start() {
        if (runner == null); {
            runner = new Thread(this);
            runner.start(); // Innesca invocazione metodo run() della applet
        }
    }

    public void stop() { runner = null; }
```

CREAZIONE DI APPLET JAVA – 31

*Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino*

JAVA

L'orologio digitale corretto (2/2)

```
public void run() {
    Thread thisThread = Thread.currentThread();
    while (runner == thisThread) {
        repaint();
        try { Thread.sleep(1000); }
        catch (InterruptedException e) { }
    }
}

public void paint(Graphics g) {
    theDate = new Date(); // aggiorno data ogni
                          // volta che devo
                          // ridipingere schermo

    g.setFont(theFont);
    g.drawString(theDate.toString(),10,50);
}
}
```

CREAZIONE DI APPLET JAVA – 32

*Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino*

JAVA

Lo sfarfallio (flicker) nelle animazioni

L'esecuzione del metodo `repaint()` innesca (in modo indiretto) la chiamata al metodo `paint()`:

1. `repaint()` chiama il metodo `update()`
2. il metodo `update()` cancella la parte di schermo dedicata alla applet (dipingendo tutto con il colore di sfondo o background) ed in seguito chiama il metodo `paint()`
3. il metodo `paint()` rinfresca lo schermo disegnando la nuova immagine

⇒ La fase in cui si disegna lo sfondo causa lo sfarfallio.

Per evitare lo sfarfallio si può:

1. ridefinire (overriding) il metodo `update()` in modo da non ridisegnare lo sfondo
2. ridefinire (overriding) il metodo `update()` in modo da disegnare solo la parte che è cambiata
3. ridefinire sia il metodo `update()` sia il metodo `paint()` ed usare la tecnica di double buffering

Il metodo `update()`

La versione standard del metodo `update()` è:

```
public void update(Graphics g) {  
    g.setColor(getBackground());  
    g.fillRect(0, 0, size().width, size().height);  
    g.setColor(getForeground());  
    paint(g);  
}
```

Quando si ridefinisce il metodo `update()` la nuova versione deve comunque eseguire tutte le istruzioni che sono necessarie per il funzionamento della applet.

Prima soluzione: non ridisegnare lo sfondo (1/3)

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;

public class ColorSwirl extends java.applet.Applet
    implements Runnable {

    Font f = new Font("TimesRoman",Font.BOLD,48);
    Color colors[] = new Color[50];
    Thread runThread;

    public void start() {
        if (runThread == null) {
            runThread = new Thread(this);
            runThread.start();
        }
    }

    public void stop() { runThread = null; }
```

CREAZIONE DI APPLLET JAVA – 35

*Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino*

JAVA

Prima soluzione: non ridisegnare lo sfondo (2/3)

```
public void init() {
    float c = 0.0F;
    for (int i = 0; i < colors.length; i++) {
        colors[i] = Color.getHSBColor(c,1.0F,1.0F);
        c += .02F;
    }
}

public void run() {
    int i = 0;        // cycle through the colors
    while (true) {
        setForeground(colors[i]);
        repaint();
        i++;
        try { Thread.currentThread().sleep(50); }
        catch (InterruptedException e) { }
        if (i == (colors.length)) i = 0;
    }
}
```

CREAZIONE DI APPLLET JAVA – 36

*Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino*

JAVA

Prima soluzione: non ridisegnare lo sfondo (3/3)

```
public void paint(Graphics g) {  
    g.setFont(f);  
    g.drawString("All the Swirly Colors", 15,50);  
}  
public void update(Graphics g) {  
    paint(g);  
}  
}
```

La applet Checkers (1/3)

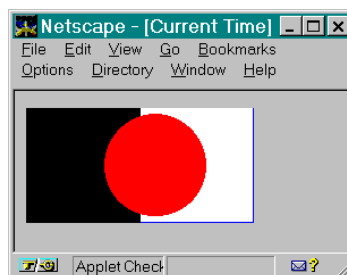
```
import java.awt.Graphics;  
import java.awt.Color;  
  
public class Checkers extends java.applet.Applet  
    implements Runnable {  
  
    Thread runner;  
    int xpos;  
  
    public void start() {  
        if (runner == null); {  
            runner = new Thread(this);  
            runner.start();  
        }  
    }  
  
    public void stop() {  
        runner = null;  
    }  
}
```

La applet Checkers (2/3)

```
public void run() {
    setBackground(Color.blue);
    while (true) {
        for (xpos = 5; xpos <= 105; xpos+=4) {
            repaint();
            try { Thread.sleep(100); }
            catch (InterruptedException e) { }
        }
        for (xpos = 105; xpos > 5; xpos -=4) {
            repaint();
            try { Thread.sleep(100); }
            catch (InterruptedException e) { }
        }
    }
}
```

La applet Checkers (3/3)

```
public void paint(Graphics g) {
    g.setColor(Color.black);      // Draw background
    g.fillRect(0,0,100,100);
    g.setColor(Color.white);
    g.fillRect(101,0,100,100);
    g.setColor(Color.red);        // Draw checker
    g.fillOval(xpos,5,90,90);
}
}
```



Seconda soluzione: ridisegnare solo le parti necessarie (1/3)

```
import java.awt.Graphics;
import java.awt.Color;

public class Checkers2 extends java.applet.Applet
    implements Runnable {
    Thread runner;
    int xpos,ux1,ux2;

    public void run() {
        setBackground(Color.blue);
        while (true) {
            for (xpos = 5; xpos <= 105; xpos+=4) {
                ux2 = xpos + 90;
                repaint();
                try { Thread.sleep(100); }
                catch (InterruptedException e) { }
                if (ux1 == 0) ux1 = xpos; // Importante per assicurare
                                           // esecuzione del metodo paint
            }
        }
    }
}
```

CREAZIONE DI APLET JAVA – 41

*Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino*

JAVA

Seconda soluzione: ridisegnare solo le parti necessarie (2/3)

```
        for (xpos = 105; xpos > 5; xpos -=4) {
            ux1 = xpos;
            repaint();
            try { Thread.sleep(100); }
            catch (InterruptedException e) { }
            if (ux2 == 0) ux2 = xpos + 90;
        }
    }
}

public void update(Graphics g) {
    g.clipRect(ux1, 5, ux2 - ux1, 95);
    paint(g);
}
```

CREAZIONE DI APLET JAVA – 42

*Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino*

JAVA

Seconda soluzione: ridisegnare solo le parti necessarie (3/3)

```
public void paint(Graphics g) {
    g.setColor(Color.black);
    g.fillRect(0,0,100,100);
    g.setColor(Color.white);
    g.fillRect(100,0,100,100);
    g.setColor(Color.red);          // Pedone
    g.fillOval(xpos,5,90,90);
    ux1 = ux2 = 0;                  // calcolo area
}
public void start() {
    if (runner == null); {
        runner = new Thread(this);
        runner.start();
    }
}
public void stop() {
    runner = null;
}
}
```

Uso delle immagini

- La classe Image in java.awt fornisce i metodi per gestire le immagini.
- Java supporta immagini in formato GIF e JPEG.
- I metodi più importanti per caricare le immagini sono:
 - getImage() carica l'immagine
 - getDocumentBase() ritorna la URL che rappresenta il direttorio in cui si trova il file HTML che contiene la applet
 - getCodeBase() ritorna, sotto forma di stringa, la URL in cui si trova la applet

```
Image img = getImage(new URL("http://www.server.com/files/image.gif"));
Image img = getImage(getDocumentBase(), "image.gif");
Image img = getImage(getCodeBase(), "image.gif");
Image img = getImage(getCodeBase(), "images/image.gif");
```

⇒ Se il file non è trovato, getImage ritorna null.

- Per disegnare un'immagine si usa il metodo drawImage() in paint().

Esempio: la applet LadyBug (1/2)

```
import java.awt.Graphics;
import java.awt.Image;

public class LadyBug extends java.applet.Applet {

    Image bugimg;

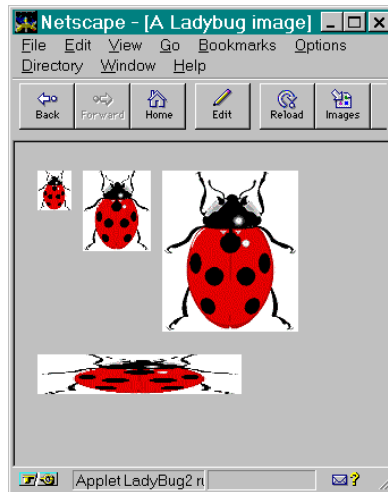
    public void init() {
        bugimg = getImage(getCodeBase(),
                           "images/ladybug.gif");
        // getImage() Torna oggetto Image.
        // Non istanzio bugimg
    }
}
```

Esempio: la applet LadyBug (2/2)

```
public void paint(Graphics g) {
    int iwidth = bugimg.getWidth(this);
    int iheight = bugimg.getHeight(this);
    int xpos = 10;

    // 25 %
    g.drawImage(bugimg, xpos, 10, iwidth/4, iheight/4, this);
    // 50 %
    xpos += (iwidth / 4) + 10;
    g.drawImage(bugimg, xpos, 10, iwidth/2, iheight/2, this);
    // 100%
    xpos += (iwidth/2) + 10;
    g.drawImage(bugimg, xpos, 10, this);
    // 150% x, 25% y
    g.drawImage(bugimg, 10, iheight + 30,
                (int)(iwidth * 1.5), iheight/4, this);
}
}
```

La applet LadyBug



Esempio di animazione: la applet Neko (1/4)

```
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Color;

public class Neko extends java.applet.Applet
    implements Runnable {

    Image nekoPics[] = new Image[9];
    Image currentImg;
    Thread runner;
    int x;
    int y = 50;

    public void init() {
        String nekoSrc[] = { "right1.gif", "right2.gif",
            "stop.gif", "yawn.gif", "scratch1.gif",
            "scratch2.gif", "sleep1.gif", "sleep2.gif",
            "awake.gif" };

        for (int i=0; i < nekoPics.length; i++) {
            nekoPics[i] = getImage(getCodeBase(),
                "images/" + nekoSrc[i]);
        }
    }
}
```


Esempio di animazione: la applet Neko (2/4)

```
public void run() {
    setBackground(Color.white);
    // run from one side of the screen to the middle
    nekoRun(0, getSize().width / 2);
    // stop and pause
    currentImg = nekoPics[2];
    repaint();
    pause(1000);
    // yawn
    currentImg = nekoPics[3];
    repaint();
    pause(1000);
    // scratch four times
    nekoScratch(4);
    // sleep for 5 "turns"
    nekoSleep(5);
    // wake up and run off
    currentImg = nekoPics[8];
    repaint();
    pause(500);
    nekoRun(x, getSize().width + 10);
}
```

Esempio di animazione: la applet Neko (3/4)

```
public void start() {
    if (runner == null) {
        runner = new Thread(this);
        runner.start();
    }
}

public void stop() { runner = null; }

void nekoRun(int start, int end) {
    for (int i = start; i < end; i += 10) {
        x = i;
        // swap images
        if (currentImg == nekoPics[0])
            currentImg = nekoPics[1];
        else currentImg = nekoPics[0];
        repaint();
        pause(150);
    }
}

void pause(int time) {
    try { Thread.sleep(time); }
    catch (InterruptedException e) { }
}
```

Esempio di animazione: la applet Neko (4/4)

```
void nekoScratch(int numTimes) {
    for (int i = numTimes; i > 0; i--) {
        currentImg = nekoPics[4];
        repaint();
        pause(150);
        currentImg = nekoPics[5];
        repaint();
        pause(150);
    }
}

void nekoSleep(int numTimes) {
    for (int i = numTimes; i > 0; i--) {
        currentImg = nekoPics[6];
        repaint();
        pause(250);
        currentImg = nekoPics[7];
        repaint();
        pause(250);
    }
}

public void paint(Graphics screen) {
    if (currentImg != null) screen.drawImage(currentImg, x, y, this);
}
}
```

CREAZIONE DI APLET JAVA – 51

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

Inclusione di suoni in una applet

- Sono supportati diversi formati: AU (codifica della Sun, detta anche codifica a legge μ , di qualità non elevatissima), AIFF, WAV, MIDI0, MIDI1 e RMF.
- È sempre possibile includere suoni mediante puntatori a file esterni nella pagina HTML.
- Per generare un suono

- mediante metodo della classe applet:

```
play(getCodeBase(), "audio/meow.au"); // plays the sound once
```

- mediante metodi della classe AudioClip:

```
AudioClip clip = getAudioClip(getCodeBase(), "audio/loop.au");
clip.play(); // plays the sound once
clip.stop(); // stops the sound
clip.loop(); // plays the sound repeatedly
```

È necessario fermare esplicitamente un suono di background (ovvero un suono che usa il metodo `loop()`) nel metodo `stop()`, altrimenti il suono continua anche quando la applet ha terminato di eseguire.

```
public void stop() {
    if (runner != null) {
        if (bgsound != null)
            bgsound.stop();
        runner.stop();
        runner = null;
    }
}
```

CREAZIONE DI APLET JAVA – 52

Elena Baralis, Andrea Bianco, Maurizio Munafò
Politecnico di Torino

JAVA

La Applet Animator

La Sun fornisce una classe `Animator` nella distribuzione standard di Java

⇒ fornisce una interfaccia di tipo generale per il supporto alle animazioni

Si inserisce in un file HTML una serie di parametri che descrivono il tipo di animazione che si intende eseguire.

La applet `Animator` permette di:

- creare un ciclo di animazione
- aggiungere una colonna sonora alla applet
- aggiungere suoni speciali ad una specifica immagine
- scegliere la velocità di animazione
- determinare l'ordine di visualizzazione delle immagini all'interno dell'animazione

Ridurre lo sfarfallio: double buffering

Il metodo più complesso per ridurre lo sfarfallio è il double buffering.

Il metodo consiste nel creare una seconda superficie fuori dallo schermo su cui disegnare la nuova immagine da visualizzare; alla fine di tale processo la superficie sarà visualizzata in un colpo solo sullo schermo.

⇒ non si rischia di visualizzare parti intermedie dell'immagine disturbando l'effetto di animazione

Siccome è una tecnica costosa in termini di memoria e di efficienza, è bene utilizzarla solo se nessuna delle altre tecniche funziona.

La applet Checkers modificata con double buffering

1. Aggiungere le variabili di istanza per l'immagine esterna ed il suo contenuto grafico

```
Image offscreenImg;  
Graphics offscreenG;
```

2. Aggiungere un metodo `init()` per inizializzare l'immagine esterna

```
public void init() {  
    offscreenImg = createImage(this.size().width, this.size().height);  
    offscreenG = offscreenImg.getGraphics();  
}
```

3. Modificare il metodo `paint` per disegnare l'immagine esterna

```
public void paint(Graphics g) {  
    // Disegno il background  
    offscreenG.setColor(Color.black);  
    offscreenG.fillRect(0, 0, 100, 100);  
    offscreenG.setColor(Color.white);  
    offscreenG.fillRect(100, 0, 100, 100);  
    // Disegno la pedina  
    offscreenG.setColor(Color.red);  
    offscreenG.fillOval(xpos, 5, 90, 90);  
  
    g.drawImage(offscreenImg, 0, 0, this);  
}
```

Osservazioni sulle applet

Il metodo `showStatus()` permette di visualizzare informazioni sulla applet mediante il metodo `getAppletContext()`, che ritorna un oggetto di tipo `AppletContext` ed abilita la applet ad accedere ad alcune funzionalità del browser che la contiene.

```
getAppletContext().showStatus("Cambio il colore");
```

Per fornire informazioni associate alla applet

```
public String getAppletInfo() {  
    return "Applet stupida, Copyright 1996 Andrea Bianco";  
}
```

Osservazioni sulle applet

Per scambiare informazioni tra applet appartenenti alla stessa pagina HTML

- si assegna un nome ad ogni applet mediante il parametro NAME= del tag APPLET
- il metodo `getApplet()` permette di accedere ai metodi ed alle variabili di istanze delle altre applet.

```
<APPLET CODE="LaMiaApplet1" WIDTH=100 HEIGHT=150 NAME="Trasmettitore"></APPLET>
<APPLET CODE="LaMiaApplet2" WIDTH=100 HEIGHT=150 NAME="Ricevitore"></APPLET>
```

```
Applet receiver = getAppletContext().getApplet("Ricevitore");
receiver.update(text, value); // innesco l'update della applet ricevitore
```

Per caricare un documento HTML e farlo visualizzare dal browser

```
String url = "http://hp0t1c.polito.it/~bianco";
theURL = new URL(url);
getAppletContext().showDocument(theURL); // Apre documento nella stessa finestra
getAppletContext().showDocument(theURL, "_blank"); // Apre documento in nuova finestra
```