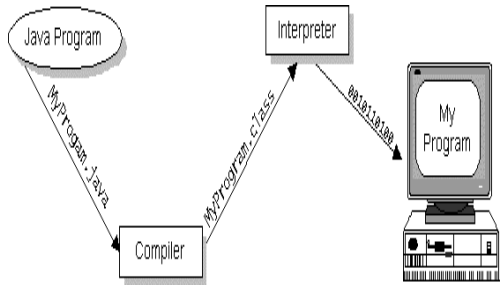
Java
Applet, animazioni ed altro.

Paolo Cignoni
cignoni@iei.pi.cnr.it
21/4/98

Introduzione

- Java is a high-level programming language that is all of the following:
- Architecture-neutral
- Object-oriented
- Portable
- Interpreted
- Multithreaded

Schema di base



Creazione di un Applet

- Creare un Java source file
- Compilare il sorgente
- Creare un html che includa l'applet
- Far girare l'applet

Solito esempio...

- Creare un Java source file di nome
- HelloWorld.java

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);
    }
}
```


Compilare ed eseguire

- Compilare:
  `javac HelloWorld.java`
- A partire da un .java si ottiene uno o più file .class;
- Per eseguire l'applet occorre includerlo in una pagina html:

```
<HTML>
<HEAD><TITLE> A Simple Program </TITLE></HEAD>
<BODY>
Here is the output of my program:
<APPLET CODE="HelloWorld.class" WIDTH=150
  HEIGHT=25>
</APPLET>
</BODY></HTML>
```

Due parole sull'applet

- Ogni applet è una sotto classe della classe Applet
- Ridefinisce vari metodi della classe, nel nostro caso abbiamo ridefinito paint()

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);
    }
}
```

7

Ciclo di vita di un applet

- Questi metodi sono invocati dal browser:
  - **`init`**() al caricamento dell'applet,
  - **`start`**() subito dopo la init() e ogni volta che l'applet riparte (senza essere ricaricato)
  - **`stop`**() quando l'utente lascia la pagina
  - **`destroy`**() prima di essere definitivamente scaricato, l'applet ha la possibilità di rilasciare tutte le risorse

8

Gestione degli eventi

- Schema di base:
- Gli eventi sono generati da *sorgenti*
- Uno o più *listener* possono registrarsi per essere notificati degli eventi di un particolare tipo da una particolare sorgente
- Le classi che implementano un interfaccia *listener* possono fare da gestori degli eventi

9

Ricetta per la Gestione Eventi

- Una classe si dichiara implementare un'interfaccia *listener*

```
public class MyClass extends SomeClass
                     implements ActionListener {
```

- Un'instanza della classe che si è dichiarata listener si registra come listener per un certo tipo di eventi per un dato componente

```
someComponent.addActionListener(instanceOfMyClass);
```

- Si implementano i metodi definiti dall'interfaccia listener

```
public void actionPerformed(ActionEvent e) {
   ...//code that reacts to the action...
   }
```

10

Esempio Beeper

L'applet fa da listener,

```
public class Beeper extends Applet
 implements ActionListener {
...
```

il bottone specifica che l'applet fa da actionlistener degli eventi prodotti dal bottone

```
//where initialization occurs:
   button.addActionListener(this);
...
 public void actionPerformed(ActionEvent e) { ...
//Make a beep sound... }}
```

11

Esempio Spot

- L'applet registra se stesso come mouselistener, e quindi processa **tutti** gli eventi mouse che lo riguardano

```
public class Spot extends Applet implements
             MouseListener {
...
addMouseListener(this);
...
public void mousePressed(MouseEvent event) {
   clickPoint = event.getPoint();
   repaint();
    }
public void mouseClicked(MouseEvent event) {}
public void mouseReleased(MouseEvent event) {}
public void mouseEntered(MouseEvent event) {}
public void mouseExited(MouseEvent event) {}
```

12

Applet e Animazioni

- Nei prossimi lucidi vedremo in dettaglio come fare (più o meno bene) piccole animazioni in java.
- Vedremo in particolare
  - Threads
  - Double buffering
  - Caricamento di immagini

13

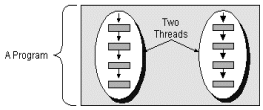
Il loop principale

- Ogni programma che fa un animazione deve aggiornare a intevalli regolari lo schermo
- Deve essere fatto con un thread separato! non dentro il metodo paint

14

Cenni sui Thread

- Un thread è un singolo flusso di controllo sequenziale all'interno di un programma



- In Java un thread è creato istanziando un oggetto della classe Thread

15

Ricetta base per gestire Threads

- La classe deve implementare runnable (o estendere Threads)

```
public class AnimatorApplet extends Applet
                            implements Runnable
```

- Creare e far partire il thread nel metodo start

```
if (animatorThread == null) {
    animatorThread = new Thread(this);
  }
animatorThread.start();
```

- Definire il comportamento del thread nel metodo run()

16

Double Buffering

- Una tecnica comune (si usa anche in Opengl) per migliorare la qualità delle animazione è quella di costruire l'immagine *offscreen* e copiarla sullo schermo solo alla fine di disegno:
- Evita problemi di *flickering*

17

Ricetta per il double buffering

- Creare un'immagine offscreen della giusta grandezza

```
Image offImage;
offImage = createImage(d.width, d.height);
```

- prendere un GraphicContext dell'immagine per poterci disegnare

```
Graphics offGraphics;
offGraphics = offImage.getGraphics();
```

- Disegnare sul GraphicContext così ottenuto e infine copiare l'immagine sullo schermo.

```
g.drawImage(offImage, 0, 0, this);
```

18

Caricare un po' di immagini

- Il metodo getimage carica un oggetto di tipo image che può poi essere disegnato su un Graphics context

```
images[i] = getImage(getCodeBase(),"pippo.jpg");
Graphics.drawImage(images[i], 0, 0, this);
```

- Nota: il metodo getimage() ritorna immediatamente, senza caricare né controllare se l'immagine esiste. Si rimanda a quando l'immagine viene usata

19

Controllare il caricamento

- Per controllare se le immagini sono già state caricate si può utilizzare la classe MediaTracker
- Ogni immagine da controllare viene aggiunta all'istanza dell'oggetto Mediatracker;

```
images[i] = getImage(getCodeBase(),
            "aloopingicon"+i+".jpg");
tracker.addImage(images[i], 1);
```

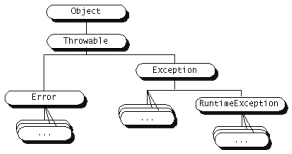
- Per controllare se le immagini sono state caricate

```
tracker.checkAll(true)
```

20

Eccezioni

- **`try { ... }`**
  - **Specifica un blocco dove possono capitare eccezioni**
- **`catch (someThrowable object ) { ... }`**
  - **Specifica il comportamento per un eccezione di una certa classe**
- **`finally { ... }`**
  - **Specifica codice di pulizia finale (rilascio risorse ecc.)**



Eccezioni

```
public void writeList() {
   PrintWriter out = null;
   try {
        System.out.println("Entering try statement");
        out = new PrintWriter( new FileWriter("OutFile.txt"));
        for (int i = 0; i < size; i++)
              out.println("Value at: " + i + " = " +
                          victor.elementAt(i));
        }
   catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("Caught ArrayIndexOutOfBoundsException: "
                           + e.getMessage()); }
   catch (IOException e) {
        System.err.println("Caught IOException: " + e.getMessage());
        }
   finally {
        if (out != null) {
              System.out.println("Closing PrintWriter");
              out.close();
        } else {
              System.out.println("PrintWriter not open");
        }
   }
}
```

22