RETE E APPLET

Dott. Denis Ferraretti

denis.ferraretti@unife.it

lezioni

- Lunedì 1 marzo 2010 ore 11.00 - 13.30
- Mercoledì 3 marzo 2010 ore 11.00 - 13.30

Tutte le lezioni sono nel
laboratorio di Informatica Grande.

Java Introduzione alla rete

- Gli **indirizzi internet** identificano in modo univoco ogni computer sulla rete. Utilizzano 4 byte (dot IP notation) come per esempio 220.210.34.7
- Il **Domain Name System** ci permette un facile modo per ricordare questo schema. I server DNS traducono il nome del dominio es "amazon.com" nella sua lista di IP.
- I **Servers** sono quei computer con delle risorse (come ad esempio stampanti e dischi) che devono essere condivisi. **Clients** sono quelle entità che vogliono usare queste risorse.
  - I Servers "ascoltano" le loro porte sulle **socket** (connessioni) aspettando che un client si connetta con una richiesta. Sono **multithreaded** per permettere a molte richieste di essere gestite simultaneamente.

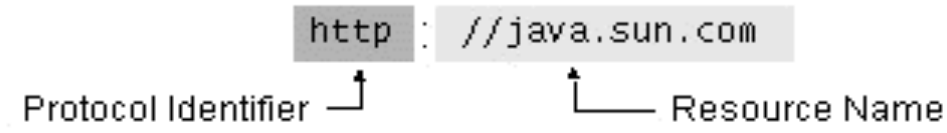
Java Networking

L'architettura Java è **network-ready** e comprende package che permettono all'utente di gestire file, richiedere risorse, e anche richiamare metodi tramite la rete.

- Package **`java.net`**
  Concetti (classi) fondamentali per TCP (connessione):
  - **`Socket, ServerSocket, URL, URLConnection`**
- Package **`java.rmi`**
  Uso di *oggetti remoti*. Questi oggetti verranno utilizzati soprattutto nei corsi avanzati del curriculum. Per ora anticipiamo solamente il funzionamento per illustrarne le potenzialità.

Java URL e connessioni

- La classe **`URL`** (Uniform Resource Locator) cattura il concetto di **puntatore a risorsa** sul World Wide Web.
- Una risorsa può essere qualcosa di semplice come un file o una directory, o un riferimento a qualche oggetto più complicato, come una query a un database o a un motore di ricerca:
  - **`http://localhost/index.html`**
  - **`file:///autoexec.bat`**
  - **`...`**
- Come si crea un URL per identificare una risorsa:



  - L'identificativo del protocollo (Protocol Identifier)
  - Il nome della risorsa (Resource Name)

Java URL e connessioni

- Il **Protocol Identifier** indica il nome del protocollo utilizzato per trovare una risorsa. L'esempio usa l'Hypertext Transfer Protocol (HTTP) che è largamente utilizzato per trasferire documenti ipertestuali. Sono supportati protocolli come quello per i file, per l'ftp, e altri ancora.
- Il **Resource Name** è l'indirizzo completo della risorsa, il formato utilizzato dipende dal protocollo. Per il protocollo HTTP, il resource name, è costituito da queste componenti:
  - *Host Name:* Il nome della macchina remota sulla quale si trova la risorsa.
  - *Filename:* Il pathname del file sulla macchina.
  - *Port Number:* Il numero della porta alla quale connettersi (tipicamente opzionale).
  - *Reference:* Un riferimento a uno specifico tag all'interno della risorsa, normalmente una locazione all'interno di un file (tipicamente opzionale).

Java URL e connessioni

- Per esempio, supponiamo di costruire un pannello browser (per altro esistente in swing: **EditorPane**) simile a un vero browser che ci permetta di utilizzare un protocollo scelto, un hostname, un numero di porta, e un nome di file. Si può costruire un URL dai componenti del pannello. Il primo costruttore crea un URL da un protocollo, host e nome di file.

```
new URL("http", "www.gamelan.com", "/pages/Gamelan.net.html");
```

Questo è equivalente a:

```
new URL("http://www.gamelan.com/pages/Gamelan.net.html");
```

- Il primo argomento è il protocollo, il secondo l'hostname, e l'ultimo il pathname del file. Notare che il filename contiene un forward slash all'inizio. Questo indica che il filename è specificato dalla directory root dell'host.
- Il costruttore finale dell'URL aggiunge il numero di porta alla lista di argomenti usati nel precedente costruttore.

```
new URL("http", "www.gamelan.com", 80, "pages/Gamelan.net.html");
```

Questo crea un oggetto URL equivalente a:

```
http://www.gamelan.com:80/pages/Gamelan.net.html
```


Java URL e connessioni

- Per aprire una connessione, si invoca sull'oggetto **URL** il metodo **openConnection():**

```
URLConnection c = url.openConnection();
```

- Il risultato è un oggetto **URLConnection,** che rappresenta una "connessione aperta"
  - in pratica, così facendo si è stabilito un canale di comunicazione verso l'indirizzo richiesto

- Per connettersi tramite tale connessione:

```
c.connect();
```

Java URL e connessioni

Per comunicare si recuperano dalla connessione i due stream (di ingresso e di uscita) a essa associati, tramite i metodi:

- public **`InputStream getInputStream()`**
  - restituisce lo stream di input da cui leggere i dati (byte) che giungono dall'altra parte

- public **`OutputStream getOutputStream()`**
  - restituisce lo stream di output su cui scrivere i dati (byte) da inviare all'altra parte

Poi, su questi stream si legge/scrive come su qualunque altro stream di byte.

Java URL e connessioni

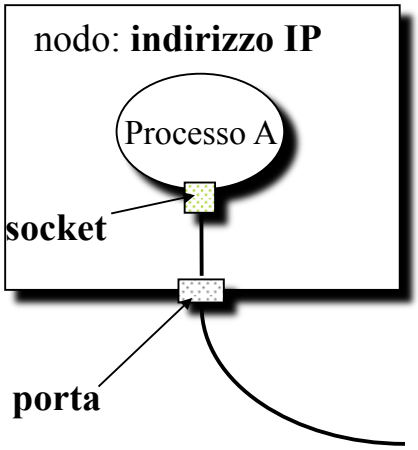
Programma per connettersi all'URL dato, e visualizzare il contenuto dello stream, supponendo che esso contenga testo.

```
import java.io.*;
import java.net.*;
public class EsempioURL {
 public static void main(String args[]) {
   URL u = null;
   try { u = new URL(args[0]); }
   catch (MalformedURLException e) {
   System.err.println("URL errato: " + u);
   }
   URLConnection c = null;
   try {
    c = u.openConnection(); c.connect();
    InputStreamReader is = new InputStreamReader(c.getInputStream());
    BufferedReader r = new BufferedReader(is);
    String line = r.readLine();
     while(line != null) {
       System.out.println(line);
       line = r.readLine();
     }
   } catch (IOException e) { System.err.println(e);}
 }
}
```

Possiamo provarlo sia coi files che con gli indirizzi internet.
NOTA: Ricordarsi il protocollo!!

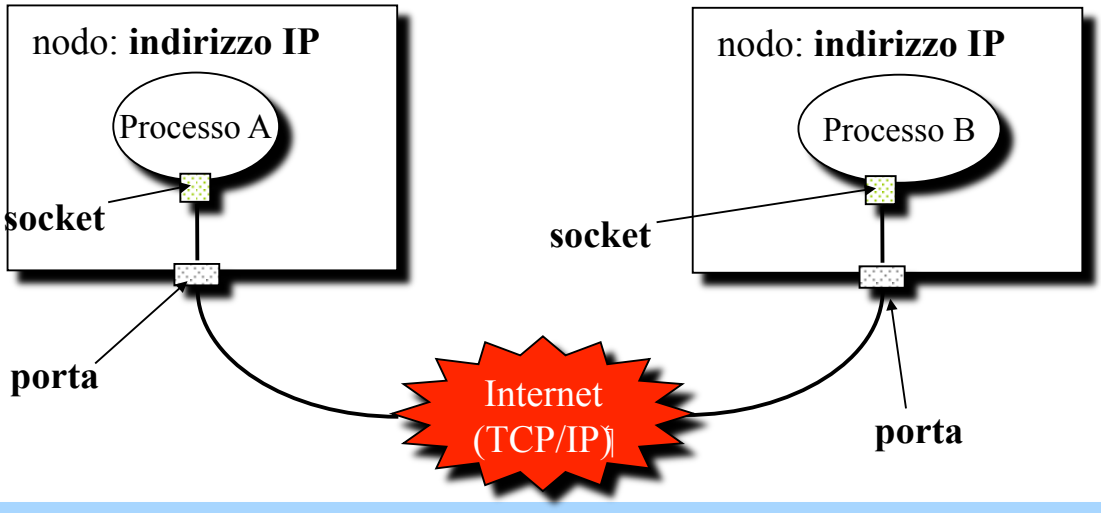
Java Socket

- Una **socket** è un terminale di comunicazione fra due macchine, concettualmente una porta, una “presa” verso la rete
- Collega un certo processo al mondo esterno
- È identificata da un **numero** (port number) unico su una data macchina (nodo o host)
- Ogni nodo è identificato dal suo **indirizzo IP**



Java Socket

- La socket permette a due **processi**, residenti sulla stessa macchina o su macchine anche molto distanti, di **comunicare** fra loro.
- Modello **cliente / servitore**:
  - il servitore deve stare in attesa di possibili comunicazioni in arrivo
  - i clienti (anche più di uno), quando vogliono, parlano con il servitore



Java Socket

- Esistono fondamentalmente due tipi di socket: **socket stream** e **socket datagram**

- Le **socket stream**
  - sono affidabili, stabiliscono una connessione stabile e bidirezionale con l'altra parte, che dura finché non si decide di chiuderla
- Le **socket datagram**
  - non sono affidabili, non stabiliscono una connessione stabile: la comunicazione è unidirezionale come un telegramma
  - sono meno "costose"

Java Socket Stream

Schema di funzionamento:

- Il servitore crea la sua **ServerSocket** con un numero noto, e si mette in attesa

- Un cliente, quando vuole comunicare col servitore, crea la sua **Socket** specificando con chi vuole parlare
  - nome dell'**host** dove risiede il servitore
  - numero di **porta** su cui è in ascolto il servitore

- Il servitore accetta la richiesta del cliente: con ciò si crea una **Socket** già collegata al cliente, tramite cui i due comunicano.

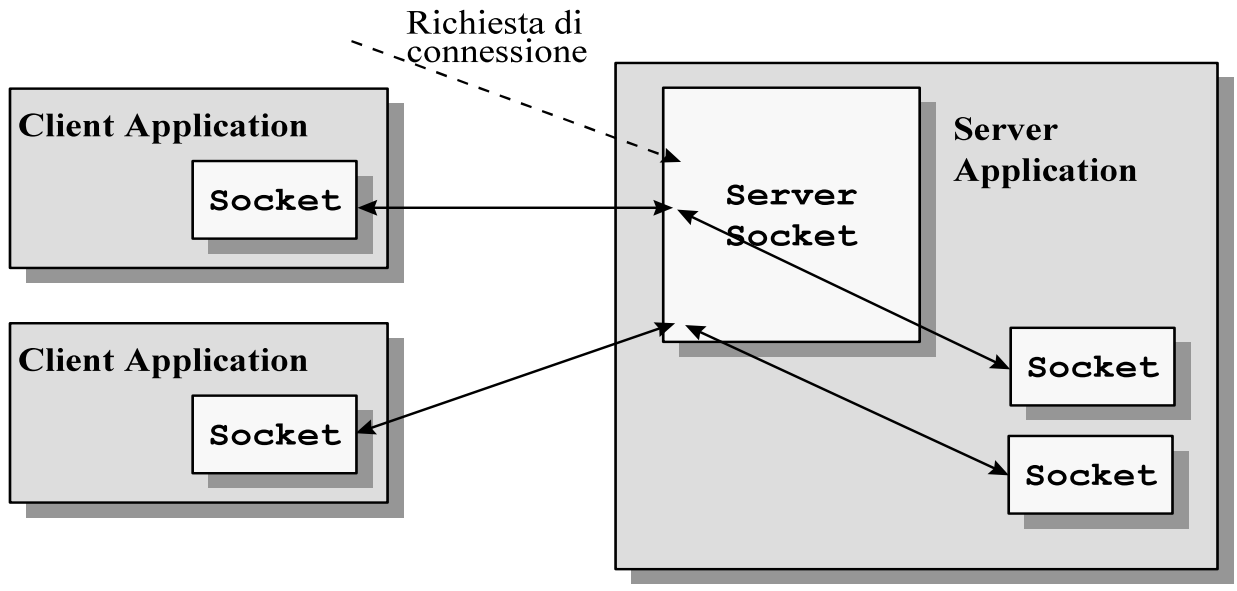
Java Socket Stream

- Alla Socket sono associati **due stream**:
  - uno dal cliente verso il servitore
  - uno dal servitore verso il cliente

- La **comunicazione** cliente/servitore è **bidirezionale**
  - i ruoli “cliente” e “servitore” sono tali solo nella fase iniziale, quando si instaura la connessione
  - una volta connessi, i due processi possono parlarsi reciprocamente “alla pari”

Java Socket Stream

Schema di funzionamento del **Socket Stream** con server.



Java Socket servizi

Moltissimi servizi di uso comune sono standardizzati e disponibili su macchine sia **Unix** sia (non sempre) Windows

- **echo** (porta 7): rimanda indietro tutto quello che gli si invia, come un'eco
- **daytime** (porta 13): restituisce data e ora
- **telnet** (porta 23): consente il collegamento remoto, da altro terminale (solo Unix)
- **smtp** (porta 25): consente la spedizione della posta (se abilitata)

Java Socket Stream

- Esempio di come interrogare un servizio daytime:

```
import java.net.*;
import java.io.*;
public class EsempioNet {
 public static void main(String args[]){
   Socket s = null;
   try {
    s = new Socket(…,13);
    InputStream is = s.getInputStream();
    InputStreamReader ir = new InputStreamReader(is);
    BufferedReader r = new BufferedReader(ir);
    String line = r.readLine();
    System.out.println(line);
    s.close();
   } catch (UnknownHostException e){
    System.err.println("Host unknown");
   } catch (Exception e){
    System.err.println(e);
   }
  }
}
```

Risultato:

```
Fri Feb 18 16:44:46 2000
```

Java Applet

- Una applet (“applicazioncina”) è una applicazione non autonoma, ma pensata per far parte di una pagina Internet
- Porta dinamicità alle pagine statiche
- Viene eseguita dal browser, che quindi deve incorporare un interprete Java

Caratteristiche:

Un’applet non è un’applicazione autonoma, quindi non deve creare un frame principale, perché usa la finestra del browser che la ospita; non ha un main, perché la sua vita è dipendente dalla pagina in cui è visualizzata, è organizzata intorno a 4 metodi standard:

- `init()`, che gioca il ruolo del costruttore
- `start()` e `stop()`, chiamati dal browser ogni volta che occorre avviare/ fermare l'applet
- `destroy()`, invocato quando il browser viene chiuso.

Java Applet

Esistono due versioni di Applet:

- la classe Applet dell'AWT standard (da Java 1.0 in poi)
- la classe JApplet di Swing (da Java 1.1.6 in poi)

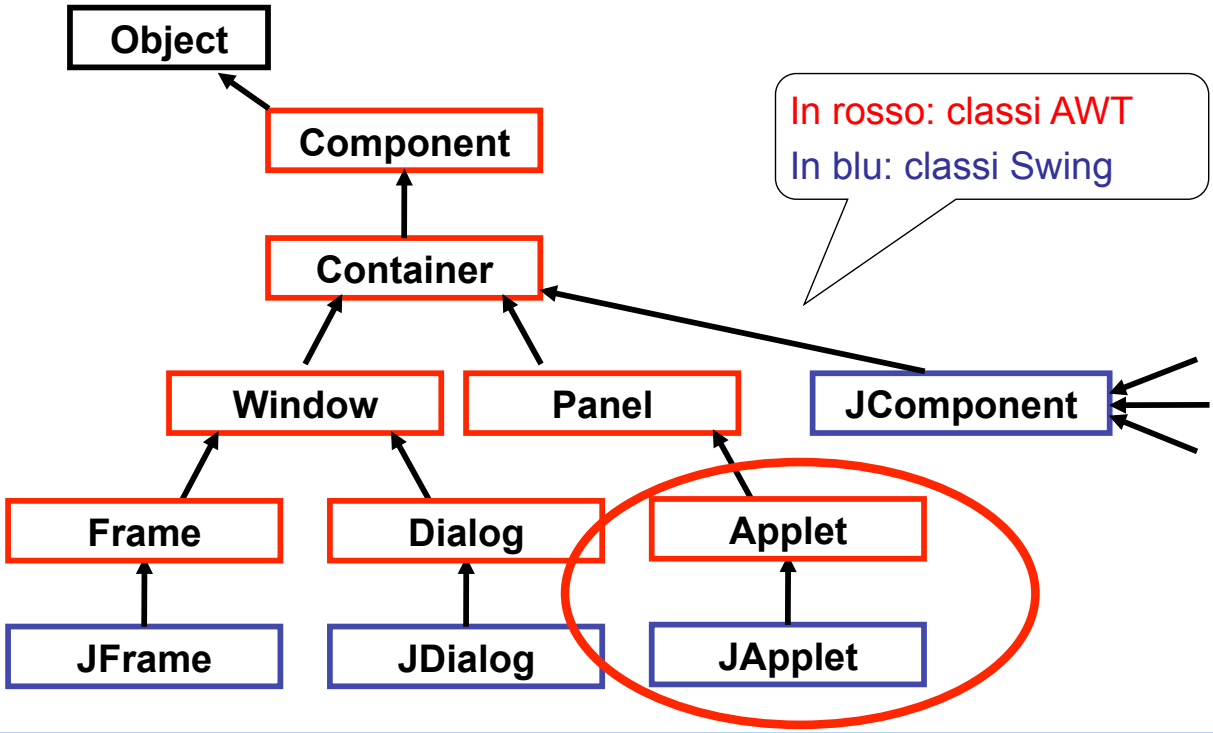
Attenzione:

- se si usano componenti **Swing**, occorre necessariamente usare **JApplet**
- una Applet con componenti Swing non viene disegnata correttamente

Infatti, Applet deriva direttamente da Panel, quindi è essa stessa un pannello.

Java Applet Gerarchia

- Lo schema gerarchico delle classi:



Java Applet Costruzione

Per costruire un'Applet (con Swing), occorre:

- creare una propria classe che estenda JApplet
- implementare i metodi opportuni (tipicamente, almeno **`init()`**)
- compilare l'applet nel modo standard
- preparare una pagina HTML che la carichi
- eseguire l'applet tramite l'AppletViewer

  - oppure -
- aprire tale pagina con un browser

Java Applet Costruzione

Un'applet è organizzata intorno a 4 metodi:

- **`init()`**, che viene chiamato dal browser quando lancia l'applet per la prima volta fa le veci di un costruttore, legge i parametri, etc

- **`start()`**, che viene chiamato dal browser ogni volta che l'applet deve essere riavviata tipicamente riavvia un'animazione o un **thread** non occorre implementarlo se non ci sono **animazioni** o thread da riattivare.

- **`stop()`**, che viene chiamato dal browser ogni volta che l'applet deve essere fermata tipicamente ferma un'animazione o un thread non occorre implementarlo se non ci sono animazioni o thread da fermare

- **`destroy()`**, che viene chiamato dal browser quando il browser stesso si chiude. Risulta utile in casi particolari, per liberare i contesti grafici (di norma non occorre implementarlo).

Java Applet Esempio

- Proviamo a vedere il codice di un semplicissimo esempio:

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

public class MyApplet extends JApplet {

  Font f = new Font("Times", Font.BOLD, 36);

  public void paint(Graphics g) {
   g.setFont(f);
   g.setColor(Color.RED);
   g.drawString("Ciao mondo!", 100, 50);
  }
}
```

Questo esempio non implementa nessuno dei metodi che abbiamo nominato.

Java Applet Esempio

Questo è il codice HTML da scrivere in un file di testo (opportunamente rinominato .html) per vedere il risultato dell'esempio precedente.

```
<HTML><HEAD>
<TITLE> Applet Hello World </TITLE>
</HEAD><BODY>
<OBJECT codetype="application/java"
   classid="java: MyApplet.class" width="100" height="150">
</OBJECT>
</BODY>
</HTML>
```

Una alternativa è utilizzare il seguente tag:

```
<APPLET CODE="MyApplet.class" WIDTH="500" HEIGHT="100">
</APPLET>
```

Lo utilizzeremo ora per passare parametri.

Java Applet Esempio

- Il file HTML può specificare parametri da passare all'applet, nella forma:

```
<HTML><HEAD>
<TITLE> Applet Hello World </TITLE>
</HEAD><BODY>
<APPLET CODE="MyApplet.class"
WIDTH="500" HEIGHT="100">
<PARAM name="testo" value="Ciao mondo!!">
</APPLET>
</BODY>
</HTML>
```

- L'applet può recuperarli con il metodo **`getParameter (nomeparametro)`**, che restituisce una **`String`**

Java Applet con parametri

- Modifichiamo il codice dell'esempio per fargli stampare un testo passato all'applet come parametro:

```
public class MyApplet extends JApplet {
   public MyApplet() { }

   public void init() {
    super.init();
    this.testo = this.getParameter("testo");
   }

   public void paint(Graphics g) {
    if(this.testo == null) this.testo = "Nessun parametro!";
    g.setColor(Color.BLUE);
    g.setFont(new Font("Verdana", Font.PLAIN, 36));
    g.drawString(this.testo, 50, 50);
   }

   private String testo;
}
```


Java Applet e Applicazioni

Un'applet può essere costruita in modo da poter funzionare anche come applicazione: basta aggiungere un main che svolga le funzioni normalmente svolte dal browser:

- creare il frame, dimensionarlo con **`setSize()`** e fissare il titolo con **`setTitle()`**
- impostare un **`WindowListener`** per gestire la chiusura della finestra (JFrame come facevamo con **`setDefaultCloseOperation(int)`**)
- invocare il metodo **`init()`**
- avviare l'applet chiamando **`start()`**

Nota:

- non occorre chiamare il metodo **`stop()`**, perché un'applicazione termina quando il suo frame viene chiuso.

Java Applet e Applicazioni

UN APPROCCIO ALTERNATIVO

- Non toccare l'applet già fatta
- Ma sfruttare l'ereditarietà per definire una nuova classe:
  - che erediti dall'applet che interessa
  - e contenga il main opportuno

Vantaggi:

- è molto semplice
- non tocca neanche un file dell'applet originale
- si può usare sempre, per qualunque applet

Java Applet Esempio

- Proviamo a modificare il codice della calcolatrice in modo da trasformarla in un applet. (modifichiamo Calculator.java)

```
public class CalculatorApplet extends JApplet {
  public CalculatorApplet() {}

    public void init() {
   super.init();
   this.add(this.createUI());
  }
  … // Vecchio codice della classe
```

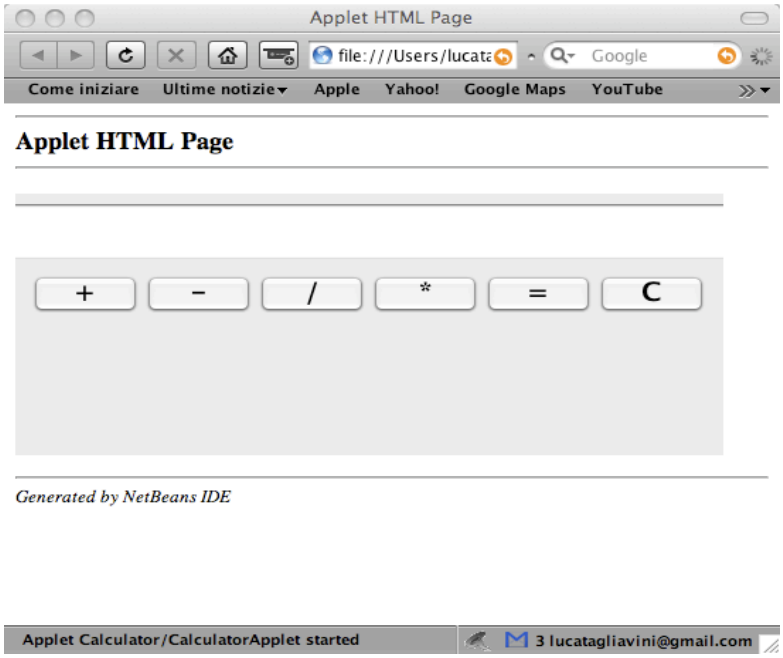
- **NOTA:** Questa volta **non serve** il metodo main

- Questo è il tag da modificare sul file html:

```
<APPLET codebase="classes" code="CalculatorApplet.class"
width="500" height="200">
</APPLET>
```

Java Applet Esempio

- Il risultato è il seguente visto dentro un browser:



Applet e Sicurezza

Un'applet **non può** fare tutto quello che fa una applicazione.

- Poiché può essere scaricata dalla rete, sarebbe troppo pericoloso permettere a un'applet di fare qualunque cosa.
- Un'applet è costretta a rispettare un ben preciso modello di sicurezza ("sandbox")
  - è eseguita in una "scatola" da cui non può uscire
  - non può contaminare (o spiare) i dati del computer dell'utente

Un'applet di norma **non può**:

- accedere al file system locale (neppure per leggere un file)
- eseguire un altro programma
- ottenere informazioni sull'utente
- connettersi via rete a un computer diverso da quello da cui è stata scaricata
- caricare la libreria Java, chiamare `System.exit()`

Questi vincoli non si applicano all'appletviewer

Applet e Sicurezza

Un'applet, inoltre:

- può aprire un'altra finestra, ma in essa compare automaticamente un avviso ("Warning: Applet window")

Problema:

- in molte situazioni, questi vincoli sono troppo rigidi
- rischierebbero di rendere impossibile la costruzioni di applet utili.

Soluzione:

- Attraverso tecnologie di cifratura, un'applet può essere firmata, ossia a essa può essere allegato un certificato che ne garantisce l'origine.
- Alle applet firmate, cui si attribuisce mag-giore fiducia, l'utente può consentire di svolgere alcune o tutte le operazioni sottoposte a vincolo.

Applet e Sicurezza

- Ogni browser può essere configurato: