Applets

- Applet = programma Java che può essere invocato all'interno di un file .html ed eseguito da
  - un browser web, es. `C:\Files> explorer` *MioFile.html*
  - un appletviewer, es. `C:\Files> appletviewer` *MioFile.html*
- Differisce da un'applicazione stand-alone nel modo in cui viene eseguito
  - un'applicazione stand-alone inizia con l'esecuzione del metodo `main()`
  - il ciclo di vita di un applet è più complesso: il browser carica una URL, carica il documento HTML, carica le classi applet incluse nel documento, esegue gli applet.

Restrizioni per la sicurezza

- La maggior parte dei browser (es.Nestcape Navigator) impediscono
  - l'esecuzione run-time di altri programmi
  - input-output di file
  - chiamata a metodi nativi
  - tentativi di aprire socket in un sistema diverso da quello che ha fornito l'applet
- JDK 1.2. permette di definire un "dominio di protezione" in cui un applet può venire eseguito: si può consentire che applet che provengano da un certo dominio abbiano privilegi speciali

Sicurezza

NL = Netscape caricando una URL **AV = Applet viewer**
NF = Netscape caricando un file locale **JA = Applicazione stand-alone**

	NL	NF	AV	JA
lettura file locali	no	no	si	si
scrittura file locali	no	no	si	si
avere info su un file	no	no	si	si
cancellare un file	no	no	no	si
far girare un altro programma	no	no	si	si
connettersi alla rete su un server	si	si	si	si
connttersi alla rete su un altro host	no	si	si	si
caricare la libreria Java	no	si	si	si
chiamare exit	no	no	si	si

Il primo esempio di applet

- ciao. java

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

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

- ciao.html

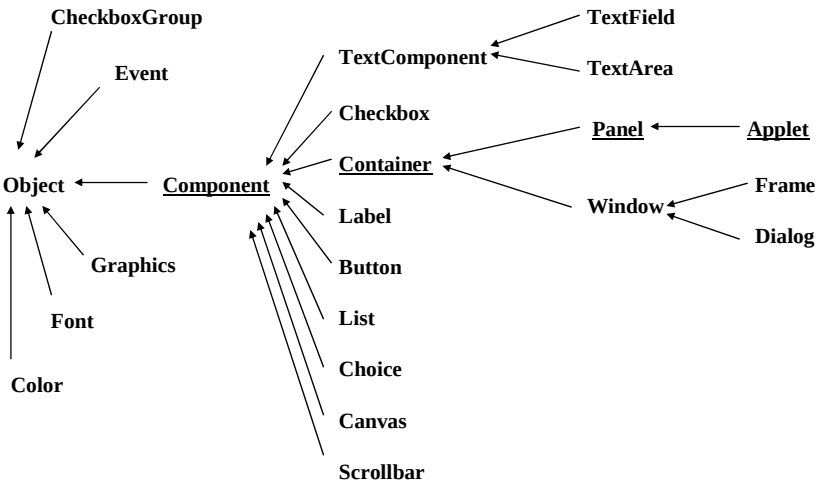
```
<html>
<applet code="Ciao.class" width=275 height=200> </applet>
</html>
```

- La classe `Ciao` deve essere pubblica, ed il suo nome deve essere uguale al nome del file che la contiene.
- Deve essere una sottoclasse di `java.applet.Applet`

La gerarchia della classe Applet

- La classe `Applet` è sottoclasse di `Panel`, nella libreria Awt.
  Questo implica in particolare che:
  - `Applet` ha come layout manager di default il Flow Layout Manager, che dispone gli elementi grafici da sinistra a destra, con allineamento centrale
  - `Applet` eredita le variabili ed i metodi delle sue superclassi `Component`, `Container` e `Panel`

La gerarchia della classe Applet …



Metodi base degli applets

- **`init()`**
  - Quando si apre un documento con un applet, viene invocato automaticamente il metodo `init()` per l'inizializzazione dell'applet.
  - Può essere usato per inizializzare le variabili.
  - Questo metodo è chiamato solo una volta per un applet
- **`start()`**
  - Quando un documento con un applet viene aperto, viene chiamato il metodo `start()` automaticamente dopo `init()`, per iniziare l'applet.
  - Questo metodo, ad esempio, è invocato ogni qual volta l'applet viene "rivisitato".

Metodi base degli applets

- **`stop()`**
  - Viene invocato automaticamente quando l'utente si sposta fuori dalla pagina su cui è situato l'applet
  - Serve per bloccare attività che possono rallentare il sistema quando l'utente non sta utilizzandole (es. animazioni)
  - I metodi `start` e `stop` formano una coppia: `start` attiva un comportamento, `stop` lo disattiva.
- **`destroy()`**
  - Viene invocato quando l'applet viene dismesso (ad es. quando viene chiuso il browser), e in questo caso viene prima invocato il metodo `stop()`.
  - Tutte le risorse legate all'applet vengono rilasciate

Metodi base degli applets

- **`paint()`**
  - Questo metodo è invocato direttamente dal browser dopo che `init()` è stato completamente eseguito e dopo che `start()` abbia iniziato l'esecuzione.
  - E' invocato ogni qual volta l'applet deve essere ridipinto
- **`repaint()`**
  - invoca il metodo `update()` al più presto (non va overridden)
- **`update()`**
  - permette di "ri-dipingere" l'applet, e può essere riscritto.
  - Per default, "pulisce" lo sfondo e chiama il metodo `paint()`

AWT Thread

- La rappresentazione del display è un thread indipendente, che resta in stato di attesa
- L'aggiornamento del display deve essere predisposto per essere invocato in tre casi
  - la "esposizione" iniziale
  - ogniqualvolta la schermata possa essere danneggiata
  - ogniqualvolta ci sia nuova informazione che deve essere mostrata

La sintassi del tag <applet>

```
<applet
    [archive=ListaArchivio]          // classi o risorse che saranno preloaded. Si puo' indicare una lista
                                     // di file JAR (Java Archive) dalla quale estrarre l'applet
    code=MioFile.class               // nome del file che contiene il compilato, relativo alla stessa URL
    width=pixels heigh=pixels        // dimensioni iniziali del display dell'applet
    [codebase=codebaseURL]           // directory che contiene il codice dell'applet
    [alt=testoAlternativo]           // se il browser legge l'applet tag ma non può eseguire l'applet
    [name=nomeIstanzaApplet]         // permette di identificare diversi applet nella stessa pagina
                                     // ad esempio, permette ad applet di comunicare tra loro…
    [align=allineamento]             // left right top texttop middle baseline bottom
    [vspace=pixels] [hspace=pixels]
>
[<param name=attributo1 value=valore>]      // valori specificati dall'esterno
[<param name=attributo2 value=valore>]      // ci sarà un getParameter() nell'applet
    . . .
</applet>
```


I parametri nel file HTML

```
<html>
<title>Etichetta</title>
<body>
<applet code = "Etichetta.class"
        width=500 height=100
        name=primo
        align = top
        vspace=30
>
<param name = testo      value ="Ciao ciao ciao ciao MARE !!!!">
<param name = carattere  value="Courier">
<param name = grossezza  value="bold">
<param name = dimens     value ="18">
</applet>

<p><hr><p>

<applet code = "Etichetta.class"
        width=500 height=100
        name=secondo
        align = right
        vspace=30
>
<param name = testo value ="Sapore di SALE, Sapore di MARE...">
<param name = stile value="italic">
</applet>
</body>
</html>
```

Una classe con parametri esterni

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

public class Etichetta extends Applet {
        private static final String LABEL     = "Testo di default!";
        private static final String FONTNAME  = "TimesRoman";
        private static final int    FONTSIZE  = 24;
        private static final int    FONTSTYLE = Font.PLAIN;

        private String  textValue     = LABEL;
        private Font    textFont      = null;
        private Color   textColor     = Color.red;

 public void init(){
        String param = getParameter("testo");

        if (param != null) { textValue = param;}

        String fontName  = FONTNAME;
        int    fontStyle = FONTSTYLE;
        int    fontSize  = FONTSIZE;

        param = getParameter("carattere");
        if (param != null) {fontName = param;}
```



```
    param =getParameter("stile");
       if (param != null &&
         (param.equals("italic") || param.equals("ITALIC"))) {
         fontStyle |= Font.ITALIC;
    }

    param = getParameter("grossezza");
    if (param != null &&
         (param.equals("bold") || param.equals("BOLD"))) {
         fontStyle |= Font.BOLD;
    }

    param = getParameter("dimens");
    if (param != null) { fontSize = Integer.parseInt(param); }

    textFont = new java.awt.Font(fontName, fontStyle, fontSize);
 }

 public void paint(Graphics g) {
    g.setFont(textFont);
    g.setColor(textColor);
    g.drawString(textValue, 50, 50);
 }

}
```

Applet con campi di testo

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class Addition extends Applet implements ActionListener{
   Label prompt;      // etichetta (1 linea, in sola lettura)
   TextField input;   // linea di scrittura che riceve input da tastiera
   Button reset;      // pulsante per annullare tutto
   int number;        // memorizza il valore in input
   int sum;           // memorizza la somma

   public void init(){
    prompt = new Label( "Scrivi un intero e vai a capo:" );
      input = new TextField( 10 );
      input.addActionListener(this);
      reset = new Button("Reset");
      reset.addActionListener(this);
      add(prompt); add(input); add(reset);
      sum = 0;
   }

   public void paint(Graphics g){
      g.drawString("La somma dei numeri inseriti = " + sum, 75, 105);
      g.drawString("(reset automatico in caso di overflow)", 75, 125);
   }
```



```
    public void actionPerformed(ActionEvent e) {
      try{
         try{
           if ( e.getSource()== input){       // oppure instanceof TextField
             number = Integer.parseInt( input.getText() );
             sum += number;
           }
           else if (e.getSource()== reset) // oppure e.getActionCommand() == "Reset"
             sum =0;
         }
         finally{
           input.setText( "" );
           repaint();
         }
      }
      catch (NumberFormatException ex){
         sum= 0;
      }
   }
 }
```

- Si osservi l'uso combinato di `try-catch` e `try-finally`
- Si osservi la chiamata del metodo `getSource` (o del metodo `getActionCommand`) per selezionare gli eventi

Applet con campi di testo

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class Comparison extends Applet implements ActionListener{
   Label     prompt1, prompt2;
   TextField input1, input2;
   int       num1, num2;

   public void init(){
     prompt1 = new Label("Un intero, grazie");
     input1 = new TextField("0",10);
     input1.addActionListener(this);
     prompt2 = new Label("Un altro, grazie");
     input2 = new TextField("0",10);
     input2.addActionListener(this);
     add( prompt1 ); add( input1 ); add( prompt2 ); add( input2 );
   }

   public void paint(Graphics g){
      g.drawString( "confronto dei valori:", 70, 75 );
      if ( num1 == num2 ) g.drawString(num1 + "=" + num2,100,105);
      if ( num1 < num2 )  g.drawString(num1 + "<" + num2,100,105);
      if ( num1 > num2 )  g.drawString(num1 + ">" + num2,100,105);
   }

   public void actionPerformed(ActionEvent e){
         num1 = Integer.parseInt(input1.getText());
         num2 = Integer.parseInt(input2.getText());
         repaint();
   }
}
```


Password...

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class AccessoPrivato extends Applet implements ActionListener {
   private TextField scritta, campo, risposta;
   private String password;

   public void init(){
      password = "AbracadabrA";         // fissa la password
      scritta = new TextField( "Scrivi la password: " );
      scritta.setEditable( false );
      campo = new TextField( 12 );
      campo.setEchoChar( '*' );         // caratteri di maschera
      campo.addActionListener(this);

      risposta = new TextField( 30 );
      risposta.setEditable( false );
      add( scritta ); add( campo ); add( risposta );
   }

   public void actionPerformed( ActionEvent e){
      if ( e.getSource() instanceof TextField )
         if ( e.getSource() == campo )
            if ( campo.getText().equals(password))
               risposta.setText( "Accesso Concesso" );
            else
               risposta.setText( "Password non valida. Accesso Negato" );
   }
}
```

Input/Output

- Come in C e C++ l'input/output è supportato da una libreria, non dal linguaggio: `java.io`
- La sicurezza del linguaggio impone che non si possa verificare un crash del programma dovuto al passaggio di un parametro di tipo errato ad un metodo di I/O
- Stream = sequenza/flusso di bytes in input o in output
- Classi astratte `InputStream`,`OutputStream` estese per lavorare su files, su dati strutturati ecc.
- Classi "wrapper" (aggiungono funzionalita' alla classe "wrapped") per bufferizzare, convertire ecc.

`InputStream`: metodi astratti

- Metodi di lettura
  - `int read()`
    restituisce un byte letto oppure -1, che indica end-of-file
  - `int read(byte[])`
    inserisce nell'array i bytes letti e restituisce il numero di bytes letti
  - `int read(byte[], int, int)`
    idem, ma con l'indicazione degli indici del subarray da riempire
- altri metodi
  - `void close()` chiude il flusso
  - `int available()` numero di bytes immediatamente disponibili
  - `skip(long)` scarta il numero specificato di caratteri dallo stream

`OutputStream`: metodi astratti

- Metodi di scrittura
  - `void write(byte)`
  - `void write(byte[])`
  - `void write(byte[], int, int)`
    analoghi ai corrispondenti metodi di lettura
- altri metodi
  - `void close()` chiude il flusso
  - `void flush()` forza la scrittura dei bytes bufferizzati

`InputStream` - `OutputStream`

- Sono classi astratte: non si possono creare oggetti di queste classi, ma si possono avere metodi che restituiscono valori di questi tipi.
- Permettono di leggere/scrivere solo singoli bytes o array di bytes.
- Non possono gestire stringhe, che in Java sono sequenze di caratteri Unicode, né numeri! Per questo ci si serve di opportune sottoclassi.

Copia di un file (bytes)

```
// Copia di un file per array di bytes. Le classi FileInputStream e
// FileOutputStream supportano solo la lettura e scrittura di array di bytes
// Il metodo int read(byte[] b) della classe InputStream restituisce
// il numero di bytes letti, e -1 alla fine dello stream.

import java.io.*;

public class CopyFile {
   public static void main(String[] args) {
      byte[] bytes = new byte[128];
      if (args.length < 2) {
          System.err.println("Uso: java CopyFile <src> <dest>");
          return;
      }
      try {
          InputStream istream  = new FileInputStream(args[0]);
          OutputStream ostream = new FileOutputStream(args[1]);
          int count = 0;
          while ((count = istream.read(bytes)) != -1) {
                    ostream.write(bytes, 0, count);
          }
          istream.close(); ostream.close();
      }
      catch (IOException e) {
          System.err.println(e); return;
      }
   }
}
```


Copia di un file di testo

```
// Copia di un file di testo. Il metodo readLine() di DataInputStream restituisce
// un valore di tipo String. Il metodo println(String line) è della classe PrintStream

import java.io.*;

public class CopyTextFile {
   public static void main(String[] args){
     if (args.length < 2) {
          System.err.println("Uso: java CopyTextFile <src> <dest>");
          return;
     }
     try {
          DataInputStream istream =
              new DataInputStream(new FileInputStream(args[0]));
          PrintStream ostream =
              new PrintStream(new FileOutputStream(args[1]));
          String line;
          while ((line = istream.readLine()) != null) {
             ostream.println(line);
          }
          istream.close(); ostream.close();
     }
     catch (IOException e) {
          System.err.println(e); return;
     }
   }
}
```

Copia file di testo (ctd.)

- In realtà nelle ultime versioni di Java sono state definite nuove classi per la gestione dei file di testo:
  - `DataInputStream` diventa `BufferedReader`
  - `FileInputStream` diventa `FileReader`
  - `PrintStream` diventa `PrintWriter`
- Se non si utilizzano queste nuove classi il compilatore darà un warning.

Files di dati tipati

Scrivere e leggere dati in forma tabellare, ad es:

```
19.99   12      T-shirt
9.99    8       Mug
15.99   13      Doll
3.99    29      Pin
24.99   50      Hat
```

```
import java.io.*;

class DataIOTest {
  public static void main(String[] args) {
    try {
      DataOutputStream dos = new DataOutputStream (new FileOutputStream("ordine.txt"));
      double[] prices = {19.99,9.99,15.99,3.99,24.99 };
      int[] units = { 12, 8, 13, 29, 50 };
      String[] descs = {"T-shirt","Mug","Doll","Pin","Hat"};

      for (int i = 0; i < prices.length; i ++) {
         dos.writeDouble(prices[i]);    // reali
         dos.writeChar('\t');           // caratteri
         dos.writeInt(units[i]);        // interi
         dos.writeChar('\t');
         dos.writeUTF(descs[i]);        // stringhe
         dos.writeChar('\n');
      }
      dos.close();
    }
    catch (IOException e) { System.out.println("DataIOTest: " + e);
    }
```

```
    try {
         DataInputStream dis = new DataInputStream (new FileInputStream("ordine.txt"));
         double price;
         int unit;
         String desc;
         double total = 0.0;
         try {
            while (true) {
              price = dis.readDouble();
              dis.readChar();       //  rimuovi tab
              unit = dis.readInt();
              dis.readChar();       //  rimuovi tab
              desc = dis.readUTF();
              dis.readChar();       //  rimuovi fine linea
              System.out.println("Ordinati: " + unit + " pezzi di" + desc + " a $" + price);
                 total = total + unit * price;
             }
            } catch (EOFException e) {
            }
          System.out.println("For a TOTAL of: $" + total);
          dis.close();
          }
          catch (FileNotFoundException e) { System.out.println("DataIOTest: " + e);
        }
        catch (IOException e) { System.out.println("DataIOTest: " + e);
        }
    }
}
```


Random-Access File Streams

```
// Appende il contenuto di un file in coda ad un altro file
// RandomAccessFile permette di leggere e scrivere dati in un file dappertutto.
// (Attenzione i file su disco sono Random-Access, mentre i flussi di rete no!)

import java.io.*;

public class AppendTextFile {
   public static void main(String[] args) {
      if (args.length < 2) {
          System.err.println("Uso: java AppendTextFile <src> <dest>");
          return;
      }
      try {
          RandomAccessFile ifile = new RandomAccessFile(args[0], "r");
          RandomAccessFile ofile = new RandomAccessFile(args[1], "rw");
          ofile.seek(ofile.length());
          String line;
          while ((line = ifile.readLine()) != null) {
                    ofile.writeBytes(line);
                    ofile.writeByte('\n');
          }
          ifile.close(); ofile.close();
      }
      catch (IOException e) {
          System.err.println(e); return;
      }
   }
}
```

Streams predefiniti

- Quando eseguiamo un programma o un applet, vengono creati automaticamente tre oggetti stream:
- `System.in` (input di bytes dalla tastiera)
  - metodo `read()`della classe `InputStream`
- `System.out` (output a video)
  - metodi `print(), println()` della classe `PrintStream`
- `System.err` (messaggi di errore a video)

Leggere una stringa da console

```
import java.io.*;
public class Leggi{
  public static void main(String[] args) throws IOException{
    String linea;
    InputStreamReader lettore;
    BufferedReader bufferizzato;
    lettore = new InputStreamReader(System.in, "8859_1");   // ISO Latin_1
    bufferizzato = new BufferedReader(lettore);
    while ((linea = bufferizzato.readLine()) != null){
      System.out.println("Ho letto: " + linea);
    }
  }
}
```