MANUALE

COMPLETO

DI

JAVA

Pietro Castellucci

Indice generale

LEZIONE 1: *Introduzione a Java*

Java appena è uscito è stato accolto con molto entusiasmo dalla comunità mondiale dei progettisti di software e dei provider di servizi Internet, questo perché Java permetteva agli utenti di Internet di utilizzare applicazioni sicure e indipendenti dalla piattaforma, che si possono trovare in qualsiasi punto della rete.

Java è quindi nato come linguaggio per la rete, per affiancare l'Hyper Text Markup Language (HTML), il quale non è un linguaggio di programmazione vero e proprio, e per dargli quella sicurezza che l'HTML non ha. Da quando è nato Java sulla rete si è iniziato a poter parlare di numeri di carte di credito e di informazioni sicure, notizia che ha molto affascinato le grosse società mondiali, le quali hanno trasformato la vecchia Internet, rete ad appannaggio delle sole università e centri di ricerca, nell'attuale mezzo di comunicazione aperto a tutti.

Il linguaggio di programmazione Java è stato creato verso la metà degli anni novanta, è il più recente tra i suoi cugini, e per questo è ancora in fase evolutiva, tanto che ogni anno circa ne viene rilasciata una nuova relase.

Da linguaggio nato solo per la rete è divenuto un vero e proprio linguaggio di **programmazione**, paragonabile, dal punto di vista delle funzionalità, al più blasonato C++. Java e la maggior parte degli altri linguaggi possono essere paragonati solo dal punto di vista delle funzionalità, perché sono fondamentalmente molto diversi, infatti Java compila i sorgenti dei suoi programmi in un codice detto Bytecode, diverso dal linguaggio della macchina su cui è compilato, mentre linguaggi come il C++ compilano i sorgenti dei programmi in un codice che è il codice della macchina (per macchina intendo computer + sistema operativo) su cui è eseguito. Quindi per eseguire un programma Java occorre avere uno strumento che è chiamato Java Virtual Machine, la quale interpreta il bytecode generato dal compilatore Java e lo esegue sulla macchina su cui è installato. Grazie alla Java Virtual Machine Java è indipendente dalla piattaforma, infatti il programma compilato Java è legato alla JVM e non al sistema operativo, sarà quindi possibile eseguire lo stesso programma Java, compilato una sola volta su una qualche macchina con un compilatore Java versione X, su una piattaforma Windows e su una piattaforma Linux, per fare questo però c'è bisogno che sia Windows che Linux abbiano installato una Java Virtual Machine che supporti la versione X di Java. Le due JVM installate sulle due piattaforme diverse sono lo stesso programma compilato una volta per Windows ed una volta per Linux, come avveniva con i programmi scritti in linguaggi come il C/C++.

Una **Java Virtual Machine** è implementata anche nei vari Browser (Come Netscape e Explorer) per poter eseguire i programmi Java incontrati nella rete, i cosidetti Applet. Questo però, unito al fatto che Java ancora si evolve, causa degli ovvi problemi di incompatibilità: capita sempre che il più moderno Browser supporti una versione precedente di Java rispetto all'ultima versione rilasciata dalla Sun Microsystem, inoltre bisogna tener presente che non tutti gli utenti di Internet navigano usando l'ultima versione di Netscape o di Explorer. Quindi volendo creare un applet ed inserirlo in un nostro documento HTML, dobbiamo tenere presente questi problemi, e cercare di scrivere un programma che sia compatibile con la maggior parte delle JVM inplementate nei vari browser.

Un altro problema da affrontare è quello della scelta del **compilatore** Java da utilizzare, infatti esistono vari ambienti integrati per editare, compilare, debuggare ed eseguire programmi Java, come quelli della Broland, della Microsoft, della Symantec. Tutti questi ambienti offrono dei tool di sviluppo eccellenti, come editori grafici di finestre, debugger molto interessanti, però hanno due problemi, il primo è che si pagano, anche molto, il secondo è sempre lo stesso della compatibilità, infatti essi spesso si trovano indietro alla relase della sun, ed inoltre aggiungono delle classi che poi le JVM implementate nei browser non hanno.

Il mio consiglio è quallo di usare le JDK (**Java Development Kit**) della Sun, le quali comprendono sia il compilatore che la Java Virtual Machine per eseguire i programmi da noi compilati, inoltre sono freeware (non costano niente) e sono scaricabili dalla rete ed i browser si adeguano pian piano a questa versione di Java. Se volete scrivere applet per i vecchi browser dovete scaricarvi la versione 1.1 di Java, però questa versione è niente alla versione attualmente più gettonata, ovvero alla 1.2.2 (Chiamata per qualche motivo a me oscuro Java 2 , la si può scaricare all'indirizzo http://java.sun.com/products/jdk/1.2/index.html), alla quale farò riferimento anche io in questo corso e per la quale Netscape versione 4.0x ed Explorer 5 hanno implementato la Java Virtual Machine (quasi tutta, io ho avuto qualche problema con le Swing, che sono una libreria standard di Java 2).

Se avete intenzione di scaricarvi Java 2 vi consiglio di farlo la mattina verso tra le nove e le undici, perché il sito è affollato e lento, inoltre il file da scaricare è una ventina di mega, se volete potete scaricarvi anche la documentazione la quale è molto utile, però anch' essa è di circa venti mega. Io ho appena scaricato e installato la prossima versione, ovvero la Relase Candidate del Java 2 Software Development Kit versione 1.3 Relase Candidate 1, e vi assicuro che se la 1.2.2 era eccezionale questa è incredibile, ho anche appena scoperto che è uscita la Relase Candidate 2 del JDK 1.3 e che a fine aprile uscirà finalmente la relase (Io la

aspetto, non ho voglia di passare un'altra nottata in bianco per scaricarmi una cosa che verrà rimpiazzata dalla relase tra meno di un mese). Per scaricarvi il prodotto dovrete iscrivervi, fatelo, l'iscrizione è gratuita.

Un ultimo problema che ha Java è la **lentezza**, infatti, come già detto, esso è interpretato, quindi le istruzioni Java prima di essere eseguite dalla macchina vengono interpretate dalla JVM, ovvero per eseguire ogni istruzione il computer eseguirà un numero di istruzioni macchina che è più del doppio delle istruzioni che eseguirebbe se la stessa istruzione fosse stata scritta in C, quindi avrete bisogno di computer veloci per eseguire bene programmi Java, e di questo vi sarete sicuramente accorti anche navigando sulla rete. Anche la memoria è importante, si compila e si esegue anche con soli 32Mb di RAM, ma per fare le cose velocemente ne occorrono almeno 64, pensate che l'ultimo ambiente integrato della Borland, il Jbuilder 3, ha come requisito minimo di RAM 96 Megabyte.

Per finire ritorno al Java Development Kit della **Sun Microsystem**, con questo è possibile produrre tutto il software che si vuole senza dover pagare diritti di uso del prodotto come avviene con il Borland Jbuilder, il Symantec Cafe, e il Microsoft Visual Java, vi consiglio di leggervi la licenza d'uso che troverete quando andrete a scaricare il JDK prima di cominciare a produrre software. A questo punto possiamo cominciare.

LEZIONE 2: *La programmazione Java*

La principale differenza tra Java e gli altri linguaggi di programmazione ad oggetti è che mentre con questi ultimi è possibile anche porgrammare ad oggetti con Java si deve assolutamente programmare ad oggetti. Quindi punto fondamentale a questo punto è spiegare cosa vuol dire programmare ad oggetti.
Sostanzialmente la programmazione avviene allo stesso modo dei linguaggi "normali", solo che sia i dati che le funzioni che li manipolano sono racchiusi in delle strutture dette classi.
Le classi sono dei prototipi di oggetti, ovvero sono delle strutture astratte (non troppo vedremo) che possono essere instanziate e quindi creare un oggetto (ma anche più di uno).
La classe definisce tutte le proprietà degli oggetti appartenenti a quella classe, detti attributi, e le funzioni che verranno usate per agire su di essi, detti metodi. Ad esempio è possibile definire una classe delle persone, come segue:

```
Inizio classe persone
   Attributo annodinascita
   Metodo calcolaetà (annoattuale)
Fine classe persone
```

La classe delle persone così definita ha un attributo che è annodinascita che sarà sicuramente un numero intero ed un metodo che in base all' anno attuale passatogli calcola l'età della persona. Usando il formalismo di Java,per definire la classe persone scriveremo:

```
class persone
{
   public int annodinascita;
   public int calcolaeta ( int annoattuale )
{
return ( annoattuale - annodinascita );
}
}
```

Come si vede abbiamo dichiarato sia il metodo che l'attributo come public, vedremo tra poco cosa significa, vediamo anche che la classe comincia con { e finisce con }, così anche i metodi. Questo ricorda molto il C, e devo dire che la sintassi di Java è molto simile, anzi quasi uguale a quella del C, mentre per chi non conosce il C, le parentesi graffe rappresentano il begin e l'end del pascal. La classe avrà un cosiddetto costruttore (o più di uno), che è un metodo particolare che di solito viene utilizzato per inizializzare gli attributi quando viene instanziata la classe in un oggetto, esso è una funzione che non ha nessun tipo di ritorno ed il nome uguale al nome della classe. Ho detto che i costruttori possono essere più di uno, che però il nome del costruttore deve essere lo stesso di quello della classe, chi è abituato a programmare con linguaggi non orientati agli oggetti troverà tutto questo strano, però è possibile perché Java fa il cosidetto overloading di funzioni, ovvero funzioni con lo stesso nome che hanno parametri diversi (detti in informatica parametri formali) sono diverse, e al momento dell'invocazione viene scelta la funzione in base al parametro (detto parametro attuale). Questo vale per ogni metodo, non solo per i costruttori.

```
class persone
{
   public int annodinascita;
   public String Cognome=new String();
```

```
   // Costruttori
public persone(int annonascita)
{
this("Non Conosco");
this.annodinascita=annonascita;
}

public persone(String Cognome)
{
this(0);
this.Cognome=Cognome;
}

public persone(int annonascita , String Cognome)
{
annodinascita=annonascita;
this.Cognome=Cognome;
}

// Funzione che calcola l'età del soggetto;
public int calcolaeta ( int annoattuale )
{
return ( annoattuale - annodinascita );
}
}
```

Le linee che cominciano con // sono dei commenti, vengono ignorati dal compilatore, vi sono altri due tipi di commenti, quelli racchiusi tra /* e / che permettono di definire commenti su più linee e quelli racchiusi tra /** e */, che permettono sempre di definire commenti su più linee, sono detti commenti di documentazione, essi si devono trovare subito prima la dichiarazione di classi, di membri di classi (attributi o metodi) o costruttori, e vengono inclusi nella eventuale documentazione del codice generata automaticamente.

Nell'esempio vediamo che ci sono tre costruttori, diversi per i parametri formali, che hanno lo stesso nome, vediamo inoltre un nuovo attributo che è Cognome, esso è una Stringa, definita come public String Cognome=new String(); la parte prima dell'uguale è chiara, lo è meno quella a destra, quel new String() crea un nuovo oggetto della classe String, e ne invoca il costruttore che non ha parametri, questo è il modo standard usato da Java per instanziare gli oggetti di una classe. Non deve sorprendere che il tipo di dato stringa sia una classe, in Java è possibile usare oggetti che rappresentano tutti i tipi di dato del linguaggio, inseriti per completezza del linguaggio, detti involucri che a volte sono molto utili, è però possibile anche usare i valori.
Quindi ad esempio ci troveremo a lavorare sia con interi che con oggetti che rappresentano interi. Un ultima cosa che salta all'occhio dall'esempio è che i costruttori hanno volutamente dei parametri che hanno lo stesso nome degli attributi, anche questo è possibile in Java, il quale stabilisce che quando c'è un assegnamento alla sinistra dell'uguale ci deve essere l'attributo, e alla destra il parametro, comunque se non vogliamo confonderci possiamo usare il riferimento this, scrivendo ad esempio this.annodinascita intendiamo l'attributo. This è un riferimento all'oggetto, e nell'esempio lo troviamo anche come invocazione di funzione this(0), in questo caso esso è un riferimento ad un costruttore dell'oggetto, in questo caso chiama il costruttore persone (int annodinascita), con il valore 0. E' quindi possibile in un costruttore chiamare un costruttore diverso della classe stessa, a patto che l'invocazione sia la prima istruzione del costruttore e che il costruttore sia diverso da quello attuale.
A questo punto siamo pronti a creare oggetti appartenenti alla classe da noi appena definita, abbiamo tre modi per farlo, perché abbiamo creato tre costruttori, essi sono:

persone Pietro=new persone(1974);

oppure

persone Pietro=new persone("Castellucci");

oppure

persone Pietro=new persone(1974,"Castellucci");

ora vogliamo creare un altro oggetto della classe persone

persone Lina=new persone(1975);

oppure

persone Lina=new persone("Marucci");

oppure

persone Lina=new persone(1975,"Marucci");

a questo punto ho creato due oggetti della classe persone, essi sono in una relazione con la classe detta di inst_of, gli oggetti si chiamano uno Pietro e l'altro Lina, è possibile anche copiare i riferimenti degli oggetti, ad esempio è possibile scrivere:

persone Pietro2=Pietro;

Costruiti gli oggetti ne poso invocare i metodi, questo si fa indicando Oggetto.Metodo, ad esempio è possibile invocare i metodi:

```
Pietro.calcolaeta(2000);
Pietro2.calcolaeta(2000);
Lina.calcolaeta(2000);
```

Introduciamo adesso degli attributi e dei metodi particolari, i cosidetti membri statici. Per come abbiamo definito i membri della classe non ci è possibile referenziare direttamente dalla classe attributi e metodi (persone.annodinascita è un errore), questo perché essi lavorano su una istanza della classe, ovvero su un oggetto, però a volte può essere utile scrivere metodi e attributi che possano essere invocati senza dover istanziare l'oggetto, ma direttamente dalla classe, per fare questo occorre dichiararli static, ad esempio:

```
class TitoliAziendaVattelapesca
{
public static int TassodiInteresse=3;
public String Proprietario=new String();
public static float InteressiMaturati (int Periodo)
{
return((Periodo * TassodiInteresse )/100)
}
TitoliAziendaVattelapesca(String nome)
{
Proprietario=nome;
}
}
```

Quindi potremo decidere ad esempio di instanziare un oggetto della classe TitoliAziendaVattelapesca solo se ci conviene, ad esempio facendo:

```
if (TitoliAziendaVattelapesca.InteressiMaturati(12)>1000)
CompraAzioni(10000);
```

Dove CompraAzioni(int X) è una funzione che instanzia X TitoliAziendaVattelapesca.

Introduciamo adesso la relazione is_a tra classi, data una classe è possibile creare una nuova classe da questa facendo come si dice in gergo una specializzazione della prima classe. La nuova classe creata è in relazione is_a con la prima. Creata una classe studente dalla classe (detta superclasse) persone, la nuova classe eredita dalla prima tutti i metodi e gli attributi, con la possibilità di definirne dei nuovo o di ridefinirne alcuni, in Java l'estensione di una classe si esplicita con la parola chiave extends.

```
class studente extends persone
{
int matricola;
// Costruttori
public studente(int annonascita)
{
super(annonascita,"Non Conosco");
```

```
}

public studente (String Cognome)
{
super(0,Cognome);
}

public studente(int annonascita , String Cognome)
{
super(annonascita,Cognome);
}

}
```

Come si vede dall'esempio la classe studente eredita tutti i metodi e gli attrubuti della classe persone, definisce un nuovo attributo matricola e nei suoi costruttori chiama i costruttori della classe persone, con super().
super() può essere, come this(), una invocazione di un altro costruttore (tra parentesi vanno i parametri eventuali) o un riferimento alla classe (alla superclasse in questo caso), quindi super.annodinascita rappresenta l'attributo annodinascita della superclasse persone. Le relazioni is_a e inst_of sono le due relazioni più importanti dei modelli ad oggetti. Adesso concludo con un esempio ed alcune considerazioni sull'esempio per spiegare il significato del public che abbiamo introdotto sopra, e del private e protected (attributi e metodi possono essere dichiarati come public, private e protected).

```
class A
{
public int a;
private int b;
protected int c;
// Suoi medodi, attributi e costruttori
}

class B extends A
{
public float a;
private float b;
protected float c;
// Suoi medodi, attributi e costruttori
}

class C
{
// Suoi medodi, attributi e costruttori
}
```

Abbiamo definito tre classi, A,B e C, B è definita da A ridefinendone i tre attributi, da interi a reali.
In A, nei suoi costruttori e nei suoi metodi ho libero accesso ai propri attributi di tipo intero, senza limitazioni, ovvero posso scriverli e leggerli a piacimento.
In B e C accade lo stesso per i propri attributi, ma vediamo cosa succede per quelli delle altre classi.
Ad esempio in B (in un suo metodo) se scrivo espressioni con a,b e c, scriverò delle espressioni per dei float (in Java è importantissimo il tipo delle espressioni), per referenziare invece gli attributi omonimi di A da cui è ereditata, dobbiamo scrivere come sappiamo, super.a, super.b e super.c (e sono degli interi).
Il nostro compilatore Java non darà problemi per le prime due ma ci darà errore per la terza, questo perché il c di A è protected, questo vuol dire che è possibile leggere e scrivere quell'attributo solo all'interno della classe a cui appartiene, ma non è possibile leggerlo e scriverlo da classi estranee o da sottoclassi.
A questo punto mettiamoci in un metodo di C, instanziamo qui dentro un oggetto della classe B (lo chiamo b) e scriviamo le espressioni b.a, b.b e b.c , il nostro simpatico compilatore a questo punto ci darà buona solo la prima, questo perché la terza è protected ed è quindi visibile solo nella classe di appartenenza e la seconda è private, questo significa che è visibile solo alla classe di appartenenza e alle sue sottoclassi, e C non è sottoclasse di B.
Sopra ho un po' barato, ovvero ho parlato di atributi di classi e di oggetti come se fossero la stessa cosa, ma in effetti lo sono, solo con la piccola differenza che se accedo all'attributo di una classe (NOMECLASSE.NOMEATTRIBUTO)questo sarà sicuramente static, e vi accederò solo in lettura, è in pratica una costante. Quando instanzierò la classe in un oggetto (NOMECLASSE NOMEOGGETTO= new NOMECLASSE (parametri);), accedendo allo stesso attributo dell'oggetto

(NOMEOGETTO.NOMEATTRIBUTO) accederò allo stesso valore di prima, e non potrò modificarlo (è static). Se cercherò invece di accedere ad un attributo non dichiarato static di una classe avrò ovviamente errore, infatti questo attributo verrà creato al momento della creazione dell'oggetto della classe, ed il più delle volte verrà inizializzato dal costruttore dell'oggetto.
Spero di essere stato abbastanza chiaro nella trattazione dell'argomento, però devo dirvi che i modelli ad oggetti sono piuttosto complicati, ed è impossibile spiegarli in dettaglio in un solo paragrafo, occorrerebbe un corso dedicato interamente a loro. Ovviamente quello che vi ho detto non è tutto sui modelli ad oggetti, ho solo introdotto alcuni concetti che ci basteranno per fare i nostri programmi in Java, se dovessi accorgermi durante il corso di avere tralasciato qualche punto importante per i nostri scopi aprirò volentieri una piccola parentesi.
Per il lettore che volesse approfondire l'argomento non mi sento di consigliare un testo, perché ve ne sono vari sull'argomento e tutti egregi, se invece vi accontentate di una introduzione all'argomento potreste vedere i capitoli iniziali di ogni buon manuale di programmazione per principianti di Linguaggi ad oggetti (C++ o Java), come ad esempio Java: Didattica e Programmazione di K.Arnold e J.Gosling (Ed. Addison-Wesley) Capitoli 2,3 e 4.
Mi raccomando però che sia per principianti, perché sui testi di programmazione avanzata l'argomento è dato per scontato.

LEZIONE 3: *Cosa sono i Package di Java*

I package sono delle collezioni di classi, racchiuse in una collezione che le accomuna. Sono in pratica delle librerie a cui l'utente può accedere e che offrono varie funzionalità.
I package possono anche essere creati dall'utente, ad esempio racchiudendovi tutte le classi che ha definito per svolgere alcune funzioni che poi userà in vari programmi, ma questo a noi non interessa, perché ci interesserà vedere i package sicuramente più interessanti definiti in Java.
E' questa la vera potenza odierna di Java, il numero di classi già definite che svolgono i più svariati compiti, ed è anche la parte che continua sempre di più a crescere e ad aggiornarsi con le nuove versioni di Java (JDK 1.3 ne ha ben 18Mb). I package di Java sono così tanti che ne descriverò per sommi capi alcuni in un intero capitolo (il capitolo 3) del corso, e ne vedremo più o meno in dettaglio un paio, ovvero quelli che ci serviranno per definire interfacce grafiche e applets. Ovviamente tutti i package del linguaggio sono descritti nella mastodontica guida in linea del JDK scaricabile insieme al pacchetto del compilatore (La JDK Documentation 1.3 beta è di 105 Megabyte zippata è di una trentina di Mega), la quale è anch'essa gratuita, e che se avete voglia di passare un paio d'ore a scaricarla vi consiglio di prenderla insieme al compilatore,comunque è a vostra disposizione anche in linea (c'è il link nella pagina www.javasoft.com o www.java.sun.com , è lo stesso, sono parenti stretti).
Il nucleo del linguaggio Java contiene solo le parole chiave per la costruzione delle classi, i commenti, i normali costrutti if, switch, while, do-while, for, etichette, break, continue e return (manca il goto),che vedremo in dettaglio nel prossimo capitolo, tutto il resto è implementato nei package del linguaggio, comprese le normali primitive di Input e di Output. In questo paragrafo vedremo la lista dei packages di Java e ne vedremo uno in particolare, quello che ci interessa vedere per scrivere la nostra prima applicazione Java ancor prima di avere visto i costrutti, ovvero quello in cui possiamo trovare le istruzioni di input e output.

La lista completa dei packages di Java 1.2.1 in ordine alfabetico è la seguente :

com.sun.image.codec.jpeg, com.sun.java.swing.plaf.windows,com.sun.java.swing.plaf.motif, java.applet, java.awt, java.awt.color, java.awt.datatransfer, java.awt.dnd, java.awt.event, java.awt.font, java.awt.geom, java.awt.im, java.awt.image, java.awt.image.renderable, java.awt.print, java.beans, java.beans.beancontext, java.io, java.lang, java.lang.ref, java.lang.reflect, java.math, java.net, java.rmi, java.rmi.activation, java.rmi.dgc, java.rmi.registry, java.rmi.server, java.security, java.security.acl, java.security.cert, java.security.interfaces, java.security.spec, java.sql, java.text, java.util, java.util.jar, java.util.zip, javax.accessibility, javax.swing, javax.swing.border, javax.swing.colorchooser, javax.swing.event, javax.swing.filechooser, javax.swing.plaf, javax.swing.plaf.basic, javax.swing.plaf.metal, javax.swing.plaf.multi, javax.swing.table, javax.swing.text, javax.swing.text.html, javax.swing.text.html.parser, javax.swing.text.rtf, javax.swing.tree, javax.swing.undo, org.omg.CORBA, org.omg.CORBA.DynAnyPackage, org.omg.CORBA.ORBPackage, org.omg.CORBA.portable, org.omg.CORBA.TypeCodePackage, org.omg.CosNaming, org.omg.CosNaming.NamingContextPackage, sun.tools.ttydebug, sunw.io, sunw.util.

Sembrano pochini in effetti, sembra quasi che io abbia detto un stupidaggine in riferimento alla potenza di Java, ma se pensate che solo il package java.io comprende 50 classi e 10 interfacce capite bene che quelli di sopra sono una collezione di classi corposa.
A questo punto vorremmo fare l'input e output da consolle, per fare questo dobbiamo usare il package java.lang .
Per usare un package in una nostra classe, prima della definizione della classe dobbiamo inserire l'istruzione import, ad esempio volendo usare il package java.awt dobbiamo inserire all'inizio del nostro file:

import java.awt.;*

La * sta ad indicare che vogliamo usarne tutte le classi, se invece vogliamo usarne solo una classe possiamo specificarlo, ad esempio, import java.awt.Frame; potremo usare solo la classe Frame dell'awt. Nel caso nostro, in cui vogliamo fare una operazione di output, dovremmo dichiarare all'inizio import java.lang.*; oppure, sapendo che la classe del package java.lang che contiene i metodi per fare questo è System potremmo scrivere

import java.lang.System;

La classe System a sua volta avrà al suo interno una import java.io (che è il package per l'input e l'output) per accedere alle classi dell'input e dell'output, e le userà per mandare quello che vogliamo su schermo.
Abbiamo visto un esempio di package che al suo interno invoca un altro package per fargli svolgere dei compiti, in queste librerie Java accade spesso questo, ed è proprio questo fenomeno che a molti fa apparire Java un linguaggio ostico, però superato questo blocco psicologico di fronte a quest'aspetto del linguaggio, la programmazione diviene semplice ed immediata.
Ho introdotto il package java.lang, vi dico anche che questo è il più importante di Java, in questo sono racchiuse le classi fondamentali del linguaggio, tanto che non serve dichiarare l'import, perché java lo importa automaticamente.
Inoltre vorrei soffermarmi su un aspetto fondamentale dell'importare i package, se io importo nel mio file il package java.lang, anche se esso importerà il package java.io, io dal mio file non potrò usare le classi di java.io, per farlo devo importarlo esplicitamente. Questo succede anche se programmo una applicazione in più file (con più classi), in ogni file devo importare i package che mi occorrono per la classe che sto definendo, non basta importarli in una sola. Vediamo le classi che contiene quindi questo package java.lang tanto importante per il linguaggio:

Boolean
abbiamo detto che i tipi di dato possono essere anche rappresentati da oggetti detti;

contenitori
questa è la classe che genera i contenitori per i tipi di dato booleani (vero, falso).

Byte
questa è la classe che genera i contenitori per i tipi di dato interi corti (di un byte).

Character
questa è la classe che genera i contenitori per i tipi di dato caratteri.

Character.Subset
questa è la classe che genera i contenitori per i tipi di dato caratteri con codifica unicode.

Character.UnicodeBlock
questa è la classe che genera i contenitori per i tipi di dato caratteri con codifica unicode 2.0.

Class
rappresenta classi e interfacce a runtime (in esecuzione), in Java è possibile a tempo di esecuzione creare, mandare in esecuzione e compilare classi, il compilatore e l'esecutore sono mischiati in modo strano, ma a noi non serviranno queste classi "particolari", perché scriveremo delle applicazioni tranquille.

ClassLoader
caricatore di classi a tempo di esecuzione.

Compiler
compilatore di classi a tempo di esecuzione.

Double
questa è la classe che genera i contenitori per i tipi di dato reali in virgola mobile di 64 bit.

Float
questa è la classe che genera i contenitori per i tipi di dato reali di 32 bit in virgola mobile.

Integer
questa è la classe che genera i contenitori per i tipi di dato interi.

Long
questa è la classe che genera i contenitori per i tipi di dato interi doppi.

Math
contiene metodi che emulano funzioni matematiche.

Number
classe di oggetti contenitori numeri generici.

Object
questa è la classe da cui derivano tutte le classi del linguaggio Java, ovvero ogni classe è sottoclasse (non per forza diretta) di questa.

Package
contiene metodi per estrapolare informazioni sui package di Java.

Process
Java è un linguaggio che permette di gestire Thread, ovvero programmini che girano in Parallelo, questa classe si occupa di loro, come le altre classi di java.lang: Runtime, RuntimePermission, Thread, ThreadGroup, ThreadLocal, Throwable

SecurityManager
per la sicurezza

Short
questa è la classe che genera i contenitori per i tipi di dato interi corti.

String
questa è la classe che genera i contenitori per i tipi di dato stringhe di caratteri.

StringBuffer
questa è la classe che genera i contenitori per i tipi di dato stringhe di caratteri modificabili.

System
è la classe che interagisce con il sistema, essa contiene molti metodi utili, e dei riferimenti ad altre classi (detti class field), è quella che useremo ad esempio noi per creare un output su schermo a caratteri.

Void
questa è la classe che genera i contenitori per i tipi di dato void, ovvero senza tipo. Questo sembra un tipo di dato inutile, ma vedremo che non lo è affatto, anzi è utilissimo, infatti è utile quando si vuole definire un metodo-procedura (un metodo che non ha valore di ritorno), in Java esistono solo funzioni, le quali sono obbligate a ritornare un valore del tipo che dichiarano. Dichiarando un metodo come void, si emula la procedura, ovvero non vi è la necessità del return.

LEZIONE 4: *Scrittura di applicazioni "a console"*

Mi rendo conto che tutta la parte descrittiva fatta fin' ora è stata abbastanza noiosa, sono d'accordo con voi, però sono sicuro che mi scusiate per esserlo stato, perché i concetti introdotti sono FONDAMENTALI per la parte meno noiosa che sta per cominciare, ovvero la programmazione.
Prima di cominciare però devo dirvi un'ultima cosa, con gli abituali linguaggi di programmazione è possibile fare due tipologie di programmi, escludendo le librerie dinamiche (.DLL) o statiche (.LIB) e varie, ovvero le applicazioni a consolle e le applicazioni a finestre, le prime sono quelle più semplici da costruire, che non hanno grafica e che hanno interfaccia a caratteri, tipo le vecchie applicazioni DOS, o javac.exe stesso (il compilatore Java), mentre le seconde sono quelle Windows o X-Windows, con tutti i bottoni, menu e via dicendo, più complicate da fare, ma estremamente gradevoli da vedere. Con Java è possibile fare oltre a queste due una terza applicazione, quella che gira su Web, ovvero l'applet. Inizialmente Java era nato solo come linguaggio per il Web, ma poi si è visto che era un linguaggio completo con cui era facile scrivere anche applicazioni a consolle e a finestre. Anche io sto facendo la mia Tesi in Informatica usando Java, e la mia è una applicazione a finestre. Le prime applicazioni che vedremo sono chiaramente le più semplici, anche se le meno spettacolari, ovvero le applicazioni a consolle.
Un applicazione può essere composta da una o più classi, tutte rinchiuse in omonimi files tipicamente, solo per una questione di semplicità di reperimento, anche se è possibile definirne più di una in un file e chiamare i files con nomi diversi dalle classi in essi contenute, ma se si scrive una applicazione a consolle o una a finestre vi è una classe speciale che deve essere contenuta in un file che ha il suo stesso nome. Questa classe che chiamerò ad esempio ciao conterrà un metodo, chiamato main che rappresenta l'entry point del programma, ovvero il punto dal quale comincerà l'esecuzione del programma.
Quando un programma in Java viene compilato genererà uno o più files .class, tra cui nel nostro caso ci sarà

anche ciao.class, per cominciare l'esecuzione del programma dovremo battere:

java ciao [INVIO]

Da notare che gli eseguibili Java non sono .EXE, ma .class, questo perché per essere eseguiti hanno bisogno della citata Java Virtual Machine, che è appunto il porgramma java.exe. Faccio un esempio:

```
class ciao
{
public static void main(String[] args)
{
// Commento, questo programma non fa niente.
}
}
```

Vediamo che main è dichiarato static void, static perché esso è invocato all'inizio dalla JVM, prima che ciao sia instanziato automaticamente in un oggetto, void perché non deve ritornate niente. Questa classe la scriverò in un file chiamato ciao.java, e la compilerò scrivendo javac ciao.java NB: E' assolutamente obbligatorio chiamare i file sorgenti di Java con l'estensione .java, altrimenti il compilatore non li compila, inoltre se ho creato ciao.java DEVO battere javac ciao.java e non javac ciao per compilarlo.
Come parametro il metodo main ha una stringa chiamata args, questa rappresenta i parametri con cui viene invocato il programma, ad esempio se scriverò una volta compilato ciao in ciao.class

java ciao Pietro Castellucci
args[0] sarà uguale a Pietro
args[1] sarà uguale a Castellucci

questo a tempo di esecuzione. Notate anche che la JVM java.exe esegue ciao.class specificandogli semplicemente java ciao, a differenza del compilatore javac.exe, anzi se si batte java ciao.class dà errore, provate a farlo.

LEZIONE 5: *Il nostro primo programma in Java*

Questo che stiamo per fare in effetti non è il nostro prima programma in Java, già qualcuno lo abbiamo fatto sopra, questo è il primo programma Java che faccia qualcosa.
Useremo le cose viste nel paragrafo precedente per creare una applicazione a consolle, e quelle del paragrafo prima per scrivere qualcosa sullo schermo.
Iniziamo a editare un file che chiameremo CiaoMondo.java, mi raccomando alle lettere maiuscole e minuscole, in Java sono importantissime, non solo a livello di programma, ma anche per i files, mi auguro inoltre che abbiate un editore che supporti java, perché se editate con l'edit del DOS non sarete molto aiutati nella stesura dei programmi, mentre con l'editore specializzato avrete le indentazioni fatte ad ok, i colori diversi per le parole chiave, per i commenti, ecc…
Potete scaricare gratuitamente degli editori appositamente creati o rivisti per Java sulla rete, potete provare al sito http://omega.di.unipi.it della facoltà di Informatica di Pisa, al settore Resources - ftp, oppure ai vari siti della tucows (www.tucows.com), dove troverete il bluette (una versione demo), che ha un bell'editore, solo mi raccomando in questo caso, il bluette è un compilatore java, ma non so se è interamente compatibile con il Java della SDK che sto esponendo io, quindi fate attenzione. Nel nostro file CiaoMondo.java scriveremo:

```
class CiaoMondo
{

public static void main(String[] args)
  {

    System.out.print ("Ciao mondo, sono il primo programma in Java ");

    System.out.println ("di "+args[0]+" "+args[1]);

}

}
```

Compiliamo il programma scrivendo al solito java CiaoMondo.java e ora possiamo eseguirlo, scriveremo:

java CiaoMondo TUONOME TUOCONGNOME

e vedremo un output simile a questo:

Ciao mondo, sono il primo programma in Java di TUONOME TUOCOGNOME

Se non mettiamo il nome e il cognome il programma genererà una eccezione, che è una sorta di errore a runtime, dovuta al fatto che nel main si riferiscono arg[0] e arg[1] che non sono presenti. Queste eccezioni sono si degli errori a runtime, ma si possono prevedere e gestire, proprio perché Java è un linguaggio "sicuro", e invece di bloccarsi e dare il solito RUNTIME ERROR deto dagli altri linguaggi, lancia una eccezione, che se prevista dal programmatore non blocca il programma. Vedremo nel prossimo cappitolo come gestire eventuali eccezioni, ed anche come usarle per i nostri scopi, si pensi ad esempio ad un programma che va a leggere da file e che non trova il suddetto file, Java lancerà la sua eccezione, noi la raccoglieremo e faremo leggere un altro file. La versione senza riscio di eccezioni e più coreografica è la seguente:

```
class CiaoMondo2
{

public static void main(String[] args)

  {

    System.out.println ("**************************************************");
    System.out.println ("** Ciao mondo, sono il primo programma in Java **");
    System.out.println ("**************************************************");
    System.out.println (" |||||");
    System.out.println ("0/ x x \\0");
    System.out.println (" | o |");
    System.out.println (" |\\___/|");
    System.out.println (" |_____|");

}
}
```

A questo punto non resta che farvi i miei complimenti, avete scritto il vostro primo programma in Java. Se vi sembra pochino, non temete dal prossimo capitolo vedremo i costrutti e allora potremo sbizzarrirci a scrivere tutti i programmi che desideriamo.

LEZIONE 6: *Tipi primitivi di Java e valori*

Per tipi primitivi di un linguaggio si intende i tipi di dato già definiti nel linguaggio e dai quali si può partire con la costruzione di espressioni, o di tipi composti.
I tipi sono molto importanti in tutti i linguaggi di programmazione tipati, e sono fondamentali in Java che è un cosiddetto linguaggio fortemente tipato. Il concetto di tipo è molto naturale, se io vedo un numero, ed esempio 15.4, posso dirne subito l'insieme di numeri a cui appartiene, in questo caso ai reali.
Se vedo una operazione 4 / 5 posso dire:

4 è di intero intero **5** è di tipo intero **e /** è una funzione che prende due interi e mi restituisce un reale, formalmente si scrive così:

/ : int x int à float

quindi posso dire che tutta l'espressione 4/5 è di tipo intero.

Questa che abbiamo fatto è all'incirca una inferenza di tipi, in informatica è una pratica molto attuata, che per essere spiegata tutta, come per i modelli ad oggetti, avrebbe bisogno di un corso intero.
I problemi non sono con queste espressioni, cosiddette chiuse, che non hanno variabili, ma con quelle con le variabili, ogni linguaggio usa una sua tecnica, chi non tipa le espressioni, chi stabilisce il tipo dal contesto della espressione e ne da un tipo in conseguenza anche alle singole espressioni e chi come Java fa dichiarare i tipi di tutte le variabili all'inizio. I tipi primitivi di Java sono gli stessi degli altri linguaggi di programmazione, solo che sono leggermente diversi come valori, essi sono.

boolean
ovvero valori che possono essere true e false

char i caratteri, sono di 16 bit, e sono codificati Unicode 1.1.5, negli altri linguaggi sono ASCII, di soli 8 bit.

byte
interi di 8 bit con segno, ovvero numeri compresi tra meno (due alla settima) e due alla ottava.

short
interi di 16 bit con segno.

int
interi di 32 bit con segno.

long
interi di 64 bit con segno.

float
reali di 32 bit in virgola mobile (IEEE 754-1985).

double
reali di 32 bit in virgola mobile (IEEE 754-1985).

Da questi ovviamente è possibile definire nuovi tipi di dato che vengono detti tipi composti, ad esempio una stringa è un vettore di caratteri, ed è un tipo composto.
Come abbiamo visto nella precedente lezione nel package java.lang sono contenuti i contenitori per questi tipi di base, ovvero se dichiaro int mi riferisco ad un intero, mentre se dichiaro Integer mi riferisco alla classe interi.

Se uso le classi, avrò anche una serie di attributi e metodi utili per lavorare su questi, ad esempio in alcuni saranno definite le costanti MIN_VALUE e MAX_VALUE (Minimo e massimo valore), nelle classi Float e Double ci saranno le costanti NEGATIVE_INFINITY e POSITIVE_INFINITY, NaN per indicare un valore non valido e il metodo isNan() per controllare se un valore è valido, ad esempio si può utilizzare dopo una divisione per controllare che non sia stata fatta una di solito catastrofica divisione per zero. Tutto questo negli altri linguaggi di programmazione non c'è.
Vediamo quali sono i letterali di ogni tipo di Java, ovvero i valori costanti che ogni tipo può assumere.

L'unico letterale per i riferimenti agli oggetti è null, si può utilizzare ovunque si aspetti un riferimento ad un oggetto, null è un oggetto non creato o non valido, esso non è di nessun tipo.

I letterali booleani sono true e false, ad indicare rispettivamente il valore vero e il valore falso.

I letterali interi sono stringhe di cifre ottali, decimali o esadecimali. L'inizio della costante serve a dichiarare la base del numero, uno zero iniziale indica la base otto, 0x o 0X indica la base 16 e niente indica la notazione decimale. Ad esempio il numero 15 in base dieci può essere rappresentato come:

15 in notazione decimale
017 in notazione ottale
0xf oppure 0XF in notazione esadecimale.

Le costanti intere che terminano con l o L sono considerate long, meglio che terminino con L, perché l si confonde facilmente con l'unità (non dal compilatore, ma da noi che leggiamo i programmi), se non hanno lettera terminale sono considerati degli int. Se un letterale int viene assegnato direttamente ad uno short o ad un byte viene trattato come se fosse tale.

I letterali in virgola mobile vengono espressi come numeri decimali, con un punto opzionale seguito eventualmente da un esponente che comincia con una e, secondo la usuale notazione mantissa-esponente. Il numero può essere seguito da una f (F) o da un d (D) per indicare che è a precisione singola o doppia, per default sono in precisione doppia.

35, 35. , 3.5e1, .35e2 denotano lo stesso numero.

Lo zero può essere positivo o negativo, confrontandoli si ottiene l'uguaglianza, però è utile nei calcoli, si pensi a 1d/0d e 1d/-0d.
Non è possibile assegnare direttamente una costante double ad una float, anche se entra nel suo range, se si vuole effettuare questa operazione si deve fare un esplicita forzatra detta casting che poi vedremo cos'è.

I caratteri sono racchiusi tra apici, come ad esempio '2', e i caratteri speciali vengono rappresentati da delle

sequenze, dette sequenze di escape, esse sono:

\n newline, va accapo

\t tab

\b backspace, cancella a sinistra

\r return, rappresenta il carattere speciale INVIO

\f form feed

**** è il carattere backslash

\' apice

\" doppio apice

\ddd è un char di cui si fornisce il valore ottale (d sono cifre ottali, ad esempio \329, devono essere di tre o meno cifre e minori di \377), si può dare anche la rappresentazione esadecimale, sempre di quattro cifre, ecco in ordine i caratteri di sopra rappresentati con il loro numero esadecimale: \u000A, \u0009, \u0008, \u000D, \u000C, \u005C, \u0027 e \u0022.

Le stringhe vengono scritte tra virgolette, come ad esempio "Pietro", tutte le sequenze di escape valide possono essere inserite in una strunga, creando cose interessanti, ad esempio:

System.out.println("\tNome:\tPietro\n\tCognome:\tCastellucci");

darà come risultato:

Nome: Pietro
Cognome: Castellucci

LEZIONE 7: *Variabili*

Le variabili sono dei valori modificabili, ovvero sono dei nomi che rappresentano un valore di un certo tipo, il valore associato al nome può essere variato. Ad esempio se dico che X è una variabile di tipo intero, e poi dico che X ha valore 10, se scrivo l'espressione 5 + X, è come se avessi scritto l'espressione 5 + 10. Credo che comunque tutti voi abbiate dimestichezza con le variabili, in quanto sono la cosa principale di ogni linguaggio di programmazione, detto ciò vediamo come Java tratta le variabili.
Innanzitutto, prima di essere usata, una variabile deve essere dichiarata, in Java una dichiarazione di variabile è composta da tre parti: modificatori, tipo e identificatori.

I modificatori sono opzionali, e sono sostanzialmente quelli che stabiliscono l'accesso alle variabili, ovvero i public, private e protected visti nel precedente capitolo, detti modificatori di accesso, e poi ci sono static, synchronized e final. L'ordine con cui vengono messi non è importante, anche se è bene usare sempre la stessa convenzione, per la leggibilità del codice. Anche static è stato visto, permette di dichiarare degli attributi e dei metodi costanti che possono essere invocati anche prima che un oggetto venga instanziato, final invece dichiara che ad un campo il valore può essere assegnato solo una volta, quando viene inizializzato (è una costante anch'essa), il modificatore synchronized per ora non lo vediamo, e forse non lo vedremo proprio, serve per dichiarare che ad un metodo, in caso di multiprogrammazione (con i thread), vi si può accedere in mutua esclusione, ovvero un thread alla volta.

Gli identificatori possono essere uno o più di uno, separati da virgole, e sono i nomi che avranno le variabili. Ad esempio la dichiarazione di X dell'esempio precedente sarà:

int X;

ma posso anche dichiararne più di uno, ad esempio è valida la seguente dichiarazione:

int X,Y,Z;
double W=3.12;

Ovviamente essendo gli attributi delle classi, devono essere dichiarati all'interno di queste, le dichiarazioni fatte all'esterno sono errate.

Come si vede dall'esempio è possibile anche inizializzare una variabile al momento della sua creazione, semplicemente aggiungendo dopo il nome l'uguale ed il valore che prende.
Dopo la dichiarazione della variabile, e la sua inizializzazione, ovunque venga scritto l'identificatore della variabile si intende il suo valore attuale, tranne che in un caso, nell'assegnamento di un nuovo valore. Ad esempio se io scrivo X = 10 intendo per X la variabile e non il suo valore, e con quell'uguale 10 dico che il nuovo valore che ha la variabile è 10.
Se dopo scrivo:
X = X+1;
Con la X a sinistra dell'uguale intendo dire la variabile chiamata X, mentre con la X a destra intendo dire il valore attuale della variabile X, quell'espressione X+1 sarà quindi in questo caso

10 + 1 e quindi 11, ovvero per X = X + 1; ho detto X = 11;

Provate ad esempio questo programmino che fa due conti:

```
class p
{
int X,Y,Z;

double W = 3.12;

public double A = 15;

static int B = 101;

private final int C = 2;

protected static boolean D = true;

public p()

{

X= 10 ;

Y= X ;

Z= X + Y ;

System.out.println ("All\'inizio ho: X="+X+", Y="+Y+", Z="+Z);

X= X + 1 ;

Y= Z - X;

System.out.println ("Effettuo le operazioni: \nX= X + 1 ;\nY= Z - X;\ned ottengo:");

System.out.println ("X="+X+", Y="+Y+", Z="+Z);

}

}

class decl

{

public static void main(String[] a)

{
p Prova=new p();

}
}
```

come sapete, visto che decl è la classe che contiene il main dovete editare il programma in un file chiamato

decl.java, poi eseguitelo e vedete cosa succede. Vi sono anche altri attributi per far vedere l'uso dei modificatori.
Dall'esempio si vede che io ho creato una classe p e ne ho instanziato l'oggetto Prova, invocandone il costruttore. Questo l'ho dovuto fare perché le variabili sono degli attributi, e quindi, se non sono static, sono attributi di un oggetto, per questo motivo ho dovuto creare un oggetto per potere lavorare con le sue variabili. Abituatevi a programmare sotto questa ottica, perché i programmi Java sono tutti così. Siccome il main del programma DEVE essere dichiarato static (e deve avere una stringa come perametro) esso viene eseguito in assenza dell'oggetto, e quindi delle variabili non statiche. Se avessi voluto creare una sola classe avrei dovuto scrivere qualcosa del tipo:

```
class decl2
{
int X,Y,Z;

// Main
public static void main(String [] a)
{

decl2 Prova=new decl2();

}

// Costruttore

public decl2()
{

X= 10 ;

Y= X ;

Z= X + Y ;

System.out.println ("All\'inizio ho: X="+X+", Y="+Y+", Z="+Z);

X= X + 1 ;

Y= Z - X;

System.out.println ("Effettuo le operazioni: \nX= X + 1 ;\nY= Z - X;\ned ottengo:");

System.out.println ("X="+X+", Y="+Y+", Z="+Z);

}

}
```

In questo secondo esempio, editato e compilato in decl2.java, non si capisce bene quando vedo decl2 come classe e quando come oggetto, infatti il main è nella classe, e nel mai ne viene creato l'oggetto, quindi invocato il costruttore e qui dentro si ragiona in termini di oggetto instanziato. Comunque è possibile, vedrete che il programma è funzionante anche in questo caso, ed un giorno quando avrete più familiarità con il linguaggio lo potrete fare anche voi, ma per ora io ve lo sconsiglio vivamente, perché è fondamentale a questo punto distinguere bene classe e oggetto, cosa che con l'esperienza viene più facilmente.

Gli identificatori Java devono cominciare con una lettera (oppure con _ o $) seguita da una sequenza lunga a piacere di caratteri e cifre, che essendo codificate Unicode sono molto più numerose delle lettere e delle cifre ASCII usate dagli altri linguaggi. Sono spiacente di non avere una mappa completa dei caratteri unicode da darvi, ma posso dirvi che tra i caratteri sono compresi i caratteri greci, cirillici, arabici, eccetera.

Vediamo adesso come dichiarare e inizializzare un array, la dichiarazione avviene mettendo un paio di parentesi quadre dopo il tipo:

```
int [] vettoreDiInteri;
```

Gli array in Java sono quasi degli oggetti, solo che essi non possono essere estesi per aggiungervi nuovi

metodi. Quello dichiarato sopra è un riferimento ad unn'array, che però il linguaggio non ha allocato, se voglio inizializzare l'array elemento per elemento, come si fa con gli altri linguaggi, devo dichiararne le dimensioni:

int [] vettoreDiInteri = new int[100];

Da questo momento in poi posso iniziare a mettere elementi nel vettore nelle posizioni da 0 a 99, se vado fuori avrò una eccezione di out of range.
Al momento della creazione posso anche inizializzare un array, basta scrivere:

int [] vettoreDiInteri = { 1,2,3,10,100,555,0, ….altri valori…};

Posso anche creare dei vettori i cui elementi sono dei vettori, e così via, questo si fa così:

```
double[] [] matriceIdentitaDiOrdine3 = { { 1.0 , 0.0 , 0.0 } ,
{ 0.0 , 1.0 , 0.0 } ,
{ 0.0 , 0.0 , 1.0 } };
```

Il prossimo programma usa gli array, lo editeremo in un file chiamato vett.java:

```
class vett
{

static int NOME = 0;

static int COGNOME = 1;

static int PAPERINO=0;

static int GAMBA=1;

static int Q1=2;

static int Q2=3;

static int Q3=4;

static int CICCIO=5;

static int COMM=6;

static int TOPO=7;

static int PIPPO=8;

static int CANE=9;

static int PAPERA=10;

static int PAPERONE=11;

static String personaggi[][] = {
{"Paolino","Paperino"},
{"Pietro","Gambadilegno"},
{"QUI","Non Specificato"},
{"QUO","Non Specificato"},
{"QUA","Non Specificato"},
{"Ciccio","Di Nonna Papera"},
{"Non Specificato","Bassettoni"},
{"Non Specificato","Topolino"},
{"Pippo","Non Specificato"},
{"Pluto","Non Specificato"},
{"Non Specificato","Paperina"},
{"Non Specificato","Paperone"},
};
```

```
public static void main(String [] a)
{

int PersonaggioTmp;

int Numero=1;

System.out.println ("Alcuni Personaggi della Walt Disney");

PersonaggioTmp=PAPERINO;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=GAMBA;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=Q1;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=Q2;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=Q3;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=CICCIO;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=COMM;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=CANE;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;
```

```
PersonaggioTmp=PIPPO;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=PAPERONE;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

Numero=Numero+1;

PersonaggioTmp=TOPO;

System.out.println("Numero:"+Numero+"\tNome:"+personaggi[PersonaggioTmp][NOME]+"
Cognome:"+personaggi[PersonaggioTmp][COGNOME]);

System.out.println("Ho contato "+Numero+" personaggi, ne erano molti di più!");

// Questo si che è davvero un programma inutile! L'autore.

}
}
```

Una piccola nota sul programma, avete visto nel main che vengono dichiarate delle variabili (è possibile farlo), vengono usate normalmente non dichiarandole static, questo perché le variabili definite nel main, o in una funzione in generale, non sono degli attributi, quindi non sono legati all'oggetto o alla classe, ma solo alla funzione in cui sono definiti. In queste definizioni non si mettono i modificatori, perché essi sono associati alle classi.
A questo punto non resta che vedere i tipi composti, ma è facile per noi identificarli ora con delle classi, ad esempio volendo definire il tipo composto Complessi definiremo:

```
class Complessi
{
private double ParteReale;

private double ParteImmaginaria;

Complessi (double Re, double Im)
{

ParteReale=Re;

ParteImmaginaria=Im;

}

// a questo punto definiremo tutti i nostri ovvi metodi sul nouvo tipo
double dammiParteReale()
{
return ParteReale;
};
double dammiParteImmaginaria()
{
return ParteImmaginaria;
};
double calcolaModulo()
{
return (java.lang.Math.sqrt((ParteImmaginaria*ParteImmaginaria)+ (ParteReale*ParteReale)));
};
}
```

Per provarla potete scrivere il seguente(provaComplessi.java):

```
class provaComplessi
{

public static void main (String[] arg)
{
Complessi N=new Complessi(3.0,1.1);

double Re=N.dammiParteReale();

double Im=N.dammiParteImmaginaria();

double mod=N.calcolaModulo();

System.out.println("Numero complesso:"+Re+"+i"+Im+" ha modulo "+mod);

}

}
```

LEZIONE 8: *Operatori*

Gli operatori del linguaggio Java sono gli stessi degli altri linguaggi, in più ogni nuovo tipo avrà i propri metodi che ne implementeranno nuovi operatori, come abbiamo visto nell'esempio della classe Complessi.
Abbiamo già visto gli operatori di accesso alle classi e agli oggetti, ovvero il semplice punto. Volendo accedere al campo X dell'oggetto x scriveremo semplicemente x.X, anche se x ha un metodo Y(int valore), noi vi accediamo, o meglio in questo caso lo invochiamo scrivendo x.Y(39); Ci sono i soliti operatori aritmetici: +,-,*,/ e %.
Da notare che le operazioni sono anch'esse tipate, ovvero se sommo due interi ho un valore intero, così per tutte le operazioni, in modo che data una espressione è sempre possibile definirne il tipo del risultato.
Java fa un forte controllo sui tipi, infatti non è possibile assegnare un carattere ad un intero, cosa possibile nel linguaggio C, ma è anche impossibile assegnare un double ad un float, senza un esplicito casting. Il casting consiste nel forzare un valore di un tipo ad assumere un tipo diverso. Se scrivo 5 io intendo un intero, mentre se ne faccio un cast a float intenderò la versione di 5 reale, per farlo scriverò (float) 5.
Facendo così posso fare i famosi assegnamenti di interi a reali e di reali double a reali float, si pensi ad esempio a :

```
double v1=10.0;
float v2;
int v3=5;
```

e si pensi agli assegnamenti errati seguenti:

```
v2=v1;
v1=v3;
```

Se effettuiamo dei cast, diventano leciti anche per Java:

```
v2=(float) v1;
v1=(double) v3;
```

Da notare che la divisione tra interi, a differenzache in altri linguaggi da un intero, quindi se si vuole il numero reale derivato dalla divisione dei due numeri interi x e y, bisogna prima trasformarli in reali:

```
float risultato= (float ) x / (float) y;
```

Per le stringhe esiste un operatore di concatenamento:

```
String a="Pietro " ;
String b= a + "Castellu";
String b+="cci"
```

Il risultato sarà Pietro Castellucci. L'operatore è il +. Abbiamo ad un certo punto usato un assegnamento particolare, ovvero +=, questa è una abbreviazione, se devo scrivere a=a+1; posso abbreviare e scrivere a+=1, che è lo stesso. Vale lo stesso per ogni operatore binario. L'operatore + appende ad una stringa anche

caratteri e numeri, si pensi agli output degli esempi precedenti, dove scrivevamo ad esempio:

System.out.println("Numero complesso:"+Re+"+i"+Im+" ha modulo "+mod);

Sembra strano questo, anche in considerazione del fatto che abbiamo detto che le espressioni Java sono ben tipate a differenza del C, questo avviene però perché il sistema effettua delle conversioni implicite di tipi primitivi e oggetti in stringhe. Ad esempio definendo un qualsiasi oggetto, se ne definisco un metodo toString (), il sistema che si troverà a lavorare con stringhe e con quest'oggetto ne invocherà automaticamente il metodo toString().

Java inoltre offre altri quattro operatori che sono delle abbreviazioni, due di incremento di variabili e due di decremento, sia X una variabile, io posso scrivere:

X++
++X
X--
--X

che stanno rispettivamente ad indicare valuta prima e poi incrementa X, incrementa prima e poi valuta X, valuta e decrementa X ed infine decrementa e valuta X, questo vuol dire che l'espressione X++ è un comando di assegnamento, ma anche una espressione che restituisce un risultato, risultato che in questo caso è X prima di essere incrementata.

Ad esempio:

X=10;
Y=X++;

mi darà come risultato X=11 e Y=10. Provate ad editare ed eseguire il seguente class incdec

```
{

public static void main(String [] a)
{

int X,Y,Z,W,V;

X=10;

System.out.println("X="+X);

Y=X++;

System.out.println("Y=X++: ho X="+X+",Y="+Y);

Z=++Y;

System.out.println("Z=++Y: ho Z="+Z+",Y="+Y);

W=Z--;

System.out.println("W=Z--: ho W="+W+",Z="+Z);

V=--W;

System.out.println("V=--W: ho V="+V+",W="+W);

}
}
```

Altri operatori sono quelli di confronto, ovvero:

> maggiore di, x>y restituisce true se x è maggiore stretto di y, false altrimenti

>= maggiore o uguale di, x>=y restituisce true se x è maggiore o uguale di y, false altrimenti

< minore di, x<y restituisce true se x è minore stretto di y, false altrimenti

<= minore o uguale di, x<=y restituisce true se x è minore o uguale di y, false altrimenti

== uguale, x==y restituisce true se x uguale a y, false altrimenti

!= diverso, x!=y restituisce true se x diverso da y, false altrimenti

Vi sono gli operatori logici &&, || e !, che rappresentano l'and, l'or e il not: Date x e y valori booleani, x && y sarà vera se x e y sono entrambe vere, sarà falsa altrimenti (se tutte e due false o se una vera e l'altra falsa). x || y sarà vera se x oppure y o entrambe sono vere (sarà falsa solo se sono ambedue false). !x sarà vera se x è falsa, mentre sarà falsa se x è vera.
Vi sono gli ovvi operatori binari orientati ai bit:

&, AND bit a bit
| , OR bit a bit
^, OR ESCLUSIVO, bit a bit
e gli shift, >>, <<, >>>.

Queste operazioni binarie sono molto utili nella costruzione di maschere.

L' ultimo operatore da vedere è l'operatore condizionale, che permette di definire espressioni con due differenti risultati a seconda di un valore booleano.
Si consideri ad esempio l'espressione seguente:

int ValoreAssoluto = (a<0 ? -a : a);

ValoreAssoluto assumerà il valore -a se a è negativo oppure a se è positivo o nullo.
A questo punto siamo in grado di scrivere tutte le espressioni che vogliamo nel linguaggio Java.

LEZIONE 9: *Istruzioni*

Siamo pronti per vedere i costrutti principali del linguaggio Java. Abbiamo già visto i commenti, i quali non possono essere annidati, ovvero è illecito il seguente commento: /* Questo è sicuramente un commento /* Anche questo lo è */ */ Si possono però mettere commenti di una linea nei commenti di più linee, perché sono diversi, è lecito ad esempio il seguente: /* Questo è sicuramente un commento // Anche questo lo è */ anche se francamente è abbastanza inutile. Le prime istruzioni che vedremo di Java sono le istruzioni di dichiarazione, che sono delle dichiarazioni di variabili locali, vengono usate per dichiarare variabili e assegnare loro un eventuale valore iniziale. Queste istruzioni le abbiamo già viste, e sono simili alla dichiarazione di attributi, tranne che per la mancanza di modificatori. Le variabili locali prima di essere usate devono essere inizializzate.
Le variabili locali possono essere dichiarate in un punto qualsiasi di un blocco, e non per forza all'inizio, ed esse esistono solo all'interno del blocco in cui sono dichiarate. Un blocco è come una istruzione, solo che inizia con { e finisce con }, ed all'interno può contenere tante istruzioni, anche eventualmente altri blocchi.

```
{
istruzione 1;
istruzione 2;
….
istruzione j-1;
{
sottoistruzione1;
sottoistruzione2;
…..
sottoistruzionen;
}; // è l'istruzione j
istruzione j+1;
….
istruzione n;
}// Questo blocco può essere il corpo di un metodo oppure un blocco
Pensate al medodo:
void pippo()
{
int j=0;
```

```
{
int k=0;
k++; // istruzione A
};
j++; // istruzione B
k++; // istruzione C
}
```

L'istruzione A non da' problemi, in quanto usa una variabile locale inizializzata, dichiarata all'interno del suo stesso blocco.
L'istruzione B lo stesso, mentre l'istruzione C è errata, in quanto la variabile che usa è dichiarata in un blocco interno e non è visibile all'esteno. Se nel blocco interno avessi scritto j++; non avrei avuto problemi perché la variabile j era stata dichiarata in un blocco esterno (Dentro si vede quello che c'è fuori ma fuori non si vede quello che c'è dentro, come accade in tutti i linguaggi di programmazione a blocchi).
Abbiamo visto nell'esempio che anche j++ è una istruzione, essa è sia una espressione che una istruzione come la sua altra forma e ++j e la sua speculare di decremento. Altre sono le espressioni che si possono considerare delle istruzioni, esse sono le chiamate di metodi che restituiscano o meno un valore, e le istruzioni di creazione di oggetti, quelle con new, ad esempio new Object(); è una istruzione valida.
Un'altra istruzione molto simile ad una espressione è l'istruzione di assegnamento, quella del tipo:

NomeVariabile = espressione;

Quindi fino ad ora abbiamo visto le prime sei istruzioni:

- Dichiarazione di variabile locale;
- Blocco;
- Forme di incremento e decremento di variabili, prefisse e postfisse;
- Creazione di oggetti;
- Invocazione di metodi;
- Assegnamento a variabili;

Tutte le istruzioni terminano con un ;

Le altre istruzioni sono dette costrutti, e sono molto simili a quelle degli altri linguaggi di programmazione, le vediamo una ad una.

Istruzione condizionale
if (EXPBOOL) IST
Questa istruzione esegue l'istruzione denominata IST, la quale può essere una qualsiasi istruzione del linguaggio, solo se l'espressione booleana EXPBOOL è vera, altrimenti viene saltata. Una variante è quella in cui se invece EXPBOOL è falsa viene eseguita un'altra istruzione, essa è:

```
if (EXPBOOL) IST
    else
    IST2
```

Da notare che IST e IST2 sono istruzioni generiche, quindi anche blocchi o altre istruzioni condizionali, in questo secondo caso si parla di if annidati.

Istruzione switch

```
Switch (EXP)
{
case CST11: …. :case CST1n: IST1
case CST21: …. : case CST2n: IST2
….
case CSTm1: …. : case CSTmn: ISTm
default ISTm+1;
};
```

Dove EXP è una espressione, CSTij sono delle costanti dello stesso tipo dell'espressione EXP e ISTi sono delle istruzioni.
L'istruzione valuta EXP, e confronta il risultato con le costanti dei case, se ne trova una uguale esegue l'istruzione corrispondente. Se invece non trova nessuna costante uguale esegue l'istruzione dopo il default, la quale è opzionale, nel caso in cui è assente e nessun case ha la costante uguale al valore di EXP si salta lo switch.

Si consideri ad esempio il seguente comando:

```
switch (5+1)
{
case 6 : System.out.println ("Bravo");
default : System.out.println ("Asino, è sei");
};
```

Istruzione for *for (exp di inizializzazione; exb booleana; exp di incremento) ISTR*
esegue l'istruzione ISTR un numero di volte pari ai valori contenuti in un intervallo. Tipicamente l'istruzione di inizializzazione inizializza una variabile ad un valore, variabile che viene incrementata (o decrementata) dalla istruzione di incremento e nell' espresione booleana viene controllato che la variabile assuma i valori voluti, infatti appena l'espressione booleana è falsa si esce dal ciclo for.
Faccio un esempio, voglio inizializzare i valori di un vettore di 100 interi tutti a zero, invece di scrivere 100 assegnamenti farò:

```
int v = new int[100];
for (int i = 0 ; i<100; i++) v[i] = 0 ;
```

Come si vede la variabile si può anche dichiarare nell'istruzione di inizializzazione, così si vedrà solo all'interno del for.

Istruzione while
while (EXPBOOL) IST
Esegue IST fintanto che EXPBOOL è vera, ad esempio l'inizializzazione di prima si può fare con il while così:

```
int v = new int[100];
int i=0;
while (i<100) {
v[i]=0;
i+=1;
};
```

oppure in forma più compatta, sfruttando l'incremento in forma compatta:

```
int v = new int[100];
int i=0;
while (i<100) v[i++]=0;
```

Istruzione do-while *do IST while (EXPBOOL);*
E' come il while, solo che prima esegue l'istruzione e poi controlla l'EXPBOOL, di nuovo l'inizializzazione precedente diviene:

```
int v = new int[100];
int i=0;
do {
v[i]=0;
i+=1;
} while (i<100);
```

oppure ancora in modo più compatto:

```
int v = new int[100];
int i=0;
do v[i++] = 0 while (i<100);
```

Etichette
E' possibile etichettare delle istruzioni, come in codice assembler, questo è utile in combinazione con istruzioni di break e di continue:

```
etichetta: istruzione;
```

Istruzione di break
L'istruzione break viene utilizzata per uscire da un blocco o da uno switch, quando si incontra un break si

esce dal blocco più interno, mentre se si mette una etichetta, si esce fino a dove è dichiarata l'etichetta.

Esempio:

```
{
while ( COND )
{
while (COND2)
{
if (QUALCOSA) break;
}
}
}
```

Si esce in questo caso dal while interno, mentre:

```
{
pippo: while ( COND )
{
while (COND2)
{
if (QUALCOSA) break pippo;
}
}
}
```

si esce fuori da tutti e due i while.

Istruzione di continue
In un ciclo salta tutte le istruzioni che lo seguono e va a valutare direttamente l'espressione booleana, ad esempio:

```
while (Condizione)
{
istruzioni;
if (ColpoDiScena) continue;
altre istruzioni;
};
```

Si entra nel ciclo , si eseguno istruzioni, se ColpoDiScena è true salta altre istruzioni e riva a valutare Condizione, se è ancora vera continua il ciclo eseguiendo istruzioni , ecc..

Istruzione di return
Provoca la conclusione dell'esecuzione del metodo in cui si trova e il ritorno al chiamante. Se il metodo è void basta scrivere return; , altrimenti deve contenere una espressione dello stesso tipo del metodo, ad esempio

```
int pippo()
{
return 10;
}
```

con l'istruzione return è possibile anche bloccare l'esecuzione di un costruttore o di un metodo static.

Abbiamo finito la nostra panoramica sulle istruzioni, siamo pronti ora a costruire una qualsiasi applicazione a consolle.
Facciamo un paio di esempi, iniziamo a scrivere un programmino che stampi i primi numeri, quelli minori di 500 della famosa serie di Fibonacci. Per chi non la conoscesse la serie di Fibonacci è una sequenza infinita di numeri, i cui primi due elementi sono: 1, 1, e gli altri elementi vengono calcolati come somma dei precedenti due, ad esempi, l'i-esimo numero di Fibonacci è dato dalla somma dell' i-1 esimo numero e dell' i-2 esimo, formalmente, vedendo Fibo come una funzione:

Fibo (i) = Fibo (i - 1) + Fibo (i - 2);

Il programma da scrivere in Fibo.java è il seguente:

```
class Fibo
```

```
{

public static void main (String[] args)

{

int basso=1, alto=1;

// int alto=1;

System.out.println (basso);

while (alto<500)
{

System.out.println(alto);

alto+=basso;

basso = alto - basso;

};

}

}
```

Per esercizio potreste provare a fare un programma che calcola 10 ! , sapendo che 1! = 1 e che dato n intero, n! = n * ((n-1)!). La soluzione è contenuta nel file Fatt.java.
Ora proviamo a scrivere un nostro crivello di Eratostene fatto in casa, ovvero un modo per calcolare tutti i numeri primi fino ad un dato numero, sia questo ad esempio 100;
L'algoritmo procede così,

1 è primo.
2 è primo, tolgo i suoi multipli
3 è primo, tolgo i suoi multipli

e così via fino a trovare il mio massimo.

```
class Eratostene
{
static int MAX=1000;

int [] numeri = new int[MAX];

boolean [] primi = new boolean[MAX];

public Eratostene()
{

int i;

for (i=0;i<MAX;i++)
{
numeri[i]=i+1;

primi[i]=true;

};

}

void elMul(int a)
{
```

```
for (int j=2;j*a<=MAX;j++)
primi[(j*a)-1]=false;

}

int prossimo(int n)
{
int tmp=n;

while (!primi[tmp]) {
tmp++;
if (tmp>=MAX) break;
}

return tmp+1;

}

void calcolaPrimi()
{
int num=2;

while (num<=MAX)
{
elMul(num);
num=prossimo(num);
};

}

void scriviPrimi()

{

System.out.println("I numeri primi fino a "+MAX+" sono:");

for (int i=0; i < MAX ; i++)
if (primi[i]) System.out.print(numeri[i]+" ");

}

public static void main(String[] args)
{

Eratostene e = new Eratostene();

e.calcolaPrimi();

e.scriviPrimi();

}

}
```

LEZIONE 10: Eccezioni e cenni sui thread

Le eccezioni sono un modo chiaro per controllare gli errori, senza confondere il codice con tante istruzioni di controllo dell'errore. Quando si verifica una situazione di errore viene lanciata una eccezione, che se viene in seguito catturata permette di gestire l'errore, altrimenti viene eseguita dal supporto a tempo di esecuzione di Java una ruotine di default.
Esiste in Java una classe Exception, e le eccezioni sono degli oggetti appartenenti a questa classe, quindi esse possono essere trattate come veri e propri componenti del linguaggio.
Vediamo come è possibile creare una eccezione.
Si deve creare una classe che estende Throwable, o meglio Exception , la quale è una estensione di Throwable. Supponiamo di volere creare una eccezione per dire che riferiamo una stringa vuota, scriveremo:

```
public class ErroreStringaVuota extends Exception
{
ErroreStringaVuota()
{
super("Attenzione, stai riferendo una stringa non inizializzata");
}
}
```

Questa classe si deve trovare in un file chiamato come la classe, ovvero ErroreStringaVuota.java Dopo averla creata vorremmo un modo per lanciarla, ovvero per mettere in risalto la situazione di errore, che a questo punto spero sia catturata da qualcuno. Le eccezioni vengono lanciate con l'istruzione throw, la quale deve essere in un metodo che deve essere dichiarato con una clausola throws, ad esempio, nel nostro caso se voglio creare un metodo di stampa della stringa che controlla anche l'errore, scriverò nella mia classe:

```
public void Stampa (String a)
throws ErroreStringaVuota /* ß Questo vuol dire che questa funzione può lanciare una eccezione di tipo
ErroreStringaVuota*/
{
if (a==null) throw new ErroreStringaVuota();
else System.out.println(a);
}
```

Quindi a questo punto il metodo Stampa manderà sullo schermo la stringa passatagli come parametro, e genererà una eccezione di tipo ErroreStringaVuota se il puntatore era null.
L'eccezione lanciata da noi si dice eccezione controllata, questa deve essere obbligatoriamente gestita, mentre quelle lanciate da Java non per forza devono essere catturate e gestite.
Vediamo ora come catturare le eccezioni, ad esempio la nostra EccezioneStringaVuota.
Le eccezioni vengono catturate quando vengono invocati i metodi che sono dichiarati con la clausola throws, per catturarle bisogna invocare il metodo in blocchi try, essi hanno la sintassi seguente:

```
try
BLOCCO // Questo è un blocco pericoloso, può lanciare eccezioni. catch (Tipo di Eccezione da catturare)
BLOCCO // Questo è un blocco di ripristino. catch (Tipo di Eccezione da catturare)
BLOCCO // Questo è un blocco di ripristino.
….
finally
BLOCCO
```

Il corpo di questa istruzione viene eseguito fino alla fine oppure fino alla visione di una eccezione, in caso di eccezione si esaminano le clausole catch per vedere se esiste un gestore per quella eccezione o per una sua superclasse, se nessuna clausola catch cattura l'eccezione, essa viene lanciata nel metodo che l'ha provocata (che l'ha lanciata).
Se nel try è presente una clausola finally, il suo codice viene eseguito dopo aver completato tutte le altre operazioni del try, indipendentemente dal fatto che questa abbiano lanciato una eccezione o no. Ad esempio il nostro metodo Stampa può essere chiamato così:

Sia X la stringa da stampare:

```
try {Stampa(X);}
catch (ErroreStringaVuota e)
{System.out.println ("Spiacente");}
```

In questo caso se X è null verrà lanciata l'eccezione e catturata, quindi verrà stampata la stringa Spiacente,

altrimenti X. Se avessi scritto Stampa(X) fuori dalla try avrei avuto errore, perché le eccezioni generate da me, devono obbligatoriamente essere catturate, questo obbliga il programmatore a scrivere codice "buono". Le eccezioni di Java possono essere invece catturate oppure no.
Il seguente esempio riprende il CiaoMondo del primo capitolo, che se invocato senza parametri dava una eccezione di ArrayIndexOutOfBoundsException, che questa volta viene gestita;

```
class CiaoMondo
{

public static void main(String[] args)
{

System.out.print ("Ciao mondo, sono il primo programma in Java ");

String Nome;

String Cognome;

try {Nome=args[0];}
catch (ArrayIndexOutOfBoundsException e)
{Nome="Non hai inserito il tuo Nome";}

;
try {Cognome=args[1];}
catch (ArrayIndexOutOfBoundsException e)
{Cognome="Non hai inserito il tuo Cognome";}
;

System.out.println ("di "+Nome+" "+Cognome);
}
}
```

Provatelo, e provate ad eseguirlo nei seguenti modi:

```
java CiaoMondo Nome Cognome
java CiaoMondo Nome
java CiaoMondo
```

e vedete cosa succede, fate lo stesso con il CiaoMondo del primo capitolo.
Le eccezioni sono un modo chiaro per controllare gli errori, senza confondere il codice con tante istruzioni di controllo dell'errore. Quando si verifica una situazione di errore viene lanciata una eccezione, che se viene in seguito catturata permette di gestire l'errore, altrimenti viene eseguita dal supporto a tempo di esecuzione di Java una ruotine di default.
Esiste in Java una classe Exception, e le eccezioni sono degli oggetti appartenenti a questa classe, quindi esse possono essere trattate come veri e propri componenti del linguaggio.

Vediamo come è possibile creare una eccezione.
Si deve creare una classe che estende Throwable, o meglio Exception , la quale è una estensione di Throwable. Supponiamo di volere creare una eccezione per dire che riferiamo una stringa vuota, scriveremo:

```
public class ErroreStringaVuota extends Exception
{
ErroreStringaVuota()
{
super("Attenzione, stai riferendo una stringa non inizializzata");
}
}
```

Questa classe si deve trovare in un file chiamato come la classe, ovvero ErroreStringaVuota.java Dopo averla creata vorremmo un modo per lanciarla, ovvero per mettere in risalto la situazione di errore, che a questo punto spero sia catturata da qualcuno. Le eccezioni vengono lanciate con l'istruzione throw, la quale deve essere in un metodo che deve essere dichiarato con una clausola throws, ad esempio, nel nostro caso se voglio creare un metodo di stampa della stringa che controlla anche l'errore, scriverò nella mia classe:

```
public void Stampa (String a)
```

```
throws ErroreStringaVuota /* ß Questo vuol dire che questa funzione può lanciare una eccezione di tipo ErroreStringaVuota*/
{
if (a==null) throw new ErroreStringaVuota();
else System.out.println(a);
}
```

Quindi a questo punto il metodo Stampa manderà sullo schermo la stringa passatagli come parametro, e genererà una eccezione di tipo ErroreStringaVuota se il puntatore era null.
L'eccezione lanciata da noi si dice eccezione controllata, questa deve essere obbligatoriamente gestita, mentre quelle lanciate da Java non per forza devono essere catturate e gestite.
Vediamo ora come catturare le eccezioni, ad esempio la nostra EccezioneStringaVuota.
Le eccezioni vengono catturate quando vengono invocati i metodi che sono dichiarati con la clausola throws, per catturarle bisogna invocare il metodo in blocchi try, essi hanno la sintassi seguente:

```
try
BLOCCO // Questo è un blocco pericoloso, può lanciare eccezioni.
catch (Tipo di Eccezione da catturare)
BLOCCO // Questo è un blocco di ripristino.
catch (Tipo di Eccezione da catturare)
BLOCCO // Questo è un blocco di ripristino.
….
finally
BLOCCO
```

Il corpo di questa istruzione viene eseguito fino alla fine oppure fino alla visione di una eccezione, in caso di eccezione si esaminano le clausole catch per vedere se esiste un gestore per quella eccezione o per una sua superclasse, se nessuna clausola catch cattura l'eccezione, essa viene lanciata nel metodo che l'ha provocata (che l'ha lanciata).
Se nel try è presente una clausola finally, il suo codice viene eseguito dopo aver completato tutte le altre operazioni del try, indipendentemente dal fatto che questa abbiano lanciato una eccezione o no. Ad esempio il nostro metodo Stampa può essere chiamato così:

Sia X la stringa da stampare:

```
try {Stampa(X);}
catch (ErroreStringaVuota e)
{System.out.println ("Spiacente");}
```

In questo caso se X è null verrà lanciata l'eccezione e catturata, quindi verrà stampata la stringa Spiacente, altrimenti X. Se avessi scritto Stampa(X) fuori dalla try avrei avuto errore, perché le eccezioni generate da me, devono obbligatoriamente essere catturate, questo obbliga il programmatore a scrivere codice "buono".
Le eccezioni di Java possono essere invece catturate oppure no.
Il seguente esempio riprende il CiaoMondo del primo capitolo, che se invocato senza parametri dava una eccezione di ArrayIndexOutOfBoundsException, che questa volta viene gestita;

```
class CiaoMondo
{

public static void main(String[] args)
{

System.out.print ("Ciao mondo, sono il primo programma in Java ");

String Nome;

String Cognome;

try {Nome=args[0];}
catch (ArrayIndexOutOfBoundsException e)
{Nome="Non hai inserito il tuo Nome";}

;
try {Cognome=args[1];}
catch (ArrayIndexOutOfBoundsException e)
```

```
{Cognome="Non hai inserito il tuo Cognome";}
;

System.out.println ("di "+Nome+" "+Cognome);
}
}
```

Provatelo, e provate ad eseguirlo nei seguenti modi:

```
java CiaoMondo Nome Cognome
java CiaoMondo Nome
java CiaoMondo
```

e vedete cosa succede, fate lo stesso con il CiaoMondo del primo capitolo.

LEZIONE 11: Il package java.lang

Il Package lang

Questo package lo abbiamo già visto precedentemente, adesso però ne esaminiamo alcune funzionalità che non abbiamo visto. Innanzitutto c'è da dire che questo è uno dei package più importanti dell'API di Java. Esso contiene moltissime classi e interfacce fondamentali per la programmazione Java, tanto che questo package viene automaticamente incluso nei nostri porgrammi. Il suo contenuto è il seguente:

Intefacce
Cloneable
Comparable
Runnable

Classi
Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
Class
ClassLoader
Compiler
Double
Float
InheritableThreadLocal
Integer
Long
Math
Number
Object
Package
Process
Runtime
RuntimePermission
SecurityManager
Short
String
StringBuffer
System
Thread
ThreadGroup
ThreadLocal
Throwable
Void

Eccezioni
ArithmeticException
ArrayIndexOutOfBoundsException

ArrayStoreException
ClassCastException
ClassNotFoundException
CloneNotSupportedException
Exception
IllegalAccessException
IllegalArgumentException
IllegalMonitorStateException
IllegalStateException
IllegalThreadStateException
IndexOutOfBoundsException
InstantiationException
InterruptedException
NegativeArraySizeException
NoSuchFieldException
NoSuchMethodException
NullPointerException
NumberFormatException
RuntimeException
SecurityException
StringIndexOutOfBoundsException
UnsupportedOperationException

Errori
AbstractMethodError
ClassCircularityError
ClassFormatError
Error
ExceptionInInitializerError
IllegalAccessError
IncompatibleClassChangeError
InstantiationError
InternalError
LinkageError
NoClassDefFoundError
NoSuchFieldError
NoSuchMethodError
OutOfMemoryError
StackOverflowError
ThreadDeath
UnknownError
UnsatisfiedLinkError
UnsupportedClassVersionError
VerifyError
VirtualMachineError

Nel primo capitolo abbiamo visto le classi involucro per i tipi primitivi inclese in questo package, in questa sezione esamineremo alcune classi del package che fanno delle cose più complesse.
La prima classe che vediamo è la classe System, la quale interfaccia il nostro programma Java con il sistema sul quale è eseguito. Innanzitutto la classe System contiene tre attributi statici, essi sono:

static PrintStream err static InputStream in static PrintStream out

questi rappresentano rispettivamente lo standard error, lo standard input e lo standard output (che abbbiamo già usato), sono degli attributi particolari, infatti essi sono di tipo classe, quindi possono essere referenziati per prendere i metodi della classe, ad esempio, nella scrittura:

System.out.println("Ciao");

viene invocato il metodo println(String) della classe PrintStream, perché out è di tipo PrintStream. Nei nostri esempi abbiamo usato indifferentemente le espressioni:

System.out.println("Ciao");
System.out.println(true);
System.out.println(10);

System.out.println('C');

In generale in Java questo non è possibile perché un metodo va invocato con il parametro attuale dello stesso tipo del parametro formale, questa non è una eccezione, infatti invocare il metodo println con tutti questi tipi di dato diversi è possibile perché PrintStream contiene tutti i metodi println dichiarati con il parametro formale adatto, infatti essa contiene:

void println()
void println(boolean x)
void println(char x)
void println(char[] x)
void println(double x)
void println(float x)
void println(int x)
void println(long x)
void println(Object x)
void println(String x)

gli stessi metodi print, oltre ovviamente ad altri.

Lo stesso vale per err, essendo anch'esso come out di tipo PrintStream, questo viene usato per segnalare gli errori a programma, è facile trovare nei programmi espressioni del tipo:

```
try {System.out.println(stringa+" vs "+altrastringa+" = " +stringa.compareTo(altrastringa));}
catch (nullPointerException e)
{System.err.println("ERRORE: La seconda stringa stringa è nulla");};
```

L'attributo in è invece di tipo InputStream, esso rappresenta come detto lo standard input.
I metodi per agire su di esso saranno quindi delle read:

```
abstract int read()
int read(byte[] b)
int read(byte[] b, int off, int len)
```

Siccome è di tipo InputStream esso legge dei byte, per fargli leggere altre cose dobbiamo specializzarlo in qualche altra cosa, nell'esempio seguente, che digiteremo in un file chiamato StandardIO.java, gli faremo leggere intere linee.

```
import java.io.*;
class StandardIO
{
public static void main(String[] argomenti)
{

System.out.println("Inserischi i tuoi input, batti end [INVIO] per uscire");

InputStreamReader a=new InputStreamReader(System.in);

BufferedReader IN=new BufferedReader(a);

String s=new String();

while(s.compareTo("end")!=0)
{

try {s=IN.readLine();}
catch (IOException e)
{s="ERRORE DI LETTURA";};

System.out.println("Letto: "+s);

}
}
}
```

Vediamo alcuni metodi della classe System:

static long currentTimeMillis(), questo metodo restituisce il tempo corrente in millisecondi
static void exit(int status), questo metodo determina l'uscita dalla Java Virtual Machine, con un codice .
static void gc(), Java alloca tanti oggetti, e li disalloca quando non sono più usati e serve nuova memoria, per farlo usa un cosidetto Garbage Collector , questo metodo esegue in qualsiasi momento il garbage collector per liberare la memoria ancora allocata dal programma non usata, può essere molto utile in grosse applicazioni.
static String getenv(String name), restituisce una stringa contenente informazioni riguardo al sistema su cui si esegue il programma, il metodo è dichiarato deprecated sulle ultime Documentazioni on line del Java Development Kit, questo perché il metodo presto scomparirà, esso è delle prime versioni del linguaggio, rimane ancora in queste per una questione di compatibilità con il vecchio software.
static Properties getProperties() , è il nuovo metodo per avere informazioni sulle proprietà del sistema.
static String getProperty(String key), come quello di prima, ma solo che prende informazioni specifiche riguardanti key
static String getProperty(String key, String def)
static void load(String filename) , carica in memoria il codice contenuto in filename, che è una libreria dinamica.
static void loadLibrary(String libname), carica la libreria di sistema indicata da libname
static String mapLibraryName(String libname), mappa una libreria in una stringa che la rappresenta.

Ci sono i metodi per riassegnare gli standard input, output ed err, in altri flussi, ad esempio file.

static void setErr(PrintStream err)
static void setIn(InputStream in)
static void setOut(PrintStream out)

I seguenti metodi settano le proprietà del sistema.

static void setProperties(Properties props)
static String setProperty(String key, String value)

Abbiamo usato degli oggetti di tipo Properties, vediamo come sono fatti: essi sono delle specializzazioni di tabelle hash di java, sono sostanzialmente delle tabelle che hanno una serie di coppie Chiave - Valore (In effetti non sono proprio così, sono delle strutture molto usate in informatica per contenere dati, però a noi basta vederle così per adesso). In particolare le Properties del sistema sono:

Chiave	**Descrizione del valore associato**
java.version	Versione dell'ambiente di Java Runtime
java.vendor	Distributore dell'ambiente di Java Runtime
java.vendor.url	URL del distributore di Java
java.home	Directory dove è installato Java
java.vm.specification.version	Versione delle specifiche della Java Virtual Machine
java.vm.specification.vendor	Distributore delle specifiche della Java Virtual Machine
	Nome delle specifiche della Java Virtual Machine
java.vm.version	Versione della implementazione della Java Virtual Machine
java.vm.vendor	Distributore della implementazione della Java Virtual Machine
java.vm.name	Nome della implementazione della Java Virtual Machine
java.specification.version	Versione dell'ambiente di Java Runtime

java.specification.vendor	Distributore dell'ambiente di Java Runtime Java Runtime
java.specification.name	Nome dell'ambiente di Java Runtime
java.class.version	Versione delle classi di Java
java.class.path	Pathname delle classi di Java
os.name	Nome Sistema Operativo
os.arch	Architettura del Sistema Operativo
os.version	Versione del sistema Operativo
file.separator	Separatore di File ("/" su UNIX, "\" su Windows)
path.separator	Separatore di Path (":" su UNIX, ";"su Windows)
line.separator	New Line ("\n" su UNIX e Windows)
user.name	Account name dell'utente
user.home	Home directory dell'utente
user.dir	Working directory dell'utente

Potremmo quindi scrivere un programmino che ci da informazioni sul sistema, come il seguente, da editare in un file chiamato Sistema.java

```
import java.io.*;

public class Sistema
{

public static void main(String[] arg)
{

// Cambio lo standard output, uso il file Sistema.txt

File outFile=new File("Sistema.txt");

FileOutputStream fw;
try {fw=new FileOutputStream(outFile) ;}
catch (IOException e)
{fw=null;};


PrintStream Output=new PrintStream(fw);

System.setOut(Output);


// Scrivo sul nuovo standard output:

// Tempo:
long tempo=System.currentTimeMillis();

System.out.println("Tempo in millisecondi: "+tempo);

long t1=tempo/1000;

System.out.println("Tempo in secondi: "+t1);

long sec=t1%60;

long t3=t1/60;

long min=t3%60;
```

```
long t4=t3/60;

System.out.println("Tempo in ore h"+t4+" m"+min+" s"+sec);

System.out.println("\nE' il tempo passato dal 1/1/1970 ad ora.\n");

// Proprietà del sistema:

System.out.println("\nProprieta' del sistema:\n");
String tmp;

System.out.println("\n\tJAVA\n");

tmp=System.getProperty("java.version ");
System.out.println("Versione dell'ambiente di Java Runtime: "+tmp);

tmp=System.getProperty("java.vendor");
System.out.println("Distributore dell'ambiente di Java Runtime: "+tmp);

tmp=System.getProperty("java.vendor.url");
System.out.println("URL del distributore di Java: "+tmp);

tmp=System.getProperty("java.home");
System.out.println("Directory dove e' installato Java: "+tmp);

tmp=System.getProperty("java.vm.specification.version");
System.out.println("Versione delle specifiche della Java Virtual Machine: "+tmp);

tmp=System.getProperty("java.vm.specification.vendor");
System.out.println("Distributore delle specifiche della Java Virtual Machine: "+tmp);

tmp=System.getProperty("java.vm.specification.name");
System.out.println("Nome delle specifiche della Java Virtual Machine: "+tmp);

tmp=System.getProperty("java.vm.version");
System.out.println("Versione della implementazione della Java Virtual Machine: "+tmp);

tmp=System.getProperty("java.vm.vendor" );
System.out.println("Distributore della implementazione della Java Virtual Machine: "+tmp);

tmp=System.getProperty("java.vm.name");
System.out.println("Nome della implementazione della Java Virtual Machine: "+tmp);

tmp=System.getProperty("java.specification.version");
System.out.println("Versione dell'ambiente di Java Runtime: "+tmp);

tmp=System.getProperty("java.specification.vendor");
System.out.println("Distributore dell'ambiente di Java Runtime Java Runtime: "+tmp);

tmp=System.getProperty("java.specification.name" );
System.out.println("Nome dell'ambiente di Java Runtime: "+tmp);

System.out.println("\n\tCLASSI\n");

tmp=System.getProperty("java.class.version");
System.out.println("Versione delle classi di Java: "+tmp);


tmp=System.getProperty("java.class.path");
System.out.println("Pathname delle classi di Java: "+tmp);

System.out.println("\n\tSISTEMA OPERATIVO\n");
```

```
tmp=System.getProperty("os.name");
System.out.println("Nome Sistema Operativo: "+tmp);


tmp=System.getProperty("os.arch");
System.out.println("Architettura del Sistema Operativo: "+tmp);


tmp=System.getProperty("os.version");
System.out.println("Versione del sistema Operativo: "+tmp);

tmp=System.getProperty("file.separator");
System.out.println("Separatore di File: "+tmp);


tmp=System.getProperty("path.separator");
System.out.println("Separatore di Pathname: "+tmp);


tmp=System.getProperty("line.separator");
System.out.println("New Line: "+tmp);

System.out.println("\n\tUTENTE\n");

tmp=System.getProperty("user.name");
System.out.println("Account name dell'utente: "+tmp);

tmp=System.getProperty("user.home");
System.out.println("Home directory dell'utente: "+tmp);

tmp=System.getProperty("user.dir");
System.out.println("Working directory dell'utente: "+tmp);


}

}
```

Eseguito sul mio sistema, l'output è stato (Sistema.txt):

Tempo in millisecondi: 956241233430
Tempo in secondi: 956241233
Tempo in ore h265622 m33 s53

E' il tempo passato dal 1/1/1970 ad ora.

Proprieta' del sistema:

JAVA

1. Versione dell'ambiente di Java Runtime: null
2. Distributore dell'ambiente di Java Runtime: Sun Microsystems Inc.
3. URL del distributore di Java: http://java.sun.com/
4. Directory dove e' installato Java: C:\PROGRAMMI\JAVASOFT\JRE\1.3
5. Versione delle specifiche della Java Virtual Machine: 1.0
6. Distributore delle specifiche della Java Virtual Machine: Sun Microsystems Inc.
7. Nome delle specifiche della Java Virtual Machine: Java Virtual Machine Specification
8. Versione della implementazione della Java Virtual Machine: 1.3.0rc1-S
9. Distributore della implementazione della Java Virtual Machine: Sun Microsystems Inc.
10. Nome della implementazione della Java Virtual Machine: Java HotSpot(TM) Client VM
11. Versione dell'ambiente di Java Runtime: 1.3
12. Distributore dell'ambiente di Java Runtime Java Runtime: Sun Microsystems Inc.
13. Nome dell'ambiente di Java Runtime: Java Platform API Specification

CLASSI

1. Versione delle classi di Java: 47.0
2. Pathname delle classi di Java: .

SISTEMA OPERATIVO

1. Nome Sistema Operativo: Windows 95
2. Architettura del Sistema Operativo: x86
3. Versione del sistema Operativo: 4.0
4. Separatore di File: \
5. Separatore di Pathname: ;
6. New Line:

UTENTE

1. Account name dell'utente: pietro
2. Home directory dell'utente: C:\WINDOWS\Profiles\Pietr000
3. Working directory dell'utente: D:\Lavoro\HTML.IT\corso\esempi\cap3\lez6

Un'altra classe molto importante del package java.lang di Java è la classe Object, questa è la classe da cui vengono create tutte le altre classi di Java, ovvero, ogni altra classe di Java è una estensione della classe java.lang.Object. La class Object non contiene alcuna variabile, contiene 11 metodi che vengono ereditati da tutte le altre classi di Java, e possono essere usate quindi in qualsiasi oggetto definito in Java, tra cui:
clone(), che crea una copia dell'oggetto identica a quella di cui è stato invocato il metodo. In particolare è da notare che il programma Java stesso è un Oggetto, quindi si possono creare un numero arbitrario di programmi tutti identici. Affinchè possa essere invocato il metodo clone() di un oggetto, questo deve implementare l'interfaccia Cloneable .
getClass(), restituisce un Oggetto di tipo Class, che rappresenta la classe di appartenenza dell'oggetto di cui è stato invocato il metodo.
toString(), trasforma l'oggetto in una stringa, questo metodo deve essere sovrascritto negli oggetti creati dall'utente, ma è molto utile.

Si pensi alla scrittura String somma="Sommando"+10+" e "+11+" ottengo "+(10+11);
Si crea una stringa usando delle stringhe e degli int, automaticamente Java in questo caso ne invoca il metodo toString() della classe involucro corrispondente. Questo accade in ogni espressione che coinvolge stringhe e oggetti del tipo "Stringa"+Oggetto.

L'altra classe del package nominata sopra è Class, questa rappresenta le classi del linguaggio, essa è molto importante perché ha più di trenta metodi che servono per gertire le classi a runtime, cosa impensabile negli altri linguaggi di programmazione. Oggetti di questo tipo, detti descrittori di classe, anche se sono anche descrittori di interfacce, vengono automaticamente creati e associati agli oggetti a cui si riferiscono.
Oggetti di tipo Class, Object e altri tipi, sono molto importanti a runtime, perché permettono di gestire il programma che è in esecuzione come se fosse una collezione di dati su cui è possibile lavorare.
Java è un linguaggio dalle potenzialità impressionanti, si pensi che stesso in java.lang, c'è una classe chaimata Compiler, la quale contiene metodi per compilare sorgenti Java, vi sono altre classi, come ClassLoader, Runtime, che permettono di caricare nuove classi a runtime, di eseguirle, e di modificare il programma stesso mentre è eseguito, potenzialmente in Java è possibile scrivere del codice che si automodifica (che si evolve).
A questo punto termino la trattazione del java.lang, anche perché vederlo in dettaglio è troppo complicato ed è sicuramente fuori dagli scopi del nostro corso, a noi basta sapere che esiste e che fa delle cose che per gli altri linguaggi di programmazione è pura fantasia, o almeno è impossibile o difficilissimo fare ad alto livello.
Prima di terminare descrivo brevemente la classe java.lang.Math (diversa dalla classe java.math).
Questa classe serve per fare calcoli matematici, essa ha due attributi:

static double E , è la e di Eulero
static double PI, è la PI Greca

I metodi sono ovviamente tutte le funzioni matematiche calcolabili, tra cui:

1. I valori assoluti di valori double, float, int e long:
2. static double abs(double a)
3. static float abs(float a)
4. static int abs(int a)
5. static long abs(long a)

Le funzioni trigonometriche

1. static double acos(double a) - arcocoseno
2. static double asin(double a) - arcoseno
3. static double atan(double a) - arcotangente

4. static double cos(double a) - coseno
5. static double sin(double a) - seno
6. static double tan(double a) - tangente

Trasformazioni di angoli

1. static double toDegrees(double angrad) - converte radianti in gradi
2. static double toRadians(double angdeg) - converte gradi in radianti
3. static double atan2(double a, double b) - converte coordinate cartesiane (b,a) in coordinate polari (r,theta)

Funzioni esponenziali e logaritmiche

1. static double exp(double a) - e elevato alla a.
2. static double log(double a) - logaritmo in base e di a

Massimi e minimi tra valori double, float, int e long

1. static double max(double a, double b)
2. static float max(float a, float b)
3. static int max(int a, int b)
4. static long max(long a, long b)
5. static double min(double a, double b)
6. static float min(float a, float b)
7. static int min(int a, int b)
8. static long min(long a, long b)

Potenze e radici

1. static double pow(double a, double b) - calcola a elevato alla b, da notare che se b è <1, questa è
2. una radice (1/b)-esima di a
3. static double sqrt(double a) - calcola la radice quadrata di a.

Numeri pseudocasuali

1. static double random() - spara un numero casuale tra 0 e 1

Arrotondamenti

1. static double rint(double a) - parte intera bassa di a, è un intero
2. static long round(double a) - a arrotondato, è un long
3. static int round(float a) - a arrotondato, è un intero

Visto che abbiamo nominato il package java.math, diciamo che questo contiene due classi, BigInteger e BigDecimal. La prima classe serve per trattare numeri interi di grandezza arbitraria, si pensi ad esempio al calcolo del fattoriale di un numero molto grande, questo sarà un numero spaventosamente grande. BigDecimal fa lo stesso per numeri particolari.

LEZIONE 12: Il package java.util

Prima di cominciare questa lezione devo fare una piccola nota.
Abbiamo visto come programmare in Java, ma mi sono accorto che le tastiere italiane non hanno le parentesi graffe, mentre sulle tastiere che uso io ci sono. Quindi devo dirvi come fare uscire le parentesi.
Per fare apparire { bisogna premere il tasto ALT, e tenendolo premuto bisogna battere sul tastierino numericoalla destra della tastiera il numeri 123 (ALT+123).
Per fare apparire } si deve premere invece ALT+125.
Per gli altri simboli utili in Java:
ALT Gr + è ci da la [e ALT Gr + + da]. I tasti sono sottolineati per non confonderli con il + che indica "premere insieme". ALT Gr è l'ALT che si trova alla destra del tasto SPAZIO, mentre ALT si trova alla sinistra (guardando la tastiera).

Questo package è molto utile, esso mette a disposizione 34 classi e 13 interfacce che implementano alcune tra le strutture dati più comuni, alcune operazioni su date e sul calendario, e altre cose.
Inoltre il package java.util include altri sottopackages che sono: java.util.mime,java.util.zip e java.util.jar che servono rispettivamente per trattare files di tipo MIME, di tipo ZIP e di tipo Java Archive (JAR), che vedremo in seguito in questo capitolo. Per struttura dati, in informatica si intende una struttura logica atta a contenere un certo numero di dati, nella quale è possibile inserire dati, toglierli, cercarli, a limite ordinarli. Una semplice struttura dati che abbiamo gia visto è l'array, esso contiene un cero numero di dati che possono essere qualsiasi cosa, da byte ad oggetti complessi, su questa si possono fare inserzioni ricerche e cancellazioni, volendo un array può anche essere ordinato.

Per vedere se in un array grande N è presente un dato X, io devo visitare tutto l'array e fare dei confronti con ogni elemento dell'array, quindi questa è una operazione abbastanza complessa. Esistono delle strutture dati che questa ricerca la fanno impiegando un solo accesso alla struttura, queste sono ad esempio le già citate tabelle hash. La scelta delle strutture dati da usare in un programma è una scelta abbastanza complicata, infatti si sceglie una struttura anziché un'altra in base all'uso che deve esserne fatto nel programma, ad esempio è difficile che in un database molto grande in cui si fanno continue ricerche si usi un array come struttura dati, perché ogni ricerca costerà tanto in termini di tempo (il computer è veloce, ma ha anch'esso i suoi limiti, se pensiamo ad un database di un miliardo di elementi, il computer per visitarlo tutto può impiegare tanto), in questo caso è forse meglio usare una tabella hash opportunamente grande.
Java quindi ci mette a disposizione tutta una gamma di strutture dati, le gestisce lui, e a noi non resta solo che usarle. Per la scelta della struttura dati più consona al nostro scopo, useremo il buon senso, perché per sceglierla in modo rigoroso occorrono delle conoscenze molto specifiche, tra cui i dettagli realizzativi delle strutture, studiate in corsi universitari quali Algoritmi e Strutture Dati , e Fondamenti di Complessità.
Io quando descriverò le classi di Java addette all'implementazione di queste strutture, accennerò a loro, e discuterò dei tempi di inserzioni, ricerche, eccetera…, ma sempre informalmente.
Vediamo cosa contiene il package java.util

Interfacce

Collection
Comparator
Enumeration
EventListener
Iterator
List
ListIterator
Map
Map.Entry
Observer
Set
SortedMap
SortedSet

Queste interfacce stabiliscono alcune proprietà delle nostre strutture dati, esse vengono implementate in alcune delle seguenti classi.

Classi

AbstractCollection
AbstractList
AbstractMap
AbstractSequentialList
AbstractSet
ArrayList
Arrays
BitSet
Calendar
Collections
Date
Dictionary
EventObject
GregorianCalendar
HashMap
HashSet
Hashtable
LinkedList
ListResourceBundle
Locale
Observable
Properties
PropertyPermission
PropertyResourceBundle
Random
ResourceBundle
SimpleTimeZone

Stack
StringTokenizer
Timer
TimerTask
TimeZone
TreeMap
TreeSet
Vector
WeakHashMap

Eccezioni

ConcurrentModificationException
EmptyStackException
MissingResourceException
NoSuchElementException
TooManyListenersException

Iniziamo a vedere qualche struttura, vediamo BitSet, o in italiano Insieme di Bit.
Questa classe fornisce un modo per creare e gestire insiemi bit (1,0) o meglio valori true, false.
L'insieme è in effetti un vettore di bit, però che cresce dinamicamente. All'inizio i bit hanno valore false, e il vettore è lungo 2^32-1 ((due alla trentadue) meno uno).
La classe ha due costruttori: BitSet(), per creare un BitSet di dimensioni standard, e BitSet(int nbits), per creare un BitSet che contiene nbits bit.
Le operazioni che si possono effettuare sono le seguenti:

void and(BitSet set), restituisce l'and tra la BitSet e l'altra BitSet individuate da set.
void andNot(BitSet set), cancella tutti i bit della BitSet che hanno il corrispondente bit settato nella BitSet set.
void or(BitSet set), restituisce l'or inclusivotra la BitSet e l'altra BitSet individuate da set.
void xor(BitSet set) , restituisce l'or esclusivo tra la BitSet e l'altra BitSet individuate da set.
void clear(int bitIndex), setta il bit specificato a false.
void set(int bitIndex), setta il bit specificato a true.
Object clone(), BitSet è dichiarato Cloneable, questo metodo ne crea una copia uguale.
boolean equals(Object obj), confronta l'oggetto con un altro oggetto.
boolean get(int bitIndex), da il valore del bit numero bitIndex.
int hashCode(), da un codice hash per questo bitset.
int length(), da la grandezza logica del bit set, ovvero il bit più alto settato a true, più uno.
int size(), da il numero di bit attuali della bitset, è il bit più alto che può essere posto a uno senza ampliare la bit set.
String toString(), trasforma la bitset in una stringa

Proviamo la bit set con il seguente programma CaratteriUsati, da editare in un file chiamato CaratteriUsati.java, il quale prende una stringa in ingresso e vede quali caratteri sono stati usati.

```
import java.util.BitSet;
public class CaratteriUsati
{

public static void main (String[] a)
{

String tmp;

try {tmp=a[0];}
catch (ArrayIndexOutOfBoundsException e)
{errore();}

elabora(a[0]);

}

public static void errore()
{

System.out.println("ERRORE, ho bisogno di una stringa:");
```

```
System.out.println("\tBattere:");
System.out.println("\t\tjava CaratteriUsati STRINGA.");
System.exit(0);

}

public static void elabora (String a)
{

String tmp = a;

BitSet usati= new BitSet();

for (int i = 0; i < tmp.length() ; i++)
usati.set(tmp.charAt(i));

String out="[";

int dim=usati.size();

for (int i = 0; i < dim; i++)
{
if (usati.get(i))
out+=(char )i;
};

out+="]";

System.out.println(out);

System.out.println("Per l'elaborazione ho usato una bit set di "+usati.size()+" bit");

System.out.println ("\t\t\tPietro ");

}

}
```

Vediamo un'altra classe di java.util, la classe Vector. Questa classe implementa degli array di Object. La cosa interessante rispetto agli array del linguaggio, è che si può modificare la lunghezza del vettore a tempo di esecuzione.
In Vector avremo quindi oltre ai costruttori tre tipi di metodi, metodi per modificare il vettore, metodi per ottenere i valori presenti nel vettore e metodi per gestire la crescita del vettore.

Ho quattro costruttori:

Vector(), che costruisce un vettore grande 10, e con possibilità di incremento pari a zero.
Vector(Collection c), costruisce un vettore contenente gli elementi della collezione specificata, nello stesso ordine in cui appaiono nella collezione.
Vector(int initialCapacity), costruisce un vettore grande quando initialCapacity, con possibilità di Incremento pari a zero.
Vector(int initialCapacity, int capacityIncrement), costruisce un vettore grande initialCapacity, con la possibilità di incremento pari a capacityIncrement.

Alcuni metodi della classe sono i seguenti:

void add(int index, Object element), aggiunge un elemanto al vettore nella posizione indicata.
boolean add(Object o), aggiunge un elementio alla fine del vettore.
boolean addAll(Collection c), aggiunge alla fine del vettore gli elementi della collezione specificata, nello stesso ordine in cui appaiono nella collezione.
boolean addAll(int index, Collection c), aggiunge gli elementi della collezione nel vettore iniziando dall'indice specificato.
Questi tre metodi ritornano true se gli elementi sono stati aggiunti, false se non c'entrano.
void addElement(Object obj), aggiunge l'elemento al vettore, e ne incrementa la capacità di uno. int capacity (), da' la capacità attuale del vettore.

void clear(), rimuove tutti gli elementi presenti nel vettore.
Object clone(), solito metodo usato per clonare l'oggetto Vector, che è chiaramente cloneable.
boolean contains(Object elem), controlla se l'elemento specificato è presente nel vettore.
boolean containsAll(Collection c), controlla se l'intera collezione specificata è presente nel vettore.
void copyInto(Object[] anArray), copia il contenuto dell'oggeto di tipo Vector in un array di oggetti del linguaggio.
Object elementAt(int index), restituisce l'elemento del vettore che si trova nella posizione specificata.
boolean equals(Object o), controlla l'uguaglianza tra il vettore e l'oggetto specificato.
Object firstElement(), da il primo elemento del vettore, quello in posizione 0.
Object get(int index), da l'elemento del vettore che si trova nella posizione specificata, e lo cancella dal vettore.
int hashCode(), da il codice hash del vettore.
int indexOf(Object elem), cerca la prima occorrenza del dato elemento nel vettore, e ne restituisce l'indice. Per prima occorrenza intendo ovviamente quella di indice più basso.
int indexOf(Object elem, int index), cerca la prima occorrenza del dato elemento nel vettore iniziando dall'indice specificato, e ne restituisce l'indice
void insertElementAt(Object obj, int index), inserisce l'oggetto specificato nella posizione voluta del vettore.
boolean isEmpty(), controlla se il vettore è vuoto.
Object lastElement(), da l'ultimo componente del vettore.
int lastIndexOf(Object elem), da l'indice dell'ultima occorrenza dell'oggetto specificato nel vettore.
int lastIndexOf(Object elem, int index), come prima iniziando dall'indice specificato e procedendo all'indietro.
Object remove(int index), cancella l'elemento che si trova nella posizione indicata nel vettore.
boolean remove(Object o), cancella la prima occorrenza nel vettore dell'oggetto specificato.
boolean removeAll(Collection c), cancella dal vettore tutti gli elementi contenuti nella collezione specificata.
void removeAllElements(), cancella tutti gli elementi del vettore.
boolean removeElement(Object obj), cancella la prima occorrenza dell'oggetto specificato nel vettore.
void removeElementAt(int index), cancella l'elemento nella posizione specificata.
boolean retainAll(Collection c), cancella tutti gli elementi del vettore che non sono contenuti nella collezione specificata.
Object set(int index, Object element), rimpiazza l'elemento nella posizione specificata del vettore con l'elemento passatogli come parametro (mette element all'indirizzo index).
void setElementAt(Object obj, int index), mette l'oggetto specificato nella posizione voluta.
void setSize(int newSize), setta la nuova dimensione del vettore.
int size(), da il numero di posizioni nel vettore.
List subList(int fromIndex, int toIndex), crea una lista con gli elementi del vettore che si trovano dalla posizione specificata come inizio a quella come fine.
List è un'altra struttura dati molto importante, che Java ha già implementata.
Object[] toArray() e Object[] toArray(Object[] a), creano un array del linguaggio con gli elementi contenuti nel vettore.
toString(), da' una rappresentazione sotto forma di stringa del vettore, contenente le rappresentazioni sotto forma di stringa di tutti gli oggetti contenuti nel vettore, in pratica invoca il toString di tutti gli oggetti.

Chi è abituato a programmare in altri linguaggi di programmazione potrà apprezzare tutte le funzionalità dei vettori, funzionalità che spesso il programmatore deve implementarsi da solo, e che spesso lo fa in modo sommario. Quello che personalmente apprezzo io, da programmatore, è oltre alla dinamicità della struttura Vector, che è stata sempre per definizione statica, la possibilità di creare vettori con elementi eterogenei, infatti questi sono vettori di oggetti, e si sa che gli oggetti di Java possono essere qualsiasi cosa, anche programmi. Ad esempio è possibile creare un vettore contenente numeri interi, valori booleani e stringhe, cosa che non è possibile fare con gli array del linguaggio.
Per farmi capire meglio faccio un esempio.
Pensate al seguente vettore:

pippo = { true , 10 , "Ciao"};

Di che tipo sarà pippo? int[]? boolean[]? O String[]? Nessuno dei tre, in Java è impossibile creare un array così fatto, usando però le famose classi involucro, che fin'ora ci sembravano abbastanza inutili, possiamo creare un array così fatto, scriveremo qualcosa del tipo.

```
Object[] pippo={new Boolean(true),
new Integer(10),
new String("Ciao")
};
```

Se inoltre usiamo la classe Vector, abbiamo anche parecchi metodi per gestire questo vettore di elementi eterogenei.

Facciamo un piccolo esempio che usa i vettori, definiamo uno spazio geometrico tridimensionale di punti, linee e facce, così definiti:

```
class punto
{
String nome;
int x,y,z;
public punto(String n, int X,int Y,int Z)
{

x=X;

y=Y;

x=Z;

nome=n;

}

public String toString()
{
return "\n"+nome+"="+x+","+y+","+z+"\n";

}
}


class linea
{
String nome;
punto inizio;
punto fine;
public linea(String n, punto i,punto f)
{

inizio=i;

fine=f;

nome=n;

}

public String toString()
{
return "\n"+nome+"=("+inizio.toString()+","+fine.toString()+")\n";

}

}


class faccia
{
String nome;
punto x1;
punto x2;
punto x3;
linea l1;
linea l2;
linea l3;
public faccia(String n, punto X1,punto X2,punto X3)
{
```

```
x1=X1;
x2=X2;
x3=X3;
l1=new linea(n+"-linea1",x1,x2);
l2=new linea(n+"-linea2",x3,x2);
l3=new linea(n+"-linea3",x1,x3);
nome=n;

}

public String toString()
{

return "\n"+nome+"={"
+l1.toString()+
","
+l2.toString()+
","
+l3.toString()+"}\n";

}
}
```

Da notare che le classi contengono dei metodi toString(), che sovrascrivono i metodi standard, questo per creare l'output su file del programma.
Queste classi le metteremo in un file chiamato Geometria.java, ove metteremo anche le import necessarie al programma e la classe Geometria contenente il main sotto definite:

```
import java.util.*;
import java.io.*;

// Definizione della class punto
// Definizione della class linea
// Definizione della class faccia

public class Geometria
{

public static int NUMERO = 3000;

public static void main(String[] arg)
{
Vector spazio=new Vector(1,1);

System.out.println("Geometria nello spazio:");

int pti = NUMERO;

System.out.println();

System.out.println("Genero "+pti+" oggetti a caso per ogni specie");

System.out.println("Cambiare la costante NUMERO in Geometria per generarne un numero diverso.");

int lin=NUMERO;

int fac;

fac=NUMERO;


System.out.println();

int d1=spazio.capacity();

System.out.println ("Capacità del vettore:"+spazio.capacity());
```

```
System.out.println("Genero i punti...");

int i=0;

for ( i = 0 ; i
{

float a=(float ) Math.random();

float b=(float ) Math.random();

float c=(float ) Math.random();

int x=Math.round(a*1000);

int y=Math.round(b*1000);

int z=Math.round(c*1000);

String nome="Punto"+(i+1);

punto tmpP=new punto(nome,x,y,z);

spazio.addElement(tmpP);

};

System.out.println ("Capacità del vettore:"+spazio.capacity());

int d2=spazio.capacity();

System.out.println("Genero le linee...");

for ( i = 0 ; i
{

float a=(float ) Math.random();

float b=(float ) Math.random();

float c=(float ) Math.random();

float d=(float ) Math.random();

float e=(float ) Math.random();

float f=(float ) Math.random();


int x=Math.round(a*1000);

int y=Math.round(b*1000);

int z=Math.round(c*1000);

int x1=Math.round(d*1000);

int y1=Math.round(e*1000);

int z1=Math.round(f*1000);

String nome="Linea"+(i+1);

punto P1=new punto ("Punto 1 della "+nome,x,y,z);
```

```
punto P2=new punto ("Punto 2 della "+nome,x1,y1,z1);
linea tmpL=new linea(nome,P1,P2);
spazio.addElement(tmpL);
};
System.out.println ("Capacità del vettore:"+spazio.capacity());
int d3=spazio.capacity();
System.out.println("Genero le facce...");
for ( i = 0 ; i
{
float a=(float ) Math.random();
float b=(float ) Math.random();
float c=(float ) Math.random();
float d=(float ) Math.random();
float e=(float ) Math.random();
float f=(float ) Math.random();
float g=(float ) Math.random();
float h=(float ) Math.random();
float j=(float ) Math.random();
int x=Math.round(a*1000);
int y=Math.round(b*1000);
int z=Math.round(c*1000);
int x1=Math.round(d*1000);
int y1=Math.round(e*1000);
int z1=Math.round(d*1000);
int x2=Math.round(g*1000);
int y2=Math.round(h*1000);
int z2=Math.round(j*1000);
String nome="Faccia"+(i+1);
punto P1=new punto ("Punto 1 della "+nome,x,y,z);
punto P2=new punto ("Punto 2 della "+nome,x1,y1,z1);
punto P3=new punto ("Punto 1 della "+nome,x2,y2,z2);
faccia tmpF=new faccia(nome,P1,P2,P3);
spazio.addElement(tmpF);
```

```
};

System.out.println ("Capacità finale del vettore:"+spazio.capacity());

int d4=spazio.capacity();

File FileOut=new File("Geometria.txt");

FileWriter Output;

try {Output=new FileWriter(FileOut);}
catch (IOException e) {Output=null;};

try {

Output.write("Geometria.txt\nnumero oggetti="+3*NUMERO+"\n");

Output.write("Dimensioni del Vector:\nall'inizio:"+d1+"\n dopo l'inserzione dei punti:"+d2);

Output.write("\ndopo l'inserzione delle linee:"+d3+"\ndopo l'inserzione delle facce:"+d4);

Output.write("\n\nContenuto:\n");

Output.write(spazio.toString());

Output.write("\n\n\n\t\tPietro Castellucci");

Output.flush();

} catch (IOException e) {};

System.out.println("\nGuarda il file Geometria.txt.\n");

};

}
```

Se mandiamo in esecuzione il programma vediamo come il Vector spazio cresce dinamicamente, con gli oggetti eterogenei punto, linea e faccia.
Alla fine avremo un file chiamato Geometria.txt che conterrà la descrizione del vettore, il file con soli 5 elementi per ogni specie sarà qualcosa del tipo:

```
Geometria.txt
numero oggetti=15

Dimensioni del Vector:

all'inizio:1
dopo l'inserzione dei punti:5
dopo l'inserzione delle linee:10
dopo l'inserzione delle facce:15

Contenuto:
[
Punto1=653,932,0

,
Punto2=100,273,0

,
Punto3=855,210,0

,
Punto4=351,702,0

,
Punto5=188,996,0

,
```

Linea1=(
Punto 1 della Linea1=680,454,0
,
Punto 2 della Linea1=69,16,0
)
,
Linea2=(
Punto 1 della Linea2=116,651,0
,
Punto 2 della Linea2=371,15,0
)
,
Linea3=(
Punto 1 della Linea3=947,335,0
,
Punto 2 della Linea3=477,214,0
)
,
Linea4=(
Punto 1 della Linea4=391,671,0
,
Punto 2 della Linea4=692,725,0
)
,
Linea5=(
Punto 1 della Linea5=762,283,0
,
Punto 2 della Linea5=582,192,0
)
,
Faccia1={
Faccia1-linea1=(
Punto 1 della Faccia1=826,235,0
,
Punto 2 della Faccia1=13,378,0
)
,
Faccia1-linea2=(
Punto 1 della Faccia1=12,950,0
,
Punto 2 della Faccia1=13,378,0
)
,
Faccia1-linea3=(
Punto 1 della Faccia1=826,235,0
,
Punto 1 della Faccia1=12,950,0
)
}
,
Faccia2={
Faccia2-linea1=(
Punto 1 della Faccia2=382,30,0
,
Punto 2 della Faccia2=224,597,0
)
,
Faccia2-linea2=(
Punto 1 della Faccia2=277,361,0
,
Punto 2 della Faccia2=224,597,0
)
,
Faccia2-linea3=(
Punto 1 della Faccia2=382,30,0

,
Punto 1 della Faccia2=277,361,0
)
}
,
Faccia3={
Faccia3-linea1=(
Punto 1 della Faccia3=139,802,0
,
Punto 2 della Faccia3=880,935,0
)
,
Faccia3-linea2=(
Punto 1 della Faccia3=643,921,0
,
Punto 2 della Faccia3=880,935,0
)
,
Faccia3-linea3=(
Punto 1 della Faccia3=139,802,0
,
Punto 1 della Faccia3=643,921,0
)
}
,
Faccia4={
Faccia4-linea1=(
Punto 1 della Faccia4=516,614,0
,
Punto 2 della Faccia4=429,210,0
)
,
Faccia4-linea2=(
Punto 1 della Faccia4=979,860,0
,
Punto 2 della Faccia4=429,210,0
)
,
Faccia4-linea3=(
Punto 1 della Faccia4=516,614,0
,
Punto 1 della Faccia4=979,860,0
)
}
,
Faccia5={
Faccia5-linea1=(
Punto 1 della Faccia5=152,663,0
,
Punto 2 della Faccia5=828,101,0
)
,
Faccia5-linea2=(
Punto 1 della Faccia5=651,761,0
,
Punto 2 della Faccia5=828,101,0
)
,
Faccia5-linea3=(
Punto 1 della Faccia5=152,663,0
,
Punto 1 della Faccia5=651,761,0
)
}
]

Pietro Castellucci

Provate a mettere la costante NUMERO a 4000 e ad eseguire il programma.
Come avete visto abbiamo tante strutture dati in java.util, descriverle tutte sarebbe un lavoraccio.
La cosa che vi deve interessare è che tutte bene o male hanno metodi per inserire, togliere, recuperare e metodi per gestire la struttura stessa.
Le strutture dati implementate sono tabelle hash, liste, pile, code, alberi ecc… e a seconda della vostra esigenza ne userete una o l'altra, per i dettagli però vi consiglio di vedere la documentazione del Java Development Kit, dove sono descritte tutte queste classi.
Per ora vi basti sapere che le tabelle hash sono velocissime nella ricerca di un oggetto, tipicamente basta un solo accesso alla struttura dati per beccare l'oggetto cercato.
Le liste sono come i vettori, gli oggetti sono legati tra di loro, e da un oggetto è possibile raggiungere il seguente, o il precedente o tutti e due, a seconda della realizzazione della lista.
Le Pile sono selle liste particolari, dove gli oggetti vengono inseriti sempre in testa alla struttura, e da qui vengono prelevati, quindi nella pila l'ultimo oggetto entrato è il primo ad uscire, immaginate gli oggetti come delle pratiche da sbrigare da un ufficio, esse vengono poste in ordine di come arrivano sulla scrivania dell'addetto, una sull'altra. L'addetto alle pratiche prende sempre la pratica in testa alla pila e la elabora.
Questa può sembrare una struttura dati stupida, ma vi assicuro che è la struttura dati più importante di tutte, essa è utilizzata da tutti i linguaggi di programmazione per la chiamata di funzioni e di procedure (anche Java la usa per l'invocazione di dei metodi), per il passaggio dei parametri e il recupero dei risultati, e senza di questa struttura sarebbe impossibile programmare in modo ricorsivo, il quale è un modo di programmare molto potente.
Le code funzionano invece al contrario, sono paragonabili alle code in banca, gli oggetti sono le persone che fanno la fila allo sportello, esse arrivano e si mettono alla fine della fila, intanto l'operatore serve le persone che sono in testa alla fila, anche questa struttura è importantissima, essa e alcune sue varianti (le code a priorità, dove gli oggetti entrano con una priorità, ed in base a questa vengono serviti, ovvero saltano la fila) sono molto usate nei sistemi operativi come Windows, Linux, Unix, eccetera, ad esempio per gestire le richieste di stampa verso una sola stampante da parte di tutti gli utenti del sistema.
Prima di passare al prossimo package però voglio farvi vedere un piccolo paragone sulla ricerca di un oggetto tra tanti oggetti, racchiusi in strutture diverse, in particolare in strutture di tipo vettore e di tipo tabelle hash.
Vedremo come le performances della ricerca cambiano in modo evidente al crescere delle dimensioni delle strutture. Immaginiamo degli oggetti fatti così:

```
class O
{
String nome=new String();
int valore;
public O(String a, int v)
{
nome=a;
valore=v;
}
}
```

Creiamo un vettore di 100000 di questi elementi, ne cerchiamo uno e vediamo quanto tempo impieghiamo:

```
import java.util.*;

class O
{

String nome=new String();

int valore;

public O(String a, int v)
{
nome=a;
valore=v;
}

}
```

```
public class Rvettore
{

public static int NUMERO = 100000;

public static void main (String[] s)
{

Vector V = new Vector(NUMERO);

int numnome=128756;

O O_30000=null;

System.out.println("Genero il vettore, inserisco "+NUMERO+" oggetti di tipo O con valori scorrelati rispetto all'indice.");

System.out.print("Inizio: O_"+numnome);

for (int i=0; i< NUMERO;i++)
{

O tmp = new O("O_"+numnome,numnome);

if (numnome==30000) O_30000 = tmp;

numnome--;

V.add(i,tmp);

};
System.out.println(" Fine: O_"+numnome);

System.out.println("Inizio la ricerca di O_30000");

long Inizio=System.currentTimeMillis();

int index=V.indexOf(O_30000);

O tmp=(O ) V.get(index);

long Fine=System.currentTimeMillis();

System.out.println("Oggetto O_30000 trovato in "+(Fine-Inizio)+" millisec. all'indice "+index);

System.out.println("Esso vale:\nNome:"+tmp.nome+"\nValore:"+tmp.valore+"\n");

}

}
```

Il programma cercherà l'oggetto O_30000, l'output del programma è:
Genero il vettore, inserisco 100000 oggetti di tipo O con valori scorrelati rispetto all'indice.
Inizio: O_128756 Fine: O_28756
Inizio la ricerca di O_30000
Oggetto O_30000 trovato in 50 millisec. …..

Adesso faccio la stessa cosa, usando però come struttura una tabella hash:

```
import java.util.*;

class O
```

```
{

String nome=new String();

int valore;

public O(String a, int v)
{
nome=a;
valore=v;
}

}


public class Rhash
{

public static int NUMERO = 100000;

public static void main (String[] s)
{

Hashtable H = new Hashtable(NUMERO+1);

int numnome=128756;

O O_30000=null;

System.out.println("Genero la tabella, inserisco "+NUMERO+" oggetti di tipo O.");

System.out.print("Inizio: O_"+numnome);

for (int i=0; i< NUMERO;i++)
{

O tmp = new O("O_"+numnome,numnome);

if (numnome==30000) O_30000 = tmp;

numnome--;

H.put(tmp,tmp);

};
System.out.println(" Fine O_"+numnome);

System.out.println("Inizio la ricerca di O_30000");

long Inizio=System.currentTimeMillis();

O tmp=(O ) H.get(O_30000);

long Fine=System.currentTimeMillis();

System.out.println("Oggetto O_30000 trovato in "+(Fine-Inizio)+" millisec. ");

System.out.println("Esso vale:\nNome:"+tmp.nome+"\nValore:"+tmp.valore+"\n");


}

}
```

Ancora una volta il programma cercherà l'oggetto O_30000, l'output del programma questa volta è:
Genero il vettore, inserisco 100000 oggetti di tipo O.
Inizio: O_128756 Fine: O_28756
Inizio la ricerca di O_30000
Oggetto O_30000 trovato in 0 millisec. …..

Il package java.util non comprende solo queste utilissime strutture dati, esso comprende anche delle classi per facilitare la gestione di date e orari, per l'interazionalizzazione, ed altre classi di utilità, come lo string tokenizer, che prende da una stringa tutte le parole, e dei generatori di numeri casuali.

LEZIONE 13: Il package java.util

In questa lezione vedremo la parte di java.util che tratta gli utilissimi file .zip e i .jar
Iniziamo a veder java.util.zip
I file .zip sono degli archivi che contengono dei file compressi, essi sono molto usati per scambiare dati in internet perché ne riducono anche notevolmante le dimensioni.
Esistono vari tipi di file compressi, e vari programmi per comprimere e decomprimere i dati, si pensi agli archivi RAR, ai CAB di Windows, agli ARJ. Questo package ci da la possibilità di trattare dati compressi secondo gli standard ZIP e GZIP, che usano l'algoritmo di compressione chiamato DEFLATE, in questo package troviamo anche utility per controllare i codici checksum CRC-32 e Adler-32 di un arbitrario stream di ingresso.
Perché usare questo package, i motivi sono tanti, innanzitutto i programmi Java che includono immagini, animazioni e suoni possono essere veramente molto grandi, ed è possibile creare un file .ZIP con tutti i file necessari al funzionamento del programma, in modo da ridurne le dimensioni, però così facendo per essere utilizzati dal programma devono essere prima scompattati, questo package da questa possibilità. Un ' altro semplice motivo per usare questo package è che la compressione di dati in informatica è un problema abbastanza complesso, vengono usati degli algoritmi che si basano sull'algebra dei gruppi, materia non molto conosciuta al di fuori delle università scientifiche, sono quindi degli algoritmi abbastanza incomprensibili a chi non è "del mestiere", questo package da la possibilità a chiunque di comprimere e decomprimere questi dati.
Vediamo quindi cosa contiene il package java.util.zip

Interfacce

Checksum, è un interfaccia che rappresenta il codice checksum dei dati.

Classi

CheckedInputStream, è uno stream di ingresso che tratta anche il checksum dei dati in ingresso.
CheckedOutputStream, è uno stream di uscita che tratta anche il checksum dei dati in uscita.
CRC32, classe usata per calcolare il codice checksum di tipo CRC-32 di uno stream di dati.
Deflater, classe che si occupa della compressionedei dati usando la compressione secondo la libreria ZLIB.
DeflaterOutputStream, stream di uscita che comprime i dati usando l'algoritmo Deflate.
GZIPInputStream, filtro per lo stream di ingresso per leggere dati compressi secondo il formato GZIP.
GZIPOutputStream, filtro per lo stream di uscita per scrivere dati zippati usando GZIP.
Inflater, supporto per la compressione di tipo ZLIB.
InflaterInputStream, filtro per stream di ingresso, per decomprimere dati compressi secondo l'algoritmo Deflate.
ZipEntry, usata per rappresentare un file di ingresso di tipo ZIP.
ZipFile, usata per leggere il contenuto di un file ZIP.
ZipInputStream, usata per leggere i file contenuti in un archivio ZIP.
ZipOutputStream, usata per scrivere dati compressi in formato ZIP.

Eccezioni

DataFormatException, errore di formato dei dati.
ZipException

Ognuna di queste classi avrà i suoi metodi, per conoscerli tutti vi rimando alla documentazione del JDK, noi ne vediamo alcuni in un piccolo esempio.
Il seguente programma apre la directory in cui si trova e cerca tutti i file .zip, per ogni file trovato ne va vedere il contenuto.

```
import java.util.*;
import java.util.zip.*;
import java.io.*;


public class ReadZip
{

public static void main(String [] a)
{
File dir=new File(".");

System.out.println("Apro la directory "+dir.getAbsolutePath());

File[] cont=dir.listFiles();

int MAX=cont.length;

for (int i = 0; i
{
String tmp=cont[i].getName();

if ((tmp.endsWith(".zip"))||(tmp.endsWith(".ZIP")))
{
// è un file .zip

System.out.println("Ho trovato "+tmp);

controllaZip(cont[i]);

};
}


}

public static void controllaZip(File f)
{

System.out.println("Contenuto del file "+f.getName());

ZipFile Zf;
try {Zf=new ZipFile(f);}
catch (ZipException e){Zf=null;}
catch (IOException e1){Zf=null;}
;

Enumeration files=Zf.entries();

while(files.hasMoreElements())
System.out.println(files.nextElement());


}


}
```

Possiamo anche decomprimere questi files,il seguente programma prende tutti i .zip della directory dove viene eseguito e li decomprime.

```
import java.util.*;
import java.util.zip.*;
import java.io.*;
```

```
public class Decomp
{

public static void main(String [] a)
{
File dir=new File(".");

System.out.println("Apro la directory "+dir.getAbsolutePath());

File[] cont=dir.listFiles();

int MAX=cont.length;

for (int i = 0; i<MAX; i++)
{
String tmp=cont[i].getName();

if ((tmp.endsWith(".zip"))||(tmp.endsWith(".ZIP")))
{
// è un file .zip

System.out.println("Ho trovato "+tmp);

try {controllaZip(cont[i]);}
catch (IOException e){};

};
}


}

public static void controllaZip(File f) throws IOException
{

System.out.println("Decompressione del file "+f.getName()+":");

ZipFile Zf;
try {Zf=new ZipFile(f);}
catch (ZipException e){Zf=null;}
catch (IOException e1){Zf=null;}
;

Enumeration files=Zf.entries();

while(files.hasMoreElements())
{
ZipEntry tmpFile=(ZipEntry ) files.nextElement();

System.out.println("decomprimo il file "+tmpFile.getName());

System.out.println("dimensione compresso "+
tmpFile.getCompressedSize()+" dimensione non compresso "+
tmpFile.getSize()+" CRC "+tmpFile.getCrc());

System.out.println("modificato "+tmpFile.getTime());

InputStream in= Zf.getInputStream(tmpFile);

FileOutputStream out= new FileOutputStream(tmpFile.getName());


for (int ch=in.read();ch!=-1;ch=in.read()) out.write(ch);
```

```
out.close();

in.close();

}

}


}
```

Il package java.util.jar mette a disposizione del programmatore delle classi e interfacce per trattare i file di tipo Java Archive (JAR), in particolare è possibile leggerli, e scriverli. I file JAR sono basati sullo standard ZIP, con un file opzionale detto manifest .
Il contenuto del package è il seguente:

Classi
Attributes
Attributes.Name
JarEntry
JarFile
JarInputStream
JarOutputStream
Manifest

Eccezioni
JarException
Rimando alla documentazione del JDK per ulteriori informazioni.

LEZIONE 14: Il package java.net

Il Package java.net

Java, come già detto in precedenza è nato come linguaggio per la rete, solo successivamente è divenuto un vero e proprio linguaggio di programmazione.
Il suo ruolo di leader per la programmazione di rete è comunque indiscusso, come tale esso mette a disposizione del programmatore tutto vari package per effettuare tale programmazione.
Visto che lo scopo finale del corso è quello di programmare Applet, una infarinatura del package la dobbiamo avere. Il package è molto ampio, il suo contenuto è il seguente:

Interfacce
ContentHandlerFactory
FileNameMap
SocketImplFactory
SocketOptions
URLStreamHandlerFactory

Classi
Authenticator
ContentHandler
DatagramPacket
DatagramSocket
DatagramSocketImpl
HttpURLConnection
InetAddress
JarURLConnection
MulticastSocket
NetPermission
PasswordAuthentication
ServerSocket
Socket
SocketImpl

SocketPermission
URL
URLClassLoader
URLConnection
URLDecoder
URLEncoder
URLStreamHandler

Eccezioni
BindException
ConnectException
MalformedURLException
NoRouteToHostException
ProtocolException
SocketException
UnknownHostException
UnknownServiceException

Di queste noi ne vedremo solo alcune, iniziamo intanto a vedere cosa succede nella rete.
I computer in Internet comunicano scambiandosi pacchetti di dati, chiamati pacchetti IP (Internet Protocol), questi pacchetti partono da un computer, attraversano vari nodi della rete (Server di rete) e arrivano a destinazione, per stabilire il percorso intermedio tra i due computer che vogliono comunicare si esegue un algoritmo di routing (ne esistono di vari tipi, a seconda delle esigenze e a seconda del tipo di rete).
Per stabilire il mittente di una comunicazione, i destinatari, i nodi intermedi occorre che ogni computer collegato in rete abbia un nome che lo identifica univocamente, questo nome, è un numero, e si chiama indirizzo IP.
L'indirizzo IP è un numero di 32 bit, che può essere scritto in vari formati, ma il più usato è il formato decimale separato da punti, un indirizzo IP è ad esempio 594. 24.114.462 (Il numero è un numero a caso).
Siccome i computer ricordano bene i numeri, ma noi umani no, sono stati inventati i DNS (Domain Name System) che associano dei nomi a questi indirizzi IP.
La prima classe che vediamo del package è la classe InetAddres, la quale gestisce questi indirizzi IP.
La classe ha vai metodi, tra i quali uno che restituisce gli indirizzi Ip del computer sul quale si lavora, uno che dato un nome di dominio ne da l'indirizzo IP.
Il seguente esempio ne mostra l'uso.

```
import java.net.*;
public class IndirizziIP
{
public static void main(String [] a)
{

String dom="developer.java.sun.com";

try {
InetAddress loc=InetAddress.getByName(dom);
System.out.println("IP di "+dom+" : "+loc.getHostAddress());
}
catch (UnknownHostException e)
{System.out.println("Non esiste il dominio "+dom);};


try {
InetAddress loc=InetAddress.getLocalHost();
System.out.println("IP locale: "+loc.getHostAddress());
System.out.println("Nome locale"+loc.getHostName());
}
catch (UnknownHostException e)
{};

}

}
```

Come esercizio provare a far prendere come nome di dominio il primo argomento del programma, ad esempio java IndirizziIP html.it

Il package fornisce utili classi per trattare i socket fondamentali basati su TCP e su UDP, i quali sono i protocolli impiegati nella maggior parte delle applicazioni Internet. Noi queste non le vedremo, ma passiamo direttamente a classi che gestiscono applicazioni Web di livello più alto.
Vediamo la classe URL.
Che cos'è un URL? Un URL, o Uniform Resource Locator, è un puntatore ad una risorsa web.
Un arisorsa web può essere qualsiasi cosa, una directory, un file, un oggetto in rete come ad esempio una interfaccia per fare delle query ad un database remoto, o per un motore di ricerca.
Gli URL sono svariati, ognuno con un protocollo diverso, ma i più usati sono quelli che usano protocolli HTTP (HyperText Transfer Protocol) e FTP (File Transfer Protocol).
La classe URL incapsula degli oggetti web, accedendovi tramite il loro indirzzo URL.
A voi che siete utenti di Html.it non sto a stressarvi con chiacchiere sul formato degli indirizzi URL del tipo http://www.cli.di.unipi.it/~castellu/index.html e che l'index.html è facoltativo, ovvero l'indirizzo url http://www.cli.di.unipi.it/~castellu è equivalente al precedente, che ~castellu è una directory che si trova sul server, e che l'indirizzo del server è identificato da http://www.cli.di.unipi.it . Però vi dico che a differenza dei web browser, l'http:// davanti all'indirizzo è indispensabile, perché individua il protocollo dell'URL, protocollo che il Navigator e l'Explorer intuiscono anche se omesso. Quando programmeremo gli applet useremo solo gli URL per accedere anche ai file locali, vedendoli come risorse di rete. Questo lo faremo perché negli applet è vietato usare i file, per motivi di sicurezza di rete, quindi per leggere un file bisogna vederlo come una risorsa di rete.
Le ultime versioni di Java stanno eliminando questa limitazione del linguaggio, usando delle firme, che rendono la lettura e la scrittura di file controllata anche sulla rete, quindi in futuro sarà possibile usare anche i file negli applet, a patto che ci si assuma qualche responsabilità.
Vediamo alcuni costruttori di oggetti URL:

URL(String spec) , crea ul oggetto URL in base alla rappresentazione a stringa, ad esempio "html.it"
URL(String protocol, String host, int port, String file), crea un oggetto URL, specificando tutto, anche la porta.
URL(String protocol, String host, int port, String file, URLStreamHandler handler), come il precedente, solo che specifica anche l'URLStreamHandler, il quale è la superclasse comune per tutti i protocolli (HTTP,FTP,Gopher).
URL(String protocol, String host, String file), crea un oggetto URL specificando protocollo, host e File sul server host, ad esempio: URL("http","www.cli.di.unipi.it","~castellu/index.html").

Discutiamo ora alcuni metodi della classe

boolean equals(Object obj), confronta due oggetti URL.
Object getContent(), da il contenuto dell'oggetto URL.
String getFile(), da il filename dell'URL.
String getHost(), l'host
int getPort(), il numero della porta
String getProtocol(), il nome del protocollo.
String getRef(), da il puntatore all'URL.
int hashCode(), da il codica hash dell'oggetto.
URLConnection openConnection(), apre una connessione con l'oggetto remoto indicato dall'URL.
InputStream openStream(), apre una connessione con l'oggetto web indicato dall'url sotto forma di stream in lettura.
String toExternalForm(), da una stringa rappresentante l'URL.
String toString(), da una rappresentazione dell'oggetto URL.

Ecco di seguito un piccolo esempio sull'uso della classe URL.

```
import java.net.*;
import java.io.*;

public class getPage
{
public static void main(String[] arg)
{
String un;

try {un=arg[0];}
catch (ArrayIndexOutOfBoundsException e)
{
un="http://www.html.it/index.asp";
System.out.println("Nessun URL definito, prendo "+un);
```

```
};

System.out.println("URL:"+un);

URL url;

boolean pippo=false;

try {url= new URL(un);}
catch (MalformedURLException e)
{
System.out.println("URL errato, prendo http://www.html.it/index.asp ");

url = null;

pippo=true;

};

if (pippo) try {url = new URL ("http://www.html.it/index.asp ");}
catch (MalformedURLException e){};

BufferedReader stream;
try {stream = new BufferedReader (new InputStreamReader (url.openStream()));}
catch (IOException e){
System.out.println("Errore di apertura del file");
stream=null;
System.exit(0);
};

File out=new File(".\\"+url.getFile());

FileWriter Output;

try {Output=new FileWriter(out);}
catch (IOException e) {Output=null;};

String l;
try
{

while ((l=stream.readLine())!=null)
{


Output.write(l);

};

Output.flush();

Output.close();

}
catch (IOException e){System.out.println("Errore di lettura.");};
}

}
```

Il programma prende un url dai parametri d'ingresso, se non è valido o non ci sono parametri apre per default l'url http://html.it/index.asp , preleva la pagina letta e la salva su disco.
Come vedete questo è una specie di web browser, basterebbe un po di grafica e si potrebbe anche visualizzare la pagina. Provate ad eseguire il programma mettendo i seguenti indirizzi:

java getPage http://www.cli.di.unipi.it/~castellu/index.htm
java getPage http://www.cli.di.unipi.it/~castellu/pietro1.htm
java getPage http://www.cli.di.unipi.it/~castellu/chisono.htm

e semplicemente java getPage, e vedete i risultati.

LEZIONE 15: conclusioni sui package

In questo capitolo abbiamo visto alcuni package contenenti le API del linguaggio Java, ma ce ne sono degli altri:

java.applet , che vedremo nel prossimo capitolo, il quale serve per creare dei programmi che girano sui web browsers, chiamati applet.
java.awt , questo package e i suoi sottopackages implementano le classi per implementare i controlli GUI, per implementare interfacce grafiche, oltre agli strumenti per il disegno, manipolazione di immagini, stampa e altre funzionalità. Inizieremo a vederlo nel prossimo capitolo
java.beans, package che permette di definire componenti Java e di utilizzarli in altri programmi.
java.rmi, package per l'invocazione di metodi remoti, ovvero di metodi di oggetti che si trovano ovunque sulla rete, per costruire delle applicazioni distribuite.
java.security, ci sono classi che implementano le funzioni di sicurezza, come ad esempio classi utilizzate per crttografare documenti prima di mandarli in rete.
java.sql, interfaccia tra il linguaggio Java e il linguaggio per basi di dati SQL.
java.text, classi molto utili per l'interazionalizzazione.
javax.accessibility, classi che supportano tecnologie a sostegno degli utenti disabili.
javax.swing, è un estensione di java.awt, per costruire applets e applicazioni grafiche è un portento.
org.omg.CORBA, permette di interfacciare il linguaggio Java con il linguaggio CORBA.

Ancora una volta vi invito per saperne di più a controllare la documantazione del JDK, disponibile On line, sia per essere scaricata che per essere consultata presso il sito della Sun Microsystem www.sun.com .

Questi sono i package standard del linguaggio Java, a questi si aggiungono le estensioni standard del linguaggio. Le estensioni standard sono dei package che nelle prossime versioni di java diventeranno package standard, e che per ora sono versioni beta. Swing è stata una estensione fino all'uscita di Java2, ora di fatto è più utilizzata delle vecchie awt.

***API Servlet**, è destinata alla programmazione di applicazioni server in Java. Questa API è costituita dai packages javax.servlet e javax.servlet.http.*

***Java 3D**, gestisce il disegno tridimensionale, è come la versione Java di OpenGL (JavaGL), la famosissima libreria della SGI (alcuni la conoscono come Glide, ovvero la libreria OpenGL per le 3Dfx) e DirectX della Microsoft.*
Si può scaricare al sito: http://java.sun.com/products/java-media/3D/index.html

***Java Media Framework**, gestisce all'interno dei programmi Java vari formati audio, video e multimediali, i file supportati sono i seguenti:*
.mov, .avi, .viv, .au, .aiff, .wav, .midi, .rmf, .gsm, .mpg, .mp2, .rtp.
Se non tutta, almenu una parte diverrà standard con Java 1.3, la quale è in uscita (fine Aprile). Si scarica al sito:
http://www.javasoft.com/products/java-media/jfm/index.html

***Speech**, funzioni di riconoscimento vocale, non solo per i comandi, ma è possibile anche editare interi file. Questo package fa anche l'output vocale.*
Si può scaricare al sito: http://java.sun.com/products/java-media/speech/index.html

***Telephony**, funzioni di telefonia, fax.*

***JavaMail**, classi per gestire la posta elettronica.*

***Java Naming and Directory Services**, per accedere ai servizi di nomi e directory usando protocolli come LDAP.*
Questo packages è diventato standard in JDK 1.3

***Java Management**, per la gestione di reti locali.*

***JavaSpaces**, ulteriori classi per la creazione di applicazioni distribuite.*

***JavaCommerce**, per il commercio elettronico.*

Personalmente non vedo l'ora che diventino standard le API Java 3D, Java Media Framework, JavaSpeech e Java Telephony, parchè le funzionalità promesse da queste API sono veramente eccezionali. Usarle adesso è possibile, ma a proprio rischio e pericolo, infatti esse sono ancora delle versioni beta, e quindi piene di errori, inoltre se si vogliono scrivere degli applet usando queste nuove funzionalità è possibile, ma per mandarle in esecuzione occorre l'appletviewer del JDK, perché sicuramente il Java implementato nei web browser ancora non le supporta. Si pensi che Swing è divenuto un package standard del linguaggio, ma ancora esistono dei browser che non lo supportano.

LEZIONE 16: Interfacce grafiche ed eventi

Eccoci finalmente arrivati alla programmazione di interfacce grafiche, quindi alla creazione di applet e di applicazioni a finestre. Cerchiamo innanzittutto di stabilire cosa è una interfaccia, e cosa significa farla grafica.
Innanzitutto ogni programma, come già detto, può essere visto come un oggetto che calcola una funzione, quindi che prende dei dati dall'esterno e restituisce dei risultati. Si pensi ad esempio ad un semplice programma che fa la somma tra due numeri. Il programma in questione prenderà in input due numeri e restituirà come output un un numero, che rappresenta la somma dei primi due.
Quanto detto è valido in generale per tutte le applicazioni, e non è un caso particolare dell'esempio precedente. Si pensi ad un videogioco, l'input sarà dato dal joystick, e l'output sarà la grafica sullo schermo, quindi concettualmente sia il videogioco, che il programma somma, che qualsiasi altro programma un programmatore possa inventarsi, sono delle funzioni calcolate su dei dati in ingresso che restituiscono dei risultati.
L'interfaccia del programma verso l'utente che lo usa è il modo in cui il programma prende i dati dall'utente e gli restituisce i risultati.

Fino ad ora abbiamo visto delle interfacce testuali, ovvero nel caso della somma dei due numeri, i dati venivano presi sallo standard input, con una System.in.read() e i risultati venivano stampati sullo standard output con una System.out.print().
Le System.in e out rappresentano una interfaccia del programma verso l'esterno, in questo caso verso l'utente.
Altre interfacce con l'esterno che abbiamo già esaminato sono i files, essi rappresentano delle interfacce di ingresso o di uscita verso utenti o altri programmi.
Facendo un piccolo paragone, neanche tanto impossibile, tra un programma e l'uomo, possiamo dire che l'interfaccia in ingresso della mente è rappresentata dalla vista, il tatto, il gusto, l'udito e l'olfatto, mentre l'interfaccia in uscita è rappresentata dalla parola e dal movimento dei muscoli.
Da notare che questa non è una cosa stranissima, infatti i robot, anche se non hanno gli stessi canali di ingresso degli uomini (hanno vista, una specie di tatto, qualcosa che segnala la prossimità) e di uscita (spesso non parlano, al posto dei muscoli hanno motori elettrici collegati a bracci meccanici, pinze, ecc...), hanno lo stesso funzionamento dell'uomo, ovvero compiono delle azioni (danno un output) in conseguenza a degli stimoli (segnali di input), anche se l'uomo è molto più complesso grazie alla capacità di pensiero, che ancora l'intelligenza artificiale non riesce a replicare in pieno (infatti un robot non è capace di prendere una decisione senza conoscere solo delle informazioni parziali di un problema, come è in grado di fare l'uomo, in base a supposizioni, a volte coscienza, ecc…).

Quindi l'interfaccia è composta da tutti quei canali da cui un programma prende informazioni e verso le quali sputa fuori dei risultati.
Alcuni dati di ingresso servono a fare cambiare solo stato al programma, altri invece inducono immediatamente un output. Noi li cosideriamo tutti dati su cui vengono calcolate funzioni, infatti una funzione viene calcolata anche in base allo stato di un programma, e quindi anche lo stato, e di conseguenza l'input che lo ha causato, è un input della funzione.
Chiarisco questo discorso con un esempio. Pensiamo ad un programma che calcola due funzioni, una di somma e una di sottrazione tra due numeri, e pensiamo ad un input che prende un numero per scegliere uno stato (1 o 2), e poi due numeri. A questi due numeri viene applicata la funzione somma o la funzione sottrazione a seconda dello stato del programma.

Ad esempio:

INPUT: <1,10,10> à OUTPUT: 20
INPUT:<2,10,10> à OUTPUT: 0

Vedete nell'esempio, come anche l'input sullo stato viene in definitiva usato per calcolare l'output.

Quello che interessa a noi è l'interfaccia utente, abbiamo visto infatti come i programmi possono interfacciarsi con altri programmi o con periferiche (inviando ad esempio comandi ad una stampante), a noi questo non interessa, ma ci interessa vedere tutto quello che serve al programma per dialogare con l'utente, ovvero la cosiddetta interfaccia utente.
In particolare ci interessa vedere l'interfaccia grafica del programma, ovvero quell'interfaccia che rende molto gradevole il programma all'utente, al posto della grigia interfaccia a caratteri che abbiamo visto.
L'interfaccia grafica del programma è composta dai cosiddetti componenti GUI (Graphics User Interface), essi sono dei componenti che servono per l'input o per l'output, ed hanno un aspetto grafico. Sono ad esempio dei componenti GUI di un programma tutti i menù, i bottoni, le label, eccetera…di questo.
Tutte queste GUI saranno ovviamente collegate ad una struttura che le contiene, le due strutture che vedremo, e che sono le più importanti, sono gli applet e le finestre.

Ogni GUI è quindi un pezzettino di interfaccia grafica, essi ricevono degli eventi e in base a questi danno dei risultati. Si pensi ad un bottone, esso può essere cliccato e rilasciato, quando un utente programma una interfaccia grafica che contiene un bottone dovrà gestire anche gli eventi associati a questo, ovvero dovrà dire cosa succede quando il bottone viene cliccato, quando viene rilasciato, quando ci si passa con il mouse sopra, eccetera.
Ogni differente GUI avrà un tipo di evento associato, alcuni sono automaticamente gestiti dalla classe che viene estesa per inserire il GUI nella finestra (ad esempio la modifica grafica del bottone cliccato che diviene evidenziato), altri possono essere definiti dall'utente (come il click di un bottone, se non c'è il gestore non succede niente) e altri ancora devono essere definiti dall'utente (come tutti gli eventi della tastiera quando si vuole ascoltare almeno uno di questi, ad esempio ci interessa il rilascio di un tasto, dobbiamo gestire anche la pressione, ecc…).
Ad ogni tipo di evento, per ogni tipo di componente GUI, deve essere dichiarato un ascoltatore dell'evento, esso è un programma che attende il verificarsi dell'evento sul componente, e quando questo si verifica lo gestisce.
La programmazione di interfacce grafiche è quindi diversa dalla usuale programmazione, infatti qui si disegnano le interfacce, e poi si gestiscono gli eventi che arrivano, mentre nelle usuali applicazioni c'era un main, che rappresentava l'intero programma. La gestione degli eventi in Java è cambiata dalla versione 1, a Java 2 (JDK 1.2 in sù), noi vedremo la nuova gestione degli eventi, quella di Java 2, perché la prima è stata cambiata in quanto a volte capitava che non si capiva un evento a quale componente era associato.

Nella nostra visita ai componenti GUI vedremo quindi come definirli, inizializzarli, inserirli in una finestra o in un applet, e quindi come gestirne gli eventi.
Come già detto Java 2, ha due collezioni di packages per le interfacce grafiche, java.awt, già presente in Java 1, e javax.swing, uscita con Java 2, costruita sulle AWT, che amplia alla grande le possibilità per le interfacce grafiche.

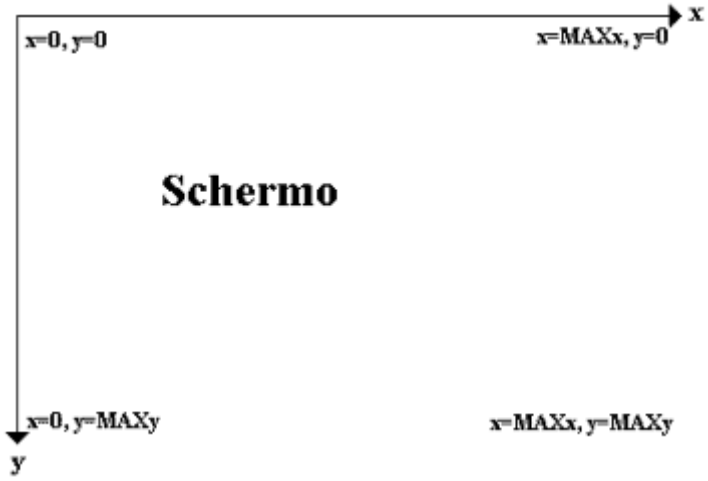
LEZIONE 17: Cosa è una applicazione a Finestre

L'applicazione a finestre è l'applicazione che più spesso usiamo quando lavoriamo al computer, essa è una applicazione che viene eseguita in locale, che come interfaccia utente utilizza le tecnologia delle finestre, tipica dei sistemi operativi Mac OS (su cui è nata), Windows (tutte le sue varianti: Windows 3.1, Windows 95, 95-OSR2, 98, NT, 2000 e il futuro Millenium), XWindows (il server grafico per Linux e Unix).
Quasi tutti i programmi che vengono usati attualmente sono applicazioni a finestre ad esempio lo sono: Word, Netscape, Explorer, Jbuilder, Visual Studio, WinZip, Paint, XV, ecc… (tutti prodotti appartenenti alle relative case produttrici).
Vediamo quindi come si crea una Finestra usando il package AWT.
Al solito creiamo una applicazione con il solito main, come abbiamo già fatto in precedenza. La classe creata però, questa volta, estenderà la classe java.awt.Frame, la quale rappresenta la finestra completa, comprendente il titolo, i bottoncini di riduci a icona, massimizza e chiudi.
La classe Frame ha vari metodi e costruttori, che verdremo tra poco, per ora creiamo la nostra prima finestra con titolo Prima Finestra e che non contiene niente, la editiamo in Finestra.java.

```
import java.awt.*;

public class Finestra extends Frame
{

public Finestra()
{
```

```
super("Prima Finestra");

setLocation(100,100);

setSize(200,100);

show();

}


public static void main(String[] arg)
{
new Finestra();

System.out.println("Ho creato la finestra");
}

}
```

All'inizio, come per ogni applicazione viene invocato il main, il quale crea un nuovo oggetto di tipo Finestra (il main e la finestra potevano stare in due file separati, uno che gestiva il main, e uno che gestiva la finestra). Nel costruttore dell'oggetto finestra, viene invocato il costruttore della superclasse, ovvero di Frame, con la stringa "Prima Finestra" (è il titolo), viene invocato il metodo setLocation che dichiara la posizione dell'angolo destro in alto della finestra sul desktop, in questo caso <100, 100> (Sono la x e la y rispettivamente, la x si misura dal lato sinistro dello schermo e cresce andando a destra, la y si misura dal lato in alto dello schermo, e cresce andando verso il basso).
Poi si chiama il metodo setSize, che permette di specificare larghezza e altezza della finestra, la nostra è larga 200 e alta 100, e alla fine viene invocato il metodo show(), che fa apparire la finestra sullo schermo.



Le coordinate dello schermo non sono coerenti con le usuali coordinate cartesiane, in effetti la y è esattamente l'opposta, cresce verso il basso e diminuisce verso l'alto. Questo è un problema che non solo Java ha, ma tutti i linguaggi di programmazione e tutte le librerie per la grafica Raster, è dovuto a motivi credo storici, legati a come vengono disegnati i pixel sullo schermo a livello hardware. Bisogna abituarcisi, le prime volte però mi rendo conto che qualche problema si incontra nel ragionare in questi termini.
Provate a compilare e ad eseguire il programma, e vedrete la vostra prima finestra prendere vita, provate poi ad eliminare a turno i metodi setLocation, setSize e show, provate anche a mettere la finestra in posizioni diverse e valutate i cambiamenti. Nell'esempio si vedono degli esempi di eventi gestiti automaticamente dal sistema, questi sono Ir Resize della finestra, e la pressione dei bottoni riduci ad icona e massimizza (minimizza), i quali sono gestiti nella classe Frame.
Non è gestito invece l'evento chiudi della finestra (il bottone con la x), se esso viene premuto non succede niente, per terminare l'esecuzione del programma bisogna andare sun prompt del dos dal quale è stata lanciata l'applicazione e premere CTRL+C (l'exit per tutti i programmi DOS).
Infine, credo sia chiaro, l'applicazione non funziona in ambienti non grafici, ovvero in DOS e in Linux, occorre Windows (e il dos caricato in una finestra, ovvero prompt di ms-dos) o Xwindows con una shell aperta.

Da notare che non è vero che ad una applicazione è associata una sola finestra, Java è un linguaggio che

supporta la multiprogrammazione, ogni finestra è un programma a se stante che gira in parallelo agli altri, quindi posso creare per la stessa applicazione più finestre Frame, come nell'esempio che segue.

```
import java.awt.*;

public class Finestre extends Frame
{

public Finestre(String Nome, int x, int y)
{

super(Nome);

setLocation(x,y);

setSize(200,100);

show();

}


public static void main(String[] arg)
{

System.out.println("Creo 4 finestre sovrapposte");

for (int i=1;i<5;i++) new Finestre("Finestra "+i,10+(i*10), 10+(i*10));

System.out.println("Creo 4 finestre a scacchiera");

for (int i=5;i<9;i++) new Finestre("Finestra "+i,(i-5)*200, 100+(i-5));


System.out.println("Ho creato le otto finestre");
}

}
```

Le stesse cose si potevano fare estendendo la classe JFrame del package javax.swing.

```
import javax.swing.*;

public class FinestraSwing extends JFrame
{

public FinestraSwing()
{

super("Prima Finestra");

setLocation(100,100);

setSize(200,100);

show();

}


public static void main(String[] arg)
{
new FinestraSwing();
```

System.out.println("Ho creato la finestra");

}

}

Questo programma è come Finestra.java. solo che estende JFrame di swing, le uniche differenze sono il contenuto della finestra, questa volta grigio, prima bianco e il bottone chiudi, che questa volta chiude la finestra (solo la finestra però, non l'applicazione intera) In effetti l'utilizzo di swing e di awt è molto simile, solo che swing è più completo, esso mette a disposizione molte classi in più, da la possibilità di cambiare il look delle finestre a runtime, e altro.
Purtroppo non tutti i web browser le supportano, noi quindi vedremo le awt, e poi discuteremo delle swing, fermo restando che chi vuole fare un applet per la propria pagina html DEVE per ora usare le awt, tra non molto, quando l'XML diverrà uno standard di fatto, e quindi si dovranno cambiare i browser, si potrà utilizzare tranquillamente swing anche per gli applet.
Vediamo quindi cosa contiene la classe Frame, innanzitutto i costruttori sono due:

Frame() , che crea un Frame senza nessun titolo, inizialmente invisibile.
Frame(String T), che crea un Frame con titolo T, anch'esso inizialmente invisibile.

Gli attributi della classe sono:
static int ICONIFIED
static int NORMAL

per indicare lo stato della finestra, e

static int CROSSHAIR_CURSOR
static int DEFAULT_CURSOR
static int E_RESIZE_CURSOR
static int HAND_CURSOR
static int MOVE_CURSOR
static int N_RESIZE_CURSOR
static int NE_RESIZE_CURSOR
static int NW_RESIZE_CURSOR
static int S_RESIZE_CURSOR
static int SE_RESIZE_CURSOR
static int SW_RESIZE_CURSOR
static int TEXT_CURSOR
static int W_RESIZE_CURSOR
static int WAIT_CURSOR

tutti dichiarati deprecated, per i cursori, rimpiazzati dalla classe Cursor. Eredita gli allineamenti dei componenti da Component.

I metodi sono:

void addNotify(), collega il frame ad una risorsa schermo, e lo rende visualizzabile.
int getCursorType(), dichiarato Deprecated. E' della versione 1.1 delle JDK
static Frame[] getFrames(), da un array contenente tutti i Frames creati dall'applicazione.
Image getIconImage(), da l'icona del Frame, essa è un oggetto di tipo Icon, che vedremo in seguito.
MenuBar getMenuBar(), restituisce la barra dei menu del frame, è un oggetto di tipo MenuBar, che vedremo tra non molto.
int getState(), da lo stato del frame.
String getTitle(), da il titolo del frame
boolean isResizable(), da true se il frame può essere allargato e ristretto con il mouse.
protected String paramString() , da la stringa contenente i parametri del Frame.
void remove(MenuComponent m), toglie la MenuBar specificata dal frame.
void removeNotify(), toglie la connessione tra il frame e la risorsa che rappresenta lo schermo, rendendolo invisualizzabile.
void setCursor(int cursorType), settava il cursore in JDK1.1, è dichiarato deprecated.
void setIconImage(Image image), setta l'icona del Frame.
void setMenuBar(MenuBar mb), assegna una MenuBar al frame.
void setResizable(boolean resizable), setta il Frame ridimensionabile oppure no, per default lo è.
void setState(int state), setta lo stato del frame

void setTitle(String title), setta il titolo del frame.

Frame è una classe che estende java.awt.Window, e quindi ne eredita metodi e attributi.
In effetti Frame è una Window con in più un bordo e una MenuBar. I metodi ereditati da window sono:

addWindowListener, applyResourceBundle, applyResourceBundle, dispose, getFocusOwner, getInputContext, getLocale, getOwnedWindows, getOwner, getToolkit, getWarningString, isShowing, pack, postEvent, processEvent, processWindowEvent, removeWindowListener, show, toBack, toFront.

Window estende java.awt.Container, esso è un contenitore di oggetti AWT, è un componente che può contenere altri componenti., quindi per transitività Frame ne eredita metodi e attributi, i metodi sono:

add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, remove, removeAll, removeContainerListener, setFont, setLayout, update, validate, validateTree

Container estende java.awt.Component, un Component è un oggetto che ha una rappresentazione grafica, ad esempio un bottone sarà una estensione di questa classe, la quale estende a sua volta java.lang.Object
Gli attributi ereditati da Component sono:

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

mentre i metodi ereditati da Component sono:

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

Mentre da Object vengono ereditati i soliti metodi:

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Alcuni di questi li vedremo, per gli altri vi consiglio di consultare la Documentazione del JDK, che li descrive tutti in dettaglio.

Il diagramma delle estensioni di Frame è il seguente:

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
```

A questo punto si capisce come è importante l'estensione delle classi in Java, infatti in Frame possono essere invocati tutti i metodi di sopra.

Mentre quello di JFrame di swing è:

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
                          |
                          +--javax.swing.JFrame
```

Quindi Jframe eredita tutti i metodi di Frame essendo derivato da essa, in più ne definisce altri.

LEZIONE 18: Cosa è un applet

Un' applet non è altro che una applicazione Java che gira su web. L'applet presenta qualche differenza con le applicazioni, infatti essi non hanno nessun main, sono delle classi, chiamate come il file che le contiene, che estendono la classe Applet del package java.applet.
Anche per gli applet esiste la versione JApplet di swing, che viene usata per inserire componenti Swing invece che componenti AWT. Un applet ha bisogno di un file html che la richiama, ad esempio sia PrimoApplet.java l'applet che vogliamo eseguire, lo compiliamo e il compilatore ci genera PrimoApplet.class, per eseguirlo abbiamo bisogno di un file html che al suo interno contenga il TAG:

<applet code="PrimoApplet.class" ></applet>

Supponiamo che questo file si chiami pippo.html, a questo punto abbiamo due modi di eseguire l'applet, il primo in fase di debug è usando il programma appletviewer di JDK, e scriveremo in questo caso dal prompt del dos:

appletviewer pippo.html

il secondo, è usando un web browser, come Explorer o Netscape richiamando il file pippo.html.
Come si vede, si lancia sempre il file html, e non come accadeva per le applicazioni il file .class, è il file html che richiama l'applicazione .java.
Per maggiori dettagli sulle pagine html vi consiglio di consultare sul sito HTML.IT le sezioni riguardanti l'argomento, noi faremo delle pagine scarne che serviranno solo a caricare i nostri applet, ad esempio per l'applet PrimoApplet.class di prima, il file pippo.html sarà qualcosa del tipo:

<html>
<head>
<title>Applet PrimoApplet
</head>
<body>

Il seguente è l'applet PrimoApplet.class

<applet code="PrimoApplet.class" width=100 height=100>Il tuo browser è vecchio, cambialo!</APPLET>
</body>
</html>

Abbiamo visto come mandare in esecuzione l'applet, adesso creiamolo. Innanzitutto dobbiamo creare una classe chiamata PrimoApplet, che estende la classe java.applet.Applet, e dobbiamo definire alcuni metodi che il sistema (appletviewer o il browser) invocherà automaticamente. Uno di questi metodi si chiama paint (Graphics O), e Graphics è un oggetto che rappresenta lo schermo dell'applet, noi lo ridefiniremo in modo da fare uscire sullo schermo quello che preferiamo.
Useremo il metodo drawString della classe Graphics per stampare una stringa su schermo.
Il programma PrimoApplet.java è il seguente

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

public class PrimoApplet extends Applet
{

public void paint (Graphics g)
{
g.drawString("Ciao, io sono il primo applet.",0,50);
}

}
```

Il package applet contiene tre interfacce ed una classe:
Interfacce

AppletContext, questa interfaccia corrisponde all'ambiente dell'applet, ovvero al documento che lo contiene e agli altri applets contenuti nello stesso documento.
AppletStub, riguarda l'ambiente di esecuzione dell'applet, sia esso il browser che l'appletviewer. AudioClip, l'interfaccia è una semplice astrazione per suonare degli audio.

Classe
Applet

Analizziamo la classe Applet più in dettaglio.
Il cosruttore è unico, senza argomenti.
Applet()

Vi sono alcuni metodi chiamati dal browser o dall'appletviewer automaticamente, essi sono: void init(), questo metodo viene invocato appena l'applet viene completamente caricato nel sistema.
Esso tipicamente viene utilizzato per inizializzare l'applet.

void start(), chiamato quando il sistema manda in esecuzione l'applet, lo informa di questo avvenimento.
void stop(), chiamato quando il sistema stoppa l'esecuzione dell'applet, lo informa dell'evento, chiamato quando si preme il bottone STOP dell'appletviewer, si cambia pagina nel browser.
void destroy(), chiamato quando l'applet viene distrutto, ovvero quando si cambia esce dal browser o dall'appletviewer

Quindi il ciclo di vita di un'applet è il seguente:

1. Viene caricato, e quindi viene chiamato init();
2. Viene mandato in esecuzione, viene chiamato start(), esso invoca il metodo paint() della superclasse Container;
3. Viene stoppato premendo lo stop del browser oppure quando la finestra che lo contiene non è più in primo piano, e viene chiamato stop(), mentre quando ripassa in primo piano ne viene richimato lo start();
4. Viene infine distrutto quando si esce dal browser che lo ha mandato in esecuzione, innanzitutto viene invocato lo stop() e poi il destroy();

Nell'esempio successivo vengono visualizzate le precedenti fasi, per vedere i risultati con un browser, visto che l'output sono delle System.out.print(), selezionare in strumenti (tool), show java consolle (o simili), con l'appletviewer invece l'output viene scritto nella finestra dal quale viene invocato l'html che invoca l'applet.
Lo editiamo nel file Stadi.java:

```
import java.applet.*;
public class Stadi extends Applet
{
public Stadi()
{

System.out.println("Invocato il costruttore di Stadi");

}
public void init()
{

super.init();

System.out.println("Eseguito public void init()");

}
public void start()
{

super.start();

System.out.println("Eseguito public void start()");

}
public void stop()
{

super.stop();

System.out.println("Eseguito public void stop()");

}
public void destroy()
{

super.destroy();

System.out.println("Eseguito public void destroy()");

}
}
```

Lo compiliamo con: javac Stadi.java
Per caricarlo creeremo il file Stadi.html che conterrà:

```
<html>
<head>
<title>Stadi.html carica Stadi.class</title>
</head>
<body>
Il seguente è l'applet Stadi, che fa vedere gli stadi attraversati dall'applet.
<BR>
<applet code="Stadi.class" width=200 height=100>Il tuo browser è vecchio, cambialo!</APPLET>
</body>
</html>
```

Per eseguirlo batteremo : appleviewer Stadi.html, oppure faremo apri sul file Stadi.html con il nostro browser preferito.

Gli altri metodi della classe Applet sono:

AppletContext getAppletContext(), da l'AppletContext associato all'applet, ovvero il documento che lo ha mandato in esecuzione e gli altri applet invocati da questo.
String getAppletInfo(), da informazioni sull'applet, deve essere sovrascritta, la normale da null.

AudioClip getAudioClip(URL url), da l'oggetto di tipo AudioClip associato all'URL inserito. Ricordo che l'URL è una risorsa del web.
AudioClip getAudioClip(URL url, String name), da l'oggetto di tipo AudioClip associato all'URL e al nome.
URL getCodeBase(), da l'url associato all'applet.
URL getDocumentBase(), da l'url del documento html che ha invocato l'applet.
Image getImage(URL url), da l' oggetto di tipo Image associata all'url inserito, essa può essere stampata sullo schermo.
Image getImage(URL url, String name), da l'oggetto di tipo Image associato all'url e al nome.
Locale getLocale(), da l'oggetto di tipo Locale associato all'applet, esso si cura dell'internazionalizzazione, si trova nel package java.util.
String getParameter(String name), da il valore del parametro chiamato name preso dalla pagina html che invoca l'applet. L'applet infatti può essere invocato con dei valori di ingresso, come l'args del main, questo metodo li preleva.

Ad esempio se invoco l'applet Clock.class così:

<applet code="Clock" width=50 height=50>
<param name=Color value="blue">
</applet>

Se nel codice dell'applet scriverò getParameter("Color") il risultato che otterrò sarà "blue".
String[][] getParameterInfo(), da un array che contiene informazioni sui parametri dell'applet. *boolean isActive ()*, dice se l'applet è attivo.
static AudioClip newAudioClip(URL url), prende un AudioClip da un dato url.
void play(URL url), suona l'audio clip preso dall'url assoluto.
void play(URL url, String name), suona il Clip dato dall'url e dal nome specificato.
void resize(Dimension d) o void resize(int width, int height), richiede all'applet di modificare le proprie dimensioni. Dimension è un oggetto awt che è una dimensione, ovvero un altezza e una larghezza.
void setStub(AppletStub stub), setta l'AppletStub dell'applet con il nuovo stub.
void showStatus(String msg), Richiede all'applet che la stringa venga stampata nella finestra di stato dell'applet.

Un applet è una estensione di Panel, il quale è un semplice contenitore, da questo eredita il metodo: addNotify

Panel estende Container da cui eredita, e fa ereditare ad Applet i metodi: add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, paramString, preferredSize, print, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, removeNotify, setFont, setLayout, update, validate, validateTree

A sua volta Container estende Component, e quindi vi sono gli attributi

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

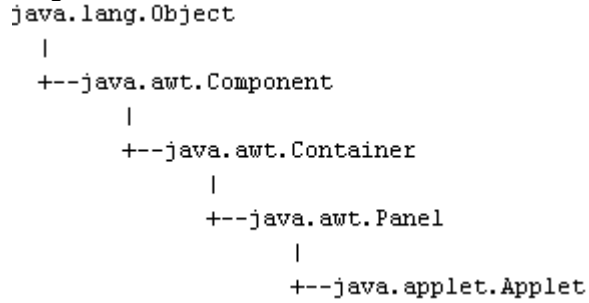
e i metodi:

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener,

removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

E quindi quelli di Object:
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

La gerarchia è:



Facciamo un altro piccolo applet chiamati Info.java

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

public class Info extends Applet implements ImageObserver
{


public Info()
{

}

public void init()
{
super.init();
setBackground(Color.yellow);
resize(400,200);
}


public void start()
{
super.start();
}

public void stop()
{
super.stop();
}


public void destroy()
{
super.destroy();
}


public void paint (Graphics g)
{
g.setColor(Color.darkGray);

String p=getAppletInfo();
```

```
if (p!=null) g.drawString(p,10,10);

g.drawString("CODE:"+getCodeBase().toString(),10,20);

g.drawString("DOC:"+getDocumentBase().toString(),10,30);

Image io=getImage(getCodeBase(),"me.JPG");

// Per visualizzare questa imagine ho bisogno che Info implementi l'interfaccia ImageObserver.
// g.drawImage(Image,x,y,ImageObserver);
g.drawImage(io,10,40,this);

g.drawString("Questo sono io",80,80);

String nome=getParameter("parametro3");

String cognome=getParameter("parametro2");

String eta=getParameter("parametro1");

g.drawString("Esegue il programma",10,150);

g.drawString(nome,10,160);

g.drawString(cognome,10,170);

g.drawString("di "+eta+" anni",10,180);

}

public String getAppletInfo()
{
return "Applet di Pietro Castellucci";
}

public String[][] getParameterInfo()
{

String[][] r={
{"parametro1","intero","Tua età"},
{"parametro2","Stringa", "Tuo Cognome"},
{"parametro3","Stringa","Tuo Nome"}
};
return r;
}

}
```

Per caricare l'applet creeremo un file chiamato Info.html, che conterrà:

```
<html>
<head>
<title>Info.html carica Info.class</title>
</head>
<body>
Il seguente è l'applet Info.
<BR>
<applet code="Info.class" width=400 height=200>
<param name=parametro1 value="ETA' DI CHI ESEGUE IL PROGRAMMA">
<param name=parametro2 value=" COGNOME DI CHI ESEGUE IL PROGRAMMA ">
<param name=parametro3 value=" NOME DI CHI ESEGUE IL PROGRAMMA ">
Il tuo browser è vecchio, cambialo!
</APPLET>
</body>
</html>
```

Mi raccomando di specificare i parametri parametro1, parametro2 e parametro3, perché nell'applet non si fa nessun controllo sulla loro definizione, quindi l'applet fa una eccezione non catturata, inoltre nella directory dove mettete l'applet, ci deve essere un file chiamato me.JPG, che è un'immagine 67x88 pixel.

LEZIONE 19: *Applicazioni miste*

A questo punto, dopo avere visto cosa sono i Frame e cosa sono gli Applet, ci viene in mente una domanda, ma sarà possibile combinare le due tecniche, se le vogliamo chamare così, per creare delle applicazioni miste?
La risposta è ovviamente si. Possiamo fare dei programmi Java che sono dei misti tra applicazione e applets. In effetti data la modularità del linguaggio possiamo creare delle appliczioni Java che usano altre applicazioni Java, cosa che non accade nei normali linguaggi di programmazione, ove è solo possibile invocare da un programma altri programmi.
Supponiamo ad esempio di avere creato un programma Java chiamato Calcolatrice.java, con il suo main e i suoi metodi, tra cui uno che prende due interi e ne restituisce la somma.
Supponiamo a questo punto di averlo compilato in Calcolatrice.class e di averlo anche eseguito.
Infine facciamo un altro programma di qualunque genere, in questo noi possiamo usare il programma Calcolatrice.class, ovvero la classe Calcolatrice, e tutte le sue funzioni pubbliche, tra cui quella della somma.

Tornando al discorso degli applet che chiamano dei frame, guardate questo esempio:

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

public class Mista extends Applet
{

Frame F;

public void init()
{

F = new Frame("Ciao, questo è un Frame");

F.setSize(100,50);

F.show();

}

public void paint(Graphics g)
{

g.drawString("Io sono un'applet",10,20);

new p();

}


public void destroy()
{

String [] param={""};

Finestra.main(param);

}


}
```

```
class p extends Frame
{

p()
{
setTitle("Un'altro Frame lanciato dall'applet");

setSize(250,100);

setLocation(200,300);

show();

new Dialog(this,"Questa è una dialog");

}

public void paint(Graphics g)
{

g.drawString("Potrei essere un banner pubblicitario",10,40);



}


}
```

Io metterete in un file chiamato Mista.java, e lo lancerede creando un file Mista.html contenente:

```
<html>
<head>
<title>mista</title>
</head>
<body>
Il seguente è un applet:<BR>
<applet code="Mista.class" width=200 height=100>Il tuo browser è vecchio, cambialo!</APPLET>
</body>
</html>
```

Lanciatelo e vedete cosa succede, ogni volta che l'applet passa in secondo piano, ovvero ci piazzate un'altra finestra sopra, e poi ritorna in promo piano.

LEZIONE 20: *Interfacce grafiche: GUI e AWT*

Prima di cominciare questo che sarà uno dei capitoli più interessanti del corso, introduciamo il package awt e cosa è un componente GUI.
Vedremo anche dei componenti che sono contenitori per altri componenti.
Il package Abstract Window Toolkit, mette a disposizione del programmatore una serie di classi e Interfacce per la gestione di interfacce grafiche.
Il package java.awt ha anche dei sottopackage, essi sono:

java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.image
java.awt.image.renderable
java.awt.print

Di cui vedremo alcune funzionalità. Sulle classi awt sono state costruite le classi del package swing, che è attualmente il package più utilizzato per costruire delle interfacce grafiche, e che vedremo tra non molto. Vediamo cosa contiene awt:

Interfacce *ActiveEvent*, interfaccia per eventi che sanno come sono stati attivati.
Adjustable, interfaccia per oggetti che hanno un valore numerico compreso in un range di valori.
Composite, interfaccia che serve per comporre le primitive grafiche di awt.
CompositeContext, ambiente ottimizzato per le operazioni di composizione di primitive.
ItemSelectable, interfaccia per oggetti che contengono un insieme di item, di cui possono essere selezionati anche zero o più di uno(Di solito si può selezionare almeno e al massimo un item, questa interfaccia elimina questa limitazione).
LayoutManager, interfaccia per le classi che conterranno dei gestori di layout.
LayoutManager2, come quello di sopra.
MenuContainer, superclasse di tutti i menu.
Paint, questa interfaccia definisce i colori da usare per le operazioni grafiche usando Graphics2D.
PaintContext, ambiente ottimizzato per le operazioni di composizione di primitive grafiche usando Graphics2D.
PrintGraphics, definisce il contesto per stampare.
Shape, definizioni per costruire una forma geometrica.
Stroke, ulteriori decorazioni per le forme.
Transparency, definisce i più comuni tipi di trasparenza.

Classi
AlphaComposite, classe che implementa le composizioni del fattore alfa dei colori trattando il blending e la trasparenza di grafica e immagini.
AWTEvent, root degli eventi di AWT.
AWTEventMulticaster, dispatcher per eventi AWT, efficiente e multiprogrammabile, effettua il multicast di eventi a vari componenti.
AWTPermission, per i permessi AWT.
BasicStroke, attributi per le linee esterne delle primitive grafiche.
BorderLayout, è il primo gestore di layout, questi gestori, che vedremo tra pochissimo, servono a disporre gli elementi (i componenti) nei contenitori. Questo divide il contenitore in 5 regioni: nord, sud, est, ovest e centro.
Button, classe dei bottoni.
Canvas, area in cui una applicazione può disegnare.
CardLayout, altro gestore di layout.
Checkbox, CheckboxGroup, CheckboxMenuItem esse gestiscono i bottoni di tipo booleano (premuto non premuto)
Choice, presenta un menu a tendina per delle scelte.
Color, colori in RGB o in altri arbitrari spazi di colori.
Component, Classe da cui derivano tutti i componenti GUI (bottoni, label,…)
ComponentOrientation, orientamento dei componenti
Container, contenitore di componenti.
Cursor, cursori.
Dialog, finestre di dialogo, sono quelle che segnalano errori e non solo.
Dimension, casse che gestisce le dimensioni dei componenti, altezza e larghezza.
Event, classe che gestisce gli eventi secondo il veccio modello di gestone degli eventi (Java 1.0), è rimasto per fare girare ancora le vecchie applicazioni e applet Java, ma è dichiarato deprecated. In effetti questo modello è stato cambiato perché in situazioni complesse, ovvero con molti componenti messi uno sull'altro, non si riusciva a capire quale componente stesse ricevendo l'evento.
EventQueue, coda di eventi.
FileDialog, dialog speciale che gestisce l'input e l'output dei file. E' molto comoda.
FlowLayout, altro gestore di layout.
Font, fonts per il testo.
FontMetrics, attributi per le font.
Frame, classe già vista che implementa le finestre, con titolo e bordo.
GradientPaint, riempie le figure con colori lineari con gradiente.
Graphics, contesto grafico in cui si può disegnare.
Graphics2D, nuovo contesto grafico, inserito con Java 2, che da altre sofisticate possibilità al disegno.
GraphicsConfiguration, descrive le caratteristiche della desitnazione della grafica che può essere il monitor o la stampante.
GraphicsConfigTemplate, usata per ottenere una GraphicsConfiguration valida.
GraphicsEnvironment, descrive gli ambienti grafici e i font utilizzabili in ogni specifica piattaforma da Java.
GraphicsDevice, descrive il GraphicDevice utilizzabile in un particolare ambiente.
GridBagConstraints, GridBagLayout, GridLayout, gestori di layout.
Image, superclasse per tutte le classi che rappresentano una immagine grafica.

Insets, rappresenta il bordo di un contenitore.
Label, etichetta.
List, lista
MediaTracker, classe di utilità per gestire oggetti multimediali.
Menu, menu di tipo pull-down (a discesa).
MenuBar, barra dei menu.
MenuComponent, superclasse di tutti i componenti dei menu,.
MenuItem, Voce di menu.
MenuShortcut, serve a rappresentare l'acceleratore per una voce di menu.
Panel, contenitore.
Point, punto nelle coordinate x,y.
Polygon, poligono.
PopupMenu, menu di tipo a scomparsa.
PrintJob, serve ad inizializzare e ad eseguire le stampe.
Rectangle, rettangolo.
RenderingHints, contiene aiuti per il rendering usati da Graphics2D.
RenderingHints.Key, definisce i tasti usati per controllare i vari aspetti del rendering.
Scrollbar, barra di scorrimento.
ScrollPane, pannello con due barre di scorrimento.
SystemColor, rappresenta i colori simbolici di un oggetto GUI in un sistema.
TextComponent, superclasse di tutti i componenti che contengono testo.
TextArea, area di testo.
TextField, singola linea di testo.
TexturePaint, come riempire una figura con una texture.
Toolkit, superclasse dell' Abstract Window Toolkit.
Window, finestra (senza bordo e barra di menu).

Eccezioni
AWTException
IllegalComponentStateException

Errori
AWTError

Vista questa panoramica sul package cerchiamo di capire a cosa servono queste classi.
Tra tutte le classi, distinguiamo alcune classi nel package per quello che fanno.
Innanzitutto abbiamo visto delle classi Frame, Dialog e Windows, le quali rappresentano delle finestre, queste finestre conterranno dei menu (le prime due) e dei componenti GUI. Per contenerli Devono avere dei contenitori, di questa specie abbiamo visto Container e Panel. Ogni Contenitore può contenere un Layout Manager, che gestisce il posizionamento di più componenti nello stesso contenitore, e può contenere anche altri contenitori, in modo da creare strutture anche complesse.
Infine abbiamo visto i componenti GUI veri e propri che vanno aggiunti ai contenitori, ad esempio Label.
Nell'esempio che segue attacchiamo una label al contenitore di un Frame.

```
import java.awt.*; public class Etichetta extends Frame
{

Label et=new Label();

public Etichetta(String etichetta)
{


et.setText(etichetta);
setTitle("Etichetta "+etichetta);
add(et);
pack();
show();

}

public static void main (String[] a)
{

try
```

```
{new Etichetta(a[0]);}
catch (ArrayIndexOutOfBoundsException e)
{System.out.println("ERRORE: battere java Etichetta STRINGA");};

}

}
```

Come abbiamo già detto, i GUI vanno sia su Frame che su Applet, quindi si può pensare alla versione applet del programma precedente.

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

public class ApplEtichetta extends Applet
{

Label et=new Label();

public void init ()
{

et.setText("Ciao");
add(et);


}


}
```

Che sarà messo in un file chiamato ApplEtichetta.java e sarà chiamato da un file html contenente:

```
<html>
<head>
<title>

</title>
</head>
<body>
<APPLET code="ApplEtichetta.class" width=200 height=100>
</APPLET>
</body>
</html>
```

Nel package swing è tutto molto simile, anche se la gestione dei contenitori è leggermente differente.

LEZIONE 21: *Le etichette ed i bottoni*

Iniziamo la panoramica sui GUI, per usare nella stessa finestra più componenti abbiamo bisogno di uno o più contenitori, questi li vedremo in seguito, però in alcuni casi di questo paragrafo dovrò usarli senza averli spiegati, perché mi necessiteranno più di un componente GUI nella finestra.
Sono sicuro che non avrete problemi nel comprendere lo stesso i programmi, perché userò in modo semplice sia i Layout che i Container, però mi scuso lo stesso se questo vi arrecherà anche un minimo disagio.
In particolare userò i Panel e userò delle costanti della classe BorderLayout, la quale divide il Container, in questo caso il Panel, in cinque zone, NORTH, SOUTH, WEST, EAST e CENTER.
In questa lezione esamineremo due componenti, i più semplici, le Label e i Bottoni. Le prime non hanno nessun evento associato mentre i secondi, a seconda del tipo di bottone hanno due tipi di eventi associati.

Etichette
Iniziamo la nostra panoramica sui componenti vedendo il primo semplice elemento, l'etichetta, che tra l'altro abbiamo già usato. L'etichetta è un contenitore per un testo, essa fa apparire sullo schermo una singola linea di testo non editabile dall'utente, che però può essere cambiata dall'applicazione.
L'etichetta, chiamata in Java Label, si trova nel package java.awt, quindi per essere usata il nostro

programma dovrà cominciare con una import java.awt.Label (per inglobare solo la classe Label del package), oppure import java.awt.* (per inglobare tutte le classi del package). All'interno del programma che la vuole usare bisognerà quindi dichiarare un oggetto di tipo Label nel seguente modo:

Label etichetta = new Label();

oppure, se non è stato importato il package java.awt:

java.awt.Label etichetta = new Label();

La Label ha tre costruttori:

Label() , che costruisce un'etichetta vuota. *Label(String text)*, che costruisce un'etichetta con testo text, giustificata a sinistra. *Label(String text, int alignment)* , che costruisce un'etichetta con testo text, giustificata secondo alignment. I valori possibili di alignment sono: Label.LEFT, Label.RIGHT, e Label.CENTER.

Alcuni metodi dell'oggetto sono:

int getAlignment(), da la giustificazione attuale dell'oggetto.
String getText() , da il testo contenuto nella label.
void setAlignment(int alignment), cambia la giustificazione della Label con alignment.
void setText(String text), setta il testo della Label.

Poi c'è il metoto AccessibleContext getAccessibleContext(), che vedremo quando vedremo i contenitori, infatti la Label è in effetti un contenitore di testo.
Label proviene dalla seguente gerarchia:

```
java.lang.Object
  |
  +--java.awt.Component
       |
       +--java.awt.Label
```

Quindi eredita i soliti metodi di Object e quelli di java.awt.Component. Tutti i componenti GUI sono ereditati da Component, quindi alla fine della panoramica dei GUI parleremo di Component, la quale contiene metodi molto utili per la gestione di questi componenti, come ad esempio metodi per settare e ottenere informazioni su dimensioni, colori dello sfondo, colori del testo, bordi, ecc… del componente.
A parte i metodi di Component, siamo pronti ad usare le Label nei nostri programmi. Per ora non faccio nessun esempio dell'uso delle Label, le vedremo ne prossimo paragrafo quando parleremo dei bottoni.
Il package swing mette a disposizione un'naloga classe denominata JLabel, essa è una estensione di Label, e mette a disposizione una vasta gamma di funzionalità supplementari, ad esempio è possibile creare JLabel che hanno sia un testo che un'immagine.

Bottoni
Un altro componente molto usato, e di semplice realizzazione è il bottone, esso è leggermente più complicato della Label, in quanto può ricevere degli eventi (Click sul bottone).
I bottoni sono di tre tipi, i bottoni normali, le Checkbox e i radiobutton, iniziamo a vedere i primi.
La classe che definisce i bottoni è Button di java.awt, essa mette a disposiozione tutti i metodi per gestire l'aspetto del bottone. I costruttori della classe sono due:

Button(), costruisce un bottone senza etichetta;
Button(String txt), costruisce un bottone con etichetta txt.

Alcuni metodi della classe sono:

void addActionListener(ActionListener l), aggiunge il bottone ad un oggetto chiamato *ActionListener*, il quale è un ascoltatore di eventi. Quando il bottone verrà cliccato, l'evento verrà gestito dall'ActionListener.
AccessibleContext getAccessibleContext(), come per le Label, i bottoni sono anch'essi dei contenitori.
String getActionCommand(), ogni bottone ha associato un comando, questo ne restituisce quello attuale.
String getLabel(), restituisce l'etichetta associata al bottone.
EventListener[] getListeners(Class listenerType), da un array contenente tutti gli ascoltatori di aventi definiti per questo bottone.
addXXXListener(), aggiunge un ascoltatore di eventi, l'ascoltatore di eventi di tipo XXX. In Java esistono vari

tipi di eventi, li vedremo tra poco, uno di questi è l'evento di tipo action, in questo caso il metodo è *addActionListener( )*
void removeActionListener(ActionListener l), elimina l'ascoltatore di eventi l precedentemente associato al bottone.
void setActionCommand(String command), setta il comando associato al bottone, inizialmente il comando è la label.
void setLabel(String label), setta la Label (etichetta) del bottone.

Siamo pronti per definire il nostro primo bottone, dobbiamo solo riuscire a capire come catturarne l'evento scaturito dalla pressione dello stesso. Come già detto in precedenza io parlerò dellagestione degli eventi secondo Java 2, perché il vecchio modello di gestione degli eventi di Java 1 è obsoleto e non funzionale.
Secondo Java 2, definito un bottone, visto che l'evento principale del bottone è il click, che è un evento appartenente alla famiglia degli eventi Action, bisogna associare un ActionListener allo stesso bottone, usando il metodo addActionListaner(ActionListener l); Quello che dobbiamo fare noi è creare un ActinListener specializzato per il nostro bottone (o per i nostri bottoni).
ActionListener è definito nel package java.awt.event (da includere nel programma) esso è una interfaccia, quindi deve essere implementata, all'interno della classe in cui è definita.
Ad esempio, sto facendo un Frame chiamato Pippo, contenente un bottone, di cui voglio ascoltare il click, lo voglio ascoltare con un ActionListener specializzato da me chiamato AscoltaPippo, dovrò scrivere qualcosa del tipo:

```
import java.awt.*;
import java.awt.event.*;
public class Pippo extends Frame
{
Button bottone=new Button("CIAO");
… metodi di Pippo, tra cui il costruttore, che conterrà un bottone.addActionListener(new AscoltaPippo());

public class AscoltaPippo implements ActionListener()
{
// Gestione dell'evento
} // Fine classe AscoltaPippo interna a Pippo
}//Fine classe Pippo
```

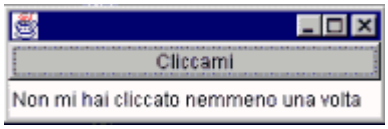
In AscoltaPippo dovrò specializzare tutti i metodi dell'interfaccia che sto estendendo, ActionListener ha un solo metodo:

void actionPerformed(ActionEvent e) quindi in AscoltaPippo dovrò definire il metodo public void actionPerformed(ActionEvent e), è questo il metodo che verrà invocato quando cliccherò il bottone.
L'ActionEvent che mi arriva nel metodo come parametro conterrà tutte le informazioni riguardanti l'evento, tra cui il comando. AscoltaPippo può essere associato anche a più bottoni, la sua actionPerformed verrà invocata al click di ognuno di questi bottoni. Chiariamo subito il concetto, che forse è un pochino più difficile delle altre cose che abbiamo fatto con un piccolo esempio, vi assicuro però che una volta assimilato il concetto, la gestione degli eventi di Java è molto semplice.
Creiamo un Frame che ha un bottone e una Label, ogni volta che il bottone viene cliccato la label da un messaggio.

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

public class Bottone extends Frame
{
// Costruttore classe Bottone

Button cliccami=new Button("Cliccami");

Label cliccato=new Label("Non mi hai cliccato nemmeno una volta");

public Bottone()
{
cliccami.addActionListener(new Ascoltatore());
// setup comando
cliccami.setActionCommand("CLICK");
// Aggiungo il bottone e la label al Frame.
// Non badate alle seguenti istruzioni,
```

```
// la add serve ad aggiungere un componente ad
// un contenitore, e il secondo parametro della
// add, ovvero BorderLayout, è un gestore di Layout,
// che serve a stabilire il modo in cui gli oggetti
// GUI vengono posti nel contenitore.

add(cliccami,BorderLayout.NORTH);
add(cliccato,BorderLayout.SOUTH);


// metodi di Frame
pack();
show();
}

// main

public static void main (String [] arg)
{

new Bottone();

}

// Ascoltatore di eventi Action

int Volte=2;

public class Ascoltatore implements ActionListener
{

public void actionPerformed (ActionEvent e)
{

String Comando=e.getActionCommand();

if (Comando.compareTo("CLICK")==0)
{

cliccato.setText("Mi hai cliccato");

cliccami.setLabel("Ricliccami");

cliccami.setActionCommand("RECLICK");

};

if (Comando.compareTo("RECLICK")==0)
cliccato.setText("Mi hai cliccato "+(Volte++)+" volte.");

}

}// Fine Ascoltatore

}// Fine Bottone
```

Il risultato è la seguente finestra:



Come si vede in Ascoltatore, vengono gestiti due comandi differenti associati allo stesso bottone, ed il bottone quando viene cliccato la prima volta cambia il comando associato all'evento click.
Provate a modificarlo cambiando il comando per stabilire con certezza le prime dieci volte che viene cliccato e stampando nella label "Hai cliccato nove volte il bottone" (la nona volta).

Ricapitolando, in una interfaccia grafica innanzitutto si programma l'aspetto grafico, ovvero si inseriscono i bottoni, i menu ecc, poi si creano e aggiungono gli ascoltatori di eventi, i quali a seconda dell'evento che si è verificato chiamano uno o l'altro metodo di gestione del programma (aggiorna l'interfaccia, calcola le funzioni del programma di cui è interfaccia grafica).
Questo è un principio generale della programmazione di Interfacce Grafiche, ovvero della programmazione ad eventi, ogni programma che usate funziona così, ed ogni programma che scriverete funzionerà così.
Il tipo di programmazione è leggermente differente da quella a cui un programmatore su piattaforma non grafiche (tipo Dos, Linux senza X-Win) sono abituati, e all'inizio può risultare non molto semplice, ma ci si deve solo abituare, che poi è lo stesso.

Il secondo tipo di bottoni che vediamo sono i Checkbox, ovvero quei bottoni di tipo booleano, che trattengono lo stato. Si usano per fare delle scelte, tra di loro non esclusive o esclusive (in questo caso sono dei radiobutton), a esempio se si devono scegliere alcuni tra gli hobby preferiti. Questi bottoni una volta cliccati rimangono tali, e bisogna ricliccarli per farli tornare nella situazione di partenza.
Per creare una Checkbox bisogna creare un oggetto appartenente alla classe Checkbox di java.awt, i costruttori possibili sono i seguenti:

Checkbox(), crea una Checkbox senza etichetta.
Checkbox(String label), crea una Checkbox con etichetta label.
Checkbox(String label, boolean state), crea una Checkbox con etichetta label, e cliccata o meno a seconda del valore booleano state.

Invece per le radiobutton:

Checkbox(String label, boolean state, CheckboxGroup group), crea una Checkbox con etichetta label inserita in una specifica
CheckboxGroup, che è un contenitore di Checkbox.
Checkbox(String label, CheckboxGroup group, boolean state) crea una Checkbox con etichetta label inserita in una specifica
CheckboxGroup, e settata a seconda del valore di state.

In pratica tra tutti i bottoni inseriti nel CheckboxGroup, solo uno può essere cliccato, gli altri devono essere non cliccati. Le Checkbox del primo tipo come ho detto vengono usate per effettuere delle scelte non esclusive, come ad esempio la scelta degli interessi tra una lista di interessi possibili, mentre le Checkbox del secondo tipo, ovvero i radiobuttons servono per fare delle scelte esclusive, ad esempio la scelta del reddito di una persona tra tante possibili fasce.

Tra i metodi della classe Checkbox alcuni più interessanti sono:

void addItemListener(ItemListener l) , associa un ItemListener alla Checkbox, l'ItemListener gestisce gli eventi dei Checkbox e non solo, sono un secondo tipo di eventi che impareremo a gestire, comunque somigliano agli ActionListener, solo che sono associati ad oggetti che hanno uno stato.
CheckboxGroup getCheckboxGroup(), da l'eventuale gruppo in cui è inserita la Checkbox.
Le solite *String getLabel()* e *EventListener[] getListeners(Class listenerType) boolean getState()*, questo metodo dice se la Checkbox è cliccataoppure no.
void removeItemListener(ItemListener l), toglie l'attuale ascoltatore di eventi Item.
void setCheckboxGroup(CheckboxGroup g), setta un gruppo per la Checkbox.
void setLabel(String label), setta l'etichetta
void setState(boolean state), setta lo stato.

Vediamo anche cosa contiene un CheckboxGroup. Il costruttore è uno senza parametri, e i metodi non dichiarati Deprecated sono tre:

Checkbox getSelectedCheckbox(), da la Checkbox selezionata nel gruppo.
void setSelectedCheckbox(Checkbox box), setta la Checkbox del gruppo indicata.
String toString(), da una stringa rappresentante l'oggetto.

Proviamo quindi a fare un esempietto che usa sia Checkbox che RadioButton.

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

public class Check extends Frame
{
// Costruttore classe Bottone

Label r=new Label("Reddito annuo");
Label i=new Label("Interessi");

Button chiudi=new Button("Chiudi Applicazione");

CheckboxGroup reddito=new CheckboxGroup();

Checkbox r1=new Checkbox("Da 0 ai 10");
Checkbox r2=new Checkbox("Dagli 11 ai 30");
Checkbox r3=new Checkbox("Dai 31 ai 70");
Checkbox r4=new Checkbox("Dai 71 ai 100");
Checkbox r5=new Checkbox("Dai 101 a 200");
Checkbox r6=new Checkbox("Dai 201 a 500");
Checkbox r7=new Checkbox("Sono ricco sfondato");

Checkbox i1=new Checkbox("Sport");
Checkbox i2=new Checkbox("Informatica");
Checkbox i3=new Checkbox("Lettura");
Checkbox i4=new Checkbox("Cinema");
Checkbox i5=new Checkbox("Animali");
Checkbox i6=new Checkbox("Fumetti");
Checkbox i7=new Checkbox("Lotterie");
Checkbox i8=new Checkbox("Donne nude");
Checkbox i9=new Checkbox("Uomini nudi");



public Check()
{
chiudi.addActionListener(new Ascoltatore());


Panel p1=new Panel();

Panel p2=new Panel();

p1.add(r);
p1.add(r1);
p1.add(r2);
p1.add(r3);
p1.add(r4);
p1.add(r5);
p1.add(r6);
p1.add(r7);

r1.setCheckboxGroup(reddito);
r2.setCheckboxGroup(reddito);
r3.setCheckboxGroup(reddito);
r4.setCheckboxGroup(reddito);
```

```
r5.setCheckboxGroup(reddito);
r6.setCheckboxGroup(reddito);
r7.setCheckboxGroup(reddito);



add(p1,BorderLayout.NORTH);


p2.add(i);
p2.add(i1);
p2.add(i2);
p2.add(i3);
p2.add(i4);
p2.add(i5);
p2.add(i6);
p2.add(i7);
p2.add(i8);
p2.add(i9);



add(p2,BorderLayout.CENTER);

add(chiudi,BorderLayout.SOUTH);

// metodi di Frame

pack();
show();

}

// main

public static void main (String [] arg)
{

new Check2();

}


// Ascoltatore di eventi Action

public class Ascoltatore implements ActionListener
{

public void actionPerformed (ActionEvent e)
{

System.out.println("Iscrizione al sito XXXXXXX.it");

System.out.print("Guadagno annuo ");
if (r1.getState()) System.out.println(r1.getLabel());
else if (r2.getState()) System.out.println(r2.getLabel());
else if (r3.getState()) System.out.println(r3.getLabel());
else if (r4.getState()) System.out.println(r4.getLabel());
else if (r5.getState()) System.out.println(r5.getLabel());
else if (r6.getState()) System.out.println(r6.getLabel());
else if (r7.getState()) System.out.println(r7.getLabel()+" (Beato te!)");
else System.out.println("NON DICHIARATO");

System.out.println("Interessi dichiarati:");
if (i1.getState()) System.out.println("\t"+i1.getLabel());
```

```
if (i2.getState()) System.out.println("\t"+i2.getLabel());
if (i3.getState()) System.out.println("\t"+i3.getLabel());
if (i4.getState()) System.out.println("\t"+i4.getLabel());
if (i5.getState()) System.out.println("\t"+i5.getLabel());
if (i6.getState()) System.out.println("\t"+i6.getLabel());
if (i7.getState()) System.out.println("\t"+i7.getLabel());
if (i8.getState()) System.out.println("\t"+i8.getLabel());
if (i9.getState()) System.out.println("\t"+i9.getLabel());

System.exit(0);
}

}// Fine Ascoltatore

}// Fine Bottone
```

Fig 4: Finestra del programma

Ho dovuto di nuovo usere i layout e anche i Panel, me ne scuso perché ancora non ne parliamo, ma è l'unico modo per inserire più GUI nella stessa finestra, per esempio provate a togliere i Panel e quindi ad aggiungere tutto direttamente alla finestra, senza usare i Layout, vedrete cosa succede.
Nell'applicazione c'è un bottone che serve a chiudere la finestra, questo perché ancora non siamo in grado di gestire gli eventi della finestra, come il close, che se lo provate vedrete che non va. Tra qualche tempo saremo in grado di gestire anche quelli.

Nell'applicazione non ho gestito gli eventi delle Checkbox, questo perché non ne ho avuto bisogno, esistono situazioni in cui si devono gestire, e sono quelle in cui oltre allo stato del bottone cambia qualcosa, ad esempio lo stato di qualche GUI, come nell'esempio seguente.

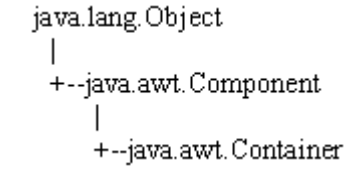
```
import java.awt.*;
import java.awt.event.*;

public class Check2 extends Frame
{
// Costruttore classe Bottone


Button chiudi=new Button("Chiudi Applicazione");

Checkbox i1=new Checkbox("primi 3");
Checkbox i2=new Checkbox("secondi 4");

Checkbox ii1=new Checkbox("uno");
Checkbox ii2=new Checkbox("due");
Checkbox ii3=new Checkbox("tre");
Checkbox ii4=new Checkbox("quattro");
Checkbox ii5=new Checkbox("cinque");
Checkbox ii6=new Checkbox("sei");
Checkbox ii7=new Checkbox("sette");



public Check2()
{
chiudi.addActionListener(new Ascoltatore());


Panel p1=new Panel();

Panel p2=new Panel();

p1.add(ii1);
p1.add(ii2);
p1.add(ii3);
p1.add(ii4);
```

```
p1.add(ii5);
p1.add(ii6);
p1.add(ii7);


ii1.addItemListener(new AItem1());
ii2.addItemListener(new AItem1());
ii3.addItemListener(new AItem1());
ii4.addItemListener(new AItem1());
ii5.addItemListener(new AItem1());
ii6.addItemListener(new AItem1());
ii7.addItemListener(new AItem1());


i1.addItemListener(new AItem2());
i2.addItemListener(new AItem2());

add(p1,BorderLayout.NORTH);

p2.add(i1);
p2.add(i2);



add(p2,BorderLayout.CENTER);

add(chiudi,BorderLayout.SOUTH);

// metodi di Frame

pack();
show();

}

// main

public static void main (String [] arg)
{

new Check();

}


// Ascoltatore di eventi Action

public class Ascoltatore implements ActionListener
{

public void actionPerformed (ActionEvent e)
{System.exit(0);}

}// Fine Ascoltatore

// Ascoltatori di eventi Item
public class AItem1 implements ItemListener
{
public void itemStateChanged(ItemEvent e)
{

i1.setState(ii1.getState()&&
ii2.getState()&&
ii3.getState());
```

```
i2.setState(ii4.getState()&&
ii5.getState()&&
ii6.getState()&&
ii7.getState());
}

}//Fine AItem1


public class AItem2 implements ItemListener
{
public void itemStateChanged(ItemEvent e)
{
ii1.setState(i1.getState());
ii2.setState(i1.getState());
ii3.setState(i1.getState());
ii4.setState(i2.getState());
ii5.setState(i2.getState());
ii6.setState(i2.getState());
ii7.setState(i2.getState());
}

}//Fine AItem2


}// Fine Bottone
```

LEZIONE 22: Contenitori e Gestione dei Layout

Nella lezione precedente abbiamo visto come inserire nelle nostre applicazioni e nei nostri applets delle etichette e dei bottoni. Per fare questo abbiamo usato dei contenitori e dei layout, senza sapere come funzionassero. In questa lezione vedremo il loro funzionamento, e come sfruttarli per rendere migliori le nostre interfacce.
Iniziamo a descrivere la classe Container del package java.awt, esso è una estensione della classe Component, dalla quale provengono tutti i componenti GUI, ed in effetti anc'esso è un componente GUI.



Un Container è un contenitore per oggetti GUI, e quindi anche per altri container. Esso se usato insieme ad un Layout Manager permette di contenere anche più di un GUI (e Contenitori), permettendo di fare apparire più oggetti nelle nostre interfacce. Vediamo cosa contiene la classe.
Container() , è l'unico costruttore della classe.

Alcuni metodi: *Component add(Component comp)*, aggiunge il componente alla fine del contenitore.
Component add(Component comp, int index), aggiunge il componente nella posizione indicata del contenitore.
void add(Component comp, Object constraints) e void add(Component comp, Object constraints, int index) , come prima, solo che si specificano delle costanti per il componente inserito.
Component add(String name, Component comp), aggiunge al contenitore il componente e gli da un nome.
void addContainerListener(ContainerListener l), setta il ContainerListener che ascolterà gli eventi occorsi al contenitore.
void doLayout(), causa la disposizione dei componenti nel contenitore.
Component findComponentAt(int x, int y), restituisce il componente che si trova nella posizione specificata del contenitore (posizione grafica).
Component findComponentAt(Point p) , restituisce il componente che si trova nella posizione specificata del contenitore (posizione grafica).
float getAlignmentX(), restituisce l'allineamento lungo l'asse X del contenitore.
float getAlignmentY(), restituisce l'allineamento lungo l'asse Y del contenitore.
Component getComponent(int n), restituisce l'ennesimo componente contenuto nel contenitore.
Component getComponentAt(int x, int y), restituisce il componente che si trova nella posizione specificata del

contenitore (posizione grafica).
Component getComponentAt(Point p), restituisce il componente che si trova nella posizione specificata del contenitore (posizione grafica).
int getComponentCount() , da il numero di componenti conteinuti nel contenitore.

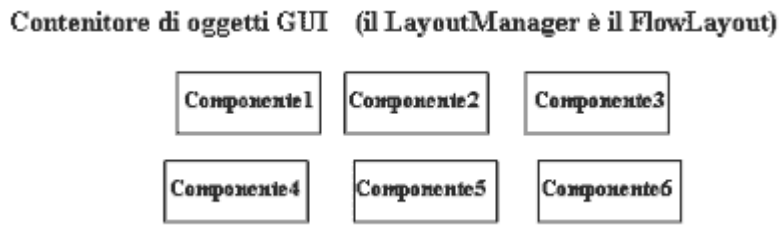
Component[] getComponents(), da tutti i componenti contenuti nel contenitore.
Insets getInsets(), da un'oggetto Insets per il contenitore, questo oggetto rappresenta le dimensioni (grafiche) del contenitore.
LayoutManager getLayout(), restituisce il LayuotManager associato al contenitore.
EventListener[] getListeners(Class listenerType), restituisce tutti gli ascoltatori di eventi associati al contenitore.
Dimension getMaximumSize(), restituisce la dimensione massima che può assumere il contenitore.
Dimension getMinimumSize(), restituisce la dimensione minima che può assumere il contenitore.
Dimension getPreferredSize(), restituisce la dimensione preferita per il contenitore.
void list(PrintStream out, int indent), stampa la lista dei componenti contenuti nel contenitore, su uno stream di output.
void list(PrintWriter out, int indent), Stampa su una stampante i componenti contenuti nel contenitore.
void paint(Graphics g), disegna il contenitore. La paint è una funzione molto importante, vedremo come ridefinirla per disegnare sulla finestra.
void paintComponents(Graphics g). disegna tutti i componenti del contenitore.
void print(Graphics g), stampa il contenitore (l'aspetto grafico).
void printComponents(Graphics g), stampa tutti i componenti contenuti nel contenitore (il loro aspetto grafico).
void remove(Component comp), elimina il componente specificato dal contenitore.
void remove(int index), rimuove il componente che si trova nella posizione specificata nel contenitore.
void removeAll(), rimuove tutti i componenti contenuti nel contenitore.
void removeContainerListener(ContainerListener l), elimina l'ascoltatore di eventi per containers da questo container.
void setFont(Font f), setta le Font usate per il testo nel contenitore.
void setLayout(LayoutManager mgr), associa al contenitore un gestore di layout.
void update(Graphics g), ridisegna il contenitore.
void validate(), invalida il contenitore e i suoi componenti.

Inserendo componenti nel contenitore, come sel successivo esempio, ci accorgiamo che solo l'ultimo viene visualizzato, questo perché non vi abbiamo dato un gestore di Layout, nel caso di applet invece, il LayoutManager di default è il FlowLayout.

```
import java.awt.*;

public class Contenitore extends Frame
{

Label l1=new Label("Etichetta1");
Label l2=new Label("Etichetta2");
Label l3=new Label("Etichetta3");
Label l4=new Label("Etichetta4");
Label l5=new Label("Etichetta5");

public Contenitore()
{
// uso add, perchè il Frame è una estensione di Window, che a sua
// votla estende Container.

add(l1);
add(l2);
add(l3);
add(l4);
add(l5);
doLayout();
pack();
show();
}

public static void main(String [] arg)
{
```

```
new Contenitore();

}

}
```

Quindi vediamo come inserire un gestore di Layout nel contenitore, il metodo della classe container che lo permette è void setLayout(LayoutManager mgr), bisogna solo vedere come è l'oggetto di tipo LayoutManager.
LayoutManager è una interfaccia, tra le calssi che la implementano vedremo GridLayout, FlowLayout.
Esiste un'altra interfaccia denominata LayoutManager2, che estende questa, e che ha altre classi che la implementano, di queste vedremo: CardLayout, BorderLayout, GridBagLayout, BoxLayout, OverlayLayout.
Quindi nel metodo setLayout (…) possiamo usare come parametro un'oggetto di queste classi.
A questo punto dobbiamo capire qual è la differenza tra i vari layout manager.
Il FlowLayout permette di disporre i componenti di un contenitore da sinistra verso destra su un'unica linea.



Per usare il LayoutManager FlowLayout nell'esempio precedente basta invocare il metodo setLayout con parametro new FlowLayout(), e poi i componenti vengono automaticamente inseriti dalla add da destra verso sinistra sulla stessa linea.

```
import java.awt.*;

public class ContEFL extends Frame
{

Label l1=new Label("Etichetta1");
Label l2=new Label("Etichetta2");
Label l3=new Label("Etichetta3");
Label l4=new Label("Etichetta4");
Label l5=new Label("Etichetta5");

public ContEFL()
{
// uso add, perchè il Frame è una estensione di Window, che a sua
// votla estende Container.

setLayout(new FlowLayout());

add(l1);
add(l2);
add(l3);
add(l4);
add(l5);

pack();
show();

}

public static void main(String [] arg)
{

new ContEFL();
```

```
}

}
```

La finestra risultante è:

Il GridLayout permette di disporre i componenti in un contenitore come se fossero disposti su una griglia.

Tutti i componenti assumeranno a stessa dimensione. La classe ha tre costruttori, uno senza parametri che crea una griglia di 1 riga, praticamente un FlowLayout, e altri due che permettono di specificare la dimensione della griglia.
L'esempio precedente cambia così:

```
import java.awt.*;

public class ContEGL extends Frame
{

Label l1=new Label("Etichetta1");
Label l2=new Label("Etichetta2");
Label l3=new Label("Etichetta3");
Label l4=new Label("Etichetta4");
Label l5=new Label("Etichetta5");

public ContEGL()
{
// uso add, perchè il Frame è una estensione di Window, che a sua
// votla estende Container.

setLayout(new GridLayout(2,3));

add(l1);
add(l2);
add(l3);
add(l4);
add(l5);

pack();
show();

}
```

```
public static void main(String [] arg)
{

new ContEGL();

}

}
```

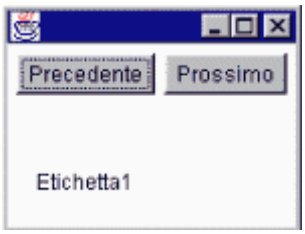
Il risultato è la seguente finestra:

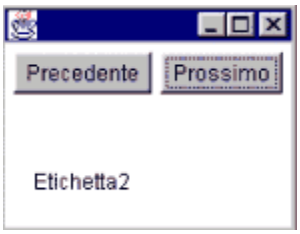
Il LayoutManager CardLayout permette di visualizzare componenti diversi in tempi diversi, ovvero esso visualizza solo un componente alla volta, ma il componente visualizzato può essere cambiato.
Ecco un esempio dell'uso di CardLayout

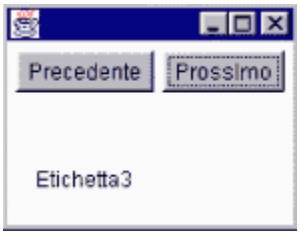
```
import java.awt.*;
import java.awt.event.*;

public class ContECL extends Frame
{

Label l1=new Label("Etichetta1");
Label l2=new Label("Etichetta2");
Label l3=new Label("Etichetta3");
Label l4=new Label("Etichetta4");
Label l5=new Label("Etichetta5");

Panel p=new Panel(new GridLayout(2,1));
CardLayout CL=new CardLayout(14,14);
Panel p1=new Panel(CL);
Panel NPB=new Panel(new FlowLayout());

public ContECL()
{



p.add(NPB);
p.add(p1);

Button N=new Button("Prossimo");
Button P=new Button("Precedente");

NPB.add(P);
NPB.add(N);

P.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
CL.previous(p1);
}
}
);

N.addActionListener(new ActionListener()
```

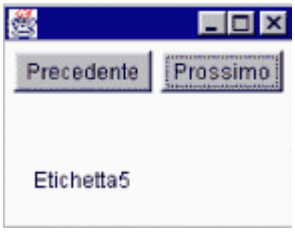
```
{
public void actionPerformed(ActionEvent e)
{
CL.next(p1);
}
}
);

p1.add("uno",l1);
p1.add("due",l2);
p1.add("tre",l3);
p1.add("quattro",l4);
p1.add("cinque",l5);

add(p);
pack();
show();

}

public static void main(String [] arg)
{

new ContECL();

}

}
```

Il risultato dell' esempietto sono le seguenti finestre:









Che mostrano come le cinque etichette vengono mostrate nella stessa posizione in tempi differenti.
Da notare nell'esempio che sono stati usati anche altri gestori di Layout, questo per inserire i bottoni Prossimo e Precedente.

BorderLayout è uno dei gestori di Layout più usati, esso permette di inserire in un contenitore cinque componenti, uno a Nord del contenitore, uno a Sud, uno a Est , uno ad Ovest, ed uno al centro.
La dimensione del contenitre verrà data dal componente centrale, le cinque posizioni sono indicate dalle costanti della classe:

BorderLayout.NORTH
BorderLayout.SOUTH
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.CENTER

Ecco un'esempio di uso di BorderLayout. Nell'esempio ho cambiato i colori di sfondo delle etichette per far vedere il componente intero.

```
import java.awt.*;

public class ContEBL extends Frame
{

Label l1=new Label("Etichetta a Nord",Label.CENTER);
Label l2=new Label("Etichetta a Sud",Label.CENTER);
Label l3=new Label("Etichetta a Est",Label.CENTER);
Label l4=new Label("Etichetta a Ovest",Label.CENTER);
Label l5=new Label("Etichetta al Centro",Label.CENTER);

public ContEBL()
{
// uso add, perchè il Frame è una estensione di Window, che a sua
// votla estende Container.

l1.setBackground(Color.pink);
l2.setBackground(Color.lightGray);
l3.setBackground(Color.green);
l4.setBackground(Color.yellow);
l5.setBackground(Color.orange);


setLayout(new BorderLayout());

add(l1,BorderLayout.NORTH);
add(l2,BorderLayout.SOUTH);
add(l3,BorderLayout.EAST);
add(l4,BorderLayout.WEST);
add(l5,BorderLayout.CENTER);

pack();
show();

}

public static void main(String [] arg)
```

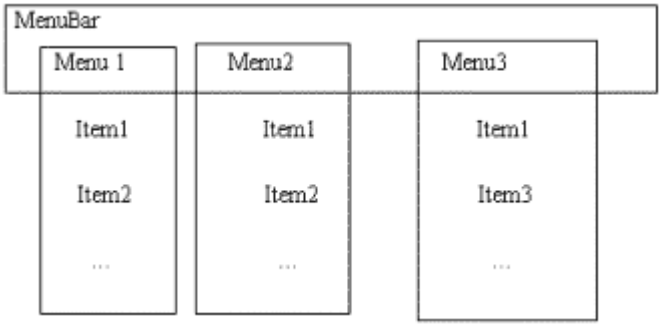
```
{

new ContEBL();

}

}
```

Il risultato è:

Esistono anche altri gestori di Layout, che non vedremo, che però potete studiare insieme a questi nella documentazione ufficiale delle Java Development Kit.

LEZIONE 23: *Menu*

Un menu in una applicazione non è altro che una MenuBar in cui vi sono vari menu.
Si pensi ad un programma qualsiasi con le voci di menu File Edit ed Help, queste tre voci In Java sono degli oggetti della classe Menu, e vanno aggiunte ad un oggetto della classe MenuBar il quale va attaccato alla finestra. Ogni menu ha varie voci, ad esempio il menu File avrà le voci: Apri, Chiudi, Salva e Esci, questi in Java sono degli oggetti della classe MenuItem (o anche Menu se conterranno altri sottomenu).



Quindi se ad una applicazione vogliamo aggiungere un menù dobbiamo in qualche ordine fare le seguenti cose:

- Creare gli oggetti MenuItem
- Creare gli oggetti menu ed attaccarvi i MenuItem
- Creare una MenuBar e attaccare i Menu

E poi bisogna al solito scrivere dei gestori per gli eventi provenienti dai menu ed associarli ai menu.
Vediamo in pratica come si costruisce un menu, iniziamo dai MenuItem:
Gli eventi dei MenuItem sono quelli che devono essere gestiti da noi, a differenza degli eventi dei menu che li gestisce il sistema, infatti mentre i secondi servono a fare apparire e scomparire le voci di menu, i primi sono i click sul comando corrispondente all'Item.
Quindi per questi dovremo scrivere degli ActionListener, come per i bottoni, infatti essi non sono altro che dei bottoni speciali. I costruttori sono tre:

MenuItem() , che costruisce un MenuItem senza etichetta.
MenuItem(String label), che costruisce un MenuItem con etichetta label.
MenuItem(String label, MenuShortcut s), che costruisce un MenuItem con etichetta label e accelleratore (tasto di scelta rapida) definito in MenuShortcut s.

Alcuni metodi sono:

addActionListener(ActionListener l), associa un ActionListener al MenuItem per sentirne gli eventi di tipo

ActionEvent (il click).
void deleteShortcut(), cancella il tasto di scelta rapida per il menuitem.
String getActionCommand(), da l'azione associata al MenuItem, l'azione è quella passata alll'actionListener del bottone per identificare il bottone stesso, infatti più item possono avere lo stesso gestore di eventi, il quale potrà distinguere il bottone cliccato in base al comandoo che gli arriva.
String getLabel(), restituisce l'etichetta del MenuItem
EventListener[]getListeners(Class listenerType) , restituisce tutti gli ascoltatori di eventi associati al MenuItem, deltipo listenerType.
MenuShortcut getShortcut(), restituisce la definizione dell'acceleratore per il MenuItem.
boolean isEnabled(), dice se il menu è abilitato o meno, se è disabilitato verrà visualizzato ingrigito.
void removeActionListener(ActionListener l), elimina l'ascoltatore associato.
void setActionCommand(String command), setta il comando associato al MenuItem, se non è specificato, il comando è l'etichetta del MenuItem.
void setEnabled(boolean b), abilita e disabilita il MenuItem.
void setLabel(String label), setta l'etichetta per il MenuItem.
void setShortcut(MenuShortcut s), definisce l'acceleratore per il menu.

Quindi per creare gli Item del menu File descritto prima dovremo scrivere:

```
MenuItem apri=new MenuItem("Apri");
MenuItem chiudi=new MenuItem("Chiudi");
MenuItem salva=new MenuItem("Salva");
MenuItem esci=new MenuItem("Esci");
```

Creiamo quindi l'oggetto Menu per attaccare i MenuItem.
Due costruttori dell'oggetto sono:

Menu(), costruisce un Menu senza etichetta
Menu(String label), costruisce un menu con etichetta label.

Tra i metodi ci sono:

MenuItem add(MenuItem mi), che aggiunge un MenuItem al Menu.
void add(String label), cheaggiunge un'etichetta al Menu (tipo separatore).
void addSeparator(), aggiunge un vero e proprio separatore al menu, ovvero una linea.

Aggiungendo oggetti ad un menù in questo modo, essi verranno posizionati nello stesso ordine in cui vengono aggiunti. Per specificare invece la posizione in cui si vogliono inserire bisogna usare i metodi:

```
void insert(MenuItem menuitem, int index)
void insert(String label, int index)
void insertSeparator(int index)
```

I metodi MenuItem getItem(int index) e int getItemCount() servono per prendere un item data una posizione e per sapere il numero di MenuItem inseriti in un menu.

Per rimuovere gli Item dal menu invece si usano i metodi:

```
void remove(int index)
void remove(MenuComponent item) e
void removeAll()
Quindi per creare il nostro menù File scriveremo:
Menu file= new Menu("File");
file.add(apri);
file.add(salva);
file.add(chiudi);
file.addSeparator();
file.add(esci);
```

A questo punto supponendo di avere creato anche Edit ed Help, dobiamo aggiungerli ad una MenuBar, la quale si crea usando il costruttore MenuBar().
In questa ci sono i metodi remove e removeAll come per i menu, e c'è la add:

```
Menu add(Menu m);
```

Quindi la nostra MenuBar sarà creata nel seguente modo:

```
MenuBar barra=new MenuBar();
barra.add(file);
barra.add(edit);
barra. setHelpMenu(help);
```

Quest'ultimo metodo è un metodo speciale per aggiungere un menu di aiuto.

Infine non rimane che associare la barra di menu alla nostra finestra, questo è semplicissimo, perché la nostra finestra sarà una estensione di Frame, e in Frame c'è il metodo:

```
setMenuBar(MenuBar mb);
```

Nel nostro caso sarà setMenuBar (barra);
ATTENZIONE: con awt non è possibile inserire menu in Dialog e in Applet (quelle che si vedono in giro sono dei finti menu, ovvero sono programmati disegnando dei menu, oppure non sono solo awt), mentre con swing si.
Di seguito c'è il listato di una applicazione che definisce un menu e ne sente gli eventi:

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

public class Finestramenu extends Frame
{

MenuItem apri=new MenuItem("Apri");
MenuItem chiudi=new MenuItem("Chiudi");
MenuItem salva=new MenuItem("Salva");
MenuItem esci=new MenuItem("Esci");

MenuItem taglia=new MenuItem("Taglia");
MenuItem copia=new MenuItem("Copia");
MenuItem incolla=new MenuItem("Incolla");
MenuItem cancella=new MenuItem("Cancella");

MenuItem cerca=new MenuItem("Cerca");
MenuItem rimpiazza=new MenuItem("Rimpiazza");

MenuItem aiuto=new MenuItem("Indice");


Menu file= new Menu("File");
Menu edit= new Menu("Edit");
Menu help= new Menu("Help");

MenuBar barra = new MenuBar();

Label risultato=new Label("Nessuna voce di menu cliccata");

public Finestramenu()
{

setupEventi();

file.add(apri);
file.add(salva);
file.add(chiudi);
file.addSeparator();
file.add("Menu Uscita");
file.addSeparator();
file.add(esci);

edit.add(taglia);
edit.add(copia);
```

```
edit.add(incolla);
edit.add(cancella);
edit.addSeparator();
edit.add(cerca);
edit.add(rimpiazza);

help.add(aiuto);

barra.add(file);
barra.add(edit);
barra.setHelpMenu(help);

setMenuBar(barra);

add(risultato);

pack();
show();
addWindowListener(new FinestramenuWindowListener());
}

void setupEventi()
{
apri.addActionListener(new AscoltatoreMenu());
salva.addActionListener(new AscoltatoreMenu());
chiudi.addActionListener(new AscoltatoreMenu());
esci.addActionListener(new AscoltatoreMenu());

taglia.addActionListener(new AscoltatoreMenu());
copia.addActionListener(new AscoltatoreMenu());
incolla.addActionListener(new AscoltatoreMenu());
cancella.addActionListener(new AscoltatoreMenu());

cerca.addActionListener(new AscoltatoreMenu());
rimpiazza.addActionListener(new AscoltatoreMenu());

aiuto.addActionListener(new AscoltatoreMenu());


}


public static void main(String[] arg)
{

new Finestramenu();

}

class AscoltatoreMenu implements ActionListener
{

public void actionPerformed (ActionEvent e)
{

risultato.setText(" Cliccato "+e.getActionCommand());
if (e.getActionCommand().compareTo("Esci")==0) System.exit(0);

}

}
class FinestramenuWindowListener implements WindowListener
{

public void windowActivated(WindowEvent e)
```

```
{
System.out.println("Sentito un Window Activated");
}

public void windowClosed(WindowEvent e)
{
System.out.println("Sentito un Window Closed");
}

public void windowClosing(WindowEvent e)
{
System.out.println("Sentito un Window Closing");
System.exit(0);
}

public void windowDeactivated(WindowEvent e)
{
System.out.println("Sentito un Window Deactivaded");
}

public void windowDeiconified(WindowEvent e)
{
System.out.println("Sentito un Window Deiconified");
}

public void windowIconified(WindowEvent e)
{
System.out.println("Sentito un Window Iconified");
}

public void windowOpened(WindowEvent e)
{
System.out.println("Sentito un Window Opened");
}

}

}
```

Come si vede, nell'esempio c'è un altro ascoltatore di eventi. Questo sente eventi di tipo WindowEvent, e serve per ascoltare eventi della finestra.
Per associarlo alla finestra si usa il metodo addWindowListener(WindowListener l); dove WindowListener è l'ascoltatore di eventi. Per definire un ascoltatore di eventi bisogna quindi implementare l'interfaccia WindowListener e ridefinirne tutti i metodi:

class MIOASCOLTATORE implements WindowListener

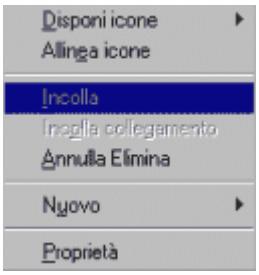
I metodi da ridefinire sono:

public void windowActivated(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowClosing(WindowEvent e)
public void windowDeactivated(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowOpened(WindowEvent e)

Bisogna ridefinirli tutti.
Se si manda in esecuzione l'esempio si vede, grazie alle System.out.println messe nei metodi ridefiniti, a quale tipo di evento sulla finestra sono associati i vari metodi, come ad esempio windowClosed è invocato quando si preme la X della finestra.

Per esercizio provare a scrivere dei WindowListener per ascoltare l'evento chiudi Finestra per tutti i Frame definiti negli esempi di questo capitolo (li ho lasciati volutamente senza).

I menu visti non sono gli unici menu implementabili in Java, un altro tipo di menù è il menu di tipo Popup, ovvero quel menu che si associa ad un componente o ad una posizione. Un esempio di menu popup è il menu che viene fuori sul desktop del sistema operativo Windows quando si preme il bottone destro del mouse (Figura sotto).



Per creare un menu popup bisogna creare un oggetto appartenente alla classe PopupMenu, usando uno dei cosrtuttori: PopupMenu(), PopupMenu(String label), che creano rispettivamente un menu senza e con etichetta.

PopupMenu è una estensione di Menu, quindi ne eredita i metodi e anche quelli per aggiungere ed eliminare dei MenuItem. Tra i suoi metodi invece ce n'è uno per visualizzare il menu in una data posizione dello schermo rispetto ad un dato componente: show(Component origin, int x, int y)

LEZIONE 24: *Liste e scelte*

Spesso nelle interfacce dei nostri Applet ci capita di dover fare effettuare all'utente una o più scelte tra varie possibilità. Per fare questo dobbiamo elencare le possibilità e poi leggere la scelta.
Per fare questo possiamo usare le liste, che rapresentano una serie di oggetti e danno la possibilità di sceglierne alcuni. Pensiamo ad esempio ad un' applet che gestisce le prenotazioni di visite guidate ad alcune località, e che faccia scegliere tra tutte quelle che si vogliono visitare.
Oltre alle solite form per il Nome, Cognome ecc.. avremo la lista delle località, vediamo come possiamo implementarla. Per costruire la lista useremo uno dei tre costruttori:

List() , crea una nuova lista
List(int rows), crea una nuova lista con rows linee
List(int rows, boolean multipleMode), crea una nuova lista con rows linee, e le dice che a seconda del valore booleano multipleMode si potranno scegliere uno o più elementi della lista.

Nel nostro caso useremo la terza forma di costruttore.

List lista=new List(0,true);

Alcuni metodi che possiamo usare sulle liste sono i seguenti:

void add(String item) e void add(String item, int index), per appendere alla fine della lista, oppure in una data posizione un nuovo componente.

void addActionListener(ActionListener l), per associare un ActionListener alla lista.
void addItemListener(ItemListener l), per associare un ItemListener alla lista.
String getItem(int index), per avere l'elemento in posizione indicata.
int getItemCount() , per avere il numero di elementi.
String[] getItems(), per avere tutti gli elementi.
EventListener[] getListeners(Class listenerType) , per avere gli ascoltatori associati alla lista del tipo voluto.
Dimension getMinimumSize(), da la dimensione minima della lista.
Dimension getPreferredSize(), da la dimensione preferita per la lista.
int getRows() , dice il numero di linee attualmente visibili nella lista.
Int getSelectedIndex(), da l'indice dell'elemento selezionato.
int[] getSelectedIndexes(), da gli indici degli elementi selezionati.
String getSelectedItem(), da l'oggetto selezionato.
String[]getSelectedItems(), da gli oggetti selezionati.
Object[] getSelectedObjects(), da la lista degli elementi selezionati in un vettore di oggetti.
boolean isIndexSelected(int index), dice se l'indice è selezionato.
boolean isMultipleMode(), dice se nella lista si possono selezionare più elementi.
void makeVisible(int index), visualizza l'item in posizione indicata.
void remove(int position), void remove(String item), eliminano gli elementi indicati dall'indice o dall'etichetta.

void removeAll(), elimina tutti gli elementi della lista.
void removeActionListener(ActionListener l) e *void removeItemListener(ItemListener l)*, eliminano gli ascoltatori di eventi definiti per la lista.
void replaceItem(String newValue, int index), modifica l'elemento specificato.
void select(int index), seleziona l'elemento indicato nella lista.
void setMultipleMode(boolean b), dice alla lista se è possibile selezionare solo un'elemento o più di uno.

Vediamo l'esempio sull'uso delle liste.

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

public class liste extends Frame
{

List lista=new List(0,true);
Label text=new Label("Bellezze che possono essere visitate nella località scelta");

public liste()
{
super("Scelta itinerario");

lista.add("Benevento");
lista.add("Foiano di Val Fortore");
lista.add("Baselice");
lista.add("San Bartolomeo in Galdo");
lista.add("San Marco dei Cavoti");
lista.add("Montefalcone in Val Fortore");
lista.add("Pesco Sannita");
lista.add("Colle Sannita");
lista.add("Castelvetere in Val Fortore");
lista.add("Castelfranco in Miscano");
lista.add("Ginestra degli Schiavoni");
lista.add("San Giorgio la Molara");
lista.add("Molinara");
lista.add("Pietrelcina");
lista.add("Fragneto Monforte");
lista.add("Circello");
lista.add("Campolattaro");

add(lista,BorderLayout.CENTER);
add(text,BorderLayout.SOUTH);

addWindowListener(new listeWindowListener());
lista.addItemListener(new ascoltaLista());

setSize(350,100);

setResizable(false);

show();

}


public static void main(String [] arg)
{

new liste();

}


class listeWindowListener implements WindowListener
{
```

```
public void windowActivated(WindowEvent e)
{}

public void windowClosed(WindowEvent e)
{}

public void windowClosing(WindowEvent e)
{


String[] s=lista.getSelectedItems();
int i=0;
System.out.println("Itinerario selezionato");
try
{
while (true)
{

System.out.println(s[i++]);

}

}
catch (ArrayIndexOutOfBoundsException er)
{System.out.println("Buon divertimento...");}
System.exit(0);
}

public void windowDeactivated(WindowEvent e)
{}

public void windowDeiconified(WindowEvent e)
{}

public void windowIconified(WindowEvent e)
{}

public void windowOpened(WindowEvent e)
{}

}


class ascoltaLista implements ItemListener
{

public void itemStateChanged(ItemEvent e)
{

int indice=((Integer) e.getItem()).intValue();

if (indice==0) text.setText("Rocca dei Rettori, arco di Traiano, anfiteatro Romano, città spettacolo");
if (indice==1) text.setText("località San Giovanni, Campanile, via Roma, lago, festa S.Giovanni, festa
dell'emigrante");
if (indice==2) text.setText("oasi di San Leonardo");
if (indice==3) text.setText("centro storico");
if (indice==4) text.setText("centro storico");
if (indice==5) text.setText("centro storico");
if (indice==6) text.setText("centro storico");
if (indice==7) text.setText("centro storico");
if (indice==8) text.setText("centro storico");
if (indice==9) text.setText("Bosco");
if (indice==10) text.setText("centro storico");
if (indice==11) text.setText("Lago di San Giorgio");
```

```
if (indice==12) text.setText("centro storico");
if (indice==13) text.setText("Piana Romana, centro storico, case di Padre Pio");
if (indice==14) text.setText("Raduno internazionale di mongolfiere, palazzo Ducale");
if (indice==15) text.setText("centro storico");
if (indice==16) text.setText("Diga di Campolattaro");

}

}

}
```

Le liste sono quindi un buon GUI per fare effettuare all'utente una scelta fra varie possibilità.
Vi è un altro GUI che serve a fare questo, che in pratica è una lista a scomparsa, nel senso che è visibile solo l'oggetto selezionato, e che si vedono gli altri solo quando si deve fare la scelta, tipo menù a scomparsa, mentre nelle liste sono sempre visibili.
Il GUI di cui sto parlando è il Choice (scelta), il cui funzionamento è simile a quello delle liste.
Il costruttore è uno, senza parametri, mentre i metodi sono:

void add(String item), aggiunge una possibilità in fondo alle scelte possibili.
void addItem(String item), come la add
void addItemListener(ItemListener l), associa un ascoltatore di eventi per la Choice.
String getItem(int index), restituisce la scelta possibile che si trova all'indice i.
int getItemCount(), da' il numero di possibilità per la Choice.
EventListener[] getListeners(Class listenerType), da tutti gli ascoltatori associati alla Choice.
int getSelectedIndex(), restituisce l'indice della scelta selezionata nella Choice.
String getSelectedItem(), restituisce la scelta selezionata nella Choice.
Object[] getSelectedObjects(), da tutti gli item selezionati.
void insert(String item, int index), inserisce una scelta nella posizione indicata della Choice.

void remove(int position), void remove(String item), void removeAll() , per rimuovere le scelte possibili dalla lista di scelte.

void removeItemListener(ItemListener l), elimina l'ascoltatore di eventi di tipo ItemListener associato alla Choice.

void select(int pos), void select(String str), per selezionare una scelta.

Facciamo un programmino che usi le Choice, però visto che finora abbiamo creato delle applicazioni a finestre, creiamo un'applet, visto che e GUI, come già detto, possono essere usate in ambedue i tipi di programmi.
Parlando di applet, visto che non lo abbiamo fatto finora, creiamo un applet che legge i parametri passatigli dal file html che lo invoca, come fanno in pratica la maggior parte degli applet presenti sul sito html.it.
Nell'applet, a titolo di esempio ho messo un piccolo controllo sul campo autore, questo solo per farvi vedere come si può fare. La cosa può risultare utile se si crea un'applet e la si distribuisce, in modo da lasciare obbligatorio il parametro autore e il suo valore, per evitare che altri possano spacciarsi per il ceratore dell'applet, e quindi per affermare i diritti di cerazione dell'applet.

Il controllo non è sofisticato, un hacker lo eviterebbe facilmente, però un hacker non ha interesse di crackare un programma free, solo per eliminare i diritti d'autore, mentre per un utente normale è impossibile eseguire il programma senza specificare il nome dell'autore. Quel nome può anche essere ad esempio un numero di serie, se si vuole vendere un programma.

Il file html, si chiamerà Choice.html, ed il suo contenuto sarà:

```
<html>
<head>
<title>
Esempio di uso della java.awt.Choice e del passaggio di parametri ad un applet
</title>
</head>
<body>
<APPLET code="Scelte.class" width=350 height=100>
<param name=loc1 value="Benevento">
<param name=loc2 value="Foiano di Val Fortore">
```

```
<param name=loc3 value="Baselice">
<param name=loc4 value="San Bartolomeo in Galdo">
<param name=loc5 value="San Marco dei Cavoti">
<param name=loc6 value="Montefalcone in Val Fortore">
<param name=loc7 value="Pesco Sannita">
<param name=loc8 value="Colle Sannita">
<param name=loc9 value="Castelvetere in Val Fortore">
<param name=loc10 value="Castelfranco in Miscano">
<param name=loc11 value="Ginestra degli Schiavoni">
<param name=loc12 value="San Giorgio la Molara">
<param name=loc13 value="Molinara">
<param name=loc14 value="Fragneto Monforte">
<param name=loc15 value="Circello">
<param name=loc16 value="Campolattaro">
<param name=autore value="Pietro Castellucci">

</APPLET>
</body>
</html>
```

Mentre il codice dell'applet, editato nel file chiamato Scelte.java, è:

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

public class Scelte extends Applet
{


Label text=new Label("Bellezze che possono essere visitate nella località scelta");
Choice scelte=new Choice();
public void init()
{

// Controllo diritti di autore:
String autore=getParameter("autore");
if (autore==null) System.exit(0);
if (autore.compareTo("Pietro Castellucci")!=0)
{
System.out.println("Parametro autore non valido, inserire come valore Pietro Castellucci");
System.exit(0);
}


String p=getParameter("loc1");
if (p==null) p="Località 1 non definita";
scelte.add(p);

p=getParameter("loc2");
if (p==null) p="Località 2 non definita";
scelte.add(p);

p=getParameter("loc3");
if (p==null) p="Località 3 non definita";
scelte.add(p);

scelte.add(p);

p=getParameter("loc4");
if (p==null) p="Località 4 non definita";
scelte.add(p);

p=getParameter("loc5");
if (p==null) p="Località 5 non definita";
```

```
scelte.add(p);

p=getParameter("loc6");
if (p==null) p="Località 6 non definita";
scelte.add(p);

p=getParameter("loc7");
if (p==null) p="Località 7 non definita";
scelte.add(p);

p=getParameter("loc8");
if (p==null) p="Località 8 non definita";
scelte.add(p);

p=getParameter("loc9");
if (p==null) p="Località 9 non definita";
scelte.add(p);

p=getParameter("loc10");
if (p==null) p="Località 10 non definita";
scelte.add(p);

p=getParameter("loc11");
if (p==null) p="Località 11 non definita";
scelte.add(p);

p=getParameter("loc12");
if (p==null) p="Località 12 non definita";
scelte.add(p);

p=getParameter("loc13");
if (p==null) p="Località 13 non definita";
scelte.add(p);

p=getParameter("loc14");
if (p==null) p="Località 14 non definita";
scelte.add(p);

p=getParameter("loc15");
if (p==null) p="Località 15 non definita";
scelte.add(p);

p=getParameter("loc16");
if (p==null) p="Località 16 non definita";
scelte.add(p);

p=getParameter("loc17");
if (p==null) p="Località 17 non definita";
scelte.add(p);

scelte.addItemListener(new ascoltaChoice());


// Visualizzo la Choice

add(scelte, BorderLayout.CENTER);
add(text, BorderLayout.SOUTH);

}

class ascoltaChoice implements ItemListener
{

public void itemStateChanged(ItemEvent e)
{
```

```
int indice=scelte.getSelectedIndex();

if (indice==0) text.setText("Rocca dei Rettori, arco di Traiano, anfiteatro Romano");
if (indice==1) text.setText("località San Giovanni, Campanile, via Roma, lago");
if (indice==2) text.setText("oasi di San Leonardo");
if (indice==3) text.setText("centro storico");
if (indice==4) text.setText("centro storico");
if (indice==5) text.setText("");
if (indice==6) text.setText("");
if (indice==7) text.setText("");
if (indice==8) text.setText("");
if (indice==9) text.setText("Bosco");
if (indice==10) text.setText("");
if (indice==11) text.setText("Lago di San Giorgio");
if (indice==12) text.setText("");
if (indice==13) text.setText("Piana Romana, centro storico, case di Padre Pio");
if (indice==14) text.setText("Raduno internazionale di mongolfiere, palazzo Ducale");
if (indice==15) text.setText("");
if (indice==16) text.setText("Diga di Campolattaro");
}
}
}
```

L'ultimo componente GUI che vediamo in questa lezione è lo ScrollPane, ovvero un pannello in cui è possibile inserire un componente grande a piacere, che deve essere visualizzato a pezzi.
Lo ScrollPane usa due Scrollbar, una orizzontale ed una verticale, per selezionare la parte del GUI che si vuole vedere. Da notare che la Scrollbar è un oggetto GUI anch'esso, quindi può essere inserito e gestito nelle proprie applicazioni. Nello ScrollPane si può scegliere se visualizzare le Scrollbar sempre, mai o solo se è necessario.
Uno ScrollPane tipico è quello che si usa per visualizzare un file di testo.
Ho due tipi di costruttori per l'oggetto:

ScrollPane(int sbp), crea uno ScrollPane con politica di visualizzazione lelle Scrollbar definita da spb.

I valori possibili sono :
ScrollPane.SCROLLBARS_ALWAYS sempre in vista
ScrollPane.SCROLLBARS_AS_NEEDED in vista solo se ce n'è bisogno
ScrollPane.SCROLLBARS_NEVER mai in vista

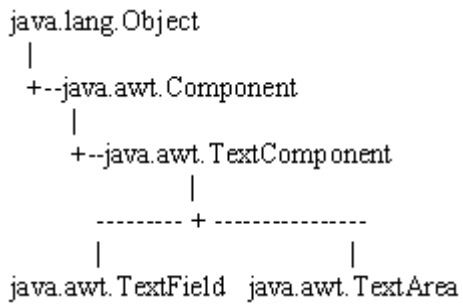
ScrollPane(), crea uno ScrollPane con politica ScrollPane.SCROLLBARS_AS_NEEDED.

Tra i metodi, oltre a quelli per gestire lo ScrollPane troviamo quelli di Container e di Component, infatti ScrollPane è una estensione di Container, quindi può essere visto come un contenitore può essere visto come un componente.

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.ScrollPane
```

LEZIONE 25: *Il Testo, gli eventi, Dialog*

Ci rimane da vedere un componente fondamentale per un interfaccia, ovvero il Testo.
Le abstract window Toolkit di Java mettono a disposizione due classi per il testo: TextArea e TextField, ambedue estendono la classe TextComponent.



In TextComponent troviamo una serie di metodi utili per la gestione del testo, sia in TextArea che in TextField, tra cui:

Color getBackground() e setBackground(Color c), per prendere e settare il colore di Background del testo.
String getSelectedText(), prende il testo selezionato.
int getSelectionEnd(), da la posizione della fine selezione del testo.
Int getSelectionStart(), da la posizione dell'inizio selezione del testo.
String getText(), da il testo del TextComponent
boolean isEditable(), dice se il testo del componente è editabile.
void select(int selectionStart, int selectionEnd), seleziona il testo da selectionStart a selectionEnd.
void selectAll(), seleziona tutto il testo.
void setEditable(boolean b), setta se il testo è editabile o meno.
void setSelectionEnd(int selectionEnd), setta la fine della selezione del testo.
void setSelectionStart(int selectionStart), setta l'inizio della selezione.
void setText(String t), setta il contenuto del componente di testo con il valore t.

Inoltre in TextComponent si possono usare tutti i metodi di Component, ovvero:

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addNotify, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, deliverEvent, disable, disableEvents, dispatchEvent, doLayout, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getAlignmentX, getAlignmentY, getBounds, getBounds, getColorModel, getComponentAt, getComponentAt, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getGraphicsConfiguration, getHeight, getInputContext, getInputMethodRequests, getLocale, getLocation, getLocation, getLocationOnScreen, getMaximumSize, getMinimumSize, getName, getParent, getPeer, getPreferredSize, getSize, getSize, getToolkit, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, invalidate, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isShowing, isValid, isVisible, keyDown, keyUp, layout, list, list, list, list, locate, location, lostFocus, minimumSize, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paint, paintAll, postEvent, preferredSize, prepareImage, prepareImage, print, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setFont, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, show, size, toString, transferFocus, update, validate

Vediamo quindi i TextField. Un TextField è in pratica una singola linea di testo editabile, con un aspetto grafico, è uno di quei campi che siamo abituati a vedere nelle Form che si trovano su internet per immettere nomi, cognomi, ecc.

Ho quattro possibili costruttori:

TextField(), crea una TextField vuota.
TextField(int c), crea una TextField vuota di c colonne, ovvero una linea di c caratteri.
TextField(String t), crea una TextField inizializzata con t.
TextField(String t, int c), crea una TextField inizializzata con t, di c caratteri.

Tra i vari metodi, quelli più interessanti per i nostri scopi, sono:

boolean echoCharIsSet(), dice se la TextField fa l'echo dei caratteri inseriti.
int getColumns(), da il numero di colonne della TextField.
char getEchoChar(), da il carattere usato per l'echo.
void setColumns(int columns), setta il numero di colonne della TextField.
void setEchoChar(char c), setta il carattere di echo per la TextField.
void setText(String t), per settare il testo contenuto.

Per le dimensioni:

Dimension getMinimumSize()
Dimension getMinimumSize(int c)
Dimension getPreferredSize()
Dimension getPreferredSize(int c)

La TextArea invece è un vero e proprio testo, non solo una riga, ma tante, comprende anche le due scrollbar per navigare sul testo se questo è più grande della finestra dove è visualizzato.

I costruttori sono:

TextArea(), costruisce una nuova TextArea.
TextArea(int r, int c) , costruisce una nuova TextArea, di r righe e c colonne.
TextArea(String text), costruisce una nuova TextArea inizializzata con il testo passato.
TextArea(String text, int rows, int columns) , costruisce una nuova TextArea inizializzata con il testo passato, di r righe e c colonne.
TextArea(String text, int rows, int columns, int scrollbars), come il precedente, solo che permette di specificare la politica di visualizzazione delle scrollbar.

Vediamone alcuni metodi:

void append(String str), appende il testo passato alla TextArea. (nelle vecchie versioni di Java è appendText (String str)).
int getColumns(), int getRows() per avere informazioni sul numero di colonne o righe.
void insert(String str, int pos), inserisce il testo passato nella posizione passata. (nelle vecchie versioni di Java è insertText(String str)).
void replaceRange(String str, int start, int end), rimpiazza il testo da start a end con la stringa passata.
void setColumns(int columns) e *void setRows(int rows)* per settare il numero di colonne o righe della TextArea.

Per le dimensioni dell'oggetto GUI:

Dimension getMinimumSize()
Dimension getMinimumSize(int rows, int columns)
Dimension getPreferredSize()
Dimension getPreferredSize(int rows, int columns)

Abbiamo visto tutti i componenti di testo, a questo punto possiamo fare un'esempio che le usa ambedue.

Creiamo un'applet, quindi il launcher è:

```
<html>
<head>
<title>
Esempio di uso dei componenti di testo.
</title>
</head>
<body>
<APPLET code="Testo.class" width=500 height=200>
<param name=testo value="Testo iniziale per la TextArea.Inserire altri caratteri.">
</APPLET>
</body>
</html>
```

mentre il codice, da editare in Testo.java, è il seguente:

```
import java.awt.event.KeyListener;

import java.awt.event.KeyEvent;

import java.awt.TextField;

import java.awt.TextArea;

import java.awt.BorderLayout;

import java.applet.*;


public class Testo extends Applet
{

TextField TF=new TextField(20);

TextArea TA=new TextArea();

public void init()
{

add(TA,BorderLayout.CENTER);

add(TF,BorderLayout.SOUTH);

String t=getParameter("testo");

if (t!=null) TA.setText(t);

TF.setEditable(false);

TA.addKeyListener(new MyKeyListener());

}


public class MyKeyListener implements KeyListener
{
public void keyPressed(KeyEvent e)
{

}

public void keyReleased(KeyEvent e)
{

int valore=e.getKeyCode();
if (valore!=10) TF.setText("Inserito carattere "+e.getKeyChar());
else TF.setText("Inserito carattere INVIO");

}

public void keyTyped(KeyEvent e)
{

}

}

}
```

LEZIONE 26: *La gestione degli eventi in Java2*

Abbiamo visto quesi tutti i componenti GUI di awt, per costruire una interfaccia basta combinare queste e ascoltarne gli eventi.
Per ogni GUI vista abbiamo visto come gestire alcuni eventi, vediamo adesso quali altri eventi possono essere ascoltati.
Innanzitutto abbiamo ascoltato degli eventi di tipo ActionEvent, ovvero eventi che consistono nel compiere un'azione sulle gui da parte dell'utente, tipo premere un bottone.
Per fare questo, ricordo che abbiamo implementato l'interfaccia ActionListener, e ne abbiamo ridefinito l'unico evento actionPerformed (ActionEvent e), che viene invocato quando viene esequita una azione sul componente GUI (o sui componenti) a cui è associato.
Per associare al GUI l'ascoltatore di eventi che abbiamo definito abbiamousato il metodo addActionListener (ActionListener ascoltatore) del GUI.
Pe usarli bisogna includere nel programma:

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

oppure

impore java.awt.event.;*

Ascoltatori di eventi di tipo ActionListener possono essere associati agli oggetti GUI di awt:

Button
List (quando si seleziona o deseleziona un elemento)
MenuItem
TextField

Abbiamo visto anche come implementare un ItemListener, per ascoltare eventi provenienti da GUI che hanno uno stato di on/off, quando cambiano il proprio stato.
Per associarlo ad un GUI abbiamo usato addItemListener(ItemListener ascoltatore) del GUI, quando l'evento è generato dal GUI, viene invocato il metodo itemStateChanged(ItemEvent e) dell'ascoltatore associato.

Da includere:

import java.awt.event.;*

oppure

import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;

I GUI awt per cui può essere ascoltato l'evento sono:

Checkbox
CheckboxMenuItem (non lo abbiamo visto, esso è come un MenuItem, solo che funziona come un Checkbox, ovvero può essere cliccato o meno, se associato ad un CheckboxGroup, diviene ancora una volta un radiobutton, solo che su un menu).
Choice
List

Abbiamo anche visto gli ascoltatori degli eventi delle finestre, ovvero di tipo WindowListener.
Essi vengono associati alle finestre usando il metodo di queste addWindowListener(WindowListener WL), e per definirle bisogna implementare l'interfaccia WindowListener e ridefinirne i metodi:

void windowActivated(WindowEvent e), invocato quando la finestra diviene attiva, ovvero viene in primo piano sul desktop.
void windowClosed(WindowEvent e), invocato quando la finestra si è chiusa.
void windowClosing(WindowEvent e), invocato quando si preme il bottone chiudiFinestra (X).
void windowDeactivated(WindowEvent e), invocato quando la finestra passa in secondo piano.
void windowDeiconified(WindowEvent e), invocato quando la finestra passa da icona ad attiva.
void windowIconified(WindowEvent e), quando viene ridotta ad icona.
void windowOpened(WindowEvent e), quando viene aperta.

Per sentire gli eventi della finestra bisogna includere:

import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;

oppure

import java.awt.event.;*

Gli eventi delle finestre avvengono sempre più o meno a coppie, ad esempio quando si preme la X in alto a destra della finestre, prima viene invocato windowClosing e poi windowClosed, oppure quando una finestra passa da icona ad attiva prima sente windowDeiconified e poi WindowActivated, eccetera.

E' possibile sentire eventi di tipo finestra per gli elementi:

Window
Frame
Dialog

Sulle Dialog apro una piccola parentesi:
Sono dei Frame senza icona , riduci a icona e massimizza, sonomolto utili per fare appunto dei dialoghi veloci con l'utente, sono ad esempio delle dialog le finestre che appaiono nelle applicazioni e dicono Sei sicuro di volere uscire?
Una Dialog può essere modale o non modale, la prima è una Dialog che blocca completamente l'applicazione, la quale non sente più gli eventi, la seconda invace va in parallelo a questa.
Le dialog di awt non possono contenere dei menu, mentre quelle di swing (javax.swing.JDialog) si.

Vediamo quali eventi possono essere ascoltati da oggetti di tipo TextComponent, ovvero oggetti di tipo TextField e TextArea. Possiamo ascoltare eventi di tipo TextEvent, implementando l'interfaccia TextListener, e ridefinendone il metodo textValueChanged(TextEvent e), invocato ogni volta che il testo viene modificato. L'interfaccia e l'evento si trovano in java.awt.event

Per oggetti di tipo container, ovvero per Panel, ScrollPane, Window (e quindi Dialog e Frame), posso ascoltare gli eventi sul contenitore, usando il metodo addContainerListener(ContainerListener l). Bisogna implementare l'interfaccia ContainerListener e ridefinire i metodi:

void componentAdded(ContainerEvent e)
void componentRemoved(ContainerEvent e)

che vengono automaticamente invocati quando vengono aggiunti o tolti elementi al contenitore.
Bisogna includere java.awt.event.ContainerListener e java.awt.event.ContainerEvent.

A questo punto vediamo quali eventi si possono ascoltare su oggetti di tipo Component, ovvero su TUTTI i componenti awt, compresi i contenitori e le finestre.

In Component troviamo i metodi:

addComponentListener(ComponentListener l)
addFocusListener(FocusListener l)
addHierarchyBoundsListener(HierarchyBoundsListener l)
addHierarchyListener(HierarchyListener l)
addInputMethodListener(InputMethodListener l)
addKeyListener(KeyListener l)
addMouseListener(MouseListener l)
addMouseMotionListener(MouseMotionListener l)
addPropertyChangeListener(PropertyChangeListener listener)
addPropertyChangeListener(String propertyName, PropertyChangeListener listener)

Il ComponentListener ascolta eventi sul componente, si devono ridefinire i metodi:

void componentHidden(ComponentEvent e)
void componentMoved(ComponentEvent e)
void componentResized(ComponentEvent e)
void componentShown(ComponentEvent e)

che vengono invocati quando il componente viene nascosto, mosso, cambiato di dimensioni e visualizzato.

FocusListener ascolta eventi di focus della tastiera sul componente, ovvero se si seleziona o meno il componente con la tastiera.
Bisogna ridefinirne i metodi:

focusGained(FocusEvent e), chiamato quando il componente è selezionato.
focusLost(FocusEvent e), chiamato dal componente quando perde la selezione della tastiera.

Si pensi ad esempio alla navigazione tra varie form di una interfaccia premendo il tasto TAB.

HierarchyBoundsListener, sente eventi di spostamento degli antenati dal componente, ad esempio un bottone su una finestra, se implementa questa interfaccia sente se la finestra si muove.
Bisogna ridefinire i metodi:

void ancestorMoved(HierarchyEvent e), chiamata quando l'antenato viene mosso.
void ancestorResized(HierarchyEvent e) , chiamata quando l'antenato viene ambiato di dimensione.

HierarchyListener ascolta i cambiamenti degli antenati del componente, bisogna ridefinire il metodo hierarchyChanged(HierarchyEvent e), sarà HierarchyEvent a contenere il cambiamento fatto.

KeyListener lo abbiamo gia visto, ascolta eventi provenienti dalla tastiera, dobbiamo ridefinire i metodi
void keyPressed(KeyEvent e), invocato quando un tasto viene premuto.
void keyReleased(KeyEvent e), invocato quando un tasto viene lasciato.
void keyTyped(KeyEvent e), invocato quando un tasto si tiene premuto.
In pratica quando si preme un tasto vengono sentiti gli eventi nell'ordine: keyPressed, keyTyped e keyReleased.

MouseListener ascolta gli eventi provenienti dal mouse, bisogna ridefinire i metodi:
void mouseClicked(MouseEvent e), invocato quando viene cliccato un bottone del muose sul componente.
void mouseEntered(MouseEvent e), invocato quando il mouse entra nel componente.
void mouseExited(MouseEvent e), invocato quando il mouse esce dal componente.
void mousePressed(MouseEvent e), invocato quando il mouse si tiene premuto sul componente.
void mouseReleased(MouseEvent e), invocato quando il mouse viene rilasciato.

MouseMotionListener, altri eventi del mouse sul componente:
mouseDragged(MouseEvent e), quando il mouse viene premuto e mosso sul componente.
void mouseMoved(MouseEvent e), invocato quando il mouse si muove sul componente, con o senza bottoni premuti.

L'ultima interfaccia per eventi del mouse è MouseInputListener, che implementa tutte e due le interfacce precedenti, quindi ascolta tutti gli eventi precedenti del mouse.

Nell'esempio successivo vengono ascoltati vari eventi, viene anche usata una Dialog.

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

class Ascolta extends Frame
{

Button b1=new Button("Bottone1");

Button b2=new Button("Visualizza Dialog");

Button b3=new Button("Esci");

Button b4=new Button("Chiudi Dialog");

Choice l1=new Choice();

TextField TF1=new TextField(15);

Dialog D;
```

```
AscoltaActionListener aal;
AscoltaItemListener ail;
AscoltaWindowListener awl;
AscoltaWindowListenerDialog awld;
AscoltaMouseListener aml;
AscoltaKeyListener akl;
public Ascolta()
{
setTitle("Frame, esempio ascoltatori eventi");

setLocation(100,100);

setLayout(new GridLayout(5,1));
add(b1);
add(b2);
add(b3);
add(l1);
add(TF1);

D=new Dialog(this,true);
D.setSize(200,100);
D.setLocation(200,100);
D.add(b4,BorderLayout.SOUTH);
D.setTitle("Dialog - modale");
preparaD(false);

l1.add("Scelta 1");
l1.add("Scelta 2");
l1.add("Scelta 3");

// Inizializzo gli ascoltatori degli eventi.
aal=new AscoltaActionListener();
ail=new AscoltaItemListener();
awl=new AscoltaWindowListener();
awld=new AscoltaWindowListenerDialog();
aml=new AscoltaMouseListener();
akl=new AscoltaKeyListener();
b1.addActionListener(aal);
b2.addActionListener(aal);
b3.addActionListener(aal);
TF1.addActionListener(aal);
l1.addItemListener(ail);
addWindowListener(awl);
addMouseListener(aml);
b1.addMouseListener(aml);
b2.addMouseListener(aml);
b3.addMouseListener(aml);
b4.addMouseListener(aml);
l1.addMouseListener(aml);
TF1.addMouseListener(aml);
TF1.addTextListener(new TextListener()
{
public void textValueChanged(TextEvent e)
{
System.out.println("TextListener: cambiato valore della TextField");
}
}
);
D.addMouseListener(aml);
addKeyListener(akl);
b1.addKeyListener(akl);
b2.addKeyListener(akl);
b3.addKeyListener(akl);
b4.addKeyListener(akl);
l1.addKeyListener(akl);
```

```
TF1.addKeyListener(akl);
D.addKeyListener(akl);
pack();
show();
}

void preparaD(boolean m)
{
D.setModal(m);
}

public static void main (String [] s)
{
System.out.println("Ascolto gli eventi.");
new Ascolta();

}

// ActionListener

class AscoltaActionListener implements ActionListener
{
public void actionPerformed(ActionEvent e)
{

String c=e.getActionCommand();
if (c.compareTo("Bottone1")==0) System.out.println("ActionListener: Premuto Bottone1");
else
if (c.compareTo("Esci")==0)
{
System.out.println("ActionListener: Premuto Esci");
System.exit(9); // è un numero qualsiasi, si chiama exit code.
} else
if (c.compareTo("Visualizza Dialog")==0)
{
System.out.println("ActionListener: Premuto Visualizza Dialog");
D.show();
b4.addActionListener(aal);
b2.removeActionListener(aal);
D.addWindowListener(awld);
} else
if (c.compareTo("Chiudi Dialog")==0)
{
System.out.println("ActionListener: Premuto Visualizza Dialog");
D.hide();
b4.removeActionListener(aal);
b2.addActionListener(aal);
D.removeWindowListener(awld);
} else
{
System.out.println("ActionListener: Premuto Invio nella TextField, letto:"+TF1.getText());
TF1.setText("");
}

}
}

// ItemListener
class AscoltaItemListener implements ItemListener
{

public void itemStateChanged(ItemEvent e)
{
String val=l1.getSelectedItem();
System.out.println("ItemListener: Selezionato "+val);
```

```
}

}

// WindowListener

public class AscoltaWindowListener implements WindowListener
{
public void windowActivated(WindowEvent e)
{
System.out.println("WindowListener: Finestra attivata");
}

public void windowClosed(WindowEvent e)
{
System.out.println("WindowListener: Finestra chiusa");
}

public void windowClosing(WindowEvent e)
{
System.out.println("WindowListener: Premuto chiudi finestra");
System.exit(0);
}

public void windowDeactivated(WindowEvent e)
{
System.out.println("WindowListener: Finestra disattivata");
}

public void windowDeiconified(WindowEvent e)
{
System.out.println("WindowListener: Finestra deiconificata");
}

public void windowIconified(WindowEvent e)
{
System.out.println("WindowListener: Finestra riduci a icona");
}

public void windowOpened(WindowEvent e)
{
System.out.println("WindowListener: Finestra aperta");
}
}


public class AscoltaWindowListenerDialog implements WindowListener
{
public void windowActivated(WindowEvent e)
{
System.out.println("WindowListener: Dialog attivata");
}

public void windowClosed(WindowEvent e)
{
System.out.println("WindowListener: Dialog chiusa");
}

public void windowClosing(WindowEvent e)
{
System.out.println("WindowListener: Premuto chiudi Dialog");
D.hide();
b4.removeActionListener(aal);
b2.addActionListener(aal);
D.removeWindowListener(awld);
```

```
}

public void windowDeactivated(WindowEvent e)
{
System.out.println("WindowListener: Finestra disattivata");
}

public void windowDeiconified(WindowEvent e)
{
System.out.println("WindowListener: Dialog deiconificata");
}

public void windowIconified(WindowEvent e)
{
System.out.println("WindowListener: Dialog riduci a icona");
}

public void windowOpened(WindowEvent e)
{
System.out.println("WindowListener: Dialog aperta");
}
}

// Mouse Listener:
class AscoltaMouseListener implements MouseListener
{

public void mouseClicked(MouseEvent e)
{
Component a=e.getComponent();
String n=a.getName();
System.out.println("MouseListener: Mouse cliccato sul componente "+n);
}
public void mouseEntered(MouseEvent e)
{
Component a=e.getComponent();
String n=a.getName();
System.out.println("MouseListener: Mouse entrato nel componente "+n);
}
public void mouseExited(MouseEvent e)
{
Component a=e.getComponent();
String n=a.getName();
System.out.println("MouseListener: Mouse uscito dal componente "+n);
}
public void mousePressed(MouseEvent e)
{
Component a=e.getComponent();
String n=a.getName();
System.out.println("MouseListener: il bottone e' schiacciato su coponente "+n);
}
public void mouseReleased(MouseEvent e)
{
Component a=e.getComponent();
String n=a.getName();
System.out.println("MouseListener: Mouse rilasciato sul componente "+n);
}
}

// Tastiera
class AscoltaKeyListener implements KeyListener
{
public void keyPressed(KeyEvent e)
{
System.out.println("KeyListener: Premuto tasto "+e.getKeyChar()+
```

```
" codice "+
e.getKeyCode());
}
public void keyReleased(KeyEvent e)
{
System.out.println("KeyListener: Rilasciato tasto "+e.getKeyChar()+
" codice "+
e.getKeyCode());
}
public void keyTyped(KeyEvent e)
{
System.out.println("KeyListener: Stai premendo il tasto "+e.getKeyChar()+
" codice "+
e.getKeyCode());
}
}
}
```

LEZIONE 27: *Introduzione a swing*

Oltre al package java.awt, Java mette a disposizione del programmatore il package javax.swing per creare delle interfacce grafiche. In questo capitolo vedremo perché la Sun Microsystem ha creato quest'altro package e quali sono le differenze con awt.
Per presentare il package parlerò del programma che sto facendo per la mia Tesi in Informatica all'Università di Pisa. Per informazioni riguardo ad Orespics potete contattare me, oppure le Prof.sse Laganà e Ricci presso il dipartimento di Informatica di Pisa (Corso Italia, 40).
Il nome del programma è Orespics Programming Language, ed in pratica è un ambiente integrato in cui è possibile programmare agenti paralleli che comunicano tra di loro scambiandosi dei messaggi. L'ambiente permette di definire ed eseguire questi agenti, esso si propone come strumento didattico di supporto all'apprendimento delle modalità di programmazione parallela, programmazione che al programmatore abituato a programmare in modo sequenziale può risultare molto difficoltosa.
Proprio perché lo strumento è uno strumento didattico, ha una interfaccia utente User Friendly. L'interfaccia è stata scritta usando il package Swing di Java.
Swing è stato interamente scrittto in Java, usando il package awt, e mette a disposizione all'utente tante classi presenti anche in awt, ma notevolmente migliorate e potenziate, mette inoltre tante altre classi non presenti in awt.
Vediamo quindi quali sono le classi e le interfacce contenute nel pacchetto:

Interfacce
Action
BoundedRangeModel
ButtonModel
CellEditor
ComboBoxEditor
ComboBoxModel
DesktopManager
Icon
JComboBox.KeySelectionManager
ListCellRenderer
ListModel
ListSelectionModel
MenuElement
MutableComboBoxModel
Renderer
RootPaneContainer
Scrollable
ScrollPaneConstants
SingleSelectionModel
SwingConstants
UIDefaults.ActiveValue
UIDefaults.LazyValue
WindowConstants

Classi
AbstractAction

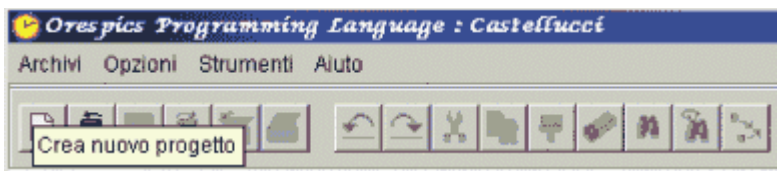
AbstractButton
AbstractCellEditor
AbstractListModel
ActionMap
BorderFactory
Box
Box.Filler
BoxLayout
ButtonGroup
CellRendererPane
ComponentInputMap
DebugGraphics
DefaultBoundedRangeModel
DefaultButtonModel
DefaultCellEditor
DefaultComboBoxModel
DefaultDesktopManager
DefaultFocusManager
DefaultListCellRenderer
DefaultListCellRenderer.UIResource
DefaultListModel
DefaultListSelectionModel
DefaultSingleSelectionModel
FocusManager
GrayFilter
ImageIcon
InputMap
InputVerifier
JApplet
JButton
JCheckBox
JCheckBoxMenuItem
JColorChooser
JComboBox
JComponent
JDesktopPane
JDialog
JEditorPane
JFileChooser
JFrame
JInternalFrame
JInternalFrame.JDesktopIcon
JLabel
JLayeredPane
JList
JMenu
JMenuBar
JMenuItem
JOptionPane
JPanel
JPasswordField
JPopupMenu
JPopupMenu.Separator
JProgressBar
JRadioButton
JRadioButtonMenuItem
JRootPane
JScrollBar
JScrollPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JTable
JTextArea

JTextField
JTextPane
JToggleButton
JToggleButton.ToggleButtonModel
JToolBar
JToolBar.Separator
JToolTip
JTree
JTree.DynamicUtilTreeNode
JTree.EmptySelectionModel
JViewport
JWindow
KeyStroke
LookAndFeel
MenuSelectionManager
OverlayLayout
ProgressMonitor
ProgressMonitorInputStream
RepaintManager
ScrollPaneLayout
ScrollPaneLayout.UIResource
SizeRequirements
SizeSequence
SwingUtilities
Timer
ToolTipManager
UIDefaults
UIDefaults.LazyInputMap
UIDefaults.ProxyLazyValue
UIManager
UIManager.LookAndFeelInfo
ViewportLayout

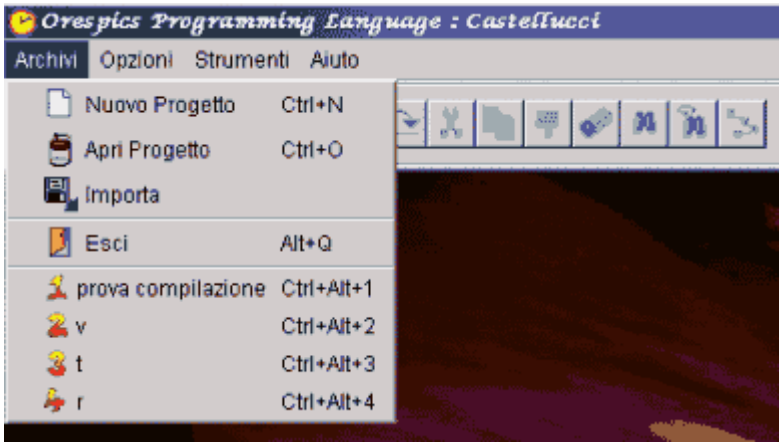
Eccezioni
UnsupportedLookAndFeelException

Notiamo in numero di classi molto superiore di quelle di awt, inoltre notiamo che ci sono tante classi che hanno lo stesso nome di alcune classi di awt, tranne che per una J iniziale, come ad esempio JFrame, JDialog, eccetera, queste classi hanno lo stesso funzionamento delle analoghe classi di awt, tranne che per il fatto che contengono tanti metodi utili in più.
Presentiamo alcune delle cose in più in swing rispetto ad awt.
In swing ci sono le Toolbar, ovvero delle piccole barre su cui ci vanno dei bottoni, esse possono essere spostate all'interno della finestra che le contiene.
I bottoni di swing, come quasi tutti gli altri JComponent hanno la possibilità olre che di avere un'etichetta anche una immagine, immagine che può cambiare a seconda dello stato del bottone.
Inoltre in swing possono essere usati i Tooltip, ovvero quei suggerimenti che escono automaticamente su un componente quando il mouse vi ci si sofferma per un po di tempo.
Vediamo l'effetto di questi componenti swing in Orespics.

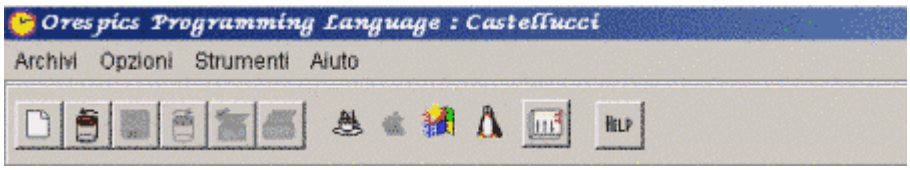
Nella Figura possiamo notare la Toolbar (in cui ho nascosto i bordi) di altre 8 Toolbar, una per ogni categoria di bottoni. Si vedono i bottoni che contengono delle immagini, alcune attive e altre inattive.



Nella Figura si può vedere una Tooltip su uno dei bottoni.

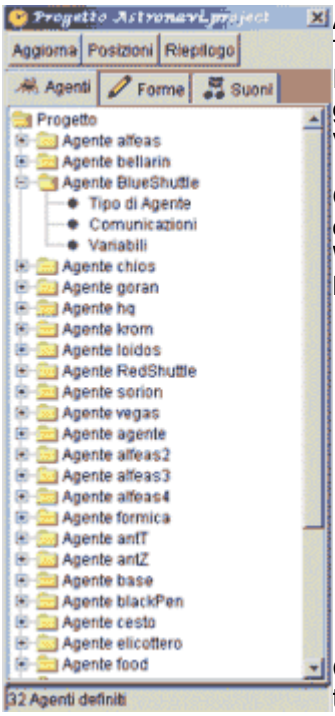


Nella Figura possiamo notare che anche nei menu possono essere inserite delle immagini, rendendoli molto gradevoli all'utente.



Le toolbar possono essere facilmente nascoste e visualizzate sullo schermo.

Nella Figura notiamo il cambiamento dell'immagine del bottone con il pinguino, che diventa più grande, con swing, come già detto è possibile cambiare l'icona a seconda dello stato del bottone.



Altri due componenti molto belli presenti in Swing e non in awt sono i TabbedPane e gli Alberi.
I primi sono i controlli a schede tipici delle interfacce di Windows, molto gradevoli, e i secondi fanno vedere delle strutture ad albero, tipo qelle che si vedono navigando sugli hard disk usando esplora di Windows.

Con swing si può anche cambiare il cosiddetto Look and Feel della finestra, ovvero il suo aspetto, rendendola simile ad una interfaccia tipica di Java, di Windows, di Macintosh (se si è su una macchina Mac) e di Linux (X-Windows - Motif).

Con Swing cambia completamente la gestione del testo, infatti si possono creare testi colorati, con vari stili di carattere.

Tante altre sono le possibilità di Swing, come ad esempio delle Dialog per navigare sui file con delle preview e delle dialog per la scelta dei colori, già implementate dal pacchetto.

Molto interessante è secondo me sapere che la dialog di scelta dei colori della Figura 16 viene creata e usata

con una singola istruzione, ovvero basta invocare il metodo:

static JDialog
createDialog(Component c, String title,
boolean modal, JColorChooser chooserPane,
ActionListener okListener,
ActionListener cancelListener)

della classe JColorChooser.

Comunque credo di avere fatto capire quali sono le potenzialità del pacchetto, pacchetto che però ancora non è stato incluso nelle versioni di Java dei browser, e che quindi non è ancora possibile creare delle applet swing senza caricare plugins nei sopracitati browser.

Nei testi di swing è possibile inserire anche immagini, e inoltre si possono leggere e decodificare direttamente file di tipo html, rtf (Un cugino del .doc).

In Swing c'è la classe Jdialog, analoga alla Applet di java.applet, essa ne è una estensione, e presenta, oltre ai metodi della class Applet (di cui è una estensione), altri metodi molto interessanti, tra cui il metodo void setJMenuBar(JMenuBar menuBar).

Ovviamente siccome ci sono tanti GUI in più in swing, c'è la necessità di gestirne gli eventi, esiste quindi il sottopacchetto javax.swing.event che estende gli eventi di java.awt.event per gestire gli eventi dei componenti di swing.

LEZIONE 28: *Fondamenti di disegno con Java*

Nel capitolo precedente abbiamo visto come usare i componenti GUI di Abstract Window Toolkit per costruire delle interfacce utente grafiche per i nostri programmi, e abbiamo detto che un componente GUI è in sostanza un oggetto con delle qualità grafiche. Il componente GUI è un componente di alto livello del pacchetto awt, esso è costruito usando i componenti a basso livello del pacchetto, ovvero le primitive grafiche, e gli eventi.
Pensiamo ad esempio ad un bottone, esso graficamente non è altro che una scatola con un testo all'interno, che cambia aspetto quando viene schiacciato, ovvero quando sente l'evento ActionEvent di esso.
Usando le caratteristiche di basso livello, possiamo costruirci i nostri GUI personalizzati, oppure modificare quelli esistenti secondo i nostri bisogni, ma non solo, possiamo estendere un componente particolare, un componente tela, su cui possiamo disegnare quello che vogliamo e metterlo nelle nostre interfacce come un qualsiasi altro GUI.
Se usiamo gli applet piuttosto che le applicazioni possiamo evitare di usare la tela, in quanto l'applet ha implicitamente una tela. Ma iniziamo a vedere concretamente come è possiblie disegnare su un componente qualsiasi o su un applet, iniziamo a parlare delle applet, per poi estendere i ragionamenti fatti ai componenti in generale.

LEZIONE 29: *Funzioni paint, repaint, update..*

Nel linguaggio Java è possibile disegnare su un'applet semplicemente ridefinendone il metodo paint, questo metodo viene automaticamente invocato dal sistema quando la finestra contenente l'applet va in primo piano, quindi ogni comando grafico all'applet sarà dato nel metodo paint dell'applet

```
void paint (Graphics g)
{
// Disegno
}
```

quando l'applet ha disegnato, e si vogliono visualizzare eventuali modifiche è possibile invocare il metodo repaint() oppure il metodo update(Graphics g) che cancella l'area in cui l'applet ha disegnato e ne reinvoca la paint.
Il metodo paint(Graphics g) non può essere invocato direttamente.
Non solo l'applet ha il metodo paint che può essere ridefinito, ma anche gli oggetti di tipo Component, ovvero tutti i GUI, quindi per cambiare l'aspetto grafico di un qualsiasi componente grafico basta ridefinirne il metodo paint.
Tra tutti i component c'è la precedentemente citata tela su cui è possibile disegnare, essa è un oggetto della classe Canvas. Chi è abituato a programmare con altri linguaggi si aspetterà che nella paint ci sia un inizializzazione del device su cui si va a disegnare e poi una serie di istruzioni tipo drawLine, drawBox eccetera, ma devo dirvi da subito che questo non è del tutto vero, ovvero nei Component e negli applet non

c'è nessun metodo grafico.
A questo punto si spiega anche il parametro g di tipo Graphics della paint, infatti esso conterrà i metodi grafici usabili da Java, in effetti esso rappresenta in qualche modo l'area in cui verranno disegnate le varie primitive.

Graphics è una classe che non può essere instanziata, ma nella paint viene ricevuta come paramtero, ed è solo qui che può essere usata, essa contiene i metodi:

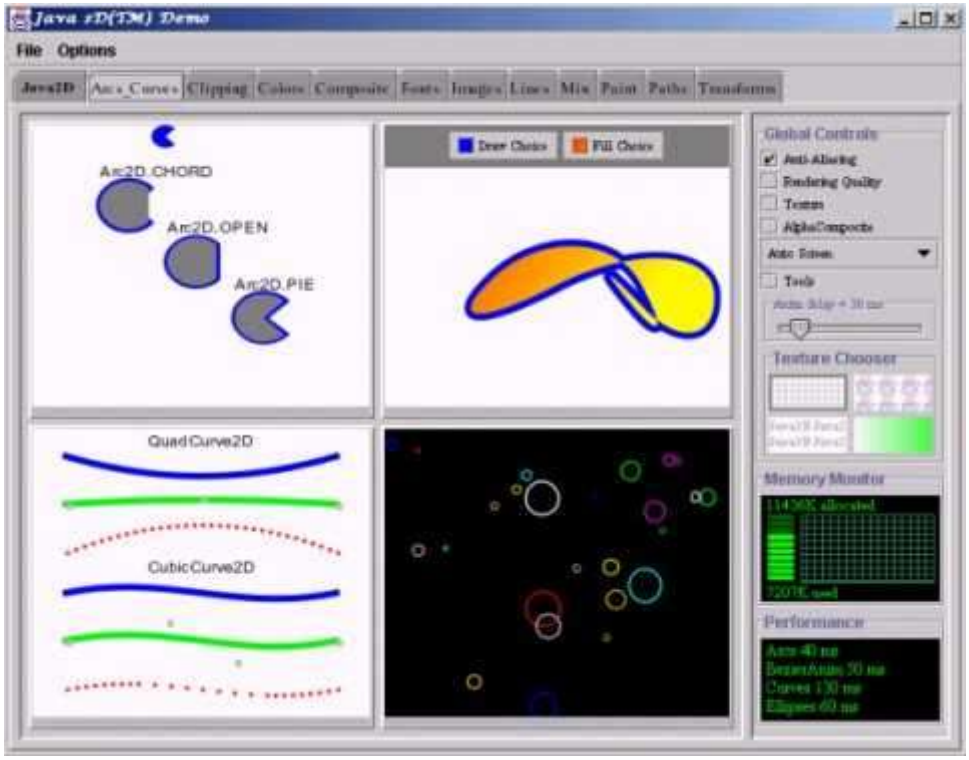
abstract void clearRect(int x, int y, int width, int height)
abstract void clipRect(int x, int y, int width, int height)
abstract void copyArea(int x, int y, int width, int height, int dx, int dy)
abstract Graphics create()
Graphics create(int x, int y, int width, int height)
abstract void dispose()
void draw3DRect(int x, int y, int width, int height,boolean raised)
abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
void drawBytes(byte[] data, int offset, int length, int x, int y)
void drawChars(char[] data, int offset, int length, int x, int y)
abstract boolean drawImage(…) Ne esistono varie versioni
abstract void drawLine(int x1, int y1, int x2, int y2)
abstract void drawOval(int x, int y, int width, int height)
abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)
void drawPolygon(Polygon p)
abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)
void drawRect(int x, int y, int width, int height)
abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
abstract void drawString(AttributedCharacterIterator iterator, int x, int y)
abstract void drawString(String str, int x, int y)
void fill3DRect(int x, int y, int width, int height, boolean raised)
abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
abstract void fillOval(int x, int y, int width, int height)
abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
void fillPolygon(Polygon p)
abstract void fillRect(int x, int y, int width, int height)
abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
void finalize()
abstract Shape getClip()
abstract Rectangle getClipBounds()
Rectangle getClipBounds(Rectangle r)
abstract Color getColor()
abstract Font getFont()
FontMetrics getFontMetrics()
abstract FontMetrics getFontMetrics(Font f)
boolean hitClip(int x, int y, int width, int height)
abstract void setClip(int x, int y, int width, int height)
abstract void setClip(Shape clip)
abstract void setColor(Color c)
abstract void setFont(Font font)
abstract void setPaintMode()
abstract void setXORMode(Color c1)
String toString()
abstract void translate(int x, int y)

Avremo modo tra non molto di vedere in dettaglio tutti questi metodi.
In Java2 la classe Graphics è stata potenziata aggiungendo la classe Graphics2D, che la estende aggiungendo notevoli potenzialità alla grafica in Java.
Per avere un'assaggio delle potenzialità di java potede vedere la Demo rilasciata insieme alle JDK, se ad esempio avete scaricato le JDK1.3 e le avete installate in c:\jdk1.3, la demo si trova in:

c:\jdk1.3\demo\jfc\Java2D

e per eseguirla bisogna battere dal prompt:

java -jar Java2Demo.jar



LEZIONE 30: *Visualizzazione di immagini*

Iniziamo a vedere i metodi drawImage della classe Graphics, con i quali è possibile visualizzare delle immagini salvate con formato gif o jpg.
Nella classe Graphics ci sono vari metodi drawImage, essi sono:

abstract boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer) *abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)*
abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)
abstract boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)
abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)
abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)

Tutti i metodi richiedono i due parametri Image e ImageObserver.
Image rappresenta l'immagine da visualizzare, essa è una classe astratta che fornisce un accesso indipendente dal formato a immagini grafiche, oggetti di questa classe vengono creati usando metodi di altre classi che creano delle immagini, a seconda del conteso in cui si vogliono usare le immagini.
Ad esempio, volendo disegnare una immagine in un componente Gui, si può usare il metodo createImage() della classe Component.
Volendo invece disegnare una immagine in un'applet, è possibile usare il metodo getImage() della classe Applet.
La classe Toolkit ha sia createImage() che getImage().
L'altro parametro richiesto da tutte le drawImage() è un oggetto che implementa l'interfaccia ImageObserver, e rappresenta in pratica l'oggetto grafico su cui verrà visualizzata l'immagine.
Component implementa l'interfaccia ImageObserver, di conseguenza la implementano tutte le sue estensioni, tra cui:

- Tutti i GUI
- Le Applets
- Tutte le finestre (tra cui i Frames)

Tra gli altiri parametri ci sono:

- la posizione x in cui verrà visualizzata l'imagine nel componente
- la posizione y in cui verrà visualizzata l'imagine nel componente il colore dello sfondo

Proviamo a fare un'esempio di visualizzazione di una immagine. Creiamo innanzitutto una immagine, ad

esempio la seguente

e la salviamo in formato gif usando un qualunque editore di immagini, chiamandola ad esempio pietro.jpg.
A questo punto creiamo la seguente applet:

```
import java.awt.*; // Per la classe Graphics
import java.applet.*; // Per la classe Applet
import java.net.*; // Per leURL
/*
Come abbiamo gia detto i files per gli applet non sono altro che degli URL
*/

public class VisualizzaImmagine extends Applet
{
Image ImmaginePietro;

public void init()
{

setBackground(Color.white);

ImmaginePietro=getImage(getDocumentBase(),"pietro.jpg");

}

public void paint(Graphics g)
{

g.drawImage(ImmaginePietro,0,0,this);
getAppletContext().showStatus("Visualizzo l'immagine pietro.jpg, Applet creata da Pietro Castellucci");

}

}
```

la mettiamo in un file chiamato VisualizzaImmagine.java e la invochiamo usando il documento html seguente (Immagine.html):

```
<html>
<head>
<title>
Applet VisualizzaImmagine, visualizza l'immagine pietro.jpg
</title>
</head>
<body>
<APPLET code="VisualizzaImmagine.class" width=385 height=100>
</APPLET>
<BR>
Quella di sopra è una applet, ma quella di sotto no!
<BR>
<IMG SRC="pietro.jpg">
</body>
</html>
```

Se eseguiamo l'applet ci accorgiamo che non vi è alcuna differenza tra l'immagine caricata nell'applet e quella caricata usando il tag IMG SRC dell'html, a parte il messaggio che appare quando vi si passa sopra con il mouse, potrebbe sembrare anche un lavoro inutile per le vostre pagine html.
Questo non è assolutamente vero, infatti l'imagine caricata nell'applet appartiene ad un programma che può

fare anche cose complesse, ad esempio è possibile aggiungere altri componenti GUI all'applet, oppure fare delle elaborazioni sull'immagine stessa, come mostra il seguente esempio (VisualizzaImmagine2.java):

```
import java.awt.*; // Per la classe Graphics
import java.applet.*; // Per la classe Applet
import java.net.*; // Per leURL
import java.awt.event.*; // Per gli eventi

/*
Come abbiamo gia detto i files per gli applet non sono altro che degli URL
*/

public class VisualizzaImmagine2 extends Applet
{
Image ImmaginePietro;

CheckboxGroup gruppo=new CheckboxGroup();

Checkbox b1=new Checkbox("Ferma",gruppo,true);

Checkbox b2=new Checkbox("Animata",gruppo,false);

Panel p=new Panel(new GridLayout(1,3));

public void init()
{

setBackground(Color.white);
setLayout(new BorderLayout());
ImmaginePietro=getImage(getDocumentBase(),"pietro.jpg");

p.add(new Label());
p.add(b1);
p.add(b2);

b1.addItemListener(new IL());
b2.addItemListener(new IL());

add(p,BorderLayout.SOUTH);

}

boolean MOVIMENTO=false;

int numero=0;

int inc=1;

public void paint(Graphics g)
{

if (!MOVIMENTO)
{
g.drawImage(ImmaginePietro,0,0,this);

getAppletContext().showStatus("Visualizzo l'immagine pietro.jpg, Immagine ferma, Applet creata da Pietro
Castellucci");

numero=0;

}
else
{

g.setPaintMode();
```

```
g.drawImage(Elabora(ImmaginePietro,numero),0,0,this);

for (int k=0;k<=99;k++) getAppletContext().showStatus("Visualizzo l'immagine pietro.jpg, Frame "+numero+", Applet creata da Pietro Castellucci");

getAppletContext().showStatus("Visualizzo l'immagine pietro.jpg, Frame "+numero+", Applet creata da Pietro Castellucci");

if (numero>=10) inc=-1;

if (numero<=0) inc=1;

numero+=inc;

}

repaint();

}


Image Elabora (Image img, int frm)
{

return img.getScaledInstance(324-(frm*20), 85-(frm*4),img.SCALE_FAST);

}


public class IL implements ItemListener
{
public void itemStateChanged(ItemEvent e)
{

if (b1.getState()) MOVIMENTO=false;
else MOVIMENTO=true;
repaint();
}

}
}
```

Caricata dal file html seguente:

```
<html>
<head>
<title>
Applet VisualizzaImmagine, visualizza l'immagine pietro.jpg
</title>
</head>
<body>
<APPLET code="VisualizzaImmagine2.class" width=385 height=100>
</APPLET>
<BR>
Quella di sopra è una applet, ma quella di sotto no!
<BR>
<IMG SRC="pietro.jpg">
</body>
</html>
```

La nuova classe Graphics2D mette a disposizione altri metodi drawImage, tra cui alcuni che hanno un parametro di tipo AffineTransform che come si capisce dal nome è una trasformazione affine dell'immagine, ovvero una scalatura, una rotazione o una traslazione, o una combinazione di queste.

Per avere una versione senza sfarfallio basta aggiungere la funzione:

```
public void update(Graphics g)
{
paint(g);
}
```

LEZIONE 31: *Disegno*

Capito il funzionamento della paint() siamo pronti a disegnare qualsiasi cosa su un'applet o su un generico componente.

La cosa più semplice da disegnare è la linea, per farlo usiamo il metodo drawLine() di Graphics:

drawLine(Iniziox, Inizioy, Finex, Finey)

il quale disegna una linea che parte dal punto (Iniziox, Inizioy) e arriva al punto (Finex, Finey)
Prima di usare la drawLine() capiamo come si possono cambiare i colori delle cose disegnate.
I colori si cambiano usando il metodo setColor(Color c) della classe Graphics.
Color è un'altra classe di awt che permette di definire un colore, alcuni suoi costruttori sono:

Color(float r, float g, float b), crea un colore specificando i valori RGB (Red - Green - Blue, ovvero Rosso - Verde - Blu) con dei valori compresi tra 0 e 1.

Color(float r, float g, float b, float a), come il precedente, solo che permette di definire anche il valore alfa

Color(int r, int g, int b) , crea un colore specificando i valori RGB (Red - Green - Blue, ovvero Rosso - Verde - Blu) con dei valori compresi tra 0 e 255.

Color(int r, int g, int b, int a), come il precedente, solo che permette di definire anche il valore alfa

Alcuni colori semplici sono dati da delle costanti nella classe color:

Color.black
Color.blue
Color.cyan
Color.darkGray
Color.gray
Color.green
Color.lightGray
Color.magenta
Color.orange
Color.pink
Color.red
Color.white
Color.yellow

Nel seguente applet si disegnano alcune linee colorate.

```
import java.awt.Graphics;

import java.awt.Color;

import java.awt.Button;

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.Panel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.applet.*;
```

```
public class linee extends Applet
{

final int SI=14;

final int NO=0;

Button R=new Button("Rosso");

Button G=new Button("Verde");

Button B=new Button("Blu");

int r_=0;

int g_=0;

int b_=0;

int ir=0;

int ig=0;

int ib=0;

public void init()
{

setLayout(new BorderLayout());

Panel np=new Panel(new GridLayout(1,3));

np.add(R);

np.add(G);

np.add(B);

R.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{

ir=SI;
ig=NO;
ib=NO;
repaint();
}
}

);



G.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{

ir=NO;
ig=SI;
ib=NO;
repaint();
}
```

```
}

);
B.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{

ir=NO;
ig=NO;
ib=SI;
repaint();
}
}

);



add(np,BorderLayout.SOUTH);

}

public void paint(Graphics g)
{


for (int i=0;i<200;i+=10)
{

g.setColor(new Color(r_,g_,b_));

r_=r_+ir;
g_=g_+ig;
b_=b_+ib;

g.drawLine(0,i,i,200);

}

g.setColor(Color.black);
g.drawLine(0,0,0,200);
g.drawLine(0,200,200,200);

r_=0;
g_=0;
b_=0;

}

}
```

Chiamiamo il file linee.java e lo carichiamo con il file html seguente (linee.html):

```
<html>
<head>
<title>
Applet Linee (linee.class)
</title>
</head>
<body>
<APPLET code="linee.class" width=200 height=300>
</APPLET>
</body>
</html>
```

LEZIONE 32: *Figure geometriche e testo*

RETTANGOLI

Graphics permette di disegnare dei rettangoli specificandone due spigoli opposti usando il metodo:

drawRect(Iniziox,Inizioy,Finex,Finey);

questo metodo è equivalente a:

drawLine(Iniziox,Inizioy,Finex,Inizioy);
drawLine(Iniziox,Inizioy,Iniziox,Finey);
drawLine(Finex,Inizioy,Finex,Finey);
drawLine(Iniziox,Finey,Finex,Finey);

è possibile anche costruire rettangoli che danno una specie di effetto rialzato o incavato usando il metodo draw3DRect(int x,int y,int larghezza,int altezza,boolean rialzato) e rettangoli con gli spigoli arrotondati con il metodo drawRoundRect(int x, int y, int larghezza, int altezza, int larghezzaArco, int altezzaArco).
E' inoltre possibile colorare all'interno tutte le figure chiuse usando i metodi fill, ad esempio per colorare un RoundRect bisogna usare fillRoundRect, che ha glistessi parametri della corrispondente draw.

CERCHI ED ELLISSI

Si può disegnare una ellisse usando la drawOval (int x, int y, int larghezza,int altezza) e colorarla usando la fillOval(int x, int y, int larghezza,int altezza).
Se altezza e altezza sono uguali si disegna un cerchio.

POLIGONI

Usando la drawPolygon si disegna un generico poligono. Esistono due tipi di drawPolygon:

drawPoligon(int[] PUNTIx, int PUNTIy, int NUMEROPUNTI);

e

drawPolygon (Polygon p)

La seconda usa un oggetto della classe Polygon, che definisce un poligono. Esistono le corrispondenti fillPolygon.

TESTO

Per disegnare una stringa si può usare il metodo drawString (String TESTO, int x, int y);
Un testo può essere disegnato con diverse Font, per cambiare le Font Graphics mette a disposizione il metodo setFont(Font font), dove font è l'oggetto che definisce il tipo di carattere.
Font è la classe che definisce i caratteri, è possibile instanziarla creando uno specifico Font, usando Font (String name, int style, int size).
Font è una classe abbastanza complicata, però è facile trovare ed usare i Fonts del sistema mentre si esegue un programma java: Nelle vecchie versioni di java baste scrivere:

String [] NOMI=Toolkit.getDefaultToolkit().getFontList();

Mentre nelle nuove versioni di Java (da JDK 1.2)

String [] NOMI=GraphicsEnvironment.
getLocalGraphicsEnvironment().
getAvailableFontFamilyNames();

è poi facile instanziare oggetti Font conoscendo i nomi.

ARCHI

Si possono disegnare degli archi usando drawArc(int x, int y, int larghezza, int altezza, int angoloIniziale, int angoloArco)

L'uso di tutti questi metodi è illustrato nell'esempio seguente.
Il file che lo manda in esecuzione è:

```
<html>
<head>
<title>
Applet grafDemo (grafDemo.class)
</title>
</head>
<body>
<APPLET code="grafDemo.class" width=500 height=400>
</APPLET>
</body>
</html>
```

LEZIONE 33: *file grafDemo.java*

L'esempio, editato nel file grafDemo.java è:

```
import java.awt.Graphics;

import java.awt.Color;

import java.awt.Button;

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.FlowLayout;

import java.awt.Panel;

import java.awt.Label;

import java.awt.Choice;

import java.awt.Font;

import java.awt.Toolkit;

import java.awt.Checkbox;

import java.awt.Scrollbar;

import java.awt.GraphicsEnvironment;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.awt.event.AdjustmentListener;

import java.awt.event.AdjustmentEvent;

import java.applet.*;

public class grafDemo extends Applet
{

Scrollbar R=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);

Scrollbar G=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);

Scrollbar B=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);
```

```
Choice FN=new Choice();

Choice GR=new Choice();

Checkbox F=new Checkbox("Figure chiuse piene",false);

int [] puntiX={1,100,200,300,399,300,200,100};

int [] puntiY={200,150,50,150,200,100,250,130};

int punti=8;

public void init()
{

// Calcolo i Font del sistema:

// String[] NOMI=GraphicsEnvironment.
// getLocalGraphicsEnvironment().
// getAvailableFontFamilyNames();

/*
Per Java presente sui browser
String [] NOMI=Toolkit.getDefaultToolkit().getFontList();
*/

String [] NOMI=Toolkit.getDefaultToolkit().getFontList();

try
{
int indice=0;

while (true)
{

FN.addItem(NOMI[indice++]);

}

}
catch (ArrayIndexOutOfBoundsException e)
{};

setLayout(new BorderLayout());

Panel np=new Panel(new GridLayout(1,3));


Panel rosso=new Panel(new FlowLayout());
rosso.add(new Label("Rosso"));
rosso.add(R);
np.add(rosso);

Panel verde=new Panel(new FlowLayout());
verde.add(new Label("Verde"));
verde.add(G);
np.add(verde);

Panel blu=new Panel(new FlowLayout());
blu.add(new Label("Blu"));
blu.add(B);
np.add(blu);

R.setUnitIncrement(10);
```

```
R.setValue(255);

G.setUnitIncrement(10);
G.setValue(255);

B.setUnitIncrement(10);
B.setValue(255);

add(np,BorderLayout.SOUTH);

GR.addItem("Linea");

GR.addItem("Rettangolo");

GR.addItem("Rettangolo 3D");

GR.addItem("Rettangolo Arrotondato");

GR.addItem("Cerchio");

GR.addItem("Ellisse");

GR.addItem("Poligono generico");

GR.addItem("Testo");

GR.addItem("Arco");

Panel up=new Panel(new GridLayout(1,3));

up.add(GR);

up.add(FN);

up.add(F);

add(up,BorderLayout.NORTH);

R.addAdjustmentListener(new AL());

G.addAdjustmentListener(new AL());

B.addAdjustmentListener(new AL());

GR.addItemListener(new IL());

FN.addItemListener(new IL());

F.addItemListener(new IL());

}


public void paint(Graphics g)
{

g.setColor(new Color(255-R.getValue(),
255-G.getValue(),
255-B.getValue()
));

g.setFont(new Font(FN.getSelectedItem(),0,40));

boolean filled=F.getState();
```

```
int pg=GR.getSelectedIndex();

g.drawString(GR.getSelectedItem(),1,300);

if (pg==0)
{
g.drawLine(1,100,399,300);
return;
}

if (pg==1)
{
g.drawRect(50,100,250,100);
if (filled) g.fillRect(50,100,250,100);
return;
}

if (pg==2)
{
g.draw3DRect(50,100,250,100,true);
return;
}

if (pg==3)
{
g.drawRoundRect(50,100,300,100,20,20);
if (filled) g.fillRoundRect(50,100,300,100,20,20);
return;
}

if (pg==4)
{
g.drawOval(100,100,200,200);
if (filled) g.fillOval(100,100,200,200);
return;
}

if (pg==5)
{
g.drawOval(100,100,200,100);
if (filled) g.fillOval(100,100,200,100);
return;
}

if (pg==6)
{
g.drawPolygon(puntiX,puntiY,punti);
if (filled) g.fillPolygon(puntiX,puntiY,punti);
return;
}


if (pg==7)
{
g.drawString("Questa è una stringa",1,200);
return;
}

if (pg==8)
{
g.drawArc(1,50,398,200,10,270);
if (filled) g.fillArc(1,50,398,200,10,270);
return;
}
```

```
}

public class IL implements ItemListener
{
public void itemStateChanged(ItemEvent e)
{
repaint();
}

}

public class AL implements AdjustmentListener
{
public void adjustmentValueChanged(AdjustmentEvent e)
{
repaint();
}

}


}
```

LEZIONE 34: *Note per compilare il programma*

Visto che il programma è stato creato per un browser (che hanno le vecchie versioni di Java) ho dovuto usare String [] NOMI=Toolkit.getDefaultToolkit().getFontList(); per avere i caratteri, se si compila con JDK 1.2 o 1.3 si deve scrivere:

Ø javac grafDemo.java -deprecation

Il quale darà un warning per dire che si usa un metodo deprecato.
Se invece userete l'appletviewer per visualizzare l'applet potete cambiare

String [] NOMI=Toolkit.getDefaultToolkit().getFontList();

con

String [] NOMI=GraphicsEnvironment.
getLocalGraphicsEnvironment().
getAvailableFontFamilyNames();

e apprezzare la grande quantità di Font presenti.

Nell'esempio ho usato per i colori tre scrollbar, che tra i GUI non avevamo visto. Per creare l'oggetto scrollbar ho usato il costruttore:

Scrollbar R=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);

che crea una scrollbar verticale con valori che vanno da 0 a 255 con valore iniziale 0.
Poi ho settato il valore della scrollbar con R.setValue(255); ed ho detto che l'incremento per ogni click deve essere di 10 con R.setUnitIncrement(10);

Per ascoltare i cambiamenti uso un oggetto che implementa la classe AdjustmentListener che ascolta eventi di tipo AdjustmentEvent. Per associare l'ascoltatore all'oggetto scrollbar si usa il metodo addAdjustmentListener.

LEZIONE 35: *Il suono di Java 1.1.x e 1.2.x*

In tutto quello fatto finora abbiamo cercato di creare programmi compatibili tra le vecchie e la nuova versione di Java, questo per creare degli applet generali che potessero girare su tutti i browser (i quali come ho già detto attualmente hanno Java 1.1.x come motore per le applet).
Nell'argomento che stiamo per trattare purtroppo questo non è possibile, infatti con le vecchie versioni di Java si possono creare delle applet che leggono e suonano file di tipo .au, usando i metodi play di applet e l'interfaccia AudioClip.
Tutto questo è ancora possibile con Java2, versione 1.3, ma sono stati inclusi dei pacchetti per gestire i suoni, che offrono delle potenzialità eccezionali.

LEZIONE 36: *Suono: javax.swing.sampled*

I pacchetti aggiunti al linguaggio sono 4:

1. javax.sound.midi
2. javax.sound.midi.spi
3. javax.sound.sampled
4. javax.sound.sampled.spi

Le parti che finiscono per spi servono a leggere e scrivere files sonori, convertirli o leggere files di strumenti nel caso di midi. Noi non vedremo queste parti, mentre vedremo javax.sound.sampled e javax.sound.midi.
Il pacchetto javax.swing.sampled permette di registrare suonare e modificare dati sonori.
Il contenuto del pacchetto è:

Interfacce

- Clip
- DataLine
- Line
- LineListener
- Mixer
- Port
- SourceDataLine
- TargetDataLine

Classi

- AudioFileFormat
- AudioFileFormat.Type
- AudioFormat
- AudioFormat.Encoding
- AudioInputStream
- AudioPermission

- AudioSystem
- BooleanControl
- BooleanControl.Type
- CompoundControl
- CompoundControl.Type
- Control
- Control.Type
- DataLine.Info
- EnumControl
- EnumControl.Type
- FloatControl
- FloatControl.Type
- Line.Info
- LineEvent
- LineEvent.Type
- Mixer.Info
- Port.Info
- ReverbType

Eccezioni

- LineUnavailableException
- UnsupportedAudioFileException

Vediamo come usare questo pacchetto solo per suonare un file sonoro, ad esempio il file chiamato italian.wav.

Inanzitutto dobbiamo reperire il file, e questo lo facciamo come al solito instanziando un oggetto della classe File (di java.io)

File sf=new File("italian.wav");

A questo punto iniziamo una try{} catch, nella quale includeremo tutto quello che ha a che fare con il suono, cattureremo le eccezioni:

catch(UnsupportedAudioFileException ee){}
catch(IOException ea){}
catch(LineUnavailableException LUE){};

Per usare il motore sonoro di Java dobbiamo partire dalla classe AudioSystem, da questa otterremo due oggetti di tipo AudioFileFormat e AudioInputStream

AudioFileFormat aff=AudioSystem.getAudioFileFormat(sf);
AudioInputStream ais=AudioSystem.getAudioInputStream(sf);

che rappresentano il contenuto del file italian.wav

Dobbiamo creare un oggetto AudioFormat da AudioFileFormat per poterlo suonare

AudioFormat af=aff.getFormat();

Il nostro scopo è quello di creare un oggetto che implementi l'interfaccia Clip, il quale contiene i metodi play, per fare questo scriveremo:

DataLine.Info info = new DataLine.Info(
Clip.class,
ais.getFormat(),
*((int) ais.getFrameLength() **
af.getFrameSize()));

Clip ol = (Clip) AudioSystem.getLine(info); Da ol, che è di tipo Clip possiamo suonare, e questo si fa aprendo la linea e invocando i metodi per suonare:

ol.open(ais); <-(APRE l'input stream, ovvero il file italian.wav)

e quindi

```
ol.loop(NUMERO);
```

dove NUMERO indica il numero di ripetizioni del dile, nel nostro caso è Clip. LOOP_CONTINUOUSLY per indicare che bisogna ripetere all'infinito.
L'esempio completo, editato in suono.java è il seguente:

```
import javax.swing.*;
import javax.sound.sampled.*;
import java.io.*;

public class suono extends JFrame
{


public suono()
{

File sf=new File("italian.wav");
AudioFileFormat aff;
AudioInputStream ais;


try
{
aff=AudioSystem.getAudioFileFormat(sf);

ais=AudioSystem.getAudioInputStream(sf);


AudioFormat af=aff.getFormat();


DataLine.Info info = new DataLine.Info(
Clip.class,
ais.getFormat(),
((int) ais.getFrameLength() *
af.getFrameSize()));

Clip ol = (Clip) AudioSystem.getLine(info);

ol.open(ais);

ol.loop(Clip.LOOP_CONTINUOUSLY);

System.out.println("Riproduzione iniziata, premere CTRL-C per interropere");

}
catch(UnsupportedAudioFileException ee){}
catch(IOException ea){}
catch(LineUnavailableException LUE){};

}

public static void main(String[] ar)
{
new suono();

}

}
```

Sembra un po macchinoso, ma seguendo questa procedura è possiblie leggere e suonare ogni file wav. Per effettuare altre operazioni il pacchetto è abbastanza complesso, e oggi non esistono manuali che ne spiegano il funzionamento (tranne la documentazione del JDK, che come avrete avuto modo di vedere non spiega come usare le cose, ma le descrive soltanto).

LEZIONE 37: *Il pacchetto javax.suond.midi*

Vediamo come leggere e suonare un file midi, per fare questo ci occorre il pacchetto javax.suond.midi, il quale contiene:

Interfacce

- ControllerEventListener
- MetaEventListener
- MidiChannel
- MidiDevice
- Receiver
- Sequencer
- Soundbank
- Synthesizer
- Transmitter

Classi

- Instrument
- MetaMessage
- MidiDevice.Info
- MidiEvent
- MidiFileFormat
- MidiMessage
- MidiSystem
- Patch
- Sequence
- Sequencer.SyncMode
- ShortMessage
- SoundbankResource
- SysexMessage
- Track
- VoiceStatus

Eccezioni

- InvalidMidiDataException
- MidiUnavailableException

La tecnica per suonare un file midi è simile a quella usata per i file wav, solo che si parte da MidiSystem.

Nel seguente esempio viene suonato il file sorpresa.mid, che è un file midi che io apprezzo molto, e credo che apprezzaranno tutti i miei coetanei.
Editatelo nel file midifiles.java

```
import javax.swing.*;
import javax.sound.midi.*;
import java.io.*;

public class midifiles extends JFrame
{

public midifiles()
{
try
{

File f2=new File("sorpresa.mid");

MidiFileFormat mff2=MidiSystem.getMidiFileFormat(f2);

Sequence S=MidiSystem.getSequence(f2);

Sequencer seq=MidiSystem.getSequencer();
```

```
seq.open();

seq.setSequence(S);

seq.start();

System.out.println("Sorpresa per tutti i miei coetanei");
System.out.println("Premere CTRL-C per interrompere");



}
catch(MidiUnavailableException ecc){}
catch(InvalidMidiDataException ecc2){}
catch(IOException ecc3){}
;
}


public static void main(String[] ar)
{
new midifiles();

}

}
```

LEZIONE 38: *Sintetizzare suoni*

Come ho gia detto è anche possibile sintetizzare suoni, questo è quello che fa il prossimo programma, da editare nel file sintetizzatore.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;
import java.util.*;
import javax.sound.sampled.*;
import javax.sound.midi.*;

public class sintetizzatore extends JFrame
{

int TEMPO=50;
int CANALE=0;
int UNS=-1;
int STRUMENTO=0;
Sintetizzatore SINTETIZZATORE=new Sintetizzatore();

// GUI

JLabel tastiera1=new JLabel(new ImageIcon("tastiera1.gif"));
JLabel tastiera2=new JLabel(new ImageIcon("tastiera2.gif"));

JButton OkB=new JButton("Ok");


JComboBox Strumenti=new JComboBox();
JComboBox Canali=new JComboBox();

JLabel uns=new JLabel("Ultima nota suonata");
JTextField msg=new JTextField(
"Nessuna nota"
```

```
);

JSlider SL=new JSlider();

public sintetizzatore()
{
setTitle("Sintetizzatore sonoro by Pietro Castellucci");

setupAspetto();

setupValori();

setupEventi();

SINTETIZZATORE.setStrumento(
Strumenti.getSelectedIndex(),CANALE);

setResizable(false);

setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

pack();

Toolkit t = Toolkit.getDefaultToolkit();

Dimension d=t.getScreenSize();

Dimension win=getSize();

win=getSize();

setLocation(d.width/2-(win.width/2)-1,d.height/2-(win.height/2)-1);

show();

}


void setupAspetto()
{
tastiera1.setCursor(new Cursor(Cursor.HAND_CURSOR));
tastiera2.setCursor(new Cursor(Cursor.HAND_CURSOR));
OkB.setCursor(new Cursor(Cursor.HAND_CURSOR));

tastiera1.setToolTipText(
"Cliccami per suonare"
);

tastiera2.setToolTipText(
"Cliccami per suonare"
);

OkB.setToolTipText(
"Esco dal programma"
);



JPanel P=new JPanel(new BorderLayout());
JPanel Pannello=new JPanel(new GridLayout(2,1));
Pannello.add(tastiera1);
Pannello.add(tastiera2);

P.add(Pannello,BorderLayout.CENTER);
```

```
JPanel SP=new JPanel(new FlowLayout());

Strumenti.setBorder(
BorderFactory.createTitledBorder(
"Strumenti"
)
);

Canali.setBorder(
BorderFactory.createTitledBorder(
"Canali"
)
);

SP.add(Strumenti);
// SP.add(Canali);

SP.add(uns);
msg.setEditable(false);
msg.setBackground(P.getBackground());
SP.add(msg);

SP.add(OkB);

P.add(SP,BorderLayout.SOUTH);

SL.setBorder(BorderFactory.createTitledBorder(
"Pressione "
));

SL.setOrientation(JSlider.VERTICAL);

SL.setMajorTickSpacing(25);
SL.setMaximum(100);
SL.setMinimum(0);
SL.setMinorTickSpacing(1);

SL.setPaintLabels(true);
SL.setPaintTicks(true);
SL.setPaintTrack(true);
SL.setSnapToTicks(true);

SL.setValue(TEMPO);

P.add(SL,BorderLayout.EAST);


setContentPane(P);
}




void setupValori()
{
try
{
int i=0;
while (true)
{
String nome= SINTETIZZATORE.Strumenti[i++].getName();
Strumenti.addItem(nome);
}
}
catch (ArrayIndexOutOfBoundsException e)
```

```
{};


try
{
int i=0;
while (true)
{
String nome="Canale"+" "+(i+1);
SINTETIZZATORE.CANALI[i++].allNotesOff();
Canali.addItem(nome);
}
}
catch (ArrayIndexOutOfBoundsException e)
{};

if (Strumenti.getItemCount()>0)
Strumenti.setSelectedIndex(STRUMENTO);
}



void setupEventi()
{
tastiera1.addMouseListener(new Tastiera1Listener());
tastiera2.addMouseListener(new Tastiera2Listener());



Strumenti.addItemListener(new ItemListener()
{
public void itemStateChanged(ItemEvent e)
{

SINTETIZZATORE.setStrumento(
Strumenti.getSelectedIndex(),CANALE);

}
}
);



Canali.addItemListener(new ItemListener()
{
public void itemStateChanged(ItemEvent e)
{
CANALE=Canali.getSelectedIndex();
SINTETIZZATORE.cambiaCanale(CANALE);
}
});



OkB.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
System.out.println("Grazie per avere suonato con me!");
System.exit(0);
}
}
```

```
);




}

// Eventi


public class Tastiera1Listener implements MouseListener
{
public void mouseClicked(MouseEvent e)
{}

public void mouseEntered(MouseEvent e)
{}

public void mouseExited(MouseEvent e)
{}

public void mousePressed(MouseEvent e)
{
// System.out.println("Tastiera 1 Punto: (x="+e.getX()+",y="+e.getY()+")");
UNS=getNota(e.getX());
// System.out.println("Nota="+nota);
TEMPO=SL.getValue();
SINTETIZZATORE.suona(UNS,TEMPO,CANALE);
msg.setText(""+(UNS+1));
}

public void mouseReleased(MouseEvent e)
{}

}


public class Tastiera2Listener implements MouseListener
{
public void mouseClicked(MouseEvent e)
{}

public void mouseEntered(MouseEvent e)
{}

public void mouseExited(MouseEvent e)
{}

public void mousePressed(MouseEvent e)
{
// System.out.println("Tastiera 2 Punto: (x="+e.getX()+",y="+e.getY()+")");
UNS=64+getNota(e.getX());
// System.out.println("Nota="+nota);
TEMPO=SL.getValue();
SINTETIZZATORE.suona(UNS,TEMPO,CANALE);
msg.setText(""+(UNS+1));
}

public void mouseReleased(MouseEvent e)
{}

}
```

```
// Utility fun

int getNota (int pos)
{
int nota;
nota=(pos/12);
return nota;
}



public static void main(String[] arg)
{

new sintetizzatore();

}


//****************************************
// SOUND MANAGER
//****************************************

//************************************************************
// Sintetizzatore:
//************************************************************

public class Sintetizzatore
{
private Synthesizer SYNT;
private Sequencer sequencer;
private Sequence seq;
private Soundbank BANK;
public Instrument[] Strumenti;
public MidiChannel[] CANALI;

public Sintetizzatore()
{
setupSintetizzatore();
}

void setupSintetizzatore()
{
try
{
SYNT=MidiSystem.getSynthesizer();
sequencer=MidiSystem.getSequencer();
seq= new Sequence(Sequence.PPQ, 10);
SYNT.open();
BANK = SYNT.getDefaultSoundbank();
if (BANK != null)
Strumenti = SYNT.getDefaultSoundbank().getInstruments();
else
Strumenti = SYNT.getAvailableInstruments();
CANALI=SYNT.getChannels();
}
catch(MidiUnavailableException ecc){tuttonull();}
catch(InvalidMidiDataException ecc2){tuttonull();}
;
}
```

```
void tuttonull()
{
SYNT=null;
sequencer=null;
seq=null;
BANK=null;
Strumenti=null;
}

public void setStrumento(int str,int can)
{

SA=str;
int prog=Strumenti[str].getPatch().getProgram();
CANALI[can].programChange(prog);

}

private int SA=0;

public void cambiaCanale(int can)
{
int prog=Strumenti[SA].getPatch().getProgram();
CANALI[can].programChange(prog);
}

public void suona(int nota,int tempo, int canale)
{
CANALI[canale].allNotesOff();
CANALI[canale].noteOn(nota,tempo);
}

public void suona(int nota,int tempo)
{
suona(nota,tempo,0);
}

public void zitto()
{
CANALI[0].allNotesOff();
}


} // End of Sintetizzatore

}
```

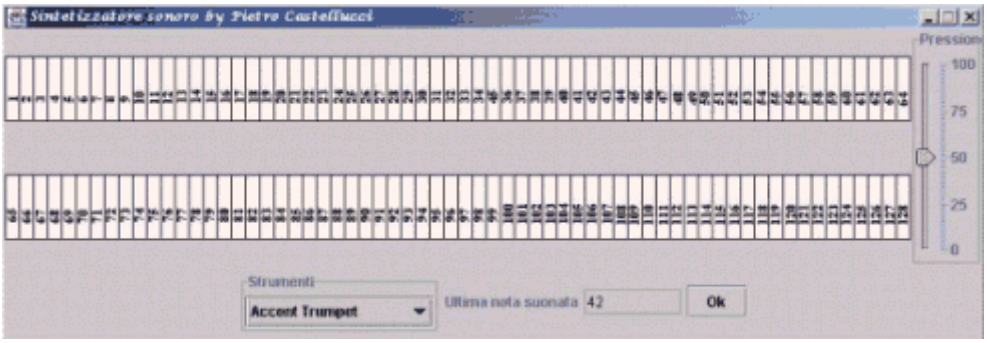
Avrete notato la corposità del programma, non fatevi spaventare, perché la maggior parte del codice serve per la grafica e per sentire gli eventi, la parte interessante per sintetizzare suoni l'ho racchiusa nella sottoclasse Sintetizzatore, ovvero in sintetizzatore.Sintetizzatore

ovvero la seguente:

```
//************************************************************
// Sintetizzatore:
//************************************************************
public class Sintetizzatore
{
private Synthesizer SYNT;
private Sequencer sequencer;
private Sequence seq;
private Soundbank BANK;
public Instrument[] Strumenti;
public MidiChannel[] CANALI;
```

```
public Sintetizzatore()
{
setupSintetizzatore();
}
void setupSintetizzatore()
{
try
{
SYNT=MidiSystem.getSynthesizer();
sequencer=MidiSystem.getSequencer();
seq= new Sequence(Sequence.PPQ, 10);
SYNT.open();
BANK = SYNT.getDefaultSoundbank();
if (BANK != null)
Strumenti = SYNT.getDefaultSoundbank().getInstruments();
else
Strumenti = SYNT.getAvailableInstruments();
CANALI=SYNT.getChannels();
}
catch(MidiUnavailableException ecc){tuttonull();}
catch(InvalidMidiDataException ecc2){tuttonull();}
;
}
void tuttonull()
{
SYNT=null;
sequencer=null;
seq=null;
BANK=null;
Strumenti=null;
}
public void setStrumento(int str,int can)
{

SA=str;
int prog=Strumenti[str].getPatch().getProgram();
CANALI[can].programChange(prog);

}
private int SA=0;
public void cambiaCanale(int can)
{
int prog=Strumenti[SA].getPatch().getProgram();
CANALI[can].programChange(prog);
}
public void suona(int nota,int tempo, int canale)
{
CANALI[canale].allNotesOff();
CANALI[canale].noteOn(nota,tempo);
}

public void suona(int nota,int tempo)
{
suona(nota,tempo,0);
}
public void zitto()
{
CANALI[0].allNotesOff();
}
} // End of Sintetizzatore
```

L'aspetto del programma è il seguente:



LEZIONE 39: *Conclusioni e bibliografia*

bbiamo fatto una carrellata su alcune parti del famoso linguaggio Java, usatissimo per scrivere programmi che girano su internet e non solo, iniziando dalle basi fino ad arrivare alle interfacce grafiche, la grafica ed infine il suono, tutti aspetti abbastanza avanzati della programmazione.
Vorrei scusarmi con i lettori meno esperti se hanno avuto qualche problema nel comprendere alcune parti della guida, e allo stesso tempo vorrei scusarmi con i più esperti se hanno trovato le alcune cose noiose o troppo semplici. Chi ha seguito il corso dovrebbe essere in grado di scriversi delle semplici applet e applicazioni da solo, non penso che siate ancora in grado di gestire grosse applicazioni complesse, anche se con un po' di pratica e i semplici concetti del corso penso che diverrete ottimi programmatori.
Io rimmarrò sempre disponibile per eventuali domande sul corso o chiarimenti (non scrivetemi per i Javascript o per la configurazione degli applet scaricati dalla rete).
Per chi volesse approfondire gli argomenti trattati o vederne di nuovi, pubblico la bibliografia completa dalla quale partono le mie conoscenze.

Un saluto a tutti

Pietro Castellucci
Foiano di Val Fortore (Benevento)

Bibliografia

- Documentazione ufficiale della Sun Microsystems di Java Development Kit Versioni 1.3, 1.3 Relase Candidate 1, 1.3 Relase Candidate 2, 1.3 Relase Candidate 3, 1.3 Beta, 1.2.2 , 1.2.1. Disponibile in rete sul sito ufficiale di Java: http://java.sun.com
- Java Didattica e Programmazione, K.Arnold e J. Gosling, Addison-Wesley Prima edizione, Marzo 1997
- Manuale QUE - Special Edition Using Java, 2nd Edition, versione trovata su internet.
- Java2 Tutto&Oltre, J. Jaworski, SAMS Publishing - APOGEO
- Javatm 2D Graphics, J. Knudsen, O'REILLY
- Ambiente per l'esplorazione di micromondi concorrenti, Pietro Castellucci, Tesi di Laurea in Informatica.