Appunti dei corsi di

Programmazione di Rete
Sistemi di elaborazione: Reti II

PROF. G. BONGIOVANNI

0) Introduzione al linguaggio Java per programmatori C++

Il linguaggio Java è un linguaggio di programmazione orientato agli oggetti, derivato dal C++ (e quindi indirettamente dal C) e creato da James Gosling e altri ingegneri di Sun Microsystems. Il gruppo iniziò a lavorare nel 1991, il linguaggio inizialmente si chiamava Oak. Il nome fu successivamente cambiato in Java a causa di un problema di copyright (il linguaggio di programmazione Oak esisteva già nel 1991). Java fu annunciato ufficialmente il 23 maggio 1995 a SunWorld. La piattaforma di programmazione Java è fondata sul linguaggio stesso, sulla Java Virtual Machine (JVM) e sulle API. Java è un marchio registrato di Sun Microsystems. Il 13 novembre 2006 la Sun Microsystems ha rilasciato la sua implementazione del compilatore Java e della macchina virtuale sotto licenza GPL. L'8 maggio 2007 SUN ha rilasciato anche le librerie (tranne alcune componenti non di sua proprietà) sotto licenza GPL rendendo Java un linguaggio di programmazione la cui implementazione di riferimento è libera.

(Da ***Wikipedia***, l'enciclopedia libera)

0.1) Caratteristiche principali

Le caratteristiche salienti della piattaforma Java sono le seguenti:

- linguaggio totalmente orientato ad oggetti;
- esecuzione indipendente dalla piattaforma di calcolo;
- presenza di un ricco insieme di API per:
  - costruzione GUI;
  - gestione eventi;
  - gestione I/O (da/verso memoria, file system, rete);
  - gestione multithreading;
  - gestione eccezioni;
- progettata per l'esecuzione sicura del codice distribuito via rete.

0.1.1) Orientamento a oggetti

Java è un linguaggio che sintatticamente assomiglia molto a C e C++, ma è strettamente object oriented. Il che significa che **tutto il codice** deve essere contenuto in ***classi***, e quindi che non esistono funzioni che non siano metodi di qualche classe (funzioni globali o top level).

Una classe contiene sia variabili che metodi. Per fare riferimento ad una variabile o ad un metodo di una classe si usa l'operatore ***dot*** (**.**)

Ad esempio, per la classe:

```
public class Prova {

      int a, b;

      public metodo1() {
         ……
      }
}
```

e per un oggetto `miaProva` istanziato da tale classe,

`miaProva.a` è un riferimento alla variabile `a` di `miaProva`
`miaProva.metodo1()` è una chiamata del metodo `metodo1()` di `miaProva`

La funzione `main()`, top-level in C++, che anche in Java viene eseguita all'avvio dell'applicazione sviluppata, deve essere definita come ***metodo statico pubblico*** di una ***classe pubblica*** (che può anche essere l'unica classe del codice):

```
public class Test {
      public static void main(String argv[]) {
            System.out.println("Hello world");
            /*
            …altro codice del main…
            */
      }
}
```

L'ereditarietà in Java è implementata mediante la parola chiave ***extends*** applicata dopo la dichiarazione di una classe:

```
public class Quadrato extends Quadrilatero {
      …
      …
}
```

Come in C++, una classe derivata può ***aggiungere*** nuovi metodi a quelli della classe da cui deriva, o anche ***ridefinire***, mantenendone la ***firma***, tutti o alcuni dei metodi di quella da cui deriva.

A differenza del C++, invece, una classe può estendere una sola altra classe. In altre parole, l'ereditarietà multipla non è consentita in Java.

Interfacce
Un'interfaccia consiste nella definizione di uno o più metodi ***non implementati***.

```
public interface Contatore{
      void start(int);
      void stop();
}
```

La classe che implementa un'interfaccia deve implementare almeno i metodi definiti nell'interfaccia:

```
public MyCont implements Contatore{
        void start(int){
                // codice della funzione start
        }
        void stop(){
                // codice della funzione stop
        }
}
```

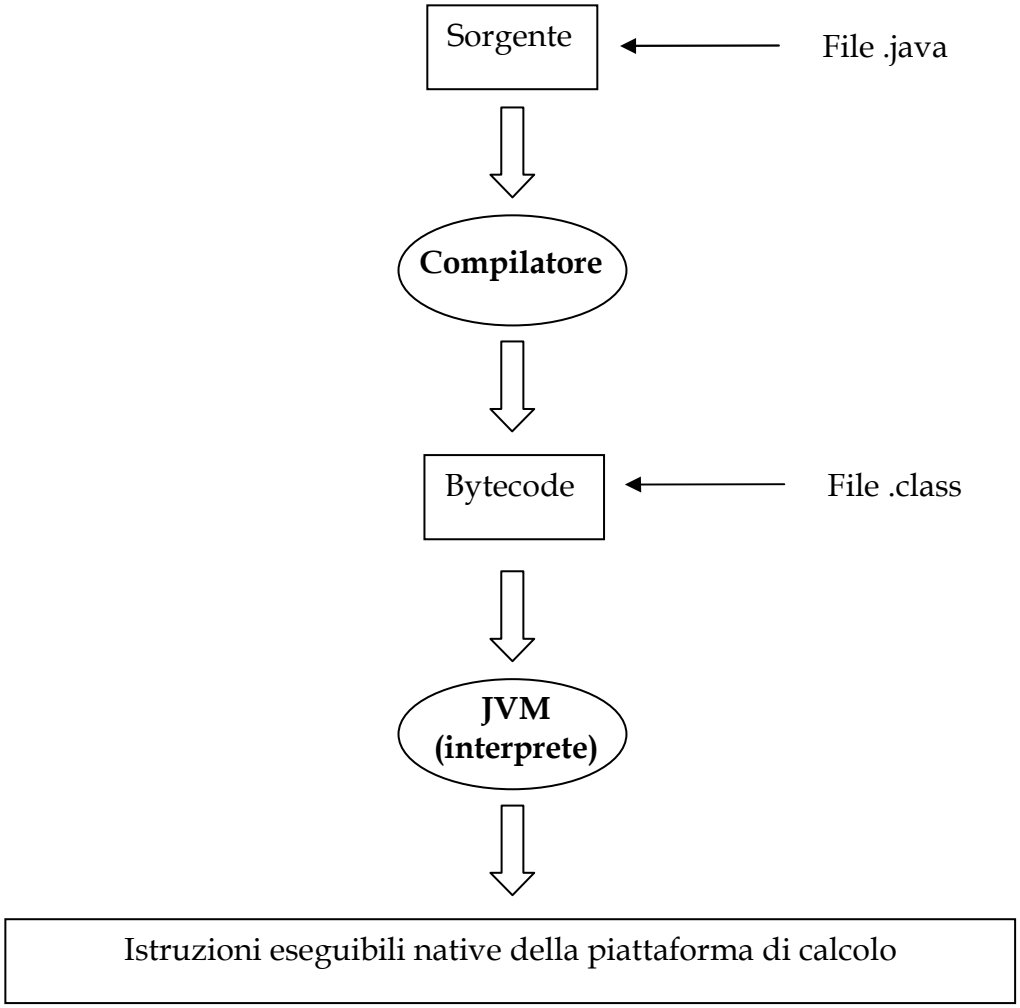
Una classe può estenderne un'altra ed implementare un numero arbitrario di interfacce:

```
public class Alpha extends Letter implements Greek1, Greek2, Greek3 {
      …
      …
}
```

Naturalmente dovranno essere implementati tutti i metodi di tutte le interfacce implementate dalla classe, altrimenti la classe sarà astratta (si veda in seguito) e non si potranno da essa istanziare oggetti.

0.1.2) Indipendenza dalla piattaforma

Java comprende sia l'idea di linguaggio compilato che quella di linguaggio interpretato. Dal codice sorgente infatti il compilatore ottiene un semicompilato (***bytecode***) che necessita di un interprete (***Java Virtual Machine, JVM***) per essere eseguito.



Essenzialmente il codice bytecode è costituito da istruzioni di un linguaggio macchina semplificato, il linguaggio macchina della JVM, che essa si incarica di trasformare in codice nativo eseguibile sulla reale piattaforma di calcolo.

I pregi più evidenti di questa architettura sono la portabilità, sia del codice sorgente che del semicompilato (entrambi sono identici per ogni piattaforma e sistema operativo), e la innata propensione cross-platform (compilazione ed esecuzione possono avvenire su piattaforme completamente diverse). Il che significa, ad esempio, che è possibile sviluppare su un PC Linux quello che poi dovrà girare su un palmare WindowsCE o su un mainframe Unix.

Il limite maggiore, almeno rispetto al C/C++, sta nella velocità di esecuzione, comunque superiore alla maggior parte dei linguaggi puramente interpretati e buona nel caso in cui l'interprete compili in codice nativo porzioni di codice che poi riusa (***Just In Time compiler***).

In Java si possono creare:

- ***applicazioni stand alone***, che devono implementare il metodo `main()`, e, se eseguite su una macchina dotata di JVM, si comportano come delle normali applicazioni C++;
- ***applet***: funzionalità da aggiungere a pagine html all'interno del tag `<APPLET CODE="nomefile.class">`. Generalmente sono scaricate attraverso Internet e vengono eseguite entro la finestra del browser, che deve implementare la JVM (o direttamente o per mezzo di un plug-in);
- ***servlet***: anche queste sono applicazioni pensate per un uso nel web, però il codice viene eseguito dal lato del server web, entro un apposito ambiente (***Servlet container***, ad es. ***Tomcat***).

0.1.3) Le API di Java

L'ambiente di programmazione Java è stato dotato fin dall'inizio di un ricco insieme di classi predefinite, volte a fornire molteplici funzionalità. Tali classi sono ragguppate in ***package***, i principali dei quali sono:

- `java.applet` : funzioni necessarie alla creazione di applet;
- `java.awt` e `javax.swing` : funzioni per la creazione di GUI;
- `java.io` : funzioni di input/output, manipolazione file, ecc.;
- `java.lang` : le classi fondamentali (importate per default);
- `java.math` : funzioni matematiche a precisione arbitraria;
- `java.net` : funzioni di networking;
- `java.sql` : funzioni orientate all'interazione con basi di dati;
- `java.util` : funzioni di utilità comune (random, list, ecc.).

Il vantaggio di questo approccio (che è lo stesso adottato, a suo tempo, nel progetto del linguaggio ***Smalltalk***) è che molte importanti caratteristiche dell'applicazione da sviluppare, come ad esempio l'interfaccia utente, vengono implementate facendo uso di strumenti standard (le opportune classi di uno o più package) e non dovendo ricorrere a strumenti proprietari (ad es. librerie esterne, che possono avere API differenti per differenti piattaforme di calcolo).

Per poter utilizzare tali classi, è necessario ***importarle*** con la direttiva ***import*** posta all'inizio del codice.

```
import java.awt.Frame           //importa la classe Frame del package java.awt
import java.awt.*               //importa tutte le classi del package java.awt

public class Quadrato extends Quadrilatero {
      …
}
```

Il meccanismo di inclusione di codice utilizzato da Java differisce notevolmente da quello del C. Nasce infatti per evitare ambiguità in caso di pubblicazione del codice, e non prevede file di definizione separati (header).

E' possibile definire propri package, con la direttiva ***package*** che deve essere posta nella prima riga non commentata del codice.

Ad esempio, La direttiva

```
package mypackage

public class Quadrato extends Quadrilatero {
      …
      …
}
```

specifica che le classi definite nel sorgente fanno parte della libreria `mypackage`. La classe `Quadrato` definita nello stesso file sarà quindi completamente identificata come `mypackage.Quadrato`, ed identificata in questo modo potrà essere utilizzata all'interno di altro codice.

Le direttive:

```
import mypackage.Quadrato;
import mypackage.*;
```

importano all'interno del codice le funzionalità della classe `Quadrato` (nel primo caso) o dell'intero `mypackage` (nel secondo caso). Funzionano cioè un po' come gli #include del linguaggio C (in realtà però non importano realmente il codice ma definiscono i ***namespace*** necessari al successivo codice).

0.1.4) Esecuzione sicura di codice (Applet) distribuito via rete

Una grande attenzione è stata posta, nel progetto del linguaggio e della JVM, ai problemi di sicurezza derivanti potenzialmente dal fatto di mandare in esecuzione sulla propria macchina un codice proveniente da una fonte ignota (e perciò non affidabile in linea di principio).

Ad esempio, si potrebbero ipotizzare questi scenari certamente indesiderabili:
- un applet cifra tutti i file del disco, e chi lo ha programmato chiede un riscatto per fornire la chiave di decifratura;
- un applet ruba informazioni riservate e le invia a qualcun altro;
- un applet cancella tutti i file del disco.

La prima linea di difesa è stata incorporata nel linguaggio, che è:
- fortemente tipato;
- con controlli sui limiti degli array;
- senza aritmetica dei puntatori.

In tal modo è impossibile accedere a zone di memoria esterne a quelle allocate all'applet.

Tuttavia, ***Trudy*** (un personaggio che conosceremo di più in seguito) si diverte a modificare un compilatore C per produrre dei bytecode in modo da aggirare i controlli effettuati dal compilatore Java.

Per questa ragione, la JVM offre la seconda linea di difesa sotto forma di una componente, detta ***bytecode verifier***, che effettua numerosi controlli sui bytecode prima di mandarli in esecuzione, verificando ad esempio che non si cerchi di:
- manipolare i valori dei puntatori;
- chiamare metodi con parametri non validi;
- usare variabili non inizializzate.

La terza linea di difesa è rappresentata dal ***class loader***, il meccanismo di caricamento delle classi. Esso impedisce, ad esempio, che una classe dell'applet vada a sostituirsi a una delle classi di sistema in modo da aggirare i meccanismi di sicurezza di quest'ultima.

Infine, un'ulteriore linea di difesa è il ***security manager***, una classe che ha il compito di stabilire dei limiti a ciò che il programma (applet o application) può fare.

In particolare, di norma la JVM usata dai client Web carica all'avvio un security manager che impedisce a tutti gli applet di:
- accedere al file system locale;
- aprire connessioni di rete con host diversi da quello di provenienza;
- lanciare altri programmi sulla macchina locale.

0.2) Aspetti sintattici

Dal punto di vista sintattico Java, in particolare per gli aspetti procedurali, è sostanzialmente identico a C e C++. Infatti, rispetto a C e C++ rimane uguale la struttura di:

- commenti;

```
//commento
/* commenti
   su più righe */
```

- istruzioni di assegnamento:

```
a = 5;
```

- condizioni logiche:

```
(a == b) && (c == d)
```

- strutture di controllo (if, for, while, ecc.):

```
if (a == b) {
   c = 1;
} else {
   c = 2;
}
```

- blocco di codice:

```
{
   istruzione1;
   istruzione1;
   …
   istruzioneN;
}
```

Inoltre, il linguaggio è più semplice da usare rispetto a C e C++, poiché ad esempio:

- non esistono gli operatori `*` e `&`;
- sono vietate condizioni logiche "implicite". Ad esempio, se `a` è un intero:

```
if (a)       // illegale
if (a == 0)  // legale
```

0.2.1) Tipi primitivi e non primitivi, reference ed istanze

In Java esistono solamente due tipi di dati:

- tipi di dati primitivi;
- tipi di dati non primitivi (oggetti ed array).

Tipi primitivi

Ogni tipo primitivo ha una dimensione definita, indipendente dalla piattaforma e dal sistema operativo. Per questo motivo in Java non esiste l'operatore `sizeof`.

Tipo	Dimensione
boolean	(true/false)
char	16bit
byte	8bit
short	16bit
int	32bit
long	64bit
float	32bit
double	64bit
void	-

Dichiarare una variabile di un tipo primitivo ha come effetto l'allocazione della memoria necessaria a contenere tale tipo primitivo, quindi:

```
int a; //alloca in memoria 4 byte, il cui contenuto
       //viene riferito per mezzo della variabile a
       //e viene interpretato come valore intero a 32 bit
```

Al momento della allocazione i tipi primitivi vengono automaticamente inizializzati ai valori:

- `false` per i boolean;
- `null` per le reference (si veda nel seguito);
- `0` per tutti gli altri.

Tipi non primitivi

Tutto ciò che non è un tipo primitivo non viene allocato automaticamente, ma deve essere esplicitamente allocato con la direttiva **`new`**

In particolare ciò si applica a:

- oggetti;
- stringhe (anch'esse in realtà sono oggetti);
- array.

Ad esempio, nel caso di una stringa, la dichiarazione:

```
String s;
```

alloca automaticamente solo 4 byte destinati a contenere la ***reference*** ad una stringa, di fatto un puntatore a quella che sarà la locazione di memoria che conterrà la stringa stessa. Il valore della reference dopo tale dichiarazione è ***null***, ossia la reference non punta a nulla.

Quindi, ad esempio, le istruzioni:

```
String s;
s = "abcd";
```

non sono legali: la seconda genera un errore di compilazione poiché si cerca di assegnare una stringa ad una variabile che invece è una reference ad una stringa.

La sequenza corretta di istruzioni è la seguente:

```
String s;               // definisce una reference ad una stringa
s = new String("abcd"); // alloca effettivamente la stringa "abcd"
                        // che viene referenziata dalla variabile s
```

oppure:

```
String s = new String("abcd");
```

Stesso discorso per gli array e gli oggetti:

```
int[] a;            // definisce una reference ad un array di interi, di
                    // dimensione non specificata
a = new int[10];    // alloca effettivamente un array di 10 interi
                    // che viene referenziato dalla variabile a


Account x;          // definisce una reference ad oggetto di tipo Account
x = new Account();  // alloca effettivamente l'oggetto
                    // che viene referenziato dalla variabile x
```

Non esiste il problema di deallocare la memoria assegnata a oggetti non più utilizzati. A questo provvede automaticamente il ***garbage collector***, una apposita componente della JVM.

Inoltre, non è possibile manipolare i valori delle reference ad oggetti, ossia non esiste alcuna forma di aritmetica dei puntatori.

Si noti che la seguente sequenza di istruzioni :

```
Account x, y;
x = new Account();
y = x;
```

ha l'effetto di istanziare **un solo oggetto** di tipo `Account`, a cui fanno da reference **entrambe** le variabili `x` ed `y`. Ossia, `x` ed `y` "puntano" allo stesso oggetto, per cui le modifiche apportate a quell'oggetto referenziandolo per mezzo di `x` saranno visibili anche referenziandolo per mezzo di `y`.

Per inciso, il test `(x == y)` applicato a due variabili reference restituisce `true` se e solo se le due variabili "puntano" allo stesso oggetto, e non va inteso come un test di uguaglianza dei valori contenuti negli oggetti puntati.

0.2.2) Scope

Java supporta le stesse regole di scope (ambito di validità delle variabili) del C e del C++. L'unica differenza è che le variabili a scope limitato non possono nascondere quelle di scope più ampio aventi lo stesso nome.

Ad esempio il seguente codice, lecito in C, è illegale in Java:

```
int x;
{
      int x;
}
```

0.2.3) Passaggio di parametri

Il passaggio di parametri in Java avviene sempre e comunque ***per valore***, cioè viene fatta una copia della variabile e la copia è passata al metodo. Quindi, modifiche al valore passato non influenzano la variabile nel codice che richiama il metodo.

Va però sottolineato il fatto che se la variabile passata è una reference ad un oggetto, la copia che viene fatta è della variabile reference, non dell'oggetto stesso. Quindi le eventuali modifiche all'oggetto effettuate dentro il metodo chiamato saranno visibili anche successivamente all'uscita da tale metodo.

0.2.4) Modificatori di accesso

I ***modificatori di accesso*** regolano l'ambito di visibilità di una classe, di un metodo o di una variabile.

Ne esistono 4:

- **`public`**: si applica a classi, metodi e variabili. Indica che l'entità in questione è visibile e utilizzabile da tutte le classi senza alcuna restrizione;
- **`private`**: si applica a metodi e variabili. Indica che il metodo o la variabile è visibile e utilizzabile solo all'interno della classe cui appartiene;
- **`protected`**: si applica a metodi e variabili. Indica che il metodo o la variabile è visibile e utilizzabile dalle classi dello stesso package ed inoltre dalle sottoclassi di quella stessa classe, anche se appartenenti ad altri package;
- **`package-protected`** (default, nessun modificatore): si applica a classi, metodi e variabili. Indica che l'entità in questione è visibile e utilizzabile da tutte le classi dello stesso package.

0.2.5) Modificatori final, static, abstract

Altri importanti modificatori sono ***`final`***, ***`static`***, ***`abstract`***.

`final`

Il modificatore `final` si applica a variabili, metodi e classi. Presenta le seguenti caratteristiche:

- quando è applicato ad una variabile indica che di essa non si può modificare il valore, quindi che si tratta in effetti di una costante (il cui nome per consuetudine è tutto maiuscolo);
- quando è applicato ad un metodo indica che esso non può essere ridefinito nelle sottoclassi;
- quando è applicato ad una classe indica che da essa non è possibile creare delle sottoclassi.

`static`

Il modificatore `static` si applica a variabili e metodi. Esso indica che l'entità in questione è "di classe", cioè è relativa alla classe e non alle singole istanze degli oggetti di tale classe.

- applicato ad una variabile: rende la variabile **condivisa** e visibile da tutte le istanze di oggetti della classe. Tale variabile può essere riferita senza bisogno di istanziare alcun oggetto della classe;
- applicato ad un metodo: è possibile accedervi senza creare un'istanza della classe. Il metodo `main()`, ad esempio, è statico proprio per questo: poiché è il metodo che avvia il programma, deve essere chiamato quando ancora non esistono istanze di oggetti della classe che lo contiene.

In pratica, le variabili (o i metodi) statici sono un l'equivalente delle variabili (o delle funzioni) globali esistenti negli altri linguaggi di programmazione. I metodi statici sono `final` per default.

Nota: all'interno di codice statico sono vietati riferimenti a entità non statiche, ossia entità di istanza.

`abstract`

Il modificatore `abstract` si applica a metodi e classi:

- applicato ad un metodo: un metodo astratto è un metodo privo di implementazione (come quelli delle interfacce). In pratica viene definito solo il suo prototipo, è compito della sottoclasse implementare correttamente il metodo. Solo le classi astratte e le interfacce possono contenere metodi astratti;
- applicato ad una classe: la qualifica come classe astratta. Le classi astratte non possono essere istanziate, ma possono contenere tutto quanto può contenere una classe normale ed in più metodi astratti. Una sottoclasse che eredita da una classe astratta deve implementare tutti i metodi dichiarati `abstract` altrimenti sarà anche essa classe astratta.

0.3) Utilizzo della piattaforma Java

0.3.1) Cosa scaricare ed installare

Tutti gli strumenti per la programmazione Java sono disponibili per il download gratuito sul sito ufficiale (***http://java.sun.com***). I pacchetti, disponibili per i più diffusi sistemi operativi, sono vari:

- ***Java Standard Edition SD***K (J2SE_SDK): pacchetto indispensabile per la programmazione, contiene l'ambiente di run-time, le API, gli strumenti a riga di comando ed alcuni esempi;
- ***Java Enterprise Edition SDK*** (J2EE_SDK): come il precedente, in più include anche un web server, un application server, ed in generale gli strumenti per la programmazione lato server (non serve nell'ambito di questo corso);
- ***Java Documentation*** (J2SDK_Doc): pacchetto contenente tutta la documentazione necessaria (API e tool) in formato HTML: utilissimo, serve da manuale di riferimento mentre si programma;
- ***NetBeans***: ambiente di sviluppo integrato (IDE) per la programmazione Java, include anche dei tool per la produzione rapida del codice relativo all'interfaccia utente;
- ***Java Runtime Environment*** (J2SE_JRE): piattaforma necessaria all'esecuzione di programmi Java. E' incluso nel pacchetto SDK, e quindi non è necessario sulla macchina di sviluppo.

Dei pacchetti elencati l'unico strettamente necessario è il primo (SDK), oltre ad un editor di testi.

0.3.2) Compilazione ed esecuzione

Per compilare un file sorgente (o meglio una ***compilation-unit***) ***Prova.java*** si deve impartire nella console il comando:

```
javac Prova.java
```

Se non vi sono errori (attenzione a maiuscole e minuscole, Java è case-sensitive), il risultato di tale comando è la generazione di uno o più file di bytecode con estensione ***class***, uno per ogni definizione di classe contenuta nella compilation unit (nel nostro caso almeno il file ***Prova.class***, più eventuali altri).

Nota: ogni classe pubblica deve essere contenuta in un omonimo file con estensione ***java***: la classe pubblica `Prova` deve essere contenuta nel file ***Prova.java***.

Per eseguire la nostra applicazione (nell'ipotesi che la classe `Prova` contenga il metodo `main()`) si deve impartire nella console il comando:

```
java Prova
```

che chiama in causa l'interprete, ossia la JVM. Va notato come alla JVM vada passato il nome della classe (`Prova`) e non il nome del file (***Prova.class***).

0.3.3) Variabili d'ambiente

Vi sono due variabili d'ambiente particolarmente importanti, il cui scorretto valore può essere fonte di problemi.

Variabile d'ambiente `PATH`
Il suo valore deve includere fra gli altri il direttorio ***bin*** del JDK, altrimenti i comandi `java` e `javac` non verranno eseguiti.

Normalmente l'installazione provvede ad aggiornare il valore di tale variabile, ma se non lo fa bisogna aggiornarlo manualmente, o dare i comandi `java` e `javac` corredandoli del path completo, ad es:

```
C:\Programmi\Java\jdk1.6.0_01\bin\javac Prova.java
```

sotto Windows.

Variabile d'ambiente `CLASSPATH`
Indica alla JVM dove reperire le classi aggiuntive, cioé tipicamente le classi sviluppate per l'applicazione che si vuole eseguire.

Di norma `CLASSPATH` include automaticamente i direttori che contengono le classi predefinite della JVM, più ***il direttorio corrente*** (ossia quello da cui si impartisce il comando `java`).

L'eventuale, tipico, errore `NoClassDefFoundError` è quasi sempre causato da un non corretto valore del `CLASSPATH`.

In luogo della variabile d'ambiente è anche possibile specificare i path aggiuntivi direttamente nel comando `java`:

```
java -classpath \path\a\classi\aggiunte\ Prova
```

0.3.3) Un esempio

Il seguente codice, compilato ed eseguito,

```
/*
      G. Bongiovanni - Corso di Programmazione di Rete
      Esercitazione n. 0

      Esempio di interfaccia utente (assolutamente inutile!)
*/

import java.awt.*;

public class UselessApp extends Frame    {

      int a, b, c, d;
      String x, y, z;
      Button button1;

//-------------------------------------------------

      public UselessApp() {

            this.setLayout(null);

            String x = new String();
            x = "Ciao";

            Button button1 = new Button(x);
            button1.reshape(270, 220, 60, 20);
            this.add(button1);

            resize(600, 460);
            show();
      }

//-------------------------------------------------

      public static void main(String args[]) {

            new UselessApp();
      }

}
```

produce un'applicazione che mostra una finestra contenente un bottone: sia la finestra che il bottone reagiscono a molti eventi standard, grazie alle funzionalità predefinite nelle classi `Frame` e `Button` offerte dalle API Java.

Il risultato visivo è il seguente (si noti il ***look and feel*** del sottostante sistema operativo):