Università di Roma "La Sapienza", Facoltà di Ingegneria

Corso di

# Progettazione del Software

Corso di Laurea in Ingegneria Gestionale

Prof. Toni Mancini & Prof. Monica Scannapieco

## AUTOV.Java.1
## Nozioni Preliminari di Programmazione e Java

Versione del 3 gennaio 2007

# Premessa

Cosa mi serve per proseguire?

- Il Java Development Kit (JDK)
- La Java API Documentation

Nella pagina del sito web del corso dedicata al materiale didattico, sono presenti tutti i link utili.

**Nota:** Per avere successo in questo corso non basta leggere il codice, bisogna scrivere codice, compilarlo ed eseguirlo.

# Usare il JDK

- La directory di installazione contiene una sottodirectory "bin" con i seguenti strumenti:
  - **javac** Compilatore Java
  - **java** Interprete java (Java Virtual Machine)

- Per poter utilizzare questi comandi da una qualsiasi directory (attraverso una finestra di comandi) bisogna impostare la variabile d'ambiente **Path** aggiungendole il percorso della directory bin:
  - Windows: da pannello di controllo-Sistema-Avanzate
  - UNIX: usare il comando setenv

# Compilazione ed Esecuzione

- per compilare il file MioProgramma.java che si trova nella directory corrente:
  - javac MioProgramma.java

- Per eseguire il file compilato (MioProgramma.class):
  - java MioProgramma (**Senza** l'estensione .class)

# Primo Programma

"MyFirstJavaApp"

# Progetti

- Directory nomeprogetto con directory src, bin, lib
- Compilare il progetto
  - javac -classpath lib -d bin src/*.java
- Eseguire il progetto:
  - java -classpath bin;lib MiaClasse
- Nota: l'opzione -classpath indica directory e file .zip/.jar in cui il compilatore e l'interprete possono cercare file .class se non li trovano nella directory corrente. Nel caso del compilatore, questo serve solo per includere clasi di libreria precompilate (presenti nella nostra directory lib). I percorsi vanno separati con ;

# Librerie di uso frequente

- Per includere directory nella classpath senza usare l'opzione -classpath della linea di comando ogni volta, bisogna impostare la variabile d'ambiente denominata CLASSPATH. Questo è utile per classi che non appartengono alla distribuzione standard, ma si usano spesso in diversi progetti.

- Se è definita la variabile classpath occorre aggiungere
  - javac -classpath %CLASSPATH%;lib -d bin src/*.java (Windows)
  - javac -classpath $CLASSPATH;lib -d bin src/*.java (Unix/Unix-like)

# Cenni sugli editor

- Vantaggi: Highlight della sintassi, supporto all'indentazione a blocchi, numerazione delle righe, compilazione ed esecuzione integrate, gestione di progetti, completamento automatico della sintassi...

- Svantaggi(?): Una cosa in più da imparare...

- Due tool freeware

  - **Tera Text Editor**: highlight, indentazione, numerazione delle righe... (http://www.terasoft.ru/en)

  - **JCreator LE**: molto completo, richiede un po' di tempo per capire come funzionano i progetti, impostare le opzioni etc. (http://www.jcreator.com/download.htm)

# Variabili

- Nei linguaggi di programmazione si fa riferimento ai dati che i programmi devono manipolare tramite nomi simbolici detti **variabili**.

- Java distingue vari *tipi di dato*. Esempi di tipo di dato sono il tipo int, che rappresenta valori interi (a 32 bit) ed il tipo boolean, che rappresenta valori booleani. Prima di essere usate, le variabili devono essere dichiarate e se ne deve specificare il tipo. Es:

  – int miaVariabile; dichiara una variabile di tipo int

- i tipi di Java si possono distinguere

  – tipi primitivi

  – tipi per riferimento

- Per ora ci limiteremo a considerare i soli tipi primitivi, e approfondiremo in seguito i tipi per riferimento e le caratteristiche delle variabili di tali tipi. Nel caso dei tipi primitivi, il concetto di variabile si può accostare metaforicamente a quello di una scatola che contiene un certo valore, che può essere rimpiazzato tramite un'opportuna operazione.

- Contestualmente alla loro dichiarazione, o in seguito, alle variabili può essere assegnato un valore del tipo appropriato. Es:

  - int miaVariabile=10;

- Notare che è necessario che ad una variabile sia stato assegnato almeno una volta un valore prima di poterla utilizzare (cioè di poter leggere il valore che contiene).

# Tipi primitivi

- tipi numerici interi
  - byte 8 bit
  - short 16 bit
  - int 32 bit
  - long 64 bit
- tipi numerici decimali
  - float 32 bit
  - double 64 bit

- altri tipi
  - char singolo carattere Unicode(16 bit)
  - boolean true/false

# Espressioni

- I valori, siano essi indicati direttamente come costanti oppure contenuti nelle variabili, possono essere utilizzati per ottenere altri valori formando, tramite opportuni operatori, delle espressioni.

- Anche le espressioni seguono regole sui tipi. Il risultato della valutazione di un'espressione ha sempre un valore ed un tipo.

- Un'operatore fondamentale è quello di assegnazione, che assegna ad una variabile il valore di un'espressione.

# Operatori aritmetici

- +, *, /, % (modulo: resto nella divisione intera)

- sono sovraccarichi, cioè accettano operandi di più tipi(numerici). In alcuni casi si comportano diversamente a seconda del tipo (es: / è la divisione intera se entrambi gli operandi sono interi, altrimenti è la divisione in virgola mobile). Il tipo restituito dipende dal tipo degli operandi.

- durante la valutazione delle espressioni, ed in particolare durante l'assegnazione, possono avvenire alcune conversioni di tipo implicite. Intuitivamente, si passa sempre da un tipo a minore precisione ad uno a maggiore precisione. Questo è vero anche nell'inizializzazione di variabili

- tuttavia se si usa una costante per inizializzare una variabile di un tipo intero, il compilatore può svolgere un check statico sul range del valore,

e quindi permettere anche assegnazioni da tipi che hanno precisione maggiore a tipi con minore precisione. Es: byte b=100; La costante 100 è intera, ma rientra nel range del tipo byte.

- precedenza fra operatori aritmetici: usate sempre le parentesi, il codice guadagna moltissimo in leggibilità es: 10+(10/100) invece di 10+10/100

- incremento e decremento

  - var++ (var- -) incrementa(decrementa) var di 1

  - ++var (- -var) incrementa(decrementa) var di 1

# operatori di relazione

- $>, <, >=, <=$, si applicano a espressioni che hanno tipo numerico.
- ==, != si applicano a espressioni di qualsiasi tipo.
- Le espressioni formate con questi operatori hanno tipo booleano.

# operatori condizionali

- & (AND), | (OR), ! (NOT), ^ (XOR)

- (esistono anche && e ||, che valutano il secondo operando solo se necessario)

- Si applicano a espressioni di tipo booleani e formano espressioni di tipo booleano.

# Istruzioni

- il programma Java è formato da una sequenza di istruzioni o statement. Ciascuna istruzione ha lo scopo di dichiarare o modificare il valore di una variabile (assegnazioni), oppure controllare il flusso del programma.

- gli statement sono terminati da ;

- Un'espressione in generale non è un'istruzione, ma alcune espressioni lo sono (assegnazioni e ++/- -)

# Istruzioni Condizionali: if

- if(condizione)istruzione;

- condizione deve essere una espressione di tipo booleano. Es:

```
if(miaVariabile>6) tuaVariabile=0;
```

- invece di un'istruzione si può usare un blocco di istruzioni

```
if(miaVariabile>6){
    tuaVariabile=0;
    suaVariabile=5;
}
```

# if..else

- if può essere seguito da un else istruzione (o blocco istruzioni):

```
if(miaVariabile>6){
    tuaVariabile=0;
}
else{
    tuaVariabile=9;
}
```

- Il ramo else viene eseguito alternativamente al ramo if

# Scope delle variabili

- Si possono dichiarare variabili internamente a blocchi di istruzioni. Una variabile dichiarata in un blocco è locale rispetto al blocco e non è visibile dall'esterno del blocco stesso.

```
if(b==true){
    int i=0;
    ...
} i=5;  //no! i è definita (e dunque esiste) solo internamente al
blocco
```

# Istruzioni Condizionali: switch

```
switch(espressione){
    case 0: sequenza istruzioni; break;
    case 5: sequenza istruzioni; break;
    ...
    default: sequenza istruzioni; break;
}
```

- esegue condizionalmente una sequenza di espressioni in base al valore dell'espressione, che deve essere di tipo intero o char. Le etichette sono delle costanti.

# Istruzioni di ciclo: while

```
while(condizione)
    istruzione;
```

- l'istruzione può essere un blocco

- la condizione è booleana. L'istruzione viene eseguita ripetutamente finchè la condizione resta vera. Quindi:

  ```
  while(true){...} //è un ciclo infinito;
  ```

- variante: `do istruzione while(condizione);` L'istruzione viene eseguita sempre almeno una volta.

# Istruzioni di ciclo: for

- E' una forma abbreviata per scrivere cicli while di uso frequente, come ad esempio quelli che ciclano n volte.

- for(inizializzazione; condizione; istruzione) istruzione (o blocco); es: `for(i=0; i<100; i++) System.out.println(‘‘ciao’’)`

- nella sezione inizializzazione la variabile può essere anche dichiarata. In questo caso è locale rispetto al blocco istruzioni.

# Array

- Gli array sono liste ordinate di variabili, ciascuna identificata da un indice.

- Gli array non sono tipi primitivi, ma per riferimento. Le variabili array non contengono un valore ma un riferimento ad una locazione di memoria (che contiene l'array vero e proprio).

```
int[] mioArray; //dichiarazione
//allocazione della memoria e inizializzazione della variabile
mioArray=new int[10];
//variabile length e accesso a componenti
for(int i=0; i<mioArray.length;i++){
    System.out.println(mioArray[i]);
}
```