Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica

Programmazione I - corso B
a.a. 2009-10

prof. Viviana Bono

Blocco 12 – Riepilogo e complementi sui tipi

Ripasso del sistema di tipi di Java

In Java, ogni valore è di un qualche tipo (o anche di più tipi, come un cavallo è un ente di tipo animale ma anche di tipo mammifero, o un quadrato è una figura di tipo rombo ma anche di tipo rettangolo)

I tipi di Java

- tipi primitivi:
  - boolean 1 bit true, false
  - tipi numerici
    - tipi interi
      - con segno:
        - byte 8 bit da -128 a 127
        - short 16 bit da -32768 a 32767
        - int 32 bit da ~ -2 miliardi a ~ 2 miliardi
        - long 64 bit ... miliardi di miliardi ...
      - assoluti:
        - char 16 bit da 0 a 65535
    - tipi con virgola mobile (floating point)
      - float 32 bit ~ 7 cifre decimali significative
      - double 64 bit ~ 15 cifre decimali signific.

- tipi di riferimenti a oggetti:
  - tipi di riferimenti a oggetti di una data classe, o tipi-classe:
    - String (classe fornita col linguaggio)
    - JOptionPane (classe fornita con il JDK)
    - Conto (classe definita dal programmatore)
    - Dipendente (classe definita dal programmatore)
    - ...
  - tipi di riferimenti ad array, o tipi-array:
    - int[] (array di valori di un tipo primitivo)
    - double[] (array di valori di un tipo primitivo)
    - String[] (array di oggetti di una data classe)
    - Conto[] (array di oggetti di una data classe)
    - ...

Esiste un terzo genere di tipi di riferimenti a oggetti,
i tipi-interfaccia, che non studieremo in questo corso.

Quindi, riassumendo, i tipi di Java sono:

- tipi primitivi:
  - boolean
  - tipi numerici

- tipi di riferimenti a oggetti:
  - tipi di riferimenti a oggetti di una data classe, o tipi-classe
  - tipi di riferimenti ad array di elementi di un dato tipo (**ricorda: gli array sono oggetti, quindi una variabile di tipo array contiene un indirizzo**)
  - tipi di riferimenti a oggetti di una data interfaccia

NOTA BENE

I nomi di classe servono sia da nomi di programma che da tipi;
un programma è (la realizzazione di) un tipo.

Riferimenti a oggetti di una data classe

Se *UnaClasse* è il nome di una classe predefinita oppure definita dal programmatore, la valutazione di un'espressione

```
new UnaClasse(...)
```

dove *UnaClasse(...)* sia una corretta invocazione di un costruttore, crea nello heap un oggetto e restituisce il riferimento a tale oggetto (cioè il suo indirizzo).

Tale riferimento potrà essere memorizzato in una opportuna variabile, oppure passato come argomento ad un opportuno metodo, ecc. In generale, un'espressione *new* si troverà quindi in istruzioni della forma:

```
UnaClasse unOggetto;
...
unOggetto = new UnaClasse(...);
```


Riferimento nullo

Fra i particolari valori di tipo riferimento a oggetto vi è il riferimento nullo, indicato dalla costante *null* (corrispondente al *nil* del Pascal).

Importante! Il riferimento nullo NON è il riferimento ad un oggetto nullo (non esistono oggetti nulli), bensì un valore che significa "nessun riferimento", e che fisicamente è realizzata dal numero ... provate a indovinare ... sì, proprio dal numero zero.

Quindi una variabile di tipo riferimento a oggetto, se contiene il valore *null*, è una variabile che in quell'istante non si riferisce ad alcun oggetto. Ogni tentativo di accedere a campi o metodi dell'oggetto da essa puntato genera un errore durante l'esecuzione, la Null Pointer Exception (incontrerete le *eccezioni* piú avanti).

La traduzione italiana di *null reference* come *riferimento a nullo*, che si trova in alcuni libri, è pertanto ERRATA.

Ulteriori osservazioni sui tipi e su null

Un tipo, in un linguaggio di programmazione, può essere pensato come un insieme. Gli elementi di un tale insieme, che sono degli enti astratti, sono i **valori** di quel tipo. Naturalmente tali enti astratti hanno, nella memoria del calcolatore, rappresentazioni concrete sotto forma di sequenze di bit.

Nel linguaggio, nel nostro caso in Java, tali enti astratti sono rappresentati da costanti numeriche come 5, -21, 3.14, ecc., oppure da parole riservate come *true* e *false*.

Attenzione: ricordare che non solo i numeri (interi, doubles, ecc.) sono valori; anche i booleani sono valori esattamente come i valori numerici; semplicemente non appartengono a un tipo numerico ma al tipo **boolean**.

Analogamente, ***null*** è (il nome di) un valore esattamente come 5, true, o false. Semplicemente, esso non appartiene né a un tipo numerico né al tipo boolean, bensì al tipo "riferimento a oggetto".

Più precisamente, esso appartiene a tutti i tipi "riferimento a oggetto" (cioè riferimento ad oggetto di qualunque classe, o anche riferimento ad array).

Si noti che ***null*** è l'unico valore di tale tipo che può essere scritto esplicitamente nel linguaggio: in un programma Java non possiamo scrivere esplicitamente l'indirizzo di un oggetto!

Il tipo char

Un carattere, in Java, è semplicemente un numero intero assoluto compreso fra 0 e 65535, il quale quando viene visualizzato per mezzo di opportuni metodi (ad es. System.out.println) viene scritto sotto forma di carattere.

Ai caratteri si applicano quindi tutte le ordinarie operazioni numeriche, in particolare gli operatori di confronto <, <=, >, ==, ecc.

Si possono anche fare somme, sottrazioni, moltiplicazioni, ecc. di caratteri, ma ovviamente non hanno molto senso!

Esempio:
le due assegnazioni `char c = 'a';` `char c = 97;`
hanno esattamente lo stesso effetto (ma la seconda è cattivo stile !)

```
c = c + 3;
System.out.println(c);   // scrive il carattere 'd';
```

naturalmente è un uso sconsigliato !

Esempio di uso degli operatori di confronto:

```
String cognome = tastiera.next();
char iniziale = cognome.charAt(0);
if(iniziale >= A && iniziale < M)
 System.out.println("cognome appartenente al corso A");
```

Il tipo String

• Il tipo **`String`** è un tipo di riferimento a oggetto, infatti String è una classe predefinita di Java

• Una stringa è “praticamente” un array di caratteri

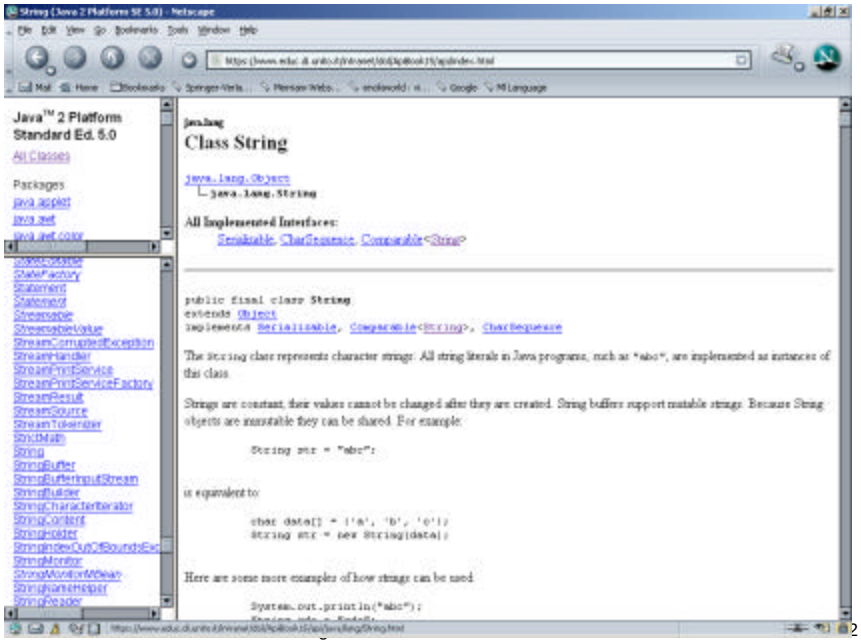
Per esempio:

```
String str = "abc";
```

è equivalente a:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```


La classe String



Java: variabili

Le variabili possono essere:

- **variabili di classe (variabili "globali",campi static)**
- **variabili d'istanza (campi)**, cioè "scatole", (strutture-dati) in cui ogni istanza di quella classe conserva i dati necessari ad eseguire le operazioni definite nella sua interfaccia
- **variabili locali** ai metodi

Dichiarazioni di variabili locali di un metodo

Hanno la stessa forma delle dichiarazioni dei campi static, ma senza il modificatore static; esse creano dei contenitori utilizzabili dal metodo. Una dichiarazione di variabile locale, poiché fa parte del corpo del metodo, viene eseguita soltanto quando il metodo viene invocato: le variabili locali pertanto non esistono e non sono visibili al di fuori del metodo.

Esempio:

```
static int myFun(int x, int y) {
  int t = 3*x - 4*y;
  return t*t*t;
}
public static void main(String args) {
  System.out.println(t); // ERRORE !
```

Java: classi come generatori di oggetti

Scrivere un **programma** in Java significa:

Definire un certo numero di **classi**

Per ogni classe, definire

1. Le **proprietà** della classe, cioè le **strutture-dati** necessarie, cioè un insieme di **campi** (normalmente privati) che sono lo **stato** dell'oggetto
2. Uno o più **costruttori**, cioè meccanismi per inizializzare istanze (oggetti) di quella classe
3. Uno o più **metodi pubblici**, cioè servizi, operazioni offerte dalla classe sullo stato
4. Eventualmente campi e metodi static

L'insieme dei costruttori e dei metodi pubblic rappresenta l'**interfaccia** della classe.

Java: classi come contenitori di metodi

Scrivere un **programma** in Java significa:

Definire un certo numero di **classi**

Per ogni classe, definire

1. Uno o più metodi static pubblici
2. Eventualmente variabili (campi) static

L'insieme dei metodi pubblic rappresenta l'**interfaccia** della classe.

Ricordatevi! (I)

Java e` il linguaggio (inteso come sintassi, ambiente di programmazione, librerie, ecc.): noi scriviamo codice in Java

javac è il programma compilatore: traduce il file .java nel corrispondente file .class (che contiene il programma tradotto in bytecode)

java è il programma inteprete: traduce il bytecode istruzione per istruzione e le fa eseguire alla "macchina"

Il bytecode definisce la macchina virtuale Java (JVM = Java Virtual Machine) : architettura a livelli del calcolatore = 1 macchina fisica (bottom) + n macchine virtuali

Ricordatevi! (II)

Il file .class che si da' in input all'interprete java deve contenere una classe (**`class`**) che contenga un **`main`** (dichiarato ***esattamente*** come abbiamo visto negli esempi)

Un programma Java può essere composto da piu' classi, deve però esserci una sola classe contenente il main (quella da cui parte l'esecuzione dell'interprete java)

In effetti abbiamo gia` visto programmi che usano piu' classi (magari organizzate in package), anche se non scritte da noi (come la System per l'output e la Scanner per l'input)