

PROGRAMMAZIONE 1

LEZIONE 1

Com'è fatto un computer

Il modo migliore per capire il funzionamento di un computer è smontarlo, aprendo il case troveremo:

- Lettore DVD/Masterizzatore (opzionale)
- Scheda madre
- Hard disk
- Alimentatore
- Cavi di alimentazione
- Cavi vari (led, altoparlanti, etc...)
- Cavi dati

Possiamo quindi rimuovere lettore DVD e hard disk. Fatto questo passiamo alla scheda madre e ai suoi componenti, rimuoviamo in ordine i seguenti componenti: - Scheda video (GPU)

- Memoria RAM
- Dissipatore (sotto di esso troveremo la CPU)
- CPU

E infine rimuoviamo la scheda madre. Questi componenti sono collegati tramite BUS di comunicazione gestiti dal processore e da un apposito chipset.

Architettura Von Neumann

- Prima proposta di architettura di un elaboratore
- Risale al 1946
- Schema generale di tipo funzionale del calcolatore

Unità di elaborazione o CPU:

- acquista, interpreta ed esegue istruzioni

Memoria centrale:

- contiene istruzioni e dati

Periferiche:

- permettono lo scambio di informazioni da e con l'esterno

Bus di sistema:

- collega i vari elementi del calcolatore

La CPU e la memoria RAM

I componenti fondamentali per l'esecuzione di un programma sono la CPU e la RAM.

- La **CPU** è un chip capace di eseguire operazioni molto semplici in tempi rapidissimi, nell'ordine dei miliardi di operazioni al secondo .
- La **memoria RAM** è una memoria volatile (si cancella quando non alimentata) che consente di memorizzare grandi quantità di dati rappresentati in formato binario

La memoria RAM

Un modulo di RAM può memorizzare miliardi di bit 0/1.

- I bit sono organizzati in gruppi di 8, detti **byte**
- I byte sono organizzati in gruppi di 4 (32 bit) oppure 8 (64 bit) detti **word** (o **parola**)

Ogni singolo byte è associato ad un unico **indirizzo di memoria** che lo distingue dagli altri.

Una parola è la quantità **massima** di memoria che una CPU può utilizzare in una singola operazione (solitamente). Operazioni che utilizzano dati più grandi vengono svolte in più passi dalla CPU.

La CPU

Le operazioni eseguibili dalla CPU sono codificate tramite un linguaggio in codice binario, il **linguaggio macchina**. Questo linguaggio macchina è solitamente rappresentato tramite una notazione simbolica detta **linguaggio assembly**.

Rappresentazione binaria dell'informazione

Per informazione intendiamo tutto quello che viene manipolato da un calcolatore: numeri, caratteri, immagini, suoni, programmi...

La più piccola unità di informazione memorizzabile/elaborabile da un calcolatore, il **bit**, corrisponde allo stato di un dispositivo fisico (es. on/off) che viene interpretato come 0 o 1. In un calcolatore tutte le informazioni sono rappresentate in forma **binaria**, come sequenze di 0 e 1. Ciò viene fatto per vari **motivi tecnologici**, uno dei più importanti è la **facilità**.

Rappresentazione posizionale

Un numero naturale può essere rappresentato mediante una **sequenza di simboli** in diversi modi, prendiamo ad esempio il numero 234:

- La **sequenza di cifre** "234" è la rappresentazione decimale del numero 234
- La **sequenza di cifre romane** "CCXXXIV" è un'altra rappresentazione dello stesso numero

La rappresentazione decimale è un esempio di rappresentazione **posizionale**:

- Ogni cifra contribuisce con un valore basato sulla posizione in cui essa si trova • Es. "2426" il primo due contribuisce di più rispetto al secondo (2000 vs 20)

Prendiamo come esempio la sequenza di cifre: $c_{n-1} c_{n-2} \dots c_1 c_0$

- La cifra C_0 viene detta cifra **meno significativa**
- La cifra C_{n-1} viene detta cifra **più significativa**

Ad esempio, nella rappresentazione "2435":

- la cifra **meno significativa** è 5
- la cifra **più significativa** è 2

Rappresentazione di numeri naturali

La rappresentazione decimale si basa su un insieme di cifre costituito da 10 simboli (0 - 9).

Il numero **b** di cifre usate è detto **base**. Ad ogni cifra è associato un valore tra 0 e $b-1$.

Base	Cifre	Sistema Esempio
2	0, 1	1001010110

8	0,1,..7	40367
10	0, 1, ..., 9	3954
16	0, 1, ..., 9, A, ..., F	2DE4

Nel sistema esadecimale **A vale 10, B vale 11, ..., F vale 15.**

Il numero N rappresentato dalla sequenza di cifre $c_{n-1} c_{n-2} \dots c_1 c_0$ **dipende dalla base B** e si ottiene tramite la seguente formula:

$$N = c_{n-1} * b^{n-1} + c_{n-2} * b^{n-2} + \dots + c_1 * b^1 + c_0 * b^0$$

Ad esempio:

- Il numero binario 1101 corrisponde a $1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13$
- Per la rappresentazione ottale si moltiplica per 8
- Per la rappresentazione decimale si moltiplica per 10
- Per la rappresentazione esadecimale si moltiplica per 16

Rappresentazione binaria di numeri naturali

Per convertire un numero binario in decimale si può costruire una semplice tabella:

Numero da convertire 10110101

Numero da convertire	1	0	1	1	0	1	0	1
Potenze di 2 corrispondenti	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Il risultato è la somma delle potenze di 2 associate ai valori 1 nella tabellina: 181

Per convertire un numero decimale in binario basterà dividere il numero per 2.

Il numero binario sarà dato dai **resti delle divisioni** presi in ordine inverso.

Numero da convertire: 25 .

N : 2	Quoziente	Resto Cifra
25 : 2	12	1 C0
12 : 2	6	0 C1
6 : 2	3	0 C2
3 : 2	1	1 C3
1 : 2	0	1 C4

Il risultato è quindi 11001.

Intervallo di rappresentazione

Nel sistema binario con **n** cifre si possono rappresentare **2^n** valori

diversi: • I valori vanno **da 0 a $2^n - 1$**

- Con 1 bit si possono rappresentare 2^1 valori
- Con 8 bit (1 byte) si possono rappresentare 2^8 valori
- Con 16 bit (2 byte) si possono rappresentare 2^{16} valori
- Con 32 bit (3 byte) si possono rappresentare 2^{32} valori

Rappresentazione binaria di numeri interi

Quando si considerano valori interi, per rappresentare il **segno** si può pensare di usare uno dei bit (quello più significativo), questa convenzione viene detta rappresentazione tramite **modulo e segno**:

- Il bit più significativo rappresenta il segno (0 equivale a +, 1 equivale a -).
- Gli altri bit rappresentano il valore assoluto

Esempio: $+7 = 0111$ $-7 = 1111$

Questa rappresentazione però presenta diversi **problemi**:

- Doppia rappresentazione dello zero (come 00 oppure 10).
- Le operazioni aritmetiche diventano complicate.

Rappresentazione in complemento a 2

Una rappresentazione alternativa è la rappresentazione **in complemento a 2**. Come prima si usa il bit più significativo per rappresentare il **segno**. Anche la rappresentazione dei **numeri positivi** non cambia.

Un numero negativo **-N** invece viene rappresentato come **$2^n - N$** , dove n è il numero dei bit a disposizione.

Ad esempio con 4 bit (**n = 4**) rappresentiamo il numero negativo -3 (**N = 3**) come **$2^4 - 3 = 16$** , $16 - 3 = 13$, 13 in binario equivale a 1101.

Vantaggi rappresentazione in **complemento a 2**:

- Lo zero ha una sola rappresentazione (00).
- Le operazioni aritmetiche risultano facili:
 - Si può trasformare un numero positivo in negativo
 - La somma rimane invariata (bit a bit)
 - La sottrazione si ottiene trasformando il secondo operando in negativo e sommandolo
 - Moltiplicazione e divisione si basano su somma e sottrazione

Per trasformare un numero positivo in negativo è sufficiente invertire tutti i bit e sommare 1.

Esempio:

- 5 è 0101
- inverti tutti i bit: 1010

- sommo uno: 1011
- quello che ottengo è -5

Rappresentazione di numeri reali

Anche in un intervallo chiuso, i numeri reali sono infiniti, quindi è impossibile rappresentarli tutti.

Ci vengono in aiuto due rappresentazioni:

- Virgola fissa
- Virgola mobile

Rappresentazione in **virgola fissa** (poco usata):

- Si rappresentano separatamente, usando un numero di cifre fissato, la **parte intera** e la **parte frazionaria**.
- Ad esempio:
 - Usando 8 bit, 4 parte frazionaria e 4 parte intera
 - 5.75 in binario diventa 0101.1100
 - Per la parte frazionaria: $1 * 2^{-1} + 1 * 2^{-2} + 0 * 2^{-3} + 0 * 2^{-4} = 0.75$

Rappresentazione in **virgola mobile**:

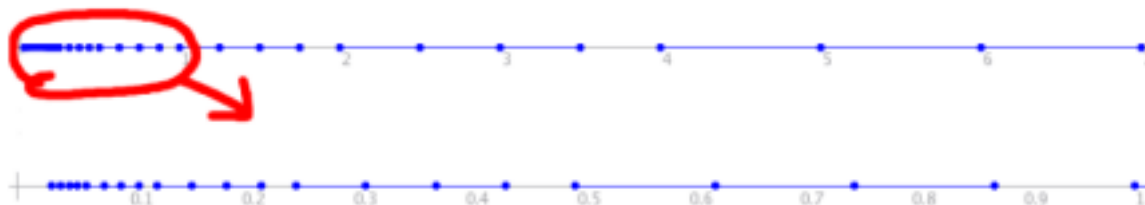
- Per rappresentare un numero reale N usa **notazione esponenziale**: $N = m * 2^e$
dove:
 - m è la **mantissa** in base 2: numero frazionario tale che: $0.5 \leq |m| < 1$ -
 - e è l'**esponente** in base 2: numero intero
- Esempio: per rappresentare 0.125 avremo
 - $m = 0.5$ (in binario $0.1 = 1 * 2^{-1}$)
 - $e = -2$ (in binario con 4 bit 1110)
 - infatti: $0.5 * 2^{-2} = 0.125$

I numeri rappresentati in virgola mobile:

- Sono distribuiti simmetricamente rispetto allo 0
- **NON** sono uniformemente distribuiti sull'asse reale
 - sono più densi intorno allo 0!
- Ad esempio, con 3 bit per la mantissa e 3 bit per l'esponente:

m \ e	-4	-3	-2	-1	0	1	2	3
0.100	0.03125	0.0625	0.125	0.25	0.5	1	2	4
0.101	0.0390625	0.078125	0.15626	0.3125	0.625	1.25	2.5	5
0.110	0.046875	0.09375	0.1875	0.375	0.75	1.5	3	6
0.111	0.0546875	0.109375	0.21875	0.4375	0.875	1.75	3.5	7

Graficamente (i punti sono numeri rappresentabili):



Problemi della rappresentazione in virgola mobile:

- Molti razionali non sono rappresentabili (ad esempio $\frac{1}{3}$)
- Non è chiuso rispetto ad addizioni e moltiplicazioni (possiamo avere due numeri perfetti che una volta moltiplicati o sommati non diano un numero perfetto)
- Per rappresentare un reale N si sceglie l'elemento rappresentabile **più vicino** ad N. ●

Perdita di precisione:

- **Arrotondamento**: mantissa non sufficiente a rappresentare tutte le cifre significative del numero
- **Errore di overflow**: esponente non sufficiente (numero troppo grande) -

Errore di underflow: numero troppo piccolo (rappresentato come 0)

Rappresentazione di caratteri e testi

I singoli caratteri di un testo possono essere rappresentati come numeri. La codifica **ASCII** è nata negli anni 60 per rappresentare con **7 bit** i simboli della tastiera americana.

Questa codifica è stata superata negli anni da varianti più ricche. In primis, la codifica **Extended-ASCII (8 bit)** o **UTF-8**:

- Usa da 1 a 4 byte per rappresentare un carattere

Le **stringhe** sono sequenze di caratteri che possono contenere un testo, possono essere rappresentate da una sequenza di caratteri terminata dal carattere **NUL** (= numero 0).

Rappresentazione di immagini e altri dati

Un'immagine può essere rappresentata nella memoria di un pc codificando **il colore dei singoli pixel** che la compongono tramite valori numerici.

Ad esempio, nel formato bmp a 24 bit ogni pixel dell'immagine viene rappresentato da 3 byte:

- ogni byte (256 valori) rappresenta il livello di un colore fondamentale RGB Altri formati più sofisticati (gif, png, jpeg) utilizzano metodi di compressione per ridurre la dimensione della rappresentazione. Approcci simili sono adottati per rappresentare suoni e filmati...

Problemi computazionali

L'informatica è una scienza che studia metodi e strumenti per la risoluzione di problemi computazionali (**problem solving**).

Un **problema computazionale** è un problema che richiede:

- di calcolare un risultato (**output**)
- a partire da determinati valori noti (**input**)

Esempi:

- Calcolo massimo comune divisore di due numeri
- Preparazione di un risotto ai funghi
- Ordinamento di una sequenza di numeri
- Etc...

Algoritmi

I problemi computazionali possono essere risolti tramite algoritmi.

Un **algoritmo** è una sequenza finita di passi di elaborazione che, dato un input, consentono di ottenere l'output ad esso corrispondente.

Un algoritmo si può esprimere in molti modi diversi:

- In linguaggio naturale (es. “testo del problema”)
- Tramite formule matematiche (es. “ $\text{mcd}(X, Y)$ ”)
- In **pseudo-codice** (es. “finchè $X \neq Y$ ripeti {etc...}”)

Un buon algoritmo deve soddisfare alcune **proprietà**:

- **Non ambiguità**: I singoli passi devono essere elementari, facilmente eseguibili e non ambigui
- **Determinismo**: Eseguito più volte dallo stesso input, l'algoritmo deve eseguire sempre la stessa sequenza di passi (quindi dare anche lo stesso risultato)
- **Terminazione**: L'esecuzione dell'algoritmo deve prima o poi terminare e fornire un risultato

Dagli algoritmi ai programmi

Per “dare in pasto” un algoritmo al computer esso deve essere scritto in **linguaggio macchina**, cosa difficile, per questo ci vengono in aiuto i **linguaggi di programmazione**.

I linguaggi di programmazione

Un **linguaggio di programmazione** è un “linguaggio formale (alto livello)” che consente di scrivere programmi che realizzano algoritmi:

- Un **linguaggio formale** è un linguaggio con regole sintattiche e semantiche ben precise che rendono i costrutti del linguaggio stesso **privi di ambiguità**
- I programmi potranno poi essere **tradotti** in linguaggio macchina per essere eseguiti
- Seguendo regole precise, la traduzione può essere effettuata da un altro programma

Terminologia:

- I linguaggi di programmazione sono detti **linguaggi di alto livello**
 - **astraggono** dai dettagli di funzionamento dell'elaboratore
- Il linguaggio macchina e l'assembly sono **linguaggi di basso livello**.

Compilazione e interpretazione

Per tradurre un programma (**sorgente**) in linguaggio macchina si può usare: ● Un **compilatore**: programma che prende in input il programma sorgente e produce in output il corrispondente programma in linguaggio macchina, **eseguibile successivamente** dal computer

- Un **interprete**: programma che prende in input il programma sorgente, traduce un comando per volta e **lo esegue** man mano

Esempi:

- Il C è un linguaggio compilato
- Il JavaScript è un linguaggio interpretato

Vantaggi della **compilazione**:

- Il programma eseguibile che si ottiene è veloce
- Il programma eseguibile può essere eseguito più volte senza ricompilare
- Il compilatore può controllare e ottimizzare il programma prima che venga eseguito

Svantaggi della **compilazione**:

- Compilare un programma può richiedere molto tempo

- Gli errori che sfuggono al controllo del compilatore non potranno più essere gestiti durante l'esecuzione
- Un compilatore produce un eseguibile che funziona su una sola architettura (Intel, ARM, ...) e sistema operativo (Windows, Linux, MacOS, ...)

Vantaggi dell'**interpretazione**:

- Non è necessario ricompilare tutto il programma ogni volta che si fa una modifica.
- Portabilità: lo stesso identico programma può essere eseguito su architetture (Intel, ARM,...) e sistemi operativi (Windows, Linux, MacOS, ...) diversi
- Gli errori che si incontrano a tempo di esecuzione possono essere gestiti meglio

Svantaggi dell'**interpretazione**:

- L'esecuzione del programma è rallentata dall'interprete, che deve tradurre ogni comando
- Nessun controllo sui programmi prima di iniziarne l'esecuzione

Alcuni linguaggi combinano compilazione e interpretazione: Java.

LEZIONE 2

Astrazione

Distinzione tra le proprietà esterne di un'entità e i dettagli della struttura interna. Esempi:

- Possiamo usare un'auto senza conoscerne i dettagli interni
- Possiamo usare una app senza sapere come è stata realizzata

Gerarchia di livelli: ogni livello è descritto tramite le sue componenti.

Algoritmo

Sequenza di operazioni eseguibili e non ambigue.

Un algoritmo deve essere **comprensibile**.

Algoritmo + astrazione:

- Divido un problema complesso in sottoproblemi più semplici
- Trovo le soluzioni di ciascun problema
- Ricompongo la soluzione del problema originale

Proprietà degli algoritmi

Correttezza: funzionamento corretto dell'algoritmo.

Efficienza: trova una soluzione rapidamente e usando meno risorse possibili.

Algoritmo e programmi

Un **computer** è un esecutore di algoritmi descritti tramite programmi.

Un **programma** è una sequenza di istruzioni scritte in un linguaggio comprensibile al calcolatore.

Algoritmo \neq programma: un programma è la rappresentazione di un algoritmo.

Linguaggio di programmazione

Primitive: sono le componenti base, i mattoni del linguaggio di programmazione.

Sintassi: come è fatta una primitiva.

Semantica: il significato della primitiva.

Regole: spiegano come combinare primitive per rappresentare idee complesse.

Linguaggio macchina

Insieme delle istruzioni macchina, cioè quelle accettate dalla CPU \rightarrow codice binario. Esempio: 16 bit per istruzione \rightarrow 4 bit codice istruzione, 12 bit operandi.

Linguaggi ad alto livello

- Primitive indipendenti dalla macchina
- Primitive ad alto livello: 1 primitiva = più istruzioni macchina

Compilatore: traduce un programma da linguaggio di alto livello a linguaggio macchina.

Interprete: traduce un'istruzione alla volta e, senza memorizzare il risultato, la esegue.

Indipendenza dalla macchina: un programma può essere utilizzato su computer diversi traducendolo con l'apposito compilatore.

Linguaggio standard:

- Specificazione chiara e universale (indipendente da hardware e sistema operativo) del linguaggio.
- Estensioni del linguaggio: aggiunte allo standard per aumentare le potenzialità o adattarlo ad una specifica macchina.

Java Bytecode

Java cerca di unire i vantaggi dei linguaggi compilati a quelli dei linguaggi interpretati. Il compilatore non traduce più in linguaggio macchina, ma in **bytecode**.

Java

Può sia compilare che interpretare. Vantaggi dell'utilizzo del bytecode:

- **Portabilità**: eseguibile su ogni macchina (indipendentemente dall'hardware)
- **Velocità**: Tradurre ed eseguire bytecode è più veloce rispetto alla norma

L'interprete Java è la Java Virtual Machine (JVM), traduce da bytecode a linguaggio macchina specifico per l'hardware.

Class Loader

Un programma Java consiste di più pezzi detti classi. Classi diverse possono avere autori diversi e ognuno è compilato separatamente. Il class loader collega tra di loro classi diverse.

Linguaggi: tipi di errori

I linguaggi di programmazione, come tutti i linguaggi, sono caratterizzati da una sintassi e una semantica che, se violate, producono errori:

1. Sintattici
2. Run-time
3. Logici

Errori sintattici

Intercettati dal compilatore che li segnala con appositi messaggi di errore - I messaggi di errore possono essere fuorvianti

- Sono i più facili da correggere
- Il codice non può essere eseguito fino a che tutti gli errori sintattici non sono stati corretti

Errori a Run-time

- Errori durante l'esecuzione del programma (run-time)
- Non sempre facili da identificare e correggere
- I messaggi di errore possono non essere di aiuto

Errori logici

Solo perché un programma è compilato ed eseguito senza messaggi di errori, questo NON significa che il programma sia corretto!

Un errore nel disegno (algoritmo) o nell'implementazione dell'algoritmo: ● Il codice è corretto, non ci sono errori a run-time ma sono effettuate azioni sbagliate durante l'esecuzione

I più difficili da trovare e correggere, bisogna provare (testare) attentamente.

LEZIONE 3

Applicazioni e Applet

Esistono due tipi di programmi Java: le **applicazioni** e le **applet**:

- Le applicazioni sono programmi regolari, pensati per essere eseguiti sul tuo computer
- Le applet sono piccole applicazioni, pensate per essere inviate via Internet ed eseguite su un computer remoto (non sono supportate da molti browser moderni)

Terminologia

Programmatore: la persona che scrive il programma.

User: la persona con cui il programma interagisce.

Package: libreria di classi definite in precedenza.

Argomenti: elementi all'interno delle parentesi.

Variabile: qualcosa che può memorizzare dati.

Statement: un'istruzione fornita al computer, terminata con un punto e virgola
“;”.

Sintassi: insieme di regole grammaticali di un linguaggio di programmazione.

Stampa a schermo

System.out è un oggetto che si utilizza per mandare un output allo schermo. **println** è un metodo per stampare a schermo (andando a capo) qualunque cosa ci sia tra le parentesi.

- Un oggetto esegue un'azione quando viene chiamato uno dei suoi metodi
- In un programma Java si ottiene un'**invocazione di metodo** (o **chiamata di metodo**) scrivendo il nome dell'oggetto, seguito da un punto (chiamato, in gergo, **dot**), seguito dal nome del metodo e infine da una coppia di parentesi tonde
- All'interno delle parentesi potrebbero essere specificati uno o più argomenti
- Se non ci sono argomenti si indicano solo le parentesi ()

Compilazione di un programma o classe Java

- Un programma Java consiste di una o più classi, che devono essere compilate prima di poterlo eseguire
- Non è necessario compilare le classi fornite da Java
- Ogni classe dovrebbe essere in un file separato
- Il nome del file dovrebbe essere lo stesso nome della classe (con estensione **.java**)

Compilazione ed esecuzione

Utilizzare un IDE (ambiente di sviluppo integrato) che combina un editor di testo con comandi per la compilazione e l'esecuzione di programmi Java. Quando un programma Java viene compilato, la versione bytecode del programma ha lo stesso nome, ma l'estensione cambia da **.java** a **.class**.

Un programma java può avere infinite classi, la classe che verrà eseguita è quella rappresentante l'intero programma, ovvero quella contenente il metodo **main**, **public static void main(String[] args)**

LEZIONE 4

Architetturalmente neutro

Il compilatore produce il bytecode, questo formato è indipendente dall'architettura della macchina (**ANDF** = Architecture Neutral Distribution Format). Il bytecode generato da java può essere eseguito da qualsiasi **JVM**.

Cos'è il bytecode?

Il compilatore Java non genera "codice macchina" bensì un codice ad alto livello (bytecode), indipendente dall'hardware. Questo codice viene interpretato dall'interprete Java in run-time.

Interpretato

L'interprete esegue il bytecode direttamente sulla macchina, in questo modo il codice risulta:

- portabile su qualsiasi sistema avente una JVM.
- sicuro
- high performance

Il sistema Java (compilazione + interpretazione + librerie run-time) è facilmente portabile grazie al bytecode interpretato.

- il compilatore è scritto in Java
- l'ambiente run-time è scritto in ANSI C con interfacce standard verso il S.O.
- no "implementation dependency"

Elementi lessicali

Si definiscono **elementi lessicali** le parole del linguaggio, cioè le piccole unità cui attribuire un valore o un significato.

In java ci sono i seguenti tipi di elementi lessicali:

- parole riservate del linguaggio e caratteri speciali (public, for, class, ...)
- stringhe (sequenze di caratteri)
- numeri senza segno
- identificatori

Identificatori

Un **identificatore** è qualcosa definito dal programmatore per identificare univocamente:

- una classe
- una variabile
- un metodo
- un oggetto

Gli identificatori sono sequenze di caratteri alfanumerici, sottostanti alle seguenti regole:

- non devono iniziare con un numero
- possono contenere solo numeri, lettere, _ e \$ (no altri caratteri speciali) - non possono essere utilizzate parole chiave del linguaggio
- non possono contenere spazi
- due variabili con lo stesso *scope* (non ci devono essere ambiguità) non possono avere lo stesso nome
- i nomi sono case-sensitive

Le keyword del linguaggio

short	if	void	package
byte	else	return	volatile
int	while	synchronized	default
char	do	class	implements
long	for	new	import
float	switch	static	instanceof
double	case	abstract	(goto)
boolean	break	final	(const)
this	finally	native	try
super	continue	private	catch
interface	label	protected	throw
extends	transient	public	throws

Che identificatore dare ad una classe?

Il nome deve essere scelto in base a cosa la classe deve rappresentare, valgono le regole sintattiche viste in precedenza.

Per convenzione:

- il nome di una classe deve avere la lettera iniziale maiuscola
- se è un nome composto: ciascuna parola deve avere l'iniziale maiuscola (MyClass)

Obblighi sul nome del file?

Se la classe ha intestazione (**public class...**), allora sarò obbligato a salvarla in un file con nome coincidente ad essa.

Se la classe ha intestazione (**class...**) posso salvarla in un file che ha nome diverso.

Variabili e loro dichiarazione

Una **variabile** è una precisa locazione di memoria:

- con un identificatore (nome variabile)
- in cui viene memorizzato un valore di un unico tipo
- che ha un ben determinato *scope*

Si utilizzano variabili quando un dato valore varia durante l'esecuzione del programma. Prima di poter usare una variabile occorre **SEMPRE** dichiararla:

- specificando il tipo (int, double, etc...)
- assegnando un identificatore (nome)

Regole per le variabili

Devono essere rispettate:

- non devono iniziare con un numero
- possono contenere solo numeri, lettere, _ e \$ (no altri caratteri speciali)
- non possono essere utilizzate parole chiave del linguaggio
- due variabili con lo stesso *scope* (non ci devono essere ambiguità) non possono avere lo stesso nome
- i nomi sono case-sensitive

Dovrebbero essere rispettate:

- usare sempre nomi estesi che descrivano bene cosa rappresentano
- iniziare il nome della variabile con una minuscola

- se il nome consiste in più di una parola le parole vanno scritte di seguito, con la lettera iniziale maiuscola (ciaoComeVa)
- evitare di usare \$

I tipi

Un tipo determina:

- il valore che la variabile può contenere
- le operazioni che possono essere effettuate su di essa

In java esistono 2 categorie principali di tipi di dati:

- primitivi (più semplici e non scomponibili)
- non primitivi (complessi e composti → classi, interfacce e array)

Nome	Tipo di valore	Memoria usata	Range di valori
byte	intero	8 bit = 1 byte	-128 : +127
short		16 bit = 2 byte	-32.768 : +32.767
int		32 bit = 4 byte	-2.147.483.648 : +2.147.483.647
long		64 bit = 8 byte	-9.223.372.036.854.775.808 : +9.223.374.036.854.775.808
float	floating point	32 bit = 4 byte	+/- 3,4028... × 10 ⁺³⁸ : +/- 1,4023... × 10 ⁻⁴⁵
double		64 bit = 8 byte	+/- 1,767... × 10 ⁺³⁰⁸ : +/- 4,940... × 10 ⁻³²⁴
char	singolo carattere	16 bit = 2 byte	Tutti i caratteri Unicode
boolean	true o false	1 bit	true o false

Assegnare un valore ad una variabile

Per assegnare un valore ad una variabile si utilizza l'operatore di assegnamento `=`. Assegna il valore alla destra di esso alla variabile posta alla sua sinistra.

Questa operazione può essere effettuata anche con una variabile **già dichiarata**.

Il valore assegnato ad una variabile deve essere dello stesso tipo della variabile.

Regole

Di norma si inizializza una variabile:

- nel momento in cui la si dichiara
- e comunque prima di usarla

Quando si dichiara solamente una variabile, il suo valore è **indefinito**.

Compatibilità di assegnamento

Java è detto "strongly typed", esempio: non si può assegnare un float ad un int.

byte → short → int → long → float → double

Si possono assegnare valori da destra verso sinistra ma non il contrario.

Type cast

Un type cast cambia temporaneamente il valore di una variabile del tipo dichiarato ad un altro.

N.B. Qualunque valore diverso da zero dopo le virgole viene **troncato**, non arrotondato.

Espressioni

Le espressioni sono un **misto di variabili** (o valori) e **operatori** che producono come risultato un **valore** (le variabili (o valori) sono chiamate **operandi**).

Il tipo di valore che restituisce un'espressione dipende dal tipo degli operandi.

Espressioni ed operatori

La maggior parte delle espressioni fa uso di **operatori**. Un operatore è un simbolo speciale che:

- regola l'esecuzione dell'espressione
- restituisce un valore calcolato il cui tipo dipende dagli operandi e dall'operatore

Gli operatori possono avere un diverso numero di operandi:

- operatori **unari**: richiedono 1 operando (es. ++)
- operatori **binari**: richiedono 2 operandi (es. =)
- operatori **ternari**: richiedono 3 operandi

Le notazioni

Gli operatori **unari** utilizzano:

- notazione **prefissa**: l'operatore appare prima dell'operando (es. ++x) -

notazione **postfissa**: l'operatore appare dopo l'operando (es. x++) Gli

operatori **binari** e **ternari** utilizzano:

- notazione **infixa**: l'operatore appare in mezzo agli operandi (es. x = y)

Tipologie di operatori

- Aritmetici
- Aritmetici di incremento e decremento
- Di assegnamento
- Relazionali
- Condizionali
- Bit a bit

Operatori aritmetici

Validi per tipi **interi** e **floating point**

Operatore	Descrizione	Uso	Significato
+	Somma	op1+op2	Somma il valore di op1 a quello di op2
-	Sottrazione	op1-op2	Sottrae al valore di op1 quello di op2
*	Moltiplicazione	op1*op2	Moltiplica il valore di op1 con quello di op2
/	Divisione	op1/op2	Divide il valore di op1 con quello di op2
%	Modulo	op1%op2	Calcola il resto della divisione tra il valore di op1 e quello di op2
-	Negazione aritmetica	-op	Trasforma il valore di op in positivo o negativo

Precedenza tra operatori

Gli operatori possono essere combinati in espressioni complesse (con più operatori etc...). Gli operatori hanno una **precedenza ben definita** implicita che determina il loro ordine: - moltiplicazione, divisione e resto sono valutati prima di somma, sottrazione e concatenazione tra stringhe

Gli operatori che hanno la stessa precedenza sono valutati da sinistra a destra.

Mediante le parentesi si può alterare l'ordine di precedenza.

Cosa posso ancora assegnare ad una variabile?

Ad una variabile, oltre quello detto prima posso assegnare il valore di un'**espressione**.

L'operatore di assegnamento ha la precedenza **più bassa** in assoluto, ovvero prima si svolge l'espressione, poi si assegna il risultato.

Il valore assegnato ad una variabile deve essere dello **stesso tipo** della variabile.

Come determinare il tipo di un'espressione?

Le espressioni con 2 o più operatori possono essere viste come una serie di passi, ognuno riguardante solamente 2 operandi per volta.

Il risultato di ogni step produce uno degli operandi da utilizzare al passo successivo.

Esempio:

- Se almeno uno degli operandi è floating point e gli altri sono interi, il risultato sarà di tipo floating point.

byte → short → int → long → float → double

Output a video

- **System.out.println()** stampa a video ciò che è stato specificato e va a capo
- **System.out.print()** stamp a video ciò che è stato specificato

Concatenazione di output

Se si vuole stampare a video oltre ad una stringa, anche una variabile, si utilizza il **+**.

System.out.println("x vale: " + x);

Casi particolari

Se si concatenano semplicemente i valori di due variabili numeri, il risultato è una stampa della somma.

System.out.println(x + y);

Inserendo una stringa questo accade comunque.

System.out.println("valori: " + x + y);

In caso non la si volesse, si utilizzerà una stringa vuota in mezzo alle due variabili. **System.out.println(x + " " + y);**

I commenti

Commenti al codice:

- **//...** per una sola linea di commento
- **/*...*/** per più linee di commento

Commenti per generare documentazione:

- **/**...*/:**
 - **@param** nomeParametro: commenti sul parametro
 - **@return**: commento sul ritorno di un metodo

LEZIONE 5

La classe String

Un valore di tipo **String** è:

- Una sequenza di caratteri
- Trattati come un unico elemento

Esiste in java un tipo che permette la memorizzazione di stringhe: la classe **String**.

Sebbene sia una classe, è particolare:

- si può dichiarare una variabile di questo tipo ed assegnargli un valore senza dover creare un oggetto

Differenze tra String e char

Una variabile di tipo:

- **char** può contenere un unico carattere
- **String** può contenere una sequenza di caratteri

Il valore è racchiuso tra:

- Singoli apici per i **char** `char carattere2 = 'd';` - Doppi apici per le **String** `String carattere1 = "d";`

Concatenazione di stringhe

Due stringhe si possono concatenare utilizzando l'operatore **+**:

```
String saluto = "Hello";
```

```
String frase;
```

```
frase = saluto + " Simone";
```

```
System.out.println(frase);
```

Un numero qualunque di stringhe possono essere concatenate utilizzando l'operatore **+**:

```
frase = saluto + " Simone" + " , " + "hai " + "visto " + "che " + "bella " + "giornata " + "è " + "oggi?";
```

I metodi di String

Un oggetto della classe **String** immagazzina dati che sono una sequenza di caratteri.

Gli oggetti hanno dei metodi oltre ai dati:

- Il metodo **length()** restituisce il numero di caratteri di uno specifico oggetto **String**

```
String greeting = "Hello";
```

```
int n = greeting.length();
```

```
System.out.println(n);
```

Il risultato sarà 5.

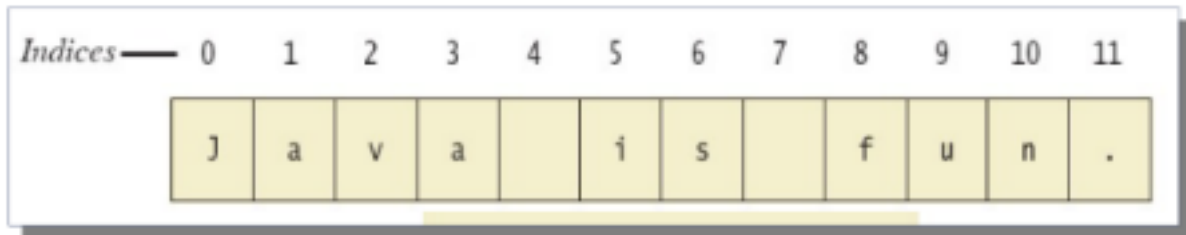
Il metodo length()

Il metodo **length()** restituisce un **int**. Esso si può usare ovunque sia possibile usare un **int**.

```
int count = command.length();
```

```
System.out.println("Length is " + command.length()); count =  
command.length() + 3;
```

Indici di String



- La posizione all'interno della stringa inizia con 0, non con 1
- Quindi, la "J" nella stringa "Java is fun." è in posizione 0
- La posizione di un carattere è spesso chiamata indice (index)
- La 'f' in "Java is fun." è all'indice 8

Alcuni metodi della classe String

Method	Return Type	Example for String s = "Java";	Description
charAt (index)	char	c = s.charAt(2); // c='v'	Returns the character at <i>index</i> in the string. Index numbers begin at 0.
compareTo (a_string)	int	i = s.compareTo("C++"); // i is positive	Compares this string with <i>a_string</i> to see which comes first in lexicographic (alphabetic, with upper before lower case) ordering. Returns a negative integer if this string is first, zero if the two strings are equal, and a positive integer if <i>a_string</i> is first.
concat (a_string)	String	s2 = s.concat("rocks"); // s2 = "Javarocks"	Returns a new string with this string concatenated with <i>a_string</i> . You can use the + operator instead.
equals (a_string)	boolean	b = s.equals("Java"); // b = true	Returns true if this string and <i>a_string</i> are equal. Otherwise returns false.
equals IgnoreCase (a_string)	boolean	b = s.equalsIgnoreCase("Java"); // b = true	Returns true if this string and <i>a_string</i> are equal, considering upper and lower case versions of a letter to be the same. Otherwise returns false.
indexOf (a_string)	int	i = s.indexOf("va"); // i = 2	Returns the index of the first occurrence of the substring <i>a_string</i> within this string or -1 if <i>a_string</i> is not found. Index numbers begin at 0.

<code>lastIndexOf (a_string)</code>	<code>int</code>	<code>i = s.lastIndexOf("a"); // i = 3</code>	Returns the index of the last occurrence of the substring <i>a_string</i> within this string or -1 if <i>a_string</i> is not found. Index numbers begin at 0.
<code>length()</code>	<code>int</code>	<code>i = s.length(); // i = 4</code>	Returns the length of this string.
<code>toLowerCase()</code>	<code>String</code>	<code>s2 = s.toLowerCase(); // s = "java"</code>	Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase. This string is unchanged.
<code>toUpperCase()</code>	<code>String</code>	<code>s2 = s.toUpperCase(); // s2 = "JAVA"</code>	Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase. This string is unchanged.
<code>replace (oldchar, newchar)</code>	<code>String</code>	<code>s2 = s.replace('a','o'); // s2 = "Jovo";</code>	Returns a new string having the same characters as this string, but with each occurrence of <i>oldchar</i> replaced by <i>newchar</i> .
<code>substring (start)</code>	<code>String</code>	<code>s2 = s.substring(2); // s2 = "va";</code>	Returns a new string having the same characters as the substring that begins at index <i>start</i> through to the end of the string. Index numbers begin at 0.
<code>substring (start,end)</code>	<code>String</code>	<code>s2 = s.substring(1,3); // s2 = "av";</code>	Returns a new string having the same characters as the substring that begins at index <i>start</i> through to but not including the character at index <i>end</i> . Index numbers begin at 0.
<code>trim()</code>	<code>String</code>	<code>s = " Java "; s2 = s.trim(); // s2 = "Java"</code>	Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

Elaborazione di stringhe

- Tecnicamente gli oggetti di tipo **String** non possono essere modificati. Ad esempio non esiste alcun metodo per scrivere un'istruzione del tipo: *cambia in z il quinto carattere della stringa*
- Si possono però scrivere programmi per cambiare il valore di una variabile **String**, grazie a questa operazione molte necessità di programmazione vengono soddisfatte
- Per modificare il valore di una stringa basta utilizzare l'operatore di assegnamento **String**
nome = "Savitch";
nome = "Walter " + nome;

Caratteri di escape

Come possiamo fare a stampare:

Il termine "Java" indica il nome di un linguaggio.

Bisogna specificare al compilatore che il doppio apice (") non segnala l'inizio o la fine della stringa, ma invece è un carattere da stampare.

System.out.println("Il termine \"Java\" indica il nome di un linguaggio.");

```
\ " Double quote.  
\ ' Single quote.  
\\ Backslash.  
\n New line. Go to the beginning of the next line.  
\r Carriage return. Go to the beginning of the current line.  
\t Tab. Add whitespace up to the next tab stop.
```

Ogni sequenza di escape è un singolo carattere anche se viene scritta con due simboli.

LEZIONE 6

Output a schermo

Conosciamo già diversi esempi di output a schermo:

- **System.out** è un oggetto che fa parte di Java
- **println()** è uno dei metodi disponibili per l'oggetto **System.out**
- Il metodo **System.out.println()** può essere utilizzato per visualizzare righe di testo
- Gli elementi visualizzati possono essere:
 - Stringhe tra apici
 - Variabili
 - Costanti (es. numeri)
 - O qualsiasi altro oggetto definibile in java

L'operatore di concatenazione (+) è utile anche quando all'interno del codice l'argomento di println() non ci sta tutto sulla stessa riga:

```
System.out.println("Numero fortunato = " + 13 +  
"Numero segreto = " + number);
```

È bene non interrompere la linea se non immediatamente prima o dopo l'operatore della concatenazione (+).

In alternativa è possibile usare **print()**:

```
System.out.print("One, two,");  
System.out.print(" buckle my shoe.");  
System.out.println(" Three, four,");  
System.out.println(" shut the door.");
```

terminando con una chiamata a **println()**.

Input da tastiera

In java la classe **Scanner** gestisce l'input da tastiera, questa classe si trova nel package **java.util** (package = libreria di classi).

Come usare la classe Scanner

- Nelle prime righe del programma occorre scrivere la riga seguente:
import java.util.Scanner;
- Bisogna poi creare un oggetto della classe Scanner:
Scanner nome_oggetto_scanner = new Scanner (System.in);
- A questo punto tramite i metodi disponibili possiamo leggere i dati:
int n1 = nome_oggetto_scanner.nextInt();
double d1 = nome_oggetto_scanner.nextDouble();

Alcuni metodi della classe Scanner

<code>nome_oggetto_scanner.next()</code> Restituisce un valore String che corrisponde al prossimo input da tastiera fino al primo carattere di delimitazione escluso. I caratteri di delimitazione di default sono i caratteri di spaziatura.
<code>nome_oggetto_scanner.nextLine()</code> Legge la parte rimanente della riga corrente e restituisce i caratteri letti come un valore di tipo String . Si noti che il carattere di terminazione di riga <code>'\n'</code> viene letto, ma scartato. Infatti non viene incluso nella stringa restituita.
<code>nome_oggetto_scanner.nextInt()</code> Legge il prossimo input da tastiera come un valore di tipo int .
<code>nome_oggetto_scanner.nextDouble()</code> Legge il prossimo input da tastiera come un valore di tipo double .
<code>nome_oggetto_scanner.nextFloat()</code> Legge il prossimo input da tastiera come un valore di tipo float .
<code>nome_oggetto_scanner.nextLong()</code> Legge il prossimo input da tastiera come un valore di tipo long .
<code>nome_oggetto_scanner.nextByte()</code> Legge il prossimo input da tastiera come un valore di tipo byte .
<code>nome_oggetto_scanner.nextShort()</code> Legge il prossimo input da tastiera come un valore di tipo short .
<code>nome_oggetto_scanner.nextBoolean()</code> Legge il prossimo input da tastiera come un valore di tipo boolean . I valori <i>true</i> e <i>false</i> devono essere scritti proprio come true e false . Viene accettata qualsiasi combinazione di lettere maiuscole e minuscole.
<code>nome_oggetto_scanner.useDelimiter(parola_di_delimitazione)</code> Fa sì che la stringa <i>parola_di_delimitazione</i> sia l'unico delimitatore utilizzato per separare l'input. Solo le stringhe corrispondenti a questa parola saranno considerate delimitatori. In particolare, i caratteri spazio, nuova riga e gli altri caratteri di spaziatura non saranno più considerati delimitatori, a meno che non facciano parte della parola <i>parola_di_delimitazione</i> . Questo è un semplice esempio d'uso del metodo <code>useDelimiter</code> . Ci sono diversi modi per cambiare i delimitatori, che tuttavia non verranno presentati in questo testo.

Attenzione al metodo `nextLine()`

Il metodo **`nextLine()`** legge tutto ciò che resta sulla linea corrente, anche se vuota.

Esempio:

```
int n = input.nextInt();
String s1 = input.nextLine();
String s2 = input.nextLine();
```

Assumiamo di voler inserire il seguente input:

```
44 gatti in
fila per 3
```

Allora verranno effettuati i seguenti assegnamenti:

```
n diventa 44
s1 diventa " gatti in"
s2 diventa "fila per 3"
```

Cambiando l'input in:

```
44
gatti in
fila per 3
```

Verranno effettuati i seguenti assegnamenti:

```
n diventa 44
```

s1 diventa "" ovvero la stringa vuota!

s2 diventa "gatti in"

Cos'è successo?

- quando terminiamo di inserire un input da tastiera, premiamo **INVIO** • quando digitiamo **44** e premiamo invio, in realtà abbiamo inserito: **44\n** • il comando **n=nextInt()** legge **44**, ma nella stessa riga resta il carattere **\n** • il comando **s1=nextLine()** legge ciò che resta della linea corrente, ovvero **\n** • da documentazione sappiamo che **\n** viene letto ma scartato, assegnando così a **s1** la stringa vuota

Questo:

- è un problema noto
- al quale non c'è una soluzione generale
- bisogna tenerne conto quando si utilizzano questi metodi

La stringa vuota

- abbiamo già visto che una stringa può avere un numero qualsiasi di caratteri, zero incluso
- La stringa con zero caratteri è chiamata stringa vuota (**empty string**).
- La stringa vuota è utile e può essere creata in diversi modi, incluso:
String vuota = "";

LEZIONE 7

Flusso di controllo

- Il **flusso di controllo** è l'ordine in cui in programma svolge le azioni
- Un'istruzione di selezione (**branching statement**) sceglie tra due o più azioni possibili
- Un ciclo (**loop**) ripete un'azione fino a che non si verifica una condizione di stop.

L'istruzione if-else semplice

È un'istruzione di selezione che sceglie tra 2 azioni possibili.

Sintassi:

```
if (Espressione_booleana){
    Istruzione_1;
} else {
    Istruzione_2;
}
```

Istruzioni composte

Se si vogliono includere più istruzioni in un solo "ramo" dell'if, è sufficiente racchiudere le diverse istruzioni tra **parentesi graffe {}**. Un insieme di istruzioni racchiuse tra parentesi graffe è considerato come un'unica istruzione "più ampia".

Espressioni booleane

Il valore di una espressione booleana è **true** o **false**.

Operatori di confronto in Java

Math Notation	Name	Java Notation	Java Examples
=	Equal to	==	balance == 0 answer == 'y'
≠	Not equal to	!=	income != tax answer != 'y'
>	Greater than	>	expenses > income
≥	Greater than or equal to	>=	points >= 60
<	Less than	<	pressure < max
≤	Less than or equal to	<=	expenses <= income

Espressioni booleane complesse

Le espressioni booleane possono essere combinate utilizzando l'operatore "and" (**&&**).

if ((score > 0) && (score <= 100))

Le parentesi sono spesso utilizzate per aumentare la leggibilità, l'espressione composta è vera quando entrambe le sotto espressioni sono vere.

(Sotto_Espressione_1) || (Sotto_Espressione_2)

L'espressione composta è vera se:

- Entrambe le sotto espressioni sono vere

- Una qualunque delle due sotto espressioni è vera

La versione di "or" implementata in Java è l' inclusive or che permette che una qualunque o entrambe delle due sottoespressioni siano vere.

Negazione di un'espressione booleana

Un'espressione booleana può essere negata utilizzando l'operatore "not"

(!). Sintassi:

!(Espressione_booleana)

Operatori logici

Name	Java Notation	Java Examples
Logical <i>and</i>	&&	(sum > min) && (sum < max)
Logical <i>or</i>		(answer == 'y') (answer == 'Y')
Logical <i>not</i>	!	!(number < 0)

Operatori booleani

Value of <i>A</i>	Value of <i>B</i>	Value of <i>A</i> && <i>B</i>	Value of <i>A</i> <i>B</i>	Value of ! (<i>A</i>)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Istruzioni if-else annidate (nested)

- un'istruzione **if-else** può contenere al suo interno qualunque tipo di istruzione
- In particolare, può contenere un'altra istruzione **if-else**:
 - un if-else può essere annidato in entrambi i rami
- ogni **else** è associato all'**if** più vicino (che ancora non è stato associato) • **se usata correttamente**, l'indentazione rende chiaro quale **if** è associato a quale **else**
- le parentesi graffe possono essere utilizzate per raggruppare le istruzioni

Istruzioni composte

Quando una sequenza di istruzioni è racchiusa tra parentesi graffe {}, queste formano una singola istruzione **composta**.

Un'istruzione **composta** può essere utilizzata ovunque sia possibile utilizzare un'istruzione (**singola**).

Istruzione if-else multi ramo

Sintassi:

if (espressione_booleana_1)
azione_1;

```
else if (espressione_booleana_2)
    azione_2;
else if (espressione_booleana_3)
    azione_3;
else if ...
else
    azione_di_default;
```

Utilizzo di ==

== può essere utilizzato per determinare se due interi o due caratteri hanno lo stesso valore.

```
int a;
```

```
a = ...;
```

```
if (a == 3)
```

== non è indicato per determinare se due valori in virgola mobile sono uguali. Si utilizza < e una determinata tolleranza sul valore assoluto.

```
if (Math.abs(b - c) < epsilon)
```

con **b**, **c**, **epsilon** di tipo floating point

== non è appropriato per determinare se due oggetti hanno lo stesso valore.

- **if (s1 == s2)**, dove **s1** e **s2** sono 2 stringhe, determina solamente se s1 e s2 si riferiscono alla stessa area di memoria
- Se **s1** e **s2** si riferiscono a stringhe con la stessa sequenza di caratteri, ma memorizzate in posizioni diverse nella memoria, l'espressione (**s1 == s2**) risulterà falsa

Per testare l'uguaglianza di oggetti della classe String, utilizzare il metodo

```
equals. s1.equals(s2)
```

oppure

```
s2.equals(s1)
```

Per testare l'uguaglianza ignorando maiuscole e minuscole, utilizzare il metodo

```
equalsIgnoreCase.
```

```
("Hello".equalsIgnoreCase("hello"))
```

Operatore condizionale

```
if (n1 > n2)
```

```
    max = n1;
```

```
else
```

```
    max = n2;
```

Può essere scritto in modo più compatto come:

```
max = (n1 > n2) ? n1 : n2;
```

Il **?** e **:** insieme sono chiamati operatore condizionale o operatore ternario.

Il metodo exit

Può capitare che si crei una situazione in cui continuare l'esecuzione del programma è privo di senso, si può terminare regolarmente in questo modo:

```
System.exit(0);
```

Esempio

```
if (numeroDiVincitori == 0)
```

```
{
```

```
    System.out.println ("Errore: Divisione per zero.");
```

```
    System.exit (0);
```

```

}
else
{
    unaQuota = montepremi / numeroDiVincitori;
    System.out.println ("Ogni vincitore riceverà una somma di €" + unaQuota);
}

```

Il tipo boolean

Il tipo **boolean** è un tipo primitivo che può avere solamente due valori: **true** oppure **false**.

Espressioni o variabili booleane

- Le variabili, le costanti e le espressioni di tipo boolean quando vengono valutate restituiscono tutte true oppure false.
- Ad una variabile booleana può essere dato il valore di una espressione booleana tramite l'operatore di assegnamento.

```
boolean isPositive = (number > 0);
```

```
...
```

```
if (isPositive) ...
```

Nomi di variabili booleane

Scegliere nomi come ad esempio **isPositive** o **systemsAreOk**, evitare nomi come **numberSign** o **systemStatus**.

Regole di precedenza

Le parentesi vengono usate per indicare l'ordine delle operazioni, quando queste vengono omesse, entrano in gioco le cosiddette **regole di precedenza**.

<i>Highest Precedence</i>
First: the unary operators +, -, ++, --, and !
Second: the binary arithmetic operators *, /, %
Third: the binary arithmetic operators +, -
Fourth: the boolean operators <, >, <=, >=
Fifth: the boolean operators ==, !=
Sixth: the boolean operator &
Seventh: the boolean operator
Eighth: the boolean operator &&
Ninth: the boolean operator
<i>Lowest Precedence</i>

Valutazioni short-circuit

Talvolta è possibile valutare solamente una parte di una espressione booleana per determinare il valore dell'espressione intera.

- Se il primo operando associato con **||** è **true**, l'espressione completa sarà **true**. - Se il primo operando associato con **&&** è **false**, l'espressione completa sarà **false**. Questa è chiamata valutazione short-circuit (corto circuito) o lazy (pigra). La valutazione

short-circuit non solo è efficiente, talvolta è anche essenziale! Può esserci un errore a run-time, ad esempio, a causa di un tentativo di dividere per zero. Supponiamo che sia **number = 0** allora:

if ((number != 0) && (sum/number > 5))

darà errore nella seconda parte dell'espressione

La valutazione short-circuit può essere fatta sostituendo **&** al posto di **&&** oppure **|** al posto di **||**.

LEZIONE 8

L'istruzione switch

L'istruzione **switch** è una selezione che effettua una scelta sulla base di un intero, di un carattere o di una stringa.

Inizia con la keyword **switch** seguita da un controllo tra parentesi. A seguire una lista di casi.

In caso nessuno di questi casi si verificasse viene eseguito il caso **default** (aggiunta opzionale, ma consigliata). Non è permesso ripetere le **case labels**.

Ogni caso viene terminato dalla keyword **break**, per evitare l'esecuzione di altri casi.

LEZIONE 9

Blocchi

Un **blocco** è una sequenza di comandi racchiusa tra parentesi graffe (es. ramo di un **if**).

Si possono creare blocchi all'interno di ulteriori blocchi.

Le variabili dichiarate in un blocco sono locali al blocco, quindi non visibili al di fuori di esso.

La visibilità di una variabile è la porzione di programma in cui tale variabile può essere utilizzata. La visibilità di una variabile locale è la porzione di programma che va dalla dichiarazione della variabile stessa fino alla fine del blocco che la contiene.

Limitare la visibilità di una variabile al blocco che la contiene permette

- di: - Riutilizzare nomi di variabili in diverse parti del programmare
- Anche con tipi diversi

Le variabili dichiarate all'interno di un blocco non possono avere lo stesso nome di una variabile già esistente all'esterno di esso.

I cicli (loops)

La porzione di programma che ripete un'istruzione (o un gruppo) è chiamata **ciclo (loop)**.

L'istruzione o gruppo di istruzioni ripetute costituiscono il **corpo (body)** del ciclo. Ogni ripetizione del corpo viene detta **iterazione**. Si deve definire quando la ripetizione del corpo deve terminare.

Istruzione while

Il ciclo **while** ripete l'azione definita nel corpo finché l'espressione booleana di controllo rimane vera. Quando questa diventa falsa esso termina. Il corpo tipicamente contiene un'azione che fa diventare falsa l'espressione di controllo.

Istruzione do-while

Simile all'istruzione **while**, eccetto che il corpo viene eseguito almeno una volta (dopo la parentesi della condizione booleana ci va il punto e virgola). Per prima cosa, viene eseguito il corpo del ciclo, poi viene valutata l'espressione booleana di controllo.

Cicli infiniti

Un ciclo che ripete il proprio corpo senza mai terminare è chiamato **ciclo infinito (infinite loop)**, normalmente all'interno del corpo del ciclo le variabili vengono alterate, in maniera tale da far diventare falsa l'espressione booleana di controllo. Se questo non accade in maniera corretta, allora si potrebbe generare un ciclo infinito.

Cicli annidati

Il corpo di un ciclo può contenere qualsiasi cosa, incluso un ulteriore ciclo.

LEZIONE 10

L'istruzione for

L'istruzione **for** esegue il corpo del ciclo un numero prefissato di volte. Il **corpo** può essere sia un'istruzione semplice che un blocco **{}**. È possibile dichiarare variabili all'interno dell'istruzione for, queste saranno locali però.

È possibile fare inizializzazioni multiple separandole con la virgola, stessa cosa per gli aggiornamenti. Questo si può fare anche per le condizioni, inserendo **&&**, **||** e **!**. **for(n = 1, prodotto = 1; n <= 10 || n > 2 cond; prodotto *= n, n++)**

Il corpo del ciclo

Per progettare il corpo di un ciclo, scrivere prima le azioni che il codice deve effettuare.

Successivamente, individuare le azioni da ripetere più volte.

- Potrebbero esserci delle azioni da effettuare prima del ciclo
- Le azioni ripetute formeranno il corpo del ciclo
- Potrebbero esserci delle azioni da effettuare a ciclo concluso

Controllare il numero delle iterazioni

Se si conosce il numero di iterazioni allora si suggerisce di usare un **for**. In caso contrario si suggeriscono un **while** o un **do-while**.

Per input che consistono lunghe liste, è consigliato utilizzare un **valore sentinella**, ovvero un valore che segnala la fine della lista.

Tracciare le variabili

Tracciare le variabili significa visualizzare come le variabili cambiano mentre si sta eseguendo il programma. È sufficiente inserire nel codice istruzioni di output temporanee.

LEZIONE 11

Alcune sequenze di istruzioni devono essere ripetute più volte.

Un **metodo** raggruppa una sequenza di istruzioni che realizzano una funzionalità del programma e assegna loro un nome. E' sufficiente chiamarla ogni volta che ci serve. Per eseguire un metodo bisogna invocarlo o chiamarlo.

Esistono due tipologie di metodi:

- 1) Quelli che restituiscono un valore
- 2) Quelli che NON restituiscono un valore: **void**

STEP:

- 1) Definire il metodo: scrivere la sequenza di istruzioni e assegnare a questa sequenza un nome
- 2) Invocazione di Metodo: tramite nome
- 3) Esecuzione: delle istruzioni del metodo
- 4) Ritorno: alla posizione dove è stato eseguito il metodo

Un metodo è definito all'interno di una classe. Tutti i metodo che vedremo saranno public, ovvero possono essere invocati in ogni parte di un programma

La parola chiave static indica che è un metodo di classe (o statico).

Per definire un void, a sinistra del nome del metodo viene specificata la keyword void. Dopo il termine void , il nome del metodo e dopo le parentesi ().

Dentro le parentesi sono specificati gli argomenti di cui il metodo ha bisogno per eseguire le istruzioni.

Se non ne necessita non si inserisce nulla tra le parentesi.

Le **variabili** all'interno del void sono dette locali.

L'invocazione del metodo avviene scrivendo il nome seguito dalle parentesi e ;

Un metodo può essere chiamato **fuori o dentro** dalla classe in cui è definito.

Se viene invocato da un'altra classe bisogna specificare anche la classe

Es: void "ciao" definito in classe "miao", bisogna scrivere ciao.miao(); dentro un'altra classe.

I metodi che restituiscono un valore hanno il **tipo di valore** che contengono

Es: public static double areaCerchio();

All'interno del corpo ci sarà un valore return, ovvero il valore che deve ritornare il metodo.

gli argomenti dentro le parentesi dei metodi sono detti **parametri**. (ad esempio quando noi dobbiamo scegliere che valore assegnare).

Il valore effettivo del parametro arriva quando viene invocato il metodo.

I parametri sono locali al metodo. Vi possono essere più parametri separati dalla virgola.

La variabile nel main che viene inserita nel metodo può avere **tipo e nome diverso** (tipo bisogna vedere se non è necessario un cast, nome non da problemi). Si può castare dentro la chiamata.

La classe math viene fornita da java e non è necessario importarla.

Name	Description	Argument Type	Return Type	Example	Value Returned
pow	Power	double	double	Math.pow(2.0, 3.0)	8.0
abs	Absolute value	int, long, float, or double	Same as the type of the argument	Math.abs(-7) Math.abs(7) Math.abs(-3.5)	7 7 3.5
max	Maximum	int, long, float, or double	Same as the type of the arguments	Math.max(5, 6) Math.max(5.5, 5.3)	6 5.5

Name	Description	Argument Type	Return Type	Example	Value Returned
min	Minimum	int, long, float, or double	Same as the type of the arguments	Math.min(5, 6) Math.min(5.5, 5.3)	5 5.3
round	Rounding	float or double	int or long, respectively	Math.round(6.2) Math.round(6.8)	6 7
ceil	Ceiling	double	double	Math.ceil(3.2) Math.ceil(3.9)	4.0 4.0
floor	Floor	double	double	Math.floor(3.2) Math.floor(3.9)	3.0 3.0
sqrt	Square root	double	double	sqrt(4.0)	2.0

LEZIONE 12

Il **record di attivazione** contiene tutte le informazioni necessarie per gestire l'esecuzione del metodo:

- i **parametri formali** del metodo con gli argomenti attuali
- le **variabili locali** al metodo con i valori assunti
- l'**indirizzo di rientro**: quando termina il metodo il controllo torna al chiamante e riprende l'esecuzione da la riga dopo all'indirizzo di rientro
- il **valore** restituito nel momento in cui è un metodo con restituzione di valore

Il record di attivazione viene creato dinamicamente quando viene chiamato il metodo. Viene posto in un'area di memoria denominata **stack** e viene rimosso al termine dell'esecuzione. Se vengono invocati metodi dentro ad altri metodi si genera una sequenza di record di attivazione gestita dalla politica LIFO (last in first out), ultimo dentro primo fuori.

Si possono più usare più return nel corpo di un metodo (si preferisce usarne uno alla fine). Tipicamente un metodo void non ha return però si possono utilizzare senza valori per terminare l'esecuzione del metodo (return;)

Il modo migliore per affrontare problemi di compilazione è dividere la task in stottotask.

Per collaudare un metodo si può usare un programma driver. Forniscono al metodo una serie di argomenti e invocano il metodo stesso.

Un primo modo per collaudare è detto testing bottom-up (dal basso verso l'alto)

Es: se A invoca B, prima controlla B e poi A.

Spesso si utilizza stub per verificare la soluzione, una versione semplificata del metodo che permette il collaudo della soluzione generale.

LEZIONE 13

Un array è un tipo particolare di oggetto; è:

- una sequenza di variabili di tipo omogeneo (tutte dello **stesso tipo**)
- Le variabili nella sequenza sono **indirizzabili** in base alla loro **posizione** all'interno della sequenza

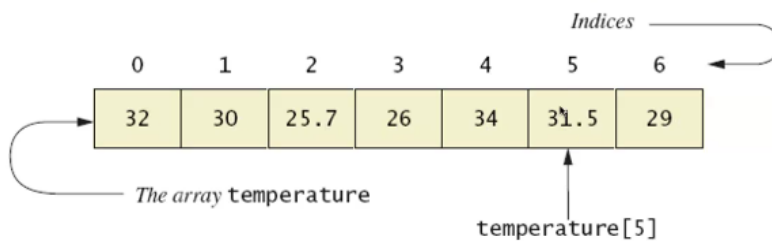
Es array:

```
double[] temperature = new double[7] //array composto da 7 variabili double
```

Dichiarazione e creazione di un array possono essere fatti separatamente

Per accedere all'elemento di un array bisogna riportare l'indice riportato tra parentesi (es. `temperature[4]`)

Gli indici degli array partono da 0.



Quattro diversi tipi di [].

1. Con il tipo di dato: `double[] pressione;` //dichiara ma non alloca memoria
2. Con il nome dell'array: `double pressione[];` //dichiara ma non alloca memoria
3. Racchiudere un tipo di dato quando si crea un nuovo array: `pressione= new double[100]` //alloca memoria
4. Indicare una variabile indicizzata dell'array: `pressione[3]`

Le proprietà dell'array sono dette variabili di istanza: `nome_oggetto.proprietà`.

L'unica proprietà accessibile è `length`. la variabile è `final`, cioè non può essere modificato. Non essendo un metodo ma variabile di istanza **NON** ha parentesi tonde

Questo serve per sapere ad esempio l'ultimo indice dell'array: `nomeArray.length-1`;

E' possibile inizializzare un array in dichiarazione:

```
double[] valore = {3.3, 15.8, 9.7};
```

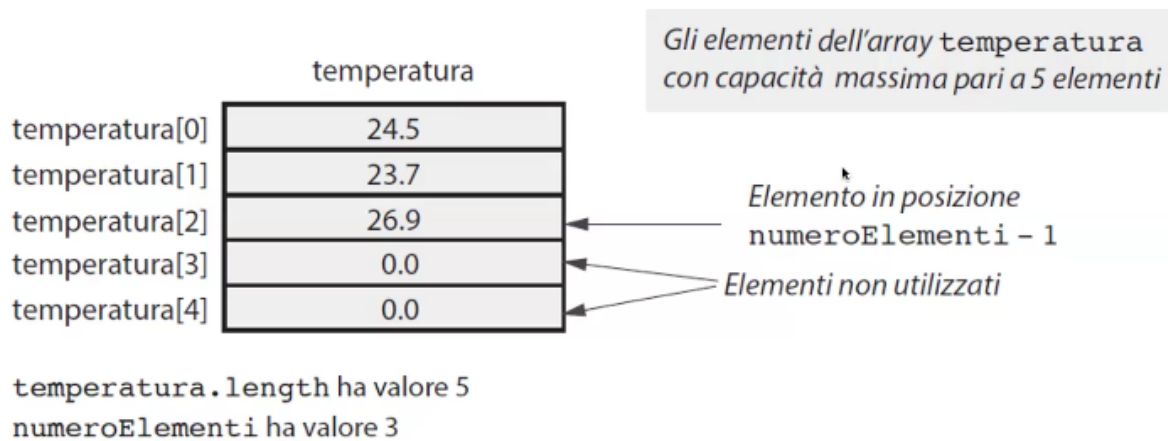
In questo caso la lunghezza viene fissata al minimo necessario per contenere i valori indicati, in questo caso 3.

Anche se non li inizializzo, vengono inizializzati con il valore di default in base al tipo di array. Ciò nonostante è preferibile eseguire l'inizializzazione. Un modo semplice per farlo è questo

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

Se un array non ha tutti gli elementi con un valore si parla di array parzialmente riempito.

La capacità dell'array e `length`, il numero di elementi creato quando abbiamo definito l'array



LEZIONE 14

Una variabile indicizzata può essere usata come argomento di un metodo in cui non si può usare il tipo base dell'array.

Il modo con cui si specifica che l'argomento di un metodo è un array è simile a come si dichiara.

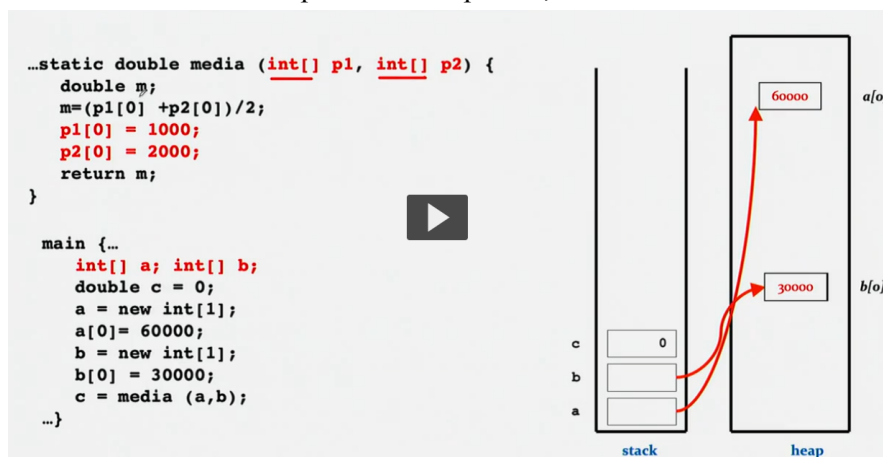
```
public static tipo nome (tipo[] nome);
```

Nell'intestazione del metodo non si specifica la lunghezza dell'array.

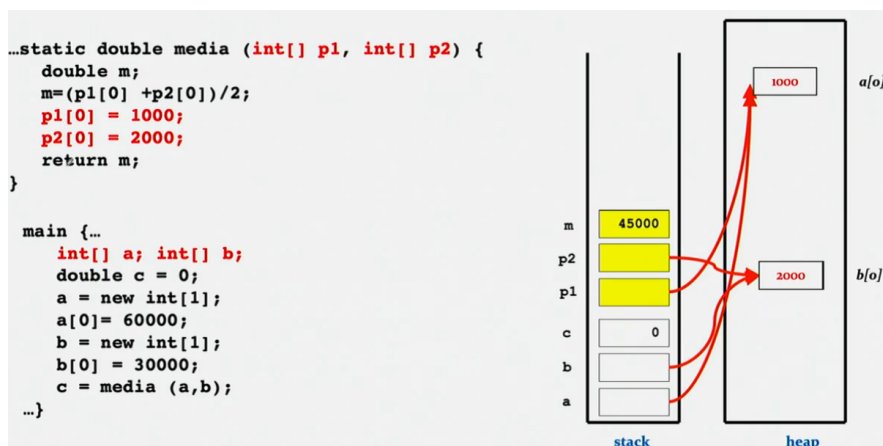
Quando si passa un intero array come argomento, non si usano le parentesi quadre, solo il nome.

Un metodo può modificare il valore degli elementi di un array passato come argomento

Main: a e b sono nell'heap con i valori passati, la media inizializzata a 0.



Media: calcolo la media con i valori passati, ma dopo li cambio a 1000 e 2000, dato che hanno lo stesso indirizzo di memoria



L'operatore di assegnamento = copia l'indirizzo dell'oggetto

L'operatore di uguaglianza == verifica se sono stati memorizzati nella stessa area di memoria

Questo indirizzo dove è memorizzato l'array è detto reference.

Un metodo può restituire un array.

```
public static char[] getVocali() {  
    char[] nuovoArray = {'a', 'e', 'i', 'o', 'u'};  
    return nuovoArray;  
}
```

Se dobbiamo memorizzare una tabella di valori si possono usare gli array multidimensionali.

Array bidimensionali: Hanno due indici[riga][colonna]

Array multidimensionali: più di un indice

Dichiarazione e creazione di array bidimensionali:
int[][] tabella = new int[10][6];


È possibile scansionare gli elementi di un array bidimensionale con due cicli innestati:

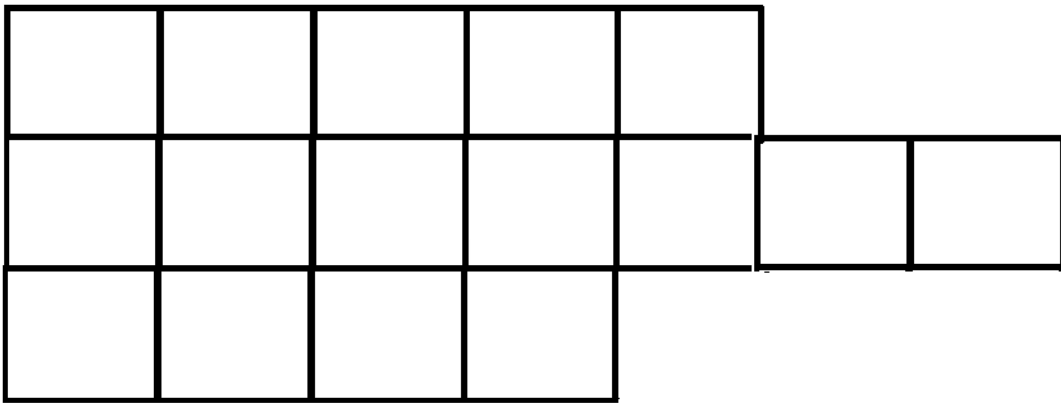
```
for (int row = 0; row < 10; row++)  
    for (int column = 0; column < 6; column++)  
        table[row][column] =  
            balance(1000.00, row + 1, (5 + 0.5 * column));
```

Un parametro del metodo o il suo ritorno possono essere array multidimensionali.

Essendo l'array multidimensionale un'array di un'array, se si usa la proprietà length, restituirà il numero di righe (primo array).

Gli array possono essere irregolari

```
int[][] b;   
b = new int[3][];  
b[0] = new int[5]; //First row, 5 elements  
b[1] = new int[7]; //Second row, 7 elements  
b[2] = new int[4]; //Third row, 4 elements
```



LEZIONE 15

Quando una parte di algoritmo contiene una versione ridotta dell'algoritmo stesso, la parte ridotta viene detta **ricorsiva**.

Un algoritmo ricorsivo può essere implementato con un metodo ricorsivo, un metodo che **chiama se stesso** dunque.

Es.

Vogliamo stampare tutti i numeri da un numero inserito all'indietro fino a 1 (conto alla rovescia). Se il numero è minore di 1, stampare una riga vuota

1. Gestire il caso più semplice, **caso base**; perchè se non funziona questo non funzioneranno le altre. In questo caso, input minore di 1.

```
public static void contoAllaRovescia(int num) {
    if (num <= 0) {
        System.out.println();
    }
}
```

2. Ora, per gestire gli altri casi, dobbiamo ridurre a una versione più semplice, ovvero dobbiamo riformulare il problema in termini di una **chiamata al metodo del caso base**.

Dato un certo numero, bisogna stampare il numero e chiamare contoAllaRovescia(num-1)

```
public class ContoAllaRovesciaRicorsivo {
    public static void main(String[] args) {
        contoAllaRovescia(3);
    }

    public static void contoAllaRovescia(int num) {
        if (num <= 0) {
            System.out.println();
        } else {
            System.out.print(num);
            contoAllaRovescia(num - 1);
        }
    }
}
```

Linee guida:

- Il nucleo della definizione del metodo deve essere costituito da un if-else o istruzioni condizionali che gestisce i casi diversi
- Almeno una delle alternative deve contenere una chiamata ricorsiva al metodo, in qualche modo più ridotto del caso originario
- Almeno una delle alternative non deve contenere una chiamata ricorsiva (caso base)

Stack:

C'è un limite alla dimensione dello stack. Se si verifica una lunga chiamata ricorsiva di un metodo ed è troppo lunga lo stack si estende oltre i limiti e si verifica lo **stack overflow**. Quindi qualche metodo ha generato una sequenza troppo lunga di chiamate ricorsive. Un esempio può essere una ricorsione infinita.

Un metodo che include una chiamata ricorsiva può essere riscritto in modo da svolgere lo stesso compito senza ricorsione. Un processo ripetitivo e non ricorsivo è definito iterazione e un metodo che lo implementa è un **metodo iterativo**.

- Il metodo ricorsivo occupa più memoria di uno iterativo.
- A causa dell'overhead a runtime (creare record di attivazione a stack) il ricorsivo è più lento dell'iterativo.
- In molti casi, però la ricorsione migliora la comprensibilità

LEZIONE 16

Le basi della ricorsione

Quando una parte di algoritmo contiene una **versione ridotta dell'algoritmo** completo, quest'ultimo è definito ricorsivo.

Un algoritmo ricorsivo può essere implementato tramite un metodo ricorsivo, ovvero un metodo che contiene una chiamata a se stesso (detta chiamata ricorsiva).

Deve essere definito correttamente altrimenti l'algoritmo potrebbe richiamare se stesso all'infinito o non farlo del tutto.

Linee guida per l'uso corretto della ricorsione

Il nucleo della definizione deve essere costituito da un blocco if-else (o altre istruzioni condizionali) che permetta di gestire casi diversi in base a qualche proprietà del parametro del metodo.

Almeno un'alternativa dovrebbe contenere una chiamata ricorsiva del metodo, almeno una delle alternative non deve contenere alcuna chiamata ricorsiva.

Casi base (o di arresto)

I **casi base** devono essere progettati in modo da terminare ogni possibile sequenza di chiamate ricorsive. Una chiamata di un metodo può produrre una chiamata ricorsiva dello stesso metodo, la quale può generare a sua volta un'altra chiamata ricorsiva e così via, per un certo numero di volte ma alla fine qualunque sequenza di questo tipo deve portare a un caso base che termina senza ulteriori chiamate ricorsive. In caso contrario, una chiamata del metodo potrebbe non terminare mai.

Lo stack e la ricorsione

Le **chiamate ricorsive** sono chiamate a metodi, l'invocazione di ognuno dei metodi comporta la creazione di un nuovo record di attivazione e il suo posizionamento in cima allo stack. Questo procedimento procede finché qualche chiamata ricorsiva del metodo non completa la propria elaborazione senza produrre ulteriori chiamate ricorsive.

C'è sempre un limite alle dimensioni dello stack.

Se si verifica una lunga catena di chiamate ricorsive di un metodo, ogni chiamata ricorsiva produrrà il salvataggio sullo stack di un'altra elaborazione sospesa.

Se la sequenza è troppo lunga, lo stack cercherà di estendersi oltre i propri limiti.

Questa condizione di errore è detta stack overflow.

Quando si ottiene un messaggio di errore di tipo stack overflow, è probabile che qualche metodo abbia generato una sequenza eccessivamente lunga di chiamate ricorsive.

Una causa tipica di questo tipo di errore è la ricorsione infinita. Se un metodo scatena una ricorsione infinita, prima o poi tenderà a far crescere lo stack oltre i limiti.

Confronto tra metodi ricorsivi e iterativi

Qualunque definizione di un metodo che include una chiamata ricorsiva può essere riscritta in modo da svolgere lo stesso compito senza ricorsione

La versione non ricorsiva implica solitamente un ciclo al posto della ricorsione

Un processo ripetitivo e non ricorsivo è definito iterazione e un metodo che implementa un processo di questo tipo è detto metodo iterativo

Un metodo ricorsivo utilizza più memoria rispetto a una versione iterativa

A causa dell'overhead a runtime, l'esecuzione di un metodo ricorsivo può essere più lenta di quella del corrispondente metodo iterativo

In molti casi la ricorsione migliora la comprensibilità di un programma

Metodi ricorsivi che ritornano un valore

Un metodo ricorsivo può restituire un valore. Stessa modalità di progettazione di un metodo void, unica differenza: almeno una delle alternative dovrebbe contenere una chiamata ricorsiva del metodo che produce il valore da restituire.