

Appunti di Fondamenti di Informatica

Giacomo Simonetto

Primo semestre 2023-24

Sommario

Appunti del corso di Fondamenti di Informatica della facoltà di Ingegneria Informatica dell'Università di Padova.

Indice

1	Storia dell'informatica	4
2	Computer	5
3	Modello di Von Neumann	5
3.1	Central Processing Unit o CPU	5
3.2	Memoria primaria e secondaria	6
3.3	Dispositivi di I/O	7
4	Sistemi operativi	8
4.1	Unix - Linux	9
5	Rappresentazione delle informazioni nei calcolatori	10
5.1	Base binaria, decimale, ottale ed esadecimale	10
5.2	ASCII e UNICODE	10
5.3	Rappresentazione in sistema posizionale	11
5.4	Rappresentazione in modulo - segno	11
5.5	Rappresentazione in complemento a 2	12
5.6	Rappresentazione in virgola fissa	13
5.7	Rappresentazione in virgola mobile	14
6	Introduzione alla programmazione	15
6.1	Algoritmo	15
6.2	Computational Thinking	15
6.3	Programmazione	15
6.4	Linguaggi di programmazione	15
7	Java	16
7.1	Introduzione	16
7.2	Struttura di un programma	16
7.3	Variabili	17
7.4	Metodi	19
7.5	Classi e oggetti	19
7.6	Interfacce	24
7.7	Commenti e JavaDoc	24
7.8	Operazioni logiche	25
7.9	Selezioni	25
7.10	Iterazioni	25
7.11	Note sulla memoria in Java	25
7.12	Reindirizzamento dei flussi i/o	25
7.13	Eccezioni in Java	25
8	Ricorsione	26
9	Complessità computazionale	26
9.1	Complessità temporale	26
9.2	Complessità spaziale	26
10	Strutture dati	27
10.1	Array	27
10.2	Matrici	27
10.3	Linked List o lista concatenata	27
10.4	ArrayList (no esame)	29

11 Strutture dati astratte - ADT	30
11.1 Container	30
11.2 Stack	30
11.3 Queue	31
11.4 Map	31
11.5 Multimap - Dictionary	31
11.6 Set	31
11.7 Table	32
11.8 Hash Table	32
11.9 Albero - no esame	33
11.10Grafì - no esame	33
12 Algoritmi di ordinamento	34
12.1 SelectionSort	34
12.2 MergeSort	34
12.3 InsertionSort	34
12.4 Confronto complessità	34
13 Algoritmi di ricerca	34
13.1 Ricerca lineare	34
13.2 Ricerca binaria	34

1 Storia dell'informatica

fine 1800	Si hanno i primi tentativi di ricerca di un linguaggio formale . La matematica è un sistema formale completo? Esiste un procedimento meccanico (passo-passo, finito) per dimostrare se una proposizione sia vera o falsa? Il primo tentativo di “ <i>formalizzazione della matematica</i> ” viene svolto da David Hilbert, con cui si scopre che la matematica possiede 23 problemi di formalizzazione chiamati “ <i>23 problemi di Hilbert</i> ”. La risposta alla prima domanda risale al 1931 quando Goedel, con il “ <i>teorema di incompletezza</i> ” conferma che la matematica non è un sistema formale.
1936	Church, Turing e Kleene elaborano dei formalismi meccanici tra cui la Macchina di Turing e la Tesi di Church-Turing che sostiene che tutto ciò che è computabile è computabile dalla macchina di Turing universale. La capacità computazionale tra una macchina di Turing e un computer odierno è la stessa (eccetto per il fatto che la macchina di Turing prevedeva uno spazio di archiviazione illimitato), cambia solo la velocità computazionale. Entrambe le macchine risolvono gli stessi problemi, ovvero tutti quelli che si possono risolvere con un algoritmo.
1943	Si arriva a costruire l' ENIAC , il primo computer (general purpose) della storia. Si programmava esclusivamente in binario, i circuiti si basavano sulle valvole termoioniche e occupava un palazzo di 5 piani.
1948	Walter Brattain, John Bardeen e William Shockley creano il primo transistor (MOSFET) in grado di sommare due bit. Grazie a ciò, durante gli anni '50 si riesce a ridurre le dimensioni dei computer a un piano.
1958	John Backus della IBM sviluppa il primo linguaggio di programmazione di alto livello Fortran per programmare uno dei computer sviluppati dall'IBM. Le novità erano quelle di poter programmare in un linguaggio simile all'inglese e l'introduzione delle selezioni e dei cicli.
1966	Viene formulato il Teorema di Jacopini-Bohm : qualsiasi algoritmo è implementabile utilizzando le strutture fondamentali di sequenza, selezione e ripetizione. In altre parole ha senso investire nell'informatica come strumento per risolvere problemi di tipo algoritmico.
1969	Viene inventato l' Internet (a carattere).
1970-71	Niklaus Wirth inventa il PASCAL , il primo linguaggio strutturato in cui scompaiono il go-to, ma a differenza dei precedenti, non può essere usato per scrivere sistemi operativi.
1970-71	Federico Faggin sviluppa il primo microprocessore .
1973	Dennis Ritchie inventa il linguaggio C , simile al Pascal, ma con la possibilità di impiegarlo per sviluppare sistemi operativi.
1977	Steve Jobs e la Apple inventano il primo personal computer .
1979	Bjarne Stroustrup sviluppa il C++ , ovvero il C con il paradigma a oggetti.
1979	Come risposta alla Apple, la IBM crea il suo primo PC. Non credendo nei PC, non volendo perdere tempo e non avendo un proprio sistema operativo, la IBM si rivolge alla Microsoft (nata nel 1974) chiedendole di sviluppare un sistema operativo per microprocessori. La Microsoft sviluppa MS-DOS (Microsoft Disk Operating System) chiedendo 50 euro per copia (praticamente nulla). Dopo 6 anni vengono vendute 300 milioni di copie e il ricavato viene investito per sviluppare Windows.
1991	Nasce internet a interfaccia grafica ed insieme ad esso c'è la necessità di avere programmi in grado di girare indipendentemente dal sistema operativo (Win, Mac OS, Unix). Si sviluppano le prime Virtual Machine in grado di eseguire codici in grado di girare su qualsiasi macchina, grazie alle Virtual Machine.
1994	Linus Torvald pubblica la prima versione stabile del kernel Linux (creato nel 1991).
1995	James Gosling nella Sun Microsystems sviluppa il linguaggio Java , dotato della particolarità di generare un codice compilato in grado di essere eseguito su qualsiasi macchina grazie alla Java Virtual Machine.

2 Computer

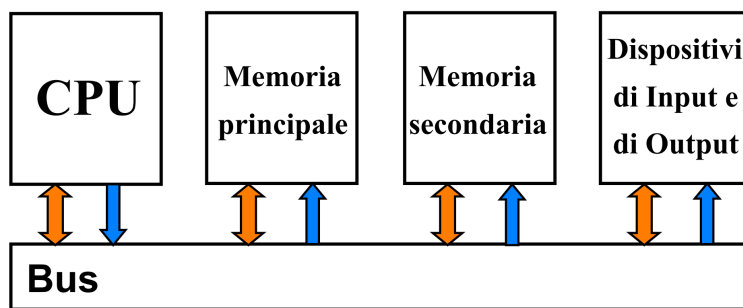
Per computer, o calcolatore, si intende un sistema di elaborazione e memorizzazione di informazioni che opera sotto il controllo di un programma. È composto da hardware (parte fisica) e software (programmi e dati). I dati possono essere di diverso tipo (immagini, testi, audio, video, ...) e sono rappresentati elettricamente in 0 e 1.

Esistono diversi tipi di computer (workstation, smartphone, ...) che possono svolgere diversi tipi di impieghi (elettrodomestici, giochi, fotografie, ...).

3 Modello di Von Neumann

Il modello di Von Neumann è una rappresentazione dell'architettura di un elaboratore. Prevede la presenza di 4 blocchi: la CPU, la memoria principale, la memoria secondaria e i dispositivi di I/O collegati insieme grazie al BUS.

Inoltre sono presenti due diversi flussi di informazioni: quello di dati è bidirezionale (nello schema è rappresentato dalle frecce arancioni), mentre quello degli indirizzi e dei segnali di controllo è unidirezionale con direzione CPU → altri blocchi (nello schema è rappresentato dalle frecce blu).



3.1 Central Processing Unit o CPU

La Central Processing Unit ha il compito di:

- individuare ed eseguire le istruzioni
- elaborare dati attraverso la ALU (Unità Logico Aritmetica)
- reperire dati di input e restituire dati di output

Componenti

È costituita da tre blocchi:

Control Unit	o <i>CU</i> , gestisce l'esecuzione dei programmi e i flussi di dati
ALU	o <i>Arithmetic Logical Unit</i> , elabora le espressioni logiche e algebriche
memorie temporanee per dati che devono essere subito elaborati:	
- l' Accumulator , o <i>ACC</i> , che memorizza i dati elaborati o che stanno per essere elaborati dalla <i>ALU</i>	
- il Program Counter , o <i>PC</i> , che memorizza l'indirizzo di memoria dell'istruzione successiva da eseguire	
Registri	- l' Instruction Register , o <i>IR</i> , che memorizza l'istruzione da decodificare
	- il Memory Data Register , o <i>MDR</i> , che memorizza i dati/le istruzioni lette o che stanno per essere scritte nella memoria primaria
	- il Memory Address Register , o <i>MAR</i> , che memorizza l'indirizzo di memoria dell'istruzione da eseguire o del dato da utilizzare

Funzionamento

La CPU ha funzionamento ciclico che si divide in tre fasi. La velocità di una CPU, chiamata frequenza di clock è espressa in cicli al secondo (dell'ordine dei GHz) ed è scandita dal *Clock*. La velocità massima è dovuta ai limiti fisici della tecnologia disponibile.

1° fase	fetch	<ul style="list-style-type: none">- viene letto l'indirizzo dell'istruzione da eseguire dal <i>PC</i> e viene salvato nel <i>MAR</i>- viene incrementato il <i>PC</i> in modo che punti all'istruzione successiva- viene letta e caricata l'istruzione prima nel <i>MDR</i> poi nell'<i>IR</i>
2° fase	decode	<ul style="list-style-type: none">- la <i>CU</i> decodifica l'istruzione salvata nell'<i>IR</i>- se necessario viene caricato nel <i>MAR</i> l'indirizzo del dato da elaborare o della posizione in cui scrivere il dato elaborato
3° fase	execute	<ul style="list-style-type: none">- viene eseguita l'istruzione:- se necessario viene caricato nel <i>MDR</i> il dato referenziato dal <i>MAR</i>- il dato può essere salvato nell'<i>ACC</i> o impiegato in un'operazione logico-algebrica eseguita dalla <i>ALU</i>- il risultato viene salvato nell'<i>ACC</i>- oppure il dato memorizzato nell'<i>ACC</i> viene scritto nell'indirizzo di memoria contenuto nel <i>MAR</i>

Limiti e parallelismo

I limiti della CPU sono principalmente due: la frequenza di clock e l'impossibilità di eseguire un'istruzione, finché non viene completata quella precedente. Per superare il secondo problema si sono cercate soluzioni come il parallelismo. Esistono due tipi di parallelismo:

- **parallelismo a livello di istruzioni:**

detto anche pipeline o multiscalari, consiste nel suddividere il ciclo di un processore in 5 stadi (lettura, decodifica, recupero operandi, caricamento, esecuzione, invio risultati) e di eseguire contemporaneamente più istruzioni su stadi diversi. In questo modo non è necessario aspettare che l'esecuzione dell'istruzione precedente termini per iniziare quella della successiva, ma è sufficiente che sia completato il primo stadio.

- **parallelismo a livello di processori:**

consiste nell'avere più processori che lavorano contemporaneamente in grado di eseguire più istruzioni nello stesso momento. In base all'architettura si distinguono in multiprocessori (se sono presenti più processori che condividono la stessa memoria) o multicomputer (se sono più processori, ciascuno con la propria memoria dedicata, collegati tra loro).

3.2 Memoria primaria e secondaria

La memoria ha il compito di memorizzare dati e programmi, sia in maniera temporanea, che permanente.

Struttura

La memoria è composta da celle chiamate allocazioni di memoria. Ogni allocazione può contenere un preciso numero di bit. Un bit (abbreviazione di Binary Digit) è l'unità minima di dimensione della memoria e corrisponde allo spazio occupato da 0 o 1. Il bit è un sottomultiplo del byte, 1byte = 8bit. Il byte è l'unità minima di accesso singolo alla memoria ed è l'unità base per la misura della dimensione dello spazio di archiviazione.

Memoria primaria

La memoria primaria è la più veloce delle due, ma anche la più costosa. Ne esistono due tipi:

RAM	o <i>Random Access Memory</i> , memoria volatile, dotata della caratteristica di avere un tempo di accesso ad una cella indipendente dal luogo in cui essa si trova (tempo di accesso “casuale”). Viene impiegata per salvare dati temporanei derivati dall'esecuzione di programmi.
ROM	o <i>Read Only Memory</i> , memoria permanente di sola lettura in cui vengono salvati i programmi necessari all'avvio della macchina, es. BIOS (<i>Basic Input Output System</i>)
Cache	o <i>memoria di località</i> , memoria estremamente veloce che permette di memorizzare celle di memoria che potenzialmente potrebbero tornare utili nelle future elaborazioni. Esistono due tipi di località: <ul style="list-style-type: none">- località temporale: accedere alla stessa cella in tempi vicini- località spaziale: accedere a celle limitrofe

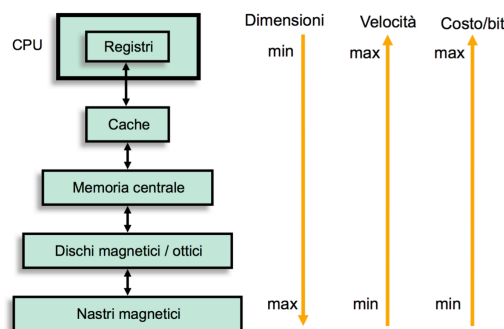
Memoria secondaria

Memoria non volatile, più lenta e molto meno costosa della memoria primaria. È riservata all'archiviazione di file, dati, programmi (tra cui anche il sistema operativo) che vengono trasferiti nella *RAM* al momento dell'esecuzione. Esistono diversi supporti di archiviazione di memoria secondaria:

- *HDD* o disco magnetico
- *SSD* o disco a stato solido (solid state drive)
- dischi ottici come *CD*, *DVD*, *Blue-Ray*
- chiavette USB
- nastri magnetici, impiegati per l'archiviazione di documenti, sono molto lenti, ma hanno costo molto basso ed elevata capacità di archiviazione

Gerarchie di memoria

Maggiore è la dimensione, minore è la velocità ed il costo.



3.3 Dispositivi di I/O

Permettono l'interazione dell'essere umano con la macchina. Comprendono mouse, tastiera, touchpad, schermo, stampante, ... Le operazioni relative ai dispositivi di I/O sono:

polling	o controllo da programma, consiste nel ripetuto e periodico controllo dello stato dei dispositivi
interrupt	richiama l'attenzione della CPU attraverso un'interruzione del flusso di esecuzione
DMA	o <i>Direct Memory Access</i> , dispositivo indipendente dalla CPU che gestisce il flusso di dati dei dispositivi di I/O ed alleggerisce il carico della CPU (la CPU indica solo indirizzi e dati da spostare)

4 Sistemi operativi

Cos'è e a cosa serve

Un sistema operativo, abbreviato *SO*, è un insieme di software che fornisce all'utente una serie di comandi e servizi per usufruire della potenza di calcolo di un elaboratore elettronico, inoltre garantisce l'operatività di base di un elaboratore, coordinando e gestendo le risorse hardware di elaborazione e memorizzazione, le periferiche, le risorse/attività software e facendo da interfaccia con l'utente, senza il quale quindi non sarebbe possibile l'utilizzo del computer stesso e dei programmi. Ogni sistema operativo è legato ad uno specifico hardware.

Bootstrap

Il bootstrap è la fase in cui viene avviato il sistema operativo, generalmente all'avvio del computer. La procedura di bootstrap è memorizzata nella *ROM*, all'avvio del computer:

- la CPU legge le istruzioni dalla ROM
- recupera il sistema operativo dal disco (memoria secondaria)
- carica il sistema operativo nella RAM
- avvia l'esecuzione dei programmi che ne permettono il funzionamento

Struttura a cipolla

Il sistema operativo è organizzato su più strati (come una cipolla), ciascuno con la caratteristica di poter interfacciarsi soltanto con quelli più interni. Questo garantisce modularità, flessibilità e più facile manutenzione. Gli strati, dal più interno sono:

nucleo o core	gestisce le risorse fisiche, comunica con l'hardware ed è scritto in linguaggio macchina
gestore I/O	gestisce i dispositivi di input e output e si occupa di trasferire i dati tra le diverse memorie del computer
gestore memoria	gestisce l'allocazione delle memorie durante l'esecuzione dei programmi
gestore archiviazione	anche chiamato filesystem, organizza la struttura di archiviazione dei file
interfaccia utente	permette all'utente di interagire con la macchina attraverso un'interfaccia grafica (GUI) o a linea di comando (CLI)

Comandi e linguaggi di controllo

Ogni sistema operativo possiede un linguaggio di controllo, ovvero un insieme di comandi che permette di interfacciarsi con il sistema operativo, eseguire operazioni o programmi, controllare le attività in corso e lo stato della macchina. I comandi sono impartiti dall'utente attraverso il terminale o attraverso l'interazione con l'interfaccia grafica. In Windows i comandi riprendono il vecchio sistema MS-DOS.

4.1 Unix - Linux

Introduzione

Linux è un sistema operativo sviluppato nel 1994 da Linus Torvald, basandosi su UNIX. Unix è un sistema operativo proprietario, Linux è la corrispettiva versione di Unix, ma open source.

Utenti e permessi

Da sempre Linux e Unix sono sistemi multiutente, ovvero ciascun file ha un utente proprietario e ogni utente può accedere e modificare solo dove è permesso. L'utente che non ha limitazioni è chiamato **root**.

Filesystem

Il filesystem è organizzato con una struttura ad albero capovolto, in cui la cartella, o *directory*, principale, che contiene tutti i file e le directory del sistema, è chiamata **root**. Ogni elemento nel filesystem è raggiungibile attraverso un percorso chiamato *path*. Il percorso della cartella **root** è `\`.

I path si distinguono in:

- **percorso assoluto:**
ovvero il percorso che separa la cartella **root** dal file in questione, inizia con `\`, ovvero il simbolo della cartella **root**, cioè con `/`, es. `/user/nomeutente/home/desktop/file.txt`
- **percorso relativo:**
ovvero il percorso che separa una cartella diversa dalla **root** dal file in questione, inizia con il nome della cartella di partenza, es. `desktop/file.txt` rispetto alla **home**

Il percorso per accedere alla stessa cartella è `./`, quello per accedere alla cartella di livello superiore è `../`

Shell o CLI

La Shell è l'interfaccia utente a linea di comando. In Linux/Unix sono presenti diverse shell: *bash*, *csh*, *ksb*, *zsh*, in base alla distribuzione utilizzata (in Windows è quella di *MS-DOS*).

I comandi della CLI si dividono in *builtin*, che sono presenti di default nell'OS, ed *esterni* che possono essere installati in un secondo momento dall'utente.

In Linux/Unix sono presenti dei metacaratteri come il simbolo `*`, che rappresenta una sequenza di uno o più caratteri, e il simbolo `?`, che rappresenta un singolo carattere.

Alcuni comandi di Linux/Unix sono:

<code>cd</code>	change directory
<code>ls</code>	list files and subdirectory of the current directory
<code>cp</code>	copy file
<code>mv</code>	move or rename file
<code>rm</code>	remove file
<code>mkdir</code>	make new directory
<code>rmdir</code>	remove directory
<code>kill</code>	end process
<code>sudo</code>	per eseguire comandi dall'utente root
<code>man</code>	manuale
<code>appropos</code>	ricerca comandi
<code>whatis</code>	descrizione comando

5 Rappresentazione delle informazioni nei calcolatori

tipo di dato	rappresentazione
numeri naturali \mathbb{N}	rappresentazione secondo il sistema posizionale
numeri interi \mathbb{Z}	rappresentazione modulo - segno rappresentazione in complemento a due
numeri reali \mathbb{R}	rappresentazione in virgola fissa rappresentazione in virgola mobile (singola e doppia precisione)
caratteri	tabella ASCII o Unicode

Dato che un computer può elaborare soltanto un numero finito di informazioni, mentre i numeri sono infiniti, significa che ci sarà un valore massimo e un valore minimo rappresentabile. Inoltre viene introdotto un errore dato dal fatto che non tutti i numeri hanno un numero di cifre limitate (es. π o $\sqrt{2}$).

5.1 Base binaria, decimale, ottale ed esadecimale

La rappresentazione decimale utilizza le cifre da 0 a 9 (10 simboli), quella binaria solo 0 e 1 (2 simboli). Le altre rappresentazioni si comportano allo stesso modo. Nella base ottale vengono impiegate le cifre da 0 a 7 (8 simboli), in quella esadecimale da 0 a F (16 simboli). Verranno approfonditi soltanto i diversi sistemi di rappresentazione binaria e le conversioni con quella decimale.

5.2 ASCII e UNICODE

Il codice ASCII e quello UNICODE sono sistemi che associano ad ogni simbolo (carattere) un numero senza segno. Nello standard ASCII vengono riservati 7 bit per ogni carattere per un massimo di 128 simboli (poi estesi a 256 con l'extended ASCII), mentre per l'UNICODE si impiegano 2 byte per un massimo di 65536 simboli rappresentabili. Il sistema UNICODE, ad oggi il più usato, comprende il vecchio codice ASCII.

5.3 Rappresentazione in sistema posizionale

Si sfrutta il principio che ogni cifra possiede un peso dato dalla posizione relativa nel numero. Nel sistema decimale ogni cifra ha, come peso, una potenza di 10, in quello binario si usano le potenze di 2.

Rappresentazione

DEC	234_{10}	$= 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$
BIN	11101010_2	$= 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

Valori rappresentabili

Il range di valori rappresentabili con n cifre è $[0, 2^n - 1]$.

Conversione decimale - binario

Per eseguire la conversione dal sistema decimale a quello binario, bisogna usare l'algoritmo di conversione:

```
1 while (numero != 0)
2     resto_nesimo = numero % base
3     numero = numero / base
```

Il numero convertito si ottiene giustapponendo i resti ottenuti al contrario, in modo che l'ultimo resto diventi la cifra più significativa e il primo resto diventi quella meno significativa.

Conversione binario - decimale

Per convertire un numero dal sistema binario a quello decimale, basta associare ciascuna cifra al suo peso in potenza di 2 e sommare i valori ottenuti.

$$\begin{aligned} 11101010_2 &= 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= 1 \cdot 128 + 1 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 \\ &= 234_{10} \end{aligned}$$

5.4 Rappresentazione in modulo - segno

Con il sistema posizionale non è possibile rappresentare valori negativi, per cui viene introdotta la rappresentazione modulo-segno. Tale sistema prevede di riservare il primo bit al segno del numero ed i restanti per il modulo rappresentato con il sistema decimale.

Rappresentazione

$\text{numeri} \geq 0$	$+12_{10} = +1 \cdot 1100_2 = 0 + 1100_2 = 01100_{2MS}$
$\text{numeri} \leq 0$	$-12_{10} = -1 \cdot 1100_2 = 1 + 1100_2 = 11100_{2MS}$

Valori rappresentabili

Il range di valori rappresentabili con n cifre è $[-(2^{n-1} - 1); +(2^{n-1} - 1)]$.

Si osserva che lo 0_{10} possiede due rappresentazioni: 10000_{2MS} e 00000_{2MS} per $n = 4$.

Conversioni

Per le conversioni da sistema decimale a binario e viceversa si ricorre al procedimento illustrato nel sistema posizionale, con l'unica particolarità di far corrispondere uno 0 davanti al numero binario se il numero decimale è positivo, oppure un 1 se il numero decimale è negativo.

Criticità

Questo sistema di rappresentazione non viene utilizzato in quanto l'algoritmo per eseguire somme (e sottrazioni) è poco efficiente e complesso.

5.5 Rappresentazione in complemento a 2

Con la rappresentazione in complemento a 2 è possibile rappresentare numeri interi positivi e negativi, eliminando la doppia rappresentazione dello zero e semplificando l'algoritmo di somma (e differenza), avendo sempre i positivi e lo 0 che iniziano per 0 e i negativi che iniziano per 1.

Rappresentazione

$$\text{numeri} \geq 0 \quad +12_{10} = 0 + 1100_2 = 01100_{C2}$$

$$\text{numeri} < 0 \quad -12_{10} = 10100_{C2} \quad \rightarrow \quad -12 + 32 = 20_{10} = 10100_2$$

Valori rappresentabili

Il range di valori rappresentabili con n cifre è $[-2^{n-1}; 2^{n-1} - 1]$.

Conversione decimale - binario

Per i numeri positivi, compreso lo 0, si impiega il classico sistema posizionale (aggiungendo uno 0 davanti al numero per il segno), mentre per i numeri negativi è necessario:

1. sommare 2^n con n numero di cifre in binario, in modo da rendere il numero positivo
2. convertire il risultato secondo il sistema posizionale (se i conti sono giusti il numero inizierà per 1)

$$+12_{10} = 0 + 1100_2 = 01100_{C2}$$

$$-12_{10} \rightarrow -12 + 2^5 = 20_{10} = 10100_2$$

$$-12_{10} = 10100_{C2}$$

In alternativa se il numero è negativo

1. convertire il modulo secondo il sistema posizionale
2. aggiungerci uno 0 davanti
3. invertire le cifre (gli 0 diventano 1 e gli 1 diventano 0)
4. sommarci 1 (se i conti sono giusti il numero inizierà per 1)

$$-12_{10} \rightarrow 1100_2 \rightarrow 00011 \rightarrow 10011 \rightarrow 10011 + 1 \rightarrow 10100$$

$$-12_{10} = 10100_{C2}$$

Conversione binario - decimale

Per i numeri che iniziano per 0 basta eseguire la conversione per sistema posizionale, per quelli che iniziano con 1 bisogna convertire il numero secondo il sistema posizionale e toglierci 2^n con n = cifre del numero.

$$\begin{aligned} 01100_{C2} &= 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^3 + 0 \cdot 2^1 + 0 \cdot 2^0 \\ &= 8 + 4 \\ &= 12_{10} \end{aligned}$$

$$\begin{aligned} 10100_{C2} &= 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^3 + 0 \cdot 2^1 + 0 \cdot 2^0 - 2^5 \\ &= 16 + 4 - 32 \\ &= 20 - 32 \\ &= -12_{10} \end{aligned}$$

5.6 Rappresentazione in virgola fissa

La rappresentazione in virgola fissa riprende il principio del sistema posizionale, usando potenze con esponenti negativi per le cifre dopo la virgola.

Rappresentazione

DEC	234.56_{10}	$= 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 + 5 \cdot 10^{-1} + 6 \cdot 10^{-2}$
BIN	11101010.1001_2	$= 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 +$ $+ 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}$

Valori rappresentabili

Il range di valori rappresentabili è $\{0\} \cup [2^{-m}, 2^n - 1]$, con n cifre intere, m cifre decimali.

Conversione decimale - binario

Per eseguire la conversione dal sistema decimale a quello binario, è necessario dividere la parte intera da quella dopo la virgola. Per la prima basta convertirla con l'algoritmo visto per il sistema posizionale, mentre per la parte decimale è necessario applicare il seguente "algoritmo":

```
1 while (numero != 0)
2     parteIntera_n-esima = parteIntera(numero)
3     numero = parteDecimale(numero) * base
```

Per ottenere il numero convertito è necessario prendere le parti intere in ordine (senza invertirle). Il risultato potrebbe essere un numero illimitato.

Conversione binario - decimale

Per convertire un numero dal sistema binario a quello decimale, basta associare ciascuna cifra al suo peso in potenza di 2 e sommare i valori ottenuti.

$$\begin{aligned} 11101010.1001_2 &= 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + \\ &\quad + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} \\ &= 1 \cdot 128 + 1 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 + \\ &\quad + 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 0 \cdot \frac{1}{8} + 1 \cdot \frac{1}{16} \\ &= 128 + 64 + 32 + 8 + 2 + 0.5 + 0.0625 \\ &= 234.5625_{10} \\ &\approx 234.56_{10} \end{aligned}$$

Criticità

Non tutti i numeri sono rappresentabili in un numero finito di cifre. Alcuni numeri che nel sistema decimale hanno un numero finito di cifre, nel sistema binario potrebbero essere illimitati, per cui la loro rappresentazione potrebbe essere un'approssimazione, ad esempio $4.35_{10} = 100.0101100_2$.

Inoltre è poco efficiente in quando per rappresentare numeri molto grandi, la parte decimale sarebbe poco significativa e i bit riservati a tale parte si potrebbero usare per la parte intera. Viceversa per numeri prossimi allo 0.

5.7 Rappresentazione in virgola mobile

Utilizza la notazione esponenziale con mantissa ed esponente che permette maggiore flessibilità per numeri molto grandi e numeri prossimi allo 0. Lo standard *IEEE 754* prevede due rappresentazioni: a singola e a doppia precisione.

Rappresentazione

DEC	$234,56_{10}$	$= 0,23456 \cdot 10^4$
BIN	$11101010,1001_2$	$= 0,111010101001 \cdot 2^8$

In Java un numero di virgola mobile è rappresentato nella forma $s \cdot m \cdot 2^{(e-N+1)}$ con:

- s segno, può assumere soltanto i valori $-1, +1$
- m mantissa nell'intervallo $[0, 2^N - 1]$
- e esponente nell'intervallo $[-(2^{K-1} - 2), 2^{K-1} - 1]$
- N numero di bit riservati alla base
- K numero di bit riservati all'esponente

Suddivisione in bit

precisione	segno	mantissa	esponente	complessivo
singola	"1 bit"	24 bit	8 bit	32 bit
doppia	"1 bit"	53 bit	11 bit	64 bit

Densità

La densità nel sistema in virgola mobile è la distanza tra due numeri adiacenti in formato binario.

Per la singola precisione, avere 23 bit di mantissa, significa che posso suddividere l'intervallo da $0.000 \dots 1_2 \cdot 2^n, 0.111 \dots 1_2 \cdot 2^n$ in 2^{23} parti. Queste parti saranno più fitte per esponenti bassi e meno fitte per esponenti alti. Per cui la densità è data dalla formula $\delta = 2^{23} \cdot 2^E$, dove E è l'esponente binario dei numeri da rappresentare.

Valori rappresentabili

precisione	mantissa	esponente	minimo	massimo	densità
singola	$[0, 2^{24} - 1]$	$[-126, 127]$	$1.4 \cdot 10^{-45}$	$3.4 \cdot 10^{38}$	$\delta = 2^{23} \cdot 2^E \approx 8 \cdot 10^6 \cdot 2^E$
doppia	$[0, 2^{53} - 1]$	$[-1022, 1023]$	$4.9 \cdot 10^{-324}$	$1.8 \cdot 10^{308}$	$\delta = 2^{52} \cdot 2^E \approx 4.5 \cdot 10^{15} \cdot 2^E$

Valori limite

Alcune combinazioni di mantissa ed esponente sono catalogate per valori particolari come:

0: esponente minimo, mantissa 0

∞ : esponente massimo, mantissa 0

NaN: esponente massimo, mantissa 1 (*Not A Number*)

Criticità

Non tutti i numeri sono rappresentabili con un numero finito di cifre, ma vengono introdotte approssimazioni, come nel caso di 4.35 (come visto precedentemente). Per questo motivo, quando si fanno confronti tra valori in virgola mobile è necessario introdurre un errore entro cui due numeri sono uguali.

Inoltre nel caso in cui si lavora con numeri molto grandi a cui vengono sommati numeri molto piccoli, si rischia di non avere abbastanza precisione per eseguire correttamente la somma, rischiando di perdere il valore dell'addendo minore. Questo avviene soprattutto quando l'addendo più piccolo è minore della precisione del numero più grande.

6 Introduzione alla programmazione

6.1 Algoritmo

Un algoritmo è un metodo di risoluzione di un problema che:

- deve essere eseguibile
- non deve essere ambiguo
- deve concludersi in un numero finito di passi

6.2 Computational Thinking

Per computational thinking, o pensiero computazionale, si intende l'insieme delle abilità che permettono di astrarre il problema e tradurlo in algoritmo. Comprende le tecniche di astrazione/risoluzione di problemi algoritmici tra cui la decomposizione di problemi complessi e la modularità.

6.3 Programmazione

Per programmazione si intende il processo di progettazione e codifica di programmi per calcolatori.

Per programma si intende l'implementazione di algoritmi in un qualche linguaggio per calcolatore.

Un programma è corretto se possiede correttezza semantica, correttezza sintattica e correttezza logica.

Il teorema di Jacopini-Bohm sostiene che un programma/algoritmo è composto da istruzioni imperative, istruzioni condizionali, o decisioni, e iterazioni, o cicli.

6.4 Linguaggi di programmazione

Per linguaggio di programmazione si intende un sistema per la scrittura di programmi/algoritmi eseguibili da un calcolatore. Deve essere rigoroso e non avere ambiguità.

Linguaggio macchina

Il linguaggio macchina è il primo esempio di linguaggio di programmazione, è diverso per ogni tipo di processore e le istruzioni sono composte da sequenze di bit. Le istruzioni si dividono in tre categorie:

- trasferimento dati (es. `LOAD` per registri, `STORE` per memorie)
- operazioni aritmetiche e logiche (es. `ADD`, `SUB`, `MUL`, `DIV`, `AND`, `OR`, `NOT`)
- salti (es. `JUMP`, `JZ` se == 0, `JGZ` se > 0)

Linguaggi assembly e di basso livello

Permettono di scrivere codice macchina attraverso codici mnemonici più simili al linguaggio umano, per cui più semplice da utilizzare. C'è una corrispondenza biunivoca tra l'instruction set del processore e il linguaggio assembly (o basso livello), per cui è diverso per ogni processore. Le istruzioni sono convertite in linguaggio macchina da un assembler. Rimane sempre il problema della portabilità.

Linguaggi di alto livello

Sono linguaggi formali (artificiali), espressivi, non ambigui, efficienti e molto più leggibili e versatili dei precedenti in quanto simili al linguaggio parlato. Posseggono precise regole grammaticali (lessicali, sintattiche e semantiche) definite dalla notazione EBNF (Extended Backus-Naur form). Sono convertiti in linguaggio macchina da un compilatore. Si dividono in:

- **linguaggi compilati:**
il programma (indipendente dalla CPU) è convertito in codice eseguibile (dipendente dalla CPU) da un compilatore, garantisce alta velocità, ma bassa portabilità
- **linguaggi interpretati:**
le istruzioni tradotte in linguaggio macchina da un interprete durante l'esecuzione, garantisce alta portabilità, ma lenta velocità di esecuzione

Breve storia sui linguaggi di programmazione

anni 50	primi linguaggi di programmazione ad alto livello (FORTRAN, BASIC, COBOL), viene coniato il termine <i>spaghetti code</i> per l'inappropriato uso delle istruzioni di salto (<code>GOTO</code>)
anni 60-70	programmazione strutturata con istruzioni imperative, selezioni e iterazioni, come previsto dal teorema di Jacopini-Bohm (1966), compaiono nuovi linguaggi strutturati PASCAL (1968) e C (1970-75)
anni 80-90	si sviluppa il paradigma di programmazione ad oggetti con nuovi linguaggi come C++ (1979) e Java (1991) in grado di supportare la creazione di classi e oggetti

7 Java

7.1 Introduzione

Java è un linguaggio di programmazione ad oggetti, sviluppato da James Gosling nel 1995. La sua particolarità è di essere un linguaggio fortemente tipizzato come il C o C++, ma più semplice e flessibile in quanto la gestione della memoria non deve essere fatta dal programmatore, ma dal Garbage Collector e presenta librerie standard molto ricche.

Non è né un linguaggio interpretato, né un linguaggio compilato. Un file con un programma scritto in java si presenta con l'estensione `.java`, viene pseudocompilato da un compilatore java e convertito in un file con estensione `.bytecode`. Il file bytecode può essere eseguito da un computer con installata la Java Virtual Machine, o JVM, ovvero l'ambiente di esecuzione che funge da interprete del file bytecode.

Per questo motivo è più veloce di un linguaggio interpretato, ma meno veloce di uno compilato e meno portabile di uno interpretato, ma più portabile di uno compilato.

Per compilare un file java si utilizza il comando `javac NomeFile.java`, per eseguire un file bytecode si usa `java NomeFile`

7.2 Struttura di un programma

Un programma in java è composto da diversi elementi (introduzione):

classi	contenitori di metodi e variabili, possono essere eseguibili se contengono un metodo <code>main</code> , come fabbriche di oggetti o entrambe (anche se poco raccomandabile); le classi sono organizzabili in pacchetti e sono importabili in file diversi da quello in cui sono implementate attraverso la parola chiave <code>import</code>
metodi	sequenze di istruzioni, chiamati anche funzioni o procedure in altri linguaggi; devono necessariamente essere contenuti in una classe, il metodo principale <code>main</code> viene eseguito all'avvio del programma
variabili	spazi di memoria riservati per memorizzare dati, ogni variabile può contenere solo un particolare tipo di dato es. <code>int</code> , <code>double</code> , <code>char</code> , ... il valore di una variabile può essere modificato nel corso dell'esecuzione del programma
costanti	spazi di memoria riservati per memorizzare valori che rimangono costanti per tutta la durata del programma, come le variabili, anche le costanti sono tipizzate
literals	insieme di valori numerici e stringhe di testo che sono contenuti nel programma
oggetti	istanze di una classe, sono variabili particolari in grado di avere dei metodi in grado di compiere operazioni su di esse, es. stringhe, array, strutture dati
commenti	parti di testo che vengono ignorate dal compilatore al momento della compilazione, si utilizzano per descrivere la funzione di una determinata parte di codice
strutture logiche	insieme di costrutti che permettono di compiere selezioni e iterazioni, es. <code>if</code> , <code>else</code> , <code>while</code> , <code>for</code> , ...
parole chiave	insieme di parole riservate con una funzione logica ben precisa, es. le strutture logiche, tipi delle variabili, <code>import</code> , <code>final</code> , <code>private</code> , <code>public</code> , <code>new</code> ...

7.3 Variabili

Una variabile è uno spazio di memoria riservato per memorizzare dati (numerici, logici o caratteri) che possono variare durante l'esecuzione del programma. Una variabile può contenere solo dati di specifico tipo indicato nella dichiarazione.

Dichiarazione, inizializzazione e assegnazione

- **dichiarazione:** `int a;`
viene riservato uno spazio in memoria in grado di contenere valori del tipo indicato al posto di **tipo**
- **inizializzazione:** `int a = 0;`
viene dichiara una variabile e le viene assegnato un valore iniziale
- **assegnazione:** `a = 345;`
viene salvato un valore in una variabile

Visibilità

Le variabili vengono create al momento della loro dichiarazione e vengono distrutte alla chiusura del blocco in cui sono state create. Inoltre sono accessibili solo all'interno del blocco (e sottoblocchi) in cui sono state dichiarate.

Tipi di dato

tipo	min	max	mem	descrizione
byte	-128	127	8 bit	numeri interi in complemento a due
short	-32 768	32 767	16 bit	numeri interi in complemento a due
int	$-2.15 \cdot 10^9$	$2.15 \cdot 10^9$	32 bit	numeri interi in complemento a due
long	$-9.22 \cdot 10^{18}$	$9.22 \cdot 10^{18}$	64 bit	numeri interi in complemento a due
float	$1.40 \cdot 10^{-45}$	$3.40 \cdot 10^{38}$	32 bit	numeri decimali in singola precisione
double	$4.94 \cdot 10^{-324}$	$1.80 \cdot 10^{308}$	64 bit	numeri decimali in doppia precisione
char	0	65535	16 bit	numeri interi ≥ 0 per standard UNICODE
boolean	false	true	1 bit	valori booleani true, false

Promozione e casting

Per convertire un valore di un determinato tipo di dato in un altro, esistono due operazioni:

- **promozione:** `double a = 123`
sempre concesso e non genera errori, avviene quando si converte un numero con minore precisione in un formato con maggiore precisione
- **casting:** `int a = (int)123,45`
se non correttamente gestito può generare errori di perdita di precisione o overflow, avviene quando si converte un numero con maggiore precisione in un formato con minore precisione, per eseguire il casting è necessario indicare tra parentesi il nuovo tipo di dato, davanti al valore da convertire

Overflow

L'overflow avviene quando si supera il valore massimo (o minimo) consentito da un determinato tipo di dato. Nei numeri interi, i valori hanno comportamento "ciclico" quando si supera il valore superiore, si ricomincia dal valore inferiore e viceversa.

Operazioni

somma	<code>a + b</code> <code>a++</code>	viene eseguita la somma (possibile overflow) al variabile viene incrementata di uno
differenza	<code>a - b</code> <code>a--</code>	viene eseguita la differenza la variabile viene decrementata di uno
prodotto	<code>a * b</code>	viene eseguito il prodotto
divisione	<code>a / b</code>	viene eseguita la divisione, se entrambe le variabili sono di tipo <code>int</code> viene eseguita la divisione intera (tralasciando la parte decimale), se almeno una delle due è <code>double</code> , allora viene eseguita la divisione reale, se <code>b</code> è 0 e le variabili sono intere, viene lanciata una <code>ArithmeticException</code>
modulo	<code>a % b</code>	calcola il resto della divisione tra <code>a</code> e <code>b</code> , se <code>b</code> è 0 e le variabili sono intere, viene lanciata una <code>ArithmeticException</code>
confronti	<code>a == b</code> <code>a < b</code> <code>a <= b</code> <code>a > b</code> <code>a >= b</code>	confronta il valore contenuto nelle due variabili
and	<code>a && b</code>	restituisce <code>true</code> se e solo se entrambi i membri sono <code>true</code>
or	<code>a b</code>	restituisce <code>true</code> se almeno uno dei due membri è <code>true</code>
not	<code>!a</code>	restituisce <code>true</code> se <code>a</code> è <code>false</code> e viceversa

Precedenze

In ordine di precedenza:

1. operatori unari: di incremento e decremento, not logico, byteshift, ...
2. prodotto, divisione e modulo
3. somma e sottrazione
4. confronti con uguaglianza
5. operatore logico and
6. operatore logico or
7. assegnazioni

Costanti

Le costanti sono un tipo particolare di variabili che contengono un valore fisso per tutta la durata dell'esecuzione. il valore è assegnato al momento della dichiarazione. In genere le costanti si indicano con lettere maiuscole. `final double PI = 3,14...`

Literals

I Literals sono l'insieme di numeri e stringhe che vengono utilizzate nel programma e che non sono contenuti in una variabile. Ad esempio i valori di inizializzazione delle variabili o i messaggi di testo da mandare in output.

I numeri interi vengono interpretati di tipo `int`, quelli decimali come `double`, i caratteri racchiusi da singoli apici es. `'c'` come `char` e le sequenze di caratteri racchiuse tra doppi apici `"text"` come oggetti della classe `String`.

7.4 Metodi

I metodi sono blocchi di codice racchiusi da parentesi graffe che contengono istruzioni che risolvono un determinato problema. Un programma eseguibile deve necessariamente avere un metodo main definito come `public static void main(String[] args) { }`.

Struttura

```
1 tipoRestituito nomeMetodo (parametriFormali) {  
2     istruzioni;  
3     return valoreRestituito;  
4 }
```

La prima riga del metodo viene chiamata firma.

Un metodo può ricevere in input dei parametri espliciti indicati tra le parentesi tonde e restituire in output un valore di ritorno il cui tipo è specificato prima del nome del metodo e il valore viene indicato alla fine del metodo dopo la parola `return`. Se un metodo non restituisce nessun valore si utilizza `void` al posto del `tipoRestituito` e il `return` viene omissso o indicato senza specificare il `valoreRestituito`.

Overloading

Due metodi possono avere lo stesso nome, ma avere parametri differenti (per tipo e per numero). Quando vengono richiamati, il compilatore riconosce il metodo corretto da utilizzare in base alla corrispondenza del tipo e del numero di parametri tra la chiamata e la firma del metodo.

7.5 Classi e oggetti

Una classe è un contenitore di metodi e variabili. Sono alla base del paradigma di programmazione ad oggetti (OOP) e permettono un maggiore livello di astrazione per risolvere problemi complessi più facilmente.

Ogni file Java deve contenere una classe pubblica con lo stesso nome del file. Questa classe può essere eseguibile se ha un metodo main, altrimenti viene detta non eseguibile.

Una classe può essere istanziabile, ovvero è possibile crearne delle copie (più o meno indipendenti) che prendono il nome di oggetti o istanze della classe.

Struttura

Una classe è composta da:

variabili d'istanza	memorizzano le informazioni e lo stato di un oggetto
metodi costruttori	contengono le istruzioni da eseguire alla creazione dell'oggetto, es. inizializzazione delle variabili
metodi di accesso	servono per accedere alle variabili d'istanza, senza poterle modificare
metodi modificatori	servono per modificare le variabili d'istanza secondo dei precisi criteri

Incapsulamento

Per evitare che l'oggetto venga messo in uno stato non valido (es. lati di una figura negativi), si sceglie di nascondere le variabili d'istanza, riducendone l'accessibilità, e creando dei metodi di accesso e di modifica. Questo principio di "nascondere" le variabili è chiamato incapsulamento e prevede i seguenti vantaggi:

- impedire che l'oggetto venga messo in uno stato inconsistente
- l'utilizzatore non deve preoccuparsi dei dettagli implementativi
- se si modifica l'implementazione di un metodo, non serve modificarne i programmi in cui è impiegato
- se c'è un errore nelle variabili d'istanza, va cercato nei metodi della classe o nelle loro invocazioni

Accessibilità

public	accessibile da qualsiasi classe
package	(default) accessibile da qualsiasi classe contenuta nello stesso pacchetto
protected	accessibile da qualsiasi sottoclasse della classe in cui è creato
private	accessibile solo dalla classe in cui è stato creato

Variabili statiche e non statiche

static	variabile unica comune a tutte le istanze di una classe, se viene modificata da un'istanza, si modifica per tutte le istanze
non-static	variabile propria di una specifica istanza, indipendente dalla corrispettiva in istanze diverse

Metodi statici e non statici

static	non può accedere alle variabili d'istanza e non ha parametri impliciti, si invoca indicando <code>NomeClasse.nomeMetodo(parametri espliciti)</code>
non-static	può accedere alle variabili d'istanza e ha come parametro implicito l'oggetto invocante si invoca indicando <code>nomeOggetto.nomeMetodo(parametri espliciti)</code>

Struttura base di una classe generatrice di oggetti

```
1 public class NomeClasse {
2     // variabili d'istanza
3     private int variabileIstanza1;
4     ...
5
6     // costruttore
7     public NomeClasse() {
8         variabileIstanza1 = 0;
9         ...
10    }
11
12    // metodi di accesso
13    public int getVariabileIstanza1(){
14        return variabileIstanza1;
15    }
16    ...
17
18    // metodi di modifica
19    public void setVariabileIstanza1(int v1) {
20        variabileIstanza1 = v1;
21    }
22    ...
23
24    // metodi statici
25    public static int metodoStatico(int a) {
26        return a;
27    }
28    ...
29 }
```

Instanziamento di un oggetto e richiamo metodi

```
1 // istanziamento di un oggetto della classe NomeClasse
2 NomeClasse nomeOggetto = new NomeClasse(parametri);
3
4 // invocazione del metodo metodo1 dall'oggetto nomeOggetto
5 nomeOggetto.metodo1(parametri);
6
7 // invocazione del metodo metodoStatico della classe NomeClasse
8 NomeClasse.metodoStatico(parametri);
```

Riferimenti ad oggetti

Quando si dichiara un nuovo oggetto, si crea uno spazio in memoria in cui vengono salvati i dati dell'oggetto (es. variabili d'istanza e metodi) e viene creata una variabile riferimento che punta alla posizione in memoria dell'oggetto.

Per cui se creo un oggetto `A a = A();` e uno `A b = a;`, le modifiche che faccio ad `a` si riflettono anche su `b` in quanto entrambi `a` e `b` puntano alla stessa posizione in memoria, per cui allo stesso oggetto.

Quando viene passato un oggetto ad un metodo come parametro esplicito, le modifiche che vengono fatte all'oggetto all'interno del metodo, si riscontrano anche sull'oggetto nel metodo chiamante.

Ereditarietà

Tra diverse classi ci può essere un rapporto di madre-figlia (superclasse-sottoclasse), quando una classe (la sottoclasse) estende un'altra classe (la superclasse) ereditandone tutti i metodi e tutte le variabili d'istanza. Per realizzare una sottoclasse, è necessario aggiungere **extends SuperClasse** nella firma della sottoclasse, dopo il nome della sottoclasse. Una sottoclasse ha un'unica superclasse, non è possibile estendere due classi contemporaneamente.

All'interno dell'implementazione dei metodi della sottoclasse si può:

- chiamare il costruttore della superclasse, tramite il riferimento **super: super(parametri);**
- accedere ai metodi non private della superclasse (se ci sono overloading di metodi è necessario utilizzare il riferimento **super** per chiamare quelli della superclasse)
- accedere alle variabili non private della superclasse (se ci sono overloading di variabili è necessario utilizzare il riferimento **super** per indicare quelle della superclasse)
- aggiungere nuovi metodi statici e non ed in caso eseguire l'overloading di metodi della superclasse
- aggiungere nuove variabili, in caso con lo stesso nome di quelle della superclasse

Attraverso l'istanza della sottoclasse (oggetto della sottoclasse) è possibile chiamare i metodi della superclasse solo se non è stato fatto l'overloading. La superclasse non può accedere alle variabili e ai metodi della sottoclasse.

Tutte le classi, in Java, sono sottoclassi della classe **Object**, da cui ereditano i seguenti metodi (riportati solo quelli ritenuti più importanti):

- **protected Object clone()** crea e restituisce una copia dell'oggetto chiamante
- **boolean equals(Object obj)** confronta i riferimenti dell'oggetto chiamante e di quello passato come parametro
- **int hashCode()** restituisce il codice hash dell'oggetto chiamante
- **String toString()** restituisce la rappresentazione in stringa dell'oggetto chiamante

È buona pratica eseguire l'overloading dei metodi ereditati dalla classe **Object** prima di utilizzarli, in modo da renderli più adatti all'oggetto chiamante (es. modificare **toString** in modo che non restituisca classe e indirizzo di memoria o modificare **equals** in modo che non confronti gli indirizzi, ma le variabili d'istanza).

Casting tra oggetti

È sempre possibile eseguire il casting da una sottoclasse ad una sua superclasse, senza generare errori in compilazione o in esecuzione. Al contrario per eseguire il casting da una superclasse ad una sua sottoclasse è necessario esplicitare il tipo della sottoclasse tra parentesi tonde e, se non è possibile eseguire la conversione, viene lanciata **ClassCastException** durante l'esecuzione.

Polimorfismo

Quando viene eseguito il casting da una sottoclasse ad una superclasse, il “nuovo” riferimento permette di utilizzare solo i metodi definiti per la superclasse, pena un errore in compilazione.

Se non è stato fatto l’overloading del metodo della superclasse, allora in fase di esecuzione verrà chiamato il metodo della superclasse. Se, invece, è stato eseguito l’overloading del metodo in qualche sottoclasse, in fase di esecuzione verrà chiamato il metodo overloade della sottoclasse più specifica. Es. se è stato fatto l’overloading di `toString` e viene chiamato da un oggetto non `Object` dopo il cast ad `Object`, verrà chiamato l’overloading del `toString` della classe originaria.

Questo fenomeno è chiamato polimorfismo e deriva dal fatto che quando viene eseguito il casting da sottoclasse a superclasse, il nuovo riferimento del tipo superclasse punta sempre allo stesso oggetto della sottoclasse e, quando viene invocato un metodo della superclasse di cui è stato eseguito l’overloading, l’interprete (in fase di esecuzione) riconosce il tipo della sottoclasse ed invoca il metodo della sottoclasse.

Questo è utile per creare dei metodi che eseguano operazioni generiche su diversi tipi di oggetti. Ad esempio il metodo `println` riceve come parametro un oggetto (es. un `String`) a cui viene fatto un cast implicito a `Object` e invoca il metodo `toString`. Se non è stato eseguito l’overloading del metodo, viene chiamato il `toString` definito in `Object` (non è questo il caso), altrimenti viene chiamato l’overloading della sottoclasse (in questo caso definito in `String`).

Classi interne

È possibile definire ed implementare una classe (classe interna) all’interno di un’altra classe (classe esterna), per cui la classe interna e quella esterna condividono un rapporto di “fiducia” in quanto possono reciprocamente accedere a tutti i metodi e tutte le variabili, anche se sono stati definiti `private`.

Non è mai possibile creare nuove istanze della classe interna, se non nella classe esterna o nelle classi che la estendono, invece è sempre concesso restituire riferimenti di istanze di classe interna, indipendentemente se la classe interna sia `public` o `private`.

In un metodo al di fuori della classe esterna (es. nel `main`), se la classe interna è `public`:

- è possibile chiamare i metodi `public` della classe interna su riferimenti di classe interna
- è possibile creare riferimenti di classe interna con `ClasseEsterna.ClasseInterna nomeOggetto;`
- i nuovi riferimenti del tipo classe interna possono puntare a `null` o ad istanze della classe interna create nei metodi della classe esterna o sue sottoclassi

Viceversa se la classe interna è `private` non è possibile fare nulla (non si possono creare riferimenti e non è possibile chiamare i metodi, anche se questi sono definiti `public`).

Pacchetti e organizzazione

Più classi che svolgono compiti simili vengono raggruppate in pacchetti. Ad esempio il pacchetto `java.lang` racchiude la classe `System`, la classe `String` e la classe `Math`.

Per utilizzare delle classi che si trovano in file diversi in cartelle diverse, bisogna inserire all’inizio del programma (fuori dalle classi) la parola chiave `import` seguita dalla classe che si vuole importare, precisandone il relativo pacchetto. Il pacchetto `java.lang.*` è importato di default su ogni progetto.

Classe `java.lang.String`

Classe che memorizza una stringa di testo e contiene i metodi per manipolarla. Ogni stringa di testo è un'istanza di `String` e ogni istanza di `String` è immutabile, ovvero non è possibile modificare la stringa memorizzata se non creando una nuova istanza.

Alcuni metodi forniti dalla classe:

<code>String()</code>	costruttore vuoto
<code>char charAt(int)</code>	restituisce il carattere all'indice passato come parametro
<code>int compareTo(String)</code>	restituisce 1 se il parametro esplicito precede quello esplicito, -1 se avviene l'opposto e 0 se sono uguali
<code>boolean equals(Object)</code>	restituisce true se le due stringhe sono uguali, false altrimenti
<code>boolean equalsIgnoreCase(String)</code>	analogo al metodo prima, ignorando maiuscole e minuscole
<code>int indexOf(char)</code>	restituisce l'indice di un dato carattere/stringa nell'intervallo di indici specificato (se specificato)
<code>int length()</code>	restituisce la lunghezza della stringa
<code>String substring(int)</code>	restituisce la sottostringa dal primo indice (incluso) alla fine
<code>String substring(int, int)</code>	restituisce la sottostringa dal primo indice (incluso) al secondo indice (escluso)
<code>String toLowerCase()</code>	converte tutte le maiuscole in minuscole
<code>String toUpperCase()</code>	converte tutte le minuscole in maiuscole

Classe `java.util.Scanner`

Classe che permette di facilitare le operazioni di input, output, lettura da file e suddivisione di stringhe in token.

Alcuni metodi forniti dalla classe:

<code>Scanner(InputStream)</code>	scanner per leggere da standard input: <code>new Scanner(System.in)</code>
<code>Scanner(String)</code>	scanner per leggere da una stringa
<code>Scanner(File)</code>	scanner per leggere da file
<code>boolean hasNext...()</code>	true se ci sono ancora dati da leggere, false altrimenti
<code>String next...()</code>	estrae e restituisce un dato dal buffer
<code>void close()</code>	metodo per chiudere lo scanner, dopo aver finito di utilizzarlo

Classe `java.io.FileReader`

Classe per leggere da file, utilizzabile come oggetto della classe `File` per utilizzare lo scanner.

Alcuni metodi forniti dalla classe:

<code>FileReader(File)</code>	costruttore che riceve un oggetto file da cui leggere
<code>FileReader(String)</code>	costruttore che riceve il nome del file da cui leggere
<code>void close()</code>	metodo per chiudere il file di input, dopo aver finito di utilizzarlo

Siccome può lanciare `IOException`, è necessario inserirlo in un try-with-resources (così viene chiuso in automatico). Questa struttura verrà approfondita più avanti.

Classe java.io.PrintWriter

Classe per scrivere su file utilizzando i metodi normalmente impiegati per lo standard output.

Alcuni metodi forniti dalla classe:

<code>PrintWriter(File)</code>	costruttore che riceve un oggetto file su cui scrivere
<code>PrintWriter(String)</code>	costruttore che riceve il nome del file su cui scrivere
<code>void print(...)</code>	metodo che scrive quanto passato come parametro nel file
<code>void println(...)</code>	come il metodo precedente, alla fine va a capo
<code>void close()</code>	metodo per chiudere il file di input, dopo aver finito di utilizzarlo

Classe java.util.Random

Classe utilizzata per generare numeri pseudocasuali

Alcuni metodi forniti dalla classe:

<code>Random()</code>	istanzia un generatore di numeri casuali, il seme è preso dall'orologio del computer
<code>Random(long)</code>	istanzia un generatore di numeri casuali, il seme è passato come parametro
<code>int nextInt()</code>	restituisce un numero casuale intero nell'intervallo $[0, 2^{32})$
<code>int nextDouble()</code>	restituisce un numero casuale decimale nell'intervallo $[0, 1)$
<code>int nextInt(int)</code>	restituisce un numero casuale nell'intervallo $[0, x)$, con x passato come parametro

7.6 Interfacce

Le interfacce sono dei particolari tipi di “classi” non istanziabili e non implementate, che contengono soltanto metodi pubblici (non serve specificare `public`, dato che è implicito). Servono per indicare i metodi che devono avere determinate classi che implementano l'interfaccia.

Una classe, per implementare l'interfaccia, deve contenere `implements NomeInterfaccia` che segue il nome della classe nell'implementazione della classe stessa. Una classe può implementare diverse interfacce.

7.7 Commenti e JavaDoc

I commenti sono parti di testo che il compilatore ignora. Si usano per descrivere quali operazioni vengono effettuate in una determinata parte del codice. Possono essere su una singola riga `// commento` o su più righe `/* commento */`.

Esiste uno standard di commentazione che permette di creare una documentazione nello stesso stile di quella di java, con il comando `javadoc NomeFile.java`. Si prevede un commento su più righe posto prima di un metodo o una classe e in tale commento vengono descritti la funzione della classe, i parametri (preceduti da `@param`), il valore di ritorno (preceduto da `@return`) ed eventuali eccezioni che possono essere lanciate.

```
1 /**
2  * Descrizione del metodo
3  *
4  * @param p1 parametro 1
5  * @param p2 parametro 2
6  * @return valore che viene restituito
7  * @throws Exception eccezione lanciata
8  */
9 public int metodo(int p1, double p2) {
10     ...
11 }
```


7.8 Operazioni logiche

Operazioni logiche

...

Confronti tra double / float

...

Confronti tra oggetti

...

De Morgan

...

Cortocircuito logico

...

7.9 Selezioni

utilizzo, struttura, else if, annidamento, else sospeso, espressioni logiche, ... switch case

7.10 Iterazioni

while, do-while, for, cicli annidati, break, continue

7.11 Note sulla memoria in Java

Heap e Stack

...

Record di attivazione

...

7.12 Reindirizzamento dei flussi i/o

Reindirizzamento del flusso di input

...

Reindirizzamento del flusso di output

...

Canalizzazioni o pipes

...

7.13 Eccezioni in Java

generale...

...

Throw

...

Try - catch

...

Try with resources

...

Eccezioni checked e unchecked

...

8 Ricorsione

...

9 Complessità computazionale

...

9.1 Complessità temporale

...

9.2 Complessità spaziale

...

10 Strutture dati

liste e array

10.1 Array

definizione, struttura, creazione, passaggio come parametro

Algoritmo di copia

copia di un array

Algoritmo di ridimensionamento

ridimensionamento di un array

Array riempiti a metà

dimensione logica vs dimensione relative

10.2 Matrici

array bidimensionali

10.3 Linked List o lista concatenata

Struttura dati che implementa il concetto di sequenza. È composta da nodi che contengono:

- il riferimento al dato associato
- il riferimento al nodo successivo

Il nodo di testa è chiamato header e ha `null` come riferimento al nodo precedente.

Il nodo di coda è chiamato tail e ha `null` come riferimento al nodo successivo.

Quando nodo di testa e nodo di coda coincidono, la lista è vuota.

Implementazione nodo

```
1 public class ListNode {
2     private Object element;
3     private ListNode next;
4
5     public ListNode(Object elm, ListNode next);
6     public ListNode();
7
8     public Object getElement();
9     public ListNode getNext();
10
11     public void setElement(Object elm);
12     public void setNext(ListNode next);
13 }
```

Implementazione lista

```
1 public class LinkedList {
2     private ListNode header;
3     private ListNode tail;
4
5     public LinkedList();
6
7     public boolean isEmpty();
8     public void makeEmpty();
9
10    public Object getFirst();
11    public Object getLast();
12 }
```

```

12
13     public void addFirst(Object elm);
14     public Object removeFirst();
15
16     public void addLast(Object elm);
17     public Object removeLast();
18
19     public ListIterator getIterator();
20 }

```

Implementazione iteratore lista

```

1 private class LinkedListIterator {
2     private ListNode current;
3     private ListNode previous;
4
5     public Object next();
6     public boolean hasNext();
7     public void add(Object obj);
8     public void remove();
9 }

```

La classe ListIterator è interna alla classe LinkedList ed è privata.
Si immagina l'iteratore come il cursore in un editor di testo.

Complessità

- getFirst $O(1)$
- getLast $O(1)$
- addFirst $O(1)$
- removeFirst $O(1)$
- addLast $O(1)$
- removeLast $O(n)$

Lista vs Array

...

10.4 ArrayList (no esame)

Classe in java.util, implementato con array riempito in parte e ridimensionabile.

Inizializzazione

```
ArrayList<OggettiDaSalvare> nomeArray = new ArrayList<OggettiDaSalvare>();
```

Metodi

`add(newObj)` aggiunta alla fine, inserimento costante secondo l'analisi ammortizzata

`remove(index)` rimozione elemento ad un dato indice

`get(index)` accesso ad un elemento dato l'indice

`set(index, newObj)` modifica di un elemento già presente

`getSize()` dimensione logica dell'array

11 Strutture dati astratte - ADT

11.1 Container

Struttura dati generica.

```
1 /**
2  * Interfaccia che definisce i metodi di una struttura dati generica
3  */
4 public interface Container {
5     /**
6      * Metodo che verifica se la struttura dati e' vuota o no
7      * @return true se la struttura dati e' vuota, false altrimenti
8      */
9     public boolean isEmpty();
10
11     /**
12      * Metodo che svuota la struttura dati
13      */
14     public void makeEmpty();
15 }
```

11.2 Stack

Struttura dati Last-In-First-Out (LIFO).

```
1 /**
2  * Interfaccia che definisce i metodi di una pila
3  */
4 public interface Stack extends Container {
5     /**
6      * Metodo che inserisce un nuovo elemento nella pila
7      * @param obj oggetto da inserire alla pila
8      * @throws FullStackException se la pila e' piena
9      */
10     public void push(Object obj);
11
12     /**
13      * Metodo che restituisce il prossimo elemento da estrarre
14      * @return prossimo elemento da estrarre
15      * @throws EmptyStackException se la pila e' vuota
16      */
17     public Object top();
18
19     /**
20      * Metodo che rimuove un elemento dalla testa della pila
21      * @return elemento rimosso
22      * @throws EmptyStackException se la pila e' vuota
23      */
24     public Object pop();
25 }
```

Complessità computazionale:

- inserimento $O(1)$
- accesso $O(1)$
- rimozione $O(1)$

11.3 Queue

Struttura dati First-In-First-Out (FIFO).

```
1 /**
2  * Interfaccia che definisce i metodi di una coda
3  */
4 public interface Queue extends Container {
5     /**
6      * Metodo che inserisce un nuovo elemento nella coda
7      * la condizione in cui la coda e' vuota e' front == back, siccome quella in
8      * cui la coda e' piena sarebbe la stessa, per evitare problemi si sceglie
9      * di scaprecare un elemento e considerare la coda piena quando si ha un
10     * solo elemento libero, ovvero se back+1 = front
11     * @param obj oggetto da inserire alla coda
12     * @throws FullQueueException se la coda e' piena
13     */
14     public void enqueue(Object obj);
15
16     /**
17      * Metodo che restituisce il prossimo elemento da estrarre
18      * @return prossimo elemento da estrarre
19      * @throws EmptyQueueException se la coda e' vuota
20     */
21     public Object getFront();
22
23     /**
24      * Metodo che rimuove un elemento dalla testa della coda
25      * @return elemento rimosso
26      * @throws EmptyQueueException se la coda e' vuota
27     */
28     public Object dequeue();
29 }
```

Complessità computazionale:

- inserimento $O(1)$
- accesso $O(1)$
- rimozione $O(1)$

11.4 Map

Definizione

Struttura dati associativa che memorizza coppie di dati composte da chiave e valore. L'associazione chiave-valore è biunivoca (1 chiave - 1 valore)

Metodi - interfaccia

Complessità computazionale

11.5 Multimap - Dictionary

Definizione

Struttura dati associativa che memorizza coppie di dati composte da chiave e valore. Possono essere presenti più valori associati alla stessa chiave.

Metodi - interfaccia

Complessità computazionale

11.6 Set

Struttura

Definizione

Metodi - interfaccia

Complessità computazionale

11.7 Table

Definizione

Metodi - interfaccia

Complessità computazionale

11.8 Hash Table

Definizione

Metodi - interfaccia

Complessità computazionale

11.9 Albero - no esame

Struttura dati in cui un elemento può avere più successori, se ogni elemento ha due successori, si dice albero binario.

11.10 Grafi - no esame

Struttura dati in cui un elemento può avere più successori e più predecessori, creando delle reti di nodi. Si può parlare di grafi diretti/indiretti, semplici/multigrafi, connessi/non connessi, pesati/non pesati, ...

12 Algoritmi di ordinamento

...

12.1 SelectionSort

...

12.2 MergeSort

...

12.3 InsertionSort

...

12.4 Confronto complessità

...

13 Algoritmi di ricerca

...

13.1 Ricerca lineare

...

13.2 Ricerca binaria

...