Ministero dell'Istruzione, dell'Università e della Ricerca
Servizio Automazione Informatica e Innovazione Tecnologica

Modulo 3

Computer software

ForTIC

Piano Nazionale di Formazione degli Insegnanti sulle Tecnologie dell'Informazione e della Comunicazione

Percorso Formativo C

Materiali didattici a supporto delle attività formative

2002-2004

Promosso da:

- Ministero dell'Istruzione, dell'Università e della Ricerca, Servizio Automazione Informatica e Innovazione Tecnologica
- Ministero dell'Istruzione, dell'Università e della Ricerca, Ufficio Scolastico Regionale della Basilicata

Materiale a cura di:

- Università degli Studi di Bologna, Dipartimento di Scienze dell'Informazione
- Università degli Studi di Bologna, Dipartimento di Elettronica Informatica e Sistemistica

Editing:

CRIAD - Centro di Ricerche e studi per l'Informatica Applicata alla Didattica

Progetto grafico:

Campagna Pubblicitaria - Comunicazione creativa

Scopo e obiettivi del modulo

In questa sezione verrà data una breve descrizione del modulo.
Gli scopi del modulo consistono nel mettere in grado di:

- Descrivere, identificare, usare software di sistema di diversi produttori su differenti piattaforme
- Descrivere e usare le più diffuse categorie di software applicativo
- Installare, disinstallare, configurare e aggiornare software di sistema e applicativi
- Identificare i più usati linguaggi di programmazione
- Saper riconoscere e usare costrutti procedurali e *object-oriented*
- Saper indicare e descrivere le tecnologie *software* emergenti

Il modulo è strutturato nei seguenti argomenti:

- ***Software* di sistema**
  - Descrivere le funzioni e le principali componenti di un sistema operativo.
  - Identificare i sistemi operativi attuali e descrivere le loro caratteristiche.
  - Usare un sistema operativo per attività quali la gestione di file e dati.
  - Identificare le *utility* di sistema e descrivere le loro principali funzioni.
  - Usare il *software* di sistema per effettuare procedure come *back-up*, deframmentazione dei dischi, ecc.
  - Usare sistemi operativi di differenti produttori su differenti piattaforme.
  - Usare sistemi operativi *stand-alone* e di rete.
  - Creare, usare e mantenere file di configurazione di sistema.
  - Modificare la configurazione di un sistema operativo per ottimizzare le prestazioni.
  - Trasmettere e scambiare file in un ambiente con piattaforme multiple.
- ***Software* applicativo**
  - Descrivere le principali caratteristiche e funzione delle più diffuse categorie di *software* applicativo (*word processing*, *spreadsheet*, *database*, *presentation*, *e -mail*, *browsers*, ecc.)
  - Usare *software* di *office automation*
  - Imparare autonomamente ad effettuare attività usando *software* di *office automation*.
  - Usare software realizzato da produttori diversi.
  - Trasmettere e scambiare dati in un ambiente con piattaforme multiple.
  - Utilizzare caratteristiche di integrazione di differenti *software* di *office automation*.
  - Usare strumenti di produttività di ufficio o altri *software* applicativi ad un livello avanzato.
- **Installazione e configurazione del *software***
  - Installare e configurare un sistema operativo per PC.
  - Descrivere le procedure per la disinstallazione di un sistema operativo.
  - Installare e configurare *software di sistema*.
  - Installare e configurare *software applicativi*.
  - Configurare *software* per garantire l'accessibilità dei disabili.
  - Installare e configurare aggiornamenti di *software applicativo*.
  - Descrivere le modifiche necessarie ad un sistema operativo (modifica dei parametri, gestione dei conflitti di *interrupt*, ecc.) nell'installazione, configurazione e aggiornamento di *software applicativo*.
  - Installare e configurare *software client* per la connessione a LAN, WAN, Internet (*network client*, *WWW browser*, emulatori di terminali, *file transfer*, ecc.).
  - Installare e configurare *software* per applicazione *client/server* e applicazioni in rete (*e-mail*, *database*, ecc.).
  - Installare applicazioni su un *server* e configurare *client* per accessi di rete.
- **Elementi di linguaggi di programmazione**
  - Identificare alcuni dei più importanti e attualmente usati linguaggi di programmazione.

    - Saper distinguere tra programmazione strutturata e programmazione *object-oriented*
    - Usare costrutti procedurali e *object-oriented* di linguaggi di programmazione e di *scripting* per creare e testare semplici programmi e *file batch*.

- **Tecnologie emergenti e tendenze**
    - Indicare alcune tecnologie *software* emergenti.
    - Descrivere il possibile impatto delle tecnologie emergenti indicate.

Introduzione

Introduzione generale

Prof. Maurizio Gabbrielli
3.1 Software di sistema

Cos'è un sistema operativo

Il **sistema operativo** (*operating system*) è un insieme di programmi che interagiscono e controllano le funzionalità della macchina fisica (*hardware*) in modo tale da offrire ai programmi applicativi ed agli utenti un insieme di funzionalità con un elevato livello di astrazione. I programmi applicativi accedono quindi alle risorse *hardware* mediante opportuni componenti del sistema operativo. Inoltre il sistema operativo offre un'interfaccia amichevole dalla quale l'utente può invocare i principali servizi.

Il **sistema operativo** aggiunge un livello (*software*) alla macchina fisica realizzando così una nuova macchina virtuale che facilita l'uso delle risorse fisiche, consente di superare alcuni problemi dovuti alla loro limitazione (ad esempio dimensioni della memoria centrale) e ne ottimizza e regolamenta l'accesso (ad esempio evitando conflitti).

Il concetto di macchina virtuale

Una macchina virtuale è un dispositivo di calcolo astratto caratterizzato da:

- un linguaggio che la macchina è in grado di capire, specificato da una opportuna sintassi, ovvero da un insieme di regole che definiscono i costrutti corretti del linguaggio;
- un interprete per tale linguaggio, ovvero un meccanismo che permetta di eseguire le istruzioni specificate nel linguaggio definendone così anche il significato, ovvero la semantica.

Il concetto di macchina virtuale è ampiamente usato in molti settori dell'informatica sia per descrivere sistemi che per specificare le caratteristiche dei linguaggi di programmazione.

Ad esempio possono essere considerate macchine virtuali:

- una macchinia fisica, ovvero un calcolatore *hardware*, definito da un opportuno linguaggio di programmazione (linguaggio macchina) e dall' interprete realizzato dal ciclo *fetch-execute* della CPU;
- un editor di testi, definito dai comandi che agiscono sull'astrazione di un documento elettronico.

Le macchine astratte possono essere combinate in vario modo per costruire macchine più complesse.

Architettura di un sistema operativo

I componenti di un sistema operativo si possono considerare organizzati a livelli secondo la struttura illustrata qui sotto. Tale struttura definisce una gerarchia di macchine virtuali: i componenti di uno stesso livello realizzano una macchina virtuale che, usando i servizi del livello sottostante, definisce nuove funzionalità.

Componenti di un sistema Operativo 1

Interprete dei Comandi (o *shell*)
: Fornisce l'interfaccia con l'utente che attraverso di esso può inviare comandi al sistema operativo (ad esempio lettura dalla memoria di massa, creazione ed attivazione di un processo, allocazione di memoria centrale, caricamento di un programma in memoria centrale ecc.). L'interfaccia dell'interprete con l'utente può essere di tipo testuale oppure grafico.

Gestore dei *file* (*file system*)
: Componente che permette l'organizzazione delle informazioni in strutture logiche, dette ***file*** , identificati da nomi e accessibili all'utente tramite opportuni comandi (apertura, lettura, scrittura, chiusura). Gestisce le operazioni di allocazione di memoria di massa necessarie per memorizzare i *file*

Gestore delle periferiche
: Componente che fornisce una visione astratta delle periferiche e dei relativi comandi di input/output, mascherando le specifiche caratteristiche (fisiche e non) di ogni dispositivo. Il particolare dispositivo *software* che controlla la comunicazione con la periferica è detto anche **Driver** della periferica.

Componenti di un sistema Operativo 2

Gestore della Memoria
: Modulo che gestisce la memoria centrale consentendo ai programmi di lavorare in un proprio spazio di indirizzamento virtuale. Permette di
  - proteggere i dati ed i programmi; in generale un programma non può accedere alla zona di memoria riservata ad un altro
  - mascherare l'allocazione fisica dei dati; il programma non vede i dettagli fisici relativi alla collocazione

dei dati
- condividere dati in modo controllato fra più programmi

Nucleo
: Componente che comunica con la macchina fisica per gestire lo stato di avanzamento dei processi corrispondenti ai vari programmi che sono contemporaneamente attivi. Il nucleo in sostanza realizza più macchine virtuali corrispondenti all'unica CPU fisica

BIOS (*Basic Input Output System*)
: A differenza degli altri componenti, questo modulo, che può essere considerato separato dal Sistema Operativo vero e proprio, non risiede nella memoria di massa ma è memorizzato permanentemente su una ROM (memoria di sola lettura). Al momento dell'accensione l'elaboratore come prima cosa cerca il BIOS che fornisce una sorta di collegamento fra l'*hardware* ed il *software*. Il BIOS gestisce varie funzionalità (gestisce le comunicazioni attraverso le porte del *computer*, interpreta i dati immessi da tastiera, visualizza i caratteri sullo schermo) in modo tale da rendere il sistema operativo indipendente dall'*hardware*.

Alcuni Sistemi Operativi esistenti

L'interfaccia verso l'utente dell'interprete dei comandi può essere molto diversa:

- nel caso più semplice (e vecchio) l'interprete dei comandi del sistema operativo si aspetta un comando testuale dal terminale (esempio DOS)
- nel caso più moderno l'interfaccia è di tipo testuale (*Windows* di *Microsoft*, *MacOS* di *Apple*, *Linux*): l'interprete dei comandi propone sullo schermo un menu di comandi selezionabili mediante il *mouse* , che permette anche di manipolare icone disposte su una scrivania virtuale (*desktop*).

• nel caso più moderno l'interfaccia è di tipo testuale (*Windows* di *Microsoft*, *MacOS* di *Apple*, *Linux*): l'interprete dei comandi propone sullo schermo un menu di comandi selezionabili mediante il mouse , che permette anche di manipolare icone disposte su una scrivania virtuale (desktop).

I sistemi operativi moderni (*Windows*, *MacOS*, *Linux*) permettono il *multitasking*, ovvero permettono l'uso della CPU ad un programma alla volta per brevi intervalli di tempo, così che l'utente vede più programmi eseguiti contemporaneamente. Quando questo non è possibile si parla di sistemi *monotasking*.

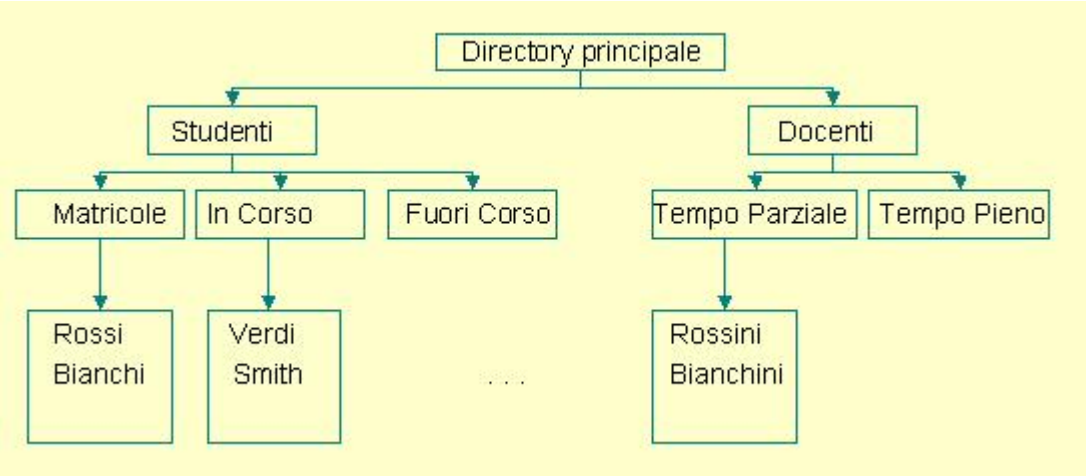
Alcuni sistemi operativi

- *Unix*: sviluppato a partire dal 1973 presso AT&T. Adatto principalmente per *server* di medie dimensioni.
- DOS: introdotto nel 1980 sul primo PC IBM, interfaccia testuale, *monotasking*.
- *MacOS*: sviluppato da *Apple* per i propri *computer* e per PC. *Apple* è stato il primo produttore ad offrire interfacce grafiche (a finestre).
- *Windows* (95, 98, NT, 2000, ME, CE) *Microsoft*: tipico sistema operativo per PC, interfaccia grafica.
- *Linux*: basato su *Unix*, sistema operativo per varie piattaforme basato sulla filosofia *freeware*, ossia è distribuito gratuitamente (si paga poi la manutenzione ed i servizi).

Gestione dei file e dei dati 1

I sistemi operativi moderni (*Windows*, *MacOS*, *Linux*) offrono in primo luogo servizi di gestione di documenti di varia natura (testi, grafica, audio, video, programmi). I documenti sono memorizzati in contenitori logici detti *file*, identificati da un nome.

L'insieme dei *file*, detto anche *file system*, è tipicamente organizzato in una struttura ad albero, i cui nodi sono detti *directory* (o cartelle) e le cui foglie contengono i singoli *file*.



Il contenuto di un *file* è di solito memorizzato su un singolo dispositivo di memoria secondaria (disco, CDROM, nastro). L'insieme degli archivi può invece essere distribuito su più dispositivi. Un *file* ha una struttura logica contigua, ma può essere memorizzato su più blocchi di memoria, non necessariamente contigui.

Gestione dei file e dei dati 2

Le varie cartelle sono rappresentate con icone, ad esempio del tipo

La struttura ad albero è rappresentata in vario modo, ad esempio facendo vedere le cartelle interne come raggiungibili dalla cartella esterna cliccando su un simbolo associato alla medesima. Inoltre cliccando su di una cartella di solito si accede al suo contenuto

Ad ogni *file* sono associati alcuni attributi. L'attributo più importante di un *file* è il **nome** che permette l'accesso al *file*. Il nome completo è specificato di solito fornendo un cammino che individua il *file* nella struttura ad albero del *file system*. Ad esempio

C:\didattica\informatica\corso.doc

indica il *file* lucidi nella cartella informatica che a sua volta si trova nella cartella didattica che si trova sull'*hard disk*. Normalmente comunque non serve specificare il nome completo in quanto si possono cercare i *file* aprendo le cartelle fino ad arrivare al *file* desiderato. Per poter accedere al contenuto di un *file* si deve effettuare un'operazione di apertura del *file*, mentre quando si è terminato l'utilizzo di un *file* esso deve essere chiuso. I *file* possono essere spostati da una cartella ad un'altra trascinando le relative icone per mezzo del mouse.

Un altro attributo importante è il **tipo del *file***, che descrive la struttura del contenuto. La maggior parte dei sistemi operativi supporta molti tipi distinti di *file*. In DOS e *Windows* il tipo di *file* è indicato da un suffisso che si appende al nome, detto **estensione**. Ad esempio

.txt
: denota un *file* in testo ASCII

.doc
: denota un *file* scritto con MS *Word*

.htm

denota un *file* HTML

Codice ASCII

Un *file* è rappresentato elettronicamente (su disco, in RAM) come sequenza di caratteri ASCII (ASCII: *American Standard Code for Information Intechange*). L'ASCII standard è su 7 bit, quindi ha 128 valori possibili. Ad esempio

Hex	Dec	Val
61	97	a
62	98	b
63	99	c
64	100	d
65	101	e
66	102	f
67	103	g

Hex
: Valore esadecimale

Dec
: Valore decimale

Val
: Simbolo rappresentato

Utilità di sistema

Oltre al componente per la gestione di *file*, i sistemi operativi comprendono programmi di utilità (o utility di sistema) che realizzano varie funzionalità utili alla gestione dell'elaboratore. Fra le principali ricordiamo le seguenti:

Formattazione.
: Un programma di utilità è adibito alla formattazione dei dischi, che consiste nella suddivisione del disco in tracce circolari e settori a spicchio in modo tale da creare una degli indirizzi che permettano di archiviare e poi recuperare i *file*. Tale programma crea anche la tabella di allocazione dei *file* (**FAT**) che permette la localizzazione dei *file* fornendo la corrispondenza nome-indirizzo sul disco.

Scanning.
: Questo programma di utilità effettua una scansione del disco per individuare eventuali porzioni di memoria rovinate e marcarle in modo tale che non siano poi usate dal sistema operativo. Inoltre permette entro certi limiti di localizzare eventuali dati persi a causa di guasti del sistema.

Eliminazione *file*.
: Nei sistemi operativi con interfaccia grafica moderni si possono eliminare i *file* che non servono più spostandoli in un cestino (lo spostamento avviene trascinando le relative icone per mezzo del mouse). Da qui i *file* possono essere recuperati fino a che non si effettua una operazione di svuotamento del cestino (dopo tale operazione il recupero è possibile solo con programmi speciali e se lo spazio disco non è stato sovrascritto).

Utilità di sistema: Deframmentazione e backup

Deframmentazione (*Defrag*).
: Quando un disco è vuoto i *file* vengono memorizzati in porzioni di disco (dette cluster) adiacenti. Tuttavia, successive cancellazione e aggiunte di *file* possono rendere sempre più piccoli gli spazi liberi adiacenti, cosi che un singolo *file* può venire memorizzato su cluster non più adiacenti ma sparsi sul disco. Questo crea un

rallentamento delle funzioni del calcolatore perchè i tempi necessari per le operazioni di lettura e scrittura vengono ad essere aumentati. Si può ovviare a questo problema usando una utility di deframmentazione, che sposta i dati sul disco in modo tale da memorizzare i dati di uno stesso *file* su cluster contigui e da ricompattare lo spazo vuoto.

Backup .
Dato che i dischi, come ogni dispositivo fisico, sono soggetti ad errori e malfunzionamenti, per evitare la perdita irrimediabile di dati è utile eseguire periodicamente il salvataggio dei dati di interesse su dispositivi di memoria ausiliari (*hard disk*, *floppy*, DVD o nastri). Tale salvataggio, detto backup, può essere fatto utilizzando un opportuno programma di sistema che di solito provvede anche alla compressione dei dati per ottimizzare l'uso dello spazio di memoria.

L'uso dei programmi di utilità di sistema di solito è semplice. Ad esempio, in *Windows* 98, per accedere alle utilità di sistema basta premere sul bottone *START*, selezionare Programmi (*Programs*) nel menù a tendina che appare, selezionare quindi Accessori (*Accessories*) e quindi Strumenti di Sistema (*System Tools*). Da qui si accede (cliccando sul corrispondente nome) ai vari programmi di utilità il cui uso è autoesplicativo. Nel caso in cui tali programmi non siano raggiungibili dal menu è semplice localizzarli usando il comando *Find* (dal menù a tendina che si ottiene cliccando sul bottone *START*)

Sistemi operativi Stand-alone e di rete

I sistemi operativi di possono distinguere anche in base alla possibilità di supportare applicazioni di rete. Si parla di sistema operativo stand-alone quando il sistema operativo gestisce un unico calcolatore mentre si parla si sistema operativo di rete quando deve gestire dei calcolatori collegati in rete. Uno dei modelli di condivisione delle risorse più comuni prevede un'artchitettura di tipo *client-server*:

client:
è il calcolatore (o, in generale, il processo) che richiede dei servizi; normalmente i *client* corrispondono ai processi attivati dagli utenti finali;

server:
è un calcolatore che ha il compito di soddisfare le richieste dei *client* fornendo i servizi richiesti e gestendo quindi la condivisione delle risorse.

Ad esempio, nel contesto del *World Wide Web*, i *browser* usati sui *computer* degli utenti che accedono al WWW hanno il ruolo di *client* mentre le macchine sulle quali sono presenti le pagine web sono i *server*.

I sistemi operativi usati sui *server* devono essere in grado di gestire grandi quantità di dati e comunicazioni, devono garantire la sicurezza e la consistenza dei dati memorizzati centralmente sul *server*, devono gestire in modo efficiente la condivisione di risorse. Fra i sistemi operativi più diffusi per l'uso in rete ricordiamo:

- *Windows* NT
- OS 2
- *Unix*
- *Linux*

Condivisione e trasferimento di file

Un'applicazione molto comune in una rete di calcolatori è il trasferimento di *file* fra *computer* diversi.

- Si dice caricare un *file* (o farne l'*upload*) il trasferirlo ad un altro calcolatore
- Si dice scaricare un *file* (o farne il *download*) il trasferirlo da un altro calcolatore al nostro.

Se si sta operando all'interno di una rete locale (LAN) di calcolatori il trasferimento di un *file* avviene essenzialmente come il trasferimento fra due cartelle diverse all'interno della stessa macchina: basta spostare il *file* in una cartella pubblica, ovvero una cartella alla quale abbiano accesso anche altri utenti: normalmente al momento della creazione di una cartella è possibile stabilirne i diritti di accesso e quindi se la cartella è condivisa (*shared*) o no. In *Windows*,

ad esempio, si possono modificare i diritti d'accesso cliccando con il bottone destro del mouse sulla cartella e quindi selezionando proprieta (*properties*) e poi condivisione (*sharing*).

Se invece si vogliono trasferire *file* fra *computer* geograficamente distanti ma connessi ad Internet (o comunque connessi da un rete geografica) si può usare il protocollo FTP. Questo è un protocollo del livello applicazione che usa TCP come protocollo di livello trasporto. Se dunque è disponibile FTP (e quindi TCP) si possono scambiare *file* mediante una sorta di *login* remoto e quindi usando i comandi di FTP.

Una terza possibilità infine è quella di inviare un *file* come *attachment* di un messaggio di posta elettronica. In questo caso si usano opportuni programmi di codifica per evitare che nel trasferimento del messaggio attraverso i vari *computer* (*gateway*) sulla rete si generino errori. La codifica è solitamente fatta automaticamente dal programma che si usa per la posta elettronica. Ad esempio, l'estensione MIME del protocollo SMTP utilizza un sistema di codifica detto Base 64 e trasforma un *file* binario in un *file* ASCII.

FTP

Ad esempio, per trasferire il *file* pluto dalla macchina pippo.cs.psu.edu alla propria macchina si può fare come segue:

- si digita
  `ftp pippo.cs.psu.edu`
- nella finiestra in cui è disponibile una *shell* (in *Unix* o *Linux*) o nella zona opportuna della finestra ottenuta cliccando sul bottone *START* e quindi selezionando Esegui (*Run*) (in *Windows*). Questo fa si che venga eseguito il programma FTP con accesso alla macchina pippo.cs.psu.edu.
- si effettua il *login* fornendo nome utente e *password*. Se non si dispone di un *account* sulla macchina pippo si può usare la parola *anonymous* come nome utente e quindi il proprio indirizzo di *e-mail* come *password*.
- si digita
  `get pluto` per trasferire il *file*
- si digita
  `quit` per terminare il collegamento quando la trasmissione è finita.

Se il *file* richiesto non fosse al *top-level* nella macchina alla quale ci si è collegati, si possono usare opportuni comandi di FTP, simili a quelli di *Unix*, per muoversi nel *file system* e localizzarlo. È possibile anche ottenere aiuto da FTP digitando help oppure il punto interrogativo.

Introduzione al software applicativo

Prof. Maurizio Gabbrielli
3.2 Software applicativo

Il software applicativo

Il ***software* applicativo** è l'insieme dei programmi che, usando le funzionalità del sistema operativo, permettono di realizzare specifici compiti quali, ad esempio: videoscrittura, navigazione in Internet, elaborazione di dati mediante fogli elettronici, gestione di basi di dati, elaborazioni multimediali, progettazione assistita dal computer (CAD), commercio elettronico e molti altri ancora.

Da un punto di vista architetturale il *software* applicativo è un livello ulteriore costruito sul sistema operativo:

Il *software* applicativo realizza quindi una nuova macchina virtuale, costruita sul sistema operativo, che definisce delle funzioni e dei servizi specifici per classi di applicazioni. Il *software* applicativo è progettato e realizzato sia da grossi produttori, per i prodotti di largo impiego, che da piccole aziende per le applicazioni specifiche.

Esempi di software applicativo 1

Si possono a grandi linee distinguere le seguenti categorie di *software* applicativo:

Office automation (automazione d'ufficio).
Ci si riferisce con questo termine all'uso coordinato di strumenti *software* per la comunicazione e l'elaborazione di dati, nel contesto di piccole reti di PC quali quelle che si hanno in ufficio. Rientrano in questa classi categoria

- programmi, di video scrittura ed elaborazione testi (*word processing*) (vedi **approfondimento**);
- le agende elettroniche ed i fogli elettronici (o *spreadsheet*). Questi sono strumenti che permettono il trattamento e l'analisi di dati numerici mediante tabelle (o griglie) nelle quali è possibile operare sulle singole caselle, sulle colone, sulle righe o su più caselle collegate fra loro (vedi **approfondimento**);
- programmi di gestione delle comunicazioni quali quelli per l'utilizzo della posta elettronica, per la realizzazione di video-conferenze ecc.

Sistemi di gestione di basi di dati (DBMS).
Questi sono programmi per la realizzazione di sistemi informativi automatizzati aziendali. Con sistema informativo si intende l'insieme dei documenti strutturati e dei processi di elaborazione che li trattano. Per poter gestire in modo automatico sistemi informativi che comportano grandi quantità di dati, quali quelle che si hanno in azienda, sono nate le cosiddette basi di dati ed i relativi programmi di gestione detti appunto DBMS (*Data Base Management System*). Su questo argomento è disponibile un **approfondimento** .

Strumenti per il commercio elettronico (*e-commerce*).
Molte aziende oramai, oltre a pubblicizzare i propri prodotti su Internet, usano la rete anche per permettere acquisti e pagamenti. Il maggior problema che si riscontra in questo settore è la possibilità di permettere transazioni finanziarie (ad esempio, usando la propria carta di credito) in modo sicuro. Per questo esistono opportuni *software* che usano vari meccanismi di crittografia per impedire l'accesso di estranei a dati sensibili.

Esempi di software applicativo 2

Sistemi di supporto alle decisioni (DSS).
Un DSS è un sistema che aiuta a prendere decisioni in situazioni complesse mediante l'impiego di opportuni modelli matematici, logici e statistici specifici del particolare dominio di applicazione in esame. Anche se sono state sviluppate molte applicazioni, dalla medicina all'analisi finanziaria, questi sistemi sono per il momento di limitata diffusione.

Data Mining.
Il *Data mining* consiste nell'estrazione di informazioni contenute in modo implicito e semi-strutturato in grandi

quantità di dati. A differenza dei DBMS e dei DSS, il *data mining* non si basa su interrogazioni specifiche ma sull'estrapolazione di relazioni significative. Ad esempio, avendo a disposizione i dati sugli acquirenti di libri, un editore puo' essere interessato a conoscere la tipologia dell'acquirente medio di un testo di informatica. Il *Data mining* è una disciplina che ha trovato un notevole impulso grazie al Web, dove i protagonisti del commercio on-line usano varie tecniche per individuare informazioni utili ai fini commerciali e, in generale, per effettuare il cosidetto *user-profiling* (ad esempio, vengono identificati i gusti e le caratteristiche di un acquirente utilizzando opportuni *cookies* che il server web del sito nel quale facciamo un acquisto memorizza nel nostro computer).

GIS (Sistemi Informativi Geografici).
: I GIS sono sistemi che gestiscono e illustrano dati su carte geografiche e che sono utilizzati per l'immediatezza della lor rappresentazione grafica.

Programmi per l'uso di Internet e del *World Wide Web*.
: In questa classe rientrano i programmi per usare la posta elettronica, i ***browser*** per la navigazione in Internet e la visualizzazione delle pagine HTML, i programmi per il trasferimento di dati fra terminli remoti ecc. Questi programmi e gli aspetti relativi verrano trattati in uno specifico modulo (12) del corso per cui qui sono omessi.

Programmi di supporto alla progettazione e produzione

Questa categoria comprende tutti quei programmi volti a migliorare l'efficienza e la gestione aziendale dal livello della produzione a quello del *marketing* e della distribuzione. Rientrano in questa classe:

CAD (*Computer Aided Design*):
: strumenti di supporto alla progettazione del prodotto che tipicamente permettono la realizzazione e la modifica veloce di disegni tecnici (anche tridimensionali). Sono usati prevalentemente in ambito meccanico, ma esistono anche applicazioni in altri settori (ad esempio, ci sono strumenti specifici per la progettazione di circuiti elettronici)

CAPP (*Computer Aided Process Planning*):
: strumenti per pianificare ed ottimizzare i processi produttivi.

CAM (*Computer Aided Manufacturing*):
: strumenti per il controllo della produzione automatica, a partire dalle macchine utensili a controllo numerico fino ai sistemi di trasporto e di immagazzinamento con la relativa logistica.

CAQ (*Computer Aided Quality Assurance*):
: strumenti per l'automazione del controllo di qualità a tutti i livelli del processo produttivo.

Tipi di software

Mentre fino a non molto tempo fà l'unica possibilità era quella di acquistare il *software*, oggi ci sono altre alternative. Infatti si possono distinguere i seguenti tipi di *software*:

***Software* proprietario** .
: È *software* coperto da *copyright* e deve essere acquistato. Se viene usato senza essere acquistato (ad esempio usando copie illegali) si può essere perseguiti a norma di legge.

***Software* di pubblico dominio** .
: Non coperto da diritti. Si può copiare liberamente senza temere alcuna conseguenza.

Software shareware.
: Coperto da *copyright*. Disponibile gratuitamente ma per continuare a usarlo occorre acquistarlo.

Software freeware .
: Coperto da *copyright*. Disponibile gratuitamente (le aziende che lo distribuiscono guadagnano sulla fornitura di servizi. Si sta sviluppando notevolmente soprattutto grazie al sistema operativo Linux che è stato concepito come free-software. Su questo argomento è disponibile un approfondimento.

software rentalware.
: Coperto da *copyright*. Il noleggio è a pagamento.

Installazione e configurazione del software

Prof. Simone Martini

3.3 Installazione e configurazione del software

Cosa avviene all'accensione di un calcolatore

Al momento dell'accensione, il controllo del calcolatore viene automaticamente preso da un piccolo programma memorizzato nella memoria centrale non volatile (ROM, o EPROM), chiamato comunemente **BIOS** (*Basic Input Output System*).

Il BIOS gestisce l'interfaccia tra il *software* e l'*hardware*, così da permettere al *software* - in particolare al sistema operativo - di essere indipendente dall'*hardware*. Con la sola modifica del BIOS è possibile far sì che uno stesso sistema operativo giri su componenti *hardware* diversi. Essendo un programma che risiede in modo permanente in memoria, il BIOS fa parte del cosiddetto **firmware** .

Una volta che il BIOS è in esecuzione, questi controlla per prima cosa l'integrità del sistema *hardware* (e in genere sullo schermo vengono segnalati quali controlli vengono eseguiti). Se i controlli hanno esito positivo, al loro termine il BIOS esegue la procedura di caricamento del sistema operativo (**boot**). Al termine del caricamento sarà il sistema operativo (SO) ad avere il controllo del processore.

Il sistema operativo continuerà ad interagire col BIOS, per esempio:

- per interpretare i caratteri immessi sulla tastiera;
- per gestire lo schermo;
- per controllare le comunicazioni sulle varie porte;

ma tutte queste interazioni sono trasparenti per l'utente (cioé avvengono senza che questi se ne accorga).

Per caricare il sistema operativo, questo deve essere memorizzato su un supporto accessibile al BIOS. Il sistema operativo non può essere memorizzato sulla RAM, come il BIOS, perché:

- è troppo grosso (diversi megabyte);
- per modificare il sistema operativo si dovrebbe modificare la RAM, un procedimento difficoltoso e dispendioso.

In condizioni normali, tale supporto è il disco rigido (o un disco accessibile in rete, nel caso di una rete locale configurata per *boot* remoto). Si chiama installazione la procedura di preparazione che:

- memorizza nella giusta posizione tutti i *file* necessari al caricamento del sistema operativo;
- configura alcuni di questi *file* in modo da adattarsi alle specifiche caratteristiche del calcolatore su cui il sistema operativo sarà eseguito.

Prima di installare un sistema operativo

Oggi la maggior parte dei PC vengono commercializzati con un sistema operativo pre-installato. Sapere installare un sistema operativo è però importante perché:

- se i *file* del sistema operativo si corrompono (per esempio a causa di errore *hardware* del disco, o di un virus), occorre ripristinare il sistema operativo ex-novo;
- aggiornare il sistema operativo ad una nuova versione comporta una procedura analoga all'installazione;
- il sistema operativo acquistato con il calcolatore può non essere quello voluto.

In particolare, si possono voler installare sistemi operativi *open-source*, come Linux, o si può voler installare un

sistema operativo compatibile (o più compatibile) con gli altri sistemi operativi di una rete locale.

La procedura di installazione di un sistema operativo è per buona parte automatizzata. Il punto di partenza è un supporto su cui la procedura di installazione sia memorizzata. In genere si tratta di un CD-ROM di installazione, che contenga la distribuzione ufficiale del sistema operativo. Descriveremo in dettaglio i vari passi da seguire, facendo riferimento a *Windows* XP. Non vi sono sostanziali modifiche nel caso di altri sistemi operativi.

Prima di installare:

- avere a disposizione il CD di installazione; eventualmente annotare il numero di serie (o di autenticità);
- verificare che il sistema su cui si installa possegga i requisiti minimi per poter far girare il sistema operativo prescelto; I requisiti più importanti: sufficiente RAM; sufficiente spazio disco. Alcune versioni recenti di *Windows* Me controllano anche la velocità della CPU. Un importante controllo avanzato: la disponibilità di driver compatibili col nuovo sistema operativo per tutti componenti del sistema.
- nel caso si stia installando una nuova versione di un sistema operativo, o in ogni caso quando non si sta effettuando la prima installazione: salvare tutti i dati importanti su un floppy, un CD-ROM, o un disco ad alta capacità (p.e. ZIP). Salvare almeno tutti i dati e documenti personali. Una copia di sicurezza del **disco di ripristino** sarebbe consigliata.

Che tipo di installazione

Sono possibili tre modi principali di installazione:

Nuova installazione
: Il nuovo sistema operativo sostituisce completamente il precedente. Tutte le vecchie impostazioni verranno perdute. A meno che non si tratti di una nuova versione del sistema operativo già in uso, la gran parte delle applicazioni non funzioneranno più e dovranno essere reinstallate. Le cartelle personali (nel caso di *Windows*: quelle esterne alla cartella *Windows*) sono in genere preservate (ma sono possibili nuove installazioni che formattano ex-novo tutto il disco rigido; p.e. in Apple Mac OS X: Erase and Install).

Aggiornamento
: Vengono aggiornati i soli *file* di sistema. Sono preservate le impostazioni, le configurazioni, le applicazioni e i dati. È un tipo di installazione possibile solo passando ad una nuova versione del sistema operativo già presente (o ad un sistema operativo vicino: p.e. da *Windows* 98 o Me a XP).

In nuova cartella
: Con questa opzione, laddove disponibile, è possibile avere due sistemi operativi installati su uno stesso calcolatore. È possibile scegliere quale dei due caricare al momento dell'accensione (attraverso un prompt del BIOS). Ciascun sistema operativo avrà le proprie applicazioni e le proprie configurazioni. In *Windows* XP i dati possono essere condivisi dai due sistemi operativi *Windows*.

Quale opzione scegliere?

La soluzione pulita è nuova installazione, a patto di:

- aver fatto una copia di *backup* di tutti i documenti;
- aver annotato tutti i **parametri di configurazione per la rete e Internet**;
- aver copia di tutte le applicazioni da (re-)installare e dei loro eventuali parametri.

In pratica si sceglierà spesso aggiornamento, ma talvolta qualche programma non funzionerà più e andrà comunque re-installato. In casi di grossi problemi di compatibilità, si cercherà di effettuare una nuova installazione.

Installare un sistema operativo

- Inserire il CD di installazione
- Se il programma di installazione non parte da solo: cercare sul CD l'icona dell'installatore e cliccarla due volte.
  Windows :
  aprire la cartella Risorse del computer, poi aprire l'icona del CD-ROM (p.e. winhphome); l'icona da aprire è **Setup**.
  Mac OS X:
  aprire l'icona del CD che compare sul *desktop*; l'icona da aprire è **Installa Mac OS X**.
- La procedura presenta alcune finestre; scegliere l'opzione corrispondente all'installazione e, quando compare la scelta del tipo di installazione, selezionare quella desiderata. Confermare la scelta, cliccando su **Avanti**. Tra le finestre presentate ce ne sarà anche una con il testo della licenza d'uso. La lettura è noiosa, ma istruttiva. Occorre confermare la sua accettazione, che ha valore legale in Italia. È possibile che venga richiesto l'inserimento del codice seriale del prodotto, che si trova nella confezione del sistema operativo. Confermare sempre le scelte.
- La successione delle finestre che si presenteranno dipende dallo specifico sistema operativo da installare (e dalla sua versione). Ecco alcune delle richieste che si possono presentare:
  Rapporto sull'aggiornamento
  Se l'installazione è di tipo aggiornamento, è possibile chiedere di mostrare i problemi *hardware* e *software* a cui si può andare incontro. Non fa mai male produrre tale rapporto.
  Opzioni avanzate di installazione
  L'utente avanzato può selezionare manualmente quali moduli del sistema operativo installare. Non è un'opzione indicata per chi non sa con esattezza cosa fare.
  Accesso facilitato
  Permette un'installazione più semplice ad alcuni utenti disabili (per esempio, permette di usare un sintetizzatore vocale che guida tra le opzioni). Riprenderemo questo aspetto più avanti.
  Lingua e località
  Scegliere la lingua principale e la nazione in base alla quale saranno impostate le preferenze relative al formato delle date e dell'ora, al fuso orario, alla tastiera in uso, ecc.
  File di installazione aggiornati
  Se il calcolatore è connesso a Internet tramite una rete locale, selezionando questa opzione l'installazione si collega coi siti dove reperire alcuni *file* aggiornati. Se non c'è una connessione attiva, si può ignorare l'opzione.
  Disco/Cartella di installazione
  Selezionare il disco su cui installare (se ne esiste più di uno). Se sul disco selezionato c'è già una copia del sistema operativo, indicare il nome della cartella su cui installare il nuovo sistema operativo. Indicando il nome della cartella del vecchio sistema operativo, il nuovo sistema sarà installato sopra la vecchia versione, cancellando preferenze e configurazioni. Se si desidera mantenere una copia del vecchio sistema operativo, indicare un nuovo nome.
  Modifiche al *file system*
  Nell'installazione di *Windows* XP, può essere richiesto se si desidera convertire il *file system* in formato NTFS. Si consiglia di **lasciare intatto il *file system* corrente** (la conversione può essere fatta in un secondo momento: si veda l'approfondimento **Aggiornare un sistema: alcuni argomenti**).

Dopo tutte queste finestre, inizia l'installazione vera e propria: i *file* vengono copiati e configurati. Apposite barre segnalano, con molta approssimazione, lo stato di avanzamento del processo.

La prima configurazione

Quando la registrazione dei *file* è terminata, vengono presentate altre finestre per la configurazione (o **impostazione**) del sistema. Le richieste principali:

Opzioni internazionali
: Le opzioni suggerite sono in genere quelle corrette. È possibile sceglierne altre relative al formato dei numeri, dei prezzi, delle date; oppure selezionare tastiere diverse da quelle standard per la lingua selezionata.

Nome del PC
: Indicare il nome col quale il PC sarà noto sulla rete locale.

Data e ora
: Inserire data e ora corrette.

Utenti
: Se il sistema è **multiutente** , come *Windows* XP o Apple Mac OS X, indicare i nomi degli *account* ed eventualmente le *password*. Almeno un utente sarà l'utente principale, che possiede i diritti di amministratore del sistema. Altri utenti potranno essere inseriti in qualsiasi momento.

Configurazione Internet
: Varie finestre, possibilmente di un programma separato, richiedono di inserire le opzioni per la connessione a Internet. Tale configurazione può essere fatta in un secondo momento e la tratteremo in modo separato.

Registrazione
: La registrazione consiste nel contattare la casa produttrice del sistema operativo per associare il numero seriale della copia installata al proprio nome o addirittura (p.e. nel caso di *Windows* XP) allo specifico *hardware*. La registrazione può avvenire via Internet, se il calcolatore è connesso alla rete, o per telefono, o per posta. In ogni caso si tratta di annotare il numero seriale e, nel caso di *Windows* XP, lo ID di installazione (un lungo numero che corrisponde all'*hardware* e che compare nella finestra di registrazione). I dati devono essere trasmessi alla casa produttiva. Nel caso di XP, in risposta alla registrazione sarà comunicato un ulteriore codice (ID di conferma) che (se si registra per telefono) andrà inserito nell'apposito campo della finestra di registrazione. Perché registrare:
  - per essere coperti dalla garanzia
  - per ricevere informazioni
  - perché le copie non registrate di *Windows* XP smettono di funzionare dopo 30 giorni dall'installazione. Per limitare la pirateria, Microsoft non solo richiede la registrazione, ma blocca il codice seriale dopo cinque installazioni fatte con la stessa copia su *hardware* diversi. In caso di blocco, contattare il servizio clienti.

Al termine di tutto questo lavoro (o, a seconda dei casi, anche durante) il calcolatore deve essere riavviato. Dopo il riavvio è pronto: compare la scrivania o la maschera per il login di un utente.

Connettesi ad internet

Per connettersi ad Internet è necessario che qualcuno ci fornisca connettività. Si tratta del nostro **ISP** , *Internet Service Provider*. Ce la può fornire in due modalità principali:

dedicata
: La connessione è sempre attiva

commutata
: La connessione è stabilita al bisogno, mediante una richiesta (una telefonata) da parte nostra

In entrambi i casi il calcolatore deve essere dotato di **modem**, una periferica che lo rende in grado di dialogare col nostro ISP attraverso la linea di collegamento.

Se la connessione è dedicata, il vostro ISP vi ha fornito tutti i dati necessari per la configurazione:
aprite il Pannello di Controllo e selezionare Rete e/o Internet e immettete tali dati.

Se la connessione è commutata, i dati di cui avete bisogno, e che vanno inseriti nel Pannello di controllo (sotto Rete e/o Internet) sono:

- numero di telefono del vostro ISP
- nome del vostro *account* presso lo ISP (*username*)

- *password*
- protocollo usato per la connessione: usualmente **PPP**
- DNS è un indirizzo IP (cioé un codice composto da quattro numeri separati da punti; per esempio 152.144.2.143); non sempre è necessario.

Lo ISP vi ha anche fornito altri dati, tecnicamente non necessari per la connessione, ma indispensabili per usare la posta elettronica (e-mail). Questi dati sono:

- *server* POP (o IMAP) un nome simbolico, per esempio pop.myisp.it;
- *server* SMTP un altro nome simbolico.

Personalizzare il sistema

Ogni sistema operativo ha molte opzioni che possono essere modificate, per adattare il suo funzionamento ai bisogni o ai gusti dell'utente. Le opzioni si trovano nel Pannello di Controllo.

- *Windows*: pulsante *Start*, poi Pannello di Controllo;
- Mac OS 9: menù mela, poi Pannello di Controllo;
- Mac OS X: Preferenze di Sistema, nella cartella Applicazioni o nel Dock.

Le diverse pagine del Pannello di Controllo sono autoesplicative; è utile divertirsi un po' a modificare i vari parametri. Le categorie principali del pannello sono riferite a:

Aspetto
: Permette di cambiare lo **sfondo** (***desktop***); il **tema** (l'aspetto grafico e i colori dei menu e delle finestre); la disposizione delle icone sul ***desktop***; lo ***screen-saver***;

hardware
: Permette di installare nuove **periferiche** (stampanti, fax, display); di controllare il funzionamento della **tastiera** (p.e. il tempo dopo quando un tasto ripete il carattere) o del **mouse** (la velocità di trascinamento); I moderni sistemi operativi (col supporto di tecnologia *hardware*, p.e. protocollo USB) permettono di installare periferiche semplicemente collegandole alla presa opportuna. Il sistema operativo rileva la presenza della periferica e si configura per permettere l'accesso (tecnologia *plug and play*).

Suono
: Controllo del **volume** e del tipo di suoni corrispondenti ad eventi particolari (errori, segnalazioni, ecc.);

Lingua
: Personalizzazioni relative alla lingua usata;

Account utenti
: Inserire nuovi utenti e relative *password*; gestione dei loro diritti;

Rete e Connessione a Internet
: Permette di inserire e modificare le opzioni necessarie al funzionamento della rete locale e dalla connessione ad un ISP;

Installazione applicazioni
: Gestisce l'installazione e la rimozione di nuove applicazioni;

Accesso universale (o facilitato)
: Gestisce alcuni ausili per l'utente disabile.

Le ultime due categorie sono descitte in modo dettagliato nelle prossime due sezioni.

Installare nuove applicazioni

Installare un'applicazione significa, come per il sistema operativo, copiare i relativi *file* in un'opportuna posizione del disco, configurare alcuni parametri, e infine far sì che il sistema operativo e le altre applicazioni sappiano

dell'esistenza della nuova applicazione.

Apple Mac OS X è il sistema che permette l'installazione più semplice. Basta spostare l'icona dell'applicazione nella cartella Applicazioni. Non vi sono pannelli di controllo relativi all'installazione di applicazioni.

Affinché tutto avvenga correttamente, in *Windows* non è sufficiente copiare l'applicazione in una cartella opportuna. Il pannello di controllo Installazione applicazioni permette di:

aggiungere nuovi programmi
: Selezionare il CD da cui si vuole installare e cliccare sull'applicazione; una serie di finestre guida l'installazione;

rimuovere/cambiare programmi
: Selezionare l'applicazione da rimuovere e confermare la scelta; una serie di finestre guida la disinstallazione Le applicazioni si devono installare (e soprattutto disinstallare) tramite il pannello di controllo perché:
  - un'applicazione è composta da molti *file*, alcuni di questi nascosti e memorizzati in varie locazioni del disco (preferenze, *driver*, storia, ecc.);
  - occorre installare/cancellare tutti questi *file* per un funzionamento corretto;
  - occorre aggiornare alcuni menù (per esempio quello delle applicazioni);
  - occorre aggiornare alcuni *file* molto importanti, trai quali il Registro di sistema.

installare nuove componenti del sistema operativo
: Permette di installare parti del sistema operativo che fossero state tralasciate al momento dell'installazione originale.

Accesso universale

La tecnologia fornisce una gamma molto ampia di ausili coi quali un utente disabile può accedere ad un calcolatore. Vi sono interfacce e periferiche specializzate (p.e. tastiere e display braille) che possono essere installate anche su calcolatori personali. Queste possibilità sono di grande rilevanza, ma non possono essere affrontate qui.

I sistemi operativi moderni forniscono alcune opzioni che possono facilitare l'uso del calcolatore ad alcuni utenti, specialmente quelli con deficit visivo. Alcune opzioni sono controllabili dalle normali categorie del pannello di controllo; altre dal pannello **Accesso facilitato** (o **universale**). Segnaliamo le seguenti (che dipendono ovviamente dal sistema operativo usato):

scritte bianche su sfondo nero:
: il contrasto maggiore consente ad alcuni miglior lettura;

icone ingrandite/zoom:
: permette di ingrandire solo alcune porzioni dello schermo, attorno al cursore;

forma del cursore sullo schermo;
sintetizzazione vocale dei comandi e dei bottoni puntati dal cursore;
flash dello schermo invece di suono in caso di segnalazione;
uso dei tasti invece del mouse per lo spostamento del cursore;
combinazione di tasti come sequenza:
: alcuni utenti hanno difficoltà a premere più tasti contemporaneamente (pe. maiusc+a; oppure alt+`);
: l'opzione permette di comporre queste combinazioni come sequenze;

ritardo dei tasti:
: introduce ritardo tra quando un tasto è premuto e quando un tasto è accettato, permettendo la non ripetizione del carattere con tasto premuto a lungo.

Configurare un client di posta elettronica

La posta elettronica è una tipica applicazione organizzata come *client/server*.

Il *server*, che risiede in genere presso lo ISP, è responsabile di:

- ricezione dei messaggi e loro memorizzazione in una casella postale;
- invio dei messaggi.

Si tratta di due servizi indipendenti che possono essere svolti da *server* diversi.

Il *client*, cioé l'applicazione che gira sul calcolatore dell'utente, richiede al *server*:

- il trasferimento dei messaggi dalla casella postale al proprio calcolatore
- l'invio di messaggi

Il *client*, inoltre, gestisce in loco altri servizi per l'utente:

- organizzazione dei messaggi in cartelle
- mantenimento di copie dei messaggi inviati
- gestione degli indirizzi (archivio, *nicknames*, etc.)

Affinché *server* e *client* possano dialogare correttamente, devono condividere lo stesso **protocollo**. Vi sono due protocolli per la ricezione della posta e uno per l'invio. Ricezione della posta. Protocolli:

POP
: Il più diffuso e supportato da tutti i *server*. Consente di scaricare la posta dalla casella del *server* verso il *client*;

IMAP
: Non supportato da tutti i *server*. Consente di leggere la posta e organizzarla in cartelle mantenendo una copia dell'organizzazione sul *server*.

POP è il protocollo comune per l'utente personale. **IMAP** è la scelta d'elezione per coloro che devono leggere la propria posta da più postazioni diverse (ufficio, casa, portatile) e vogliono avere sempre a disposizione lo stesso archivio di posta. Richiede un più consistente traffico tra *server* e *client* ed è pertanto sconsigliato se si dispone di una connessione lenta commutata.

Per configurare un cliente **POP** , occorre inserire le seguenti informazioni:

- il nome simbolico del *server* (p.e. pop.myisp.it) nella casella del *server* per la posta in ingresso;
- il nome del proprio *account* sul *server* (p.e. mariarusso);
- la propria ***password***;
- se usare una **connessione sicura**, qualora il *server* la richieda/permetta (controllare con lo ISP): con una connessione normale la *password* viene inviata al *server* in chiaro e potrebbe essere intercettata lungo il tragitto (*sniffing*). Una connessione sicura usa in genere il protocollo SSL (*Secure Server Layer*), che cifra la *password* prima di inviarla. Selezionando l'opzione **SSL** si deve indicare anche il modo di autenticazione; il più comune è quello mediante *password*. Modi più sofisticati (p.e. *kerberos*) sono rari: in ogni caso lo ISP fornisce le informazioni in merito.
- se lasciare o meno la posta sul *server*. È possibile lasciare una copia dei messaggi sul *server*, in modo da poterli ritrovare controllando la posta da un'altra postazione. Non tutti i *server* supportano questa opzione, perché appesantisce l'uso del disco sul *server*.

Per configurare un cliente **IMAP** , occorre inserire le informazioni come per **POP** . Inoltre deve essere inserito il prefisso del path IMAP, cioè un path del *file system* del *server* che indica la posizione delle cartelle in cui il *server* memorizza i messaggi e la loro organizzazione. Lo ISP che supporta **IMAP** fornisce anche questa informazione.

Invio della posta. Protocollo SMTP. Per inviare la posta il *client* si collega ad un *server* SMTP. La configurazione è semplice; occorre inserire le seguenti informazioni:

- il nome simbolico del *server* (p.e. smtp.myisp.it) nella casella del *server* per la posta in uscita;
- se usare una connessione sicura, qualora il *server* la richieda/permetta (controllare con lo ISP). In tal caso indicare se usare SSL, il tipo di autenticazione, il nome dell'*account* (potrebbe essere diverso da quello per la ricezione) e la *password*. La connessione sicura per SMTP non è molto diffusa, perché è sostituita da un'autenticazione automatica basata sul riconoscimento del numero IP del calcolatore su cui gira il *client*: sono accettati messaggi in partenza solo da calcolatori il cui numero IP è stato fornito dallo ISP che fornisce anche il *server* SMTP.

Conclusioni

Abbiamo visto le modalità operative per la configurazione elementare di un sistema operativo e di alcune applicazioni. Informazioni più dettagliate si troveranno nei moduli seguenti, in specie per quanto riguardo la configurazione di rete, che sarà trattata dopo avere introdotto le definizioni e i concetti fondamentali.
Il modulo **aggiornare un sistema** contiene alcuni complementi relativi all'aggiornamento di un sistema di calcolo.

Elementi di linguaggi di programmazione

Prof. Simone Martini
3.4 Elementi di linguaggi di programmazione

Manipolazione di simboli

Un calcolatore manipola sequenze di caratteri senza comprenderle.

Le regole secondo le quali avviene la manipolazione costituiscono il programma che il calcolatore esegue.

Le regole - i programmi - sono scritte in un linguaggio formale artificiale (cioè progettato espressamente per questo scopo), nel quale le modalità di manipolazione sono esprimibili in modo semplice, chiaro e non ambiguo.

Linguaggio di programmazione :

un linguaggio formale nel quale esprimere comandi e **flusso di controllo**

per esempio:

```
Leggi x; Per 10 volte trasforma x in x*x; Stampa x;
```

L'esecutore di uno specifico linguaggio di programmazione è la macchina astratta di quel linguaggio.

Gerarchie di linguaggi di programmazione

Lo *hardware* ha un suo linguaggio di programmazione per il quale la macchina astratta coincide con quella concreta. è il **linguaggio macchina** (di quello specifico *hardware*).

Il linguaggio macchina manipola direttamente le sequenze di bit fornite dall'*hardware*, utilizzando le operazioni primitive dell'*hardware* stesso (operazioni aritmetiche, salti).

Programmare in linguaggio macchina è faticoso, costoso ed espone ad errori.

Fin dagli anni 50 sono stati sviluppati linguaggi più evoluti, ciascuno dei quali progettato sopra un linguaggio più rudimentale.

Possiamo suddividere i linguaggi di programmazione in livelli, a seconda di quanto essi sono vicini al linguaggio macchina.

Basso livello:

- **Linguaggio macchina** Le operazioni disponibili sono quelle direttamente fornite dall'*hardware*; ogni operazione è codificata da una sequenza di bit; ogni dato è indicato dall'indirizzo binario della parola di memoria in cui è memorizzato. Ogni indirizzo è espresso in modo assoluto (rispetto a tutta la memoria disponibile). Un programma è una sequenza di bit, che viene direttamente interpretata dall'*hardware*.
- **Linguaggio assembler** (o assemblativo) Le operazioni sono quelle direttamente fornite dall'*hardware*, ma sono indicate da nomi convenzionali (mnemonici); i dati sono indicati da nomi, che corrispondono a indirizzi. Gli indirizzi sono espressi in modo relativo rispetto all'inizio del programma, permettendo così più semplici modifiche. Il programma, per essere eseguito, viene tradotto in linguaggio macchina da un programma traduttore detto **assemblatore** .

Alto livello:

- **Linguaggi procedurali** Le operazioni disponibili sono ampie e non legate a quelle fornite dall'*hardware*. I dati sono indicati in modo totalmente indipendente dalla loro memorizzazione. Si tratta di linguaggi progettati affinchè la scrittura dei programmi sia semplice, elegante e, dunque, di facile comprensione e verifica. Per essere eseguito, un programma deve essere tradotto in linguaggio macchina da un programma traduttore detto **compilatore** , oppure deve essere interpretato (cioè eseguito da un altro programma, l'interprete). Linguaggi procedurali di rilievo sono (o sono stati) FORTRAN, COBOL, BASIC, Pascal, C. Tra i linguaggi procedurali alcuni sono detti orientati agli oggetti; tra questi ricordiamo C++ e Java.

Oggi tutti i progetti industriali si realizzano in linguaggi ad alto (ed altissimo) livello.

La programmazione in linguaggio assembler è limitata alla realizzazione di semplici driver.

Da un punto di vista didattico - insegnare a programmare - i linguaggi ad alto livello sono evidentemente la scelta d'elezione:

- permettono di concentrarsi sulla logica dell'algoritmo;
- forniscono costrutti per descrivere in modo conciso tale logica;
- mettono a disposizione strumenti per il test del programma.

Linguaggio di alto livello

A basso livello (livello macchina o assembler) un programma è una sequenza di istruzioni operative interrotta da istruzioni di salto:

```
"se si verifica una certa condizione, vai all'istruzione numero xx"
```

La scrittura di programmi complessi diviene difficile, perché il programma non rispetta la struttura della soluzione: i salti si incrociano l'uno con l'altro ed è impossibile seguire la logica della soluzione. Il risultato si chiama in gergo "programmazione a spaghetti".

Per risolvere un problema complesso:

- lo si decompone in parti più semplici, possibilmente indipendenti;
- si risolve ogni parte separatamente;
- si combinano le varie soluzioni per ottenere la soluzione al problema originario.

I **linguaggi ad alto livello** mettono a disposizione strumenti linguistici per descrivere, all'interno del programma stesso, la struttura logica della decomposizione.

I principali strumenti sono:

- strutture di controllo Invece del semplice salto dell'assembler, troviamo costrutti che gestiscono il flusso del controllo in modo strutturato;
- strutture di modularizzazione Parti distinte della soluzione sono programmate come programmi distinti e indipendenti;
- strutture dati Il linguaggio fornisce costrutti per definire e manipolare dati strutturati.

Prendiamo in considerazione queste tre caratteristiche in successione.

Costrutti di controlli strutturato

Solo alcune forme particolari di controllo (p.e. di iterazione) sono permesse nei moderni linguaggi ad alto livello. Le più rilevanti sono:

- **iterazione determinata** : *for* i := <inizio> *to* <fine> *do* <corpo del *for*>
  <corpo del *for*> è eseguito <fine>-<inizio> volte, al variare della variabile i
- **iterazione indeterminata** : *while* <condizione> *do* <corpo del *while*>
  <corpo del while> è eseguito fino a quando <condizione> non diventa vera
- **condizionale** : *if* <condizione> *then* <ramo *then*> *else* <ramo *else*>
  se <condizione> è vera, si esegue <ramo then>; altrimenti si esegue <ramo else>
- **eccezione** : *try* <comando> *catch* <gestore-errore>
  si esegue <comando>; se si verifica un errore (specificato in <gestore-errore>) si esegue <gestore-errore>

è chiaro che queste strutture di controllo si possono realizzare anche a basso livello.

Un linguaggio ad alto livello impone che tutti i programmi siano redatti usando solo queste strutture, che sono più semplici da analizzare della "programmazione a spaghetti".

Costrutti di modularizzazione

Per poter efficacemente decomporre la soluzione di un problema, i linguaggi ad alto livello permettono di risolvere una parte del problema in modo indipendente dalla soluzione (cioè dal programma) di altre parti. Per far ciò mettono a disposizione:

- **procedure** e **funzioni** : Porzioni di programmi relativemente autosufficienti contraddistinte da un nome e da argomenti; codificano la soluzione di una parte del problema: per invocarli non è necessario conoscere i dettagli della soluzione, ma solo il nome e quali argomenti richiedono;
- **parametri** : Argomenti delle procedure, mediante i quali è gestito il flusso delle informazioni dal programma principale alla procedura e viceversa;
- **ambiente dinamico** : I nomi delle variabili e i loro spazi di memorizzazione sono gestiti al livello della singola procedura, senza necessità di aver nomi unici in tutto il programma.

Costrutti per la strutturazione dei dati

Il linguaggio fornisce in modo diretto costrutti per definire e manipolare dati strutturati.

- nuovi tipi, oltre a dati numerici, si possono definire ed usare nel linguaggio nuovi tipi di dato:
  - enumerazioni;
  - insiemi;
  - matrici;
  - aggregati eterogenei (record);
  - ecc.

L'uso di questi tipi permette di rappresentare in modo naturale la logica del problema, invece di ricorrere a codifiche artificiali;

- controllo automatico dei tipi La presenza di un ricco sistema di tipi permette di rilevare alcuni errori semantici, che si manifestano come errori di tipo. Per esempio:
  3+pippo deve essere sbagliato, non si possono sommare interi e stringhe.

Il controllo dei tipi è analogo al controllo dimensionale in fisica: se una formula che esprime una velocità non è uno spazio diviso un tempo, la formula deve essere sbagliata.

Il controllo può essere:

- statico prima dell'esecuzione del programma;
- dinamico durante l'esecuzione del programma.

In ogni caso è il sistema che rileva e segnala l'errore.

Tecnologie di programmazione

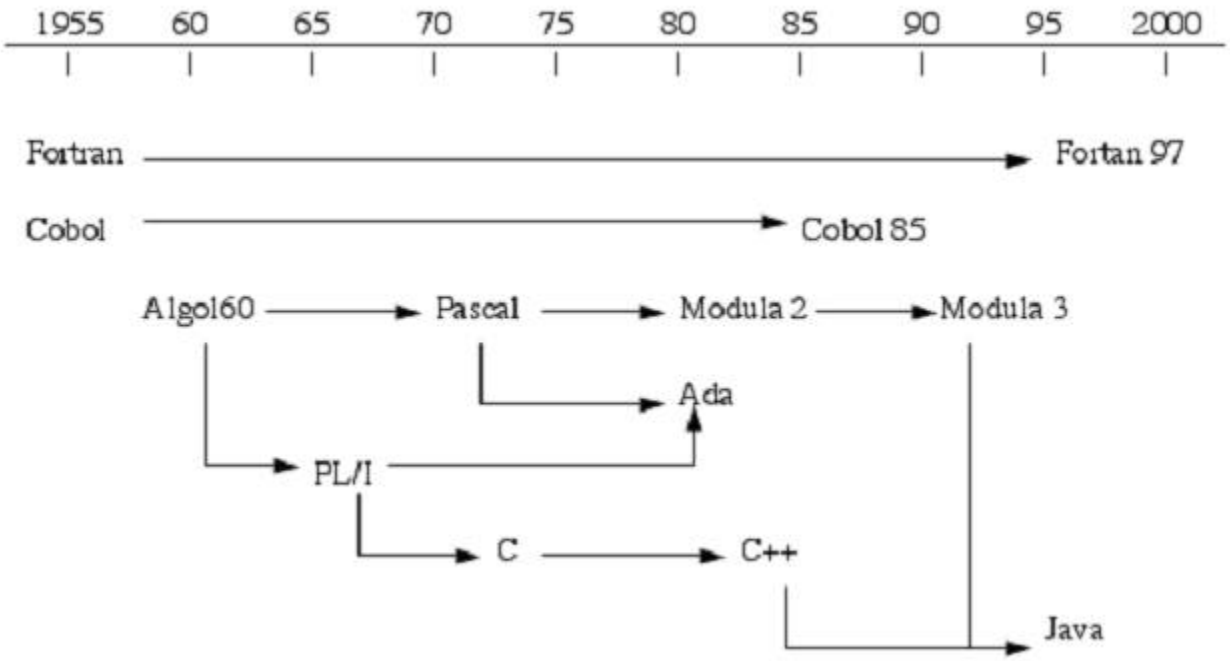
Prof. Simone Martini
3.5 Tecnologie di programmazione

Evoluzione dei linguaggi di programmazione

La seguente tabella mostra alcuni (pochissimi!) delle centinaia di linguaggi di programmazione ad alto livello che sono stati introdotti, a partire dal primo di essi, Fortran. Tutti i linguaggi indicati sono ancora in uso, in una qualche

loro versione, con l'eccezione di Algol, PL/I e, probabilmente, Modula. Ma ciascuno di essi ha dato contributi essenziali per i successori. Algol è stato il primo linguaggio con costrutti di controllo veramente strutturati, Modula 3 è stato il banco di prova di alcuni concetti orientati agli oggetti che troviamo in Java.

Fortan e Cobol, due linguaggi che hanno dominato la scena fino alla fine degli anni 70, oggi rimangono in vita perché esistono grandi quantità di *software* scritto in essi che non è conveniente trasformare in altri linguaggi.



Linguaggi orientati agli oggetti

I linguaggi di programmazione come Pascal o C si basano su una chiara distinzione tra dati e procedure che agiscono su di essi.

è possibile adottare una prospettiva diversa:

- un programma è costruito mediante una collezione di entità (**oggetti**);
- ciascuna entità ingloba sia un dato che le procedure (**metodi**) che lo manipolano;
- il dato non è immediatamente accessibile;
- l'accesso al dato è permesso solo attraverso le procedure fornite dall'entità che lo ingloba;
- le entità comunicano mediante **messaggi** coi quali richiedono al destinatario l'esecuzione di un metodo.

Un **oggetto** consiste dunque di:

- dati nascosti;
- operazioni pubbliche, dette metodi.

Un programma in un linguaggio orientato agli oggetti consiste in una serie di **messaggi** agli oggetti: un messaggio è la richiesta di eseguire un certo metodo.

Un esempio.

Possiamo immaginare un cerchio come un oggetto capace di rispondere a due messaggi, uno che richiede l'area del

cerchio, un altro che richiede il raggio del cerchio. Abbiamo dunque un oggetto con due metodi: area e raggio. Se c è un cerchio, nello specifico senso appena definito e usando la sintassi del linguaggio Java, possiamo richiedere l'area del cerchio come:

```
c.area();
```

Osserviamo che non abbiamo detto come un cerchio è rappresentato all'interno dell'oggetto. Questo fa parte dei dati nascosti, che non sono manipolabili se non vi sono metodi che lo fanno. Questo incapsulamento dell'informazione è una caratteristica molto importante dei linguaggi orientati agli oggetti. Essa garantisce, per esempio, che ogni programma che usa un cerchio non dipende dallo specifico modo in cui un cerchio viene rappresentato.

Linguaggi orientati agli oggetti:

- precursori: Simula, Smalltalk;
- in uso: C++, Java.

Metodologia ad oggetti

L'oggetto è un costrutto di incapsulamento molto flessibile.

Possiamo applicare la prospettiva ad oggetti a realtà molto diverse:

- numeri interi
- strutture dati
- *file systems*
- basi di dati
- finestre sullo schermo
- ecc.

La metodologia ad oggetti permette la costruzione di sistemi facilmente estensibili, in modo particolare perché i linguaggi orientati agli oggetti forniscono strumenti sofisticati (ereditarietà) per il riuso del codice (cioè di parti di programma).

Ingegneria del software

Il costo dell'hardware diminuisce drasticamente ogni anno.

Il costo globale di un sistema di calcolo è di gran lunga dominato dal costo del suo *software*.

Costi di:

- progettazione iniziale
- realizzazione iniziale
- aggiornamento
- correzione degli errori

Di questi costi, quello della manutenzione, cioè dell'aggiornamento del sistema e della correzione dei suoi errori (veri o dovuti a specifiche mal poste), è di gran lunga quello più oneroso.

L'evoluzione dei linguaggi di programmazione può essere letta come una ricerca di ambienti linguistici nei quali:

- sia semplice programmare;
- sia semplice modificare programmi già scritti.

Semplice = poco costoso

Tecnologie *software*:

strumenti informatici per la produzione di grossi sistemi

Sono basate sui linguaggi di programmazione, ma comprendono molti altri aspetti. Alcuni di essi:

- la specifica
- la modularizzazione
- la ricerca degli errori (*debugging*)
- la prova (*testing*)
- la gestione delle varie versioni
- la produzione da parte di vari gruppi di lavoro
- la documentazione
- la manutenibilità

Della definizione, realizzazione e valutazione di queste tecnologie si occupa il settore dell'informatica detto **ingegneria del *software*** .

Conclusioni

I linguaggi di programmazione sono linguaggi artificiali che permettono di esprimere algoritmi in modo non ambiguo. I moderni linguaggi di programmazione da una parte forniscono al programmatore costrutti sofisticati per descrivere all'interno del programma la struttura del problema da risolvere. Dall'altra costringono il programmatore ad usare schemi di programmazione strutturata, che sono più facilmente verificabili della programmazione a spaghetti.

L'ingegneria del *software* ha come obiettivo la realizzazione e la valutazione di tecnologie mediante le quali la produzione di programmi sia sempre meno costosa e, soprattutto, meno soggetta al verificarsi di errori dopo la posa in opera del sistema.

La programmazione orientata agli oggetti è oggi una delle maggiori tecnologie in uso. Le caratteristiche fondamentali di un linguaggio orientato agli oggetti saranno affrontate in un approfondimento.

Sebbene non facciano parte direttamente dei linguaggi di programmazione, negli anni recenti hanno avuto grande sviluppo i linguaggi per la descrizione della struttura di documenti e, più in generale, di dati non strutturati. Tra questi il più noto è certamente HTML, un linguaggio che permette l'annotazione di documenti e dati multimediali. Moduli successivi affonteranno queste tecnologie (HTML, XML, ecc.), che stanno divenendo lo standard per la definizione e l'annotazione di dati non strutturati.

Approfondimenti

L'elaborazione di documenti

Prof. Maurizio Gabbrielli
3.2 Software applicativo

Il documento e il documento elettronico

Un **documento** è una frase scritta in un qualche linguaggio che abbia un contenuto, una struttura, una semantica e che può essere in relazione con altri documenti.

Un **documento elettronico** è un documento la cui rappresentazione fisica è in forma di bit all'interno di un sistema informatico.

Esempi di documenti:

Testi debolmente strutturati:
romanzi, racconti, poesie, saggi, articoli, eccetera.
Testi fortemente strutturati: elenchi telefonici, schede cliniche, programmi di computer, eccetera.

Ipertesti:
testi contenenti collegamenti interni che ne permettono una lettura non sequenziale (vedi più avanti).
Non-testi:
immagini, schemi, progetti, fotografie, filmati, animazioni, eccetera.

Operazioni tipiche sui documenti

Creazione:
del contenuto, della struttura, dei metadati.
Cancellazione:
del contenuto, della struttura, dei metadati.
Acquisizione:
via tastiera, rete, scanner, penna ottica, eccetera.
Modifica:
aggiornamento, correzione, pulitura, montaggio,strutturazione, eccetera.
Condivisione:
stampa, trasmissione, conversione, eccetera.
Interazione:
esecuzione di istruzioni di I/O contenute all'interno del documento.
Navigazione:
accesso a documenti correlati mediante una delle relazioni che li collegano.

Componenti dei documenti e operazioni

I programmi di elaborazione di documenti e, in particolare, gli editori di testo, operano a vari livelli sui componenti di un documento, ad esempio:

carattere:
unità minima di rappresentazione, corrispondente solitamente ad un tasto della tastiera (ed ad un valore di un opportuno codice di rappresentazione). Operazioni tipiche sui caratteri sono: selezione del tipo di carattere (Arial, Times New Roman, Courier ecc) della dimensione (9pt, 10pt., 12 pt., 14 pt., ecc), dello stile (normale, **grassetto**, *corsivo* ecc) e del colore (rosso, nero, verde ecc) .
parola:

sequenza di caratteri divisa da separatori di parola dalle altre parole. Come separatori di parole si usano di solito spazi, fine linea, fine pagina, e segni di punteggiatura ma ci sono eccezioni (ad esempio nei linguaggi di programmazione). Conteggio, controllo ortografico e divisione delle parole (hyphenation) sono operazioni tipiche sulle parole.

frase:

sequenza di parole.

paragrafo:

frase delimitata da separatori di paragrafo. Un ritorno a capo, o CR (carriage return) è un separatore di paragrafo. Sistemi operativi diversi utilizzano separatori diversi per i paragrafi. Ad esempio:

- DOS e *Windows*: Carriage Return + Line Feed (ASCII 13 e ASCII 10)
- Mac: Carriage Return (ASCII 13)
- Unix: Line Feed (ASCII 10)

Queste differenze fanno sì che lo stesso documento, usato su sistemi operativi diversi anche con lo stesso tipo di applicazione, possa dare dei problemi di conversione.

I programmi di elaborazione testi inoltre usano introducono altre strutture quali sezioni, colonne, oggetti grafici, eccetera, che richiedono ulteriori separatori specifici.

Componenti dei documenti e operazioni

Per alcuni sistemi di elaborazione di documenti, orientati alla stampa, è importante il concetto di **pagina**.

Le pagine sono di dimensioni prefissate, ma esistono formati diversi a seconda degli scopi ed anche del paese. Ad esempio, in Europa esistono i formati A, definiti a partire dal formato A0 (84x118.8 cm). Il formato più comune è A4 (21x 29.7 cm). In USA invece sono più comuni i formati lettera e legale. Altri tipi di pagina possono essere usati per scopi particolari (buste, biglietti da visita, eccetera)

Per alcuni sistemi di elaborazione di documenti, orientati alla visualizzazione su schermo, è importante invece il concetto di **pagina visualizzabile** (o **schermata**). Dato che gli schermi sono di capacità diverse per quanto riguarda dimensioni, risoluzione, numero di colori. Una schermata che risulta adatto ad uno schermo può risultare inadeguata per un altro.

Ipertesti

Un testo può essere considerato come una sequenza di caratteri, strutturati opportunamente in parole, frasi, paragrafi ecc,. che è inteso per un accesso ed una lettura sequenziale in una qualche direzione (ad esempio da destra a sinistra, dall'alto al basso) che dipende dalla cultura di riferimento.

Un **ipertesto** è un testo che contiene dei collegamenti diretti (detti *link*, o collegamenti ipertestuali) fra alcune sue parti. Tali collegamenti permettono un accesso ed una lettura non sequenziale dell'ipertesto, navigando attraverso di esso. Queste pagine sono un esempio di ipertesto: ad esempio, cliccando su una linea dell'indice ci si porta alla pagina che contiene la trattazione dell'argomento specificato nell'indice.

I documenti ipertestuali, esistenti da molto tempo, si sono diffusi negli ultimi anni grazie all'avvento del *World Wide Web* (WWW), dato che le pagine di un sito *Web* sono tipicamente documenti ipertestuali multimediali. In effetti lo stesso WWW può essere visto come un ipertesto multimediale distribuito.

I documenti ipertestuali vengono creati usando i cosiddetti linguaggi di marcatura (di *markup*). Questi sono dei formalismi che permettono di usare dei marcatori (detti anche tag) per definire la struttura logica del documento, le relazioni che legano documenti diversi e gli aspetti di presentazione del documento (stile, formattazione, eccetera). *Standard Generalized Markup Language* (SGML, ISO 8879) è un linguaggio per descrivere linguaggi che codificano documenti. Fra i linguaggi di marcatura più diffusi ricordiamo HTML (che è una specifica istanza di SGML).

Uso di un editore di testi: funzioni di base

I programmi di videoscrittura (o editori di testi) sono fra i più diffusi programmi applicativi e permettono di scrivere, modificare, formattare documenti di vario genere,

I programmi per l'elaborazione di testi (o editori di testi), quali ad esempio *Microsoft Word* oppure *Corel WordPerfect*, sono di norma di uso estremamente semplice in quanto tutte le principali funzioni sono disponibili da opportuni menu ed inoltre è disponibile un manuale ed un sistema di aiuto on-line (anch'esso accessibile tramite menu). L'autoapprendimento di questi programmi applicativi è quindi abbastanza facile. Ricordiamo qui alcune caratteristiche comuni ai vari editori di testi.

Tutti gli editori di testi permettono le seguenti funzioni:

a capo automatico:
: una volta che sono stati impostati i margini del documento (usando i comandi grafici su un opportuno righello) quando arriviamo alla fine di una riga il cursore, che indica la posizione nella quale verrà inserito il prossimo carattere, va a capo automaticamente, eventualmente dividendo in modo opportuno una parola.

modifiche al testo: per modificare una porzione di testo basta spostare il cursore (usando il mouse) nella zona nella quale si vuole fare la modifica e quindi aggiungere il testo (digitandolo da tastiera) o modificare quello esistente. Per cancellare del testo basta selezionare il testo ce si vuole cancellare con il cursore (trascinando il mouse mentre si tiene premuto il pulsante sinistro del medesimo) e quindi usare il tasto *Delete* (Cancella) o *Backspace*. La parte selezionata del testo può essere copiata in un'altra parte del documento o in un altro documento usando le funzionlità di *Cut* (Taglia) e *Paste* (Incolla) disponibili dal menu *Edit* (Modifica).

Uso di un editore di testi: controlli e visualizzazione

controllo ortografico:
: se questa funzionalità è attiva, le parole inserite nel testo sono confrontate con quelle presenti in un dizionario memorizzato nel computer. La lingua di riferimento ovviamente deve essere selezionata in precedenza, e questo può essere fatto dal menu *Tools* (Strumenti). Le parole che non sono presenti nel dizionario sono segnalate come possibili errori (di solito mediante una sottolineatura in un colore diverso da quello del testo). è disponibile anche una funzionalità di correzione automatica che permette di ottenere suggerimenti su possibili correzioni della parola errata ed anche di realizzare la correzione in modo automatico. Inoltre è disponibile un thesaurus che permette di ottenere sinonimi e contrari.

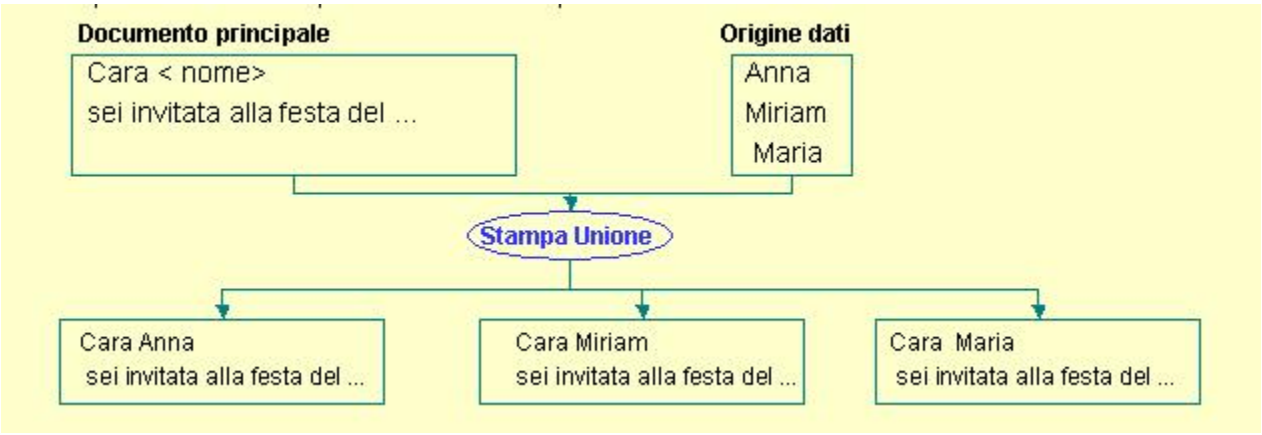
ricerca e sostituzione :
: tramite gli opportuni comandi *Search* (Trova) e *Replace* (Sostituisci) del menù *Edit* (Modifica) è possibile cercare le occorenze di una sequenza di caratteri all'interno del testo ed eventualmente sostituirla con un'altra sequenza di caratteri. I comandi Trova e Sostituisci possono essere usati anche per cambiare lo stile.

comandi di visualizzazione:
: il menù *View* (visualizza) permette di visualizzare il documento secondo varie modalità che permettono di evidenziare la struttura del documento (sostanzialmente mostrando i titoli ai vari livelli) oppure di visualizzare il documento normale secondo vari formati di presentazione.

Uso di un editore di testi: stampa unione

comandi relativi al file:
: come tutti i programmi che operano su dei file, anche gli editori di testi permettono (dal menu file) di aprire, chiudere, salvare e stampare il file contente il documento. Di particolare interesse è la stampa unione che permette di stampare dati provenienti da due file. Più precisamente, si ha un documento principale che contiene la struttura del testo da stampare (ad esempio un modello di lettera) ed un documento (detto origine dei dati) contente i dati da includere nel documento principale (ad esempio i dati dei vari destinatari della

lettera). Mediante opportuni campi unione, di solito indicati racchiudendo del testo fra < >, si indicano nel documento principale i tipi di dati che devono essere prelevati dall'origine dei dati ed i punti dove tali dati devono essere inseriti. Al momento della stampa unione verranno quindi generate (e stampate) varie copie del documento principale, ottenute istanziandolo con tutti i dati del file origine dei dati nel modo specificato dai campi unione. Ad esempio



Uso di un editore di testi: formattazione

Vi sono vari comandi che permettono di definire il formato di un documento. Oltre ai possibili formati sui caratteri, già visti, ricordiamo qui le seguenti funzioni, accessibili dal menu Formato:

margini:
: si possono definire i margini, ovvero gli spazi bianche che circondano il testo usando gli opportuni comandi sul righello;

numeri di pagina:
: si possono definire secondo vari formati e sono inseriti automaticamente dal programma;

elenchi:
: si possono inserire elenchi numerati, puntati etc. quali quello contenuto in questa pagina;

tabelle:
: (vedi più avanti);

stili:
: è possibile memorizzare tutte le informazioni relative alla formattazione di un documento e creare così uno stile (al quale sarà dato un nome). Tutte le volte che si vorranno usare tali formattazioni per un nuovo documento sarà sufficiente usare lo stile memorizzato in precedenza (rintracciabile dal menu stile, nel menu formato, usando il nome dello stile scelto).

Uso di un editore di testi: immagini, tabelle

Di solito gli editori di testi permettono anche di inserire immagini, grafici e tabelle. Ad esempio, in *Word* è possibile:

- inserire un'immagine selezionando Immagine nel menu Inserisci. Si ottiene così un menu a tendina dal quale si possono selezionare i vari tipi di immagine che si possono usare. Ad esempio, l'immagine può essere contenuta in una file e copiata (oppure collegata), oppure l'immagine può essere disegnata usano delle forme predefinite (accessibili selezionando Forme). Notare che quando si usano le forme per realizzare delle figure convesse (cerchi, ellissi, rettangoli ecc) se si vuole inserire del testo entro il perimetro della figura, il colore della medesima deve essere impostato a nessun colore. Per fare questo basta cliccare due volte sulla figura stessa e quindi usare il menu che appare.
- Inserire una Tabella come, ad esempio, la seguente

Nome	Cognome	Via	Città
Mario	Rossi	Vivaldi	Roma
Anna	Bianchi	Corelli	Firenze

Per inserire una tabelle si usa il comando Inserisci Tabella del menu Tabella. Il formato della tabella può essere scelto usando il comando Formattazione Automatica fra vari formati predefiniti. è possibile cambiare le dimensioni usando il comando Altezza e larghezza Celle, e si possono fare operazioni sugli elementi delle varie tabelle (ad esempio, ordinamenti per i valori di un certo campo).

Oggetti OLE

Con i moderni editori di testo è possibile inserire nel documento non solo testo ma anche grafici, fogli di lavoro, immagini, video e animazioni, creando così documenti composti. Questo è possibile usando la tecnologia **OLE** (*Object Linking and Embedding*) o un altro sistema simile chiamato OpenDoc.

Un oggetto OLE è un generico oggetto (testo, immagine, informazione multimediale) che, tramite OLE viene sempre gestito dall'applicazione che lo ha creato: ovvero, quando viene selezionato un oggetto OLE in un documento viene invocata l'applicazione che ha creato l'oggetto OLE e che, in generale, può esser diversa dall'applicazione in uso per elaborare il documento. Ad esempio, se abbiamo creato un grafico con uno sreadsheet e lo abbiamo inserito come oggetto OLE in un file che stiamo editando con un editore di testi, facendo doppio clic sul grafico attiviamo lo *spreadsheet*. Ci sono due modalità per inserire e gestire un oggetto OLE

Incorporamento (*Embedding*).
: I dati dell'oggetto da inserire vengono copiati nel documento composto che stiamo creando. Facendo doppio clic sull'oggetto incorporato si attivano i menu dell'applicazione che lo ha creato.

Collegamento (*Linking*).
: L'oggetto da inserire non viene copiato e nel documento composto viene inserito solo un collegamento a tale oggetto. Facendo doppio clic sull'oggetto collegato si attiva l'applicazione che lo ha creato (sull'oggetto collegato).

Rappresentazione interna dei documenti

Ogni documento elettronico ha una o più rappresentazioni interne (in termini di byte, o caratteri) ed una o più rappresentazioni esterne (in termini di pixel). Internamente, i caratteri che costituiscono un documento, una volta codificati opportunamente, sono memorizzate in struttrure logiche dette **file** che, come abbiamo visto, sono gestire dal sistema operativo.

Per rappresentare i caratteri si usano i seguenti codici:

Il codice **ASCII** ,
: utilizza 7 bit per i caratteri, permettendo così di rappresentare 128 valori diversi (27 = 128). L'ottavo bit viene utilizzato come codice di controllo. Esistono anche codici ASCII a 8 bit e vari produttori hanno usato estensioni diverse (questo spiega i problemi di conversione tra file di sistemi diversi, ad esempio fra *Windows* e *OS-Macintosh*).

Il codice EBCDIC
: utilizza 8 bit e permette quindi di rappresentare 28= 256 valori diversi.

Il codice Unicode
: utilizza 16 bit. I possibili valori non sono ancora stati tutti assegnati (34168 su 655236). Unicode raddoppia lo spazio occupato da testi scritti in caratteri latini, ma permette la rappresentazione dei caratteri di molti alfabeti non latini (arabo, ebraico, cinese, ecc) e di caratteri speciali (matematici, tecnici, elementi di disegno ecc).

Il codice ISO Latin 1
è un'estensione a 8 bit del codice ASCII ed è molto diffuso negli usi di rete. ISO Latin 1 corrisponde ai caratteri Unicode i cui primi 8 bit sono tutti zero.

Formati proprietari e non

Il codice **ASCII** si può usare per rappresentare qualsiasi struttura dati o programma, usando le particolari sequenze di caratteri tipiche di quella struttura. Tuttavia molte applicazioni usando un formato privato e proprietario, di solito binario, dei documenti che creano. Anche se si adducono spesso ragioni di efficienza per tali formati in realtà la vera ragione è commerciale, dato che il formato proprietario impedisce che tale formato sia usabile anche da altre applicazioni.

Esempio:
Il formato .doc, prodotto con il programma di scrittura *Microsoft Word*, è un formato proprietario. Se guardiamo un documento .doc come file di testo (ad esempio usando un editor puramente testuale quale il Notepad disponibile in *Windows* solitamente dal menu accessori nel menu ottenibile cliccando su Start), vediamo una sequenza di caratteri illeggibile. Riusciamo a leggere un file .doc usando il programma *Word*, perchè tale programma interpreta correttamente le varie sequenze d caratteri. Per ragioni di interoperabilità e portabilità dei documenti tuttavia si stanno affermando formati di interscambio.
Esempio:
Word ed altri programmi di scrittura sono capaci di produrre file in un formato chiamato RTF (Rich Text Format) che è leggibile anche come file di testo e che è comprensibile a varie applicazioni.

Il linguaggio HTML, nato nel contesto del *World Wide Web*, si sta affermando come un importante sistema di rappresentazione di documenti. A causa di alcuni suoi limiti è stato introdotto un nuovo linguaggio per documenti chiamato XML. HTML e XML definiscono formati "aperti", non proprietari e leggibili.

Alcuni formati

Esistono molti formati diversi di file di dati, quasi sempre distinguibili dall'estensione .xxx che segue il nome del file
Esempi di estensione:

.txt o .asc
file in formato ASCII
.doc
file in formato MS *Word*
.rtf
file in *Rich Text Format*
.htm o .html
file in formato HTML
.ps
file in formato *PostScript*
.eps
file in formato *Encapsulated PostScript*
.pdf
Portable Document Format (Acrobat)
.zip
file compresso (usare *unzip* per decomprimere)
.gz
file compresso (gunzip per decomprimere)
.z o .Z
file compresso (uncompress per decomprimere)
.tar
archivio realizzato col programma TAR

.tar.z
: archivio realizzato con TAR e poi compresso

.tgz
: archivio realizzato con TAR e poi compresso

.uu
: file ASCII codificato con uuencode

.gif
: file contenente immagine in formato GIF

.jpg
: file contenente immagine in formato JPEG

.pic
: file contenente immagine in formato PIC

.wav
: file contenente audio

.au
: file contenente audio

.mpg
: file contenente filmato MPEG

.mov
: file contenente filmato *QuickTime*

Compressione dei file

Spesso prima di spedire o trasferire dei file, oppure prima di memorizzarli su disco, se ne effettua la **compressione** . Tale tecnica permette di mantenere il contenuto informativo dei dati riducendo però lo spazio di memoria occupato. è possibile la compressione perché i file normalmente contengono delle ridondanze che possono essere eliminate usando opportuni algoritmi. Per file prodotti da programmi di videoscrittura si possono ottenere riduzioni dello spazio occupato fino al 50%, mentre per file di grafica bitmap si può arrivare al 90%. Ovviamente esiste un limite alla possibilità di compressione, oltre il quale si perde il contenuto informativo.

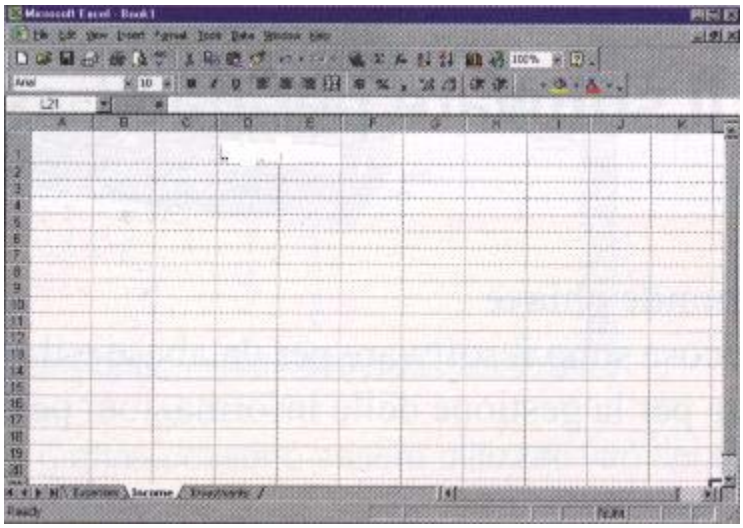
Alcuni formati sono già compressi per cui sono inutili ulteriori compressioni. Per esempio è inutile comprimere i file pdf, gif o mp3.

Per *Windows* uno dei programmi di compressione *shareware* più diffusi si chiama Winzip (è possibile reperire tale programma in rete). Per Unix il metodo più comune per produrre archivi compressi è usare i programmi TAR e GZIP.

Ovviamente, prima di usare un file compresso questo dovrà essere decompresso: a tale scopo si usano gli stessi programmi usati per la compressione con modalità diverse.

I fogli elettronici

I **fogli elettronici** (detti anche *spreadsheet*) sono strumenti che permettono il trattamento e l'analisi di dati numerici mediante tabelle (o griglie) nelle quali è possibile operare sulle singole caselle, sulle colone, sulle righe o su più caselle collegate fra loro. Si possono effettuare varie operazioni numeriche, dalle più semplici (quali la somma dei valori contenuti in una colonna) a quelle più complesse quali si hanno in ambito statistico e finanziario. I folgi elettronici permettono inoltre di definire grafici (secondo vari formati) in modo abbastanza immediato. I più diffusi fogli elettronici sono Microsoft Excel, Corel Quattro Pro e Lotus 1-2-3. Un foglio elettronico si presenta come nella figura seguente:



Immissione dei dati nei fogli elettronici

I dati presenti in un **foglio elettronico** sono molto più strutturati di quelli presenti in un elaboratore di testi: l'area di lavoro di un foglio elettronico infatti è organizzata in colonne e righe. Ogni colonna è identificata da un nome riportato nella parte superiore (A, B, C ecc) mentre le righe sono identificate da numeri presenti nel lato sinistro (1,2,3 ecc). è possibile inserire opportune etichette all'inizio delle righe e delle colonne, così come in altri punti del foglio, semplicemente digitandole da tastiera.

Il punto di intersezione fra una riga e una colonna è detto **cella**, e la sua posizione è ottenuta dalla colonna e dalla riga che la individuano (ad esempio A4, B7 ecc). Un intervallo è costituito da un gruppo di celle adiacenti, per esempio quelle celle comprese fra A3 e D6

È possibile **immettere dei valori** in una cella semplicemente posizionando il cursore sulla cella e quindi digitando il valore. è possibile anche immettere contemporaneamente valori in gruppi di celle mediante operazioni di taglia e incolla, oppure trasferendoli per mezzo di un programma da un altro file. I valori possono essere molte cose: valori monetari, percentuali, gradi di temperatura, misure di angoli, eccetera.

Cosa si può fare con un foglio elettronico

Il motivo principale dell'utilità di un foglio elettronico consiste nella possibilità di immettere nella caselle formule, funzioni e nella possibilità di ricalcolare con estrema facilità nuovi valori.

Ad esempio, supponendo di avere nella colonna A, nelle righe da 1 a 10, dei valori numerici possiamo inserire nella cella A11 una formula del tipo = SOMMA(A1:A10). Questa formula ci dice che il valore della casella 11 è costituito dalla somma dei valori presenti nella prime 10 righe della colonna A. Se cambiamo uno di questi valori il valore della somma in A11 viene aggiornato automaticamente.

Nelle formule possiamo usare delle funzioni predefinite quali ad esempio la somma, la media di un insieme di valori, l'ammortamento, eccetera. Sono possibili molte funzioni di varo tipo: finanziarie, matematiche e trigonometriche, statistiche, eccetera, tutte selezionabili dal menù Inserisci.

La caratteristica essenziale di un foglio elettronico è la possibilità di usare all'interno di una cella riferimenti ad altre celle (come ad esempio A1 e A10 nella cella A11, nell'esempio precedente). Questo permette di poter ricalcolare valori in modo immediato: basta immettere un nuovo valore in un operando per avere automaticamente aggiornato il risultato in tutte le formule che lo usano. Da questa facilità di ricalcolo deriva la possibilità di effettuare la cosiddetta analisi condizionale, ovvero si possono esaminare più scenari diversi semplicemente immettendo valori diversi e quindi ricalcolando istantaneamente i risultati che ci interessano.

È possibile anche inserire nei fogli elettronici vari tipi di grafici quali istogrammi, grafici a barre, a torta, eccetera.

Introduzione alle basi di dati

Prof. Maurizio Gabbrielli
3.2 Software applicativo

Introduzione

Le basi di dati ed i relativi programmi di gestione (detti **DBMS**) in prima approssimazione possono essere considerati come sistemi che permettono la gestione di grandi quantità di informazioni per raggiungere gli scopi di una qualche organizzazione (azienda, istituzione etc.), dove per gestione intendiamo le seguenti attività :

- Raccolta, acquisizione;
- Archiviazione, conservazione;
- Elaborazione, trasformazione, produzione;
- Distribuzione, comunicazione, scambio.

Queste attività di gestione delle informazioni, così come le informazioni stesse, mentre a livello umano possono essere gestite usando principalmente idee informali e linguaggio naturale, a livello informatico devono essere opportunamente formalizzate e codificate. Si noti che tali formalizzazioni sono anteriori alla realizzazione dei sistemi informatici. Nei sistemi informatici le informazioni vengono rappresentate in modo strutturato ed essenziale (mediante i dati).

Sistemi informativi, sistemi informatici e basi di dati.

È utile chiarire la distinzione fra i seguenti termini:

Sistema informativo :
Componente di una organizzazione che gestisce (acquisisce, elabora, conserva, produce) le informazioni interessanti per l'organizzazione in questione. Si noti che, in principio, l'esistenza di un sistema informativo prescinde da qualsiasi processo di automazione. In effetti istituzioni vecchie di secoli (quali ad esempio le banche) usavano sistemi informativi prima che esistesse l'energia elettrica. Anche alcune formalizzazioni usate per il trattamento sistematico dei dati sono anteriori all'informatica (ad esempio, le schede degli archivi).

Sistema informatico :
Parte automatizzata del sistema informativo, ovvero quella parte che gestisce le informazioni usando l'informatica.

Nei sistemi informatici, le informazioni sono rappresentate in termini di dati. È per questo che si usa la terminologia base di dati.

Basi di dati

Base di dati :
Insieme organizzato di dati utilizzati per il supporto allo svolgimento delle attività di una organizzazione (azienda, ufficio ecc). In senso più tecnico, una base di dati identifica l'insieme dei dati gestiti da un programma DBMS (*Database Management System*).

DBMS:
Programma che gestisce grandi insiemi di dati, in modo persistente e condiviso, in modo tale da garantire privatezza, affidabilità ed efficienza.

Vediamo i vari concetti introdotti nella definizione di DBMS:

grandi:
molte aziende hanno oramai basi di dati dell'ordine del terabyte (1 terabyte = mille miliardi di byte).
persistente:
i dati che si gestiscono hanno un tempo di vita maggiore di quello dei programmi che li gestiscono, deve quindi essere garantita la loro persistenza nel tempo. Si noti che in molti i casi i dati sono una risorsa strategica per un'organizzazione e la loro perdita può costituire un danno irreparabile.
privatezza:
visto che vengono gestiti grandi quantità di dati, di solito ci sono molti utenti che hanno accesso alla base di dati. Ogni utente deve poter accedere solo ai dati per i quali ha una specifica autorizzazione.
affidabilità:
i DBMS devono permettere di gestire eventuali malfunzionamenti (hardware o software) mediante opportune funzionalità di salvataggio (*backup*) e ripristino (*recovery*) dei dati.
efficienza:
l'accesso alla base di dati deve essere possibile in tempi ragionevoli, così come devono essere ragionevoli le risorse di calcolo e di memoria necessarie (in generale comunque i DBMS sono programmi abbastanza pesanti e richiedono adeguate risorse).

Fra i DBMS disponibili sul mercato ricordiamo *Access*, DB2, Oracle, Informix .

Cosa si può fare con una base di dati

Per l'utente finale l'uso più comune di una base di dati è l'esecuzione di opportune interrogazioni (*query*) predefinite. Esempi tipici di interrogazioni che molti di noi hanno già usato sono l'accesso ad un catalogo elettronico di una biblioteca, la consultazione (on-line oppure sulle apposite macchine nelle stazioni) degli orari dei treni, la prenotazione di un volo aereo, la consultazione delle informazioni relative all'andamento dei titoli in borsa ecc.

Queste interrogazioni possono essere molto semplici (ad esempio, dimmi tutti i libri di Calvino che sono presenti in biblioteca) ma possono essere anche molto complesse. Ad esempio, si potrebbe voler sapere quali sono i libri di Calvino che nel titolo hanno una parola che compare anche nel titolo di un libro di Pavese e che inoltre sono stati presi in prestito almeno 10 volte. Oppure, in ambito finanziario, si potrebbe voler conoscere i titoli che hanno perso almeno il 50% del valore negli ultimi due anni, che hanno un rapporto prezzo/utili minore di quello medio dei titoli dello stesso settore e che hanno un volume superiore a quello medio dei titoli di aziende con la stessa capitalizzazione.

Questo tipo di interrogazioni possono essere espresse usando opportuni linguaggi (*query languages*) che differiscono a seconda del modello dei dati adottato (vedi più avanti).
La possibilità di usare tali linguaggi, che in sostanza sono dei linguaggi di programmazione specializzati, costituisce la maggiore differenza fra le basi di dati ed i fogli elettronici, dato che questi ultimi invece permettono di esprimere delle interrogazioni molto più semplici (usando le formule e le funzioni).
Altra differenza rilevante fra DBMS e fogli elettronici è nella quantità dei dati che si possono trattare, molto inferiore nel caso dei fogli elettronici.

Le transazioni

Le basi di dati, date le loro dimensioni, di solito sono utilizzate da più utenti e da più programmi applicativi, permettendo così un uso condiviso dei dati. In tali circostanze possono verificarsi problemi di ridondanza (stessi dati ripetuti inutilmente) e incoerenza (visione diversa dello stesso dato). Per ovviare a questi possibili problemi si usa il cataolgo e il meccanismo basato sulle transazioni:

Catalogo.
I dati di un DBMS sono organizzati in un opportuni file che il DBMS gestisce usando anche le funzionalità del sistema operativo. Tuttavia, a differenza di quanto avviene con un normale programma che usi dei file, nei DBMS esiste una porzione della base di dati, detta catalogo o dizionario, che contiene una descrizione centralizzata dei dati e che può essere utilizzata dalle varie applicazioni. Questo permette di ridurre

i problemi di incoerenza e ridondanza dei dati.

Transazione.

Una transazione è un insieme di operazioni sulla base di dati che devono essere considerate in modo indivisibile (o "atomico"); ovvero, o tutte le operazioni dell'insieme sono eseguite correttamente oppure nessuna.

L'effetto di una transazione deve essere corretto anche in presenza di altre transazioni concorrenti eseguite nello stesso intervallo di tempo. Questo si può garantire, ad esempio, facendo si che l'effetto di due transazioni concorrenti sia equivalente alla loro esecuzione sequenziale.

Infine gli effetti di una transazione conclusasi positivamente devono essere definitivi, ovvero devono essere garantiti anche in presenza di guasti ed di esecuzione concorrente

Come esempi di transazioni si pensi alle operazioni effettuate sulla base di dati di una banca: le operazioni di addebitamento dell'importo prelevato da un Bancomat e l'erogazione del denaro devono costituire una transazione. Analogamente, nello spostamento di fondi fra due conti, le operazioni di accreditamento su un conto e addebitamento su un altro devono costituire una transazione.

Il modello dei dati

I programmi che realizzano un DBMS fanno riferimento ad una struttura logica, detta modello dei dati, che descrive i dati ad a livello piùastratto di quello che invece corrisponde alla loro rappresentazione fisica. Questo permette una maggiore indipendenza delle applicazioni dai dettagli implementativi. Ad esempio, se si passa da un file-system ad un altro, i programmi che realizzano le applicazioni non necessariamente devono essere cambiati (saranno cambiate le componenti, di livello più basso, che accedono direttamente ai dati).

Modello dei dati.

Un modello dei dati è costituito da un insieme di costrutti utilizzati per organizzare i dati e per descriverne l'evoluzione. La parte fondamentale di un modello dei dati è costituita da opportuni meccanismi di strutturazione dei dati.

Si distinguono tre tipi di modelli dei dati

modelli concettuali:

sono i modelli più astratti, usati per rappresentare i dati e, più in generale, i concetti esistenti nel mondo reale, secondo un formalismo che è completamente indipendente da ogni sistema informatico. Il modello concettuale più diffuso è il modello E-R (Entità -Relazioni). Sono usati nella prima fase della progettazione di una base di dati.

modelli logici:

sono i modelli adottati nei DBMS per definire l'organizzazione dei dati utilizzati nei programmi, in modo indipendente dalle strutture fisiche di memorizzazione. I modelli principali sono i seguenti:

- relazionale
- a oggetti
- reticolare,
- gerarchico, a oggetti

I modelli più importanti oggi sono quello relazionale e quello a oggetti

Schema e istanza

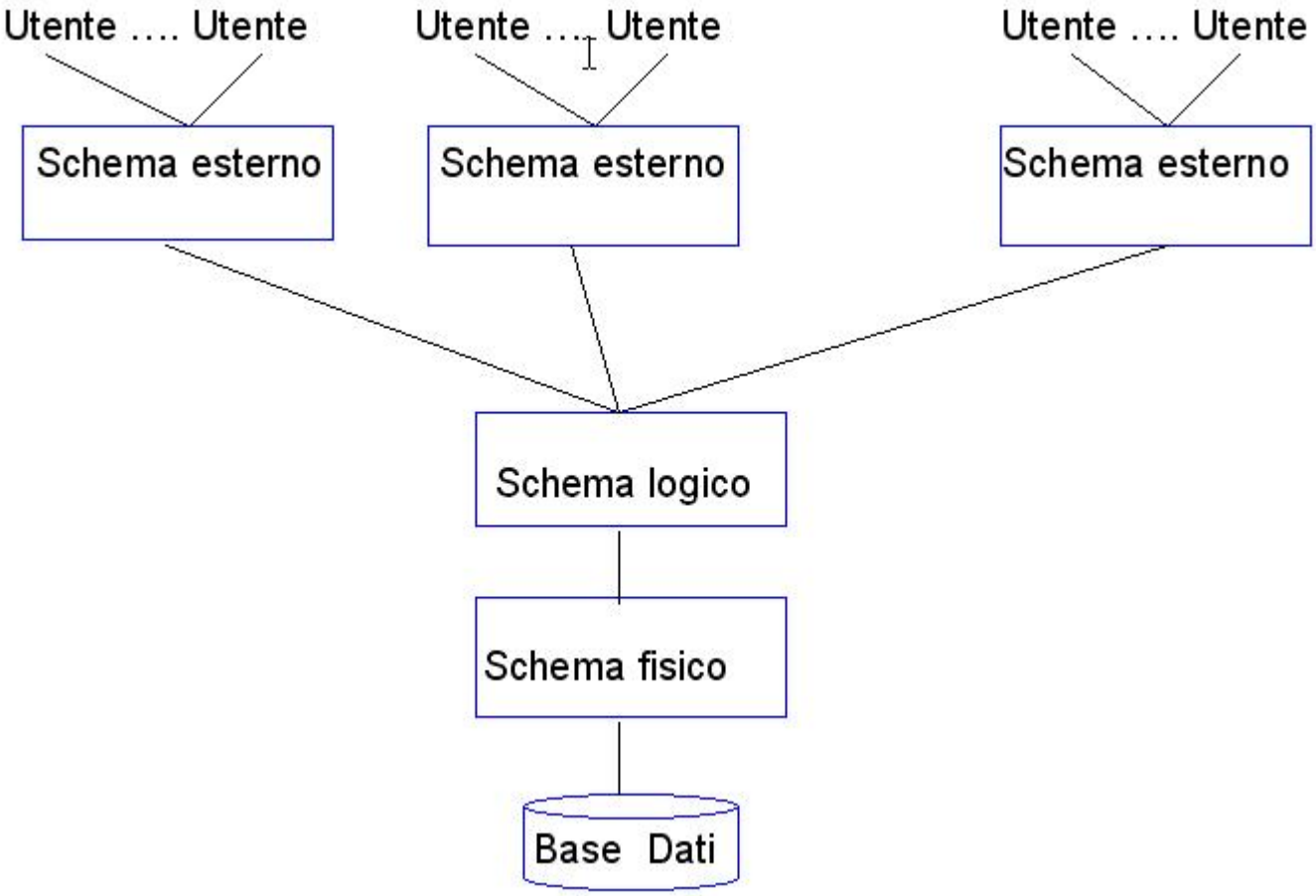
Lo **schema** di una base di dati ne descrive la struttura invariante nel tempo, ovvero l'aspetto intensionale. Distinguiamo

Schema logico:
descrizione della base di dati nel modello logico di riferimento.
Schema interno (o fisico):
descrizione delle strutture di memorizzazione usate per memorizzare lo schema logico.
Schema esterno:
descrizione, allo stesso livello di astrazione dello schema logico, di una parte della base di dati (ad esempio, la parte accessibile ad una classe di utenti)

L'istanza di una base di dati invece descrive i valori contenuti nella base di dati, che possono variare nel tempo (aspetto estensionale).

Si ha dunque lo schema nella pagina successiva

Architettura standard di una base di dati



Indipendenza dei dati

Con l'architettura vista in precedenza, che è uno standard ANSI/SPARC, l'accesso da parte degli utenti avviene solo a livello più esterno, ovvero attraverso lo schema esterno.

Questa architettura garantisce l'indipendenza dei dati in due sensi:

indipendenza fisica.
Il livello logico e quello esterno sono indipendenti da quello fisico, ovvero l'accesso ai dati avviene in

modo indipendente dai dettagli di allocazione fisica e memorizzazion. Questo è molto importante in quanto, ad esempio, permette di modificare le modalità di organizzazione dei supporti di memorizzazione senza che questo influisca sui programmi delle applicazioni che accedono ai dati.

indipendenza logica.

Il livello esterno è indipendente da quello logico per cui modifiche allo schema esterno (ad esempio, per aggiungere classi di utenti o per limitare i diritti di alcuni utenti) non comportano modifiche dello schema logico.

Linguaggi per basi di dati

Nell'ambito dei DBMS si distinguono due tipi di linguaggi (anche se, nella pratica, entrambi i tipi sono presenti nello stesso formalismo).

Data Manipulation Language (DML):

linguaggi per la manipolazione di dati, ovvero per esprimere interrogazioni e aggiornamenti di (istanze di) basi di dati

Data Definition Language (DDL):

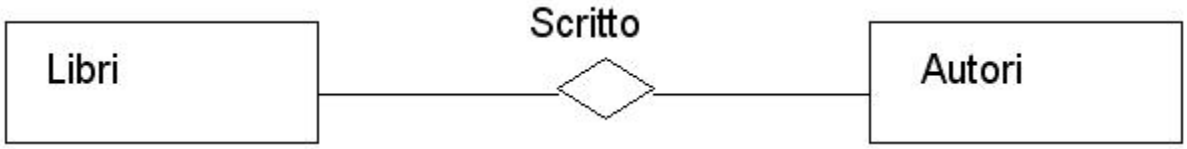
linguaggi per la per la definizione degli aspetti strutturali di una base di dati, ovvero per la definizione di schemi logici, schemi esterni, schemi fisici) ecc

I linguaggi usati dai DBMS fanno riferimento ad un particolare modello dei dati. Uno dei più diffusi è **SQL** (*Structured Query Language*), un linguaggio basato sul modello relazionale che è disponibile in varie versioni, con interfacce sia grafiche che testuali, e che può essere a sè stante, oppure immerso in linguaggio di programmazione ospite (generico come Pascal, Java e C oppure specifico per alcune applicazioni, come ad esempio per realizzare grafici).

Il modello relazionale dei dati: introduzione

Il modello gerarchico e quello reticolare utilizzano riferimenti espliciti (puntatori) fra strutture di memorizzazione (*record*) per esprimere le relazioni e le dipendenze esistenti fra i dati. Il modello relazionale è invece basato unicamente sui valori, ovvero i riferimenti fra dati diversi sono realizzati usando valori.

Ad esempio, supponiamo di avere la seguente situazione, descritta in termini concettuali usando il modello E-R



dove abbiamo l'entità Libro (con gli opportuni attributi quali, titolo, editore ecc.), l'entità Autore (con attributi nome, cognome, anno di nascita ecc.) e l'associazione Scritto che collega ogni libro ai suoi autori (e quindi ogni autore ai libri che ha scritto).
Questa associazione, nel modello gerarchico dei dati è rappresentata esplicitamente, mediante un riferimento (o puntatore) che collega il record contenente le informazioni su un singolo libro al record contenente le informazioni sull'autore del libro. Avremo quindi una situazione di questo tipo:

dove il tratto verticale indica il riferimento esplicito.

Nel modello relazionale invece tale riferimento è espresso implicitamente in termini di valori. Ad esempio, nel record Libro potremmo aggiungere un attributo autore dove sia memorizzato un valore che permetta di individuare univocamente un autore, e quindi di accedere alle informazioni sull'autore del libro, che saranno contenute in un altro record. La situazione è dunque la seguente

e non abbiamo più riferimenti espliciti, ma solo riferimenti impliciti dati dai valori (nel nostro caso, il cognome Hesse).

N.B. In alcuni DBMS relazionali che permettono un'interfaccia grafica (ad esempio ***Access***) si possono tracciare degli archi fra le varie tabelle. Tali archi non servono per rappresentare dei riferimenti espliciti fra i dati ma solo per abbreviare alcune operazioni (di *Join*).

Il modello relazionale dei dati: la nozione di relazione

Il modello relazionale è stato proposto da E. F. Codd nel 1970 per favorire l'indipendenza dei dati, ma è disponibile in DBMS reali nel 1981 e si basa sul concetto matematico di relazione (da cui il nome), leggermente modificato .

Dato che le relazioni possono essere rappresentate per mezzo di tabelle, nel linguaggio delle basi di dati spesso si usano i termini relazione e tabella come sinonimi.

In senso strettamente matematico una relazione su n insiemi anche non distinti D1 ,..., Dn è definita come un qualsiasi sottoinsieme del prodotto cartesiano D1 ×...× Dn .

Il prodotto Cartesiano D1 ×...× Dn è l'insieme di tutte le n-uple (d1 ,..., dn) tali che d1 è un elemento in D1 , d2 è

un elemento in D2 ,..., Dn è un elemento in Dn.

Esempio. Se A è l'insieme dei cognomi, B è l'insieme dei numeri di telefono, una il seguente insieme di coppie
{ (Rossi, 06 12134), (Bianchi, 02 12115) }
è una relazione su A,B. Dal punto di vista strettamente matematico la coppia
(Rossi, 06 12134) è diversa dalla coppia (06 12134, Rossi)
dato che l'ordine dei componenti è importante. Difatti è tale ordine che ci permette di identificare il tipo dei valori (sappiamo che Rossi è un cognome, perchè appare nella prima posizione).

Per poter ottenere una rappresentazione indipendente dalla posizione possiamo attribuire ad ogni dominio un nome che ne precisa il ruolo. Abbiamo quindi una rappresentazione di tipo tabellare che nel caso dell'esempio precedente è

Cognome	Numero Telefono
Rossi	06 12134
Bianchi	02 12115

Si noti che tale tabella contiene esattamente la stessa informazione di

Numero Telefono	Cognome
06 12134	Rossi
02 12115	Bianchi

Relazioni e Tabelle

La definizione di **relazione** che si usa nel modello relazionale è dunque derivata dalla nozione matematica, introducendo dei nomi di dominio per ottenere una notazione non posizionale. Dal punto di vista pratico tale nozione coincide con quella di **tabella** se nella definizione della tabella sono rispettate le seguenti condizioni:

- ogni colonna della tabella ha una intestazione (detta attributo) e le intestazioni delle colonne sono diverse fra di loro;
- i valori di ogni colonna (ovvero di ogni attributo) sono omogenei fra di loro;
- le righe sono diverse fra loro;
- l'ordinamento tra le righe è irrilevante;
- l'ordinamento tra le colonne è irrilevante.

Come detto in precedenza, il modello relazionale è basato sui valori: i riferimenti fra varie tabelle diverse (quali ad esempio autori e libri, dell'esempio precedente, sono realizzati mediante valori). Questo ha i seguenti vantaggi:

- indipendenza dalle strutture fisiche di memorizzazione;
- si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione;
- i dati sono portabili più facilmente da un sistema ad un altro;
- le associazioni fra dati sono rappresentate in modo non direzionale .

Il modello relazionale: Definizioni

Schema di relazione:
uno schema di relazione è costituito da un nome R con un insieme di attributi A1 , ..., An e si indica con R(A1 ,..., An). Dal punto di vista della tabelle uno schema di relazione è costituito al nome della tabella e dal nome delle singole colonne.
Schema di base di dati:

uno schema di base di dati è insieme di schemi di relazione R = {R1 (X1), ..., Rk (Xk)}. Dal punto di vista della tabelle, uno schema della base di dati è costituito dagli schemi di tutte le tabelle presenti.

Ennupla :

una ennupla su un insieme di attributi X èuna funzione che associa a ciascun attributo A in X un valore del dominio di A. Dal punto di vista della tabelle, una ennupla è costituita da una riga della tabella (diversa dalla prima riga che contiene le intestazioni delle colonne).

Istanza di relazione :

una istanza di una relazione su uno schema R(X) è costituita da un insieme di ennuple su X. Dal punto di vista della tabelle, una istanza di relazione è costituita dall'insieme delle righe di una tabella.

Istanza di base di dati:

una istanza di una base di dati su uno schema R= {R1 (X1), ..., Rn (Xn)} è costituita da un inisieme di istanze di relazioni r = {r1 ,..., rn}, dove ri è una relazione su Ri . Dal punto di vista della tabelle, una istanza di base di dati è costituita da un insieme di istanze di tabelle, con una istanza per ogni schema di tabella presente nello schema della base di dati.

Esempio

Consideriamo il caso (molto semplificato) della gestione di una biblioteca, dove si vogliano memorizzare delle informazioni sui libri, sugli utenti e sui prestiti. Qui e nel seguito useremo una notazione tabellare dove il nome della tabella è posto immediatamente sopra la medesima. Lo schema della base di dati è costituito dai seguenti tre schemi

Libro	Titolo	Autore	Codice ISBN

Utente	Nome	Indirizzo	Codice Fiscale

Prestito	Libro	Utente	Data Prestito

Uno specifica istanza della base di dati sarà ottenuta inserendo dei valori nelle righe delle celle. Quella qui sotto è un esempio di istanza (i codici ISBN e fiscali sono fittizi).

Libro

Titolo	Autore	Codice ISBN
Siddharta	Hesse	ISBN 1456
La luna e i falò	Pavese	ISBN 1347
Macbeth	*Shakespeare*	ISBN 1237

Utente

Nome	Indirizzo	Codice Fiscale
Rossi	Via Vivaldi	MRZRSSI 134
Bianchi	Via Lulli	ANMBCI 125
Smith	Via Corelli	JOHSMH 167

Libro	Utente	Data Prestito
ISBN 1456	ANMBCI 125	12 Dicembre 2002
ISBN 1237	MRZRSSI 134	3 Gennaio 2003

Esempio

Si noti come nell'esempio precedente, i valori contenuti nella tabella Prestito permettano di identificare univocamente le righe della tabella Libro mediante il codice ISBN e le righe della tabella Utente mediante il codice fiscale. Inoltre i valori contenuti nelle righe della tabella Prestito realizzano le seguenti associazioni fra le righe delle tabelle Libro e Utente

Libro

Titolo	Autore	Codice ISBN
Siddharta	Hesse	ISBN 1456
La luna e i falo'	Pavese	ISBN 1347
Macbeth	Shakespeare	ISBN 1237

Utente

Nome	Indirizzo	Codice Fiscale
Rossi	Via Vivaldi	MRZRSSI 134
Bianchi	Via Lulli	ANMBCI 125
Smith	Via Corelli	JOHSMH 167

Come detto in precedenza, l'uso dei valori è l'unica possibilità che si ha nel modello relazionale per esprimere dei riferimenti.

Chiave

Come visto nell'esempio precedente, spesso è utile poter disporre di un attributo (ad esempio il codice fiscale) che permetta di identificare univocamnente le righe di una tabella. Un tale attributo è una chiave. Con più precisione, possiamo definire una chiave come segue:

- Una **chiave** è un insieme minimale di attributi che permette di identificare univocamente le ennuple di una relazione (ovvero le righe di una tabella).

Nell'esempio precedente, il Codice ISBN è una chiave per la tabella Libro, perchè non ci sono due libri diversi con lo stesso codice ISBN. Se supponiamo che la nostra base di dati non contenga due libri che abbiano stesso titolo e stesso autore, allora anche l'insieme di attributi {Titolo, Autore} costituisce una chiave. L'insieme {Titolo, Autore, Codice ISBN} invece non è una chiave perchè non è minimale, ovvero se togliamo un attributo da tale insieme l'insieme rimasto permette ancora di identificare le righe della tabella (tecnicamente si dice che {Titolo, Autore, Codice ISBN} è una super-chiave).

Per ogni tabella esiste sempre una chiave: infatti dato che le righe di una tabella sono considerate come un insieme, non possono esistere in una tabella due righe diverse che hanno gli stessi valori per tutte le colonne. Quindi l'insieme di tutti gli attributi (di tutte le intestazioni delle colonne cioè) permette di identificare univocamente ogni riga della tabella. Se un tale insieme è minimale allora è una chiave, se non è minimale conterrà un insieme più piccolo che

permette ancora di identificare univocamente ogni riga della tabella. Ripetendo lo stesso ragionamento su tale insieme più piccolo si arriva ad un insieme minimale che costituisce la chiave.

L'esistenza delle chiavi è fondamentale perchè la chiave è l'unico meccanismo che garantisce l'accessibilità a ciascun dato della base di dati e permette di correlare i dati presenti in tabelle diverse (si ricordi che i riferimenti nel modello relazionale sono basati solo sui valori).

Fra tutte le chiavi ne viene scelta una, detta **chiave primaria**: sugli attributi della chiave primaria non sono permessi valori nulli (ovvero tutti i valori devono essere specificati).

I vincoli d'integrità

Per cercare di limitare l'immissione di dati scorretti nella base di dati si usano i vincoli d'integrità . Questi permettono di formulare delle proprietà che devono essere soddisfatte dai dati presenti nella base di dati. I vincoli vengono definiti a livello di schema, ovvero devono essere soddisfatti da tutte le possibili istanze corrette dello schema. Con i vincoli quindi si cerca di modellare delle caratteristiche rilevanti della realtà che vogliamo rappresentare nella base di dati.

Ci sono tre tipi di vincoli di integrità :

- vincoli su valori (o di dominio)
- vincoli di ennupla (o di riga)
- vincoli di integrità referenziale

I vincoli di dominio esprimono delle condizioni sui valori di un singolo attributo (ovvero di una singola colonna) di una singola ennupla (o riga).
Ad esempio, considerando l'attributo Data Prestito nella precedrnte tabella Prestiti possiamo imporre il vincolo che la data sia inferiore (ovvero più vecchia) della data odierna (i DBMS reali di solito offrono particolari rappresentazioni per i valori data, con un opportuno ordinamento sui medesimi)

I vincoli di ennupla esprimono delle condizioni sui valori di una singola ennupla (riga).
Ad esempio, nella precedente tabella Utenti possiamo imporre il vincolo che l'attributo Codice Fiscale contenga una sequenza di caratteri che corrispondono all'attributo Nome secondo una certa regola .

I vincoli di integrità referenziale infine permettono di correlare i dati in tabelle diverse. Più precisamente, un vincolo di integrità referenziale fra gli attributi X di una relazione R1 e un'altra relazione R2 impone ai valori su X in R1 di comparire come valori della chiave primaria di R2. Ad esempio, nelle precedenti tabelle Prestito e Utenti possiamo imporre il vincolo che i valori dell' attributo Utente che compaiono nella tabella Prestito compaiano anche come valori dell'attributo Codice Fiscale nella tabella Utenti. Questo fa sì che non si possano inserire delle informazioni relative al prestito per un utente non registrato.

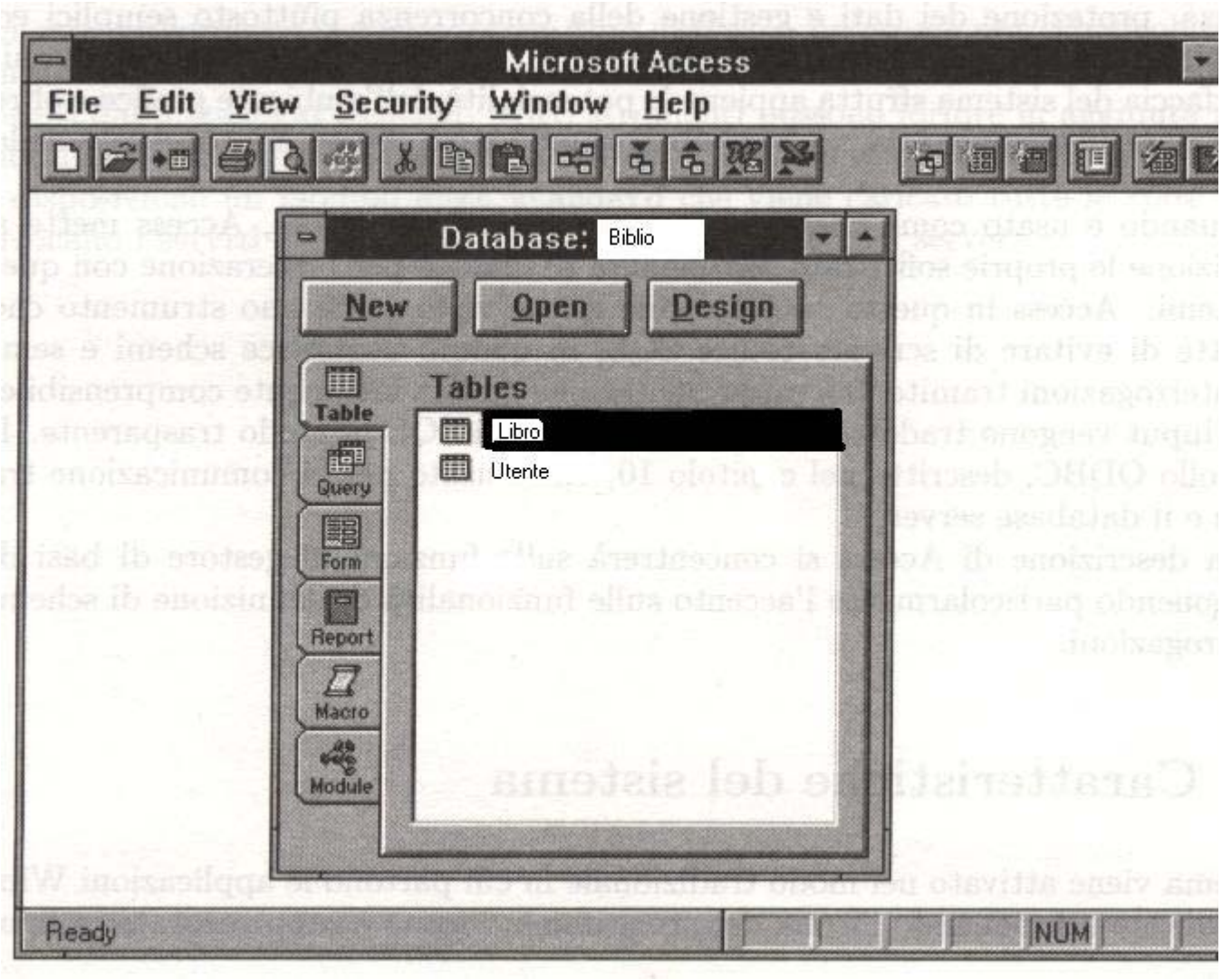
Come si crea una base di dati

Tralasciando la fase (in realtà molto importante) della progettazione concettuale che dovrebbe procedere la fase di effettiva realizzazione di una base di dati, la progettazione di una base di dati relazionale avviene definendo le tabelle specifiche per la particolare applicazione che si vuole realizzare e quindi gli eventuali vincoli di integrità che vogliamo imporre. Questa fase, detta progettazione logica, fa solo riferimento al modello dei dati usato (quello relazionale nel nostro caso) e prescinde dal particolare DBMS usato.

Una volta definite le tabelle si passa alla loro realizzazione fisica usando il DDL (*Data Definition Language*) dello specifico DBMS che useremo. Tutti i moderni DBMS offrono un linguaggio di definizione dei dati con interfaccia grafica, di uso relativamente semplice.

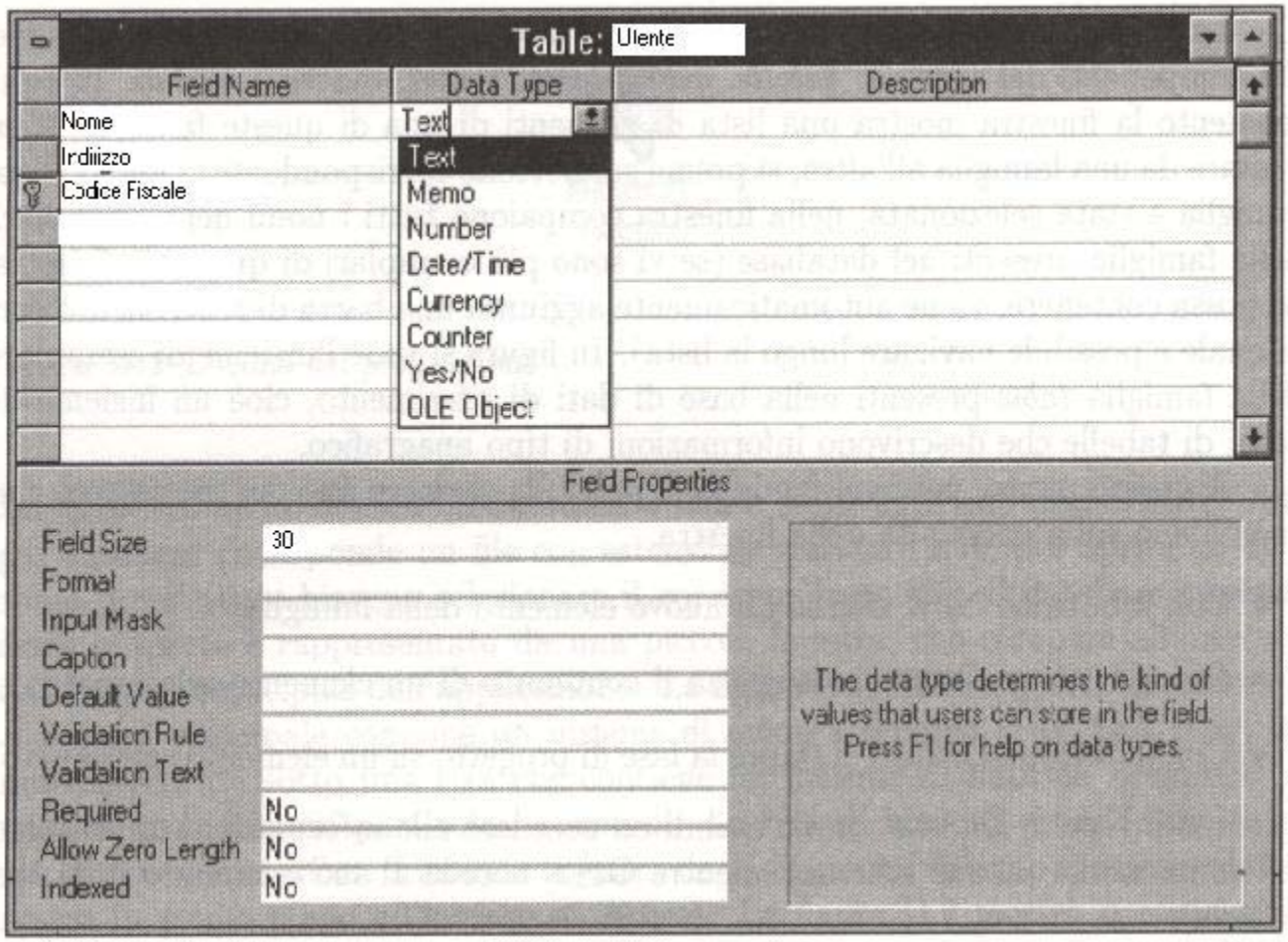
Come si crea una base di dati in Access

In *Access*, una volta fatto partire il programma e creata una nuova base di dati con il comando *New Database* del menu *File*, otteniamo una finestra (come quella nella figura seguente) nella cui parte sinistra abbiamo un elenco dei principali componenti di sistema: *Table , Query, Form, Report, Macro* e *Module*, mentre nella parte superiore abbiamo tre comandi *New, Open* e *Design* che servono per creare un nuovo componente, per visualizzare il contenuto di un componente e per modificare il progetto di un componente (lo schema, se si tratta di tabelle), rispettivamente.



Selezionando *Table* e dando quindi il comando *New* nella finestra sopra mostrata otteniamo la finestra mostrata nell'immagine seguente che permette di definire lo schema della tabella, come specificato più avanti..

Per riempire di dati una tabella, una volta che questa sia già stata definita, basta selezionare nella finestra iniziale la tabella voluta e quindi usare il comando *Open*. Si otterrà una tabella (con i nomi degli attributi come intestazioni) nella quale si potranno inserire i dati riga per riga usando il cursore (spostando il cursore fuori da una riga si indica che l'inserimento per quella riga è terminato). Se i dati inseriti non rispettano i vincoli imposti al momento di definizione della tabella verrà impedito l'inserimento e visualizzato un opportuno messaggio di errore.
Ovviamente si possono riempire le tabelle anche in modo non manuale, usando un opportuno programma che prelevi ad esempio i dati da un file.



Definizione delle tabelle in Access

Dalla finestra vista in precedenza possiamo definire gli attributi (cioè le intestazioni delle colonne) specificandone il nome ed il tipo.

Il tipo fornisce una sorta di primo vincolo di integrità di dominio, in quanto permette di specificare il fatto che i valori per un certo attributo possono esser numerici, testuali ecc. Si possono specificare ulteriormente le caratteristiche di un tipo utilizzando la parte in basso della finestra, parte che varia a seconda del tipo selezionato.

Per inserire il nome di un attributo basta digitarlo nella casella opportuna, mentre per inserire i tipi si possono utilizzare i menu a tendina. Si noti che nel gergo di *Access* un attributo è detto *Field*, cioè campo, e nella rappresentazione dello schema di una base di dati (così come nella fase di definizione del medesimo) gli attributi sono elencati su una stessa colonna (su righe diverse) invece che su una stessa riga come nel modello a tabelle che abbiamo visto.

Dopo avere definito gli attributi per completare la definizione dello schema della tabella occorre indicare quali di essi costituiscono la chiave primaria. Questo si può fare usando il comando *Set Primary Key* nel menu *Edit*, oppure premendo il bottone che rappresenta un chiave nella barra degli strumenti, dopo aver selezionato (con il mouse) gli attributi rilevanti.

Infine, usando il comando *Table Properties* nel menu *View*, si possono specificare dei vincoli di integrità di ennupla (*Validation Rule*) e l'eventuale messaggio che deve essere visualizzato se un vincolo di integrità e violato (*Validation Text*).

NOTA: per specificare i vincoli di integrità si usa sostanzialmente la logica proposizionale, ovvero si usano proposizioni booleane (cioè proposizioni che possono essere vere o false) e operatori logici (*AND*, *OR*, *NOT* ecc). La sintassi che si usa per esprimere i vincoli è la stessa usata per esprimere le condizioni nel linguaggio delle interrogazioni (QBE, vedi più avanti). è disponibile un menu che facilita la scrittura di tali condizioni.

Tipi e parametri in Access

Fra i tipi che *Access* permette ricordiamo i seguenti:

Text:
: testo, ovvero sequenze di caratteri;

Number:
: rappresenta l'insieme dei tipi numerici, che si possono specificare ulteriormente utilizzando la parte in basso della finestra;

Date/Time:
: tipi che permettono di rappresentare date ed intervalli temporali;

Counter:
: è un tipo che associa automaticamente ad ogni riga della tabella un valore numerico che viene incrementato per ogni nuova riga inserita;

OLE *Object*:
: rappresenta un generico oggetto (testo, immagine, informazione multimediale) che può essere gestito tramite OLE (*Object Linking and Embedding*). Come già detto nella parte introduttiva generale, gli oggetti OLE sono gestiti dall'applicazione che li creati, ovvero quando viene selzionato un oggetto OLE viene invocata l'applicazione che lo ha creato.

Fra i parametri che si possono specificare per ogni attributo (utilizzando la parte in basso della finestra) ricordiamo :

Field size:
: dimensione dell'attributo (per i tipi *Text* e *Number*). Per *text* la dimensione specifica il numero massimo di caratteri che si possono memorizzare (1 carattere = 1 byte) mentre per *Number* possiamo specificare

Byte:
: interi su 1 byte

Integer:
: interi su 2 byte

Long Integer:
: interi su 4 byte

Single:
: permette di memorizzare numeri reali in virgola mobile su 32 bit, single fa riferimento alla precisione (singola o doppia)

Double:
: come sopra con 64 bit (doppia precisione).

Format:
: permette di descrivere vari formati di visualizzazione degli attributi

Default Value:
: permette di inserire un valore di default che verrà dato all 'attributo automaticamente se l'utente non ne specifica alcuno.

Validation Rule:
: permette di definire un vincolo di integrità di dominio per l'attributo in questione.

Required:
: con questo parametro si specifica se l'immissione di un valore per l'attributo è obbligatoria o no.

Indexed:
: con questo parametro si specifica se deve essere costruito un indice sull'attributo. Un indice è una struttura dati ausiliaria che permette un accesso efficiente ai dati e che, di norma, è costruita sugli attributi che costituiscono la chiave primaria.

Le interrogazioni e i relativi linguaggi

Lo scopo principale di una base di dati è quello di permettere interrogazioni, dette anche *query*, con le quali accedere al contenuto della basi di dati stessa.
Le interrogazioni si possono esprimere usando un opportuno DML (*Data Manipulation Language*). Di norma le interrogazioni sono definite dai progettisti della base di dati e l'utente finale le usa attraverso delle maschere nella quale vengono immessi i parametri in base ai quali verrà fatta l'interrogazione (come, ad esempio, quando si consulta un orario ferroviario on-line). Non è comunque troppo difficile formulare proprie *query* o modificarne di esistenti, usando un DML (specie se si tratta di linguaggi con interfacce grafiche). Nel seguito vedremo quindi le caratteristiche essenziali dei DML relazionali e quindi il linguaggio QBE usato in *Access*.

Nell'ambito delle basi di dati relazionali possiamo distinguere i seguenti formalismi principali per esprimere interrogazioni:

- formalismi teorici quali algebra relazionale e calcolo relazionale;
- SQL (*Structured Query Language*): il più diffuso linguaggio per DBMS relazionali;
- QBE (*query by example*): linguaggi di tipo grafico abbastanza semplice (il QBE è usato ad esempio in *Access*).

Tutti questi formalismi sostanzialmente permettono di manipolare relazioni usando opportuni operatori. Inoltre, gli operatori principali che si usano sono gli stessi in tutti i formalismi, ovvero sono gli operatori dell'algebra relazionale.

Algebra relazionale

L' **algebra relazionale** è costituita da un insieme di operatori su relazioni (ovvero tabelle) che producono come risultati altre relazioni (o tabelle). Questi operatori possono essere composti per formare espressioni complesse analogamente a quello che possiamo fare con gli operatori aritmetici. Nel seguito, per non appesantire la notazione, useremo spesso il termine relazione (o tabella) come sinonimo di istanza di relazione (o istanza di tabella).

Gli operatori dell'algebra relazionale sono i seguenti

- operatori insiemistici (unione, intersezione, differenza)
- ridenominazione
- selezione
- proiezione
- *join*

Gli operatori insiemistici semplicemente permettono di applicare le usuali operazioni insiemistiche alle tabelle, visto che quest'ultime possono essere considerate come insiemi (ricordiamo che una relazione è un insieme di ennuple, ovvero una tabella è costituita dall'insieme delle sue righe). Si noti che si possono applicare gli operatori insiemistici solo a tabelle definite sugli stessi attributi (ovvero che hanno le stesse intestazioni).

Ridenominazione

L'operatore di **ridenominazione** è un operatore unario (con un solo operando) che modifica lo schema della tabella, ridenominando il nome di alcuni attributi (cioè alcune intestazioni) e lascia inalterata l'istanza dell'operando.
Ad esempio, facendo riferimento alle tabelle dell'esempio della biblioteca viste in precedenza, possiamo applicare la ridenominazione alla tabella Prestito cambiando il nome di attributo Utente in Codice Utente. Il risultato di questa operazione sarà un nuova tabella con le stesse righe della tabella originaria, salvo la riga di intestazione che sarà cambiata come detto. Quindi il risultato dell'operazione

Ridenomina{Utente -> Codice Utente} (Presito)

sarà la tabella

Libro	Codice Utente	Data Prestito
ISBN 1456	ANMBCI 125	12 Dicembre 2002
ISBN 1237	MRZRSSI 134	3 Gennaio 2003

Selezione

L'operatore di **selezione** è un operatore unario che, applicato ad un tabella X, produce come risultato una tabella che ha lo stesso schema di X e che contiene solo quelle righe di X che soddisfano una determinata condizione (specificata dall'operazione di selezione stessa).

Ad esempio, sempre facendo riferimento alla tabella Prestito, potremmo voler selezionare solo le righe per cui la Data Prestito è maggiore del 1 Gennaio 2003 (supponendo di avere un opportuno operatore di confronto fra date). Una tale operazione, che possiamo indicare con

Seleziona(Data Prestito > 1 Gennaio 2003) (Prestito)

produce come risultato la tabella

Libro	Utente	Data Prestito
ISBN 1237	MRZRSSI 134	3 Gennaio 2003

mentre

Seleziona(Autore = Hesse *OR* Autore = Pavese) (Libro)

produce come risultato la tabella

Titolo	Autore	Codice ISBN
Siddharta	Hesse	ISBN 1456
La luna e i falò	Pavese	ISBN 1347

Proiezione

L'operatore di **proiezione** è un operatore unario che, applicato ad un tabella X, produce come risultato una tabella che ha uno schema costituito da un sottoinsieme degli attributi della tabella X (quali attributi devono rimanere è specificato nell'operazione, ovviamente). Il risultato inoltre contiene tutte le righe di X , limitatamente agli attributi (ovvero alle colonne) che sono rimasti.

Ad esempio, facendo riferimento alla tabella Utente, se non ci interessa il codice fiscale possiamo fare una proiezione di tale tabella sugli attributi Nome e Indirizzo. Una tale operazione, che possiamo indicare con

Proietta (Nome, Indirizzo) (Utente)

produce come risultato la tabella

Nome	Indirizzo
Rossi	Via Vivaldi
Bianchi	Via Lulli
Smith	Via Corelli

Quindi, mentre l'operatore di selezione opera una decomposizione orizzontale, l'operatore di proiezione effettua una

decomposizione verticale.

Join

L'operatore di ***join*** è l'operatore più importante dato che è quello che permette di mettere in relazione dati in tabelle diverse. Diamo qui una definizione semi-formale per un operatore di *join* che è detto usualmente theta-*join* e che di solito in algebra relazionale è visto come operatore derivato dall'operatore di *join* naturale. Preferiamo definire il theta-*join* perchè questo è l'operatore più rilevante dal punto dal vista pratico.

Supponiamo di avere due relazioni A, con attributi X, e B, con attributi Y, dove X e Y si suppongono disgiunti (ovvero non ci sono attributi in comune: questo è facilmente ottenibile per mezzo della ridenominazione, inoltre nei sistemi reali, i nome di attributi di tabelle diverse sono considerati diversi). Il risultato dell'operazione

join Cond(A,B)

è una tabella definita sugli attributi X,Y, le cui righe sono ottenute come segue:
si costruisce il prodotto cartesiano di A x B, ovvero si costruiscono le righe ottenute concatenando le righe di A e quelle di B in tutti i modi possibili. Da queste righe si selezionano quindi quelle che soddisfano la condizione Cond e solo queste appariranno nel risultato finale.

Di solito la condizione è una eguaglianza fra attributi di A e B, nel qual caso si parla di equi-*join*. Vediamo un esempio (sempre con riferimento all'esempio della biblioteca).
Se facciamo il *join* della tabella Utenti e della tabella Prestito, specificando la condizione CodiceFiscale = Utente, colleghiamo le informazioni sugli utenti a quelle sui prestiti in modo tale da sapere quali utenti hanno preso in prestito dei libri ed in quale data. Dunque facendo

Utente *join* (CodiceFicale = Utente) Prestito

otteniamo la tabella

Nome	Indirizzo	Codice Fiscale	Libro	Utente	DataPrestito
Rossi	Via Vivaldi	MRZRSSI 134	ISBN 1237	MRZRSSI 134	3 Gennaio 2003
Bianchi	Via Lulli	ANMBCI 125	ISBN 1456	ANMBCI 125	12 Dicembre 2002

si noti che alcune righe non contribuiscono al risultato (ad esempio la riga dell'utente *Smith*) in quanto non soddisfano la condizione. Esiste la possibilità di inserire anche queste ennuple nel risultato usando il *join* esterno che estende, con valori nulli, le ennuple che nel Theta-*join* normale non sarebbero considerate. Esistono tre versioni del *join* esterno:

sinistro:
: mantiene tutte le ennuple del primo operando, estendendole con valori nulli, se necessario

destro:
: come sopra per il secondo operando ;

completo:
: come sopra per entrambi gli operandi.

NOTA: Nei sistemi reali di solito il nome di una attributo è preceduto dal nome della tabella e da un punto (ad esempio, Utente.Nome indica l'attributo Nome della tabella Utenti). Questo fa si che attributi di tabelle diverse siano considerati diversi. Noi omettiamo il nome della tabella perchè tutti i nomi di attributo usati nelle tabelle degli esempi sono diversi.

Interrogazioni

Usando gli operatori visti in precedenza è possibile formulare interrogazioni anche complesse. Vediamo qualche esempio, sempre facendo riferimento al nostro esempio della biblioteca.

1) Vogliamo trovare mentre tutti i titoli dei libri (presenti nella nostra base di dati) il cui autore è Pavese. Dovremo quindi selezionare le righe della tabella Libro, specificando nella condizione che l'autore deve essere Pavese; inoltre, dato che ci interessa solo il titolo, dovremmo fare una proiezione sull'attributo Titolo. Dovremmo quindi fare

Proietta{Titolo} (Seleziona(Autore = Pavese) (Libro))

dove gli apici indicano il fatto che Pavese è un valore e non un nome. Il risultato sarà la tabella

Titolo
La luna e i falò

2) Vogliamo adesso visualizzare le date in cui Rossi ha preso in prestito dei libri. Come prima cosa dovremo selezionare la riga che riguarda l'utente Rossi nella tabella Utente (visto che ci interessa solo lui). Poi, come visto in precedenza, per poter collegare le informazioni presenti nella tabella Utente con quelle nella tabella Prestito dobbiamo fare un *join*. Infine dovremo fare una proiezione per restringerci agli attributi che ci interessano. Dunque avremo:

Proietta{Nome, DataPrestito} (Seleziona Nome = Rossi (Utente) *join* (CodiceFiscale = Utente) Prestito)

ed il risultato sarà la tabella

Nome	DataPrestito
Rossi	3 Gennaio 2003.

3) Infine vogliamo sapere i nomi degli utenti che hanno in prestito dei libri di Hesse ed il titolo di tali libri. Notiamo che adesso, per collegare le informazioni nella tabella Libro con quelle nella tabella Utente, dobbiamo usareanche la tabella Prestito. Quindi dovremo fare due operazioni di *join* e ripetendo per il resto gli stessi ragionamenti di prima avremo:

Proietta{Nome,Titolo} ((Seleziona Autore = Hesse (Libro) *join* (CodiceISBN = Libro) Prestito) *join* (CodiceFiscale = Utente) Utente)

che produrrà come risultato la tabella

Nome	Titolo
Bianchi	Siddharta

Le interrogazioni in Access

In *Access* è disponibile un'interfaccia grafica per la formulazione di interrogazioni, detta QBE (*Query By Example*) che permette di esprimere in modo molto semplice ed intuitivo gli operatori dell'algebra relazionale visti in precedenza.

Per definire una nuova interrogazioni (o *Query*) in *Access* dalla solita finestra iniziale si seleziona *Query* e quindi si usa il comando *New*. Questo apre una nuova finestra nella quale è possibile scegliere fra varie modalità di definizione della *query*, in particolare è possibile usare una creazione guidata (usando un cosiddetto *Wizard*) oppure, selezionando *Design View*, è possibile procedere alla definizione diretta della *query* senza alcun aiuto da parte del sistema. Scegliendo questa seconda modalità viene aperta una finestra per la progettazione di una nuova *query* che consiste di due parti. Nella parte superiore si inseriscono (selezionandole da una finestra che appare separatamente) le tabelle rilevanti per la *query* che dobbiamo progettare. Nella parte inferiore della finestra invece vengono visualizzate un insieme di righe e colonne nella quali dovremo inserire gli operatori opportuni per costruire la *query*.

Più precisamente, ogni colonna nella parte inferiore sarà usata per definire delle condizioni su un singolo attributo, usando le righe che compaiono e che hanno le seguenti intestazioni (almeno nella versione più semplice):

Field.
: Qui va inserito (per ogni colonna, ovvero per ogni attributo) il nome dell'attributo che ci interessa;

Table.
: Qui va inserito il nome della tabella nella quale compare l'attributo (può anche essere omesso in alcuni casi);

Show.
: Questa riga contiene, per ogni attributo, un campo booleano rappresentato da un quadratino. Cliccando nel quadratino il medesimo viene riempito da una crocetta, il che sta a indicare che l'attributo indicato nella stessa colonna sarà visualizzato nel risultato finale. In pratica questa operazione corrisponde alla Proiezione dell'algebra relazionale.

Sort.
: In questa riga possiamo inserire delle condizioni di ordinamento sui valori dell'attributo della stessa colona.

Criteria.
: Nella celle di questa riga sono inserite le condizioni che devono essere soddisfatte dalle righe del risultato. Ad esempio, se l'attributo indicato nel campo *Field* è Nome, l'attributo indicato nel campo Table è Utenti e (nella stessa colonna) inseriamo nella cella Criteria il valore Rossi, vengono selezionate solo le righe della tabella Utenti per cui il valore dell'attributo Nome è Rossi. Questa riga quindi permettte di esprimere l'operatore di Selezione dell'algebra relazionale. Se vogliamo usare più condizioni in *AND*, ovvero vogliamo indicare più condizioni che devono essere tutte soddisfate dalle righe del risultato, dobbiamo riempire più campi della riga Criteria (riempendo anche le altre righe in modo opportuno, con i nomi degli attributi e delle tabelle giusti). Se non è indicato alcun operatore (come nell'esempio) si intende =, altrimenti possono essere indicati operatori quali >, < etc e possono essere composte condizioni complesse usando gli operatori logici.

Or.
: In questa riga possono essere indicate condizioni aggiuntive che vengono considerate in *OR* rispetto alla riga precedente. Ad esempio nel caso detto sopra, se aggiungiamo anche il valore Bianchi nella cella riga *OR* (stessa colonna), vengono selezionate solo le righe della tabella Utenti per cui il valore dell'attributo Nome è Rossi oppure è Bianchi.

Per modificare invece la struttura di una *query* già definita si usa il bottone Design sempre dalla finestra iniziale.

Join ed esecuzione delle interrogazioni in Access

L'operatore di *Join* dell'algebra relazionale si esprime in *Access* inserendo nella cella Criteria il nome dell'attributo che si vuole eguagliare a quello presente nella riga *Field*. Ad esempio, per effettuare il *join*

Utente *join* (CodiceFiscale = Utente) Prestito

visto in precedenza, dovremo indicare nella parte alta della finestra le tabelle Utente e Prestito, e nella parte inferiore dovremo scrivere

Field	Codice Fiscale
Table	Utente
Sort	
Show	
Criteria	[Prestito.Utente]
OR	

dove la parte in azzurro è quello che appare nella finestra di definizione delle *query* (i nomi di attributi devono essere racchiusi fra parentesi quadre). Si noti che gli attributi che vogliamo che siano visualizzati, come detto in precedenza,

devono essere opportunamente indicati nella riga Show. Ad esempio, per effettuare la *query*

Proietta{Nome,Titolo} ((Seleziona Autore = Hesse (Libro) *join* (CodiceISBN = Libro) Prestito) *join* (CodiceFiscale = Utente) Utente)

dovremo indicare nella parte alta le tabelle Libro, Utente e Prestito e nella parte bassa dovremo scrivere

Field	Nome	Titolo	Autore	Libro	Utente
Table	Utente	Libro	Libro	CodiceISBN	CodiceFiscale
Sort					
Show	[X]	[X]			
Criteria			Hesse	[Prestito.Libro]	[Prestito.Utente]
OR					

dove la seconda e la terza colonna corrispondono alla proiezione, la quarta alla selezione, la quinta al primo *join* e la sesta al secondo *join*.

In pratica l'interprete di *Access* per eseguire una *query* effettua il prodotto cartesiano di tutte le tabelle che sono state indicate nella fase di progettazione della *query*, quindi seelziona le righe opportune in base alle condizioni specificate ed infine visualizza le rughe del risultato sugli attributi indicati.

Dato che spesso si usando le stesse operazioni di *join* in più *query*, in *Access* si può indicare un cosidetto *join* predefinito, in modo grafico, direttamente sullo schema della base di dati: selezionando *Relationship* nel menu *Edit* si ottiene una finestra contente gli schemi delle tabelle già definite; qui si possono definire degli archi che collegano due attributi di due tabelle diverse, cliccando con il mouse su un attributo e quindi trascinandolo (tenendo sempre premuto il mouse) fino all'altro. Fatto questo si apre una finestra nella quale è possibile specificare ulteriori caratteristiche del *join* (ad esempio si può indicare un *join* esterno). Una volta dato il comando OK in questa ultima finestra viene inserito un *join* fra le due tabelle. Ad esempio, se abbiamo unito come detto sopra gli attributi Libro della tabella Prestito e Codice ISBN della tabella Libro, tutte le volte che useremo queste due tabelle avremo implicitamente un'operazione di *join* che corrisponde a quella indicata nella quinta colonna dello schema visto poc'anzi.

Una volta che è stata completata la definizione di una *query*, la sua esecuzione avviene premendo il bottone che contiene il punto esclamativo dalla barra degli strumenti, oppure selezionandola dalla finestra iniziale e quindi usando il bottone *Open*.

Aggiornare un sistema: alcuni argomenti

Prof. Simone Martini
3.3 (Aggiornare un sistema: alcuni argomenti)

Perché aggiornare

Un sistema di calcolo ha bisogno di continua manutenzione per essere al meglio della propria funzionalità. Infatti:

- si rendono disponibili nuove versioni del sistema operativo in uso;
  - Si tratta spesso di aggiornamenti che non modificano l'aspetto esterno del sistema, ma correggono errori scoperti nel frattempo, o aumentano la sicurezza o la stabilità del sistema. Di tanto in tanto, sono distribuiti aggiornamenti più importanti, che intervengono anche sull'interfaccia utente e le funzionalità del sistema.
- sono necessarie nuove funzionalità;
  - Il calcolatore che prima era usato soltanto isolato, in un secondo momento viene collegato in rete. Se i moduli di rete del sistema operativo non erano stati installati, o configurati, si tratterà di intervenire su

di essi.
- vengono installate nuove applicazioni.
    - Dopo un certo periodo dall'installazione originale del sistema operativo, le applicazioni disponibili sul mercato richiederanno, per funzionare correttamente, di versioni recenti del sistema operativo.

L'evoluzione della tecnologia *software* e *hardware*, insomma, è così veloce, che dobbiamo ternerle dietro continuamente, se non vogliamo che il nostro sistema diventi rapidamente obsoleto.

In questo modulo di approfondimento toccheremo brevemente alcuni argomenti collegati con l'aggiornamento di un calcolatore. Non si tratta di uno studio organico, che richiederebbe ben altro tempo, anche perché dovrebbe tener conto dell'aggiornamento dell'*hardware* e delle periferiche, un argomento anch'esso estremamente importante. In **bibliografia** si citeranno alcuni testi che potranno essere letti per approfondire l'argomento.

Prima di aggiornare

Prima di iniziare qualsiasi manovra di aggiornamento (*software* e, a maggior ragione, *hardware*), occorre esser sicuri di poter ristabilire lo stato del calcolatore, nel caso malagurato che qualcosa vada storto.

La prima cosa da fare è avere a disposizione un disco di ripristino, cioè un disco dal quale re-installare il sistema operativo in uso, possibilmente nello stato in cui è al momento in cui si decide l'aggiornamento.

Lo stato del sistema operativo varia nel tempo, perché l'installazione di nuove applicazioni e il loro uso modifica alcuni parametri del sistema. La soluzione ideale, pertanto, è quella di ottenere una fotografia del sistema in un certo momento, per poterlo riportare, al bisogno, in quello stato. In mancanza, ci si dovrà riferire al disco originale di installazione del sistema operativo, ma in questo caso perderemo la "storia" del sistema.

Creare un disco di ripristino per Windows 9x e ME.

Un disco di ripristino è un floppy che consente di riavviare il sistema e ripristinarne lo stato. Lo si può creare durante la fase di installazione del sistema operativo. Altrimenti:

- Dal Pannello di Controllo, entrare in Installazione Applicazioni.
- Selezionare la scheda (il "tab") Disco di ripristino e creare il disco (è necessario un floppy nuovo e formattato).

Per ripristinare il sistema operativo dal disco di ripristino, occorre inserire il floppy nel suo lettore (unità A:) e avviare il calcolatore. Può essere il caso che il sistema operativo non sia predisposto per l'avvio dal floppy come prima unità. In tal caso:

- durante l'avvio del BIOS, premere il tasto che permette di accedere alla maschera di configurazione del BIOS (consultare il manuale del sistema);
- cercare l'opzione *Boot order* (o *Boot sequence*, o *First boot device*), che si trova nella categoria di *setup* avanzato;
- selezionare A: come periferica di avvio.

Creare il disco di ripristino per Windows XP *Home Edition*.

Windows XP non fornisce una funzionalità per la creazione di un disco di ripristino allo stesso modo delle altre versioni di Windows. In XP la funzione di ripristino è integrata con quella di *backup* e consente di definire dei punti di ripristino ai quali riportare il sistema in caso di bisogno. è senza dubbio consigliabile creare tali punti di ripristino prima dell'installazione di nuove periferiche o nuovo *software*.

Per creare un punto di ripristino:

- selezionare Programmi a partire dal menù Start;
- scegliere Accessori e poi Utilità di sistema;
- scegliere Ripristino configurazione;
- selezionare Crea punto di ripristino.

Per ripristinare lo stato del sistema:

- nella finestra Ripristino configurazione selezionare Riprista stato precedente;
- selezionare il punto di ripristino desiderato scegliendolo dal calendario.

Il registro del sistema

Windows mantiene le informazioni sullo stato del sistema nel suo Registro, un elenco di tutte le particolarità *hardware* e *software* di un certo sistema. Sul registro vengono memorizzate le periferiche installate o rimosse, le applicazioni installate o rimosse, le preferenze sui colori dello schermo, l'aspetto delle cartelle, perfino i file aperti. Il registro è letto all'avvio del sistema operativo, così da impostare il sistema nel modo voluto dall'utente.

La presenza del Registro di sistema è il motivo per cui installazione e rimozione di *hardware* e applicazioni devono sempre essere compiute attraverso le apposite finestre del Pannello di controllo (Installazione applicazioni e Gestione periferiche).

Dove risiede il Registro in Windows 9x e ME:

Il Registro di sistema è in realtà composto da due file distinti e nascosti che si trovano nella cartella Windows:

- *User.dat*, che memorizza le informazioni relative all'utente;
- *System.dat*, che memorizza le informazioni relativa al sistema.

Per vederli (allo scopo, per esempio, di salvarli) si deve modificare l'opzione di sistema che permette di visualizzare tutti i file, anche quelli nascosti:

- dal menù Start, selezionare Programmi e quindi Esplora risorse;
- aprire la cartella Windows dell'unità C:;
- click su Visualizza, Opzioni Cartella;
- click su scheda Visualizza e selezionare Mostra tutti i file.

Windows 95 memorizza, oltre al Registro corrente, anche la sua versione precedente, creata durante l'ultimo avvio normale, coi nomi *User.da0* e *System.da0*. Questi file possono venire usati per ripristinare il Registro alla situazione precedente.

Windows 98 mantiene cinque versioni precedenti del Registro, nella cartella Sysbckup e col nome *Rbyyy.cab*. Il più recente si chiama *Rb000.cab*; il più antico *Rb005.cab*. Per ripristinare uno dei registri salvati, occorre usare il comando Scanreg:

- avviare il sistema in modalità MS-DOS;
- digitare il comando *Scanreg/restore*
- sono visualizzate le date dei *backup* del Registro disponibili;
- selezionare quella desiderata;
- riavviare il sistema.

In Windows ME e XP non è necessario andare così a basso livello per ripristinare lo stato precedente del sistema. Come abbiamo visto al punto precedente, è possibile usare la funzione Ripristino configurazione di sistema.

I backup

Un *backup* è una copia di tutto o parte il *file system* del sistema operativo. I dati presenti sul disco sono in genere di gran lunga la parte più preziosa del sistema di calcolo. In caso di guasto del disco, di dati corrotti a causa di errori locali o di virus, di cancellazione in caso di re-installazione di *software*, è di vitale importanza poter ripristinare il contenuto del disco (o di alcuni file) nello stato più recente possibile.

è importante definire ed attenersi con scrupolo ad una politica di *backup*. Dovremo definire:

- cosa salvare;
- quando salvare;
- dove salvare;
- come salvare.

Cosa salvare

- Possiamo fare un *backup* dei soli dati dell'utente: documenti, lettere, posta.
- Possiamo fare un *backup* di livello zero: salvare tutto il contenuto del disco. è l'opzione più sicura, ma è costosa in termini di tempo e di spazio di memorizzazione. Pur non trascurando l'importanza di avere copie delle applicazioni, queste ultime vengono modificate molto più raramente dei loro dati (i documenti dell'utente). E spesso per esse avremo a disposizione i dischi per la re-installazione.
- Possiamo fare un *backup* dell'intero sistema, includendo anche le impostazioni del BIOS e delle periferiche, nel caso si debba reinstallare davvero tutto da capo.

La terza opzione è raramente seguita, ma non è da trascurare prima di fare aggiornamenti importanti dell'*hardware* (installazione di nuove unità interne, nuove schede, ecc.). Oltre al *backup* del *file system*, richiede di annotare manualmente tutte le opzioni dei vari componenti *hardware*. Rimandiamo ai testi in bibliografia per informazioni su dove andare a cercarle.

Quando salvare:

La periodicità dei *backup* è molto importante se vogliamo davvero garantire di non perdere una quantità rilevante di informazioni. In un sistema usato giornalmente, possiamo immaginare di fare:

- un *backup* di livello zero almeno una volta al mese;
- un *backup* dei dati dell'utente su base almeno settimanale.

è evidente che *backup* più ravvicinati saranno tanto più raccomandati quanto più sono rilevanti e preziosi i dati memorizzati.

In contesti particolari si manterrà più di un *backup* (in ordine di tempo), per poter ripristinare dati risalenti ad una certa data. Non è questo in genere il caso dei semplici sistemi di calcolo che stiamo trattando.

Dove salvare:

Sono a disposizione molti supporti di memorizzazione su cui effettuare un *backup*. Non tutti hanno la stessa

convenienza in termini di semplicità d'uso, convenienza di archiviazione, costo al MB. Ne elenchiamo alcuni:

- **floppy magnetici**: costituiscono la soluzione più semplice e rudimentale, ma anche di difficile gestione e dal costo non trascurabile se si vuole salvare tutto il contenuto di un disco: per salvare un disco da 4GB occorrono più di 2000 dischetti! Sono utili solo per il salvataggio casalingo, giornaliero, di alcuni dati dell'utente.
- **superfloppy di grande capacità**: unità quali Iomega Zip o Superdisk possono memorizzare su un dischetto (apposito) 120MB o 250MB. Sono una buona soluzione per il *backup* dei dati, ma non per il *backup* dell'intero sistema. Le unità Superdisk sono compatibili coi vecchi floppy.
- **CD-R e CD-RW**: un CD può contenere anche 700MB e costa meno di un euro. Dal momento che molti sistemi sono dotati oggi di una masterizzatore, cioè un'unità che permette di scrivere (bruciare) un CD-R (registrabile una sola volta) o un CD-RW (riscrivibile molte centinaia di volte), si tratta di una soluzione conveniente e razionale per il *backup*, anche se si devono gestire una decina di CD per ogni *backup* di livello zero.
- **unità a cartucce di alta capacità**: sono la soluzione più efficiente, ma non a basso costo. Usano cartucce di dimensione paragonabile a quella di un disco rigido (alcuni GB); il costo di una cartuccia è dell'ordine del centinaio di euro, a cui si deve aggiungere quello dell'unità. Sono molto veloci e permettono una gestione razionale dell'archivio dei *backup*.

Come salvare:

La soluzione più rudimentale, ma anche la più semplice ed efficace per pochi dati, è quella di copiare direttamente i file o le cartelle sul supporto di *backup*. è particolarmente efficace per il ripristino di singoli file dal *backup*: basta andare a cercarli e copiarli di nuovo sul disco.

Si è tentati di usare i programmi di *backup* forniti dal sistema operativo, ma le vecchie versioni di Windows sono molto carenti al riguardo (Windows 95 supporta solo floppy; Windows 98 supporta certe unità esterne, ma non si comporta bene coi CD...).

La soluzione migliore, se si vogliono eseguire *backup* con regolarità e di una certa consistenza è quella di dotarsi di un programma di *backup* di un produttore terzo. Questi supportano in genere una molteplicità di supporti di memorizzazione, permettono di scegliere agevolmente quale parte del *file system* salvare, consentono anche di produrre dei dischi di ripristino. Se si usa un'unità speciale, insieme all'unità sarà stata quasi certamente fornita anche un'applicazione di *backup*. Anche per fare *backup* delle sole cartelle dell'utente può essere conveniente usare un programma di questo tipo. Alcune applicazioni, infatti, mantengono file di configurazione o di lavoro in locazioni diverse da quella dei documenti utente: un *backup* ben fatto deve archiviare anche questi file, per poterli ripristinare correttamente al bisogno. Tipici esempi di applicazioni che memorizzazioni informazioni in molti luoghi diversi sono i clienti di posta elettronica.

Convertire il file system

Si è visto nel modulo introduttivo che la gestione del *file system* è una funzionalità molto importante di un sistema operativo. Il modo in cui i file vengono fisicamente allocati sul disco, infatti, influenza non poco la performance globale del sistema.

Le varie versioni di Windows, che non ha mai brillato per efficienza dei suoi *file systems*, hanno usato tre diverse tecnologie:

- **FAT16** (o semplicemente FAT, *File Access Table*), usato in Windows 3.1 e nei primi Windows 95;
- **FAT32** , usato in tutti i sistemi operativi a partire dalle ultime versioni di Windows 95;
- **NTFS** (*New Technology File System*), disponibile su Windows 2000 e XP.

In **FAT16** si possono indirizzare come unità singole solo dischi fino a 2GB (dischi di capacità maggiore devono essere

suddivisi in più unità virtuali, cioè "lettere" C:, D:, ecc.). Inoltre la quantità minima di allocazione di spazio su disco (cluster) è (o meglio può essere) molto elevata, perché in FAT16 possono essere indirizzati solo poco più di 64mila file o cartelle per unità. Ne segue che in un'unità di 1GB, per ogni file vengono allocati almeno 16KB (e 32KB per file sono allocati in un'unità di 2GB)!

FAT32 è un *file system* più razionale. Rispetto a FAT16 può indirizzare molti più file e cartelle per unità e lo spazio è utilizzato in modo più efficiente. La quantità minima di allocazione è di 4KB per dischi fino a 8GB e poi raddoppia al raddoppiare della capacità del disco. Siccome in un *file system* vi sono molti file di piccole dimensioni, gli stessi file memorizzati in FAT32 occupano molto meno spazio (circa il 30% in meno) di quello necessario in FAT16. Un calcolatore su cui gira Windows 98 (o ME) potrebbe avere il *file system* FAT16, se Windows 98 è il risultato di un aggiornamento di una precedente installazione di Windows 95.

Per convertire il *file system* di qualche unità da FAT16 a FAT32 si utilizza l'utilità Convertitore di unità FAT, che si avvia con Start, Programmi, Accessori e Utilità di Sistema. Possono essere convertite alcune o tutte le unità FAT16 in FAT32.

Non ci sono buoni motivi per conservare un *file system* FAT16 eccetto quello di far girare dei vecchi programmi MS-DOS non supportati da Windows.

NTFS è un *file system* più moderno. Non solo è più efficiente nell'allocazione dello spazio, ma permette alcune funzionalità non disponibili in FAT32, per esempio:

- proteggere file e cartelle con autorizzazioni di accesso;
- cifrare file medianti tecniche crittografiche;
- assegnare quote massime di uso del disco a determinati utenti;
- recuperare file e dati dopo un crash del sistema.

La conversione del *file system* da FAT32 a NTFS è possibile al momento dell'installazione del sistema operativo. Se la conversione non si è fatta durante l'installazione, è possibile usare l'utilità convert, che deve essere invocata dal Prompt dei comandi (Start, Programmi, Accessori, Prompt dei comandi).

L'operazione di conversione cancellerà il *backup* del sistema operativo precedente. è bene fare un *backup* almeno dei dati dell'utente prima di iniziare. Supponiamo che il sistema operativo sia installato su C:. Nella finestra a linea di comando digitare:

convert c: /FS:NTFS

Viene richiesto il nome (simbolico: p.e. MioDisco) del volume da convertire e poi la conferma alla cancellazione del *backup* precedente.

A questo punto un messaggio di errore ci informa che non è possibile effettuare la conversione del volume C:, perché è quello su cui risiede il sistema operativo (il messaggio non si presenta, ovviamente, se convertiamo un volume diverso). Il programma ci propone di smontare il volume, al qual punto rispondiamo NO.

Il programma di conversione ci informa che non è possibile procedere alla conversione e ci chiede se vogliamo programmare la conversione al prossimo riavvio del sistema. A questa domanda rispondiamo SI.

Il programma termina e possiamo chiudere l'interprete dei comandi e riavviare il calcolatore. Al riavvio saremo informati che la conversione è in atto (richiede un certo tempo). Al termine comparirà la solita maschera e il *file system* sarà stato convertito.

I driver

Ogni periferica ha il proprio protocollo per interagire con le altre componenti del sistema. Non è né conveniente né pratico inserire tutti i dettagli di questi protocolli all'interno un sistema operativo. Oltretutto, la disponibilità di nuove periferiche costringerebbe ad un continuo aggiornamento del sistema operativo.

Per interagire con una periferica, il sistema operativo usa invece l'intermediazione di un **driver** (pilota) il cui compito è quello di tradurre le istruzioni generiche che provengono dal sistema operativo (o da un programma utente, per il tramite del sistema operativo) in istruzioni specifiche per una data periferica. Al contempo, un driver rende disponibili al sistema operativo le funzionalità specifiche della periferica.

Per esempio, non tutte le stampanti permettono la stampa fronte-retro. Il driver di una stampante che lo permette, avverte di questo il sistema operativo, che rende così disponibile questa opzione all'utente al momento in cui richiede una stampa.

Al momento della comparsa sul mercato di una nuova periferica, invece di modificare tutto il sistema operativo, si deve semplicemente scrivere il driver per essa, un compito che la casa produttrice della periferica ha tutto l'interesse a svolgere in proprio, visto che l'uso della periferica dipende dalla disponibilità del suo driver.

Al momento dell'installazione, un sistema operativo ha una libreria di driver per tutti i tipi di periferiche montabili (stampanti, modem, dispositivi audio, Webcam, ecc.). Se manca il driver di una periferica, questa non è utilizzabile.

Col tempo, i driver della libreria originale del sistema operativo divengono obsoleti:

- divengono disponibili nuove versioni dei driver esistenti;
- vengono installate nuove periferiche, i cui driver non erano presenti;
- si installa una nuova versione del sistema operativo, che può non contenere un driver di una periferica precedentemente installata.

In questi casi, è importante cercare (nella distribuzione della periferica, sul Web, sui siti delle case produttrici delle periferiche) e installare i driver aggiornati di tutte le periferiche usate. è importante fare *backup* dei driver in uso, nel caso si dovesse reinstallare tutto il sistema operativo dai dischi originali.

Se una periferica non funziona correttamente, nonostante sia stata correttamente installata e configurata, si controlli se il driver in uso è quello più aggiornato disponibile.

Esistono anche driver generici, cioé che funzionano con "ogni" periferica di un certo tipo, e che mettono a disposizione le sole funzionalità di base (per esempio, la sola stampa di testo per una stampante).

Conclusioni

L'aggiornamento di un sistema, *hardware* e *software*, è un'attività la cui importanza non deve essere sottovalutata. In mancanza di aggiornamenti significativi, dopo tre anni un sistema è obsoleto. Con aggiornamenti mirati, è possibile anche raddoppiare la vita di un sistema, a patto di intervenire sia sulle prestazioni *hardware* che sul *software* di sistema.

I **riferimenti bibliografici** contengono due siti che discutono molti aspetti dell'aggiornamento di un PC e un testo orientato soprattutto all'aggiornamento dell'*hardware*, ma il cui approccio molto pragmatico e il continuo insistere anche sull'aspetto *software*, lo rende una lettura molto utile.

Linguaggi orientati agli oggetti

Prof. Simone Martini
3.4 Linguaggi orientati agli oggetti

Introduzione

Buongiorno, sono Simone Martini, insegno linguaggi e paradigmi di programmazione all'Università di Bologna e oggi ci intratteniamo su una lezione che tratterà di linguaggi orientati agli oggetti. È chiaro che in questo modulo non possiamo entrare nei dettagli dello specifico linguaggio: non possiamo imparare a programmare in un linguaggio. Vedremo soprattutto delle tecniche metodologiche, i concetti fondamentali di questi linguaggi e perché sono utili, in generale, all'interno dell'informatica e non solo.

Motivazioni

Vediamo dunque, per cominciare, alcune motivazioni. L'informatica è la scienza che costruisce delle soluzioni eseguibili. L'ingegneria costruisce manufatti, costruisce un ponte tra due argini di un fiume. La soluzione del problema è un manufatto. L'informatica costruisce delle soluzioni *software*, delle soluzioni virtuali. I linguaggi di programmazione sono lo strumento fondamentale con cui queste soluzioni sono prodotte. Ma sarebbe veramente riduttivo pensare che i linguaggi di programmazione servono solo per costruire la soluzione, un po' come, per riprendere la metafora di prima, il calcestruzzo serve a costruire il ponte. I linguaggi di programmazione non servono solo a costruire il programma.

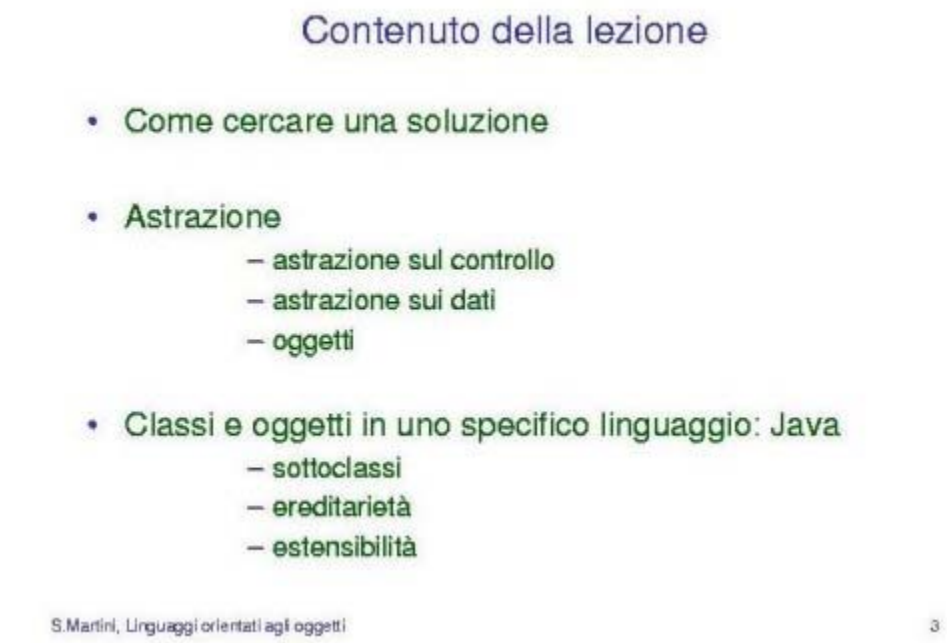
Motivazioni

I linguaggi moderni sono uno strumento per il progettista anche per ricercare le soluzioni: devono fornire degli strumenti che facilitino la ricerca della soluzione. Ovviamente vi dovremo descrivere la soluzione, che rimane certamente la parte fondamentale del programma che viene eseguito. Ma i linguaggi delle ultime generazioni, che vanno dagli anni '60 in poi, sono pensati in modo tale che la correttezza di questa soluzione sia ragionevolmente semplice da verificare, cioè che soddisfi i requisiti per la quale è stata scritta e che non contenga errori. Infine, molto importante è il fatto che sono linguaggi progettati per far sì che il mantenimento, la manutenzione di quella soluzione non sia gravosa.

Motivazioni

Quello che vediamo in questa lezione è il modello ad oggetti che è una specifica metodologia tra quelle maggiormente in uso oggi per realizzare programmi in informatica. La vediamo sia per il suo ruolo importante all'interno dell'informatica moderna, ma anche perché è una metodologia che fornisce dei concetti che sono validi ben oltre l'informatica e utilizza strumenti per risolvere e gestire un problema complesso in modo ordinato.

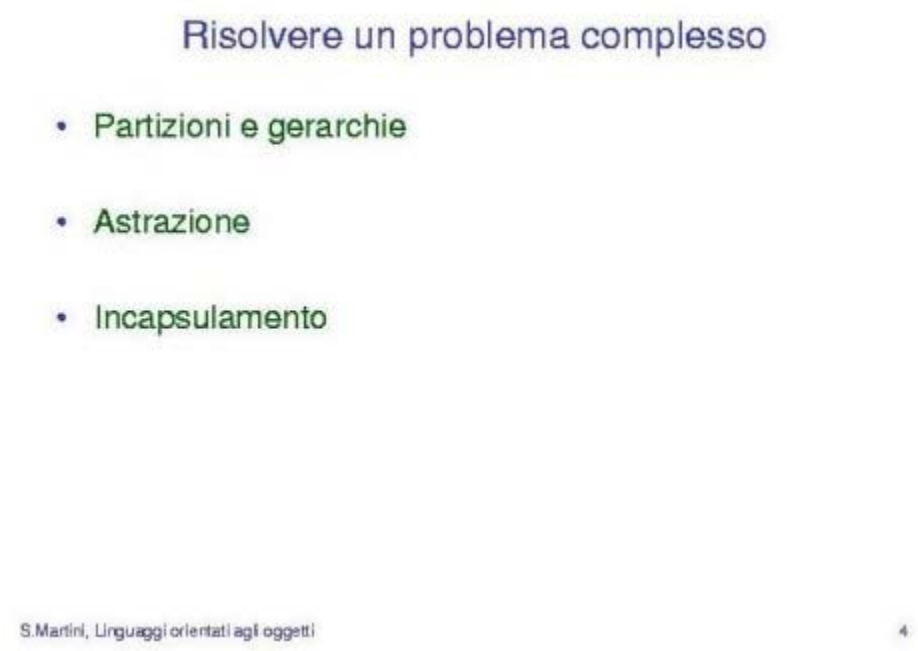
Contenuto della lezione



Ecco il contenuto della lezione. Buona parte della lezione, direi quasi la metà, sarà dedicata a quali sono le caratteristiche di una soluzione di un problema complesso e, per cercare una soluzione, dovremo parlare del concetto di astrazione. Questo è un concetto fondamentale all'interno dei linguaggi di programmazione che l'informatica ha

prodotto nel suo sviluppo teorico. Parleremo di astrazione sul controllo e di astrazione sui dati. Introdurremo poi il concetto di oggetto come un costrutto linguistico che mette insieme tutti e due i tipi di astrazione. Vedremo poi come il costrutto dell'oggetto prende la sua forma in questo tipo di linguaggio, che è il linguaggio Java, e parleremo di aspetti molto importanti tipo: le sottoclassi, l'ereditarietà, l'estensibilità e cercheremo di trarre da questo qualche conclusione.

Risolvere un problema complesso

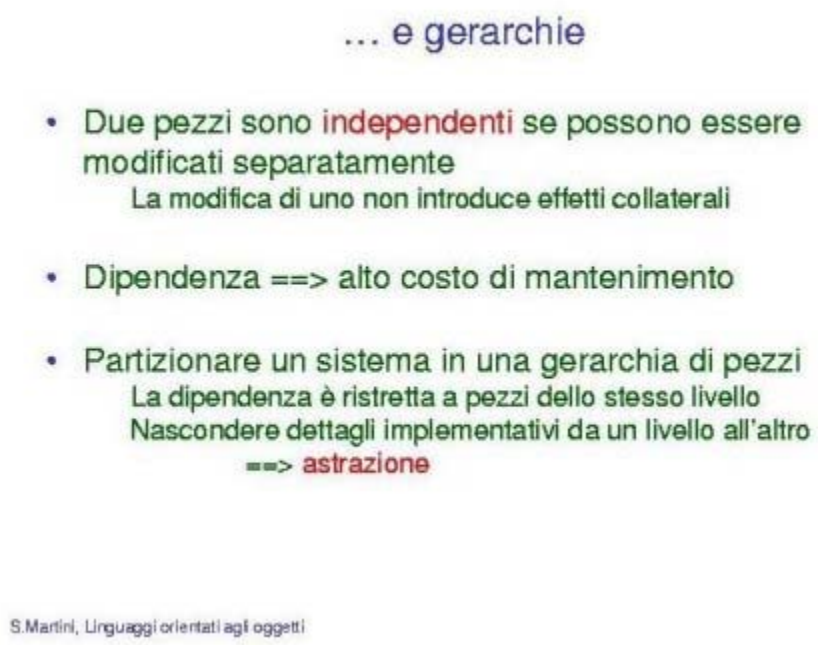


Cominciamo dal primo punto: come si cerca la soluzione ad un problema complesso. Risolvere un problema complesso richiede un notevole sforzo e capacità di invenzione. Le parole chiave di una tecnica risolutiva di un problema complesso sono: partizionare un problema, suddividerlo in livelli gerarchici e, facendo questo, astrarre aspetti importanti del problema. Deve essere un'astrazione che permetta di nascondere determinati dettagli della soluzione ad altri livelli; in termini tecnici viene chiamato "incapsulamento".

Partizioni ...

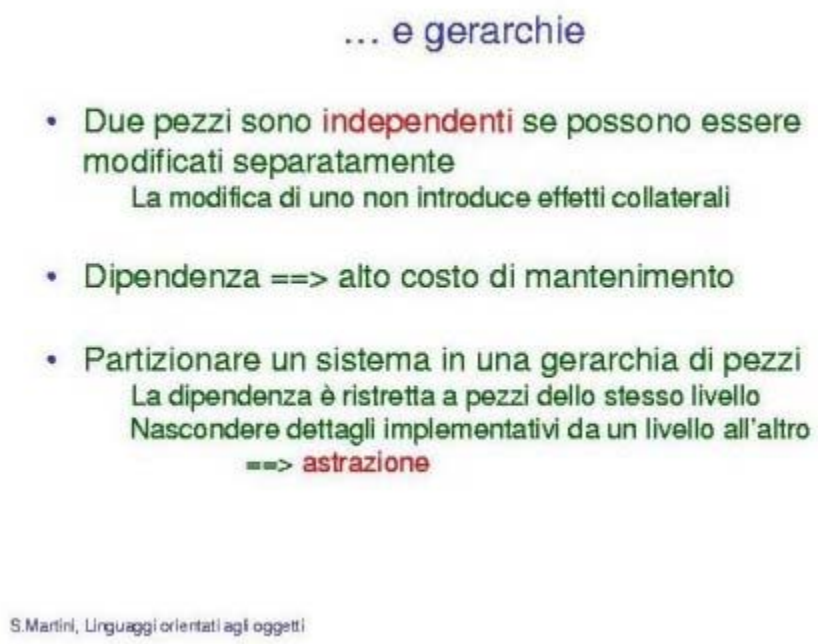
Partizioni: per risolvere un problema complesso lo si deve dividere, perché in generale, solo i problemi piccoli si risolvono bene. Si risolvono bene i problemi per i quali già si conosce la soluzione, quelli che sono varianti di un problema noto e poi ci sono quelli per i quali si vede immediatamente una soluzione. "Dividi et impera" è un vecchio motto che ogni progettista informatico deve avere sempre davanti. Un problema complesso si divide in tanti piccoli, si conquista la soluzione di ogni pezzo in modo separato e poi si rimettono insieme i pezzi. Qui, le parole separatamente e rimettere insieme sono le parole chiave di questa osservazione. Dividere è molto difficile: non c'è una tecnica che ci dice come si fa a dividere un problema in molti pezzi. Quello che sappiamo è l'obiettivo per cui dividiamo un problema: ottenere pezzi ragionevolmente indipendenti che siano ciascuno risolubile in modo indipendente dagli altri. Poi dobbiamo rimettere insieme i pezzi, ciò vuol dire farli comunicare, far scambiare loro informazioni. Ovviamente, quando i pezzi comunicano, le soluzioni che erano indipendenti ritornano in qualche modo a dipendere l'una dall'altra. La comunicazione quindi aggiunge la complessità, la quale aumenta in modo direttamente proporzionale al numero di pezzi indipendenti. In un certo senso c'è un *trade off*: si suddivide per avere pezzi semplici ma poi, tanto più sono i pezzi semplici, tanto più questi dovranno comunicare, quindi ci sarà un sovraccarico (un costo) per la comunicazione. Il progettista astuto è quello che trova il giusto bilanciamento tra questi due aspetti: il risparmio che si ottiene dalla divisione e il costo aggiunto dalla comunicazione.

... e gerarchie



Parlavamo di indipendenza dei pezzi, dei sottoproblemi. Due pezzi sono tra loro indipendenti se possono essere modificati separatamente. Capite che questo è un aspetto estremamente importante se vogliamo garantire che un sistema sia facilmente mantenibile. Infatti, se cambiando un pezzo, tale modifica si ripercuote su gran parte degli altri, la suddivisione in pezzi fatta risulta praticamente inutile. Ciò che vogliamo sono sottoproblemi indipendenti, cioè sottoproblemi per i quali la modifica della soluzione di uno introduca il minor numero possibile di effetti collaterali sulle soluzioni degli altri. È evidente che questo è un problema di costi: quanto maggiore è la dipendenza tanto maggiore sarà il costo della manutenzione.

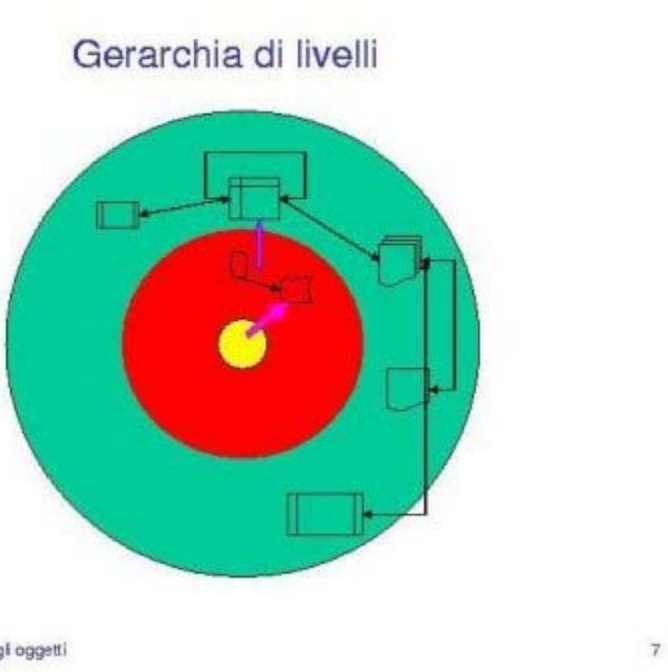
... e gerarchie



Dunque non basta partizionare: bisogna farlo in modo indipendente. Il metodo migliore, perché spesso è il più

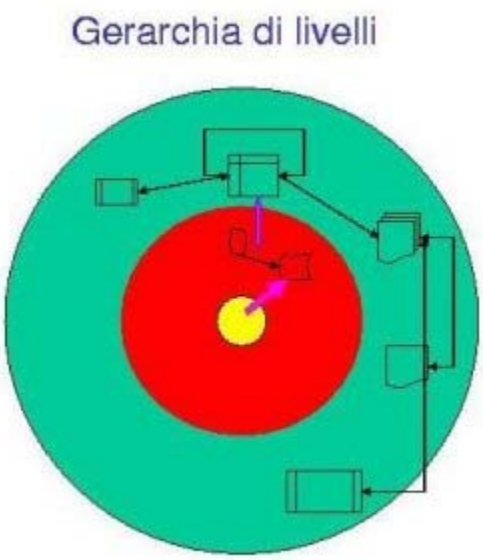
semplice, è quello di partizionare il sistema in una gerarchia di sottoproblemi, cioè strutturare il sistema in livelli in modo tale che i dettagli della soluzione di un livello non siano visibili ad un altro livello, soprattutto a quello soprastante.

Gerarchia di livelli



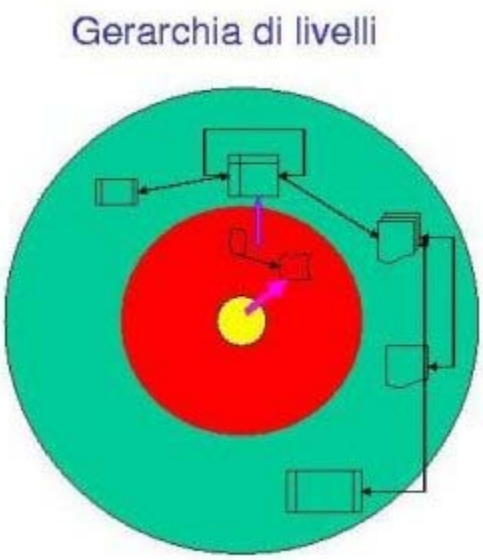
Su questa trasparenza vediamo ora una sorta di esemplificazione. C'è un sistema a tre livelli rappresentati da quelle fasce colorate che si vedono. Concentriamoci sulla parte più esterna, la fascia verde. Per sommi capi un problema è suddiviso in sottoproblemi (quelle scatole che si vedono). Le soluzioni dei sottoproblemi comunicano fra di loro con delle frecce che rappresentano dei flussi informativi (esempio: variabili condivise, sincronizzazioni) tra questi sottoproblemi.

Gerarchia di livelli



Vedete che questi sottoproblemi dialogano allo stesso livello. Tutto ciò che avviene a livello inferiore è nascosto e può essere utilizzato esclusivamente attraverso delle interfacce (vedremo più avanti cosa intendiamo con questo), che sono in pratica dei modi di interazione (in figura il livello verde verso il livello rosso) che agiscono esclusivamente sulla membrana che separa questi due livelli. È evidente poi che al livello sotto utilizzeremo la stessa tecnica. Ad esempio questo pezzo, quello puntato dalla freccia di colore lilla in figura, sarà realizzato a livello inferiore come una molteplicità di pezzi e per essere risolto sarà a sua volta suddiviso in altri pezzi. Anche a livello rosso avremo una struttura di pezzi (sottoproblemi) che dialogano e interagiscono tra di loro. Qui è identificata esclusivamente quella parte di sottoproblema (i due pezzi) che corrispondono alla soluzione del problema del livello superiore, quello puntato dalla freccia lilla.

Gerarchia di livelli

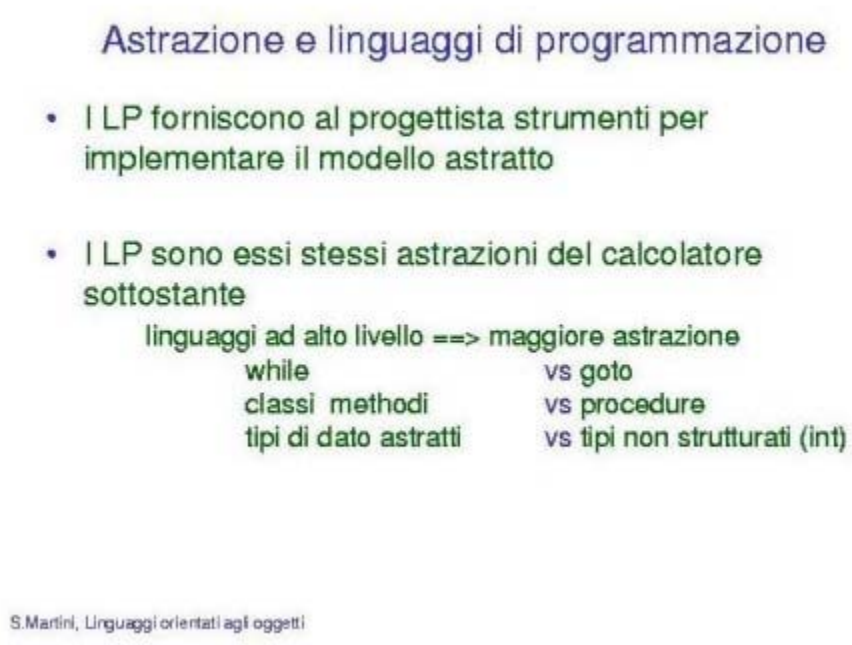


La cosa essenziale è che questi livelli sono tra loro separati. Essi interagiscono esclusivamente attraverso l'interfaccia: il modulo puntato dalla freccia al livello superiore interagisce esclusivamente con i moduli che stanno sotto di lui. Non ci deve mai essere un flusso informativo diretto tra moduli del livello rosso e moduli del livello verde, perché questo introdurrebbe una complessità di gestione che renderebbe il sistema difficilmente manutenibile. Come vedete stiamo parlando ad altissimo livello, non stiamo ancora parlando di informatica, ma di risolvere problemi mediante strutturazione. Quello che l'informatica offre sono degli strumenti linguistici perché questa strutturazione, questa gerarchia di livelli, venga rappresentata all'interno di un testo che, nel caso specifico dei linguaggi di programmazione, è un testo eseguibile (un programma).

Il ruolo dell'astrazione

Ritorniamo a questo problema della gerarchia. Per garantire che i livelli siano separati, il concetto chiave è quello di astrazione. Astrarre vuol dire identificare alcune proprietà importanti di quello che si vuol descrivere, di quello che si vuole risolvere, e concentrarsi solo su queste proprietà ignorando le altre. È un procedimento ben noto della matematica e della fisica (le loro leggi sono un modello astratto della realtà) ed è una tecnica di estrema importanza all'interno della progettazione. Non ci sono tecniche standard per astrarre, ciò che è rilevante dipende certamente da ciò che si deve descrivere, ma dipende anche in modo essenziale dallo scopo del progetto. Tornando all'esempio della fisica, è evidente che il modello galileiano è utile perché fornisce una definizione a macro-livello della realtà, se si vuole andare a livello atomico quella astrazione non funziona più, ma bisogna utilizzarne altre. Questo ovviamente è vero anche all'interno dell'informatica, dove ci sono vari livelli di astrazione.

Astrazione e linguaggi di programmazione



I linguaggi di programmazione forniscono al progettista strumenti per realizzare un modello astratto e per descriverlo al livello di astrazione scelto. Un buon linguaggio di programmazione permette di descrivere un modello a vari livelli; tornando all'esempio di prima, a livello verde oppure a livello rosso. Infine un linguaggio di programmazione descrive anche l'interfaccia da eseguire. Dopo tutto, i linguaggi di programmazione sono, a loro volta, astrazioni dalla macchina su cui essi girano. Abbiamo visto nel modulo precedente, quello che descrive per sommi capi la struttura dei linguaggi di programmazione, come i linguaggi ad alto livello forniscono strumenti linguistici; ad esempio l'iterazione determinata *while* che è un'astrazione su un costrutto di controllo sottostante. Così, a livello di linguaggi più evoluti, come ad esempio quelli ad oggetti, ci sono dei concetti come le classi e i metodi che sono delle astrazioni su costrutti più rudimentali che sono i sottoprogrammi, o le procedure, o le funzioni.

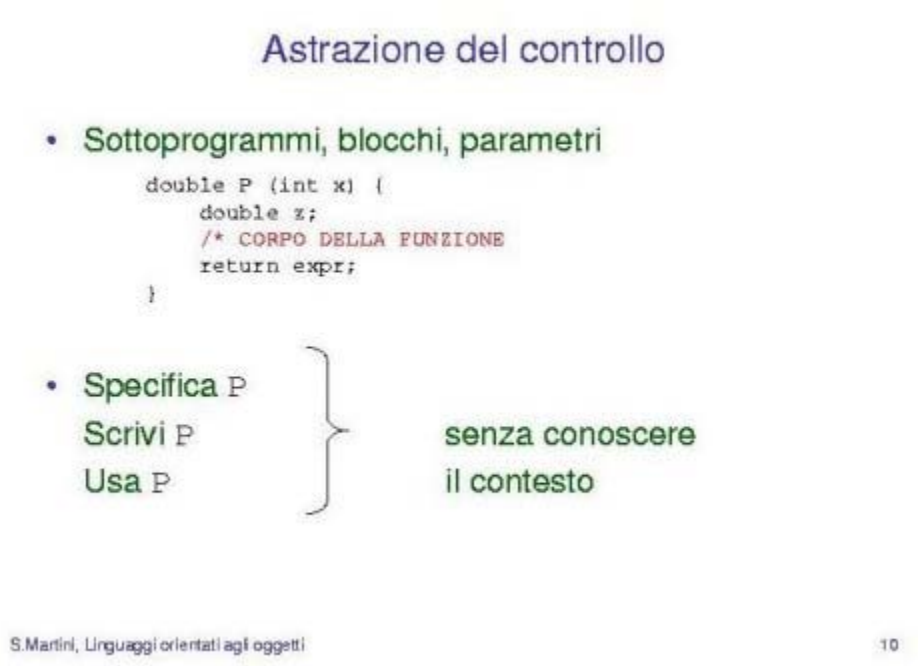
Astrazione del controllo

Stiamo parlando di strumenti linguistici che i linguaggi di programmazione forniscono per l'astrazione. L'astrazione nei linguaggi di programmazione avviene in due sapori: il primo è quello dell'astrazione sul controllo: tipico esempio sono i sottoprogrammi (procedure o funzioni). Quello che vedete sulla trasparenza è lo schema della definizione di una procedura, di una funzione definita in Java o C. Come vedete c'è un nome P; il pezzo di programma successivo, detto corpo della funzione, serve proprio per definire tale nome P, cioè viene dato il nome P proprio a quel pezzo di programma. Il corpo interagisce con l'esterno mediante l'interfaccia, che è costituita dal parametro chiamato x, dove in questo esempio è definito come un numero intero, mentre il risultato dell'esecuzione del corpo è un numero reale, o meglio un numero razionale in doppia precisione.

Astrazione del controllo

La cosa rilevante, che mi sta a cuore comunicarvi in questo contesto, è che qui stiamo astraendo sul corpo della funzione. Il corpo della funzione è un pezzo di programma (un pezzo di codice) che viene nascosto, dal quale si astrae mediante l'intestazione. Questa intestazione P con parametro x è tutto ciò che si vede del corpo della funzione dall'esterno di questo sottoprogramma: è un astrazione su un controllo, sul corpo della funzione. Questo vuol dire che per usare la procedura P non c'è bisogno di conoscere ciò che c'è scritto dentro il corpo della funzione, basta conoscere la sua interfaccia: il suo nome P e che parametro vuole, un numero intero.

Astrazione del controllo



L'uso di P è indipendente dal corpo e da chi e come è stato scritto. Dualmente possiamo scrivere il corpo della funzione senza preoccuparci di dove sarà usata, né del perché sarà usata. L'unica cosa di cui dobbiamo tener è sapere bene cosa dovrà fare P, la sua specifica, e scrivere quindi il suo corpo in modo tale che P soddisfi la sua specifica. Questo è un potentissimo strumento di astrazione, che è poi assistito da altri meccanismi di questo linguaggio che sono per esempio i blocchi, costituiti dalle parentesi graffe, al cui interno è possibile definire delle variabili locali. Ad esempio z è una variabile che può essere usata nel corpo della funzione, ma che non è vista all'esterno; all'esterno di P la variabile z non esiste. Di nuovo astraiamo qualcosa dall'esterno verso l'interno.

Astrazione del controllo, II

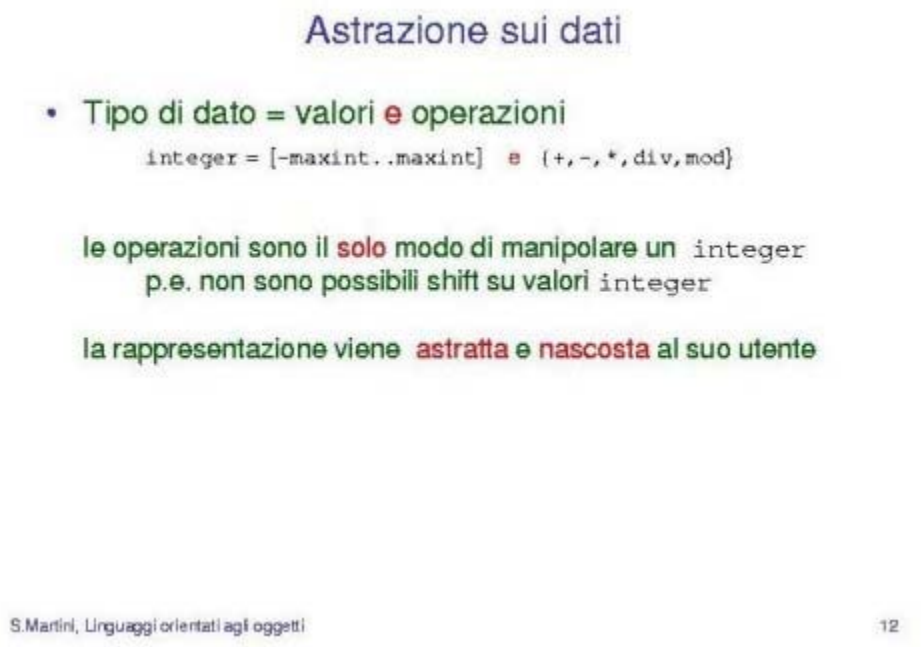
L'astrazione sul controllo è un modo per nascondere pezzi di programma; fornisce quella che si chiama astrazione funzionale: ogni componente, in questo caso ogni procedura P, fornisce dei servizi al suo ambiente. L'astrazione descrive il comportamento esterno, il nome e i parametri che usa; nasconde dentro il blocco interno, dentro il corpo, tutti i dettagli. L'interazione tra due componenti è limitata al loro comportamento esterno: si chiamano per nome ed interagiscono tramite i parametri, senza alcun riferimento diretto tra il corpo dell'una e il corpo dell'altra. Ciò che il linguaggio di programmazione fornisce è un meccanismo per garantire che le procedure siano separate l'una dall'altra.

Astrazione del controllo, II

Il linguaggio di programmazione controlla che non si possa accedere all'interno di P se non attraverso il suo nome;

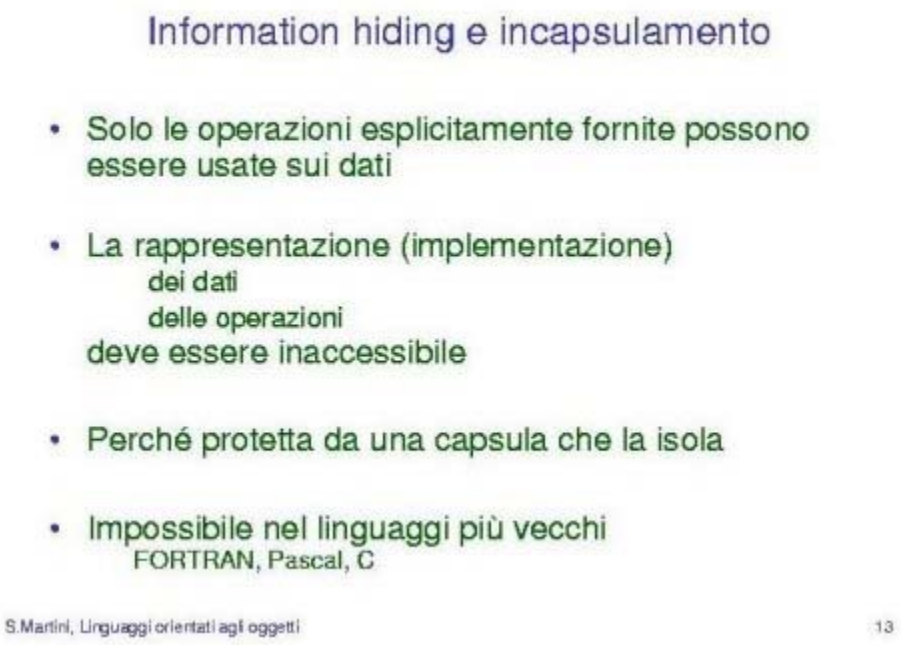
questi sono i meccanismi di astrazione forniti dal linguaggio. Come essi siano garantiti non possiamo affrontarlo qui (si chiamano blocchi, ambiente, ambiente statico, tipi di dato). La cosa che è importante sottolineare è che il compito fondamentale di un linguaggio di programmazione non è solo fornire le istruzioni elementari con le quali risolvere un problema, ma anche e soprattutto fornire strumenti per garantire che il programmatore utilizzi un progetto il più possibile astratto per i dati. Dicevamo che l'astrazione viene in due sapori: la prima è l'astrazione sul controllo, la seconda è l'astrazione sui dati.

Astrazione sui dati



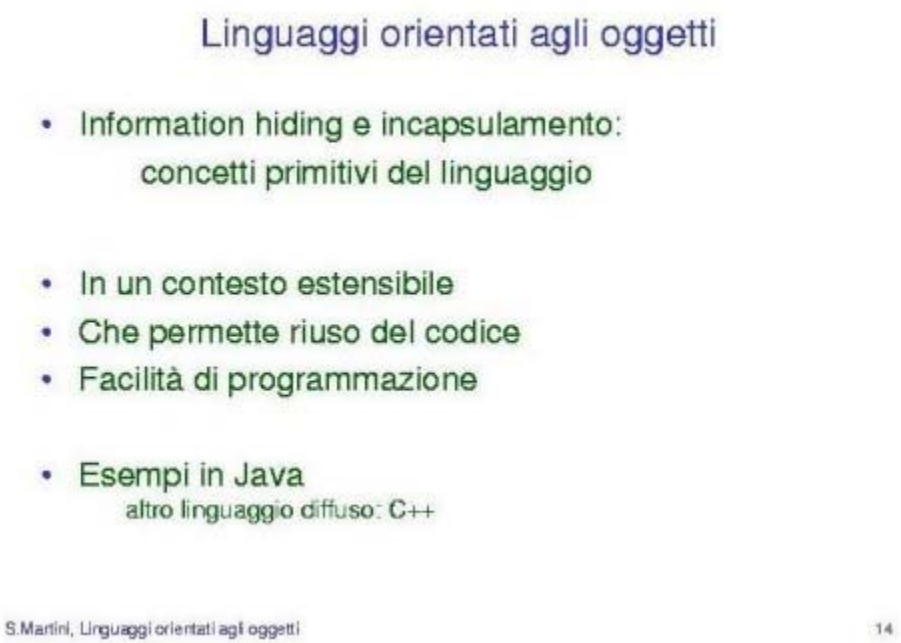
Dobbiamo introdurre brevemente il concetto di tipo di dato. Un tipo di dato è una coppia valori e operazioni. Ad esempio il tipo di dato interi (qui *integer* è un tipo di dato definito in Pascal) è un sottoinsieme, un intervallo finito, dei numeri interi e delle operazioni che agiscono su di essi. La cosa essenziale è che questi interi possono essere manipolati esclusivamente con quelle operazioni. Le operazioni fornite sono il solo modo per manipolare un intero, non è possibile, per esempio, utilizzare *shift* logici o *and* booleani, perché la rappresentazione degli interi viene nascosta all'utente, viene astratta dal linguaggio di programmazione. Mentre in un linguaggio di basso livello, ad esempio il linguaggio *Assembler*, posso accedere ad un intero come sequenza di bit, posso farci tutte le porcherie che voglio (*shift* logici, *and* logici e così via), in un linguaggio astratto, di alto livello, la rappresentazione è nascosta. Per usare qualche altra parola chiave, questa tecnica di separare l'implementazione dal suo uso si chiama nascondimento dell'informazione o *information hiding*.

information hiding e incapsulamento



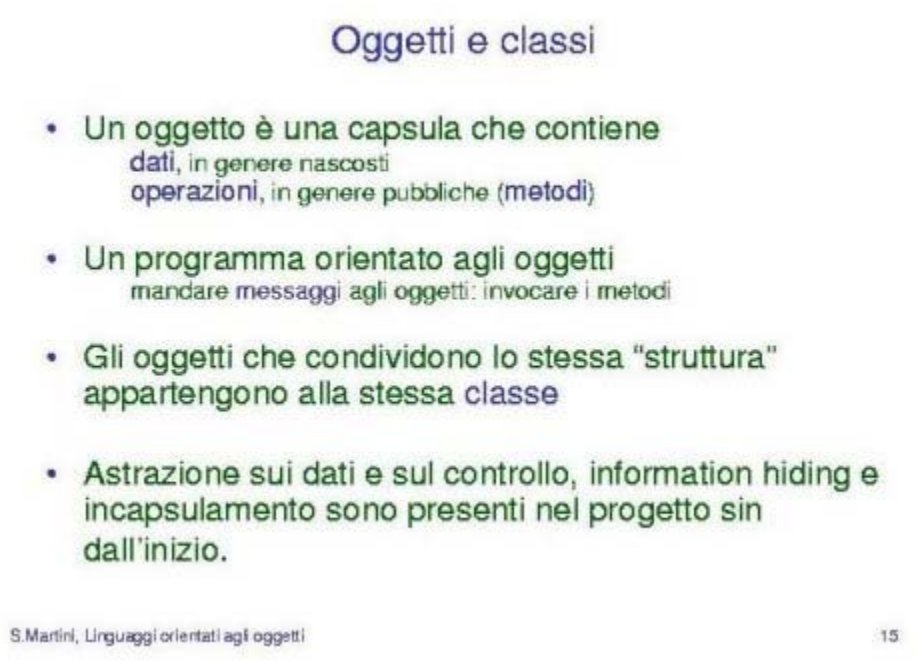
Solo le operazioni esplicitamente fornite possono essere usate sui dati; abbiamo visto l'esempio degli interi e la rappresentazione è del tutto inaccessibile all'utente. Ora, quello che noi vorremmo, è che questo non fosse possibile solo per i tipi, per le strutture predefinite del linguaggio, ma anche per quelle strutture necessarie per i sottoproblemi che l'utente definisce ed usa nel proprio programma. Vogliamo far sì che questa separazione fra uso ed implementazione sia non solo fornita per le strutture predefinite, ma l'utente possa esso stesso definire delle capsule che isolino la rappresentazione dal suo uso. Questo incapsulamento è impossibile nei linguaggi vecchi: in Fortran, in C, è largamente impossibile in Pascal, ma è possibile nei linguaggi moderni. I linguaggi orientati agli oggetti rappresentano una metodologia nella quale tale incapsulamento è realizzabile.

Linguaggi orientati agli oggetti



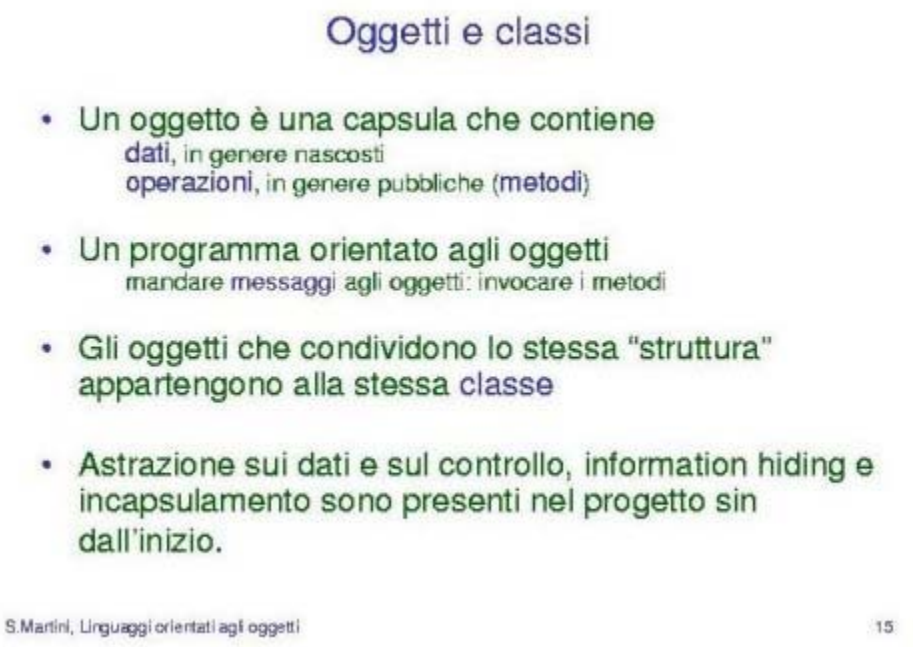
Nei linguaggi orientati agli oggetti l'*information hiding* e l'incapsulamento sono concetti primitivi del linguaggio stesso. Esso è costruito intorno a costrutti linguistici per garantire il nascondimento dell'informazione. Tutto questo è possibile anche in altri linguaggi non orientati agli oggetti. La cosa fondamentale che contraddistingue i linguaggi orientati agli oggetti è che questo avviene in un contesto estensibile, cioè una volta incapsulato un oggetto, questo non è una capsula scolpita nella pietra che non può essere più modificata, ma una capsula che può essere estesa, permettendo il riuso del codice e la facilità di programmazione. Questa è la chiave di volta della tecnologia orientata agli oggetti che la differenzia dalle altre tecniche di programmazione che riescono a garantire l'incapsulamento. Noi vedremo qualche esempio nel linguaggio Java, che è uno dei più diffusi; l'altro linguaggio ampiamente usato orientato anch'esso agli oggetti è il C++, ma per eleganza di progetto, per semplicità d'uso, per tutta una serie di motivi che forse non è il caso di elencare, Java è il linguaggio più indicato per introdurre la programmazione orientata agli oggetti.

Oggetti e classi



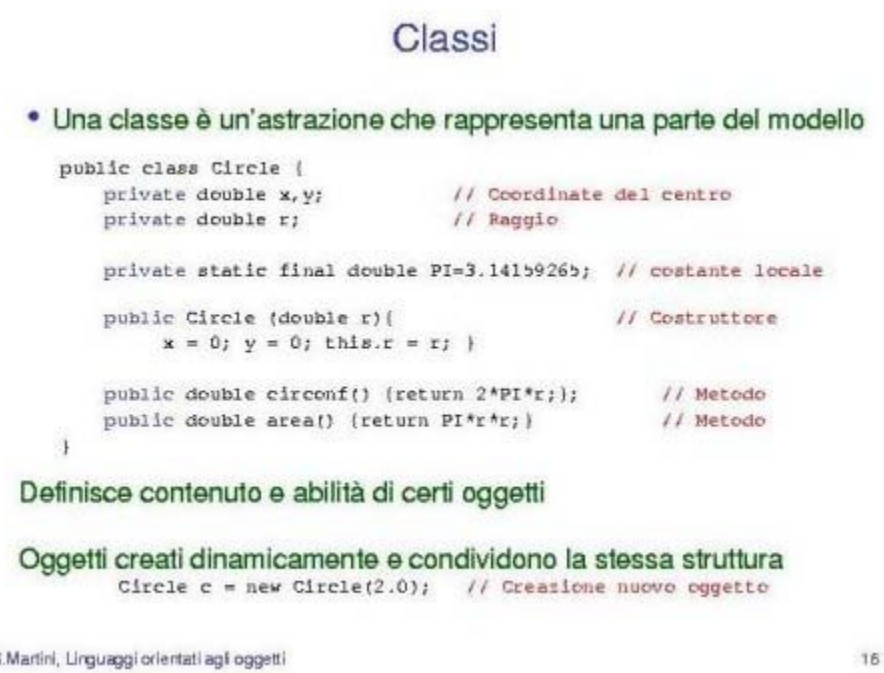
Scendiamo un po' nel dettaglio di cosa sono gli oggetti e introduciamo un nuovo concetto che è quello di classe. Abbiamo già detto che la cosa fondamentale è l'incapsulamento e un oggetto è infatti una capsula che contiene due cose: i dati e le operazioni su quei dati. Un oggetto è quindi qualcosa di unitario che garantisce nello stesso momento sia l'astrazione sul controllo, sia l'astrazione sui dati. Questi due tipi di astrazione vengono quindi fusi in un oggetto, che astrae sui dati che in genere sono nascosti e sulle operazioni che possono agire su quei dati. In generale sono pubblici i nomi e l'uso delle operazioni, mentre invece i dati sono spesso totalmente nascosti. Nel gergo *object-oriented* le operazioni si chiamano metodi. In sostanza un oggetto incapsula dei dati, sui quali si può agire attraverso dei metodi.

Oggetti e classi



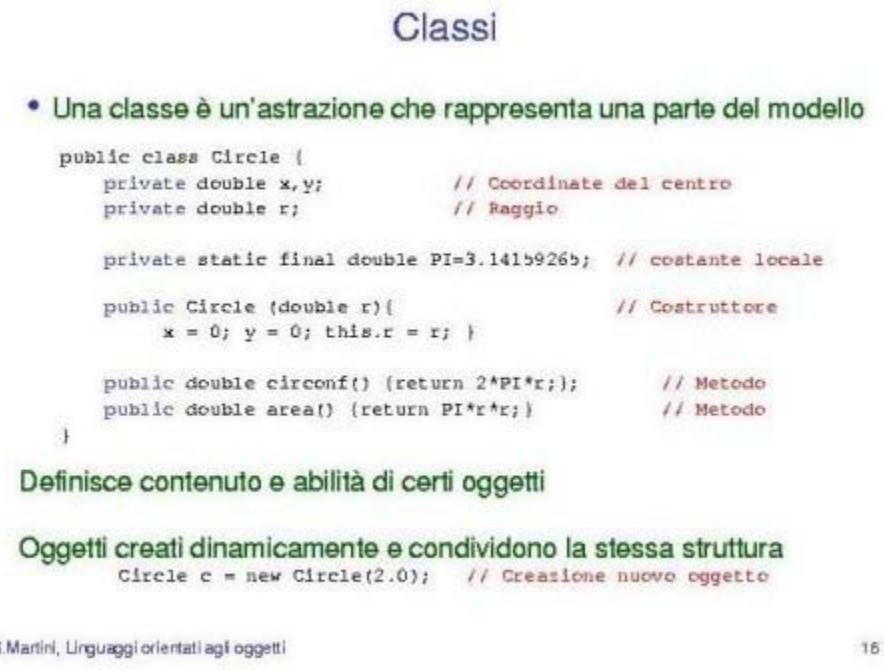
Come vengono invocati questi metodi? Come possiamo far sì che un metodo agisca sui dati? Ciò viene fatto inviando un messaggio all'oggetto. Un programma orientato agli oggetti è una collezione di oggetti che comunicano mediante messaggi. Un messaggio è la richiesta di invocazione di un metodo: un oggetto A chiede ad un oggetto B, mediante un messaggio, di eseguire un metodo di B sui dati di B e di restituire il risultato ad A. Ora, questi oggetti non costituiscono una collezione caotica, molti di essi sono simili tra loro, in particolare, se condividono una stessa struttura, si dice che appartengono alla stessa classe. In realtà la progettazione avviene in modo inverso, cioè vengono prima definite alcune strutture comuni a più oggetti, le classi appunto, e poi vengono creati gli oggetti a partire dalla classe. Prima definiamo la struttura del nostro mondo, definendo varie classi e poi si creano gli oggetti che popolano questo mondo, partendo dalle classi, dalla struttura che questi oggetti devono avere.

Classi



La classe, dal punto di vista progettuale è un'astrazione che rappresenta una parte del modello. Qui abbiamo, nella trasparenza, un esempio di una classe definita nel linguaggio Java. Una classe che abbiamo chiamiamo *circle* e che rappresenta un cerchio. Cominciamo col vedere come è strutturata questa classe. Abbiamo già detto che un oggetto è un'astrazione sia sui dati che sul controllo. Una classe è la definizione della struttura di molti oggetti. Innanzitutto vediamo che la classe considerata è composta di due parti, la parte superiore (le prime tre righe della trasparenza, quelle contraddistinte dalla parola *private*) è la parte che specifica i dati, l'astrazione sui dati.

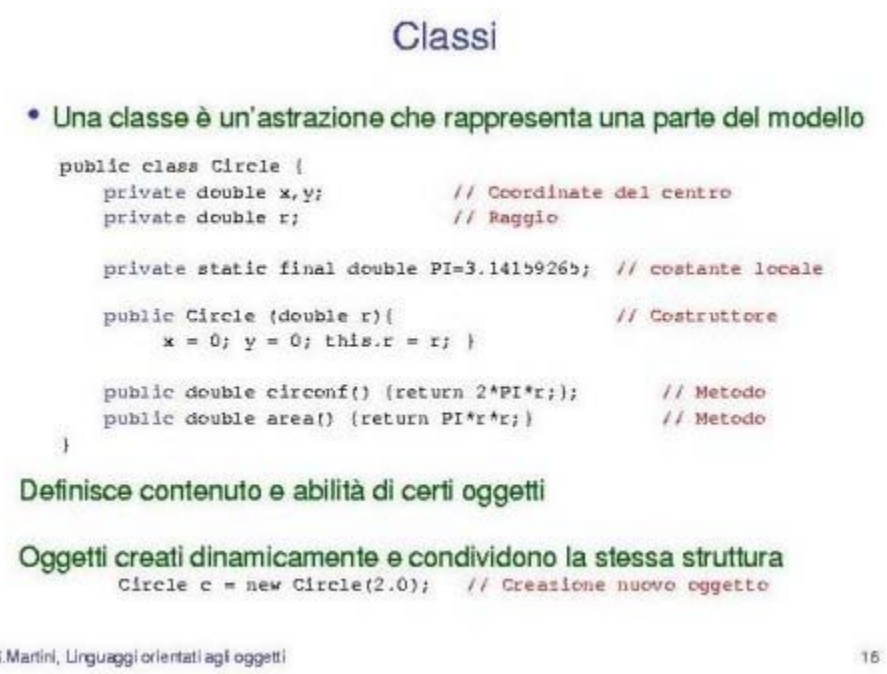
Classi



La seconda parte (le seconde tre linee della classe, quelle contraddistinte dalla parola *public*) sono le operazioni, i metodi, cioè l'astrazione sul controllo. Vediamo ora i dati: sono due variabili di tipo reale x, y, un'altra variabile r e un

altro nome p che viene fissato al valore 3.1415. Un cerchio in questa astrazione viene quindi individuato da una coppia di reali x e y, che interpretiamo come le coordinate del centro del cerchio, e un reale che è il raggio del cerchio. Fa parte dei dati di questo cerchio anche una costante p (= p-greco). Quali operazioni manipolano questi dati? Sono tre: l'operazione *circle* che costruisce un cerchio. Vuole come parametro un numero reale r e costruisce un cerchio di centro l'origine degli assi cartesiani e di raggio r. Tale metodo, in gergo *object-oriented*, viene detto costruttore, perché serve a inizializzare (a costruire) un oggetto, che è appunto un cerchio che ha centro nell'origine degli assi e raggio r, quest'ultimo fornito al momento della creazione.

Classi



Ci sono poi altri due metodi: circonf, che restituisce 2*pi-greco*r (la circonferenza di quel cerchio su cui agirà questo metodo) e area, che ritorna pi-greco*r^2 (restituisce l'area di quel cerchio su cui questo metodo agirà). Senza entrare in troppi dettagli, osservate come la parte dei dati è stata contraddistinta dalle parole *private*, che non sono indispensabili ma in questo caso stabiliscono che quei dati sono inaccessibili dall'esterno, sono effettivamente astratti, sono nascosti dall'esterno della classe. Mentre invece i metodi (il costruttore *circle*, Circonf, Area) sono pubblici, cioè accessibili dall'esterno. C'è quindi un nascondimento totale dei dati ed una pubblicità dei metodi, che ovviamente si limita solo al loro nome e parametri; non si potrà mai vedere all'esterno cosa si trova dentro il corpo di questi metodi. Una classe definisce: la struttura, il contenuto, le capacità e le abilità di determinati oggetti.

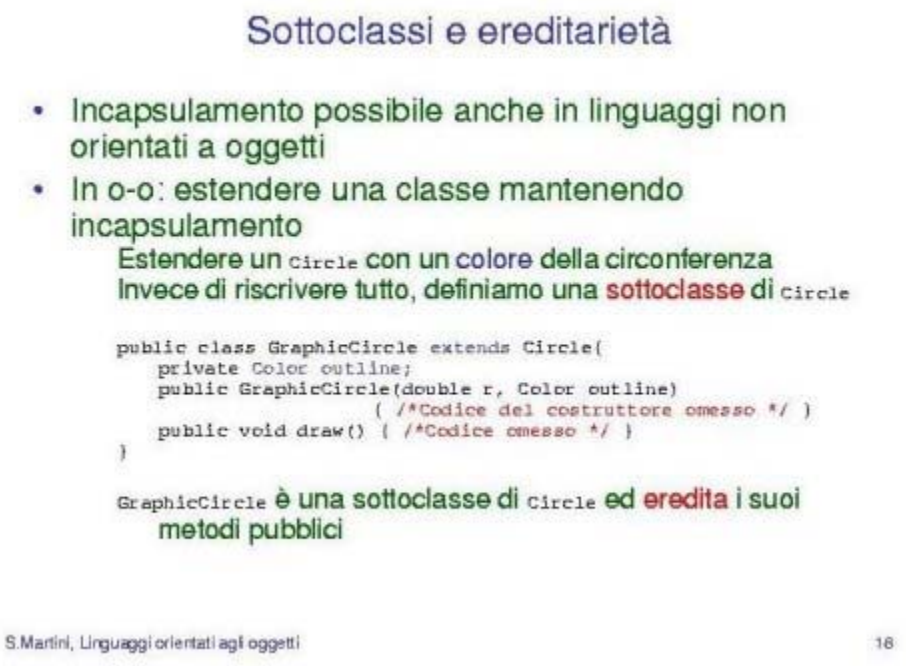
Oggetti

Gli oggetti vengono creati dinamicamente, condividono tutti la stessa struttura, quella fissata dalla classe. Vediamo in questo esempio (nell'ultima riga della trasparenza) che viene costruito un nuovo cerchio di nome c e ciò viene fatto a partire dal costruttore *circle* con parametro 2. c sarà quindi un oggetto che ha questa struttura: avrà al suo interno quattro numeri reali (compreso pi-greco) e tre metodi per agire su di esso. In particolare c è stato creato col parametro 2, quindi è un cerchio con centro nell'origine degli assi e raggio 2. Questi dati non sono accessibili dall'esterno, l'unica cosa che c ci forniscono è la capacità di rispondere a questi metodi: il metodo *circle* che lo ha costruito, il metodo circonf che ci restituisce la circonferenza di questo cerchio di raggio 2 e il metodo area che ci restituisce l'area di questo cerchio di raggio 2.

Oggetti

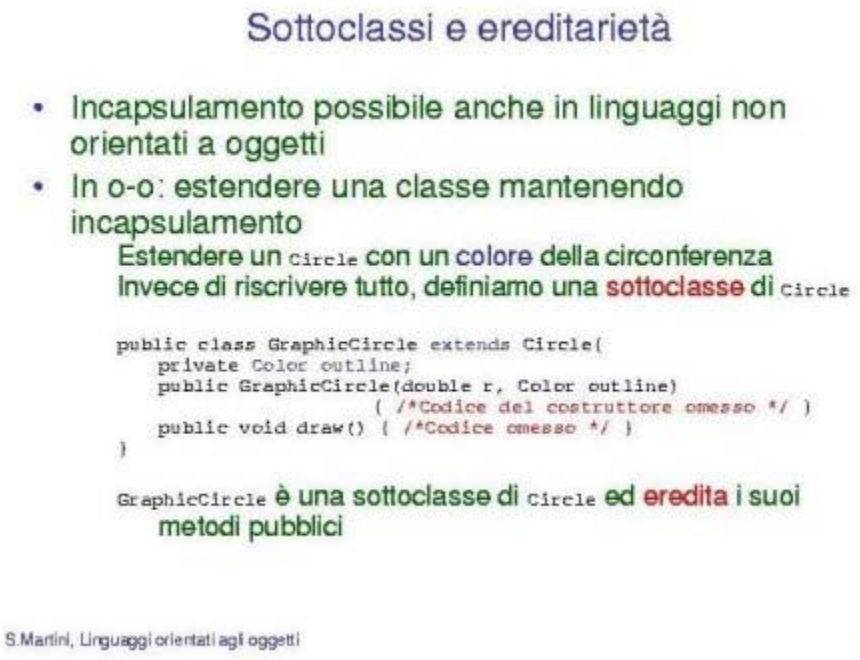
Gli oggetti sono creati dinamicamente come istanza di una classe. Abbiamo creato prima l'oggetto c, ora creiamo c1 e c2. c1 è un altro cerchio di raggio 2, ha le stesse caratteristiche di c, è diverso da c perché ha un nome distinto ed è stato creato come nuova istanza. c2 è un altro cerchio di raggio 1. Quindi, in questo momento, il nostro mondo è popolato da tre oggetti, tutti con la stessa struttura, ma diversi come valori: c e c1 sono cerchi di raggio 2, c2 è un cerchio di raggio 1. Questi cerchi, o oggetti, rispondono ai messaggi che corrispondono ai loro eventi pubblici; qui abbiamo due esempi, c1 risponde al metodo area. Questa sintassi, che vedete nella trasparenza ,c1.area sta ad indicare che posso mandare il messaggio corrispondente al metodo area a c1. Significa che l'oggetto c1 risponde al metodo area restituendo la propria area. Lo stesso possiamo fare con c e c2 inviando lo stesso messaggio a questi altri due oggetti. c, c1 e c2 rispondono quindi agli stessi metodi, ovviamente con risultati diversi: c1.area ci da l'area di un cerchio di raggio 2, c2.area ci da l'area del cerchio c2 che ha raggio 1. I dati sono del tutto inaccessibili all'utente, il nascondimento dell'informazione è in questo caso perfetto.

Sottoclassi e ereditarietà



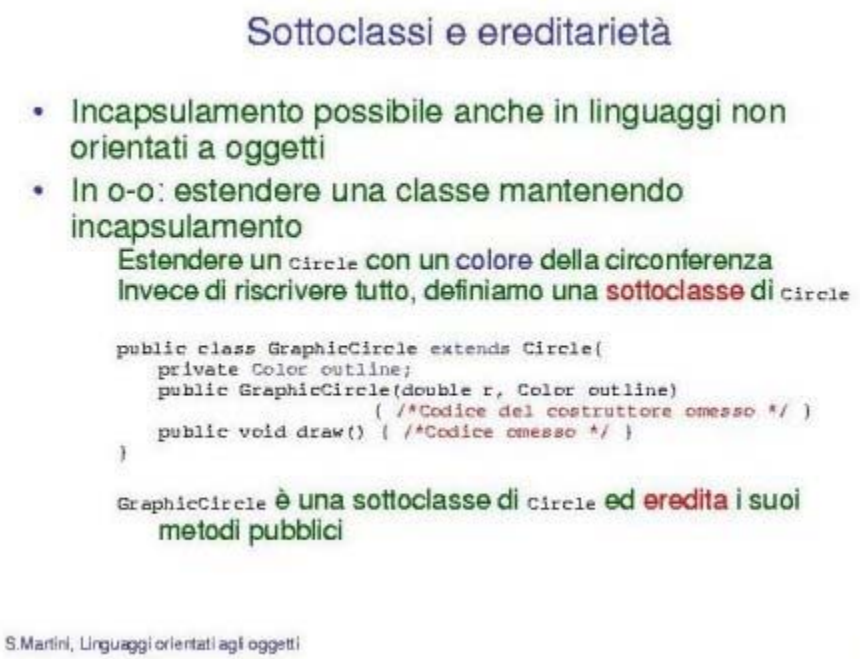
L'incapsulamento, che è uno dei concetti sui quali abbiamo insistito maggiormente fino adesso, è possibile in linguaggi che non sono orientati agli oggetti. Ciò che invece caratterizza la metodologia orientata agli oggetti è la possibilità di estendere una classe mantenendo l'incapsulamento. Se si ha una classe alla quale voglio aggiungere delle funzionalità non ho bisogno di rompere la capsula, aggiungere le funzionalità e ricostruire una nuova capsula. Ciò risulterebbe una cosa terribile: nel momento in cui apro una capsula per ricostruircene un'altra intorno, do la possibilità di accedere al suo contenuto. Quello che è possibile fare con i linguaggi *object-oriented* è di prendere una classe ed estenderla, cioè aggiungerci delle funzionalità in una nuova capsula, senza aprirla. Questo viene fatto attraverso il concetto di sottoclasse, o estensione di una classe.

Sottoclassi e ereditarietà



Vediamo un esempio. Vogliamo estendere un cerchio aggiungendo un colore alla sua circonferenza. Immaginando che i cerchi possano essere disegnati sullo schermo; non hanno solo un centro e un raggio ma hanno anche un colore della circonferenza. Potremmo definire una nuova classe che oltre ad x, y e r abbia anche un colore. Ciò è ovviamente possibile, ma non è un buon modo di procedere. Un modo corretto per risolvere questo problema è di estendere il cerchio. Creiamo una nuova classe, chiamata *GraphicCircle* come una classe che estende *circle*.

Sottoclassi e ereditarietà



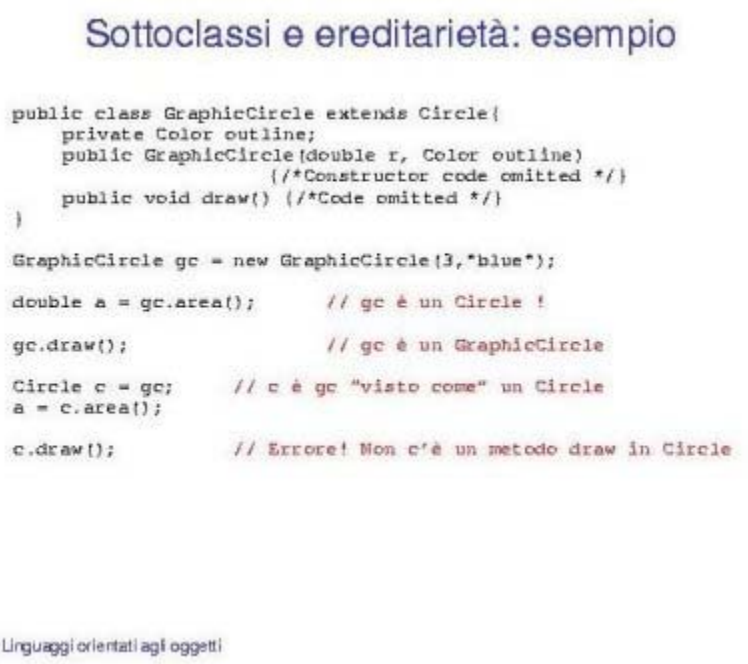
Estendere vuol dire che mantiene tutto ciò che c'è in *circle*, aggiungendo alcune nuove proprietà. Un *GraphicCircle* è tutto quello che è un *circle*, sia le variabili (x,y,r) sia i metodi (*circle*, Area, Circonf), più un nuovo dato che io ho chiamato *outline*, che è di tipo Colore, più un metodo *GraphicCircle* che è di tipo costruttore. Costruirà un oggetto

della classe *GraphicCircle*, che avrà come parametri un raggio e un colore, che saranno appunto il raggio e il colore dell'oggetto creato. Abbiamo infine un altro nuovo metodo chiamato *draw* che serve per disegnare questo cerchio su un piano cartesiano.

Sottoclassi e sottotipi

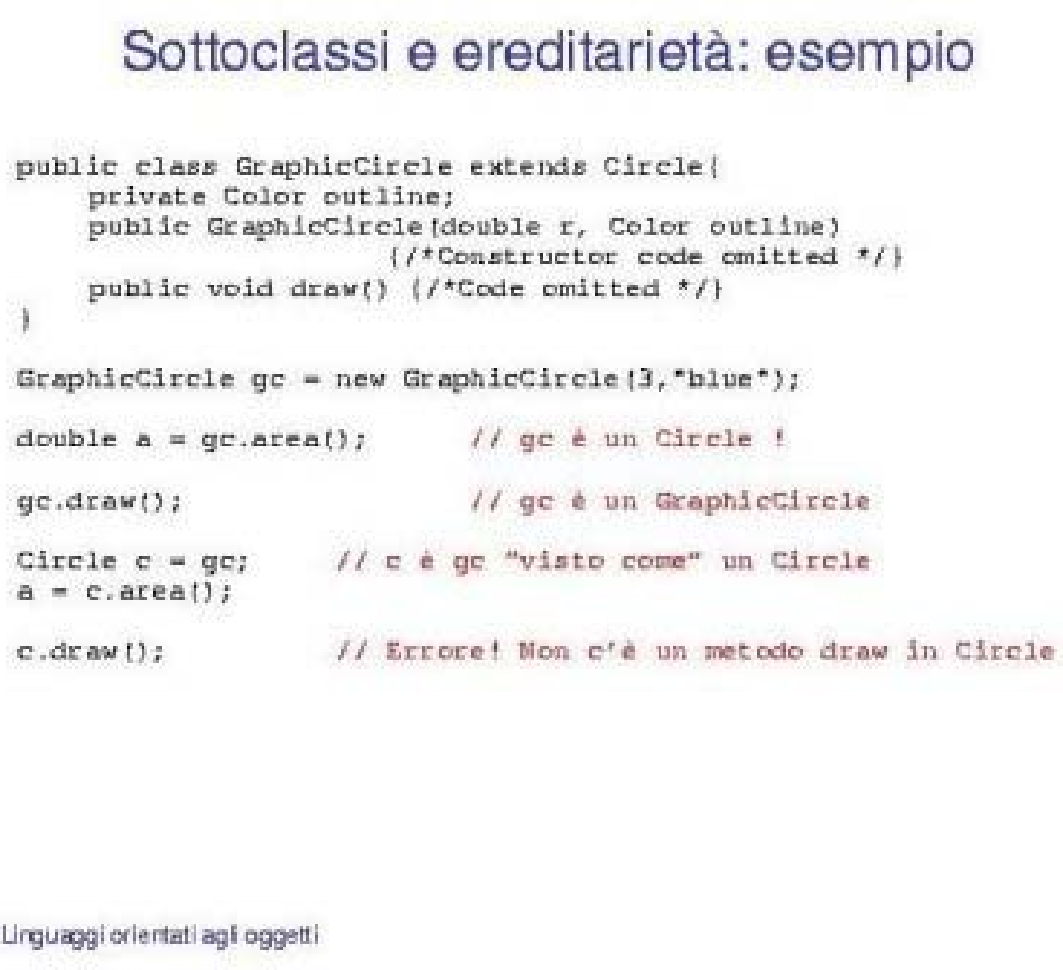
Osservate come *GraphicCircle* è definita come una estensione di *circle*: *GraphicCircle* è una sottoclasse di *circle*. In gergo si dice che eredita tutti i metodi pubblici di *circle*; ciò vuol dire che un oggetto *GraphicCircle* sa rispondere non solo al metodo *draw*, ma sa rispondere anche al metodo area e al metodo circonf. Questo non è esplicitato, ma discende dal fatto che *GraphicCircle* è un'estensione di *circle*. Nella prossima trasparenza c'è il riassunto di quanto abbiamo appena detto: una sottoclasse è definita come una estensione, ogni oggetto della classe *GraphicCircle* è anche un oggetto della classe *circle*. Potremmo quindi usare un cerchio grafico ogni volta che viene richiesto un semplice cerchio, semplicemente ci dimentichiamo del fatto che ha anche caratteristiche in più. Questa è una dote di grandissima flessibilità.

Sottoclassi e ereditarietà: esempio



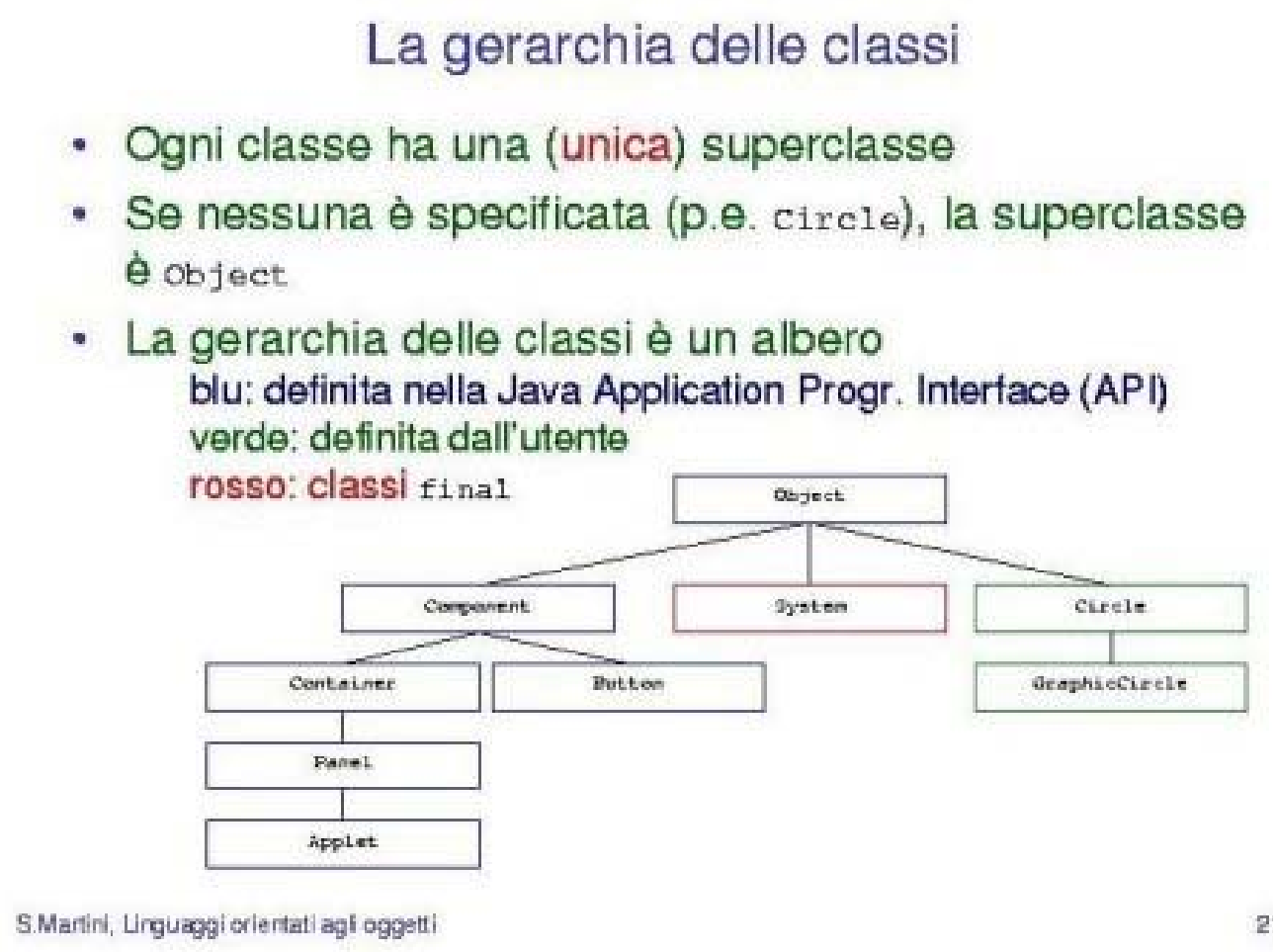
Vediamo un esempio di sottoclassi un po' più esteso. Viene mantenuta la stessa struttura, tra parentesi ho lasciato la definizione della classe *GraphicCircle*, che è la stessa che abbiamo discusso precedentemente. Innanzitutto possiamo creare un cerchio grafico chiamato gc. Per far questo ci serviamo del suo costruttore *GraphicCircle* e lo definiamo di raggio 3, di colore blu e centro nell'origine degli assi. Possiamo ora mandare dei messaggi a gc. Ad esempio, inviando un messaggio del tipo gc.area, gc riesce a rispondere in quanto appartiene anche alla classe *circle*, essendo un oggetto di una sottoclasse di *circle*. Naturalmente, essendo un *GraphicCircle*, saprà rispondere anche al metodo *draw* che lo disegnerà nello schermo.

Sottoclassi e ereditarietà: esempio



Possiamo definire c come un altro nome per gc, ma portando gc ad essere soltanto un cerchio e non un cerchio grafico. Pertanto c potrà ricevere dei messaggi soltanto come cerchio, risponderà per esempio al metodo area, ma non al metodo *draw*. Se viene chiesto a c di eseguire il metodo *draw* viene generato un errore, che sarà segnalato dal traduttore, dal controllore sintattico del linguaggio, prima che il programma possa venire eseguito e segnalerà appunto che *draw* non è un metodo che c può eseguire. Quindi c, sebbene sia un altro nome di gc, viene visto soltanto come cerchio, perché dichiarato nella classe *circle* e quindi non riesce a vedere alcuna proprietà o metodo di *GraphicCircle*.

La gerarchia delle classi

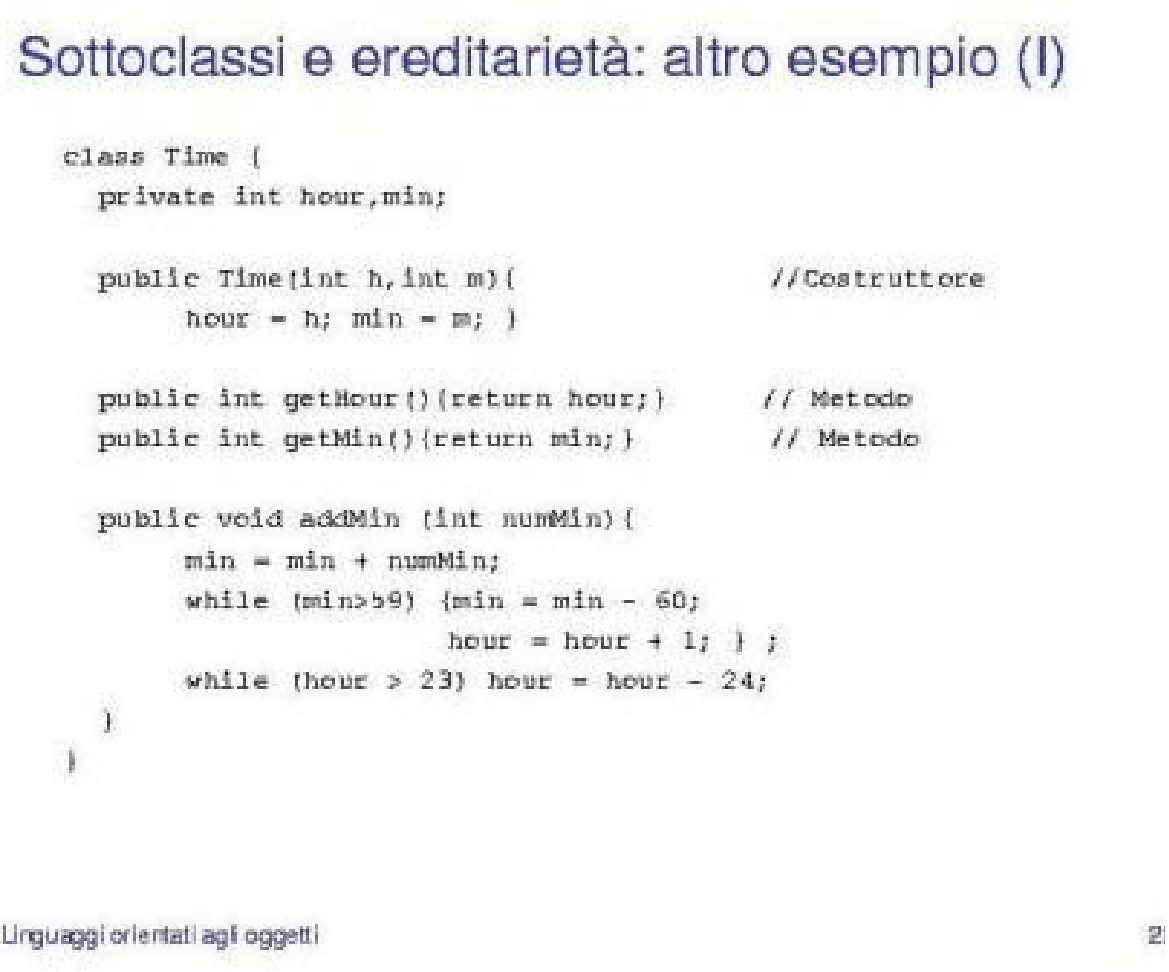


Abbiamo visto questo semplice esempio di sottoclasse (la classe *GraphicCircle* sottoclasse di *circle*), spostiamoci ora ad un contesto più ampio. Le classi in Java hanno una struttura molto elegante perché ogni classe può avere un'unica superclasse. Questo fa sì che la gerarchia delle classi in Java sia un albero, dove ogni classe che non ha alcuna sopraclasse (nel nostro esempio era la classe *circle*) viene considerata dal linguaggio come una sottoclasse della classe *object*, che è una generica classe definita dal linguaggio e che costituisce quindi la radice dell'albero della gerarchia tra classi.

La gerarchia delle classi

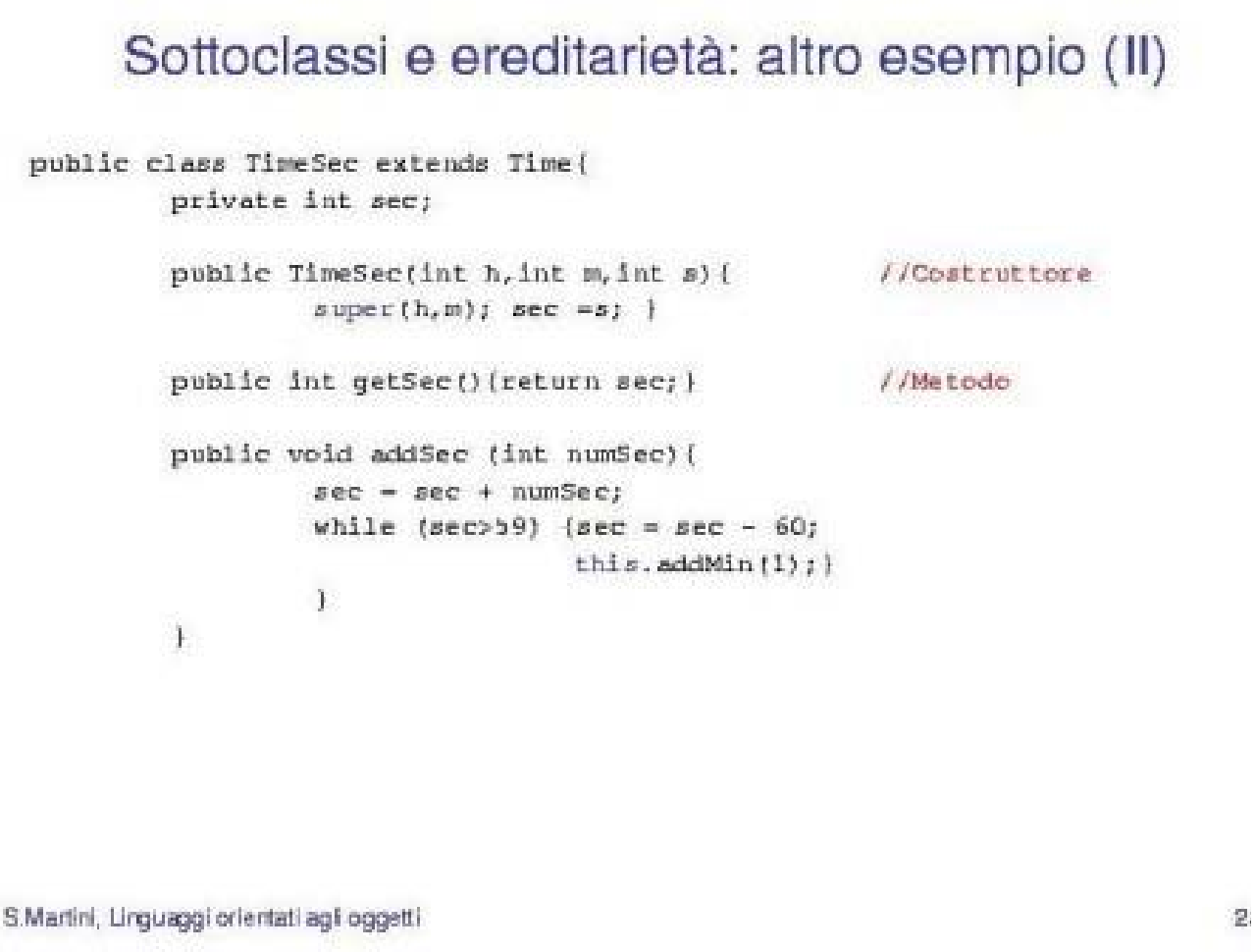
La gerarchia delle classi è un albero; abbiamo delle gerarchie che sono quelle definite dall'utente ad esempio *circle*, che è una sottoclasse di *object* per definizione, poi *GraphicCircle* che è una sottoclasse di *circle* perché così l'abbiamo definita. Ci sono poi delle gerarchie che fanno parte della definizione del linguaggio, ad esempio *Component* che ha come sottoclassi *Button* (bottone che serve per realizzare delle interfacce utente) oppure *Container* e così via. Ci sono poi delle classi che sono definite finali e sono le classi che non possono essere estese, cioè si fissa al momento della loro definizione che non possono avere sottoclassi. La struttura della gerarchia è abbastanza complicata, questa che vedete è una porzione dell'albero estremamente ridotta, ma quest'albero è estensibile dall'utente.

Sottoclassi e ereditarietà: altro esempio (I)



Sulle trasparenze che trovate nel materiale, è descritto un altro esempio che io non sto a discutere in dettaglio. Abbiamo una classe *Time* che introduce dei dati privati, un tempo in ore e minuti. Avremo quindi un metodo costruttore che assegnerà per ogni nuovo oggetto una certa ora e minuti e dei metodi che permettono di leggere il campo ora o il campo minuti e che consentono di aggiungere un certo numero di minuti all'oggetto. Capite che per aggiungere i minuti non basta sommare i numeri che si vogliono aggiungere ai minuti dell'oggetto, bisogna naturalmente fare tale somma modulo 60, cioè occorrerà aumentare l'ora nel caso in cui il valore dei minuti sia diventato maggiore di 60.

Sottoclassi e ereditarietà: altro esempio (II)



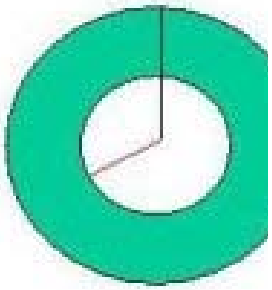
Questa classe può essere estesa, in questo caso estendiamo *Time* con una nuova classe: *TimeSec*. Le istanze sono ancora dei tempi, dove abbiamo indicati, oltre ai minuti e alle ore, anche i secondi. In tal caso avremo un nuovo costruttore che avrà in ingresso un certo valore per le ore, un certo valore per i minuti e uno per i secondi ed inizializza un nuovo oggetto con quei valori dati in *input*. Abbiamo usato una parola importante in Java, che è super: permette di invocare il costruttore di quella classe. Abbiamo anche in questo caso, come per *Time*, un metodo per leggere il tempo, ed un nuovo metodo che aggiunge i secondi.

Estensibilità: ridefinire un metodo

Estensibilità: ridefinire un metodo

- Una sottoclasse può anche ridefinire un metodo

```
public class Corona extends Circle{
    private double raggiointerno;

    public Corona(double r1, r2) {/*code omitted */}

    public void area() {/* codice per l'area della corona */}
}

Corona dc = new Corona(3, 5);

double a = dc.area();        // area della corona !

Circle c = dc;    // c è dc "visto come" un Circle
```



S.Martini, Linguaggi orientati agli oggetti 2

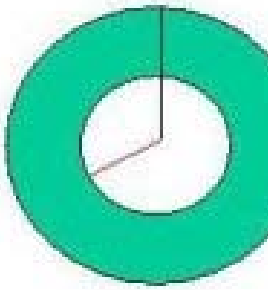
L'esempio del tempo con i secondi lo potete guardare in dettaglio da soli, più interessante è discutere un altro concetto che è la ridefinizione di un metodo, che garantisce l'estensibilità delle classi. Fino adesso abbiamo definito una sottoclasse per sola estensione, aggiungendo informazione, in questa aggiunta una sottoclasse può anche ridefinire un metodo. Vediamo questo esempio in cui definiamo una classe Corona (intendiamo la corona circolare) che estende la classe *circle*. Una corona è costituita da due cerchi concentrici, aggiungiamo quindi un altro parametro che sarà il raggio interno.

Estensibilità: ridefinire un metodo

Estensibilità: ridefinire un metodo

- Una sottoclasse può anche ridefinire un metodo

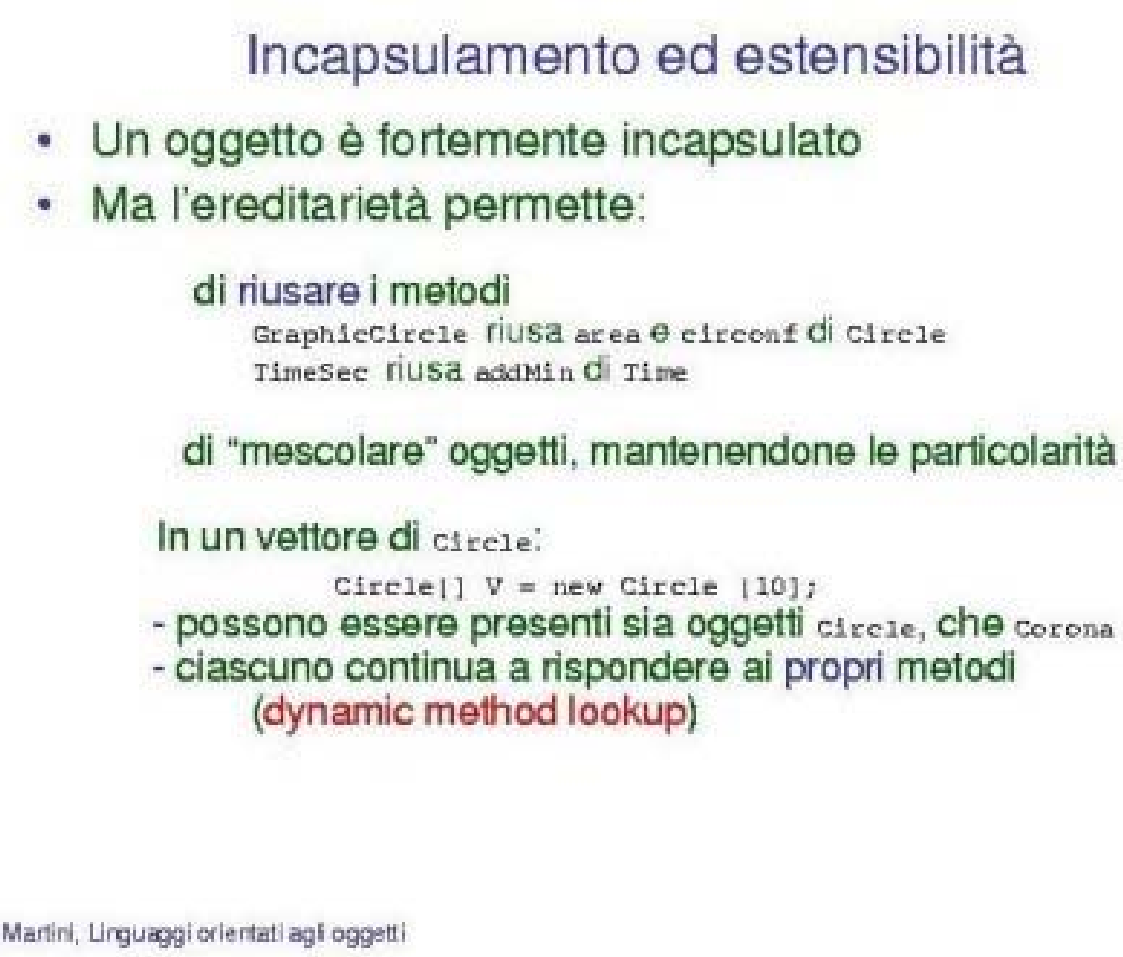
```
public class Corona extends Circle{
    private double raggiointerno;

    public Corona(double r1, r2) {/*code omitted */}

    public void area() {/* codice per l'area della corona */}
}

Corona dc = new Corona(3, 5);

double a = dc.area();        // area della corona !

Circle c = dc;    // c è dc "visto come" un Circle
```



S.Martini, Linguaggi orientati agli oggetti 2

Abbiamo il costruttore, chiamato anch'esso Corona, che avrà in ingresso due raggi: il primo interno r1 e il secondo esterno r2. Abbiamo poi il metodo area che ci restituisce l'area della corona. Questo metodo era anche definito nel *circle*, in tal caso lo stiamo ridefinendo. Ogni linguaggio orientato agli oggetti permette la ridefinizione dei metodi. In questo modo gli oggetti della nuova sottoclasse corona rispondono al metodo area definito dentro la sottoclasse e non quello definito nella sopraclasse. Riprendendo l'esempio abbiamo la costruzione di una nuova corona dc con raggio interno 3 e raggio esterno 5; dc è un oggetto della sottoclasse Corona della classe *circle* e quindi risponde al metodo area, ma il metodo invocato non è quello della sopraclasse *circle*, ma il metodo della sottoclasse Corona, cioè viene calcolata l'area della corona.

Incapsulamento ed estensibilità



Concludiamo con un po' di osservazioni topologiche. Un oggetto è fortemente incapsulato, ma l'ereditarietà, cioé la possibilità di definire sottoclassi, di ereditare i metodi della sopraclasse nella sottoclasse, consente di riusare i metodi: *GraphicCircle* riusa area e circonf che erano definiti in *circle*; *TimeSec* riusa addMin che era definito in *Time*. Oltre a consentire il riuso, viene ammessa anche la ridefinizione, riuscendo a mescolare oggetti mantenendone le particolarità. Ad esempio possiamo costruire un vettore di cerchi, chiamato cv, dove possono esserci oggetti che siano sia cerchi che corone. Una cosa importante su cui possiamo insistere poco, perché è un concetto abbastanza avanzato, è che, nonostante facciano tutti parte di un vettore definito come *circle*, ciascuno di questi oggetti risponderà però ai propri metodi: un oggetto cerchio risponde ad Area come area del cerchio, una corona risponderà invece con l'area della corona. Questa proprietà molto importante viene chiamata nel gergo orientato agli oggetti *look up* dinamico dei metodi (ricerca dinamica dei metodi).

Dynamic method lookup

Dynamic method lookup

- Immaginiamo che in `TimeSec` che estende `Time`
  `print()` di `Time` sarà ridefinito in `TimeSec`
  per stampare anche i secondi

  In un vettore di oggetti `Time`, alcuni possono essere `TimeSec`

```
Time[] time_vect = new Time[10];

time_vect[1] = Time(20,30);
time_vect[5] = TimeSec(21,30,14);

for(int i=0; i<10; i++)
      time_vect[i].print();
```

  Ogni oggetto usa il proprio metodo `print` !
- La selezione avviene in modo dinamico
  durante l'esecuzione

S.Martini, Linguaggi orientati agli oggetti 26

Su questa trasparenza insiste su questo argomento, come al solito i dettagli li potete vedere più comodamente da soli, anche perché riprende le classi *TimeSec* e *Time*. Immaginiamo che dentro queste classi siano stati inseriti anche dei metodi *Print*, naturalmente in *Time* verranno stampati solo ore e minuti, in *TimeSec* ore, minuti e secondi. Preso un vettore di oggetti *Time*, alcuni dei suoi elementi possono essere *TimeSec*; nel caso volessimo stampare quel vettore, vorremo che i *Time* vengano stampati come *Time* e i *TimeSec* come *TimeSec*. Ogni oggetto userà automaticamente il proprio metodo e la selezione avviene in modo dinamico durante di esecuzione, ecco perché viene chiamata ricerca dinamica dei metodi, questa infatti non può avvenire a tempo di compilazione, ma durante l'esecuzione.

Conclusioni

Conclusioni

Classi e sottoclassi
Oggetti e ereditarietà

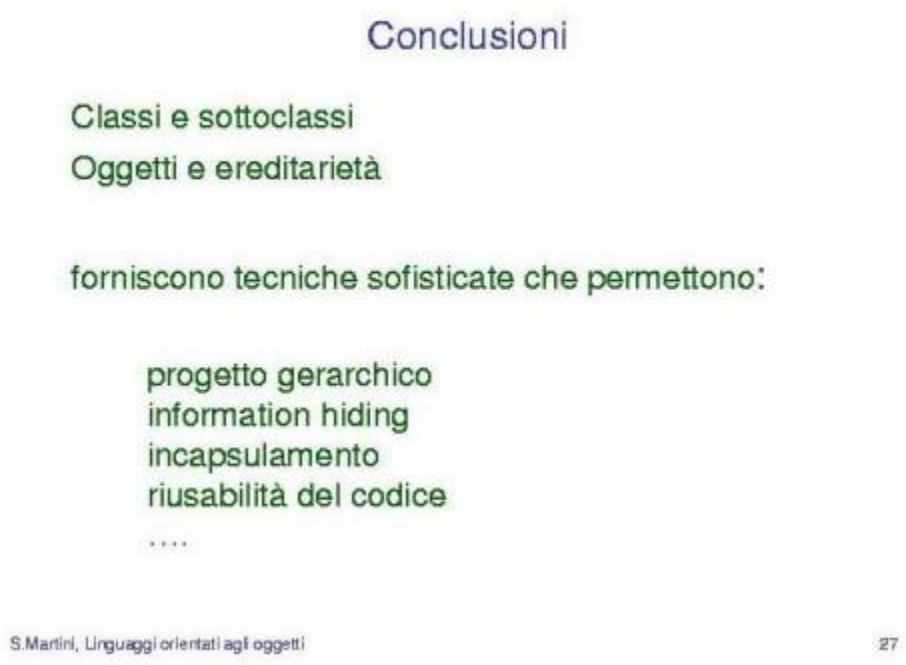
forniscono tecniche sofisticate che permettono:

progetto gerarchico
information hiding
incapsulamento
riusabilità del codice
....

S.Martini, Linguaggi orientati agli oggetti 27

In conclusione i linguaggi orientati agli oggetti, Java in particolare, sono uno strumento molto flessibile che permette non solo la costruzione di programmi complessi, ma sono uno strumento metodologico che insegna a strutturare soluzioni e, in questo ruolo sono utili anche al di fuori dell'informatica, intesi come una tecnologia per costruire sistemi complessi.

Conclusioni



Possiamo concludere questa lezione con un po' di osservazioni metodologiche. Abbiamo visto come i concetti fondamentali della programmazione *object oriented* siano quelli di classe e di sottoclasse, cioé la possibilità di definire gerarchie di classe e quindi di avere oggetti che ereditano proprietà da classi superiori. I costrutti linguistici forniscono delle tecniche molto sofisticate che permettono di raggiungere all'interno di un progetto quelle caratteristiche che avevamo visto essere molto importanti per la progettazione di sistemi complessi. In un progetto gerarchico, le classi permettono di definire gerarchie all'utente, consentono il nascondimento dell'informazione, sia perché le classi stesse sono un meccanismo di incapsulamento, sia perché si possono utilizzare i due parametri *private* o *public* in modo tale da garantire una totale o una parziale *information hiding*. Un progetto gerarchico consente inoltre di riutilizzare il codice già scritto in modo pulito, senza doverlo stanziare ogni volta in contesti diversi; naturalmente qui gioca un ruolo essenziale il concetto di ereditarietà del linguaggio. Potremmo elencare ancora tante altre qualità che non abbiamo visto.

Bibliografia

Introduzione

Installazione e configurazione del software

D.P. Curtin e altri; *Informatica di base, seconda edizione;* 2002McGrawHill
Autori Vari; *Collana NO PROBLEM e MINI NO PROBLEM;* McGrawHill

Elementi di linguaggi di programmazione e Tecnologie di programmazione

R. Sethi; *Linguaggi di programmazione;* 1994Zanichelli
J. C. Mitchell; *Concept in programming languages;* 2003Cambridge Univ. Press

Approfondimenti

Introduzione alle basi di dati

Atzeni, Ceri, Paraboschi, Torlone; *Basi di dati;* 2002McGrawHill
E.F. Codd; *A relational model for large shared data banks. vol. 13, n. 6, pag. 377-387;* 1970Communications of the ACM
M. R. Irwin, C. N. Prague, J. Reardon; *Microsoft Access 2002 Bible Gold Edition;* 2001Wiley & Sons
Microsoft Corporation; *Manuale in linea di Microsoft Access;* Microsoft Corporation

Aggiornare un sistema: alcuni argomenti

M. Soper, P. Norton; *Upgrade PC, manuale completo;* 2002Jackson Libri
Forum su Windows di ZDNet; **http://www.zdnet.it**
PCWORLD; **http://www.pcworld.com**

Linguaggi orientati agli oggetti

Judy Bishop; *Java Gently, Corso Introduttivo, Seconda edizione;* 1999Addison-Wesley
David Flanagan; *Java in a Nutshell, 4th Edition;* 2002O'Reilly
Sito ufficiale di Java; **http://java.sun.com**

Software Libero

Cos'è il Software Libero?; Free Software Foundation 1996; **http://www.it.gnu.org/philosophy/free-sw.it.html**
Il Progetto GNU; Richard Stallman 1998; **http://www.it.gnu.org/gnu/thegnuproject.it.html**
Classificazione del software libero e non; Free Software Foundation 1996;
http://www.it.gnu.org/philosophy/categories.it.html
Free as in Freedom; Richard Stallman 2002; **http://www.oreilly.com/openbook/freedom/**
Associazione Software Libero: Cos'è il Software Libero?; F. Potortì 2002;
http://www.softwarelibero.it/documentazione/softwarelibero.shtml
Materiale sul software libero 1; A. Rubini; **http://www.softwarelibero.it/GNU/**
Materiale sul software libero 2; A. Rubini; **http://www.linux.it/GNU/**
Materiale sul software libero 3; A. Rubini; **http://www.prosa.it/philosophy/**
Glossario Essenziale del Software Libero; Associazione Software Libero;
http://www.softwarelibero.it/documentazione/glossario.shtml
Un'economia del software libero: vantaggi e pericoli; Robert J. Chassell 2001;
http://www.softwarelibero.it/altri/economia-sl.shtml
I testi principali sul software libero; Associazione Software Libero;
http://www.softwarelibero.it/documentazione/intermedio.shtml
Altri documenti sul software libero; Associazione Software Libero;
http://www.softwarelibero.it/documentazione/avanzato.shtml
Appunti di Informatica Libera; D.Giacomini; **http://a2.swlibero.org/**
Trappola nel Cyberspazio; R. Di Cosmo; **http://www.pps.jussieu.fr/~dicosmo/Piege/PiegeIT.html**
Hijacking the World. The Dark Side of Microsoft.; R. Di Cosmo; **http://www.00h00.com/direct.cfm?titre=7201990401**
Cos'è il Software Libero?; Free Software Foundation 1996; **http://www.it.gnu.org/philosophy/free-sw.it.html**
Cos'è il Software Libero?; Free Software Foundation 1996; **http://www.it.gnu.org/philosophy/free-sw.it.html**
Breve Storia del Software Libero.; OTE, Osservatorio Tecnologico per la Scuola. Raffaele Meo.;
http://www.osservatoriotecnologico.net/software/opensource/meo_brevestoria.htm
GNU: Free Software Foundation; **http://www.fsf.org/home.it.html**
Free Software Foundation Europe; **http://www.fsfeurope.org/index.it.html**
ASSOLI: Associazione Software Libero; **http://www.softwarelibero.it/**
GNUTEMBERG: Progetto Gnutemberg!; **http://www.gnutemberg.org/**
ILS: Italian Linux Society; **http://www.linux.it//**
Software Libero nella Scuola; **http://scuola.linux.it/**
Linux a Scuola; **http://www.linuxascuola.it/**
OpenOffice; **http://www.openoffice.org/**
Debian Linux; **http://www.debian.org/**
Mandrake Linux!; **http://www.mandrakelinux.com/it/**

SUSE Linux; **http://www.suse.de/it/**
RedHat Linux; **http://www.redhat.com/**
Bononia; **http://www.bononia.it/**

Glossario

Adsware: *Software* liberamente scaricabile dalla rete ma proprietario e a Sorgente chiuso. La ditta produttrice viene retribuita da inserzioni pubblicitarie che devono essere visibili durante l'utilizzo.

Algebra relazionale: Insieme di operatori su relazioni (ovvero tabelle) che producono come risultati altre relazioni (o tabelle). Questi operatori possono essere composti per formare espressioni complesse analogamente a quello che possiamo fare con gli operatori aritmetici.

Ambiente: Collezione delle associazioni tra nomi e valori durante l'esecuzione di un programma. L'ambiente è dinamico se nuovi nomi e locazioni possono venire creati durante a tempo d'esecuzione. I linguaggi di programmazione moderni sono tutti dotati di ambiente dinamico.

AMD: Produttore di Processori compatibili con quelli della ditta *Intel*. È concorrente di *Intel* ma la scelta di mantenere la piena compatibilità fa sottostare AMD a tutte le scelte di *Intel* in materia di compatibilità *software* e *hardware*.

ASCII: Codice a 7 bit per la rappresentazione interna dei caratteri.

Assemblatore: Traduttore che trasforma un programma scritto in linguaggio *assembler* nel corrispondente in linguaggio macchina.

Backdoor: Porta di ingresso non autorizzata per accedere a sistemi o a dati. Può risultare presente una *backdoor* o per volontà di chi ha creato il programma o messa ad arte da chi ha fatto instrusione in un sistema per poterci rientrare. L'unico modo per poter controllare o far controllare se nei propri programmi siano presenti *backdoor* è di poter accedere ai programmi sorgenti e rigenerare a partire da essi il codice eseguibile.

Backup: Procedura di salvataggio del *file system* (tutto o parte), per permettere un ripristino dei dati in caso di danneggiamento dei dischi.

Base di dati: Insieme organizzato di dati utilizzati per il supporto allo svolgimento delle attività di una organizzazione (azienda, ufficio, eccetera). In senso più tecnico, una base di dati identifca l'insieme dei dati gestiti da un programma DBMS (*Database Management System*).

BIOS: Un semplice programma residente nella memoria centrale non volatile (ROM) che provvede all'interfaccia tra il sistema operativo e l'*hardware*; in particolare, lancia il caricamento (*boot*) del Sistema Operativo.

Boot: Più correttamente *bootstrapping*, la procedura di caricamento di un Sistema Operativo. Operazione di caricamento del nucleo del sistema operativo e dei programmi residenti. Viene fatto il *boot* del sistema operativo al momento dell'accensione del *computer* o a ogni riattivazione (*reset*) fisico.

Brevetto sul Software: Esiste negli Stati Uniti, forti *lobby* vorrebbero fosse permesso anche nell'Unione Europea. Si vuole concedere di poter registrare brevetti su algoritmi e procedimenti informatici. È fortemente osteggiato dai sostenitori del *software* libero perché ne limiterebbe fortemente la diffusione e la crescita.

Browser: Programma che permette la navigazione in Internet agendo come *client* nel protocollo HTTP e interperatndo le pagine scritte in HTML. Programma per navigare nel Web. Ne esistono molte versioni: *Internet Explorer*, *Netscape*, *Mozilla*, *Opera*, *Konqueror*...

CAD: (*Computer Aided Design*) strumenti di supporto alla progettazione del prodotto che tipicamente permettono la realizzazione e la modifica veloce di disegni tecnici (anche tridimensionali). Sono usati prevalentemente in ambito meccanico, ma esistono anche applicazioni in altri settori.

Catalogo: (in un DBMS) Porzione della base di dati che contiene una descrizione centralizzata dei dati e che può essere utilizzata dalle varie applicazioni che accedono ai dati.

Cavalli di Troia: Funzionalità malefiche nascoste in programmi che appaiono utili, innocui. L'unico modo per poter controllare o far controllare se nei propri programmi siano presenti cavalli di troia è di poter accedere ai programmi sorgenti e rigenerare a partire da essi il codice eseguibile.

CD-Live: CD dal quale è possibile fare *boot* ad un *computer*. Contiene un intero sistema operativo che lavora senza modificare il contenuto della macchina. (Attenzione. Normalmente i *CD-Live* consentono l'accesso ai dati registrati nel *computer* ospite: per mezzo di azioni volontarie, e.g. comandi di cancellazione, si possono comunque danneggiare i contenuti.).

Chiave: (nelle basi di dati relazionali) Insieme minimale di attributi che permette di identificare univocamente le ennuple di una relazione (ovvero le righe di una tabella).

Classe: Costrutto orientato agli oggetti che definisce il comportamento delle proprie istanze. In genere contiene la definizione dei metodi condivisi dalle istanze.

Codice Eseguibile: Forma del programma da eseguire, illeggibile. È impossibile quindi comprenderne il reale contenuto e il funzionamento. Quando acquistate o scaricate da rete dei programmi proprietari vi viene dato solamente il codice eseguibile dei programmi.

Codice Sorgente: Forma del programma leggibile e modificabile. I programmatori scrivono i programmi in questa forma usando un linguaggio di programmazione (come per esempio C, C++, Pascal, Fortran, Cobol, Java, ...). Se non avete il codice sorgente di un programma non potete capire quali azioni in realtà il programma compie. Se avete il codice sorgente potete modificare il programma per rimediare a errori o problemi di sicurezza quando ve ne fosse necessità, altrimenti no.

Compilatore: Traduttore che trasforma un programma scritto in un linguaggio ad alto livello in uno corrispondente scritto in un linguaggio di più basso livello (per esempio in linguaggio *assembler*, o direttamente in linguaggio macchina).

Compressione: Tecnica che permette di mantenere il contenuto informativo dei dati riducendo però lo spazio di memoria occupato.

Condizionale: Costrutto di controllo che permette la scelta tra alcune porzioni di programma in dipendenza dal verificarsi di una condizione. Un condizionale è spesso espresso mediante la sintassi *if-then-else*.

Configurazione: Fase dell'installazione di un programma durante la quale si determinano alcuni parametri, a seconda delle caratteristiche *hardware* e *software* del sistema in uso.

Copyleft: v. **Permesso d'Autore** .

Copyright: v. **Diritto d'Autore** .

Cracker: Delinquente informatico. Intrude, diffonde programmi malefici come virus e vermi (*worm*), modifica, distrugge dati o solamente viola la *privacy* altrui. I *mass media* generalmente confondono erroneamente questo termine con *Hacker* (v.).

Data Mining: Il *Data Mining* consiste nell'estrazione di informazioni contenute in modo implicito e semi-strutturato in grandi quantità di dati. A differenza dei DBMS e dei DSS, il *Data Mining* non si basa su interrogazioni specifiche ma

sull'estrapolazione di relazioni significative.

DBMS: Programma che gestisce (nel senso detto prima) grandi insiemi di dati, in modo persistente e condiviso, in modo tale da garantire privatezza, affidabilità ed efficienza.

Debian: La più libera delle grandi distribuzioni *Linux*. In *Debian* anche tutta la gestione dei pacchetti *software*, degli aggiornamenti e delle scelte strutturali è effettuata con gli stessi metodi dello sviluppo del *software* libero. Per statuto costitutivo in *Debian* non c'è alcuna società commerciale coinvolta e tutta la distribuzione è completamente libera.

Demolinux: Distribuzione *Linux* creata da Roberto di Cosmo per diffondere il *software* libero nella scuola e nella pubblica amministrazione francese. Non necessita di installazione. Viene diffuso con la tecnica dei *CD-Live*.

Diritto d'Autore: È la norma che tutela le opere originali della creatività umana. Testi, musica e anche i programmi (il *software*) sono giustamente tutelati dalle leggi sul diritto di autore. Spetta all'autore decidere con quali modalità desidera che la sua opera venga tutelata. L'autore può decidere di volere un compenso per l'utilizzo o no, di concedere che l'opera venga successivamente modificata o no, che la redistribuzione dell'opera modificata debba prevedere un compenso per l'autore originario, non debba prevedere compenso neanche per il modificatore, che debba in ogni caso contenere l'indicazione dell'autore originario o che possa non essere indicato. Il diritto d'autore tutela l'opera in quanto tale, non i concetti in essa espressi. Chiunque con carta bianca, spartito bianco o *file* nuovo scriva da zero senza plagio una sua opera su questa ha il diritto di autore. Il diritto delle opere pubblicate oggi si estingue in Italia a 75 anni dalla morte dell'autore (o dell'autore morto per ultimo se l'opera è cofirmata).

Distribuzione: Stuttura organizzata di un sistema operativo con tutti i programmi di corredo. Le distribuzioni forniscono il servizio di aggiornamento di tutti i programmi contenuti. Vengono gestite automaticamente anche le dipendenze fra programmi (per installarne uno c'è la necessità della presenza di un altro) o l'incompatibilità (per esempio fra molteplici implementazioni della stessa funzionalità).

Documento elettronico: Documento la cui rappresentazione fisica è in forma di bit all'interno di un sistema informatico.

Driver: Il particolare dispositivo *software* che controlla la comunicazione con una periferica è detto *driver* della periferica.

Eccezzione: Condizione eccezionale che si verifica durante l'esecuzione di un programma. I linguaggi di programmazione moderni hanno costrutti che permettono di gestire un'eccezione, espressi spesso mediante la sintassi *try-catch*.

Editore di testi: Programma (detto anche di videoscrittura) che permette di scrivere, modificare, formattare documenti di vario genere.

Ereditarietà: Meccanismo mediante il quale gli oggetti di una sottoclasse possono usare (rispondono a) i metodi definiti nella superclasse.

Ext2, Ext3: sono i tipi di ***file system*** tipici di *Linux*. *Ext2* è quello più storico. Sono *file system* moderni. Entrambi mantengono i dati ordinati e non hanno alcuna necessità di deframmentazione anche se usati per anni da migliaia di utenti. *Ext3* è consigliabile per *server* con grandi capacità di memorizzazione perché riesce a ripristinarsi molto velocemente anche se il sistema non è stato disattivato correttamente (guasto, mancanza di tensione, eccetera...)

FAT: *File access table*, organizzazione del *file system* di *Windows*. Esiste nella versione FAT16 (*Windows* 3.1 e prime versioni di 95) e FAT32 (le successive versioni). *Windows* 2000 e XP supportano anche il più moderno NTFS.

FAT (File Allocation Table): Il *file system* di tutti i sistemi *Windows* non professionali. È molto primitivo. Ammette solo nomi brevi (11 caratteri+3 di estensione) anche se da *Windows* 95 è stato introdotto un meccanismo chiamato *Joliet* (molto contorto) per gestire nomi più lunghi. I dati non vengono tenuti ordinati e quindi le prestazioni decadono durante l'utilizzo. L'unico modo per ripristinare l'efficienza è l'uso del deframmentatore (*defrag*) che riordina i dati. Se

durante l'opera del *defrag* qualcosa va storto (e.g. manca corrente) si potranno avere perdite di dati.

File System: Componente di un sistema operativo che permette l'organizzazione delle informazioni in strutture logiche, dette file, e che gestisce le operazioni di allocazione di memoria di massa necessarie per memorizzare i file. Viene chiamato *file system* lo strumento *software* che consente di dare all'utente l'astrazione di *file*(archivio) e *directory* (cartelle) all'interno di un disco. Esistono vari modelli di *file system* con caratteristiche differenti: un *computer* per poter leggere un disco deve conoscere come i dati siano stati memorizzati su di esso deve quindi avere il supporto per il tipo di *file system* usato per la registrazione. Esempi di *file system* sono: *ext2*, *ext3* tipici di *Linux*, FAT, NTFS usati da *Microsoft*, ISO9660 usato sui CD-ROM.

File: Struttura logica, identificata da un nome, per la memorizzazione di dati. È accessibile all'utente tramite opportuni comandi del sistema operativo (apertura, lettura, scrittura, chiusura del file).

Firmware: L'insieme dei programmi che risiedono nella memoria centrale non volatile (ROM). Uno dei suoi componenti più importanti è il BIOS.

Foglio elettronico: (o *spreadsheet*) Programma che permette il trattamento e l'analisi di dati numerici mediante tabelle (o griglie) nelle quali è possibile operare sulle singole caselle, sulle colone, sulle righe o su più caselle collegate fra loro.

Formati Aperti: Sono formati le cui specifiche sono disponibili pubblicamente. Disponendo di un formato aperto è possibile sapere esattamente quali dati sono contenuti e scrivere applicazioni capaci di gestire dati di tale formato. La definizione si applica ai formati di registrazione di *file* e ai protocolli di comunicazione. Sono formati aperti il *Postscript*, il Pdf, il testo ASCII, l'html. Per i protocolli le specifiche di IP, *TCP*, UDP, http, nntp e di tutti i principali protocolli di Internet sono pubblici.

Formati Chiusi: Sono formati dei quali non è possibile conoscere le specifiche. Non è quindi possibile conoscere quali dati in realtà contengano/trasportino né è possibile scrivere programmi che gestiscano questi dati. Sono chiusi i formati di registrazione di *Microsoft Office* (doc, xls, ppt...).

FreeBSD: È al tempo stesso un nome di un sistema operativo (completo di *kernel* e applicazioni) e della sua licenza d'uso. Sviluppato a partire da una versione resa libera dalla Università di California a *Berkeley*, *FreeBSD* viene divulgato come *software* libero ma con la possibilità per aziende di inglobare i programmi all'interno dei loro prodotti proprietari chiusi. È un ottimo prodotto ma con un futuro meno sicuro di *GNU-Linux*: funzionalità interessanti oggi accessibili come *software* libero potranno domani essere aggiornate solo in prodotti proprietari.

Free Software: v. **Software Libero** .

Freeware: Il *software* in assoluto più pericoloso. È un *software* proprietario a sorgente chiuso divulgato gratuitamente. Quasi sicuramente c'è un secondo scopo. Non sappiamo cosa contenga, spesso non si sa chi l'abbia costruito. Può servire per invogliare gli utenti a usare un determinato prodotto prima che diventi a pagamento, può servire per svolgere concorrenza sleale per esempio fornendo il *browser* per il Web come *freeware* e distribuendolo integrato coi propri sistemi operativi così che nessuno ne vada a cercare altri.

Funzione: A seconda dei linguaggi può essere sinonimo di procedura, oppure essere una procedura che restituisce un valore.

Gerarchia delle classi: La struttura ad albero (in Java) che si ottiene mediante la definizione di sottoclassi. La radice dell'albero è la classe di sistema *Object*.

Gnome: Interfaccia grafica libera sviluppata come progetto *GNU*. Molto potente e versatile è nata in un ambiente di informatici: inizialmente c'era più attenzione all'efficienza rispetto alla facilità di utilizzo e alla usabilità per utenti alle prime armi. Nelle varie versioni è notevolmente migliorato anche questo secondo aspetto. Usa ***X-Window*** .

GNU: *GNU* vuole dire *GNU is Not Unix* (cioé *GNU* Non è *Unix*). La definizione è ricorsiva: viene da chiedersi allora

cos'è *GNU*, e si può ripetere e così all'infinito (vi ricordate è come: C'era una volta un Re, che disse al servitor, raccontami una fiaba e questi incominciò: C'era una volta....). *GNU* è il progetto di *Richard Stallman* per la creazione di un intero sistema operativo libero (diverso da *Unix* che era fornito chiuso dalle case produttrici). Il progetto è nato nel 1984: ora grazie a *GNU* abbiamo molti sistemi operativi ed applicazioni libere.

GNU-Linux: Quello che tutti chiamano erroneamente *Linux*. È un sistema operativo completo composto dal *kernel* di *Linux* e dalle applicazioni di *GNU*. Utilizzando il *kernel* di *Linux* da solo si otterrebbe un sistema privo di ogni programma, anche di quelli per la comunicazione con l'utente. Tutti quelli che dicono di usare *Linux* in realtà utilizzano *GNU-Linux*.

GPL: È la licenza pubblica di *GNU*: è il contratto che si approva per poter usare il *software* libero fornito da *GNU*. Contiene le quattro libertà fondamentali del *software* libero e il vincolo di non limitazione futura della libertà.

Hacker: È un esperto informatico desideroso di conoscere profondamente la propria materia in modo libero senza costrizioni o informazioni nascoste. Vuole conoscere esattamente come funzionano gli strumenti *hardware* e *software* che ha a disposizione. Rifugge tutti i *software* che non consentono di poter essere studiati, controllati e adattati alle proprie esigenze. Non fa nulla di illegale, non viola sistemi o la *privacy* di altri. Questo termine viene usato purtroppo ormai comunemente al posto della parola ***Cracker*** .

Hardware: È la parte fisica, materiale di un *computer*, quella elettronica, meccanica. In caso di dubbio ricordatevi il detto informatico: L'*hardware* si riconosce dal *software* in caso di guasto: l'*hardware* si può prendere a calci, contro il *software* di può solo inveire.

HTML: Linguaggio di marcatura che permette di usare dei marcatori opportuni (detti anche tag) per definire la struttura logica del documento, le relazioni che legano documenti diversi ed anche gli aspetti di presentazione del documento (stile, formattazione, eccetera). È il linguaggio ustao per definire le pagine Web che sarnno poi visulizzate da un *browser*.

IBM (International Business Machines Co.): Famosa e storica società nel campo dell'Informatica. Recentemente ha adottato *software* libero sui suoi *server* come strumento di concorrenza dei confronti di *Microsoft*. Sta facendo molto per la divulgazione del *software* libero ma non ha completamente mutato la sua politica. Detiene molti brevetti *software* e ha molti prodotti proprietari che vengono distribuiti con licenze molto restrittive e privi del codice sorgente. Le strade di IBM e del *Software* libero corrono parallele e vicine per l'attuale comune interesse di divulgare **GNU-Linux** , in futuro potranno convergere completamente o anche divergere.

IMAP: *Internet Message Access Protocol*, protocollo usato per la ricezione di posta elettronica che consente di organizzare e mantenere sul *server* la posta in arrivo e già letta.

Intel: Il più grande produttore di Microprocessori al mondo. Il suo prolungato successo si deve anche alla simbiosi in essere con *Microsoft* (tanto da venir nominati comunemente come un tutt'uno sotto il nome di *Wintel*). Tutti gli studiosi di architettura degli elaboratori sanno che il disegno dei processori *Intel* come i *Pentium* è meno lineare dei concorrenti Motorola (*PowerPC*), *Digital* (*Alpha*), *Crusoe* o MIPS. Molte delle complicazioni introdotte nei procesori *Intel* si devono alla scelta politica di avere completa compatibilità verso i precedenti prodotti della stessa marca: un programma in grado di funzionare su un 8086 dovrebbe funzionare anche sul più moderno *Pentium* IV. Forti investimenti sulle tecnologie ingegneristiche di produzione, possibili per l'elevato volume di vendita rispetto ai concorrenti mantengono i processori *Intel* competitivi nelle prestazioni.

Incapsulamento: Meccanismo presente in alcuni linguaggi mediante il quale si restringe l'accesso a determinate informazioni (dati e operazioni).

Ingegneria del software: Discipina informatica che studia le tecniche di realizzazione di grossi sistemi di calcolo, cercando di razionalizzare il progetto e di garantire che il prodotto finale risponda ai requisiti per i quali è stato progettato.

Installazione: Procedura che memorizza sul disco i *file* necessari al caricamento di un programma e li configura in modo da adattarsi alle caratteristiche del sistema.

Ipertesto: Testo che contiene dei collegamenti diretti (detti *link*, o collegamenti ipertestuali) fra alcune sue parti. Tali collegamenti permettono un accesso ed una lettura non sequenziale dell'ipertesto, navigando attraverso di esso.

ISP: *Internet Service Provider*, un fornitore di connettività a Internet. Usualmente fornisce anche servizi di base, come un *server* di posta elettronica (POP e SMTP).

Istanza: Un'istanza di una classe è ogni oggetto ottenuto da essa o da una delle sue sottoclassi.

Iterazione determinata: Costrutto di iterazione nel quale il numero di volte che il corpo viene eseguito è noto prima di iniziare l'iterazione. L'iterazione determinata è spesso espressa mediante la sintassi *for*.

Iterazione indeterminata: Costrutto di iterazione nel quale il numero di volte che il corpo viene eseguito non è noto prima di iniziare l'iterazione. L'iterazione indeterminata è spesso espressa mediante la sintassi *while* oppure *repeat*.

Iterazione: Costrutto di programmazione che consiste nella ripetizione di una determinata porzione di codice (corpo dell'iterazione).

Join: Operatore fondamentale dell'algebra relazionale che permette di mettere in relazione dati in tabelle diverse sulla base dei valori di attributi comuni oppure (nel caso del theta-*join*) sulla base di valori di attributi indicati esplicitamente nell'operazione di *join*.

Kernel: v. **Nucleo** .

Knoppix: È una distribuzione di *GNU-Linux* che non necessita di alcuna installazione. È sufficiente porre il CD nel lettore e riaccendere il *computer*. È molto più completa di *Demolinux* ma richiede *computer* abbastanza recenti per poter funzionare. L'attuale versione prevede l'interfaccia grafica con localizzazione italiana ma *OpenOffice* è incluso solamente in lingua inglese.

KDE: Interfaccia grafica molto potente, versatile, di facile utilizzo. Si presenta anche molto bene dal punto di vista estetico. È nata per consentire l'utilizzo di sistemi *GNU-Linux* (o altri *Unix*) anche per applicazioni personali. Giunta alla versione 3 viene fornita con un ricco corredo di applicazioni di ufficio, di configurazione e ludiche. Anche KDE usa ***X-Window*** .

Libero Pensiero Algoritmico: Terminologia coniata dall'autore per indicare la libertà di poter tradurre le proprie idee originali in programmi per *computer* senza dover sottostare a vincoli. Deve essere portata nel mondo moderno allo stesso livello della libertà di espressione, di stampa, di parola. I brevetti nel campo del *software* sarebbero la negazione della libertà di pensiero algoritmico. Il diritto di autore è perfettamente compatibile con la libertà di pensiero algoritmico perché tutela gli autori dal solo plagio. Chiunque desideri scrivere un testo, un libro, un articolo esprimendo con proprie parole concetti noti e meno noti è libero di farlo; così anche chi vuole scrivere programmi propri deve poterlo fare anche se sono presenti metodi risolutivi già usati altrove. Negare questo significa poter pretendere di brevettare formule matematiche, idee politiche, fedi religiose e in ultima analisi numeri. In Informatica Teorica si dimostra che ogni programma equivale a un numero. Provate a continuare a insegnare matematica se vi brevettano il pigreco o il numero e base dei logaritmi naturali.

Licenza d'uso: È il contratto che ogni utente deve stipulare per utilizzare un programma. Ci sono scritti i diritti e i doveri dell'utente. Per i programmi proprietari di solito la licenza prevede il diritto ad usare una copia del programma, il dovere di non modificarlo, non divulgarlo e di non analizzarne il funzionamento. Per i programmi liberi si prevede il diritto di piena conoscenza del programma, di duplicazione, di modificazione; l'utente ha il dovere di non limitare la libertà del programma. I programmi proprietari e i programmi liberi hanno una clausola in comune nella licenza d'uso: i produttori non si prendono responsabilità per i danni provocati da possibili errori presenti.

Linguaggio ad alto livello: Linguaggio di programmazione indipendente dalla macchina. Nei linguaggi moderni molta attenzione è posta ai meccanismi per la strutturazione del controllo, la modularizzazione, l'astrazione.

Linguaggio assembler: Linguaggio di programmazione a basso livello, le cui operazioni - direttamente eseguibili

dall'*hardware* - sono indicate in modo mnemonico.

Linguaggio di programmazione: Linguaggio artificiale progettato per potervi scrivere algoritmi in modo chiaro e non ambiguo.

Linguaggio macchina: Linguaggio di programmazione rudimentale, che viene interpretato direttamente dall'*hardware*.

Linus Torvalds: È un *hacker*. È l'autore del primo *kernel* denominato *Linux* e ancora oggi ne coordina lo sviluppo che viene portato avanti da decine e decine di sviluppatori coadiuvati da migliaia di utilizzatori esperti che trovano problemi e propongono soluzioni. *Torvalds* è finlandese ma attualmente vive e lavora negli Stati Uniti.

Linux: È il *kernel* di sistema *Unix* oggi più usato al mondo. *Linux* è altamente portabile su macchine di architettura molto diversa (palmari, *personal computer* con i processori più diversi, *server*, *supercomputer*). È *software* libero. Chiunque voglia guardare, studiare, modificare il codice di *Linux* può scaricare i sorgenti da www.kernel.org.

Login: Procedura di accesso ad un sistema multiutente. Talvolta è sinonimo di *account*, cioè del nome col quale un utente è noto al sistema.

Mandrake: È una distribuzione di *GNU-Linux*. Simile a *RedHat* è sempre stata famosa per la sua semplicità di installazione e di configurazione. È stata sviluppata in Francia. Attualmente la società *MandrakeSoft* che gestisce la distribuzione non naviga in ottime acque dal punto di vista finanziario.

Messaggio: Nel contesto di un linguaggio orientato agli oggetti, il meccanismo mediante il quale gli oggetti comunicano.

Metodo privato: Un metodo che non può essere invocato dall'esterno di una classe. La possibilità di definire metodi privati garantisce una certa forma di incapsulamento.

Metodo: procedura incapsulata in una classe; risponde a un determinato messaggio.

Modello dei dati: Un modello dei dati è costituito da un insieme di costrutti utilizzati per organizzare i dati e per descriverne l'evoluzione. Nei DBMS si distinguono i modelli concettuali e quelli logici.

Motorola: Uno dei più importanti costruttori di processori non *Intel* compatibili. Sebbene la fetta di mercato nell'ambito dei *personal computer* sia piccola (gli *Apple Macintosh* per esempio montano questo tipo di processori) Motorola ha un ampio mercato in installazioni industriali. Sui processori Motorola *GNU-Linux* funziona perfettamente, i sistemi operativi *Windows* no.

Multiutente: Una caratteristica di alcuni Sistemi Operativi, che permette la presenza di più di un utente. Gli utenti hanno spazi di lavoro distinti e (relativamente) protetti.

Nucleo: Comunemente indicato col termine inglese *kernel*. È il programma principale di ogni *computer*. Attivato al momento dell'accensione il nucleo rimane in memoria fino allo spegnimento o alla riattivazione da parte dell'utente. Il nucleo gestisce tutte le unità periferiche del sistema (dischi, stampanti, video-grafici, mouse, tastiera, terminali, ...) e fornisce dei modi di interazione standard che possono essere utilizzati dai programmi in esecuzione (le cosiddette *system call*). Il nucleo gestisce anche la sicurezza di accesso e la protezione dei dati.

NTFS: *New Technology file system*, organizzazione del *file system* disponibile in *Windows* 2000 e XP. Supporta funzionalità aggiuntive rispetto al precedente FAT e consente un migliore utilizzo dello spazio disco. È il *file system* dei sistemi professionali *Windows*. Il formato di registrazione è chiuso. Non ha il problema della frammentazione e gestisce la multiutenza. L'organizzazione appare complessa e questo lo rende meno efficiente dei concorrenti *file system ext3* o *raiserfs* presenti nei sistemi liberi.

Office automation (automazione d'ufficio): Ci si riferisce con questo termine all'uso coordinato di strumenti *software* per la comunicazione e l'elaborazione di dati, nel contesto di piccole reti di PC quali quelle che si hanno in ufficio.

Oggetto OLE: È un generico oggetto (testo, immagine, informazione multimediale) che può essere gestito tramite OLE (*Object Linking and Embedding*). Gli oggetti OLE permettono di creare documenti composti, dove ogni oggetto è gestito dall'applicazione che lo ha creato.

Oggetto: Entità dinamica che incapsula sia dati che operazioni su di essi. È un'istanza della propria classe.

OpenOffice: Derivata da *StarOffice*, prodotto della *Sun Microsystems*, *OpenOffice* è oggi una *suite* di prodotti da ufficio di potenzialità del tutto comparabili a *Microsoft Office*. È *software* libero. È un ottima scelta nel mondo della scuola per utilizzare *software* libero, formati aperti e, perché no, risparmiare il costo delle licenze d'uso in completa legalità.

Open Source: È un movimento con finalità simili a quello del *Software* Libero ma non identiche. Le regole previste dalle licenze *Open Source* sono più lasche di quelle previste da *GNU*: vengono consentite licenze che ammettono l'inserimento di *software Open Source* in prodotti proprietari o licenze che prevedono la revoca da parte del primo creatore. Vengono talvolta denominati erroneamente *Open Source* programmi proprietari per i quali è disponibile ma non riutilizzabile, modificabile il codice sorgente. Questa disponibilità spesso è non totale e rivolta a classi ben determinate di utenti. Questo errore fa comodo a molti grandi fornitori di *software* e viene il legittimo dubbio che si tratti di una interpretazione deliberatamente erronea per creare confusione fra le terminologie.

Palladium: Progetto che prevede il completo controllo dei programmi e dei dati elaborati su ogni *computer* tramite un meccanismo di certificazione. È un sistema certamente incompatibile con l'uso di *software* libero. Solo programmi e dati firmati potranno essere eseguiti o riprodotti sarà quindi possibile escludere interi gruppi di prodotti *software*, contenuti anche di rete dalla fruibilità dei possessori di sistemi tipo *Palladium*. Viene proposto come standard per i *personal computer* nei prossimi anni. (**Per saperne di più** □).

Parametro: Argomento di una procedura o funzione.

Patch: In italiano viene tradotto con il temine pezza. L'idea espressa è infatti quella di rammendare il programma ponendo una pezza dove questo presenti un difetto. Serve per modificare un programma per risolvere un malfunzionamento. Solo chi conosce il codice sorgente di un programma può produrre una *patch*.

Patent: v. **Brevetto sul software** .

Payware: *Software* proprietario con licenza d'uso onerosa. È il classico programma proprietario. Non potrete mai essere sicuri che faccia ciò che dichiara perché non potete farlo analizzare, può contenere ogni sorta di funzionalità benevola o malefica a vostra insaputa. L'utilizzatore non può modificarlo. Gli errori, in modo particolare quelli che comportano rischi per la sicurezza, vengono mantenuti riservati fino a quando la casa costruttrice non abbia disponibile la **patch** per ripararli. Questo dà agli utilizzatori di *software* proprietario una falsa sensazione di sicurezza.

Permesso d'autore: o *copyleft*. Viene posto in contrapposizione con il *copyright* o Diritto di Autore. In realtà è una particolare applicazione del diritto di autore nel quale l'autore invece che rivendicare a sé il diritto sull'opera in senso restrittivo rivendica solo la paternità dell'opera ma ne lascia (*to leave*, lasciare, di cui *left* è il participio passato) la libera distribuzione e copia.

Pezza: v. ***Patch*** .

POP: *Post Office Protocol*, protocollo usato per la ricezione di posta elettronica.

PowerPC: È una famiglia di processori sviluppata da IBM e Motorola. È incompatibile con i processori *Intel*.

PPP: *Point to Point Protocol*, protocollo usato per la connessione via modem tra un calcolatore ed un ISP.

Procedura: Sottoprogramma dotato di un nome.

Programma di videoscrittura: Programma che permette di scrivere, modificare, formattare documenti di vario genere.

Proprietario: Si dice proprietario un sistema operativo, un *software* applicativo o un formato di dati del quale non è possibile conoscere le specifiche. Deriva dall'inglese *proprietary* che significa di proprietà privata. L'uso di formati di dati proprietari rende vulnerabili al ricatto delle ditte detentrici del formato stesso. Il tempo proprio o aziendale investito nell'inserire dati in un determinato formato è un investimento, talvolta cospicuo. Coi formati proprietari non vi è la garanzia di poter ottenere restituiti i propri dati.

RedHat: Distribuzione molto famosa di *GNU-Linux*. È gestita dall'omonima società americana. Molto ben costruita, nelle ultime versioni ha migliorato considerevolmente la facilità di installazione e l'interfaccia grafica che risulta essere l'unione di KDE e *Gnome*. Fornisce versioni vendibili con manuali e servizi di consulenza e assistenza a pagamento. Anche i servizi di aggiornamento del *software* potrebbero in futuro non essere gratuiti.

Registro di sistema: *File* presenti in *Windows* che mantengono informazioni sullo stato del Sistema Operativo.

Relazionale: (modello dei dati) Modello logico dei dati nel quale i dati sono rappresentati mediante relazioni (intese nel senso del modello relazionale, vedi **relazione**). I riferimenti fra dati diversi nel modello relazionale sono realizzati usando solo valori.

Relazione: (in senso matematico) Una relazione su n insiemi anche non distinti D1 , ... , Dn è definita come un qualsiasi sottoinsieme del prodotto cartesiano D1 x ... x Dn .

Relazione: (nel senso del modello relazionale dei dati) Struttura di rappresentazione dei dati visualizzabile come una tabella che rispetti i seguenti vincoli; ogni colonna della tabella ha una intestazione (nome di attributo) e le intestazioni delle colonne sono diverse fra di loro; i valori di ogni colonna sono omogenei; le righe sono diverse fra loro; l'ordinamento tra le righe è irrilevante; l'ordinamento tra le colonne è irrilevante.

Richard Stallman: Padre fondatore dell'idea del *Software* Libero e del progetto *GNU*. Programmatore all'*MIT* lasciò l'ente di ricerca per fondare la *Free Software Foundation*.

Schema: (di una base di dati) Oggetto descrive la struttura invariante nel tempo di una serie di dati, ovvero l'aspetto intensionale.

Shareware: Sono programmi proprietari divulgati gratuitamente in forma di codice eseguibile. Hanno funzionalità ridotte o funzionano per un tempo limitato. Per ottenere la versione completamente e indefinitamente funzionante del programma occorre pagarne il prezzo. Hanno le stesse caratteristiche di insicurezza di tutto il *software* proprietario con l'aggravante che spesso il costruttore è poco conosciuto.

Sistema informatico: Parte automatizzata del sistema informativo, ovvero quella parte che gestisce le informazioni usando l'informatica.

Sistema informativo: Componente di una organizzazione che gestisce (acquisisce, elabora, conserva, produce) le informazioni interessanti per l'organizzazione in questione. Si noti che, in principio, l'esistenza di un sistema informativo prescinde da qualsiasi processo di automazione. In effetti istituzioni vecchie di secoli (quali ad esempio le banche) usavano sistemi informativi prima che esistesse l'energia elettrica.

Sistema operativo: Insieme di programmi che interagiscono e controllano le funzionalità della macchina fisica (*hardware*) in modo tale da offrire ai programmi applicativi ed agli utenti un insieme di funzionalità con un maggiore livello di astrazione.

SMTP: *Simple Mail Transfer Protocol*, protocollo usato per trasferire messaggi di posta elettronica. È usato da un *client* per inviare posta ad un *server*.

Software: È la parte logica, immateriale di un *computer*, quella fatta da programmi e dati. In caso di dubbio ricordatevi il detto informatico: L'*hardware* si riconosce dal *software* in caso di guasto: l'*hardware* si può prendere a calci, contro il *software* di può solo inveire.

Software applicativo: Insieme dei programmi che, usando le funzionalità del sistema operativo, permettono di realizzare specifici compiti quali, ad esempio: vidoescrittura, navigazione in Internet, elaborazione di dati mediante fogli elettronici, gestione di basi di dati, elaborazioni multimediali.

Software di pubblico dominio: *Software* non coperto da diritti. Si può copiare liberamente senza temere alcuna conseguenza.

Software freeware: *Software* coperto da *copyright* ma disponibile gratuitamente (le aziende che lo distribuiscono guadagnano sulla fornitura di servizi). Si sta sviluppando notevolmente soprattutto grazie al sistema operativo *Linux* che è stato concepito come *free-software*.

Software Libero: Se siete arrivati fino a questo punto nella lettura e non l'avete compreso... Siccome è un glossario per correttezza comunque lo rispiego: è un *software* che rispetta tutte le quattro condizioni seguenti: 1- libertà di esecuzione del programma senza vincoli, 2-libertà di conoscere come funziona il programma, 3- libertà di modificare il programma, 4- libertà di redistribuire il programma modificato. Il *software* libero è un modello culturale etico ed economico.

Software proprietario: *Software* coperto da *copyright* e deve essere acquistato. Se viene usato senza essere acquistato (ad esempio usando copie illegali) si può essere perseguiti a norma di legge.

Sorgente: v. **Codice Sorgente** .

Sorgente Aperto: Programmi il cui codice sorgente è aperto, i.e. leggibile liberamente. Raramente viene usato come sinonimo di *Open Source*.

Sorgente Chiuso: Programmi il cui codice sorgente è chiuso. Vengono distribuiti solamente nella forma di codice eseguibile. Non sono controllabili, studiabili, modificabili.

Sottoclasse: Una classe derivata da un'altra (la superclasse) mediante estensione o ridenominazione (di metodi).

SQL: (*Structured Query Language*) Linguaggio di interrogazione per basi di dati relazionali, disponibile in varie versioni, con interfacce sia grafiche che testuali. Può essere a sé stante oppure immerso in linguaggio di programmazione ospite (Pascal, Java, C, oppure un linguaggio specifico per alcune applicazioni).

Superfloppy: Dischetti ad alta capacità (120 o 150 MB) utili per gestire *backup* di media dimensione. Alcune unità (*SUPERDISK*) sono compatibili (cioè sono in grado di leggere e scrivere) coi tradizionali *floppy*.

Suse: Distribuzione *GNU-Linux* molto ben costruita e professionale. È di origine tedesca, viene gestita dalla ditta omonima ed è stata scelta come distribuzione per la pubblica amministrazione dal governo tedesco. Anche questa distribuzione è liberamente scaricabile da Internet ma è anche acquistabile in confezioni corredate da manuali, esistono servizi di consulenza a pagamento per imprese.

Tipo di dato: Concetto dei linguaggi di programmazione, costituito da un insieme di valori e da un repertorio di operazioni definite su tali valori.

Transazione: Insieme di operazioni su di una base di dati che devono essere considerate in modo indivisibile (o atomico).

Unix: *Unix* è oggi una classe di sistemi operativi. *Linux* è uno di questi. L'idea di *Unix* nacque nei laboratori della *Bell* (Oggi AT&T, società telefonica). In realtà venne alla luce nonostante l'AT&T. *Ritchie* e *Thomson* portarono avanti volontariamente il progetto *Multics* che la *Bell* aveva dichiarato chiuso. Era il 1969 ma la vera rivoluzione che ha portato *Unix* ai giorni nostri si deve a *Ritchie* ed è datata 1973. Il sistema operativo, fino ad allora scritto espressamente per ogni diverso *computer* nel linguaggio proprio di quel *computer*, viene riscritto in un linguaggio nuovo: il C, inventato apposta da *Ritchie*. Anche il traduttore del linguaggio C in codice macchina è scritto in C. *Unix* diventa in questo modo portabile in macchine diverse con poco sforzo. Il sistema operativo non è più specifico di una

macchina ma adattabile a macchine diverse. I programmatori e gli utenti possono così avere una interfaccia unica e un insieme di strumenti unico anche cambiando tipo di *computer*. Ormai il C è diventato di uso universale per la scrittura dei sistemi operativi. Anche i sistemi operativi e gli applicativi della *Microsoft* sono scritti in C o in un suo dialetto.

Uova di Pasqua: Sono simili ai **cavalli di Troia** . Non hanno però funzionalità malefiche. Molto note sono le uova di pasqua con le quali i veri sviluppatori di *software* proprietario firmano le proprie opere per rifuggire la frustrazione di non poter vedere attribuito il proprio lavoro fuori dall'azienda. Le ditte produttrici di *software* proprietario infatti vogliono che il prodotto sia associato alla marca e non alle capacità dei progettisti e sviluppatori che l'hanno effettivamente creato che dovrebbero sempre rimanere anonimi.

Vermi: Sono programmi che si autoreplicano e si propagano da un sistema all'altro sfruttando errori di programmazione dei programmi. Famosi sono il verme di *Morris*, il primo conosciuto creato nel 1988 che mise in ginocchio l'allora nascente Internet, il *code Red* e il recentissimo *Microsoft SQL Slammer*. L'infezione può essere trasferita anche fra macchine di architettura differente. Molti vermi non producono danni esplicitamente ma bloccano le attività consumando la potenza elaborativa. Possono esistere comunque vermi che danneggiano e vermi che creano canali per successive intrusioni. Ci si difende con personale addetto a aggiornare o riparare i programmi appena venga conosciuta una nuova fragilità del sistema. Occorre libero accesso alle informazioni sui possibili problemi riscontrati.

Virus: Frammenti di programma in grado di autoreplicarsi e di attaccarsi a programmi esistenti. La replicazione avviene per copia di programmi infetti. Sono strettamente legati all'architettura *software* del sistema: un virus per *Windows* non funziona in *GNU-Linux*. Come i virus biologici quelli informatici rimangono inizialmente inattivi per un periodo di latenza e cercano di installarsi nei punti nevralgici del sistema per massimizzare il contagio. Ci si difende utilizzando sistemi che non consentano la modificazione dei programmi di sistema o l'accesso ai dispositivi durante il normale uso; sistemi professionali quali *Windows* NT, 2000, XP, *GNU-Linux* o altri *Unix*. Occorre anche identificarsi come un utente di prova (che non abbia accesso ai dati) quando si vogliono provare programmi della cui sicurezza non si è certi. Occorre evitare anche l'uso programmi di lettura della posta che attivino automaticamente contenuti (per esempio *OutLook*) e di strumenti di elaborazione di ufficio che nascondano nei documenti la possibilità di inserire interi programmi capaci di danneggiare il sistema e i propri dati (e.g. le macro di *Microsoft Office*). Un *file* .doc che dovrebbe contenere solamente dati può contenere delle macro, che sono un programma, e quindi anche virus.

Web: v. **WWW** .

Windows: È una categoria di sistemi operativi diversissimi fra loro accomunati dal solo fatto di essere costruiti dalla *Microsoft*. I sistemi *Windows* 3.11, 95, 98, ME sono per elaborazioni personali, privi di ogni protezione per l'accesso ai dati o alle periferiche. I sistemi W. NT, 2000, XP sono professionali. Sono sistemi proprietari a sorgente chiuso.

Wintel: È un modo comune di nominare la associazione *Windows*|*Intel*.

Worm: v. **Vermi** .

WWW (World Wide Web): È un'altra conquista libera. Il WWW è la ragnatela grande come il mondo, un ipertesto di dimensioni globali ottenuto collegando informazioni contenute in pagine scritte in linguaggio html (formato libero) tramite collegamenti ipertestuali. Le pagine possono includere anche elementi di altro formato (secondo uno standard aperto chiamato *mime*) anche multimediale. La navigazione nell'ipertesto avviene tramite l'uso di programmi cliente chiamati *Browser* che richiedono le pagine ai *server* tramite il protocollo (aperto) http. Nell'ambito del WWW è anche stata standardizzata la modalità di riferire documenti locali o remoti per mezzo dell'uso dell'URL (*Uniform Resource Locator*). Un esempio di URL è http://www.fsf.org. È stato creato da *Tim Berners-Lee* del CERN di Ginevra nel 1990.

X-Window: Progettato all'*MIT* il prestigioso *Massachussets Institute of Technology*, *X-Window* è un sistema di interazione grafica *client-server*. In pratica consente di poter separare il programma in esecuzione e la sua interfaccia in due macchine diverse. Con *X-Window* è possibile interagire graficamente con programmi in funzione in altri punti della rete. Questo consente per esempio di poter fare elaborazioni grafiche interattive su *computer* molto potenti e costosi senza dover essere fisicamente presso un terminale grafico del *computer* stesso. Esiste un solo elaboratore in

Emilia Romagna capace di fare le elaborazioni metereologiche accurate. Senza *X-Window* tutti i metereologi dovrebbero andare fisicamente al CINECA che gestisce questo *computer*, con *X-Window* fanno le elaborazioni comodamente nel loro ufficio. Tutte le interfacce grafiche di *Unix* e di *GNU-Linux* sono basate su *X-Window*. I sistemi *Windows* possono avere *X-Window* solo come applicazione.

Autori

Hanno realizzato il materiale di questo modulo:

Prof. Renzo Davoli
Avrà 40 anni nel 2004.
Per ora fa il Professore Associato Universitario al Dipartimento di Scienze dell'Informazione dell'Università di Bologna.
Insegna Sistemi Operativi al corso di Laurea in Informatica (Facoltà di Scienze Matematiche Fisiche e Naturali).
È docente anche di Progettazione di Sistemi Operativi presso il corso di Laurea Specialistica in Informatica, di Tecnologie, servizi e programmazione Internet e di Sicurezza Informatica al Master in Comunicazione e Tecnologia dell'Informazione.
Studia la sicurezza dei sistemi *wireless* per reti cellulari basate su IPv6, la migrazione ad Internet2, la *privacy* nei protocolli di comunicazione, i sistemi di simulazione per la didattica dei sistemi operativi.
Coordina le reti e i sistemi del Dipartimento di Scienze dell'Informazione.
È un accanito sostenitore della libertà di pensiero algoritmico.
È un *hacker*.
Ha tre splendidi figli Silvia, Giulia e Giorgio che nel 2004 avranno rispettivamente 11, 9 e 4 anni.
Ha ovviamente anche una **home page**.

Prof. Maurizio Gabbrielli
Maurizio Gabbrielli è professore straordinario di informatica presso il Dipartimento di Scienze di Informazione dell'Università di Bologna. Ha conseguito il titolo di dottore di ricerca in informatica nel 1992 presso l'Università di Pisa ed è stato quindi ricercatore presso il CWI (centro di ricerca per la matematica e l'informatica) di Amsterdam, ricercatore presso l'Università di Pisa e professore associato presso l'Università di Udine. I suoi interessi di ricerca includono i linguaggi di programmazione, i metodi formali per la verifica e l'analisi, la programmazione con vincoli e la teoria della concorrenza. È autore di più di 50 articoli pubblicati su riviste e atti di congressi internazionali, ha partecipato a numerosi progetti italiani ed internazionali ed è stato membro e presidente del comitato di programma di varie conferenze internazionali. È attualmente presidente del comitato esecutivo di PPDP, presidente della associazione italiana di programmazione logica, membro del comitato esecutivo di ALP (*Association for Logic Programming*) e membro del comitato esecutivo di EAPLS (*European Association for Programming Languages and Systems*).

Prof. Simone Martini
Simone Martini è professore ordinario di Informatica nell'Università Bologna. Laureato in Scienze dell'Informazione e Dottore di Ricerca in Informatica (Università di Pisa), ha insegnato nelle Università di Pisa e Udine. È visitatore presso il *Systems Research Center* della *Digital Equipment Corporation* a Palo Alto, la *Stanford University*, l'*École normale supérieure* di Parigi. I suoi interessi di ricerca riguardano i fondamenti logici dei linguaggi di programmazione.