PAOLO CAMAGNI
RICCARDO NIKOLASSY

# NUOVO TECNOLOGIE E PROGETTAZIONE DI SISTEMI INFORMATICI E DI TELECOMUNICAZIONI

3

Per l'articolazione INFORMATICA
degli Istituti Tecnici settore Tecnologico

HOEPLI TECNICA PER LA SCUOLA

EDIZIONE **OPENSCHOOL**

1 **LIBRO** DI TESTO
2 **E-BOOK** +
3 RISORSE **ONLINE**
4 **PIATTAFORMA**

HOEPLI

PAOLO CAMAGNI RICCARDO NIKOLASSY

# Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni

## Per l'articolazione Informatica degli Istituti Tecnici settore Tecnologico

**Volume 3**

EDITORE ULRICO HOEPLI MILANO

**Copyright © Ulrico Hoepli Editore S.p.A. 2021**
Via Hoepli 5, 20121 Milano (Italy)
tel. +39 02 864871 – fax +39 02 8052886
e-mail hoepli@hoepli.it

**www.hoepli.it**

Tutti i diritti sono riservati a norma di legge
e a norma delle convenzioni internazionali

# Indice

## Unità I

## Architettura di rete e formati per lo scambio dei dati

**L1 I sistemi distribuiti** 2
I sistemi distribuiti 2
Classificazione dei sistemi distribuiti 5
Benefici legati alla distribuzione 5
Svantaggi legati alla distribuzione 7
**Verifica... le conoscenze** 9

**L2 Evoluzione dei sistemi distribuiti e dei modelli architetturali** 10
Premessa 11
Architetture distribuite hardware: dalle SISD al cluster di PC 11
Architetture distribuite software: dai terminali remoti ai sistemi completamente distribuiti 17
Architettura a livelli 18
Conclusioni 19
**Verifica... le conoscenze 20**

**L3 La comunicazione nel Web con protocollo HTTP** 22
HTTP e il modello client-server 23
Il protocollo HTTP 23
Conversazione client-server 24
Tipi di connessioni 24
I messaggi HTTP 25
Messaggio di richiesta: HTTP Request 25
Messaggio di risposta: HTTP Response 27
Header HTTP 27
Metodi (verbi) HTTP 28
Le rappresentazioni HTTP 30
I codici di stato 31
Come vedere il funzionamento di HTTP 32
**Verifica... le conoscenze** 35

**L4 Le applicazioni Web e il modello client-server** 36
Applicazioni Web: generalità 36
Il modello client-server 38
Distinzione tra server e client 39
Livelli e strati 40
**Verifica... le conoscenze** 43

**L5 Le applicazioni di rete** 44
Il modello ISO/OSI e le applicazioni 44
Applicazioni di rete 45
Scelta dell'architettura per l'applicazione di rete 48
Servizi offerti dallo strato di trasporto alle applicazioni 50
Conclusioni 51
**Verifica... le conoscenze** 52

**Esercizi in Laboratorio**
**1 Il linguaggio XML** 53
**2 Lo schema DTD delle pagine XML** 62
**3 La navigazione nei documenti con XML Path Language** 69
**4 Il parsing XML con Java: le specifiche JAXP** 81
**5 Il linguaggio JSON** 93
**6 Web server Apache**
**7 Web server Apache in Linus**
**8 Web server IIS su Windows**


**Puoi scaricare gli Esercizi 6, 7 e 8 anche da** hoepliscuola.it

**AREA DIGITALE**

• Creare cluster HPC con Parallelknoppix

- Dettaglio per tipologia di header
- Le operazioni CRUD di HTTP
- La sicurezza con HTTPS
- Applicazioni P2P
- Comparison chart TPC vs UDP
- Esempio di utilizzo di XSchema
- XML e vincoli di Integrità in XML schema
- Caratteristiche delle tre tipologie di interfacce
- Le costanti dell'interfaccia Node
- Diagrammi sintattici delle istruzioni/strutture dati JSON
- JSON e HTMLS
- Select dinamiche con PHP e JSON

- Esercizi interattivi

- Esercizi per il recupero e l'approfondimento
- Esercizi aggiuntivi


## Unità 2

## Il socket e la comunicazione con i protocolli TCP/UDP

**L1 I socket e i protocolli per la comunicazione di rete** 106
Generalità 106
Le porte di comunicazione e i socket 108
**Verifica... le conoscenze** 112

**L2 La connessione tramite i socket** 113
Generalità 113
Famiglie e tipi di socket 114
Trasmissione unicast e multicast 117
**Verifica... le conoscenze** 119

**Esercizi in Laboratorio**
**1 Java socket** 120
**2 Java socket: realizzazione di un server TCP** 128
**3 Realizzazione di un server multiplo in Java** 131
**4 Java socket: un'animazione client-server** 136
**5 Il protocollo UDP nel linguaggio Java** 144
**6 Applicazioni multicast in Java** 149
**7 Un esempio completo con i Java socket:"la chat"** 153
**8 I socket nel linguaggio C** 160
**9 Server TCP in C** 167
**10 Client TCP in C** 172
**11 Il protocollo UDP nel linguaggio C** 180
**12 Un esempio completo con i socket: asta online** 186
**13 Applicazioni multicast in C** 191


### AREA DIGITALE


- Confronto ISO/OSI e Internet
- Connectionless e connection-oriented
- Socket: maschile o femminile
- API: Application Programming Interface
- Un esempio di verifica
- Funzioni di conversione Big-Endian/Little-Endian
- Abilitare telnet in Windows 10
- Funzioni GETSOCKOPT() e SETSOCKOPT()

- Esercizi interattivi

- Esercizi proposti


## Unità 3

## Applicazioni lato server in PHP e AJAX

**L1 Classi e oggetti in PHP** 196
Generalità su PHP 196
Il paradigma a oggetti in PHP 197
I costruttori da PHP 4 a PHP 7 200
L'ereditarietà 202
**Verifica... le competenze** 203

**L2 Comunicazione client-server in PHP con AJAX** 204
Generalità 204
Il funzionamento di AJAX 205
**Verifica... le competenze** 211

**L3 La connessione al database MySQL**

**Esercizi in Laboratorio**

**1 Installazione di EasyPHP e XAMPP**
**2 Utilizzo di AJAX nella interrogazione di database MySQL** 212
**3 Upload/download dei file e utilizzo di file JSON** 217
**4 Client-server in PHP: invio di messaggi e file mediante socket** 225
**5 Connessione FTP con uno script PHP**
**6 Leggere e scrivere file Excel e Word**
**7 Inviare una mail con PHP connesso a MySQL**
**8 Creare un file pdf con PHP**

Puoi scaricare la Lezione 3 e gli Esercizi 5, 6, 7 e 8 anche da hoepliscuola.it

**AREA DIGITALE**

- Rename della classe
- Altri esercizi con i file

## Unità 4

## Applicazioni lato server con codice separato: CGI e servlet

**L1 Le applicazioni lato server** 236
La programmazione server-side 236
Modello a codice separato 237
Struttura di una servlet 239
La classe HttpServlet 241
Ciclo di vita di una servlet 243
Output sul client 243
Deployement di un'applicazione Web 244
Il Context XML descriptor o Deployement descriptor 246
Esecuzione di una servlet 248
Servlet concorrenti 249
Vantaggi e svantaggi delle servlet 249
**Verifica... le conoscenze** 250

**L2 Servlet e database** 251
La connessione ai database 251
La connessione con JDBC Java Database Connectivity 252
Tipi di driver JDBC 253
Utilizzare JDBC standalone 254
Servlet con connessione a MySQL 257
Applicazione e servlet con connessione ad Access 258
**Verifica... le conoscenze** 260

**L3 Servlet con database embedded** 261
I database embedded 261
Uno strumento comune per l'amministrazione dei database 264
Utilizzare Derby 264
**Verifica... le competenze** 268

**Esercizi in Laboratorio**

**1 Applicazioni CGI** 269
**2 XAMPP e il server Engine Tomcat** 274
**3 L'inizializzazione delle servlet** 282
**4 L'interazione get/post tra client e servlet** 286
**5 L'interazione con client AJAX e lo scambio dati in formato JSON** 292
**6 La permanenza dei dati con le servlet: i cookie** 298
**7 La permanenza dei dati con le servlet: le sessioni** 302
**8 JDBC e MySQL** 307
**9 Servlet e database MDB con parametri** 312
**10 Servlet con JavaDB Derby** 316

**AREA DIGITALE**

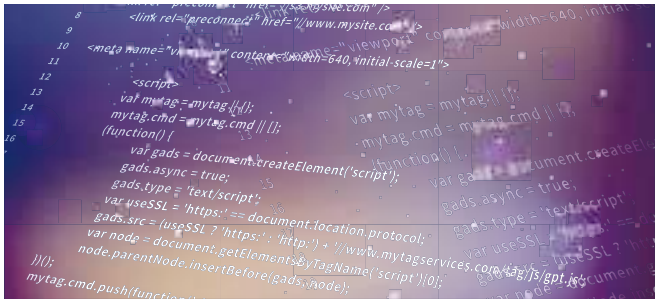
- Schema UML della gerarchia delle classi
- ODBC
- Installazione della libreria UCanAccess

- Procedura per la migrazione tra due DMBS differenti
- Installazione di Tomcat su Apache
- NetBeans, Eclipse e il file web.xml
- Metodi principali classe Cookie
- Elenco dei metodi dell'oggetto HttpSession
- Configurazione manuale di JDBC
- Registrazione database in Windows mediante ODBC

- Esercizi interattivi

## Unità 5

## Applicazioni a codice embedded in HTML con JSP




**L1 JSP: Java Server Pages** 322
Le Java Server Pages (JSP) 322
Tag in una pagina JSP 324
Tag scripting-oriented 325
Oggetti impliciti 327
**Verifica... le conoscenze** 332

**L2 Java Server Pages e JavaBean** 333
JavaBean 333
Uso di JavaBean 335
Configurazione derll'applicazione 337
Passaggio parametri al Bean 338
Conclusioni 341
**Verifica... le conoscenze** 342
**Verifica... le competenze** 343

**Esercizi in Laboratorio**
**1 JSP: primi esempi di programmazione** 344
**2 JSP con parametri da HTML** 349
**3 JSP e database MySQL** 354
**4 JSP e MDB** 356
**5 JSP e database Derby** 358
**6 JSP Bean e database Access, MySQL e Derby** 360


## AREA DIGITALE

- Esempio di codice Java generato da Tomcat: la classe oggi_jsp.java
- XML-oriented tag o JSP Standard Actions
- Elenco attributi

- Esercizi interattivi

## Unità 6

## I Web Service e le API di Google

**L1 Introduzione ai Web Service: protocolli SOAP e REST** 368
Introduzione 369
Che cos'è un Web Service 370
Un nuovo modello basato su XML: l'architettura SOA 372
Il protocollo SOAP 373
Il protocollo REST 375
Conclusione 378
**Verifica... le conoscenze** 380

**L2 La geolocalizzazione con le API di Google e LeafletJS** 381
La geolocalizzazione 381
Usare le API di Google Maps 383
Associare eventi agli oggetti Google Maps 386
Le mappe con LeafletJS 390
**Verifica... le competenze** 394

**Esercizi in Laboratorio**
**1 Strumenti per lo sviluppo e il testing di Web Service: framework e SDK (Software Development Kit)** 395
**2 Realizzazione di un Web Service SOAP in Java** 405
**3 Realizzazione di un Web Service REST in Java** 409
**4 Realizzazione di un Web Service REST in PHP** 418
**5 Integrazione dei servizi API nelle applicazioni PHP** 427

**6** Utilizzo avanzato delle API di Google in PHP

**7** Le API di Google: interazione tra JavaScript, AIAX, PHP e SQL 430

Puoi scaricare l'**Esercizio 6** anche da hoepliscuola.it

## AREA DIGITALE

- W3C – Standardizzazione
- Categorie di elementi SOA
- Caratteristiche e struttura dei componenti pila protocollare SOAP
- Come ottenere la API key
- Un progetto avanzato WSDL

- Esercizi interattivi

- Esercizi per il rinforzo e l'approfondimento

# Unità 7

## Nuove tipologie di database: i database NoSQL

**L1 NoSQL: una nuova proposta di database** 436

Introduzione 436

SQL vs NoSQL 437

Alcuni esempi di database NoSQL 441

**Verifica... le conoscenze** 445

### Esercizi in Laboratorio

**1** Firebase: caratteristiche di base e creazione degli archivi 446

**2** Utilizzare Firebase in JSP 456

## AREA DIGITALE

- Esercizi interattivi

# Unità 8

## Android e i dispositivi mobili

**L1 Dispositivi e reti mobili** 468

Premessa 468

Reti mobili 469

Software per dispositivi mobili 471

Sistemi operativi per dispositivi mobili 472

Ambienti di sviluppo per dispositivi mobili 474

Xamarin 476

React native 476

Unity 476

**Verifica... le conoscenze** 477

**L2 Android: un sistema operativo per applicazioni mobili**

### Esercizi in Laboratorio

**1** Android Studio: installazione e configurazione

**2** L'interfaccia grafica di Android Studio

**3** Utilizzare i widget nelle app Android

**4** Un'app completa: la calcolatrice

**5** Utilizzare i sensori nelle app

Puoi scaricare la **Lezione 2** e gli **Esercizi 1**, **2**, **3**, **4** e **5** anche da hoepliscuola.it

## AREA DIGITALE

- I sistemi operativi Palm OS, Symbian e Linux
- Set che compongono Xcode

- Esercizi interattivi

# Presentazione

L'opera in questa nuova edizione presenta un'organizzazione e una struttura simili all'edizione precedente, che grande riscontro positivo ha ottenuto, mantenendo lezioni teoriche e lezioni pratiche specifiche per l'attività laboratoriale, individuabili anche dalla colorazione del bordo pagina. Si è volutamente cambiato il formato del volume offrendo più "respiro" al testo per facilitare la lettura delle singole pagine che risultano così più "leggere" e meno dense di contenuti.

L'opera recepisce le indicazioni ricevute dai docenti che hanno in uso la precedente edizione; in particolare il testo è stato arricchito di tre nuovi argomenti per offrire più alternative alla realizzazione di percorsi didattici personalizzati:
- la definizione e l'analisi dei documenti in formato XML, dalla definizione con lo schema DTD al parsing con XML Path Language, e la libreria JAXP di Java;
- i Web Service, nei quali vengono presentati i protocolli SOAP e REST, e lo sviluppo e l'utilizzo delle API nei diversi linguaggi di programmazione;
- i database NOSQL, in particolare viene proposto Firebase come esempio di database a documenti e descritto il suo utilizzo con il linguaggio JavaScript.

Anche la parte riguardante JSON è stata rivista e integrata con la lettura dei documenti utilizzando il linguaggio Java con le specifiche librerie appositamente rilasciate.
Oltre all'utilizzo dei servizi di Google per la geolocalizzazione, viene anche presentato LeafletJS, che offre la possibilità di utilizzare mappe a più layers.

## STRUTTURA DEL TESTO, METODOLOGIA E STRUMENTI DIDATTICI

Il volume è così articolato:
- Unità 1: Architettura di rete e formati per lo scambio dei dati
- Unità 2: I socket e la comunicazione con i protocolli TCP/UDP
- Unità 3: Applicazioni lato server in PHP e AJAX
- Unità 4: Applicazioni lato server con codice separato: CGI e servlet
- Unità 5: Applicazioni a codice embedded in HTML con JSP
- Unità 6: I Web Service e le API di Google
- Unità 7: Nuove tipologie di database: i database NoSQL
- Unità 8: Android e i dispositivi mobili

Le **8 Unità di Apprendimento** sono suddivise in **Lezioni**, ciascuna delle quali ha una struttura innovativa e cerca di essere una reale guida all'apprendimento: infatti risulta essenziale nei contenuti ma ricca di esempi e di procedure guidate.
Ogni Unità si apre con le finalità e i contenuti dei diversi argomenti descritti negli obiettivi disciplinari.
Ogni Lezione si apre con *In questa lezione impareremo*, seguita da una *mappa concettuale* che offre una sintetica anticipazione dei contenuti sviluppati, fungendo da schema riepilogativo e di sistematizzazione dei saperi sia per abituare lo studente all'apprendimento e alla sintesi dei contenuti attraverso un percorso ragionato, sia per la didattica inclusiva.

Ricapitolando, il volume risulta idealmente strutturato in **4 sezioni tematiche**:
- la prima descrive i **sistemi distribuiti** a partire dalla loro storia ed evoluzione, con la presentazione dei diversi **modelli architetturali**;

- la seconda sezione propone **applicazioni lato server** utilizzando il **linguaggio PHP**, in particolar modo gli aspetti avanzati con l'utilizzo di **AJAX**, degli oggetti e delle **API** di **Google** in **Javascript** e la realizzazione di **Web Service REST e SOAP**;
- la terza analizza i **socket** e la modalità di comunicazione con i **protocolli TCP/UDP** utilizzando il **linguaggio Java** e **C**; in particolare un'Unità di Apprendimento è dedicata alle applicazioni lato server con codice separato (**CGI** e **servlet**) e una per le **JSP**;
- la quarta sezione propone la programmazione per dispositivi mobili **Android**, con un insieme di esercitazioni che permettono di realizzare **app** anche complesse connesse a database remoti.

A ogni argomento è stata riservata una **sezione laboratoriale** che propone esercitazioni specifiche a difficoltà crescente in tutti i linguaggi di programmazione utilizzati nel volume: **C**, **Java**, **JavaScript**, **PHP e AJAX** ed è corredata da **numerosi esempi** completamente svolti, come, per esempio, l'esercizio sulle "Applicazioni multicast in C".

L'esposizione dei contenuti è semplice e dettagliata. Il testo, integrato da contenuti online, accompagna lo studente in un preciso percorso didattico e costituisce un riferimento chiaro per il suo apprendimento e per l'acquisizione delle competenze previste dalle linee guida. I materiali collegati all'eBook+ sono attivabili attraverso le icone nella versione digitale.

## ESPANSIONI DIGITALI

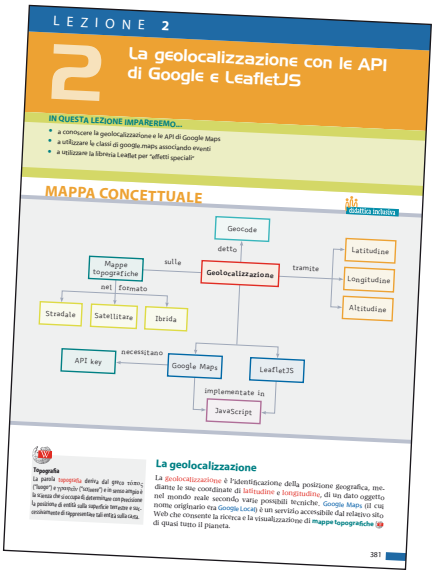
La nuova edizione Openschool consente di:
- scaricare gratuitamente il libro digitale arricchito; l'eBook+ permette in particolare di:
  - eseguire tutte le esercitazioni a risposta chiusa in modo interattivo;
  - scaricare gli approfondimenti tematici e gli esercizi di recupero, rinforzo e approfondimento;
  - scaricare le lezioni e le esercitazioni integrative;
- disporre di ulteriori esercizi online, utilizzabili a discrezione del docente per classi virtuali gestibili attraverso la piattaforma.

## RISORSE ONLINE E PIATTAFORMA DIDATTICA

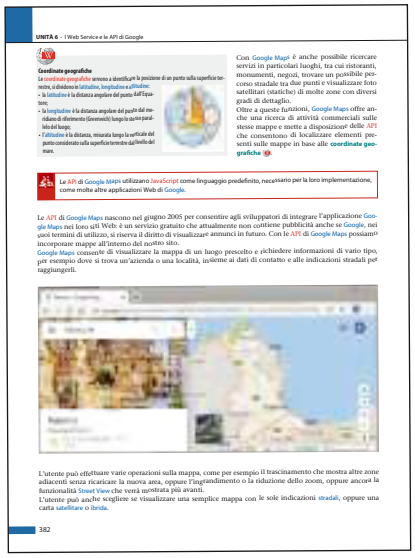
Sul sito **www.hoepliscuola.it** sono disponibili numerosi materiali. In particolare, per lo studente: lezioni integrative, approfondimenti, esercizi di recupero, rinforzo e approfondimento.
Anche i file richiamati nelle lezioni e nelle esercitazioni sono scaricabili dal sito.

# Struttura del corso per immagini

**APERTURA UNITÀ**
L'Unità si apre con l'indice delle lezioni sviluppate e l'indicazione degli obiettivi generali suddivisi in conoscenze, competenze e abilità.

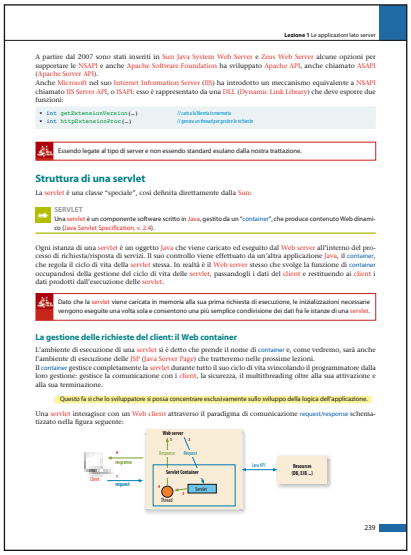



**APERTURA LEZIONE**
La Lezione si apre con una breve sintesi degli argomenti trattati e con la schematizzazione dei contenuti attraverso una MAPPA CONCETTUALE.

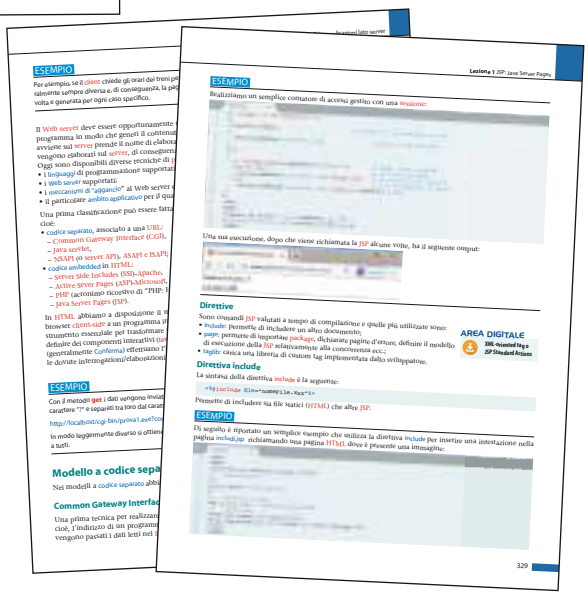


**WIKI**
Specifica il significato di un termine.

**ATTENZIONE**
Individua aspetti su cui focalizzare l'attenzione.



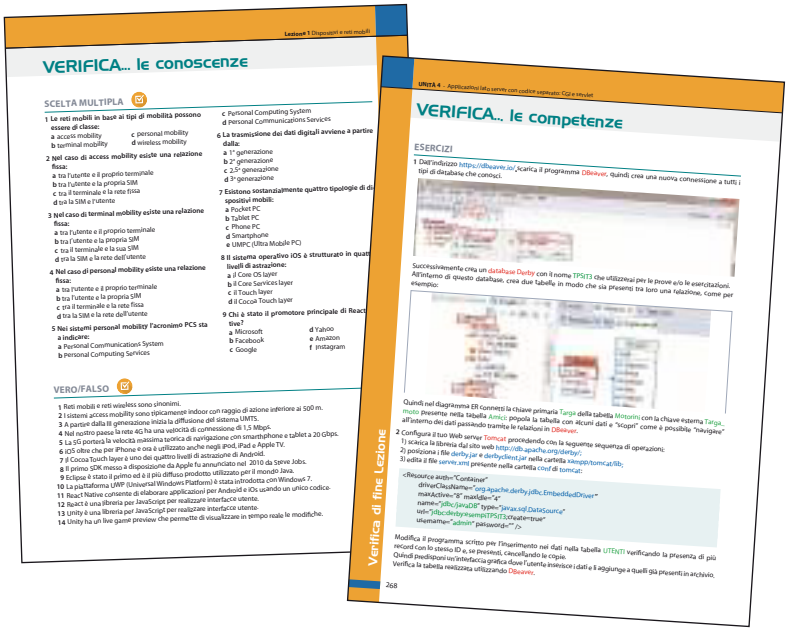
**DEFINIZIONI**
Spiegazione delle proprietà essenziali dei principali termini, funzioni e concetti trattati nel testo.



**ESEMPIO**
Gli esempi chiariscono i concetti appena esposti e svolgono la funzione di traccia di svolgimento per lo studente.

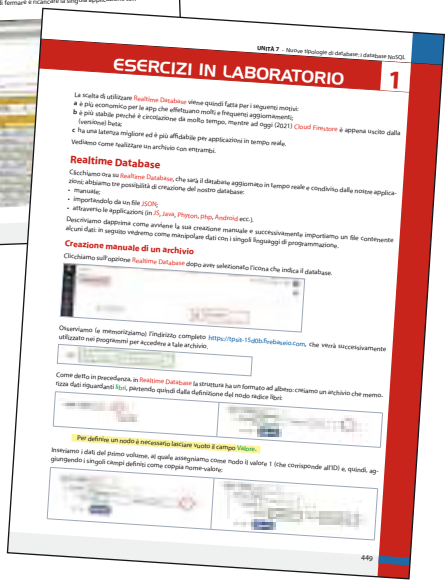
**EVIDENZIAZIONE**
Connota alcuni concetti da ricordare.

**PER SAPERNE DI PIÙ**
Schede di approfondimento degli argomenti sviluppati nel volume.

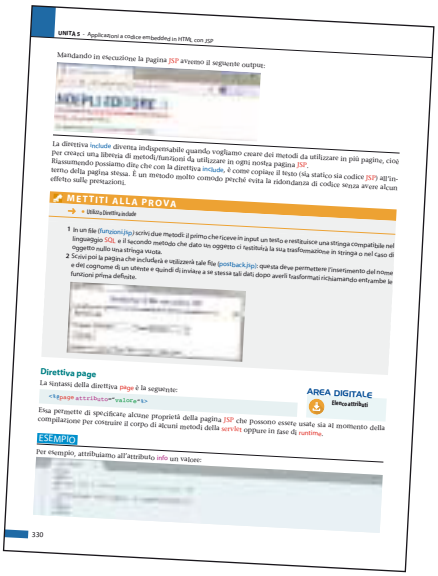



**VERIFICA LE CONOSCENZE / LE COMPETENZE**
Presenti a fine Lezione, propongono varie tipologie di test (quelli a risposta chiusa sono interattivi e autocorrettivi) e di esercizi a difficoltà crescente.

**ESERCIZI IN LABORATORIO**
Esercitazioni progettate per rafforzare i concetti assimilati, attraverso lezioni operative.




**METTITI ALLA PROVA**
Appendice esercitativa che prende spunto dal problema di partenza accrescendone le funzionalità e il campo applicativo.



# L'OFFERTA DIDATTICA
# HOEPLI

## EDIZIONE **OPENSCHOOL**

### LIBRO DI TESTO

Il libro di testo è l'**elemento cardine** dell'offerta formativa, uno strumento didattico **agile** e **completo**, utilizzabile autonomamente o in combinazione con il ricco **corredo digitale** offline e online.
Come da indicazioni ministeriali, volume cartaceo e apparati digitali **sono integrati in un unico percorso didattico**.
Le espansioni accessibili attraverso l'eBook+ e i materiali integrativi disponibili nel sito dell'editore sono puntualmente richiamati nel testo tramite apposite icone.

### E-BOOK +

L'**eBook+** è la versione digitale e interattiva del libro di testo, utilizzabile su **tablet, LIM e computer**.
Aiuta a comprendere e ad approfondire i contenuti, rendendo l'apprendimento più attivo e coinvolgente.
Consente di leggere, annotare, sottolineare, effettuare ricerche e accedere direttamente alle numerose **risorse digitali integrative**.
> Scaricare l'eBook+ è molto **semplice**.
È sufficiente seguire le istruzioni riportate nell'ultima pagina di questo volume.

### RISORSE ONLINE

Il sito della casa editrice offre una ricca dotazione di **risorse digitali** per l'approfondimento e l'aggiornamento.
Nella pagina web dedicata al testo è disponibile **My-BookBox**, il contenitore virtuale che raccoglie i materiali integrativi che accompagnano l'opera.
> Per accedere ai materiali è sufficiente registrarsi al sito **www.hoepliscuola.it** e inserire il codice coupon che si trova nell'ultima pagina di questo volume.
**Per il docente** nel sito sono previste ulteriori risorse didattiche dedicate.

### PIATTAFORMA

La **piattaforma didattica** è un ambiente digitale che può essere utilizzato in modo duttile, a misura delle esigenze della classe e degli studenti.
Permette in particolare di **condividere contenuti** ed **esercizi** e di partecipare a **classi virtuali**. Ogni attività svolta viene salvata sul **cloud** e rimane sempre disponibile e aggiornata.
La piattaforma consente inoltre di consultare la versione online degli eBook+ presenti nella propria libreria.
> Per informazioni sull'accesso alla piattaforma consulta **www.hoepliscuola.it/libri_digitali**

# Architettura di rete e formati per lo scambio dei dati

UNITÀ 1

**LEZIONE 1**
I sistemi distribuiti

**LEZIONE 2**
Evoluzione dei sistemi distribuiti e dei modelli architetturali

**LEZIONE 3**
La comunicazione nel Web con protocollo HTTP

**LEZIONE 4**
Le applicazioni Web e il modello client-server

**LEZIONE 5**
Le applicazioni di rete

**Esercizi in Laboratorio**

L1 Il linguaggio XML

L2 Lo schema DTD delle pagine XML

L3 La navigazione nei documenti con XML Path Language

L4 Il parsing XML con Java: le specifiche JAXP

L5 Il linguaggio JSON

L6 Web server Apache

L7 Web server Apache in Linux

L8 Web server IIS su Windows


## CONOSCENZE

- Conoscere gli stili architetturali fondamentali per sistemi distribuiti
- Comprendere il modello client-server
- Avere chiaro il concetto di elaborazione distribuita
- Conoscere il concetto di middleware
- Le caratteristiche del modello client-server
- L'evoluzione del modello client-server
- Avere chiaro il concetto di applicazione di rete
- Conoscere le specifiche JAXP

## COMPETENZE

- Saper riconoscere le diverse tipologie di sistemi distribuiti
- Saper classificare le architetture distribuite
- Individuare i benefici della distribuzione
- Confrontare la distribuzione con l'elaborazione concentrata
- Individuare le diverse applicazioni distribuite
- Saper classificare le applicazioni di rete
- Saper validare un documento XML
- Saper navigare in un documento XML
- Eseguire il parsing di documenti XML con Java

## ABILITÀ

- Scegliere i protocolli per le applicazioni di rete
- Definire strutture dati in XML
- Definire strutture dati in JSON
- Saper installare e configurare IIS
- Saper installare e configurare Apache e MySQL
- Scrivere e interpretare documenti XML
- Scrivere e interpretare documenti JSON

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

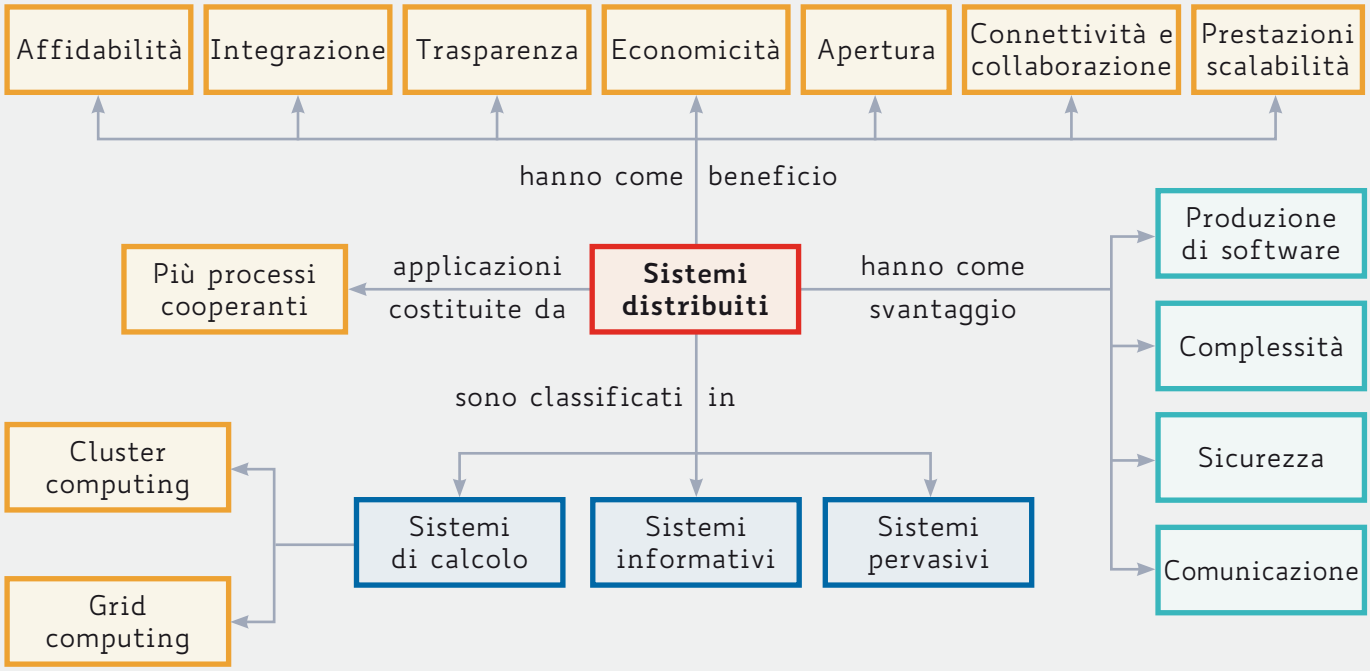
# 1 I sistemi distribuiti

**IN QUESTA LEZIONE IMPAREREMO...**

- il concetto di elaborazione distribuita
- i benefici della distribuzione
- gli svantaggi rispetto alla elaborazione concentrata

## MAPPA CONCETTUALE

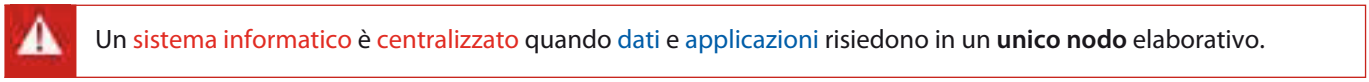
didattica inclusiva



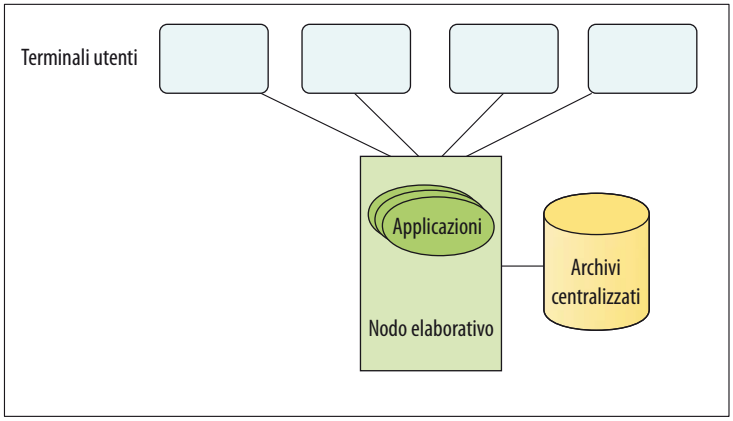

## I sistemi distribuiti

A metà degli anni Quaranta inizia l'era dei calcolatori elettronici moderni e a partire dalla metà degli anni Ottanta due sviluppi nella tecnologia hanno iniziato a cambiare la situazione: l'avvento dei microprocessori e la realizzazione di reti di elaborazione ottenute connettendo singoli computer per realizzare sistemi di calcolo complessi e potenti.
Le architetture dei sistemi informativi si sono sviluppate ed evolute nel corso degli anni successivi passando da schemi centralizzati a modelli distribuiti più aderenti alla necessità di decentralizzazione e cooperazione delle moderne organizzazioni.

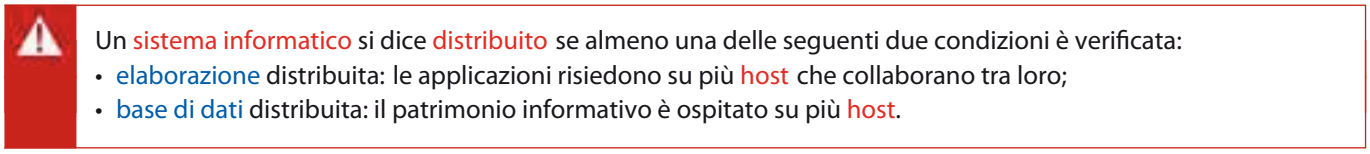
Nei sistemi centralizzati le applicazioni girano in un singolo processore, o comunque su un solo host, che costituisce l'unico componente autonomo nel sistema che è condiviso da vari utenti e tutte le risorse del componente sono sempre accessibili.



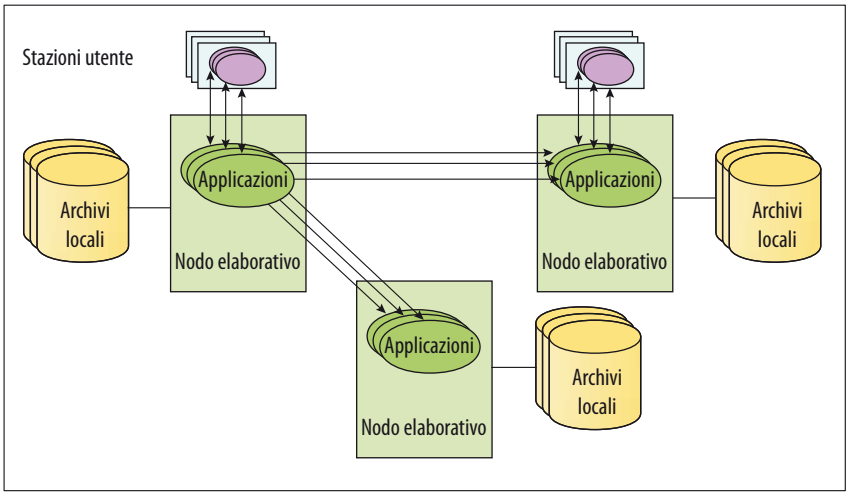
Un sistema informatico è centralizzato quando dati e applicazioni risiedono in un **unico nodo** elaborativo.




Nei sistemi distribuiti le applicazioni sono costituite da più processi, cooperanti, eseguiti in parallelo su un insieme di unità di elaborazione autonome: sono quindi sistemi ottenuti dall'aggregazione di singole CPU, sistemi di memorizzazione e periferiche.



Un sistema informatico si dice distribuito se almeno una delle seguenti due condizioni è verificata:
- elaborazione distribuita: le applicazioni risiedono su più host che collaborano tra loro;
- base di dati distribuita: il patrimonio informativo è ospitato su più host.




Numerose sono state le definizioni di sistema distribuito date a partire dagli anni Settanta ma nessuna di queste è pienamente soddisfacente: ognuna risente delle tecnologie e della visione prevalente (hardware o software) all'epoca in cui è stata formulata.
Riportiamo le più note.

### Sistema distribuito

1 Un sistema distribuito consiste di un insieme di calcolatori indipendenti che appaiono all'utente del sistema come un singolo calcolatore (Tanenbaum).
2 È un sistema in cui i componenti hardware o software posizionati in calcolatori collegati in rete comunicano e coordinano le proprie azioni solo tramite lo scambio di messaggi (Coulouris & Dollimore).
3 È un sistema in cui il fallimento di un calcolatore di cui nemmeno conosci l'esistenza può rendere inutilizzabile il tuo calcolatore (Lamport).

Nella nostra trattazione adotteremo la definizione più astratta riportata di seguito.

### Sistema distribuito

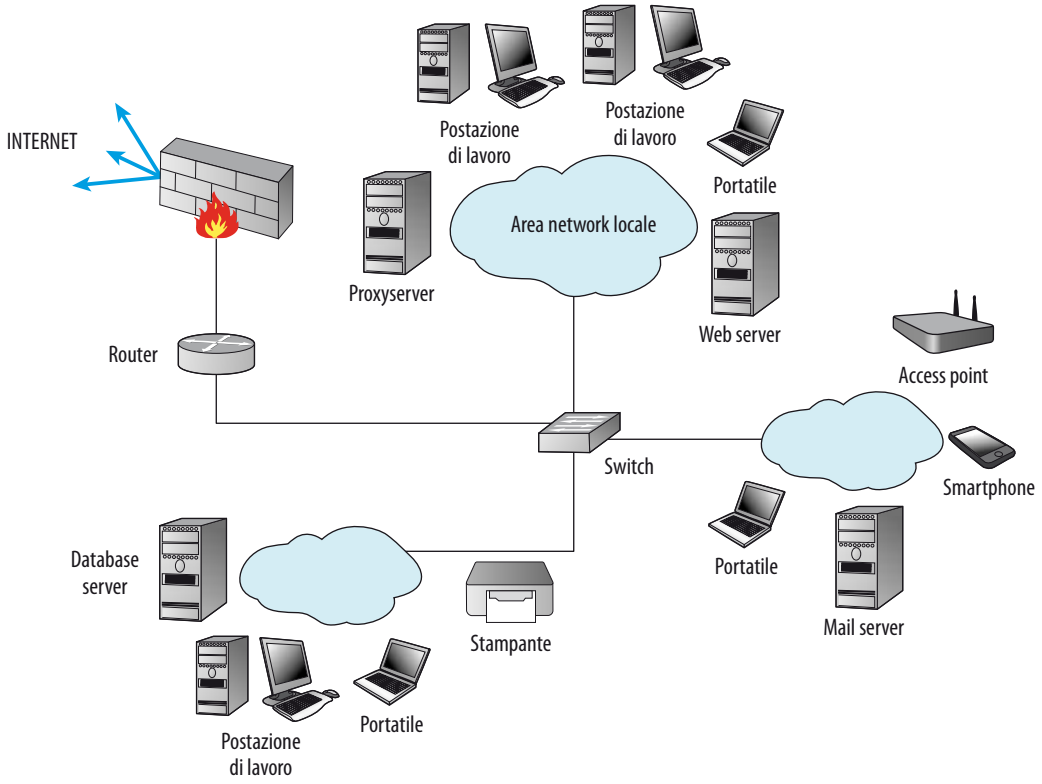
Un sistema distribuito è costituito da un insieme di applicazioni logicamente indipendenti che collaborano per il perseguimento di obiettivi comuni attraverso un'infrastruttura di comunicazione hardware e software.

Su ogni componente del sistema distribuito viene eseguito un programma che può essere differente sia per il compito che svolge che per il ruolo di elaborazione che, grazie a esso, assume nella rete. Alle applicazioni vengono dati quindi nomi diversi proprio in funzione dei diversi ruoli:

- cliente (client): un'applicazione assume il ruolo di cliente quando è utilizzatore di servizi messi a disposizione da altre applicazioni;
- servente (server): un'applicazione assume il ruolo di servente quando è fornitore di servizi usati da altre applicazioni;
- attore (actor): un'applicazione assume il ruolo di attore quando assume in diverse situazioni nel contesto del sistema sia il ruolo di cliente che quello di servente.

## ESEMPIO

Internet e il Web sono due esempi di sistemi distribuiti, ma non tutti i sistemi distribuiti sono collegati a Internet e Web e sono utilizzabili da tutti gli utenti, come per esempio le reti Intranet specifiche dei sistemi aziendali, sistemi di calcolo scientifico, server farm ecc.



Nella Lezione 4 affronteremo lo studio del modello architetturale che prende proprio il nome dai "ruoli dei suoi componenti", cioè modello client-server.

Esiste una "sottile" differenza tra sistemi distribuiti e sistemi paralleli:
- un sistema distribuito è un insieme di computer indipendenti che comunicano e possono cooperare per risolvere problemi;
- un sistema parallelo è un insieme di elementi di elaborazione che comunicano e cooperano per risolvere velocemente problemi di grandi dimensioni.

## Classificazione dei sistemi distribuiti

È possibile classificare i sistemi distribuiti in tre grandi famiglie:

1 sistemi di calcolo distribuiti che generalmente sono configurati per il calcolo ad alte prestazioni; nel seguito ne descriveremo due configurazioni:
- cluster computing: sono realizzati dalla connessione di computer omogenei tramite una rete allo scopo di parallelizzare l'elaborazione,
- grid computing: la cooperazione di calcolo può essere effettuata anche utilizzando macchine eterogenee;

2 sistemi informativi distribuiti; tra questi sistemi, oltre al Web che risulta essere il più grande sistema distribuito, le nuove tecnologie mobile hanno fatto da volano nell'evoluzione dei sistemi informativi tradizionali, che integrando **sistemi legacy** con nuove tecnologie di comunicazione, hanno generato i moderni sistemi informativi e i sistemi transazionali, dove ogni transazione consiste in un insieme di operazioni elementari incapsulate in due comandi di inizio-fine transazione;

3 sistemi distribuiti pervasivi che sono una nuova generazione di sistemi che hanno tipicamente connessioni di rete wireless e che generalmente sono sottoparti di sistemi più grandi; tra essi rientrano i sistemi domestici, le personal area network (PAN), le wearable computing e le reti di sensori.

**Sistemi legacy**
I legacy system (sistemi legacy) sono i sistemi presenti in azienda, realizzati su architetture basate su mainframe al quale si collegano terminali poco sofisticati, generalmente con interfaccia a caratteri. Si tratta di sistemi obsoleti, ereditati dalle prime fasi di informatizzazione dell'aziende che spesso rivestono ruoli critici e hanno dimensioni notevoli. Sebbene abbiano subito una lunga storia di interventi di manutenzione, non sono mai stati rimpiazzati: hanno la caratteristica di… funzionare correttamente(!).

## Benefici legati alla distribuzione

Tutti i grandi sistemi informatici sono oggi dei sistemi distribuiti e presentano come ogni sistema un insieme di vantaggi e di svantaggi che illustriamo sommariamente, senza specificare se sono dovuti all'hardware oppure al software.

### L'affidabilità

Il principale vantaggio dei sistemi distribuiti è l'affidabilità: grazie alla sua ridondanza intrinseca un SD è in grado di "sopravvivere" a un guasto di un suo componente; è comunque necessario prevenire queste situazioni e predisporre strumenti in grado di intervenire automaticamente al verificarsi di situazioni indesiderate, per esempio con l'implementazione di algoritmi che permettano alle entità non guaste di "sostituire" quella danneggiata e continuare l'elaborazione.

Non sempre però questa strada è percorribile: la difficoltà di realizzazione cresce con l'aumentare della autonomia delle entità.

I sistemi distribuiti possono essere suddivisi in due importanti categorie, ad accoppiamento debole o forte, in base alle risorse informative ed elaborative utilizzate, che nel primo tipo sono controllate da una o più unità organizzative comunque facenti riferimento a un unica autorità mentre nel secondo tipo sono soggetti organizzativi fra loro autonomi che hanno interesse a cooperare per fornire servizi a valore aggiunto per motivi istituzionali o di business.

## Integrazione

Un secondo vantaggio non meno importante del precedente è la capacità che ha un sistema distribuito di integrare componenti spesso eterogenei tra loro, sia per tipologia hardware (da PC a mainfame, da smatphone a tablet), sia per sistema operativo.
Ogni componente deve poter interfacciarsi allo stesso modo con il sottosistema di comunicazione del sistema distribuito ed è proprio l'interfaccia che permette di rendere "trasparenti" i componenti dello "strato inferiore" del sistema all'intera rete.
Di fondamentale importanza è la possibilità di connettere dispositivi di nuova generazione con legacy system che, di fatto, utilizzano tecnologie in altro modo incompatibili. È noto a tutti che con l'utilizzo della tecnologia Ethernet oppure con la piattaforma Web sistemi operativi e architetture hardware diverse possono interconnettersi e scambiarsi informazioni.

Sono stati definititi appositi linguaggi, come XML (eXtensible Markup Language) e notazioni JSON (JavaScript Object Notation) per favorire lo scambio di informazioni nel Web e permettere un'agevole ed efficiente pubblicazione di dati complessi: li descriveremo entrambi in seguito.

## Trasparenza

Come trasparenza si intende il concetto di "vedere" il sistema distribuito non come un insieme di componenti ma come un unico sistema di elaborazione: l'utente non deve accorgersi che sta interagendo con un sistema distribuito ma deve avere la percezione di utilizzare un singolo elaboratore, senza percepire eventuali mutamenti che si verificano nel tempo nel sistema.
L'ANSA nell'ISO 10746, Reference Model of Open Distributed Processing, identifica otto forme di trasparenza:

- di accesso (*access transparency*): si devono nascondere le differenze nella rappresentazione dei dati e nelle modalità di accesso alle risorse, in modo da permettere di accedere a risorse locali e remote con le stesse operazioni, in modo unico e uniforme;
- di locazione (*location transparency*): si deve nascondere dove è localizzata una risorsa e permettere di accedere a essa senza conoscerne la locazione (per esempio, l'URL nasconde l'indirizzo IP);
- di concorrenza (*concurrency transparency*): si permette ai processi di operare in maniera concorrente, cioè facendo in modo che l'accesso concorrente a una risorsa condivisa la lasci sempre in uno stato consistente, implementando per esempio meccanismi di locking;
- di replicazione (*replication transparency*): le operazioni di duplicazione vengono effettuate all'insaputa degli utenti mantenendo gli stessi nomi per ogni risorsa, in modo da avere copie di risorse che aumentano l'affidabilità e le prestazioni del sistema;
- ai guasti (*failure transparency*): viene mascherato il guasto e il recovery di una risorsa;
- alla migrazione (*mobility transparency*): si nasconde l'eventuale spostamento (logico o fisico) di una risorsa senza interferire sulla sua modalità di accesso, cioè senza influenzare le operazioni degli utenti che la riguardano;
- alle prestazioni (*performance transparency*): vengono nascoste le operazioni necessarie per riconfigurare il sistema al variare del carico per migliorarne le prestazioni;
- di scalabilità (*scaling transparency*): il sistema viene espanso senza interrompere o modificare il funzionamento.

I più importanti sono i primi due, cioè la trasparenza di accesso e di locazione.

## Economicità

Generalmente i sistemi distribuiti offrono spesso un miglior rapporto prezzo/qualità dei sistemi centralizzati basati su mainframe: una rete di PC connessi ha un prezzo di alcuni ordini di grandezza inferiore rispetto a quello di un mainframe e con le tecnologie odierne la capacità computazionale è paragonabile. Inoltre, come già abbiamo detto, introduce la possibilità di connettere con un costo molto basso i sistemi legacy in modo da non dover abbandonare una tecnologia su cui è stato realizzato un certo investimento e poter convivere con la tecnologia più recente.

### Apertura

Con la definizione di protocolli standard si favorisce l'apertura all'hardware e software di fornitori diversi in modo da avere:

- interoperabilità: implementazioni diverse su elaboratori di diverso tipo possono coesistere per comporre un unico sistema;
- portabilità: un'applicazione sviluppata su un sistema operativo può funzionare su di un altro presentando la medesima interfaccia all'utente in modo da non modificare l'operatività;
- ampliabilità: è relativamente semplice "far crescere" il sistema aggiungendo sia componenti hardware che software.

### Connettività e collaborazione

Nei sistemi distribuiti la possibilità di condividere risorse hardware e software comporta vantaggi economici. Per esempio è possibile condividere apparecchiature speciali di costo elevato, quali stampanti particolari, plotter, oppure sistemi di storage e di recovery. Lo scambio e la condivisione di informazioni arricchisce ogni utilizzatore: per esempio vengono creati groupware con specifici interessi.

### Prestazioni e scalabilità

Un sistema distribuito può essere composto semplicemente da due workstation e un file server oppure da una LAN che può contenere anche molti server, migliaia di workstation, molte stampanti ecc. La crescita di un sistema distribuito con l'aggiunta di nuove risorse e, quindi, di nuovi servizi, fornisce a tutti i suoi componenti un miglioramento delle prestazioni e permette di sostenere l'aumento del carico di richieste: questa possibilità prende il nome di scalabilità orizzontale.
Con la scalabilità si superano i problemi visti nel caso di un sistema centralizzato dove spesso alcune risorse condivise sono disponibili in misura limitata: nei SD concettualmente non esiste una limitazione nel numero di risorse disponibili, basta che il sistema abbia la "possibilità di espandersi".

### Tolleranza ai guasti

La possibilità di replicare risorse offre una certa garanzia di tolleranza ai guasti; la presenza di un componente guasto non deve pregiudicare il funzionamento del sistema ma, al limite, introdurre solo inefficienze in termini di tempo di risposta.

In questa situazione si parla di "guasto parziale", cioè di un guasto che non influenza pesantemente il funzionamento del sistema.

Un primo approccio è quello di duplicare interamente le risorse, però questa è una soluzione spesso molto costosa. Quindi è opportuno optare per un sistema software di recupero, che sia in grado di riportare il sistema in uno stato consistente precedente e di trovare un'alternativa al componente guasto.

## Svantaggi legati alla distribuzione

I sistemi distribuiti hanno però anche una serie di svantaggi che le nuove tecnologie cercano di ridurre o eliminare in fase di progettazione dei nuovi dispositivi di rete. Descriviamo sinteticamente i principali problemi.

### Produzione di software

I programmatori del "secolo scorso" hanno dovuto modificare il proprio stile di programmazione e aggiornarsi con lo studio dei nuovi linguaggi e dei nuovi strumenti di sviluppo per poter realizzare applicazioni distribuite. Il passaggio è avvenuto in tre tappe:

- la definizione dello standard TCP/IP (e quindi della modalità di comunicazione tramite socket) è divenuta la base per lo sviluppo di applicazioni di rete;

- lo sviluppo di architetture Web e di tutti i linguaggi a esse connesse, sia lato client come HTML, CSS, JavaScritp, sia lato server, come CGI, servlet, JSP, PHP, ASP ecc. che hanno permesso di far effettuare un salto di qualità alla produzione del software;
- il diffondersi del linguaggio Java che, grazie alla sua tecnologia a *macchina virtuale*, permette ai programmi di essere eseguiti su macchine anche completamente differenti tra loro.

## Complessità

Proprio per la struttura hardware i sistemi distribuiti sono più complessi di quelli centralizzati: richiedono strumenti per l'interconnessione degli host e tecniche per l'instradamento corretto dei messaggi e dei dati. È anche più complesso valutare le performance di un sistema distribuito piuttosto che di un unico elaboratore.

## Sicurezza

Con la connessione di più host tra loro si crea la possibilità di accedere a dati e risorse anche a chi non ne ha il diritto: nascono nuove problematiche connesse alla sicurezza che nel caso di sistemi centralizzati erano inesistenti. Infatti, nei vecchi sistemi per lo più bastava proteggere il sistema dall'accesso fisico delle persone ai locali dove erano presenti i dispositivi da proteggere (hard disk e supporti di memorizzazione). Oggi l'accesso avviene via etere e via cavo e anche le trasmissioni sono soggette a rischio di intercettazione (sniffing) e quindi richiedono l'applicazione di appositi accorgimenti per tutelare tutti gli utenti e garantire sicurezza e riservatezza nei dati, sia memorizzati sui proprio computer personali, sia trasmessi per transazioni commerciali o semplicemente personali (email).

## Comunicazione

Il trasferimento a distanza delle informazioni richiede nuove tipologie di sistemi di telecomunicazione, sia cablati sia wireless, e l'aumento esponenziale degli utenti fa sì che giornalmente aumenti la richiesta di bande trasmissive, anche per migliorare la qualità del servizio offerto e offrire nuove tipologie di applicazioni sempre più performanti (alta velocità, alta definizione, video streaming ecc.). La difficoltà maggiore sta nel fatto che si è sempre di fronte a nuove problematiche delle quali non si ha "ben chiaro" né il dominio di definizione ne l'ambito delle soluzioni: la mancanza di prevedibilità delle richieste genera situazioni "casuali" di carico e quindi di risposta del sistema che possono essere molto diverse anche a breve distanza di tempo. A volte l'utilizzo di tecniche "sperimentali" porta all'aumento di complessità in modo accidentale anche a causa del fatto che le tecniche di analisi e progettazione sono spesso basate su vecchi metodi di sviluppo di applicazioni monoprocesso, non adeguate alle situazioni di sistema distribuito.

# VERIFICA... le conoscenze

**AREA DIGITALE**
Esercizi per il recupero e l'approfondimento

## SCELTA MULTIPLA

**1 In un sistema informatico distribuito almeno una delle seguenti condizioni è verificata** (*2 risposte*)**:**
**a** elaborazione distribuita
**b** calcolatori distribuiti
**c** base di dati distribuita
**d** informazioni distribuite

**2 Quale delle seguenti affermazioni per i sistemi legacy sono false?**
**a** Sono quelli presenti in azienda realizzati generalmente su architetture basate su mainframe
**b** A essi si collegano terminali poco sofisticati, generalmente con interfaccia a caratteri
**c** Spesso rivestono ruoli critici e hanno dimensioni notevoli
**d** Devono essere rimpiazzati perché iniziano a dare problemi di funzionamento

**3 Quali tra le seguenti forme di trasparenza non sono previste dall'ANSA nella ISO 10746?** (*2 risposte*)
**a** di accesso (access transparency)
**b** di locazione (location transparency)
**c** di concorrenza (concurrency transparency)
**d** di prezzo (price transparency)
**e** di replicazione (replication transparency)
**f** ai guasti (failure transparency)
**g** di sicurezza (security transparency)
**h** alla migrazione (mobility transparency)
**i** alle prestazioni (performance transparency)
**j** di scalabilità (scaling transparency)

**4 Le forme di trasparenza più importanti sono:**
**a** accesso e locazione
**b** locazione e concorrenza
**c** accesso e concorrenza
**d** locazione e scalabilità
**e** accesso e scalabilità

**5 I principali svantaggi legati alla distribuzione sono** (*più risposte*)**:**
**a** produzione di software
**b** complessità
**c** affidabilità
**d** sicurezza
**e** costo

## VERO/FALSO

1 In un sistema informatico centralizzato dati e applicazioni risiedono in un unico nodo. V F
2 Un sistema distribuito è anche un sistema parallelo. V F
3 Il principale vantaggio dei sistemi distribuiti è l'affidabilità dovuta alla sua ridondanza intrinseca. V F
4 Con sistemi legacy si intendono i sistemi informativi obsoleti presenti in azienda. V F
5 Generalmente i sistemi distribuiti offrono un miglior rapporto prezzo/qualità che i sistemi centralizzati. V F
6 Nei vecchi sistemi non era sufficiente proteggere il sistema dall'accesso fisico delle persone ai locali dove erano presenti i dispositivi da proteggere. V F
7 L'utilizzo di tecniche di sviluppo "sperimentali" porta all'aumento di complessità in modo accidentale. V F

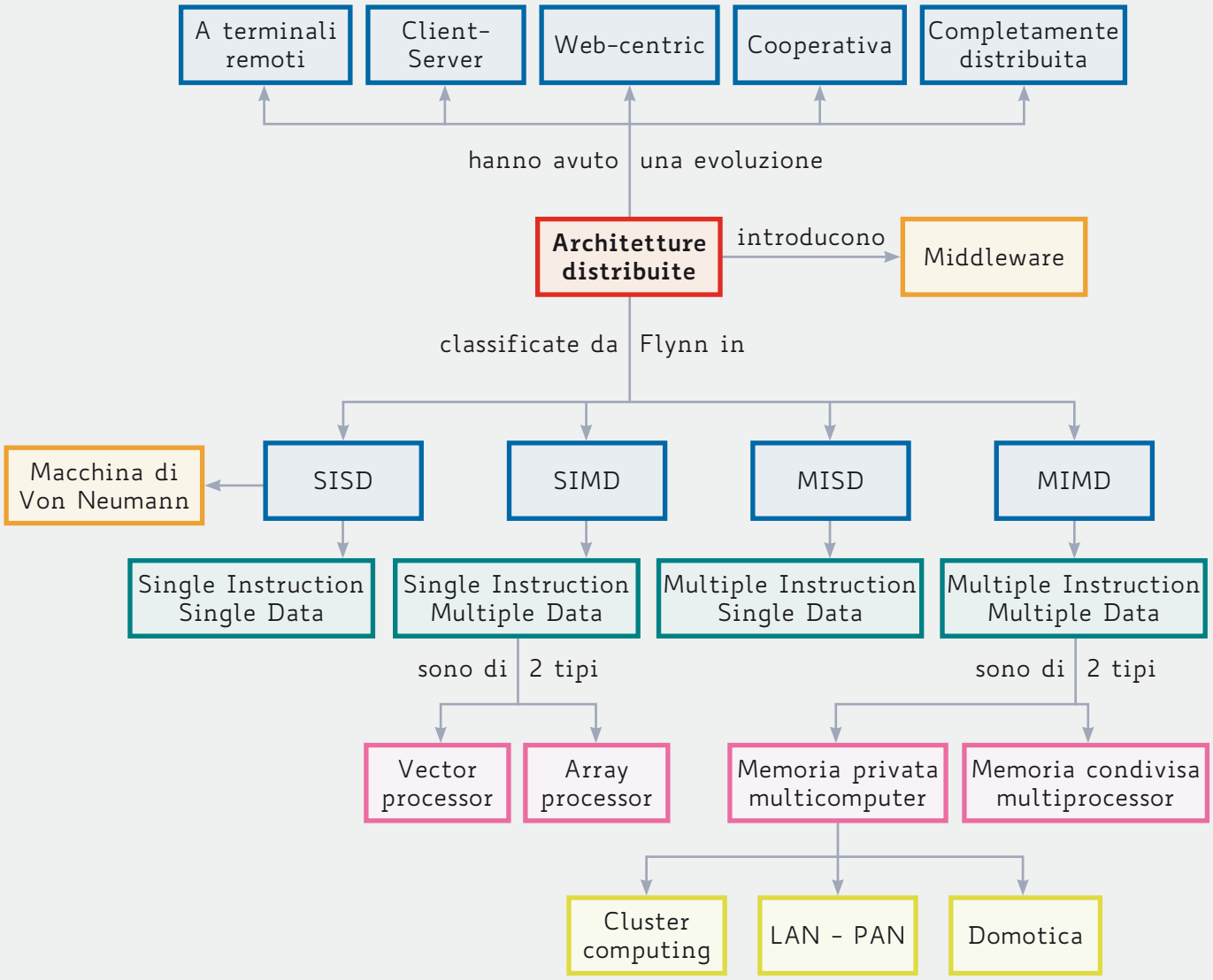
# 2 Evoluzione dei sistemi distribuiti e dei modelli architetturali

**IN QUESTA LEZIONE IMPAREREMO...**

- la classificazione delle architetture distribuite hardware e software
- il concetto di middleware

## MAPPA CONCETTUALE

didattica inclusiva



## Premessa

Un sistema distribuito è ottenuto dalla connessione di più personal computer che per molti anni hanno avuto una crescita tecnologica legata alla legge di Moore ma che nel prossimo futuro sarà disattesa in quanto non sarà più possibile raddoppiare la potenza degli elaboratori ogni due anni.

> Il problema maggiore è quello legato alla miniaturizzazione dei microprocessori e alla crescita delle frequenze di lavoro!

Attualmente i microprocessori più potenti hanno circuiti larghi 14 nanometri (un miliardesimo di metro) e si sta lavorando per farne di ancora più piccoli, ma ci sono limiti fisici con i quali si dovrà fare i conti quando si arriverà nell'ordine dei 2-3 nanometri dato che a quelle dimensioni i componenti sarebbero larghi quanto una decina di atomi!
Le leggi della fisica pongono dei limiti anche a partire dal fatto che non è possibile superare la velocità della luce e quindi, dato che questa nel rame è di circa 200.000 km/s, si ha:

$$v = \frac{200000 \square 10^3}{10^9} = 0,2 \text{ [m/ns]}$$

Questo significa che nel tempo di 1 ns vengono percorsi 20 cm di conduttore.
L'unica soluzione è quelle di andare quindi verso architetture di elaborazione di diverso tipo, sia dal punto di vista costruttivo (hardware) cha del punto di vista logico (software).

> Se vogliamo avere frequenze di lavoro dell'ordine dei Ghz non possiamo superare la distanza di 20 cm in trasmissione altrimenti verrebbe generato un ritardo e se si restringono i circuiti avremmo problemi di instabilità e di surriscaldamento in quanto il calore prodotto viene difficilmente dissipato in una architettura così densa.

## Architetture distribuite hardware: dalle SISD al cluster di PC

Dal punto di vista hardware si è passati alla realizzazione di macchine e sistemi dotati di più di una CPU in modo da avere più potenza di calcolo senza "esasperare" i limiti di velocità di ogni singola CPU: stiamo parlando di macchine parallele o macchine ad architettura parallela.
Esistono diverse possibilità per classificare le architetture hardware a seconda dei fattori che si prendono come riferimento. Noi ricordiamo quella di Flynn (1972) che si basa sui due flussi di informazioni normalmente presenti nei calcolatori:

- flusso delle istruzioni;
- flusso dei dati.

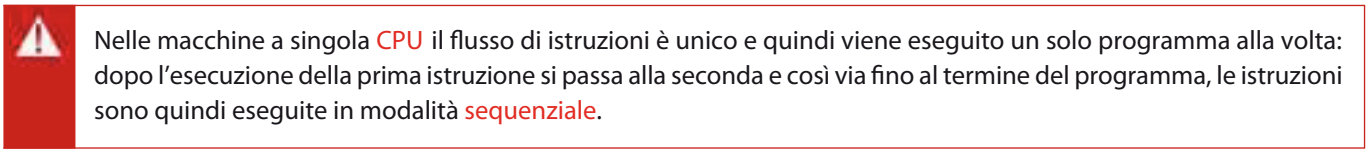
A seconda di come si "combinano" flusso di dati e di istruzioni abbiamo quattro possibili situazioni:

| | DATI SINGOLI | DATI MULTIPLI |
|---|---|---|
| Istruzioni singole | SISD | SIMD |
| Istruzioni multiple | MISD | MIMD |

- macchine SISD (Single Instruction Single Data): flusso di istruzioni unico e flusso di dati unico;
- macchine SIMD (Single Instruction Multiple Data): flusso di istruzioni unico e flusso di dati multiplo;
- macchine MISD (Multiple Instruction Single Data): flusso istruzioni multiplo e flusso dati unico;
- macchine MIMD (Multiple Instruction Multiple Data): flusso istruzioni multiplo e flusso dati multiplo.

### SISD

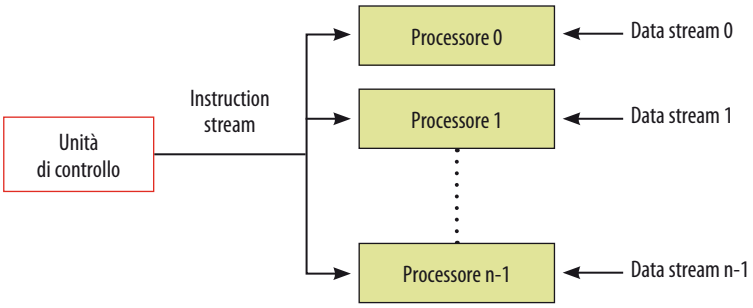
Un elaboratore come la macchina di Von Neumann che ha un solo flusso dati e un solo flusso istruzioni è una macchina Single Instruction stream – Single Data stream (SISD): tutte le macchine che hanno una singola CPU come i personal computer, le workstation e i mainframe appartengono a questa categoria.



Nelle macchine a singola CPU il flusso di istruzioni è unico e quindi viene eseguito un solo programma alla volta: dopo l'esecuzione della prima istruzione si passa alla seconda e così via fino al termine del programma, le istruzioni sono quindi eseguite in modalità sequenziale.

## SIMD

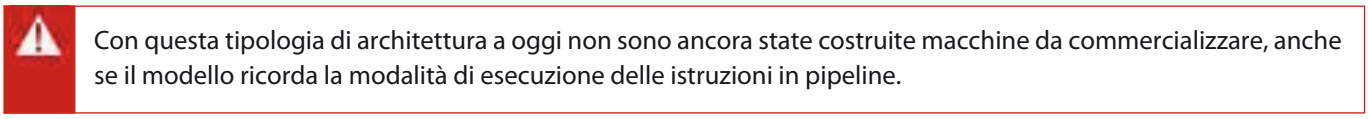
Nelle macchine SIMD l'elaborazione avviene su più flussi dati in contemporanea ma con un singolo flusso di istruzioni: generalmente sono presenti più processori (macchine a vector processor o ad array processor) che eseguono la stessa istruzione su flussi di dati diversi (si sfrutta il parallelismo a livello di dati).




Un elaboratore SIMD è particolarmente adatto per realizzare calcoli vettoriali e matriciali soprattutto in ambiente grafico.

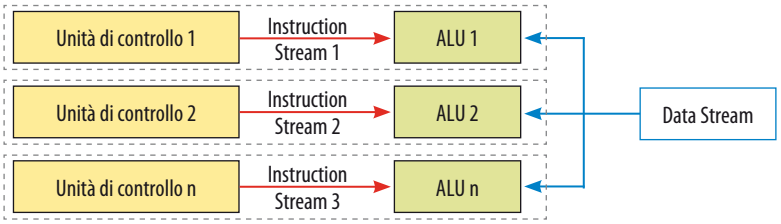
## MISD

A questa categoria appartengono gli elaboratori che eseguono più istruzioni sullo stesso flusso dati, cioè Multiple Instruction stream-Single Data stream (MISD): nel modello di calcolo MISD esistono quindi più processori, ognuno con una propria memoria (registri), la quale a sua volta avrà un proprio flusso di istruzioni che verranno eseguite sullo stesso flusso di dati.



Con questa tipologia di architettura a oggi non sono ancora state costruite macchine da commercializzare, anche se il modello ricorda la modalità di esecuzione delle istruzioni in pipeline.

Avere a disposizione una macchina con questa caratteristiche potrebbe essere utile nell'ambito della crittografia, dove vengono richieste elaborazioni simili da eseguirsi sugli stessi dati in parallelo.



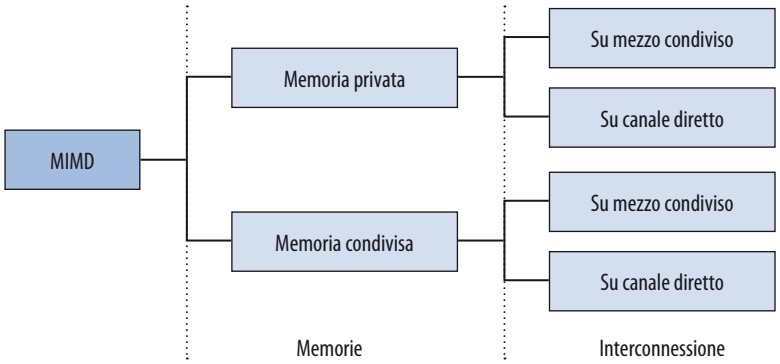

## MIMD

L'architettura Multiple Instruction stream-Multiple Data stream (MIMD) comprende tutte le tipologie di elaboratori composti da più unità centrali di elaborazione indipendenti che possono lavorare su stream di dati anch'essi indipendenti. Viene effettuata una ulteriore classificazione delle macchine MIMD in riferimento a come è suddivisa la memoria fisica, e cioè possiamo avere:

- macchine MIMD a memoria fisica condivisa;
- macchine MIMD a memoria privata.

Le prime sono anche conosciute con il nome di multiprocessor, mentre le seconde con quello di multicomputer.

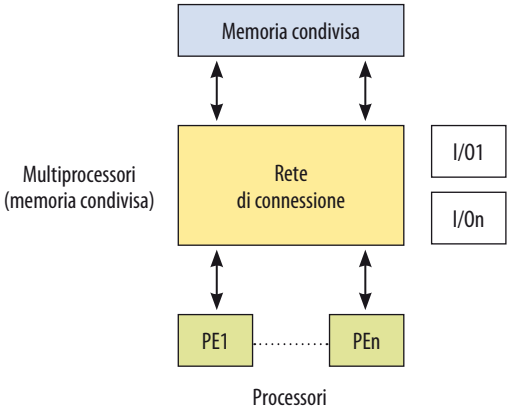
È possibile introdurre una successiva classificazione dei multicomputer e multiprocessor a seconda di come sono interconnesse le unità di elaborazione, cioè se utilizzano un unico mezzo fisico condiviso (per esempio a bus oppure ad anello) o attraverso un canale diretto fisico tra coppie di unità centrali: la classificazione completa è riportata nella figura che segue.




In queste macchine sia il flusso delle istruzioni che dei dati è multiplo: ogni processore legge le proprie istruzioni e opera sui propri dati e quindi sono presenti contemporaneamente più processi in esecuzione. Il parallelismo è fisico ed è molto flessibile dato che ogni singola CPU evolve in modo autonomo.

### MIMD: multiprocessori

I sistemi a multiprocessori sono architetture MIMD a memoria condivisa (shared memory): la memoria è in comune, e quindi esiste un unico spazio di indirizzamento condiviso tra tutti i processori.

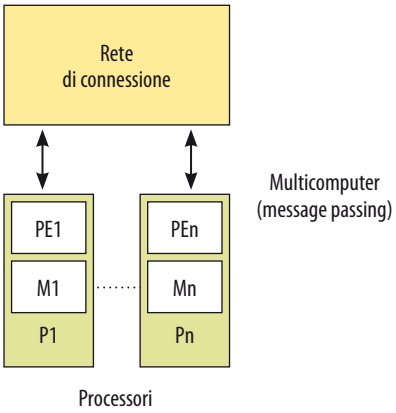



Dato che la comunicazione tra processi avviene mediante variabili condivise è necessario implementare gli opportuni meccanismi di sincronizzazione per regolare gli accesi alla memoria in modo da coordinare i diversi processi per gestire la competizione alle risorse comuni.

Non necessariamente i processori devono avere una unica memoria in comune (centralizzata): possono avere ciascuno una propria memoria e condividerne parte con gli altri processori, in modo da realizzare una memoria condivisa distribuita.

### MIMD: multicomputer

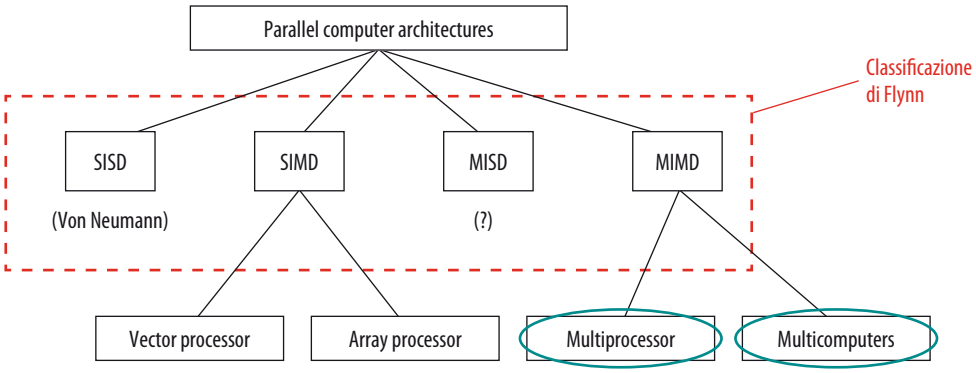
Nei multicomputer non è presente una memoria condivisa a livello di architettura e quindi la comunicazione avviene mediante lo scambio di messaggi (message passing) esplicito, effettuato mediante apposite procedure (send e receive). Ogni computer possiede una propria area di memoria privata, non indirizzabile da parte dei processori remoti.




## ESEMPIO

Le LAN di un personal computer sono da considerarsi sistemi MIMD dato che hanno una memoria privata e sono tra loro interconnessi con mezzo condiviso.

Possiamo ora estendere la classificazione di Flynn inserendo anche le sottoclassificazioni:




## Cluster computing

Con cluster computing si intende un sistema distribuito costituito da un insieme di nodi ad alte prestazioni interconnessi tramite una rete locale ad alta velocità: devono essere omogenei, cioè i singoli nodi hanno lo stesso sistema operativo, hardware molto simile, e sono connessi attraverso la stessa rete.
Teoricamente un cluster di PC ha una potenza di calcolo pari alla somma di quelle dei singoli computer che lo costituiscono, e differisce da una rete di PC principalmente per:

- la potenza di elaborazione ad alte prestazioni (HPC o High Performance Computing);
- la velocità del trasferimento dati (oltre 1 Gbit/s);
- la centralizzazione fisica delle macchine, tutti i PC sono montati sullo stesso rack;
- la presenza di una applicazione di management, residente su un singolo PC, che permette di lanciare processi su altri PC, monitorare il loro comportamento ecc.

**AREA DIGITALE**
**Creare cluster HPC con ParallelKnoppix**

Abbiamo due tipiche possibili architetture:

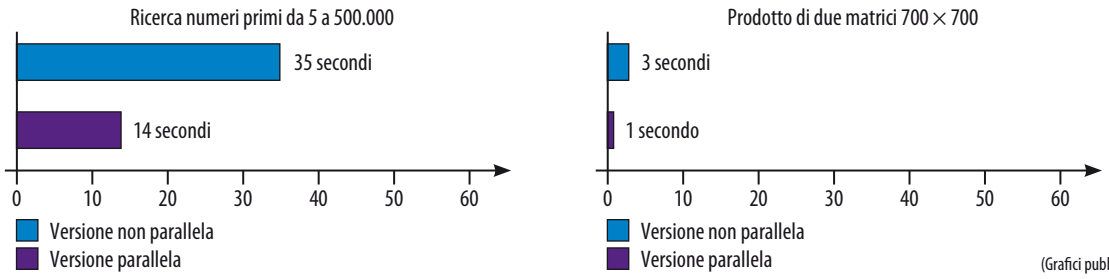
1 organizzazione gerarchica con singolo nodo principale: per esempio Beo/wulf, dove spesso sono usate librerie di message passing per il calcolo parallelo (MPI);
2 organizzazione Single System Image: per esempio MOSIX, che effettua un bilanciamento automatico del carico effettuando una eventuale migrazione di processi.

In base alla definizione e considerando le unità centrali come entità, un sistema cluster di PC corrisponde all'insieme delle macchine MIMD a memoria privata.

L'enorme vantaggio dei cluster di PC è quello di affrontare calcoli particolarmente onerosi che sarebbero molto lunghi o impossibili con un solo computer.

## ESEMPIO

Vediamo per esempio il confronto di prestazioni ottenuto con il calcolo parallelo effettuato su un cluster di 4 PC in rispetto all'elaborazione sequenziale in due classiche situazioni di calcolo matematico: *ricerca di numeri primi e il prodotto di matrici*.
Come è possibile osservare dai grafici riportati di seguito, nella ricerca dei numeri primi si ha una riduzione del tempo di elaborazione da 35 secondi a 14 secondi, mentre per il prodotto di due matrici di ordine 700 il tempo di computazione scende da 3 secondi a 1 secondo!




(Grafici pubblicati su Rottanuova)

## Grid computing

Con grid computing si intende un sistema distribuito di calcolo altamente decentralizzato, composto da un gran numero di nodi disposti a griglia (grid) e caratterizzati da un grado elevato di eterogeneità sia per hardware, per il software, la tecnologia di rete, le politiche di sicurezza ecc.
Il termine "griglia" è stato coniato intorno alla metà degli anni Novanta: il vero problema alla base del concetto di griglia è la condivisione coordinata di risorse all'interno di una dinamica e multi-istituzionale organizzazione virtuale (Virtual Organization, brevemente indicata con VO).

> La condivisione non è limitata solo allo scambio dei file, ma si estende all'accesso diretto a computer, al software, in generale a tutto l'hardware necessario alla risoluzione di un problema scientifico, ingegneristico o industriale.

Si è passati da un'architettura a 4 livelli di specificità alla sua evoluzione service-oriented che ha portato alla definizione della Open Grid Service Architecture (OGSA) dove risorse computazionali, di storage, reti, programmi, database ecc., sono tutti dei servizi (grid services).

## Sistemi distribuiti pervasivi

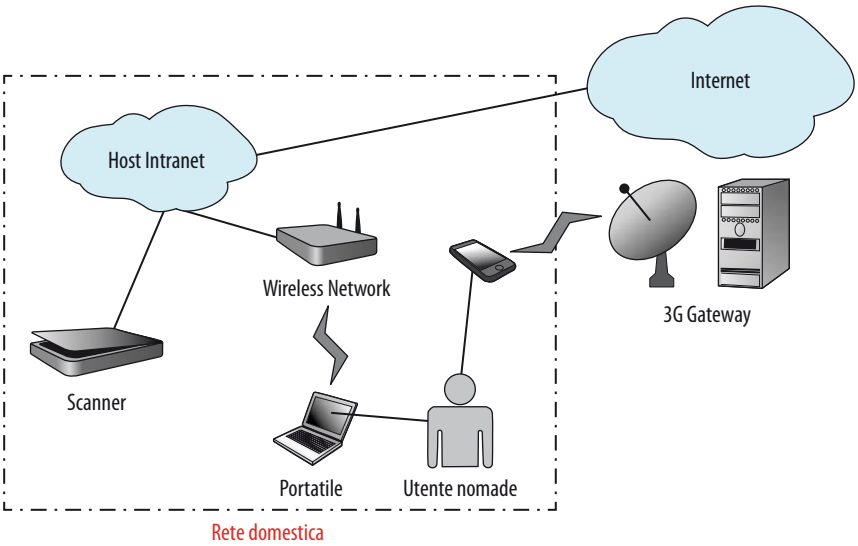
Questa è una nuova generazione di SD i cui nodi sono piccoli, mobili, con connessioni di rete wireless e spesso facenti parte di un sistema più grande:
- sistemi domestici, sistemi elettronici per l'assistenza sanitaria;
- reti di sensori.

Alcuni requisiti per sistemi pervasivi:
- *cambi di contesto*: il sistema è parte di un ambiente che può cambiare in ogni momento;
- *composizione ad hoc*: ogni nodo può essere usato in modi molto diversi da utenti differenti;
- richiesta la facilità di configurazione;
- *condivisione come default*: i nodi vanno e vengono, fornendo informazioni e servizi da condividere.

Pervasività e trasparenza della distribuzione non sono facilmente coordinabili: è preferibile esporre la distribuzione piuttosto che cercare di nasconderla.

## Reti domestiche e domotica

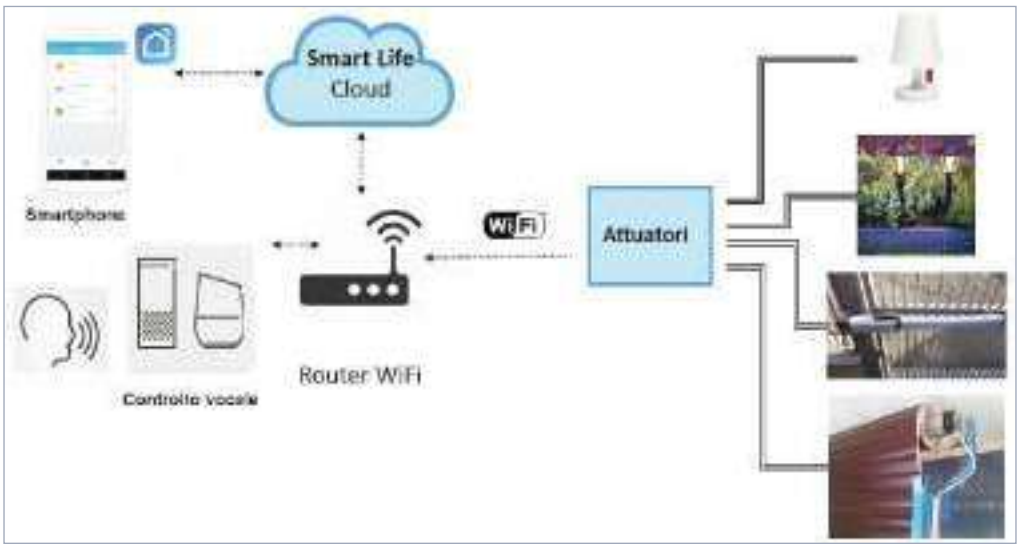
Le reti domestiche sono caratterizzate dall'assenza di un amministratore di sistema e generalmente sono sistemi completamente auto-configuranti e autogestiti in quanto generalmente gli utenti non hanno conoscenze specifiche di connettività.
La soluzione più semplice e quella di una home box centralizzata alla quale si connettono wireless tutte le periferiche ed eventuali dispositivi "ospiti".



L'espansione d'ambito applicativo che sta investendo l'IT con l'interesse crescente verso l'integrazione del mondo fisico con quello digitale ha dato un forte impulso innovativo alla domotica favorendone una rapida evoluzione tecnologica. Anche le esigenze dei consumatori hanno fatto segnare una aumentata richiesta di intelligenza della casa e l'attesa di ritrovare negli impianti domestici gli stessi pattern di interazione e semplicità tipici degli smartphone.
La domotica è un neologismo nato da "*domus*" (casa) e "automatica" e si occupa delle tecnologie capaci di migliorare la qualità di vita all'interno della casa.
Un modello che meglio descrive l'impianto domotico contemporaneo è quello introdotto da D.J.Cook ovvero l'Ambient Intelligence proposto con la definizione che segue:

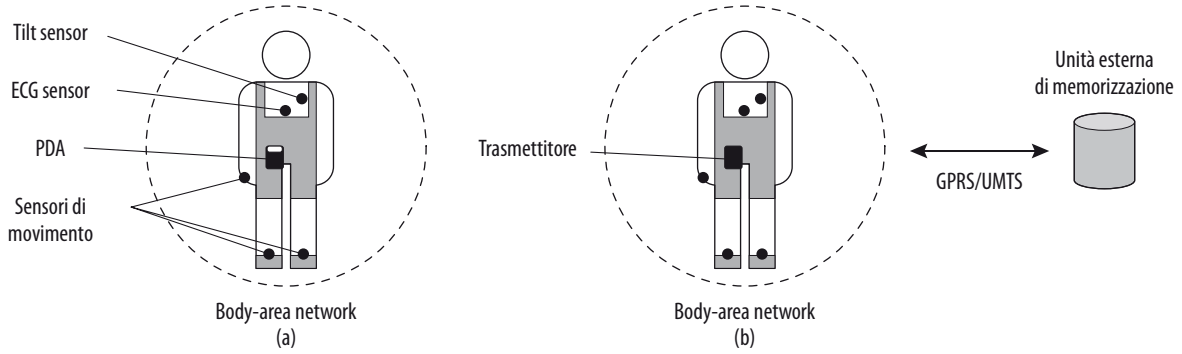
"an Ambient Intelligence system is a digital environment that proactively, but sensibly, supports people in their lives"

che possiamo tradurre liberamente come: "un sistema che proattivamente, ma in modo ragionato, supporta le persone nella loro vita".
La domotica consente di ottenere un notevole incremento delle prestazioni e delle possibilità offerte dai diversi impianti presenti nell'abitazione, ottimizzando i consumi e permettendo l'integrazione di diverse funzioni quali controllo, comfort, sicurezza, risparmio energetico e comunicazione.
Generalmente una rete domotica è senza fili e comunica con tutti i dispositivi compatibili presenti fra le mura domestiche: l'utilizzatore può quindi automatizzare tutte quelle operazioni che svolge quotidianamente come, per esempio, accendere le luci, alzare le tapparelle, regolare la temperatura della propria abitazione o gestire l'allarme domestico sia da smartphone sia con le nuove tecnologie di controllo vocale (Alexa, Domoki, Apple Home, Google Home ecc.).

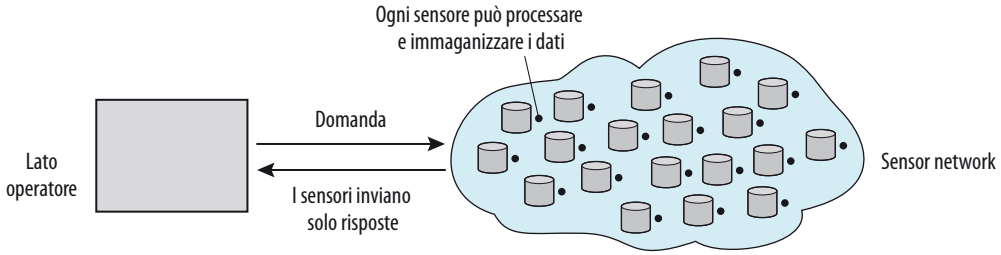



## Wearable computing

Oggi i sistemi wearable computing sono essenzialmente utilizzati per l'assistenza sanitaria ed effettuano il monitoraggio di parametri biologici o memorizzando i dati in un computer palmare (PDA *Personal Digital Assistant*) come nell'esempio riportato nel disegno a) oppure trasmettendo i dati a un sistema di archiviazione remoto, come nel caso b).




I principali problemi che vanno affrontati in questi sistemi, oltre alla memorizzazione dei dati è la prevenzione contro la perdita di dati cruciali, la gestione della sicurezza e le modalità di generazione e propagazione di eventuali allarmi.
Le reti di sensori sono composte da molteplici semplici sensori (da 10 a 1000) e ogni sensore corrisponde a un nodo della rete: generalmente i dati raccolti vengono elaborati e memorizzati in una base di dati centralizzata ma in alcuni casi i dati vengono proprio memorizzati in essi stessi, realizzando di fatto anche una base di dati distribuita.

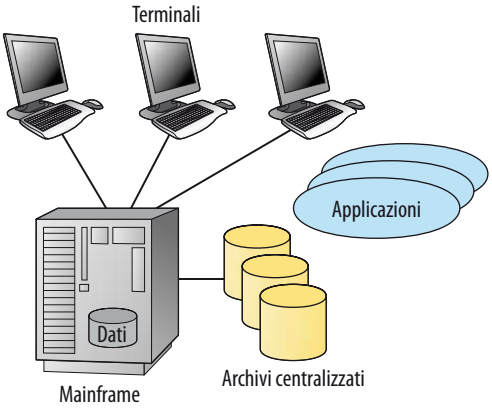


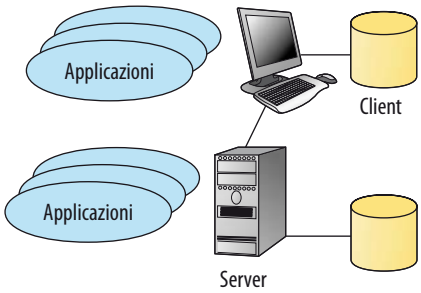

## Architetture distribuite software: dai terminali remoti ai sistemi completamente distribuiti

Come per l'hardware, anche per il software abbiamo avuto una evoluzione nelle architetture distribuite che spesso hanno anticipato i cambiamenti delle architetture hardware e dei loro sistemi operativi. Ricordiamo brevemente le tappe fondamentali.

### Architettura a terminali remoti

La prima tipologia di elaboratori ha questa architettura dove tutte le operazioni vengono effettuate da un'unica entità centrale alla quale sono collegati terminali privi di capacità di elaborazione che si limitano a visualizzare e inviare e ricevere le informazioni alla entità centrale.
In questi sistemi si hanno terminali omogenei e quindi il software di gestione svolge dei compiti relativamente semplici, replicando per ciascuno di essi una area di memoria riservata e facendo evolvere singolarmente i singoli task avviati dalle diverse unità remote.




### Architettura client-server

A differenza dell'architettura precedente i client non sono "macchine stupide" ma hanno una loro capacità di elaborazione: questi richiedono un servizio a un server "di competenza", che è in grado di esaudirne la richiesta, la elabora e invia la risposta al client. Come vedremo in seguito, possono essere presenti più server che offrono più servizi contemporaneamente così come più client possono richiedere lo stesso servizio o servizi differenti sia a server diversi che allo stesso server.

Inoltre un server può essere contemporaneamente anche client: quindi può essere server per uno (o più) servizi e client per altri richiedendo i servizi che necessita ad altri server.
Client e server possono essere tecnologicamente diversi, sia come hardware che come sistema operativo: questa architettura è quella che meglio si presta a far comunicare e cooperare entità non omogenee.

### Architettura WEB-centric

L'evoluzione che negli ultimi anni ha caratterizzato i sistemi distribuiti riguarda il Web: la diffusione capillare delle piattaforme mobili ha, di fatto, provocato uno spostamento delle applicazioni sul server facendo "in qualche modo" regredire gli host quasi "a terminali stupidi del secolo passato", utilizzati solo per l'interfaccia grafica verso l'utente.
Molte applicazioni gestionali sono state "convertite al Web" rendendo il Web server come centro del sistema distribuito (Web-centric) al quale i client vi accedono per ottenere dei servizi: tutta la computazione avviene sui server che restituisce ai client il risultato.
I componenti fondamentali di un'applicazione Web sono analoghi per certi versi a quelli di una tradizionale applicazione client-server: infatti il Web è un sistema distribuito composto da client e server che accedono a documenti collegati. I server gestiscono insiemi di documenti, mentre i client forniscono agli utenti una interfaccia facile da usare per accedere e presentare questi documenti.
Le architetture possono essere classificate in:
- architetture Web tradizionali;
- architetture Web multilivello.

### Architettura cooperativa

L'evoluzione dell'architettura client-server è l'architettura cooperativa che si basa su entità autonome che esportano e richiedono servizi secondo il modello di sviluppo a componenti per la programmazione lato server proprio della programmazione a oggetti.
Di fatto è stato preso uno dei principi chiave della programmazione orientata agli oggetti, l'incapsulamento, applicandolo all'abbattimento delle differenze tra i prodotti usati nell'infrastruttura hardware, software, di programmazione e di rete di un sistema distribuito.
Per rendere efficace l'utilizzo di questi meccanismi è necessario introdurre una standardizzazione sulle modalità con le quali i servizi vengono offerti e come i client possano richiederli: OdP (Open Distributed Processes) e CORBA (Common Object Request Broker Architecture) sono due esempi di standardizzazione.

### Architettura completamente distribuita

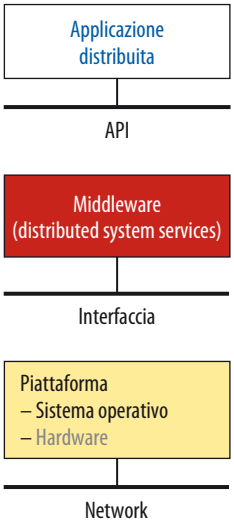
In opposizione alla architettura Web-centric troviamo l'architettura completamente distribuita, tipica di sistemi dove è necessaria la cooperazione di gruppi di entità paritetiche, come nei sistemi groupware, che dialogano tra loro offrendo ciascuno i propri servizi e richiedendo servizi che spesso risultano essere duplicati per garantire l'immunità ai guasti (vengono introdotte ridondanze anche nei dati).
Le tecnologie più importanti sono:
- OMG (Object Management Group): è un consorzio creato nel 1989 da varie aziende tra cui Microsoft, HP, NCR, SUN, allo scopo di creare sistemi di gestione di architetture distribuite;
- RMI (Remote Method Invocation): è una tecnologia che consente a processi Java distribuiti di comunicare attraverso una rete; è specifica appunto del mondo Java;
- DCOM (Distributed Component Object Model): è una tecnologia informatica introdotta nel 1996 da Microsoft ed è basato sull'estensione in rete della precedente tecnologia COM.

## Architettura a livelli

Per alleggerire il carico elaborativo dei serventi sono state introdotte le applicazioni multilivello, nelle quali avviene la separazione delle funzionalità logiche del software in più livelli.
Si introducono gli *strumenti di* middleware, cioè uno strato software "in mezzo" che si colloca sopra al sistema operativo ma sotto i programmi applicativi, e sono l'evoluzione dei sistemi operativi distribuiti.

Con middleware si intende una classe di tecnologie software sviluppate per aiutare gli sviluppatori nella gestione della complessità e della eterogeneità presenti nei sistemi distribuiti (D.E. Bakken). Il middleware ha lo scopo di realizzare la comunicazione e le interazioni tra i diversi componenti software di un sistema distribuito.

Questo software si interpone quindi negli elaboratori locali tra le applicazioni e il sistema operativo locale creando un'architettura a tre livelli, può quindi essere diverso da host a host, e sarà descritta nella prossima Lezione.
Lo scopo principale di middleware è di permettere e garantire l'interoperabilità delle applicazioni sui diversi sistemi operativi.
Inoltre permette la connettività tra servizi che devono interagire e collaborare su piattaforme distribuite sulla base di meccanismi di programmazione e API relativamente semplici.
Di fatto la presenza di questo strato rende facilmente programmabili i sistemi distribuiti offrendo una specifica modalità di interazione, che può essere per esempio un paradigma di interazione basato sulla chiamata di procedure remote (RPC, Remote Procedure Call) oppure un paradigma di programmazione basata sullo scambio di messaggi.
Schematicamente possiamo collocare il middleware come mostrato nella figura a lato.




Tra le funzionalità del middleware ricordiamo:
- i servizi di astrazione e cooperazione;
- i servizi per le applicazioni;
- i servizi di amministrazione del sistema;
- il servizio di comunicazione;
- l'ambiente di sviluppo applicativo.

Un servizio di middleware è un servizio general-purpose che si colloca tra piattaforme e applicazioni (Bernstein).
Nonostante i molti benefici offerti dal middleware, questo però non è la risposta a tutti i problemi legati ai sistemi distribuiti: i problemi connessi alle decisioni "affrettate" o alla "progettazione approssimativa" e/o spesso non completamente competente degli sviluppatori. Per esempio, chi sviluppa applicazioni distribuite spesso non conosce la differenza nella realizzazione di una chiamata di procedura remota e di una chiamata locale!
A volte il middleware è focalizzato solo sulla comunicazione tra componenti e quindi non ha "nessuna responsabilità" sui problemi derivati da errori e malfunzionamenti della applicazione e non è in grado di gestire fallimenti della rete e guasti nei server.

## Conclusioni

Nella seguente tabella ricapitoliamo le caratteristiche essenziali delle diverse architetture distribuite descritte sinora:

| CARATTERISTICHE ESSENZIALI | ARCHITETTURE MIMD | | OS NETWORK | MIDDELWARE |
|---|---|---|---|---|
| | Multiprocessori | Multicomputer | | |
| Gradi di trasparenza | Molto alto | Alto | Basso | Alto |
| Unico SO su tutti i nodi | SÌ | SÌ | NO | NO |
| NR. di copie del SO | 1 | N | N | N |
| Forma di comunicazione | Memoria condivisa | Messaggi/ memoria condivisa | Files | Modelli specifici |
| Gestione delle risorse | Globale centralizzata | Globale distribuita | Per nodo | Per nodo |
| Scalabilità | NO | Modesta | SÌ | Variabile |
| Apertura ai diversi SO | CHIUSO | CHIUSO | APERTO | APERTO |

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 La classificazione di Flynn si basa sui due flussi normalmente presenti nei calcolatori:**
**a** flusso delle istruzioni
**b** flusso di controllo
**c** flusso dei dati

**2 Alla categoria SISD appartengono i seguenti calcolatori:**
**a** personal computer
**b** video terminali
**c** workstation
**d** mainframe

**3 Quali delle seguenti affermazioni sono false riferite all'architettura MISD?**
**a** Utilizza elaboratori con più istruzioni sullo stesso flusso dati
**b** Esistono più processori
**c** Ogni processore ha una sua memoria
**d** Vengono utilizzati principalmente per l'intelligenza artificiale

**4 Nella macchine MIMD viene effettuata una ulteriore classificazione:**
**a** macchine a memoria fisica condivisa
**b** macchine ad accesso parallelo
**c** macchine a memoria privata
**d** macchine a controllo numerico
**e** macchine a canale condiviso
**f** macchine a mezzo diretto

**5 Un cluster di PC differisce da una rete di PC principalmente perché:**
**a** ha una potenza di calcolo pari alla somma di quelle dei singoli computer che lo costituiscono
**b** ha una velocità del trasferimento dati di oltre 1 Gbit/s
**c** ha una centralizzazione fisica delle macchine
**d** esiste una applicazione di management, residente su un singolo PC

**6 Nei sistemi wearable computing possiamo avere:**
**a** sensori di movimento
**b** PDA
**c** connessioni cablate
**d** ECG sensor
**e** architetture MISD

**7 Quali delle seguenti affermazioni sono false riferite alla domotica?**
**a** Neologismo nato da *casa* e *automatica*
**b** Consente di avere risparmio energetico
**c** Ammette accesso remoto
**d** Rientra nelle specifiche dell'industria 4.0

**8 Le architetture WEB-centric possono essere:** (*2 risposte*)
**a** Web tradizionali
**b** Web avanzate
**c** Web remote
**d** Web multilivello

**9 Le tecnologie con architettura completamente distribuita più importanti sono:**
**a** OMG
**b** OdP
**c** CORBA
**d** RMI
**e** DCOM

**10 L'acronimo CORBA significa:**
**a** Comunication Object Request Basic Architecture
**b** Common Object Request Basic Architecture
**c** Common Object Request Broker Architecture
**d** Comunication Object Request Broker Architecture

**11 Tra le funzionalità del middleware ricordiamo:** (*indica quelle non presenti*)
**a** i servizi di astrazione e cooperazione
**b** i servizi per le applicazioni
**c** i meccanismi di sincronizzazione
**d** i servizi di amministrazione del sistema
**e** il servizio di comunicazione
**f** l'ambiente di sviluppo applicativo
**g** la gestione degli errori

**12 Lo scopo principale di middleware è quello di:** (*indica quello inesatto*)
**a** aiutare gli sviluppatori nella gestione della complessità
**b** permettere di garantire l'interoperabilità
**c** ridurre la quantità di comunicazione
**d** permettere la connettività tra servizi che devono interagire su piattaforme distribuite

# VERIFICA... le conoscenze

## VERO/FALSO

1 Il limite inferiore ottenibile con l'ottimizzazione dell'hardware è legato alla velocità delle luce. V F

2 La velocità della luce nel rame è di circa 200.000 km/s. V F

3 Con frequenze di lavoro dell'ordine dei Megahertz non possiamo superare la distanza di 20 cm senza introdurre ritardi. V F

4 Nelle macchine a singola CPU il flusso di istruzioni è unico. V F

5 Un elaboratore SIMD non ha trovato a oggi applicazioni commerciali. V F

6 Un elaboratore MISD è particolarmente adatto per realizzare calcoli vettoriali e matriciali. V F

7 Le macchine MIMD sono anche chiamate multicomputer. V F

8 Le macchine MIMD sono anche chiamate multiprocessor. V F

9 I sistemi multicomputer sono architetture MIMD a memoria condivisa (shared memory). V F

10 Lo scambio di messaggi espliciti viene effettuato mediante apposite procedure (send e receive). V F

11 Nei multiprocessori ogni computer possiede una propria area di memoria privata, non indirizzabile da parte dei processori remoti. V F

12 Le LAN di PC sono da considerarsi sistemi MIMD. V F

13 Con i cluster di PC è possibile affrontare calcoli particolarmente onerosi che sarebbero molto lunghi o impossibili con un solo computer. V F

14 Nelle architetture client-server due client possono collaborare tra loro unicamente attraverso uno o più server che permettono la coordinazione e la condivisione dei dati. V F

15 La casa domotica deve avere una connessione Internet. V F

16 La rete domestica può essere considerata un elemento della casa domotica. V F

17 Tra le aree di interesse della domotica troviamo la climatizzazione e il riscaldamento. V F

18 Tra le aree di interesse della domotica troviamo la gestione degli spazi esterni (irrigazione, piscina ecc.). V F

## COMPLETAMENTO

Completa la tabella che segue.

| CARATTERISTICHE ESSENZIALI | ARCHITETTURE MIMD | | OS NETWORK | MIDDELWARE |
|---|---|---|---|---|
| | Multiprocessori | Multicomputer | | |
| Gradi di trasparenza | | | | |
| Unico SO su tutti i nodi | | | | |
| NR. di copie del SO | | | | |
| Forma di comunicazione | | | | |
| Gestione delle risorse | | | | |
| Scalabilità | | | | |
| Apertura ai diversi SO | | | | |

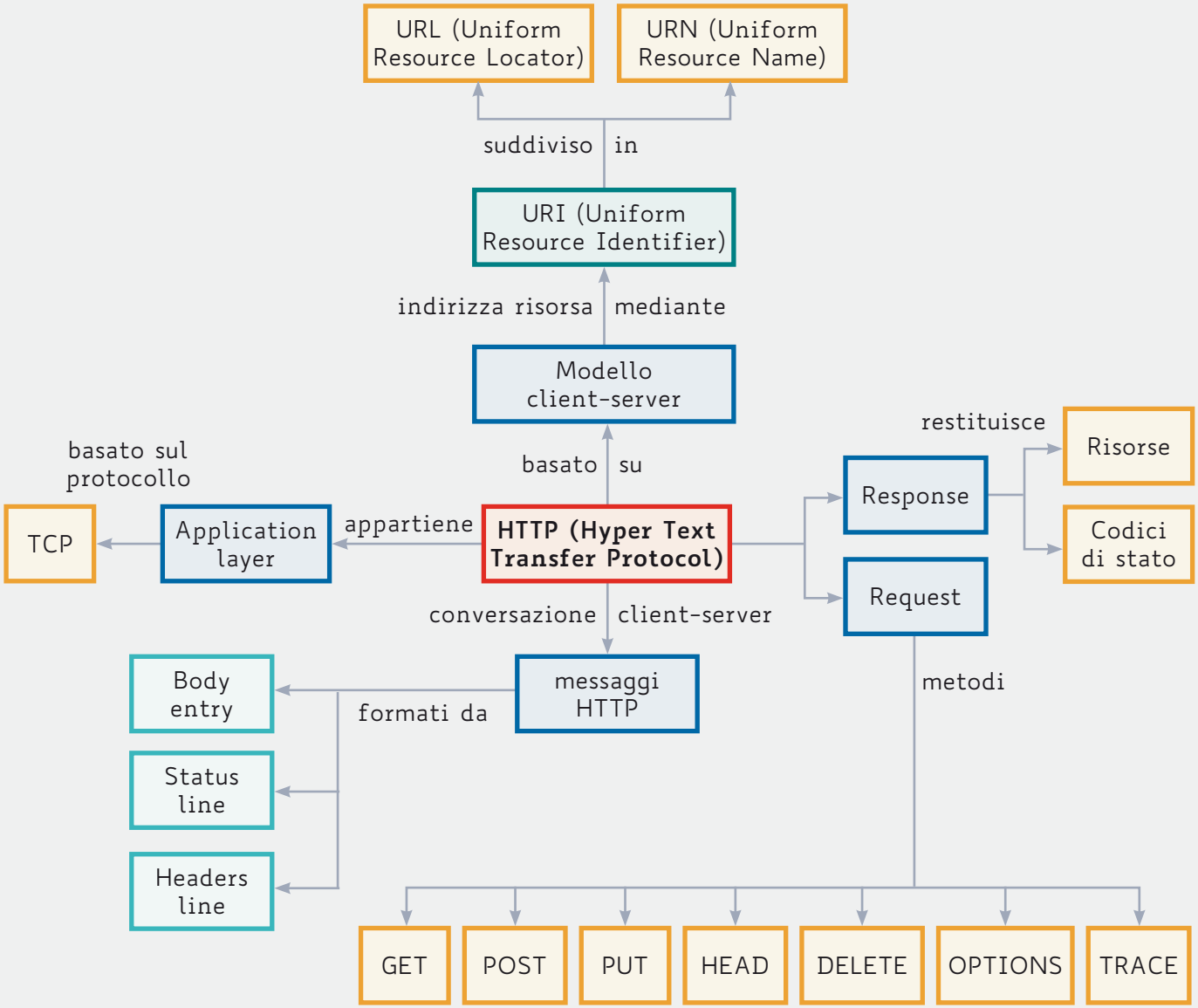
# 3 La comunicazione nel Web con protocollo HTTP

**IN QUESTA LEZIONE IMPAREREMO...**

- le caratteristiche del protocollo HTTP
- la struttura dei messaggi HTTP
- I codici di stato

## MAPPA CONCETTUALE

didattica inclusiva



## HTTP e il modello client-server

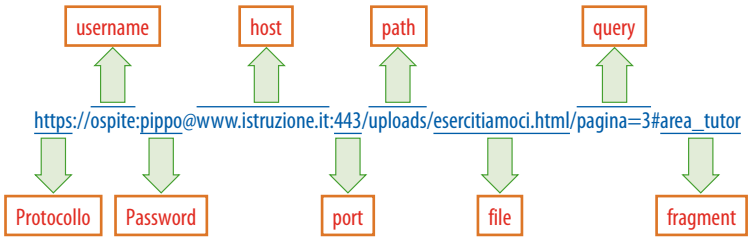
Il Web è basato sul modello client-server che ha due differenti elementi:
- client;
- server.

I client sono elementi ATTIVI (Web browser) che:
- utilizzano il protocollo **HTTP** per connettersi ai server, secondo un modello a client attivo;
- usano un URL per identificare le risorse;
- richiedono pagine Web ai server e ne visualizzano il contenuto.

I server sono elementi PASSIVI (Web server o HTTP server) che:
- rimangono in ascolto di eventuali connessioni di nuovi client su una determinata porta TCP, secondo un modello a server passivo;
- utilizzano il protocollo HTTP per interagire con i client;
- forniscono ai client le pagine Web o le risorse richieste.

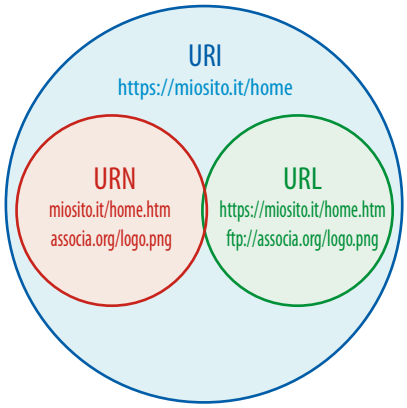
HTTP è un protocollo usato per trasmettere risorse, non solo file. Una risorsa è identificata da un URI o URL. Il tipo più comune di risorsa è un file, ma può anche essere il risultato di una query generato dinamicamente, l'uscita di uno script CGI, un documento disponibile in diversi linguaggi, o altro ancora.
Vediamo le definizioni del W3C (World Wide Web Consortium):
- URI (Uniform Resource Identifier): "set generico di nomi o indirizzi che rappresentano stringhe assegnate alle risorse";
- URL (Uniform Resource Locator): "termine informale, utilizzato solo nelle specifiche tecniche, associato a popolari protocolli quali HTTP, FTP, mailto, ... ecc.".

Vediamo un esempio di indirizzo URI:




**HTTP**
Sono state definite due versioni del protocollo HTTP:
- HTTP/1.0 (anno 1996),
- HTTP/1.1 (anno 1999).

La versione 1.1, rispetto alla versione 1.0, permette di specificare una connessione permanente, oltre a consentire il criptaggio di alcuni dati. Per connessione permanente si intende una richiesta e la risposta all'interno della stessa connessione, grazie al server che lascia aperta la connessione TCP dopo aver spedito la risposta.




**Server e Client**
Client è il programma applicativo che stabilisce una connessione al fine di inviare delle richieste, mentre server è il programma applicativo che accetta connessioni al fine di ricevere richieste e inviare specifiche risposte con le risorse richieste.

**Sessioni**
Le sessioni HTTP sono basate sul protocollo TCP: sia le richieste inviate al server sia le risposte restituite ai client sono trasmesse mediante i seguenti passaggi:
1 il server rimane in ascolto sulla porta 80;
  1.1 il client apre una connessione TCP sulla porta 80;
2 il server accetta la connessione;
  2.1 il client invia una richiesta;
3 il server invia la risposta al client e, solo nel caso dell'HTTP 1.0., chiude la connessione.

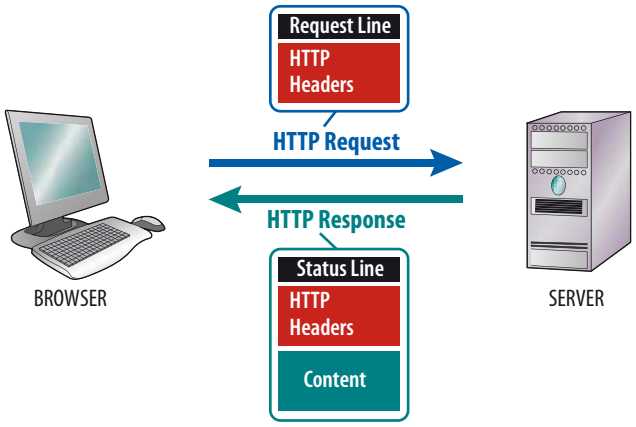
## Il protocollo HTTP

WWW = HTML + URL + HTTP

HTTP è un protocollo di testo che fornisce il livello di trasporto a tutti i protocolli applicativi basati su di esso. Possiamo dire che il Web (WWW) è nato dall'insieme di diverse tecnologie, le principali sono HTML, URL e HTTP.
HTTP (acronimo di Hyper Text Transfer Protocol) è un protocollo che consente di inviare e ricevere le risorse Web, rappresentate da documenti, pagine o elementi di una singola pagina Web, da un host chiamato **server** a un altro chiamato **client**.
Il protocollo HTTP per comunicare utilizza **sessioni**, dove ciascuna sessione inizia stabilendo per prima cosa una connessione TCP al server, utilizzando di default la porta TCP 80, effettuando poi una richiesta (request) contenente un URL. In risposta, il server produce e restituisce il file richiesto e, nella situazione più semplice, chiude la connessione TCP immediatamente dopo.

## Conversazione client-server

Ogni conversazione tra client e server sul Web inizia con una richiesta (request) rappresentata da un messaggio di testo creato dal client in formato HTTP. Il client invia la richiesta al server, quindi attende la risposta (response).




### ESEMPIO

**RICHIESTA DI UNA PAGINA**

Ipotizziamo di volere richiedere una pagina composta da un file HTML e da alcune immagini in formato png all'indirizzo www.istitutovolta.edu/informatica/home.index.

| | |
|---|---|
| **1a** http (client) richiede una connessione TCP con il WEB server all'indirizzo: www.istitutovolta.edu. La porta 80 è la porta di default per i server http. | |
| | **1b** http (server) all'indirizzo www.istitutovolta.edu è in attesa sulla porta 80. Accetta la richiesta di connessione. |
| **2** http (client) invia un messaggio di richiesta (request message) con la URL nella connessione TCP. | |
| | **3** http (server) riceve la richiesta e invia il messaggio di risposta (response message) con l'informazione richiesta (la pagina WEB informatica/home.index), mediante connessione TCP. |
| **5** http (client) riceve la risposta con il file html. Analizzando il file html (parsing) il client rileva alcune immagini (png). | **4** http (server) chiude la connessione TCP. |
| **6** Vengono ripetuti i passi da 1 a 5 per tutti gli elementi png. | |

## Tipi di connessioni

Il protocollo HTTP utilizza TCP come protocollo di trasporto per effettuare le connessioni tra client e server. La connessione HTTP tra client e server viene attivata dal client che attraverso lo scambio di tre segmenti TCP con il server. Dopo tale scambio, il client può mandare la sua richiesta HTTP al server. Una volta soddisfatta la richiesta con un opportuno messaggio di risposta, è il server a chiudere la connessione tramite un ulteriore scambio di segmenti TCP.

Nella connessione HTTP 1.1, di tipo **permanente**, il server lascia aperta la connessione TCP dopo aver spedito la risposta. Le successive richieste e risposte fra client e server utilizzano sempre la stessa connessione. Una pagina Web, anche se formata da file distinti (immagini, file html ecc.) può essere inviata in una singola connessione TCP permanente.
Esistono due tipi di connessione permanente:
- connessione permanente incanalata;
- connessione permanente non incanalata.

Nella versione non incanalata il client passa una nuova richiesta solo quando la risposta alla precedente è stata ricevuta. Nella versione incanalata le richieste vengono via via aggiunte a una coda chiamata pipeline, mentre le risposte vengono processate e inviate nello stesso ordine delle richieste.

**Connessione permanente**
Nella connessione HTTP permanente il server chiude automaticamente la connessione nel caso di non utilizzo per un determinato tempo, chiamato intervallo di timeout.

L'HTTP 1.1 utilizza di default la connessione incanalata: il client può fare richieste consecutive, effettuando una nuova richiesta non appena incontra un nuovo riferimento (metodo back-to-back).

## I messaggi HTTP

Durante una connessione e una comunicazione HTTP client e server si scambiano messaggi di richiesta e di risposta, entrambi formati da:
- una riga iniziale;
- un'intestazione (header), non obbligatoriamente presente;
- una riga vuota;
- un corpo del messaggio (body), non obbligatoriamente presente.

## Messaggio di richiesta: HTTP Request

In generale la prima riga della request contiene la versione del protocollo HTTP utilizzato, mentre nelle righe successive vengono indicate gli header (intestazioni), rappresentate da diversi elementi, ciascuno dei quali composto da un nome, seguito dai due punti (:) e da un valore, dove ogni riga rappresenta un distinto header. Gli header più comuni che possiamo trovare nelle richieste sono:
- la versione del browser che prende il nome di User-Agent;
- l'host presente nell'URL;
- il parametro l (per i tipi di documento che supportano il tipo **MIME**).

**MIME**
MIME (Internet media type) è un dato formato da due componenti che identifica la classe e il formato di ogni tipo di file. Per esempio **text/plain** o **audio/mpeg** sono alcuni dei formati disponibili.

Gli header si concludono con una riga vuota, la quale può essere seguita da un qualunque dato da inviare al server. La dimensione massima del dato inviato è stabilito dall'header Content-Length. Generalmente HTML utilizza questo spazio per l'invio dei campi dei form.

Una HTTP request è un messaggio testuale inviato dal client al server HTTP ed è formato da tre elementi:
- riga di richiesta,
- intestazione HTTP (header),
- corpo del messaggio (message body),

secondo la sintassi seguente:

```
<Method> <URI ><Version>        // riga di richiesta
[Header]                        // intestazione
CRLF                            // riga vuota
[Body]                          // corpo del messaggio
```

CRLF indica l'invio a capo per ottenere una riga vuota e si ottiene con la sequenza di caratteri ASCII:
- CR = Carriage Return,
- LF = Line Feed.

## Riga di richiesta

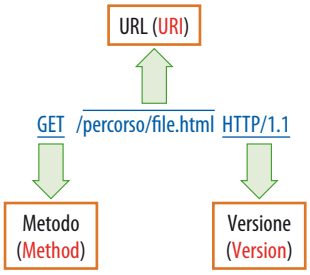
La riga di richiesta contiene:
- il **metodo** (Method) che può essere:
  - GET (versione HTTP 0.9),
  - DELETE, GET, HEAD, LINK, POST, PUT, UNLINK (versione HTTP 1.0),
  - CONNECT, DELETE, GET, HEAD, POST, PUT, OPTIONS, TRACE (versione HTTP 1.1);
- l'URI (o URL) è l'identificativo di risorsa richiesta al server;
- la versione (Version) può assumere i valori HTTP/1.0 o HTTP/1.1.

**Metodo**
Il nome del metodo è case sensitive pertanto deve essere sempre scritto in maiuscolo.

### ESEMPIO

Un esempio di riga di richiesta HTTP è la seguente:

```
GET  /percorso/file.html HTTP/1.1
```




In questo caso possiamo notare che il metodo richiesto è GET, l'URI è rappresentato dal pathname nel formato /percorso/file.html, si tratta di un file in formato html presente nella sottodirectory della radice del server chiamata /percorso.

## Intestazione HTTP (header)

L'intestazione HTTP (header) contiene tutte le informazioni necessarie per l'identificazione del messaggio. È formata da diverse righe, ciascuna delle quali rispetta il formato:

```
nome_header : valoreCRLF
```

e sono classificate in:
- intestazioni generali;
- intestazioni della richiesta;
- intestazioni del corpo dell'entità.

Quando si utilizza il metodo GET non appare nessun corpo dell'entità (entity body).

## Corpo del messaggio (Message body)

Il corpo del messaggio (message body) di richiesta contiene i dati trasportati.

### ESEMPIO

**ESEMPIO DI RICHIESTA COMPLETA**

request line (GET, POST, HEAD commands)

```
GET / HTTP/1.1
```

header lines

```
Accept: image/gif, image/jpeg, image/png, */*
Accept-Language: it
Accept-Encoding: gzip, deflate
Host: www.magistricumacini.it
If-Modified-Since: Fri, 14 Mar 2018 10:54:03 GMT
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
```

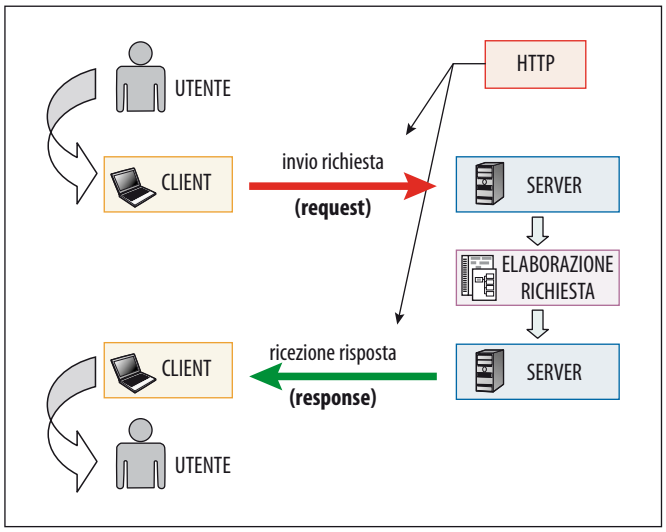
Indicano la fine del messaggio

(carriage return, line feed)

## Messaggio di risposta: HTTP Response

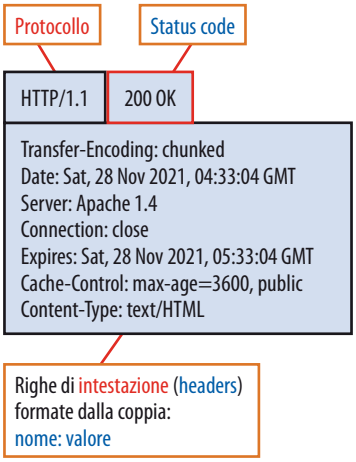
**Server-side**
Le tecnologie lato server (server-side) consentono l'elaborazione di script sul server HTTP, in modo quindi indipendente dall'ambiente di esecuzione dell'utente. Le principali e più comuni tecnologie dal lato server sono: CGI (Common Gateway Interface), PHP, ASP, JSP, Perl.

Dopo che il server HTTP ha ricevuto una richiesta da un client, intraprende le azioni necessarie a fornire la risorsa richiesta. Queste possono includere l'esecuzione di uno script in tecnologia **server-side**. Fatto ciò, il server Web risponde al client con una risposta HTTP.




La response (risposta) HTTP è organizzata in maniera analoga rispetto a una richiesta. L'unica differenza è che le risposte iniziano con una riga di stato al posto della riga di richiesta.
La response è formata da:
- una riga iniziale con versione protocollo HTTP e stato;
- un'intestazione (header ) facoltativa;
- un corpo (body) facoltativo.

Nella seguente figura vediamo un'intestazione tipica di una response HTTP.




## Header HTTP

Gli header della response sono formati, come nel caso della richiesta, da insiemi di coppie, formate da un nome separato dal valore dai due punti (:), che specificano le caratteristiche del messaggio.

**AREA DIGITALE**
**Dettaglio per tipologia di header**

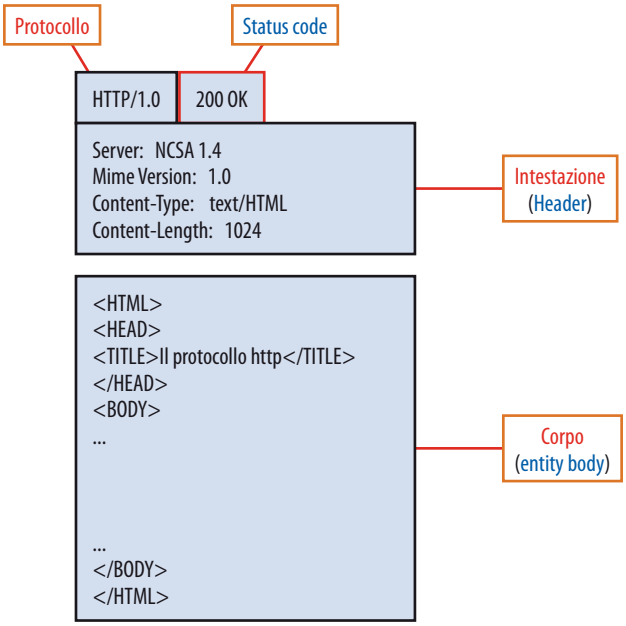
- Header generali della trasmissione, come per esempio:
  - MIME-Version: la versione MIME usata per la trasmissione, normalmente 1.0;
  - Transfer-Encoding: indica la compressione dei dati trasmessi e si riferisce a tutto il messaggio.

- Header relativi all'entità trasmessa, come per esempio:
  - Content-Type: indica il MIME e specifica se si tratta di un testo, un'immagine PNG, un suono WAV, un filmato MPG ecc.;
  - Expires: indica la data di validità della risorsa.
- Header riguardo la richiesta effettuata, come per esempio:
  - User-Agent: indica la versione e il tipo di browser oltre alla versione e al sistema operativo del client;
  - Host: indica il nome di dominio del server e la porta TCP sulla quale il server è in ascolto.
- Header della risposta generata, come per esempio:
  - Server: una stringa che descrive il server: tipo, sistema operativo e versione;
  - WWW-Authenticate: indica il Tipo di autenticazione e le credenziali per autenticare un utente con il server.

Il protocollo HTTP utilizza messaggi in formato ASCII.

### Body della risposta

La sezione body del messaggio di risposta (entity body response) contiene in genere la pagina HTML richiesta dal client. L'entity body è quasi sempre presente nella risposta. La figura che segue mostra un esempio di risposta di un server a una richiesta di un client.




## Metodi (verbi) HTTP

Nella conversazione HTTP, la prima riga di intestazione di una request contiene un elemento chiamato **metodo** o verbo HTTP, che viene sempre digitato tutto in maiuscolo, per esempio:

```
GET / HTTP/1.1
```

In questo caso viene richiesto il metodo GET durante l'invio del messaggio di request. In questo secondo caso viene invece richiesto il metodo DELETE:

```
DELETE /prove/risorsa HTTP/1.1
```

**Metodo**

Il metodo di richiesta è uno dei più importanti attributi di una richiesta HTTP, in quanto rappresenta l'"intenzione" presente nella richiesta del client. Sebbene molti metodi siano usati raramente nella pratica, sono tutti importanti per scopi diversi.

Vediamo una tabella che riassume i principali metodi utilizzabili nelle request HTTP.

| METODO | VERSIONE HTTP | MODIFICHE IN HTTP/1.1 | DESCRIZIONE |
|---|---|---|---|
| GET | 1.0 e 1.1 | Richiesta di parti di entità | Richiede un file al server. La risposta è composta da vari header separati con due CR (Carriage Return) dal path effettivo della risorsa richiesta, in genere rappresentato dal file HTML che contiene la pagina richiesta. Tra i dati della risposta si troverà il protocollo usato (HTTP/1.1) e lo status code che indica l'esito della richiesta. Possiamo anche specificare alcuni parametri, rappresentati da una coppia: `attributo=valore&attributo=valore ... ecc.` |
| HEAD | 1.0 e 1.1 | Uso di header di richiesta condizionali | Richiede solo l'header, senza alcun percorso di file. Utilizzato solo in ambito di fase di testing per pagine Web dinamiche. |
| POST | 1.0 e 1.1 | Gestione di connessione, trasmissione di messaggio | Invia informazioni all'URL specificato. Le informazioni sono presenti in coppie: `attributo=valore&attributo=valore ... ecc.` e vengono elaborate dal server. |
| PUT | 1.1 (1.0 non standard) | | Esegue l'upload di un file sul server, creandolo o riscrivendolo, con il nome indicato e i contenuti specificati nella parte che segue gli header. |
| DELETE | 1.1 (1.0 non standard) | | Cancella una risorsa, generalmente rappresentata da un file, sul server. L'utente con cui è in esecuzione il Web server deve poter avere permessi in scrittura sul file indicato e il server deve essere configurato opportunamente (permesso di cancellazione). |
| OPTIONS | 1.1 | Estensibilità | Richiede l'elenco dei metodi concessi dal server, per esempio: `OPTIONS * HTTP/1.0` |
| TRACE | 1.1 | Estensibilità | Traccia una richiesta, visualizzando come viene trattata dal server, per esempio `TRACE * HTTP/1.0` |

I più importanti metodi HTTP necessari per la conversazione attraverso applicazioni Web server-side (**API**) di tipo RESTful sono GET, POST, PUT e DELETE. Le API per HTTP consentono di eseguire operazioni sui dati presenti sui server.

Le applicazioni RESTful (REpresentational State Transfer) consentono di effettuare operazioni da remoto sui dati presenti nei server, che lo permettono, sfruttando i metodi del protocollo HTTP. Le operazioni del tipo CRUD (Create, Retrieve, Update, Delete), cioè **crea**, **recupera**, **aggiorna** e **cancella**, possono essere così descritte:

- chiedere dati al server (GET);
- creare dati sul server (POST);
- modificare o sostituire i dati sul server (PUT);
- cancellare un oggetto sul server (DELETE).

**API**

Le API (Application Programming Interface) sono delle librerie di procedure, utilizzabili dai programmatori. Ne esistono moltissime, realizzate per diversi linguaggi, che vanno da Java, C++, Python, JavaScript, PHP ecc. Sono sostanzialmente dei frammenti di codice sorgente che possono essere richiamati nei programmi per svolgere scopi diversi.

Lo studio dettagliato dei metodi CRUD del protocollo HTTP esula dagli obiettivi di questo corso: ci focalizziamo invece sulla codifica URL, che verrà utilizzata in ogni applicazione Web.

**AREA DIGITALE**

**Le operazioni CRUD di HTTP**

## La codifica URL

Nella codifica URL, chiamata anche URL encode, i dati (campi modulo) di un form HTML vengono codificati per essere impacchettati in una trasmissione GET o POST.

La URL encode è una procedura che prepara opportunamente una stringa di dati, codificandola, in modo che venga riconosciuta durante la trasmissione HTTP con metodo GET o POST. In altre parole, questo tipo di codifica viene utilizzata nelle QueryString per inserire, all'interno delle URL, delle stringhe di testo in modo sicuro e conforme allo standard previsto per le URL.

La codifica URL viene applicata anche nel caso in cui la richiesta venga generata, per esempio, da un link HTML (tag <a href>) e non solo da campi di un modulo (tag <form>).
Vediamo i passaggi che devono essere effettuati per codificare la stringa da inviare. La stringa da inviare è sempre formata da una serie di coppie nome campo e valore: possiamo paragonare il nome campo a una specie di variabile e il valore al dato in essa contenuto; le operazioni da fare sono le seguenti:

1. convertire tutti i caratteri non sicuri presenti nella stringa in simboli formati dalla percentuale e dal codice ASCII relativo: %xx, dove xx è il valore ASCII del carattere, in esadecimale. I caratteri non sicuri includono =, &, %, @, spazio, +, e altri caratteri non stampabili;
2. eliminare gli spazi vuoti sostituendoli con il carattere %20;
3. unire i nomi e valori con uguale (=) e separare i nomi con ampersand (&), per esempio:

```
nome1=valore1&nome2=valore2&nome3=valore3 ... ecc.
```

Vediamo una tabella che riepiloga i principali caratteri di URL encode:

| CARATTERE | CODIFICA URL ENCODE | CARATTERE | CODIFICA URL ENCODE | CARATTERE | CODIFICA URL ENCODE |
|---|---|---|---|---|---|
| | %20 | # | %23 | % | %25 |
| " | %22 | $ | %24 | & | %26 |
| * | %2A | / | %2F | | |
| 0 | %30 | 1 | %31 | 2 | %32 |
| 3 | %33 | 4 | %34 | 5 | %35 |
| 6 | %36 | 7 | %37 | 8 | %38 |
| 9 | %39 | < | %3C | > | %3E |
| @ | %40 | [ | %5B | ] | %5D |

Un form contiene un campo chiamato utente che contiene il valore Luca e un campo chiamato societa che contiene Rossi & Martini, vediamo come viene codificata la stringa secondo URL encode:

```
utente=Luca&societa=Rossi%26Martini
```

## Le rappresentazioni HTTP

Come abbiamo visto client e server HTTP si scambiano informazioni riguardanti risorse identificate da un indirizzo (URL o URI): sia la richiesta sia la risposta contengono una rappresentazione della risorsa.

La rappresentazione della risorsa HTTP è un'informazione che riguarda non solo la risorsa intesa come indirizzo di localizzazione, bensì il formato e lo stato della risorsa attuale e nella sua evoluzione futura. I due elementi cardine della rappresentazione sono l'intestazione (header) e il corpo (body) del messaggio.

L'header HTTP contiene informazioni chiamate metadati, definite strettamente dalle specifiche HTTP: contengono testo e devono essere formattate, come abbiamo visto, in maniera specifica. Il body HTTP contiene informazioni che rappresentano dati in qualunque formato (testo semplice, immagini, HTML, XML, JSON ecc.). La risposta HTTP deve specificare il tipo di contenuto del body, attraverso l'intestazione, nel campo Content-Type, come per esempio in questo caso in cui viene richiesta una risorsa JSON:

```
Content/Type: application/json
```

## I codici di stato

I codici di stato, o status code, sono dei codici restituiti dai server HTTP per indicare al client l'esito di una richiesta. Sono stati definiti dall'**IETF**.

L'RFC 2616 elenca circa 50 codici di stato che possono essere restituiti dai server HTTP, nella response. La maggior parte degli stati di questo elenco non corrisponde con esattezza al comportamento delle moderne applicazioni Web, alcuni codici di risposta sono stati infatti pensati per usi futuri.

**IETF**
IETF (Internet Engineering Task Force) è un organismo internazionale, formato da specialisti informatici, che si riunisce e delibera standard per lo sviluppo di Internet, in collaborazione con il W3C (World Wide Web Consortium) e con l'ISO/IEC, in particolare per la suite di protocolli Internet.

Lo status code è un numero formato da tre cifre, di cui la prima indica la classe della risposta e le altre due la risposta specifica. Esistono cinque classi principali di codici di stato, suddivisi in base alla tipologia di risposta, vediamoli riassunti nel seguente elenco dove sono stati suddivisi per categoria:

1. codici da 100-199 (Information): questi codici di risposta forniscono informazioni temporanee alla richiesta, durante il suo svolgimento e a partire dall'HTTP versione 1.0 viene sconsigliato il loro utilizzo;
2. codici da 200-299 (Successful): questo intervallo di codici di stato indica il completamento di una richiesta con successo;
3. codici da 300-399 (Redirection): questi codici vengono utilizzati per comunicare una varietà di stati che non indicano un errore, ma che richiedono un trattamento particolare da parte del browser;
4. codici da 400-499 (Client error): questo intervallo di valori viene utilizzato per indicare le condizioni di errore provocate dal comportamento del client;
5. codici da 500-599 (Server error): è una classe di messaggi di errore, prodotti come risposta a problemi verificatisi sul server.

**AREA DIGITALE**
La sicurezza con HTTPS

### ESEMPIO

**CODICI DI STATO SPECIFICI**

A titolo di esempio riportiamo i principali o frequenti messaggi dell'HTTP.

```
100 Continue
```

Il client può chiedere al server di accettare una richiesta prima di inviare l'intero corpo del messaggio. L'intestazione Expect: 100-Continue chiede al server di segnalare la sua accettazione: quando il client riceve lo status code 100 dal server, continua la trasmissione inviando il resto della richiesta.

```
200 OK
```

È la risposta di default a una richiesta GET o POST che indica un esito positivo. Il browser visualizza i dati ricevuti immediatamente dopo o li processa.

```
201 Created
```

Questo codice indica che la richiesta era valida e che una risorsa è stata creata. Viene utilizzato per confermare il successo di una richiesta PUT o POST.

```
204 No content
```

Questo codice identifica un esito positivo, quando non sono attesi dati di risposta, e blocca la navigazione sull'URL che l'ha generato, mantenendo l'utente sulla pagina di origine.

```
206 Partial Content
```

Il codice viene generato dal server in risposta a richieste di blocchi da parte del client. Il client deve già essere in possesso degli altri "blocchi" (chunk) del documento e esamina l'header di risposta Content-Range per ricomporlo prima di effettuarne l'elaborazione.

```
301 Moved Permanently
302 Found
303 See Other
```

Questi codici comunicano al client di ripetere la richiesta a un nuovo server, specificato nell'header Location presente nella response: il client sostituisce POST con GET, eliminano il **payload** e inoltrando nuovamente la richiesta.

**Payload**
Se con carico utile intendiamo la parte di dati trasmessi che è il messaggio, con payload si intendono le informazioni aggiuntive necessarie per effettuare la trasmissione, cioè il "tributo aggiuntivo da pagare".

```
400 Bad request
```

Attraverso i codici 400 il server informa che non è in grado di processare la richiesta per qualche motivo non specificato: il payload della risposta in genere illustra il problema e generalmente viene gestito dal client come una risposta 200.

```
401 Unauthorized
```

Questa risposta indica che l'utente deve fornire delle credenziali di autenticazione a livello di protocollo HTTP per accedere alla risorsa. In genere a questo punto il browser chiede all'utente le informazioni per fare il login e visualizza il corpo della risposta solo se il processo di autenticazione non va a buon fine.

```
403 Forbidden
```

L'URL richiesto esiste ma non può essere scaricato per ragioni diverse dall'autenticazione HTTP, per esempio, permessi di accesso al filesystem non sufficienti, una regola di configurazione sul server che vieta l'accesso alla risorsa, credenziali insufficienti come, per esempio,quelle rappresentate da cookie non validi o da indirizzi IP non validi. Questa risposta viene generalmente mostrata all'utente.

```
404 non trovato
```

Il codice 404 è forse il più noto status code di errore, in quanto indica che l'indirizzo (URL) richiesto non esiste. Questa risposta viene generalmente mostrata all'utente.

```
405 metodo non permesso
```

Il metodo utilizzato non è supportato per la risorsa in questione.

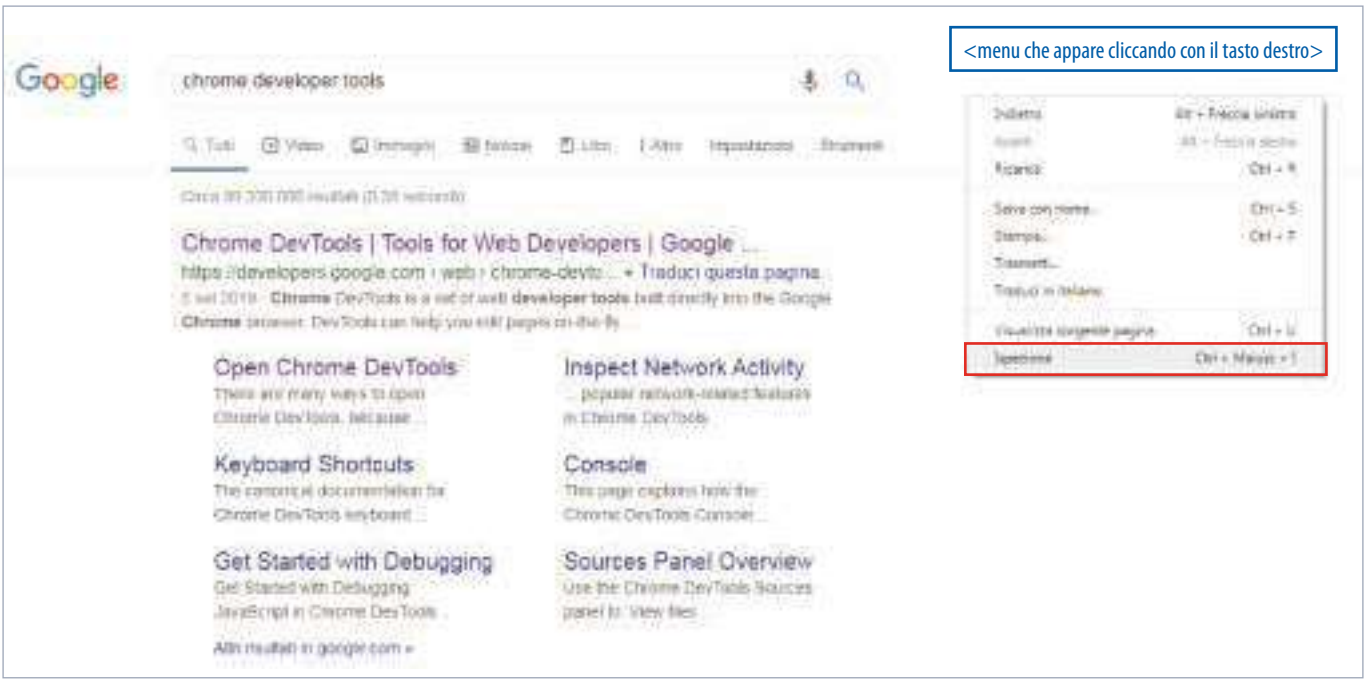
```
500 errore interno al server
```

La risposta 500 è usata quando il processo fallisce dal lato server, in circostanze impreviste, che causano l'insorgere dell'errore.

## Come vedere il funzionamento di HTTP

Se usate Chrome e i suoi strumenti, è disponibile Chrome Developer Tools, una suite di tool inclusi nel browser Google Chrome le cui funzionalità sono del tutto analoghe a quelle offerte da Firebug, che nella sua versione completa permette:

- la navigazione all'interno del DOM delle pagine Web;
- l'analisi e la modifica dell'HTML e dei fogli di stile in tempo reale;
- il monitoraggio,il debug e le modifiche del codice JavaScript in tempo reale.



## cURL

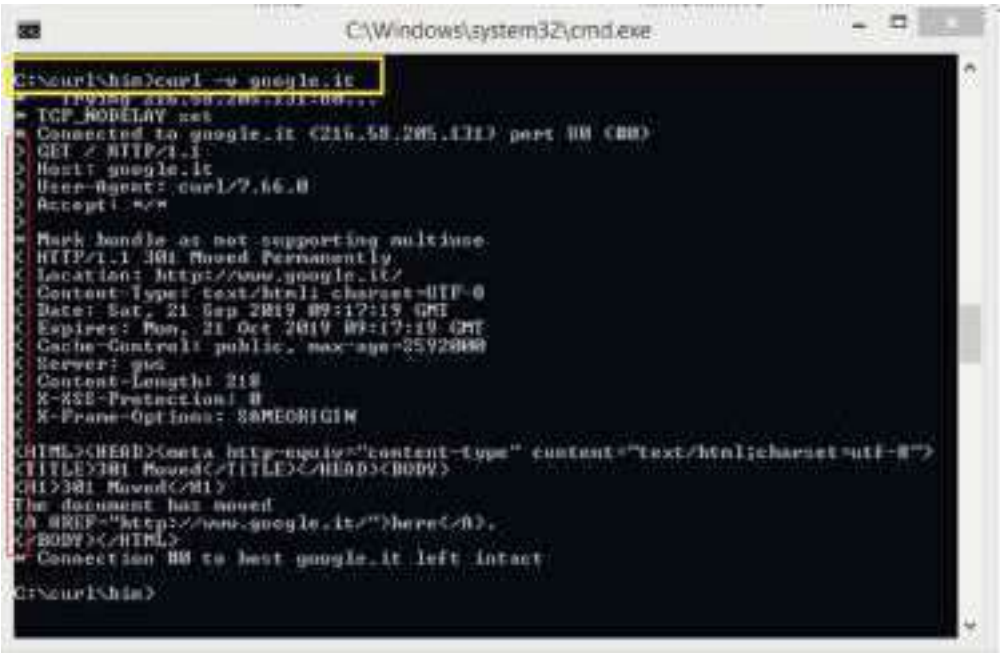
Un altro utile metodo per prendere confidenza con l'HTTP è di usare un client dedicato, come per esempio cURL, scaricabile dall'indirizzo https://curl.haxx.se/.

cURL è un software gratuito e open source definito dai sui stessi autori come "command line tool and library for transferring data with URLs".
cURL viene utilizzato nelle righe di comando o negli script per trasferire i dati in tutti i tipi di dispositivi, dal PC al tablet, dai televisori alle stampanti, dai router alle apparecchiature Audio EDD.
cURL viene utilizzato per effettuare i test dei siti Web, sia semplici come quelli composti da poche pagine che noi realizzeremo, sia estremamente complessi come Facebook o YouTube.
cURL è un software da riga di comando per Shell/DOS che, oltre a HTTP, interagisce con svariati protocolli, come FTP e TELNET.

Con cURL è possibile effettuare molte operazioni:
- recuperare una pagina Web e il suo contenuto;
- inviare in modo fittizio i dati di un form;
- autenticarci in modo fittizio in un'area riservata;
- connetterci in modo generico a un Web service.

## ESEMPIO

Dopo aver installato cURL, digitiamo il seguente comando:




Ciò vi mostrerà l'intera conversazione HTTP. Le richieste sono precedute da >, mentre le risposte sono precedute da <.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Quale tra le seguenti richieste del metodo GET non è permessa:**
**a** richiesta assoluta **c** richiesta parziale
**b** richiesta di test **d** richiesta condizionale

**2 La codifica URL del carattere "e commerciale" (ampersand o "&"):**
**a** non può essere codificata in URL encode
**b** è %20 **c** è %AC **d** è %26

**3 Indica quale campo non è contenuto della seguente query string:**
**pippo.php?uno=b%2A&esse=ciao%3E&A=a%26b**
**a** E **b** uno **c** esse **d** A

**4 Quale tra i seguenti è un argomento valido del metodo PUT?**
**a** Risorsa da ottenere nella response del server
**b** Risorsa da ottenere facendo un GET in seguito
**c** Risorsa da ottenere facendo un POST in seguito
**d** Un header di test della connessione

**5 Qual è il limite della dimensione dei campi di una richiesta POST, in HTTP/1.1?**
**a** 1024 byte **c** 1024 bit
**b** Nessun limite **d** 256 kB

**6 I campi GET vengono inviati al server:**
**a** memorizzati nel body message
**b** memorizzati nell'intestazione
**c** non possono essere inviati campi
**d** dalla successiva richiesta GET

**7 I campi POST vengono inviati al server:**
**a** memorizzati nel body message
**b** memorizzati nell'intestazione
**c** non possono essere inviati campi
**d** dalla successiva richiesta GET

## VERO/FALSO

1 I client sono elementi passivi. V F
2 Un URL identifica sempre una risorsa. V F
3 Il protocollo TCP utilizza HTTP come protocollo di trasporto. V F
4 Le righe di header HTTP sono formate da un nome, seguito dai due punti (:) e da un valore. V F
5 La versione del browser prende il nome di User-Agent. V F
6 Nei messaggi HTTP gli header si concludono con una riga vuota che è facoltativa. V F
7 Una riga di richiesta HTTP può contenere anche un pathname di una risorsa. V F
8 La prima riga di una request HTTP contiene solo la versione http. V F
9 Il metodo TRACE può essere usato per API di applicazioni RESTful. V F
10 La codifica URL si applica solo ai campi form. V F
11 Il metodo PUT non può essere utilizzato per effettuare un upload di un file. V F
12 Nel metodo POST non esistono limiti di lunghezza nei parametri della richiesta. V F

## COMPLETAMENTO

*GET – versione – header – RESTful – PUT – client – body – server – POST – MIME*

1 Nella prima riga di una HTTP request vi è la _____ del protocollo HTTP utilizzato, mentre nelle righe successive vengono indicate le _____.
2 Il tipo _____ è un dato formato da due componenti che identifica la classe e il formato di ogni tipo di file. Per esempio text/plain o audio/mpeg sono alcuni dei formati disponibili.
3 Nella sezione _____ della risposta del server viene indicato il codice HTML della pagina richiesta.
4 Le applicazioni _____ consentono di effettuare operazioni di tipo CRUD sui dati presenti nei server sfruttando i metodi _____, POST, _____, DELETE del protocollo HTTP.
5 Attraverso il metodo GET inizia lo scambio di due messaggi: il _____ inizia mandando un messaggio al _____ che identifica l'oggetto richiesto tramite il campo URI.
6 Il metodo _____ consente di trasmettere informazioni dal client al server (form).

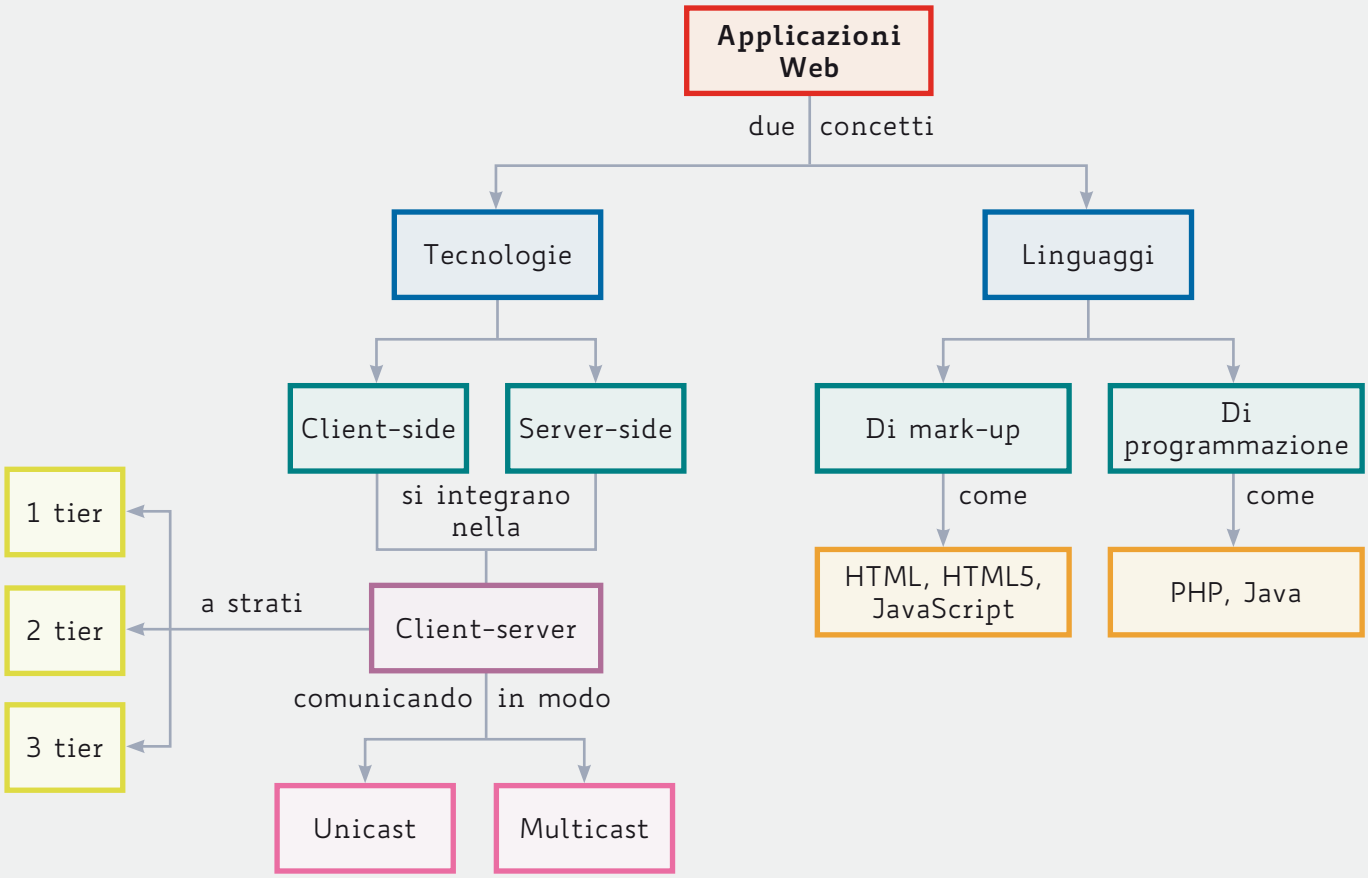
# 4 Le applicazioni Web e il modello client-server

**IN QUESTA LEZIONE IMPAREREMO...**

- gli aspetti essenziale delle tecnologie per il Web
- le caratteristiche del modello client-server
- l'evoluzione del modello client-server

## MAPPA CONCETTUALE

didattica inclusiva



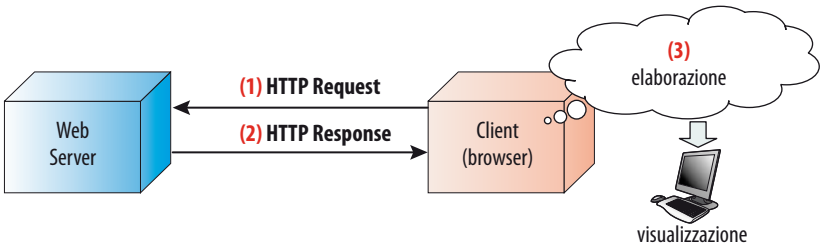

## Applicazioni Web: generalità

Con il termine di applicazione Web viene definito il software sviluppato e utilizzato attraverso tecnologie Web e linguaggi specifici.
Sintetizziamo due concetti fondamentali che stanno alla base delle applicazioni Web:
**1** tecnologie client-side e server-side;
**2** linguaggi di mark-up e linguaggi di programmazione.

## Tecnologie del Web

Possiamo distinguere le tecnologie del Web in due gruppi, in base al posto in cui avvengono le elaborazioni.

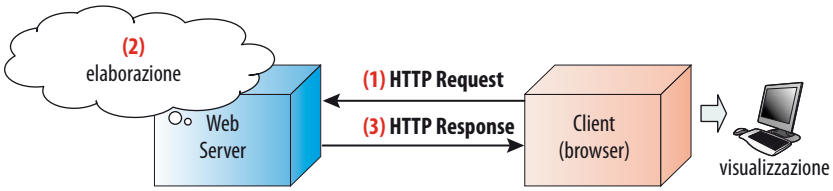
1 Tecnologie client-side: sono le strutture tecnologiche in cui l'elaborazione avviene sul client, tipicamente nel browser; possiamo schematizzare le tre fasi dell'elaborazione in:




Per visualizzare una pagina che utilizza una tecnologia client-side (che già conosciamo essere HTML, JavaScript ...) potremmo anche non utilizzare un Web server, memorizzando e, quindi, aprendo la pagina WEB e fornendo al browser il path sul file system locale:

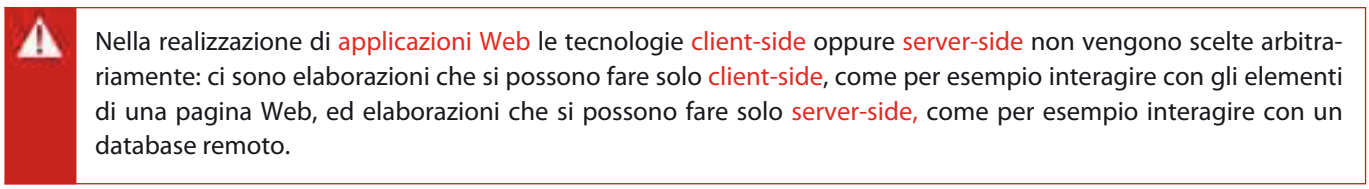
Inoltre, se chiediamo al browser di visualizzare il codice sorgente della pagina, le parti di codice che utilizzano tecnologie client-side **sono visibili**.

2 Tecnologie server-side: sono le strutture tecnologiche in cui l'elaborazione avviene sul server, tipicamente nel Web server; possiamo schematizzare le tre fasi dell'elaborazione in:




Per visualizzare una pagina che utilizza una tecnologia server-side (scritta in linguaggi dinamici come PHP, Java Servlet, ... che descriveremo in questo volume) abbiamo bisogno di un Web server che elabori il codice della pagina ed è, quindi, necessario connettersi e richiedere la pagina tramite un URL:

Se, invece, chiediamo al browser di visualizzare le parti di codice che utilizzano tecnologie server-side queste **NON sono visibili** perché il server ha elaborato il codice (che è memorizzato sullo stesso server) e quello che vediamo sul nostro browser è il risultato di tale elaborazione.



Nella realizzazione di applicazioni Web le tecnologie client-side oppure server-side non vengono scelte arbitrariamente: ci sono elaborazioni che si possono fare solo client-side, come per esempio interagire con gli elementi di una pagina Web, ed elaborazioni che si possono fare solo server-side, come per esempio interagire con un database remoto.

## Linguaggio del Web

Nel Web vengono utilizzate due tipologie di linguaggi: i linguaggi di mark-up e i linguaggi di programmazione.

1 I linguaggi di mark-up servono a scrivere documenti strutturati ("formattati"), dove un documento "formattato" è un contenuto generalmente testuale corredato di indicazioni (tag) che ne definiscono la struttura (e spesso la visualizzazione), come il linguaggio HTML, XML.

**2** I linguaggi di programmazione servono a scrivere programmi: un programma è una sequenza di istruzioni, come per esempio Java e PHP.

Con il passare degli anni l'evoluzione dei linguaggi ha fatto sì che attualmente nel Web queste distinzioni non siano più così "rigide":

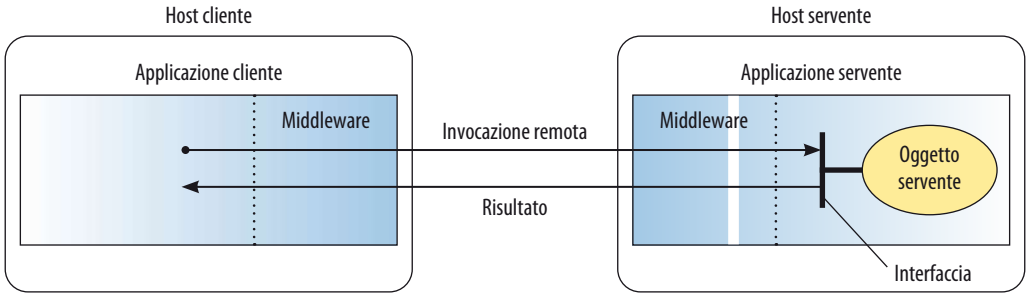
**a** esistono approcci "ibridi" come AJAX, ma non sono una vera "via di mezzo" tra client-side e server-side in quanto "fondono in un unico linguaggio" le tecnologie client-side e server-side effettuando un uso congiunto;

**b** in HTML5 è possibile utilizzare tag (mark-up) per scrivere istruzioni (programmi), rendendo più "difficile" la distinzione tra mark-up e programmazione.

È quindi necessario avere chiaro la distinzione tra le due tipologie di linguaggio e capire il loro funzionamento per poterle correttamente utilizzare per la realizzazione di applicazioni Web.

## Il modello client-server

Il modello client-server è costituito da un insieme di host che gestiscono una (o più) risorse, i serventi o server, e da un insieme di clienti (o client) che richiedono l'accesso ad alcune risorse distribuite gestite dai server. Inoltre ogni processo server può a sua volta diventare client per richiedere accesso ad altre risorse gestite da altri (processi) server.



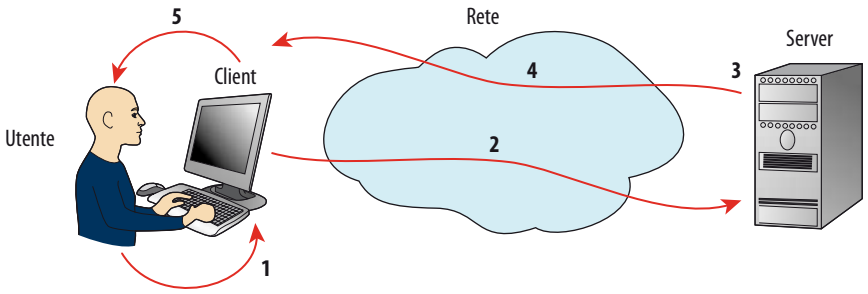

Per essere precisi, non sono gli host a essere server o client ma i processi che sono in esecuzione su di essi, dove come processo si intende un programma in esecuzione: dato che su un host possono essere in esecuzione più processi, un host può essere contemporaneamente sia client che server.

Non bisogna confondere servizio e server: un servizio è un'entità astratta che viene fornito da uno o più server che lavorano su macchine spesso differenti e che cooperano via rete.

Descriviamo il modello client server anche analizzando la sua evoluzione che oggi gli permette di gestire applicazioni molto complesse (*applicazioni* di tipo enterprise) che hanno le seguenti caratteristiche:

- molti utenti concorrenti che richiedono i servizi;
- una logica applicativa complessa;
- archivi di grandi dimensioni con organizzazione di dati complessa e distribuita;
- notevoli requisiti di sicurezza;
- sistemi transazionali.

Lo schema di funzionamento di un modello client-server è sintetizzato nella seguente figura:



**1** il client *manda* una richiesta al server;
**2** il server (in attesa) *riceve* la richiesta;
**3** il server *esegue* il servizio richiesto (generando un thread concorrente);
**4** il server *manda una risposta* ed eventualmente dei dati;
**5** il client *riceve* la risposta ed eventualmente i dati.

Tutte queste attività avvengono in modo trasparente a entrambi i processi.

Questo meccanismo è utilizzato sia che i processi fisicamente si trovino in esecuzione su due calcolatori diversi connessi in rete (come nell'esempio descritto) sia che risiedano sul medesimo calcolatore.
Di seguito sono riportati alcuni servizi tipici delle architetture client-server:

- Telnet: mediante un tale programma (programma client) è possibile operare su un computer remoto come si opera su un computer locale; questo è possibile se sulla macchina remota è presente un programma server in grado di esaudire le richieste del client Telnet;
- HTTP: il browser è un client HTTP (Web), che richiede pagine Web ai computer su cui è installato un Web server, il quale esaudirà le richieste spedendo la pagina desiderata;
- FTP: tramite un client FTP è possibile copiare e cancellare file su un computer remoto, purché qui sia presente un server FTP;
- altri servizi di questo tipo sono SMTP, IMAP4, NFS, NIS e così via.

La programmazione di rete assume sempre più importanza in quanto la maggior parte delle applicazioni ha bisogno di reperire o scambiare dati presenti (o recuperabili) su altri PC e quindi la necessità prima è quella della connessione reciproca.

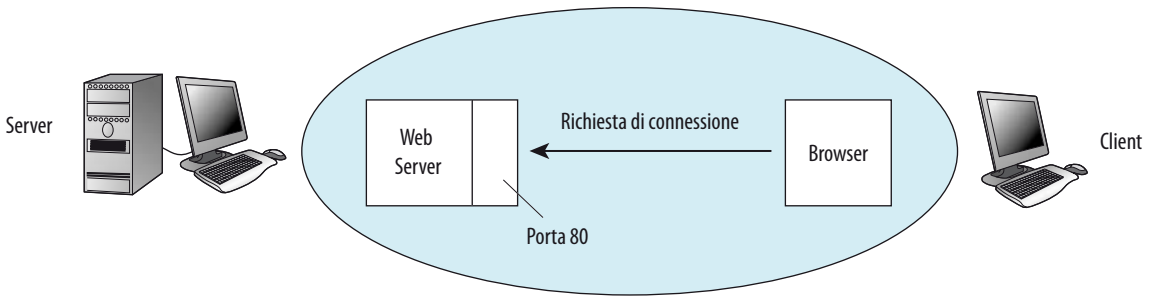
## Distinzione tra server e client

Un programma chiamato client richiede dei servizi a un altro programma chiamato server. Quest'ultimo è ospitato su un computer chiamato host ed è in ascolto tramite un **socket** su una determinata porta, in attesa che un client richieda la connessione: il client invia la richiesta al server tentando la connessione proprio tramite tale porta, quella cioè su cui il server è in ascolto.

**Socket**
Un socket è formato dalla coppia <indirizzo IP: numero della porta> che permette di individuare univocamente il gestore di un servizio: verrà descritto nella prossima lezione.

Un client, quindi, per comunicare con un server usando il protocollo TCP/IP deve, per prima cosa, "connettersi" al socket dell'host dove il server è in esecuzione specificando l'indirizzo IP della macchina e il numero di porta sulla quale il server è in ascolto.

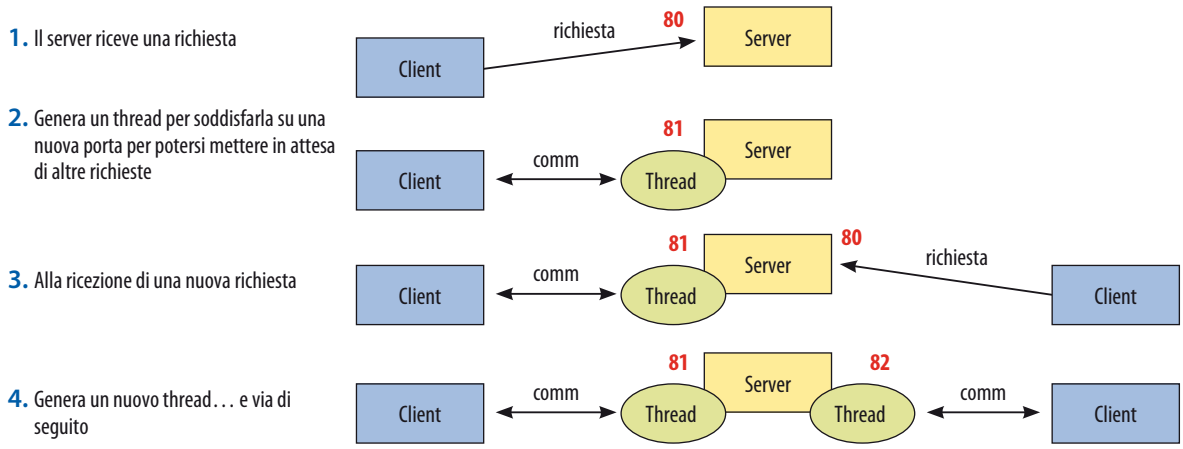
Naturalmente su uno stesso computer possono essere in esecuzione server diversi, in ascolto su porte diverse (per semplificare con un'analogia, si può pensare al fatto che più persone abitano allo stesso indirizzo, ma a numeri di interno diversi: i numeri di interno rappresentano le porte).
Un server "rimane in ascolto" su una determinata porta finché un client non crea una comunicazione con il socket specificando la porta sulla quale è disponibile il servizio; quindi esegue le richieste del client con le risorse che ha a disposizione e rispedisce, se richiesto, i risultati al client.
La figura seguente riporta un esempio di servizio HTTP, dove il Web server rimane in attesa della connessione dei browser dei client sulla porta 80.



## Comunicazione unicast e multicast

Ora che è stato introdotto il concetto client-server, possiamo passare a distinguere due tipi di comunicazione:
- unicast: il server comunica con un solo client alla volta accettando una richiesta di connessione solo se nessun altro client è già connesso;
- multicast: al server possono essere connessi più client contemporaneamente.

Nel caso di trasmissione multicast se la richiesta di connessione tra client e server va a buon fine il server, prima di stabilire il canale di connessione con il client, sposta la richiesta dalla porta nella quale è stata effettuata, port address, su una nuova porta, così lascia libera la prima in attesa di altre connessioni, e manda in esecuzione un thread che soddisfa la richiesta.




### ESEMPIO

Un classico esempio di applicazione client-server che implementa trasmissioni multicast sono browser e Web server: infatti il Web server deve garantire gli stessi sevizi a più utenti e deve sempre restare in ascolto sulla porta 80, in attesa di nuove richieste di connessione da parte di nuovi utenti.

## Livelli e strati

Le architetture client-server sono normalmente organizzate a "livelli" (tier) dove ogni livello corrisponde a un nodo o gruppo di nodi di calcolo su cui è distribuito il sistema: ciascun livello funziona da server per i suoi client nel livello precedente e da client per il livello successivo ed è organizzato in base al tipo di servizio che fornisce. Spesso il modello client-server *a* livelli è combinato con quello *a* strati dove ogni strato viene definito dal punto di vista funzionale come un "livello di astrazione".
Ricapitolando, un sistema adotta un'architettura a livelli e il software in ciascun livello è spesso organizzato internamente a strati (punto di vista di deployment).
In generale nelle applicazioni possiamo individuare tre tipi principali di funzionalità che corrispondono a una struttura in tre strati o livelli (modello three-tier):
- front-end o presentation tier: è l'interfaccia verso l'utente;
- logica applicativa o middle tier;
- back-end con l'accesso alle risorse/ai dati, anche detto data tier.

La nomenclatura indicata è quella tipica delle applicazioni Web, ma la suddivisione in tre strati può essere effettuata per ogni tipo di applicazione informatica, con la terminologia elencata di seguito.
- Presentation Layer (PL): è composta dall'insieme delle procedure o moduli dedicate all'acquisizione e alla presentazione dei dati all'utente (maschere di input, organizzazione di tabelle e tabulati video/cartacei).

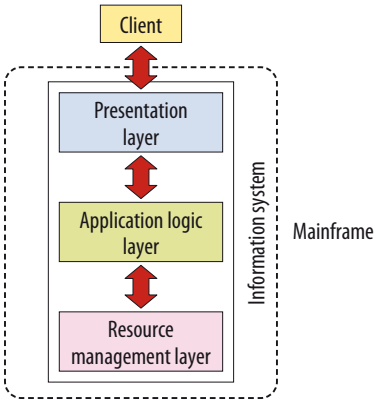
Per esempio, nei sistemi Web che visualizzano pagine HTML, il Presentation Layer è costituito dai moduli del web server che concorrono a creare i documenti HTML, come le Java Servlet, gli script PHP e ASP, mentre il client può essere identificato con il browser.

- Resource Management Layer: è composto dall'insieme delle procedure che gestiscono i dati, cioè memorizzano e recuperano le informazioni persistenti dagli archivi di massa delle basi di dati.
  Nel caso in cui esso è implementato tramite un DBMS, è detto semplicemente Data Access Layer (DAL).
- Business Logic Layer (BLL) o Resource Management Layer (RML): è il "corpo centrale" della applicazione che comprende la logica della elaborazione e le definizione delle relazioni esistenti tra le diverse entità.
  Per esempio, comprende l'algoritmo che implementa le operazioni legate a un prelievo su un conto corrente bancario, o la sequenza di passi da compiere per effettuare un acquisto online.

Nelle moderne architetture client-server è possibile individuare una corrispondenza tra livelli e strati e nella nostra trattazione, dopo aver ripercorso storicamente la sua evoluzione, la prenderemo come modello di riferimento.

## Architettura a un livello – 1 tier

Storicamente, a partire dagli anni Settanta, le architetture si riducevano a un solo mainframe al quale erano collegati i terminali "stupidi": quindi tutta l'elaborazione era effettuata dall'elaboratore centrale e i terminali servivano solo per le fasi di I/O.
Questa architettura non rientra nella tipologia client-server e può essere classificata come architettura a un solo livello (1 tier) ed è la situazione che si presentava prima dell'avvento dei sistemi distribuiti, come mostrato nella figura a lato.



Architettura a un livello – 1 tier

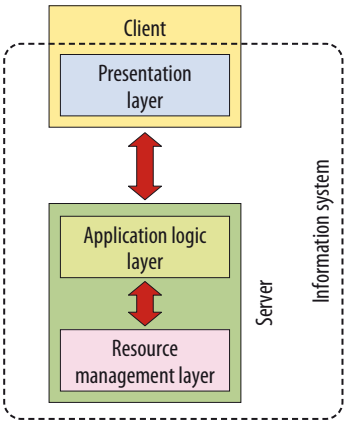
## Architettura a due livelli – 2 tier

Con l'avvento delle reti locali, a partire dagli anni Ottanta, sono nate le architetture client-server dove le funzionalità e le responsabilità erano suddivise su due livelli:

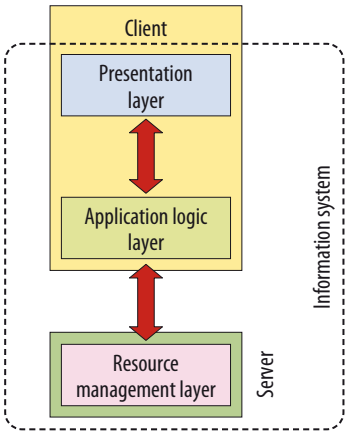
- un livello server;
- un livello client.

Possiamo individuare due sottocategorie di architetture a due livelli:

1. il modello thin-client, dove il server è responsabile della logica applicativa e gestione dei dati e il client è responsabile dell'esecuzione del software di presentazione.
2. il modello thick-client (o fat-client), dove il server è responsabile della gestione dei dati mentre il client è responsabile di presentazione e logica applicativa.



Architettura a due livelli – 2 tier – modello thin



Architettura a due livelli – 2 tier – modello thick

Il passaggio dal modello thin-client al modello thick-client è avvenuto verso la fine degli anni Ottanta con l'aumentare della potenza di calcolo dei PC: mentre il modello thin-client è stato il primo passo per effettuare il passaggio dai sistemi mainframe alle architetture distribuite, con il thick-client si è spostata parte della applicazione sul client favorendo la connessione di host di tipo diverso che era di fatto praticamente impossibile nei modelli precedenti.

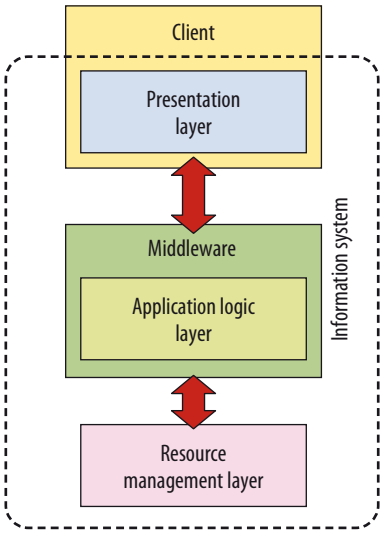
Il limite delle architetture client-server a due livelli è che sono poco scalabili dato che il server deve gestire la connessione e lo stato della sessione di ciascun client: questo carico di elaborazione porta alla limitazione del numero limitato di client che possono essere gestiti contemporaneamente.

Il passaggio al thick-client è stato alla base delle architetture distribuite moderne.

## Architettura a tre livelli – 3 tier

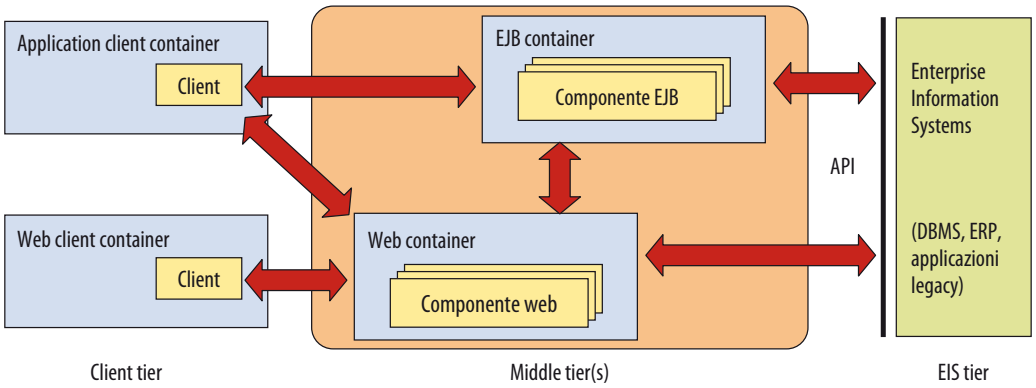
A partire dagli anni Novanta l'architettura client-server è a tre livelli e a ogni livello corrisponde uno strato architetturale, come precedentemente descritto:

- front-end o presentation tier: è l'interfaccia verso l'utente;
- logica applicativa o middle tier (business-tier);
- back-end con l'accesso alle risorse/ai dati, anche detto data tier (o resource-tier).



Architettura a tre livelli – 3 tier

I vantaggi dell'introduzione del middleware sono notevoli, soprattutto in termini di prestazioni, in quanto in questo modo si favorisce la distribuzione della quantità di elaborazione a scapito, però, dei tempi di comunicazione. Inoltre il sistema è facilmente scalabile in quanto all'aumentare delle richieste di un servizio è possibile aggiungere qualche server in grado di compensare il carico di lavoro ed è inoltre più tollerante ai guasti.
Anche in termini di sicurezza il modello a tre livelli porta notevoli vantaggi in quanto rende possibile l'introduzione di sicurezza a livello di servizio e quindi più facilmente gestibile.
Nei sistemi 3 tier è però più difficile la loro progettazione, lo sviluppo e l'amministrazione.
Vediamo per esempio l'architettura 3 tier a componenti della piattaforma Java EE (Enterprise Edition), rappresentata nel seguente schema:




È anche necessario osservare che in questo caso le corrispondenze tra strati e livelli non sono sempre così nette, come si può vedere dai due esempi di seguito riportati.

## Architettura a n tier

Le architetture client-server a *N* livelli sono una generalizzazione del modello client-server a tre livelli dove vengono scomposti e introdotti un numero qualunque di livelli e server intermedi.

Questa scomposizione viene effettuata per suddividere ulteriormente i compiti dei vari strati: prende anche il nome di applicazione multi-tier.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Nelle applicazioni Web con le tecnologie client-side:**
- **a** l'elaborazione avviene sul client
- **b** l'elaborazione avviene sul server
- **c** l'elaborazione avviene sia sul client sia server
- **d** l'elaborazione avviene prima sul client e successivamente sul server

**2 Indicare quali di queste affermazioni sono false per un modello client-server:**
- **a** una macchina server non può essere anche client
- **b** non sono gli host a essere server o client ma i processi
- **c** un host può essere contemporaneamente sia client che server
- **d** ogni processo server può a sua volta diventare client
- **e** un processo client non può diventare server
- **f** un servizio è un'entità astratta che viene fornito da uno o più server

**3 Quali caratteristiche non sono caratteristiche delle applicazioni di tipo enterprise:**
- **a** molti utenti concorrenti che richiedono i servizi
- **b** una logica applicativa poco complessa
- **c** archivi di grandi dimensioni con organizzazione di dati complessa e distribuita
- **d** richiedono modesti requisiti di sicurezza
- **e** gli utenti possono riferirsi a tutte le risorse in un modo esclusivo

**4 Le applicazioni client-server di tipo enterprise hanno (indicare la risposta errata):**
- **a** molti utenti concorrenti che richiedono i servizi
- **b** una logica applicativa complessa
- **c** archivi di grandi dimensioni con organizzazione di dati centralizzata
- **d** notevoli requisiti di sicurezza
- **e** sistemi transazionali

**5 Quale tra le seguenti non è una tipica applicazione delle architetture client-server?**
- **a** Telnet
- **b** HTTP
- **c** FTP
- **d** DNS
- **e** SMTP

**6 Nelle applicazioni possiamo individuare tre tipi principali di funzionalità che corrispondono a una struttura in tre strati o livelli (modello 3 tier):**
- **a** front-end, presentation tier, back-end
- **b** middle tier, front-end, presentation tier
- **c** back-end, data tier, front-end
- **d** presentation tier, middle tier, data tier
- **e** front-end, logica applicativa, middle tier

## VERO/FALSO

1 Un processo server può richiedere accesso ad altre risorse a un altro server. V F
2 Un'applicazione si dice distribuita se è installata su diversi PC. V F
3 Un socket è formato dalla coppia <indirizzo IP: numero della porta>. V F
4 In una comunicazione unicast un solo client si può connettere a un port. V F
5 Una applicazione client-server che implementa trasmissioni multicast è il server Web. V F
6 Nel modello thick-client il server è responsabile della gestione dei dati e della logica applicativa. V F
7 Il livello back-end con l'accesso alle risorse/ai dati è anche detto data tier. V F
8 Un'applicazione Java si colloca nell'application layer. V F

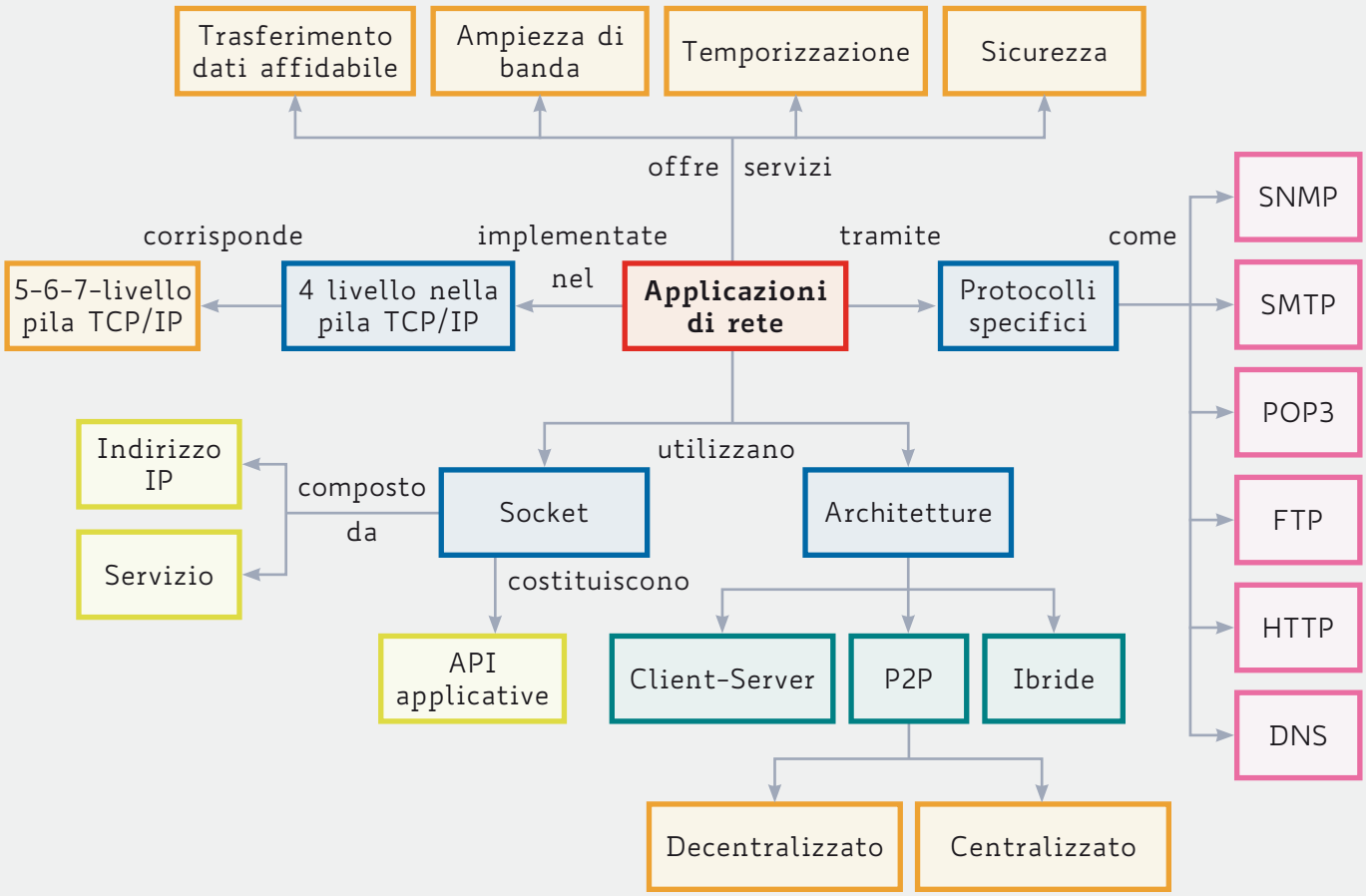
# 5 Le applicazioni di rete

**IN QUESTA LEZIONE IMPAREREMO...**

- il concetto di applicazione di rete
- le tipologie di applicazione
- a scegliere i protocolli per le applicazioni di rete

## MAPPA CONCETTUALE

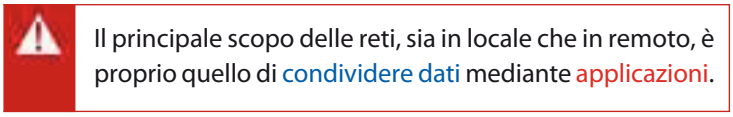
didattica inclusiva



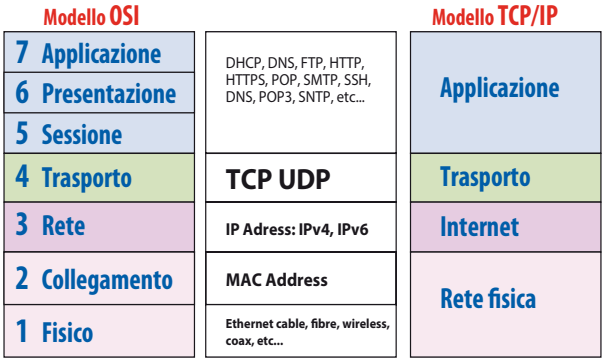

## Il modello ISO/OSI e le applicazioni

Nel modello ISO/OSI e TCP il livello delle applicazioni si occupa di implementare le applicazioni di rete che vengono utilizzate dall'utente finale, in cui il programmatore non si deve preoccupare dei livelli inferiori ma soltanto utilizzare le primitive di comunicazione messe a disposizione dai diversi protocolli degli altri strati: a tale livello possiamo individuare Internet come la struttura più complessa esistente.

Il modello a pila (stack) ISO/OSI può essere messo in relazione con altri stack protocollari, in modo da poterli comparare: la figura a fianco mostra la pila TCP/IP, formata da 4 livelli messa a confronto con la pila ISO/OSI. Possiamo notare che la pila TCP/ IP come strutturazione è analoga alla pila OSI, a eccezione del livello applicazione che racchiude i livelli 5, 6 e 7 e il livello che corrisponde alla rete fisica che ingloba i primi due livelli OSI (è anche chiamato di Network Interface o link layer).



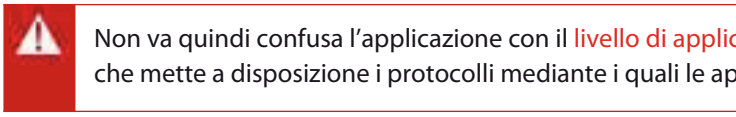
Il principale scopo delle reti, sia in locale che in remoto, è proprio quello di condividere dati mediante applicazioni.




Il livello applicazione implementa i vari protocolli, tra cui:

- SNMP: Simple Network Management Protocol;
- SMTP: Simple Mail Transfer Protocol;
- POP3: Post Office Protocol;
- FTP: File Transfer Protocol;
- HTTP: HyperText Transfer Protocol;
- DNS: Domain Name System.

Questi protocolli vengono utilizzati da tutte le applicazioni di rete generali come la posta elettronica, il Web, la condivisione di file P2P, i giochi multiutente via rete, la messaggistica istantanea, la telefonia via Internet, la videoconferenza in tempo reale, lo streaming di video-clip memorizzati, le autenticazioni in un calcolatore remoto (Telnet e SSH) ecc.
Oltre alle applicazioni generali offerte al pubblico di Internet, sulla rete "gira" un infinito numero di applicazioni proprietarie, cioè sviluppate all'interno di un'organizzazione per la gestione privata, come per esempio il collegamento tra filiali remote di una società commerciale, la connessione in tempo reale degli agenti col magazzino centrale ecc.

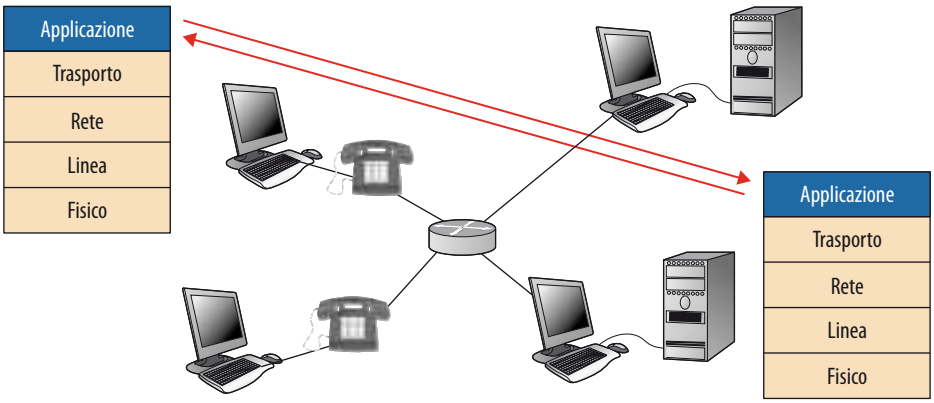


Non va quindi confusa l'applicazione con il livello di applicazione: il livello di applicazione è lo strato protocollare che mette a disposizione i protocolli mediante i quali le applicazioni possono comunicare tra host remoti.

## Applicazioni di rete

In generale un'applicazione di rete è costituita da un insieme di programmi che vengono eseguiti su due o più computer contemporaneamente: questi operano interagendo tra loro utilizzando delle risorse comuni, accedendo cioè concorrentemente agli archivi (database), mediante la rete di comunicazione che li connette.

L'applicazione di rete prende anche il nome di applicazione distribuita dato che non viene eseguita su di un solo elaboratore (concentrata).




Esempi di applicazioni di rete sono la posta elettronica, Telnet, FTP, newsgroup, chat www, telefonia Internet (Voice over IP), videoconferenze, audio e video on demand.

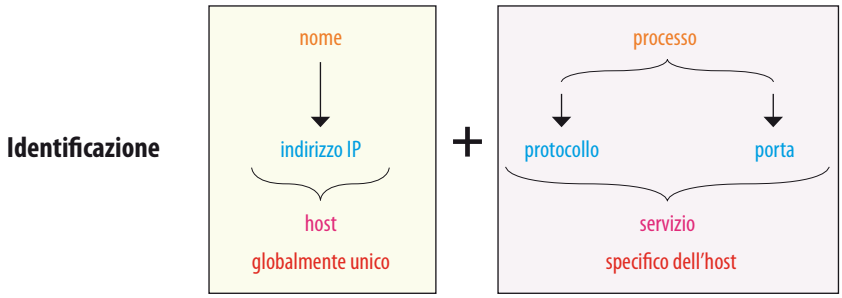
I processi hanno la necessità di scambiare messaggi con gli altri processi della medesima applicazione, sia che essi appartengano alla stessa rete locale oppure che siano remoti e quindi dislocati dall'altra parte del globo: per comunicare tra loro questi processi devono mettersi in "contatto" tramite i loro indirizzi e utilizzando i servizi offerti dal livello di applicazione.

## Identificazione di un servizio mediante socket

Affinché un processo, presente su un determinato host, invii un messaggio a un qualsiasi altro host, il processo mittente deve identificare il processo destinatario in modo univoco.

L'identificazione non può avvenire soltanto mediante l'indirizzo IP del destinatario, in quanto quest'ultimo individua soltanto l'host ma non il processo specifico e su un host sono sicuramente in esecuzione più processi contemporaneamente.

Pertanto l'identificazione deve tenere conto di **due informazioni**, l'indirizzo IP e il processo appartenente a quel determinato host: per individuare il processo a ogni servizio offerto da un server è assegnato un identificatore unico per l'host, chiamato numero di porta, il server si registra con il software del protocollo di trasporto usando il numero di porta (bind) e il client lo contatta usando identificatore dell'host (indirizzo IP) + l'identificatore del servizio (numero di porta).




Un numero di porta non può essere associato a più processi (server o client) ma un server può offrire più servizi, ciascuno su un'apposita porta.

### ESEMPIO

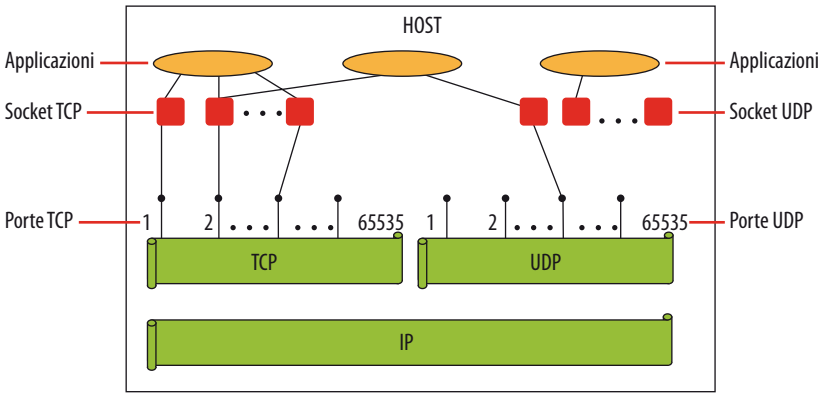
Un caso tipico è quello rappresentato da un server sul quale sono disponibili più servizi, tra i quali:

- e-mail: viene inviata usando il protocollo applicativo SMTP, quindi occorre inviare un messaggio opportunamente codificato alla porta TCP 25 del server;
- sito Web: per richiedere una pagina Web si usa il protocollo applicativo HTTP, che invia un opportuno messaggio di richiesta alla porta TCP 80 del server;
- per trasformare un nome di un calcolatore in un indirizzo IP si invia una opportuna richiesta alla porta UTP 53 del server che offre il servizio DNS.

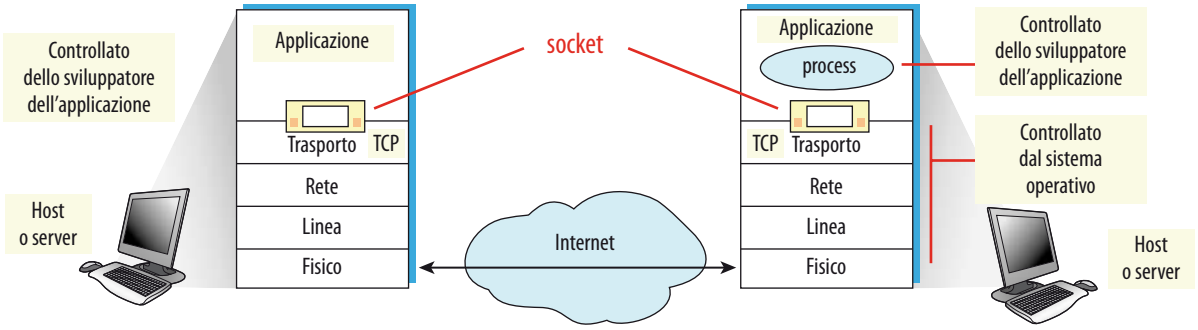
Per richiedere il servizio a questo server è quindi necessario specificare l'indirizzo dell'host e il tipo di servizio desiderato, quindi:

gli identificatori del server devono essere noti a priori al client per potersi connettere.

L'identificazione univoca avviene conoscendo sia l'indirizzo IP che il numero di porta associato al processo in esecuzione su un host: questo meccanismo è stato già introdotto nelle Lezioni precedenti e prende il nome di meccanismo dei socket.




Il processo mittente si affida all'infrastruttura offerta dallo strato di trasporto per consegnare il messaggio al socket del processo ricevente.




Come abbiamo visto in precedenza un socket è formato dalla coppia <indirizzo IP:numero della porta>; si tratta di un identificatore analogo a una porta, cioè a un punto di accesso/uscita; un processo che vuole inviare un messaggio lo fa uscire dalla propria "interfaccia" (socket del mittente) sapendo che un'infrastruttura esterna lo trasporterà attraverso la rete fino all'"interfaccia" del processo di destinazione (socket del destinatario).

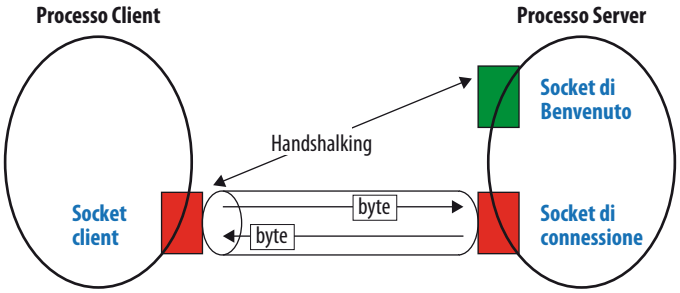
Un socket consente quindi di comunicare attraverso la rete utilizzando la pila TCP/IP ed è quindi parte integrante del protocollo: le API mettono a disposizione del programmatore gli strumenti necessari a codificare la connessione e l'utilizzo del protocollo di comunicazione.

L'applicazione di rete può essere vista come composta da due parti:
**1** una user agent, che funge da interfaccia tra l'utilizzatore dell'applicazione e gli aspetti comunicativi;
**2** l'implementazione dei protocolli che permettono all'applicazione di integrarsi con la rete.

## ESEMPIO

In un browser Web possiamo individuare queste due componenti:
**1** l'interfaccia utente che serve a visualizzare i documenti ricevuti dai client, a permettere la loro navigazione e a richiedere nuovi documenti specificando la loro URL;
**2** il motore del browser che è la parte che si preoccupa di inviare le richieste ai vari server e di ricevere le risposte.

Il server offre il servizio a più client e, quindi, deve gestire la concorrenza: il socket di connessione è anche chiamato socket di benvenuto e alla richiesta accettata di un client il server crea dinamicamente un nuovo thread e gli assegna un socket di connessione.




Il thread concorrente del server, con il suo flusso di controllo autonomo operante su memoria condivisa, soddisfa la richiesta del client attraverso la socket di connessione e, contemporaneamente, si rende disponibile ad attendere sul socket di benvenuto le successive richieste di altri client.

## Scelta dell'architettura per l'applicazione di rete

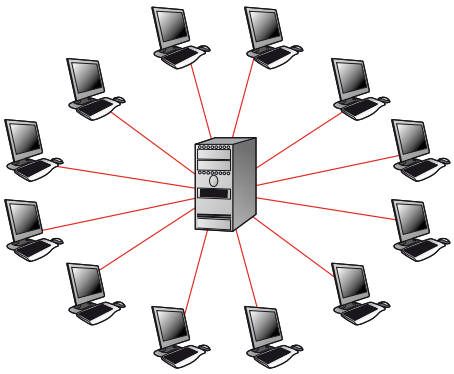
Il primo passo che il programmatore deve effettuare per progettare una applicazione di rete è la scelta della architettura dell'applicazione; sintetizziamo le principali caratteristiche delle architetture attualmente utilizzate, e cioè:

- client-server;
- peer-to-peer (P2P);
- architetture ibride (dove convivono client-server e P2P).

### Architettura client-server

Nella architettura client-server la caratteristica principale è che deve sempre esserci un server attivo che offre un servizio, restando in attesa che uno o più client si connettano a esso per poter rispondere alle richieste che gli vengono effettuate.

Un tipico esempio di questa architettura è il WWW, dove molteplici server (al limite uno per ogni sito pubblicato) possiedono le pagine (statiche o dinamiche) che saranno inviate ai client che ne fanno richiesta tramite i browser.



> Il server deve sempre essere attivo e deve possedere un indirizzo IP fisso dove può essere raggiunto dagli host client: quindi l'indirizzo IP deve essere statico, contrariamente a quello dei client che generalmente è dinamico.

Un client non è in grado di comunicare con gli altri client ma solo con il server: più client possono invece comunicare contemporaneamente con lo stesso server.

Se un server viene consultato contemporaneamente da molti client potrebbe non essere in grado di soddisfare tutte le richieste e potrebbe entrare in stato di congestione: è necessario virtualizzare la risorsa realizzando una server farm. Questa non è altro che un server con un unico hostname ma con più indirizzi IP, trasparenti rispetto al client, sui quali vengono dirottate le richieste di connessione (viene utilizzato per esempio da Google, Amazon, Facebook e tutti i siti che hanno una elevata affluenza di visite).

### Architettura peer-to-peer (P2P)

Nelle architetture peer-to-peer (P2P) abbiamo coppie di host chiamati peer (letteralmente "*persona di pari grado, coetaneo*") che dialogano direttamente tra loro.

**AREA DIGITALE**
**Applicazioni P2P**

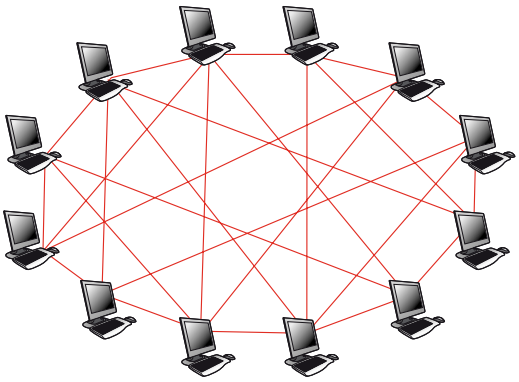
Un sistema P2P è formato da un insieme di entità autonome (peers), capaci di auto organizzarsi, che condividono un insieme di risorse distribuite presenti all'interno di una rete. Il sistema utilizza tali risorse per fornire una determinata funzionalità in modo completamente o parzialmente decentralizzato.

Nei sistemi P2P gli host possono essere visti come una comunità che collabora con il binomio dare e ricevere: ogni peer fornisce una risorsa e ottiene in cambio altre risorse. Gli esempi più noti sono rappresentati dalle P2P in ambito di condivisione di file, come Emule e Gnutella.

Un peer può anche decidere di offrire gratuitamente risorse, magari per la partecipazione a iniziative benefiche o di solidarietà, come per esempio gli aiuti ai terremotati, oppure di ricerca scientifica o sanitaria, come per esempio la ricerca sul cancro, mediante donazioni.

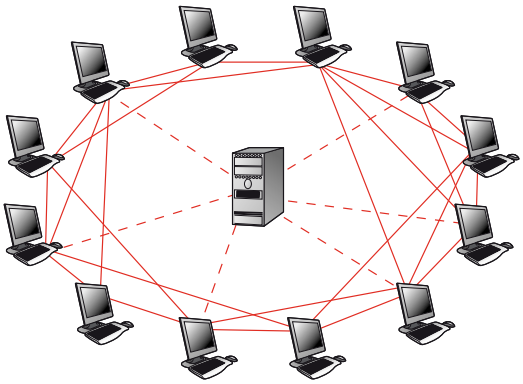
## P2P decentralizzato

Nella architettura completamente decentralizzata un peer ha sia funzionalità di client che di server (hanno funzionalità simmetrica e sono anche chiamati servent), ed è impossibile localizzare una risorsa mediante un indirizzo IP statico: vengono effettuati nuovi meccanismi di indirizzamento, definiti a livello superiore rispetto al livello IP.
Le risorse che i peer condividono sono i dati, la memoria, la banda ecc.: il sistema P2P è capace di adattarsi a un continuo cambiamento dei nodi partecipanti (churn) mantenendo connettività e prestazioni accettabili senza richiedere l'intervento di alcuna entità centralizzata (come un server).



## P2P centralizzato

Il P2P centralizzato è un compromesso tra il determinismo del modello client-server e la scalabilità del sistema puro: ha un server centrale (directory server) che conserva informazioni sui peer (index, cioè il mapping resorse-peer) e risponde alle richieste su quelle informazioni effettuando quindi la ricerca in modalità centralizzata.
I peer sono responsabili di conservare i dati e le informazioni (il server centrale non memorizza file), di informare il server del contenuto dei file che intendono condividere e di permettere ai peer che lo richiedono di scaricare le risorse condivise.

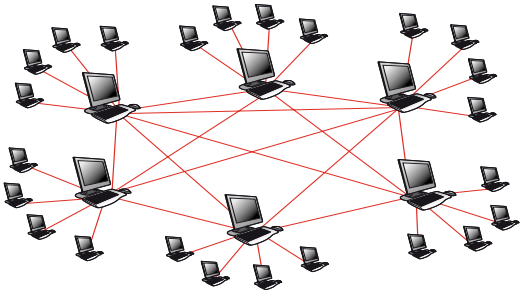


**ESEMPIO**

L'implementazione più famosa del P2P centralizzato è Napster, dove gli utenti si connettono a un server centrale nel quale pubblicano i nomi delle risorse che condividono.

## P2P ibrido (o parzialmente centralizzato)

Il P2P ibrido è un P2P parzialmente centralizzato dove sono presenti alcuni peer (detti supernodi o super-peer o ultra-peer) determinati dinamicamente (tramite un algoritmo di elezione) che hanno anche la funzione di indicizzazione: gli altri nodi sono anche chiamati leaf peer.



# Servizi offerti dallo strato di trasporto alle applicazioni

Le applicazioni richiedono allo strato di trasporto un insieme di servizi specifici oltre ai protocolli necessari per realizzarli, che possono essere standard o realizzati ad hoc.
Tutti i protocolli, sia standard che specifici, hanno in comune una particolarità: trasferire dei messaggi da un punto a un altro della rete.
Ogni applicazione deve scegliere tra i protocolli di trasporto quale deve adottare per realizzare un protocollo applicativo in base ai servizi che sono necessari alle specifiche esigenze della applicazione, che possono essere riassunte in:

- trasferimento dati affidabile;
- ampiezza di banda;
- temporizzazione;
- sicurezza.

## Trasferimento dati affidabile

Con trasferimento di dati affidabile intendiamo un servizio che garantisce la consegna completa e corretta dei dati: da una parte sappiamo che alcune applicazioni, come per esempio quelle di audio/video possono tollerare qualche perdita di dati senza compromettere lo scopo dell'applicazione mentre altre, come per esempio il trasferimento di file, richiedono un trasferimento dati affidabile al 100%.
A tale scopo il livello di trasporto mette a disposizione due protocolli:

3 UDP (User Datagram Protocol): il protocollo di trasporto senza connessione da utilizzarsi quando la perdita di dati è un fatto accettabile in quanto non è affidabile, non offre il controllo di flusso, il controllo della congestione, del ritardo e una banda minima;

4 TCP (Transmission Control Protocol): il protocollo orientato alla connessione da utilizzarsi quando la perdita di dati è un evento inaccettabile, ovvero quando il trasferimento deve essere affidabile; dà la garanzia di un trasporto senza errori o perdita di informazioni, effettua il controllo di flusso in quanto se il ricevente è più lento del mittente esso rallenterà per non sommergere il ricevente, esegue anche il controllo della congestione limitando il mittente se la rete è sovraccarica, ma non dà garanzie di banda minima.

Riportiamo una tabella dove sono indicati i protocolli utilizzati da alcune applicazioni:

| APPLICAZIONE | PROTOCOLLO A LIVELLO APPLICAZIONE | PROTOCOLLO DI TRASPORTO SOTTOSTANTE |
|---|---|---|
| Posta elettronica | SMTP (RFC 2821) | TCP |
| Accesso a terminali remoti | Telnet (RFC 854) | TCP |
| Web | HTTP (RFC 2616) | TCP |
| Trasferimento file | FTP (RFC 959) | TCP |
| Multimedia in streaming | Proprietario (RealNetworks) | TCP o UDP |
| Telefonia Internet | Proprietario (Vonage, Dialpad) | Tipicamente UDP |

**AREA DIGITALE**
**Comparison chart TPC vs UDP**

## Ampiezza di banda (Bandwidth) o Throughput

Alcune applicazioni, come per esempio quelle multimediali, per poter "funzionare" hanno bisogno di avere una garanzia sulla larghezza di banda minima disponibile, cioè possono richiedere un throughput garantito di R BSP: si pensi alla trasmissione di un evento in diretta in una Web-TV.
Altre applicazioni, invece, non hanno questo bisogno come prioritario, ma utilizzano in modo elastico l'ampiezza di banda che si rende disponibile: tipici esempi sono la posta elettronica o i sistemi FTP.

Internet ha una natura eterogenea e quindi i protocolli di trasporto non sono in grado di garantire la presenza di una certa quantità di banda per tutta la durata necessaria per trasmettere un messaggio: vedremo che mette a disposizione la possibilità di implementare un protocollo applicativo flessibile che realizza un circuito virtuale che si adatta se la banda è insufficiente.

### Temporizzazione

Alcune applicazioni, come la telefonia VoIP, i giochi interattivi, gli ambienti virtuali, per essere "realistiche" ammettono solo piccoli ritardi per essere efficaci: lo strato di trasporto non è in grado di garantire i tempi di risposta perché le temporizzazioni presenti assicurano un certo ritardo end-to-end tra le applicazioni.
Il protocollo TCP garantisce la consegna del pacchetto, ma non il tempo che ci impiega e neppure il protocollo UDP, nonostante sia più veloce del TCP, è temporalmente affidabile.
La soluzione, come vedremo, è quella di utilizzare un protocollo di trasporto in tempo reale, come RTP (Real Time Protocol), che è in grado di studiare i ritardi di rete e calibrare gli apparati e i collegamenti per garantire di restare nei limiti di tempo prefissati, scegliendo alternativamente quando utilizzare UDP e quando TCP.

### Sicurezza

Un'applicazione può richiedere allo strato di trasporto la cifratura di tutti i dati trasmessi in modo tale che anche se questi venissero intercettati da malintenzionati non si perda la riservatezza.
È quindi possibile che vengano richiesti dei servizi di sicurezza da applicare per garantire l'integrità dei dati e l'autenticazione end-to-end.

Il problema della sicurezza nelle reti riveste una grande importanza dato che le reti per loro natura non sono sicure: molteplici sono le minacce e i pericoli per i dati che sono presenti nei diversi host e che circolano sulla rete. Garantire la sicurezza di un sistema informativo significa impedire a potenziali soggetti attaccanti l'accesso o l'uso non autorizzato di informazioni e risorse.

## Conclusioni

Riportiamo in una tabella i requisiti richiesti al servizio di trasporto da parte di alcune applicazioni:

| APPLICAZIONI | TOLLERANZA ALLA PERDITA DEI DATI | AMPIEZZA DI BANDA | SENSIBILITÀ DAL TEMPO |
|---|---|---|---|
| Trasferimento file | No | Variabile | No |
| Posta elettronica | No | Variabile | No |
| Documenti Web | No | Variabile | No |
| Audio/Video in tempo reale | Sì | Audio: da 5 Kbps a 1 Mbps<br>Video: da 10 Kbps a 5 Mbps | Sì, centinaia di ms |
| Audio/Video memorizzati | Sì | | Sì, pochi secondi |
| Giochi interattivi | Sì | Fino a pochi K | Sì, centinaia di ms |
| Messaggistica istantanea | No | Variabile | Sì e no |

# VERIFICA... le conoscenze

**AREA DIGITALE**
Esercizi per il recupero e l'approfondimento

## SCELTA MULTIPLA

**1 Quale tra i seguenti non è un protocollo applicativo?**
**a** HTTP
**b** POP3
**c** DNS
**d** SMNP
**e** SMTP
**f** FTP

**2 Quale tra le seguenti non è un'applicazione di rete?**
**a** Posta elettronica
**b** Condivisione di file P2P
**c** Scheduler dei processi
**d** Telefonia via Internet
**e** Videoconferenza in tempo reale
**f** TV in streaming
**g** Telnet

**3 Che cosa significa SMTP?**
**a** Simple Message Transfer Protocol
**b** System Mail Transfer Protocol
**c** Simple Mail Transfer Protocol
**d** System Message Transfer Protocol

**4 Qual è il significato di API?**
**a** Application Protocol Internet
**b** Application Protocol Interface
**c** Application Programming Interface
**d** Application Programming Internet

**5 Quali tra le seguenti non sono architetture di una applicazione di rete?**
**a** client-server
**b** B2B
**c** B2C
**d** P2P

**6 Quale tra i seguenti servizi offerti non è garantito dallo strato di trasporto alle applicazioni?**
**a** trasferimento dati affidabile
**b** ampiezza di banda
**c** velocità di comunicazione
**d** temporizzazione sicurezza
**e** sicurezza

## VERO/FALSO

**1** Il protocollo HTTP è a livello di trasporto. V F
**2** Il protocollo TCP è a livello di applicazione. V F
**3** L'applicazione di rete prende anche il nome di applicazione distribuita. V F
**4** L'applicazione di rete prende anche il nome di applicazione multiutente. V F
**5** Sullo stesso host possono essere in esecuzione molti processi. V F
**6** Sullo stesso host può essere in esecuzione una sola applicazione. V F
**7** Un protocollo è una parte integrante di una applicazione ed è sviluppato all'interno di essa. V F
**8** In un'architettura client-server gli indirizzi IP devono essere statici. V F
**9** Il termine servent è ottenuto dalla contrazione server-client. V F
**10** Nel P2P decentralizzato i servent hanno indirizzo IP statico. V F

# ESERCIZI IN LABORATORIO 1

## IL LINGUAGGIO XML

### Generalità

L'eXtensible Markup Language (XML) non è un linguaggio di markup né l'evoluzione HTML: è un metalinguaggio di mark-up, cioè un linguaggio che permette di definire altri linguaggi di markup.
XML non ha tag predefiniti e non serve né per programmare né per definire pagine Web, è costituito da un insieme standard di regole sintattiche per modellare la struttura di documenti e dati.

> Le specifiche ufficiali sono state definite dal W3C (World Wide Web Consortium) e sono consultabili all'indirizzo http://www.w3.org/XML.

Gli obiettivi iniziali che diedero vita a XML erano rivolti alla soluzione di un problema di standard per il Web ma, una volta definito, grazie alla caratteristica di essere abbastanza generale, venne subito utilizzato nei più disparati contesti, dalla definizione della struttura di documenti allo scambio di informazioni tra sistemi diversi, dalla rappresentazione di immagini alla definizione di formati di dati.

**ESEMPIO**

Nella programmazione lato server, come vedremo, viene utilizzato per la definizione del deployment descriptor, cioè del file web.xml che viene utilizzato per informare il server in cui è installata un'applicazione J2EE riguardo alla configurazione dell'applicazione stessa.
Inoltre in esso possono essere definiti parametri di inizializzazione, la mappatura dei percorsi e, in generale, tutto ciò che contiene informazioni importanti per la corretta esecuzione dell'applicazione.

Vale la pena sottolineare che un documento XML è un documento di testo e può essere letto e modificato con qualsiasi editor di testo.

> In particolare, XML non è:
> - un linguaggio di presentazione come HTML: il markup XML definisce il significato dei dati e non dice nulla sullo stile dei dati;
> - un linguaggio di programmazione come Java: un documento XML non è in grado di eseguire calcoli;
> - un protocollo di trasporto come http: XML non trasferisce i dati attraverso la rete;
> - un DBMS come Access o MySql: XML non memorizza e recupera i dati.

### Il linguaggio XML

XML è un linguaggio formale, cioè come tutti i linguaggi è definito da un insieme di regole formali (una grammatica) che dicono esattamente come comporre un documento XML ma non possiede tag predefiniti: la principale differenza tra l'XML e l'HTML è che l'HTML si occupa di visualizzare i dati, mentre l'XML si occupa di descrivere la natura dei dati che compongono le informazioni.
XML consente di eseguire il markup dei dati: i dati sono inclusi nei documenti XML come stringhe di testo ed è circondato da un markup di testo che li descrive.

> Possiamo definire anche l'XML come un metalinguaggio che ha lo scopo di rappresentare contenuti testuali organizzati in modo gerarchico.

XML è semplicemente un formato flessibile per contrassegnare i dati con tag leggibili dall'uomo.

# 1 ESERCIZI IN LABORATORIO

La definizione dell'XML prescinde dal suo campo di applicazione e quindi anche dalla tecnologia e dall'ambiente di lavoro in cui viene utilizzato: l'XML non è un linguaggio per il Web, così come non è un meccanismo per la sola rappresentazione di contenuti provenienti da basi di dati.
Poiché, proprio per la sua natura, il linguaggio consente un set estensibile di tag markup, può essere adattato per soddisfare molte esigenze diverse.
La caratteristica principale dell'XML è quella che permette di rappresentare contenuti in un formato compatibile con qualsiasi sistema hardware e software in quanto si tratta di un semplice file in formato testo.

## File XML

I file XML possono essere trasmessi tra aziende e utenti diversi, senza alcun vincolo, dato che sono file di testo: hanno come suffisso .XML.

Quindi i file in formato XML:
- consentono di separare i dati dalla rappresentazione grafica;
- possiedono un insieme unico di regole di sintassi che consentono di leggere e scrivere, nello stesso modo e con strumenti analoghi, i contenuti rappresentati;
- analogamente a HTML sono basati su tag del tipo <nome></nome>; i tag permettono di leggere le informazioni in essi contenute, in base al loro nome e non necessariamente alla loro posizione;
- a ogni documento XML può essere associata la definizione della struttura utilizzata per renderlo facilmente leggibile e per rendere rapido il processo di verifica della struttura del documento.

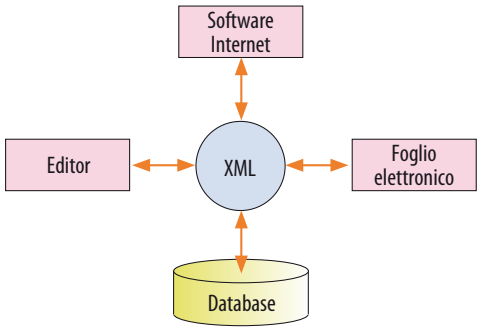
# Utilizzo dell'XML

Quando si usa l'HTML per visualizzare dati, questi ultimi sono memorizzati all'interno del codice HTML; con l'XML, invece, i dati possono essere memorizzati in un file XML separato dal documento HTML.
In questo modo possiamo dedicarci all'uso dell'HTML per il layout e la presentazione dei dati e avere la garanzia che i dati "sottostanti" non richiedano al file HTML alcun cambiamento.

Con l'XML i dati strutturati possono essere memorizzati all'interno delle pagine HTML: si definiscono così le cosiddette aree data islands (isole dei dati).

## Scambio di dati con l'XML

I sistemi di elaborazione e i database contengono dati spesso incompatibili tra loro: questo è stato uno dei maggiori problemi per gli sviluppatori, che impiegavano una notevole quantità di tempo per scambiare dati via Internet tra computer diversi. La conversione dei dati in formato XML può ridurre notevolmente la complessità della comunicazione di informazioni e può creare dati leggibili da diversi sistemi di elaborazione. Il formato XML è infatti divenuto compatibile con un numero sempre più ampio di applicazioni.
L'XML è ormai uno standard per quanto concerne lo scambio di informazioni di tipo commerciale via Internet.




## Condivisione dei dati

Poiché i dati XML sono memorizzati in formato di testo puro, l'XML fornisce un modo di condividere i dati slegato sia dal software sia dall'hardware. Ciò rende molto più facile creare dati con cui possono lavorare applicazioni diverse.

# ESERCIZI IN LABORATORIO 1

## Memorizzazione dei dati

L'XML può essere utilizzato per memorizzare dati in un file o in un database; consente di scrivere applicazioni per archiviare e recuperare informazioni dagli archivi, mentre applicazioni di tipo generico possono essere usate per visualizzare i dati. L'XML può rendere dunque più accessibili gli archivi di dati.

## La sintassi XML

L'XML è leggibile su qualsiasi piattaforma, ma per ottenere questo risultato occorre prestare un minimo di attenzione nella costruzione dei documenti.
Un documento è costituito da due parti:

1. prologo: contiene una dichiarazione XML e il riferimento (opzionale) ad altri documenti che ne definiscono la struttura o direttive di elaborazione;
2. corpo: è il documento XML vero e proprio.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="greco.css"?>

<radice>
  <!--  Questo è un commento  -->
  <figlio>
  ...
  </figlio>
  <figlio>
  ...
  </figlio>
</radice>
```

Marcatore di inizio

Marcatore di fine

Prologo

Corpo

Il prologo inizia con una prima riga, detta XML declaration, che può anche contenere informazioni sul set di caratteri:

```
<?xml version="1.0" ?>
<?xml version="1.0" encoding= "UTF-8"?>
<?xml version="1.0" encoding= "UTF-8" standalone="yes"?>
```

```
<?xml version="1.0" encoding= "UTF-8" standalone="yes"?>
```

Versione di XML

Codifica dei caratteri

Indica se il documento è conforme a una sintassi esterna (standalone="no")

Inoltre può contenere indicazioni riferimenti a documenti esterni utili per il trattamento del documento.

```
<?xml-stylesheet type="text/css" href="greco.css"?>      // istruzioni di elaborazione (Processing instruction)
<!DOCTYPE prodotti SY\STEM "prodotti.dtd">               // istruzioni di validazione (Doctype declaration)
```

# 1 ESERCIZI IN LABORATORIO

Nell'XML si possono comunque inserire anche commenti liberi, come nell'HTML, utilizzando la stessa simbologia utilizzata nell'HTML(<!-- commento -->).
Il corpo, cioè il contenuto del documento, deve corrispondere a una gerarchia di marcatori (tag) non sovrapposti, quindi i tag che si aprono per ultimi devono chiudersi per primi e ogni tag deve avere una chiusura.

```
- <pagina>
    <titolo>esempio di documento in stile XML</titolo>
  - <testo>
      <paragrafo>testo testo testo</paragrafo>
    </testo>
  </pagina>
```

Deve sempre esserci un tag che racchiuda tutti gli altri, che viene chiamato root (radice). Inoltre l'XML è case sensitive, pertanto un **tag maiuscolo** è diverso dallo stesso **tag scritto in minuscolo**.

## Documenti ben formati e documenti validi

In XML ci sono regole sintattiche, cioè come bisogna scrivere le informazioni all'interno dei documenti, e ci possono essere regole semantiche, cioè cosa possiamo scrivere in un documento XML che, però, non sono obbligatorie da rispettare.

Un documento XML che rispetta le regole sintattiche si dice ben formato (well-formed).
Un documento XML che rispetta le regole sintattiche e le regole semantiche si dice valido.

Per avere un documento XML ben formato le regole sintattiche da rispettare sono le seguenti:
- deve contenere una dichiarazione corretta;
- il corpo deve avere un unico elemento radice (esclusi eventuali commenti);
- ogni elemento deve avere un tag di apertura e uno di chiusura; se l'elemento è vuoto si può utilizzare la forma abbreviata (<nometag/>);
- gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'ordine inverso dei rispettivi tag di apertura;
- i nomi dei tag di apertura e chiusura devono coincidere, anche in termini di maiuscole e minuscole;
- i nomi dei tag non devono iniziare con il carattere underscore (_) o con un numero e non devono contenere spazi al loro interno;
- i valori degli attributi devono sempre essere racchiusi tra singoli o doppi apici.

Un documento ben formato può non essere valido rispetto a una grammatica, mentre un documento valido è necessariamente ben formato.

### ESEMPIO

**DOCUMENTO XML PER DEFINIRE UNA RUBRICA**

Supponiamo che si vogliano scambiare informazioni di database su Internet utilizzando un browser e inviare al server le informazioni, per esempio un questionario compilato dagli utenti. Questo processo richiede un formato che può essere personalizzato per un utilizzo specifico. L'XML è la soluzione per questo tipo di problema: infatti la rappresentazione visiva di un documento XML risulta elementare, come mostra l'esempio che segue.

# ESERCIZI IN LABORATORIO 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <rubrica>
  - <persona>
      <nome>Paolo</nome>
      <cognome>Camagni</cognome>
      <telefono_casa>0288888888</telefono_casa>
      <telefono_lavoro>02999999999</telefono_lavoro>
      <indirizzo_email>paolo@camagni.it</indirizzo_email>
    </persona>
  - <persona>
      <nome>Riccardo</nome>
      <cognome>Nikolassy</cognome>
      <telefono_casa>0277777777</telefono_casa>
      <telefono_lavoro>0299999999</telefono_lavoro>
      <indirizzo_email>riccardo@nikolassy.it</indirizzo_email>
    </persona>
  </rubrica>
```

La prima linea del codice indica la versione dell'XML e il set di caratteri utilizzato: caratteri ISO- 8859-1 (Latin-1/West European).
La riga successiva descrive l'elemento radice del documento (rubrica). L'elemento rubrica possiede come elemento figlio persona che a sua volta si compone di altri elementi figli (nome, cognome, telefono ecc.).

## Elementi dell'XML

Gli elementi dell'XML sono modificabili ed espandibili (possono essere definiti, e si potrebbe dire "inventati", da chi crea il documento XML senza alcuna limitazione) e devono seguire regole molto semplici per specificare i loro nomi.

### ESEMPIO

**MESSAGGIO TRA DUE UTENTI**

Esempio di documento XML che indica un messaggio tra due utenti:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <messaggio>
    <mittente>Mario</mittente>
    <destinatario>Maria</destinatario>
    <oggetto>promemoria</oggetto>
    <testo> ricordati che siamo d'accordo di vederci venerdì sera </testo>
  </messaggio>
```

Se l'autore del documento XML aggiunge alcune informazioni ulteriori, per esempio la data:

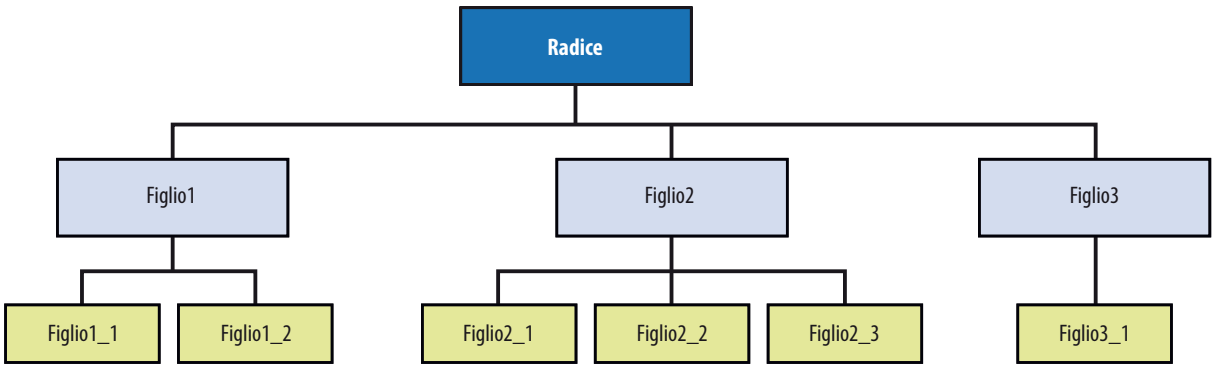
```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <messaggio>
    <data>2017-07-12</data>
    <mittente>Mario</mittente>
    <destinatario>Maria</destinatario>
    <oggetto>promemoria</oggetto>
    <testo> ricordati che siamo d'accordo di vederci venerdì sera </testo>
  </messaggio>
```

l'applicazione è ancora in grado di effettuare l'estrazione degli elementi <messaggio>, <mittente>, <destinatario>, <testo> per produrre il medesimo output senza difficoltà.

# 1 ESERCIZI IN LABORATORIO

## Gerarchia degli elementi

Gli elementi di un documento XML sono da considerarsi come nodi appartenenti alla struttura di un albero.



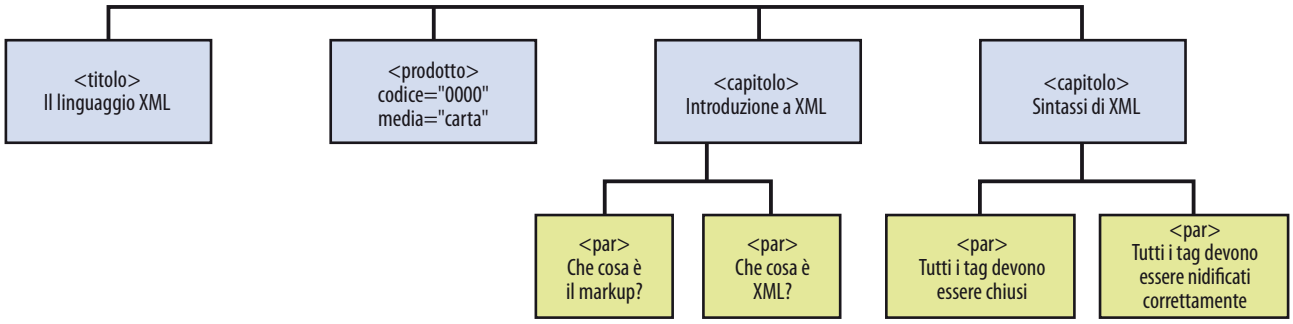

### ESEMPIO

Si consideri la descrizione del contenuto di un libro che parla di XML:

```
il linguaggio XML
Introduzione a XML
  Che cosa è il markup
  Che cosa è XML
Sintassi di XML
  Tutti i tag devono essere chiusi
  Tutti i tag devono essere nidificati correttamente
```

La precedente struttura di dati può essere codificata in XML mediante il codice che segue:

Dal punto di vista grafico, il documento XML può essere rappresentato con un albero:



# ESERCIZI IN LABORATORIO 1

Nell'esempio precedente, <libro> è l'elemento XML radice; <titolo>, <prodotto> e <capitolo> sono elementi figlio di <libro> e sono fratelli perché hanno lo stesso padre.

> Un elemento è tutto ciò che va dal tag iniziale al tag finale e può avere diversi tipi di contenuto: può essere formato da altri elementi, può avere un contenuto misto oppure semplice e può anche essere vuoto.

Nell'esempio precedente:
- <libro> ha un contenuto formato da altri elementi figli;
- <capitolo> ha un contenuto di tipo misto perché contiene sia testo sia altri elementi;
- <par> ha un contenuto di tipo semplice perché contiene solo testo;

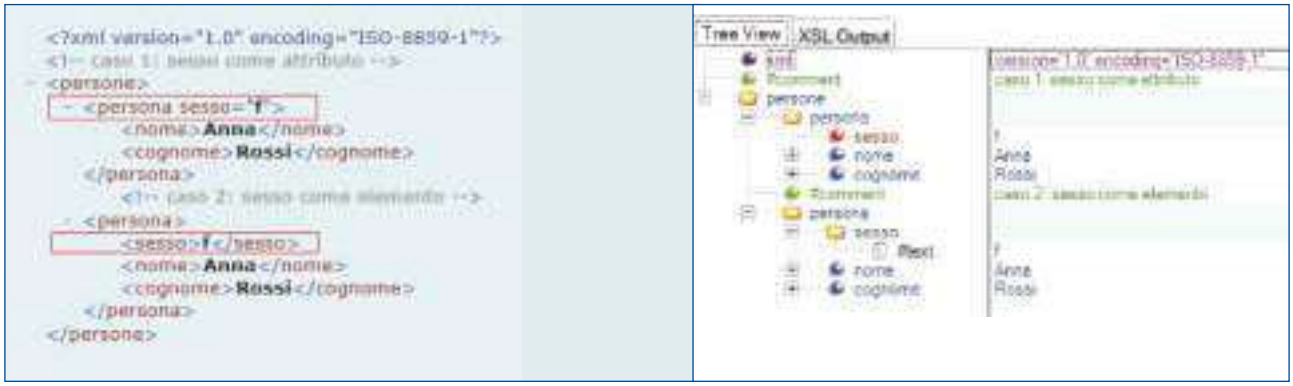
Solo <prodotto> ha attributi:
- l'attributo di nome <media> ha valore "carta";
- l'attributo di nome <codice> ha valore 1234.

## Differenza tra attributi ed elementi

I dati possono essere rappresentati attraverso gli attributi oppure attraverso gli elementi: è da ricordare che in XML la regola sintattica per gli attributi prevede sempre che siano racchiusi tra virgolette.

### ESEMPIO

Vediamo un esempio dove riportiamo a confronto le due possibili rappresentazioni:




Nel primo caso, <sesso> è un attributo.
Nel secondo caso <sesso> è un elemento figlio della radice <persona>.
Entrambe le codifiche forniscono la medesima informazione e non esistono regole precise per decidere quando è meglio usare attributi e quando è meglio scegliere elementi figlio.
In generale, si può dire che gli attributi sono opportuni in HTML, ma è meglio evitarne l'uso con l'XML; infatti utilizzando gli attributi potrebbero nascere alcuni problemi dato che:
- gli attributi non possono contenere valori multipli (gli elementi figlio sì);
- gli attributi non descrivono la struttura (gli elementi figlio sì);
- gli attributi sono più difficili da elaborare da parte di un codice di programma.

## Conflitti sui nomi

Spesso capita di dare nomi ed elementi (o attributi) con significati diversi all'interno dello stesso documento.

# 1 ESERCIZI IN LABORATORIO

ESEMPIO

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <libro>
    <titolo>Libro di informatica</titolo>
  - <capitolo>
      <titolo>Documento in stile XML</titolo>
      <testo>testo testo testo</testo>
    </capitolo>
  </libro>
```

Il problema di omonimie viene risolto ricorrendo al concetto di "spazio dei nomi" (namespace): si usano prefissi che identificano il vocabolario di appartenenza di elementi e attributi, dove a ogni prefisso viene associato un URI (Uniform Resource Identifier) che "è un alias per l'URI stesso", che altro non è che un URL.
Per definire un namespace si usa la seguente sintassi:

```
xmlns:NamespacePrefix="NamespaceURI"
```

Modifichiamo l'esempio precedente aggiungendo dove necessario gli appositi URL.

ESEMPIO

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <lb:libro xmlns:lb="mioSito.com/libri">
    <lb:titolo>Libro di informatica</lb:titolo>
  - <ca:capitolo xmlns:ca="mioSito.com/capitoli">
      <ca:titolo>Documento in stile XML</ca:titolo>
      <ca:testo>testo testo testo</ca:testo>
    </ca:capitolo>
  </lb:libro>
```

## Un editor e visualizzatore per XML

L'editor di file XML può essere creato con un qualunque editor di testi e la sua visualizzazione viene effettuata di solito direttamente con un browser HTML.
Chi normalmente lavora con una grande quantità di file XML utilizza specifici programmi come XML viewer o un editor XML, che aiutano il programmatore e semplificano la gestione dei file XML più complessi.
Tra i software presenti indichiamo XMLNotepad, un prodotto open source scaricabile gratuitamente all'indirizzo https://archive.codeplex.com/?p=xmlnotepad, che permette sia di visualizzare sia di editare i file XML.
Nei nostri esempi questo prodotto è stato utilizzato per visualizzare l'albero XML.

## Esercitazione: il magazzino

1. **Apri il file magazzino.**
   - Modifica il file XML aggiungendo il prologo corretto.
   - Aggiungi due articoli a piacere.
   - Modifica l'intero file sostituendo gli elementi visualizzati con gli attributi.
2. Crea lo schema ad albero per la struttura dei dati che descrivono le caratteristiche di un'automobile (nome, marca, cilindrata, prezzo, dimensioni, tipo motore e così via).
   Crea il file XML corrispondente allo schema ad albero appena realizzato.

# ESERCIZI IN LABORATORIO 1

**3.** Crea uno schema ad albero per la struttura dei dati presenti in una fattura (si tenga conto che una fattura è composta da tre parti: l'intestazione che contiene i dati anagrafici del destinatario e del mittente, la parte tabellare che contiene la lista degli articoli venduti e il piede che presenta il totale della fattura e i totali per l'IVA).
Crea il file XML corrispondente allo schema ad albero appena realizzato.

**4.** Data la seguente tabella Categorie:

| ID | NOME CATEGORIA | DESCRIZIONE |
|---|---|---|
| 1 | Beverages | Soft drinks, coffees, teas, beers and ales |
| 2 | Condiments | Sweet and savory sauces, relishes, spreads and seasonings |
| 3 | Confections | Desserts, candies and sweet breads |
| 4 | Dairy Products | Cheeses |
| 5 | Grains/Cereals | Breads, crackers, pasta and cereal |
| 6 | Meat/Poultry | Prepared meats |
| 7 | Produce | Dried fruit and bean curd |

costruisci il file XML corrispondente e associa a esso il file CSS opportuno per mostrare i dati nel modo seguente:

| Nome categoria | Descrizione |
|---|---|
| Beverages | Soft drinks, coffees, teas, beers, and ales |
| Condiments | Sweet and savory sauces, relishes, spreads, and seasonings |
| Confections | Desserts, candies, and sweet breads |
| Dairy Products | Cheeses |
| Grains/Cereals | Breads, crackers, pasta, and cereal |
| Meat/Poultry | Prepared meats |
| Produce | Dried fruit and bean curd |

**5.** Dato il seguente documento testuale, in formato ASCII, di un curriculum vitae:

```
                    Curriculum Vitae di Mario Rossi
            Dati personali
Nome        Mario
Cognome     Rossi
Residenza   Via Milano, 40 - 10100 Roma
e-mail      mario.rossi@email.it
tel.        01-2345678
cell.       111-222333444
Nato il     11-6-1960 a Como (Co)
Nazionalità Italiana
Stato civile Coniugato
            Esperienze professionali
1990-1995 Computer Software s.r.l – Como
Ruolo: Analista programmatore
Attività: Sviluppo di Sistemi Gestionali per piccole imprese tessili
1995-2020 Docente ITIS Magistri Cumacini – Como
Ruolo: Docente
Attività: Tecnologie e progettazione – Corso informatica 1
```

effettua la sua codifica in XML.

# 2 ESERCIZI IN LABORATORIO

## LO SCHEMA DTD DELLE PAGINE XML

### La validazione con il Document Type Definition (DTD)

È possibile definire con uno schema la struttura dei tag di markup consentiti in una particolare applicazione XML: il linguaggio più ampiamente supportato che permette di descrivere lo schema definito dalla specifica XML 1.0 è il Document Type Definition (DTD).

> Un file DTD fornisce uno strumento per la validazione dei documenti XML ma non è scritto rispettando la sintassi XML.

Un DTD consente di porre alcuni vincoli sulla struttura di un documento XML: in esso sono elencati tutti gli elementi, gli attributi e le entità utilizzati dal documento e il contesto in cui li si utilizza.
I DTD non dicono mai nulla sul tipo di contenuto di un elemento o sul valore di un attributo: non si può, per esempio, imporre che il prezzo di un prodotto sia un numero reale, che il nome sia una stringa o che la data di nascita sia una data.
I vantaggi che si hanno usando XML validati con i linguaggi di schema sono l'aumento della produttività e la possibilità di sviluppare sistemi aperti e interoperabili.
Per applicare un DTD a un documento XML dobbiamo inserire nel suo prologo, sotto la riga del'XML Declaration, una dichiarazione con questa sintassi:

```
<!DOCTYPE <root-element> SYSTEM "<filename>">
```

dove:
- root-element è il nome dell'elemento radice;
- SYSTEM definisce documenti di utilizzo locale;
- filename è il file che contiene il DTD.

Lo schema DTD è costituito da un elenco di dichiarazioni (markup declaration) che descrivono la struttura del documento, dove ciascuna dichiarazione definisce:
- gli elementi (element) di un documento XML;
- il modello di contenuto di ogni elemento (content model), ovvero gli elementi che contiene e le loro relazioni (un elemento può essere vuoto);
- la lista degli attributi associati a ciascun elemento, il loro tipo e il loro valore.

Per dichiarare un elemento, che può essere indicato una sola volta, si usa la sintassi:

```
<!ELEMENT element-name content-model>
```

Il contenuto (content-model) può essere di 4 tipi:
**1** EMPTY: parola chiave che indica l'elemento vuoto;
**2** ANY: indica che si può inserire testo o elementi qualsiasi (purché dichiarati nel DTD);
**3** l'elenco di elementi figli specifici con ordine determinato (content-model Children);
**4** un testo più l'elenco di elementi figli senza ordine specifico (content-model Mixed).

> ⚠ I due ultimi tipi non sono indicati da una parola chiave ma attraverso la notazione usata nella definizione.

# ESERCIZI IN LABORATORIO 2

Nel content-model Children viene specificata:
- la sequenza di elementi figli che devono comparire separati da virgola, specificando il loro ordine (E1,E2,…,En);

oppure
- la lista di elementi figli che possono comparire in alternativa: (E1|E2|…|En).

Per ogni elemento è possibile stabilire la sua occorrenza utilizzando gli operatori ?, +, *:
- ? = zero o 1;
- + = 1 o più;
- * = zero o più.

## ESEMPIO

In questo primo esempio realizziamo lo schema DTD da applicare a un file XML che contiene i seguenti record:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE posta SYSTEM "posta.dtd">
- <posta>
  - <messaggio>
      <mittente>Romeo</mittente>
      <destinatario>Giulietta</destinatario>
      <intestazione>Appuntamento</intestazione>
      <testo>Ricordati che stasera verro' sotto il tuo balcone!</testo>
    </messaggio>
  - <messaggio>
      <mittente>Adamo</mittente>
      <destinatario>Eva</destinatario>
      <intestazione>Merenda</intestazione>
      <testo>Porta tu la frutta, basta che non siano mele!</testo>
    </messaggio>
  </posta>
```

Osserviamo che nella seconda riga viene inserito il nome del file dtd che conterrà tale descrizione.

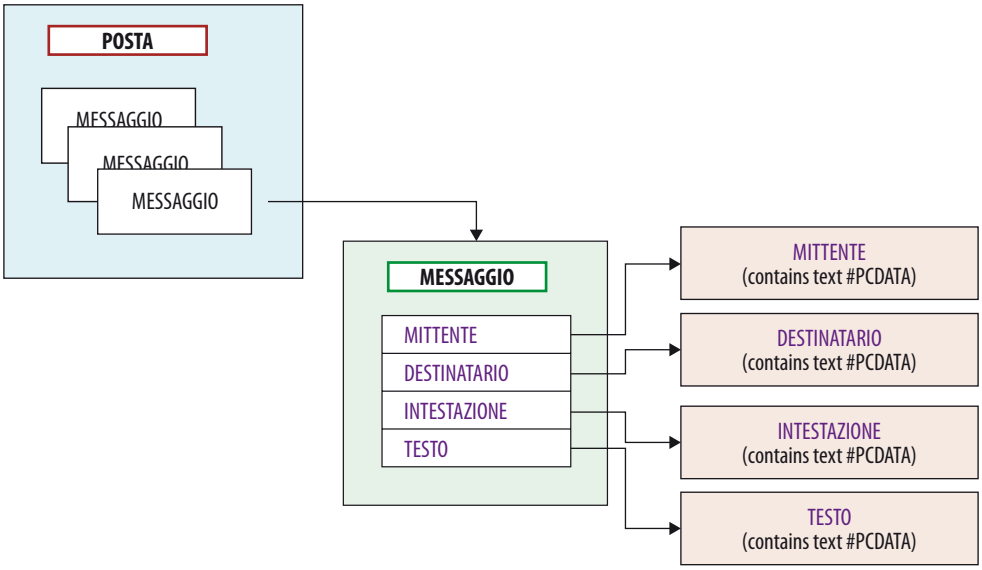
Nel file XML Individuiamo <posta> come elemento radice composto da due record <messaggio>: nello schema DTD si deve specificare la struttura del record che è composto da:
- un elemento mittente contenente testo;
- un elemento destinatario contenente testo;
- un elemento intestazione contenente testo;
- un elemento testo contenente testo.

Lo schema che descrive questa situazione è il seguente:

```
<!ELEMENT posta (messaggio)*>
<!ELEMENT messaggio (mittente, destinatario, intestazione, testo)>
<!ELEMENT mittente (#PCDATA)>
<!ELEMENT destinatario (#PCDATA)>
<!ELEMENT intestazione (#PCDATA)>
<!ELEMENT testo (#PCDATA)>
```

La prima riga del DTD definisce l'elemento messaggio che è vincolato a contenere i quattro elementi specificati che devono rispettare l'ordine di apparizione.

# 2 ESERCIZI IN LABORATORIO




Riportiamo il file XML che viene visualizzato in un editor generico, per esempio SublimeText, oppure in un prodotto specifico come XML Notepad: entrambi i prodotti sono open source e, quindi, scaricabili gratuitamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE posta SYSTEM "posta.dtd">
<posta>
  <messaggio>
    <mittente>Romeo</mittente>
    <destinatario>Giulietta</destinatario>
    <intestazione>Appuntamento</intestazione>
    <testo>Ricordati che stasera verro' sotto il tuo balcone!</testo>
  </messaggio>
  <messaggio>
    <mittente>Adamo</mittente>
    <destinatario>Eva</destinatario>
    <intestazione>Merenda</intestazione>
    <testo>Porta tu la frutta, basta che non siano mele!</testo>
  </messaggio>
</posta>
```

Questi elementi sono di tipo PCDATA (Parsed Character Data), che rappresenta l'unico tipo di dato possibile, ovvero di tipo carattere immune al parsing.

## Definire l'elenco di attributi con ATTLIST

Si può anche definire un elenco di possibili attributi per uno specifico elemento della lista mediante la clausola ATTLIST, con la seguente sintassi:

```
<!ATTLIST element-name
  AttrName1 AttrType1 Value1
  AttrName2 AttrType2 Value2
  ...
>
```

dove

- element-name è il nome dell'elemento;
- $AttrName_n$ è il nome dell'elemento n-esimo;

# ESERCIZI IN LABORATORIO 2

- *AttrType*$_n$ è il tipo dell'elemento n-esimo, che può assumere i seguenti valori:

| | |
|---|---|
| CDATA | : testo; |
| ($en_1$\|$en_2$\|…\|$en_n$) | : valore scelto da una lista di enumerazione; |
| ID | : identificatore univoco a livello di documento che non può iniziare con un numero e che viene generalmente indicato come obbligatorio (#REQUIRED). |

- *Value*$_n$ indica il valore di default dell'attributo n-esimo o modificatore di presenza che può essere:

| | |
|---|---|
| "valore" | : l'attributo ha valore di default pari a valore; |
| #REQUIRED | : l'attributo deve essere presente; |
| #IMPLIED | : l'attributo è opzionale e non è possibile stabilire un valore di default; |
| #FIXED "valore" | : l'attributo deve avere un valore fisso pari a valore. |

Nell'esempio seguente utilizziamo la clausola ATTLIST.

## ESEMPIO

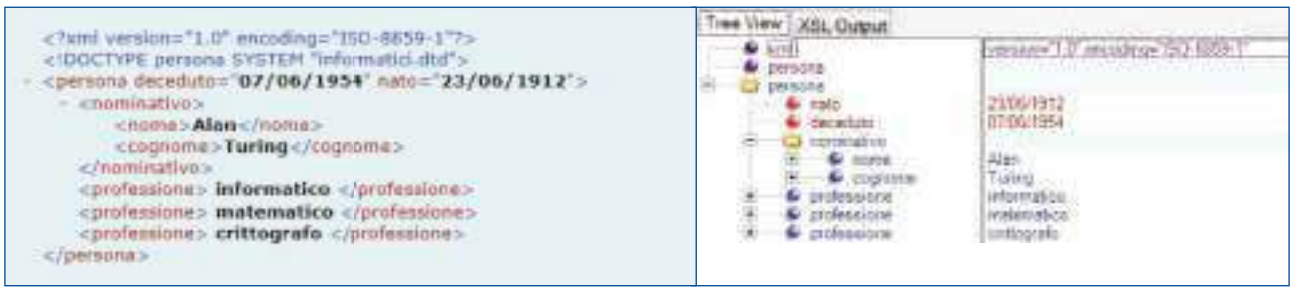
È dato lo schema che descrive un record anagrafico di una persona memorizzato in un file informatici.dtd:

```
<!ELEMENT persona (nominativo, professione*)>
<!ATTLIST persona nato CDATA #REQUIRED deceduto CDATA #IMPLIED>
<!ELEMENT nominativo (nome, cognome)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT professione (#PCDATA)>
```

Vediamo il significato di ogni singola riga del DTD:

- la prima riga dichiara che l'elemento persona deve contenere esattamente un elemento figlio nominativo e zero o più elementi figlio professione, in questo ordine;
- la seconda riga dice che lo stesso elemento persona deve avere un attributo nato e può avere un attributo deceduto (l'ordine non è rilevante);
- la terza riga dichiara che il nominativo deve avere due figli chiamati nome e cognome, riportati in questo ordine;
- le ultime tre righe specificano che gli elementi nome, cognome e professione devono contenere dati di carattere analizzati (PCDATA), ovvero testo non elaborato che potrebbe contenere riferimenti a entità e caratteri ma non contenente tag.

Il seguente esempio di file XML che utilizza tale DTD riporta i dati di Alan Turing.
Visualizziamo l'esempio in XML Notepad dove il file XML viene interpretato e ne viene visualizzato il suo contenuto.



# 2 ESERCIZI IN LABORATORIO

## Conclusioni

Il DTD è un documento opzionale che consiste in un insieme di regole specifiche per definire la struttura di un documento XML, in particolare definiosce:
- quali elementi possono e/o devono essere usati nel documento;
- quali attributi si possono usare per ogni elemento;
- i vincoli sulle relazioni tra elementi.

Il DTD ha grande utilità per:
1 verificare la correttezza e la validità di un documento XML;
2 chi deve interpretare il documento XML, che sia il programmatore oppure un parser automatico;
3 definire i fogli di stile da associare ai documenti XML e/o HTML;
4 creare interfacce dinamiche per documenti XML;
5 l'interscambio di documenti.

Il DTD ha alcuni svantaggi, evidenziamo i più importanti:
1 non è possibile definire il tipo degli elementi o attributi foglia che possono assumere come valore un qualsiasi dato carattere (o anche nessuno);
2 non è possibile limitare il numero di occorrenze di elementi ed è difficile specificare contenuti non ordinati;
3 il meccanismo ID/IDREF è troppo semplice e non permette di limitare l'ambito di unicità degli attributi ID a un frammento ma solo all'intero documento;
4 le chiavi utilizzate per il meccanismo ID/IDREF devono essere solo attributi singoli;
5 dato che la sua definizione precede la raccomandazione del W3C non supporta gli spazi dei nomi;
6 fornisce un supporto limitato per la modularità, il riutilizzo e l'evoluzione degli schemi;
7 non è possibile verificare la correttezza della sua descrizione dato che non è scritto in notazione XML e, quindi, su di esso non si possono utilizzare gli strumenti che permettono di manipolare schemi con strumenti XML, come, per esempio, per verificare che sia ben formato.

Vediamo un ultimo esempio.

### ESEMPIO

**CATALOGAZIONE ARTICOLI**

Modelliamo un documento XML di catalogazione articoli e relativo DTD di validazione in cui:
1 Un Articolo può contenere uno o più Paragrafi.
2 Un Paragrafo ha almeno un testo, zero o più immagini ed eventualmente un codice.
3 Un Articolo è dotato di proprietà quali:
- un codice identificativo univoco (*cod*),
- un titolo (*obbligatorio*),
- una dataPubblicazione (*opzionale*).

4 Un Paragrafo è dotato di proprietà quali:
- un titolo (*opzionale*),
- un'indicazione del tipo (*obbligatorio*) che può essere abstract, bibliografia, note.

**SOLUZIONE**

1 Un Articolo può contenere uno o più Paragrafi:

```
<!ELEMENT Articolo (Paragrafo+)>
```

# ESERCIZI IN LABORATORIO 2

**2** Un Paragrafo ha almeno un testo, zero o più immagini ed eventualmente un codice:

```
<!ELEMENT Paragrafo (codice*,testo+,immagine*)>
<!ELEMENT testo (#PCDATA)>
<!ELEMENT codice (#PCDATA)>
<!ELEMENT immagine EMPTY>
```

**3** Un Articolo è dotato di proprietà quali:

- un codice identificativo univoco (*cod*),
- un titolo (*obbligatorio*),
- una dataPubblicazione (*opzionale*):

```
<!ATTLIST Articolo
cod ID #REQUIRED
titolo CDATA #REQUIRED
dataPubblicazione CDATA #IMPLIED>
```

**4** Un Paragrafo è dotato di proprietà quali:

- un titolo (opzionale).
- un'indicazione del tipo (obbligatorio) che può essere abstract, bibliografia, note:

```
<!ATTLIST Paragrafo
titolo CDATA #REQUIRED
tipo (abstract|bibliografia|note) #IMPLIED>
```

**5** Un'immagine ha come unica proprietà il nome del file:

```
<!ATTLIST immagine file CDATA #REQUIRED>
```

Salviamo il DTD completo nel file articoli.dtd:

```
<!ELEMENT Articolo (Paragrafo+)>
<!ELEMENT Paragrafo (codice*,testo+,immagine*)>
<!ELEMENT testo (#PCDATA)>
<!ELEMENT codice (#PCDATA)>
<!ELEMENT immagine EMPTY>

<!ATTLIST Articolo
cod ID #REQUIRED
titolo CDATA #REQUIRED
dataPubblicazione CDATA #IMPLIED>

<!ATTLIST Paragrafo
titolo CDATA #REQUIRED
tipo (abstract|bibliografia|note) #IMPLIED>

<!ATTLIST immagine file CDATA #REQUIRED>
```

Un documento XML valido secondo il DTD appena definito è il seguente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE articolo SYSTEM "articoli.dtd">
- <Articolo dataPubblicazione="10/10/2020" titolo="Il DTD per XML" cod="a1">
  - <Paragrafo titolo="introduzione" tipo="abstract">
      <codice>par1</codice>
      <testo>"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."</testo>
      <immagine>schemaDTD.gif</immagine>
    </Paragrafo>
  - <Paragrafo titolo="Ti consiglio di utilizzare i DTD">
      <codice>par2</codice>
      <testo>Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur."</testo>
      <immagine>esempio1.gif</immagine>
      <immagine>esempio2.gif</immagine>
    </Paragrafo>
  </Articolo>
```

# 2 ESERCIZI IN LABORATORIO

Come sempre visualizziamo l'esempio anche con XML Notepad.

## METTITI ALLA PROVA

Modella un documento XML di catalogazione Film e relativo DTD di validazione in cui:

1. un Catalogo può contenere zero o più Film;
2. un Film è descritto da un Titolo, almeno un Regista, zero o più Attore ed eventualmente un Genere;
3. un Film è dotato di proprietà come:
   - un codice identificativo univoco (*cod*);
   - un'indicazione di "originalità" del supporto (originale *sì – no*) in cui si assume di default l'acquisto legale;
   - un'indicazione del tipo di formato (*obbligatoria*) che può essere VHS, DVD, DIVX;
   - un voto (*opzionale*).

Puoi confrontare la tua soluzione con quella riportata nei file catalogo.dtd e catalogo.xml.

## Cenni su XML Schema

Un altro linguaggio proposta dal W3C per la descrizione dello schema è XSchema.
A differenza di DTD, XSchema offre un potente sistema di tipi che consente di definire tipi semplici e complessi e anche di ereditare tipi da altri tipi, nello stile di linguaggi di programmazione orientati agli oggetti: la tipizzazione può essere associata sia agli elementi sia agli attributi, aggiungendo significato alle loro interpretazioni.

Ma l'aggiunta di queste descrizioni comporta la complicazione della lettura dei file di formattazione prodotto in questo formato, anche da parte di programmatori esperti, rendendolo, di fatto, quasi inutilizzato.

**AREA DIGITALE**

**Esempio di utilizzo di XSchema**

# ESERCIZI IN LABORATORIO 3

## LA NAVIGAZIONE NEI DOCUMENTI CON XML PATH LANGUAGE

### Cos'è l'XML Path Language

Con la grande diffusione dell'utilizzo di XML sono stati resi disponibili prodotti e strumenti in grado di intercettare e analizzare questi documenti sia in uscita che in ingresso ai sistemi informatici.
Tra questi è stato definito XPath, un linguaggio per analizzare i file XML: XPath, acronimo di XML Path Language, è un linguaggio semplice per identificare e recuperare particolari nodi o set di nodi all'interno di un singolo documento XML.
La struttura di un documento XML è un albero e, quindi, in esso sono individuabili una serie di nodi: il concetto che sta alla base di XPath è quello di avere a disposizione uno strumento che, con una sintassi **NON** XML, ma in forma abbreviata, permetta di effettuare la navigazione simile a quella che viene eseguita su un file system (Unix).
XPath può essere sfruttato in diverse tecnologie XML:

- da solo come un semplice linguaggio di query per XML;
- in XQuery per recuperare elementi XML che possono essere ulteriormente elaborati per risolvere una query;
- in XSLT per recuperare gli elementi a cui vengono applicate le regole del modello per trasformare un documento XML;
- nel W3C XML Schema per individuare chiavi e riferimenti chiave;
- in XPointer per puntare a particolari elementi XML nel documento XML collegato.

Esistono strumenti per verificare la correttezza di una espressione/query XPath, che in inglese prendono il nome di tester o evaluator.

**ESEMPIO**

Alcuni siti Web consentono di caricare e/o copiare e incollare documenti XML e di immettere e valutare una stringa XPath: ne riportiamo alcuni tra I più utilizzati.

http://codebeautify.org/Xpath-Tester
http://www.freeformatter.com/xpath-tester.html
http://xpath.online-toolz.com/tools/xpath-editor.php
http://www.xpathtester.com/xpath

### Gli elementi del linguaggio XPath

XPath consente di creare espressioni, dette espressioni XPath o pattern, che individuano i vari nodi dell'albero di rappresentazione di un documento XML, dove le espressioni XPath risultano essere costituite da token e delimitatori.

#### Elementi del linguaggio: i token

I token sono insiemi di caratteri Unicode che possono assumere due differenti significati:
**1** di stringa se delimitati da virgolette;
**2** di nodi-elemento se privi di virgolette.

In XPath è previsto l'utilizzo del carattere * come carattere jolly (wildcard) che permette di selezionare tutti i nodi del contesto indipendentemente dal loro nome.

# 3 ESERCIZI IN LABORATORIO

ESEMPIO

L'espressione:

```
/ITIS2021/tpsit3/*
```

permette di selezionare tutti i nodi figlio degli elementi tpsit3 presenti sotto l'elemento radice ITIS2021 del documento XML.

Come nella sintassi dei percorsi del file system, il primo carattere **/** identifica il nodo radice, e un percorso aperto da **/** viene considerato assoluto.

## Elementi del linguaggio: i nodi

Il modello di dati XPath rappresenta ogni documento XML come un albero di nodi, classificabili in tre tipologie fondamentali.

1 Nodo Radice (Root Node): il nodo radice è un nodo virtuale che non corrisponde a nessun componente nel documento XML ed è la radice che contiene tutti gli altri nodi. Il nodo radice non ha parent (fratelli) e il suo valore è il valore stringa del nodo dell'elemento documento.
2 Nodo Elemento (Element Node): ogni elemento XML del documento è un nodo elemento ed è etichettato con il tag dell'elemento XML; ha sempre un genitore (o il nodo radice o un nodo elemento) e può avere elementi di tipo elemento, commento, istruzione di elaborazione e testo. Il valore stringa di un nodo elemento è il testo contenuto tra i tag iniziale e finale dell'elemento, esclusi tutti i tag, i commenti e le istruzioni di elaborazione.
3 Nodo Attributo (Attribute Node): un nodo attributo corrisponde agli attributi di ciascun elemento; il suo genitore è il nodo dell'elemento che contiene l'attributo (ma "non sono figli del loro genitore") e il valore stringa di un nodo attributo è il valore dell'attributo.

Sono inoltre definiti altri quattro tipi di nodi:

1 Spazio dei nomi (Namespace Node): un nodo dello spazio dei nomi rappresenta uno spazio dei nomi e, come i nodi degli attributi, i nodi dello spazio dei nomi hanno un genitore ma "non sono figli del loro genitore" e il valore di stringa di un nodo è l›URI dello spazio dei nomi.
2 Istruzioni di elaborazione (Processing Instruction Node): un nodo dell'istruzione di elaborazione rappresenta un'istruzione di elaborazione; ha un genitore e nessun figlio e il valore di stringa di un nodo dell'istruzione di elaborazione è il contenuto dell'istruzione esclusa la destinazione.
3 Testo (Text Node): rappresenta una stringa massima di testo tra tag, commenti e istruzioni di elaborazione; ha un nodo padre ma nessun figlio e il suo valore di stringa è il testo del nodo.
4 Commento (Comment Node): rappresenta un commento, ha un genitore e nessun figlio e il valore di stringa di un nodo di commento è il contenuto del commento.

## Elementi del linguaggio: i tipi di dato e delimitatori

Il modello di dati XPath considera le dichiarazioni XML e DTD e tutti i risultati delle espressioni hanno come risultato uno dei seguenti tipi di dato:

- node-set: un insieme di nodi di un documento XML;
- boolean: un tipo di dato binario che può assumere il valore true o false ed è generalmente prodotto mediante operatori di confronto o logici;
- number: un tipo di dato floating point (compresi i valori speciali, Inf, –Inf e NaN);
- string: una sequenza di caratteri Unicode racchiusi tra apici singoli o doppi.

I token di un'espressione XPath sono separati da specifici caratteri delimitatori:

- / separa un passo/livello di localizzazione da quello a lui inferiore/interno;
- [ ... ] definiscono un predicato;

# ESERCIZI IN LABORATORIO 3

- =, !=, <, >, <=, >= sono utilizzati all'interno di un predicato per verificare il valore booleano del suo contesto;
- :: separa l'asse di un'espressione XPath dal nome di uno specifico nodo;
- //, @, ., .. sono forme abbreviate per indicare rispettivamente "se stesso o discendente", "attributo", "se stesso", "genitore";
- | permette di definire un'espressione XPath composta dall'unione di più espressioni XPath;
- ( ... ) raggruppano sotto-espressioni XPath o delimitano gli argomenti di una funzione XPath;
- +, –, *, div, mod hanno la funzione degli operatori numerici ed eseguono le rispettive operazioni.

## Elementi del linguaggio: location step e path

XPath si compone di espressioni (o query) che vengono di volta in volta valutate in un determinato contesto restituendo uno o più nodi del documento XML.
Un'espressione è costituita da combinazioni di uno o più chunk di testo separati da caratteri speciali '/', e vengono letti da sinistra a destra.
Le espressioni XPath individuano contenuti del documento XML o parte di questi: tale processo avviene in uno o più passi, detti location step, che messi insieme danno vita a un percorso completo di localizzazione, ossia un location path:

```
LocationPath = LocationStep/LocationStep/.../LocationStep
```

A ogni passo di valutazione dei location step (ossia procedendo da sinistra verso destra del location path) vengono esclusi alcuni nodi dalla selezione e vengono considerati i rimanenti.

Un location step identifica un nuovo node-set relativo al contesto node-set corrente.

L'obiettivo di XPath è quello di individuare un nodo (o un insieme di nodi) o una porzione di documento XML utilizzando espressioni formate mediante un insieme di regole principali.

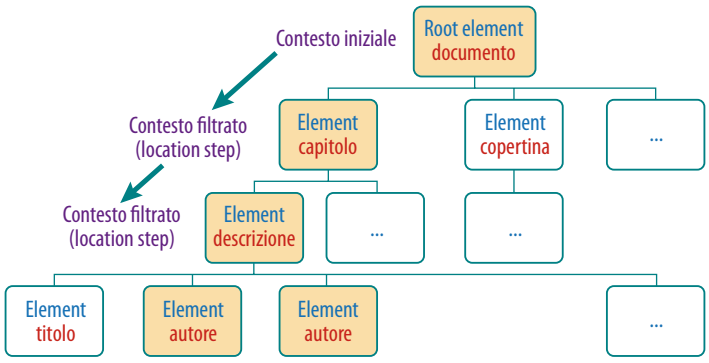
Il location path, che può essere assoluto o relativo, ammette due forme di scrittura per le espressioni XPath:
- la forma abbreviata che ricorda molto la sintassi della shell Unix con simboli come / . ..;
- la forma estesa che ha una sintassi particolare, e divide il suo costrutto in 3 parti:;

```
axis::node_test[pred1][pred2]...[predN]
```

– un axis (asse), che individua la direzione di specifica del location step nell'albero: è opzionale;
– un node_test (test di nodo), che individua il tipo e il nome completo del nodo identificato dal location step;
– zero o più predicati che raffinano ulteriormente l'insieme di nodi selezionati dal location step.

La figura mostra il risultato della seguente location path:

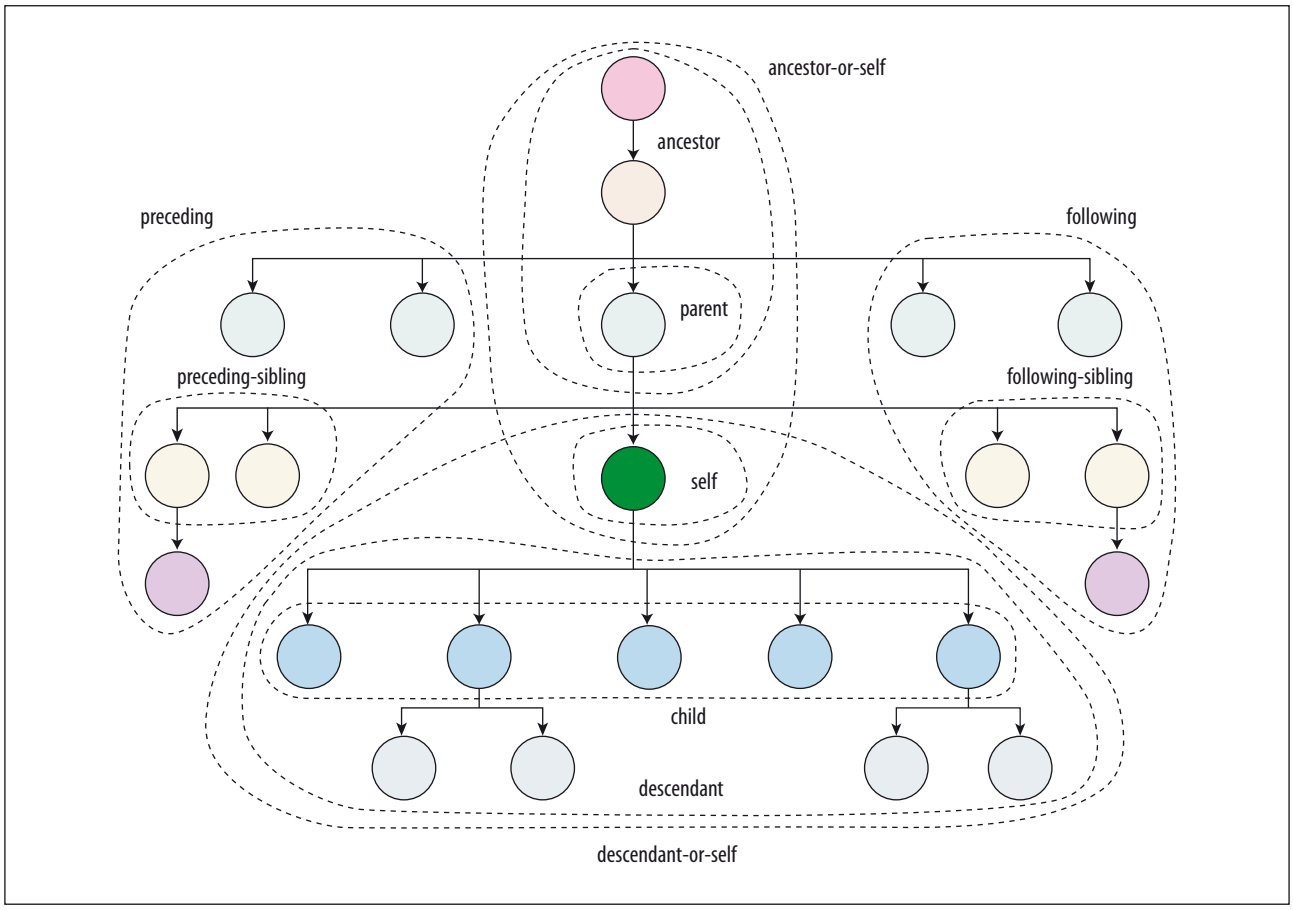
```
/documento/capitolo/descrizione/autore
```



# 3 ESERCIZI IN LABORATORIO

## Elementi del location step: gli assi

Gli assi identificano la direzione rispetto alla struttura del documento in cui andare a cercare l'oggetto da restituire rispetto al nodo contesto NC (o nodo corrente).

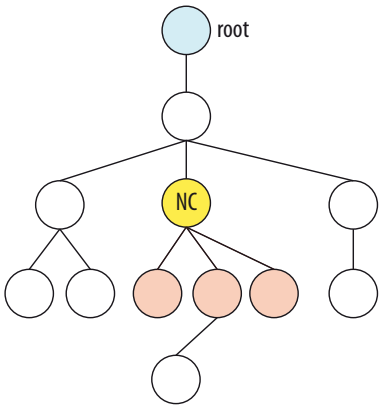
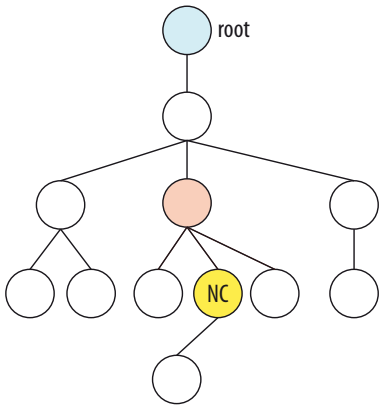


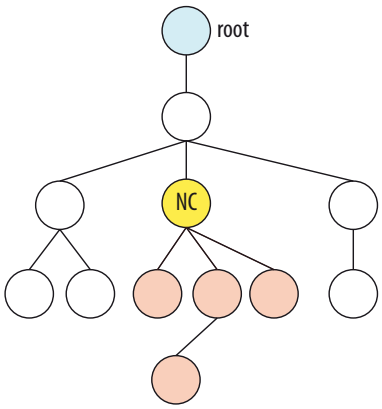
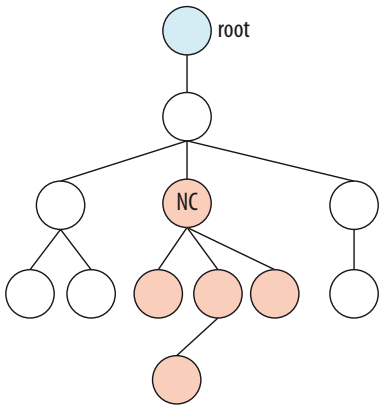

XPath definisce 13 tipi differenti di assi:

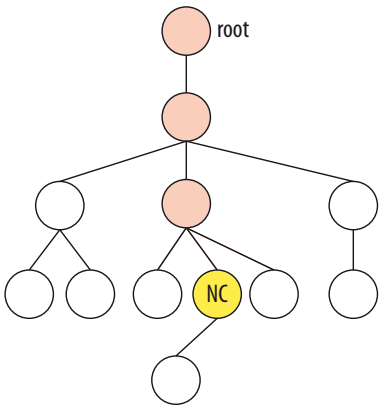
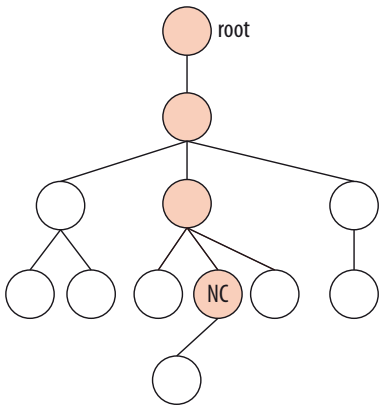
- child: individua il nodo immediatamente discendente dal nodo contesto;
- parent: individua il nodo genitore del nodo contesto;
- descendant: individua tutti i nodi discendenti del nodo contesto a qualsiasi profondità (i figli, i figli dei figli ...);
- ancestor: individua tutti gli antenati del nodo contesto fino all'elemento radice (il padre, il padre del padre ...);
- descendant-or-self: individuano il nodo contesto stesso e tutti i suoi discendenti;
- ancestor-or-self: individuano il nodo contesto stesso e tutti i suoi antenati;
- following: individuano tutti i nodi che all'interno del documento XML seguono il nodo contesto esclusi i discendenti/antenati del nodo contesto;
- preceding: individuano tutti i nodi che all'interno del documento XML precedono il nodo contesto esclusi i discendenti/antenati del nodo contesto;
- following-sibling: individuano tutti i nodi che seguono il nodo contesto e che condividono con questo lo stesso nodo parent;
- preceding-sibling: individuano tutti i nodi che precedono il nodo contesto e che condividono con questo lo stesso nodo parent;
- self: il nodo stesso (Nodo Corrente: NC);
- namespace: il nodo namespace del set dei nodi NC;
- attribute: gli attributi del set dei nodi di NC.

# ESERCIZI IN LABORATORIO 3

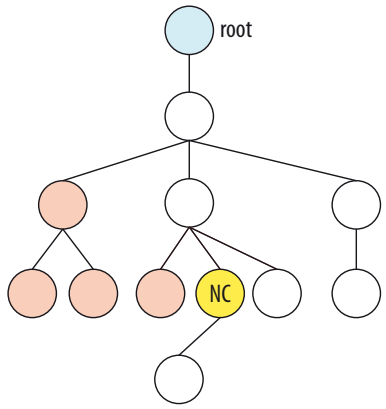
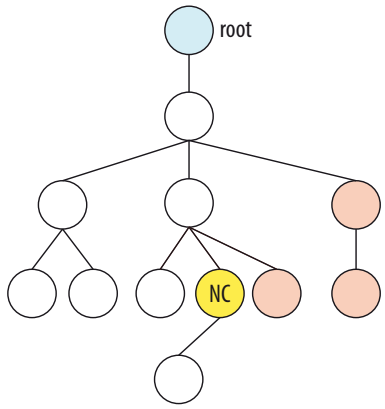
Di seguito illustriamo quelli per i quali la grafica ci viene in aiuto.

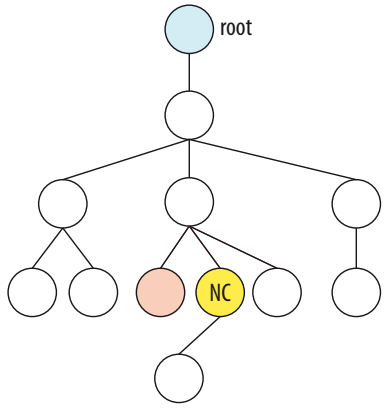
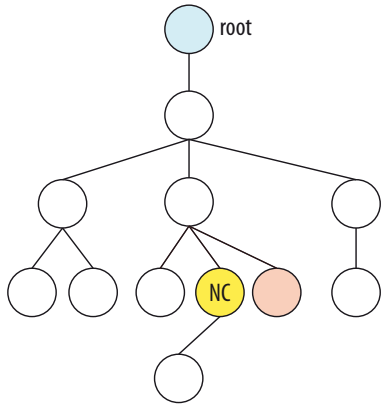
| child: tutti i nodi figli diretti del NC | parent: il genitore che contiene il set dei nodi |
|---|---|
|   |   |

| descendant: tutti i figli a qualunque livello del NC (figli, nipoti, ecc.) | descendant-or-self: il nodo contesto stesso e tutti i figli a qualunque livello del NC (esiste l'abbreviazione //) |
|---|---|
|   |   |

| ancestor: gli antenati di qualunque livello del NC | ancestor-or-self: il nodo contesto stesso e gli antenati di qualunque livello del NC |
|---|---|
|   |   |

# 3 ESERCIZI IN LABORATORIO

| preceding: i nodi a qualunque livello (ma fuori al NC) che precedono il NC | following: i nodi a qualunque livello (ma fuori al NC) che seguono il NC |
|---|---|
|   |   |

| preceding-sibling: i nodi allo stesso livello (fratelli) ma non il NC | following-sibling: i nodi allo stesso livello (fratelli) ma seguenti il NC |
|---|---|
|   |   |

Possiamo indicare l'asse utilizzando la forma estesa sopra indicata oppure la forma breve, preferita per la sua semplicità, anche se non permette di indicare tutti i tipi di assi, ma solo quelli relativi alla radice dell'albero oppure in riferimento relativo rispetto alla posizione corrente.

| ABBREVIATA | ESTESA |
|---|---|
| name | child::name |
| @name | attribute::name |
| / | descendent::name |
| // | descendent-or-self::name |
| . | self::node() |
| .. | parent::node() |

## Elementi del location step: il node_test

Il node_test indica quali nodi includere lungo l'axis specificato prima dei ::, che può essere uno dei seguenti tipi possibili:

- name - ogni elemento o attributo con quel nome lungo l'asse specificato;
- *- ogni elemento lungo l'asse specificato;
- prefix:* - ogni elemento o attributo con il namespace 'prefix' lungo l'asse specificato;
- comment() - ogni commento lungo l'asse specificato;
- text() - ogni nodo di testo lungo l'asse specificato;
- node() - ogni nodo lungo l'asse specificato;

# ESERCIZI IN LABORATORIO 3

- processing-instruction() - ogni istruzione di elaborazione lungo l'asse specificato;
- processing-instruction('target') - ogni istruzione di elaborazione che si riferisce al relativo 'target' lungo l'asse specificato.

## Elementi del location step: i predicati

Il predicato è solitamente racchiuso fra parentesi quadre ed espresso come un'espressione booleana con la seguente struttura:

```
[val1 operatore val2]
```

dove val1 e val2 sono delle espressioni XPath e operatore è uno degli operatori booleani descritti in precedenza.

Inoltre ogni location path che appare all'interno del predicato fa riferimento al contesto stabilito dai location step che precedono il predicato stesso.

### ESEMPIO

Le seguenti espressioni sono differenti tra loro:

```
//descrizione/autore[@tipo="compositore"]
//descrizione[autore/@tipo="compositore"]
```

- Il primo predicato seleziona i nodi autore con attributo tipo uguale a "compositore";
- il secondo predicato seleziona gli elementi descrizione aventi almeno un elemento figlio autore con attributo tipo uguale a "compositore".

Il node_test indica il tipo di nodi che vogliamo selezionare; esistono due possibili approcci:

**1** identificazione dei nodi in base al nome degli elementi;
**2** identificazione dei nodi in base al tipo degli elementi.

L'utilità dei predicati sta nella possibilità di poter filtrare ulteriormente i nodi dal contesto corrente in base all'impostazione (e al verificarsi) di opportune condizioni.

Riportiamo alcuni esempi di XPath.

| XPATH | ELEMENTI SELEZIONATI |
|---|---|
| `paragrafi[@tipo="riassunto"]` | tutti i paragrafi figli del NC che abbiano l'attributo tipo uguale a "riassunto" |
| `paragrafi [@tipo =" riassunto"][5]` | il quinto paragrafo figlio di NC ad avere l'attributo tipo uguale a "riassunto" |
| `paragrafi [5][@ tipo =" riassunto"]` | il quinto paragrafi figlio di NC, ma solo se ha l'attributo tipo uguale a "riassunto" |
| `capitolo[titolo]` | il "capitolo" figlio del NC che contenga uno o più elementi "titolo" |
| `//peso[@netto="11"]` | tutti i nodi discendenti della root chiamati peso aventi attributo netto il cui valore è «11». |

Per selezionare gli elementi con un particolare nome è sufficiente indicare come node_test il nome dell'elemento.

# 3 ESERCIZI IN LABORATORIO

## ESEMPIO

Facendo riferimento al file maestri.xml seguente:

```
<maestri>
<pittori>
<artista nome="Leonardo da Vinci">
<opera anno="1483" genere="olio su tela">Vergine delle Rocce</opera>
<opera anno="1488" genere="olio su tavola">Dama con ermellino</opera>
<opera anno="1494" genere="olio su parete">Ultima cena</opera>
</artista>
<artista nome="Michelangelo">
<opera anno="1497" genere="gruppo sculture">Pieta vaticana</opera>
<opera anno="1506" genere="tempera">Tondo Doni</opera>
<opera anno="1513" genere="statua">Mose</opera>
<opera anno="1537" genere="affresco">Giudizio universale</opera>
</artista>
</pittori>
<musicisti>
<artista nome="Giuseppe Verdi">
<opera anno="1871" atti="4">Aida</opera>
<opera anno="1851" atti="3">Rigoletto</opera>
<opera anno="1857" atti="3" prologo="1">Simon Boccanegra</opera>
</artista>
<artista nome="Giacomo Puccini">
<opera anno="1896" atti="4">La boheme</opera>
<opera anno="1887" atti="3">Tosca</opera>
<opera anno="1926" atti="3">Turandot</opera>
</artista>
</musicisti>
</maestri>
```

scriviamo alcuni location path:

- per selezionare tutti gli elementi figli dell'elemento radice che hanno nome pari a "pittori" (sintassi con solo node_test) basta indicare semplicemente:

```
/maestri/pittori
```

- per selezionare tutti gli elementi a prescindere dal loro nome è possibile utilizzare il carattere wildcard *: selezioniamo tutti i figli dell'elemento radice:

```
/*
```

- per selezionare soltanto i nodi-testo degli elementi che hanno nome "autore" (figli a loro volta dell'elemento pittori della radice maestri):

```
/maestri/pittori/artista/text()
```

Analogamente si può utilizzare node() per selezionare tutti i nodi indipendentemente dal tipo, comment() per selezionare i nodi-commento ecc.

# ESERCIZI IN LABORATORIO 3

## Elementi del linguaggio: le funzioni

XPath non è un linguaggio di programmazione, ma in esso è definito un elenco completo di funzioni per elaborare/manipolare i node-set, ne ricordiamo le più utilizzate:

- funzioni sul node-set: last(), position(), count(), id(), local-name(), namespace-uri(), name();
- funzioni sulle stringhe: string(), concat(), starts-with(), contains(), substring-before(), substring-after(), substring(), string-length(), normalizespace(), translate();
- funzioni booleane e numeriche: boolean(), not(), true(), false(), lang();
- funzioni numeriche: number(), sum(), floor(), ceiling(), round():

Vediamo nel dettaglio le tre funzioni principali che operano sul node-set:

- last() restituisce il position-context dell'ultimo nodo del node-set corrispondente al contesto attualmente selezionato;
- position() restituisce il position-context di un determinato nodo all'interno del node-set corrispondente al contesto attualmente selezionato;
- count(node-set) restituisce il numero di nodi appartenenti a un node-set, agisce come last() ma a differenza di quest'ultimo accetta in ingresso un'espressione XPath che individua il node-set sul quale la funzione deve operare.

Riportiamo i comandi di location step più utilizzati, sempre facendo riferimento al file maestri.xml precedente:

| COMANDO DI LOCATION STEP | NODI/ATTRIBUTI SELEZIONATI |
|---|---|
| `child::maestri` | figli dell'elemento maestri del nodo di contesto |
| `child::*` | elementi figlio del nodo di contesto |
| `child::text()` | figli del nodo di testo del nodo di contesto |
| `child::node()` | figli del nodo di contesto, qualunque sia il loro tipo di nodo |
| `attribute::nome` | attributo nome del nodo di contesto |
| `attribute::*` | attributi del cenno del contesto |
| `ancestor::pittori` | antenati pittori del nodo di contesto |
| `ancestor-or-self::pittori` | antenati pittori del nodo di contesto ed, eventualmente, anche il nodo di contesto |
| `descendant::pittori` | discendenti dell'elemento pittori del nodo di contesto |
| `descendant-or-self::pittori` | discendenti dell'elemento pittori del nodo di contesto ed, eventualmente, anche il nodo di contesto |
| `/` | radice del documento (che è sempre il genitore dell'elemento documento) |
| `/descendant:: pittori` | elementi parametri nello stesso documento del nodo CN |
| `self:: pittori` | nodo di contesto se è un elemento pittori, e altrimenti non seleziona nulla |
| `child::*/child:: pittori` | pittori nipoti del nodo di contesto |
| `child:: pittori [position()=1]` | primo parametro secondario del nodo di contesto |
| `child:: pittori [position()=last()]` | ultimo parametro secondario del nodo di contesto |
| `child:: pittori [position()=last()-1]` | ultimo, tranne uno, figlio secondario del nodo di contesto |
| `child:: pittori [position()>]` | figli secondari del nodo di contesto diversi dal primo figlio secondario del nodo di contesto |

# 3 ESERCIZI IN LABORATORIO

## ESEMPIO

Sempre con riferimento al file maestri.XML, eseguiamo le seguenti espressioni:

**1.** contiamo gli artisti presenti nel documento XML:

```
count(//artista)
Int64='4'
```

**2.** selezioniamo il secondo pittore:

```
//pittori//artista[position()=2]
Element='<artista nome="Giuseppe Verdi">
 <opera anno="1871" atti="4">Aida</opera>
 <opera anno="1851" atti="3">Rigoletto</opera>
 <opera anno="1857" atti="3" prologo="1">Simon Boccanegra</opera>
 </artista>'
```

**3.** selezioniamo l'ultimo artista di ogni nodo:

```
//artista[last()]
Element='<artista nome="Michelangelo"">
 <opera anno="1497" genere="gruppo scultore">Pieta' vaticana</opera>
 ...
 </artista>'
Element='<artista nome="Giacomo Puccini">
 <opera anno="1896" atti="4">La bohème</opera>
 ...
 </artista>'
```

**4.** visualizziamo gli attributi dell'ultima opera del secondo artista tra i pittori:

```
//pittori//artista[2]/opera[last()]/text()
Text='Giudizio universale'
```

**5.** selezioniamo tutte le opere del '400, cioè quelle con il valore-stringa dell'attributo anno che inizi per "14"):

```
//opera[starts-with(@anno,'14') = true()]
Element='<opera anno="1483" genere="olio su tela">Vergine delle Rocce</opera>'
Element='<opera anno="1488" genere="olio su tavola">Dama con ermellino</opera>'
Element='<opera anno="1494" genere="olio su parete">Ultima cena</opera>'
Element='<opera anno="1497" genere="gruppo scultore">Pieta' vaticana</opera>'
```

**6.** visualizziamo il titolo di tutte le opere successive al 1880, convertendo in numero del valore dell'attributo anno:

```
//opera[number(@anno) > 1880]/text()
Text='La bohème'
Text='Tosca'
Text='Turandot'
```

**7.** sommiamo gli anni di creazione di tutte le opere (operazione naturalmente priva di significato!):

```
sum(//opera/@anno)
Double='21806'
```

# ESERCIZI IN LABORATORIO 3

La verifica delle espressioni XPath può essere effettuata online utilizzando un valutatore come, per esempio, https://www.freeformatter.com/xpath-tester.html:

| Si inserisce (o si carica) il codice XML | si inserisce l'espressione | si visualizza il risultato |
|---|---|---|
| | | |

Un utilizzo particolare dei predicati è quello di verificare o meno l'esistenza di un determinato nodo figlio del contesto attuale.

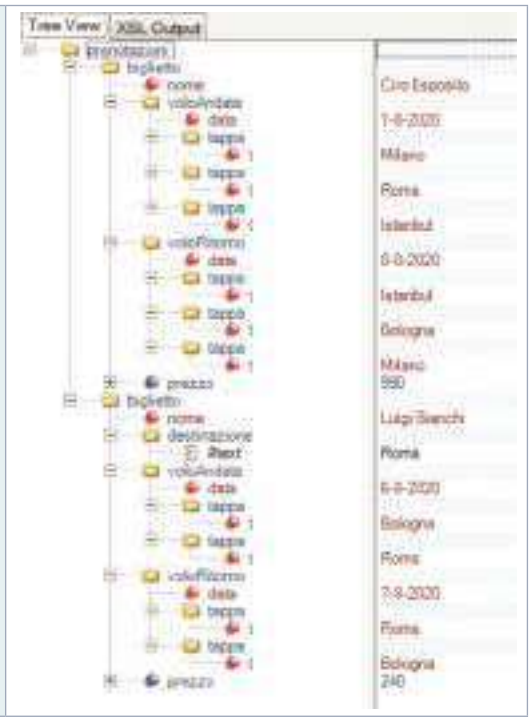
## ESEMPIO

**AREA DIGITALE**

**XML e vincoli di Integrità in XML schema**

Dato il seguente documento voli.xml:

```
<prenotazioni>
<biglietto nome="Ciro Esposito">
<voloAndata data="1-8-2020">
<tappa t="Milano"/><tappa t="Roma"/><tappa t="Istanbul"/>
</voloAndata>
<voloRitorno data="8-8-2020">
<tappa t=" Istanbul"/><tappa t="Bologna"/><tappa t=" Milano "/>
</voloRitorno>
<prezzo>700</prezzo>
</biglietto>
<biglietto nome="Luigi Bianchi">
<destinazione>Roma</destinazione>
<voloAndata data="6-8-2020">
<tappa t="Bologna"/><tappa t="Roma"/>
</voloAndata>
<voloRitorno data="7-8-2020">
<tappa t="Roma"/><tappa t="Bologna"/>
</voloRitorno>
<prezzo>240</prezzo>
</biglietto>
<!-- ... -->
</prenotazioni>
```



# 3 ESERCIZI IN LABORATORIO

Scriviamo le espressioni XPath per individuare:

1. Tutti i biglietti di Ciro Esposito

```
prenotazioni/biglietto[@nome="Ciro Esposito"]
```

2. Tutti i biglietti che costano più di 500 euro

```
/prenotazioni/biglietto[prezzo > 500]
```

3. Tutte le tappe di tutti i voli

```
//tappa
```

4. Tutti i biglietti del 6 agosto 2020

```
//biglietto[*/@data="6-8-2020"]
```

5. Tutti i biglietti che passano per Roma

```
//biglietto[//tappa/@t="Roma"]
```

6. Le date di tutti i biglietti che arrivano a Bologna

```
/prenotazioni/biglietto/*[./tappa[last()]/@t="Bologna"]/@data
```

**AREA DIGITALE**

**Esercizi aggiuntivi**

## METTITI ALLA PROVA

Utilizzando il file personaggi.xml dell'esempio disponibile tra le risorse digitali e utilizzando un software per la valutazione di espressioni XPath ti viene richiesto di determinare l'espressione per selezionare:

**1** l'elemento radice;
**2** tutti gli attori e i diretti elementi figli 'attore';
**3** tutti gli elementi "cantanti" indipendentemente dalle loro posizioni nel documento;
**4** gli attributi 'id' degli elementi 'cantante' indipendentemente dalle loro posizioni nel documento;
**5** il valore testuale del primo elemento "attore";
**6** l'ultimo elemento "attore";
**7** il primo e il secondo elemento "attore" usando la loro posizione;
**8** tutti gli elementi 'attore' che hanno un attributo 'id';
**9** l'elemento 'attore' con il valore dell'attributo 'id' di '3';
**10** tutti i nodi 'attore' con il valore dell'attributo 'id' inferiore o uguale a '3';
**11** tutti i figli del nodo "cantanti";
**12** tutti gli elementi nel documento;
**13** tutti gli elementi "chitarrista" e gli elementi "cantante";
**14** il nome del primo elemento nel documento;
**15** il valore numerico dell'attributo "id" del primo elemento "attore";
**16** il valore di rappresentazione della stringa dell'attributo 'id' del primo elemento 'attore';
**17** la lunghezza del valore testuale del primo elemento 'attore';
**18** il nome locale del primo elemento "cantante", ovvero senza lo spazio dei nomi;
**19** il numero di elementi "chitarrista";
**20** la somma degli attributi "id" degli elementi " chitarrista ".

# ESERCIZI IN LABORATORIO 4

## IL PARSING XML CON JAVA: LE SPECIFICHE JAXP

### Elaborazione di documenti XML

Un'operazione estremamente importante sui documenti XML è il parsing, che oltre a essere molto diffusa ha una notevole utilità.

> In generale, eseguire il parsing di un documento significa costruire l'albero sintattico a esso associato, verificandone, eventualmente, la rispondenza con lo schema del documento stesso.

**CSV**
Il formato CSV (Comma-Separated Values) è un formato di file utilizzato per l'importazione ed esportazione di una tabella di dati in fogli elettronici o database: viene generato un semplice file di testo con separatori di campo e di record.

Il risultato dell'elaborazione è un albero che costituisce la base di ulteriori elaborazioni, come, per esempio, la trasformazione dello stesso in un altro formato (per esempio in **CSV** direttamente importabile all'interno dei database), oppure la ricerca veloce di particolari informazioni direttamente sul documento XML.
La realizzazione dell'albero richiede la valutazione del markup XML originale: il parser deve elaborare tutti i documenti nella loro interezza per leggere ed estrarre i dati da ogni parte del testo corrispondente; la procedura più semplice è quella di leggere il documento XML, farne l'analisi sintattica e utilizzarlo all'interno di un programma.
L'analisi del documento avviene scandendo la sequenza dei simboli terminali, aggregandoli in entità più complesse sulla base dei tag secondo le semplici regole dell'XML.

Poiché le regole sintattiche dell'XML sono sempre le stesse, il codice che effettua l'analisi sintattica (parsing) può essere scritto una volta per tutte.

I principali controlli che il parser può eseguire sui documenti XML sono:
- verificare se il documento è ben formato;
- controllare che il documento sia valido in base a una definizione DTD oppure XML Schema.

> Queste elaborazioni possono richiedere però molteplici risorse computazionali ed è quindi necessario utilizzare strumenti che ci permettano di scegliere quando attivarle o non eseguirle.

### Approcci dei parser Java: DOM e SAX

**Callback**
Con callback si intende una funzione (o metodo) che viene passato come parametro a un'altra funzione (o metodo).

In Java esistono sostanzialmente i seguenti due approcci.

1 Interfacce basate su eventi: Interfacce SAX (Simple API for XML).
Si tratta di un parser che, *durante la lettura* del file XML, ne individua i vari elementi e per ognuno di essi esegue le opportune azioni sfruttano un modello a **callback**; de facto è ora uno standard per il parsing di documenti XML.
Questo approccio si presta bene sia al trattamento di documenti molto semplici sia per le elaborazioni di documenti molto grandi che, per la loro lunghezza, non potrebbero essere contenuti nella memoria centrale.

2 Interfacce Object Model: W3C DOM Recommendation (Document Object Module).
Si tratta di un parser che legge l'*intero documento* XML e lo trasforma in un albero che risiede nella memoria centrale; è quindi adatto nel caso di documenti che richiedano un'elaborazione complessa ma di dimensioni modeste.

> Possiamo quindi affermare che SAX utilizza al meglio le risorse, ma non consente di navigare il documento al contrario né di modificare o creare documenti XML, operazioni queste invece facilmente realizzabili con DOM poiché tutto l'albero del documento viene caricato in memoria e si può lavorare su di esso come meglio preferiamo.

# 4 ESERCIZI IN LABORATORIO

## Interfacce a eventi (SAX)

Nell'approccio a eventi l'applicazione implementa un insieme di metodi di callback che vengono invocati dal parser mentre elabora il documento al verificarsi di specifici eventi, cioè quando durante la scansione del documento vengono individuati particolari elementi, come il tag-iniziale e finale ecc.; al metodo callback vengono passati un insieme di parametri necessari a specificare e fornire ulteriori informazioni sull'evento, come per esempio

- il nome dell'elemento;
- il nome e valore assunto dagli attributi;
- il valore delle stringhe contenute ecc.

### ESEMPIO

In SAX il metodo che individua il tag iniziale è startElement() e ha la seguente scrittura:

```
public void startElement (String namespace, String rawName, String dato, Attributes attributi)
```

dove i parametri sono:

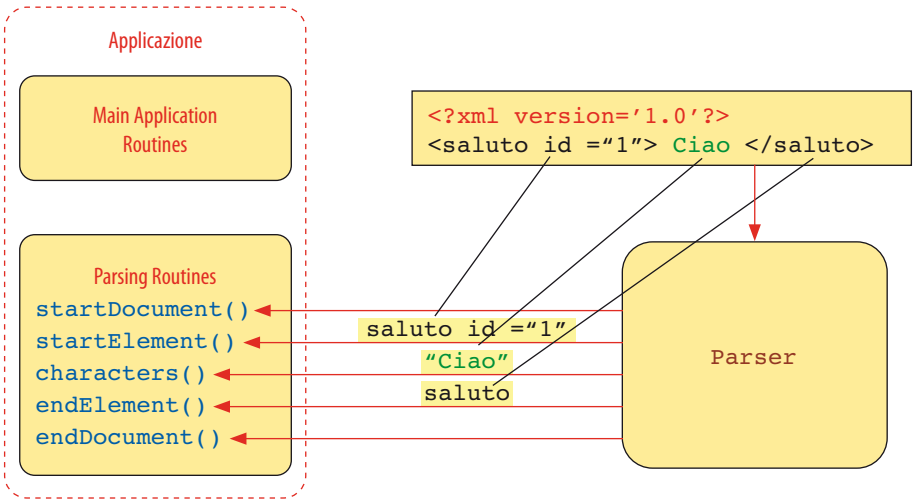
- namespace : URI del namespace associato all'elemento;
- rawName : nome nel formato XML 1.0;
- dato : nome dell'elemento;
- attributi : lista degli attributi.

Se si vuole "intercettare" ogni occorrenza del tag <cognome> nel corpo del metodo inseriamo il seguente codice:

```
public void startElement (String namespace, String rawName, String dato, Attributes attributi) {
    if (dato.equals("cognome"))
       System.out.println("Individuato l'inizio del tag <cognome>");
}
```

L'interfaccia a eventi è una modalità semplice ed efficiente, non mantiene in memoria una rappresentazione del documento ed è usato per grossi file XML o per accessi veloci: è il modello adottato da SAX.

Lo schema di parsing a eventi è riportato nella figura che segue applicato a un semplice esempio: possiamo osservare i momenti nei quali vengono richiamate le singole routine.



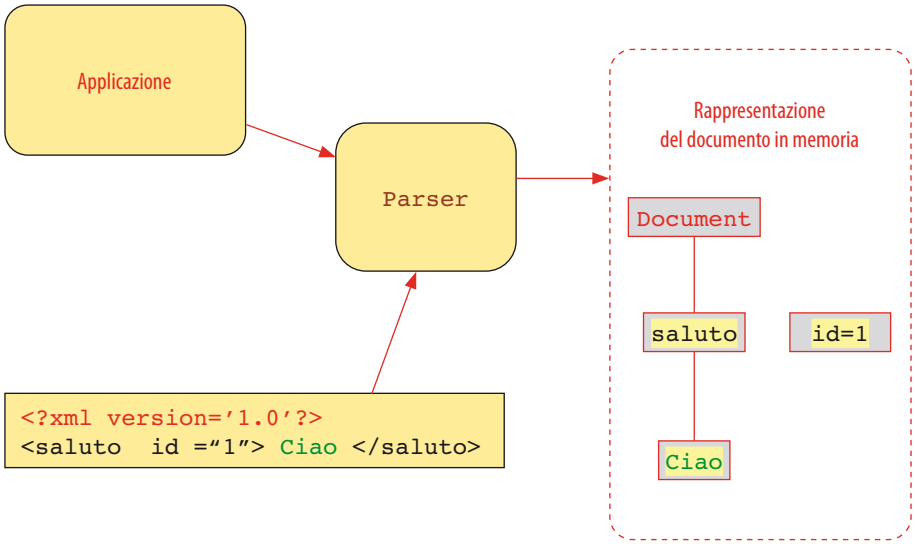
# ESERCIZI IN LABORATORIO 4

## Interfacce Object Model (DOM)

L'applicazione interagisce con una rappresentazione object-oriented del documento che viene interamente "ricostruito in memoria" in una struttura ad albero chiamata parse tree, nel quale sono presenti oggetti per ciascun componente del file XML, cioè document, element, attribute, text ecc.

La rappresentazione astratta realizzata mediante un albero favorisce la sua elaborazione anche grazie alle interfacce disponibili nel modello a oggetti: è facile visitare l'albero, accedere ai figli, ai fratelli ecc. di un elemento e/o aggiungere, rimuovere dinamicamente nodi; l'unico limite che possiamo indicare per questo metodo è quello legato alla *dimensione del file*, che deve essere completamente contenuto nella memoria dinamica.
Il Document Object Model (DOM) è anche il modello adottato dal W3C (WWW Consortium) per le specifiche del Web.

Lo schema di parsing object model dell'esempio descritto in precedenza è riportato nella figura seguente:




In conclusione possiamo affermare che i parser XML rendono disponibile alle applicazioni la struttura e il contenuto dei documenti XML e si interfacciano mediante due tipologie di API:

1 Event-based API (adottato da SAX): le API di questa tipologia notificano alle applicazioni eventi generati nel parsing dei documenti usando poche risorse (parsing routine); non sono sempre comodissime da usare e non richiedono di avere tutto il documento in memoria;

2 Object-model based (adottato da DOM): questa tipologia di API realizza un parse tree che rappresenta il documento XML richiedendo maggiori risorse in termini di memoria dinamica in quanto il documento deve essere completamente presente in essa.

I parser più diffusi supportano sia SAX sia DOM: spesso i parser DOM sono sviluppati su parser SAX.

Nel seguito della trattazione vedremo un esempio di parsing effettuato con SAX e uno con DOM, nei quali utilizzeremo come file da analizzare studenti.xml che contiene i seguenti dati:

# 4 ESERCIZI IN LABORATORIO

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <scuola>
  - <studente ID="01">
    - <nominativo>
        <nome>Mario</nome>
        <cognome>Rossi</cognome>
      </nominativo>
      <indirizzo>via Verdi 15</indirizzo>
      <citta>Genova</citta>
      <telefono>339.1223344</telefono>
    </studente>
  - <studente ID="02">
    - <nominativo>
        <nome>Paolo</nome>
        <cognome>Verdi</cognome>
      </nominativo>
      <indirizzo>via Roma, 22</indirizzo>
      <citta>Genova</citta>
      <telefono>339.1223344</telefono>
    </studente>
  - <studente ID="03">
    - <nominativo>
        <nome>Ciro</nome>
        <cognome>Meo</cognome>
      </nominativo>
      <indirizzo>via Milano 1</indirizzo>
      <citta>Napoli</citta>
    </studente>
  </scuola>
```

## Parser in Java: JAXP

JAXP (Java API for XML Processing) è un framework che ci consente di istanziare e utilizzare parser XML (sia SAX che DOM) nelle applicazioni Java: JAXP 1.1 è inclusa nella JDK a partire dalla versione JDK 1.4.
Fornisce vari package:

- org.xml.sax: interfacce SAX;
- org.w3c.dom: interfacce DOM;
- javax.xml.parsers: inizializzazione e uso dei parser;

e quest'ultimo package mette a disposizione due classi astratte factory, una per SAX e una per DOM:

- SAXParserFactory;
- DocumentBuilderFactory.

In esse è presente il metodo statico newInstance() per creare un'istanza:

```
SAXParserFactory spf =SAXParserFactory.newInstance();
DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
```

Creata una factory possiamo invocarla per creare un parser con interfaccia SAX o DOM:

```
SAXParser saxParser = spf.newSAXParser();
DocumentBuilder builder = dbf.newDocumentBuilder();
```

La documentazione completa di JAXP è consultabile all'indirizzo:
http://download.oracle.com/javase/6/docs/technotes/guides/xml/jaxp/index.html.

# ESERCIZI IN LABORATORIO 4

Vediamo singolarmente i due metodi realizzando un'applicazione con SAX e una con DOM.

## Interfacce SAX

Un parser SAX è orientato al flusso di dati, cioè legge il documento XML generando degli eventi quando incontra alcuni elementi strutturati; le applicazioni interagiscono con parser conformi a SAX utilizzando interfacce stabilite, classificabili in tre categorie:

1 Interfacce Parser-to-Application (callback): consentono di definire funzioni di callback e di associarle agli eventi generati dal parser mentre questi processa un documento XML;
2 Interfacce Application-to-Parser: consentono di gestire il parser;
3 Interfacce Ausiliarie: facilitano la manipolazione delle informazioni fornite dal parser.

Un parser SAX non mantiene quindi tutto l'albero del documento XML e gli eventi possono essere catturati e gestiti da rispettivi metodi Java: riportiamo quelli presenti nella versione 10:

**AREA DIGITALE**
**Caratteristiche delle tre tipologie di interfacce**

### Attivazione di un parser SAX

La struttura del main() di un'applicazione può essere schematizzata in quattro passi: si crea prima una factory, quindi il parser e infine il reader, successivamente si procede con la lettura e l'analisi del documento:

```
// a) creazione della Factory col metodo static newInstance
    SAXParserFactory spf = SAXParserFactory.newInstance()
// b) creazione del SAXparser
   SAXParser saxParser = spf.newSAXParser();
// c) creazione del Reader
   XMLReader xmlReader = saxParser.getXMLReader();
// d) lettura e analisi del documento
   try
   {
   . . .  // lettura e parse del file
   }
   catch (Exception e)
   {
    System.err.println(e.getMessage());
    System.exit(1);
   };
```

Creiamo una nuova classe StudentiSAXParser che discende da DefaultHandler nella quale ridefiniremo i metodi di callback utili nella nostra applicazione: come prime istruzioni importiamo le classi che ci servono per utilizzare le interfacce SAX.

```
3 import org.xml.sax.XMLReader;
5 import org.xml.sax.Attributes;
4 import org.xml.sax.ContentHandler;
5 import org.xml.sax.helpers.DefaultHandler;   // implementazione di default di ContentHandler
6 import javax.xml.parsers.*;                   // classi JAXP usate per accedere al parser SAX
```

# 4 ESERCIZI IN LABORATORIO

Il codice completo del main() è riportato di seguito, dove analizziamo il file studenti.xml presente nella directory dati.

```
65  public static void main (String args[]) throws Exception{
66    String xmlFilename = "dati/studenti.xml";
67    // usiamo SAXParserFactory per istanziare un XMLReader
68    SAXParserFactory spf = SAXParserFactory.newInstance();
69    SAXParser saxParser = spf.newSAXParser();
70    XMLReader xmlReader = saxParser.getXMLReader();
71    try{
72      // creiamo un oggetto sovrascrivendo la ContentHandler
73      ContentHandler handler = new StudentiSAXParser();
74      xmlReader.setContentHandler(handler);
75      System.out.println("\nParser del file "+xmlFilename);
76      xmlReader.parse(xmlFilename);
77    }
78    catch (Exception e){
79      System.err.println(e.getMessage());
80      System.exit(1);
81    }
82  }
```

Supponiamo di voler stampare a video nome, cognome e identificativo di ogni persona nel seguente formato:

```
Parser del file dati/studenti.xml
Mario Rossi (01)
Paolo Verdi (02)
Ciro Meo (03)
```

A tale scopo definiamo nella classe due attributi booleani, che serviranno da flag per stabilire quale tag XML stiamo analizzando, e tre stringhe atte a contenere i dati man mano che li leggiamo, prima di visualizzarli sullo schermo.

```
8  public class StudentiSAXParser extends DefaultHandler{
9    // flag per ricordare il campo che stiamo analizzando
10   private boolean ilNome = false;
11   private boolean ilCognome = false;
12   // attributi per memorizzare i valori da visualizzare
13   private String sNome, sCognome, sID;
```

Ora sovrascriviamo i metodi che ci permettono di sfruttare l'event-based secondo la seguente strategia:

- l'inizio del nuovo record viene individuato alla lettura del tag iniziale *studente* e in questa situazione memorizziamo il suo codice nella variabile sID;
- quando individuiamo l'inizio dei due campi desiderati, e cioè *cognome* oppure *nome*, settiamo a vero il corrispondete flag (ilCognome = true oppure ilNome = true) così da "sapere" durante la lettura dei caratteri successivi in che attributo memorizzare il testo;
- quando individuiamo il termine del campo/elemento *cognome* oppure *nome* settiamo a falso il corrispondete flag così da "terminare" la sua lettura;
- se individuiamo il tag "indirizzo" sappiamo di aver letto i dati desiderati, cioè l'ID, il *cognome* e il *nome* del record e, quindi, li possiamo stampare e passare all'analisi del record successivo.

# ESERCIZI IN LABORATORIO 4

I tre metodi di callback da implementare sono:

**A** startElement(), che ci serve per "registrare" l'inizio del riconoscimento degli elementi:

```
25  public void startElement (String namespace, String rawName, String dato, Attributes attrib){
26    if (dato.equals("nome")){
27      ilNome = true;
28    }
29    if (dato.equals("cognome")){
30      ilCognome = true;
31    }
32    if (dato.equals("studente")){
33      sID = attrib.getValue("ID");
34    }
35  }
```

**B** characters(), necessario per estrarre il contenuto testuale: lo registriamo in modo opportuno a seconda del valore di flag inizializzati nel metodo precedente:

```
42  public void characters (char stringaParserizzata[], int inizio, int lungo) {
43    if (ilNome)
44      sNome = new String(stringaParserizzata, inizio, lungo);
45    if (ilCognome)
46      sCognome = new String(stringaParserizzata, inizio, lungo);
47  }
```

**C** endElement(), che ci serve per "registrare" la terminazione degli elementi:

```
54  public void endElement(String namespaceURI, String rawName, String dato){
55    if (dato.equals("indirizzo"))
56      System.out.println(sNome + " " + sCognome + " (" + sID + ")" );
57    // aggiorna i flag di contesto del dato elaborato
58    if (dato.equals("nome"))
59      ilNome = false;
60    if (dato.equals("cognome"))
61      ilCognome = false;
62  }
```

Il codice completo del programma è presente nel file StudentiSAXParser.java.

## METTITI ALLA PROVA

Dato il seguente segmento di file XML:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/xhtml1/strict">
   <xsl:template match="/">
    <html>
     <xsl:value-of select="//total"/>
   </html>
  </xsl:template>
</xsl:stylesheet>
```

che cosa vengono passati come parametri attuali a un'invocazione del metodo di SAX endElement() per l'elemento xsl:template?

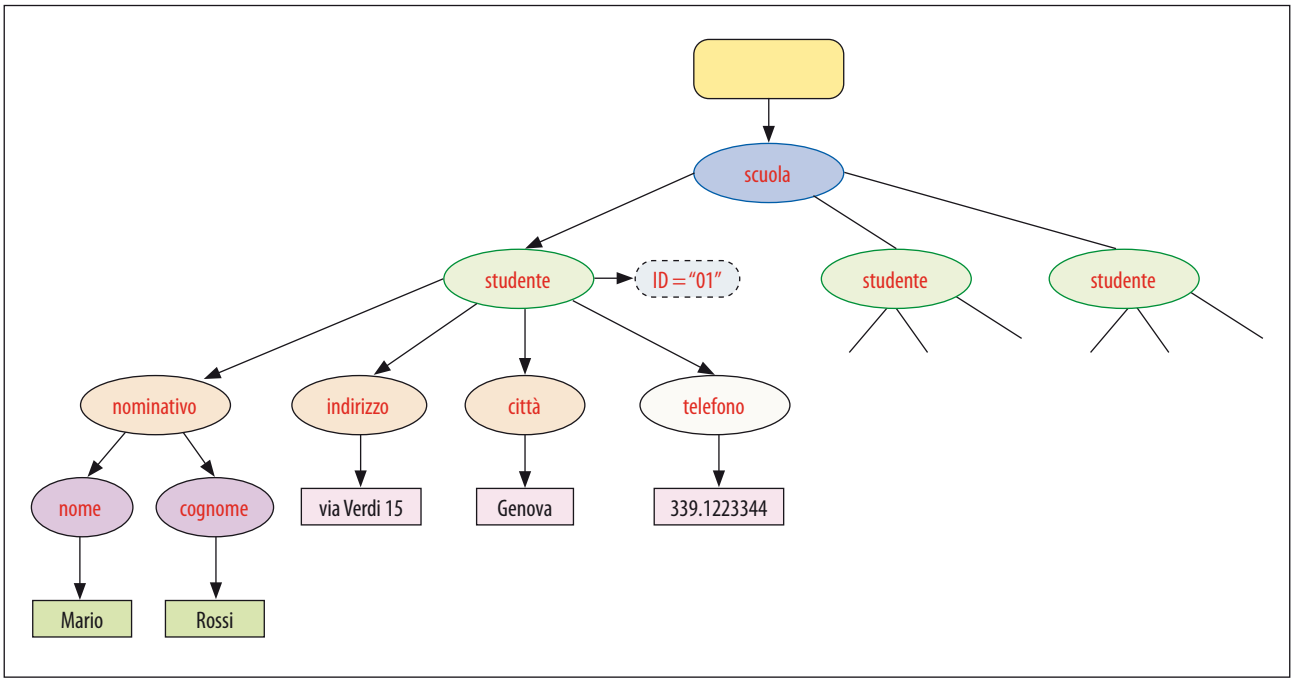
# 4 ESERCIZI IN LABORATORIO

## Attivazione di un parser DOM

A differenza d SAX, con DOM tutto l'albero del documento che viene parserizzato è caricato in memoria e, quindi, lo possiamo analizzare a piacere, come un "normale albero", e quindi visitarlo in ampiezza o profondità, o semplicemente ramo per ramo.

Sono evidenti i vantaggi di DOM rispetto a SAX, ma dobbiamo sempre avere presente i limiti imposti dalla dimensione del documento da analizzare.

Un ulteriore vantaggio di DOM rispetto a SAX è la possibilità di modificare e, quindi, di salvare il documento con nuovi dati e/o valori oppure di creare nuovi documenti.
L'albero creato da DOM è una struttura dinamica a oggetti rappresentato da una classe che implementa l'interfaccia org.w3c.dom.Node.
Ogni elemento del documento XML diventa un nodo dell'albero e l'interfaccia mette a disposizione i metodi opportuni per esplorare o modificare l'albero e per reperire e/o modificare le proprietà di ogni singolo nodo.




## Prerequisiti: le interfacce Document, Node e NodeList

Come precedentemente detto, DOM è basata sui concetti tradizionali della programmazione a oggetti: noi utilizziamo tre interfacce messe a disposizioni di Java.

**A** L'interfaccia NodeList, che è una lista di oggetti Node, offre i seguenti metodi:

`int getLength()`: restituisce il numero di nodi della lista;
`NodeList getChildNodes()`: restituisce un riferimento a una NodeList dei figli del nodo;
`Node item(int index)`: riferimento al nodo della lista alla posizione specificata.

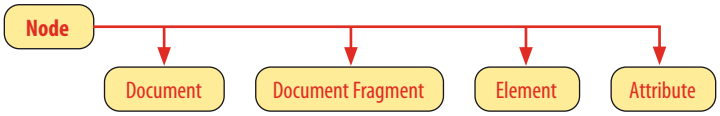
# ESERCIZI IN LABORATORIO 4

**B** L'interfaccia Node che ha un ricco set di funzionalità:

`NamedNodeMap getAttributes()`: restituisce un riferimento a una NamedNodeMap degli attributi del nodo se è un elemento o null altrimenti;
`Node getFirstChild()`: restituisce un riferimento al primo figlio del nodo;
`Node getLastChild()`: restituisce un riferimento all'ultimo figlio del nodo;
`Node getNextSibling()`: restituisce un riferimento al nodo successivo di pari livello;
`String getNodeName()`: restituisce il nome del nodo;
`String getNodeValue()`: valore del nodo per i nodi di testo, CDATA, commento e attributo;
`Document getOwnerDocument()`: riferimento all'oggetto Document che contiene il nodo;
`Node getParentNode()`: restituisce un riferimento al nodo di livello superiore;
`Node getPreviousSibling()`: restituisce un riferimento al nodo precedente di pari livello;
`boolean hasAttributes()`: restituisce true se il nodo ha attributi;
`boolean hasChildNodes()`: restituisce true se il nodo ha nodi figli;
`short getNodeType()`: restituisce il tipo del nodo.

**C** L'interfaccia Document (sotto-interfaccia di Node) che ha i seguenti metodi:

`Element getDocumentElement()`: restituisce un riferimento all'elemento radice del documento;
`Element getElementById(String elemento)`: restituisce un riferimento all'elemento con ID specificato;
`Element getElementsByTagName(String tagname)`: restituisce una NodeList degli elementi con il tag specificato
`DocumentType getDocType()`: restituisce un riferimento alla DTD associata al documento.




**AREA DIGITALE**
Le costanti dell'interfaccia Node

## Elaborazione con DOM

Realizziamo un'applicazione che analizza il documento precedente studenti.xml, ma per meglio comprendere le potenzialità di DOM, ne visualizza tutto il contenuto.
Creiamo una nuova classe StudentiDOMParser e come prime istruzioni importiamo i package fondamentali:

```
1 // Parser del documento XML con DOM
2 // autore: paolo 23/09/2019
3 import java.io.*;
4 import javax.xml.parsers.*;
5 import org.w3c.dom.*;
6 import org.xml.sax.*;
```

Nel main(), dopo aver definito il nome del file da analizzare, creiamo un oggetto che conterrà l'albero, che è proprio un oggetto della classe che stiamo definendo.
Quindi è necessario istanziare un oggetto DocumentBuilderFactory e su di esso un oggetto DocumentBuilder, che ci permetterà di effettuare il parsing di un documento XML.
Creiamo quindi l'oggetto Document che, tramite il metodo builder.parse(), crea l'albero DOM che verrà analizzato grazie al metodo stampaNodo() descritto in seguito.

# 4 ESERCIZI IN LABORATORIO

```
public class StudentiDOMParser{
  public static void main(String[] args){
    String nomeFile = "dati/studenti.xml";
    StudentiDOMParser dp = new StudentiDOMParser();
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    try {
      DocumentBuilder builder = dbf.newDocumentBuilder();
      File xmlFile = new File(nomeFile);
      Document documento = builder.parse(xmlFile);
      dp.stampaNodo(documento);
    }catch (Exception e)
    {
      System.err.println(e.getMessage());
      System.exit(1);
    }
  }
```

È anche possibile impostare le caratteristiche del parser con una serie di metodi setter che, tra le altre cose, consentono di ignorare i commenti, di impostare la non validazione del documento XML, come per esempio:

```
dbf.setValidating(false);                          // stiamo validando il documento
dbf.setFeature(schemaFeature,false);               // stiamo validando tramite XML Schema
dbf.setNamespaceAware(true);
dbf.setFeature(ignorableWhitespace,false);         // elementi che devono essere ignorati
```

Abbiamo effettuato il parsing del documento utilizzando il metodo parse() che restituisce un oggetto di tipo Document, che rappresenta l'albero del documento XML: ora possiamo interrogare l'albero con i metodi messi a disposizione dalle varie classi.

## ESEMPIO

In questo primo esempio scriviamo un segmento di codice che recupera la lista dei nodi da un documento DOM: dopo aver ottenuto un oggetto di tipo Document, possiamo analizzarlo navigando nel suo albero con il seguente codice:

```
NodeList lista = documento.getElementsByTagName("*");
for (int conta = 0; conta < lista.getLength();conta ++) {
  Element elemento = (Element)lista.item(conta);
}
```

Chiamiamo il metodo getElementsByTagName() sull'oggetto documento di tipo Document in modo da creare in un oggetto di tipo NodeList i soli nodi principali, e non i loro sottonodi.

In questo secondo esempio riportiamo invece un metodo ricorsivo che accede a tutti i nodi, anche a quelli innestati in profondità:

```
public void visitaNodo(Node nodo, int livello) {
  NodeList lista = nodo.getChildNodes();
  for (int conta = 0; conta < lista.getLength();conta ++) {
    Node nodoFiglio = lista.item(conta);
    visitaNodo (nodoFiglio, livello + conta);
  }
}
```

# ESERCIZI IN LABORATORIO 4

Questo metodo va invocato con la seguente istruzione:

```
trovaNodo (doc,0);
```

---

Scriviamo ora un metodo ricorsivo stampaNodo() simile a quello appena descritto per visitare tutti i nodi e, contemporaneamente, stampare a video il loro contenuto.

```
// Stampa le info sui nodi, in modo ricorsivo passando il nodo corrente
public void stampaNodo(Node nodoCorrente){
  String elencoAttributi;
  short sNodeType = nodoCorrente.getNodeType();
  //se il nodo e di tipo Element ricavo le informazioni e le stampo
  if (sNodeType == Node.ELEMENT_NODE){
    // analizzo il nodo: estraggo il nome del nodo (campo)
    String nomeNodo = nodoCorrente.getNodeName();
    if (nomeNodo=="studente")                          // nuovo record studente
      System.out.print("\n");
    System.out.print("Elemento: " + nomeNodo);
    String valoreNodo = analizza(nodoCorrente);        // estraggo il valore
    if (!valoreNodo.trim().equalsIgnoreCase("")){
      System.out.print(" = " + valoreNodo);
    }
    // estraggo gli attributi
    NamedNodeMap attributi = nodoCorrente.getAttributes();
    elencoAttributi=formattaAttributi(attributi);
    if (elencoAttributi!="assenti")
      System.out.print("\tAttributi: " + elencoAttributi);
    System.out.print("\n");
  }
  int nrFigli = nodoCorrente.getChildNodes().getLength();
  //se non si tratta di una foglia continua l'esplorazione per tutti i figli
  if (nodoCorrente.hasChildNodes()){
    NodeList figli = nodoCorrente.getChildNodes();
    for (int conta = 0; conta < nrFigli; conta++){
      stampaNodo(figli.item(conta));                   // chiamata ricorsiva
    }
  }
}
```

Osserviamo che al suo interno richiamiamo due metodi:

**1** il primo metodo analizza il nodo per estrarne il valore del suo primo figlio, se presente:

```
// estrae il testo da ogni nodo: e' il primo item
private static String analizza(Node elementNode){
  String testo = "";
  if (elementNode.hasChildNodes()) {
    Node testoFiglio = elementNode.getChildNodes().item(0);
    testo = testoFiglio.getNodeValue();
  }
  return testo;
}
```

# 4 ESERCIZI IN LABORATORIO

**2** il secondo metodo predispone gli attributi in modo da visualizzarli formattati sullo schermo:

```
66    // predispone l'elenco degli attributi di un nodo
67    private static String formattaAttributi(NamedNodeMap nnm){
68      String elenco = new String();
69      if (nnm != null && nnm.getLength() > 0){
70        for (int conta=0; conta < nnm.getLength(); conta++){
71          elenco += nnm.item(conta).getNodeName();
72          elenco += "=";
73          elenco += nnm.item(conta).getNodeValue();
74          elenco += "; ";
75        }
76        return elenco;
77      }
78      else  return "assenti";
79    }
80  }
```

Avviamo l'esecuzione e sullo schermo vengono visualizzati i tre record presenti nel file come riportato nella figura seguente:

```
BlueJ: Blue3: Terminale - JAVA
Opzioni
Elemento: scuola

Elemento: studente      Attributi: ID=01;
Elemento: nominativo
Elemento: nome = Mario
Elemento: cognome = Rossi
Elemento: indirizzo = via Verdi 15
Elemento: citta = Genova
Elemento: telefono = 339.1223344

Elemento: studente      Attributi: ID=02;
Elemento: nominativo
Elemento: nome = Paolo
Elemento: cognome = Verdi
Elemento: indirizzo = via Roma, 22
Elemento: citta = Genova
Elemento: telefono = 339.1223344

Elemento: studente      Attributi: ID=03;
Elemento: nominativo
Elemento: nome = Ciro
Elemento: cognome = Meo
Elemento: indirizzo = via Milano 1
Elemento: citta = Napoli
```

Il codice completo del programma è presente nel file StudentiDOMParser.java.

# ESERCIZI IN LABORATORIO 5

**AREA DIGITALE**

- **ESERCIZI IN LABORATORIO 6** Web server Apache
- **ESERCIZI IN LABORATORIO 7** Web server Apache in Linux
- **ESERCIZI IN LABORATORIO 8** Web server IIS su Windows

## IL LINGUAGGIO JSON

### Cos'è JSON

JSON è l'acronimo di JavaScript Object Notation ed è un formato standard, aperto, adatto a immagazzinare varie tipologie di informazioni e a scambiare queste informazioni tra applicazioni, sia stand alone che nel Web.
La facilità di scrittura e di analisi della notazione JSON ne ha fatto uno dei suoi punti di forza e ne ha diffuso l'utilizzo tra gli sviluppatori.

> Già dal 2005 Yahoo! supporta JSON come alternativa dei suoi servizi Web e oggi è supportato praticamente da tutte le applicazioni Web.

Rispetto al formato di dati XML, la dimensione del documento JSON è molto inferiore: infatti JSON è privo di schemi e quindi è necessario "gestire e manovrare" elementi strutturali come spazi dei nomi e involucri.
JSON si basa su un sottoinsieme del linguaggio di programmazione JavaScript, Standard ECMA-262 Terza Edizione (1999), ma è un formato indipendente dalla piattaforma e viene utilizzato in più linguaggi di programmazione tra cui Java, Python, PHP, Ruby ecc.

Qualsiasi linguaggio in grado di analizzare una stringa può gestire JSON.

La popolarità di JSON ha fatto sì che le ultime versioni di importanti database come PostgreSQL e MySQL contengano il supporto nativo per l'interrogazione dei dati memorizzati nei campi JSON.
Anche database NoSQL, come MongoDB, sono stati costruiti partendo da questo formato e utilizzano documenti JSON per archiviare i record, proprio come tabelle e righe archiviano i record in un database relazionale.

### Differenze tra XML e JSON

Sappiamo che XML è uno strumento versatile, dotato di grande flessibilità, in grado di adattarsi a ogni situazione e di crescere insieme a qualsiasi progetto, poiché con esso è sempre possibile creare nuovi tag e nidificarli nelle strutture XML già definite, in base alle proprie esigenze.

> Negli ultimi anni XML ha aumentato la sua diffusione anche grazie al fatto che si è imposto come un veicolo fondamentale per lo scambio di messaggi tra i diversi nodi di applicazioni distribuite, utilizzato per la gestione dei dati

Nei casi molto articolati, però, XML risulta a volte poco comprensibile la "vista d'insieme del documento" e richiede un'accurata analisi in tutta la sua interezza per comprendere l'organizzazione dei dati.
Anche JSON è un formato per lo scambio di dati, ma possiede una struttura ancora più semplice di XML, che consente di capire quasi immediatamente il significato di ciò che descrive: questa particolarità è utile specialmente quando si lavora a progetti molto grandi e, soprattutto, quando al progetto collaborano più programmatori che utilizzano linguaggi diversi, da C a Java oppure Perl o Python.

> Un'ulteriore caratteristica di JSON è quella di utilizzare JavaScript come linguaggio e, quindi, di poter essere semplicemente integrato nelle applicazioni Web dato che tutte le informazioni ricavate da JSON possono essere usate in JavaScript avvalendosi semplicemente della funzione eval().

# 5 ESERCIZI IN LABORATORIO

Naturalmente la conoscenza di JavaScript è un prerequisito necessario per l'utilizzo di JSON, mentre XML non richiede nessuna conoscenza pregressa.

## Formato di JSON

JSON è costituito da due sole strutture:
- un insieme di coppie (nome, valore);
- una lista ordinata di valori.

Il vantaggio di avere solo queste due strutture lo rende facilmente utilizzabile dato che praticamente tutti i linguaggi possono rappresentare le coppie o mediante array (nel Web abbiamo i "comodi" array associativi) oppure come record, e ammettono la definizione di strutture a lista.
JSON è il formato maggiormente utilizzato per i messaggi di risposta dei servizi Web anche per il fatto di utilizzare JavaScript: ne vediamo un primo esempio utilizzando la sintassi degli oggetti letterali.

### Oggetti letterali JSON

Un oggetto letterale può essere definito con un insieme di coppie proprietà/valori separate dalla virgola: l'intero oggetto viene racchiuso tra parentesi graffe.

```
var oggetto JSON = {
 proprieta1: 'valore',
 proprieta2: 'valore',
 . . .
 proprietaN: 'valore'
}
```

Come differenza rispetto a JavaScript sottolineiamo che JSON ammette solo valori semplici e atomici (stringhe, numeri, array, oggetti letterali, true, false, null) e, quindi, non può contenere funzioni.

## Tipo dei dati

**AREA DIGITALE**

**Diagrammi sintattici delle istruzioni/strutture dati JSON**

JSON supporta i tipi di dati analizzati di seguito.

### Number

Viene utilizzato un unico formato per tutti i numeri ed è quello in virgola mobile a doppia precisione utilizzato da JavaScript: non sono previsti l'ottale e l'esadecimale.
Anche il NaN e l'Infinity non sono contemplati.

```
var giorni = {marzo: 31}
```

### String

Si tratta di una sequenza di zero o più caratteri Unicode delimitati da virgolette e terminante con backslash, dove ogni carattere è una stringa di caratteri singola cioè una stringa di lunghezza 1.

```
var alunno = {nome: 'Andrea'}
```

# ESERCIZI IN LABORATORIO 5

## Boolean

Può assumere i due valori true oppure false.

```
var alunno = {nome: 'Andrea', promosso: true }
```

## Array

È un insieme ordinato di valori racchiuso tra parentesi quadre: i valori sono separati da "," (virgola) con l'indicizzazione dell'array che può iniziare sia con indice 0 oppure 1.

```
{
  "books": [
    { "linguaggio":"Java" , "edizione":"seconda" },
    { "linguaggio":"Python" , "edizione":"prima" },
    { "linguaggio":"C++" , "edizione":"seconda" }
  ]
}
```

## Object

È un insieme non ordinato di coppie nome/valore racchiusi tra parentesi graffe: ogni nome è seguito da ':' (due punti) e le coppie nome/valore sono separati da "," (virgola).
I nomi devono essere stringhe diverse tra loro perché svolgono la funzione di chiave.

```
{
 "id": "01669",
 "languggio": "JAVA",
 "prezzo": 15.50
}
```

## Whitespace

Lo spazio bianco può essere inserito tra una qualsiasi coppia di token per migliorare la leggibilità del codice: confrontiamo le due dichiarazioni seguenti:

```
var x = " ascissa";
var y = " ordinata ";
```

## null

Utilizzato per l'inizializzazione a vuoto delle variabili.

```
var x = null;
if(x==1)
{
  document.write("<h1>il valore di x è 1</h1>");
}
else
{
  document.write("<h1>il valore di x è null</h1>");
}
```

# 5 ESERCIZI IN LABORATORIO

Riassumiamo i tipi di dati in una tabella:

| TIPO | DESCRIZIONE |
|---|---|
| Number | formato in virgola mobile a doppia precisione formato in JavaScript |
| String | doppio byte – Unicode con backslash finale |
| Boolean | vero o falso |
| Array | una sequenza ordinata di valori |
| Object | una collezione non ordinata di coppie chiave: valore |
| Whitespace | lo spazio bianco può essere utilizzato tra una qualsiasi coppia di token |
| null | vuoto |

## Creare oggetti JSON in JavaScript

Ci sono tre modalità per creare un oggetto JSON utilizzando JavaScript:

**1** creare un oggetto vuoto:

```
var oggetto1 = {};
```

**2** creare un nuovo oggetto:

```
var oggetto2 = new Object();
```

**3** creare un nuovo oggetto assegnando i valori:

```
var oggetto3 = { “titolo”:”TPSIT vol.3”, “prezzo”:23.50 };
```

Vediamo un primo esempio in cui inseriamo del codice JSON all’interno del codice JavaScript di una pagina HTML a definizione di un singolo letterale e a definizione di un array.

**Definizione di un letterale semplice**

**Definizione di un array di letterali**

# ESERCIZI IN LABORATORIO 5

## METTITI ALLA PROVA

Modifica la struttura del record aggiungento il campo prezzo in formato intero; quindi modifica la stuttura gerarchica aggiungendo ai libri anche l'argomento, ottendendo il seguente schema:

```
var libri = {
 "informatica" : [
  { "linguaggio" : "C++", "autore": "Rossi", "prezzo" : 15.50},
  { "linguaggio" : "JavaScript", "autore": "Verdi","prezzo" : 17.50},
  { "linguaggio" : "Java","autore": "Bianchi", "prezzo" : 12.50},
 ],
 "cucina" : [
  { "lingua" : "italiano","portata" : "antipasti", "prezzo" : 25.50},
  { "lingua" : "cinese","portata" : "dolci", "prezzo" : 13.00 }
 ]
}
```

Quindi completa lo script aggiungendo un nuovo oggetto in modo che sullo schemo venga visualizzato il seguente output:

**Array di oggetti JSON**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| linguaggio | C++ | linguaggio | JavaScript | linguaggio | Java | portata | antipasti | portata | dolci |
| autore | Rossi | autore | Verdi | autore | Bianchi | lingua | italiano | lingua | cinese |
| prezzo | 15.5 | prezzo | 17.5 | prezzo | 12.5 | prezzo | 25.5 | prezzo | 13 |

Confronta la tua soluzione con quella presente nel file JSON_libri.htm.

È anche possibile realizzare oggetti nidificati, come nel seguente esempio:

```
var alunni = {                                  // definiamo un oggetto alunni composto da tre "elementi"
 "pino" : {                                     // ogni elemento è un oggetto
   "nome" : "Giuseppe",
   "eta" : "21"
  },
 "mario" : {
   "nome" : "Mariolino",
   "eta" : "24"
  },
 "nino" : {
   "nome" : "Giannino",
   "eta" : "20"
 }
}
```

In questo caso, l'oggetto alunni contiene tre "proprietà": pino, mario, gianni; quest'ultime, a loro volta, sono degli oggetti che contengono una serie di nomi/valori.

## Notazioni a confronto

Riportiamo un esempio con tre diverse modalità di definizione di un oggetto per la codifica dei colori RGB in esadecimale in modo da poterne confrontare la sintassi:

# 5 ESERCIZI IN LABORATORIO

**Codifica dei colori RGB in esadecimali**

```
{
   "colorsArray":
   [
     {
       "colorName":"red",
       "hexValue":"#f00"
     },
     {
       "colorName":"green",
       "hexValue":"#0f0"
     },
     {
       "colorName":"blue",
       "hexValue":"#00f"
     },
     {
       "colorName":"black",
       "hexValue":"#000"
     }
   ]
}
```

```
{
   "colorsArray":
   [
     {
       "red":"#f00",
       "green":"#0f0",
       "blue":"#00f",
       "black":"#000"
     }
   ]
}
```

```
{
   "red"    :"#f00",
   "green"  :"#0f0",
   "blue"   :"#00f",
   "black"  :"#000"
}
```

Ricordiamoci di separare una proprietà dall'altra con una virgola "," mentre dopo l'ultima proprietà questa non è richiesta.

## File con suffisso JSON

Il formato JSON è anche utilizzato per rendere disponibili archivi di dati che vengono memorizzati in file con suffisso JSON. A titolo di esempio alleghiamo il file colorName.JSON dove sono presenti i nomi e la codifica esadecimale di tutti i colori RGB del quale ne riportiamo una piccola parte:

```
{
 "aliceblue": "#f0f8ff",
 "antiquewhite": "#faebd7",
 "aqua": "#00ffff",
 "aquamarine": "#7fffd4",
 "azure": "#f0ffff",
 "beige": "#f5f5dc",
 "bisque": "#ffe4c4",
 "black": "#000000",
 "blanchedalmond": "#ffebcd",
 "blue": "#0000ff",
 "blueviolet": "#8a2be2",
 . . .
}
```

# ESERCIZI IN LABORATORIO 5

## Conversione con eval()

Per convertire un testo JSON in un oggetto JSON si utilizza la funzione eval():

```
var mioOggettoJSON = eval('('+ mioTestoJSON + ')');
```

La concatenazione di parentesi è un artificio per eludere l'ambiguità della grammatica di JavaScript: eval() invoca il compilatore JavaScript e poiché JSON è un sottoinsieme corretto di JavaScript, il compilatore analizzerà il testo e produrrà una struttura a oggetti.
Nell'esempio che segue convertiamo una stringa di testo in un oggetto.

```
JSON/JSON_oggetto.html
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <h2>Crea un oggetto JSON da una stringa </h2>
5  <script>
6  var testo = '{"autori":[' +
7  '{"nome":"Paolo","cognome":"Camagni" },' +
8  '{"nome":"Riccardo","cognome":"Nikoolaeny" }]}';
9  var oggetto = eval ("(" + testo + ")");
10 document.write("<h3>Il primo autore = " + oggetto.autori[0].nome +"</h3>");
11 document.write("<h3>Il secondo autore = " + oggetto.autori[1].nome +"</h3>");
12 </script>
13 </body>
14 </html>
```

Avviando l'esecuzione viene visualizzato il seguente output:

**Crea un oggetto JSON da una stringa**

**Il primo autore = Paolo**

**Il secondo autore = Riccardo**

Allo stesso risultato si giunge utilizzando la funzione parse() di JSON:

```
var oggetto = JSON.parse (testo);
```

utilizzabile, però, solo nei browser più aggiornati che lo supportano.

**AREA DIGITALE**

**JSON e HTML5**

## JSON e Java

Il linguaggio Java utilizza il formato JSON grazie alle molteplici librerie che nel corso degli anni sono state realizzate per permettere un'agevole manipolazione dei dati: inoltre i più famosi framework supportano JSON.

Noi utilizziamo la libreria json-simple-1.1.1.jar che è scaricabile dall'indirizzo:
https://code.google.com/archive/p/json-simple/downloads.

Un oggetto JSON è una raccolta non ordinata di coppie nome/valore: segnaliamo alcune delle sue operazioni fondamentali:

- creazione di un oggetto JSON con il metodo JSONObject();
- creazione di un vettore di elementi JSON con il metodo JSONArray().

# 5 ESERCIZI IN LABORATORIO

Tra i metodi più utilizzati ricordiamo:
- il metodo .put() per creare oggetti JSON;
- il metodo .get() per interrogare le proprietà degli oggetti;
- il metodo .parse() per leggere un JSON presente in una stringa;
- il metodo .toJSONString() per convertire un oggetto JSON in una stringa.

Come primo esempio scriviamo un programma che crea un documento JSON e lo salva su disco: tra le prime istruzioni inseriamo le direttive per includere le librerie necessarie:

```
1 package JSON;
2 import org.json.simple.JSONArray;
3 import org.json.simple.JSONObject;
4 import java.io.FileWriter;
5 import java.io.IOException;
```

Procediamo con tre operazioni:
1 creiamo dapprima un oggetto JSON nel quale andremo a scrivere singolarmente alcuni elementi formati da coppie <nome>:<dato>;
2 creiamo quindi un array di elementi JSON e in esso inseriamo tre elementi JSON;
3 richiamiamo infine il metodo che andrà a scrivere questo oggetto in un file e ne visualizziamo il contenuto sullo schermo.

```
7   public static void main(String[] args) {
8     String nomeFile= "dati/prova1.json";
9     JSONObject mioOggetto = new JSONObject();   // creo l'oggetto JSON
10    mioOggetto.put("cognome", "Verdi");         // metto singolo elemento
11    mioOggetto.put("nome", "Pino");
12    mioOggetto.put("eta", 40);
13    JSONArray elenco = new JSONArray();         // creo un array JSON
14    elenco.add("lingua 1: cinese");
15    elenco.add("lingua 2: russo");
16    elenco.add("lingua 3: turco");
17    mioOggetto.put("lingue", elenco);           // metto un oggetto Array
18    System.out.print(mioOggetto);               // visualizza tutti i dati
19    scriviFileJSON(nomeFile, mioOggetto);       // creo il file JSON
20  }
```

Il metodo ScriviFileJSON() riceve come parametri il nome del file che deve creare e l'oggetto JSON che contiene i dati da salvare: dopo aver creato un oggetto FileWriter vi scriviamo il contenuto dell'oggetto che abbiamo appena realizzato, utilizzando il metodo toJSONString() che lo converte in una stringa.

```
22  public static void scriviFileJSON(String nomeFile,JSONObject mioOggetto ){
23    try (FileWriter mioFile = new FileWriter(nomeFile)) {
24      mioFile.write(mioOggetto.toJSONString());
25    } catch (IOException e) {
26      e.printStackTrace();
27    }
28  }
```

Avviamo l'esecuzione del main() ottenendo il seguente output:

```
Output
{"lingue":["lingua 1: cinese","lingua 2: russo","lingua 3: turco"],"eta":40,"cognome":"Verdi","nome":"Pino"}
```

Il codice completo è riportato nel file Scrivi_JSON.java.

# ESERCIZI IN LABORATORIO 5

Scriviamo ora un programma che legge da disco il documento JSON che abbiamo appena creato: come al solito inseriamo tra le prime istruzioni le direttive per includere le librerie necessarie:

```
1 package JSON;
2 import java.io.Reader;
3 import java.io.FileReader;
4 import java.util.Iterator;
5 import java.io.IOException;
6 import org.json.simple.JSONArray;
7 import org.json.simple.JSONObject;
8 import org.json.simple.parser.*;
```

Per effettuare la lettura del file scriviamo un apposito metodo leggiFileJSON() che riceve in input il nome del file da leggere e lo utilizza come parametro per il costruttore dell'oggetto FileReader: quindi predisponiamo un oggetto della classe JSONParser() al quale passiamo il file appena letto per parserizzarlo e trasformarlo in un oggetto JSON:

```
  public static Object leggiFileJSON(String nomeFile) throws Exception {
    FileReader reader = new FileReader(nomeFile);
    JSONParser parserJSON = new JSONParser();
    return parserJSON.parse(reader);
  }
}
```

Il main() elabora il contenuto dell'oggetto JSON: dapprima ne visualizza l'intero contenuto semplicemente con il metodo println(), quindi ne estrae i singoli componenti utilizzando il metodo get().

```
  public static void main(String[] args) throws Exception {
    //leggo il file JSON
    String nomeFile= "dati/prova1.json";
    JSONObject oggettoJSON = (JSONObject) leggiFileJSON(nomeFile);
    // visualizzo contenuto del file
    System.out.println("Contenuto intero oggetto:");
    System.out.println(oggettoJSON);
    // visualizzo singoli elementi
    System.out.println("Singoli elementi:");
    String cognome = (String) oggettoJSON.get("cognome");
    System.out.print("il sig." + cognome);
    long anni = (Long) oggettoJSON.get("eta");
    System.out.println(" di anni " + anni +" parla le seguenti lingue:");
    // ciclo e scorro il contenuto dell'array
    JSONArray lingue = (JSONArray) oggettoJSON.get("lingue");
    Iterator<String> iterator = lingue.iterator();
    while (iterator.hasNext()) {
      System.out.println("-"+ iterator.next());
    }
  }
```

Abbiamo tra gli elementi del file JSON un array di dati contenente le lingue parlate: creiamo un oggetto JSONArray() dove memorizziamo questi dati che visualizzeremo scorrendolo con un iteratore. Avviamo l'esecuzione e otteniamo:

```
Contenuto intero oggetto:
{"lingue":["lingua 1: cinese","lingua 2: russo","lingua 3: turco"],"eta":40,"cognome":"Verdi","nome":"Pino"}
Singoli elementi:
il sig.Verdi di anni 40 parla le seguenti lingue:
-lingua 1: cinese
-lingua 2: russo
-lingua 3: turco
```

Il codice completo è riportato nel file Leggi_JSON.java.

# 5 ESERCIZI IN LABORATORIO

## JSON e PHP

A partire da PHP 5.2 sono integrate nativamente delle funzioni per la codifica e la decodifica dello stream JSON; esse sono json_ encode() e json_ decode():

- json_encode( mixed value ) si occupa di trasformare un valore PHP nella stringa JSON che lo rappresenta;
- json_decode( string json [, bool assoc] ) effettua l'operazione contraria trasformando una stringa JSON in un valore PHP valido.

### La funzione json_encode()

Vediamo un primo esempio dove convertiamo un array PHP in formato JSON:

```
1 <?php
2 // array per la codifica
3 $vettore = array("sequenza", "selezione", "iterazione");
4 // codifica JSON
5 $stringa = json_encode($vettore);
6 // stampa del risultato
7 echo $stringa;
8 ?>
```

L'esecuzione produce il seguente output:

Effettuiamo ora sempre la conversione di un array associativo:

```
1 <?php
2 $vettore = array('qui' => 1, 'quo' => 2, 'qua' => 3);
3 // codifica JSON
4 $stringa = json_encode($vettore);
5 // stampa del risultato
6 echo $stringa;
7 ?>
```

L'esecuzione produce il seguente output:

Descriviamo il funzionamento dello script: la funzione json_ encode() svolge il compito di restituire una stringa contenente la rappresentazione di un valore partendo da una semplice array: il server invia una richiesta di stampa a video dell'operazione di codifica dei dati operata tramite json_encode() e il client effettuerà una decodifica per restituirli nuovamente sotto forma di vettore: i valori rappresentati nella stringa saranno gli stessi che compongono l'array.

Nel secondo esempio definiamo un oggetto con PHP e lo convertiamo in formato JSON:

# ESERCIZI IN LABORATORIO 5

```
1  <?php
2  class Atleta {
3   public $nome = "";
4   public $nazione = "";
5   public $sport = "";
6   public $dataNascita = "";
7  }
8  $e = new Atleta();
9  $e->nome = "Pele";
10 $e->nazione = "Brasile";
11 $e->sport = "calcio";
12 $e->dataNascita = date('m-d-Y h:i:s a', strtotime("23-10-1940 12:10:03"));
13 echo json_encode($e);
14 ?>
```

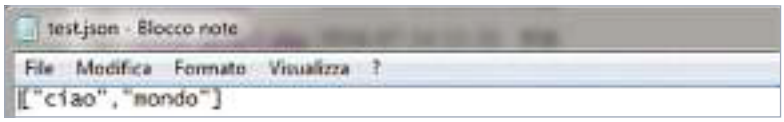
L'esecuzione produce il seguente output:

Vediamo un ulteriore esempio dove aggiungiamo la definizione MIME per il formato di ritorno (può essere sia JSON che JavaScript): questo script ritorna al programma chiamante un vettore che contiene il parametro che ha ricevuto al momento della sua chiamata:

```
1  <?php
2  // definizione MIME per JSON
3  header('Content-type: application/json');
4  // leggiamoil parametro passato
5  $id = $_GET['id'];
6  // creiamo un array e lo ritorniamo al chiamante
7  $data = array("ciao", $id);
8  // Send the data.
9  echo json_encode($data);
10 ?>
```

Chiamiamo la funzione passando direttamente la variabile:

Possiamo visualizzare l'array che viene generato aprendo il risultato col Blocco note:




## La funzione json_decode()

La funzione json_decode() viene utilizzata per trasformare dati JSON in PHP e, a seconda del suo parametro, ritorna o meno un array associativo (se tale parametro viene omesso equivale a false).

```
mixed value json_decode ($obj_json [,$associativo = false ])
```

# 5 ESERCIZI IN LABORATORIO

Vediamo un primo esempio dove omettiamo il parametro, quindi l'array non sarà associativo:

```
1 <?php
2 // definizione dell'oggetto per il parsing
3 $mesi = '{"gennaio":1,"febbraio":2,"marzo":3,"aprile":4}';
4 var_dump(json_decode($mesi));
5 echo "<p>";
6 // lettura dello stream JSON
7 $oggetto = json_decode($mesi);
8 echo $oggetto->{'aprile'};
9 ?>
```

L'esecuzione produce il seguente output:

```
localhost/json/decode1.php
object(stdClass)#1 (4) { ["gennaio"]=> int(1) ["febbraio"]=> int(2) ["marzo"]=> int(3) ["aprile"]=> int(4) }
4
```

Nel secondo esempio dove mettiamo il parametro a true:

```
1 <?php
2 // definizione dell'oggetto per il parsing
3 $mesi = '{"gennaio":1,"febbraio":2,"marzo":3,"aprile":4}';
4 var_dump(json_decode($mesi, true));
5 echo "<p>";
6 // lettura dello stream JSON
7 $oggetto = json_decode($mesi);
8 echo $oggetto->{'marzo'};
9 ?>
```

l'esecuzione produce il seguente output:

```
localhost/json/decode2.php
array(4) { ["gennaio"]=> int(1) ["febbraio"]=> int(2) ["marzo"]=> int(3) ["aprile"]=> int(4) }
3
```

In entrambi gli esempi abbiamo estratto il valore di un elemento tramite una chiamata al primo elemento della coppia: nella seconda parte il nostro script dopo l'esecuzione della json_decode() visualizza solamente il numero corrispondente al mese indicato nella istruzione 8.

**AREA DIGITALE**

**Select dinamiche con PHP e JSON**

## Conclusioni

Abbiamo visto che JSON offre la possibilità di effettuare codifiche e decodifiche praticamente "al volo", cosa impossibile con XML dato che, con esso, è necessario generare un documento intermedio che penalizza le prestazioni degli script: infatti non deve effettuare il parsing di file che, con l'aumento del numero dei dati da gestire così come del traffico generato dai client, può diventare anche estremamente "pesante".

**AREA DIGITALE**

**Esercizi per il recupero e l'approfondimento**

# I socket e la comunicazione con i protocolli TCP/UDP

UNITÀ 2

**LEZIONE 1**
I socket e i protocolli per la comunicazione di rete

**LEZIONE 2**
La connessione tramite socket

**Esercizi in Laboratorio**

- L1 Java socket
- L2 Java socket: realizzazione di un server TCP
- L3 Realizzazione di un server multiplo in Java
- L4 Java socket: un'animazione client-server
- L5 Il protocollo UDP nel linguaggio Java
- L6 Applicazioni multicast in Java
- L7 Un esempio completo con le Java socket: "la chat"
- L8 I socket nel linguaggio C
- L9 Server TCP IN C
- L10 Client TCP in C
- L11 Il protocollo UDP nel linguaggio C
- L12 Un esempio completo con i C socket: "asta on line"
- L13 Applicazioni multicast in C


## CONOSCENZE

- Conoscere i protocolli di rete
- Acquisire il modello di comunicazione in una network
- Avere il concetto di socket e conoscere le tipologie di socket
- Conoscere la comunicazione multicast
- Sapere le caratteristiche della comunicazione con i socket Java
- Sapere le caratteristiche della comunicazione con i socket C

## COMPETENZE

- Effettuare la connessione con il protocollo TCP e UDP
- Acquisire il protocollo UDP nei linguaggi C e Java
- Utilizzo delle classi Classe Socket e ServerSocket
- Progettare applicazioni client-server in Java
- Progettare applicazioni client-server in C

## ABILITÀ

- Realizzare un server e client TCP in Java
- Realizzare un server UDP in Java
- Realizzare un server multiplo in Java
- Realizzare un server e un client TCP in C
- Realizzare un server UDP in C
- Realizzare un server multiplo in C

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

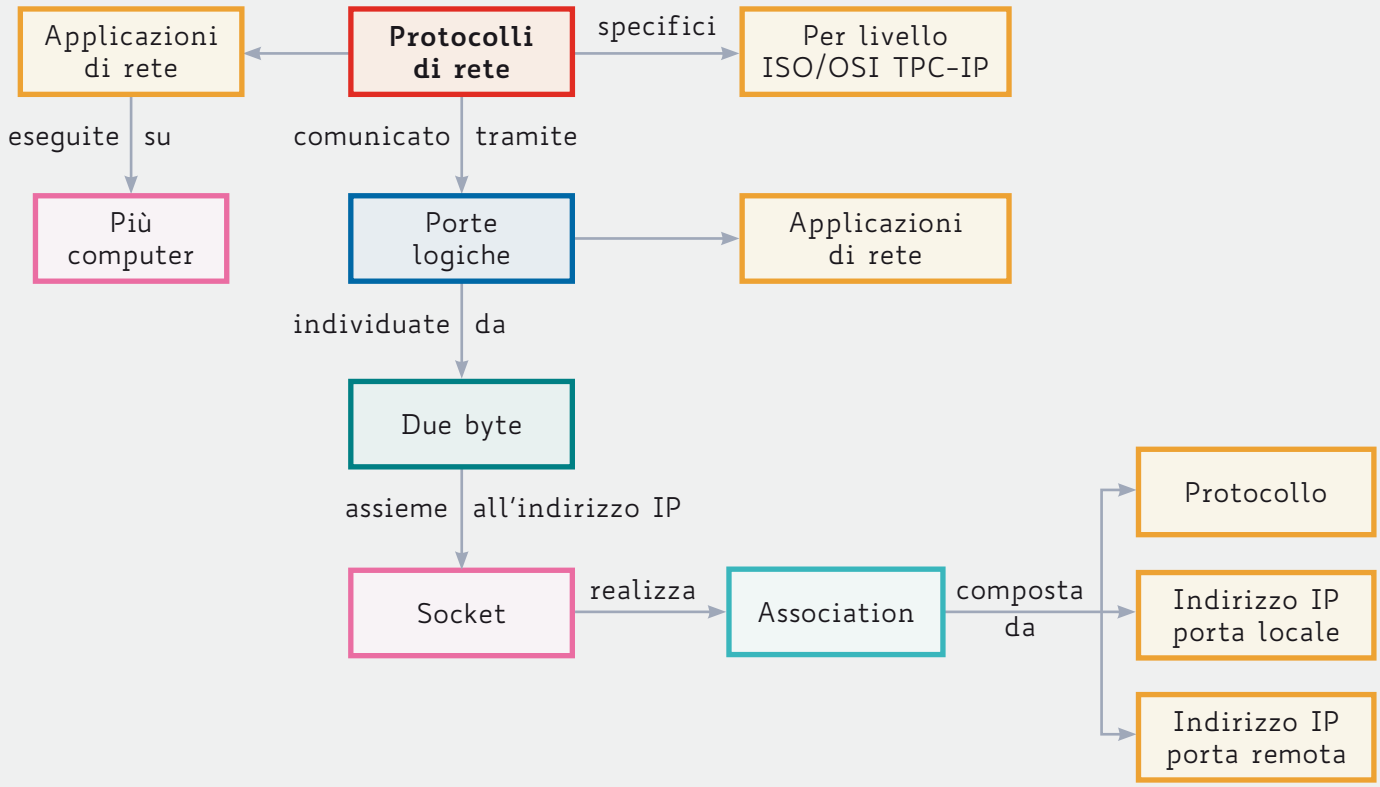
# 1 I socket e i protocolli per la comunicazione di rete

**IN QUESTA LEZIONE IMPAREREMO...**

- I protocolli di rete
- Il modello di comunicazione in una network
- Il concetto di socket

## MAPPA CONCETTUALE

didattica inclusiva



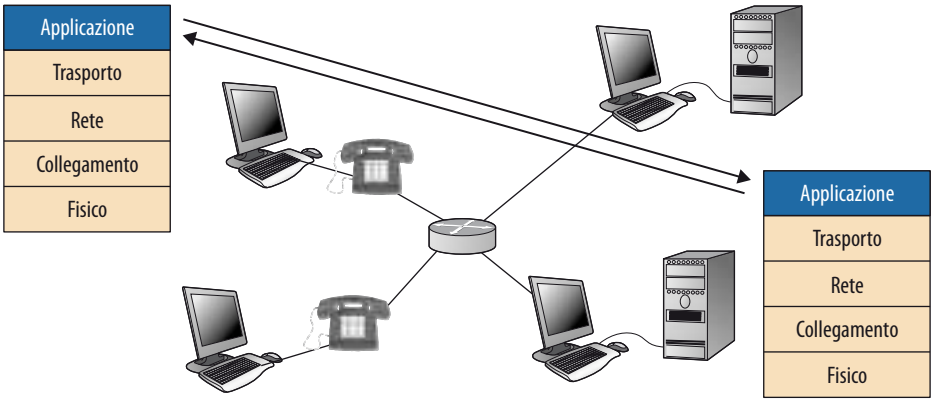

## Generalità

Riprendiamo alcune definizioni già note, come il concetto di applicazione di rete e di protocollo.

### Applicazione di rete

In generale un'applicazione di rete è costituita da un insieme di programmi che vengono eseguiti su due o più computer contemporaneamente: questi operano interagendo tra loro utilizzando delle risorse comuni, accedendo cioè concorrentemente agli archivi (database), mediante la rete di comunicazione che li connette.

L'applicazione di rete prende anche il nome di applicazione distribuita dato che non viene eseguita su di un solo elaboratore (concentrata).




Un'applicazione distribuita è un'applicazione composta da più elementi cooperanti, posti in esecuzione su macchine diverse all'interno di una rete di calcolatori.

I processi hanno la necessità di scambiare messaggi con gli altri processi della medesima applicazione, sia che essi appartengano alla stessa rete locale oppure che siano remoti e quindi dislocati dall'altra parte del globo: per comunicare tra loro questi processi devono mettersi in "contatto" tramite i loro indirizzi e utilizzare i servizi offerti dal livello di applicazione.

## Protocollo di comunicazione

Due o più computer possono cooperare se sono collegati attraverso un sistema di comunicazione e utilizzano un *meccanismo comune di dialogo*: il protocollo di comunicazione.

Con protocollo di comunicazione si indica l'insieme di regole di comunicazione che devono essere seguite da due interlocutori affinché possano comprendersi.

In una rete di calcolatori il protocollo di comunicazione stabilisce tutti gli aspetti della comunicazione, da quelli fisici (per esempio supporto fisico, meccanismo di segnalazione ecc.) a quelli più specificamente logici (per esempio meccanismo di commutazione, regole di codifica dell'informazione ecc.).
I protocolli utilizzati dai calcolatori sono organizzati secondo una gerarchia che prevede che ogni protocollo si appoggi a protocolli di livello più basso per fornire un servizio di qualità superiore

### ESEMPIO

Il protocollo che stabilisce le *regole di codifica dell'informazione* si appoggia a un protocollo di trasporto che stabilisce *come debbano essere trasportati i dati*.

Ricordiamo che la pila protocollare di Internet (o pila TCP/IP) è costituita da cinque livelli, a differenza della pila ISO/OSI che ha sette livelli, e l'unità di informazione che viene gestita a ogni livello assume *forma e nome diverso*, dato che a ogni passaggio vengono aggiunte le informazioni necessarie a ciascun protocollo per poter portare termine i propri compiti:

- al livello applicazione viene elaborato il tipo di dato messaggio;
- al livello trasporto viene elaborato il tipo di dato segmento;
- al livello rete viene elaborato il tipo di dato datagram;
- al livello collegamento (link) viene elaborato il tipo di dato frame;
- al livello fisico non corrisponde alcun tipo di dato (ma solo un segnale fisico).

| | |
|---|---|
| 5 Applicazione | Messaggio |
| 4 Trasporto | Segmento |
| 3 Rete | Datagram |
| 2 Collegamento | Frame |
| 1 Fisico | |

Lo strato di trasporto della rete Internet mette a disposizione delle applicazioni attive in ciascun host due distinti protocolli di trasporto:
1. TCP (Transmission Control Protocol);
2. UDP (User Datagram Protocol).

**AREA DIGITALE**
**Confronto ISO/OSI e Internet**

TCP e UDP svolgono funzioni diverse, cioè offrono servizi diversi allo strato applicativo: ogni applicazione (processo) in esecuzione, prima di poter trasmettere messaggi sulla rete, sceglie il protocollo da usare per poter realizzare la comunicazione. La differenza fondamentale tra i due protocolli, che saranno descritti dettagliatamente nelle prossime due lezioni, è che:

- il protocollo TCP è orientato alla connessione (connection-oriented), ed è affidabile dato che consente il controllo dell'integrità dell'informazione contenuta nei pacchetti e dispone di un sistema per segnalare l'errore al mittente (scambio di messaggi di acknowledge);
- il protocollo UDP è senza connessione (connectionless) e quindi non affidabile.

**AREA DIGITALE**
**Connectionless e connection-oriented**

Sia il TCP che l'UDP usano IP a livello di network. Il nome del protocollo completo di comunicazione viene ottenuto dalla combinazione delle sigle di tali protocolli: per esempio, con TCP/IP si individua l'insieme di questi standard.

## Le porte di comunicazione e i socket

Prima di entrare nelle specifiche e di analizzare le primitive che permettono la realizzazione della comunicazione in rete è necessario riprendere alcuni concetti fondamentali necessari per realizzare la comunicazione tramite network.

Affinché un processo, presente su un determinato host, invii un messaggio a un qualsiasi altro host, il processo mittente deve identificare il processo destinatario in modo univoco.

Generalmente ogni PC ha una sola porta fisica di connessione al network: se più applicazioni necessitano di utilizzare la rete, devono essere riconosciute in qualche modo, permettendo così di ricevere e spedire dati in modo corretto.
Questo metodo di riconoscimento viene effettuato tramite le cosiddette porte logiche identificate da un numero detto port address oppure, sinteticamente, port; in questo modo è possibile instaurare simultaneamente più comunicazioni cosicché due applicazioni possono comunicare l'una con l'altra indipendentemente dal fatto che sulla rete stiano avvenendo altre comunicazioni: è sufficiente utilizzare porte logiche diverse.
Il numero di port è specificato su 2 byte (da 0 a 65535) e identifica un particolare canale utilizzabile per la comunicazione.

| Well-Known Ports<br>0 — 1023 | Registered Ports<br>1024 — 49151 | Dynamic and/or Private Ports<br>49152 — 65535 |
|---|---|---|
| I port da **0** a **1023** sono riservati ad applicazioni particolari (quali HTTP, FTP, DNS, TelNet ecc.), costituiscono i cosiddetti well-known port numbers e possono essere usati solo dai server in apertura passiva. | I numeri di porta da **1024** a **49151** sono riservati a porte registrate e sono usati da alcuni servizi ma possono anche essere utilizzati dai client (tutti i client usano normalmente le porte a partire dalla numero 1024 per collegarsi a un sistema remoto). | I numeri di porta da **49152** a **65535** sono liberi per essere assegnati dinamicamente dai processi applicativi. |

La lista completa dei well-known port numbers si trova su http://www.iana.org/assignments/port-numbers.

Di seguito sono riportati alcuni esempi.

| Porta logica | Protocollo di rete |
|---|---|
| 7 | ECHO |
| 21/tcp | FTP (file transfer protocol) |
| 22/tcp | SSH (Secure SHell) |
| 23/tcp | TELNET |
| 25/tcp | SMTP (Simple Mail Transfer Protocol) |
| 42 | WINS (Windows Internet Naming Service) |

| Porta logica | Protocollo di rete |
|---|---|
| 53 | DNS (Domain Name Service) |
| 80/tcp | HTTP (Hyper Textual Transfer Protocol) |
| 110/tcp | POP3 (Post Office Protocol, v3) |
| 143/tcp | IMAP (Internet Message Access Protocol) |
| 161 | SNMP (Simple Network Management Protocol) |
| 443/tcp | HTTPS (Secure HTTP) |

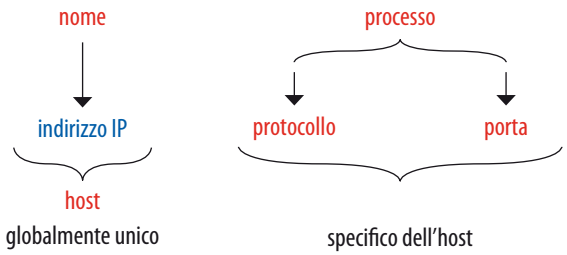
È bene sottolineare il fatto che i numeri di porta logica sono relativi soltanto al protocollo considerato: una determinata porta per il protocollo TCP è diversa dallo stesso numero di porta per il protocollo UDP (si tratta effettivamente di porte diverse) anche se in genere viene utilizzato lo stesso numero di porta per un servizio che gestisce entrambi i protocolli.

Ma il solo numero di port *non è però sufficiente* a descrivere in maniera univoca una connessione.

Infatti più processi su host differenti potrebbero avere il medesimo port ed essere quindi riconosciuti come un processo unico: è un dato identificativo dell'host!

L'identificazione di un servizio, quindi, avviene combinando l'indirizzo IP dell'host e della porta logica che viene utilizzata per renderlo disponibile in rete, quindi:

- indirizzo IP: identifica il nodo su cui opera il processo con cui si desidera comunicare;
- porta logica: identifica la porta utilizzata dal particolare processo all'interno di quel nodo.




## ESEMPIO

Un caso tipico è quello rappresentato da un server sul quale sono disponibili più servizi, tra i quali:

- e-mail: viene inviata usando il protocollo applicativo SMTP, quindi occorre inviare un messaggio opportunamente codificato alla porta TCP 25 del server;
- sito Web: per richiedere una pagina Web si usa il protocollo applicativo HTTP, che invia un opportuno messaggio di richiesta alla porta TCP 80 del server;
- la trasformazione di un nome di un calcolatore in un indirizzo IP: che consiste nell'inviare un'opportuna richiesta alla porta UTP 53 del server che offre il servizio DNS.

Se un client desidera richiedere uno di questi servizi offerti da questo server è necessario che specifichi non solo l'indirizzo dell'host ma anche il servizio desiderato.

L'identificazione univoca avviene conoscendo sia l'indirizzo IP che il numero di porta associato al processo in esecuzione su un host: questo meccanismo prende il nome di meccanismo dei socket.

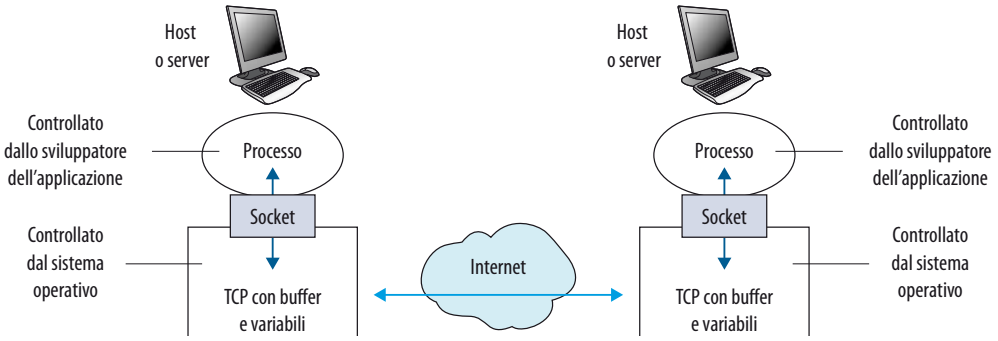
**Socket**

Un socket è formato dalla coppia <indirizzo IP: numero della porta>: è un identificatore analogo a una porta, cioè a un punto di accesso/uscita, unico per ogni rete.

Un processo che vuole inviare un messaggio lo "fa uscire" dalla propria "interfaccia" (socket del mittente) sapendo che un'infrastruttura esterna lo trasporterà attraverso la rete fino alla "interfaccia" del processo di destinazione (socket del destinatario).

**AREA DIGITALE**

**Socket: maschile o femminile?**




Un socket consente quindi di comunicare attraverso la rete usando la pila TCP/IP ed è perciò parte integrante del protocollo: le API mettono a disposizione del programmatore gli strumenti atti a codificare la connessione e l'uso del protocollo di comunicazione.

**AREA DIGITALE**

**API – Application Programming Interface**

## ESEMPIO

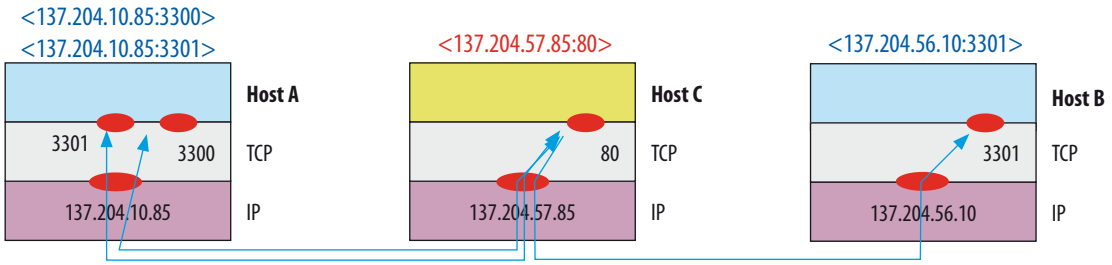
Nell'esempio schematizzato nella figura seguente, due applicazioni dell'host A e una dell'host B con i seguenti socket:

host A: <137.204.10.85:3300>
host A: <137.204.10.85:3301>
host B: <137.204.56.10:3301>

si connettono alla stessa porta 80 dell'host C richiedendo un servizio http al socket:

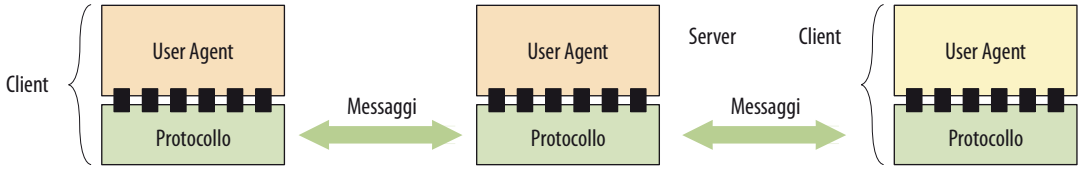
host C: <137.204.57.85:80>

avendo due il medesimo indirizzo IP sono univocamente individuati mediante il valore del socket.




L'applicazione di rete può essere vista come composta da due parti:

**1** una user agent, che funge da interfaccia tra l'utilizzatore dell'applicazione e gli aspetti comunicativi;
**2** l'implementazione dei protocolli, che permettono all'applicazione di integrarsi con la rete.




## ESEMPIO

Possiamo individuare queste due componenti per esempio in un browser Web:

- l'interfaccia utente che serve a visualizzare i documenti ricevuti, a permettere la loro navigazione e a richiedere nuovi documenti specificando la loro URL;
- il motore del browser che è la parte che si preoccupa di inviare le richieste ai vari server e di ricevere le risposte.

Esiste una struttura chiamata association che consente d'identificare ogni singola connessione in modo univoco e che contiene le seguenti informazioni:

| | | |
|---|---|---|
| Protocollo (TCP, UDP…) | Indirizzo IP (locale) | Indirizzo IP (remoto) |
| | Porta (locale) | Porta (remota) |

## ESEMPIO

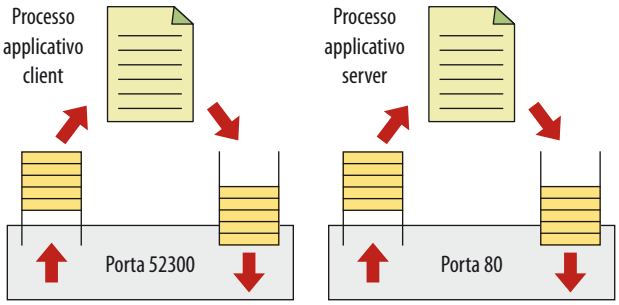
Un esempio di association è: TCP, 192.168.1.2, 1500, 192.168.1.14, 21. Nel dettaglio:
- TCP è il protocollo da utilizzare;
- 192.168.1.2 è l'indirizzo IP del PC mittente;
- 1500 è il port address del processo locale;
- 192.168 .1.14 è l'indirizzo IP del PC destinatario;
- 21 è il port address del processo remoto.

## Socket e i processi client-server

Il modello client-server è organizzato in due moduli, chiamati appunto server e client, operanti su macchine diverse:
- il server svolge le operazioni necessarie per realizzare un servizio;
- il client, generalmente tramite un'interfaccia utente, acquisisce i dati, li elabora e li invia al server richiedendo un servizio.

Quindi in rete sono presenti calcolatori su cui girano processi server che erogano servizi e sono in attesa di ricevere richieste di connessione da parte di processi client interessati a usufruire di tali servizi.

Il client deve conoscere l'indirizzo IP e il numero di porta usati dal server per potersi collegare al socket di destinazione: per esempio, il servizio HTML del server è sulla porta 80 mentre quello del client è libero, a scelta tra quelli disponibili (per esempio 52300), ma deve essere trasmesso al client nel segmento di richiesta in modo che il server possa successivamente inviare la risposta.

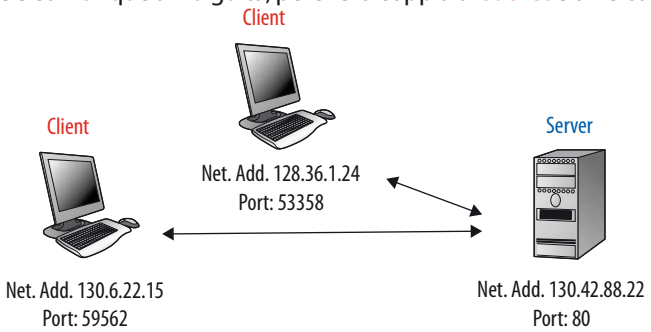



Naturalmente allo stesso server arrivano contemporaneamente richieste da client diversi che potrebbero anche utilizzare la stessa porta, come nel precedente esempio la 3301, o richieste da porte diverse dello stesso host: il collegamento avviene univocamente tramite i socket.

## ESEMPIO

- Un client si connette alla porta di un server SMTP remoto (servizio di posta elettronica).

- Due client accedono alla stessa porta di un server HTTP; non c'è comunque ambiguità, perché la coppia di socket è diversa.




Quando un client apre un socket, non dovrebbe utilizzare una porta nota, in quanto riservata per l'offerta di servizi.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Quale tra le seguenti non è una applicazione di rete?**
- **a** Posta elettronica
- **b** Condivisione di file P2P
- **c** Scheduler dei processi
- **d** Telefonia via Internet
- **e** Videoconferenza in tempo reale
- **f** TV in streaming
- **g** Telnet

**2 Qual è il significato di API?**
- **a** Application Protocol Internet
- **b** Application Protocol Interface
- **c** Application Programming Interface
- **d** Application Programming Internet

**3 Quale tra i seguenti servizi offerti non è garantito dallo strato di trasporto alle applicazioni?**
- **a** Trasferimento dati affidabile
- **b** Ampiezza di banda
- **c** Velocità di comunicazione
- **d** Temporizzazione
- **e** Sicurezza

**4 In cosa consiste una association?**
- **a** Nella coppia protocollo di connessione – socket locale
- **b** Nella coppia protocollo di connessione – socket remoto
- **c** Nella coppia socket locale – socket remoto
- **d** Nella ennupla protocollo di connessione – socket locale – socket remoto

**5 Che cosa si intende per port address?**
- **a** L'indirizzo fisico di una porta
- **b** Il numero associato a una porta logica
- **c** L'indirizzo di memoria di una porta
- **d** L'indirizzo TPC/IP di un PC

**6 A che cosa serve l'association?**
- **a** Consente di identificare ogni singola connessione in modo univoco
- **b** Consente di identificare ogni singolo pc nella rete
- **c** Consente di associare una porta logica a una porta fisica
- **d** Consente di associare un PC a un indirizzo IP

## VERO/FALSO

1 Il protocollo HTTP è a livello di trasporto. V F
2 Il protocollo TCP è a livello di applicazione. V F
3 L'applicazione di rete prende anche il nome di applicazione distribuita. V F
4 L'applicazione di rete prende anche il nome di applicazione multiutente. V F
5 Sullo stesso host possono essere in esecuzione molti processi. V F
6 Sullo stesso host può essere in esecuzione una sola applicazione. V F
7 Un protocollo è una parte integrante di un'applicazione ed è sviluppato all'interno di essa. V F
8 In una architettura client-server gli indirizzi IP devono essere statici. V F
9 Un socket è composto da un numero di porta di un processo e dall'indirizzo IP del terminale. V F
10 Prende il nome di indirizzo del socket un numero di porta concatenato a un indirizzo IP. V F

## ASSOCIAZIONE

**Associa le seguenti porte (7, 21, 22, 80, 110) ai servizi.**

**a** ..... HTTP
**b** ..... FTP
**c** ..... POP3
**d** ..... ECHO
**e** ..... TELNET

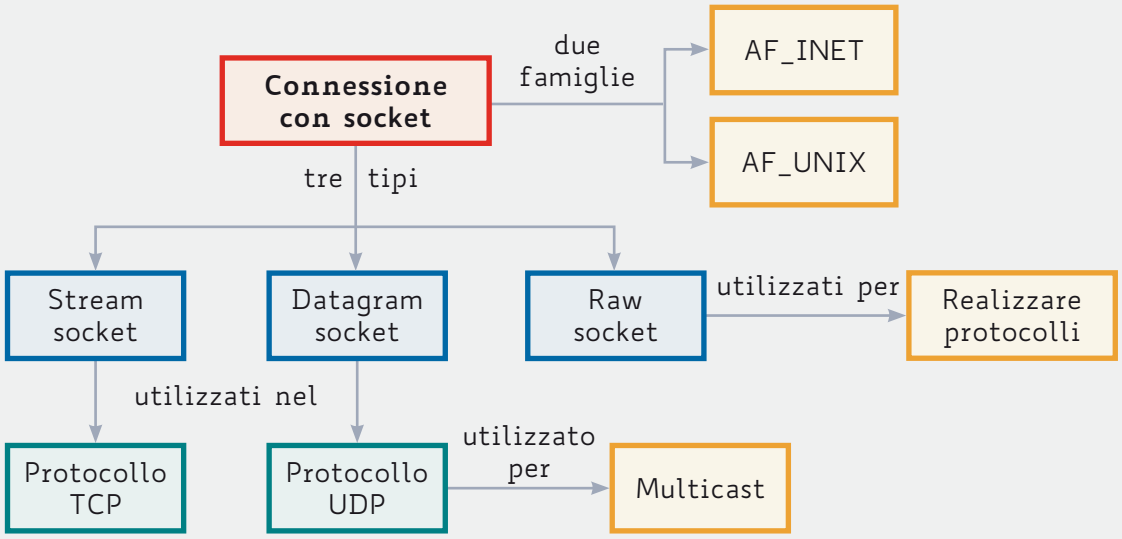
# 2 La connessione tramite socket

**IN QUESTA LEZIONE IMPAREREMO...**

- le famiglie e le tipologie di socket
- le modalità di connessione col protocollo TCP e UDP
- la comunicazione multicast

## MAPPA CONCETTUALE

didattica inclusiva




## Generalità

Il concetto di socket è stato sviluppato come estensione diretta del paradigma UNIX di I/O su file, che si basa sulla sequenza di operazioni open-read-write-close:
- open: permette di accedere a un file;
- read/write: accedono ai contenuti del file;
- close: terminazione dell'utilizzo del file.

L'utilizzo dei socket avviene pressoché con la stessa modalità ma aggiungendo a questa struttura l'insieme dei parametri necessari a realizzare la connessione tra macchine remote, cioè richiedendo:
- gli indirizzi;
- il protocollo e numero di porta;
- il tipo del protocollo.

Ogni sistema operativo mette a disposizione nelle API i meccanismi per realizzare l'interfacciamento tra diversi

protocolli: le socket API sono delle specifiche API di protocollo che hanno origine con Berkeley BSD UNIX e che oggi sono disponibili in Windows, Solaris e Linux.
Elenchiamo le funzioni presenti sia in C che in Java che verranno utilizzate in seguito:
- socket()/server socket(): crea un nuovo socket;
- close(): termina l'utilizzo di un socket;
- bind(): collega un indirizzo di rete a un socket;
- listen(): aspetta messaggi in ingresso;
- accept(): comincia a utilizzare una connessione in ingresso;
- connect(): crea una connessione con un host remoto;
- send()/write(): trasmette dati su una connessione attiva;
- recv()/read(): riceve dati da una connessione attiva.

# Famiglie e tipi di socket

## Famiglie di socket

Esistono varie famiglie di socket (chiamate anche domini) dove ogni famiglia riunisce i socket che utilizzano gli stessi protocolli (Protocol Family) sottostanti: ogni famiglia supporta un sottoinsieme di stili di comunicazione e possiede un proprio formato di indirizzamento (Address Family).
Tra le famiglie di socket ricordiamo:
- Internet socket (AF_INET): è quella che utilizzeremo nelle nostre applicazioni e permette il trasferimento di dati tra processi posti su macchine remote connesse tramite una LAN o Internet;
- Unix Domain socket (AF_UNIX): permette il trasferimento di dati tra processi sulla stessa macchina Unix.

La principale differenza tra un socket nei domini AF_UNIX e AF_INET è l'indirizzo:
- AF_UNIX è praticamente il pathname valido nel file system della macchina;
- AF_INET è specificato da due valori:
  - indirizzo IP (32 bit), che individua un unico host Internet,
  - numero di porta (16 bit), che specifica una particolare porta dell'host.

Noi tratteremo solo i socket AF_INET.

## Tipi di socket

I socket sono fondamentalmente di tre tipi e per ciascuna tipologia abbiamo una diversa modalità di connessione:
- stream socket;
- datagram socket;
- raw socket.

Gli stream socket e datagram socket sono usati a livello applicativo mentre i raw sockets sono utilizzati nello sviluppo di protocolli e quindi esulano dagli obiettivi di questa trattazione.

## Stream socket

Con gli stream socket (SOCK_STREAM) si realizza una connessione sequenziale tipicamente asimmetrica, affidabile e full-duplex basata su stream di byte di lunghezza variabile.
Operativamente, ogni processo crea il proprio endpoint richiamando la primitiva socket() in C o creando l'oggetto socket in Java e successivamente:
- il server si mette in ascolto, in attesa di un collegamento, e quando gli arriva una richiesta la esaudisce mediante la primitiva accept() che crea un nuovo socket dedicato alla connessione;
- il client si pone in coda sul socket del server e quando viene "accettato" dal server crea implicitamente il binding con la porta locale.

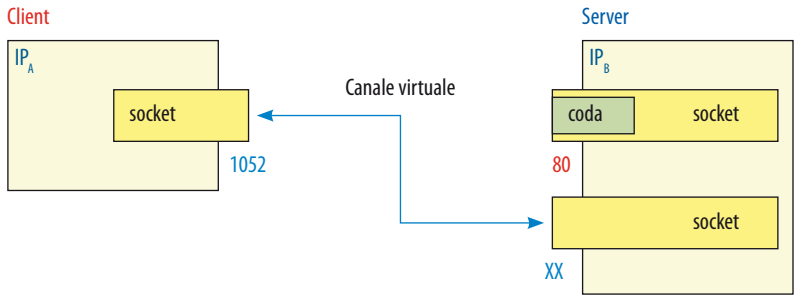
Tra i due processi server e client, il server è quello che ha controllo maggiore poiché è il processo che inizialmente crea il socket: più client possono comunicare attraverso lo stesso socket ma solo un server può essere associato a uno specifico socket.
Inoltre il client ha bisogno di conoscere l'indirizzo del server mentre il server acquisisce le informazioni del client (e, quindi, anche il suo indirizzo) solo dopo che viene stabilita la connessione.

Vediamo graficamente i singoli passaggi che realizzano la connessione tra server e client.

**A** Inizializzazione dei processi.

**B** Il client effettua la richiesta di collegamento.

**C** Il server accetta la richiesta e stabilisce il collegamento.

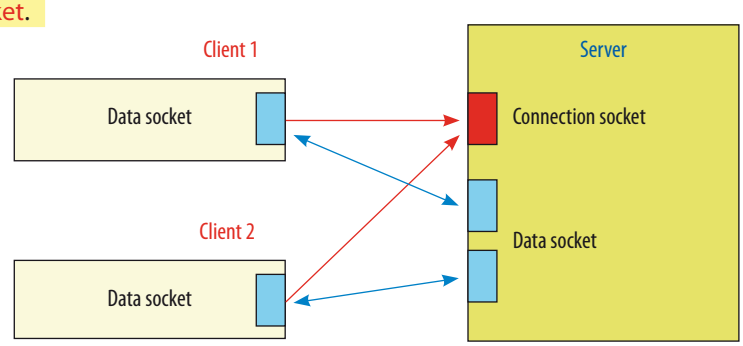



Il server accetta la richiesta e realizza "un canale virtuale" tra il client e un nuovo socket del server (indicato XX), in modo da lasciare il primo libero per le ulteriori richieste da parte di altri client.
I due processi si scambiano i dati (funzioni read() e write()) fino alla chiusura del canale, che viene effettuata rispettivamente mediante la chiamata della primitiva close().

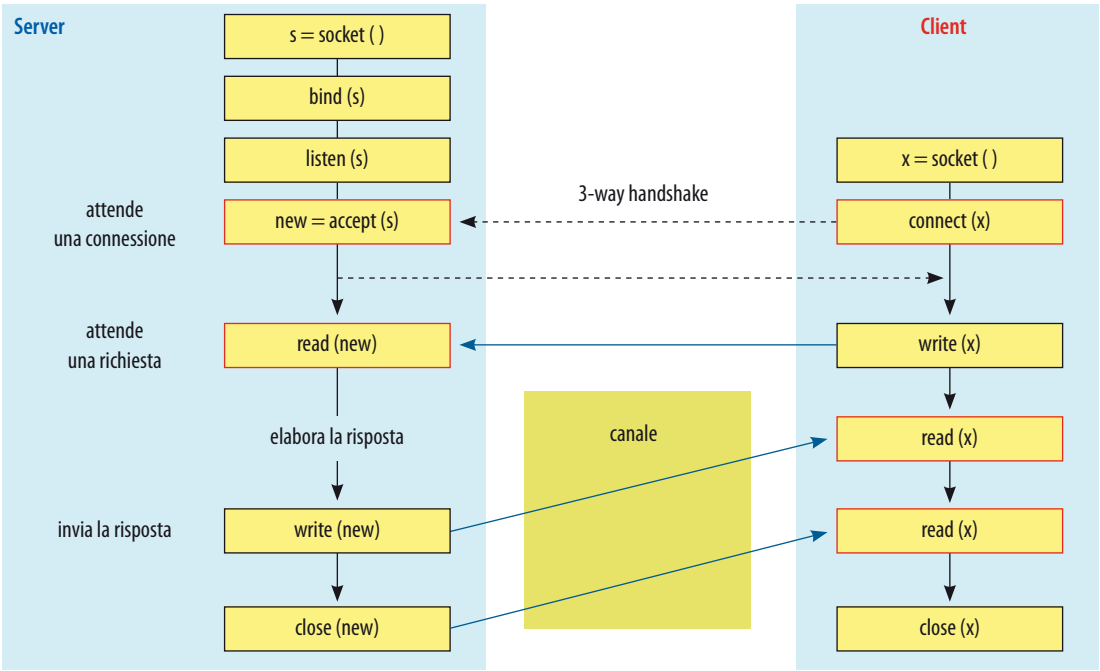
Il protocollo TCP è basato su questo tipo di socket.

Nella figura a lato possiamo osservare che in un server TCP abbiamo due tipi di socket:

- un tipo per accettare connessioni (condiviso), che chiamiamo connection socket;
- un tipo per le operazioni send() e receive() (non condiviso), che chiamiamo data socket.



Lo schema logico completo della comunicazione TCP mediante socket nel linguaggio C è il seguente:



Le API del linguaggio Java differenziano i due tipi di socket e noi utilizzeremo due entry differenti:
- ServerSocket(int port): per creare una connection socket legata alla porta specificata per accettare connessioni da parte del server;
- Socket(InetAddress address, int port): per creare una data socket e richiede come parametro una server socket, cioè l'esistenza di una connessione (l'indirizzo e la porta).

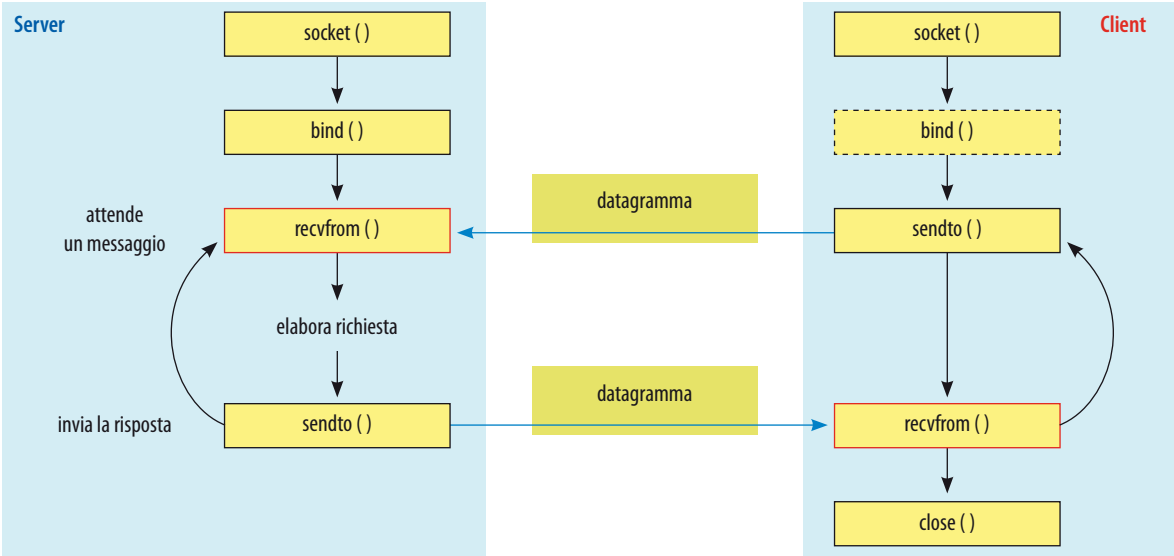
## Datagram socket

Con i datagram socket (SOCK_DGRAM) viene realizzata la comunicazione che permette di scambiare dati senza connessione (i messaggi contengono l'indirizzo di destinazione e provenienza) mediante il trasferimento di datagrammi che inoltrano messaggi di dimensione variabile, senza garantire ordine o arrivo dei pacchetti, quindi si ha una comunicazione inaffidabile.
Permettono quindi di inviare da un socket a più destinazioni e ricevere su un socket da più sorgenti: in generale realizzano quindi il modello "molti a molti" e sono supportate nel dominio Internet dal protocollo UDP.
Operativamente, ogni processo crea il proprio endpoint richiamando la primitiva che crea il socket e successivamente:
- il server si mette in attesa di ricevere i dati (mediante la primitiva receive() in Java e recvfrom() in C) e alla loro ricezione può inviare un risposta (mediante la primitiva send() in Java e sendto() in C);
- il client invia il pacchetto di dati al server (mediante la primitiva send() in Java e sendto() e in C) e può mettersi in attesa di una risposta con le stesse primitive che ha utilizzato il server (cioè mediante la primitiva receive() in Java e recvfrom() in C).

Al termine della comunicazione il socket viene chiuso.

Il loro vantaggio è quello di trasferire velocemente i dati e non esistono differenze tra le chiamate effettuate dai vari processi coinvolti nella comunicazione: quindi "decade" anche il concetto di client e server dato che utilizzano le stesse primitive e solo il loro ordine stabilisce nella applicazione quale processo fa la funzione di server e quale di client.

Lo schema logico completo della comunicazione UDP in linguaggio C mediante socket è il seguente:




## Trasmissione unicast e multicast

L'unicast è la "normale" situazione in cui un mittente invia e un destinatario riceve con eventualmente l'inversione dei ruoli (relazione one-to-one), mentre il multicast è utilizzato per trasmettere informazioni a più host contemporaneamente (relazione one-to-many).

> Il TCP è, per la sua natura di protocollo orientato alla connessione, sempre l'unicast.
> L'UDP è utilizzato normalmente a livello di trasporto e supporta più paradigmi di comunicazione anche se il più utilizzato è sempre il multicast.

Nella comunicazione di tipo multicast un insieme di processi formano un gruppo di multicast e un messaggio spedito da un processo a quel gruppo viene recapitato a tutti gli altri partecipanti appartenenti al gruppo.
Tipiche applicazioni multicast sono per esempio:

- usenet news: pubblicazione e diffusione di nuove notizie in tempo reale;
- videoconferenze: ogni host genera un segnale audio video che viene ricevuto dagli host associati ai partecipanti alla videoconferenza;
- massive multiplayer games: giochi di ruolo e Internet game dove un elevato numero di giocatori interagiscono in un mondo virtuale;
- DNS (Domain Name System): aggiornamenti delle tabelle di naming inviati a gruppi di DNS;
- altre applicazioni quali chat, instant messaging, applicazioni P2P ecc.

Per implementare un sistema multicast è necessario poter definire uno schema di indirizzamento dei gruppi e un supporto che registri la corrispondenza tra un gruppo e i partecipanti oltre alla possibilità di ottimizzare l'uso della rete nel caso di invio di pacchetti a un gruppo (tramite multicast router).
Il protocollo più utilizzato per il multicast in Internet è l'IGMP (Internet Group Management Protocol) che serve a garantire la trasmissione, tra host e multicast router a essi direttamente collegati, dei messaggi relativi alla costituzione dei gruppi: per esempio fornisce a un host i mezzi per informare il multicast router a esso più vicino che un'applicazione vuole unirsi a un determinato gruppo multicast utilizzando normali datagrammi IP.

Le API multicast devono quindi contenere primitive per:

- unirsi a un gruppo di multicast (join): identificare e indirizzare univocamente un gruppo;
- lasciare un gruppo di multicast (leave);
- spedire messaggi a un gruppo, cioè a tutti i processi che in quel momento fanno parte del gruppo;
- ricevere messaggi indirizzati a un gruppo del quale l'host fa parte.

La gestione dei gruppi è di tipo dinamico:
- un host può unirsi o abbandonare un gruppo in qualsiasi momento e può appartenere contemporaneamente a più gruppi;
- non è necessario appartenere a un gruppo per poter inviare a esso dei messaggi;
- i membri del gruppo possono appartenere alla medesima rete o a reti fisiche differenti.

Come indirizzo di multicast viene utilizzato un indirizzo IP di classe D, del tipo:

| 1 | 1 | 1 | 0 | 28 bit<br>multicast group ID |
|---|---|---|---|---|

cioè con intervallo tra 224.0.0.0 – 239-255-255-255

Gli indirizzi possono essere:
- permanenti: l'indirizzo di multicast viene assegnato dalla IANA (Internet Assigned Numbers Authority) e rimane assegnato a quel gruppo anche se in un certo momento non ci sono più partecipanti connessi; questi indirizzi sono detti well-known;
- temporanei: richiedono la definizione di un opportuno protocollo per evitare conflitti nell'attribuzione degli indirizzi ai gruppi ed esistono solo fino al momento in cui esiste almeno un partecipante.

Nei router viene mantenuta la corrispondenza tra l'indirizzo di multicast e gli indirizzi IP dei singoli host che partecipano al gruppo di multicast.

La comunicazione multicast utilizza il paradigma connectionless dato che devono essere gestite contemporaneamente un alto numero di connessioni.
È anche possibile associare un nome simbolico a un gruppo di multicast: nella schermata a fianco, ottenuta all'indirizzo http://ip-lookup.net digitando come IP address **240.0.0.1**, è possibile vedere il nome di una parte di questi gruppi.

| | IP address | Host name |
|---|---|---|
| 1 | 224.0.0.0 | base-address.mcast.net |
| | **224.0.0.1** | all-systems.mcast.net |
| 1 | 224.0.0.2 | all-routers.mcast.net |
| 2 | 224.0.0.3 | ? |
| 3 | 224.0.0.4 | dvmrp.mcast.net |
| 4 | 224.0.0.5 | ospf-all.mcast.net |
| 5 | 224.0.0.6 | ospf-dsig.mcast.net |
| 6 | 224.0.0.7 | st-routers.mcast.net |
| 7 | 224.0.0.8 | st-hosts.mcast.net |
| 8 | 224.0.0.9 | rip2-routers.mcast.net |
| 9 | 224.0.0.10 | igrp-routers.mcast.net |
| 10 | 224.0.0.11 | mobile-agents.mcast.net |
| 11 | 224.0.0.12 | dhcp-agents.mcast.net |
| 12 | 224.0.0.13 | pim-routers.mcast.net |
| 13 | 224.0.0.14 | rsvp-encapsulation.mcast.net |

Il broadcast può essere considerato come un caso estremo di multicast dove la comunicazione avviene da uno a tutti (relazione one-to-all) e per la sua realizzazione viene normalmente usato un indirizzo speciale per identificare tutte le possibili destinazioni (indirizzo broadcast)

## ESEMPIO

Java, come vedremo, mette a disposizione un servizio multicast nelle API standard (la classe MulticastSocket) che estende la DatagramSocket con alcune funzionalità utili per il multicast e permette di definire socket su cui spedire/ricevere i messaggi verso/da un gruppo di multicast.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Quali sono le operazioni del paradigma UNIX di I/O su file?**
**a** open-send-write-close
**b** open-recv-read -close
**c** open-read-write-close
**d** open-send- receive-close
**e** open-accept- connect-close

**2 L'utilizzo dei socket richiede l'insieme dei parametri necessari a realizzare la connessione, ovvero:**
**a** gli indirizzi
**b** il protocollo e numero di porta
**c** il tipo del protocollo
**d** il tipo di sistema operativo
**e** il tipo di processore

**3 Possiamo individuare tre tipi fondamentali di modalità di connessione mediante socket:**
**a** stream socket
**b** datagram socket
**c** packet socket
**d** raw socket

**4 Con gli stream socket (SOCK_STREAM) si realizza una connessione:** (*indica quella errata*)
**a** sequenziale
**b** asimmetrica
**c** affidabile
**d** full-duplex
**e** half-duplex
**f** con stream di byte di lunghezza fissa

**5 In un server TCP abbiamo due tipi di socket:**
**a** connection socket
**b** stream socket
**c** packet socket
**d** data socket
**e** raw socket

**6 Con i datagram socket (SOCK_DGRAM) si realizza una connessione:** (*Indica quelli errati*)
**a** molti a molti
**b** affidabile
**c** full-duplex
**d** con stream di byte di lunghezza fissa
**e** senza connessione

**7 Tipiche applicazioni multicast sono, per esempio:** (*indica quella errata*)
**a** usenet news
**b** videoconferenze
**c** pop mail
**d** massive multiplayer games
**e** DNS (Domain Name System)
**f** chat e instant messaging

**8 Le API multicast devono quindi contenere primitive per:**
**a** unirsi a un gruppo di multicast
**b** formare un gruppo di multicast
**c** lasciare un gruppo di multicast
**d** spedire messaggi a un gruppo
**e** ricevere messaggi indirizzati a un gruppo

## VERO/FALSO

1 Le famiglie dei socket sono anche chiamate domini V F
2 La funzione bind collega un indirizzo di rete a un socket. V F
3 AF_UNIX è praticamente il pathname valido nel file system della macchina. V F
4 AF_INET è specificato da due valori, indirizzo IP (32 bit) e numero di porta (8 bit). V F
5 Il linguaggio Java di default utilizza il dominio AF_INET nella versione v4. V F
6 Il server si mette in ascolto mediante la primitiva di receive V F
7 Gli stream e datagram socket sono usati a livello applicativo. V F
8 Il protocollo TCP è basato su stream socket. V F
9 Con i datagram socket viene garantito l'ordine di arrivo dei pacchetti. V F
10 Nella comunicazione di tipo multicast un messaggio spedito da un processo a quel gruppo viene recapitato a tutti gli altri partecipanti appartenenti al gruppo. V F
11 La comunicazione di tipo multicast coinvolge un gruppo di host. V F
12 Come indirizzo di multicast viene utilizzato un indirizzo IP di classe E. V F

# 1 ESERCIZI IN LABORATORIO

## JAVA SOCKET

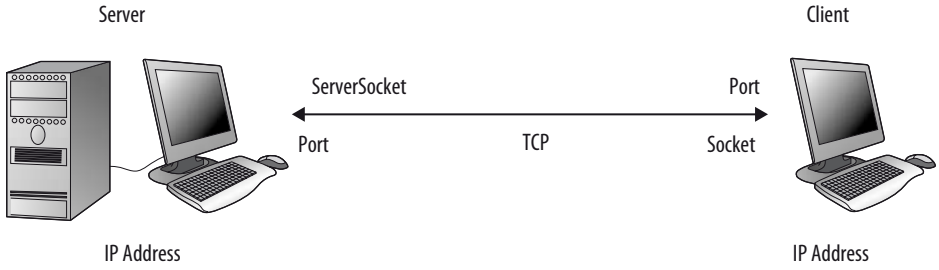
### Java socket: caratteristiche della comunicazione

Si è detto che un socket è come una porta di comunicazione e tutto ciò che è in grado di comunicare tramite il protocollo standard TCP/IP può collegarsi a un socket e comunicare attraverso di esso; sappiamo anche che le informazioni "spedite" da un socket a un altro prendono il nome di pacchetti TCP/IP.
Riassumiamo le caratteristiche fondamentali della comunicazione:

- tipicamente, la comunicazione è punto-punto (unicast), ma esistono estensioni che permettono comunicazioni multi-punto (multicast);
- la sorgente della comunicazione deve conoscere l'identità (indirizzo IP e numero di porta) del destinatario;
- la serializzazione (trasformazione tra dati strutturati e sequenze di byte) e il marshalling (conversione tra rappresentazioni di dati diversi) sono a carico delle applicazioni.

Le operazioni necessarie per un trasferimento di dati tra due host sono le seguenti:

1. un host che si comporta da server apre un canale di comunicazione su una determinata porta e rimane in ascolto, in attesa di una richiesta di connessione (ServerSocket);
2. un host client fa una richiesta di connessione a un server (con indirizzo IP e address port conosciuti: socket sul client);
3. il server accetta la connessione del client e così viene instaurato un canale di comunicazione tra i due host.




Java utilizza la classe Socket per la creazione degli oggetti che permettono di utilizzare i socket e quindi di stabilire un canale di comunicazione tra un client e un server attraverso il quale si comunica utilizzando due stream particolari, specializzati per i socket, uno per l'input e l'altro per l'output.
Vediamo dapprima singolarmente le classi che Java mette a disposizione per l'utilizzo dei socket e successivamente implementiamo un server e un client:

- classe InetAddress;
- classe ServerSocket;
- classe Socket.

### Classe InetAddress

Un indirizzo di rete IP v4 è costituito da 4 numeri (da 0 a 255) separati ciascuno da un punto e il DNS permette di utilizzare in alternativa il nome simbolico per individuare un particolare host.
La classe InetAddress mette a disposizione diversi metodi per astrarre dal particolare tipo di indirizzo specificato (a numeri o a lettere), occupandosi essa stessa di effettuare le dovute traduzioni.

# ESERCIZI IN LABORATORIO 1

Il diagramma completo della classe è il seguente:

| Classe InetAddress | |
|---|---|
| **Metodi modificatori** | |
| `InetAddress[] getAllByName(String host)`<br>`InetAddress getByAddress(byte[] addr)`<br>`InetAddress getByAddress(String host, byte[] addr)`<br>`InetAddress getByName(String host)`<br>`InetAddress getLocalHost()` | `boolean equals(Object obj)`<br>`int hashCode()`<br>`String toString()` |
| `byte[] getAddress()`<br>`String getCanonicalHostName()`<br>`String getHostAddress()`<br>`String getHostByAddr(byte[] addr)`<br>`boolean isAnyLocalAddress()`<br>`boolean isLinkLocalAddress()`<br>`boolean isLoopbackAddress()`<br>`boolean isMCGlobal()`<br>`boolean isMCLinkLocal()`<br>`boolean isMCNodeLocal()`<br>`boolean isMCOrgLocal()`<br>`boolean isMCSiteLocal()`<br>`boolean isMulticastAddress()`<br>`boolean isSiteLocalAddress()` | `byte[][] lookupAllHostAddr(String host)`<br>`Object run()` |

Non sono previsti costruttori e l'unico modo per creare un oggetto di classe InetAddress prevede l'utilizzo di *metodi statici*, e in particolare:

- `public static InetAddress getByName(String host)`
  restituisce un oggetto InetAddress rappresentante l'host specificato nel parametro host; l'host può essere specificato sia con il nome sia con l'indirizzo IP numerico e se si specifica null come parametro, ci si riferisce all'indirizzo di default della macchina locale;
- `public static InetAddress getLocalHost()`
  viene restituito un InetAddress corrispondente alla macchina locale; se tale macchina non è registrata, oppure è protetta da un firewall, l'indirizzo è quello di loopback: 127.0.0.1.

Se l'indirizzo specificato non può essere risolto tramite il DNS; questi metodi possono sollevare l'eccezione UnknownHostException.

Di particolare utilità sono i tre metodi illustrati di seguito:

1. `public String getHostName()`
   restituisce il nome dell'host che corrisponde all'indirizzo IP dell'InetAddress; se il nome non è ancora noto (per esempio se l'oggetto è stato creato specificando un indirizzo IP numerico), verrà cercato tramite il DNS; se tale ricerca fallisce, verrà restituito l'indirizzo IP numerico sotto forma di stringa.
2. `public String getHostAddress()`
   simile al precedente, restituisce però l'indirizzo IP numerico, sotto forma di stringa, corrispondente all'oggetto InetAddress.
3. `public byte[] getAddress()`
   l'indirizzo IP numerico restituito sarà sotto forma di matrice di byte; l'ordinamento dei byte è high byte first (che è l'ordinamento tipico della rete).

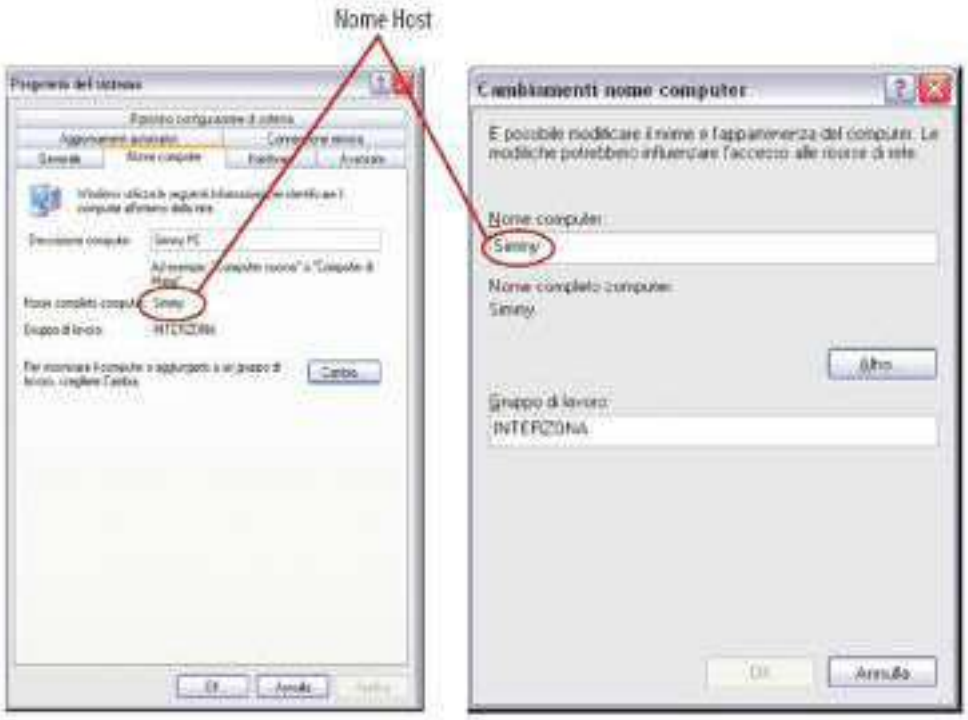
Per esempio, con:

```
String indirizzo = InetAddress.getLocalHost().getHostAddress();
```

l'oggetto *indirizzo* conterrà l'indirizzo IP della macchina locale.

# 1 ESERCIZI IN LABORATORIO

La figura che segue rappresenta un esempio del nome dell'host riconosciuto dalla classe InetAddress e dal DNS.




## Classe ServerSocket

La classe ServerSocket, il cui diagramma è il seguente, viene utilizzata per accettare connessioni da parte del client.

| Classe ServerSocket | |
|---|---|
| **Metodi costruttori** | **Metodi modificatori** |
| `ServerSocket()`<br>`ServerSocket(int port)`<br>`ServerSocket(int port, int backlog)`<br>`ServerSocket(int port, int backlog, InetAddress bindAddr)` | `void setSocketFactory(SocketImplFactory fac)` |
| | `ServerSocketChannel getChannel()`<br>`InetAddress getInetAddress()`<br>`int getLocalPort()`<br>`SocketAddress getLocalSocketAddress()`<br>`int get/setReceiveAddress()`<br>`boolean get/setReuseAddress()`<br>`int get/setSoTimeout()`<br>`boolean isBound()`<br>`boolean isClosed()` |
| | `String toString()` |
| | `Socket accept()`<br>`void bind(SocketAddress endpoint)`<br>`void bind(SocketAddress endpoint, int backlog)`<br>`void close()`<br>`void implAccept(Socket s)` |

Questa classe deve essere istanziata passando come parametro il numero della porta su cui il server sarà in ascolto: ServerSocket(int port). L'unico metodo realmente necessario è accept(): mediante tale metodo l'oggetto rimane in attesa di richiesta di connessioni da parte di un client sulla porta specificata nel costruttore.
Quando una richiesta di connessione va a buon fine viene creato il canale di collegamento e il metodo restituisce un oggetto Socket connesso con il client.

# ESERCIZI IN LABORATORIO 1

Per esempio:

```
// crea un server sulla porta 6789
ServerSocket server = new ServerSocket(6789);
// rimane in attesa di un client
Socket client = server.accept();
// chiudo il server per inibire altri client
server.close();
```

Vediamo a lato la chiosa delle singole istruzioni:

- il server si mette in ascolto sulla porta 6789;
- quando riceve una richiesta di connessione, viene creato un oggetto client di classe Socket (che vedremo in seguito); questo oggetto rappresenta il canale di comunicazione server-client;
- con la terza istruzione si chiude il server: ciò implica una comunicazione di tipo Unicast, dato che, dopo avere instaurato una connessione, il server non rimane più in ascolto.

Per poter utilizzare una connessione multicast (che realizzeremo più avanti), si dovrebbe istanziare un thread per ogni connessione effettuata, in modo da lasciare il server sempre in ascolto.

## Classe Socket

La classe Socket permette di definire una connessione client-server via TCP su entrambi i lati, sia client che server.
La differenza tra client e server sta nella modalità di creazione di un oggetto di questo tipo:

- nel server l'oggetto Socket viene creato dal metodo accept() della classe ServerSocket;
- il client provvede a creare un'istanza di Socket: per creare un socket con un server in esecuzione su un dato host basta creare un oggetto di classe Socket specificando nel costruttore l'*indirizzo Internet dell'host* e il *numero di porta*.

Dopo che l'oggetto Socket è stato istanziato, è possibile ottenere (con appositi metodi) due stream (uno di input e uno di output): con essi è possibile comunicare con l'host, e riceverne messaggi. Il diagramma della classe Socket è il seguente:

| Classe Socket | |
|---|---|
| **Metodi costruttori** | **Metodi modificatori** |
| `Socket()`<br>`Socket(SocketImpl port)`<br>`Socket(String host, int port)`<br>`Socket(InetAddress address, int port)`<br>`Socket(String host, int port, InetAddress localAddr, int localPort)`<br>`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)` | `void setSocketImplFactory(SocketImplFactory fac)` |
| | `SocketChannel getChannel()`<br>`InetAddress getInetAddress()`<br>`InputStrem getInputStream()`<br>`boolean get/setKeepAlive()`<br>`InetAddress getLocalAddress()`<br>`int getLocalPort()`<br>`SocketAddress getLocalSocketAddress()`<br>`boolean get/setOOBInline()`<br>`OutputStream getOutputStream()`<br>`int getPort()`<br>`int get/setReceiveBufferSize()`<br>`SocketAddress get/getRemoteSocketAddress()`<br>`boolean get/setReuseAddress()`<br>`int get/setSendBufferSize()`<br>`int getSolLinger()`<br>`int get/setSoTimeout()`<br>`boolean get/setTopNoDelay()`<br>`int get/setTrafficClass()`<br>`boolean isBound()`<br>`boolean isClosed()`<br>`boolean isConnected()`<br>`boolean isInputShutDown() / isOutputShutDown()`<br>`void setSoLinger(boolean on, int linger)` |
| | `String toString()` |
| | `void bind(SocketAddress bindpoint)`<br>`void close()`<br>`void connect(SocketAddress endpoint)`<br>`void sendUrgentData(int data)`<br>`void shutdownInput() / shutdownOutput()` |

# 1 ESERCIZI IN LABORATORIO

Qualsiasi metodo che prenda in ingresso un InputStream o un OutputStream può comunicare con l'host in rete: una volta creato il Socket si può comunicare in rete tramite l'utilizzo degli stream.
I costruttori della classe Socket sono i seguenti:

- `public Socket (String host, int port) throws IOException;`
- `public Socket (InetAddress address, int port) throws IOException.`

Viene creato un oggetto Socket connettendosi con l'host specificato (sotto forma di stringa o di InetAddress) alla porta specificata. Se sull'host e sulla porta specificata non c'è un server in ascolto, viene generata un'IOException e viene specificato il messaggio connection refused.
Alcuni metodi tra i più utilizzati sono i seguenti:

- `public InetAddress getInetAddress()`
  restituisce un oggetto Inet-Address corrispondente all'indirizzo dell'host con il quale il socket è connesso;
- `public InetAddress getLocalAddress()`
  restituisce un oggetto Inet-Address corrispondente all'indirizzo locale al quale il socket è collegato;
- `static InetAddress getByName(String hostname)`
  restituisce una istanza di InetAddress rappresentante l'host specificato;
- `public int getPort()`
  restituisce il numero della porta dell'host remoto con il quale il socket è collegato;
- `public int getLocalPort()`
  restituisce il numero di porta locale con la quale il socket è collegato.

Quando si crea un socket, come già detto, ci si collega con un server su un determinato host, che è in ascolto su una certa porta; sulla macchina locale viene creato il socket su una determinata porta assegnata dal sistema operativo (il primo numero libero):

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Quando si comunica attraverso connessioni TCP, i dati vengono suddivisi in pacchetti (IP packet), quindi è consigliabile utilizzare degli stream "bufferizzati" evitando così di avere pacchetti contenenti poche informazioni.
La definizione dei due stream, rispettivamente di input e di output, da parte di un server verso un client, è per esempio la seguente:

```
BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
DataOutputStream out = new DataOutputStream(client.getOutputStream());
```

Dopo questa definizione l'uso dei socket diventa quindi trasparente: su questi oggetti si utilizzano i metodi readLine() e println()!

L'ultimo metodo che descriviamo è:

```
public synchronized void close() throws IOException
```

Con questo metodo viene chiuso il socket (e quindi la connessione) e tutte le risorse che erano in uso vengono rilasciate.
I dati bufferizzati verranno comunque spediti prima della chiusura del socket, chiusura anche solo di uno dei due stream associati ai socket che comporterà automaticamente la chiusura del socket stesso.

# ESERCIZI IN LABORATORIO 1

Nei metodi precedenti può essere lanciata un'IOException, a significare che ci sono stati problemi sulla connessione: questo succede quando uno dei due programmi che utilizza il socket chiude la connessione e quindi l'altro programma potrà ricevere una tale eccezione.

## La realizzazione di un client TCP in Java

Possiamo ora realizzare un client: per comunicare con un host remoto usando il protocollo TCP/IP, si deve creare, per prima cosa, un oggetto Socket con tale host, specificando l'indirizzo IP dell'host e il numero di porta (naturalmente sull'host remoto dovrà essere presente un server in "ascolto" su tale porta).
Possiamo utilizzare indistintamente uno dei due costruttori della classe Socket sopra descritta:

```
public Socket (String host, int port) throws IOException;
public Socket (InetAddress address, int port) throws IOException.
```

Con questa istruzione viene creata una connessione con un'applicazione server (che nel frattempo è in attesa in una accept()) e restituisce il relativo socket.
Definiamo le variabili della nostra classe:

```
import java.io.*;
import java.net.*;

public class Client2 {
  String nomeServer ="nomeServer";      // indirizzo del server
  int portaServer   = 6789;             // porta a servizio
  DataInputStream in;                   // stream di input
  DataOutputStream out;                 // stream di output
```

Realizziamo ora la parte di un metodo che esegue le operazioni per aprire un socket con un server che si trova a un certo indirizzo (nomeServer) ed è in ascolto su una certa porta (portaServer) restituendo come parametro di ritorno un oggetto della classe Socket.
Una volta creato un oggetto Socket otteniamo gli stream a esso associati tramite i due metodi prima descritti della classe Socket: a questo punto, la comunicazione può avere inizio, cioè il client può scrivere sull'OutputStream, come si fa con un normale stream, e ricevere dati dal server leggendo dall'InputStream.

```
protected Socket connetti () throws IOException
{
  Socket socket = new Socket (nomeServer, portaServer);

  out = new DataOutputStream (socket.getOutputStream ());
  in = new DataInputStream (socket.getInputStream ());
  . . .
  return socket;
}
```

L'esempio seguente permette di leggere una stringa di testo dalla tastiera di un client. Esso invia la stringa a un server, il quale la modifica trasformandola in maiuscolo e la restituisce al client.
La comunicazione avviene sulla porta 6789 sul server locale localhost.

Naturalmente, non possiamo mandarlo in esecuzione senza aver realizzato il server, ma per ora lo scriviamo e lo commentiamo, realizzando il server successivamente.

# 1 ESERCIZI IN LABORATORIO

Definiamo le variabili da utilizzare nella classe.

```
import java.io.*;
import java.net.*;
public class ClientStr {
  String nomeServer ="localhost";                  // indirizzo server locale
  int portaServer   = 6789;                        // porta x servizio data e ora
  Socket miosocket;
  BufferedReader tastiera;                         // buffer per l'input da tastiera
  String stringaUtente;                            // stringa inserita da utente
  String stringaRicevutaDalServer;                 // stringa ricevuta dal server
  DataOutputStream outVersoServer;                 // stream di output
  BufferedReader inDalServer;                      // stream di input
```

Quindi definiamo il metodo che stabilisce la connessione con il server:

```
  public Socket connetti(){
    System.out.println("I CLIENT partito in esecuzione ...");
    try
    {
      // per l'input da tastiera
      tastiera = new BufferedReader(new InputStreamReader(System.in));
      // creo un socket
      miosocket = new Socket(nomeServer,portaServer);
      // miosocket = new Socket(InetAddress.getLocalHost(), 6789);
      // associo due oggetti al socket per effettuare la scrittura e la lettura
      outVersoServer = new DataOutputStream(miosocket.getOutputStream());
      inDalServer    = new BufferedReader(new InputStreamReader (miosocket.getInputStream()));
    }
    catch (UnknownHostException e){
        System.err.println("Host sconosciuto"); }
    catch (Exception e)
    {
      System.out.println(e.getMessage());
      System.out.println("Errore durante la connessione!");
      System.exit(1);
    }
    return miosocket;
  }
```

Abbiamo creato un socket mediante il costruttore:

```
miosocket = new Socket(nomeserver,portaserver);
```

Trattandosi di server locale avremmo anche potuto utilizzare la seguente istruzione:

```
miosocket = new Socket(InetAddress.getLocalHost(), 6789);
```

Per le comunicazioni I/O abbiamo definito tre oggetti, il primo della classe BufferedReader per effettuare l'input da tastiera e gli altri due, rispettivamente, della classe DataOutputStream e BufferedReader per scrivere e leggere sul socket. Definiamo un secondo metodo che effettua la conversazione:

- leggiamo la stringa inserita dall'utente e la scriviamo sullo stream del miosocket: quindi ci mettiamo in attesa della risposta del server mediante la inDalserver.readLine();
- quando riceviamo la risposta la visualizziamo sullo schermo e terminiamo l'elaborazione chiudendo la porta con miosocket.close().

# ESERCIZI IN LABORATORIO 1

```
public void comunica() {
  try                          // leggo una riga
  {
    System.out.println("4 ...  inserisci la stringa da trasmettere al server:"+'\n');
    stringaUtente = tastiera.readLine();
    //la spedisco al server
    System.out.println("5 ... invio la stringa al server e attendo ...");
    outVersoServer.writeBytes( stringaUtente+'\n');
    //leggo la risposta dal server
    stringaRicevutaDalServer=inDalServer.readLine();
    System.out.println("8 ... risposta dal server "+'\n'+stringaRicevutaDalServer );
    // chiudo la connessione
    System.out.println("9 CLIENT: termina elaborazione e chiude connessione" );
    miosocket.close();
  }
  catch (Exception e)
  {
    System.out.println(e.getMessage());
    System.out.println("Errore durante la comunicazione col server!");
    System.exit(1);
  }
}
```

Concludono il metodo le istruzioni del costrutto catch che ci segnalano gli eventuali errori.
Il main() è il seguente:

```
public static void main(String args[]) {
  ClientStr cliente = new ClientStr();
  cliente.connetti();
  cliente.comunica();
}
```

## METTITI ALLA PROVA

- Realizzare un client TCP in Java

Sulla base del client sopra descritto, scrivi un tuo client predisposto per connettersi a un server sempre presente sul tuo PC che trasmette e riceve una stringa.
La verifica della connessione verrà fatta al termine del successivo Esercizio in Laboratorio oppure utilizzando come server ServerStr.class che puoi scaricare dal sito www.hoepliscuola.it nella cartella MATERIALI nella sezione riservata al presente volume (file UDA2vol3java.rar).

# 2 ESERCIZI IN LABORATORIO

## JAVA SOCKET: REALIZZAZIONE DI UN SERVER TCP

### Realizzazione di un server

In questa esercitazione realizzeremo il server corrispondente al client definito nella precedente Esercizio in Laboratorio, dopo aver definito le variabili:

```
1  import java.io.*;
2  import java.net.*;
3  import java.util.*;
4
5  public class ServerStr
6  {
7    ServerSocket server      = null;
8    Socket client            = null;
9    String stringaRicevuta   = null;
10   String stringaModificata = null;
11   BufferedReader   inDalClient;
12   DataOutputStream outVersoClient;
```

Definiamo un metodo che effettua la connessione: per prima cosa si deve creare un oggetto ServerSocket che indichi il numero di porta sulla quale si mette in ascolto (nel nostro esempio, 6789) e mediante il metodo accept() aspetta un client.

```
14   public Socket attendi()
15   {
16     try
17     {
18       System.out.println("I SERVER partito in esecuzione ...");
19       // crea un server sulla porta 6789
20       server = new ServerSocket(6789);
21       // rimane in attesa di un client
22       client = server.accept();
23       // chiude il server per inibire altri client
24       server.close();
25       //associa due oggetti al socket del client per effettuare la scrittura e la lettura
26       inDalClient = new BufferedReader(new InputStreamReader (client.getInputStream()));
27       outVersoClient = new DataOutputStream(client.getOutputStream());
28     }
```

L'esecuzione dell'istruzione successiva avviene solamente se si è instaurata la connessione col client su tale porta: prima di iniziare la comunicazione, il server deve chiudere il ServerSocket per inibire ad altri client il collegamento, in quanto quell'indirizzo è già utilizzato (e ora noi stiamo implementando una comunicazione Unicast).
Conclude il metodo la gestione dell'errore di connessione:

```
29     catch (Exception e)
30     {
31       System.out.println(e.getMessage());
32       System.out.println("Errore durante l'istanza del server !");
33       System.exit(1);
34     }
35     return client;
36   }
```

# ESERCIZI IN LABORATORIO 2

Il metodo che effettua la comunicazione è molto simile a quello descritto per il client: dopo il saluto di benvenuto, il server si pone in attesa di lettura di una stringa dal canale di "input dal client"; al suo arrivo, la converte in maiuscolo e la trasmette al client.
Quindi si commiata dal client e chiude la connessione terminando la sua elaborazione.

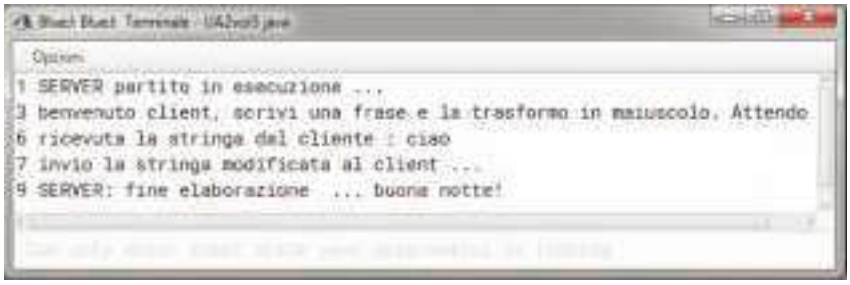
```
public void comunica()
{
  try
  {
    // rimango in attesa della riga trasmessa dal client
    System.out.println("3 benvenuto client, scrivi una frase e la trasformo in maiuscolo. Attendo ...");
    stringaRicevuta = inDalClient.readLine();
    System.out.println("6 ricevuta la stringa dal cliente : "+stringaRicevuta);

    //la modifico e la rispedisco al client
    stringaModificata=stringaRicevuta.toUpperCase();
    System.out.println("7 invio la stringa modificata al client ...");
    outVersoClient.writeBytes(stringaModificata+'\n');

    //termina elaborazione sul server : chiudo la connessione del client
    System.out.println("9 SERVER: fine elaborazione  ... buona notte!");
    client.close();
  }
```

Il main() è il seguente:

```
public static void main(String args[]) {
  ServerStr servente = new ServerStr();
  servente.attendi ();
  servente.comunica();
}
```

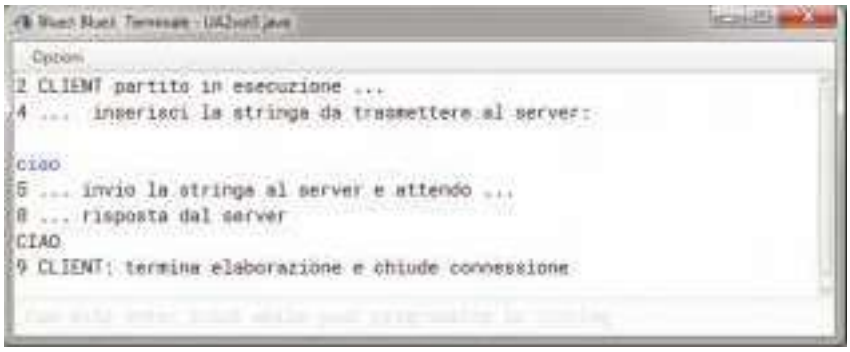
Mandiamo ora in esecuzione le due classi: dato che utilizziamo la medesima macchina per entrambe le funzioni, dovremo attivare due istanze di BlueJ: una che ci permetterà di mandare in esecuzione il server e l'altra per il client.
L'output si presenta in due finestre diverse: per meglio comprendere la corretta sequenza di esecuzione delle operazioni queste sono state numerate in ordine progressivo. Ricordiamo di mandare in esecuzione prima il server altrimenti il client termina immediatamente con il seguente messaggio:

Effettuiamo una sequenza completa di operazioni, digitando come stringa da tradurre "ciao server".



# 2 ESERCIZI IN LABORATORIO

Nella finestra precedente sono riportate tutte le operazioni effettuate dal server e nella finestra seguente dal client.




Il codice completo è nel file ServerSrt.java e ClienStr.java.

**AREA DIGITALE**

**Un esempio di verifica**

## METTITI ALLA PROVA

Realizza un'applicazione client-server dove il client invia un messaggio al server, questo conta il numero di vocali e di consonanti, e gli ritorna tali valori: il client continua a inviare frasi fino a che il numero di consonanti sia esattamente la metà del numero di vocali.
In questo caso termina l'applicazione.

## Esercizi proposti

1 Realizza una semplice calcolatrice: il client invia gli operandi e l'operatore al server, il quale esegue l'operazione e restituisce il risultato.
2 Realizza un sistema in cui il client riceve dal server un numero progressivo (tipo dispenser dei numeri per la coda dal panettiere).
3 Bomba a orologeria: un server spedisce a un client una bomba innescata con una miccia con valore random() e il client la rispedisce successivamente al server; a ogni operazione, entrambi riducono la miccia finché… la bomba scoppia.
4 Realizza il noto gioco della battaglia navale tra due giocatori che si trovano su due PC connessi in rete: il server provvede alle operazioni di avvio del gioco per poi diventare a tutti gli effetti il secondo giocatore.
5 Realizza il gioco dello "spara all'orso": un orso si muove orizzontalmente in modo casuale e un cacciatore gli spara spostandosi anch'esso orizzontalmente (il cacciatore è mosso dal giocatore mediante i tasti freccia).
6 Gioco del Tris: realizza il gioco del Tris tra due giocatori connessi in rete locale.

# ESERCIZI IN LABORATORIO 3

## REALIZZAZIONE DI UN SERVER MULTIPLO IN JAVA

Nella definizione del server nell'Esercizio in Laboratorio precedente abbiamo visto che all'inizio della comunicazione con il client viene chiusa la porta sulla quale il server era in attesa (cioè il ServerSocket, inizializzato sull'indirizzo **6789**): infatti, tale server deve essere fermato per inibire ad altri client il collegamento contemporaneo sul medesimo indirizzo:

```
System.out.println("1 SERVER partito in esecuzione ...");
// creo un server sulla porta 6789
server = new ServerSocket(6789);
// rimane in attesa di un client
client = server.accept();
// chiudo il server per inibire altri client
server.close();
```

Spesso al server devono però connettersi più client contemporaneamente, a volte anche in numero non predeterminato (per esempio in una chat tra un server e un molteplice numero di client). La realizzazione di questo sistema avviene mediante un meccanismo multithread per la gestione della ricezione e dell'invio dei messaggi, in quanto non è prevedibile il numero di client da servire e il momento in cui ogni singolo client fa richiesta di invio di un nuovo messaggio.
I server multithreaded vengono tipicamente realizzati seguendo uno dei seguenti schemi:

- on demand: viene creato un thread per ogni client al momento della richiesta di connessione e viene terminato alla chiusura della comunicazione;
- thread pool: viene predeterminato e avviato un numero fisso di thread e ciascuno di essi viene assegnato alla richiesta di connessione da parte dei client; alla chiusura di una connessione il corrispondente thread non viene terminato ma viene sospeso per essere riutilizzato in una nuova connessione.

La scelta dello schema più consono varia in funzione delle situazioni, che quindi vanno accuratamente analizzate prima dell'implementazione. Infatti, ciascuno degli schemi sopra descritti presenta pregi e difetti; per esempio: il primo schema è più semplice ma più costoso (in termini di tempo, in quanto ogni volta deve essere creato un thread) mentre il secondo richiede un dimensionamento preventivo del pool (che potrebbe essere errato, o sottostimato e sovradimensionato).

Una soluzione ottimale può essere ottenuta con una forma intermedia, ovvero adottando uno schema misto. Modifichiamo ora l'esempio precedente per realizzare una situazione on demand: riscriviamo quindi il server.

Definiamo la classe con le necessarie variabili.

```
1 import java.net.*;
2 import java.io.*;
3 class ServerThread extends Thread {
4   ServerSocket server      = null;
5   Socket client            = null;
6   String stringaRicevuta   = null;
7   String stringaModificata = null;
8   BufferedReader   inDalClient;
9   DataOutputStream outVersoClient;
```

# 3 ESERCIZI IN LABORATORIO

Modifichiamo il costruttore in questo modo:

```
public ServerThread (Socket socket){
  this.client = socket;
}
```

Anche il metodo run() viene limitato alla chiamata di un nuovo metodo comunica() e alla gestione delle eccezioni:

```
public void run(){
try{
  comunica();
}catch (Exception e){
  e.printStackTrace(System.out);  }
}
```

Il metodo che effettua la comunicazione è costituito da un ciclo infinito dal quale si esce quando il client ha trasmesso la stringa FINE: per il resto, si limita ad acquisire una riga dallo stream di input e ritrasmetterla al client.

La numerazione dei messaggi aiuta a seguire il flusso della comunicazione.

```
public void comunica ()throws Exception{
  inDalClient      = new BufferedReader(new InputStreamReader (client.getInputStream()));
  outVersoClient   = new DataOutputStream(client.getOutputStream());
  for (;;){
    stringaRicevuta = inDalClient.readLine();
    if (stringaRicevuta == null || stringaRicevuta.equals("FINE")){
      outVersoClient.writeBytes(stringaRicevuta+" (->server in chiusura...)" + '\n');
      System.out.println("Echo sul server in chiusura  :" + stringaRicevuta);
      break;
    }
    else{
      outVersoClient.writeBytes(stringaRicevuta+" (ricevuta e ritrasmessa)" + '\n');
      System.out.println("6 Echo sul server :" + stringaRicevuta);
    }
  }
  outVersoClient.close();
  inDalClient.close();
  System.out.println("9 Chiusura socket" + client);
  client.close();
}
}
```

Realizziamo ora la classe più importante, cioè quella che rimane in attesa che un client si connetta alla sua porta (come nell'esempio precedente la `6789`), mediante la:

```
// rimane in attesa di un client
client = server.accept();
```

Al momento della connessione, facciamo effettuare un'echo del socket ora istanziato che ci permette di osservare come a ogni nuovo client che si connette viene mantenuta attiva la *local port 6789* mentre troveremo volta per volta un valore diverso per la port dove viene "trasferito".

# ESERCIZI IN LABORATORIO 3

Quindi viene creato un thread passando il socket al costruttore e successivamente viene avviato.

```
public class MultiServer{
  public void start(){
    try{
      ServerSocket serverSocket = new ServerSocket(6789);
      for (;;)
      {
        System.out.println("1 Server in attesa ... ");
        Socket socket = serverSocket.accept();
        System.out.println("3 Server socket " + socket);
        ServerThread serverThread = new ServerThread(socket);
        serverThread.start();
      }
    }
    catch (Exception e){
      System.out.println(e.getMessage());
      System.out.println("Errore durante l'istanza del server !");
      System.exit(1);
    }
  }
```

In questo esempio il server non termina mai la propria esecuzione: il corpo del metodo start() è costituito da un ciclo infinito d'attesa sulla accept() per la generazione di un numero indefinito di client.
Il main() si riduce alla seguente istruzione:

```
  public static void main (String[] args){
    MultiServer tcpServer = new MultiServer();
    tcpServer.start();
  }
```

Facciamo alcune modifiche anche al client per realizzare il metodo che permette di inviare più messaggi al server: il metodo modificato è comunica().

In questo metodo realizziamo un ciclo infinito mediante l'istruzione for(;;), dove ogni iterazione di tale ciclo esegue:
- input di una riga da parte dell'utente;
- trasmissione al server;
- ricezione dell'echo da parte del server;

e termina quando l'utente inserisce la parola FINE chiudendo la connessione.

```
  public void comunica() {
    for (;;)                                  // ciclo infinito: termina con FINE
    try{
      System.out.println("4 ... utente, inserisci la stringa da trasmettere al server:");
      stringaUtente = tastiera.readLine();
      //la spedisco al server
      System.out.println("5 ... invio la stringa al server e attendo ...");
      outVersoServer.writeBytes( stringaUtente+'\n');
      //leggo la risposta del server
      stringaRicevutaDalServer=inDalServer.readLine();
      System.out.println("7 ... risposta dal server "+'\n'+stringaRicevutaDalServer );
      if (stringaUtente.equals("FINE")) {
        System.out.println("8 CLIENT: termina elaborazione e chiude connessione" );
        miosocket.close();                    // chiude la connessione
        break;
      }
    }
```

# 3 ESERCIZI IN LABORATORIO

E in caso di errore viene catturata l'eccezione:

```
    }
    catch (Exception e)
    {
      System.out.println(e.getMessage());
      System.out.println("Errore durante la comunicazione col server!");
      System.exit(1);
    }
```

Il metodo connetti(), non molto diverso da quello precedentemente descritto nell'Esercizio di Laboratorio 1, è il seguente:

```
  public Socket connetti(){
    System.out.println("2 CLIENT partito in esecuzione ...");
    try{
      // input da tastiera
      tastiera = new BufferedReader(new InputStreamReader(System.in));
      //  miosocket = new Socket(InetAddress.getLocalHost(), 6789);
      miosocket = new Socket(nomeServer,portaServer);
      // associo due oggetti al socket per effettuare la scrittura e la lettura
      outVersoServer = new DataOutputStream(miosocket.getOutputStream());
      inDalServer    = new BufferedReader(new InputStreamReader (miosocket.getInputStream()));
    }
    catch (UnknownHostException e){
      System.err.println("Host sconosciuto"); }
    catch (Exception e){
      System.out.println(e.getMessage());
      System.out.println("Errore durante la connessione!");
      System.exit(1);
    }
    return miosocket;
  }
```

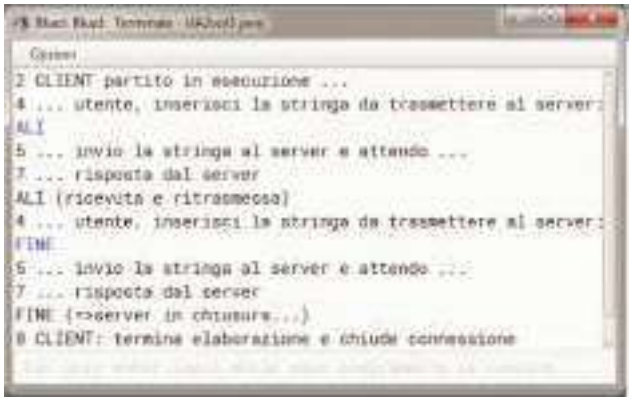
Il resto della classe (chiamata public class Multiclient()) è identico al client descritto nell'Esercizio di Laboratorio precedente.
Vediamo gli output generati da una esecuzione: mandiamo dapprima in esecuzione il server e successivamente due client.

```
Opzioni
1 Server in attesa ...
3 Server socket  Socket[addr=/127.0.0.1,port=50845,localport=6789]
1 Server in attesa ...
6 Echo sul server :ALI
3 Server socket  Socket[addr=/127.0.0.1,port=50848,localport=6789]
1 Server in attesa ...
6 Echo sul server :BABA
Echo sul server in chiusura  :FINE
9 Chiusura socketSocket[addr=/127.0.0.1,port=50845,localport=6789]
Echo sul server in chiusura  :FINE
9 Chiusura socketSocket[addr=/127.0.0.1,port=50848,localport=6789]
Type input and press Enter to send to program
```

I client comunicano al server solamente una parola (rispettivamente ALI e BABA): nel primo (ALI) digitiamo FINE mentre apriamo una nuova finestra e mandiamo in esecuzione un terzo client.

# ESERCIZI IN LABORATORIO 3

Di seguito, a sinistra, è riprodotto l'echo() sul client ALI e, a destra, l'echo() sul client BABA:



## METTITI ALLA PROVA

Realizza un'applicazione che simuli una lotteria di 90 numeri; alcuni giocatori si collegano e "acquistano" alcuni numeri (per esempio 5) estratti causalmente dal server: quando si raggiunge un numero adeguato di giocatori (almeno 4) e non avvengono successive connessioni per almeno 60 secondi il server provvede alla estrazione di 5 premi, comunicando ai giocatori l'esito della estrazione; a loro volta i client, se hanno un numero vincente, lo segnalano al server e ritirano la vincita.

## Esercizi proposti

1 Realizza l'agenzia stampa PANZA: ogni redazione locale invia a un server una notizia catalogata secondo lo schema Settore/Argomento/Area geografica; l'agenzia server le deve smistare in base a tale classificazione e inviare come risposta tutte le notizie analoghe.

2 Realizza un sistema in cui il server sia il banditore di un'asta: accoglie le offerte e comunica se sono accettabili o meno, comunica a richiesta la migliore offerta corrente; i client sono i partecipanti all'asta che possono richiedere quale sia l'offerta migliore (e di chi) e possono effettuare rilanci se il loro budget a disposizione lo consente.

3 Realizza un sistema che simuli la Borsa Valori: il server comunica ciclicamente un listino azionario con la quotazione dei singoli titoli a tutti i client a lui connessi, i quali possono comunicare una variazione (verso l'alto o verso il basso di una certa quantità fissa) del valore di un titolo del listino.

4 Realizza un sistema per la gestione del gioco della tombola dove il server gestisce il tabellone e assegna a ogni singolo client che si connette una cartella casuale; ogni client controlla i numeri estratti e segnala al server le eventuali vincite.

5 Realizza un sistema di estrazioni del lotto: N giocatori scelgano tre numeri e li comunichino alla ricevitoria. Quindi effettua le estrazioni del lotto comunicando eventuali vincite.

6 I contatori dell'energia elettrica sono connessi in rete in modo che in giorni ben determinati vengano interrogati per comunicare i consumi che sono stati effettuati e sono raggruppati per aree geografiche: ogni contatore ha un numero di matricola che viene comunicato assieme al valore della lettura.
Scrivi una applicazione che invii un codice di area e si ponga in attesa delle risposte dei diversi contatori; poi memorizzi i dati ricevuti in un file di record con la seguente struttura:
dataLettura, matricolaContatore, .valoreLettura.
Al file viene dato come nome il codice dell'area.

# 4 ESERCIZI IN LABORATORIO

## JAVA SOCKET: UN'ANIMAZIONE CLIENT-SERVER

### Una palla che "attraversa" le finestre

Realizziamo ora un'applicazione costituita da due finestre indipendenti affiancate e una pallina in movimento che si sposta nella prima finestra: quando raggiunge il bordo destro, anziché rimbalzare, "esce" dalla prima finestra ed "entra" nella seconda (come mostrato nella sequenza di immagini).

Si avvicina al bordo destro della prima finestra:

inizia l'attraversamento:

procede nella seconda finestra:

termina l'attraversamento passando completamente nella seconda finestra dove continua il suo movimento, rimbalza sui suoi bordi e ritorna nella prima finestra.

# ESERCIZI IN LABORATORIO 4

## Descrizione della soluzione

Realizziamo il sistema con un'applicazione client-server, quindi definiamo due classi e facciamole comunicare tra loro. La pallina viene generata dal server non appena si accorge che il client si è connesso.
Dovendo gestire un'animazione, abbiamo problemi di sfarfallio che annulliamo con la tecnica del doppio buffering: realizziamo l'animazione gestendo l'evento generato dall'oggetto Timer e, dato che è proprio il metodo che gestisce tale evento a incaricarsi del movimento della pallina, e quindi riconoscere il momento in cui "sta per lasciare" la finestra, demandiamo a tale metodo la realizzazione della comunicazione tra client e server per ricevere la segnalazione.
Implementiamo due classi:

```
class PannelloAnimazione extends JPanel implements ActionListener{...}

class PannelloClient extends JPanel implements ActionListener( ...)
```

rispettivamente per il server e per il client che si occupano, quindi, di:
- animare la pallina;
- eliminare lo sfarfallio;
- comunicare tra loro le situazioni di transazione.

Nel complesso le due classi sono abbastanza simili: descriviamo solamente la classe server dove evidenzieremo le differenze con la classe del client.

La classe "starter" è la seguente, cioè PallaNetServer, che definisce la finestra e avvia un thread che attende la connessione del client:

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.io.*;
4  import java.net.*;
5  import javax.swing.*;
6  public class PallaNetServer extends JFrame{
7    private Socket connessione = null;
8    private DataOutputStream out = null;
9    private DataInputStream input = null;
10   public PallaNetServer() {
11    super("PallaNet - Server");
12    this.setSize(500,400);              //setto la grandezza della finestra
13    this.setLocationRelativeTo(null); //setto la posizione al centro dello schermo
14    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
15    //instanzio un oggetto per attendere la connessione di un client
16    ThreadConnessione attendiConnessione = new ThreadConnessione(this);
17    this.setVisible(true);
18   }//fine costruttore
```

Unica osservazione nella definizione di classe è che estende la classe JFrame; nulla da sottolineare nel costruttore tranne che per l'utilizzo di un oggetto della classe ThreadConnessione() che rimane in attesa sulla porta **6789** mediante il metodo setConnessione(), dopo aver ricevuto dal costruttore il riferimento alla finestra.

Nel caso di singolo client non è necessario utilizzare i thread, ma questo server è stato così impostato a titolo di esempio, in modo da poter essere utilizzato intervenendo con "piccole modifiche" anche nel caso di connessione tra più client, come viene richiesto nell'esercizio proposto al termine della spiegazione di questo esempio.

# 4 ESERCIZI IN LABORATORIO

Con l'istruzione 73 l'avvenuta connessione col client richiama il metodo setConnessione() che, utilizzando il riferimento alla connessione con il client, crea gli stream di input e output e definisce e istanzia un PannelloAnimazione() avviando, di fatto, il movimento della pallina.

```
61 class ThreadConnessione implements Runnable{
62   private PallaNetServer finestra;
63   private Thread me;
64   public ThreadConnessione(PallaNetServer finestra){
65     //ottengo il riferimento alla JFrame
66     this.finestra = finestra;
67     me = new Thread(this);                          //instanzio il Thread
68     me.start();                                     //attivo il Thread
69   }
70   public void run(){
71     try{                  //instanzio un oggetto in ascolto sulla porta 6789
72       ServerSocket server = new ServerSocket(6789);
73       finestra.setConnessione(server.accept());     //attesa connessione
74       server.close();                               //chiudo il server
75     }catch(Exception e){
76       JOptionPane.showMessageDialog(null,e.toString());
77       System.exit(-1);
78     }
79   }//fine metodo run del Thread
80 }
```

Non appena il client si connette viene "chiuso" il server oppure si segnala un errore: naturalmente nel caso si dovessero attendere più client è necessario intervenire in questa prima parte di codice.

```
19   public void setConnessione(Socket connessione){
20     //ricevo il riferimento per la connessione con il client
21     this.connessione = connessione;
22     //ricevo lo stream di output e di input
23     try{
24       out = new DataOutputStream(connessione.getOutputStream());
25       input = new DataInputStream(connessione.getInputStream());
26     }catch(Exception e){
27       JOptionPane.showMessageDialog(null,e.toString());
28       System.exit(-1);
29     }
30     //inizio l'animazione su un oggetto di classe PannelloAnimazione
31     PannelloAnimazione pan = new PannelloAnimazione(this,this.getSize());
32     //lo aggiungo alla JFrame
33     this.add(pan);
34   }
```

Completano la classe due metodi definiti per rispettare l'incapsulamento.

```
46
47   public DataInputStream getInput(){
48     return input;
49   }
50
51   public DataOutputStream getOutput(){
52     return out;
53   }
54 } /* --Fine classe PallaNetServer */
```

# ESERCIZI IN LABORATORIO 4

Le differenze con la classe del client, la PallaNetClient, si limitano alle istruzioni di connessione. Al posto del thread che effettua la connessione con le seguenti istruzioni:

```
ServerSocket server = new ServerSocket(6789);
finestra.setConnessione(server.accept());     //attesa connessione
server.close();                               //chiudo il server
```

nel client mettiamo semplicemente la seguente istruzione che crea la connessione:

```
//mi connetto al server sul socket(<nomecomputer>,<porta>)
connessione = new Socket("localhost",6789);
```

La definizione della classe e del suo costruttore è la seguente:

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
public class PallaNetClient extends JFrame{
  private Socket connessione = null;
  private DataOutputStream out = null;
  private DataInputStream input = null;
  public PallaNetClient(){
    super("PallaNet - Client");            // costruttore della super classe
    this.setSize(500,400);                 // grandezza della finestra
    this.setLocationRelativeTo(null);      // posizione al centro schermo
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);//modalità di chiusura
    connetti();                                  //mi connetto al server
    this.setVisible(true);                       //rende visibile la finestra
  }//fine costruttore
```

Il metodo che effettua la connessione e avvia l'animazione della pallina è il seguente:

```
public void connetti(){
  try {
    //mi connetto al server sul socket(<nomecomputer>,<porta>)
    connessione = new Socket("localhost",6789);
    //ottengo le stream di input dal server e di output verso il server
    out = new DataOutputStream(connessione.getOutputStream());
    input = new DataInputStream(connessione.getInputStream());
  }catch(Exception e){
    JOptionPane.showMessageDialog(null,e.toString());
    System.exit(-1);
  }
  //inizio l'animazione
  Container contenitore = this.getContentPane();
  PannelloClient pan = new PannelloClient(this,contenitore.getSize());
  contenitore.add(pan);    //lo aggiungo alla JFrame
}
```

Conclude la classe la definizione dei due metodi getter:

```
public DataInputStream getInput(){
 return input;
}
public DataOutputStream getOutput(){
  return out;
}
```

# 4 ESERCIZI IN LABORATORIO

I pannelli che gestiscono l'animazione sono simili: descriviamo quindi solo quello del server, cioè la classe PannelloAnimazione.
Iniziamo definendo la classe con le variabili necessarie per l'animazione:

```
class PannelloAnimazione extends JPanel implements ActionListener{
  private PallaNetServer finestra;
  private Dimension dimensioni;
  private Image bufferVirtuale;            // per il doppio buffering
  private Graphics offScreen;
  private Timer tim = null;
  private int xPallina = 0;                // coordinate iniziali
  private int yPallina = 0;
  private int diametroPallina = 40;
  private int spostamento = 3;
  private int timerDelay = 10;
  private boolean destra,basso,pallinaPresente,arrivoComunicato;
```

Gli attributi sono "parlanti" e non necessitano di spiegazioni aggiuntive.
Anche il suo costruttore non presenta particolarità degne di nota:

```
  public PannelloAnimazione(PallaNetServer finestra,Dimension dimensioni){
    super();
    this.finestra = finestra;        // riferimento alla finestra corrente
    this.setSize(dimensioni);        // setto la grandezza
    destra = true;                   // direzioni iniziali della pallina
    basso = true;
    pallinaPresente = true;          //la pallina parte dal questo schermo
    arrivoComunicato = false;
    tim = new Timer(timerDelay,this); // instanzio il Timer per il movimento
    tim.start();                      // attivo il Timer
  }
```

Completano la classe i due metodi che si occupano del disegno evitando lo sfarfallio mediante la tecnica del doppio buffering:

```
  public void update(Graphics g){      // aggiorno il pannello
   paint(g);                           // eseguo il disegno in Doppio Buffering
  }
  public void paint(Graphics g){
    super.paint(g);
    //definisco lo spazio esterno per il doppioBuffering
    bufferVirtuale = createImage(getWidth(),getHeight());
    offScreen = bufferVirtuale.getGraphics();
    Graphics2D screen = (Graphics2D) g;
    offScreen.setColor(new Color(254,138,22));  // setto il colore della palla
    if(pallinaPresente){                        // disegno la palla
      offScreen.fillOval(xPallina,yPallina,diametroPallina,diametroPallina);
    }
    //disegno l'immagine modificata "bufferVirtuale" sul Component
    screen.drawImage(bufferVirtuale,0,0,this);
    offScreen.dispose();
  }
```

Più articolato è il metodo che gestisce l'evento generato dal time e che quindi si occupa del movimento della pallina ed eventualmente della "migrazione" tra le due finestre. Lo analizziamo in tre segmenti:

- aggiornamento dello spostamento della palla in verticale;
- aggiornamento dello spostamento della palla in orizzontale;
- palla assente nella finestra.

# ESERCIZI IN LABORATORIO 4

Nulla da segnalare quando aggiorniamo il moto verticale in quanto dobbiamo solo controllare quando avviene la collisione con i bordi per invertire la direzione:

```
public void actionPerformed(ActionEvent e){
  if(pallinaPresente){
    /*direzione in verticale*/
    if(basso){
      //si muove verso il basso
      if(yPallina > (this.getHeight()-diametroPallina)){// urto inferiore
        basso = false;                                  //cambia direzione
        yPallina -= spostamento;
      }else{
        yPallina += spostamento;
      }
    }else{
      //si muove verso l'alto
      if(yPallina <= 0){                                // urto superiore
        basso = true;                                   //cambia direzione
        yPallina += spostamento;
      }else{
        yPallina -= spostamento;
      }
    }
```

Durante la direzione orizzontale abbiamo due diverse situazioni.

**A** Nel caso di *movimento verso destra*, quando la pallina tocca il bordo destro, inizia la "migrazione" nella finestra del client; dobbiamo quindi:

- comunicare al client che inizia il trasferimento;
- comunicare al client la coordinata verticale di "passaggio";
- comunicare al client la direzione verticale.

Quindi impostiamo una variabile booleana (arrivoComunicato) utilizzata dal server per non ripetere la comunicazione al client delle successive posizioni della pallina durante la transazione:

```
    /*direzione orizzontale*/
    if(destra){
      if(!(arrivoComunicato)&&(xPallina > (this.getWidth()-diametroPallina))){
        /* comunico il verso di direz. verticale e la coordinata verticale
        *  in cui la pallina si trova attualmente                        */
        try{
          finestra.getOutput().writeBoolean(basso);
          finestra.getOutput().writeInt(yPallina);
          arrivoComunicato = true;
        }catch(Exception ecc){
          JOptionPane.showMessageDialog(null,ecc.toString());
          System.exit(-1);
        }
      }else{
        xPallina += spostamento;
        if(xPallina > this.getWidth()){
          pallinaPresente = false;
          arrivoComunicato = false;
        }
      }
```

Durante la transazione avremo due palline presenti, una nel client che va "scomparendo" e una nel server che si sta "formando".

Quando la pallina ha lasciato completamente la finestra del server vengono modificati i valori delle due variabili booleane pallinaPresente e arrivoComunicato che vengono posti al valore di falso.

# 4 ESERCIZI IN LABORATORIO

**B** Nel caso di movimento verso sinistra è sufficiente controllare se si è raggiunto il bordo laterale dello schermo per invertire la direzione:

```
}else{                          // la pallina sta andando a sinistra
  if(xPallina <=0 ){            // fine finestra a sinistra
    destra = true;              // cambio direzione
    xPallina += spostamento;
  }else{
    xPallina -= spostamento;
  }
}
```

Quando la pallina non è presente, il server rimane in attesa con l'istruzione 177 di una eventuale comunicazione da parte del client dell'inizio di "ritorno pallina": in questa occasione vengono aggiornate le coordinate e la variabile di stato pallinaPresente = true.

```
}else{                                  // la pallina non e presente
  // rimango in attesa del ritorno della pallina nel mio schermo
  boolean direzione = false;
  int y = 0;
  try{
    direzione = finestra.getInput().readBoolean();
    y = finestra.getInput().readInt();  // attendo che arriva la pallina
    basso = direzione;                  // direzione di ingresso
    destra = false;                     // si muove verso sinistra
    yPallina = y;                       // coord. iniziali della pallina
    xPallina = this.getWidth();
    pallinaPresente = true;
  }
  catch(Exception ecc){
    JOptionPane.showMessageDialog(null,ecc.toString());
    System.exit(-1);
  }
}
```

L'ultima operazione del metodo effettua il ridisegno invocando il metodo repaint().

```
    repaint();
  }
}
```

Il codice del client è simile: sono naturalmente invertite le situazioni di movimento della pallina, cioè se dal server esce dal lato destro, nel client entrerà dal lato sinistro e da questo lato successivamente uscirà per rientrare nel server dallo stesso lato da cui è uscita.

Anche le classi di prova sono simili e le riportiamo a titolo di esempio.

```
public class ProvaPallaServer{
  public static void main(String args[]){
    PallaNetServer server = new PallaNetServer();
  }
}
```

```
public class ProvaPallaClient{
  public static void main(String args[])  {
    PallaNetClient server = new PallaNetClient();
  }
}
```

# ESERCIZI IN LABORATORIO 4

Maggior effetto scenografico si ottiene mandando in esecuzione i due thread su due macchine diverse, connesse in rete, ponendo i due monitor vicini: unica modifica da apportare alla classe client è l'aggiornamento del nome del PC su cui viene mandato in esecuzione il thread server.
La fotografia seguente ne riporta una esecuzione

```
class PannelloAnimazione extends JPanel implements ActionListener{...}

class PannelloClient extends JPanel implements ActionListener( ...)
```

## METTITI ALLA PROVA

1. Modifica il programma precedente aggiungendo un secondo client, cioè una terza finestra, facendo in modo che la pallina esca prima a destra e successivamente a sinistra dalla videata centrale.
2. Realizza la simulazione di una corsa dei cavalli: quattro cavalli percorrono uno o più giri di un ippodromo ovale fino al termine della gara. Aggiungi successivamente la possibilità di effettuare puntate sui cavalli.
3. Realizza un sistema che gestisca il movimento di automobili in rete: il server attende la connessione di N giocatori che muovono la propria auto mediante i tasti freccia.

# 5 ESERCIZI IN LABORATORIO

## IL PROTOCOLLO UDP NEL LINGUAGGIO JAVA

### Client e server UDP in Java

Il protocollo UDP ha caratteristiche diverse dal protocollo TCP: UDP non è orientato alla connessione e quindi tra due host non si crea uno stream stabile.
L'avvio di una comunicazione avviene senza handshaking e la consegna non è garantita; al contrario i controlli sull'integrità vengono eseguiti come in TCP.
Uno dei principali vantaggi di UDP è che, grazie alla sua semplicità offre un servizio molto rapido e non richiede handshaking (per questo viene spesso utilizzato per DNS).
In questa esercitazione realizzeremo la comunicazione mediante protocollo UDP dove un mittente (UDPClient.java) trasmette un messaggio (datagram) a un destinatario (UDPServer.java).
Ricordiamo che, in una comunicazione dati "connectionless o datagram", il canale:

- trasporta messaggi;
- non è affidabile;
- ha il socket condiviso;
- non preserva l'ordine delle informazioni.

Inoltre l'ordine di arrivo dei pacchetti non è necessariamente lo stesso di invio.

Realizzare un sistema client-server UDP consiste nell'implementare un programma che effettui i seguenti passi fondamentali.

Per il client:

- crea il socket;
- manda richiesta sul socket (composto da indirizzo, porta e messaggio);
- riceve dati dal socket;
- chiude il socket.

Per il server:

- crea il socket
- ripete le seguenti operazioni:
  - si mette in attesa delle richieste in arrivo;
  - riceve il messaggio dal socket;
  - invia la risposta sul socket al client che ha fatto una richiesta;
- chiude il socket.

La comunicazione UDP viene realizzata in Java mediante la classe DatagramSocket.

### Classe DatagramSocket

La classe Java DatagramSocket usa il protocollo UDP per trasmettere dati attraverso i socket: permette a un client di connettersi a una determinata porta di un host per leggere e scrivere dati impacchettati attraverso la classe DatagramPacket.

L'indirizzo dell'host destinatario è sul DatagramPacket.

I principali metodi della DatagramSocket(..) sono i seguenti.

- Metodi costruttori:

```
void DatagramSocket() throws SocketException
void DatagramSocket(int porta) throws SocketException
```

# ESERCIZI IN LABORATORIO 5

- Metodi per ricevere/trasmettere:

```
void send(DatagramPacket pacchettoDati) throws IOException
void receive(DatagramPacket pacchettoDati) throws IOException
```

Il metodo receive() blocca il chiamante fino a quando un pacchetto è ricevuto

```
void setSoTimeout(int timeout) throws SocketException
```

usando setSoTimeout() il chiamante di receive() si blocca dopo timeout millisecondi.

Infine, il "solito" metodo per chiudere la connessione:

```
void close()
```

## Classe DatagramPacket

La classe DatagramPacket è la classe Java che permette di rappresentare i pacchetti UDP da inviare e ricevere sui socket di tipo datagram.

Il costruttore della classe è il seguente:

```
public DatagramPacket(byte buffer[], int lunghezza, InetAddress indirizzo, int porta)
```

Il client crea un oggetto di DatagramPacket passando al costruttore:
- il contenuto del messaggio: un array di lunghezza caratteri;
- l'indirizzo IP del server;
- il numero di porta su cui il server è in ascolto.

Il server, per ricevere un messaggio, crea un oggetto di DatagramPacket definendone la lunghezza massima:

```
public DatagramPacket(byte buffer[], int lunghezza)
```

I principali metodi della classe DatagramPacket sono:

**A** Per la gestione dell'indirizzo IP:

`void setAddress(InetAddress indirizzo)`: setta il valore dell'indirizzo `IP`;
`InetAddress getAddress()`: restituisce l'indirizzo `IP` del'host da cui il pacchetto è stato ricevuto.

**B** Per la gestione della porta:

`void setPort(int porta)`: setta il valore della porta;
`int getPort()`: restituisce la porta della macchina remota da cui il pacchetto è stato ricevuto.

**C** Per la gestione dei dati:

`void setData(byte[] buffer)`: inserisce i dati nel pacchetto;
`byte[] getData()`: restituisce i dati del pacchetto.

# 5 ESERCIZI IN LABORATORIO

## Server UDP

Scriviamo il server UDP: creiamo un DatagramSocket() sulla solita porta 6789 e definiamo i due buffer per i dati, rispettivamente per la ricezione e per la trasmissione, per esempio della medesima dimensione di 1024 byte.

Una variabile booleana viene definita per essere utilizzata come condizione di ripetizione/uscita dal ciclo.

```
import java.io.*;
import java.net.*;

class UDPServer{
  public static void main(String args[]) throws Exception {
    DatagramSocket serverSocket = new DatagramSocket(6789);
    boolean attivo = true;                  // per la ripetizione del servizio
    byte[] bufferIN = new byte[1024];       // buffer spedizione e ricezione
    byte[] bufferOUT = new byte[1024];

    System.out.println("SERVER avviato...");
    while(attivo)
```

Definiamo ora il datagramma e ci poniamo in attesa di ricevere dati da qualche client:

```
      // definizione del datagramma
      DatagramPacket receivePacket =
          new DatagramPacket(bufferIN,bufferIN.length);
      //attesa della ricezione dato dal client
      serverSocket.receive(receivePacket);
```

Quando viene ricevuto un pacchetto, viene analizzato estraendo il messaggio, individuando la lunghezza impostata alla trasmissione per eliminare i caratteri superflui presenti nel buffer, che ricordiamo essere di 1024 byte;

```
      // analisi del pacchetto ricevuto
      String ricevuto = new String(receivePacket.getData());
      int numCaratteri = receivePacket.getLength();
      ricevuto=ricevuto.substring(0,numCaratteri); //elimina i caratteri in eccesso
      System.out.println("RICEVUTO: " + ricevuto);
```

Sempre dal pacchetto ricevuto vengono individuati i parametri per la trasmissione della risposta, cioè l'indirizzo e il numero di porta del client:

```
      // riconoscimento dei parametri del socket del client
      InetAddress IPClient = receivePacket.getAddress();
      int portaClient = receivePacket.getPort();
```

Viene preparata la risposta, che in questo caso non è altro che il messaggio ricevuto trasformato in maiuscolo, viene creato il datagramma in uscita e viene inviato al client:

```
      // preparo il dato da spedire
      String daSpedire = ricevuto.toUpperCase();
      bufferOUT = daSpedire.getBytes();
      // invio del Datagramma
      DatagramPacket sendPacket =
          new DatagramPacket(bufferOUT, bufferOUT.length, IPClient, portaClient);
      serverSocket.send(sendPacket);
```

# ESERCIZI IN LABORATORIO 5

Se il client ha inviato la parola "fine", viene conclusa l'elaborazione del server, si esce quindi dal ciclo e si chiude il socket.

```
      // controllo termine esecuzione del server
      if (ricevuto.equals("fine"))
      {
        System.out.println("SERVER IN CHIUSURA. Buona serata.");
        attivo=false;
      }
    }
    serverSocket.close();
  }
```

## Client UDP

Scriviamo ora il client UDP: come per il server definiamo i buffer per la comunicazione e indichiamo i riferimenti del socket del server:

```
import java.io.*;
import java.net.*;

class UDPClient
{
  public static void main(String args[]) throws Exception
  {
    int portaServer   = 6789;              // porta del server
    InetAddress IPServer = InetAddress.getByName("localhost");

    byte[] bufferOUT = new byte[1024];     // buffer di spedizione e ricezione
    byte[] bufferIN = new byte[1024];
    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
```

Creiamo un socket e invitiamo l'utente a inserire un dato:

```
    // creazione del socket
    DatagramSocket clientSocket = new DatagramSocket();
    System.out.println("Client pronto - inserisci un dato da inviare:");
```

Leggiamo la stringa inserita dall'utente e predisponiamo il buffer di uscita:

```
    // preparazione del messaggio da spedire
    String daSpedire = input.readLine();
    bufferOUT = daSpedire.getBytes();
```

Creiamo ora un DatagramPacket con il messaggio da inviare e i parametri del server e lo inviamo a destinazione:

```
    // trasmissione del dato al server
    DatagramPacket sendPacket =
       new DatagramPacket(bufferOUT, bufferOUT.length, IPServer, portaServer);
    clientSocket.send(sendPacket);
```

Quindi ci mettiamo in attesa della risposta, predisponendo il DatagramPacket con il buffer per la ricezione:

```
    // ricezione del dato dal server
    DatagramPacket receivePacket = new DatagramPacket(bufferIN, bufferIN.length);
    clientSocket.receive(receivePacket);
    String ricevuto = new String(receivePacket.getData());
```

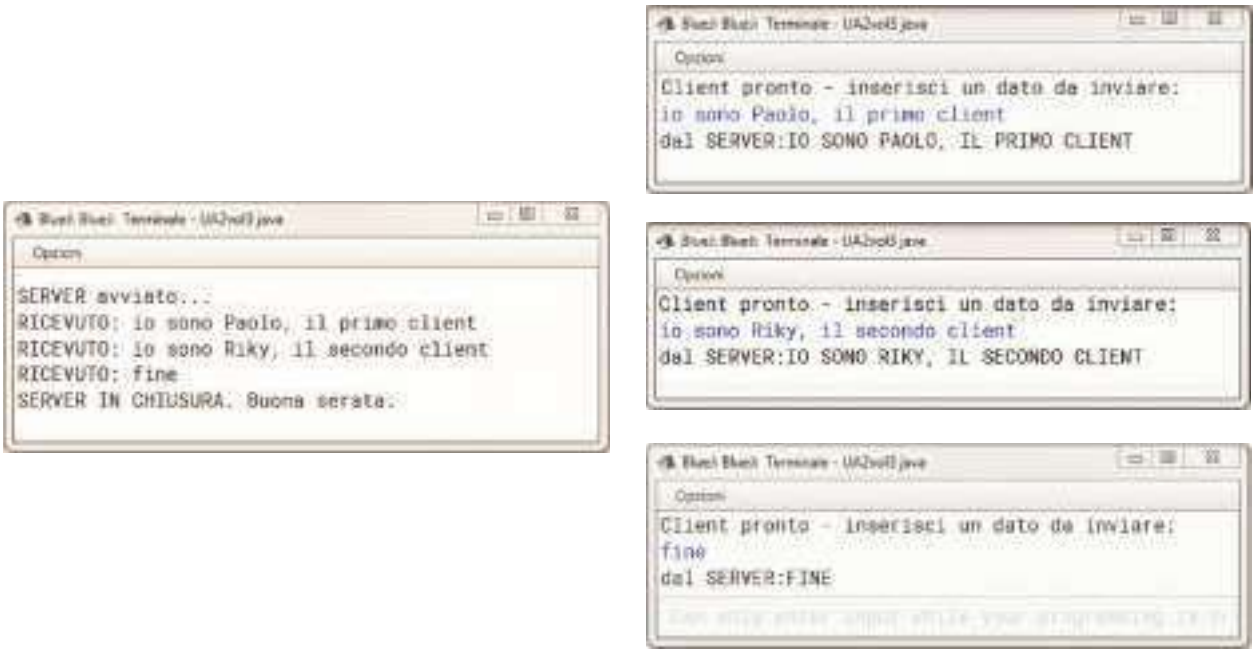
# 5 ESERCIZI IN LABORATORIO

Alla ricezione del messaggio da parte del server, questo viene analizzato, vengono tolti i caratteri eccedenti presenti nel buffer di ricezione e viene visualizzato il risultato:

```
// elaborazione dei dati ricevuti
int numCaratteri = receivePacket.getLength();
ricevuto=ricevuto.substring(0,numCaratteri); //elimina i caratteri in eccesso
System.out.println("dal SERVER:" + ricevuto);

// termine elaborazione
clientSocket.close();
```

Il client termina l'elaborazione chiudendo la connessione.

Un'esecuzione della applicazione è la seguente, dove sono stati mandati in esecuzione tre client e l'ultimo ha comunicato la stringa "fine":




## METTITI ALLA PROVA

1. Il servizio daytime (udp/13) fornisce data e ora correnti in formato comprensibile da un essere umano: scrivi un server che invia data e ora in un pacchetto UDP a ogni client che gli invia un qualsiasi pacchetto UDP, anche vuoto.
2. Successivamente effettua il conteggio delle volte che un medesimo client si è connesso memorizzandolo su di un file: alla decima connessione segnala al client che ha "esaurito i bonus" gratuiti e da questo momento il servizio "orario" è a pagamento.

# ESERCIZI IN LABORATORIO 6

## APPLICAZIONI MULTICAST IN JAVA

### Client e server multicast

La comunicazione tra gruppi di processi realizzata mediante multicasting (one to many communication):
- un insieme di processi formano un gruppo di multicast;
- un messaggio spedito da un processo a quel gruppo viene recapitato a tutti gli altri partecipanti appartenenti al gruppo.

L'implementazione del multicast richiede:
- uno schema di indirizzamento dei gruppi;
- un supporto che registri la corrispondenza tra un gruppo e i partecipanti;
- un'implementazione che ottimizzi l'uso della rete nel caso di invio di pacchetti a un gruppo di multicast.

Le API Multicast contengono le primitive per far unire un client a un gruppo di multicast (join) e per lasciare il gruppo (leave) quando non è più interessato a ricevere i messaggi relativi a quel gruppo.
La libreria java.net mette a disposizione la classe MulticastSocket per inviare messaggi multicast e con i socket datagram si può usare il multicast per mandare un pacchetto a un insieme di processi con una sola invocazione a send().

I principali metodi della classe MulticastSocket sono:

| | |
|---|---|
| `joinGroup(InetAddress addr) throws IOException` | : per unirsi al gruppo |
| `leaveGroup(InetAddress addr) throws IOException` | : per lasciare il gruppo |
| `send(DatagramPacket p)` | : per inviare un pacchetto |
| `setTimeToLive(int tlive) throws IOException` | : per settare il tempo di vita dei pacchetti inviati sul socket |
| `int getTimeToLive() throws IOException` | : ritorna il tempo restante (tra 0 e 255) |

### Client multicast

I client che vogliono ricevere messaggi multicast devono unirsi a un gruppo specifico e per far questo devono conoscere:
- la porta del socket usata dal server (per noi la solita 6789);
- l'indirizzo del gruppo a cui vengono inviati messaggi (per esempio "225.4.5.6").

Vediamo nello specifico come realizzare una classe client che si connette a un gruppo e riceve i messaggi a esso destinati:

```
1  import java.net.*;
2  import java.io.*;
3
4  public class MulticastClient
5  {
6    public static void main(String[] args) throws IOException
7    {
8      byte[] bufferIN = new byte[1024];          // buffer di ricezione
9      //parametri del server
10     int porta = 6789;
11     String gruppo = "225.4.5.6";
```

# 6 ESERCIZI IN LABORATORIO

Creiamo un socket multicast e uniamo il client nello specifico gruppo definito sul server:

```
// creazione del socket sulla porta
MulticastSocket socket = new MulticastSocket(porta);
// mi aggiungo al gruppo Multicast
socket.joinGroup(InetAddress.getByName(gruppo));
```

Creiamo un oggetto datagramma e ci poniamo in attesa dei messaggi del server:

```
// creo il DatagramPacket e mi metto in ricezione
DatagramPacket packetIN = new DatagramPacket(bufferIN, bufferIN.length);
socket.receive(packetIN);
```

Sullo schermo visualizziamo il messaggio ricevuto e alcuni parametri della porta del socket:

```
// Visualizzo i  parametri ed i dati ricevuti
System.out.println("Ho ricevuto dati di lunghezza: "+packetIN.getLength()
   + " da : " + packetIN.getAddress().toString()
   + " porta :" + packetIN.getPort());
System.out.write(packetIN.getData(),0,packetIN.getLength());
System.out.println();
```

Al termine delle operazioni il client si scollega dal gruppo e chiude il socket.

```
// al termine della ricezione lascio il gruppo
socket.leaveGroup(InetAddress.getByName(gruppo));
// chiudo il socket
socket.close();
```

Naturalmente un client può continuare a ricevere quanti pacchetti vuole: è sufficiente includere la receive() in un ciclo ed eseguirla periodicamente.

## Server multicast

Il processo server crea il socket per una porta e invia, quando vuole, pacchetti datagram al gruppo: nel nostro esempio invieremo la data e l'ora a intervalli di un secondo per un determinato tempo prefissato nella variabile conta (per esempio per 20 secondi).

Definiamo "225.4.5.6" come indirizzo di gruppo e apriamo un socket sulla porta **6789**:

```
import java.net.*;
import java.io.*;
import java.util.*;

public class MulticastServer
{
  public static void main(String[] args) throws IOException
  {
    boolean attivo = true;                  // per la ripetizione del servizio
    byte[] bufferOUT = new byte[1024];      // buffer di spedizione e ricezione
    int conta = 20;                         // secondi di attività del server
    int porta = 6789;
    InetAddress gruppo = InetAddress.getByName("225.4.5.6");
```

# ESERCIZI IN LABORATORIO 6

Creiamo il socket:

```
// crea il socket multicast
MulticastSocket socket = new MulticastSocket();
```

e prepariamo la variabile che conterrà il dato da trasmettere.

```
// contenitore per il dato da trasmettere
String dString = null;
```

Il ciclo di trasmissione genera il dato, crea un datagramma e lo spedisce al gruppo:

```
// ciclo di trasmissione
while (attivo)
{
  // come messaggio viene inviata la data e l'ora di sistema
  dString = new Date().toString();
  bufferOUT = dString.getBytes();
  // crea il DatagramPacket
  DatagramPacket packet;
  packet = new DatagramPacket(bufferOUT, bufferOUT.length, gruppo, porta);
  // invio il dato
  socket.send(packet);
```

Il ciclo di trasmissione viene ripetuto ogni secondo grazie all'attesa del metodo sleep() e a ogni iterazione aggiorna il contatore che modifica la condizione di terminazione:

```
// introduco un ciclo di attesa di 1 secondo
try {
  Thread.sleep(1000);   //attesa di 1000 millisecondi
} catch (InterruptedException ie) {
  ie.printStackTrace();
}
conta--;
if (conta==0){
  System.out.println("SERVER IN CHIUSURA. Buona serata.");
  attivo=false;
}else{
  System.out.println("SERVER attivo per altri secondi "+conta);
}
```

Alla fine, come ultima istruzione, il server chiude la connessione.

```
// alla fine chiudo il socket
socket.close();
}
```

Una possibile esecuzione è la seguente:

**A** per il server:

```
SERVER attivo per altri secondi 7
SERVER attivo per altri secondi 6
SERVER attivo per altri secondi 5
SERVER attivo per altri secondi 4
SERVER attivo per altri secondi 3
SERVER attivo per altri secondi 2
SERVER attivo per altri secondi 1
SERVER IN CHIUSURA. Buona serata.
```

# 6 ESERCIZI IN LABORATORIO

**B** per il client:

## METTITI ALLA PROVA

Realizza un'applicazione di diffusione in multicast della "Divina Commedia": il server inizia a trasmettere a intervalli di 1 secondo pacchetti contenenti ciascuno una riga di un file di testo così strutturata:

*cantica # numero canto # numero riga # una riga del canto*

Il client che riceve i datagrammi aspetta l'inizio di un nuovo canto e successivamente scrive un file per ogni canto, con la seguente struttura:

cantica_nrcanto.txt (per esempio inferno_canto5.txt).

Il file da trasmettere è anche disponibile nella cartella MATERIALI della sezione del sito www.hoepliscuola.it dedicata a questo volume.

## Esercizi proposti

Realizza un sistema per la gestione delle estrazioni del lotto:

- il server gestisce il tabellone ed effettua una nuova estrazione delle 11 ruote ogni giorno;
- l'estrazione su una nuova ruota, tra le 11 definite, avviene ogni 2 minuti, ed è composta da cinque numeri differenti in un range 1÷90, e ha la seguente struttura:

***<nome><estratto1>,<estratto2>,<estratto3>,<estratto4>,<estratto5>***

I singoli numeri vengono inviati man mano che vengono estratti ai diversi client: se un utente si collega mentre si è nel mezzo di un'estrazione, viene messo in attesa dell'inizio di un'estrazione su di una nuova ruota.
A fine giornata il server memorizza i numeri estratti in un file, che ha per nome il numero dell'estrazione e la data (est20_15_02_2020.bin).

# ESERCIZI IN LABORATORIO 7

## UN ESEMPIO COMPLETO CON I JAVA SOCKET: "LA CHAT"

Come esempio completo di applicazione Web realizziamo una semplice chat tra un server e un numero N di client (per esempio 10 client). Per la gestione della ricezione e dell'invio dei messaggi utilizziamo quindi il meccanismo multithread, in quanto non si può prevedere il momento in cui ogni singolo client fa richiesta di invio di un nuovo messaggio. Definiamo un thread con il compito di rimanere in ascolto su una determinata porta (la nota 6789) in attesa di connessioni da parte di un client; il server, a ogni nuova connessione, provvede a istanziare un nuovo oggetto dedicato con il compito di mantenere la connessione client-server per tutta la durata del suo collegamento e di occuparsi di ricevere e spedire i messaggi a quel client.
Vediamo ora il thread che si occupa di ricevere le connessioni dei client:
Il metodo run() si occupa di ricevere le connessioni dei client: al costruttore passiamo come parametro il numero massimo di connessioni e la lista dei messaggi.

```
public class ThreadGestioneServizioChat implements Runnable
{
  private int nrMaxConnessioni;
  private List lista;
  private ThreadChatConnessioni[] listaConnessioni;
  Thread me;
  private ServerSocket serverChat;

  public ThreadGestioneServizioChat(int numeroMaxConnessioni,List lista)
  {
    this.nrMaxConnessioni = nrMaxConnessioni-1;
    this.lista = lista;
    this.listaConnessioni = new ThreadChatConnessioni[this.nrMaxConnessioni];
    me = new Thread(this);
    me.start();
  }
```

Vediamo ora il thread che si occupa di ricevere le connessioni dei client: per prima cosa si crea un socket sulla porta 6789:

```
  public void run()
  {
    boolean continua = true;
    //instanzio la connessione dal server per la chat
    try{
      serverChat = new ServerSocket(6789);
    }catch(Exception e){
      JOptionPane.showMessageDialog(null,"Impossibile instanziare il server");
      continua = false;
    }
```

Se questa operazione va a buon fine, si procede con la creazione di un thread per ogni connessione:

```
    if(continua){
      //accetto le connessioni chat
      try {
        for(int xx=0;xx<nrMaxConnessioni;xx++){
          Socket tempo=null;
          tempo = serverChat.accept();
          listaConnessioni[xx] = new ThreadChatConnessioni(this,tempo);
        }
        serverChat.close();
      }catch(Exception e){
        JOptionPane.showMessageDialog(null,"Impossibile instanziare server chat");
      }
    }
  }//fine metodo "run"
```

# 7 ESERCIZI IN LABORATORIO

L'elenco delle connessioni viene memorizzato nell'array listaConnessioni.
La classe viene completata con un metodo che invia un messaggio a tutti i client:

```
47
48  public void spedisciMessaggio(String mex)
49  {
50    //scrivo il mex nella mia lista
51    lista.add(mex);
52    lista.select(lista.getItemCount()-1);
53    //Mando il messaggio agli altri
54    for(int xx=0; xx<this.nrMaxConnessioni;xx++)
55    {
56      if(listaConnessioni[xx] != null)
57      {
58        listaConnessioni[xx].spedisciMessaggioChat(mex);
59      }
60    }
61  }
62 }//fine classe ThreadGestioneServizioChat
```

Descriviamo ora la classe che viene associata a ogni client connesso con il server: si ottengono gli stream di I/O e si crea un metodo che permette di spedire i messaggi al client.
Le variabili di classe non sono commentate in quanto i loro identificativi "parlanti" indicano a che cosa servono.

```
1  import java.net.*;
2  import java.io.*;
3
4  public class ThreadChatConnessioni implements Runnable
5  {
6    private ThreadGestioneServizioChat gestoreChat;
7    private Socket client = null;
8    private BufferedReader input = null;
9    private PrintWriter output = null;
10   Thread me;
```

Le operazioni eseguite nel costruttore sono comunque note:

- creazione e inizializzazione dello stream input come oggetto BufferedReader;
- creazione e inizializzazione dello stream output come oggetto PrintWriter;
- inizializzazione di gestoreChat ricevuto come parametro;
- inizializzazione di Socket client ricevuto come parametro.

Quindi alla riga 23 viene creato un thread che viene mandato in esecuzione alla riga 24.

```
13   public ThreadChatConnessioni(ThreadGestioneServizioChat gestoreChat,Socket client)
14   {
15     this.gestoreChat = gestoreChat;
16     this.client= client;
17     try
18     {
19       this.input = new BufferedReader(new InputStreamReader(client.getInputStream()));
20       this.output = new PrintWriter(this.client.getOutputStream(),true);
21     }catch(Exception e){
22       output.println("Errore nella lettura.");
23     }
24     me = new Thread(this);
25     me.start();
26   }
```

# ESERCIZI IN LABORATORIO 7

Vediamo di seguito l'implementazione del metodo run().

```
  public void run()
  {
    while(true){
      try
      {
        String mex=null;
        //rimango in attesa dei messaggi mandati dai client
        while((mex = input.readLine())==null)
        { }
        //invoco il metodo del gestoreChat per ripetere a tutti il messaggio ricevuto
        gestoreChat.spedisciMessaggio(mex);
      }catch(Exception e)
      {
       output.println("Errore nella spedizione del messaggio a tutti.");
      }
    }
  }
```

Completa la classe un semplice metodo che spedisce un messaggio a un singolo client.

```
  public void spedisciMessaggioChat(String messaggio)
  {
    try{
      output.println(messaggio);
    }catch(Exception e){
      output.println("Errore nella spedizione del singolo messaggio.");
    }
  }
}
```

Vediamo ora l'implementazione del servizio client della chat. Il client ha solamente il compito di connettersi con il server, di rimanere in attesa di messaggi inviati dal server e, su richiesta dell'utente, deve inviare nuovi messaggi verso il server. Analizziamo in dettaglio il codice: la classe definisce le variabili private utilizzate da ogni singolo thread, come mostrato nella seguente schermata, e riceve nel costruttore la lista dei thread e i parametri del server al quale collegarsi.

```
import java.awt.*;
import java.net.*;
import java.io.*;
import javax.swing.*;

public class ThreadChatClient implements Runnable
{
  private List lista;
  Thread me;
  private Socket client;
  private BufferedReader input=null;
  private PrintWriter output=null;

  public ThreadChatClient(List lista, String ipServer, int porta)
  {
    this.lista = lista;
    try{
      client = new Socket(ipServer,porta);
      this.input = new BufferedReader(new InputStreamReader(client.getInputStream()));
      this.output = new PrintWriter(client.getOutputStream(),true);
    }catch(Exception e){
          JOptionPane.showMessageDialog(null,"Impossibile connettersi al server");
    }
    me = new Thread(this);
    me.start();
  }

```

# 7 ESERCIZI IN LABORATORIO

Il metodo run() rimane in attesa che l'utente scriva un messaggio, quindi invoca il metodo del server affinché il messaggio venga inoltrato a tutti coloro che sono connessi alla chat.

```
public void run()
{
  //aspetto le ricezioni di messaggi e aggiungo i messaggi alla lista
  while(true){
    try{
      String mex=null;
      while((mex = input.readLine())==null)
        { }
      lista.add(mex);
          lista.select(lista.getItemCount()-1);
    }catch(Exception e){

    }
  }
}
```

Conclude la classe un metodo che invia il messaggio al singolo client:

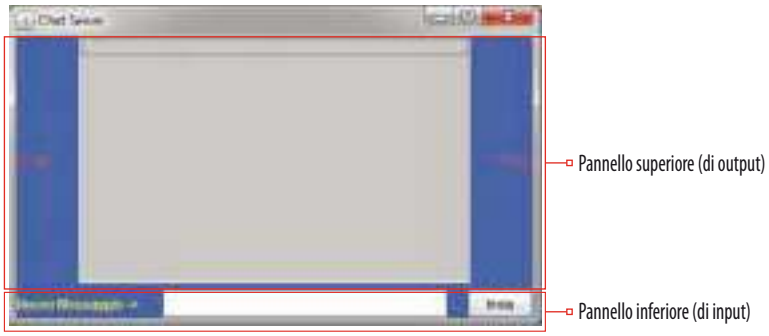
```
public void spedisciMessaggioChat(String messaggio){
  try{
        output.println(messaggio);
  }catch(Exception e){

  }
}
}
```

Riportiamo il codice che realizza l'interfaccia grafica solamente per il server, poiché è pressoché identica a quella del client:

```
public class ChatServer extends JFrame
{
  public ChatServer()
  {
    super("Chat Server");
        this.setSize(new Dimension(500,300)); // setto la grandezza della finestra
        this.setLocationRelativeTo(null);     // la metto al centro dello schermo
        this.setEnabled(true);                // setto la proprietà enable
        this.setBackground(Color.blue);       // setto il colore di sfondo
        //creo il pannello per l'inserimento e la visualizzazione dei messaggi
        PannelloChatServer pan = new PannelloChatServer();
        this.getContentPane().add(pan);
        this.setVisible(true);
  }
}
```

L'interfaccia grafica ha la struttura mostrata a lato, essa è costituita da due parti:

1. un pannello superiore (di output) dove vengono visualizzati tutti i messaggi;
2. un pannello inferiore (di input) dove l'utente scrive il proprio messaggio per inviarlo al server (e quindi agli altri utenti collegati).



# ESERCIZI IN LABORATORIO 7

La parte superiore del pannello è realizzata con il segmento di codice a lato:

```
public PannelloChatServer()
{
  super();
  this.setBackground(new Color(50, 100, 255));
  // pannello superiore lista messaggi
  JPanel panlista = new JPanel(new BorderLayout(20,5));
  panLista.setBackground(new Color(50, 100, 255));
  lista = new List();
  lista.setBackground(Color.lightGray);
  lista.setSize(100,50);
  lista.setVisible(true);
  // scritte laterali
  JLabel chat1 = new JLabel("  Chat  ",JLabel.CENTER);
  chat1.setForeground(new Color(200,100,100));
  JLabel chat2 = new JLabel("  Chat  ",JLabel.CENTER);
  chat2.setForeground(new Color(200,100,100));
  // aggiungiamo gli oggetti sul pannello
  panLista.add(chat1,BorderLayout.WEST);
  panLista.add(lista,BorderLayout.CENTER);
  panLista.add(chat2,BorderLayout.EAST);
```

La parte inferiore del pannello è realizzata con il segmento di codice a lato:

```
  //pannello inserimento nuovo messaggio
  JPanel nuovoMex = new JPanel(new BorderLayout(20,5));
  nuovoMex.setBackground(new Color(50, 100, 255));

  JLabel labNuovo = new JLabel("Nuovo Messaggio -> ",JLabel.CENTER);
  labNuovo.setForeground(new Color(255,255,0));

  textNuovo = new JTextField("");

  JButton buttonInvia = new JButton("Invia");
  buttonInvia.addActionListener(this);
  // aggiungiamo gli oggetti sul pannello
  nuovoMex.add(labNuovo,BorderLayout.WEST);
  nuovoMex.add(textNuovo,BorderLayout.CENTER);
  nuovoMex.add(buttonInvia,BorderLayout.EAST);

  this.setLayout(new BorderLayout(0,5));
  add(panLista,BorderLayout.CENTER);
  add(nuovoMex,BorderLayout.SOUTH);

  connetti();
}//fine costruttore classe PannelloChat
```

Concludono la classe due metodi, rispettivamente per avviare il server e predisporre la lista delle possibili connessioni (10 utenti massimo) e la gestione dell'evento sul pulsante di invio che inoltra il messaggio.
Infatti anche l'utente che avvia il server può, di fatto, comunicare con gli altri utenti.

```
public void connetti()
{
  //instanzio il Thread per le connessioni : numero massimo giocatori = 10
  gestioneServizio = new ThreadGestioneServizioChat(10,lista);
}

public void actionPerformed(ActionEvent e)
{
  String bottone = e.getActionCommand();
  if(bottone.equals("Invia"))
  {
   gestioneServizio.spedisciMessaggio(textNuovo.getText());
   textNuovo.setText("");
  }
}
```

# 7 ESERCIZI IN LABORATORIO

Un esempio di esecuzione è il seguente:

In questa prima situazione abbiamo avviato il server e un client e inviato un primo messaggio.
Aggiungiamo ora altri due client e inviamo un messaggio da ciascuno di essi: possiamo osservare come la comunicazione viene inviata a tutte le finestre.

L'unico problema da evidenziare è che non sono presenti i nomi dei mittenti, e quindi "non si sa di chi sono i messaggi"!

## METTITI ALLA PROVA

1 Modifica l'esempio sopra descritto aggiungendo un form di login in modo che l'utente inserisca il proprio nome e questo venga visualizzato nella sezione di output vicino a ogni messaggio, in modo da riconoscerne la "paternità".

2 Quindi fai le opportune modifiche per poter utilizzare questa applicazione in rete, creando uno "sniffer" che ricerca sulla rete l'indirizzo IP del server che offre questo servizio sulla porta 6789.

# ESERCIZI IN LABORATORIO 7

## Esercizi proposti

**1** Progetta e realizza un'applicazione client-server che realizza una chat TPSIT_chat, dove ogni utente interagisce nella chat attraverso un client:
- il server riceve le richieste di connessione da parte dei client e funziona come punto di raccolta dei messaggi;
- ogni client, al momento della richiesta di connessione, chiede all'utente e fornisce al server un *nickname* con cui l'utente comparirà in chat;
- il server invierà agli altri client connessi i messaggi ricevuti da un client specifico, anteponendo a essi il nickname che l'utente ha scelto al momento della connessione;
- il server, *per ogni client*, mantiene una variabile inizializzata a zero, per contare le *interazioni* avvenute con il client nel corso dell'esecuzione. alla ricezione di ogni nuovo messaggio, il server incrementa la variabile di un'unità;
- quando il client si disconnette, il server stampa a video un messaggio finale che riporta il numero di interazioni complessivamente avvenute con il client.

**2** Progetta e realizza un'applicazione client-server per la gestione del social network TPSIT_book:
- i client si connettono al server e inviano il proprio nickname;
- i client possono inviare tre tipologie di comando: LEGGI, POST e COMMENTO:
  - con il comando LEGGI il server gli trasmette gli ultimi 5 POST inseriti dagli altri utenti,
  - con il comando POST <stringa> si invia un messaggio al server per la pubblicazione di un nuovo POST sulla propria bacheca, che viene numerato progressivamente,
  - con il comando COMMENTO <#post><stringa> si invia un messaggio al server per la pubblicazione di un commento al post pubblicato sulla bacheca di un altro utente;
- il server memorizza la bacheca di ogni utente in un file (per esempio, bacheca.mario contiene la bacheca dell'utente Mario) e quando un utente invia un COMMENTO questo viene aggiunto al file che contiene la bacheca dell'utente.

**3** Progetta e realizza l'applicazione client-server TPSIT-drop-box, per l'**upload** e il **download** di file di testo verso/da un server remoto.
In particolare si deve garantire il seguente schema di funzionamento:
- l'utente inserisce le credenziali di login, che memorizza e invia al server dal quale, dopo la connessione, riceve un messaggio di benvenuto;
- l'utente ha a disposizione due attività caricamento di un file (UP_FILE) e scaricamento di un file (DOWN_FILE):
  1. l'utente digita il comando UP_FILE <nomefile>, dove <nomefile> comprende, oltre al nome del file, il percorso completo: il client apre il file indicato, lo legge e lo invia, riga per riga, al server, che lo riceve e lo memorizza sul proprio disco, aggiungendo al nome del file l'ID del mittente (userID_<nomefile>);
  2. l'utente digita il comando DOWN_FILE <nomefile>, dove <nomefile> è il nome di un file precedentemente memorizzato dall'utente sul server: il client inoltra la richiesta al server <nomefile> e ne riceve il contenuto del file, riga per riga; quindi visualizza sullo schermo le righe che riceve e le memorizza nel disco locale;
- il client chiude la connessione e termina la propria esecuzione alla ricezione del comando BYE.

# 8 ESERCIZI IN LABORATORIO

## I SOCKET NEL LINGUAGGIO C

### Il dominio Internet e gli indirizzi IP

La gestione dei socket in linguaggio C è più complessa rispetto a Java: sappiamo che un socket è solitamente legato a un indirizzo e il formato dell'indirizzo dipende dal dominio di comunicazione.
Ciacun dominio ha un suo nome simbolico che inizia con PF_, iniziali di protocol family, altro nome con cui si indicano i domini di comunicazione: a ciascun dominio viene inoltre associata una seconda costante, che inizia invece AF_ (da address family), che identifica il formato degli indirizzi usati in quel dominio.
I principali domini implementati sono descritti in <sys/socket.h> e sono:

- il dominio UNIX (AF_UNIX);
- il dominio Internet (AF_INET, AF_INET6);
- il dominio Novell (AF_IPX);
- il dominio AppleTalk (AF_APPLETALK).

I processi per comunicare nello stesso dominio usano lo stesso formato di indirizzi.

Nel dominio Internet abbiamo due possibili situazioni per un indirizzo IP:

- AF_INET: è costituito da quattro decimali interi separati da punti e consente di individuare univocamente la posizione di una sottorete e di un host all'interno di quest'ultima.
  Viene memorizzato in un record cosi definito nell'header file <netinet/in.h>:

```
struct in_addr
{
  u_long s_addr;              // 4 byte / 32 bit IP V4
}
```

- AF_INET6: utilizza il formato IP V6 per gli indirizzi e quindi saranno necessari per questo dominio record con 6 byte.

Dato che i valori sono memorizzati come numeri, nella libreria <arpa/inet.h> sono disponibili alcune funzioni per la conversione di indirizzi IP sia nella forma dotted-decimal ASCII string (cioè nella forma 192.168.0.1) che nella forma 32-bit network byte ordered (formato binario).

#### inet_addr()

Ritorna un indirizzo IP come numero a 32 bit a partire dal formato stringa dotted decimal.

```
int inet_add (const char *str)
```

#### inet_aton()

Converte la stringa dell'indirizzo dotted decimal in un indirizzo IP e lo memorizza.

```
int inet_aton (const char *str, struct in_addr *addrptr);
```

Scrive nel record puntato da addrptr il valore numerico a 32 bit ottenuto dalla conversione della stringa presente in str. Restituisce zero in caso di errore, 1 se tutto va bene.

# ESERCIZI IN LABORATORIO 8

## inet_ntoa()

Converte un indirizzo IP presente nel record addrptr in una stringa dotted decimal.

```
char* inet_ntoa(struct in_addr addrptr)
```

Restituisce il puntatore alla stringa che contiene l'espressione in formato dotted decimal. Per effettuare l'assegnazione a una variabile è comodo utilizzare la funzione sprintf() che memorizza in una stringa quello che è destinato all'output formattato sullo schermo:

```
char s_IP_Remoto[15];                    // indirizzo formato stringa
struct in_addr remoto_addr               // indirizzo formato numerico
...
sprintf((char*)s_IP_Remoto,"%s", inet_ntoa(remoto_addr));
```

Se volessimo effettuare una assegnazione diretta dovremmo necessariamente eseguire complesse operazioni di casting.

## inet_ntop()

Anche se ufficialmente inet_ntoa() non è deprecata se si vuole salvare l'indirizzo in formato stringa nelle variabili è conveniente utilizzare inet_ntop() che risulta essere maggiormente compatibile nelle versioni di Linux più recenti.

```
char *inet_ntop (AF_INET, const void *src, char *dest, sizeof(dest);
```

Restituisce il puntatore alla stringa che contiene l'espressione in formato dotted decimal che assume il valore NULL in caso di errore.

**ESEMPIO**

```
char s_IP_Remoto[15];                    // indirizzo formato stringa
struct in_addr remoto_addr               // indirizzo formato numerico
inet_ntop(AF_INET, &remoto_addr, s_IP_Remoto, sizeof(s_IP_Remoto));
```

## Strutture dati per i socket

Come abbiamo detto i socket di tipo AF_INET/PF_INET vengono usati per la comunicazione attraverso Internet; la struttura predisposta per memorizzare i dati per un socket Internet è definita nell'header file <netinet/in.h> come:

```
struct sockaddr_in
{
 short sin_family;                       // valore AF_INET
 u_short sin_port;                       // numero porta TCP/UDP (16 bit)
 struct in_addr sin_addr;                // indirizzo IP
 char sin_zero[8];                       // riempimento (non usato)
};
```

dove :

- sin_family deve essere sempre impostato a AF_INET, altrimenti si avrà un errore di EINVAL;
- sin_addr contiene un indirizzo Internet a cui si può accedere sia come struttura sia direttamente come numero intero; se viene impostato a INADDR_ANY si connette automaticamente all'indirizzo locale che ha l'interfaccia di rete dell'host al momento dell'esecuzione del programma;
- sin_port specifica il numero di porta, che deve essere superiore a 1024.

# 8 ESERCIZI IN LABORATORIO

## ESEMPIO

Vediamo per esempio per un server come potrebbe essere composto questo record:

```
// creazione di un elemento di tipo sockaddr_in
struct sockaddr_in datiSocket;
// inizializzazione dell'elemento creato
datiSocket.sin_family = AF_INET;
datiSocket.sin_addr.s_addr = inet_addr( "127.0.0.1" );
datiSocket.sin_port = htons( 27015 )
```

dove la funzione htons() trasforma un numero dal formato interno a quello della rete (Big-Endian).

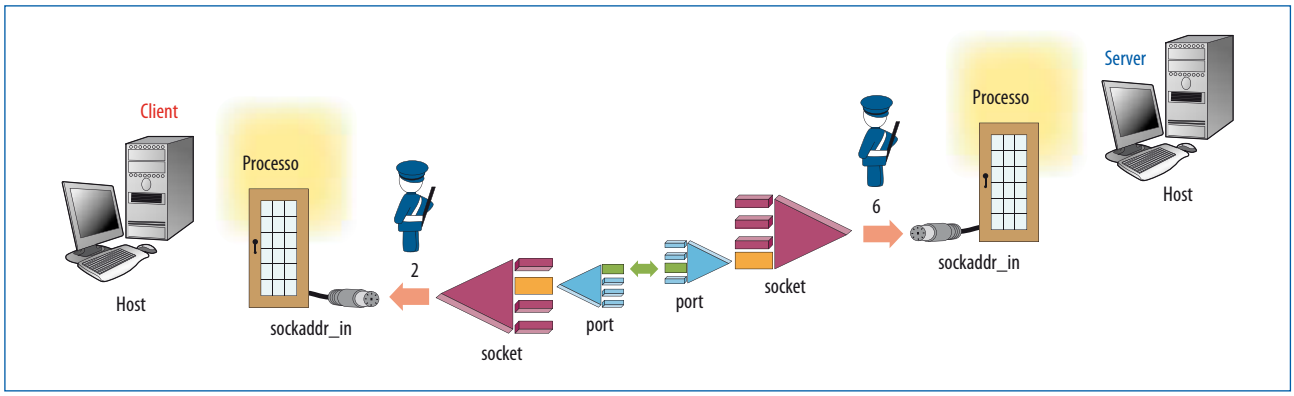
**AREA DIGITALE**

**Funzioni di conversione Big-Endian/Little-Endian**

Per la creazione di un socket server è anche disponibile la definizione di una struttura ridotta:

```
struct sockaddr
{
unsigned short sa_family;          // valore address family (AF_INET)
char sa_data[ ];                   // per memorizzare indirizzo famiglia
};
```

La visione generale dell'architettura client-server è la seguente:




## Comunicazione mediante socket

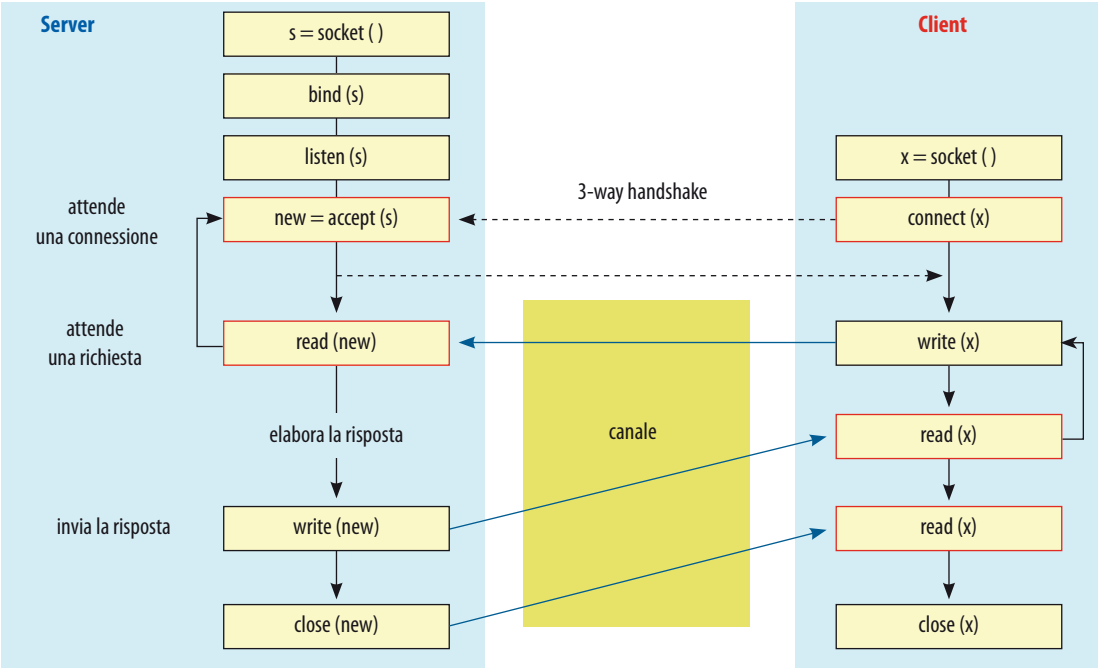
Nella realizzazione della comunicazione utilizzando i socket come primo attore viene avviata l'esecuzione del server che si predispone alla connessione e, solo successivamente, viene avviato il client che chiede la connessione al server che, essendo in esecuzione, la può attivare.
Il "principio di funzionamento che utilizza i socket" è molto simile a quello utilizzato per la gestione dei file:

- apertura del canale;
- lettura/scrittura dei messaggi;
- chiusura del canale.

L'esempio tipico di funzionamento è quello descritto di seguito dove il client spedisce una *richiesta di connessione* al server alla quale quest'ultimo risponde trasmettendo alcuni dati necessari per instaurare la connessione: quindi si procede con una sequenza di trasmissioni bidirezionali che termina quando uno dei due interlocutori decide di interrompere la connessione inviando un apposito comando (mediante la primitiva close()).

# ESERCIZI IN LABORATORIO 8

La sequenza completa delle operazioni è descritta nel diagramma di figura dove sono indicate le diverse funzioni chiamate rispettivamente dal server e dal client.




Descriviamo una per una le singole funzioni.

## Funzione socket()

I socket costituiscono gli estremi del canale di comunicazione e prendono il nome di transport endpoint: i due processi che vogliono comunicare attraverso la rete devono definire i rispettivi socket tramite una chiamata di sistema che restituisce un numero (in una variabile intera indicata con identificatore mysocketfd) maggiore o uguale a zero mentre in caso di errore restituisce (–1) e setta errno con uno dei seguenti valori in modo da riconoscere le cause del fallimento:

- EPROTONOSUPPORT: il tipo di socket o il protocollo scelto non sono supportati nel dominio;
- ENFILE: il kernel non ha memoria sufficiente a creare una nuova struttura per il socket;
- EMFILE: si è ecceduta la tabella dei file;
- EACCES: non si hanno privilegi per creare un socket nel dominio/protocollo specificato;
- EINVAL: protocollo sconosciuto o dominio non disponibile;
- ENOBUFS: non c'è sufficiente memoria per creare il socket(può essere anche ENOMEM).

Quindi la prima azione per "avviare la comunicazione" in rete è la chiamata alla funzione socket() con la specifica del tipo di protocollo da utilizzare, ad esempio TCP con IPv4, UDP con IPv6 ecc.:

```
int socket(int domain, int type, int protocol);
```

dove:

- domain indica il dominio del socket, che per internet è AF_INET;
- type indica la modalità di comunicazione, cioè:
  – SOCK_STREAM se verrà usato il paradigma connection oriented;
  – SOCK_DGRAM se verrà utilizzato il paradigma connection less;
- protocol specifica il protocollo da utilizzare e nel caso che venga lasciato al valore 0 il sistema automaticamente sceglie il protocollo più adatto in base ai primi due parametri.

# 8 ESERCIZI IN LABORATORIO

ESEMPIO

```
#include <sys/socket.h>
// creazione e definizione del socket di tipo stream TCP
mysocketfd = socket(AF_INET, SOCK_STREAM, 0);
```

## Funzione bind()

Per legare l'indirizzo dell'host a un socket identifier in modo da poter essere contattato da un client si utilizza la funzione bind() che "crea un legame" tra il socket identifier e il record che contiene i dati del socket:

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int mysocketfd, const struct sockaddr *address, size_t add_len);
```

dove:
- mysocketfd è il valore restituito dalla system call socket();
- address è il puntatore alla struttura dati che contiene l'indirizzo e la porta del socket;
- add_len è la lunghezza in byte dell'indirizzo.

La funzione restituisce 0 in caso di successo, altrimenti –1.

La funzione bind() viene utilizzata dal server per "effettuare il collegamento" a una porta specifica: se non viene eseguita la chiamata alla funzione bind(), il sistema operativo assegnerà al socket una porta qualunque e uno degli indirizzi IP dell'host.
Inoltre, nel caso di protocollo TCP/IP, se nel record la porta è specificata come zero, il fornitore di servizi assegna una porta tra 1024 e 5000.

Successivamente alla bind() l'applicazione server può usare la funzione getsockname() per apprendere l'indirizzo IP e la porta assegnati.

## Funzione listen()

Il server si mette in ascolto in attesa dei client per mezzo della seguente system call:

```
#include <sys/socket.h>
int listen(int mysocketfd, int queue_size);
```

dove:
- mysocketfd è il socket creato sul server;
- queue_size è il numero massimo di client che si possono accodare in attesa della connessione.

La funzione restituisce 0 in caso di successo e –1 in caso di errore con i seguenti valori restituiti in errno:
- EBADF l'argomento mysocketfd non è un file descriptor valido;
- ENOTSOCK l'argomento mysocketfd non è un socket;
- EOPNOTSUPP il socket è di un tipo che non supporta questa operazione.

# ESERCIZI IN LABORATORIO 8

## ESEMPIO

Utilizzando quanto detto sino a ora, iniziamo a vedere la struttura di un programma server che utilizza i socket:

```
#include <sys/socket.h>
void main(){
 int mysocketfd, ris;                          // definizione variabili
 struct sockaddr_in socketServer;              // record dati del server
// inizializzazione con i dati del server
 socketServer.sin_family = AF_INET;
 socketServer.sin_addr.s_addr = inet_addr( "127.0.0.1" );
 socketServer.sin_port = htons( 27015 )

// creazione e definizione del tipo di socket (stream TCP)
 mysocketfd = socket(AF_INET, SOCK_STREAM, 0);

// assegnazione dell'indirizzo al socket
 ris = bind(mysocketfd, (struct sockaddr*) &socketServer, sizeof(socketServer));

//  pone il server in ascolto sul socket
 ris = listen(mysocketfd, 10 );
}
```

## Funzione accept()

Il server accetta una connessione tramite la chiamata di sistema `accept`():

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int mysocketfd, struct sockaddr* address, size_t* add_len);
```

La funzione estrae la prima connessione dalla coda di pendenza delle connessioni sul socket in input: crea un nuovo socket che verrà usato per comunicare con il client e il suo valore viene passato come parametro di ritorno di accept().

Il socket in input non cambia e continua a restare in ascolto per nuove richieste.

address e add_len sono usati dal server per ottenere l'indirizzo del client da cui proviene la richiesta di connessione.
La funzione restituisce un numero di socket descriptor positivo in caso di successo e –1 impostando errno ai valori:

- EBADF l'argomento mysocket non è un file descriptor valido;
- ENOTSOCK l'argomento mysocket non è un socket;
- EOPNOTSUPP il socket è di un tipo che non supporta questa operazione;
- EAGAIN o EWOULDBLOCK il socket è stato impostato come non bloccante e non ci sono connessioni in attesa di essere accettate.

Aggiungiamo all'esempio precedente il comando accept():

```
// definizione delle variabili
int clientSocketfd;                    // identificatore del socket client
struct sockaddr_in client;             // struttura del client address
int clientLen;                         // dimensione del client address
…
clientLen = sizeof(client);
clientSocketfd = accept(mysocketfd,(struct sockaddr*)&client, &clientLen);
```

# 8 ESERCIZI IN LABORATORIO

Prima della chiamata della funzione accept() deve essere inizializzato clientLen alle dimensioni della struttura il cui indirizzo è passato come argomento.

## Funzione close()

Un socket viene distrutto chiudendo il file descriptor associato alla connessione.
La funzione standard UNIX close() che si usa sui file può essere usata con lo stesso effetto anche sui file descriptor associati a un socket.

```
#include <unistd.h>
int close(int mysocketfd);
```

L'azione di questa funzione quando applicata a un socket è di marcarlo come chiuso e ritornare immediatamente al processo: dopo la sua esecuzione il socket descriptor non è più utilizzabile dal processo e non può essere usato come argomento per le operazioni di lettura e/o scrittura.

**ESEMPIO**

```
/* chiusura*/
close(mysocketfd);
```

## L'ambiente di sviluppo e di esecuzione

Come già descritto nel volume 2, UDA1, Esercizi in Laboratorio 1 a pag. 59, se non si dispone di una partizione con Linux è possibile utilizzare l'emulatore Cygwin che è installabile seguendo le istruzioni presenti alla pagina http://www.cygwin.com.
Nei nostri esempi faremo riferimento a entrambe le situazioni.
Dato che tutti i programma richiameranno lo stesso gruppo di librerie e utilizzeranno alcuni parametri comuni, predisponiamo un file librerie.h che richiameremo come prima istruzione nei nostri programmi.

```
#ifndef __LIBRERIE__
#define __LIBRERIE__
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <type.h>
#include <time.h>          // timespec{}
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>      // timeval{}
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#define IPADDRESS "127.0.0.1" // indirizzo del server
#define SERVER_PORT 1313      // numero di porta del server
#define MAXLINE 256           // dimensione massima della riga
#define MAXSIZE 10            // dimensione del messaggio
#define LINESIZE 80           // dimensione della riga
#define BACKLOG 10            // max numero di client
#define SIZEBUF 10000         // dimensione dei buffer
#define SOCKET_ERROR ((int)-1)
#endif
```

# ESERCIZI IN LABORATORIO 9

## SERVER TCP IN C

### Funzioni per inviare e ricevere dati

Nella libreria <sys/socket.h> sono disponibili due gruppi di funzioni per effettuare le operazioni di comunicazione con il server: noi le useremo entrambe per testarne le funzionalità.

#### Funzioni read() e write()

##### read()

La struttura di una istruzione di read() è la seguente:

```
int read (int socketfd, void *buffer, int tanti);
```

Cerca di leggere tanti byte dal file descriptor socketfd, scrivendoli nel buffer puntato da buffer: se tanti è zero la read() restituisce zero altrimenti viene effettuata la lettura e viene restituito il numero di byte letti, e se il valore di ritorno è:

- maggiore di zero, significa che tutto è OK;
- uguale a **0 (zero)**, significa end-of-file (fine stream), cioè l'altro end system ha volutamente chiuso la connessione e il socket non può essere più utilizzato per leggere;
- uguale a **–1**, significa che si è verificato un errore e viene settata la variabile globale errno definita in <errno.h> con un valore che indica quale errore è avvenuto.

La funzione read(), applicata a un socket, presenta una particolarità: può accadere che la read() restituisca meno byte di quanti richiesti, anche se lo stream è ancora aperto e questo si verifica nel caso in cui il buffer a disposizione del socket nel kernel è stato esaurito. In questo caso è necessario ripetere la read() richiedendo il numero dei byte mancanti.

##### write()

La struttura di una istruzione di write() è la seguente:

```
int write (int socketfd, void *buffer, int tanti);
```

Cerca di scrivere fino a tanti byte nel buffer di sistema corrispondente al file descriptor socketfd perché siano poi trasmessi leggendoli dal buffer puntato da buffer.

#### Funzioni send() e recv()

Esistono altre due funzioni per trasmettere e ricevere dati attraverso i socket che oltre a effettuare le medesime operazioni delle coppia read()/write() offrono la possibilità di configurare in modo più specifico il comportamento in casi particolari utilizzando un ulteriore parametro (int flag): sono le system call send() e recv().

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int mysockedfd, const char *buffer, int length, int flag);
int recv(int mysockedfd, void *buffer, int length, int flag);
```

dove:

- buffer contiene la stringa da trasmettere;
- length è la dimensione in byte del buffer;
- flag assume valori diversi per modificare il funzionamento della funzione:
  - 0: send() e recv() equivalgono, rispettivamente alle system call write() e read();
  - altri valori per la send():
    - MSG_OOB: il processo invia dati "out of band";

# 9 ESERCIZI IN LABORATORIO

- MSG_DONTROUTE: vengono ignorate le condizioni di routing dei pacchetti sottostanti al protocollo utilizzato;

– altri valori per la recv():
- MSG_PEEK: i dati vengono letti, ma non "consumati" in modo che una successiva `recv`() riceverà ancora le stesse informazioni;
- MSG_OOB: legge soltanto i dati "out of band";
- MSG_WAITALL: la recv() ritorna soltanto quando la totalità dei dati è disponibile.

## Realizzazione di un server

Per poter collaudare un sistema client-server deve naturalmente essere scritto il codice di entrambi gli attori: come primo esempio completo creiamo una applicazione connection oriented dove si vuole realizzare un sistema con un server che riceve un carattere dal client e lo restituisce trasformato in maiuscolo fino a che gli arriva il carattere "x", che chiude la trasmissione.

In questa esercitazione realizzeremo il server, mentre nell'Esercizio in Laboratorio 10 descriveremo il client.

Dopo aver incluso le librerie indichiamo come costante il carattere che farà terminare la comunicazione: scegliamo il carattere "X" con valore ASCII 88.

```
serverMaiu.c
1  #include "librerie.h"                    // include elenco librerie
2  #define FINE_RICE 88                     // valore ASCII carattere X
3
4  // funzione che riceve un carattere, lo trasforma in maiuscolo, lo ritrasmette
5  int upperiChar(int in, int out);
```

Il main() definisce alcune variabili di utilità e la struttura del socket:

```
7  int main (unsigned argc, char **argv) {
8    int socketAttesa, client_len, socketComunica;
9    int asciiletto;                        // valore ASCII ultimo char ricevuto
10   struct sockaddr_in server, client;     // record con i dati del server e del client
11   printf ("Server che trasforma un carattere in maiuscolo(usare come client Telnet)\n");
```

Viene quindi effettuata la sua inizializzazione:

```
13   // impostazione del transport endpoint del server
14   printf ("socket()\n");
15   if((socketAttesa = socket(AF_INET, SOCK_STREAM, 0)) == -1){
16     perror("chiamata della system call socket() fallita");
17     return(1);
18   }
```

```
19   // definizione dei dati del socket
20   server.sin_family = AF_INET;
21   server.sin_addr.s_addr = htonl(INADDR_ANY);
22   server.sin_port = htons(SERVER_PORT);
```

Viene effettuato il bind() passando come parametro il socket e la sua dimensione:

```
24   // leghiamo l'indirizzo al transport endpoint
25   printf ("bind()\n");
26   if (bind(socketAttesa, (struct sockaddr*)&server, sizeof server) == -1){
27     perror("chiamata della bind() fallita");
28     return(2);
29   }
```

# ESERCIZI IN LABORATORIO 9

Quindi ci si pone in attesa di un client:

```
31    // poniamo il server in ascolto delle richieste dei client
32    printf ("listen()\n");
33    listen(socketAttesa, 10);            // puo'attendere fino a 10 client
```

Alla richiesta di connessione da parte di un client questa viene valutata:

```
35    // gestione delle connessioni dei client
36    printf ("accept()\n");
37    // trasferimento della comunicazione su un nuovo socket dedicato per il client
38    client_len = sizeof(client);
39    if ((socketComunica = accept(socketAttesa, (struct sockaddr*)&client, &client_len)) < 0){
40      perror("connessione non accettata");
41      return(3);
42    }
```

e quando l'esito è positivo si esce dal ciclo di attesa e inizia la comunicazione;

```
44    // inizio dialogo tra server e client
45    fprintf(stderr, "Aperta connessione col server.\n");  // echo nel client
46    send(socketComunica, "Hello dal ServerMaiu: trasformo caratteri in maiuscolo!\n", 56, 0);
47
48    // ciclo di attesa/comunicazione di un carattere alla volta col client
49    do     // funzione che riceve un carattere, lo trasforma in maiuscolo, lo ritrasmette
50      asciiLetto = upperChar(socketComunica, socketComunica);
51    while (asciiLetto != FINE_RICE);      // se  arriva x termina l'elaborazione
```

Quando riceve una "x" la comunicazione termina e il server chiude la connessione:

```
53    // fine della comunicazione
54    close(socketComunica);
55    fprintf(stderr, "Chiusa connessione col server.\n");  // echo nel client
56  }
```

Manca ora da riportare la funzione che effettua la ricezione dei dati e li trasforma in maiuscolo; una possibile codifica è la seguente:

```
58  int upperChar(int in, int out){
59    char ch, chMaiu;
60    int len, asciiVal;
61    if(recv(in, &ch, 1, 0) > 0){          // riceve un carattere
62      chMaiu  = toupper(ch);              // lo trasforma in maiuscolo
63      asciiVal = (unsigned int)ch;        // prende il valore ASCII
64      printf("\nil carattere ricevuto e': %c con valore ASCII %d ", ch, asciiVal);
65      printf(" maiuscolo %d - %c\n", (unsigned int)chMaiu, chMaiu);
66      send(out, &chMaiu, 1, 0);           // invia il carattere maiuscolo
67      asciiVal= (unsigned int)chMaiu;
68    }
69    return asciiVal;
70  }
```

Il codice completo del server è memorizzato nel file serverMaiu.c.

## Avvio del server

Verifichiamo il funzionamento del programma: per prima cosa ci posizioniamo nella directory socket dove è presente il codice sorgente serverMaiu.c e lo compiliamo.

```
PaoloPCwin8 ~
$ cd ua2vol3

PaoloPCwin8 ~/ua2vol3
$ gcc -o serverMaiu serverMaiu.c

PaoloPCwin8 ~/ua2vol3
$ |
```

# 9 ESERCIZI IN LABORATORIO

Se ci viene segnalata la mancanza di qualche libreria con un messaggio simile a quello riportato:

```
Messaggio
[Error] sys/socket.h: No such file or directory
compilation terminated.
```

significa che abbiamo erroneamente avviato la compilazione in ambiente Dev-cpp dove le librerie indicate non sono presenti dato che riguardano socket in ambiente Linux.

Avviamo il server e come echo ci riporta la sequenza delle funzioni che ha eseguito ponendosi infine in attesa della richiesta da parte di un client:

```
Paolo@PCwin8 ~/ua2vol3
$ ./serverMaiu
socket()
bind()
listen()
accept()
```

## METTITI ALLA PROVA

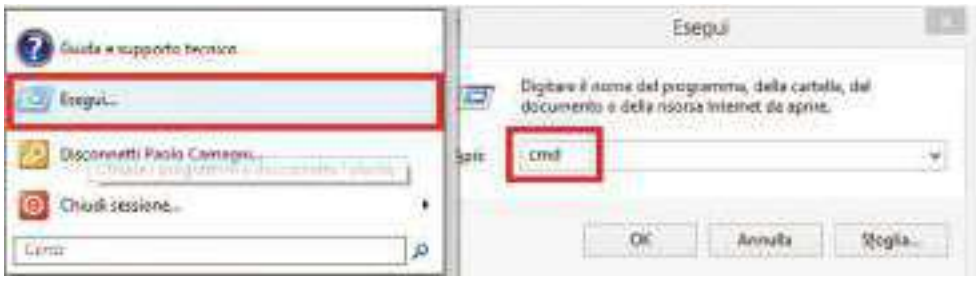
• Server TCP • Funzioni read() e write()

Modifica l'esempio descritto utilizzando per la comunicazione le funzioni read() e write() prima descritte.

Confronta il tuo codice con quello riportato nel file serverMaiuRWSol.c.

## Telnet come client

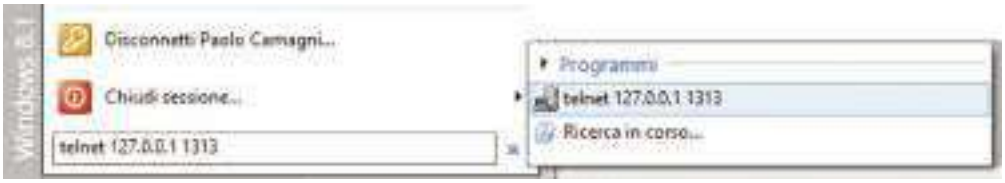
Per verificare il funzionamento del server, dato che non abbiamo ancora realizzato il codice del client, possiamo servirci del programma telnet posizioniamoci nella finestra di comando:




Al prompt dei comandi digitiamo: telnet 127.0.0.1 1313.

```
Microsoft Windows [Versione 6.3.9600]
(c) 2013 Microsoft Corporation. Tutti i diritti riservati.

C:\Windows\System32>telnet 127.0.0.1 1313_
```

È anche possibile richiamare e avviare telnet direttamente dalla riga di ricerca:



# ESERCIZI IN LABORATORIO 9

Confermiamo con invio e se la connessione va a buon fine avremo i messaggi che seguono.
Sul server:

```
$ cd ua2vol3

$ ./serverMaiu
socket()
bind()
listen()
accept()
Aperta connessione col server.
```

Sul client:

```
Hello al ServerMaiu: trasformo caratteri in maiuscole!
```

Qualunque carattere che ora digitiamo ci viene "rispedito" convertito in maiuscolo:

```
Hello al ServerMaiu: trasformo caratteri in maiuscole!
```

e sul server avremo l'echo dei caratteri inviati dal client.

Per terminare la sessione premere CTRL + e impartire nella nuova riga il comando quit:

```
Microsoft Telnet Client

Il carattere di Escape è 'CTRL++'

Microsoft Telnet> quit
```

Nella finestra dove abbiamo in esecuzione il server viene visualizzato un messaggio sia nel momento dell'inizio della connessione telnet che alla chiusura oltre agli echo delle singole istruzioni che vengono eseguite:

```
$ cd ua2vol3

$ ./serverMaiu
socket()
bind()
listen()
accept()
Aperta connessione col server.

Il carattere ricevuto e': a con valore ASCII 97  maiuscolo 65

Il carattere ricevuto e': b con valore ASCII 98  maiuscolo 66

Il carattere ricevuto e': c con valore ASCII 99  maiuscolo 67

Il carattere ricevuto e': d con valore ASCII 100  maiuscolo 68

Il carattere ricevuto e': s con valore ASCII 115  maiuscolo 83
Chiusa connessione col server.
```

## METTITI ALLA PROVA

**AREA DIGITALE**

**Abilitare telnet in Windows 10**

• Realizzare un server • Realizzare un client • Comunicare tra server e client

1 Modifica l'esempio descritto in modo che la terminazione avvenga quando il server riceve un qualunque carattere non alfabetico.

2 Successivamente bufferizza i caratteri che vengono ricevuti dal server, per esempio in un vettore di caratteri:

```
#define LINESIZE 80                    // dimensione della linea di caratteri
char inputline[LINESIZE];
```

Questo viene spedito interamente convertito al client al termine della ricezione.

# 10 ESERCIZI IN LABORATORIO

## CLIENT TCP IN C

### Struttura di un client

Negli esercizi precedenti abbiamo visto come avviene la procedura di comunicazione tra server e client. Per il client parte delle operazioni sono identiche a quelle effettuate sul server nell'Esercizio precedente, come l'inizializzazione del socket con la definizione del transport endpoint del client.

#### Operazioni preliminari alla connessione

Nel client devono però essere definiti due record di tipo sockaddr_in in quanto uno deve contenere i dati del client e l'altro i dati del server al quale deve connettersi.
Per prima cosa includiamo le librerie e definiamo le costanti:

```
1  #include "librerie.h"                    // include elenco librerie
2
```

Successivamente nel main() definiamo le variabili di servizio:

```
3  int main(int argc, char *argv[]){
4    char indirizzoServer[] = "127.0.0.1";     // indirizzo del server
5    char messaggio[] = "ciaox";               // messaggio da inviare
6    char ch, chMaiu;                          // singolo carattere inviato e ricevuto
7    int ris, nwrite, lungo;
8
9    char buffer[MAXSIZE];                     // messaggio ricevuto
10   for (int x=0; x<MAXSIZE; x++)             // inizializzo buffer
11     buffer[x]=' ';
```

Definiamo ora le variabili necessarie per la connessione e ne inseriamo i valori:

```
13   int socketfd;                             // identificatore del socket
14   struct sockaddr_in locale, remoto;        // dati del socket
15   // settaggio del socket locale
16   locale.sin_family      = AF_INET;
17   locale.sin_addr.s_addr = htonl(INADDR_ANY);
18   locale.sin_port        = htons(0);
19   // assegnazione parametri del server
20   remoto.sin_family      = AF_INET;
21   remoto.sin_addr.s_addr = inet_addr(indirizzoServer);
22   remoto.sin_port        = htons(SERVER_PORT);
```

Naturalmente i valori di indirizzoServer e SERVER_PORT devono essere noti al 3 per poter richiedere la connessione al server.

Creiamo il socket sul client:

```
24   // impostazione del transport endpoint
25   printf("creazione del socket()\n");
26   socketfd = socket(AF_INET, SOCK_STREAM, 0);
27   if (socketfd == SOCKET_ERROR){
28     printf ("fallito socket(), errore: %d \"%s\"\n", errno, strerror(errno));
29     return(1);
30   }
```

# ESERCIZI IN LABORATORIO 10

Effettuiamo quindi le operazioni di bindig:

```
32    // Leghiamo l'indirizzo al transport endpoint
33    printf("bind()\n");
34    ris = bind(socketfd, (struct sockaddr*) &locale, sizeof(locale));
35    if (ris == SOCKET_ERROR){
36      printf ("fallito il bind(), errore: %d \"%s\"\n", errno, strerror(errno));
37      fflush(stdout);
38      return(3);
39    }
```

Siamo quindi pronti per effettuare la connessione con il server.

## Connessione con il server: la funzione connect()

Mentre il codice del server a questo punto del programma si mette in attesa che "un qualche client" lo contatti, il client deve effettuare la richiesta di connessione e questa viene eseguita mediante la funzione connect(); il client si connette al server in corrispondenza del socket indicato dalla struttura che contiene gli indirizzi del server remoto:

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int socketfd, const struct sockaddr *remoto, sizeof(remoto));
```

La funzione connect() restituisce zero in caso di successo e –1 per un errore, nel qual caso errno assumerà i seguenti valori:

- ECONNREFUSED: non c'è nessuno in ascolto sull'indirizzo remoto;
- ETIMEDOUT: si è avuto timeout durante il tentativo di connessione;
- ENETUNREACH: la rete non è raggiungibile;
- EINPROGRESS: il socket è non bloccante e la connessione non può essere eseguita;
- EALREADY: il socket è non bloccante e un tentativo precedente non si è ancora concluso;
- EAGAIN: non ci sono più porte locali libere;
- EAFNOSUPPORT: l'indirizzo non ha una famiglia di indirizzi corretta nel relativo campo;
- EACCES, EPERM: si è tentato di eseguire una connessione a un indirizzo broadcast senza che il socket fosse stato abilitato per il broadcast.

Il client del nostro esempio effettua la seguente richiesta:

```
41    // richiesta di connessione al server
42    printf("connect()\n");
43    ris = connect(socketfd, (struct sockaddr*) &remoto, sizeof(remoto));
44    if (ris == SOCKET_ERROR){
45      printf("fallita la connect(), errore: %d \"%s\"\n", errno, strerror(errno));
46      fflush(stdout);
47      return(4);
48    }
```

Se tutto è andato a buon fine, i due programmi procederanno con lo scambio di dati fino alla chiusura della connessione, che viene eseguita mediante la seguente istruzione:

```
65    // chiusura socket
66    close(socketfd);
67    return(0);
68  }
```

# 10 ESERCIZI IN LABORATORIO

## Operazioni di lettura e scrittura

Vediamo ora ad esempio il segmento di codice che ci permette di inviare un messaggio al server, di riceverlo elaborato dal server, e quindi di memorizzarlo nel buffer.
Come primo esempio inviamo il messaggio un carattere alla volta; prima di trasmettere un nuovo carattere attendiamo che il precedente ci arrivi convertito in maiuscolo dal server:

```
50  // ciclo che invia la stringa (1 char alla volta) al server
51  lungo = strlen(messaggio);
52  printf("\nstringa lunga: %d \n", lungo);
53  for (nwrite = 0; nwrite < lungo; nwrite++){
54    write(socketfd, &(messaggio[nwrite]), 1);      // invia un carattere
55    if (read(socketfd, &chMaiu, 1) > 0){           // lettura della risposta
56      printf("carattere ricevuto: %c\n", chMaiu); // stampa singolo carattere
57      buffer[nwrite] = chMaiu;
58    }
59  }
```

Il codice non necessita di ulteriori chiarimenti: l'unica osservazione sta nel fatto che se nel messaggio che stiamo inviando è presente il carattere "x" il server cessa la sua elaborazione dato che questo viene interpretato come "fine trasmissione!".
Il codice completo del client è memorizzato nel file main1x1Client.c.

Per verificare il funzionamento del programma è necessario modificare il server descritto nel precedente laboratorio togliendo la frase di saluto inviata alla riga 46 in quanto verrebbe contata dal client come carattere inviato al client: il codice del server è memorizzato nel file main1x1Server.c.

### METTITI ALLA PROVA

• Comunicazione client-server • Funzioni connect()

1. Completa il codice descritto inserendo l'invio di un carattere di chiusura "x" da parte del client in modo da far terminare il programma anche nel server una volta che il messaggio è stato completamente trasformato in maiuscolo.
2. Modifica client e server facendo in modo che possa inviare una qualunque parola digitata dall'utente e ne attenda la conversione in maiuscolo.
   Confronta la tua soluzione con quella presente nei file mainParolaClientSol.c e mainParolaServerSol.c.
3. Modifica il client facendo in modo che riceva come parametro sia l'indirizzo del server che la porta alla quale connettersi.
   Confronta la tua soluzione con quella presente nel file mainParola2ClientSol.c.
4. Successivamente modifica il programma main1x1Server.c in modo che accetti più connessioni contemporaneamente (utilizzando la fork).

## Un server che fornisce data/ora

Realizziamo un'applicazione server che invia ai client la data e l'ora di sistema.
Scriviamo dapprima il codice del client, che risulta essere più semplice del precedente in quanto non invia nessuna dato al server ma riceve semplicemente il dato che questo gli invia.

# ESERCIZI IN LABORATORIO 10

client_ data Ora

apre la connessione con il server
ripete finché *end-of-file*:
    ricevi testo
    stampa la data
chiude la connessione
esce

server_ data Ora

crea socket d'ascolto e si pone in attesa
ripete *forever*.
    accetta nuova connessione,
    usa socket di connessione
    invia la data
    chiude il socket del client

Utilizziamo sempre la porta 1313 come nell'esempio precedente e, perciò, il record di identificazione del server contiene gli stessi dati; inoltre, creiamo un server che risponde a un unico client e successivamente termina l'esecuzione.

Analizziamo in parallelo il codice client e server che effettua le medesime operazioni, cioè vediamo passo passo il meccanismo di handshake.

## Passo 1

Definiamo dapprima i parametri dei socket di comunicazione:

Client

```
10  // parametri per il socket del server
11  remoto.sin_family = AF_INET;                          // assegna il tipo di indirizzo
12  remoto.sin_addr.s_addr = inet_addr(indirizzoServer); // assegna indirizzo del server
13  remoto.sin_port = htons(SERVER_PORT);                 // assegna la porta del server
```

Server

```
10  // parametri per il socket del server
11  memset((void *)&server, 0, sizeof(server));
12  server.sin_family = AF_INET;
13  server.sin_addr.s_addr = htonl(INADDR_ANY); // il server accetta connessioni su
14                                              // una qualunque sua interfacce di rete
15  server.sin_port = htons(SERVER_PORT);       // numero di porta del server
```

## Passo 2

Quindi creiamo di socket:

Server

```
17  // impostazione del transport endpoint del server
18  if ((socketAttesa = socket(AF_INET, SOCK_STREAM, 0)) < 0) { // crea il socket
19    perror("errore in socket");
20    exit(-1);
21  }
22  else
23    printf ("socket creato\n");
```

Client

```
15  // si imposta il transport endpoint
16  printf("Client: creazione del socket()\n");
17  if ((socketClient = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
18    perror("errore in socket");
19    exit(-1);
20  }
```

# 10 ESERCIZI IN LABORATORIO

## Passo 3

Dopo aver assegnato l'indirizzo al socket con la funzione bind():

```
25    // assegna l'indirizzo al socket
26    if (bind(socketAttesa, (struct sockaddr *) &server, sizeof(server)) < 0) {
27      perror("errore in bind");
28      exit(-1);
29    }
30    else
31      printf ("bind()\n");
```

il server si pone in attesa dei client tra i BACKLOG possibili:

```
33    // poniamo il server in ascolto delle richieste del client
34    if (listen(socketAttesa, BACKLOG) < 0 ) {
35      perror("errore in listen");
36      exit(-1);
37    }
38    else
39      printf("listen()\n");
```

## Passo 4

Il client prova a effettuare la connessione:

```
22    // si  stabilisce la connessione con il server
23    printf("prova la connect()\n");
24    if (connect(socketClient, (struct sockaddr *) &remoto, sizeof(remoto)) < 0) {
25      perror("errore in connect");
26      exit(-1);
27    }
```

e quando tale richiesta arriva al server, questo crea uno specifico socket per il client "accettato":

```
40
41    // trasferimento della comunicazione su un nuovo socket dedicato per il client
42    printf ("in attesa di un client su accept()\n");
43    if ((socketComunica = accept(socketAttesa, (struct sockaddr *) NULL, NULL)) < 0){
44      perror("errore in accept");
45      exit(-1);
46    }
47    else
48      printf ("apre connessione col client ...\n");
49
```

## Passo 5

Il client si mette in attesa della risposta e quando lo riceve lo visualizza:

```
29    // si legge dal socket fino a quando non trova EOF
30    printf("inizio la lettura \n");
31    while ((n = read(socketClient, datoRicevuto, MAXLINE)) > 0){
32      datoRicevuto[n] = 0;                    // aggiunge il char di terminazione
33      if (fputs(datoRicevuto, stdout) == EOF){  // stampa il dato sullo stdout
34        perror("errore in fputs");
35        exit(-1);
36      }
37    }
```

# ESERCIZI IN LABORATORIO 10

contemporaneamente il server prepara il messaggio da inviare al client e lo scrive nel socket:

```
50    // prepara il dato da inviare mettendolo nel buffer
51    dataOra = time(NULL);        // Legge l'orario usando la chiamata di sistema time
52    snprintf(buffer, sizeof(buffer), "%.24s\r\n", ctime(&dataOra));
53
54    // invia il dato presente nel buffer
55    if (write(socketComunica, buffer, strlen(buffer)) != strlen(buffer)) {
56      perror("errore in write");
57      exit(-1);
58    }
59    else
60      printf (" - dato inviato\n");
```

## Passo 6

Per comodità facciamo terminare le operazioni del server dopo che ha soddisfatto un solo client: quindi come ultima operazione chiudiamo le connessioni in entrambi i processi.

Server

```
62    // chiusura della connessione
63    if (close(socketComunica) == -1) { //chiude la connessione
64      perror("errore in close");
65    }
66    else{
67      printf ("chiusura connessione col client\n");
68    }
69    close(socketAttesa);
70    exit(0);
71  }
```

Client

```
39    // chiusura socket
40    printf("chiusura connessione \n");
41    close(socketClient);
42    exit(0);
43  }
```

Riportiamo quanto è presente sullo schermo rispettivamente del server e del client.

```
Sono il server della data-ora
socket creato
bind()
listen()
accept()
apre connessione col client ...
 - dato inviato!
chiusura connessione col client
apre connessione col client ...
 - dato inviato!
chiusura connessione col client
apre connessione col client ...
 - dato inviato!
chiusura connessione col client
```

```
$ ./dataC
Client: creazione del socket()
prova la connect()
inizio la lettura
Mon Oct  7 18:08:54 2010
chiusura connessione
```

Il codice completo del client è memorizzato nel file dataOraClient.c, mentre quello del server è memorizzato nel file dataOraServerTCP.c.

# 10 ESERCIZI IN LABORATORIO

## METTITI ALLA PROVA

Abbiamo scritto il server in modo tale da rimanere in attesa in un ciclo infinito, modifica ora il sistema in modo che:
**a** il server chiude la connessione dopo aver servito 5 client;
**b** il server chiude la connessione dopo che da un client arriva un messaggio di chiusura <END>.

## Esercizi proposti

**1** Modifica il client in modo che legga da terminale un carattere digitato dall'utente e lo invii direttamente al server utilizzando la funzione getche(), in modo da ottenere una situazione simile a quella riportata nella figura:

```
Paolo@PCwin8 ~/ua2vol3
$ gcc -o cms clientMaiuSol.c

Paolo@PCwin8 ~/ua2vol3
$ ./cms
Programma che invia un carattere letto con getche() e lo riceve in maiuscolo
Aperta la connessione col server:

Inserisci il carattere da inviare: a -> A
Inserisci il carattere da inviare: b -> B
Inserisci il carattere da inviare: f -> F
Inserisci il carattere da inviare: x -> X

chiusura socket: termine elaborazione.
```

Confronta la tua soluzione con quella presente nel file clientMaiuSol.c.

**2** Realizza un client (e il corrispondente server) con queste specifiche.
Accetta all'avvio una riga di comando con la seguente sintassi:

```
clienteA.exe [IP] [porta] [messaggio]
```

**Per esempio**

```
clienteA.exe 139.204.72.5 1313 messaggiodiprova).
```

Si connette a IP+porta .
Legge una stringa A di 10 byte trasmessi dal server (p.e.: 0000001313).
Invia una stringa B di 100 byte concatenando il proprio nome, cognome, telefono alla stringa A, riempiendo con gli spazi i caratteri mancanti e ricevendo un saluto col proprio cognome e nome con le iniziali in maiuscolo.

**Per esempio**

```
invio
0000001313 giuseppe, garibaldi, 333987654
ricezione
Hello 1313, Giuseppe Garibaldi
```

Invia un intero C di 4 byte (header) indicando quanti caratteri N spedirà in futuro.

```
N=strlen(messaggio)
```

**Per esempio**

```
int N = strlen(messaggiodiprova)
```

Invia una stringa D di N caratteri con la parola presa come parametro.

# ESERCIZI IN LABORATORIO 10

Legge una stringa E di 2 bytes:
– se E="OK" il client ha avuto successo;
– se E="ER" il client ha commesso un errore nel protocollo.
Stampa a console la stringa E.
Chiude la connessione.

**3** Realizza un sistema client-server che implementa una calcolatrice:
- il client si connette al server;
- il server ITERATIVO accetta la connessione e scrive nel file connessioni.log i dati del client (IP, PORTA e ora della connessione);
- il client invia al server 2 interi e l'operazione da effettuare (struttura);
- il server effettua l'operazione e invia il risultato al client;
- il server scrive nel file connessioni.log il tipo di operazione effettuata;
- il client stampa a video il risultato ricevuto dal server e si pone in attesa di inviare altri dati… quindi esce con ^D;
- quando il client esce, il server scrive nel file connessioni.log l'ora di uscita del client;
- prima di terminare del tutto, il server stampa a video lo storico delle operazioni effettuate fino a quel momento.

**4** Scrivi un'applicazione client-server per raccogliere le iscrizioni a una gara di atletica (per esempio per una maratona): l'utente (client) richiede di effettuare l'iscrizione e il server gli comunica il numero di pettorina.
Si ipotizzi di avere tre categorie:
- junior: età inferiore a 16 anni;
- senior: età superiore ai 60 anni;
- medium: tutti gli altri.

Il concorrente comunica, oltre al suo nome, anche la sua età e il numero che gli viene assegnato che è un progressivo del tipo J23, S23 oppure M345 a seconda della categoria alla quale viene iscritto.

**5** Realizza un sistema client-server che implementa un "dispenser di numeri":
- il server conta il numero di client che accedono al suo servizio;
- il client contatta il server per conoscere tale numero;
- il client invia un messaggio ASCII stampabile.

Il server termina l'elaborazione dopo 5 client serviti.

**6** Realizza un sistema client-server che implementa una "calcolatore importo posteggio":
- il client si connette al server (entra nel "posteggio");
- il server ITERATIVO accetta la connessione e scrive nel file connessioni.log i dati del client (IP, PORTA e ora della connessione);
- il client si connette una seconda volta al server (esce dal "posteggio");
- il server effettua il calcolo del "tempo di posteggio" e invia l'importo da pagare al client;
- il client riceve l'importo da pagare ed effettua il pagamento inviando un importo;
- il server controlla l'importo ricevuto: se è superiore invia in resto al client assieme all'<OK> per uscire dal posteggio";
- quando il client riceve <OK>, termina l'elaborazione;
- quando tutti i client sono usciti dal posteggio, il server chiude la connessione.

# 11 ESERCIZI IN LABORATORIO

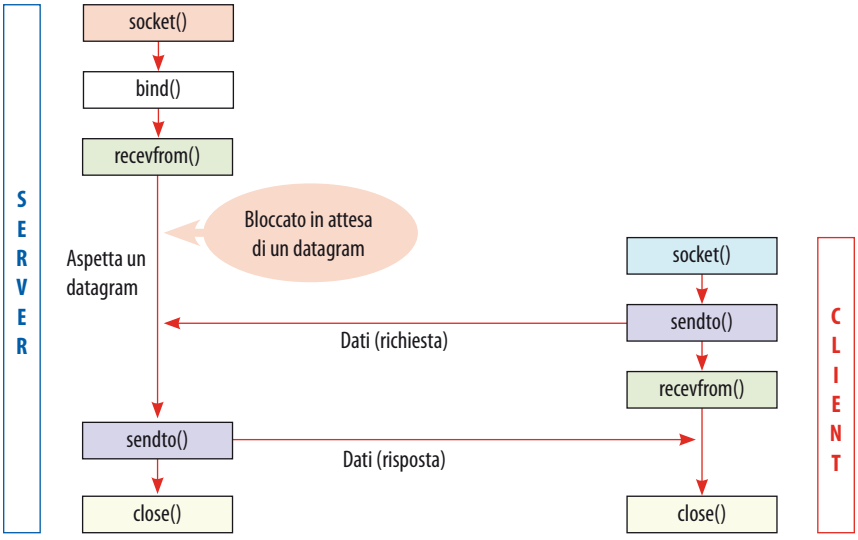
## IL PROTOCOLLO UDP NEL LINGUAGGIO C

### Struttura del protocollo UDP

Realizziamo una comunicazione mediante protocollo UDP dove un mittente (inviaUDP.c) trasmette un messaggio (datagram) a un destinatario (riceviUDP.c).
Ricordiamo che in una comunicazione dati connectionless o datagram il canale:

- trasporta messaggi;
- non è affidabile;
- il socket è condiviso;
- non preserva l'ordine delle informazioni.

La struttura di un'applicazione UDP è la seguente:




Per leggere o scrivere su un socket UDP si utilizzano funzioni di sistema differenti da TCP che devono specificare l'indirizzo del server e la lunghezza del messaggio; descriviamole nel dettaglio.

### Funzioni sendto() e recvfrom()

Per usare UDP è necessario utilizzare le funzioni sendto() e recvfrom() in quanto in UDP *non esiste* il concetto di connessione e quindi non è possibile usare direttamente read() e write() non avendo già stabilito quale sia la sorgente e quale la destinazione dei dati. Le funzioni sendto() e recvfrom() sono utilizzabili per qualunque tipologia di socket e sono le sole utilizzabili da UDP: hanno tre argomenti aggiuntivi attraverso i quali è possibile specificare la destinazione dei dati trasmessi oppure ottenere l'origine dei dati ricevuti.

#### Funzione sendto()

La prima di queste funzioni, sendto(), serve per trasmettere un messaggio a un altro socket, e ha il seguente prototipo:

```
int sendto(int socketfd, const char *buffer, int len, int flags, const struct: sockaddr* to, unsigned int tolen);
```

- `socketfd`: descrittore di un socket connesso
- `buffer`: buffer contenente i dati da trasmettere
- `len`: lunghezza in byte dei dati in buffer
- `flags`: 0 = comportamento di default
- `to`: puntatore a una struttura che contiene l'indirizzo del socket target
- `tolen`: lunghezza dei dati in `to`, in byte (valore!)

# ESERCIZI IN LABORATORIO 11

Ha come parametro di ritorno il numero di byte inviati che deve sempre corrispondere alla dimensione totale specificata dal parametro len in quanto i dati non possono essere spezzati in invii successivi ma vengono sempre inviati in forma di pacchetto.
I primi tre argomenti sono identici a quelli della funzione write() e definiscono il socket socketfd a cui si fa riferimento, il buffer *buffer che contiene i dati da inviare e la relativa lunghezza len.
L'argomento flags è un intero usato come maschera binaria che permette di impostare una serie di modalità di funzionamento avanzate della comunicazione attraverso il socket: nei casi standard si lascia sempre uguale a 0.
I due argomenti to e tolen servono a specificare la destinazione del messaggio da inviare, e indicano rispettivamente la struttura contenente l'indirizzo di quest'ultima e la sua dimensione.

Questi argomenti vanno specificati nella stessa forma in cui si sarebbero usati con connect(): to punta alla struttura contenente l'indirizzo IP e la porta di destinazione dei dati da inviare.

La sendto() termina dopo aver completato la spedizione e non garantisce che i dati arrivino a destinazione: restituisce il numero di caratteri inviati in caso di successo e –1 per un errore, nel qual caso errno viene impostata al rispettivo codice di errore:

- EAGAIN: il socket è in modalità non bloccante, ma l'operazione richiede che la si blocchi;
- ECONNRESET: l'altro capo della comunicazione ha resettato la connessione;
- EDESTADDRREQ: il socket non è di tipo connesso e non si è specificato un indirizzo di destinazione;
- EISCONN: il socket è già connesso, ma si è specificato un destinatario;
- EMSGSIZE: il tipo di socket richiede l'invio dei dati in un blocco unico, ma la dimensione del messaggio lo rende impossibile;
- ENOBUFS: la coda di uscita dell'interfaccia è già piena (di norma Linux non usa questo messaggio ma scarta silenziosamente i pacchetti);
- ENOTCONN: il socket non è connesso e non si è specificata una destinazione;
- EOPNOTSUPP: il valore di flags non è appropriato per il tipo di socket usato;
- EPIPE: il capo locale della connessione è stato chiuso, si riceverà anche un segnale di SIGPIPE, a meno di non aver impostato MSG_NOSIGNAL nel parametro flags.

## Funzione recvfrom()

La funzione utilizzata per ricevere i dati nella comunicazione fra socket UDP è recvfrom(); il suo prototipo è:

```
int recvfrom(int socketfd, char *buffer, int len, int flags, struct sockaddr* from, unsigned int* fromlen);
```

- socketfd: descrittore di un socket connesso
- buffer: buffer contenente i dati in ingresso
- len: lunghezza in byte dei dati in buffer
- flags: 0 = comportamento di default
- from: puntatore a una struttura che contiene l'indirizzo del socket target
- fromlen: lunghezza dei dati in from, in byte (puntatore!)

I primi tre argomenti sono identici a quelli della funzione di read(): dal socket vengono letti len byte che vengono salvati nel buffer.
I due argomenti from e fromlen permettono di ottenere l'indirizzo del mittente del pacchetto che è stato ricevuto, e devono essere inizializzati con i puntatori alle variabili della struttura dove saranno scritti l'indirizzo IP e la sua lunghezza.

# 11 ESERCIZI IN LABORATORIO

La funzione restituisce il numero di byte ricevuti in caso di successo e –1 in caso di errore; nel qual caso errno assumerà il valore:

- EAGAIN, il socket è in modalità non bloccante, ma l'operazione richiede che la funzione si blocchi, oppure si è impostato un timeout in ricezione e questo è scaduto;
- ECONNREFUSED, l'altro capo della comunicazione ha rifiutato la connessione (in genere perché il relativo servizio non è disponibile);
- ENOTCONN, il socket è di tipo connesso, ma non si è eseguita la connessione;
- …oltre ai soliti EFAULT, EBADF, EINVAL, EINTR, ENOMEM, ENOTSOCK e gli eventuali altri errori relativi ai protocolli utilizzati.

A differenza delle read() e write() che abbiamo usato con i socket TCP in questo caso è possibile inviare con sendto() un pacchetto vuoto specificando un valore nullo per len: la ricezione con recvfrom() di un valore di ritorno di 0 byte viene interpretato come una chiusura della connessione o come una cessazione delle comunicazioni.

## Destinatario UDP

Realizziamo un host che si pone in attesa di un messaggio sulla porta locale 1313: la prima parte del codice è simile a quella descritta negli Esercizi precedenti, dove vengono definite le variabili e viene creato il socket: una differenza è il parametro utilizzato, cioè SOCK_DGRAM mediante il quale si definisce la modalità UDP.

A differenza di Java, che ha una struttura completamente diversa tra TCP e UDP, in C per passare da un protocollo all'altro basta solo sostituire un parametro nella definizione del socket:
- TCP viene settato a SOCK_STREAM;
- UDP viene settato a SOCK_DGRAM.

Dopo aver incluso le librerie, iniziamo a descrivere il main():

```
1  #include "libreria.h"                    // include elenco librerie
2
3  int main(int argc, char *argv[]){
4    struct sockaddr_in locale, remoto;
5    int socketfd, msglen, len, ris;
6    char msg[SIZEBUF];
```

Nella definizione dei parametri dell'host destinatario inizializziamo il campo dell'IP con valore INADDR_ANY che indica la disposizione a ricevere/effettuare connessioni qualunque sia l'indirizzo IP della macchina. Come numero di porta si deve invece specificare il numero di porta sulla quale si desidera porsi in ascolto. La configurazione è la seguente:

```
8    printf ("SERVER che riceve datagram UDP unicast\n");
9    // configurazione parametri porta locale
10   locale.sin_family    = AF_INET;                  // internet
11   locale.sin_addr.s_addr= htonl(INADDR_ANY);       // qualunque IP
12   locale.sin_port    = htons(SERVER_PORT);
```

Come al solito creiamo il socket indicando come parametro SOCK_DGRAM:

# ESERCIZI IN LABORATORIO 11

```
14    // creazione del socket
15    socketfd = socket(AF_INET, SOCK_DGRAM, 0);
16    if (socketfd == SOCKET_ERROR) {
17      printf ("socket() fallito err: %d \"%s\"\n", errno, strerror(errno));
18      exit(1);
19    }
20    printf ("socket() ok\n" );
21    fflush(stdout);
```

Effettuiamo ora l'operazione di bind():

```
23    ris = bind(socketfd, (struct sockaddr*) &locale, sizeof(locale));
24    if (ris == SOCKET_ERROR)  {
25      printf ("bind() fallita err: %d \"%s\"\n",errno, strerror(errno));
26      exit(1);
27    }
28    printf ("bind() sul server ok, ora mi pongo in attesa ...\n" );
29    fflush(stdout);
```

Quindi il server si pone in attesa dell'arrivo di messaggi da parte dei client:

```
31    // attesa della ricezione di un datagramma
32    len = sizeof(struct sockaddr_in);
33    msglen = recvfrom(socketfd, msg, (int)SIZEBUF, 0, (struct sockaddr*)&remoto, &len);
34    if (msglen<0) {
35      char msgerror[1024];
36      sprintf(msgerror,"recvfrom() failed [err %d] ", errno);
37      perror(msgerror);
38    }
```

Man mano che arriva un messaggio questo viene visualizzato con i parametri dell'host mittente:

```
40    // lettura datagramm e parametri del mittente
41    printf ("ho ricevuto un messaggio " );
42    printf (" msg: \"%s\" lunghezza: %d \n", msg, msglen);
43
44    // variante I: solo visualizzazione utilizzando inet_ntoa()
45    printf("host mittente: %s:%d\n", inet_ntoa(remoto.sin_addr), ntohs(remoto.sin_port)
```

Una possibile esecuzione è la seguente:

```
utente@PC-Win7 ~/ua2vol3c
$ ./server
SERVER che riceve datagram UDP unicast
socket() ok
bind() sul server ok, ora mi pongo in attesa ...
ho ricevuto un messaggio  msg: "ciao" lunghezza: 4
host mittente: 127.0.0.1:59149
```

## Mittente UDP

Il codice del mittente è molto simile a quello del client TCP descritto negli Esercizi precedenti; dopo l'inclusione delle librerie definiamo le variabili del main():

```
1   #include "librerie.h"                    // include elenco librerie
2
3   int main(int argc, char *argv[]){
4     struct sockaddr_in locale, remoto;
5     int socketfd, optVal, len;
6     int ris;
7     // passare la stringa come arg[1]
```

# 11 ESERCIZI IN LABORATORIO

Per poter inviare il datagram il client oltre a definire il proprio socket deve specificare l'indirizzo del server; viene così inizializzato anche il record che contiene i dati del server.

```
9   // configurazione parametri porta locale
10  locale.sin_family     = AF_INET;
11  locale.sin_addr.s_addr = htonl(INADDR_ANY);
12  locale.sin_port     = htons(0);
13
14  // assegnazione parametri del destinatario
15  remoto.sin_family       = AF_INET;
16  remoto.sin_addr.s_addr = inet_addr(IPADDRESS);   // indirizzoServer
17  remoto.sin_port     = htons(SERVER_PORT);        // porta remota
```

Si procede con la creazione del socket con l'accortezza di indicare come parametro SOCK_DGRAM:

```
24  // creazione del socket
25  socketfd = socket(AF_INET, SOCK_DGRAM, 0);
26  if (socketfd == SOCKET_ERROR){
27    printf ("socket() fallita, err: %d \"%s\"\n", errno, strerror(errno));
28    exit(1);
29  }
30  printf ("socket() ok \n");
31  fflush(stdout);
```

Il socket viene inizializzato e viene effettuato in bind(), segnalando eventuali errori:

```
33  // bind del socket
34  ris = bind(socketfd, (struct sockaddr*) &locale, sizeof(locale));
35  if (ris == SOCKET_ERROR){
36    printf ("bind() fallita, err: %d \"%s\"\n", errno, strerror(errno));
37    exit(1);
38  }
39  printf ("bind() ok \n");
40  fflush(stdout);
```

Ora è possibile inviare un messaggio; nel nostro esempio inviamo l'argomento letto come parametro nella linea di comando all'avvio del client:

```
42  // invia una parola a destinazione
43  len = sizeof(struct sockaddr_in);
44  ris = sendto(socketfd,argv[1],strlen(argv[1]),0,(struct sockaddr*)&remoto, len);
45  if (ris < 0)  {
46    printf ("errore sendto(): %d \"%s\"\n", errno,strerror(errno));
47    exit(1);
48  }
49  else
50    printf("datagram UDP \"%s\" inviato a %s:%d\n",argv[1],IPADDRESS,SERVER_PORT);
```

Una possibile esecuzione è la seguente:

```
$ ./client ciao
avvio esecuzione client che invia datagram UDP unicast
socket() ok
bind() ok
datagram UDP "ciao" inviato a 127.0.0.1:1313
```

**AREA DIGITALE**

**Funzioni GETSOCKOPT() e SETSOCKOPT()**

# ESERCIZI IN LABORATORIO 11

## METTITI ALLA PROVA

**1** Realizza un'applicazione echo UDP dove il server replica tutti i messaggi inviatigli dal client: il client legge linee di testo dallo standard input e le invia al server:
- il server legge linee di testo dal socket e le rimanda al client;
- il client legge righe di testo dal socket e le invia allo standard output.

Confronta la tua soluzione con quella presente nel file inviaUDPSol.c.

**2** Modifica il client facendo in modo che riceva come parametro sia l'indirizzo del server che quello della porta alla quale connettersi e dopo aver inviato un messaggio si metta in attesa di una risposta.
Modifica il codice del server creando un'applicazione che è in ascolto su una porta UDP, accetta tutti i datagrammi ricevuti ma deve scartare quelli che provengono da indirizzi IP non noti: per esempio, vengono accettati i datagrammi che sono trasmessi degli host con un numero dispari nell'ultimo ottetto dell'indirizzo IP e a questi si risponde con "OK" mentre agli altri si invia un "KO".
Dopo aver accettato tre messaggi, il server termina la sua esecuzione.

**3** Realizza un'applicazione multicast di datagrammi UDP dove il server invia a un indirizzo multicast e a intervalli di 1 secondo pacchetti UDP contenenti ciascuno una riga di un file di testo contenente *La Divina Commedia*; l'indirizzo e la porta di comunicazione vengono letti dalla linea di comando:

UDPserver IPaddress UDPport (per esempio UDPserver 127.0.0.1 6916)

Il client riceve i datagrammi e scrive un file per ogni canto, con la seguente struttura:

cantica_nrcanto.txt (per esempio inferno_canto5.txt).

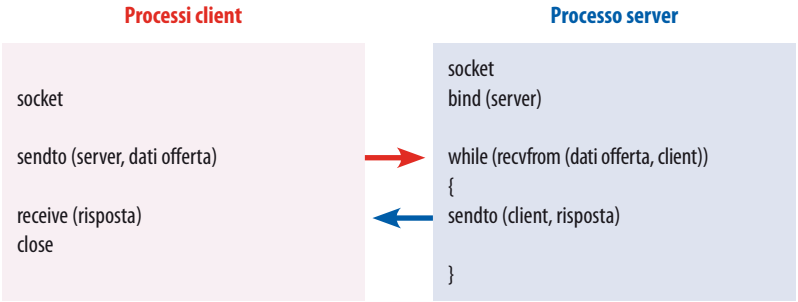
# 12 ESERCIZI IN LABORATORIO

## UN ESEMPIO COMPLETO CON I SOCKET: ASTA ONLINE

Scriviamo un'applicazione client-server UDP per gestire le offerte pervenute a un'asta: i client inviano al server l'importo dell'offerta e l'identificativo dell'offerente e il server mantiene in memoria solamente l'offerente e l'offerta più alta, rispondendo ai diversi client se l'offerta è stata accettata oppure l'importo da superare per potersi aggiudicare l'asta. La struttura del sistema, composta da un processo server e da *n* processi client, è la seguente:

**A** il processo server, dopo aver inizializzato il sistema, si pone in attesa delle offerte costituite da due dati (nome offerente e importo offerta) e a ogni nuova ricezione analizza i dati, controllando se l'importo è superiore alla migliore offerta sino ad allora pervenuta, comunicando l'esito al nuovo offerente; l'asta termina per esempio dopo che sono pervenute 10 offerte consecutive;

**B** i singoli client effettuano una offerta e si pongono in attesa dell'esito: quindi terminano la loro esecuzione.




### Codifica del server

Nella prima sezione del codice, oltre alle librerie, vengono definite alcune costanti che non necessitano di spiegazioni:

```
1  #include "libreria.h"
2
3  #define MAXOFFERTE   9          // numero massimo di offerte per sessione
4  #define MAXLUNGO     40         // lunghezza nominativo offerente
5  #define FILENAME "asta.txt"     // file con i dati del miglior offerente
```

Per la realizzazione del server definiremo tre funzioni:

```
7   struct sockaddr_in offerente;
8   int socketlen;
9
10  // creazione del socket sul server
11  int creaServer (char *ip_address, int porta){
12    ...
13    return socketfd;
14  }
15
16  // funzione che trasmette un messaggio al client
17  int inviaMsg (int socketfd, char *buffer){
18    ...
19    return esito;          // 1 = ok,  0 = errore
20  }
21
22  // funzione che riceve un messaggio dal client
23  int riceviMsg (int socketfd, char *buffer){
24    ...
25    return numCaratteri;
26  }
```

# ESERCIZI IN LABORATORIO 12

La prima funzione crea il socket, la seconda invia un messaggio che riceve come parametro nel buffer e la terza rimane in attesa di una comunicazione da parte di un client.
La codifica viene lasciata come esercizio al lettore: una soluzione è presente nel file astaServer.c che è contenuto in UDA2vol3C.rar, scaricabile dal sito www.hoepliscuola.it nella cartella MATERIALI.
Il programma principale può essere organizzato in tre sezioni:

```
28  // programma principale
29  int main (int argc, char *argv[]){
30    // definizione delle variabili locali
31    ...
32    // 1 - creazione del socket sul server
33    ...
34    // 2 - recupero dati da file
35    ...
36    // 3 - ricezione di una offerta
37    ...
38    // chiusura del socket
39    ...
40  }
```

La prima parte di codice è la seguente:

```
60  // programma principale
61  int main (int argc, char *argv[]){
62    int  socketfd, numOfferte;              // definizione delle variabili locali
63    char buffer [BUFSIZ + 1];
64    char nomeOfferente [MAXLUNGO], nomeVince [MAXLUNGO];
65    int  nuovaOfferta, offertaVince;
66    FILE *fileOfferte;
67    numOfferte = 0;                         // contatore offerte pervenute
```

Sul file fileOfferte vengono memorizzati solo i dati del miglior offerente, e in particolare solo il nome e l'importo dell'offerta, che vengono successivamente letti e inseriti rispettivamente nelle variabili nomeVince e offertaVince.
I dati della nuova offerta pervenuta vengono temporaneamente memorizzati in nomeOfferente e nuovaOfferta e quindi vengono elaborati.

La creazione del socket sul server avviene semplicemente con la chiamata alla funzione prima definita:

```
69    // creazione del socket sul server
70    if ((socketfd = creaServer (IPADDRESS, SERVER_PORT)) == 0){
71      fprintf (stderr, "Impossibile creare il socket sul server\n");
72      exit (EXIT_FAILURE);
73    }
```

Viene ora letto dal fileOfferte il valore dell'offerta vincente, se presente, sennò le rispettive variabili vengono azzerate:

```
75    // recupero dati da file
76    if ((fileOfferte = fopen (FILENAME, "r" )) == NULL){
77      nomeVince [0] = '\0';
78      offertaVince = 0;
79      printf ("L'asta e' aperta: nessuna offerta pervenuta\n");
80    }
81    else{
82      fscanf (fileOfferte, "%s %d", nomeVince, &offertaVince);
83      fclose (fileOfferte);
84      printf ("L'asta e' aperta: fate le vostre offerte\n");
85      printf ("Migliore offerta pervenuta: %s %d\n", nomeVince, offertaVince);
86    }
```

# 12 ESERCIZI IN LABORATORIO

Mediante riceviMsg() il server si mette ora in attesa di ricevere una nuova offerta da parte di un client; quando gli perviene un messaggio lo analizza:

- se l'importo è minore di 0, l'offerta è errata, e segnala questa situazione al mittente;
- se l'importo è corretto ed:
  - è superiore all'offerta che sta "vincendo" **deve essere aggiornato** il file con il nuovo importo e il nuovo offerente;
  - è inferiore all'offerta che sta "vincendo" viene solamente inviato un messaggio al client.

```
88    // ricezione di una offerta
89    while (riceviMsg (socketfd, buffer) > 0){
90      nuovaOfferta = 0;
91      sscanf (buffer, "%s %d", nomeOfferente, &nuovaOfferta);
92      if (nuovaOfferta <= 0)
93        inviaMsg (socketfd, "Offerta non valida\n");
94      else{
95        numOfferte++;
96        printf ("Nuova offerta # %d di Euro  %d\n", numOfferte, nuovaOfferta);
97        if (nuovaOfferta > offertaVince) {   // la nuova offerta diviene vincente
98          offertaVince = nuovaOfferta;
99          strcpy (nomeVince, nomeOfferente);
100         if ((fileOfferte = fopen (FILENAME, "w" )) == NULL) { // aggiorno il file
101           inviaMsg (socketfd, "Errore di apertura del file\n");
102           printf ("errore di apertura del file\n");
103         }
104         else{
105           fprintf (fileOfferte, "%s %d", nomeVince, offertaVince);
106           fclose (fileOfferte);
107         }
108         inviaMsg (socketfd, "Offerta accettata\n");
109         printf ("Nuova offerta da superare: %s %d\n", nomeVince, offertaVince);
110       }
111       else{                                          // l'offerta è inferiore
112         inviaMsg (socketfd, "Spiacente, e' stata fatta un'offerta superiore\n");
113       }
114     }
```

Il server continua a ricevere offerte fino a quando si raggiunge il valore MAXOFFERTE: a quel punto l'asta termina.

```
116     if (numOfferte>MAXOFFERTE){                    // l'asta e' terminata
117       printf ("Termine asta : si aggiudica il sig. %s con l'offerta di Euro %d \n",
118                                                     nomeVince, offertaVince);
119       exit (0);
120     }
121   }
```

Quindi viene chiusa la connessione e termina l'esecuzione del processo server.

```
123   // chiusura del socket
124   if (close (socketfd) != 0)  {
125     printf ("errore nella chiusura del socket");
126     return 0;
127   }
128   return EXIT_SUCCESS;
129 }
```

# ESERCIZI IN LABORATORIO 12

Una possibile esecuzione è la seguente:

```
Utente@PC-Win7 ~/ua2vol3c
$ ./server
L'asta e' aperta: fate le vostre offerte
Nuova offerta # 1 di Euro  200ro 100
Nuova offerta da superare: paolo 200
Nuova offerta # 2 di Euro  270
Nuova offerta da superare: mario 270
Nuova offerta # 3 di Euro  300
Nuova offerta da superare: piero 300
Nuova offerta # 4 di Euro  330
Nuova offerta da superare: ciro 330
Nuova offerta # 5 di Euro  380
Nuova offerta da superare: mario 380
Nuova offerta # 6 di Euro  300
Nuova offerta # 7 di Euro  400
Nuova offerta da superare: piero 400
Nuova offerta # 8 di Euro  420
Nuova offerta da superare: anna 420
Nuova offerta # 9 di Euro  440
Nuova offerta da superare: rita 440
Nuova offerta # 10 di Euro  500
Nuova offerta da superare: oscar 500
Termine asta : si aggiudica il sig. oscar con l'offerta di Euro 500
```

## Codifica del client

Come per il server definiamo due funzioni, la prima per ricevere dati dal server, la seconda per trasmettere la nostra offerta.
La codifica viene lasciata come esercizio al lettore: una soluzione è presente nel file astaClient.c contenuto in UDA2vol3C.rar, scaricabile dal sito www.hoepliscuola.it nella cartella MATERIALI.

```
3   struct sockaddr_in client;
4   int socketlen;
5   // funzione che riceve un messaggio dal server
6   int riceviMsg (int socketfd, char *buffer){
7     ...
8     return numCaratteri;
9   }
10  // funzione che trasmette un messaggio al server
11  int inviaMsg (int socketfd, char *buffer, char *ip_address, int porta){
12    ...
13    return esito;          // 1 - ok,  0 - errore
14  }
```

Il programma principale è abbastanza semplice; riceve in ingresso come argomento l'indirizzo IP del server al quale inviare l'offerta:

```
18  int main (int argc, char *argv[]){
19    // variabili locali
20    char buffer[BUFSIZ + 1];
21    int socketfd;
22    //  per leggere IP del server da parametro
23    if (argc != 2){
24      fprintf (stderr, "richiesto un argomento: indirizzo IP del server\n");
25      exit (EXIT_FAILURE);
26    }
```

# 12 ESERCIZI IN LABORATORIO

e quindi definisce il socket del client:

```
28    //  creazione del socket del client
29    if ((socketfd = socket (AF_INET, SOCK_DGRAM, 0)) < 0){
30      printf ("errore fatale nella creazione del socket()");
31      return 0;
32    }
```

L'utente può ora inserire il nome e l'offerta che viene inoltrata al server che provvederà a rispondere con un messaggio che contiene l'esito dell'operazione, cioè se si è superata o meno l'offerta degli altri competitori:

```
34    // creazione della offerta
35    printf ("Inserire nome e offerta: ");
36    if (fgets (buffer, BUFSIZ, stdin) != NULL){
37      inviaMsg (socketfd, buffer, argv[1], PORTA);      // invio messaggio
38      if (riceviMsg (socketfd, buffer) > 0)             // attesa risposta
39        fputs (buffer, stdout);
40    }
```

Il codice del client termina con la chiusura del socket:

```
42    // chiusura socket del client
43    if (close (socketfd) != 0){
44      printf ("errore fatale nella chiusura del socket()");
45      return 0;
46    }
47
48    return EXIT_SUCCESS;
49  }
```

Un'esecuzione del client è riportata nella figura seguente:

```
Utente@PC-Win7 ~/ua2vol3c
$ ./astaClient
richiesto un argomento: indirizzo IP del server

Utente@PC-Win7 ~/ua2vol3c
$ ./astaClient 127.0.0.1
Inserire nome e offerta: ciro $40
Offerta accettata

Utente@PC-Win7 ~/ua2vol3c
$ ./astaClient 127.0.0.1
Inserire nome e offerta: mario $50
Offerta accettata
```

```
Utente@PC-Win7 ~/ua2vol3c
$ ./astaClient 127.0.0.1
Inserire nome e offerta: anna $00
Spiacente, e' stata fatta un'offerta superiore

Utente@PC-Win7 ~/ua2vol3c
$ ./astaClient 127.0.0.1
Inserire nome e offerta: paolo $90
Offerta accettata

Utente@PC-Win7 ~/ua2vol3c
$ ./astaClient 127.0.0.1
Inserire nome e offerta: rosario
Offerta non valida
```

## METTITI ALLA PROVA

1. Modifica il sistema facendo in modo che, nel caso in cui una nuova offerta superi il valore della migliore offerta presente, il nuovo valore massimo venga comunicato a tutti i partecipanti attivi, cioè coloro che abbiano effettuato almeno un'offerta.

2. Modifica il sistema precedente offrendo la possibilità di effettuare un'offerta per un oggetto scelto da un elenco dove sono presenti diversi articoli, ciascuno identificato tramite un proprio codice identificatore di lunghezza 8 caratteri (per esempio ABC12345): il sistema deve controllare che il codice corrisponda effettivamente a un articolo posto all'asta, verificare il valore dell'offerta e comunicare l'esito sia al nuovo offerente che al precedente detentore dell'offerta massima nel caso in cui questa venga superata.

# ESERCIZI IN LABORATORIO 13

## APPLICAZIONI MULTICAST IN C

### Premessa: una funzione per settare il socket attivo

Tra le primitive per la gestione dei socket abbiamo le funzioni getsockopt() e setsockopt() che permettono di rappresentare alcune caratteristiche proprie dei socket per configurare la trasmissione/ricezione di datagram per un certo indirizzo UDP di multicast, quali le dimensioni dei segmenti, i controlli sulla funzionalità della connessione, la modalità di utilizzo degli indirizzi.

Queste opzioni possono essere settate solo quando il socket è reso disponibile all'interfaccia di programmazione, quindi dopo che si è effettuato il binding.

Il prototipo delle due funzioni è il seguente:

```
int setsockopt(int sockfd, int level, int optname, void *optval, socklen_t optlen);
int getsockopt(int sockfd, int level, const void *optval, socklen_t optlen);
```

dove:
- il primo argomento sockfd è il descrittore di socket aperto con un socket();
- l'argomento level indica a che livello di protocollo deve agire l'opzione:
  – a livello di socket generale (level=SOL_SOCKET);
  – a livello IPv4 (IPPROTO_IP), IPv6 (IPPROTO_IPV6), ICMPv6 (IPPROTO_ICMPV6);
  – a livello TCP (level=IPPROTO_TCP);
- optname specifica l'opzione da settare per lo specifico livello;
- optval è un puntatore a una variabile di tipo dipendente dall'opzione specificata, che contiene il valore nuovo da settare dell'opzione da configurare;
- optlen contiene la dimensione della variabile puntata da optval.

### Opzioni di livello IPv4 (IPPROTO_IP)

Le opzioni utilizzate per configurare un socket UDP per la trasmissione/ricezione di datagram per di livello IPv4 (IPPROTO_IP) sono riportate nella tabella che segue.

| OPZIONE (optname) | TIPO DI DATO | USO |
|---|---|---|
| IP_ADDR_MEMBERSHIP | struct ip_mreq | Collega il socket UDP a uno specifico gruppo di multicast |
| IP_DROP_MEMBERSHIP | struct ip_mreq | Toglie il socket UDP dal gruppo di multicast |
| IP_MULTICAST_IF | struct in_addr | Indica l'interfaccia di rete da usare per spedire i datagram di multicast |
| IP_MULTICAST_TTL | u_char | Specifica il TTL dei datagram di multicast da spedire |
| IP_MULTICAST_LOOP | u_char | Indica se i datagram di multicast spediti devono andare anche al mittente, se questo appartiene al gruppo |

# 13 ESERCIZI IN LABORATORIO

Noi utilizzeremo il primo parametro che, come tipo di dato, richiede un record ip_mreq avente la seguente struttura costituita da due elementi in_addr:

```
struct ip_mreq {
  struct in_addr multiaddres;          // indirizzo IP del gruppo multicast
  struct in_addr interfaccia;          // indirizzo locale dell'interfaccia/socket
};
```

ricordando che la struttura del record in_addr è la seguente:

```
struct in_addr {
  unsigned long s_address;             // dato caricato con la funzione inet_aton()
};
```

Come vedremo, utilizzeremo IPPROTO_IP e IP_ADDR_MEMBERSHIP nella funzione setsockopt():

```
setsockopt(<socket>, IPPROTO_IP, IP_ADD_MEMBERSHIP, <optionVal>,<optionSize>)
```

## La comunicazione multicast attivo

Il multicast consente di inviare uno stesso datagram UDP a più destinatari, i quali devono essersi preventivamente registrati come membri del gruppo di multicast caratterizzato da un certo indirizzo di multicast (classe D, 224.0.0.0 - 239.255.255.255).
Per ricevere datagram UDP via multicast, dopo la creazione di un socket UDP, si esegue il collegamento a una porta del protocollo UDP con bind() e si effettua il join a un indirizzo di multicast con setsockopt().

Se con bind() non specifico un indirizzo IP, posso ricevere datagram UDP sia originati da multicast sia da unicast, per la porta UDP specificata.
Se con bind() specifico l'indirizzo IP di multicast a cui farò il join, potrò ricevere solo datagram UDP di multicast per quell'indirizzo di multicast.

### Il codice del server

Il codice del server non presenta particolarità: dopo aver importato le librerie, definiamo l'indirizzo del gruppo IP:

```
multicastServer.c
1  #include "librerie.h"
2  #define IP_GRUPPO "239.0.0.1"
```

Quindi nel main(), dopo aver definito le variabili necessarie, inizializziamo il socket:

```
4  int main (int argc){
5    struct sockaddr_in remoto;
6    int lungo, mioSocket, cnt;
7    char msg[SIZEBUF] = {0};
8    /* impostiamo il socket del server */
9    mioSocket = socket (AF_INET, SOCK_DGRAM, 0);
10   if (mioSocket <0) {
11     printf ("socket() fallito ");
12     return(1);
13   }
```

# ESERCIZI IN LABORATORIO 13

Come al solito inizializziamo i valori degli indirizzi nella nostra istanza del sockaddr_in:

```
15    /* configuriamo i parametri della porta locale del server */
16    bzero ((char *) &remoto, sizeof (remoto));   // azzero il record
17    remoto.sin_family = AF_INET;
18    remoto.sin_addr.s_addr = htonl (INADDR_ANY);
19    remoto.sin_port = htons (SERVER_PORT);
20    remoto.sin_addr.s_addr = inet_addr (IP_GRUPPO);
21    lungo = sizeof (remoto);
```

e in un ciclo infinto, a intervalli di 5 secondi, inviamo il messaggio contenente data/ora ai client:

```
26    // inviamo in multicast in data/ora di sistema ogni cinque secondi
27    while (1) {
28      time_t dataOra = time (0);
29      sprintf (msg, "data/ora esatta: %-24.24s", ctime (&dataOra));
30      printf ("invio: %s \n", msg);
31      cnt = sendto (mioSocket, msg, sizeof (msg), 0,(struct sockaddr *) &remoto, lungo);
32      if (cnt <0) {
33        perror ("sendto");
34        return (1);
35      }
36      sleep (5);
37    }
```

Avviamo l'esecuzione e otteniamo qualcosa simile alla seguente schermata:

```
utente@PC-Win7 ~/ua2vol3c
$ ./server
invio: data/ora esatta: Fri Oct 11 18:19:58 2019
invio: data/ora esatta: Fri Oct 11 18:20:03 2019
invio: data/ora esatta: Fri Oct 11 18:20:08 2019
invio: data/ora esatta: Fri Oct 11 18:20:13 2019
invio: data/ora esatta: Fri Oct 11 18:20:18 2019
invio: data/ora esatta: Fri Oct 11 18:20:23 2019
invio: data/ora esatta: Fri Oct 11 18:20:28 2019
invio: data/ora esatta: Fri Oct 11 18:20:33 2019
invio: data/ora esatta: Fri Oct 11 18:20:38 2019
invio: data/ora esatta: Fri Oct 11 18:20:43 2019
invio: data/ora esatta: Fri Oct 11 18:20:48 2019
invio: data/ora esatta: Fri Oct 11 18:20:53 2019
invio: data/ora esatta: Fri Oct 11 18:20:58 2019
invio: data/ora esatta: Fri Oct 11 18:21:03 2019
```

Il codice completo del server è riportato nel file multicastServer.c.

## Il codice del client

Analogamente al server le prime istruzioni del client importano le librerie e definiscono l'indirizzo del gruppo: quindi il main() definisce le variabili e imposta il socket:

```
5   int main (int argc){
6     struct sockaddr_in remoto;
7     int lungo, mioSocket, cnt;
8     struct ip_mreq mioMreq;
9     char msg[SIZEBUF]= {0};;
10    // impostiamo il socket del client
11    mioSocket = socket (AF_INET, SOCK_DGRAM, 0);
12    if (mioSocket < 0) {
13      printf ("socket() fallito ");
14      return (1);
15    }
```

# 13 ESERCIZI IN LABORATORIO

Anche nel client inizializziamo i valori degli indirizzi del server nella nostra istanza del sockaddr_in:

```
18  // configuriamo parametri del server
19  bzero ((char *) & remoto, sizeof (remoto));   // azzero il record
20  remoto.sin_family = AF_INET;
21  remoto.sin_addr.s_addr = htonl (INADDR_ANY);
22  remoto.sin_port = htons (SERVER_PORT);
23  lungo = sizeof (remoto);
```

Per ricevere un datagram multicast, un processo deve fare il join al gruppo di multicast: configuriamo i valori del gruppo inizializzando un'istanza del record ip_mreq:

```
25  // configuriamo record ip_mreq del gruppo multicast
26  mioMreq.imr_multiaddr.s_addr = inet_addr (IP_GRUPPO);
27  mioMreq.imr_interface.s_addr = htonl (INADDR_ANY);
```

Successivamente il client deve quindi fare il bind() del socket UDP al numero di porta che viene utilizzato come destinazione dei datagram inviati al gruppo a cui appartiene:

```
29  // leghiamo l'indirizzo al transport endpoint
30  printf ("bind()\n");
31  if (bind (mioSocket, (struct sockaddr *) &remoto, sizeof(remoto)) <0) {
32    printf ("fallito il bind()");
33    return (1);
34  }
```

Impostiamo con la funzione setsockopt() i dati precedentemente settati nel record ip_mreq:

```
36  // impostiamo sul socket i dati del gruppo definiti in ip_mreq
37  if (setsockopt(mioSocket, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mioMreq, sizeof(mioMreq))<0){
38    perror ("setsockopt mreq");
39    return (1);
40  }
```

Possiamo ora porre il client in attesa delle comunicazioni del server:

```
41  // riceviamo l'orario dal server multicast per cinque volte
42  for (int x = 0; x < 5; x++){
43    cnt = recvfrom (mioSocket, msg, sizeof(msg), 0, (struct sockaddr *) &remoto, &lungo);
44    if (cnt < 0) {
45      printf("errore in recvfrom()" );
46      return (1);
47    }
48    printf ("%s: message = \"%s \" \n", inet_ntoa (remoto.sin_addr), msg);
49  }
```

Avviamo l'esecuzione e otteniamo qualcosa simile alla seguente schermata:

```
utente@PC-win7 ~/ua2vol3c
$ ./client
bind()
192.168.1.2: message = "data/ora esatta: Fri Oct 11 18:20:13 2019 "
192.168.1.2: message = "data/ora esatta: Fri Oct 11 18:20:18 2019 "
192.168.1.2: message = "data/ora esatta: Fri Oct 11 18:20:23 2019 "
192.168.1.2: message = "data/ora esatta: Fri Oct 11 18:20:28 2019 "
192.168.1.2: message = "data/ora esatta: Fri Oct 11 18:20:33 2019 "
```

**AREA DIGITALE**

**Esercizi proposti**

Il codice completo del client è riportato nel file multicastClient.c.

# Applicazioni lato server in PHP e AJAX

UNITÀ 3

**LEZIONE 1**
Classi e oggetti in PHP

**LEZIONE 2**
Comunicazione client-server in PHP con AJAX

**LEZIONE 3**
La connessione al database MySQL

**Esercizi in Laboratorio**

L1 Installazione di EasyPHP e XAMPP

L2 Utilizzo di AJAX nella interrogazione di database MySQL

L3 Upload/download dei file e utilizzo di file JSON

L4 Client-server in PHP: invio di messaggi e file mediante socket

L5 Connessione FTP con uno script PHP

L6 Leggere e scrivere file Excel e Word

L7 Inviare una mail con PHP connesso a MySQL

L8 Creare un file pdf con PHP


## CONOSCENZE

- Conoscere i file e l'upload in PHP
- Conoscere la programmazione a oggetti di PHP
- Apprendere il ruolo del Web server
- Comprendere il ruolo di AJAX nel dialogo client-server

## COMPETENZE

- Realizzare applicazioni client-server in PHP con l'uso dei socket
- Realizzare la connessione a MySQL in PHP con la OO MySQLi
- Realizzare un servizio di mailing con PHP
- Utilizzare AJAX con PHP
- Realizzare applicazioni Web dinamiche che realizzino interazioni con le mappe di Google

## ABILITÀ

- Applicare le API di Google in pagine Web dinamiche
- Scrivere pagine Web con i socket
- Realizzare server FTP con PHP
- Realizzare pagine in formato PDF con PHP
- Realizzare file in formato Excel e Word da PHP

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

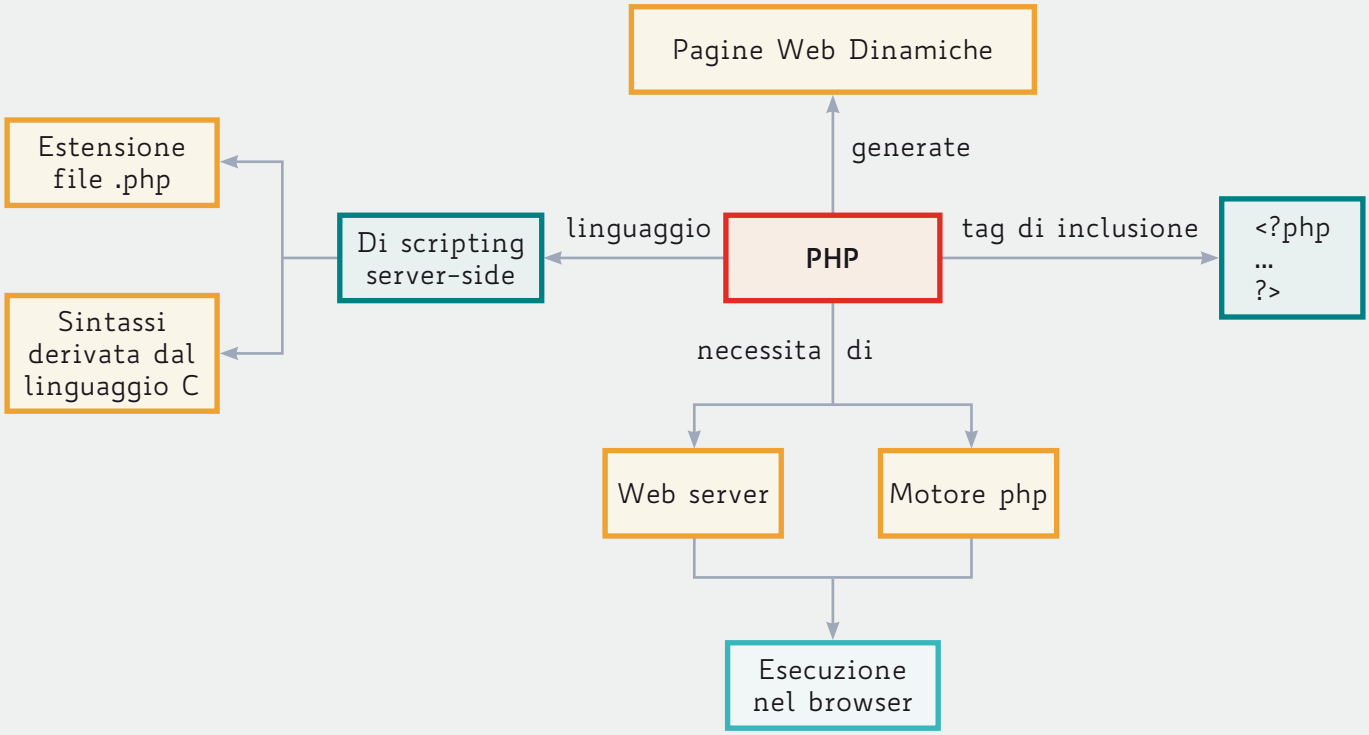
# 1 Classi e oggetti in PHP

**IN QUESTA LEZIONE IMPAREREMO...**

- a conoscere il paradigma a oggetti in PHP
- a creare classi, metodi e proprietà
- a istanziare oggetti

## MAPPA CONCETTUALE

didattica inclusiva




## Generalità su PHP

Le pagine che formano il Web possono essere statiche oppure dinamiche: sono statiche quando presentano gli stessi contenuti a tutti gli utenti che vi accedono, sono dinamiche quando possono mostrare contenuti diversi in relazione alle azioni effettuate dagli utenti stessi (per esempio, il login di accesso a una sezione riservata di un sito).

PHP è un linguaggio "general-purpose" che però, in pratica, viene utilizzato solo per creare siti Web dinamici; viene interpretato dal motore PHP (php engine), chiamato Zend, residente sul server Web: è quindi un linguaggio di scripting server-side.

L'interprete PHP è un software libero distribuito sotto la PHP License.

La sintassi di PHP è simile a quella di C, C++ e Java e al linguaggio a cui possono essere aggiunte estensioni che realizzano funzionalità specifiche come gli aspetti di sicurezza/crittografia, interazione con altri siti Web ecc.

La sintassi di un segmento di codice PHP prevede che l'inizio dello script venga marcato dal tag di apertura script (<?php): in questo modo il motore PHP riconosce il tag di apertura script ed esegue tutto il codice fino a quando non trova il tag di chiusura dello script, rappresentato da (?>).

Gli script PHP sono eseguiti sul server ma devono inviare le loro risposte al client, e quindi produrre dinamicamente file HTML che vengono trasferiti al browser del client in modo da poter essere correttamente visualizzati dall'utente.
Le pagine PHP possono anche contenere parti di codice HTML e questa caratteristica prende il nome di HTML embedded: tutto ciò che si trova all'esterno dei tag che delimitano il codice PHP viene lasciato inalterato, mentre tutto quello che si trova all'interno dei tag <?php viene interpretato da Zend e i risultati, espressi in formato HTML, vengono inviati in risposta al browser che aveva richiesto la pagina.
Dal 1995, anno in cui fece la sua comparsa PHP, a oggi, la comunità degli sviluppatori ha aggiornato periodicamente il linguaggio, sia per mantenere il passo con l'evoluzione tecnologia sia per adeguarsi a soddisfare tutte le richieste delle nuove esigenze della programmazione Web.

A volte, però, le nuove versioni (nel febbraio 2020 è stata rilasciata la 7.4.3) non hanno mantenuto la completa compatibilità con le vecchie creando notevoli disagi ai programmatori: ricordiamo, per esempio, il cambiamento epocale con l'introduzione della OOP effettuato nella versione 4 e il cambiamento del naming delle variabili superglobali nella versione 5.

La nostra trattazione inizia proprio dall'introduzione della OOP in PHP dalla sua prima versione fino a descrivere le novità introdotte dalla versione 7, come il "nuovo naming dei costruttori".

## Il paradigma a oggetti in PHP

PHP implementa il paradigma degli oggetti (Object Oriented Paradigm) a partire dalla versione 5.0.
Per la creazione di una nuova classe di oggetti si deve utilizzare il costrutto class, all'interno del quale possiamo collocare i metodi e le proprietà, rappresentate rispettivamente da funzioni e variabili: tuttavia le proprietà devono essere dichiarate anteponendo la chiave var.

### ESEMPIO

**UNA CLASSE CARRELLO DELLA SPESA**

Come primo esempio realizziamo una classe Carrello con due proprietà rappresentate da due array che contengono gli articoli presenti nel carrello e la relativa quantità:

```
carrello.php
1  <?php
2  class Carrello
3  {
4     // definizione dei membri articoli e quantita' come array
5     var $articoli = array();
6     var $quantita = array();
```

Nella classe sono definiti quattro metodi:

- aggiungi_carrello(): aggiunge al carrello un ordine di un nuovo articolo;
- togli_carrello(): elimina un elemento dal carrello;
- aggiorna(): effettua l'aggiornamento di un prodotto in termini di quantità;
- stampa(): mostra a video tutti gli elementi dell'oggetto carrello, cioè quanto inserito nella nostra borsa della spesa virtuale.

Il metodo aggiungi_carrello() riceve come parametro il nome dell'articolo da aggiungere e la quantità: dopo aver letto la dimensione dell'array attraverso la funzione predefinita count() (riga 11) viene effettuato un ciclo (riga 22) per verificare se l'elemento che si vuole aggiungere è già presente (riga 26) e in caso positivo viene chiamato il metodo aggiorna() (riga 30) che provvederà solamente ad aggiornare la quantità del prodotto aggiungendo la nuova quantità. Se l'elemento non è presente, questo viene aggiunto ai due array (righe 32 e 33).

```
19    // metodo che aggiunge $n articoli di tipo $a al carrello
20    function aggiungi_carrello($a, $q){
21      // ottengo la lunghezza dell'array
22      $position = count($this->articoli);
23      $trovato = 0;
24      // ciclo di ricerca elemento nell'array
25      for ($i = 0; $i < count($this->articoli); $i++){
26        if ($this->articoli[$i] == $a) // ctr se il prodotto e' presente nel carrello
27          $trovato = 1;
28      }
29      if ($trovato == 1)              // se il prodotto e' presente aggiorno quantita'
30        $this->aggiorna($a,$q);
31      else{                           // altrimenti aggiunge il prodotto al carrello
32        $this->articoli[$position] = $a;
33        $this->quantita[$position] = $q;
34      }
35    }
```

Come possiamo notare nella riga 30 davanti al nome del metodo aggiorna() è stato anteposto l'operatore di autoreferenza $this e la notazione freccia (->): l'operatore di autoreferenza serve per indicare che il metodo appartiene alla classe. La notazione rappresentata da una freccia formata da un trattino e dal simbolo maggiore (->) è chiamata brackets notation.

Il metodo togli_carrello() riceve come parametro il nome dell'articolo da eliminare (riga 38) che viene cercato nell'array articoli (riga 42): se l'elemento è presente viene salvata la posizione nella variabile posizione (riga 44) e successivamente (righe 49÷53) si copiano gli elementi degli array articoli e quantita negli array temporanei (app_ar e app_qt) tranne l'elemento da cancellare (riga 50): ultime operazioni sono quelle di ripristinare il contenuto degli array-attributi partendo dagli array temporanei appena aggiornati.

```
37    // metodo che elimina l'articolo di nome $a
38    function togli_carrello($a){
39      // variabile utilizzata per cercare posizione elemento; settata a non trovato (-1)
40      $posizione = -1;
41      // ciclo di ricerca elemento nell'array
42      for ($i = 0; $i < count($this->articoli); $i++){
43        if ($this->articoli[$i] == $a)  // ctr se il prodotto e' presente nel carrello
44          $posizione = $i;              // se trovato salvo la posizione
45      }
46      if ($posizione != -1){            // se trovato procedo a toglierlo
47        $cont = 0;
48        // copia negli array temporanei gli elementi eccetto quello da cancellare
49        for ($i = 0; $i < count($this->articoli); $i++){
50          if ($this->articoli[$i] != $a){
51            $app_ar[$cont] = $this->articoli[$i];
52            $app_qt[$cont] = $this->quantita[$i];
53            $cont++;
54          }
55        }
56        // elimino i "vecchi" array
57        unset($this->articoli);
58        unset($this->quantita);
59        // rimetto i valori dagli array temporanei agli array articoli e quantita'
60        for ($i = 0; $i < count($app_ar); $i++){
61          $this->articoli[$i] = $app_ar[$i];
62          $this->quantita[$i] = $app_qt[$i];
63        }
64      }
65      else echo "Prodotto non trovato!<BR>";
66    }
```

Il metodo aggiorna() riceve due parametri ($n e $q) che rappresentano il nome del prodotto da cercare e la quantità alla quale aggiornarlo. Il ciclo (riga 71) ricerca la presenza dell'elemento nell'array, quando l'elemento è stato trovato (righe 73 e 74) viene salvata la posizione.
Se l'elemento è presente si aggiorna la quantità oppure viene segnalato l'errore.

```
68    // metodo che aggiorna l'array
69    function aggiorna($n,$q){
70      $posizione = -1;
71      for ($i = 0; $i < count($this->articoli); $i++){
72        // prelevo la posizione del prodotto nell'array
73        if ($this->articoli[$i] == $n)
74          $posizione = $i;
75      }
76      // aggiorno le informazioni del prodotto
77      if ($posizione == -1)
78        echo "Prodotto non trovato!<BR>";
79      else
80        $this->quantita[$posizione] = $q;
81    }
```

Il codice che segue è relativo al metodo stampa() e non necessita di chiarimenti:

```
83    // metodo di stampa a video dei prodotti
84    function stampa(){
85      for ($i = 0; $i < count($this->articoli); $i++){
86        echo "<B>Articoli:</B> " .$this->articoli[$i] ."<BR>";
87        echo "<B>Quantita':</B> " .$this->quantita[$i] ."<BR>";
88      }
89    }
```

## Il programma di prova

Scriviamo ora un programma di prova carrelloTest.php dove utilizziamo la classe Carrello: è necessario includere il file carrello.php che contiene il codice relativo alla classe da istanziare (riga 3). L'istanza della classe Carrello avviene per mezzo dell'operatore new (riga 5) e l'invocazione dei metodi avviene nel codice sull'oggetto $acquisti con la brackets notation (->).
Eseguiamo quindi le tre operazioni dello script ottenendo il risultato mostrato, a destra, nella finestra riportata a lato.

```
carrelloTest.php
1   <?php
2     // require file che contiene la classe
3     require_once("carrello.php");
4     // istanza oggetto di classe Carrello
5     $acquisti = new Carrello;
6     // invocazione metodo aggiungi_carrello con diversi prodotti
7     echo "<HR>Aggiungo alcuni prodotti<BR>";
8     $acquisti->aggiungi_carrello("tempera",3);
9     $acquisti->aggiungi_carrello("pennello",15);
10    $acquisti->aggiungi_carrello("tempera",3);
11    $acquisti->aggiungi_carrello("fogliA4",999);
12    $acquisti->stampa();
13    echo "<HR>Cancello prodotto fogliA4<BR>";
14    $acquisti->togli_carrello("fogliA4");
15    $acquisti->stampa();
16    echo "<HR>Modifico prodotto pennello<BR>";
17    $acquisti->aggiorna("pennello",132);
18    $acquisti->stampa();
19  ?>
```

Aggiungo alcuni prodotti
**Articoli:** tempera
**Quantita':** 3
**Articoli:** pennello
**Quantita':** 15
**Articoli:** fogliA4
**Quantita':** 999

Cancello prodotto fogliA4
**Articoli:** tempera
**Quantita':** 3
**Articoli:** pennello
**Quantita':** 15

Modifico prodotto pennello
**Articoli:** tempera
**Quantita':** 3
**Articoli:** pennello
**Quantita':** 132

## METTITI ALLA PROVA

• Utilizzare le classi • Istanziare oggetti • Usare metodi e proprietà

**Apri il file carrelloTest.php**
Modifica il file in modo che mostri all'utente una pagina in cui può scegliere se aggiungere, cancellare o aggiornare i prodotti dall'elenco della spesa virtuale, quindi a ogni aggiunta o eliminazione visualizzare quanto presente nel carrello.

# I costruttori da PHP 4 a PHP 7

I costruttori sono metodi di una classe aventi il nome della stessa classe: un costruttore viene richiamato automaticamente quando viene creata una nuova istanza.
Il codice seguente riporta un costruttore per la classe Carrello e quello che effettua la creazione di un nuovo carrello e contemporaneamente aggiunge un prodotto:

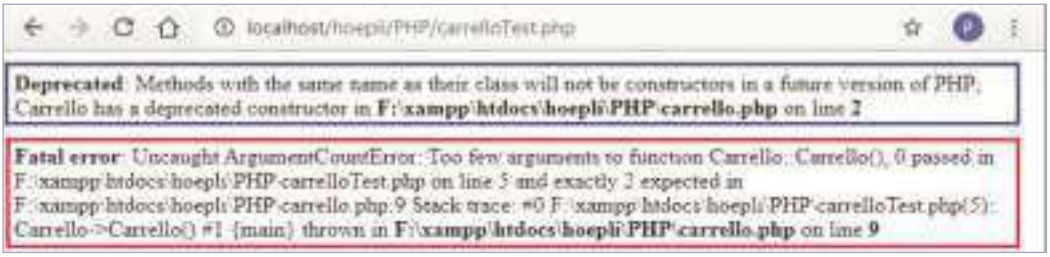
**Nella classe**

```
8    // metodo costruttore con due parametri
9    function Carrello($p1, $p2){
10     $this-> aggiungi_carrello($p1, $p2);
11   }
```

**Nel programma di prova**

```
20   // creazione con costruttore
21   echo "<HR>Creo nuovo carrello <BR>";
22   $acquisti1 = new Carrello("tempera",5);
23   $acquisti1->stampa();
```

Avviando l'esecuzione, sullo schermo ci vengono visualizzati alcuni messaggi: analizziamoli uno alla volta:




## Costruttori deprecati

PHP 7 ha introdotto molte novità e forse quella che ha creato maggiori disagi ai programmatori è stata quella legata ai costruttori degli oggetti con la sintassi di PHP 4, deprecata da PHP 7: questa è la causa del primo messaggio che viene visualizzato sullo schermo.
In PHP 4 i costruttori usavano lo stesso nome della classe, come naturalmente avviene in tutti i linguaggi OOP, e questa convenzione sappiamo che a volte può creare qualche problema perché, se in seguito viene cambiato il nome della classe, è necessario rivedere tutto il codice per cambiare tutti i nomi dei costruttori e quello negli extends.

## Sintassi __construct()

Per ovviare a questo inconveniente PHP 5 ha poi introdotto il costruttore generico __construct, ma ignorato dalla "quasi totalità dei programmatori", amanti della classica notazione OOP.
Dalla versione PHP 7 i "vecchi costruttori sono stati deprecati e, quindi, è necessario migrare alla nuova sintassi e utilizzare lo stile di PHP 5 sia per evitare spiacevoli segnalazioni in output ma anche per adeguarsi all'evoluzione del linguaggio, che probabilmente dalla futura PHP 8 toglierà completamente il vecchio formalismo e le istruzioni ancora presenti provocheranno un fatal error: la nuova sintassi per i costruttori è la seguente:

```
<?php
class NuovoNome
{
  public function __construct() // costruttore stile php 7
  {
    echo "costruttore classe NuovoNome stile php7:";
  }
  ...
}
?>
```

## Aggiornamento vecchio codice

**AREA DIGITALE**

**Rename della classe**

Se abbiamo un codice già scritto e vogliamo mantenere i vecchi costruttori basta aggiungere il nuovo costruttore e far richiamare da questo il vecchio costruttore:

```
<?php
  class NomeClasse{
    public function __construct()  // costruttore stile php 5
    {
      $this->NomeClasse();          // richiama il vecchio costruttore
    }
    // costruttore notazione php 4
    public function NomeClasse()
    {
      . . .     // istruzioni del vecchio costruttore
    }
  }
?>
```

## ESEMPIO

### CLASSE CRONOMETRO

Supponiamo di avere una classe Cronometro scritta in PHP 4: aggiorniamo il suo costruttore aggiungendo la notazione PHP 5 in modo da "eliminare" il messaggio di warning.

**La classe**

```
cronometro.php
1  <?php
2  class Cronometro
3  {
4    var $inizio, $fine;
5    public function __construct(){  // notazione php 5
6      echo "costruttore Cronometro stile php 5 <br>";
7      $this->Cronometro();
8    }
9    public function Cronometro(){  // notazione php 4
10     echo " -> si esegue costruttore stile php 4 <br>";
11     $this->azzera();
12   }
13   function azzera(){
14     return $this->inizio = $this->fine = 0;
15   }
16   function val(){
17     $this->inizio += time() - $this->fine;
18     return $this->fine;
19   }
20   function parziale(){
21     return time() - $this->inizio;
22   }
23   function fermati(){
24     return $this->fine = time() - $this->inizio;
25   }
26 }
```

**Il programma di prova**

```
28 //istanza di una classe Cronometro
29 $chrono = new Cronometro;
30 echo 'Vai:        ', $chrono->vai(),'<BR>';
31 sleep(2);
32 echo 'parziale:   ', $chrono->parziale(),'<BR>';
33 sleep(2);
34 echo 'fermati:    ', $chrono->fermati(),'<BR>';
35 echo 'azzera:     ', $chrono->azzera(),'<BR>';
36 echo 'Vai:        ', $chrono->val(),'<BR>';
37 sleep(1);
38 echo 'parziale:   ', $chrono->parziale(),'<BR>';
39 sleep(2);
40 echo 'fermati:    ', $chrono->fermati(),'<BR>';
41
42 ?>
```

**L'output**

```
costruttore Cronometro stile php 5
-> si esegue costruttore stile php 4
Vai: 0
parziale: 2
fermati: 4
azzera: 0
Vai: 0
parziale: 1
fermati: 3
```

## Costruttore con più parametri

Vediamo ora di eliminare la seconda scritta che, a differenza della prima, indica un fatal error, ed è dovuta al fatto che nel programma i primo oggetti sono creati con 0 parametri mentre l'ultimo oggetto con 2 parametri: questa differenza non è ammessa da PHP.

> In PHP è possibile definire un unico costruttore, a differenza degli altri linguaggi OOP dove i costruttori si distinguono per il tipo e il numero dei parametri.

È quindi necessario modificare il costruttore per fare in modo che "si comporti" differentemente in base al numero dei parametri con il quale viene invocato: di seguito scriviamo il codice che distingue la presenza di 0 oppure di due parametri:

```
8     public function __construct() // costruttore stile php 5-7
9     {
10      echo "costruttore Carrello stile php 5-7:";
11      if (0 === func_num_args()) {                // costruttore senza parametri
12        echo " -> senza parametri <br>";
13      } else if (2=== func_num_args()) {          // costruttore con 2 parametri
14        echo " -> con 2 parametri <br>";
15        $this->aggiungi_carrello (func_get_arg(0), func_get_arg(1));
16      }
17    }
```

## L'ereditarietà

Secondo il concetto di ereditarietà della OOP, le classi possono essere estese ad altre classi, le classi "estese" possiedono tutte le proprietà e i metodi della classe base e in più quello che viene aggiunto nella definizione estesa.

PHP non possiede tuttavia l'ereditarietà multipla.

La parola riservata per definire un'estensione di classe, come in Java, è extends.
La sintassi che segue mostra una classe MioCarrello che eredita dalla classe Carrello e la estende aggiungendone il metodo setPrezzo():

```
carrelloEredita.php   ×   carrello.php   ×
1  <?php
2  include "Carrello.php";
3  class MioCarrello extends Carrello
4  {
5    // definizione di una nuova proprietà
6    var $prezzi = array();
7    // metodo aggiuntivo
8    function setPrezzo($a,$p){
9      // ciclo di ricerca del prodotto
10     for ($i = 0; $i < count($this->articoli); $i++){
11       if ($this->articoli[$i] == $a)     // verifico se è presente nel carrello
12         $this->prezzi[$i] = $p;          // salvo la posizione
13     }
14   }
```

Per poter visualizzare il contenuto del carrello compreso il nuovo attributo riscriviamo il metodo stampa():

```
15   // metodo di stampa a video dei prodotti (sovrascrive il precedente)
16   function stampa(){
17     for ($i = 0; $i < count($this->articoli); $i++){
18       echo "<B>Articoli:</B> " .$this->articoli[$i] ."<BR>";
19       echo "<B>Quantita':</B> " .$this->quantita[$i] ."<BR>";
20       echo "<B>Prezzo:</B> " .$this->prezzi[$i] ."<BR>";
21     }
22   }
23 }
```

Una sequenza di attivazione è la seguente, dove viene richiamato il metodo della classe padre per inserire un prodotto nel carrello:

```
25 //istanzio un nuovo oggetto della nuova classe derivata
26 $nuovo_c = new MioCarrello;
27 //aggiungo un articolo
28 $nuovo_c->aggiungi_carrello("Acquarelle", 3);
29 //aggiungo il prezzo all'articolo Acquarelle
30 $nuovo_c->setPrezzo("Acquarelle", 12);
31 $nuovo_c->stampa();
```

Nelle classi derivate il costruttore della classe padre non viene chiamato automaticamente quando viene chiamato il costruttore della classe derivata: se vogliamo richiamare il costruttore della classe ereditata dobbiamo indicarlo espressamente.

# VERIFICA... le competenze

## PROBLEMI

**1** Definisci una classe Convertitore, in grado di convertire valute diverse, per esempio Dollaro-Euro. Per il programma si richiede la scrittura di almeno un metodo per il calcolo, un metodo di stampa dei valori risultanti e la definizione di uno o più attributi privati ove memorizzare i dati.

**2** Definisci una classe Libro contenente i seguenti attributi: nome del libro (array di caratteri), prezzo, numero di scaffale, numero di pagine, casa editrice. Inoltre definisci i seguenti metodi:
- inizializza
- mostra
- applicaSconto

  i quali, rispettivamente, hanno i seguenti compiti:
  - inizializzare i campi dati dell'oggetto classe
  - stampare tutti i dati dell'oggetto
  - diminuire del 10% il prezzo del libro in oggetto

**3** Crea la classe Massimi che contenga i metodi max() e min(), che calcolino rispettivamente il valore massimo e minimo di due numeri passati come parametri.

**4** Crea una classe Negozio con due costruttori che inizializzino lo stato dell'oggetto, dotato dei seguenti parametri:
- proprietario
- nomeNegozio

**5** Crea una classe ContoCorrente che contenga tre costruttori che inizializzino lo stato dell'oggetto, dotato dei seguenti parametri:
- nome
- codiceConto
- saldo

**6** Crea la classe Prodotto, utilizzando meno codice possibile, con tre costruttori che inizializzino lo stato dell'oggetto, dotati dei seguenti parametri:
- codice
- descrizione;
- codice
- descrizione

**7** Crea la classe Fattura, utilizzando meno codice possibile, con tre costruttori, che inizializzino lo stato dell'oggetto, dotati dei seguenti parametri:
- cliente
- cliente
- numeroProdotti
- numeroProdotti

**8** Definisci una classe LineaBus per rappresentare oggetti linea di autobus urbano con il numero identificativo, il nome dei due capolinea e con opportuni metodi d'istanza per la descrizione del suo percorso.

**9** Definisci una classe Auto per rappresentare oggetti automobile con il nome della marca, il nome del modello, la targa e l'anno di immatricolazione e con opportuni metodi d'istanza.

**10** Definisci una classe Studente per rappresentare oggetti studente con il cognome, il nome, il codice fiscale, il numero di matricola e con opportuni metodi d'istanza.

**11** Scrivi una classe AggiornaData che in base al tipo e al numero dei parametri richiama costruttori differenti che modificano il valore di una variabile $timestamp: a tal fine utilizza le funzioni func_num_args(), func_get_args() e func_get_arg().
Il costruttore multiplo così ottenuto deve:
**a** con 0 parametri assegnare a $timestamp il valore di sistema;
**b** con 1 parametro numerico: assegnare tale valore a $timestamp;
**c** con 1 parametro stringa: interpretarlo come data e assegnare tale valore a $timestamp;
**d** con 3 parametri numerici: interpretarlo come GG,MM,AA, trasformali in formato data e assegnare tale valore a $timestamp.

Una soluzione è riportata nel file costruttoriMultipliSol.php.

# 2 Comunicazione client-server in PHP con AJAX

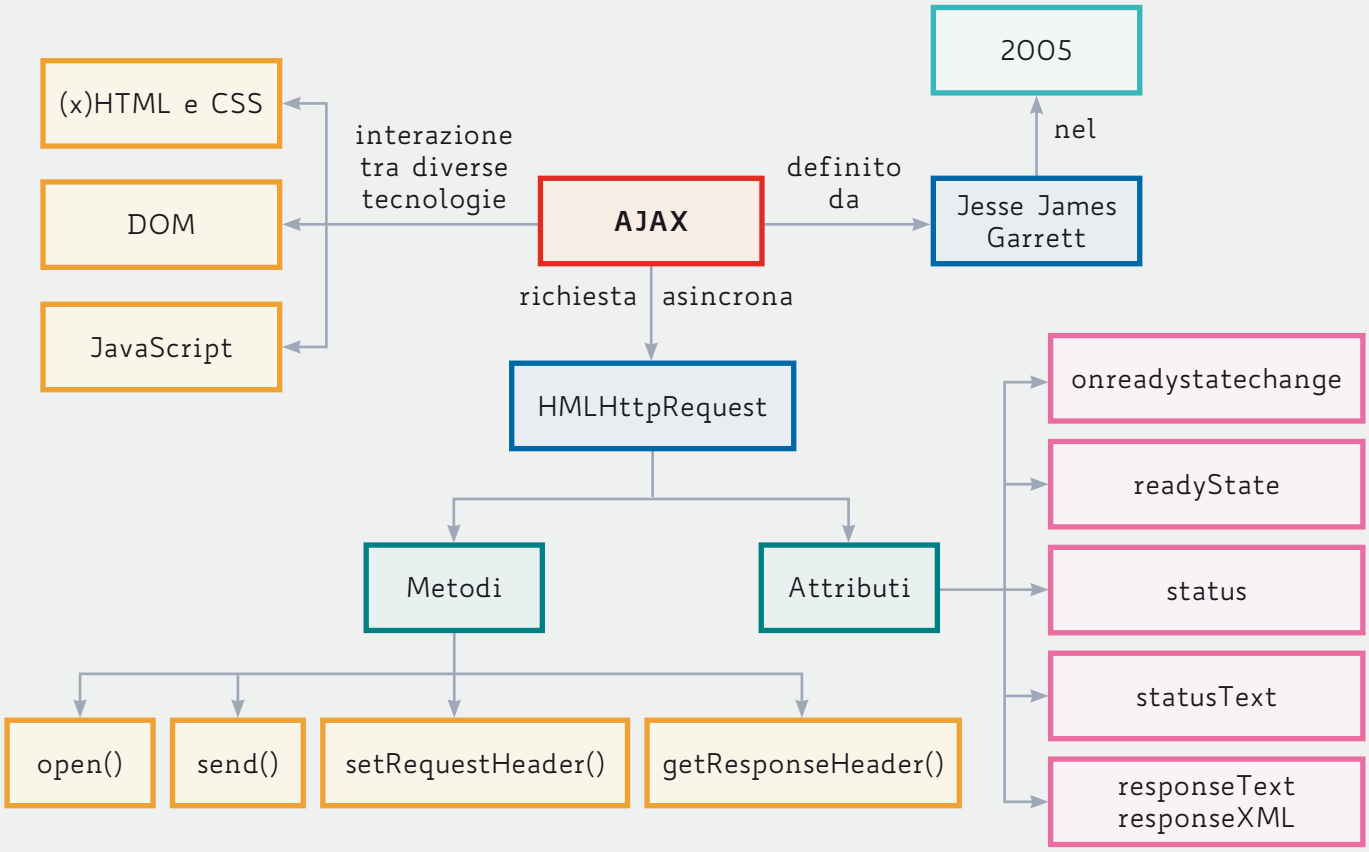
AREA DIGITALE
LEZIONE 3
La connessione al database MySQL

**IN QUESTA LEZIONE IMPAREREMO...**

- la tecnica AJAX
- l'utilizzo della classe XMLHttpRequest

## MAPPA CONCETTUALE

didattica inclusiva




## Generalità

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo per la realizzazione di applicazioni Web interattive: non è **una nuova tecnologia** né un nuovo linguaggio, bensì di un concetto utilizzato per sviluppare applicativi avanzati limitando lo spreco di risorse, permettendo a un client di richiamare informazioni dal lato server in modo veloce e trasparente.

Mediante AJAX possiamo consentire l'aggiornamento di una sezione di pagina HTML senza che questa venga caricata nuovamente dal server tramite richieste Http effettuate in JavaScript.

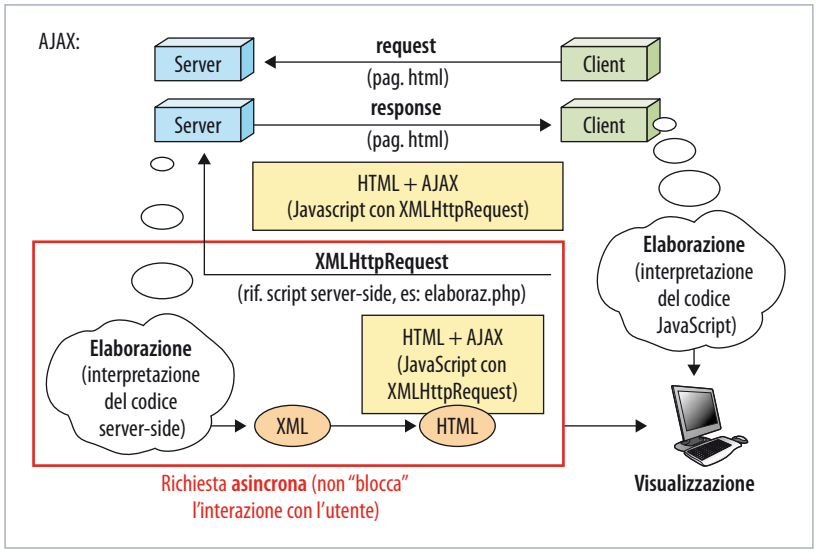
AJAX è stato progettato per creare applicazioni Web interattive, cioè per aggiornare in modo dinamico una pagina Web senza che l'utente debba ricaricarla: lavorando in modo asincrono, una volta chiamata la richiesta non è necessario aspettare che finisca la comunicazione per eseguire altre operazioni.
Dato che si basa su JavaScript, con AJAX è possibile chiamare pagine PHP, eseguirle e stampare a video il risultato senza dover ricaricare completamente la pagina: infatti per eseguire un codice JavaScript non serve il caricamento della pagina.
È quindi possibile richiede dati al server, mentre il PHP ha bisogno di inviare i dati al server, elaborarli e rimandarli indietro, mostrandoli solo se la pagina viene ricaricata.

L'unico problema che si riscontra è la presenza di molteplici browser e non tutti lo supportano direttamente: in alcuni è necessario effettuare un aggiornamento o scaricare un apposito add-on.

## Il funzionamento di AJAX

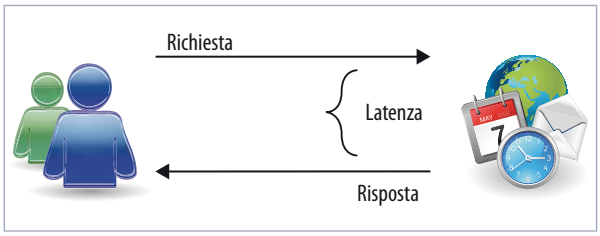
Il termine AJAX è stato coniato per la prima volta nel febbraio del 2005 da Jasse James Garrett, per descrivere un insieme di applicazioni Web dinamiche, basate sull'interazione tra diverse tecnologie:
- (x)HTML e CSS per la visualizzazione della pagina;
- DOM, modificato attraverso JavaScript, per offrire dinamicità alla pagina Web;
- XMLHttpRequest, che consente al browser e al server di comunicare *senza che la pagina venga ricaricata*, permettendo la creazione di pagine Web dinamiche più veloci.

Quindi il core di AJAX è rappresentato dalla classe XMLHttpRequest ma purtroppo l'utilizzo di questa classe varia a seconda del browser utilizzato: nell'esempio che segue vedremo come istanziare un oggetto di questa classe per versioni di browser differenti (i problemi maggiori si riscontrano con Explorer).
Lo schema a fianco illustra il ruolo di questa tecnologia nel tipico dialogo client-server attraverso il Web e il protocollo Http.

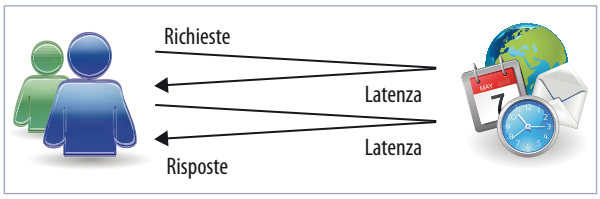



La classe XMLHttpRequest che consente di effettuare, con il protocollo Http, la richiesta di una risorsa a un Web server in modo indipendente dal browser. Nella richiesta è possibile inviare informazioni, ove opportuno, sotto forma di variabili di tipo GET o di tipo POST in maniera simile all'invio dati di un form.
La richiesta è asincrona, il che significa che non bisogna necessariamente attendere che sia stata ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso dati tipico di una pagina Web.

Infatti, generalmente il flusso è racchiuso in due passaggi alla volta, richiesta dell'utente (link, form o refresh) e risposta da parte del server, per poi passare, eventualmente, alla nuova richiesta da parte dell'utente. La figura a lato rappresenta una richiesta a un server con latenza e risposta. Inevitabile l'inconveniente del tempo di attesa/latenza necessario al server per produrre la risposta.




Con AJAX si perde questo schema: l'utente all'interno della stessa pagina può effettuare richieste multiple completamente indipendenti; teoricamente è possibile effettuare decine di richieste simultanee al server per operazioni differenti con o senza controllo da parte del navigatore.
Il nuovo schema è quello a fianco.




Rispetto al modello precedente rimane solo il tempo di attesa della risposta, che passa tuttavia in secondo piano rispetto ai vantaggi offerti da questa tecnologia. Questo tempo è anche uno dei maggiori problemi dell'utilizzo di AJAX, sia per gli sviluppatori sia per i navigatori: i primi potrebbero trovarsi in difficoltà qualora l'operazione asincrona dovesse attendere assolutamente una risposta al fine di completare una serie di operazioni più o meno sensibili, mentre i secondi potrebbero non avere idea di cosa stia accadendo alla pagina, chiudendola, ignari di aver richiesto un'informazione.

L'oggetto di classe XMLHttpRequest si aspetta una risposta dal server, dopo una chiamata, che non deve essere necessariamente di tipo XML ma che può essere semplicemente testuale (come nel nostro esempio).
Una volta aperta la richiesta, AJAX passa attraverso quattro stadi:

1 richiesta aperta, ma i dati non sono ancora stati inviati (OPENED: il metodo open() della XHR è stato invocato);
2 richiesta (dati) inviata, cioè il metodo send() è stato invocato e sono stati ricevuti gli header della risposta;
3 i dati sono ricevuti e comincia la loro lettura (LOADING: sta avvenendo il download del body della risposta Http);
4 operazione completata (DONE: l'operazione di fetch è terminata);

Il codice di stato è il valore che viene aggiornato nel suo attributo leggibile con il metodo readyState().

Riportiamo in una tabella i principali metodi della classe XMLHttpRequest.

| NOME METODO | DESCRIZIONE |
|---|---|
| open(metodo, URL)<br>open(metodo, URL, async)<br>open(metodo, URL, async, userName)<br>open(metodo, URL, async, userName, password) | Specifica il metodo, l'URL e altri parametri opzionali per la richiesta.<br>Il parametro metodo può assumere valore di "GET", "POST", oppure "PUT" ("GET" è utilizzato quando si richiedono dati, mentre "POST" è utilizzato per inviare dati).<br>Il parametro URL può essere sia relativo che assoluto.<br>Il parametro "async" specifica se la richiesta deve essere gestita in modo asincrono oppure no:<br>• con "true" lo script può proseguire l'elaborazione senza aspettare la risposta dopo il metodo send();<br>• con "false" lo script è costretto ad aspettare una risposta dal server prima di continuare. |
| send(content) | Invia la richiesta. |
| setRequestHeader(chiave, valore) | Aggiunge la coppia chiave/valore alla richiesta da inviare. |
| getResponseHeader(DOMString) | Restituisce gli header dei dati ricevuti dopo l'ultima request. |

La tabella che segue riporta i principali attributi della classe XMLHttpRequest.

| NOME PROPRIETÀ | DESCRIZIONE |
|---|---|
| onreadystatechange | Gestore dell'evento lanciato a ogni cambiamento di stato. |
| readyState | Contiene lo stato corrente dell'istanza di XMLHttpRequest:<br>0 (UNSET) = l'oggetto non è ancora stato inizializzato<br>1 (OPENED) = la richiesta è in fase di caricamento<br>2 (HEADERS_RECEIVED) = la richiesta è stata caricata completamente<br>3 (LOADING) = la richiesta è in attesa di interpretazione da parte dell'utente<br>4 (DONE) = la richiesta è stata completata |
| status | Contiene il codice Http restituito dal server:<br>200 = ok<br>404 ="Not Found" |
| statusText | Contiene l'etichetta dello status. |
| responseText | Contiene la risposta del server in formato stringa. |
| responseXML | Contiene il documento XML caricato (oggetto XML document); è significativo solo se readyState = 4, altrimenti è null. |

Una delle prime applicazioni a utilizzare questo componente è stato il servizio Google suggest che consente di ricercare una parola attraverso continui suggerimenti ricevuti in modo asincrono, proprio grazie alla tecnologia AJAX (ed è quella che realizzeremo anche noi).

Vediamo un primo semplice esempio.

## ESEMPIO

Scriviamo un documento HTML che permette di selezionare, tramite click su 4 bottoni, il caricamento asincrono di 4 diversi documenti HTML in una zona del documento stesso.
Scriviamo la prima parte di codice HTML dove visualizziamo quattro bottoni, rispettivamente con nome documento1, documento2, documento3 e documento4, e predisponiamo un  che chiamiamo zonaDinamica che sarà la nostra area dinamica nella quale verranno caricati, mediante AJAX, i documenti .html "connessi" ai quattro bottoni.

```
esempioAjax1.html
1  <!DOCTYPE HTML>
2  <html>
3    <head>
4      <title>AJAX: esercizio 1</title>
5    </head>
6    <body>
7     <font color="blue" size="4" ><strong>
8     Html con AJAX: caricamento dinamico di documenti</strong> </font><br/>
9      <hr/>
10     <div>
11       <button>documento1</button>
12       <button>documento2</button>
13       <button>documento3</button>
14       <button>documento4</button>
15     </div>
16     <hr/>
17     <div id="zonaDinamica">
18       <em>Qui viene caricato il documento scelto
19     </em> </div>
20     <hr/>
21     <font color="blue" size="3" >Parte inferiore della pagina<br/></font>
22     ...
23   </body>
```

Il codice della pagina prosegue con la sezione nella quale collochiamo il codice JavaScript: con il ciclo individuiamo il pulsante premuto e richiamiamo la funzione caricaDocumento che avvia le operazioni per effettuare il caricamento del relativo documento:

```
25    <script>
26      var documenti = document.getElementsByTagName("button");
27      for (var i = 0; i < documenti.length; i++) {
28        documenti[i].onclick = caricaDocumento;
29      }
```

La funzione che carica il documento crea un oggetto XMLHttpRequest, con il metodo open() predispone la richiesta e il metodo send() la inoltra.

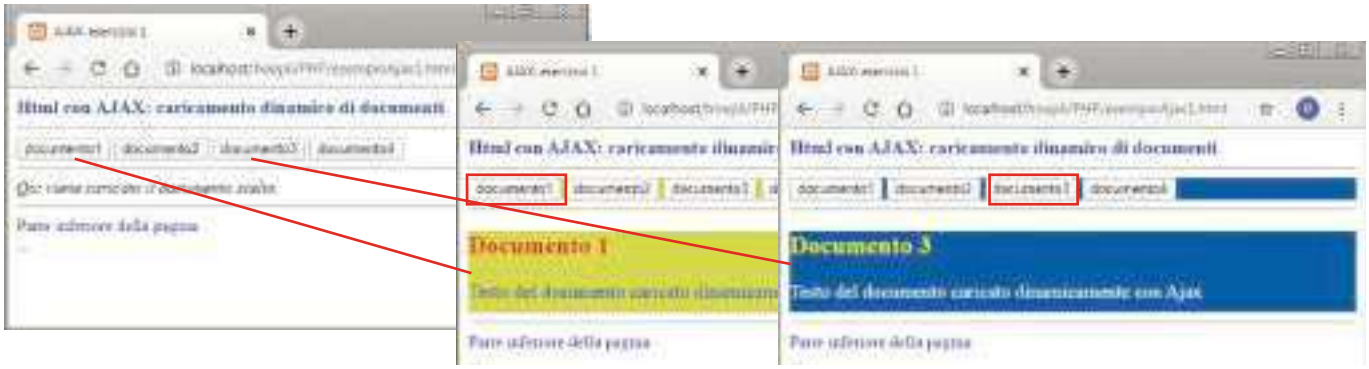
Rimane sempre "in ascolto" della modifica dello stato (riga 32) e se questo cambia valore richiama la funzione gestisciResponse che analizza il suo valore e si comporta di conseguenza:

```
30      function caricaDocumento(e) {
31        var httpRequest = new XMLHttpRequest();
32        httpRequest.onreadystatechange = gestisciResponse; // funzione ctr risposta
33        // il metodo open() ha 3 parametri:
34        //  -il primo definisce quale metodo usare (GET/POST)
35        //  -il secondo è l'uri dove risiede lo script server-side
36        //  -il terzo (boolean) specifica che la richiesta deve essere asincrona
37        httpRequest.open("GET", "http://localhost/hoepli/PHP/docAJAX/" + e.target.
            innerHTML + ".htm", true);
38        httpRequest.send();                                  // invia la richiesta
39      }
```

La funzione gestisciResponse controlla il valore degli attributi status, per verificare che tutto sia andato bene (OK), e readyState per verificare che la richiesta sia stata esaudita/completata (DONE):

```
40      function gestisciResponse(e) {
41        if (e.target.readyState == XMLHttpRequest.DONE &&  // costante con valore 4
42          e.target.status == 200) {                        // OK restituito dal server
43        // trasmette dal server documento responseText (tipo testo/stringa)
44          document.getElementById("zonaDinamica").innerHTML = e.target.responseText;
45        }
46      }
47    </script>
48  </html>
```

Dopo aver caricato il programma e i file sul server, carichiamo la pagina html e esaminiamo alcuni documenti per verificare il corretto funzionamento del nostro codice.




Vediamo ora un esempio completo che richiede anche un codice PHP; l'esempio consiste in una ricerca di un nome simulando il funzionamento tipo Google suggest.

## ESEMPIO

### SIMULAZIONE DI GOOGLE SUGGEST

#### Situazione

Si vuole realizzare un programma che aiuta l'utente a ricercare un nome compreso in un insieme di nomi, visualizzando a ogni carattere inserito solo l'elenco di quelli che "sono possibili soluzioni".

#### Soluzione

Il programma è diviso in due sezioni, client e server, la prima contenuta nel file esempioAjax.html che utilizza HTML, JavaScript e AJAX e la seconda nel file php_con_ajax.php che utilizza il solo linguaggio PHP.
Il funzionamento è semplice: la pagina HTML riceve dall'utente una stringa che viene ricercata nell'array di nomi presente sul server nel file PHP attraverso AJAX, viene effettuata la ricerca nell'array e restituita una stringa che contiene tutti i nomi che iniziano con la stringa di ricerca; tale stringa aumenta di dimensione man mano che l'utente compone il nome desiderato.

> Con AJAX la comunicazione è assai efficiente in quanto la richiesta al server avviene a ogni tasto premuto, dinamicamente, simulando il funzionamento di Google suggest.

Descriviamo il codice a partire della pagina esempioAjax.html: nella ultime righe del file abbiamo la sezione HTML che mostra una casella di testo in cui l'utente deve inserire la stringa da cercare (riga 45).
A ogni pressione del tasto, mediante l'evento onkeyup, viene richiamata la funzione mostra() alla quale viene passato il testo digitato fino a quell'istante; un tag contenitore  risposta (riga 47) servirà per posizionare i risultati.

```
42  <BODY>
43  <P><B>Scrivere nella casella le iniziali:</B></P>
44  <FORM>
45  Nome: <INPUT TYPE="text" onkeyup="mostra(this.value)">
46  </FORM>
47  <P>Nominativi trovati: <DIV ID="risposta"></DIV></P>
48  </BODY>
49  </HTML>
```

La funzione mostra(), se la stringa ricevuta non esiste (righe 5÷8), colloca una stringa vuota nel  risposta. A questo punto viene istanziato l'oggetto di classe XMLHttpRequest in modalità diversa a seconda delle versioni del browser (righe 10÷15).

```
1   <HTML><HEAD>
2   <SCRIPT>
3   // funzione attivata a ogni pressione di un tasto sulla casella di testo
4   function mostra(str){
5     if (str.length == 0){  // se stringa letta vuota viene inviata risposta blank
6       document.getElementById("risposta").innerHTML = "";
7       return;
8     }
9     // creazione oggetto XMLHttpRequest per vari tipi di Browser
10    if (window.XMLHttpRequest){  // Browser IE7+, Firefox, Chrome, Opera, Safari
11      xmlhttp = new XMLHttpRequest();
12    }
13    else{                        // browser IE6, IE5
14      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
15    }
```

Alla proprietà onreadystatechange associamo una funzione anonima (righe 18÷31) nella quale viene scritto il codice processato quando il server risponderà alla chiamata: si tratta di un codice che viene eseguito in modo asincrono, solo quando il server restituisce qualcosa, come una *sorta di interrupt*.
Dopo aver verificato le proprietà readyState e status, che ci informano se e come il server ha risposto (riga 27), il risultato ottenuto, rappresentato dalla proprietà responseText (riga 29), viene collocato nel  risposta.

```
16    // dopo una richiesta serve una funzione a ricevere i dati restituiti dal server
17    // onreadystatechange contiene la funzione che processerà la risposta del server
18    xmlhttp.onreadystatechange = function(){
19      // readyState contiene lo stato della risposta del server. Ogni volta che
20      // readyState cambia, la funzione onreadystatechange viene eseguita
21      // 0 la richiesta non è inizializzata
22      // 1 è stabilita la connessione col server
23      // 2 la richiesta è ricevuta
24      // 3 la richiesta è processata
25      // 4 la richiesta è completa
26      // se status=200 OK    -   404 Not found
27      if (xmlhttp.readyState == 4 && xmlhttp.status == 200){
28        // i dati trasmessi dal server sono ricevuti da responseText (tipo stringa)
29        document.getElementById("risposta").innerHTML = xmlhttp.responseText;
30      }
31    }
```

L'ultimo segmento di codice deve essere posizionato dopo la sezione onreadystatechange ma viene eseguito per primo, che è quello che predispone e apre la comunicazione.
Il metodo open() (riga 36), infatti, imposta l'invio della richiesta al server: possiamo notare che tra i suoi parametri vi sono il metodo di invio (GET o POST) e la pagina a cui inviare la richiesta, in questo caso php_con_ajax.php; il metodo send() (riga 39) realizza infine l'invio effettivo della richiesta.

```
36      xmlhttp.open("GET", "php_con_ajax.php?stringa=" + str, true);
37      xmlhttp.send();
38    }
39    </SCRIPT></HEAD>
```

Descriviamo ora il codice della pagina php_con_ajax.php che riceve la richiesta asincrona.

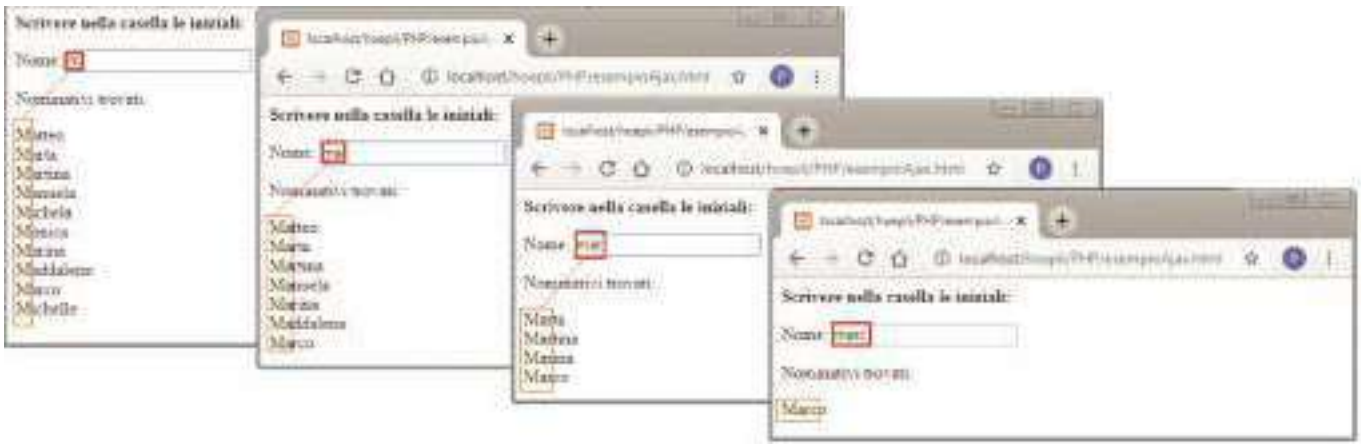
```
1   <?php
2   //array che contiene i nomi
3   $a = array("Anna","Brigitta","Claudio","Carmela","Evelina","Franco","Giovanna","Luca","Matteo","Gabriele","Marta","Martina","
      Manuela","Michela","Monica","Nanni","Pamela","Paolo","Paola","Pietro","Vincenzo","Alessandro","Riccardo","Chiara","Eva","
      Marina","Maddalena","Marco","Michelle","Antonio","Giuseppe","Lisa","Elisabetta","Giuliana","Arnoldo","Oberto","Umberto");
4   // salvo campo GET letto dalla pagina precedente
5   $q = $_GET["stringa"];
6   $response = "";
```

Nella prima parte viene dichiarato un array che contiene l'elenco dei nomi su cui cercare (riga 3); successivamente viene effettuata la ricerca (righe 8÷18) e restituito l'elenco dei nomi separati dal tag <BR>.

Osserviamo che la restituzione avviene con una normale echo (riga 27).

```
7   // verifica se campo esiste
8   if (strlen($q) > 0){
9     $risposta = "";
10    // ciclo di ricerca nell'array
11    for($i = 0; $i < count($a); $i++){
12      // controllo se nome trasformato in minuscolo è stato trovato
13      if (strtolower($q) == strtolower(substr($a[$i], 0, strlen($q)))){
14        // viene aggiunto il nome alla stringa separandolo dal tag di invio a capo
15        $risposta .= $a[$i] . "<BR>";
16      }
17    }
18  }
19  // verifica se $risposta è vuota restituisce stringa "Nessun nome trovato"
20  if ($risposta == ""){
21    $response = "Nessun nome trovato";
22  }
23  else{                        // assegnazione stringa di risposta
24    $response = $risposta;
25  }
26  // invio risposta alla pagina chiamante
27  echo $response;
28  ?>
```

Caricando la pagina esempioAjax.html osserviamo come a ogni cambiamento della parola scritta nella casella di testo venga mostrato un elenco che si aggiorna in tempo reale:



# VERIFICA... le competenze

## ESERCIZI

**1** Scrivi un documento HTML che permetta di caricare in modo asincrono, mediante posizionamento del mouse su due stringhe di testo, due diversi documenti HTML in due zone distinte del documento stesso.

**2** Dato il seguente documento HTML:

```
51  <html>
52    <body>
53      inizio documento
54      <br/><div>zona di visualizzazione di source1.htm</div><br/>
55      <br/><div>zona di visualizzazione di source2.htm</div><br/>
56      <br/><div>zona di visualizzazione di source3.htm</div><br/>
57      fine documento
58    </body>
59  </html>
```

modificalo in modo da poter selezionare, tramite un menu a 3 opzioni, il caricamento asincrono dei 4 documenti HTML source1.htm, source2.htm, source3.htm, che vanno visualizzati rispettivamente all'interno del primo, del secondo e del terzo elemento .

**3** Scrivi un documento HTML che presenti due elenchi di 4 stringhe di testo ciascuno, e che permetta di caricare in modo asincrono, mediante posizionamento del mouse su tali stringhe, 8 diversi documenti HTML in due zone distinte del documento stesso (una zona per i documenti relativi alla prima lista, l'altra per i documenti relativi alla seconda lista).

**4** Dato il seguente documento HTML:

```
61  <html>
62    <body>
63      inizio documento
64      <br/><br/><span>zona di visualizzazione 1</span><br/>
65      <br/><br/><span>zona di visualizzazione 2</span><br/>
66      <br/><br/><span>zona di visualizzazione 3</span><br/>
67      <br/>fine documento
68    </body>
69  </html>
```

modificalo in modo da poter selezionare, tramite un menu a 6 opzioni, il caricamento asincrono dei 6 documenti HTML index1.htm, index2.htm, …, index6.htm: i primi due documenti vanno visualizzati all'interno del primo elemento , il terzo e il quarto all'interno del secondo elemento , e gli ultimi due documenti all'interno del terzo elemento .

**5** Dato il seguente documento HTML:

```
72  <html>
73    <body>
74      inizio documento...
75      <p>prima zona</p><p>seconda zona</p><p>terza zona</p><div>quarta zona</div>
76      ...fine documento
77    </body>
78  </html>
```

modificalo in modo da poter selezionare, tramite 9 bottoni, il caricamento asincrono dei 9 documenti HTML newDoc1.htm, newDoc2.htm, …, newDoc9.htm: i primi tre documenti vanno visualizzati all'interno del primo elemento <p>, il quarto, quinto e sesto documento all'interno del secondo elemento <p>, e il settimo, ottavo e nono documento all'interno del terzo elemento <p>. Inoltre l'elemento  deve sempre visualizzare l'ultimo documento caricato.

**6** Apri i file esempio Ajax.html e php_con_ajax.php:

**a** modifica il codice dell'esempio in modo che il nome venga selezionato da un elenco a tendina;

**b** modifica il codice della pagina PHP dichiarando a inizio codice una matrice formata da nome e cognome. Il nome ricevuto dallo script deve essere ricercato nella matrice e devono essere restituiti i cognomi corrispondenti.

# 2 ESERCIZI IN LABORATORIO

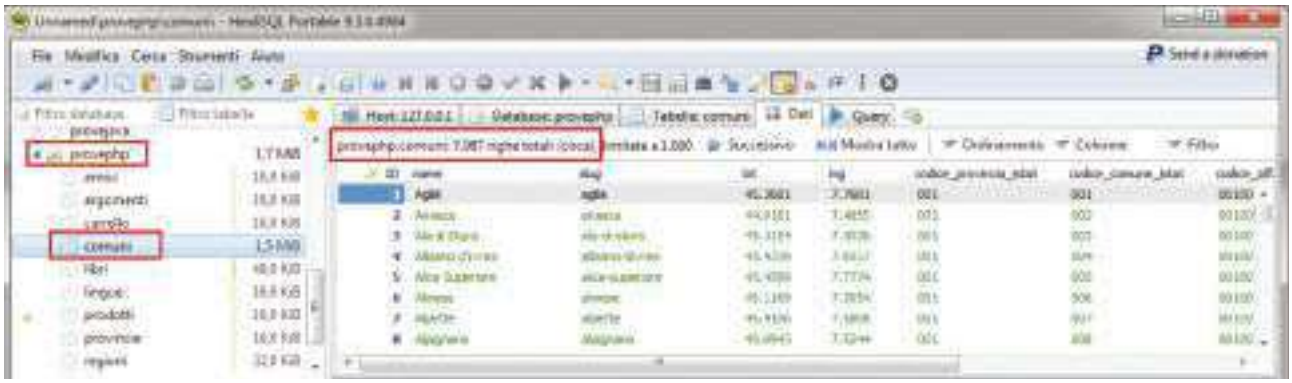
**AREA DIGITALE**

**ESERCIZI IN LABORATORIO 1**
**Installazione di EasyPHP e XAMPP**

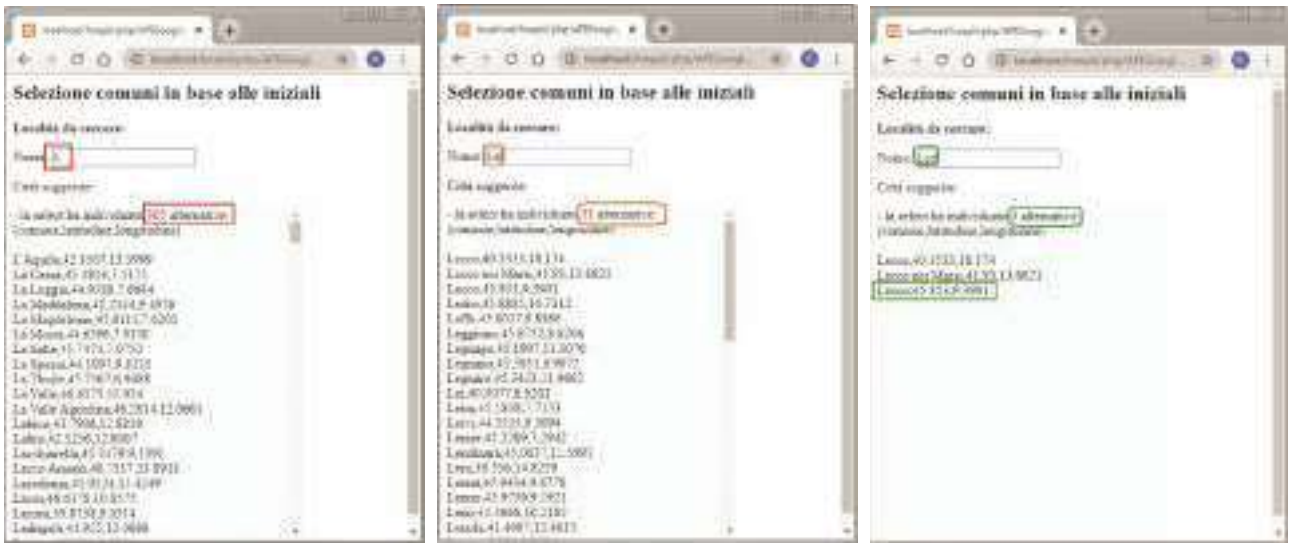
## UTILIZZO DI AJAX NELLA INTERROGAZIONE DI DATABASE MySQL

### Connessione con l'utilizzo di AJAX

Scriviamo ora un programma che utilizza AJAX per realizzare un meccanismo simile a Google suggest per individuare le coordinate di geo-referenziazione di un comune italiano letto dal una apposita tabella comuni presente sempre nel database provephp.



L'utente farà la sua ricerca digitando in una casella di testo il nome della città desiderata e, in tempo reale, utilizzando AJAX, il programma visualizza sullo schermo il risultato della interrogazione fatta alla tabella comuni.
Per esempio, ricercando "LECCO" otteniamo sullo schermo le seguenti situazioni, dove da un insieme di 305 possibili risultati alla digitazione del terzo carattere individuiamo tre sole città che hanno le iniziali digitate.



# ESERCIZI IN LABORATORIO 2

Per fare questo la pagina utilizza i linguaggi HTML, JavaScript, PHP, SQL e le librerie per AJAX.
L'esempio è formato da due file:

**1** il primo, ajaxMysql.html**,** lato client, che ha il compito di:
- leggere il nome del comune da cercare con una casella di testo e trasferirlo alla pagina elabora.php questa pagina;
- ricevere l'elenco di comuni trovati e collocare i nomi restituiti dal database in un riquadro (chiamato box) posto all'interno di un  della pagina HTML che viene formattato con un sfondo definito con uno stile CSS;
- mostrare il risultato in un ;

**2** il secondo, elabora.php, lato server, che ha il compito di:
- ricevere la stringa da cercare;
- effettuare un'interrogazione alla tabella comuni del database provephp;
- restituire l'elenco di comuni trovati all'interno di una stringa alla pagina HTML**.**

Iniziamo a vedere il codice della prima pagina ajaxMysql.html**:**

```
<HTML><HEAD>
<STYLE>
div.box{
   background-color:azure;
   width:350px;
   height:400px;
   overflow:auto;
}
</STYLE>
```

Prima di tutto creiamo il foglio di stile necessario alla visualizzazione del  chiamato 'box' che conterrà l'elenco dei comuni visualizzati nella pagina HTML.
La pagina HTML può essere divisa in 2 sezioni:

**1** la sezione in cui utilizziamo AJAX per collegarci con la pagina elabora.php;
**2** la sezione di HTML che visualizza i risultati.

Descriviamo la sezione di AJAX che è "racchiusa" nella funzione mostra(): questa funzione viene chiamata dalla sezione HTML (riga 56) ogni volta che l'utente digita un carattere nella casella di testo ricevendo come parametro str la stringa digitata fino a quel momento. L'istruzione della riga 13 valuta il valore della stringa e se ha lunghezza nulla con l'istruzione della riga 14 viene cancellato il contenuto del  'elenco'.

```
11  <SCRIPT TYPE="text/javascript">
12  function mostra(str){
13    if (str.length == 0){
14      document.getElementById("elenco").innerHTML = "";
15      return;
16    }
17    if (window.XMLHttpRequest){     // se il browser IE7+, Firefox, Chrome, Opera, Safari
18      ajax=new XMLHttpRequest();
19    }
20    else{                           // se il browser IE6, IE5
21      ajax=new ActiveXObject("Microsoft.XMLHTTP");
22    }
```

In base al tipo di browser viene creato un oggetto xmlhttp con proprietà onreadystatechange che contiene la funzione anonima necessaria per elaborare i dati in provenienza dal server (riga 24):

- la proprietà readyState contiene l'esito della richiesta: se ha valore 4 significa che è stata completata correttamente;
- la proprietà status invece contiene lo stato della risposta: se vale 200 significa che la risposta è corretta, se invece vale 404 indica "Not found".

# 2 ESERCIZI IN LABORATORIO

Nella riga 27 viene verificato il valore di questi attributi e, in caso di esito positivo, viene collocata la proprietà responseText nel riquadro riservato per il testo (riga 30), cioè viene inserito l'elenco dei nomi ricevuti dal server all'interno del  'elenco'.

```
23    // onreadystatechange contiene la funzione che elabora i dati ricevuti dal server
24    ajax.onreadystatechange=function(){
25      // readyState 4: la richiesta e' completa
26      // status: 200 OK, 404 Not found
27      if (ajax.readyState==4 && ajax.status==200){
28        // i dati ricevuti sono ricevuti da responseText di tipo di stringa
29        // e vengono inseriti nel div chiamato 'elenco'
30        document.getElementById("elenco").innerHTML = ajax.responseText;
```

L'ultima parte della sezione di AJAX è assai interessante: viene dichiarato un array di comodo chiamato tmp[] che conterrà le singole righe dell'elenco dei comuni (riga 32) e una variabile stringa che conterrà l'elenco così come ricevuto dal server (riga 34).

```
31        // utilizzo di un array temporaneo
32        var tmp = new Array()
33        // variabile stringa contiene tutto l'elenco di citta' e coord ricevute dalla pag.php
34        var stringa = ajax.responseText;
35        // viene creato un array formato dalle righe dell'elenco di citta'
36        tmp = stringa.split("<BR>");
37        conta = 0;
38        posti = [];
```

La variabile stringa è formata da diverse righe, ciascuna delle quali termina con il tag <BR> necessario per andare a capo: il metodo split("<BR>") della riga 36 riempie l'array tmp dove ogni elemento è una riga dell'elenco, come vediamo dall'esempio che segue che riporta il contenuto di stringa alla riga 34:

```
Millesimo,44.3641,8.2059<BR>Milano,45.4612,9.1878<BR>Milzano,45.2745,10.2003<BR>Milazzo,
38.2194,15.2404<BR>Militello Rosmarino,38.0472,14.6778<BR>Milena,37.4708,13.7361...
```

e il contenuto di tmp dopo l'operazione di split di riga 36:

| | |
|---|---|
| 0 | Millesimo,44.3641,8.2059 |
| 1 | Milano,45.4612,9.1878 |
| 2 | Milzano,45.2745,10.2003 |
| 3 | Milazzo,38.2194,15.2404 |
| 4 | Militello Rosmarino,38.0472,14.6778 |
| ... | |

Nel ciclo for successivo (righe 40÷43) viene effettuata una conversione dell'array tmp, dove ciascuna riga viene aggiunta alla matrice posti, così che l'array posti possiede 3 colonne e un numero di righe pari agli elementi letti dall'array posti.

```
39        // creo la matrice: ogni riga ha 3 elementi (nome, latitudine, longitudine)
40        for (i = 0; i < tmp.length; i++) {
41          posti[conta] = tmp[i].split(",");  // hanno "," come separatore
42          conta++;
43        }
44      }
45    }
```

# ESERCIZI IN LABORATORIO 2

Il metodo open() (riga 47) invia la richiesta alla pagina elabora.php del server e infine il metodo send() trasmette effettivamente la richiesta al server (riga 48).

```
46    // apriamo la connessione con il server e inoltriamo con GET i parametri
47    ajax.open("GET","elabora.php?stringa=" + str, true);
48    ajax.send();    // invio della richiesta al server
49  }
50  </SCRIPT>
```

La parte finale del codice è rappresentata dal codice HTML che mostra la casella di testo in cui digitare il comune da ricercare (riga 56) alla quale è associato l'evento onkeyup che richiama la funzione mostra(this.value) di cui abbiamo già parlato in precedenza.
Viene poi mostrato il <DIV> per l'elenco dei risultati (riga 57):

```
52  </HEAD>
53  <BODY>
54  <H2>Selezione comuni in base alle iniziali</H2>
55  <P><B>Localita' da cercare:</B></P>
56  <FORM>Nome: <INPUT TYPE="text" onkeyup="mostra(this.value)"></FORM>
57  <P>Citta' suggerite: <div class="box" id="elenco"></DIV></P>
58  </BODY>
59  </HTML>
```

La pagina elabora.php riceve il campo GET stringa e lo salva nella variabile $q (riga 3), quindi viene inizializzata la stringa $citta che verrà restituita con i nomi delle località (riga 4).
Siccome la pagina può essere richiamata senza che venga inviata alcuna stringa, lo script effettua preliminarmente un controllo sulla lunghezza della stringa (riga 5), se maggiore di 0 viene effettuata la connessione al database provephp di MySQL (righe 6÷10).

```
1   <?php
2   // ricevo il campo GET con inizio del nome della citta'
3   $q = $_GET["stringa"];
4   $citta = "";                          // per mettere l'elenco nomi delle localita'
5   if (strlen($q) > 0){                  // se e' presente una testo
6     //genero un array che contiene i nomi
7     $db = new mysqli("localhost", "root", "", "provephp");
8     if (mysqli_connect_errno()) {       // verifica l'avvenuta connessione
9       printf("Connessione non effettuata: %s\n", mysqli_connect_error());
10      exit();
11    }
```

Quindi viene memorizzata la query di selezione (riga 14) ed eseguita (riga 15).

```
13    // stringa di interrogazione
14    $query ="SELECT name,lat,lng FROM comuni WHERE name like '".$q."%' ORDER BY name;";
15    if ($res = $db->query($query)) {        // esecuzione query
16      printf("- la select ha individuato %d alternative:","<BR>", $res->num_rows);
17    }
```

Come possiamo notare, nella query di selezione viene usata la clausola like seguita dalla stringa di ricerca che termina con il simbolo di percentuale (%): questo consente di effettuare un confronto parziale tra il nome della località e la sottostringa indicata prima del simbolo di percentuale, otteniamo come risposta le località che iniziano con i caratteri indicati.

# 2 ESERCIZI IN LABORATORIO

Se la query produce un risultato, cioè è stata trovata almeno una tupla che soddisfa la condizione di selezione (riga 19), il ciclo foreach (righe 21÷23) legge e analizza tali risultati e concatena i nuovi dati ricevuti alla variabile $citta separando ogni riga con il tag HTML <BR>:

```
19    if ($res->num_rows > 0){                    // se presente almeno un comune
20      printf("(comune,latitudine,longitudine)<BR><BR>");
21      foreach( $res as $riga ) {                // ciclo che estrae i record restituiti da SQL
22        $citta.=$riga["name"].",".$riga["lat"].",".$riga["lng"]."<BR>";
23      }
24    }
25    $db->close();                               // chiusura connessione
26  }
```

Se al termine dell'iterazione la stringa $citta ha lunghezza maggiore di 0 viene inviata in risposta alla pagina precedente (righe 28÷33).

```
28  if (strlen($citta) == 0){                     // controllo se esiste un risultato
29    echo "nessun nome trovato!";
30  }
31  else{
32    echo $citta;
33  }
34  ?>
```

Il risultato di un'elaborazione è stato riportato in apertura dell'Esercizio.

## METTITI ALLA PROVA

• Utilizzare gli ascoltatori di evento con AJAX • Utilizzare PHP • Utilizzare il codice SQL di MySQL

1 Apri i file ajaxMysql.html ed elabora.php e modifica il codice in modo tale che vicino a ogni città venga visualizzata anche la provincia e la regione di appartenenza: a tale scopo utilizza le tabelle provincie e regioni del database provephp.

2 Apri il file esercizio1Ajax.html e esercizio1Ajax.php presente nella cartella esercizi di htdocs/php, dove viene realizzata una semplice implementazione del meccanismo di selezione intellisense simile a Google suggest per la scelta di alcuni nomi presenti in un array.

a Modifica il codice dell'esempio in modo tale che il nome venga selezionato da un elenco a tendina invece che digitato in una casella di testo.

b Modifica il codice della pagina PHP dichiarando a inizio codice una matrice formata da nome e cognome: il nome ricevuto dallo script deve essere ricercato nella matrice e devono essere restituiti gli eventuali cognomi corrispondenti presenti nell'array.

3 Scrivi una pagina PHP che, utilizzando il database comuni.mdb, deve mostrare una casella combinata con l'elenco delle regioni. L'utente sceglie la regione e, a quel punto, appare l'elenco delle province di quella regione, sempre in una casella combinata. Dopo aver scelto la provincia appariranno in una casella combinata l'elenco dei comuni di quella regione.
Il tutto deve essere contenuto in una sola pagina, utilizzando la tecnica postback.

# ESERCIZI IN LABORATORIO 2

## Esercizi proposti

**1** Scrivi una pagina PHP che, utilizzando il database giochi.mdb per memorizzare i punteggi realizzati in videogiochi da parte di un utente, deve offrire le seguenti opzioni:

**a** presentare un form di login in cui l'utente può accedere solo se conosce i propri dati di autenticazione (tabella Utenti);

**b** una volta effettuato il login l'utente può compiere una delle seguenti azioni, cliccando su uno dei due link presenti nella stessa pagina:

– inserire il nome del videogioco e il suo punteggio numerico aggiornando le tabelle giochi e ; se l'utente invia più di una volta un punteggio per lo stesso videogioco nella stessa sessione, il punteggio memorizzato è uguale al massimo dei valori inseriti;

– visualizzare un riepilogo con l'elenco di tutti i record dei vari utenti in ordine, con indicato anche il videogioco dell'utente loggato (quest'ultimo, magari di colore diverso).

**2** Scrivi una pagina PHP che, utilizzando il database medagliere.mdb, deve mostrare un menu a discesa (<select>) con la lista dei paesi partecipanti e un menu con il tipo di medaglia (oro, argento e bronzo). Ogni volta che viene selezionato un paese e una medaglia, la medaglia di quel tipo verrà aggiunta al paese selezionato.
Deve anche essere presente un link a una pagina di riepilogo del medagliere, con i paesi ordinati per quantità di medaglie con gli stessi criteri utilizzati alle Olimpiadi.

**3** Scrivi una pagina PHP che, dopo la pressione di un pulsante, generi un numero casuale a tre cifre, lo mostri a video sul pulsante, lo ricerchi nel campo latitudine (Lat) della tabella comuni del database comuni.mdb, quindi estragga tutti i comuni che hanno le prime tre cifre decimali della latitudine uguali a quelle generate inserendole in una casella combinata (<select MULTIPLE>).
A ogni click sulla località devono apparire tutti gli altri comuni che iniziano con la stessa lettera della stessa provincia.

**4** Realizza una pagina Web in tecnologia AJAX che consenta di visualizzare il prezzo e le caratteristiche tecniche dei prodotti di un'azienda di informatica. La pagina è costituita da un form contenente un campo di testo editabile dall'utente. Il campo di testo deve consentire all'utente di inserire il codice identificativo del prodotto di interesse. Il codice è costituito da una stringa composta da un carattere alfabetico seguito da tre caratteri numerici. Il form inoltre è dotato di un bottone che consente di inviare la request al server; a seguito della pressione del bottone la pagina deve:

**A** verificare che il codice identificativo del prodotto immesso dall'utente rispetti il formato specificato;

**B** nel caso in cui il codice identificativo del prodotto verifichi il formato specificato, ti si chiede di inserire sotto il form una lista puntata in cui vengano riportati prezzo e caratteristiche del prodotto cercato. Per semplicità assumiamo che il prodotto sia sempre presente sul server e che i dati che questi ci restituisce siano in formato XML e organizzati come segue:

```
<?xml version='1.0' encoding='UTF-16'?>
<prodotto>
<prezzo> prezzo </prezzo>
<marca> marca </marca>
<modello> modello </modello>
</prodotto>
```

dove prezzo, marca e modello rappresentano rispettivamente il prezzo del prodotto, la sua marca e il suo modello. Le informazioni devono essere richieste al server in modo asincrono tramite una chiamata GET e passando il codice del prodotto richiesto;

**C** nel caso in cui il codice identificativo del prodotto non verifichi il formato specificato, ti si chiede che venga presentato un pop-up che segnali il problema all'utente.

# 3 ESERCIZI IN LABORATORIO

## UPLOAD/DOWNLOAD DEI FILE E UTILIZZO DI FILE JSON

### I file in php

I file vengono gestiti in php con istruzioni analoghe a quelle del linguaggio. L'apertura di un file avviene ricorrendo alla funzione fopen(), con la quale al file aperto viene associato un puntatore che rappresenta l'identificatore del file per operazioni di lettura e scrittura.

```
$idmiofile=fopen("nome del file","modalità di apertura");
```

La chiusura avviene con la funzione fclose(), indicando come parametro il puntatore relativo al file.
Il nome del file è rappresentato dal percorso del file da aprire, mentre la modalità di apertura può essere una delle seguenti:

- r sola lettura
- r+ lettura e scrittura
- w sola scrittura: il contenuto viene perso (*)
- w+ lettura e scrittura, ma perdendo il contenuto (*)
- a solo per aggiunta (modalità append) (*)
- a+ per lettura e aggiunta (*)
- b da aggiungere ai precedenti per trattare il file come file binario

(*) Se il file non esiste php tenta di crearlo.

Per verificare se l'apertura del file è andata a buon fine, è necessario controllare il contenuto del puntatore al file: se esso è falso significa che l'apertura è fallita.

```
$idfile=fopen("accessi.txt","r");
// Se l'apertura del file fallisce la funzione die() visualizza il messaggio e chiude lo script
if (!$idfile)
 die ("Si è verificato un errore durante l'apertura del file");
```

Di seguito, riportiamo le principali funzioni per la gestione dei files in php.

```
fopen($filedaaprire, $mode)                    // Apre un file
fread($filedaleggere)                          // Legge da un file
unlink($filedaeliminare)                       // Elimina un file
file_exists($filedacontrollare)                // Controlla se un file esiste
is_writable($filedacontrollare)                // Controlla se un file è riscrivibile
is_readable($filedacontrollare)                // Controlla se un file è leggibile
fwrite($testodascrivere, $fileincuiscrivere)   // Scrive in un file eliminandone il contenuto precedente
fsize($filedacontrollare)                      // Restituisce la dimensione del file in byte
fclose($filedachiudere)                        // Chiude un file
```

La funzione che consente di leggere un file è fread(), che possiede due parametri, identificatore del file e numero di byte da leggere. Riportiamo qui di seguito un codice di esempio per leggere da un file contaccessi.txt i primi "n" caratteri:

# ESERCIZI IN LABORATORIO 3

```
// Apertura file in lettura
$idmiofile = fopen("/percorso/contaccessi.txt","r");
$n=100;
// Lettura di cento caratteri
$datiletti = fread($idmiofile,$n);
```

La funzione che permette di scrivere all'interno di un file è fwrite(). Possiede due parametri, il puntatore al file e la stringa da scrivere.
Per scrivere una riga e andare a capo è necessario far terminare la stringa con i caratteri di escape \r\n, come in questo esempio:

```
// Apertura file in lettura e scrittura con percorso relativo
// In questo caso il file deve trovarsi nella cartella del codice
$id = fopen("contaccessi.txt","w");
// Scrive la stringa indicata nel file mandando il testo a capo
$bytescritti = fwrite($id,"testo \r\n");
```

Vediamo di seguito un esempio in cui viene utilizzato un file per memorizzare il numero di accessi a una determinata pagina Web.

## ESEMPIO

### CONTATORE ACCESSI CON FILE

La prima operazione che viene effettuata per memorizzare il numero di accessi tramite file riguarda la verifica dell'esistenza del file mediante la funzione file_exists() (riga 3).
Quando il file esiste, verificato alla riga 4, dopo aver appurato se si riesce ad aprirlo (riga 6), si legge il valore in esso contenuto (riga 9); se il file non esiste, viene creato memorizzando 0 al suo interno (righe 13÷19).

```
contaAccessi.php
1  <?php
2    $nomefile = "dati/accessi.txt";          // file dove si memorizza il totale accessi
3    if (file_exists($nomefile)){             // verifica dell'esistenza del file
4      $idfile = fopen($nomefile, "r+");      // se esiste gia viene aperto in lettura
5      // se non si riesce ad aprirlo viene creato un messaggio di errore
6      if (!$idfile)
7        die ($msg = "Il file $nomefile non ?stato aperto <BR>");
8      // se il file viene aperto si leggono dieci caratteri e messi in $conta_accessi
9      $conta_accessi = (int)fread($idfile, 10);     // il casting da stringa in intero
10     fclose($idfile);                              // chiusura file
11   }
12   else {
13     // se il file non esiste viene creato e aperto contemporaneamente
14     $idfile = fopen($nomefile, "w+");
15     if (!$idfile)
16       die ($msg = "Il file $nomefile non e' stato aperto<BR>");
17     // se il file viene aperto correttamente
18     $conta_accessi = 0;                     // si inizializza la variabile conta_accessi
19     fclose($idfile);                        // chiusura del file
20   }
```

A questo punto viene incrementata la variabile $conta_accessi che contiene il totale di visite (riga 22) e, dopo aver riaperto il file in modalità scrittura distruttiva (riga 23), viene scritto il nuovo valore (riga 27); lo script termina visualizzando il totale sul browser (riga 30).

# 3 ESERCIZI IN LABORATORIO

```
22    $conta_accessi++;                          // incrementa contatore di accessi
23    $idfile = fopen($nomefile, "w+");          // apertura del file in scrittura
24    if (!$idfile)
25      die ($msg = "il file $nomefile non e' stato aperto<BR>");
26    // se il file aperto correttamente si scrive nel file del contatore di accessi
27    fwrite($idfile, $conta_accessi);
28    fclose($idfile);                           // chiusura file
29    echo($conta_accessi);                      // visualizza il contatore accessi
30    echo ('<p><a href=index.html><img src=../images/return.gif  width=41 height=30 align=right
        border=0></a></font></p>');
31  ?>
```

Per verificare il funzionamento dello script proviamo ad aprire più volte il browser sulla pagina php:

Puoi trovare il codice di questo esempio nel file contaAccessi.php.

## METTITI ALLA PROVA

→ • Utilizzo dei tag HTML in php • Utilizzo delle funzioni di lettura e scrittura sui file

**Apri il file Conta_accessi.php**

1 Modifica il codice dell'esempio in modo tale che mostri dei numeri grafici al posto del numero per il totale delle visite effettuate alla pagina. Per fare questo, utilizza le immagini (0.gif, 1.gif, ... 9.gif) presenti nella sottodirectory cifre della cartella MATERIALI.

2 Confronta la tua soluzione con quella presente nel file ContaAccessiSol.php.

## L'array associativo $_FILES

La variabile predefinita $_FILES contiene i nomi e le informazioni che riguardano i file provenienti dal campo di tipo file di un form. Ciò consente di effettuare l'upload di file dal client al server. In una comunicazione client/server che permetta di effettuare l'upload dei file, gli elementi fondamentali sono i seguenti:

- form HTML con campo file e proprietà enctype = multipart/form-data;
- funzione php move_uploaded_file(nome del file da caricare, percorso in cui caricarlo);
- array associativo $_FILES.

L'array $_FILES presenta la seguente struttura:

```
$_FILES[' nomecampofile ']["name"]        // nome e estensione del file caricato
$_FILES[' nomecampofile ']["type"]        // tipo di file caricato (in formato MIME type)
$_FILES[' nomecampofile ']["size"]        // dimensione del file caricato
$_FILES[' nomecampofile ']["tmp_name"]    // percorso completo del file temporaneo sul server
$_FILES[' nomecampofile ']["error"]       // codice numerico compreso fra 0 e 8 indicante il tipo di errore
```

Vediamo ora un esempio di come effettuare l'upload di un file immagine da un client a un server con un form HTML.

# ESERCIZI IN LABORATORIO 3

## ESEMPIO

### UPLOAD DI FILE

Scriviamo un programma per caricare file contenenti immagini in una directory predefinita (nel nostro caso c:\www) che alla fine dell'operazione visualizza il contenuto di questa directory.
Lo script si compone di due parti e ha come principio di funzionamento la tecnica PostBack, come si può vedere dall'istruzione di riga 19: al primo accesso, cioè quando il campo $_POST['invia'] non è stato ancora settato, viene eseguita la prima parte (righe 20÷25), nella quale viene presentato un form con un pulsante di invio (submit) e un campo di tipo file per selezionare l'immagine da inviare al server.

```
19  // scelta operazione da eseguire: tecnica PostBack
20  if (!isset($_POST['invia'])){
21    // viene mostrato il form
22    echo "<FORM ACTION='".$_SERVER['PHP_SELF']."' ENCTYPE='multipart/form-data' METHOD='post'>";
23    echo "<B>Caricamento di una immagine nella cartella c:\www\</B><br><br>";
24    echo "<INPUT TYPE='file' NAME='file_caricato'>";
25    echo "<INPUT TYPE='submit' value='Upload file' NAME='invia'>";
26    echo "</FORM>";
27  }
```

La seconda parte racchiude lo script che viene elaborato con la pressione del tasto di submit: come prima operazione vengono salvati i dati del file proveniente dal client in tre variabili di comodo (righe 30÷32).
Quindi viene verificato il tipo di file da caricare (riga 34): se il file non è una immagine, viene segnalato un errore e lo script termina, altrimenti viene usata la funzione move_uploaded_file(), che carica effettivamente il file dal client al server nel percorso specificato come secondo parametro, cioè contenuto nella variabile $percorso (che contiene C:/www).

```
28  else{
29    // trasferisco il tipo, il percorso completo e il nome del file in 3 variabili di comodo
30    $percorso ="C:/www/";
31    $f=$_FILES['file_caricato']['type'];
32    $nome=$_FILES['file_caricato']['name'];
33    $nome_tmp=$_FILES['file_caricato']['tmp_name'];
34    // verifico il tipo di file, se si tratta di un'immagine (jpg, gif o png)
35    if (($f=="image/jpeg")||($f=="image/gif")||($f=="image/png"))  {
36      // con move_uploaded_file il file viene caricato dal client al server
37      // nel percorso specificato come secondo parametro
38      move_uploaded_file($nome_tmp,$percorso.$nome);
39    }
40    else{
41      // se tipo file non consentito
42      echo "Estensione non consentita";
43    }
```

Completa il programma una sezione che visualizza sullo schermo il contenuto della directory che stiamo elaborando: a tal fine scriviamo una funzione leggiDirectory() che effettua il caricamento dei nomi dei file presenti nella directory che riceve come parametro e restituisce un array con il nome dei file stessi.

```
upload.php
1  <?php
2  Function leggiDirectory($dir){
3    $fileTrovati= array();
4    // apro la cartella che deve essere letta
5    $aperturaPercorso = opendir($dir);
6    // scorro tutti i file presenti nella cartella e li inserisco nell'array
7    while ($file = readdir($aperturaPercorso)){
8      if(is_file($dir.$file)){
9        array_push($fileTrovati,$file);
10     }
11   }
12   // chiudo la cartella che ho letto
13   $aperturaPercorso = closedir($aperturaPercorso);
14   // ritorno l'array con tutti i file
15   return $fileTrovati;
16 }
```

# 3 ESERCIZI IN LABORATORIO

La funzione leggiDirectory() viene chiamata alla fine dello script e l'array ricevuto in risposta viene visualizzato sullo schermo, a titolo di esempio, in due modalità:

1 utilizzando l'apposita funzione print_r();
2 analizzando con il comando foreach i singoli elementi dell'array, uno alla volta, e presentandoli incolonnati sotto forma di collegamento iperlink.

```
43    // leggo i file presenti nella directory
44    $fileContenuti = leggiDirectory($percorso);
45    // stampo direttamente il contenuto dell'array
46    echo "<strong>contenuto dell'array \$fileContenuti</strong><br>";
47    print_r($fileContenuti);
48    echo "<br><br><strong>\nFile presenti nella directory: [$percorso] </strong><br><ul>";
49    // "cicla" tutti gli elementi dell'array, li visualizza e crea un link
50    foreach ($fileContenuti as $file) {
51      echo "<li><a href={$percorso}{$file}>{$file}</a>";
52    }
53    echo "</ul>\n";
54  ?>
```

Riportiamo come esempio la sequenza delle videate ottenute al caricamento del file pinguini.jpg:

Al termine dell'esecuzione dello script verifichiamo la presenza del file aprendo la directory direttamente dal desktop del nostro PC:

## METTITI ALLA PROVA

**Apri il file Download.php**

1 Il programma carica i file all'interno della sottodirectory ./programmi.
Indenta il codice e modificalo in modo tale che si possano caricare i file in due cartelle (programmi e lavori).

2 Inserisci un secondo contatore che conti i download effettuati dai file della cartella lavori dopo averne visualizzato il suo contenuto.

# ESERCIZI IN LABORATORIO 3

## Gestione file JSON

Data l'importanza del formato JSON descriviamo due script, rispettivamente, il primo per leggere un file e decodificare i dati presentati in tale formato e il secondo per predisporre e scrivere su file un insieme di dati in JSON.

### Lettura di un file JSON

I file JSON sappiamo essere dei "normali" file di testo che contengono dati formattati secondo una specifica sintassi: nella prima parte dello script apriamo in lettura un file memorizzato in uno specifico percorso (nel nostro caso il file regioni.json, che contiene il numero di provincie di alcune regioni italiane, presente nella directory dati) dando eventualmente un messaggio di errore se tale operazione non va a buon fine.

```
1  <!DOCTYPE html>
2  <html lang="it">
3  <head>
4  <title>Lettura file JSON</title>
5  </head>
6  <body>
7  <h2>Lettura file formato JSON</h2>
8  <?php
9    $nomefile = "../JSON/dati/regioni.json";   // nome del file dove sono memorizzati i dati
10   if (file_exists($nomefile)){               // verifica dell'esistenza del file
11     $idfile = fopen($nomefile, "r+");        // se esiste viene aperto in lettura
12     // se non si riesce ad aprirlo viene creato un messaggio di errore
13     if (!$idfile)
14       die ($msg = "il file $nomefile non e' stato aperto <BR>");
15     // se il file viene aperto si leggono 100 caratteri e messi in $regioni
16     $regioni = fread($idfile, 100);          // leggo tutti i caratteri
17     fclose($idfile);                         // chiusura file
18   }
19   else{
20     // se il file non esiste viene creato messaggio errore
21     die ($msg = "il file  $nomefile non e' presente <BR>");
22   }
```

Visualizziamo il file appena letto dapprima con una semplice operazione di echo (riga 26) e, quindi, convertendo il file letto in un oggetto mediante la funzione json_decode() (riga 28) appositamente predisposta a tal scopo dal linguaggio PHP e, quindi, lo visualizziamo con la specifica funzione var_dump():

```
23   // visualizzazione dei dati e loro analisi
24   echo ("<p> Lettura file  $nomefile OK </p>");
25   echo ("<h4><p>a) visualizzazione come file di testo:</p></h4>");
26   echo ($regioni);
27   // lo trasformiamo in un oggetto
28   $oggetto = json_decode($regioni);
29   echo "<h4><p>b) visualizzazione con la funzione var_dump()</p></h4>";
30   var_dump($oggetto);
```

Come verifica/esempio del funzionamento della funzione json_decode() leggiamo un singolo attributo dell'oggetto utilizzando la sintassi specifica "->"(riga 34):

```
32   // analisi oggetto
33   echo "<h4><p>c) analisi dell'oggetto: legge il singolo dato con chiave Veneto :</h4>";
34   echo $oggetto->{'Veneto'};
35   echo "</p>";
36   ?>
```

Un'esecuzione del programma è riportata nella pagina seguente, dove sono state evidenziate le tre attività svolte dallo script.

# 3 ESERCIZI IN LABORATORIO

## Scrittura di un file JSON

Effettuiamo ora l'operazione inversa, cioè quella di predisporre un insieme di dati in formato JSON e di scriverli in un file: predisponiamo un'apposita funzione scriviFile() che riceve come parametri il nome del file da scrivere compreso di percorso relativo, e la variabile che contiene il dato da scrivere:

```
codiceJSON.php
1
2  <?php
3    function scriviFile($nomefile, $dati){
4      $fp = fopen($nomefile, 'w');
5      fwrite($fp, json_encode($dati));
6      fclose($fp);
7    }
8  ?>
```

Lo script propone tre possibili alternative per la preparazione dei dati da scrivere.

**1** La prima possibilità è quella di predisporre i dati in un array e di convertirli in formato JSON utilizzando la funzione json_encode() (riga 21) appositamente predisposta a tale scopo dal linguaggio PHP:

```
10  <!DOCTYPE html>
11  <html lang="it">
12  <head>
13  <title>Esempio di codifica JSON</title>
14  </head>
15  <body>
16  <h2>Esempio di codifica JSON</h2>
17  <?php
18    // prima modalita' di realizzazione file JSON
19    echo "<h4><p>a) visualizzo un array convertito in JSON:<br></h4>";
20    $linguaggi = array("PHP", "Java", "JS", "HTML", "Python", "VisualBasic");
21    $datiJSON = json_encode($linguaggi);
22    echo $datiJSON;
23    echo "<br>";
24    // scrittura dei file
25    $nomefile = "../JSON/dati/linguaggi.json";   // nome file destinazione
26    scriviFile($nomefile,$datiJSON);
```

**2** La seconda possibilità è quella di predisporre i dati in un oggetto: creiamo quindi una classe (righe 30÷37);

```
29    // seconda modalita' di realizzazione file JSON
30    echo "<h4><p>b) visualizzo un oggetto convertito in JSON:<br></h4>";
31    class dati {
32      public $java = "JAVA";
33      public $php  = "PHP";
34      public $cpp  = "C++";
35      public $js = "Javascript";
36      public $VB = "VisualBasic";
37      public $data = "";
38    }
```

# ESERCIZI IN LABORATORIO 3

ne istanziamo un oggetto (riga 38) e lo convertiamo in formato JSON sempre utilizzando la funzione json_encode() prima descritta; dopo averlo visualizzato, lo scriviamo in un file sempre utilizzando la funzione scriviFile() (riga 45):

```
38    $linguaggi = new dati();                    // creazione di un oggetto
39    $linguaggi->data = new DateTime();          // modifica di un attributo
40    $datiJSON = json_encode($linguaggi);
41    echo $datiJSON;
42    echo "<br>";
43    // scrittura del file
44    $nomefile = "../JSON/dati/linguaggi2.json";   // nome file destinazione
45    scriviFile($nomefile,$datiJSON);
```

**3** La terza possibilità è quella di leggere una tabella di un database, convertirla in formato JSON e quindi scriverla nella nuova formattazione: per la tabella amici del database provephp utilizziamo il meccanismo offerto da PHP Data Objects (PDO) dapprima per leggere questa tabella (riga 55) e quindi per memorizzarla in una variabile $stmt (riga 57). Quindi facciamo scorrere i dati estratti dal database facendo corrispondere a ogni riga la variabile $riga grazie al metodo fetch() associato a PDO::FETCH_ASSOC che "ci dice" che stiamo creando un array associativo usando come chiavi i nomi dei campi (righe 58 e 59); lo convertiamo in formato JSON sempre utilizzando la funzione json_encode() e lo scriviamo in un file usando sempre la funzione scriviFile() (riga 65):

```
48    // terza modalità di realizzazione file JSON
49    echo "<h4><p>c) visualizzo una tabella di un database MySql convertita in JSON;<br></h4>";
50    $user = 'root';
51    $password = '';
52    $db = 'provephp';
53    $host = 'localhost';
54    $port = 3306;
55    $conn = new PDO("mysql:host=$host; dbname=$db; port=$port", $user, $password);
56    $datiJSON = [];
57    $stmt = $conn->query("SELECT * FROM amici");
58    while($riga = $stmt->fetch(PDO::FETCH_ASSOC)){
59      $datiJSON[] = $riga;
60    }
61    $datiJSON = json_encode($linguaggi);
62    echo $datiJSON;
63    // scrittura del file
64    $nomefile = "../JSON/dati/amici1.json";   // nome file destinazione
65    scriviFile($nomefile,$datiJSON);
66  ?>
```

In questa esecuzione del programma sono state evidenziate le tre codifiche JSON, prima di scriverle su file:

## METTITI ALLA PROVA

**AREA DIGITALE**

**Altri esercizi con i file**

1 Scrivi un programma che, dopo aver fatto selezionato una provincia tramite un combo box, memorizza in un file JSON i dati di tutti i comuni della provincia presenti nelle rispettive tabelle del database provaphp.

2 Scrivi un programma che, dopo aver fatto selezionato un argomento tramite un combo box, memorizza in un file JSON i dati dei libri in lingua italiana che trattano l'argomento desiderato, disponendoli in ordine alfabetico di autore.

# 4 ESERCIZI IN LABORATORIO

## AREA DIGITALE

**ESERCIZI IN LABORATORIO 5**
Connessione FTP con uno script PHP

**ESERCIZI IN LABORATORIO 6**
Leggere e scrivere file Excel e Word

**ESERCIZI IN LABORATORIO 7**
Inviare una mail con PHP connesso a MySQL

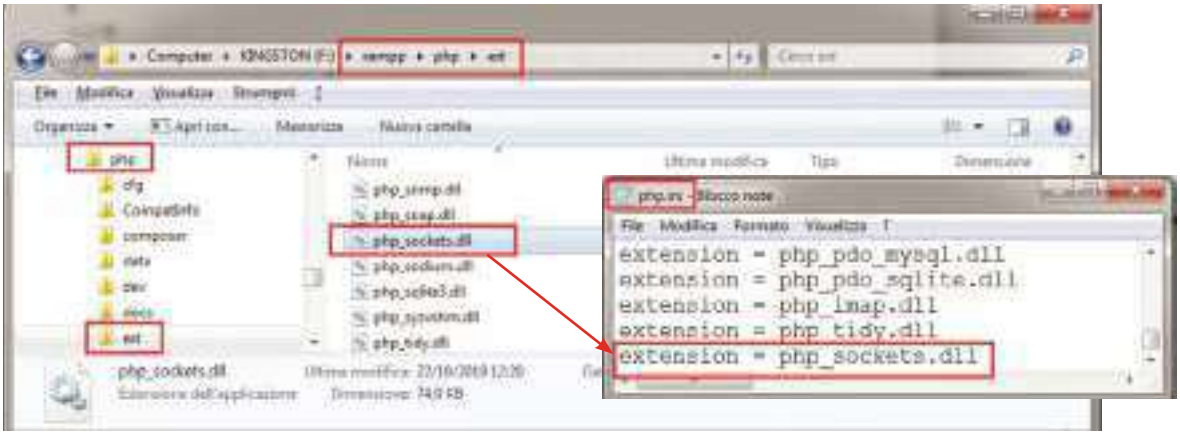
**ESERCIZI IN LABORATORIO 8**
Creare un file pdf con PHP

## CLIENT-SERVER IN PHP: INVIO DI MESSAGGI E FILE MEDIANTE SOCKET

### I socket in PHP

In questa esercitazione intendiamo utilizzare i socket per inviare un file di testo a un server. I socket rappresentano un valido meccanismo che consente di trasmettere e ricevere dati in una rete in uno script mediate i protocolli di trasmissione TCP e UDP. PHP ci mette a disposizione alcune API che permettono di implementare i socket nei nostri script per far comunicare programmi diversi in rete.

La tipica installazione XAMPP su Windows pone la libreria php_sockets.dll nella directory C:\xampp\php\ext ma, generalmente, deve essere abilitata nel file php.ini presente nella directory \xampp\php, aggiungendo la riga come nella figura:




Riportiamo alcune tra le principali funzioni presenti in PHP per creare, distruggere e gestire un socket:

- socket_accept: accetta una connessione su un socket;
- socket_bind: bind di un nome a un socket;
- socket_clear_error: azzera gli errori di un socket, oppure l'ultimo codice d'errore;
- socket_close: chiude una risorsa di tipo socket;
- socket_connect: inizia una connessione su un socket;
- socket_create_listen: apre un socket per accettare connessioni su una porta;
- socket_create: crea un socket (punto terminale di una comunicazione);
- socket_get_option: ottiene le opzioni per un socket;
- socket_getpeername: interroga il lato remoto di un dato socket per ottenere o la combinazione host/porta o un percorso Unix, in base al tipo di socket;
- socket_getsockname: Interroga il lato locale di un dato socket e restituisce o la combinazione host/porta o un percorso Unix in base al tipo di socket;
- socket_last_error: restituisce l'ultimo errore su un socket;
- socket_listen: attende una richiesta di connessione su un socket;
- socket_read: legge fino a un massimo di byte predefiniti da un socket;
- socket_recv: riceve i dati da un socket collegato;
- socket_recvfrom: riceve i dati da un socket, a prescindere che sia connesso o meno;

# ESERCIZI IN LABORATORIO 4

- socket_select: esegue la system call select() su un set di socket con un dato timeout;
- socket_send: invia i dati a un socket collegato;
- socket_sendto: invia un messaggio a un socket, a prescindere che sia connesso o meno;
- socket_set_block: attiva la modalità di "block" su un socket;
- socket_set_nonblock: attiva la modalità "nonblocking" per il descrittore di file (fd);
- socket_set_option: valorizza le opzioni per un socket;
- socket_shutdown: chiude un socket in ricezione, in invio o in entrambi i sensi;
- socket_strerror: restituisce una stringa con la descrizione dell'errore;
- socket_write: scrive su un socket.

Utilizzeremo quasi tutte queste funzioni nei due esempi che ci accingiamo a descrivere.

## Invio di messaggi

Il primo esempio mette in comunicazione tramite socket un client e un server: il primo invia un testo al secondo che lo elabora a glielo rimanda modificato: è composto da due script rispettivamente per gestire le fasi di client e server, che vengono memorizzati in due file con nomi proprio client1.php e server1.php.
Il server rimane in attesa di essere contattato (fase di listening o ascolto) e il client invece contatta il server (nel nostro esempio per inviargli un file): pertanto il server è sempre in ascolto su una determinata porta TCP e quando riceve una richiesta di connessione decide se accettarla o meno, a seconda del tipo di connessione predefinita nel programma.

### Lo script del server

Nelle prime istruzioni del server inizializziamo alcuni parametri per mantenere attivo lo script mentre è in attesa del client: l'istruzione di (riga 5), per esempio, disabilita il blocco sul tempo di esecuzione dello script, ponendo il **timeout** (W) a zero (tempo infinito).

**Timeout**
Il termine timeout Indica il tempo limite che i server assegnano per l'esecuzione di uno script. Tale valore cambia da server a server, in genere è attorno a 100/120 secondi.

```
server1.php    client1.php
1  <?php
2  // non terminare/uscire mentre attende una connessione
3  error_reporting(E_ALL);
4  // disabilita il blocco dello script
5  set_time_limit(0);
6  // abilita lo scarico dell'output
7  ob_implicit_flush();
```

Successivamente viene creato il socket con la funzione socket_create() (righe 12÷15), che ha la seguente sintassi:

```
socket_create (dominio, tipo, protocollo)
```

dove
- dominio indica la famiglia di protocolli da usare nella comunicazione tra socket (AF_INET è il protocollo Internet basato su IPv4 con protocolli TCP e UDP);
- tipo indica invece il tipo di comunicazione da usare con il socket (SOCK_STREAM fornisce una connessione sequenziale, full-duplex);
- protocollo indica il protocollo usato nella comunicazione (SOL_TCP indica il protocollo TCP, cioè un protocollo affidabile, basato sulla connessione, orientato al flusso, full-duplex. Il TCP garantisce che tutti i pacchetti siano ricevuti nel medesimo ordine in cui siano stati inviati. Se un pacchetto viene perso durante la trasmissione, il TCP provvederà automaticamente alla ritrasmissione fino a quando l'host remoto non conferma la ricezione dello stesso).

# 4 ESERCIZI IN LABORATORIO

La funzione socket_set_option() (righe 18÷20) viene utilizzata per configurare alcune opzioni del socket, come per esempio il timeout in lettura o scrittura, il riutilizzo dell'indirizzo.

```
9   echo "<h2>Server - connessione TCP/IP </h2><HR>\n";
10  // creo il socket con 3 parametri:
11  // famiglia del protocollo, tipo di comunicazione, protocollo
12  if(!($socket=@socket_create(AF_INET, SOCK_STREAM, SOL_TCP))){
13    echo socket_strerror(socket_last_error($socket));
14    exit;
15  }
16  echo "a) socket creato<BR>";
17  // configuro socket, timeout in lettura o scrittura e riutilizzo indirizzo
18  if(!(@socket_set_option($socket, SOL_SOCKET, SO_REUSEADDR,1))){
19    echo socket_strerror(socket_last_error($socket));
20    exit;
21  }
```

Le funzioni che gestiscono i socket possono restituire messaggi di errore non attesi: l'uso del simbolo chiocciola (@) posizionato prima della funzione consente di intercettarlo e gestirlo grazie alla funzione socket_last_error().
Per esempio nella riga 13 viene stampato il codice di errore che si è manifestato con la funzione socket_strerror() che lo trasforma in una stringa descrittiva.

Definiamo i valori dell'indirizzo IP e la porta TCP del server (righe 24 e 25) e li associamo al socket (righe 27÷29): questa operazione viene anche chiamata binded è ed eseguita dalla funzione socket_bind().
Successivamente indichiamo al socket di mettersi in ascolto attraverso la funzione socket_listen() (righe 33÷35).

```
23  // definiamo indirizzo IP e porta TCP del server
24  $indirizzo = '127.0.0.1';
25  $porta = 10000;
26  // associo l'IP address e il port al socket
27  if(!(@socket_bind($socket,$indirizzo,$porta))){
28    echo socket_strerror(socket_last_error($socket));
29    exit;
30  }
31  echo "b) in attesa di un client<BR>";
32  // inizio ascolto sulla connessione
33  if(!(@socket_listen($socket)) ){
34    echo socket_strerror(socket_last_error($socket));
35    exit;
36  }
```

Ora il server rimane in attesa di una connessione di un client: innanzitutto viene attivata una funzione che blocca l'esecuzione dello script in attesa che arrivi un messaggio da un client, si tratta di socket_accept() (righe 39÷41).

```
38  // accetta la richiesta di connessione client e crea un apposito socket
39  $socketOK = socket_accept($socket)
40    or die("fallito inizializzazione attesa connessione con client \n");
41  echo "c) effettuata connessione al client <BR>";
42
43  // lettura input inviato dal client
44  $ricevuto = socket_read($socketOK, 1024)
45    or die("errore in lettura sul socket di input\n");
46  echo "d) messaggio ricevuto dal client : ".$ricevuto."<BR>";
```

Dopo aver accettato la comunicazione, creando un apposito oggetto $socketOK su cui viene spostata la comunicazione, lo script legge il messaggio inviato dal client e lo visualizza sullo schermo.

# ESERCIZI IN LABORATORIO 4

Ora il messaggio viene elaborato, per esempio lo si converte in maiuscolo, e lo si scrive nel socket come messaggio di risposta al client (righe 54÷56).

```
48  // elaborazione del testo ricevuto
49  $ricevuto = trim($ricevuto);
50  $daInviare =strtoupper($ricevuto) . "\n";  // trasforma in maiuscolo
51
52  // inoltro risposta al client
53  echo "e) risposta al client <BR>";
54  if(!(socket_write($socketOK, $daInviare, strlen ($daInviare)))){
55    echo socket_strerror(socket_last_error($socket));
56    exit;
57  }
```

Infine avviene la chiusura dei due socket (righe 61 e 62) e termina lo script.

```
59  // chiusura dei sockets
60  echo "f) chiusura socket <BR>";
61  socket_close($socketOK);
62  socket_close($socket);
63  echo "<HR>Termine operazioni del server<BR>";
64  ?>
```

## Lo script del client

Lo script del client, dopo aver effettuato le medesime inizializzazioni del server, definisce le variabili che individuano il socket del server e predispone il testo da inoltrare.

```
9   echo "<h2>Client - connessione TCP/IP </h2><HR>\n";
10  // settaggio variabili client
11  $host = '127.0.0.1';
12  $porta = 10000;
13  $testo = "testo scambiato client server";
14  echo "messaggio per il Server :".$testo."<BR>";
```

Viene quindi creato il socket (riga 18) e, di seguito, avviene la connessione con il socket (righe 22÷24).

```
16  // creazione del socket
17  echo "a) socket creato<BR>";
18  $socket = socket_create(AF_INET, SOCK_STREAM, 0)
19    or die("socket non creato\n");
20  // connetti(t) al server; usiamo l'IP address e il port al socket
21  echo "b) connessione con socket del server <BR>";
22  if(!(@socket_connect($socket, $host, $porta))){
23    echo socket_strerror(socket_last_error($socket));
24    exit;
25  }
```

Poi il testo viene inviato al socket con l'istruzione di riga 29 e si pone in attesa della risposta da parte del server (righe 34÷36). Il testo ricevuto viene memorizzato nella variabile $risposta e visualizzata sullo schermo (riga 38).

```
27  // invio stringa al server
28  echo "c) attesa risposta del server <BR>";
29  if(!(@socket_write($socket, $testo, strlen($testo)))){
30    echo socket_strerror(socket_last_error($socket));
31    exit;
32  }
33  echo "d) lettura risposta del server <BR>";
34  if(!($risposta =@socket_read ($socket, 1024)))){
35    echo socket_strerror(socket_last_error($socket));
36    exit;
37  }
38  echo "e) risposta del server :".$risposta."<BR>";
```

# 4 ESERCIZI IN LABORATORIO

Infine vengono eseguite le istruzioni di chiusura:

```
40  // chiudi socket
41  echo "f) chiusura socket <BR>";
42  socket_close($socket);
43  echo "<HR>Termine operazioni del client<BR>";
44  ?>
```

L'esecuzione dei due script mostra come viene cambiato il messaggio tra client e server che, in questo esempio, viene semplicemente trasformato in maiuscolo.

## Invio di file

Il secondo esempio permette a un client di trasferire un file al server in blocchi di 1024 byte alla volta: è composto da due script rispettivamente per gestire le fasi di client e server, in parte uguali ai precedenti, memorizzati in due file client2.php e server2.php.

### Lo script del server

La logica di funzionamento del server è assai semplice: come per l'esempio precedente, inizialmente viene settato il server e si predispone un file in scrittura dove verrà memorizzare il file trasmesso da client.

```
server2.php   config.php
1   <?php
2   ob_end_clean();          // svuota il buffer di output
3   set_time_limit(0);       // disabilita il blocco dello script
4   echo "<b>Esecuzione socket server ricezione file</B><HR>";
5
6   // apro il file da creare in scrittura
7   if(!($file=@fopen("./dati/fileViaSocket.txt_ricevuto","w"))){
8     echo "Errore di apertura file.<BR>";
9     exit;
10  }
11  else
12    echo "Ho aperto il file dove scrivere quello che invierà il client<BR>";
```

Quindi viene creato il socket e viene configurato con i parametri iniziali.

```
14  // creo il socket con 3 parametri
15  if(!($socket=@socket_create(AF_INET, SOCK_STREAM, SOL_TCP))){
16    echo socket_strerror(socket_last_error($socket));
17    exit;
18  }
19  // configurare socket, timeout in lettura e scrittura e riutilizzo indirizzo
20  if(!(@socket_set_option($socket, SOL_SOCKET, SO_REUSEADDR,1))){
21    echo socket_strerror(socket_last_error($socket));
22    exit;
23  }
```

# ESERCIZI IN LABORATORIO 4

Successivamente si imposta l'indirizzo IP e la porta TCP del server (righe 26 e 27) e si effettua il binding con la funzione socket_bind().

```
25  // definizione indirizzo IP e porta TCP del server
26  $indirizzo = '127.0.0.1';
27  $porta = 10000;
28  // associo l'IP address e il port al socket
29  if(!(@socket_bind($socket,$indirizzo,$porta))){
30    echo socket_strerror(socket_last_error($socket));
31    exit;
32  }
33  echo "CREO IL SOCKET<BR>";
```

Ora il server si pone in ascolto sul socket attraverso la funzione socket_listen() (righe 31÷33).

```
35  // il server è in listening (ascolto) sul socket
36  if(!(@socket_listen($socket))){
37    echo socket_strerror(socket_last_error($socket));
38    exit;
39  }
40  echo "SONO IN ASCOLTO SU PORT 10000<BR>";
```

Alla ricezione della richiesta del client, questa viene accolta con la funzione socket_accept(): con la funzione socket_getpeername() si individua l'indirizzo del client che si è connesso (righe 45÷47).

```
42  // controllo che arrivi una richiesta del client (funzione bloccante)
43  if(!($client=@socket_accept($socket))){
44    echo socket_strerror(socket_last_error($socket));
45    exit;
46  }
47  // get_peername restituisce l'indirizzo del client che si è connesso
48  if(!(@socket_getpeername($client,$ind))){
49    echo socket_strerror(socket_last_error($socket));
50    exit;
51  }
```

Dopo aver mostrato a video l'indirizzo IP del client che si è connesso (riga 53), lo script entra in un ciclo che legge i dati dal socket fino a quando non incontrerà la stringa di fine invio ('//FINEFILE//').
Siccome è in invio, il client scrive sul socket pacchetti di 1024 byte alla volta, questi vengono letti dal server (righe 58÷60) sempre in blocchi di 1024 byte e memorizzati nel file predisposto (righe 63÷65).

```
53  echo "SI E' CONNESSO IL CLIENT:".$ind."<BR>";
54  // ciclo di lettura dal socket e scrittura nel file
55  // termina quando riceve messaggio di fine trasmissione
56  do{
57    // leggo dal socket 1024 Byte
58    if(!($ricevo=@socket_read($client, 1024, PHP_BINARY_READ))){
59      echo socket_strerror(socket_last_error($socket));
60      exit;
61    }
62    // scrivo nel file i 1024 byte ricevuti dal socket
63    if(!(@fwrite($file,$ricevo,1024))){
64      echo socket_strerror(socket_last_error($socket));
65      exit;
66    }
67  } while($ricevo!="//FINEFILE//");
```

# 4 ESERCIZI IN LABORATORIO

Terminata la ricezione dei dati, il server scrive nel socket un messaggio di cortesia in risposta al client (righe 70÷72),

```
69  // restituzione messaggio di conferma al client
70  if(!(@socket_write($client,"FILE RICEVUTO CORRETTAMENTE!"))){
71    echo socket_strerror(socket_last_error($socket));
72  }
```

quindi chiude il socket del client

```
74  echo "HO RICEVUTO IL FILE<BR>";
75  socket_close($client);
76  echo "<HR>Termine comunicazione<BR>";
```

e, infine, chiude il file e il socket del server, terminando lo script.

```
78  // chiusura socket e file
79  fclose($file);
80  socket_close($socket);
81  ?>
```

## Lo script del client

Anche lo script del client è molto simile a quello precedente: dopo le inizializzazioni viene aperto in lettura il file che si desidera inoltrare:

```
1   <?php
2   // Disabilita il blocco dello script
3   set_time_limit(0);
4   echo "<B>Esecuzione socket client invio file</B><HR>";
5   // Apro il file da inviare e controllo se esiste
6   if(!($file=@fopen("./dati/fileViaSocket.txt","r"))){
7     echo "ERRORE DI APERTURA FILE<BR>";
8     exit;
9   }
10  else
11    echo "APERTURA FILE fileViaSocket.txt<BR>";
```

Viene quindi creato il socket e viene richiesta la connessione al server.

```
13  // definisco indirizzo IP e port TCP del server
14  $indirizzo='127.0.0.1';
15  $porta=10000;
16
17  // creo il socket con 3 parametri
18  // famiglia del protocollo, tipo di comunicazione, protocollo per quel dominio
19  if(!($socket=@socket_create(AF_INET, SOCK_STREAM, SOL_TCP))){
20    echo socket_strerror(socket_last_error($socket));
21    exit;
22  }
23  // connessione al server e verifica errore eventuale
24  if(!($conn=@socket_connect($socket,$indirizzo,$porta))){
25    echo socket_strerror(socket_last_error($socket));
26    exit;
27  }
```

Lo script del client inizia con la trasmissione del file a blocchi di 1024 byte finché il file non termina e, in questa situazione, si invia il messaggio che segnala la fine della trasmissione.

# ESERCIZI IN LABORATORIO 4

```
30  echo "CARICO IL FILE SUL SERVER<BR>";
31  // ciclo che carica il file sul server 1KB per volta
32  do{
33    // verifico se il file è finito
34    if(feof($file)){
35        echo "INVIO COMPLETATO...<BR>";
36        // invio al server di un messaggio speciale che indica la fine dell'invio
37        socket_write($socket,"//FINEFILE//");
38        //Necessario per uscire dal ciclo
39        break;
40    }
41    // leggo 1024Byte dal file con controllo eventuale errore
42    if(!($blocco=@fread($file,1024))){
43        echo socket_strerror(socket_last_error($socket));
44        exit;
45    }
46    // scrivo sul socket quanto letto dal file con controllo errore
47    if(!(@socket_write($socket,$blocco,1024))){
48        echo socket_strerror(socket_last_error($socket));
49        exit;
50    }
51  }while(true);
```

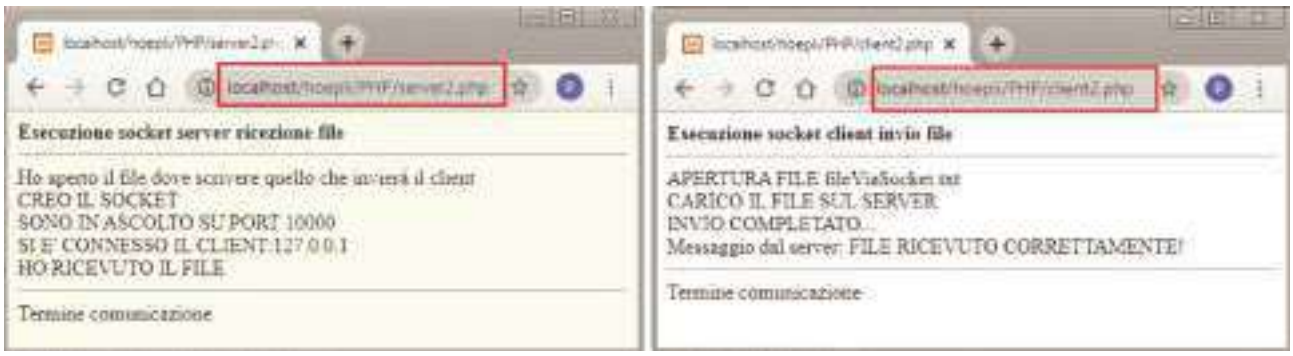
Il client si mette ora in attesa della risposta del server che gli conferma *"il buon esito dell'inoltro"*:

```
53  // leggo dato restituito dal server per verifica corretto invio
54  if(!($ricevuto=@socket_read($socket,1024))){
55      echo socket_strerror(socket_last_error($socket));
56      exit;
57  }
```

dopo averla visualizzata sullo schermo si procede con la chiusura del socket e alla terminazione dello script.

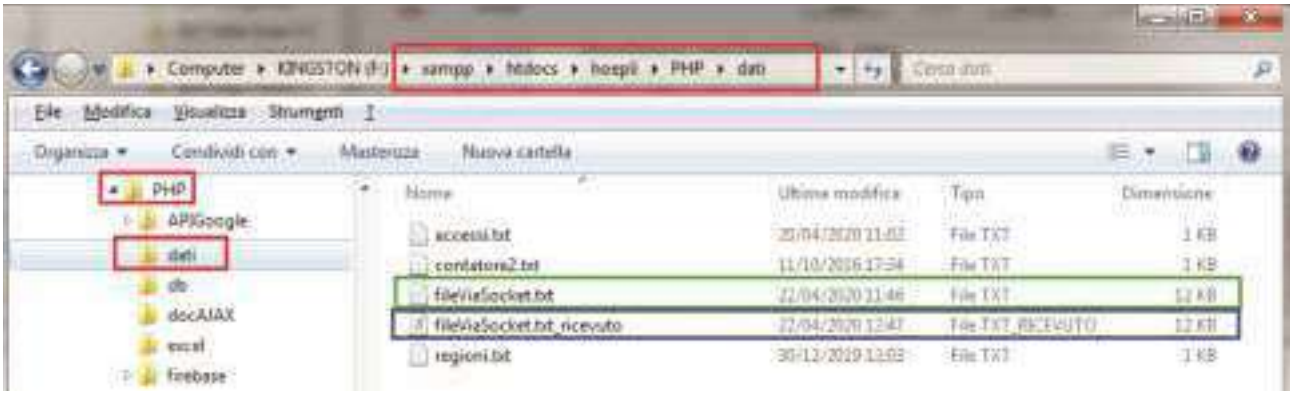
```
58  echo "Messaggio dal server: ".$ricevuto."<BR>";
59  fclose($file);
60  socket_close($socket);
61  echo "<H1>Termine comunicazione<BR>";
62  ?>
```

L'esecuzione dello script mostra l'echo delle operazioni avvenute tra client e server:




Possiamo verificare come viene caricato un file dal client al server: per comodità, nel nostro esempio, il client inoltra il file fileViaSocket.txt, presente nella sottodirectory /dati della document root del client, che viene memorizzato in quella del server (che è la stessa dato che client e server sono in esecuzione sulla stessa macchina 127.0.0.1), dopo averlo opportunamente da lui rinominato.

# 4 ESERCIZI IN LABORATORIO 4




## METTITI ALLA PROVA

→ • Utilizzare i socket in PHP

**Apri i file client2.php e server2.php**

1 Modifica il codice dei due script in modo da far scegliere all'utente il file da caricare con un form, utilizzando la tecnica del PostBack, l'indirizzo del server e il port TCP.

2 Effettua una connessione tra due computer, in cui uno fa da server e l'altro da client per realizzare l'invio del file.

3 Scrivi un'applicazione client-server in PHP che utilizzi i socket per simulare un convertitore di valuta:
- il client invia al server le due valute desiderate (sigla internazionale) e l'importo da convertire;
- il server gli risponde:
  - l'importo convertito nella valuta richiesta;
  - un segnale di errore nel caso di una sigla valuta errata o mancante.

4 Scrivi un'applicazione client-server in PHP che utilizzi i socket per simulare un cifratore di testi usando un metodo di Cesare e uno polialfabetico:
- il client invia al server un file di testo, un numero intero che indica lo spiazzamento e una parola chiave utilizzata per la cifratura;
- il server gli risponde con due file contenenti testi cifrati.

# Applicazioni lato server con codice separato: CGI e servlet

UNITÀ 4

**LEZIONE 1**
Le applicazioni lato server

**Lezione 2**
Servlet e database

**Lezione 3**
Servlet e database embedded

**Esercizi in Laboratorio**

- L1 Applicazioni CGI
- L2 XAMPP e il server Engine Tomcat
- L3 L'inizializzazione delle servlet
- L4 L'interazione get/post tra client e servlet
- L5 L'interazione con client AJAX e lo scambio dati in formato JSON
- L6 La permanenza dei dati con le servlet: i cookie
- L7 La permanenza dei dati con le servlet: le sessioni
- L8 JDBC E MySQL
- L9 Servlet e database MDB con parametri
- L10 Servlet con JavaDB Derby


## CONOSCENZE

- Acquisire le caratteristiche delle servlet
- Conoscere il ciclo di vita di una servlet
- Conoscere le caratteristiche di web.xml
- Acquisire le caratteristiche dell'interfaccia JDBC
- Conoscere i tipi di driver per la connessione ai database

## COMPETENZE

- Installare e utilizzare XAMPP
- Realizzare un'applicazione Web
- Riconoscere i componenti di una pagina lato server
- Ricevere e analizzare dati in formato JSON con JavaScript
- Generare un file .WAR

## ABILITÀ

- Realizzare un'applicazione WEB dinamica con CGI
- Realizzare un'applicazione WEB dinamica con servlet
- Utilizzare le servlet in un'applicazione AJAX/JSON
- Utilizzare cookie e sessioni con le servlet
- Connettere le applicazioni Web con MySQL e Access
- Scrivere, installare e configurare una servlet

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

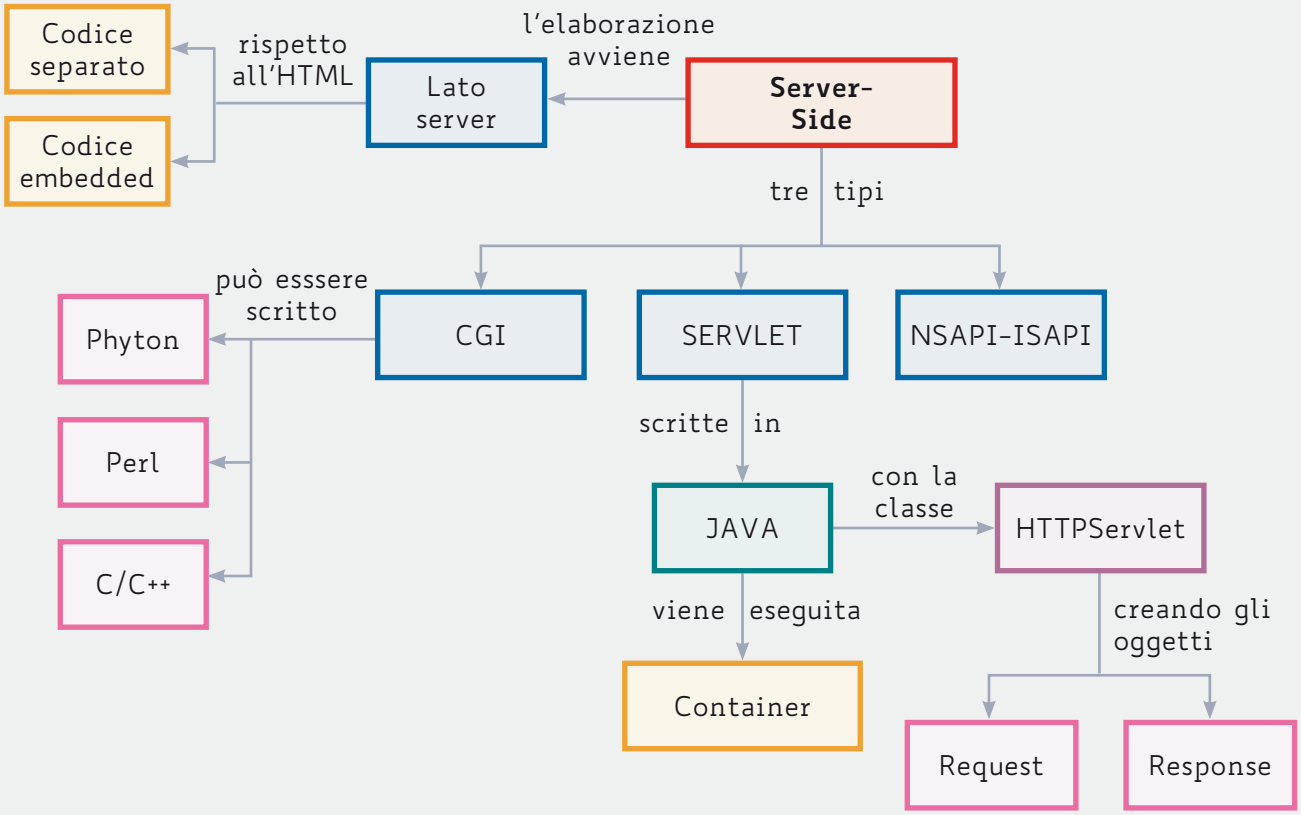
# 1 Le applicazioni lato server

**IN QUESTA LEZIONE IMPAREREMO...**

- le caratteristiche di servlet e CGI
- il ciclo di vita di una servlet
- a scrivere, installare e configurare una servlet

## MAPPA CONCETTUALE

didattica inclusiva




## La programmazione server-side

Con il termine programmazione server-side viene indicato il meccanismo mediante il quale tutto o parte del documento richiesto dal client, e atteso in formato HTML, viene generato in seguito a un'elaborazione che viene eseguita sul server, cioè il processo Web server genera dinamicamente la risposta in funzione della richiesta e dei parametri che vengono inclusi con essa.

ESEMPIO

Per esempio, se il client chiede gli orari dei treni per una certa destinazione e per una certa data la risposta del server è naturalmente sempre diversa e, di conseguenza, la pagina che verrà generata e inviata al client dovrà essere elaborata volta per volta e generata per ogni caso specifico.

Il Web server deve essere opportunamente configurato e strutturato per mandare in esecuzione uno specifico programma in modo che generi il contenuto "dinamico" al momento della richiesta: questa elaborazione che avviene sul server prende il nome di elaborazione server-side e la programmazione di componenti software che vengono elaborati sul server, di conseguenza, viene chiamata programmazione server-side.
Oggi sono disponibili diverse tecniche di programmazione server-side che si differenziano per:
- i linguaggi di programmazione supportati;
- i Web server supportati;
- i meccanismi di "aggancio" al Web server e il loro impatto sulle prestazioni;
- il particolare ambito applicativo per il quale sono concepite.

Una prima classificazione può essere fatta in base alla "relazione" che questi programmi hanno con l'HTML, e cioè:
- codice separato, associato a una URL:
  - Common Gateway Interface (CGI),
  - Java servlet,
  - NSAPI (o server API), ASAPI e ISAPI;
- codice embedded in HTML:
  - Server Side Includes (SSI)-Apache,
  - Active Sever Pages (ASP)-Microsoft,
  - PHP (acronimo ricorsivo di "PHP: Hypertext Preprocessor"),
  - Java Server Pages (JSP).

In HTML abbiamo a disposizione il meccanismo dei form che consente di inviare informazioni da un Web browser client-side a un programma in esecuzione server-side e, anche se "rudimentale, datato e obsoleto", è lo strumento essenziale per trasformare un Web browser in un'interfaccia grafica universale; in HTML possiamo definire dei componenti interattivi (text box, combo-box, check-box ecc.) che alla pressione di un pulsante di submit (generalmente Conferma) effettuano l'invio dei dati inseriti come parametri da far pervenire al server per eseguire le dovute interrogazioni/elaborazioni.

ESEMPIO

Con il metodo **get** i dati vengono inviati attraverso la URL mediante una query string dove i parametri sono accodati con il carattere "?" e separati tra loro dal carattere "&".

http://localhost/cgi-bin/prova1.exe?cognome=Rossi&nome=Mario

In modo leggermente diverso si ottiene lo stesso scopo con il metodo **post**, ma facendo in modo che i dati non siano visibili a tutti.

## Modello a codice separato

Nei modelli a codice separato abbiamo tre possibili alternative: CGI, Java servlet e NSAPI.

### Common Gateway Interface (CGI)

Una prima tecnica per realizzare questa modalità operativa è quella di invocare procedure remote indicando, cioè, l'indirizzo di un programma Common Gateway Interface (CGI) che viene eseguito sul server e al quale vengono passati i dati letti nel form.

Questa tecnica era già presente fino da Http v1.0 ed è uno standard per interfacciare applicazioni esterne con Web server.

Le applicazioni che usano questo standard prendono il nome di programmi CGI e sono scritte in un qualunque linguaggio di programmazione che può leggere da standard input e scrivere su standard output, come Python, Perl, C, C++ ecc.
Il modello di interazione CGI è il seguente:

Un programma CGI viene eseguito dinamicamente in risposta alla chiamata e produce un output che costituisce la risposta alla richiesta Http (informazione dinamica).
Le operazioni si svolgono nel seguente ordine:

- in una pagina HTML il client invia al server la richiesta di eseguire un programma CGI con alcuni parametri e dati in ingresso, utilizzando il protocollo Http;
- il server, attraverso l'interfaccia standard CGI, chiama il programma indicato dal client, passandogli i parametri letti dal form;
- il programma CGI esegue le operazioni opportune e comunica al server i dati elaborati in una pagina HTML, sempre utilizzando l'interfaccia CGI;
- il server invia al client i dati elaborati dal programma CGI tramite protocollo Http.

Per comunicare con l'utente, il programma CGI produce una pagina HTML e la passa al Web server che la invia all'utente, in modo che possa essere visualizzata dal suo browser.

## Servlet

Un'alternativa al modello CGI può essere il linguaggio Java mediante le servlet, che sono classi particolari con molte affinità alle "deprecate" applet.
La principale differenza fra servlet e CGI è che gli script CGI sono eseguiti dal sistema operativo e, quindi, sono potenzialmente meno portabili delle servlet che, essendo scritte in Java, vengono eseguite dalla JVM integrata nel Web server.

Tutti i moderni Web server hanno integrata la JVM e, quindi, le servlet sono di fatto portabili su tutte le macchine.

Una seconda differenza è legata all'efficienza: gli script CGI vengono caricati ed eseguiti una volta per ogni richiesta, quindi richiedendo sempre un notevole tempo di latenza, mentre le servlet vengono caricate solo una volta e viene creato un thread per ogni richiesta (una servlet una volta caricata rimane in memoria e può ottimizzare l'accesso alle risorse attraverso caching, pooling ecc.).
A vantaggio degli script CGI c'è che danno la possibilità di utilizzare pressoché qualsiasi linguaggio: quindi il programmatore può scegliere volta per volta il linguaggio più adatto alla particolare applicazione che deve realizzare, mentre le servlet sono classi Java e, di conseguenza, devono essere scritte in Java, con tutti i vantaggi (platform independence, OO, GC ecc.) e gli svantaggi (bytecode interpretato) intrinseci di questo linguaggio.
Le servlet possono essere utilizzate qualunque sia il servizio espletato dal server - ovvero qualunque sia il protocollo di interazione client-server (per esempio sia Http che FTP) - e, facendo parte integrante dello stesso processo del Web server, si può immaginare che sia una procedura dello stesso server; questo perché il Web server, dopo aver effettuato un caricamento dinamico della classe Servlet, a seguito della prima richiesta alla servlet da parte di un client, la mantiene residente in memoria a disposizione delle successive richieste.

## NSAPI e ASAPI

Il modello Netscape Server Application Programming Interface (NSAPI) è stato inizialmente sviluppato da Rob McCool per Netscape per l'utilizzo in Netscape Enterprise Server: non esiste uno standard formale e, quindi, le applicazioni che utilizzano NSAPI possono non essere portabili in tutti i Web server.

A partire dal 2007 sono stati inseriti in Sun Java System Web Server e Zeus Web Server alcune opzioni per supportare le NSAPI e anche Apache Software Foundation ha sviluppato Apache API, anche chiamato ASAPI (Apache Server API).
Anche Microsoft nel suo Internet Information Server (IIS) ha introdotto un meccanismo equivalente a NSAPI chiamato IIS Server API, o ISAPI: esso è rappresentato da una DLL (Dynamic Link Library) che deve esporre due funzioni:

- `int getExtensionVersion(…)` // carica la libreria in memoria
- `int httpExtensionProc(…)` // genera un thread per gestire le richieste

Essendo legate al tipo di server e non essendo standard esulano dalla nostra trattazione.

## Struttura di una servlet

La servlet è una classe "speciale", così definita direttamente dalla Sun:

**SERVLET**
Una servlet è un componente software scritto in Java, gestito da un "container", che produce contenuto Web dinamico (Java Servlet Specification, v. 2.4).

Ogni istanza di una servlet è un oggetto Java che viene caricato ed eseguito dal Web server all'interno del processo di richiesta/risposta di servizi. Il suo controllo viene effettuato da un'altra applicazione Java, il container, che regola il ciclo di vita della servlet stessa. In realtà è il Web server stesso che svolge la funzione di container occupandosi della gestione del ciclo di vita delle servlet, passandogli i dati del client e restituendo ai client i dati prodotti dall'esecuzione delle servlet.

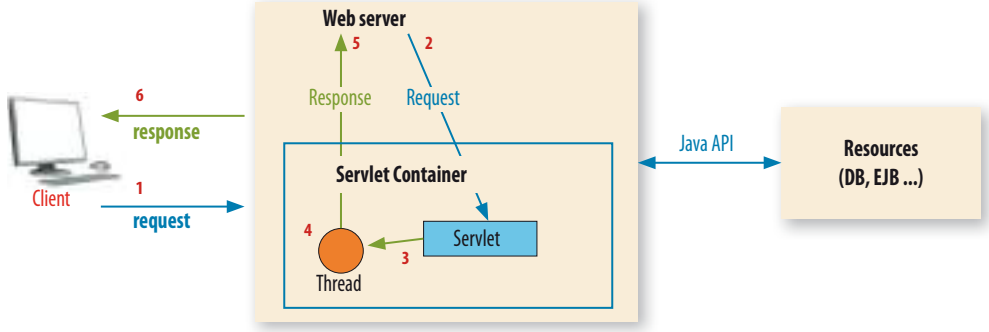
Dato che la servlet viene caricata in memoria alla sua prima richiesta di esecuzione, le inizializzazioni necessarie vengono eseguite una volta sola e consentono una più semplice condivisione dei dati fra le istanze di una servlet.

### La gestione delle richieste del client: il Web container

L'ambiente di esecuzione di una servlet si è detto che prende il nome di container e, come vedremo, sarà anche l'ambiente di esecuzione delle JSP (Java Server Page) che tratteremo nelle prossime lezioni.
Il container gestisce completamente la servlet durante tutto il suo ciclo di vita svincolando il programmatore dalla loro gestione: gestisce la comunicazione con i client, la sicurezza, il multithreading oltre alla sua attivazione e alla sua terminazione.

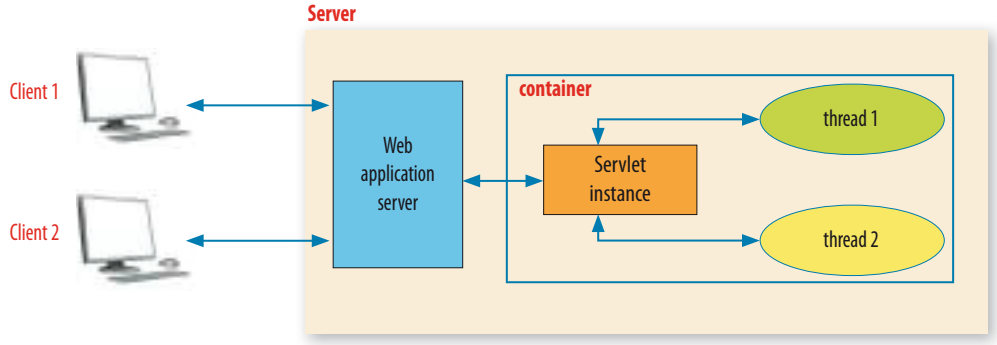
Questo fa sì che lo sviluppatore si possa concentrare esclusivamente sullo sviluppo della logica dell'applicazione.

Una servlet interagisce con un Web client attraverso il paradigma di comunicazione request/response schematizzato nella figura seguente:



Descriviamo sinteticamente il flusso di esecuzione:
**1** un client invia una richiesta (request) per una servlet a un Web application server;
**2** se si tratta di una prima richiesta allora il server istanzia e carica la servlet;
**3** si attiva un thread che gestisce la comunicazione tra il client e la servlet stessa;
**4** il server invia al thread-servlet la richiesta pervenutagli dal client;
**5** il thread-servlet costruisce e imposta la risposta (response) e la inoltra al server;
**6** il server invia la risposta al client.

Nel caso si verifichi una richiesta contemporanea di più client vengono attivati nel servlet container più thread:



Generalmente le applicazioni Web, oltre a eseguire semplici algoritmi contenuti all'interno della servlet per generare la risposta al client, interagiscono con i Database oppure con altre servlet.

Il servlet container open source più diffuso e utilizzato è Tomcat, del "gruppo" Apache (del quale riportiamo il logo a lato).
Tomcat è un servlet container open source e free software interamente scritto in Java disponibile per diverse piattaforme che si integra con i più diffusi Web server, come Apache e IIS. È scaricabile direttamente da: http://tomcat.apache.org/: la sua installazione e configurazione sarà descritta nell'Esercizio in Laboratorio 2 a essa appositamente dedicata.

L'esecuzione delle servlet richiede inoltre che sulla macchina server sia installata una versione di Java (anche semplicemente la standard edition).

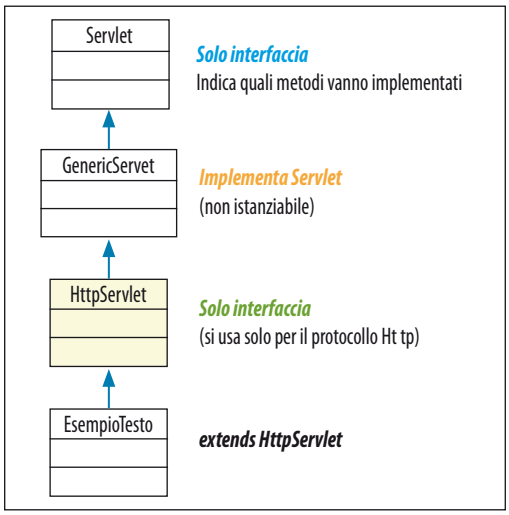
## Realizzazione di una servlet

Esistono diverse tipologie di servlet, che ereditano da un supertipo presente nel package javax.servlet: la loro estensione dipende dal protocollo di comunicazione utilizzato. Le servlet Http sono il tipo più comune di servlet e possono processare richieste (Httprequest) producendo risposte (Httpresponse).
Il codice con l'intestazione di una generica servlet ha la seguente struttura:

Quindi le servlet sono in relazione con classi e interfacce contenute nei package:

- `javax.servlet` // indipendenti dal protocollo di comunicazione
- `javax.servlet.http` // specifiche per il protocollo http

Il package javax.servlet è la base per le API Servlet e contiene la definizione dell'interfaccia servlet e le classi utili alla comunicazione fra client e server, a partire dai prototipi di tutti i metodi necessari alla gestione del ciclo di vita di una servlet e all'esecuzione delle operazioni implementate dalla servlet.
Il package javax.servlet.http fornisce classi che estendono le funzionalità di base di una servlet adottando le caratteristiche del protocollo Http come la gestione dei metodi GET e POST o degli header Http.
Tutte le servlet estendono HttpServlet o un'altra classe e sono quindi classi che implementano l'interfaccia servlet: la gerarchia completa è rappresentata nella seguente figura:

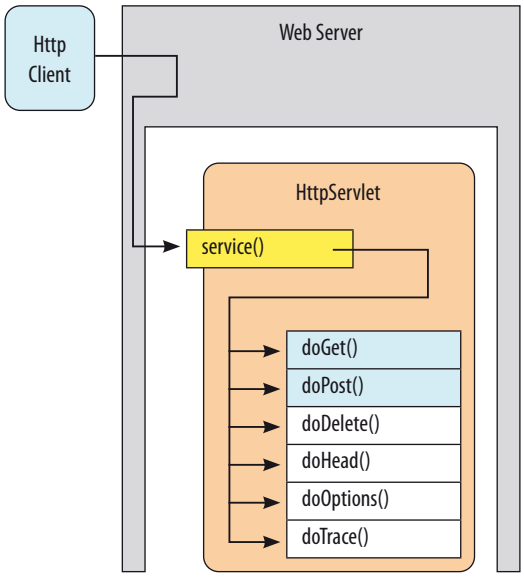



**AREA DIGITALE**
**Schema UML della gerarchia delle classi**

Naturalmente è necessario che la libreria javax (file javax.servlet.jar) sia installata sulla macchina e sia configurata correttamente la sua visibilità all'interno delle applicazioni Java: se non venisse automaticamente installata con Tomcat è possibile scaricarla dal sito www.hoepliscuola.it, nella cartella MATERIALI della sezione riservata a questo libro e copiarla nelle librerie di Java.

## La classe HttpServlet

La classe HttpServlet implementa service() e invoca un insieme di metodi, tra i quali quelli necessari per servire le richieste dal web. Il metodo service() viene invocato ogni volta che un client accede a una servlet ed è in grado di interpretare il tipo di richiesta Http e invocare il metodo opportuno.
Per le applicazioni che utilizzano il protocollo Http non è necessario sovrascrivere il metodo service(). La classe che stiamo scrivendo per la nostra applicazione eredita quindi tale metodo definito all'interno della classe HttpServlet oltre che i metodi mostrati nella figura

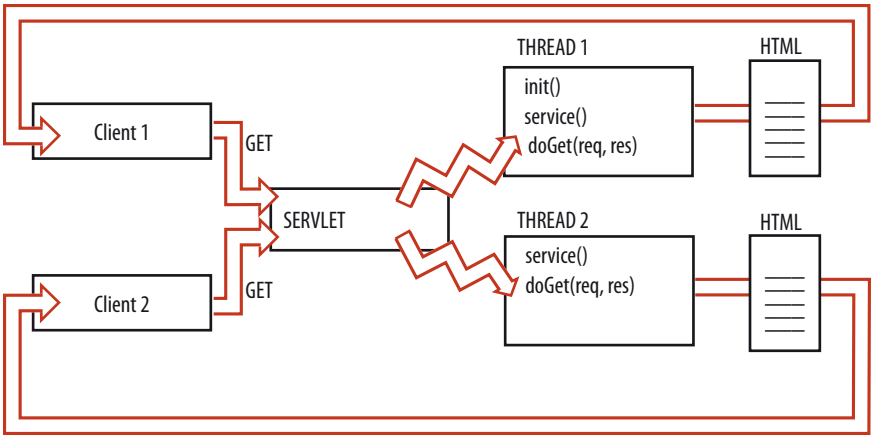



I due metodi principali sono:
- doGet(): processa le richieste di tipo GET effettuate dalla pagina HTML;
- doPost(): processa le richieste di tipo POST effettuate dalla pagina HTML.

Questi due metodi vengono automaticamente chiamati in corrispondenza delle rispettive richieste effettuate dalla pagina HTML.

## ESEMPIO

Se service() riconosce una richiesta GET viene richiamato automaticamente doGet() passandogli due oggetti (per esempio req e res), rispettivamente il primo del tipo HttpServletRequest, che contiene i dati passati dal client e HttpServletResponse nel quale verranno "inserite" le risposte per il client.




## HTTPServletRequest

Un oggetto HttpServletRequest estende ServletRequest, viene passato da service() e contiene la richiesta del client: permette di ottenere i parametri inviati dal client, il riferimento alla sessione utente e il flusso dei dati inviati dal client oltre che riconoscere l'utente autenticato.
I metodi più utilizzati su questi oggetti sono:

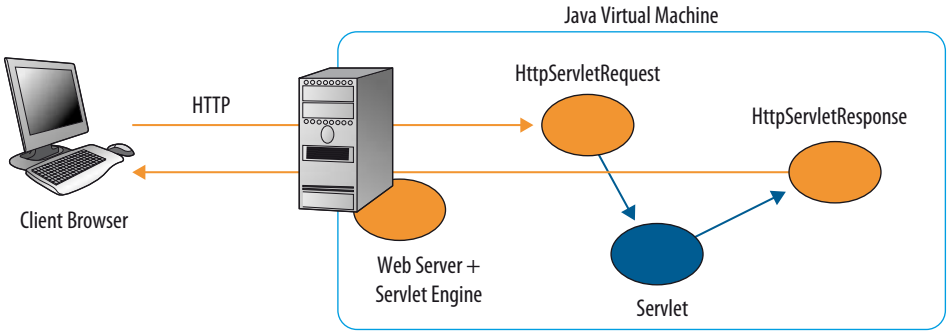
- getRequestURI() che restituisce l'URI richiesto;
- getMethod() che fornisce il metodo Http utilizzato per ri-inoltrare la richiesta;
- getParameter(String) che consente di accedere per nome ai parametri contenuti nella query string del client;
- getInputStream() e getReader() che permettono di creare gli stream di input e la possibilità quindi di ricevere i dati della richiesta Http.

## HttpServletResponse

Un oggetto HttpServletResponse estende ServletResponse, viene passato da service() e contiene la risposta per il client: permette di inviare dati al client in formato HTML oppure come flusso binario, di inviare codici di errore e codici di controllo nell'intestazione della risposta Http per controllare il comportamento del browser.
I metodi per creare gli stream di output e la possibilità quindi di inviare i dati della risposta sono:

- getOutputStream() per l'invio di dati in forma binaria;
- getWriter() per l'invio attraverso il canale System.out.

Inoltre la classe HttpServletResponse fornisce dei metodi per accedere e fissare gli header della risposta, come il setContenType(), che deve essere definito prima di poter inviare il body al client.



ESEMPIO

Iniziamo a scrivere una servlet di esempio dove utilizziamo doGet() ipotizzando che il client invii i dati dal form HTML mediante GET: lo completiamo con i suoi due parametri e il "rilancio" delle eventuali eccezioni IOException e ServletException:

```
8     public void doGet(HttpServletRequest req, HttpServletResponse res)
10        throws IOException, ServletException
```

## Ciclo di vita di una servlet

Prima di procedere con il codice della servlet è necessario analizzare il suo ciclo di vita, che può essere riassunto in tre fasi: init() → service() → destroy().

### init()

Al momento del caricamento della servlet in memoria il container delle servlet invoca il metodo init() che provvede all'inizializzazione della servlet e al settaggio delle sue variabili globali in modo da renderla disponibile per rispondere alla prima richiesta.
Il metodo init() riceve come parametro un oggetto di tipo ServletConfig che contiene la configurazione iniziale di una servlet e per salvare la configurazione richiama il metodo super.init.

Il metodo init() viene chiamato una sola volta, all'atto del caricamento che avviene alla prima chiamata: se la servlet non riesce a completare l'inizializzazione viene generata un'eccezione.

Nel nostro primo esempio non riscriviamo il metodo init() ma vedremo come utilizzarlo in seguito.

### service()

Le funzionalità della servlet, come ricevere e rispondere alle richieste dei client, sono effettuate dal metodo service(), o dai suoi sostituti nel protocollo Http doGet() e doPost(), che gestiscono le richieste mediante i due oggetti ServletRequest e ServletResponse prima descritti.

### destroy()

La terminazione di una servlet avviene tipicamente solo quando termina l'esecuzione del Web server, che si occupa anche di rilasciare le risorse occupate: questa operazione avviene tramite la chiamata al metodo destroy(). Può anche essere utilizzato per salvare tutte quelle informazioni che possono essere utili al successivo riavvio della servlet stessa.

## Output sul client

Nei nostri esempi invieremo sempre ai client come risposta una pagina HTML utilizzando il metodo getWriter(): prima di utilizzarlo è però necessario settare nell'oggetto response il tipo MIME della risposta con il metodo setContentType() e solo successivamente possiamo creare un oggetto PrintWriter sul quale scriveremo i tag HTML da inviare al client:

```
8     response.setContentType("text/html");          // content type
9     PrintWriter output = response.getWriter();     // get writer
```

Sull'oggetto PrintWriter creiamo la pagina HTML da inviare al client: creiamo l'oggetto output di comunicazione dopo aver indicato la tipologia con il metodo:

```
10    // crea una pagina HTML e la invia al client
11    output.println("<html>");
12    output.println("<head>");
13    output.println("<title>Primo esempio di servlet </title>");
14    output.println("</head>");
15    output.println("<body bgcolor=\"white\">");
16    output.println("<h1>Buongiorno, questa e la prima servlet!</h1>");
17    output.println("</body>");
18    output.println("</html>");
```

La pagina HTML può essere creata anche definendo un oggetto buffer del tipo StringBuffer e "aggiungendo" su di esso le diverse righe per poi visualizzarlo sull'oggetto PrintWriter:

```
public class EsempioBuffer extends HttpServlet{
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws IOException, ServletException  {
    res.setContentType("text/html");          // content type
    PrintWriter output = res.getWriter();     // get writer
    // crea una pagina HTML e la invia al client
    StringBuffer buffer = new StringBuffer();
    buffer.append( "<HTML><HEAD><TITLE>\n" );
    buffer.append( "Secondo esempio di servlet\n" );
    buffer.append( "</TITLE></HEAD><BODY bgcolor=\"white\">\n" );
    buffer.append( "<h1>Buongiorno, questa è la seconda servlet!</h1>" );
    buffer.append( "</BODY></HTML>" );
    output.println( buffer.toString() );
    output.close(); // chiusura del PrintWriter stream
  }
}
```

I codici di questi due esempi sono nei file EsempioTesto.java ed EsempioBuffer.java.

A questo punto compiliamo la classe e procediamo con le operazioni successive di configurazione necessarie per poterla eseguire su di un server: questa operazione prende il nome di deployment.

## Deployment di un'applicazione Web

Per poter far funzionare il nostro esempio è necessario effettuare tutti i passaggi previsti dal deployment, e cioè:

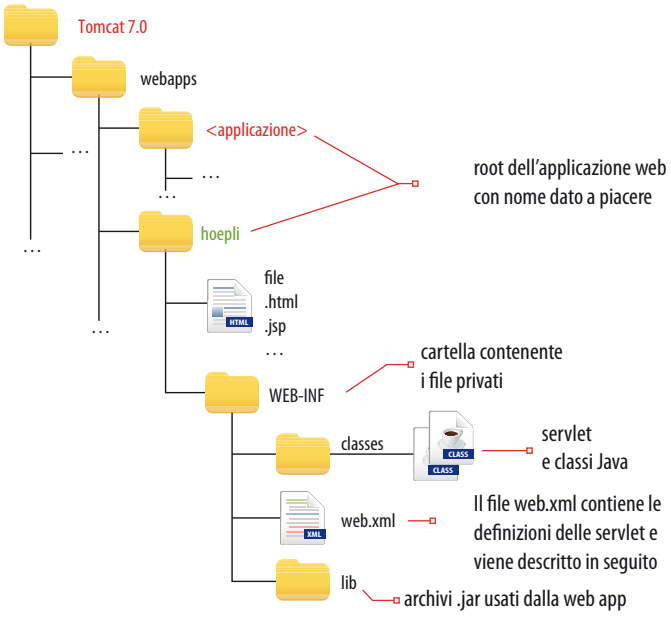
- la definizione del run time environment di una Web application (WAP);
- il posizionamento di tutti i file dell'applicazione nelle opportune directory;
- la mappatura delle URL sulle servlet;
- la definizione delle impostazioni di default di un'applicazione (per esempio la welcome page e la error page);
- la configurazione dei vincoli di sicurezza dell'applicazione.

Nel nostro calcolatore deve essere già stato installato il Web server: il procedimento viene descritto nell'Esercizio in Laboratorio 2 di questa Unità.

Supponiamo di avere già installato Tomcat come Web server: questo richiede per ogni WAP una particolare strutturazione delle directory collocate a partire dalla sua cartella webapps, che è la top level directory di tutte le applicazioni, che deve essere rispettata dal programmatore:

- nella root directory dell'applicazione, nel nostro caso hoepli, vengono copiati i file: *.html, *.jsp ecc.;
- viene creata una subdirectory WEB-INF con due sottocartelle classes e lib:
  - in WEB-INF si memorizza il file web.xml (Deployment Descriptor);
  - in WEB-INF/classes i file *.class delle servlet (bytecode);
  - in WEB-INF/lib eventuali file di libreria *.jar.

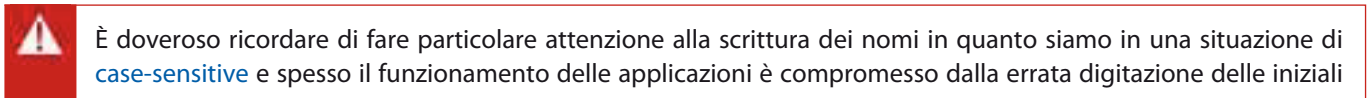
L'albero completo di una Web application è raffigurato a lato.



Predisponiamo la struttura destinata a raccogliere la nostra prima applicazione: nel nostro disco ci posizioniamo nella directory programmi dove individuiamo la cartella che contiene il software del Web server (per esempio Apache Software Foundation/Tomcat); in quella posizione creiamo sotto webapps la directory esempioTesto, che sarà la root della nostra applicazione.
Replichiamo la struttura richiesta da Tomcat, cioè in sequenza:

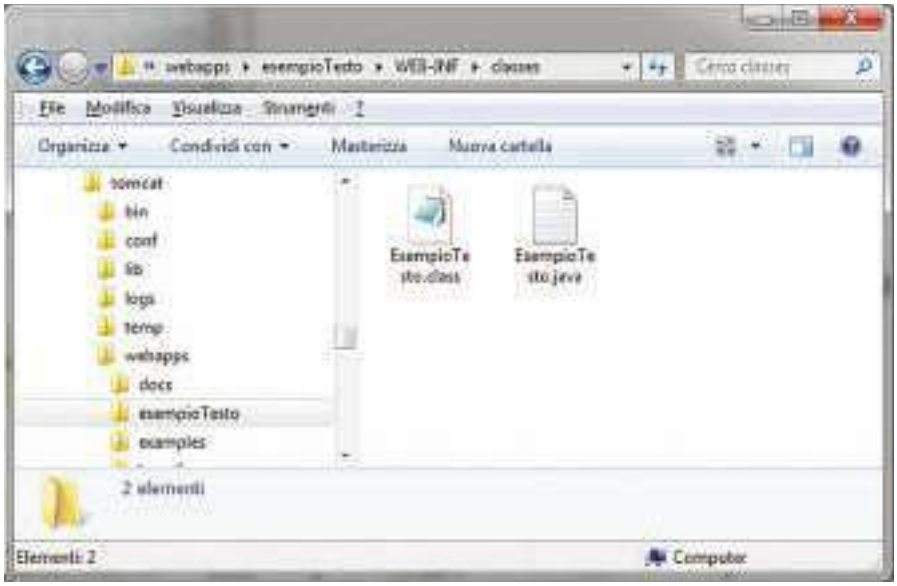
1 nella cartella webapps creiamo la cartella col nome della nostra applicazione (noi realizzeremo due applicazioni, esempioTesto e hoepli);
2 creiamo la cartella WEB-INF;
3 ci posizioniamo al suo interno e creiamo la cartella classes;
4 ci posizioniamo nella cartella classes e vi copiamo la nostra classe EsempioTesto.class.

È comodo (ma non necessario) copiare nella cartella anche il file sorgente EsempioTesto.java, in modo da poterlo consultare ed eventualmente modificare.



È doveroso ricordare di fare particolare attenzione alla scrittura dei nomi in quanto siamo in una situazione di case-sensitive e spesso il funzionamento delle applicazioni è compromesso dalla errata digitazione delle iniziali minuscole e/o maiuscole.

A questo punto siamo nella seguente situazione:




Per mandare in esecuzione l'applicazione abbiamo due modalità:

1 digitare nel browser il nome della pagina HTML di inizio della applicazione che al suo interno richiama la servlet:

```
http://localhost:8080/<applicazione>/index.html
```

2 digitare nel browser il nome della servlet connessa alla applicazione:

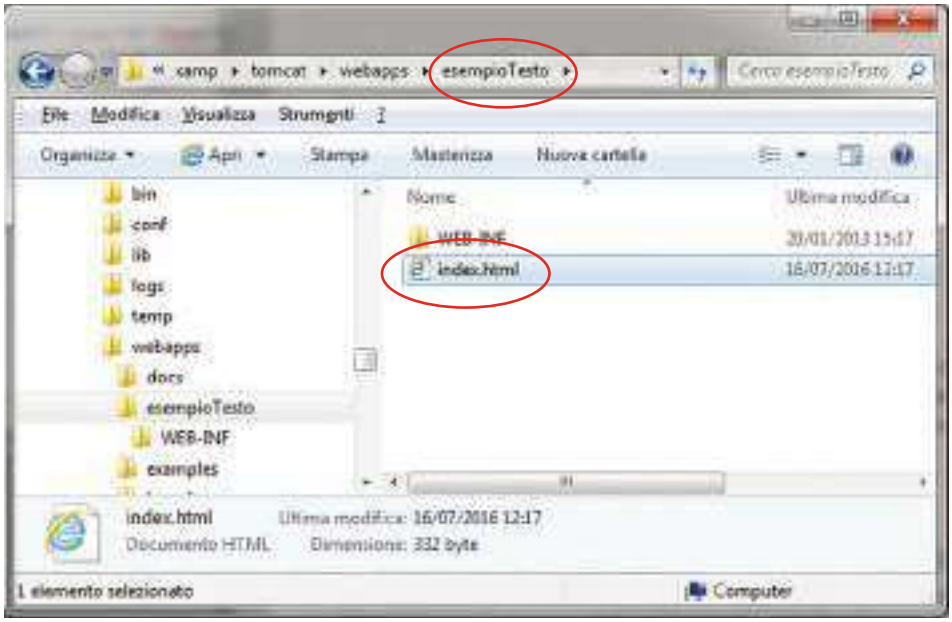
```
http://localhost:8080/<applicazione>/servlet/<nome_servlet.class>
```

## ESEMPIO

Realizziamo una semplice pagina HTML, la index.html, con un hyperlink che non fa altro che richiamare la nostra servlet presente in servlet/EsempioTesto.

```
<html>
<head>
<title>Primo esempio di servlet</title>
</head>
<body>
<p><strong><font color="#0000FF" size="4" face="Arial, Helvetica, sans-serif">Primo
  esempio di utilizzo della servlet</font></strong></p>
<p><a href="servlet/EsempioTesto">Per eseguire la servlet clicca qui!</a> </p>
<p> </p>
</body>
</html>
```

La posizioniamo nella cartella principale della nostra applicazione Web:




A questo punto abbiamo stabilito il contesto che deve essere registrato in un apposito file XML, il file web.xml, dove vengono riportate le Path dei singoli file in modo che il Web server li possa ritrovare al momento della richiesta di esecuzione nella applicazione: questo file prende il nome di Deployment descriptor o Context XML descriptor.

## Il Context XML descriptor o Deployment descriptor

Il file web.xml, chiamato anche Deployment descriptor, serve per configurare l'applicazione Web in modo che Tomcat possa gestire le richieste da parte dei client: contiene l'elenco dei servlet e per ogni servlet permette di definire una serie di parametri come coppie nome-valore che costituiscono la descrizione del Context associato a una Web application.
Nel file web.xml vengono specificati diversi parametri come:

- il nome della classe che definisce la servlet;
- i percorsi che causano l'invocazione della servlet da parte del container;
- una serie di parametri di configurazione (coppie nome-valore).

Contiene anche la mappatura fra URL e servlet che compongono l'applicazione.

La struttura completa è la seguente:

<web-app> definisce la configurazione di tutte le servlet dell'applicazione web

<display-name> specifica un nome che viene utilizzato dall'amministratore del server

<description> specifica una descrizione dell'applicazione che viene visualizzata nel pannello del manager

<servlet-name> definisce un nome per la servlet all'interno di questo file

<description> fornisce una descrizione di questa servlet

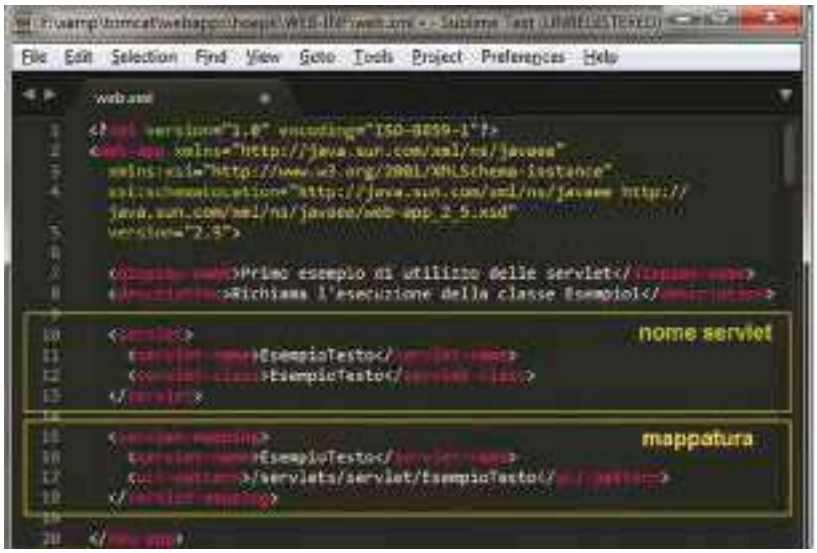
<servlet-class> specifica il nome completo della classe servlet compilata

<servlet-mapping> specifica l'associazione tra il nome dato alla servlet nell'interno del file web.xml (<servlet-name>) e l'<url-pattern> con cui la servlet potrà essere invocata

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
  <!--descrizione generale della applicazione web -->
  <display-name>
    Primo esempio di utilizzo delle servlet
  </display-name>

  <description>
    Richiama l'esecuzione della classe Esempio0
  </description>

  <!-definizione delle servlet -->
  <servlet>
    <servlet-name>Esempio0</servlet-name>

    <description>Visualizza una riga</description>

    <servlet-class>Esempio0</servlet-classe>

  </servlet>

  <servlet-mapping>
     <servlet-name>Esempio0</servlet-name>
     <url-pattern>/servlet/Esempio0</url-pattern>
  </servlet-mapping>

</web-app>
```

## ESEMPIO

Nel nostro esempio il file web.xml può essere semplicemente il seguente:




L'URL può anche essere scritto in formato assoluto:

```
http://localhost:8080/EsempioTesto/servlet/EsempioTesto
```

Lo scriviamo con un editor e lo collochiamo nella cartella WEB-INF.

## Esecuzione di una servlet

Al momento dell'invocazione di una servlet, Apache Tomcat esegue diverse operazioni:

- legge il file web.xml;
- trova un'entry <servlet-mapping> da cui capisce che una servlet deve essere eseguita in risposta all'URL richiesto;
- trova un'entry <servlet> che indica il nome della classe Java;
- controlla se esiste in memoria il file .class corrispondente o se deve essere ricaricato;
- viene creato un thread e mandato in esecuzione.

Ora siamo pronti per mandare in esecuzione la nostra servlet: come prima esecuzione richiamiamo la pagina index.html presente nella cartella esempioTesto digitandone nel browser semplicemente il nome dell'applicazione:

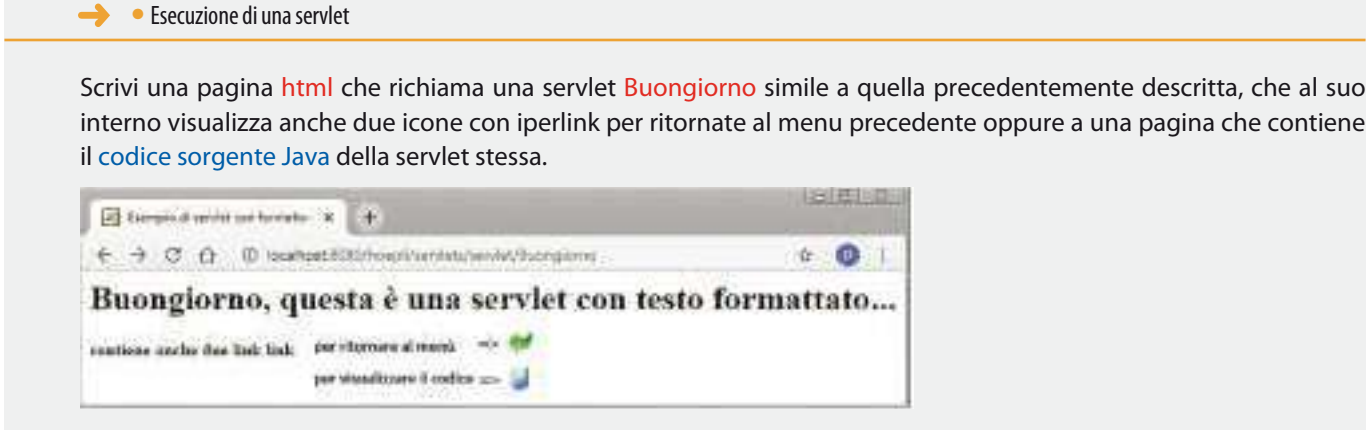
Cliccando sul collegamento hyperlink viene mandata in esecuzione la servlet che produce il risultato mostrato a lato.

Al medesimo risultato si giunge anche richiamando direttamente la servlet dalla linea di comando del browser:

Quando viene ricompilata una servlet e copiata nella cartella classes è necessario riavviare Tomcat per rendere effettiva la sostituzione con quella precedentemente caricata nella RAM del server.

L'installazione di Tomcat e la descrizione completa dell'utilizzo dell'ambiente di sviluppo viene descritto nell'Esercizio di Laboratorio 2 di questa Unità.

### METTITI ALLA PROVA



- Esecuzione di una servlet

Scrivi una pagina html che richiama una servlet Buongiorno simile a quella precedentemente descritta, che al suo interno visualizza anche due icone con iperlink per ritornate al menu precedente oppure a una pagina che contiene il codice sorgente Java della servlet stessa.

## Servlet concorrenti

Le servlet hanno il compito di servire le richieste contemporanee dei client e il Web server per definizione utilizza una sola istanza di servlet condividendola fra le varie richieste: a fronte di ogni richiesta da parte di client della stessa servlet viene creato un threads differente da parte del Web server capace di eseguire la servlet in maniera concorrente.

È però necessario tenere conto dei meccanismi di accesso concorrente alle servlet e ai loro dati in fase di programmazione, mediante l'operatore synchronized, come studiato nei classici problemi di concorrenza.

## Vantaggi e svantaggi delle servlet

### Vantaggi delle servlet

Come detto in precedenza, ancor oggi le tecnologie CGI sono molto utilizzate. Ricapitoliamo in breve i vantaggi più evidenti delle servlet rispetto alla chiamate CGI.

- Efficienza: si è detto che le servlet vengono istanziate e caricate una volta soltanto, alla prima invocazione e alle successive richieste da parte di nuovi client vengono creati nuovi thread e quindi non deve essere ricaricato ogni volta tutto il codice della classe, con il meccanismo delle CGI a ogni richiesta da parte di un client il server effettua il caricamento completo del processo CGI con un notevole degrado delle prestazioni.
- Portabilità: la tecnologia Java è facilmente portabile su ogni piattaforma senza problematiche connesse alla compatibilità in quanto sono intrinsecamente risolte dalla JVM.
- Persistenza: quando una servlet viene caricata rimane in memoria, mantenendo intatte determinate informazioni anche alle successive richieste.
- Gestione delle sessioni: il protocollo Http è un protocollo stateless e quindi non è in grado di ricordare i dettagli delle precedenti richieste provenienti da uno stesso client: con le servlet è possibile superare queste limitazioni.

### Svantaggi delle servlet

I principali inconvenienti che denunciano gli sviluppatori di applicazioni Web che utilizzano le servlet sono i seguenti:

- con le servlet si "mescola" la logica dell'applicazione con la presentazione;
- se il codice HTML è complesso, a volte è "particolarmente oneroso" doverlo "inglobare" all'interno del codice Java per mezzo di istruzioni di println();
- non è possibile la "prototipazione rapida": ogni modifica del codice richiede una ricompilazione esplicita e ogni aggiunta il cambiamento del file web.xml con il riavvio dell'applicazione o del server.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Quale di queste affermazioni sulle servlet è falsa?**
**a** Sono molto simili alle applet
**b** Sono eseguite dalla JVM sul server
**c** Richiedono un basso tempo di latenza
**d** A ogni servlet è associato un solo thread
**e** Sono più efficienti delle CGI

**2 Quale dei seguenti compiti non è relativo al container?**
**a** Gestisce la comunicazione con i client
**b** Gestisce la sicurezza
**c** Realizza il multithreading
**d** Gestisce la attivazione della servlet
**e** Effettua la compilazione della servlet
**f** Gestisce la terminazione della servlet

**3 Quale tra i seguenti metodi non viene utilizzato nel protocollo Http?**
**a** doGet()
**b** doPost()
**c** doTrace()
**d** doService()
**e** doDelete()
**f** doHead()

**4 Nella subdirectory WEB-INF non è presente:**
**a** la cartella webapps
**b** la cartella classes
**c** la cartella lib
**d** il file web.xml

**5 Quale tra i seguenti non è un vantaggio delle servlet rispetto alle CGI?**
**a** Efficienza
**b** Portabilità
**c** Persistenza
**d** Prototipazione rapida
**e** Gestione delle sessioni

**6 Quale tra i seguenti non è uno svantaggio delle servlet rispetto alle CGI?**
**a** Con le servlet si "mescola" la logica dell'applicazione con la presentazione
**b** Se il codice è "particolarmente oneroso" doverlo "inglobare" all'interno del codice Java
**c** Non è possibile superare la limitazione stateless del protocollo Http
**d** Non è possibile la "prototipazione rapida"

## ORDINAMENTO

**1 Ordina la gerarchia completa di una servlet:**
**2** ..... classe <applicazione>
**3** ..... classe astratta GenericServlet
**4** ..... classe astratta HttpServlet
**5** ..... interfaccia Servlet

## VERO/FALSO

1 Il programma CGI è scritto in un linguaggio di programmazione server oriented. V F
2 Gli script CGI vengono eseguiti dal sistema operativo. V F
3 Le servlet vengono eseguite dal sistema operativo. V F
4 Il Web server svolge la funzione di container occupandosi della gestione del ciclo di vita delle servlet. V F
5 Una servlet interagisce con un Web client attraverso il paradigma request/response. V F
6 Le servlet usate nel Web sono estensioni della classe javax.servlet.Servlet. V F
7 Con deployment si intendono le operazioni eseguite per collaudare una servlet. V F
8 La cartella webapps è la top level directory di tutte le applicazioni su Tomcat. V F
9 Il file web.xml è chiamato anche deployment descriptor. V F

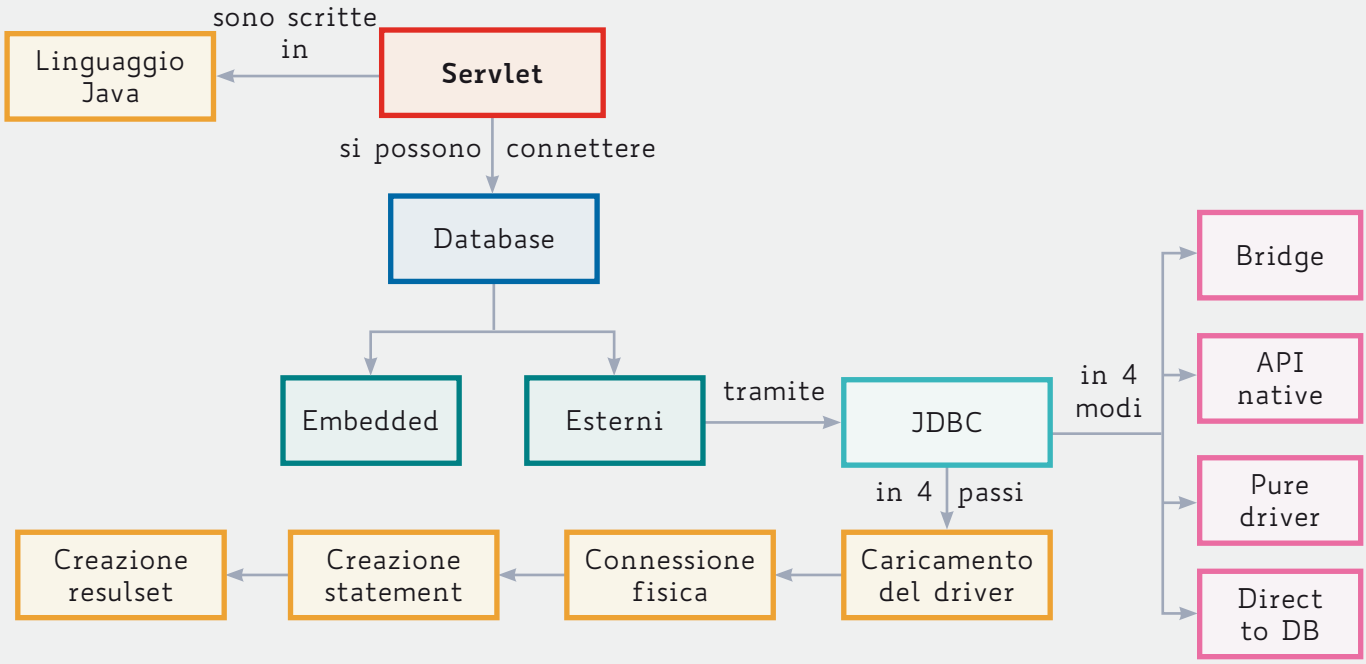
# 2 Servlet e database

**IN QUESTA LEZIONE IMPAREREMO...**

- le caratteristiche dell'interfaccia JDBC
- la connessione a concetti MySQL e Access con UCanAccess

## MAPPA CONCETTUALE

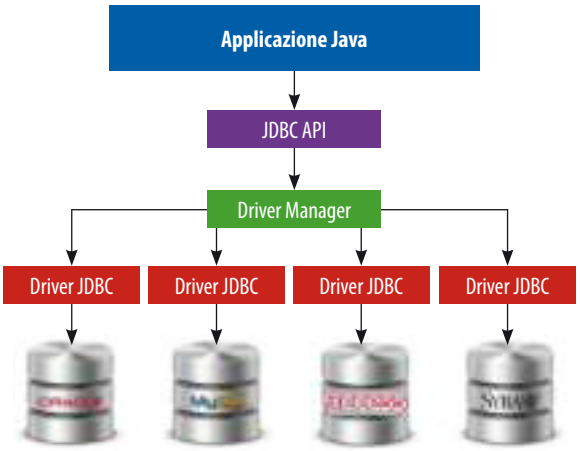
didattica inclusiva



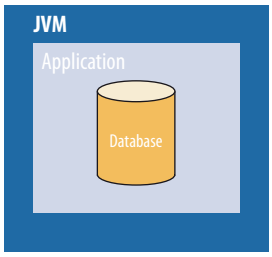

## La connessione ai database

Un'applicazione Java può connettersi e comunicare con due "famiglie " di database:
**a** database "esterni";
**b** database embedded.

Con Java è possibile accedere a database esterni, come Oracle, MySQL, Sysbase, Access, tramite il protocollo JDBC, uno standard industriale indipendente dal DBMS impiegato: il suo utilizzo permette di impostare il codice in maniera che rispetti il principio, caro al mondo Java, del "Write once, run anywhere", cioè un software riutilizzabile al di là delle tecnologie e dei contesti utilizzati inizialmente.




Da alcuni anni sono anche disponibili database embedded: con questa tecnologia i DBMS sono incorporati nell'applicazione, o nel linguaggio stesso, anziché fornite come sistemi di database autonomi.




> I database incorporati possono essere molto utili durante le fasi di sviluppo o test perché sono leggeri, veloci, migliorano la testabilità e offrono semplicità di configurazione.

Tra le possibili alternative presenti segnaliamo SQLite, H2 e HSQLDB (HyperSQL Database), Apache Derby (JavaDB), che saranno descritte nella prossima Lezione.

## La connessione con JDBC Java Database Connectivity

Sin dalle prime versioni di Java, la SUN ha messo a disposizione le API che permettono alle applicazioni scritte in questo linguaggio di avere un livello di astrazione uniforme verso i database: le JDBC.
JDBC racchiude una serie di classi per poter accedere alle basi di dati con metodi e schemi di funzionamento molto intuitivi, in linea con lo stile di programmazione tipico di Java; sfruttando un apposito driver JDBC costituito da una classe Java è possibile connettersi a un particolare database e oggi tutti i principali DBMS dispongono di uno specifico driver JDBC.

> Spesso JDBC viene interpretato come un acronimo (Java DataBase Connectivity) ma la Sun non ne rivendica la paternità e sostiene che JDBC è semplicemente il nome di un'interfaccia, cioè di uno "strato di programmazione" che si interpone tra il codice Java e i database.

A corredo di JDBC vi è inoltre un particolare driver, chiamato ponte JDBC-ODBC, che permette l'utilizzo di qualsiasi fonte di dati per la quale è disponibile un driver ODBC: mediante JDBC e il ponte ODBC è quindi possibile interagire con tutti i DBMS presenti sul mercato da codice Java, sia dalle applicazioni stand-alone, che dalle servlet e dalle JSP.

**AREA DIGITALE**
**ODBC**

La Sun ha incluso JDBC nel JDK a partire dalla versione 1.1, rimarcando la filosofia multipiattaforma che ha caratterizzato questo linguaggio non solo nei confronti dei sistemi operativi ma anche verso i database; lo scopo è quello di offrire la medesima modalità per connettersi ai database più diffusi e utilizzati in ambito industriale.
Non ci addentriamo nei dettagli, ma dal punto di vista dello sviluppatore sono stati messi a disposizione tre strumenti che permettono di interagire omogeneamente con tutti i diversi database:
- gli strumenti per la connessione al database desiderato;
- i metodi per inviare i comandi SQL;
- alcuni potenti meccanismi per gestire le risposte ricevute dalla query effettuate sui database.

Sottolineiamo che dal punto di vista del programmatore è completamente trasparente il tipo di DB che sta utilizzando in quanto interagisce con tutti mediante la stessa interfaccia, cioè JDBC, che si connette al DB fisico semplicemente cambiando il "nome del driver", specifico per ogni database.

# Tipi di driver JDBC

I driver JDBC possono essere di quattro diverse tipologie.

## Tipo 1. Bridge JDBC-ODBC

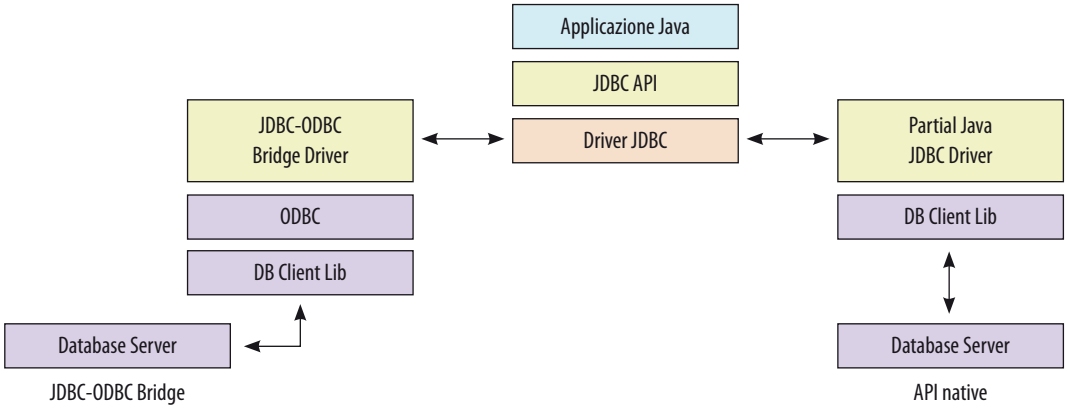
È il driver che permette l'accesso al DB nel caso in cui non sia stata realizzata una vera e propria implementazione JDBC verso un particolare database JDBC: si utilizza quindi un JDBC-ODBC bridge cioè un "driver-ponte" che converte le chiamate JDBC in ODBC.
Il codice ODBC binario e il codice del DBMS client devono essere caricati su ogni macchina client che usa JDBC-ODBC bridge.
È consigliabile utilizzare questa strategia solo quando non esistono soluzioni alternative dato che richiede l'installazione delle librerie sul client violando la filosofia Write Once, Run aAnywhere (**Wora**) alla base delle applicazioni Java.

**Wora**
Wora è uno slogan creato da Sun Microsystems per illustrare i benefici cross-platform del linguaggio Java. Fin dall'inizio l'intenzione di Sun è stata quella di realizzare Java come linguaggio indipendente dalla piattaforma grazie alla macchina virtuale: "Lo scrivi una volta, lo esegui dappertutto" significa svincolare il programmatore dall'hardware e quindi dalla configurazione della macchina e questo obiettivo viene a mancare se per "far funzionare" un programma Java su una certa macchina è necessario installare componenti aggiuntivi.

Dalla versione Java 8 viene deprecato il bridge JDBC-ODBC e quando si passa a questa versione è necessario sostituire la tipologia di connessione: noi useremo la libreria UCanAaccess.

## Tipo 2. API Native

Quando si ha disposizione API native, come per esempio nel caso di Oracle, le chiamate JDBC non vengono tradotte in ODBC ma vengono utilizzate queste API appositamente per il database.
Le API native sono generalmente scritte in C o C++ e sono molto performanti ma, come per il caso precedente, devono essere presenti nel client, quindi anche questo tipo di driver compromette la filosofia Wora.

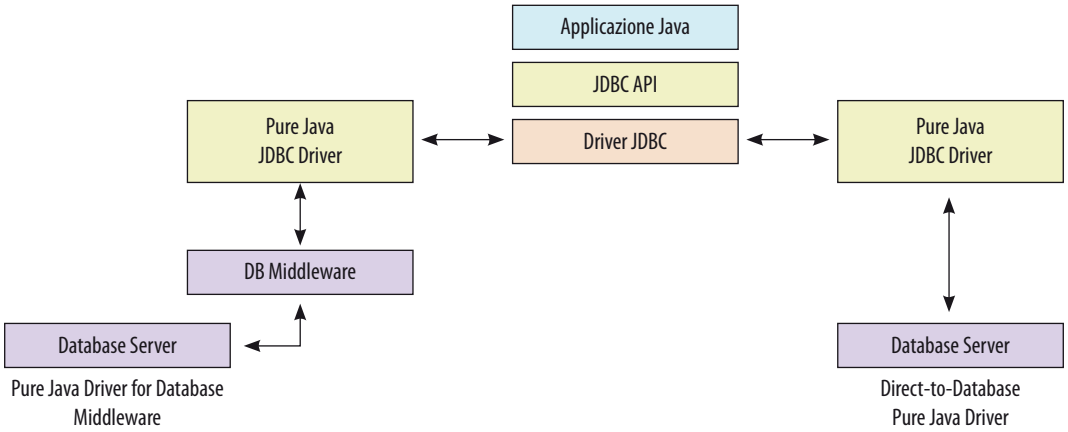



## Tipo 3. Pure Java Driver for Database Middleware

Rispetto ai precedenti tipi, questi driver, scritti interamente in Java, sono particolarmente indicati da utilizzarsi con le applet/servlet, convertendo le chiamate JDBC nel protocollo di un'applicazione middleware che, nel caso di una LAN non congestionata, potrebbe essere un vantaggio: è l'applicazione middleware che traduce la richiesta del client nel protocollo proprietario del DBMS sottostante garantendo l'accesso a diversi DBMS.

### Tipo 4. Direct-to-Database Pure Java Driver

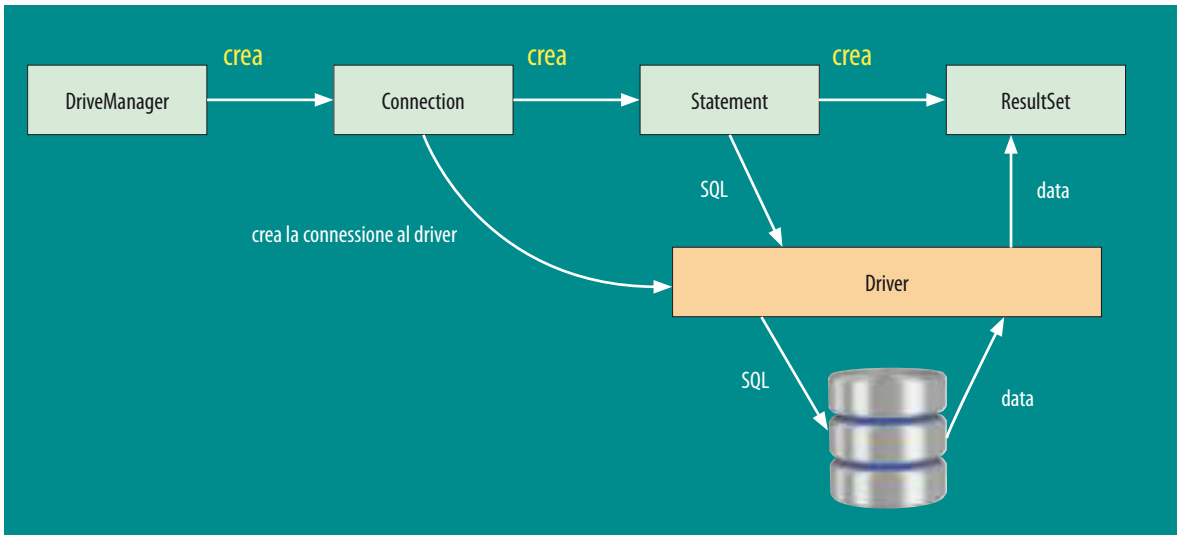
Questa tipologia è quella maggiormente indicata per le applicazioni Java in quanto il driver è realizzato totalmente in Java e trasforma le chiamate JDBC direttamente nel protocollo del database, quindi permette un collegamento diretto senza bisogno di middleware o client installati.



UcanAccess è un Pure Java Driver, è quindi noi "sposeremo" questa modalità di connessione.

## Utilizzare JDBC standalone

L'ultima versione disponibile di JDBC è 5.1.48 ed è quella che noi utilizzeremo in questo volume: il suo utilizzo è molto semplice e può essere schematizzato in quattro operazioni, rappresentate nella figura seguente.




### 1. Driver

Per prima cosa è necessario caricare il driver idoneo per l'utilizzo del particolare database che si intende sfruttare: il nostro primo esempio di connessione sarà effettuato servendosi di MySQL, che dispone sia di un driver JDBC sia di un driver ODBC.
Utilizziamo il driver JDBC chiamato Connector/J scaricabile da: http://dev.mysql.com/downloads/connector/j/.
Affinché tale driver possa essere visto dalle applicazioni Java, è necessario notificare alla macchina virtuale la presenza dell'archivio JAR contenuto nel pacchetto scaricato.
È possibile operare in diversi modi e il più semplice è depositare una copia dell'archivio JAR di Connector/J

all'interno della directory jre/lib/ext, nella cartella che contiene l'installazione di Java. La tecnica più efficace (e più corretta) è quella di aggiungere il percorso dell'archivio JAR nella variabile di sistema CLASSPATH.

## ESEMPIO

Riportiamo un primo segmento di codice che permette di realizzare la connessione: dopo aver importato il contenuto del pacchetto java.sql, che contiene le API JDBC, per prima cosa definiamo una stringa dove scriviamo il nome del driver che utilizzeremo per la connessione, che nel nostro caso è com.mysql.jdbc.Driver.

```
1  import java.sql.*;
2  public class EsempioDB1
3  {
4    public static void main(String[] args)
5    {
6      // carico il driver per la connessione al DB mysql
7      String DRIVER = "com.mysql.jdbc.Driver";
```

Proviamo a effettuare il suo caricamento in un costrutto try… cath per intercettare l'eventuale errore di mancato caricamento: una java.lang.ClassNotFoundException, infatti, può essere propagata nel caso in cui non sia possibile trovare il driver nel CLASSPATH della macchina virtuale.

```
8        try                          // carico il driver
9        {
10         Class.forName(DRIVER);
11       }catch (ClassNotFoundException e1)
12       {                            // il driver non può essere caricato
13         System.out.println("Driver non trovato...");
14         System.exit(1);
15       }
```

## 2. Connessione

Come secondo passo si apre una connessione verso un database necessario all'applicazione, sfruttando il driver caricato al passo precedente. Definiamo due variabili.

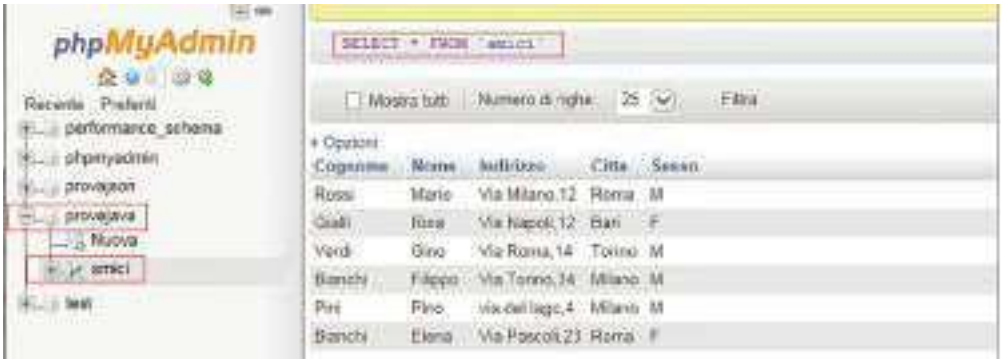
```
19       // nome ed indirizzo del database
20       String URL_mioDB = "jdbc:mysql://localhost:3306/provejava";
21       // definizione delle query
22       String query = "SELECT Cognome, Nome, Indirizzo, Citta FROM amici";
```

- La prima variabile per memorizzare l'indirizzo del database; la sintassi desiderata da Connector/J è la seguente:

  ```
  jdbc:mysql://[hostname][:port]/[dbname][?param1=val1][&param2=val2]...
  ```

  nel nostro caso stiamo eseguendo MySQL nella stessa macchina che eseguirà la classe (local host) e la porta ascoltata dal DBMS è la numero 3306, valore di default predefinito di MySQL.

- La seconda per contenere la stringa di query in linguaggio SQL che estrae i nominativi della tabella amici riportata di seguito.



Dopo aver creato un oggetto Connection necessario per stabilire una connessione con il DBMS cerchiamo di stabilire la connessione in un blocco try... catch, poiché è a rischio di eccezione.

```
// stabilisco la connessione
System.out.println("Connessione con: " + URL_mioDB);
Connection connessione = null;
try                                  // apro la connessione verso il database
{
  connessione = DriverManager.getConnection(URL_mioDB, "root", "");
}
catch (Exception e)
{                                    // gestione dell'errore
  System.out.println("Errore nella connessione: " + e);
  System.exit(1);
}
```

I parametri che passiamo al metodo getConnection() sono riportati nella seguente tabella.

| | |
|---|---|
| **nome e indirizzo del database:** | "jdbc:mysql://localhost:3306/provejava" |
| **nome utente:** | "root" |
| **password (nessuna):** | "" |

Siccome le connessioni occupano risorse, in un ambiente multiutente e multitasking è opportuno adottare la seguente politica:

- aprire una connessione solo quando necessario;
- assicurarsi di chiuderla al termine del suo utilizzo;
- e soprattutto non aprire/chiudere connessioni inutilmente.

## 3. Statement

Se la connessione va a buon fine, viene creato un oggetto java.sql.Statement, necessario per interagire con il DBMS attraverso delle query espresse in linguaggio SQL che memorizzano il risultato in un oggetto java.sql.ResultSet:

```
try
{
  // creo uno Statement per interagire con il database
  Statement statement = connessione.createStatement();
  // interrogo il DBMS mediante una query SQL
  ResultSet resultSet = statement.executeQuery(query);
```

## 4. ResultSet

I risultati ottenuti possono essere manipolati sfruttando i metodi dell'oggetto ResultSet, per esempio con un ciclo scorriamo il contenuto del ResultSet o lo visualizziamo sullo schermo.

```
while (resultSet.next()) {
  String cognome = resultSet.getString(1);
  String nome = resultSet.getString(2);
  String indirizzo = resultSet.getString(3);
  String citta = resultSet.getString(4);

  System.out.println("Lette informazioni...");
  System.out.println("Cognome: " + cognome);
  System.out.println("Nome: " + nome);
  System.out.println("Indirizzo: " + indirizzo);
  System.out.println("Citta': " + citta);
  System.out.println();
}
```

Il metodo next() di un oggetto resultSet, ha duplice funzionalità:

- scorre in avanti l'elenco dei record ottenuti;
- restituisce true fin quando non si giunge al termine dell'esplorazione.

Il risultato a video è riportato a lato.

Al termine di ogni operazione la connessione viene chiusa nel ramo della clausola finally, dopo aver controllato se essa è ancora attiva (!=null).

```
finally{
  if (connessione != null){
    try {
      connessione.close();                    // chiusura connessione
    }
    catch (Exception e)
    {                                         // gestione dell'errore
      System.out.println("Errore nella chiusura della connessione");
    }
  }
}
```

Il codice completo è nel file EsempioDB1.java che puoi scaricare dalla cartella MATERIALI nella sezione del sito www.hoepliscuola.it dedicata a questo volume.

## Servlet con connessione a MySQL

Per connettere una servlet al database MySQL non ci sono particolari differenze rispetto alla applicazione stand-alone prima descritta: naturalmente è necessario aggiungere le direttive HTML per visualizzare i risultati nel browser del client.
L'unica rettifica/integrazione risulta essere il completamento del comando di connessione dove per le applicazioni lato server devo essere indicati l'utente e la password:

```
// apro la connessione verso il database
connessione = DriverManager.getConnection(URL_mioDB, "root", "");
```

Riportiamo di seguito solamente la parte del codice inerente alla visualizzazione, nel quale è stato aggiunto il codice HTML, necessario per visualizzare sullo schermo il risultato della interrogazione, in modo formattato utilizzando il tag <pre>:

```
// scorro e mostro i risultati.
out.println("<pre>");
out.println("<b>cognome"+"&#9;"+"nome"+"&#9;"+"indirizzo"+"&#9;"+"citta</b>");
while (resultSet.next()) {
  String cognome = resultSet.getString(1);
  String nome = resultSet.getString(2);
  String indirizzo = resultSet.getString(3);
  String citta = resultSet.getString(4);
  out.println(cognome+"&#9;"+nome+"&#9;"+indirizzo+"&#9;"+citta);
}
```

Mandando in esecuzione la servlet otteniamo il seguente output:

## Applicazione e servlet con connessione ad Access

**AREA DIGITALE**

**Installazione della libreria UCanAccess**

Come abbiamo già accennato, dalla versione 8 di Java viene deprecato il bridge JDBC-ODBC che, in ogni caso, ha sempre dato qualche problema per le differenze tra le versioni a 32bit e 64bit e per le diverse versioni di rilascio da parte della Microsoft del database Access.

In questa Lezione noi proponiamo di utilizzare quello che di fatto è divento lo "standard post ODBC", che sono le librerie UCanAccess. Esse sono scaricabili gratuitamente all'indirizzo http://ucanaccess.sourceforge.net/site.html e alla data di uscita di questo volume è disponibile la versione riportata nella figura a lato:

### Codice per classi standalone

Le rettifiche da apportare al programma precedentemente descritto sono minime: per prima cosa è necessario indicare la nuova "libreria di connessione" con l'istruzione di riga 20:

```
17    // parametri per la connessione al database access
18    String DNS = "";
19    String URL_mioDB = "";
20    DNS = "jdbc:ucanaccess://";                // connessione alla libreria
21    URL_mioDB = "F:/xamp/tomcat/webapps/hoepli/esempioDB/proveJava.mdb"; // path del db
22    URL_mioDB = DNS + URL_mioDB;
23    System.out.println("Connessione con: " + URL_mioDB);
```

Quindi si procede con la creazione della connessione, ricordando che in locale non è presente nessun utente e nessuna password per connettersi al file .mdb.

```
25    // stabilisco la connessione
26    String utente = "";
27    String psw    = "";
28    Connection connessione = null;
29    try                                        // apro la connessione verso il database
30    {
31      connessione = DriverManager.getConnection(URL_mioDB, utente, psw);
32    }
33      catch (SQLException e)                   // gestione dell'errore
34    {
35      System.out.println("Errore nella connessione");
36      System.exit(2);
37    }
```

Una volta definita la query, si procede creando un oggetto di tipo Statement necessario per inviare l'interrogazione SQL: e il suo risultato verrà memorizzato in un oggetto ResultSet che, di fatto, è una tabella composta da righe e colonne.

```
// definizione della query
String query = "SELECT Cognome, Nome, Indirizzo, Citta FROM Amici";

try                                    // esecuzione della query
{
  Statement statement = connessione.createStatement();
  // interrogo il DBMS mediante una query SQL
  ResultSet resultSet = statement.executeQuery(query);

  // scorro il resultSet e mostro i risultati
  while (resultSet.next()) {
    String cognome = resultSet.getString(1);
    String nome = resultSet.getString(2);
    String indirizzo = resultSet.getString(3);
    String citta = resultSet.getString(4);

    System.out.println("Lette informazioni...");
    System.out.println("Cognome: " + cognome);
```

Come già descritto nell'esempio precedente utilizziamo un ciclo per scorrere il contenuto del ResultSet e visualizzarlo sullo schermo.

## Codice per servlet

Per poter utilizzare la libreria UCanAccess nel codice delle servlet sono necessarie tre modifiche rispetto a quello scritto per l'accesso a MySQL:

**1** indicare il nome della libreria, come indicato alla riga 9;

**2** aggiungere al nome del database il parametro che indica le modalità di gestione della memoria: nel nostro caso è necessario porre il parametro memory=false;

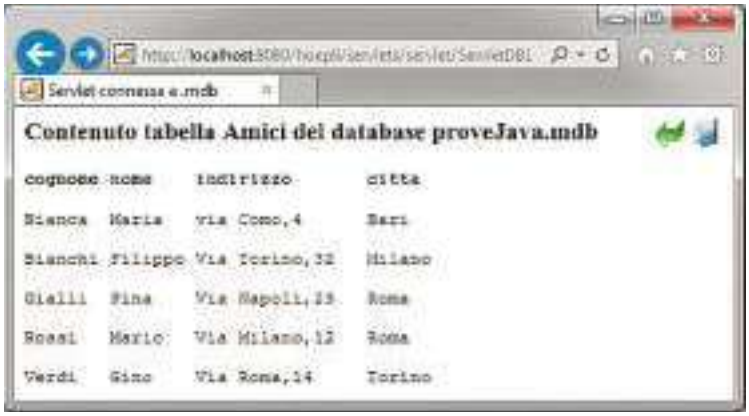
**3** indicare il nome completo del driver, come riportato nella riga 17.

```
public class ServletDB1 extends HttpServlet
{
  private final String protocollo = "jdbc:ucanaccess://";      // connessione alla libreria
  private final String mdbpath = "tomcat/webapps/hoepli/esempioDB/"; // percorso relativo
  private final String mdbName = "proveJava.mdb;memory=false";       // come database
  private final String user    = "";                                 // come utente
  private final String pw      = "";                                 // password
  // riferimento al database
  private final String URL_mioDB = protocollo + mdbpath + mdbName;
  // definizione del driver per la connessione al DB Access
  private final String DRIVER = "net.ucanaccess.jdbc.UcanaccessDriver";
```

Il resto della servlet non richiede interventi degni di particolari osservazioni: negli Esercizi in Laboratorio saranno presentati esempi completi e proposte situazioni da risolvere come esercitazione.

## Output dei programmi

Il risultato delle due esecuzioni è riportato nelle seguenti figure in modo da poter effettuare il confronto.



# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 ODBC è l'acronimo di:**
- **a** Open DataBase Connectivity
- **b** Object DataBase Connectivity
- **c** Open DataBase Comunication
- **d** Object DataBase Comunication

**2 ODBC offre all'applicazione un'interfaccia che consente l'accesso a un DB indipendente da:**
- **a** il particolare dialetto di SQL
- **b** il tipo di sistema operativo
- **c** il protocollo di comunicazione da usare con il DBMS
- **d** la posizione del DBMS (locale o remoto)

**3 I driver JDBC possono essere di quattro diverse tipologie: Indica quello non corretto**
- **a** Tipo Bridge JDBC-ODBC
- **b** Tipo API Native
- **c** Tipo Direct Bridge for Database Middleware
- **d** Tipo Pure Java Driver for Database Middleware
- **e** Tipo Direct-to-Database Pure Java Driver

**4 L'utilizzo di JDBC è molto semplice e può essere schematizzato in quattro operazioni nell'ordine:**
- **a** Connection-driver-statement-resulSet
- **b** Driver-statement-connection-resulSet
- **c** Driver-connection-statement-resulSet
- **d** Driver-connection-statement-resulSet
- **e** Connection-statement-driver-resulSet

**5 Le connessioni JDBC occupano risorse, in un ambiente multiutente e multitasking è opportuno:**
- **a** aprire una connessione solo quando necessario
- **b** assicurarsi di chiuderla al termine del suo utilizzo
- **c** aprire una connessione alla volta
- **d** non aprire/chiudere connessioni inutilmente

**6 Quale tra i seguenti è un percorso corretto per un driver JDBC-ODBC?**
- **a** sun.odbc.Jdbc.JdbcOdbcDriver
- **b** sun.jdbc.odbc.OdbcJdbcDriver
- **c** sun.jdbc.odbc.JdbcOdbcDriver
- **d** sun.jdbc.odbc.OdbcJdbcDriver

## VERO/FALSO

1 JDBC racchiude una serie di classi per poter accedere alle basi di dati. V F
2 La Sun ha incluso JDBC nel JDK a partire dalla versione 2.1. V F
3 Il codice ODBC binario e il codice del DBMS client devono essere caricati su ogni macchina client che usa JDBC-ODBC bridge. V F
4 Il Bridge JDBC-ODBC rispetta la filosofia Wora. V F
5 Le Bridge API Native rispettano la filosofia Wora. V F
6 La tecnica più corretta per l'utilizzo di Connector/J è quella di copiarla all'interno della directory jre/lib/ext, nella cartella che contiene l'installazione di Java. V F
7 Se la connessione va a buon fine, viene creato un oggetto java.sql.Statement. V F
8 Le query SQL memorizzano il risultato in un oggetto java.sql.ResultSet. V F
9 Con Access non è possibile evitare la definizione di un DSN per puntare a file MDB. V F

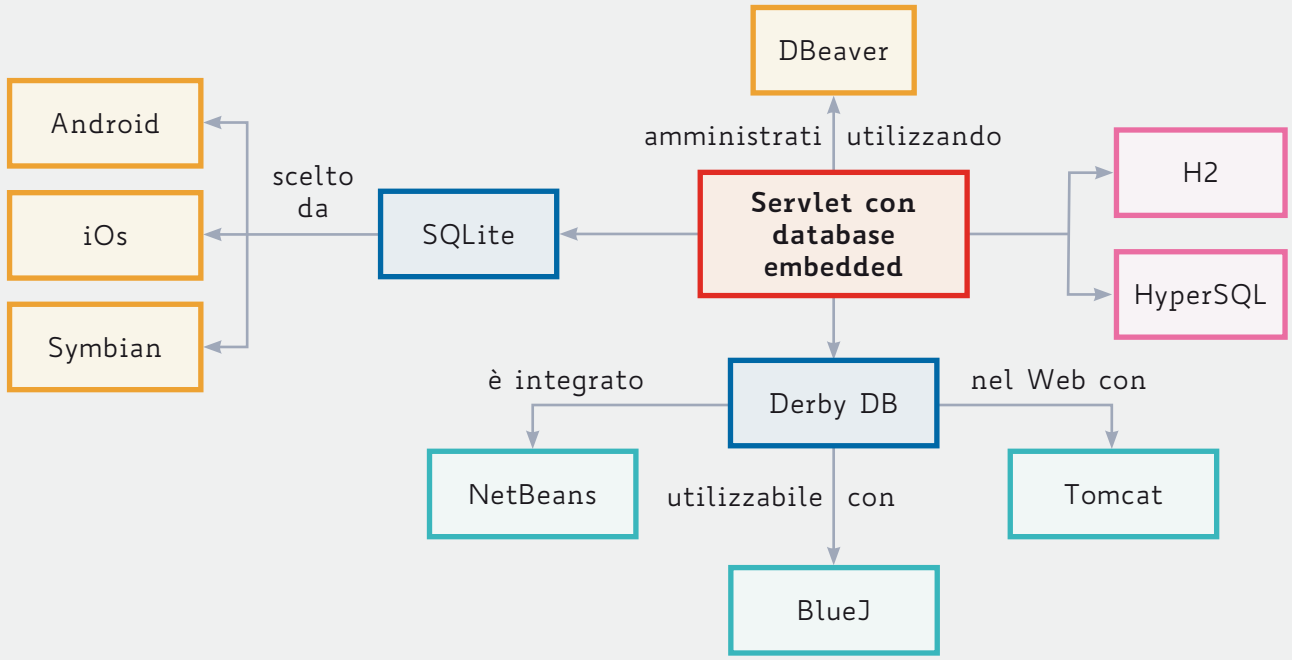
# 3 Servlet con database embedded

## IN QUESTA LEZIONE IMPAREREMO...

- i database embedded
- la connessione ai database embedded
- l'utilizzo di DBeaver per l'amministrazione dei database
- connessioni a Derby DB in locale e in remoto

## MAPPA CONCETTUALE

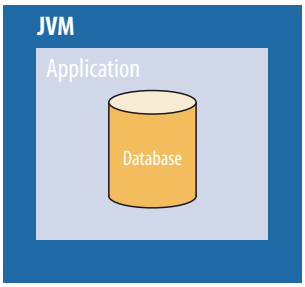
didattica inclusiva




## I database embedded

**Database incorporati**
I database incorporati sono integrati all'interno dell'applicazione, non sono pertanto sistemi di database autonomi. I DB incorporati vengono spesso utilizzati durante la fase di sviluppo o test, rendendo più facile la configurazione dei dispositivi mobili o degli ambienti IoT.

In molte situazioni vi è l'esigenza di dover utilizzare sistemi di gestione di DB snelli, di dimensioni ridotte e strettamente legati all'applicazione in uso, come per esempio nelle applicazioni per dispositivi mobili; infatti non sempre è necessario avere a disposizione sistemi complessi in grado di gestire migliaia di utenti, con terabyte di dati. I **database incorporati** (o embedded) possono essere la soluzione ottimale quando sono richieste elevate prestazioni e dimensioni ridotte.

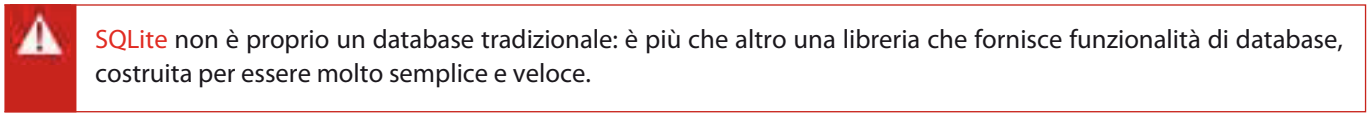


## I database embedded per Java

Presentiamo ora sinteticamente i sistemi di database più utilizzati, e infine quello che scelto per essere applicato negli esempi.

### SQLite

SQLite è il database scelto da quasi tutti i sistemi operativi mobili: Android, iOS e Symbian vengono forniti con il DB SQLite incorporato e già ottimizzato per il processore presente nei dispositivi mobili.
È un database relazionale a tutti gli effetti, basato sul linguaggio SQL, ma ottimizzato per l'utilizzo in ambienti con poche risorse (telefoni cellulari o piccole applicazioni): le query in SQLite sono quasi identiche ad altre chiamate SQL.



SQLite non è proprio un database tradizionale: è più che altro una libreria che fornisce funzionalità di database, costruita per essere molto semplice e veloce.

SQLite realizza la base di dati incorporata in un unico file, così come Access, e non è un processo standalone, ma viene incorporato all'interno di un altro programma.
La particolare struttura architetturale di Java permette di utilizzare i database embedded non solo per i dispositivi mobili ma anche per le applicazioni client-server realizzate mediante servlet, in quanto il codice eseguibile risulta essere fisicamente "incorporato" con le primitive che accedono al database.
Comunque per le applicazioni di rete SQLite non è "la migliore soluzione" ed esistono molte alternative: a tal proposito riportiamo un elenco di domande che ci si deve porre prima di decidere quale DB utilizzare per la nostra applicazione.

- *È adatto a essere impiegato per alte prestazioni (elevato numero di utenti e dati da gestire in contemporanea)?*
- *È in grado di* **scalare** *per soddisfare le richieste di crescita dell'applicazione?*
- *Il sistema di database prevede un apparato di protezione dei dati per il disaster recovery?*
- *I dati possono essere protetti in ambienti distribuiti?*
- *Il sistema di database è sicuro durante la comunicazione con altri dispositivi?*
- *Quali sono i requisiti richiesti in termini di risorse di sistema?*
- *Il sistema di database offre una gamma di opzioni di connettività?*
- *Il sistema di database ha il supporto Unicode per distribuzioni internazionali?*
- *Quanto è facile identificare e diagnosticare problemi di prestazioni?*
- *Quali sono i costi e le opzioni di licenza?*
- *Quale livello di supporto è disponibile dal fornitore del DB?*

**Scalare**
La scalarità di un database è la caratteristica che lo rende modificabile in maniera agevole quando la quantità (mole) e la tipologia dei dati trattati varia notevolmente.

### HSQLDB (HyperSQL Database)

Il database HyperSQL è un RDBMS (Relational DataBase Management System), un DBMS basato sul modello relazionale scritto interamente in Java puro, supporta JDK 5, 6, 7 e 8 in HyperSQL 2.x, è gratuito e distribuito con una licenza simile alla **BSD**, compatibile con le principali licenze open source.
HyperSQL è conforme allo standard SQL e alle specifiche JDBC e supporta tutte le funzionalità di base previste da un moderno database relazionale e può essere eseguito in modalità integrata o server.
La documentazione di HyperSQL è disponibile all'indirizzo http://hsqldb.org/web/hsqlDocsFrame.html dove è anche possibile effettuare il download: il pacchetto include un file .jar che contiene i componenti richiesti come HyperSQL RDBMS Engine e il driver JDBC per incorporare HyperSQL in un'applicazione Java.

**BSD**
La licenza BSD è una delle più semplici e permissive: richiede solo di essere inclusa sempre nel software per poterlo utilizzare.

La connessione è molto simile a quella vista in MySQL, in quanto utilizza anch'essa API JDBC:

```
Class.forName("org.hsqldb.jdbcDriver");
Connection miaConnessione = DriverManager.getConnection("jdbc:hsqldb:testdb", <user>, <psw>);
```

## H2

H2 è un database Java open source molto leggero: può essere incorporato in applicazioni Java o eseguito in modalità client-server e può essere anche utilizzato come "database temporaneo", cioè che viene eseguito in memoria, senza che i dati persistano sul disco.
H2 è uno dei migliori prodotti SQL di prossima generazione: open source, multimodello e supporta l'API JDBC; la sua home page ne riporta le caratteristiche confrontandole con altri database di largo utilizzo.

| | H2 | Derby | HSQLDB | MySQL | PostgreSQL |
|---|---|---|---|---|---|
| Pure Java | Yes | Yes | Yes | No | No |
| Memory Mode | Yes | Yes | Yes | No | No |
| Encrypted Database | Yes | Yes | Yes | No | No |
| ODBC Driver | Yes | No | No | Yes | Yes |
| Fulltext Search | Yes | No | No | Yes | Yes |
| Multi Version Concurrency | Yes | No | Yes | Yes | Yes |
| Footprint (embedded) | ~2 MB | ~3 MB | ~1.5 MB | — | — |
| Footprint (client) | ~500 KB | ~600 KB | ~1.5 MB | ~1 MB | ~700 KB |

La documentazione ufficiale è disponibile all'indirizzo http://www.h2database.com/ dove è anche possibile effettuarne il download; è anche disponibile una pagina performance.html dove è presente un benchmark che permette di confrontare le prestazioni di H2 sia con i suoi concorrenti per applicazioni embedded HSQLDB e Derby sia con i database utilizzati in ambiente client-server.
La connessione è molto simile a quella vista in MySQL, in quanto utilizza anch'essa API JDBC:

```
Class.forName("org.h2.Driver");
Connection miaConnessione = DriverManager.getConnection("jdbc:h2:miaDir/testdb", <user>,<psw>);
```

## Apache Derby

Apache Derby è un database relazionale open source implementato interamente in Java e disponibile con licenza Apache, versione 2.0. Abbiamo deciso di utilizzarlo in questo volume per la sua completezza ed estrema semplicità d'uso. Essendo integrato con NetBeans può essere usato per lo sviluppo e il test delle applicazioni, decidendo solo successivamente, in fase di distribuzione, se installarlo nella versione server in quanto supporta il meccanismo client-server e ha incorporate tutte le funzionalità standard di un database relazionale.
La connessione è molto simile a quella vista in MySQL, in quanto utilizza anch'essa API JDBC:

```
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
Connection miaConnessione = DriverManager.getConnection("jdbc:derby:testdb;create=true");
```

L'opzione create=true è particolarmente comoda per effettuare la connessione e la creazione contestuale del database, solo qualora non esista già.

Nei sistemi embedded dobbiamo sempre ricordarci di chiudere la connessione al database prima di uscire dall'applicazione.

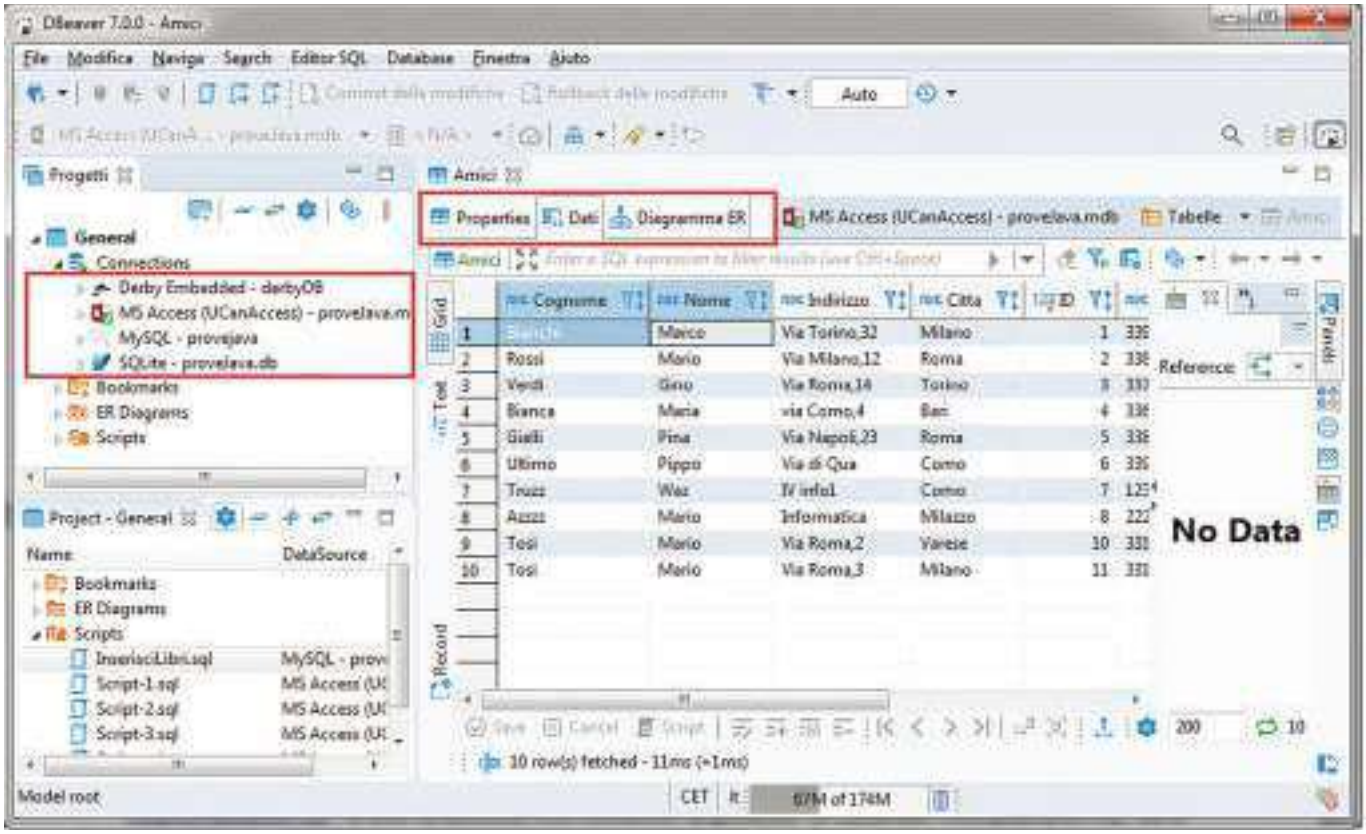
A differenza delle versioni precedenti, a partire da Java 9, Apache Derby (e quindi JavaDB) non è più incluso in JDK: è possibile effettuare il download dal sito https://db.apache.org/derby/.

## Uno strumento comune per l'amministrazione dei database

Il principale problema riscontrato nella gestione dei database embedded è quella della scarsa maneggevolezza, non tutti gli ambienti permettono di effettuare le operazioni di DDL e di controllo dei dati.
Un programma completo, gratuito e "multi database" è **DBeaver**, scaricabile da https://dbeaver.io/: è uno strumento che consente la gestione di database multipiattaforma, ed è largamente utilizzato da sviluppatori, programmatori SQL, amministratori di database e analisti.

**DBeaver**
DBeaver supporta tutti i database più diffusi: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Firebird, Presto, Derby ecc.

Come possiamo osservare dalla schermata che segue, noi lo abbiamo utilizzato sia per Derby sia per Access, MySQL e SQLite, e abbiamo riscontrato che offre tutte le opzioni presenti nel più conosciuto HeidiSQL, che però riduce le sua operatività a MySQL e i suoi fork, nonché per Microsoft SQL Server e PostgreSQL.




È anche disponibile una versione enterprise edition a pagamento che permette di gestire i database NoSQL/Big-Data come MongoDB, Amazon DynamoDB, Google Bigtable, InfluxDB, Apache Cassandra, Redis e Apache Hive.

Tra le possibili funzionalità che offre DBeaver è doveroso sottolineare quella che permette di trasferire interi database da un DBSM a un altro, importando automaticamente con pochi semplici passaggi sia la struttura delle tabelle (gli schemi) sia i dati in esse contenuti.

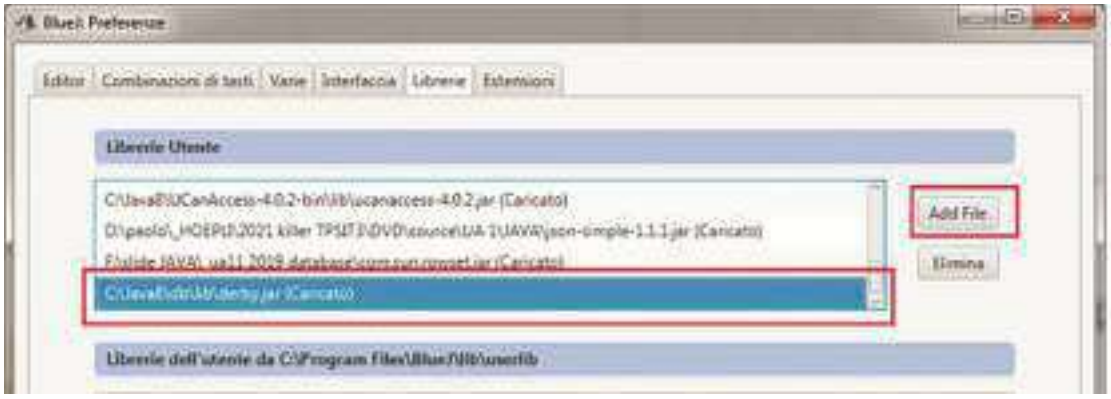
**AREA DIGITALE**
**Procedura per la migrazione tra due DMBS differenti.**

## Utilizzare Derby

Prima di descrivere un semplice programma che utilizza Derby, vediamo come includere le librerie necessarie per l'utilizzo negli ambienti di sviluppo, quando non sono già presenti in forma nativa.

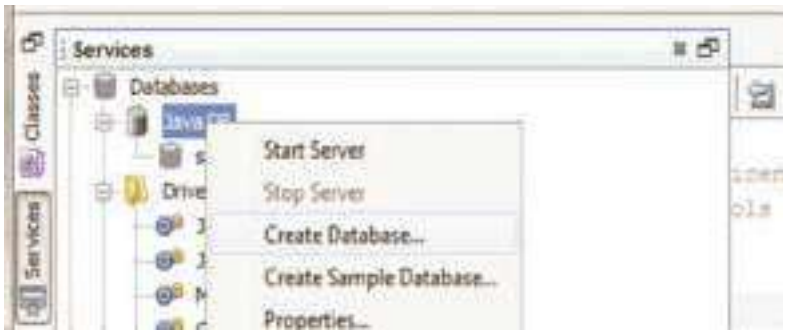
## Utilizzo con BlueJ

Per utilizzare Derby con BlueJ, come già visto per UCanAccess, è necessario importare le libreria derby.jar, che scaricheremo da https://db.apache.org/derby/ selezionando l'opzione Preferenze dal menu Strumenti.

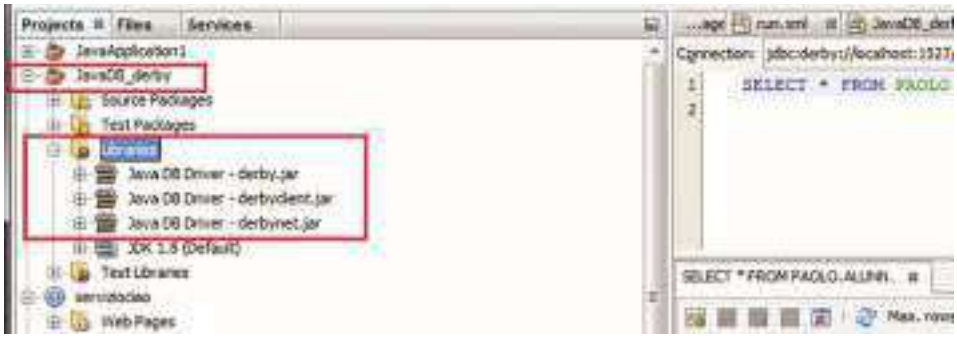



## Utilizzo con NetBeans

Derby è integrato in NetBeans ed è immediatamente utilizzabile; per creare il database e le rispettive tabelle è sufficiente posizionarsi nella scheda Services e selezionare l'opzione Create Database… dal menu che appare cliccando con il tasto destro sull'opzione JavaDB.



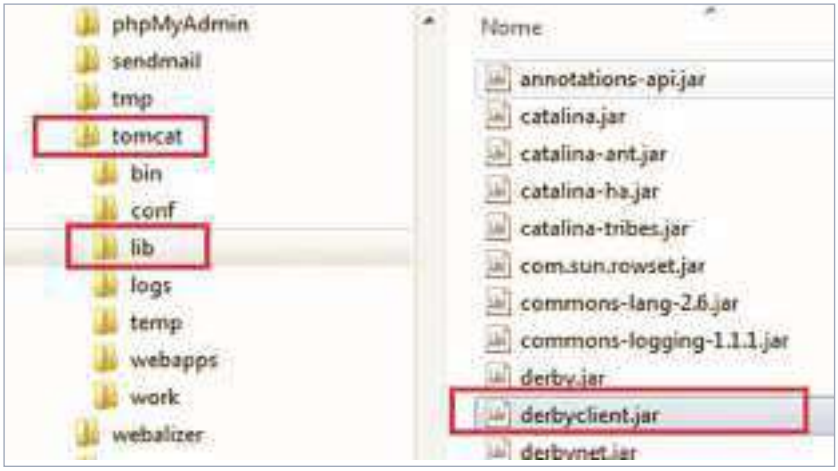

Se durante l'installazione, in NetBeans non vengono scaricati i driver necessari per l'utilizzo di Derby, è possibile aggiungerli in un secondo tempo cliccando semplicemente con il tasto destro sulla cartella Libraries e scegliere Aggiungi JAR / Cartella ..., quindi accedere al file .jar e importare i tre componenti mostrati nella seguente figura:



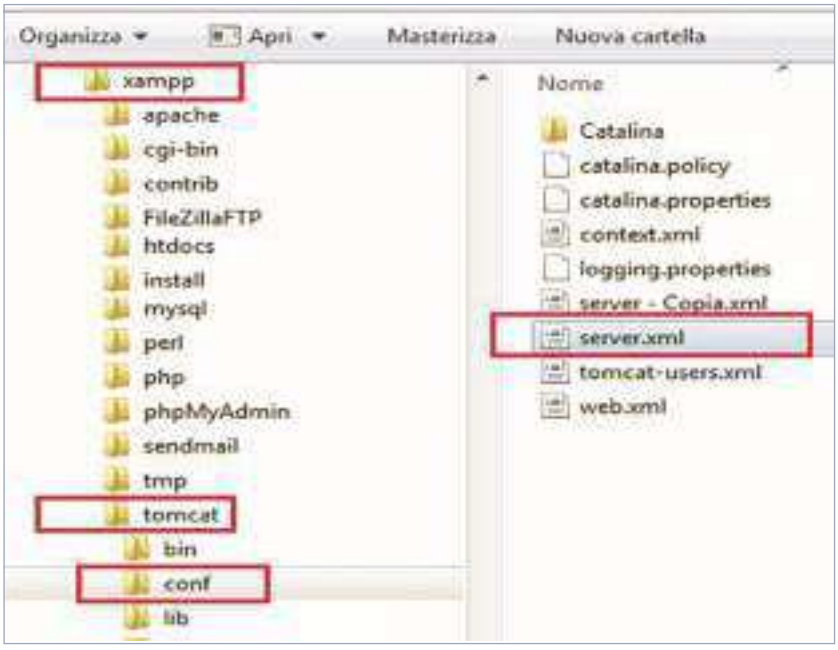

## Utilizzo in remoto: configurazione di Derby nel Web server

Per configurare Derby con il Web server Tomcat si procede con la seguente sequenza di tre passaggi:

1 effettuare il download della libreria dal sito http://db.apache.org/derby/;
2 posizionare i file derby.jar e derbyclient.jar nella cartella ${TOMCAT_HOME}/lib, come mostrato nella figura di pagina seguente:




**3** editare il file server.xml presente nella cartella conf di tomcat, come mostrato nella figura che segue:




È necessario aggiungere nel tag <GlobalNamingResources> il riferimento al nuovo database, e cioè il nome del driver, il suo percorso e le credenziali per accedervi:

```
-->
<GlobalNamingResources>
  <!-- Editable user database that can also be used by
     UserDatabaseRealm to authenticate users
  -->
  <Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />

<Resource auth="Container"
      driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
      maxActive="8" maxIdle="4"
      name="jdbc/javaDB" type="javax.sql.DataSource"
      url="jdbc:derby:esempiTPSIT3;create=true"
      username="paolo" password="123" />

</GlobalNamingResources>
```

## Utilizzare il database Derby in locale

Descriveremo come utilizzare Derby con le servlet negli Esercizi in Laboratorio che seguono: ora vediamo semplicemente come creare un database, scrivere dei dati e visualizzarli in un programma "locale".
Realizziamo una semplice classe che apre una tabella in un database Derby (provaDB) e se questo non è presente lo crea, a tal fine aggiungiamo il parametro create=true prima descritto: quindi visualizziamo i dati presenti nella tabella UTENTI.
Scriviamo direttamente il codice nel main(): come prime istruzioni definiamo le stringhe con il nome del driver e il protocollo e carichiamo il driver appena definito.

```
import java.sql.*;
public class CreaDerby {
  public static void main(String[] args) throws Exception {
    // parametri per la connessione al database Derby
    String driver       = "org.apache.derby.jdbc.EmbeddedDriver";    // stand alone
    String protocollo   = "jdbc:derby:";                             // connessione alla libreria
    String miaQuery;
    // Step 1: caricamento e registrazione Derby driver class
    System.out.println("Registrazione del driver: " + protocollo);
    try {
      Class.forName(driver);
    }
    catch(ClassNotFoundException excep) {
      System.out.println("----- Errore: problemi nel caricamento del driver " + driver);
      excep.printStackTrace();
      System.exit(1);
    }
```

Se l'operazione va a buon fine, procediamo con la connessione al database provaDB: in caso contrario, se il DB non è non presente, viene generata l'eccezione e nel codice di gestione della stessa inseriamo la query per la sua creazione con l'opzione create=true e per la creazione della tabella UTENTI, nella quale inseriremo il nostro dato:

```
    // Step 2: connessione al database DB o sua creazione
    java.sql.Connection con = null;
    try {
      con = DriverManager.getConnection("jdbc:derby:provaDB");
    }
    catch (java.sql.SQLException sqle) {
      con = DriverManager.getConnection("jdbc:derby:provaDB;create=true");
      miaQuery = "CREATE TABLE UTENTI (userID INT NOT NULL, mail VARCHAR(100),password VARCHAR(20))";
      con.createStatement().executeUpdate(miaQuery);
    }
```

Quindi predisponiamo la query parametrica di inserimento, definiamo il contenuto dei tre attributi e la mandiamo in esecuzione.

```
    // Step 3: inserimento dati
    miaQuery = "INSERT INTO UTENTI (userID,mail,password) VALUES(?,?,?)";
    PreparedStatement stm = con.prepareStatement(miaQuery);
    stm.setInt(1, 1);
    stm.setString(2, "ciro@vesuvio.org");
    stm.setString(3, "ciro@CMNmm#");
    stm.executeUpdate();
```

Visualizziamo il contenuto della tabella e chiudiamo il database, dato che stiamo operando in locale:

```
    // Step 4: visualizzazione dati presenti nel database
    miaQuery = "select * from UTENTI";
    ResultSet rs = con.createStatement().executeQuery(miaQuery);
    while (rs.next()) {
      System.out.println( " id:" + rs.getInt(1)+"   e-mail: " + rs.getString(2));
    }
    try {                          // è una versione embedded: va chiusa quando l'applicazione termina
      DriverManager.getConnection("jdbc:derby:provaDB;shutdown=true");
    } catch (SQLException se) {    // stato = 08006 chiusura OK, altrimenti si è verificato un errore
      if (!se.getSQLState().equals("08006")) {
        throw se;
      }
    }
  }
}
```

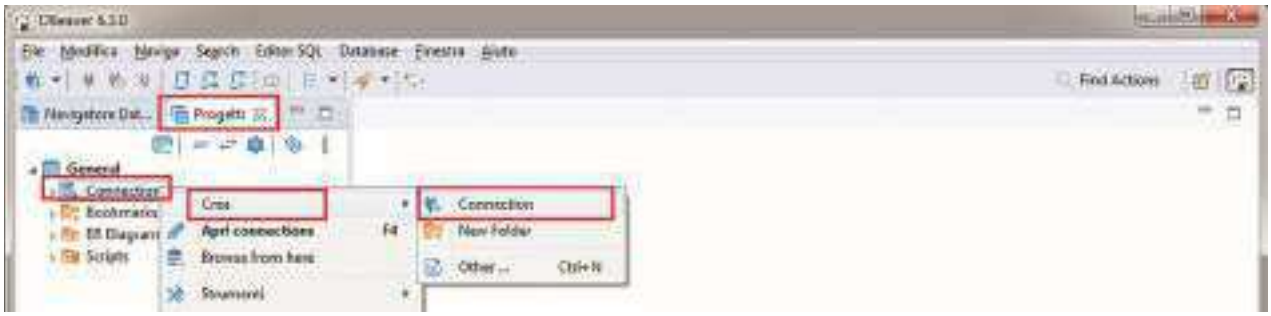
```
Registrazione del driver: jdbc:derby:
 id:1   e-mail: ciro@vesuvio.org
 id:1   e-mail: ciro@vesuvio.org
 id:1   e-mail: ciro@vesuvio.org
```

Dopo tre esecuzioni nel DB sono presenti tre record uguali:

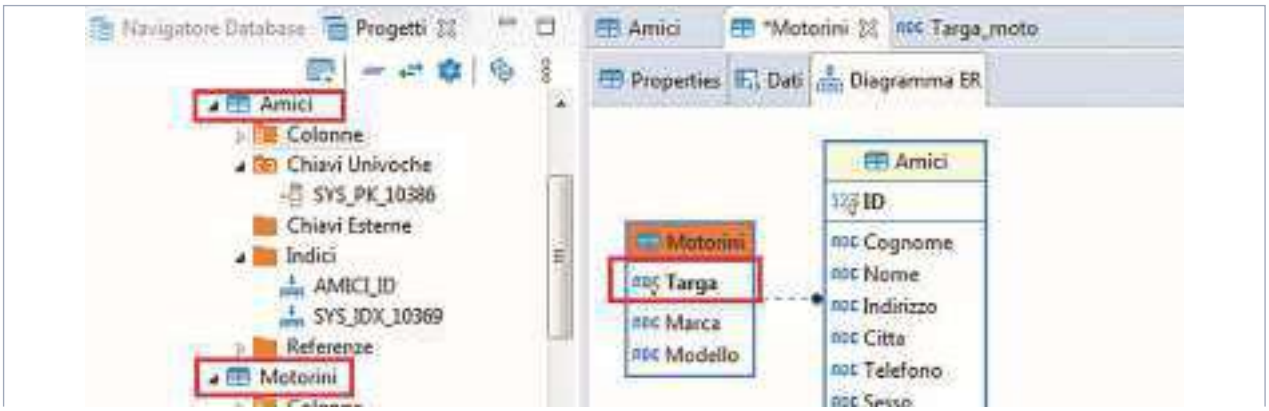
# VERIFICA... le competenze

## ESERCIZI

**1** Dall'indirizzo https://dbeaver.io/ scarica il programma DBeaver, quindi crea una nuova connessione a tutti i tipi di database che conosci.




Successivamente crea un database Derby con il nome TPSIT3 che utilizzerai per le prove e/o le esercitazioni. All'interno di questo database, crea due tabelle in modo che sia presenti tra loro una relazione, come per esempio:




Quindi nel diagramma ER connetti la chiave primaria Targa della tabella Motorini con la chiave esterna Targa_moto presente nella tabella Amici: popola la tabella con alcuni dati e "scopri" come è possibile "navigare" all'interno dei dati passando tramite le relazioni in DBeaver.

**2** Configura il tuo Web server Tomcat procedendo con la seguente sequenza di operazioni:
1) scarica la libreria dal sito web http://db.apache.org/derby/;
2) posiziona i file derby.jar e derbyclient.jar nella cartella xampp/tomcat/lib;
3) edita il file server.xml presente nella cartella conf di tomcat:

```
<Resource auth="Container"
        driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
        maxActive="8" maxIdle="4"
        name="jdbc/javaDB" type="javax.sql.DataSource"
        url="jdbc:derby:esempiTPSIT3;create=true"
        username="admin" password="" />
```

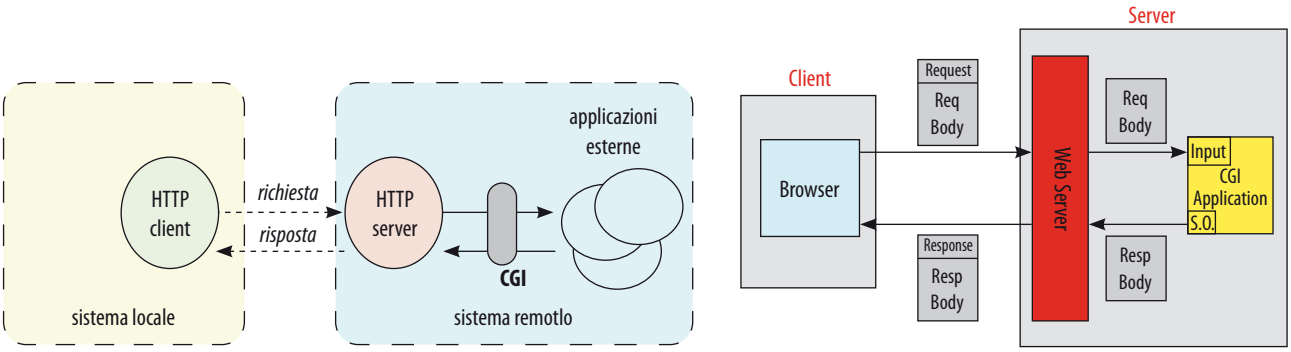
Modifica il programma scritto per l'inserimento nei dati nella tabella UTENTI verificando la presenza di più record con lo stesso ID e, se presenti, cancellando le copie.
Quindi predisponi un'interfaccia grafica dove l'utente inserisce i dati e li aggiunge a quelli già presenti in archivio.
Verifica la tabella realizzata utilizzando DBeaver.

# ESERCIZI IN LABORATORIO 1

## APPLICAZIONI CGI

### Applicazioni lato server: le CGI

I programmi CGI (Common Gateway Interface) introducono la possibilità di avere una risposta con informazioni dinamiche grazie all'esecuzione sul server di programmi che vengono lanciati come conseguenza di una specifica richiesta effettuata da una pagina HTML del client.
In passato i programmi CGI dovevano essere memorizzati in una specifica directory del server chiamata cgi-bin e collocata nella sua root: ora questa ristrettezza è decaduta e, quindi, è possibile memorizzare tali file in una directory a piacere; nei nostri esempi utilizzeremo la directory esempiCGI come sottodirectory della cartella hoepli.
Riportiamo lo schema di funzionamento già descritto nella lezione teorica:




Il nostro compito ora è quello di realizzare:
**a** come sistema locale una pagina HTML che effettui la chiamata del programma CGI;
**b** come sistema remoto un programma che esegua qualche elaborazione, predisponga una pagina HTML con i risultati e la invii come risposta al client.

### Prima CGI

Come prima applicazione scriviamo un esempio dove il client richiama semplicemente il programma CGI senza passargli alcun parametro: realizziamo un contatore degli accessi.
Il codice del client è molto semplice:

```
<HTML>
<HEAD>
<TITLE>Contatore accessi CGI</TITLE>
</HEAD>
<BODY>
<form method="POST" action="http://localhost/hoepli/esempiCGI/contaAccessi.cgi">
  <p>Clicca per sapere il tuo numero utente:</p>
  <p>
    <input type="submit" value="Invia">
  </p>
</form>
</BODY>
</HTML>
```

Scegliamo di scrivere in linguaggio C il codice del programma CGI che sarà eseguito sul server: possiamo vedere come all'interno del programma viene prodotto un documento HTML sullo standard output, anticipato dalla linea che contiene l'header "Content-Type", che è obbligatorio

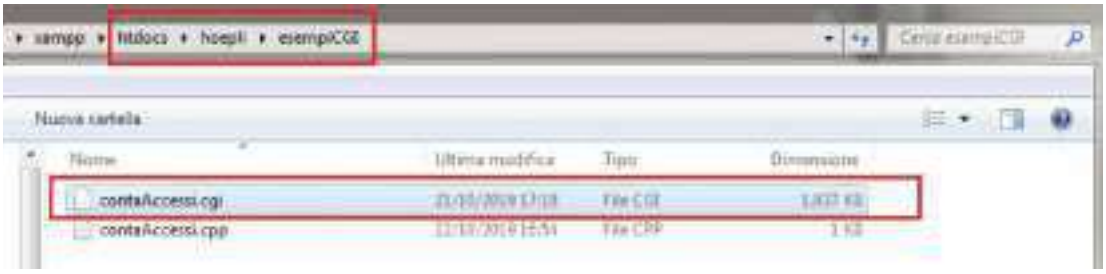
```
Content-Type: text/html
```

# 1 ESERCIZI IN LABORATORIO

Quindi vengono prodotti i tag html di apertura di una normale pagina HTML: tra i tag viene inserita una sezione di codice che esegue le operazioni, cioè apre il file, aggiorna il contatore e lo salva; successivamente inserisce tale valore in un tag della pagina che si sta "costruendo".

```
contaAccessi.cpp
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main(){
5    cout << "Content-Type: text/html" << endl;
6    cout << endl;                          // indica la fine dell'header HTTP
7    cout << "<HTML>" << endl;
8    cout << "<HEAD>" << endl;
9    cout << " <TITLE>Applicazioni CGI </TITLE>" << endl;
10   cout << "</HEAD>" << endl;
11   cout << "<BODY>" << endl;
12   cout << " <H1>Benvenuto al corso di TPSIT-3 - classe 5</H1>" << endl;
13   cout << " <HR>" << endl;
14   // legge i dati nel file
15   ifstream file1("counter.txt");     // dichiaro la variabile per il file lettura
16   int conta = 0;
17   if (file1)
18     file1 >> conta;
19   conta++;
20   file1.close();
21   // aggiorna e scrive i dati nel file
22   ofstream file2("counter.txt");     // dichiaro la variabile per il file scrittura
23   file2 << conta;
24   file2.close();
25   // visualizza il contatore
26   cout << "<H3>Sei l'utente numero: " << conta << "</H3>" << endl;
27   cout << "</BODY>" << endl;
28   cout << "</HTML>" << endl;
29   return 0;
30 }
```

Dopo aver compilato il programma, modifichiamo il nome del suffisso da .exe a .cgi, e lo memorizziamo nella cartella che abbiamo predisposto nel server Apache:




Sul client carichiamo la pagina contaAccessi.html:

Clicchiamo sul pulsante Invia ottenendo come risultato la schermata di pagina seguente:

# ESERCIZI IN LABORATORIO 1

## CGI con passaggio dati

Vediamo un secondo esempio dove realizziamo il caso più generale nel quale la pagina client invia uno (o più) parametri al programma CGI; sostanzialmente abbiamo tre modalità per acquisire i dati:

**1** prelevandoli dalle variabili d'ambiente se il client usa GET con parametri su QUERY_STRING;
**2** leggendoli dallo stdin se il client usa il metodo POST di Http;
**3** da command line con ISINDEX (più raro).

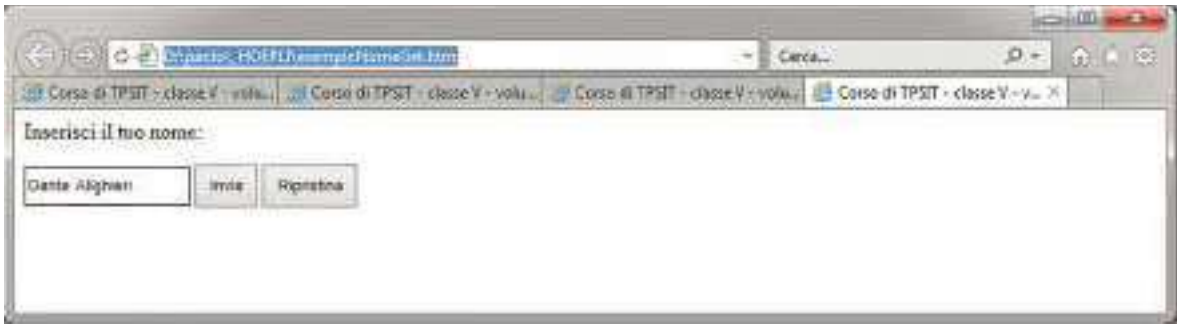
Vediamo i primi due casi in un esempio dove la pagina HTML permette di introdurre un nome in una casella di testo e la pagina generata dal programma CGI lo visualizza indicando la modalità con la quale lo ha ricevuto.

### Passaggi dati con GET

L'utilizzo di GET viene ottenuto sostituendo una singola istruzione nella pagina HTML (istruzione 6):

```
1  <HTML>
2  <HEAD>
3  <TITLE>Corso di TPSIT - classe V - volume 3</TITLE>
4  </HEAD>
5  <BODY>
6  <FORM method="GET" action="http://localhost/homp11/esempiCGI/esempioName.cgi">
7    <p>Inserisci il tuo nome:</p>
8    <p>
9      <input type="text" name="EditBox" width="20">
10     <input type="submit" value="Invia">
11     <input type="reset" value="Ripristina">
12   </p>
13 </FORM>
14 </BODY>
15 </HTML>
```

Il browser prepara una Http request il cui BODY contiene i dati (opportunamente formattati) inseriti nei campi di input e la manda al server.
Avviamo la pagina nel browser:



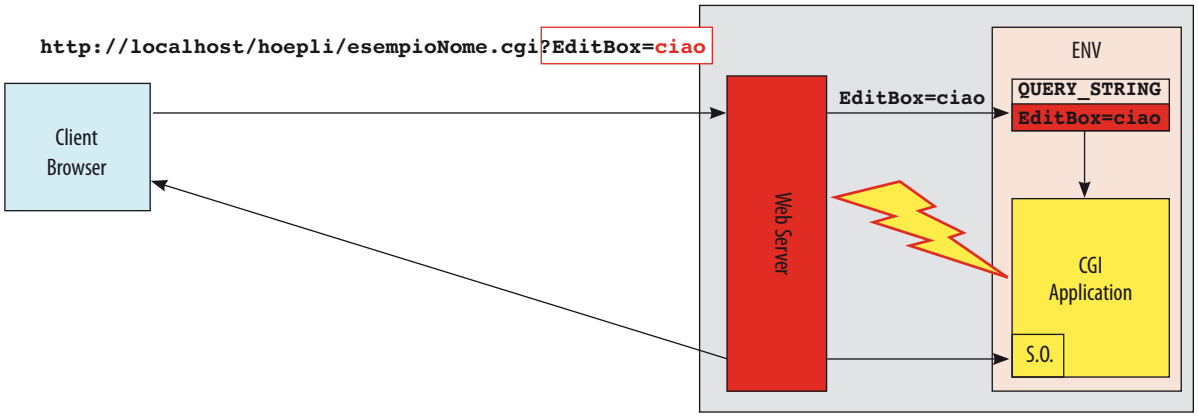

Inseriamo un nome nella casella di testo, per esempio Dante Alighieri, e clicchiamo il pulsante di Invia che effettua il commit.

# 1 ESERCIZI IN LABORATORIO

Scriviamo il codice in linguaggio C dove prevediamo di acquisire il dato sia leggendolo dalla stringa di query utilizzando l'apposita funzione getenv() (metodo GET), sia leggendolo dallo standard input (metodo POST):

```
#include <iostream>
#include <string>
#include <stdlib.h>                // getenv
using namespace std;
int main(){
  string parametro;
  if (!cin.eof())
    cin >> parametro;              // legge la prima riga dei dati di ingresso
  cout << "Content-Type: text/html" << endl;
  cout << endl;
  cout << "<HTML>" << endl;
  cout << "<HEAD>" << endl;
  cout << " <TITLE>Corso di TPSIT - classe V </TITLE>" << endl;
  cout << "</HEAD>" << endl;
  cout << "<BODY>" << endl;
  cout << " <H1>QUERY_STRING: " << getenv("QUERY_STRING") << "</H1>" << endl;
  cout << " <H1>Standard Input: " << parametro << "</H1>" << endl;
  cout << "</BODY>" << endl;
  cout << "</HTML>" << endl;
  return 0;
}
```

Il browser in questo caso, alla pressione del tasto submit, costruisce una URL che punta alla CGI e che contiene anche tutti i dati da inviarle: subito dopo invoca la URL proprio come se l'avesse digitata l'utente a mano.




Quando il server riceve la URL così composta, crea una variabile d'ambiente "QUERY_STRING", prende la parte di URL contenente la richiesta e la assegna alla variabile: subito dopo invoca la CGI la quale, interrogando la variabile, verrà a conoscenza dei dati inviati dal client.

I programmi CGI hanno accesso anche alle altre variabili d'ambiente del protocollo Http, come quelle riportate nella seguente tabella:

# ESERCIZI IN LABORATORIO 1

| NOME | DESCRIZIONE | ESEMPIO |
|---|---|---|
| REQUEST_METHOD | Metodo usato dalla richiesta (p.e. GET o POST) | GET o POST |
| SCRIPT_NAME | Percorso relativo dell'applicazione CGI che viene eseguita (basato sull'indirizzo del server) | /cgi-bin/post_photo.tcl |
| QUERY_STRING | Segue il carattere "?" nell'URL | nome=pippo.jpg& expires=never |
| CONTENT_TYPE | Tipo MIME dei dati, quando il metodo di richiesta è POST | img/jpeg |
| CONTENT_LENGTH | Lunghezza in byte dei dati passati tramite standard input, quando il metodo di richiesta è POST | 17290 |

## Passaggi dati con POST

Modifichiamo un'istruzione nella pagina HTML del client in modo da passare i dati mediante POST:

```
5   <BODY>
6   <FORM method="POST" action="http://localhost/hoepli/esempiCGI/esempioNome.cgi">
7     <p>Inserisci il tuo nome:</p>
```

e la richiamiamo all'interno del browser:

Cliccando Invia si invoca l'esecuzione dello stesso programma esempioNome.cgi del caso precedente, ma gli si passa i dati attraverso lo standard Input (POST) e *non* attraverso la stringa di query, come possiamo vedere dall'output generato alla sua esecuzione:

## METTITI ALLA PROVA

1 Si vuole realizzare un'applicazione CGI che svolge il servizio di traduttore italiano-inglese: a tal fine si predispone un file di nome dizionario.txt contenente per ogni riga un termine italiano e la sua traduzione in inglese, separati dal carattere di tabulazione e si realizza un programma CGI che da una pagina Web accetta un termine inserito dall'utente in una text-box e restituisce la sua traduzione in inglese oppure un messaggio di termine non trovato.

2 Migliora il programma precedente introducendo un controllo in modo che il programma CGI venga richiamato solo se l'utente inserisce una parola con lunghezza di almeno 3 caratteri: in caso contrario, viene visualizzato un messaggio di warning.

4 Aggiungi anche l'opzione che permetta di effettuare la traduzione a rovescio, cioè da inglese a italiano, aggiungendo un pulsante nella pagina Web e modificando appositamente il codice CGI.

# 2 ESERCIZI IN LABORATORIO

## XAMPP E IL SERVER ENGINE TOMCAT

Per poter eseguire una servlet, il Web server deve avere il servlet container (o servlet engine), che fornisce l'ambiente di esecuzione a una servlet: nei nostri esempi noi utilizzeremo Tomcat, che è il servlet container open source più utilizzato interamente scritto in Java.

Mentre scriviamo queste esercitazioni è disponibile la versione 9.0 scaricabile direttamente all'indirizzo http://tomcat.apache.org/ oppure dalla cartella MATERIALI nella sezione del sito www.hoepliscuola.it dedicata a questo volume.

**AREA DIGITALE**

**Installazione di Tomcat su Apache**

## Pacchetti con Web server, motore PHP e DBMS server

Oltre ai Web server che abbiamo descritto nell'Unità di Apprendimento 1 di questo volume, sono disponibili ambienti "integrati" scaricabili gratuitamente da Internet che contengono tutti gli strumenti per la realizzazione e la gestione di applicazioni Web: per esempio contengono il Web server, un motore per il linguaggio di programmazione dinamico PHP, un Database e un Database Administrator (in alcune versioni persino Tomcat e Perl).

Vengono anche chiamati generalmente con il termine LAMP, acronimo di Linux, Apache, MySQL, PHP.

### Il pacchetto XAMPP

**XAMPP**

XAMPP è un acronimo dal seguente significato: la X sta per cross-platform, la A sta per Apache HTTP Server, la M sta per MySQL, la P sta per PHP e l'ultima P sta per Perl.

Tra i pacchetti più diffusi c'è il pacchetto **XAMPP** che è una piattaforma software gratuita che consente di installare in una sola volta il Web server Apache, il motore PHP, il server DBMS MySQL Admin e altri componenti, tra cui un server FTP (FileZilla) e nella versione Full anche Tomcat.

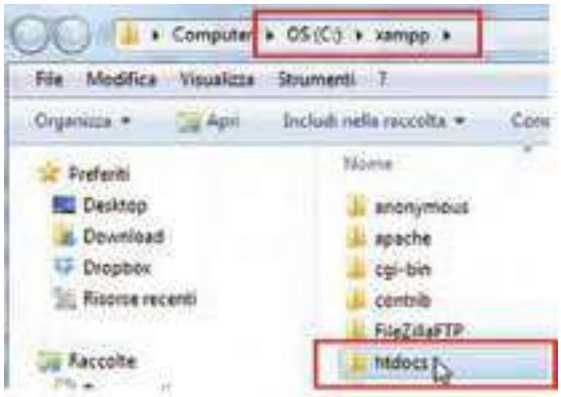
L'installazione è molto semplice e può essere effettuata direttamente dal sito https://www.apachefriends.org/it/download.html oppure partendo dall'indirizzo https://sourceforge.net/projects/xampp/files/latest/download e confermando tutto quanto viene proposto di default: contemporaneamente vengono installati e configurati tutti i suoi componenti.

Una volta installato il programma appare la tipica icona nella system tray di Windows:

Facendo clic su tale icona appare la finestra di monitor che consente di avviare o fermare i vari componenti. In questo caso abbiamo avviato Apache, e lo verifichiamo dallo sfondo verde e dalla label Stop sul pulsante delle Actions che ci permette di fermarne l'esecuzione.

# ESERCIZI IN LABORATORIO 2

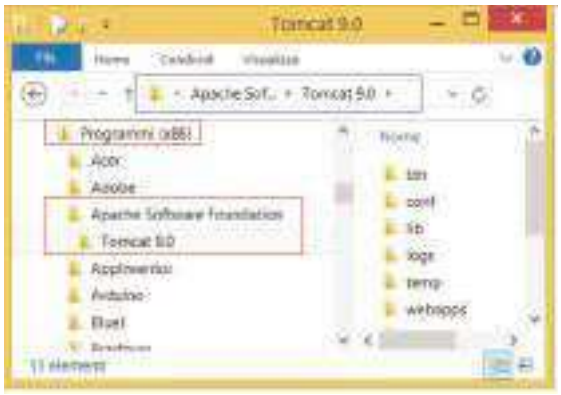
Il pulsante Explore mostra la directory in cui è stato installato XAMPP.
Vedremo in seguito che la directory htdocs sarà la document root, deve saranno memorizzati i file per le applicazioni scritte in linguaggio PHP.



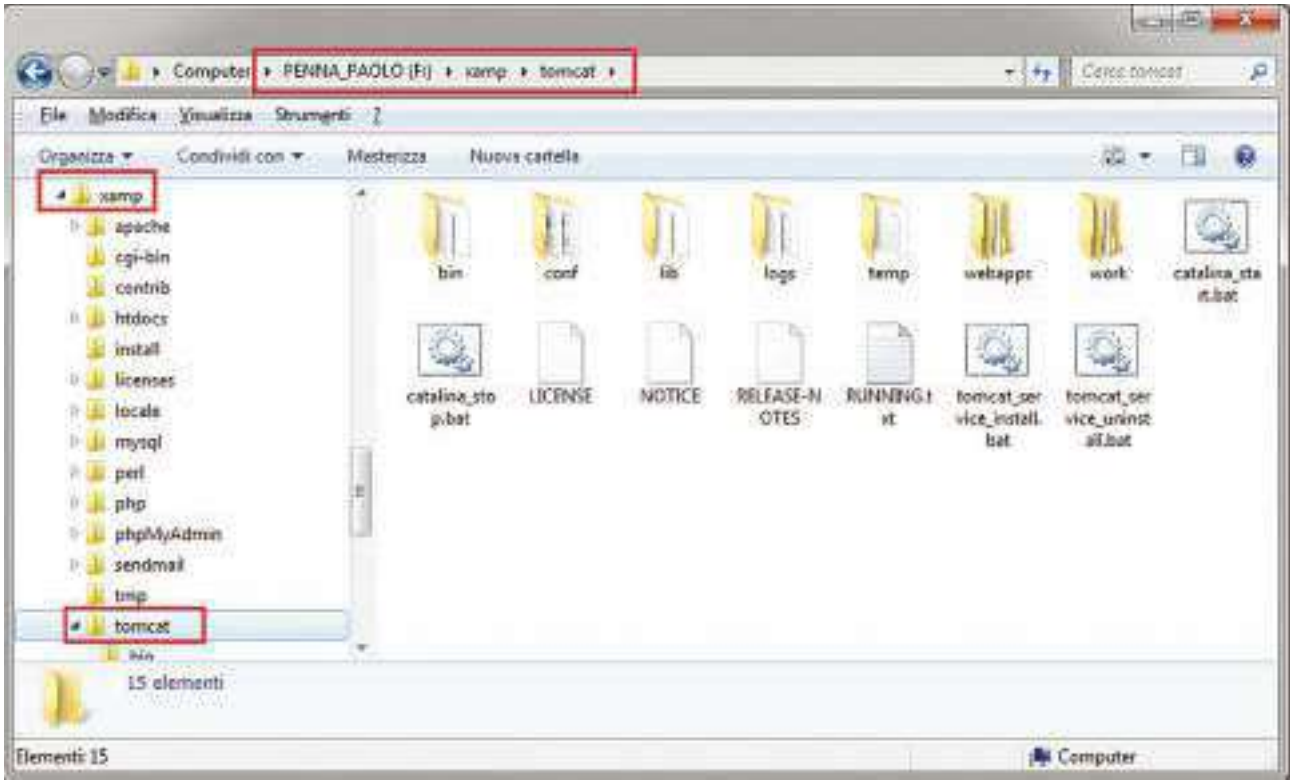

> È anche disponibile una comoda versione portable di XAMPP scaricabile all'indirizzo: http://portableapps.com/apps/development/xampp.

## Struttura di Tomcat

Se Tomcat è stato installato nella directory di default, cioè in una sottocartella della directory Programmi, siamo nella situazione mostrata nella figura a lato,



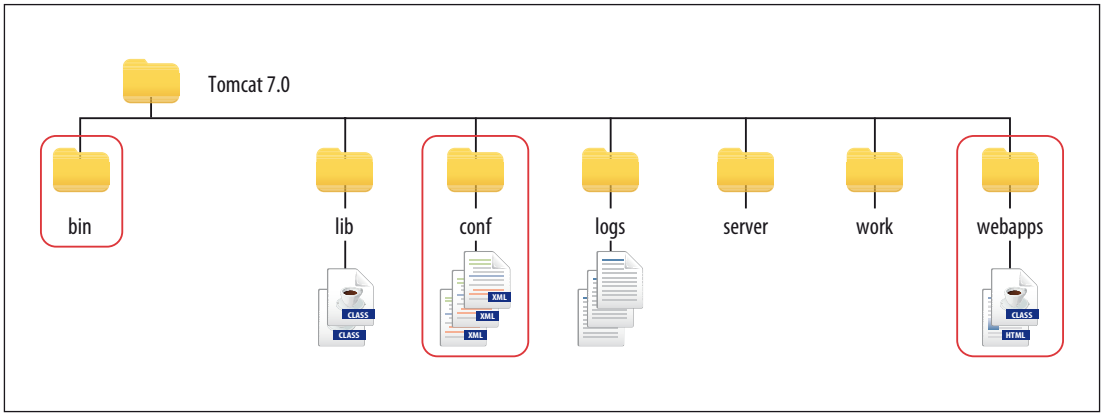

In alternativa, se abbiamo installato XAMPP sulla pen drive e abbiamo scelto come directory di installazione proprio quella che lo contiene, siamo nella seguente situazione:



# 2 ESERCIZI IN LABORATORIO

Nel seguito non faremo distinzioni tra le due possibili situazioni, cioè ambiente Tomcat installato su hard disk oppure su pen drive, in quanto hanno le medesime sottocartelle a partire dalla cartella Tomcat.

L'interno della directory Tomcat ha una struttura più complessa rispetto ad Apache: la riportiamo di seguito indicando per ciascuna sottocartella i file contenuti.




Le sottocartelle più significative sono:

- tomcat/bin è la cartella che contiene gli eseguibili per il funzionamento del Web server;
- tomcat/lib contiene le classi per la JVM di Jakarta/Tomcat;
- tomcat/conf contiene i file di configurazione di Tomcat;
- tomcat/webapps è la cartella dove saranno posizionate le applicazioni Web, ciascuna a partire da una specifica directory avente come nome proprio l'applicazione.

## Configurazione di Tomcat

La configurazione di Tomcat viene effettuata mediante file formato XML presenti nella cartella conf, tra i quali ricordiamo:

- server.xml contiene la configurazione globale dei server (specifica e configura i connector);
- context.xml permette di definire i parametri per avere dati persistenti per le servlet;
- web.xml contiene i dati per la configurazione delle servlet;
- tomcat-users.xml contiene le informazioni per il controllo d'accesso.

Non entriamo nel dettaglio di ciascuno di essi ma, a titolo di esempio, visualizziamo il contenuto del file tomcat-users.xml che può essere modificato per aggiungere gli utenti e assegnare i ruoli per l'accesso e la manutenzione del Web server.
I possibili ruoli sono sostanzialmente due, con diverse opzioni:

**1** admin
**2** manager
  – gui
  – script
  – status
  – jmx

Modifichiamo la configurazione di default e aggiungiamo un nuovo utente (noi abbiamo aggiunto un utente con password 123 e con username 123 e un utente paolo sempre con psw = 123).

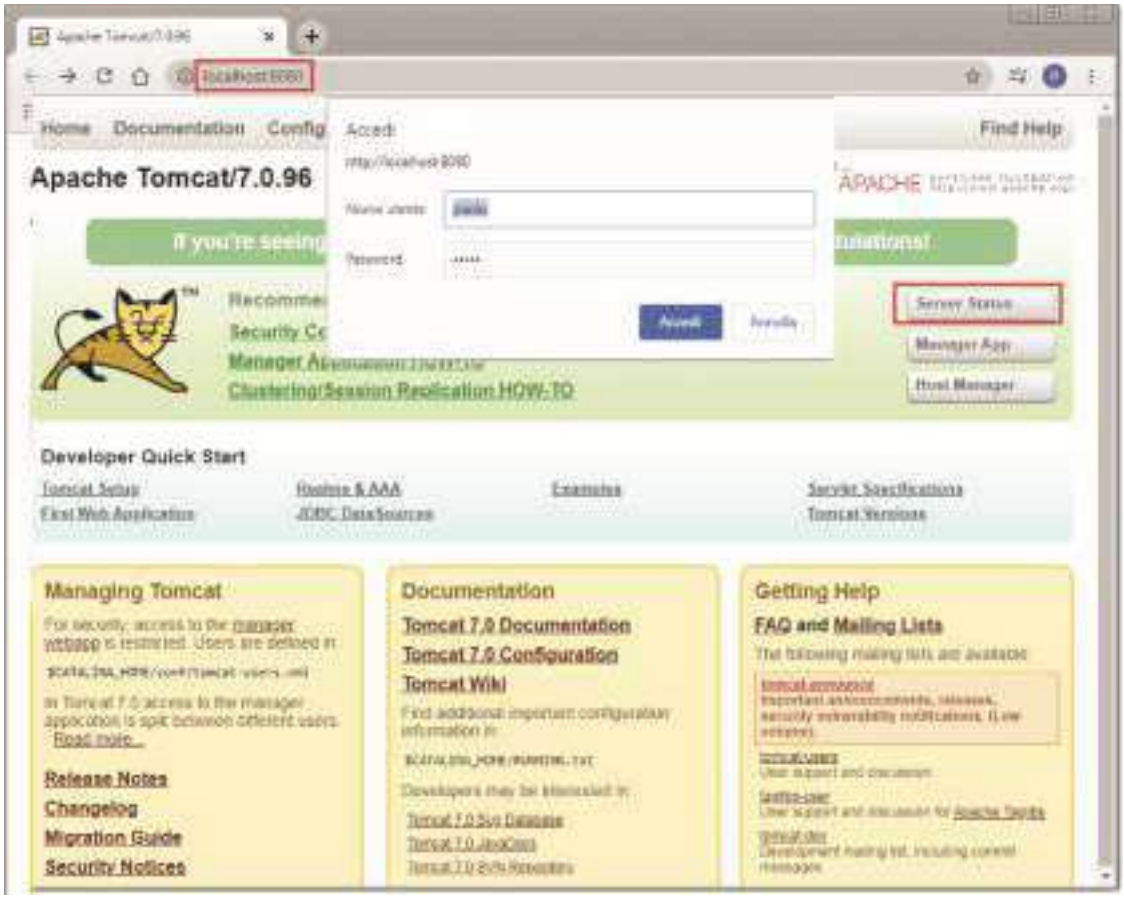
# ESERCIZI IN LABORATORIO 2

```
tomcat-users.xml
<tomcat-users>
<!--
  NOTE:  By default, no user is included in the "manager-gui" role required
  to operate the "/manager/html" web application.  If you wish to use this app,
  you must define such a user - the username and password are arbitrary. It is
  strongly recommended that you do NOT use one of the users in the commented out
  section below since they are intended for use with the examples web
  application.
-->
    <role rolename="manager-gui"/>
    <role rolename="manager-script"/>
    <role rolename="manager-status"/>
    <user password="123" roles="manager-gui,manager-script,manager-status,admin" username="paolo"/>
    <user password="123" roles="manager-script,admin" username="123"/>
</tomcat-users>
```

In questo modo "ci siamo dati le autorizzazioni" per effettuare tutte le operazioni con Tomcat.

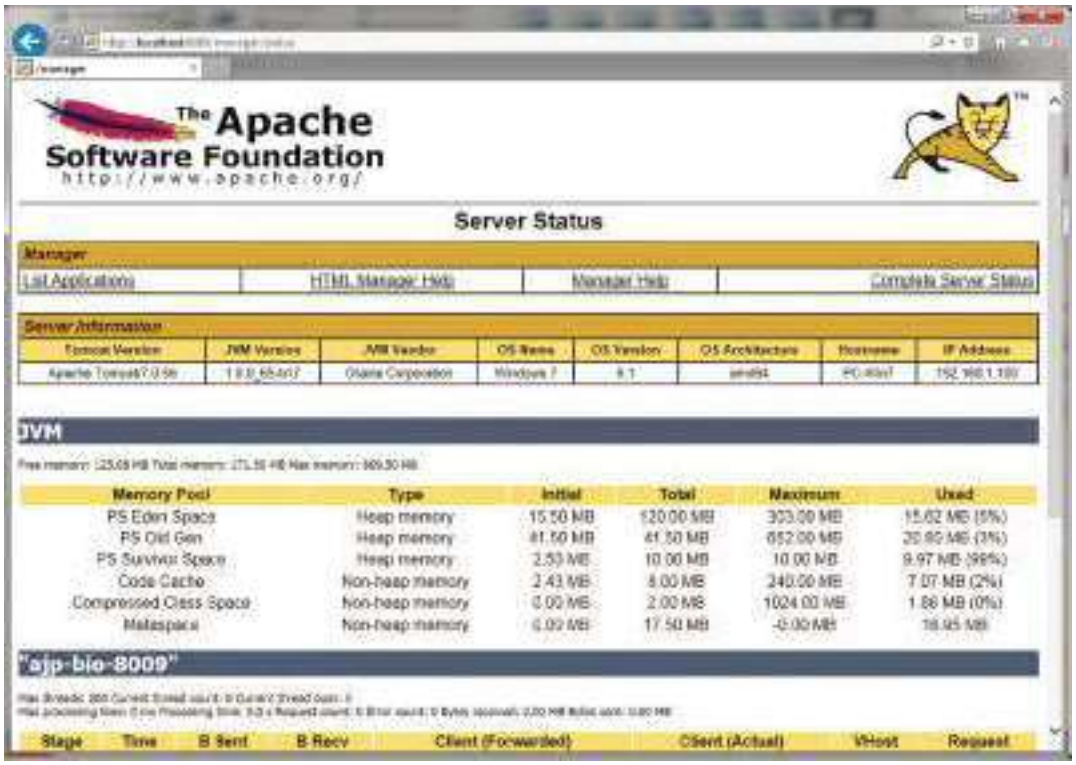
Possiamo verificare la nuove possibilità operative accedendo alla sezione riservata cliccando sul pulsante Admin dopo aver avviamo Tomcat.

Nella schermata principale di Tomcat clicchiamo su Server Status:

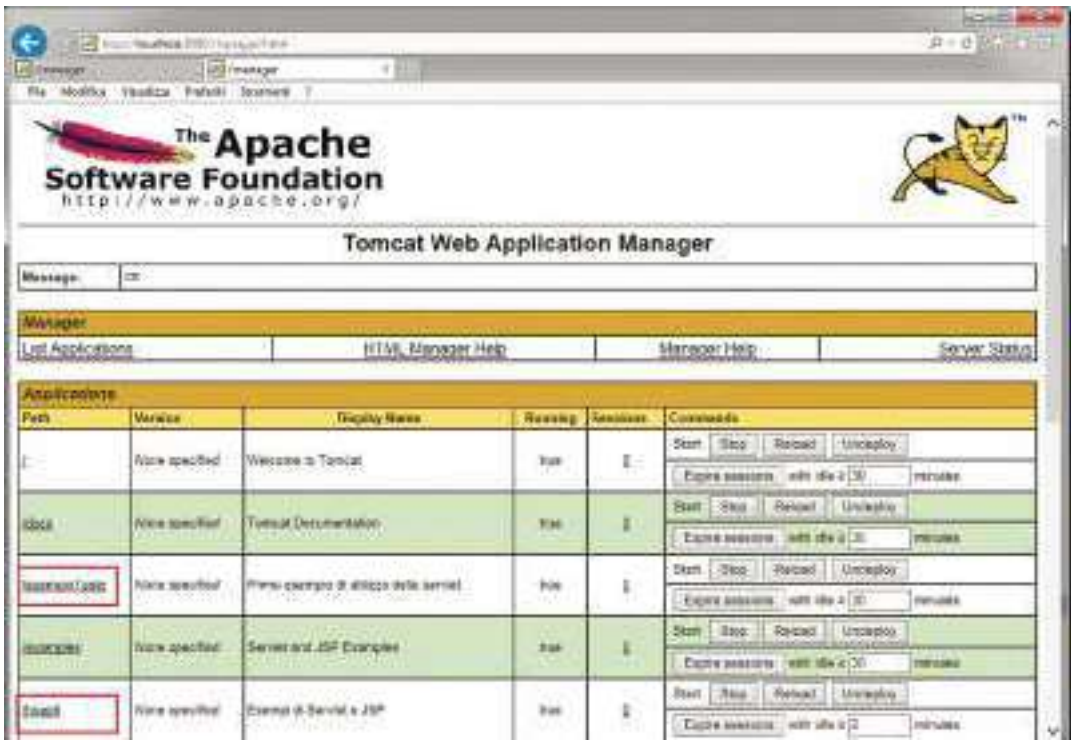


# 2 ESERCIZI IN LABORATORIO

e digitiamo i dati appena inseriti nel file di configurazione alla richiesta di autenticazione:




Cliccando invece su Manager App la videata presentata sarà quella riportata di seguito; tale videata è molto utile in fase di messa a punto dell'applicazione Web in quanto permette di fermare e ricaricare la singola applicazione senza bisogno di fermare il Web server.

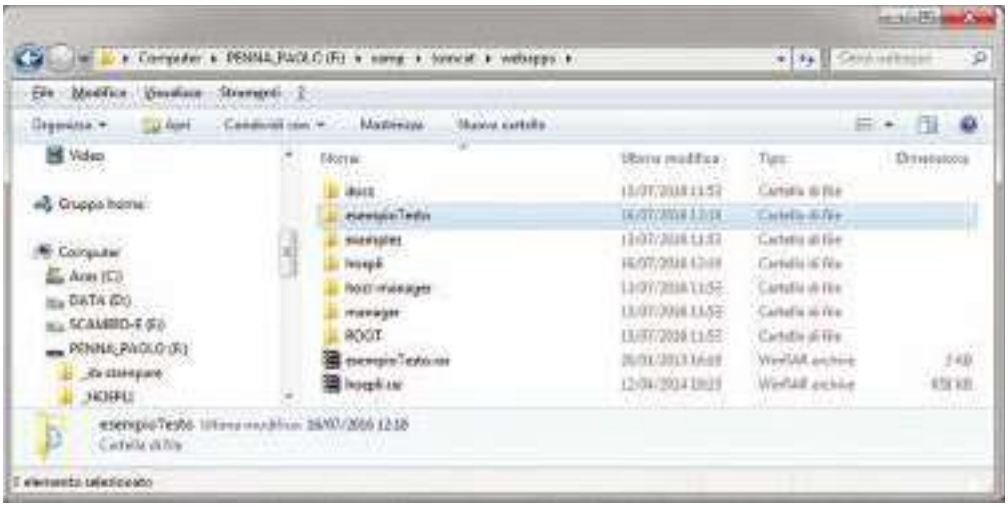


# ESERCIZI IN LABORATORIO 2

## Deployment di un'applicazione

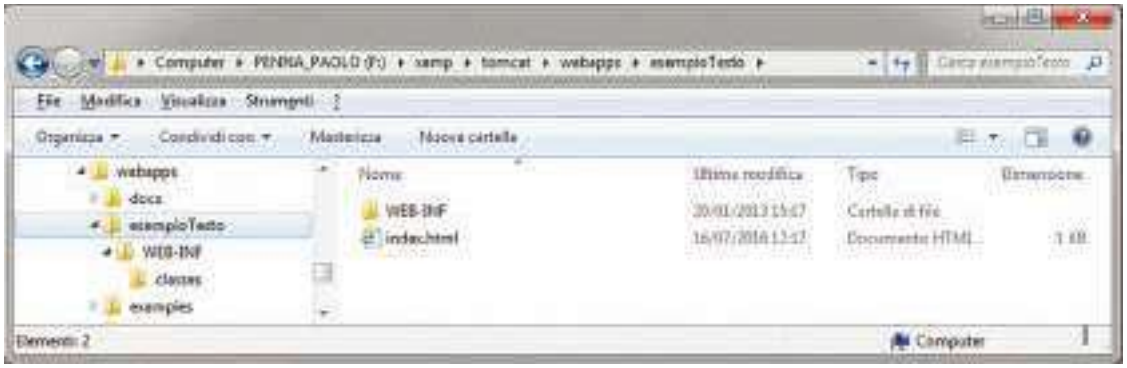
Il deploy di un'applicazione è già stato descritto nella Lezione 1 dove si sono analizzate tutte le sue componenti; rivediamo sinteticamente dove posizionare i file.

**1** Nella root directory dell'applicazione vengono collocati i file *.html, *.jsp.
Tutte le applicazioni devono essere collocate nella root directory, la cartella webapps: nel nostro caso nella cartella F:\xamp\tmcat\webapps sono presenti tutte le nostre applicazioni che verranno organizzate ciascuna in una sottodirectory, tra le quali la cartella esempioTesto che contiene i file dell'esempio descritto nella Lezione 1.



**2** All'interno creiamo una cartella chiamata WEB-INF e WEB-INF\classes.
Inseriamo i file del nostro progetto:

- nella cartella inseriamo il file "di avvio" dell'applicazione, cioè index.html;
- nella sottodirectory WEB-INF il file web.xml, cioè il Deployment descriptor analizzato nella Lezione 1;
- in WEB-INF/classes i file *.class delle servlet (bytecode), nel nostro esempio EsempioTesto.class;
- in WEB-INF/lib gli eventuali file di libreria *.jar.




Dopo aver riavviato il server possiamo mandare in esecuzione l'applicazione dal browser digitando il percorso della root directory e, se tutto è andato a buon fine, verrà caricata la pagina index.html della nostra Web application che mandata in esecuzione produce il risultato della figura di destra.

# 2 ESERCIZI IN LABORATORIO

Attenzione ai caratteri maiuscoli/minuscoli perché qui è tutto case-sensitive (in realtà sarebbe meglio abituarsi a gestire le maiuscole e le minuscole anche nei sistemi dove non incidono).

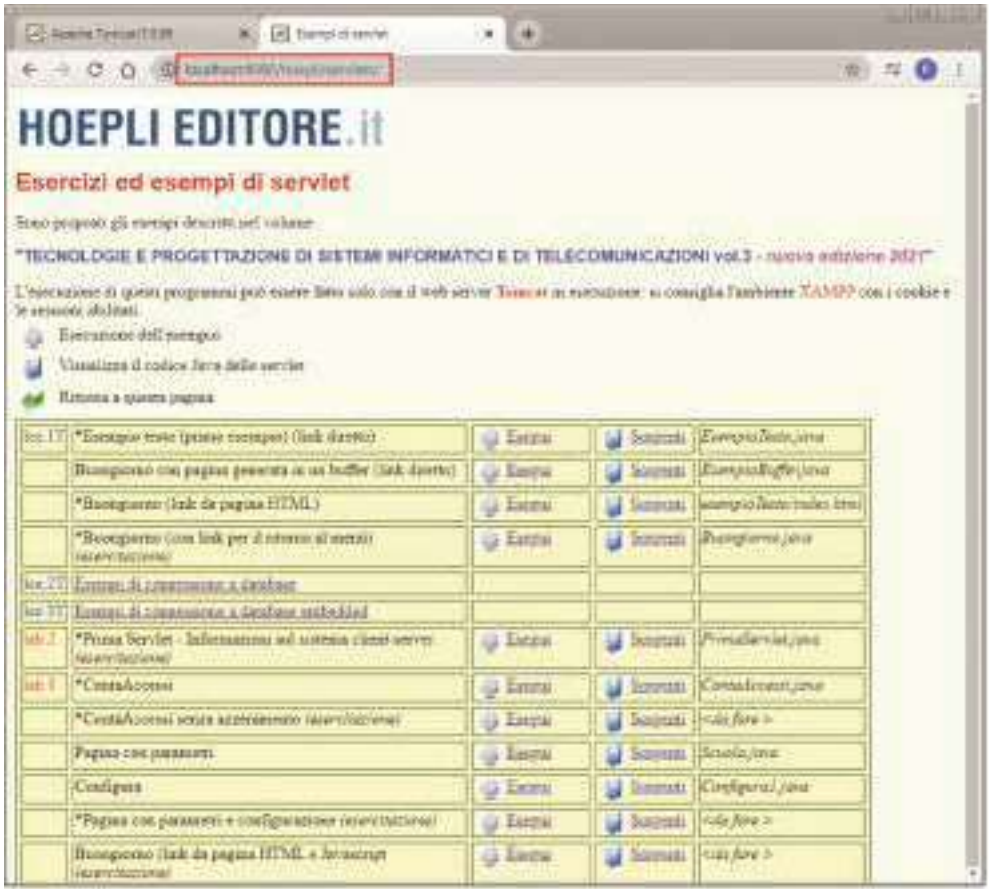
Nella sezione Materiali del sito www.hoepliscuola.it riservato al presente volume sono presenti tre file compressi:

- esempioTesto.rar
- primaServlet.rar
- hoepli.rar

Il primo contiene i file di questo esempio, il secondo un esempio di una servlet singola, mentre il terzo è il progetto completo con tutti gli esempi e alcune delle soluzioni degli esercizi proposti per le applicazione servlet e JSP.

È sufficiente decompattare il file nella cartella webapps per avere a disposizione la seguente home page all'indirizzo localhost/hoepli (oppure localhost:8080/hoepli):

La home page che abbiamo a disposizione contiene i link alle seguenti pagine:



# ESERCIZI IN LABORATORIO 2

La seconda pagina raccoglie gli esempi per le JSP presentate nella Lezione 2: i codici sorgente Java sono nel file servlet_JSP.rar (una copia è anche presente nella directory classes).

## METTITI ALLA PROVA

• Realizzare una servlet • Deployment di un'applicazione

Realizza una prima servlet che visualizzi sullo schermo le informazioni riguardanti il colloquio client-server. Per esempio:

```
out.println("<h3>Informazioni sul sistema client-server</h3>");
out.println("Porta del server   : " +request.getServerPort()+"<BR>");
out.println("Carattere encoding : " +request.getCharacterEncoding()+"<BR>");
out.println("Metodo : " + request.getMethod()+"<BR>");
out.println("Indirizzo della richiesta : " + request.getRequestURI()+"<BR>");
out.println("Informazioni sul percorso : " + request.getPathInfo()+"<BR>");
out.println("Protocollo : " + request.getProtocol()+"<BR>");
out.println("Nome host remoto     : " + request.getRemoteHost()+"<BR>");
out.println("Indirizzo host remoto: " + request.getRemoteAddr()+"<BR>");
```

Per la sua realizzazione esegui la seguente procedura passo passo.

**1** Crea una cartella dentro TOMCAT/webapps che conterrà la tua applicazione (per esempio: primaServlet).

**2** Crea una semplice pagina index.html dentro webapps/PrimaServlet con un link alla servlet come il seguente:

```
<p><a href="servlet/PrimaServlet">Per eseguire la servlet clicca qui!</a> </p>
```

**3** Crea la cartella WEB-INF dentro la home-directory PrimaServlet.

**4** Scrivi il descrittore di deployment web.xml e mettilo dentro WEB-INF.

**5** Crea la cartella classes dentro WEB-INF.

**6** Copia in questa cartella il file PrimaServlet.class.

**7** Collauda la tua prima applicazione da un browser digitando: http://localhost:8080/PrimaServlet.

Confronta la tua soluzione con quella riportata PrimaServletSol.java.

# 3 ESERCIZI IN LABORATORIO

## L'INIZIALIZZAZIONE DELLE SERVLET

Nella servlet che abbiamo descritto come primo esempio nell'Esercizio precedente non abbiamo inserito il metodo che ne permette la sua inizializzazione, cioè void init (ServletConfig config): tale metodo, ereditato direttamente dalla classe GenericServlet presente nel package javax.servlet.http, viene chiamato automaticamente all'atto del caricamento della servlet in memoria, cioè alla prima esecuzione della sua prima istanza, come avviene per il costruttore delle classi Java.

Se la servlet non riesce a completare l'inizializzazione viene generata una eccezione.

### Metodo init()

Il metodo init() riceve come parametro un oggetto di tipo ServletConfig che contiene la configurazione iniziale di una servlet.

```
5  public class ContaAccessi extends HttpServlet
6  {
7    int conta;
8    public void init(ServletConfig config)
9    {
10     conta = 0;
11   }
12   public void doGet(HttpServletRequest req, HttpServletResponse res)
13      throws IOException, ServletException
14   {
15     res.setContentType("text/html");
16     PrintWriter out = res.getWriter();  // definizione dell'output
17     out.println("<html><body>Volte" + conta ++ +"</html></body>");
18     out.close();
19   }
20 }
```

Nel sovrascrivere init() è importante ricordare di richiamare init() della classe padre col metodo super.init(), come riportato in seguito, dove abbiamo rettificato il codice precedente: senza questo invito al super, il codice all'interno della classe genitore GenericServlet non funziona e si possono avere dei problemi.

```
8    public void init(ServletConfig config)
9      throws ServletException
10   {
11     super.init(config);
12     conta = 0;
13   }
```

### METTITI ALLA PROVA

• Inizializzazione della servlet • Configurazione web.xml

Completa la servlet posizionandola in una cartella a piacere di webapps e aggiornando di conseguenza il corrispondente file web.xml; successivamente modifica il codice per fare in modo che il valore del contatore non venga azzerato a ogni nuova esecuzione del motore delle servlet.

# ESERCIZI IN LABORATORIO 3

## Configurazione di una servlet

Molte applicazioni hanno come esigenza comune quella di avere la possibilità di "parametrizzare" il funzionamento dell'applicazione stessa senza dover riscrivere il codice.
Molteplici sono i possibili vantaggi, come quello di:

- adattare il funzionamento dell'applicazione con piccole modifiche di alcuni fattori;
- personalizzare il funzionamento per i vari clienti/utenti.

Supponiamo di scrivere una servlet per gestire una scuola: naturalmente la stessa servlet potrà essere riusata per scuole diverse, dato che queste hanno molte cose in comune e il "riutilizzo del codice" è la prima "legge dello sviluppatore". Dobbiamo però gestire alcune piccole differenze tra le varie installazioni, come per esempio personalizzare il *nome della scuola*, senza dover modificare il codice della servlet.

Se mettiamo nella classe della servlet una variabile d'istanza con il nome della scuola:

```
public class Scuola extends HttpServlet{
  String intestazione="ITIS Fibonacci";
```

Sicuramente risolviamo il problema in modo "elementare", ma per una scuola differente saremo costretti a modificare il codice Java e ricompilarlo, inoltre dovremo tenere una versione per ogni installazione che faremo ed effettuare eventuali rettifiche a tutte le copie, una alla volta!

> Una soluzione migliore è quella di mettere tutti i parametri in un file che andrà letto prima di iniziare a gestire le richieste dei client.

Le API delle servlet forniscono un formato di file apposito e delle funzionalità specifiche che permettono di accedere al contenuto dei parametri senza bisogno di conoscere la struttura del file e/o effettuare complesse operazioni di parsing ma semplicemente accedendo a delle semplici proprietà di oggetti Java.
Sostanzialmente ci sono due possibilità per passare dei parametri di inizializzazione a una servlet:
**a** lettura di file ResourceBundle;
**b** lettura di parametri costanti di configurazione ServletConfig dal file web.xml.

## Utilizzo di file ResourceBundle

Un file ResourceBundle è un file di testo di tipo .properties che viene associato a un'applicazione Java che mette a disposizione la classe astratta java.util.ResourceBundle che consente di definire delle risorse contenenti le informazioni utili a localizzare un'applicazione.
Un file ResourceBundle contiene le informazioni nella forma chiave/valore dove la chiave identifica uno specifico elemento.

> L'utilizzo tipico dei ResourceBundle è quello che permette di gestire le diverse lingue in un'applicazione: per esempio un file Resources.properties conterrà i valori di default per tutte le stringhe dell'applicazione e un file Resources_fr_FR.properties per la versione francese ecc.

La tecnica del file ResourceBundle può essere utilizzata anche per inizializzare la servlet (che noi, tuttavia, non utilizzeremo).

# 3 ESERCIZI IN LABORATORIO

## Lettura di parametri tipo ServletConfig

Le servlet possono essere configurate aggiungendo particolari righe nella pagina web.xml; è possibile definire coppie chiave/valore che vengono lette dall'oggetto ServletConfig semplicemente con i metodi:

- getInitParameterNames()
- getInitParameter(String)

Per aggiungere uno o più parametri "costanti" di inizializzazione nella servlet è sufficiente aggiungere per ciascuno di essi la seguente coppia di meta-tag:

```
<init-param>
<param-name>nomeParametro</param-name>
<param-value>valoreParametro</param-value>
</init-param>
```

### ESEMPIO

Rendiamo parametrico il nomeScuola, la citta e il colore desiderato per visualizzarla, aggiungendo tre coppie di parametri nel file XML e in ciascuna ne inizializziamo un valore:

```
<servlet>
    <servlet-name>Scuola</servlet-name>
    <servlet-class>Scuola</servlet-class>
    <init-param>
      <param-name>citta</param-name>
      <param-value>Como</param-value>
    </init-param>
    <init-param>
      <param-name>nomeScuola</param-name>
      <param-value>ITIS Alan Turing</param-value>
    </init-param>
    <init-param>
      <param-name>colore</param-name>
      <param-value>#FF0000</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>Scuola</servlet-name>
    <url-pattern>/servlets/servlet/Scuola</url-pattern>
</servlet-mapping>
```

Per accedere ai propri parametri di configurazione la servlet utilizza l'interfaccia ServletConfig che viene passata come parametro al metodo init(): mediante il suo metodo getInitParameter() è possibile ottenere il valore di un parametro in base al nome con il quale è stato definito nel file XML:

```
String getInitParameter(String nomeParametro)
```

# ESERCIZI IN LABORATORIO 3

## ESEMPIO

Completiamo la definizione del metodo init() del nostro esempio dopo aver definito due variabili globali che ci serviranno per memorizzare i parametri letti con getInitParameter():

```
22  public void init(ServletConfig config)
23    throws ServletException
24  {
25    super.init(config);
26    citta = config.getInitParameter("citta");
27    nomeScuola = config.getInitParameter("nomeScuola");
28    colore = config.getInitParameter("colore");
29  }
```

Ora i parametri sono nelle due variabili e possono essere utilizzati all'interno della servlet per definire, per esempio, il titolo della pagina HTML di risposta:

```
37    // genera codice html
38    out.println(PAGE_TOP);
39    out.println("<h1><font color=" + colore + ">" + nomeScuola + "</font></h1>");
40    out.println("<h3> di "+ citta +"</h3>");
41    out.println(PAGE_BOTTOM);
```

Si può accedere ai parametri di configurazione anche tramite il metodo getServletConfig() che inoltre può essere invocato in qualunque momento, non necessariamente solo all'interno di init().

Completiamo la servlet e la mandiamo in esecuzione ottenendo:

## METTITI ALLA PROVA

**AREA DIGITALE**

**NetBeans, Eclipse e il file web.xml**

• Configurare una servlet • Definizione parametri

1. Modifica il file XML cambiando il valore dei parametri e quindi rimanda in esecuzione al servlet: aggiungi inoltre il nome e il cognome del preside.
2. Successivamente cancella i tag di definizione, una riga alla volta, osservando volta per volta cosa accade all'out rieseguendo la servlet.
   *Osservazione:* ricordati di riavviare sempre Tomcat!
3. Modifica l'esempio Configura1.java aggiungendo la chiamata di getServletConfig() per leggere il contenuto delle variabili di inizializzazione all'interno della servlet.

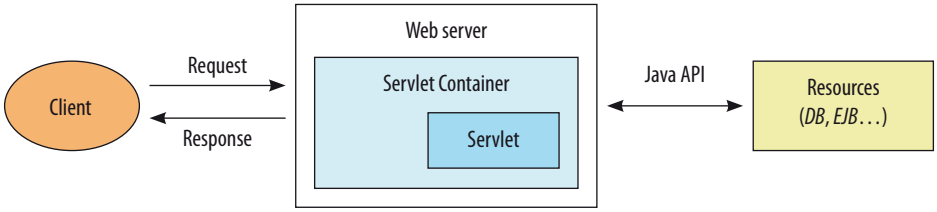
# 4 ESERCIZI IN LABORATORIO

## L'INTERAZIONE GET/POST TRA CLIENT E SERVLET

### Servlet con passaggio dei parametri

In questa esercitazione ci occuperemo del passaggio di parametri alla servlet.
Oltre ai parametri di configurazione letti dal file web.xml mediante l'oggetto ServletConfig le servlet devono acquisire i parametri dal client e comportarsi di conseguenza.

Lo schema di interazione tra client e server è il seguente:
**1** il client fa una richiesta Http al Web server (request);
**2** il Web server carica la servlet (solo la prima volta) e crea un thread per eseguirla;
**3** il container esegue la servlet richiesta;
**4** la servlet genera la risposta (response);
**5** la risposta viene restituita al client.




I form del client hanno sostanzialmente due modalità di invio dei dati alla servlet:
- agganciati alla URL come query string (GET);
- inseriti nel body del pacchetto Http (POST).

Ogni servlet estende la classe HttpServlet per gestire il protocollo Http implementando i metodi definiti per gestire l'interazione Http col client che si ritengono necessari, che sono:
- doGet, per gestire richiesta di tipo GET o HEAD;
- doPost, per gestire richiesta di tipo POST.

Tali metodi ricevono tutti due argomenti:
- l'oggetto HttpServletRequest per la gestione dei dati ricevuti dal client;
- l'oggetto HttpServletResponse per la gestione dei dati da inviare al client.

A seconda del metodo Http utilizzato per la richiesta, la servlet accede ai dati nei seguenti modi:

**A** con tutti i metodi Http viene utilizzato il metodo getParameter(<parametro>) che ritorna una stringa con il contenuto del parametro indicato; sono anche disponibili i metodi getParameterNames() e getParameterValues() che rispettivamente forniscono i nomi dei parametro e un array che ne contiene i valori.

La loro sintassi è la seguente:

```
String getParameter(String)          // ritorna il valore del parametro dal suo nome
Enumeration getParameterNames()      // ritorna una enumerazione dei parametri
String[] getParameterValues()        // ritorna una array di valori del parametro
```

**B** con il metodo GET si può utilizzare il metodo getQueryString() che restituisce una stringa che però deve successivamente essere analizzata per individuare i nomi dei parametri e i rispettivi valori.

```
String getQueryString()              // ritorna la query string completa
```

# ESERCIZI IN LABORATORIO 4

## ESEMPIO

Come primo esempio scriviamo una pagina HTML come quella della figura seguente, dove mediante due textbox inviamo alla servlet due campi, nome e cognome, utilizzando il metodo GET.

Nella servlet implementiamo il metodo doGet() e leggiamo i parametri con il metodo getParameter() e le seguenti istruzioni:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException
{
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();     // get writer
  // i parametri nome e cognome vengono passati dal browser
  String nome = request.getParameter("nome");
  String cognome = request.getParameter("cognome");
```

Con queste variabili realizziamo la pagina HTML da inviare come risposta al client:

```
// contenuto della pagina HTML
out.println("Ciao "+nome+" "+cognome+"!</p>");
out.println("<b>La data di oggi e': " + new java.util.Date()+ "</b></p>");
```

A schermo il risultato è il seguente:

Analogo risultato lo si ottiene utilizzando il metodo POST.

# 4 ESERCIZI IN LABORATORIO

## METTITI ALLA PROVA

• Passaggio parametri con doPost() • Ulizzo metodo doGet()

Modifica la pagina Servlet6.html e la classe Form1.java per effettuare le medesime operazioni utilizzando il metodo POST e scrivendo il metodo doPost().
A video devi ottenere il seguente risultato:

dove osserviamo che non è presente la query string a differenza del caso precedente.

Un suggerimento per risolvere la richiesta precedente senza dover riscrivere due doGet() e doPost(), per accettare entrambe le possibilità offerte dal client HTML, è quello di scrivere nella servlet solo uno di essi e richiamarlo dentro l'altro: per esempio, dopo aver scritto il metodo doPost() lo richiamiamo nel metodo doGet() in questo modo:

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
throws IOException, ServletException
{
  doPost(request, response);
}
```

## METTITI ALLA PROVA

• Passaggio parametri con doPost() • Passaggio parametri con doGet()

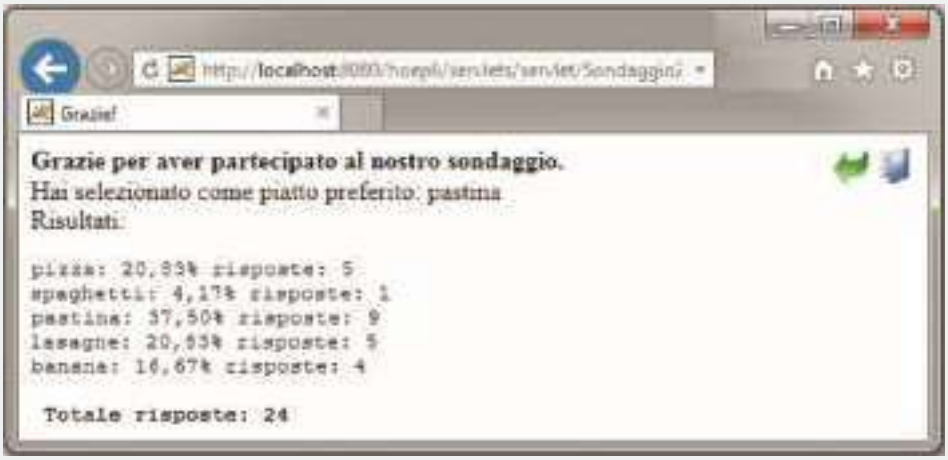
1 Aggiungi alla classe che hai appena realizzato il metodo sopra scritto e richiama la classe sia mediante POST che mediante GET in modo da verificarne il funzionamento.

2 Successivamente aggiungi dei radio button come nella figura:

# ESERCIZI IN LABORATORIO 4

e visualizza sullo schermo il piatto prescelto:

3 Infine memorizza i risultati in un file in modo da poter effettuare un sondaggio via Web, sottoponendo la domanda a tutti i tuoi compagni di classe e visualizzando i risultati con una pagina simile alla seguente:




## Parametri passati dal client con query string diretta

È anche possibile passare dei parametri al momento della chiamata della servlet, cioè accodandoli alla linea di comando, mediante la sintassi:

```
http://.../NomeServlet?nomeParametro=Paolo
```

dove:
- NomeServlet è il nome della servlet;
- nomeParametro è il nome del parametro utilizzato nella classe;
- Paolo è il valore attuale del parametro.

### ESEMPIO

Nel nostro esempio la servlet si chiama Diretta1, il nome del parametro para1 e il valore attuale Paolo; l'attivazione viene effettuata con la seguente riga di comando:

```
http://localhost:8080/hoepli/servlets/servlet/Diretta1?para1=Paolo
```

A tale scopo scriviamo la pagina diretta1.html che richiama la servlet passando il nome Paolo come parametro e leggendo altri due parametri dal file web.xml.

# 4 ESERCIZI IN LABORATORIO

In alternativa alla digitazione dell'istruzione nel browser, può essere comodo avviare le servlet Configura1 mediante un richiamo all'interno di una pagina HTML con il seguente comando:

```
<p>Richiama la classe <font color="#FF0000">Diretta1.class</font></p>
<p><a href="servlet/Diretta1?para1=Paolo">Per eseguire la servlet clicca qui!</a></p>
```

Nella servlet viene definito come nel caso precedente il metodo doGet() che legge il parametro para1:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException
{
  // il parametro para1 viene passato dal browser accodato con ?para1=Paolo
  String nomeUtente = request.getParameter("para1");
```

che viene visualizzato semplicemente con:

```
  // contenuto della pagina HTML
  out.println(saluti+" "+nomeUtente+"!");
```

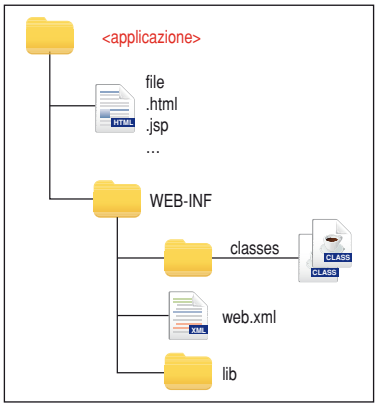
Il risultato è quello riportato di seguito.

---

I parametri sono stati passati dalla pagina HTML con il GET, e quindi "ben visibili" nella stringa di query.

## Il file.war (Web Archives)

Un'applicazione Web può essere memorizzata in un unico file compresso di estensione .war (Web Archives) che permette di trasferire semplicemente le applicazioni Web dal team di sviluppo al Web server per essere pubblicata in Internet, memorizzandola nella cartella webapps di Tomcat.
All'interno del file .war saranno presenti tutti i file del progetto memorizzati secondo l'albero descritto dalla "Servlet 2.4 specification" che riportiamo a lato:




Per crearli ci sono due possibilità:
**a** creare un archivio .zip e quindi rinominarlo, soluzione semplice e veloce;
**b** utilizzare il programma jar.exe (presente nella directory bin dell'ambiente Java) con il comando che segue:

# ESERCIZI IN LABORATORIO 4

```
Amministratore: C:\Windows\system32\cmd.exe
C:\java8\bin>jar
Usai jar {ctxui}[vfm0PMe] [jar-file] [manifest-file] [entry-point] [-C dir] fil
es ...
Opzioni:
    -c  crea un nuovo archivio
    -t  visualizza l'indice dell'archivio
    -x  estrae i file con nome (o tutti i file) dall'archivio
    -u  aggiorna l'archivio esistente
    -v  genera output commentato dall'output standard
    -f  specifica il nome file dell'archivio
    -m  include informazioni manifest dal file manifest specificato
    -n  esegue normalizzazione Pack200 dopo la creazione di un nuovo archivio
    -e  specifica il punto di ingresso per l'applicazione stand-alone
        inclusa nel file jar eseguibile
    -0  solo memorizzazione; senza compressione ZIP
    -P  conserva i componenti iniziali '/' (percorso assoluto) e "..\" (directo
ry padre) dai nomi file
    -M  consente di non creare un file manifest per le voci
    -i  genera informazioni sull'indice per i file jar specificati
    -C  imposta la directory specificata e include il file seguente
Se un file è una directory, verrà elaborato in modo ricorsivo.
Il nome del file manifest, del file di archivio e del punto di ingresso devono
essere specificati nello stesso ordine dei flag 'm', 'f' ed 'e'.
```

Un esempio di esecuzione è il seguente:

```
F:\xampp\tomcat\webapps>jar -cvf secondaEsempio.war niaapp2/
aggiunta manifest
aggiunta in corso di: niaapp2/(in = 0) (out = 0)(memorizzato 0%)
aggiunta in corso di: niaapp2/index.html(in = 332) (out = 211)(compresso 36%)
aggiunta in corso di: niaapp2/WEB-INF/(in = 0) (out = 0)(memorizzato 0%)
aggiunta in corso di: niaapp2/WEB-INF/classes/(in = 0) (out = 0)(memorizzato 0%)

aggiunta in corso di: niaapp2/WEB-INF/classes/EsempioTesto.class(in = 1177) (out
= 635)(compresso 46%)
aggiunta in corso di: niaapp2/WEB-INF/classes/EsempioTesto.java(in = 948) (out =
419)(compresso 55%)
aggiunta in corso di: niaapp2/WEB-INF/web.xml(in = 738) (out = 333)(compresso 54
%)
```

Adesso è sufficiente collocare il file nella root di Tomcat (NON nella directory webapps!) e automaticamente, grazie ai dati contenuti nel **manifest file**, la nostra applicazione verrà configurata.

**Manifest file**

Il manifest file è un file di testo non formattato che contiene alcune informazioni utili a definire le proprietà di un archivio JAR: per esempio, in esso viene indicato quale file contiene il metodo main() e il CLASSPATH da usare per l'esecuzione del JAR.

## METTITI ALLA PROVA

• Creazione file .war • Utilizzo manifest file

Realizza una semplice applicazione Web dove una pagina HTML richiama una servlet (per esempio Login1.class) mediante un pulsante e una routine Javascript, come nell'esempio riportato nella figura, per inviare i dati di accesso a un'area riservata:

# 5 ESERCIZI IN LABORATORIO

## L'INTERAZIONE CON CLIENT AJAX E LO SCAMBIO DATI IN FORMATO JSON

### Premessa

In questa esercitazione descriveremo come utilizzare le servlet all'interno delle pagine HTML tramite JavaScript: nello specifico utilizzeremo AJAX (Asynchronous JavaScript and XML), descritto nella Lezione 2 dell'Unità di Apprendimento 3, per effettuare le richieste, ricevere e analizzare le risposte dalla servlet.
Vedremo nel secondo esempio come scambiare dati tra server e client in formato JSON (descritto nell'Esercizio in Laboratorio 5 dell'Unità di Apprendimento 1) e come analizzarli in AJAX e visualizzarli all'interno della pagina HTML.

### Client AJAX che interagisce con una servlet

Riprendiamo l'esempio fatto in precedenza che effettuava un sondaggio sulle preferenze alimentari, visualizzando un gruppo di radiobutton e un pulsante per confermare la scelta:

La pagina HTML è composta da due <DIV>, rispettivamente il primo per presentare all'utente le operazioni che deve effettuare e il secondo per indicare l'area nella quale verranno visualizzati i risultati.

```
31  <TITLE>Esempio di Servlet con interazione in AJAX</TITLE>
34  </HEAD>
35  <BODY>
36    <DIV id = "scelta">
37    <FORM>
38    <P><STRONG><FONT color="#000000" size="4" face="Arial, Helvetica, sans-serif">Esempio di
      utilizzo delle servlet - GET con AJAX e radiobutton  </FONT></STRONG></P>
39    <P>Quale e' il tuo piatto preferito?<BR>
40      <BR>
41      <INPUT TYPE="radio"  NAME="piatto" VALUE="pizza">
42      pizza<BR>
43      <INPUT TYPE="radio"  NAME="piatto" VALUE="spaghetti">
44      spaghetti<BR>
45      <INPUT TYPE="radio"  NAME="piatto" VALUE="pastina">
46      pastina<BR>
47      <INPUT TYPE="radio"  NAME="piatto" VALUE="lasagne">
48      lasagne<BR>
49      <INPUT TYPE="radio"  NAME="piatto" VALUE="tortellini"CHECKED id="piatto">
50      tortellini<BR>
51      <BR>
52      <INPUT TYPE="button"  onclick="conferma()" VALUE="Inoltra" />
53    </P>
54    </FORM>
55    </DIV>
56
57    <DIV id = "risposta"> </DIV>
58  </BODY>
```

<DIV id = "scelta">
con la presentazione
del questionario

<DIV id = "risposta">
con la presentazione
dei risultati

# ESERCIZI IN LABORATORIO 5

Alla conferma del piatto selezionato viene richiamata la funzione conferma() che, come prima operazione, effettua l'analisi dei singoli elementi del radiobutton per individuare quello che è stato selezionato (checked) e prelevare il suo valore:

```
<!DOCTYPE HTML PUBLIC "-//DOC//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<script type "text/javascript">
  function conferma(){
    // recupero l'opzione selezionata dal radiobutton
    var radioButton = document.getElementsByName("piatto");
    for (x= 0 ; x <radioButton.length; x++){   // scorro tutti i radiobutton
      corrente = radioButton[x];
      if (corrente.checked){                   // individuo quello selezionato
        piattoScelto= corrente.value;
      }
    }
```

Predisponiamo ora la stringa con il parametro che avvia la comunicazione con la servlet Sondaggio1:

```
    // esegue la comunicazione con la servlet
    var URL = "http://localhost:8080/hosll/servlets/servlet/Sondaggio1?piatto="+piattoScelto;
    var xmlhttp = new XMLHttpRequest();          // oggetto che gestisce la risposta dal server
    xmlhttp.onreadystatechange = function(){     // risposta alla modifica dello stato
      if (this.readyState == 4 ) {               // di richiesta eseguita correttamente
        if (this.statusText =="OK"){             // OK: codice di stato HTTP 200
          var ricevuto = xmlhttp.responseText;   // estrae il testo ricevuto
          visualizza(ricevuto);                  // visualizza il risultato
        }
      }
    }
    xmlhttp.open("GET", URL, true);              // apre la connessione e ottiene event listener
    xmlhttp.send();                              // invia la richiesta al server
  }
```

l'oggetto xmlhttp avvia la comunicazione con la servlet con i metodi open() e send()

l'evento onreadystatechange viene generato dalla risposta del server

La ricezione della risposta da parte della servlet viene sentita dal listener dell'evento onreadystatechange: a esso associamo la funzione di risposta al verificarsi di tale evento; il valore della proprietà readyState di xmlhttp ci permette di analizzare la risposta HTTP ricevuta dal server: se il valore di readyState è uguale a 4, cioè la richiesta è stata completata con successo, il browser ha ricevuto una risposta che può essere analizzata.
Inoltre, per maggior sicurezza, testiamo anche il valore della proprietà status associata al codice di stato di Http che contiene 200 se tutto è andato a buon fine e ha settato a "OK" il corrispondente statusText.
In caso di esito positivo, leggiamo la stringa inviataci dal server e la passiamo alla funzione che provvede alla sua visualizzazione nel <DIV> risposta.

```
  function visualizza(ricevuto){
    document.getElementById("risposta").innerHTML = ricevuto;
  }
</script>
```

Un'esecuzione mostra nella seconda parte della pagina, nel <DIV> risposta, il seguente testo ricevuto dalla servlet:

**Sondaggio gastronomico**

Grazie per aver partecipato al nostro sondaggio.
Hai selezionato come piatto preferito: **pastina**

Il codice completo è riportato nel file servletAJAX1.html che utilizza la servlet Sondaggio1.java.

# 5 ESERCIZI IN LABORATORIO

## Scambio dati in formato JSON con una servlet

Scriviamo ora un'applicazione che interagisce con una servlet ricevendo da essa come risposta un insieme di dati in formato JSON che, quindi, viene elaborato da JavaScript estraendone i singoli record che vengono presentati all'utente in una tabella nella pagina HTML del client.
Nel nostro esempio chiediamo all'utente di scegliere un editore, tra quelli predisposti in un combo box, in modo da ottenere un elenco dei suoi libri presenti nel catalogo/archivio.

La struttura della pagina HTML è praticamente identica a quella dell'esempio precedente, dove al posto del radiobutton abbiamo ora collocato un combo box:

```
40  <TITLE>Esempio di Servlet con risposta in AJAX-JSON</TITLE>
41  </HEAD>
42  <BODY>
43    <DIV id = "scelta">
44    <FORM>
45      <P><STRONG><FONT color="#000000" size="4" face="Arial, Helvetica, sans-serif">Esempio
46      di utilizzo delle servlet con GET, AJAX e JSON </FONT></STRONG></P>
47      <P>Seleziona l'editore desiderato:<BR><BR>
48      <SELECT id="editore" name="editore">
49      <OPTION value="Sergio Bonelli">Sergio Bonelli</OPTION>
50      <OPTION value="Rusconi">Rusconi</OPTION>
51      <OPTION value="Hoepli">Hoepli</OPTION>
52      <OPTION value="Il saggiatore">Il saggiatore</OPTION>
53      </SELECT>
54      <BR><BR>
55      <INPUT TYPE="button" onclick="trova()" VALUE="Cerca libri" />&lt;/P&gt;
56      <P><STRONG>Nota</STRONG>: viene restituito un file <FONT color="#FF0000">JSON</FONT>
57      generato dalla servlet <FONT color="#FF0000">ServletCercaLibri1.java</FONT><BR>
58      &amp;nbsp;&amp;nbsp; -verrà visualizzato dalla funzione <font color="#FF0000">visualizza.js</FONT></P>
59    </FORM>
60    </DIV>
61    <DIV id = "tabella"></DIV>
62  </BODY>
```

Anche la funzione trova() ha la stessa struttura: unica differenza è l'analisi dei dati ricevuti che, sapendo che sono in formato JSON, saranno parserizzati dall'apposita funzione che li memorizza in un oggetto, che abbiamo chiamato objJSON:

```
1  <HTML>
2  <HEAD>
3
4  <script type "text/javascript">
5    function trova(){
6      var editoreScelto = document.getElementById("editore").value;
7      var URL ="http://localhost:8080/hoepli/servlets/servlet/ServletCercaLibri1?editore="+
8        editoreScelto;
9      var xmlhttp = new XMLHttpRequest();          // oggetto che gestisce la risposta del server
10     xmlhttp.onreadystatechange = function(){     // risposta alla modifica dello stato
11       if (this.readyState == 4){                 // 4: richiesta eseguita correttamente
12         if (this.statusText =="OK"){             // OK: codice di stato HTML 200
13           var objJSON=JSON.parse(this.responseText);// analizza la stringa JSON ricevuta
14           visualizza(objJSON,editoreScelto);     // chiama funzione che si occupa dell'output
15         }
16       }
17     }
18     xmlhttp.open("GET", URL, true);              // apre connessione e attende event listener
19     xmlhttp.send();                              // invia la richiesta al server
20   }
```

analisi dati JSON ricevuti

chiamata funzione di visualizzazione

# ESERCIZI IN LABORATORIO 5

A differenza dell'esempio precedente, la funzione che visualizza i dati nell'apposito <DIV> è più articolata in quanto deve creare una tabella analizzando uno alla volta i record presenti nell'oggetto objJSON che contiene i dati ricevuti dalla servlet ServletCercaLibri1.java:

oggetto con record libri

nome editore scelto

```
22  function visualizza(oggettoJSONtext, editoreScelto){
23    var x = 0;
24    var tabellaEsito = "<font color=\"#000000\">Editore selezionato: "+ editoreScelto + "<br>";
25    tabellaEsito += "<table border=1>";
26    tabellaEsito += "<tr><th>Autore</th><th>Titolo</th><th>Argomento</th></tr>";
27    for (x = 0; x < oggettoJSONtext.libri.length; x++){          // scorre tutti gli elementi
28      libro = oggettoJSONtext.libri[x];                          // analizza record corrente
29      tabellaEsito += "<tr>";
30      tabellaEsito += "<td>"+libro.autore +"</td>";
31      tabellaEsito += "<td>"+libro.titolo +"</td>";
32      tabellaEsito += "<td>"+libro.categoria +"</td>";
33      tabellaEsito += "</tr>";
34    }
35    tabellaEsito += "</table>";
36    document.getElementById("tabella").innerHTML = tabellaEsito;  // visualizza nel DIV "tabella"
37  }
38  </script>
```

creazione della tabella con i dati desiderati

visualizzazione del risultato nel DIV "tabella"

Selezionando, per esempio, il primo autore, otteniamo come risultato la seguente tabella:

Riportiamo la codifica Java della servlet, anche se non presenta particolari difficoltà nella sua realizzazione; definiamo innanzitutto una classe "minimalista" per poi definire gli oggetti Libri:

```
import java.util.*;
public class Libri{
  public String titolo;   // per comodità li mettiamo public
  public String autore;
  public String editore;
  public String categoria;
  public Libri (String s1,String s2,String s3,String s4){
    titolo = s1;
    autore = s2;
    editore = s3;
    categoria = s4;
  }
```

```
  // metodi getter e setter
  public void setCategoria(String categoria){
    this.categoria = categoria;
  }
  public String getCategoria(){
    return categoria;
  }
  public void setEditore(String editore){
    this.editore = editore;
  }
  public String getEditore(){
    return editore;
  }
}
```

Per semplicità predisponiamo manualmente in un vettore di oggetti un elenco di libri: naturalmente in un'applicazione "reale" e non ai soli fini didattici questi dati andrebbero letti da un database.

# 5 ESERCIZI IN LABORATORIO

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class ServletCercaLibri extends HttpServlet{
  private Vector elencoLibri;
  public void init(ServletConfig config)
  throws  ServletException
  {
    elencoLibri =new Vector();
    elencoLibri.addElement(new Libri("Algoritmi I","Paolo Camagni","Hoepli","informatica"));
    elencoLibri.addElement(new Libri("Algoritmi II","Paolo Camagni" ,"Hoepli","informatica"));
    elencoLibri.addElement(new Libri("Java","Nikolassy Riccardo","Hoepli","informatica"));
    elencoLibri.addElement(new Libri("Il nostro caro Lucio","Donato Zoppo","Hoepli","musica"));
    elencoLibri.addElement(new Libri("Fra la Via Emilia e il West","Paolo Talanca","Hoepli","musica"));
    elencoLibri.addElement(new Libri("Una vita","Italo Svevo","Rusconi","romanzo"));
    elencoLibri.addElement(new Libri("Il fu Mattia Pascal","Luigi Pirandello","Rusconi","romanzo"));
    elencoLibri.addElement(new Libri("I promessi sposi","Alessandro Manzoni","Rusconi","romanzo"));
    elencoLibri.addElement(new Libri("Gli incubi di Dylan Dog","Tiziano Sclavi","Sergio Bonelli","fumetti"));
    elencoLibri.addElement(new Libri("Corto fucile","Carlo Berardi","Sergio Bonelli","fumetti"));
  }
```

Il metodo doGet() viene richiamato dalla pagina HTML con parametro l'editore desiderato: tale parametro viene memorizzato in una variabile strEditore e, dopo aver creato un oggetto StringBuffer() nel quale verranno memorizzati i dati della risposta in formato JSON, viene utilizzato dal metodo doGet() per effettuare la selezione dei libri presenti in archivio:

- viene effettuata la scansione dell'elenco dei libri confrontando i dati presenti con la chiave di ricerca, cioè l'editore richiesto;
- i libri che soddisfano la ricerca vengono aggiunti al risultato memorizzandoli come coppie *attributo–valore* nell'albero JSON:

```
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException{
    // prepara i dati
    int quanti = 0;
    Libri libroCor;
    // leggo il parametro
    String strEditore = request.getParameter("editore");  // editore scelto
    // stringa di risposta in formato JSON
    StringBuffer datiJSON = new StringBuffer(); // utilizzeremo metodo append
    // apertura stringa JSON
    datiJSON.append("{\"libri\":[");
    // selezioniamo i libri e creiamo la stringa JSON
    for (int x = 0; x < elencoLibri.size(); x++){
      libroCor = (Libri) elencoLibri.elementAt(x);   // analisi libro corrente
      if (strEditore.equals(libroCor.editore)){
        if (quanti>0) datiJSON.append(",");          // dopo il primo elemento
        quanti++;
        datiJSON.append("{");
        datiJSON.append("\"titolo\": \""+libroCor.titolo+"\",");
        datiJSON.append("\"autore\": \""+libroCor.autore+"\",");
        datiJSON.append("\"categoria\": \""+libroCor.categoria+"\"");
        datiJSON.append("}");
      }
    }
```

Al termine dell'analisi dei dati, viene "chiusa" la stringa JSON, viene indicata la natura della risposta settando i due specifici attributi setContentType() e setCharacterEncoding() e viene inoltrata la risposta al client che ne aveva fatta richiesta con l'operazione print().

# ESERCIZI IN LABORATORIO 5

```
    // chiusura stringa JSON
    datiJSON.append("] }");
    // risposta al client con specifiche di tipo/single
    PrintWriter out = response.getWriter();
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    out.print(datiJSON);
    out.close();
  }
}
```

Il codice completo è riportato nel file servletJSON.html che utilizza la servlet CercaLibri1.java.

## METTITI ALLA PROVA

**1** Completa il primo esempio aggiungendo la memorizzazione in un file da parte della servlet delle scelte effettuate nel sondaggio e proponendo la "classifica", indicando per ciascuna opzione il numero delle volte che è stata selezionata e la % rispetto al totale.

**2** Completa il secondo esempio aggiungendo alla servlet la lettura dei dati dei libri prelevandoli dal database provejava e interrogandolo con una query di selezione utilizzando il parametro ricevuto dal client.

**3** Aggiungi successivamente il caricamento degli editori nel combo box leggendo i loro dati anch'essi dal database provejava, la struttura dell'interazione client-server avviene quindi in queste fasi:
– richiesta al server dell'elenco delle case editrici con interrogazione del database (tabella editori);
– visualizzazione delle stesse nel combo box del client;
– richiesta al server dei libri della casa editrice selezionata dall'utente;
– interrogazione del database (tabella libri);
– visualizzazione nel client dell'elenco dei libri presenti in archivio.

**4** Aggiungi successivamente il caricamento degli argomenti nel combo box leggendo i loro dati anch'essi dal database provejava, la struttura dell'interazione client-server avviene quindi in queste fasi:
– richiesta al server dell'elenco degli argomenti con interrogazione del database (tabella argomenti);
– visualizzazione degli stessi nel combo box del client;
– richiesta al server dell'elenco delle case editrici che hanno pubblicato qualche libro di un argomento scelto con interrogazione del database (tabella libri);
– visualizzazione delle stesse nel combo box del client;
– richiesta al server dei dati della casa editrice selezionata dall'utente (tabella editori);
– visualizzazione nel client della scheda anagrafica della casa editrice selezionata.

# 6 ESERCIZI IN LABORATORIO

## LA PERMANENZA DEI DATI CON LE SERVLET: I COOKIE

### I cookie con le servlet

Quasi tutte le applicazioni Web sono composte da più di una pagina e, nel soddisfare le richieste dello stesso client, è utile "ricordarsi" delle precedenti pagine che sono state visitate o dei parametri che abbiamo fatto scegliere all'utente per personalizzare la sua visita.
Tra i dati che dobbiamo ricordare è l'avvenuto accesso a una sezione riservata, in modo da non richiedere le credenziali "a ogni successiva pagina" che il nostro visitatore richiede.
Sappiamo che il protocollo Http è stateless, cioè dopo una richiesta effettuata dal client e una risposta soddisfatta dal server si "riparte da capo", cioè l'interazione client-server viene "dimenticata" e si passa a considerare la richiesta successiva.
Abbiano alcune alternative per risolvere il problema, tra le quali ricordiamo:

- inserire informazioni inerenti allo stato direttamente negli URL, accodandoli come stringa di query: questa soluzione, oltre che a essere complessa è intrinsecamente insicura;
- aggiungere campi nascosti (hidden) nella pagina HTML, ma anche questa soluzione è insicura;
- utilizzare dei cookie, che vedremo in questa esercitazione;
- utilizzare il meccanismo delle sessioni, che vedremo nel prossimo Esercizio in Laboratorio.

### I cookie

I cookie sono dei piccoli file di testo (che possono anche essere limitati a una lunghezza di 4096 byte) che vengono gestiti e memorizzati dal browser, quindi dal client.
Un browser è tenuto ad accettare al massimo 20 cookie per sito e 300 in totale per ogni utente: dato che i cookie vengono inviati nell'header Http, questi devono essere creati soltanto prima che qualsiasi altro contenuto sia inviato al client.
Server e client si scambiano l'informazione presente nei cookie inglobandola negli header del protocollo Http: il browser allega i cookie scaricati da un dato host a ogni richiesta effettuata da questo host.

I cookie, oltre a essere usati dal motore delle servlet per gestire le sessioni e quindi risolvere tutti quei problemi legati alla sessione che descriveremo, come detto, nella prossima esercitazione, possono servire per:

- customizzazione di un sito in base alle pagine precedentemente visitate;
- pubblicità mirata alle pagine precedentemente visitate;
- eliminazione di username e password per l'accesso al sito.

### I cookie in Java

In Java l'API delle servlet fornisce la classe javax.servlet.http.Cookie che ci consente di utilizzare e manipolare facilmente i cookie.
Un cookie è un'istanza della classe Cookie che ha come costruttore:

```
Cookie (String nome, String valore)
```

# ESERCIZI IN LABORATORIO 6

Un cookie viene trasmesso dal server al client come oggetto di output, ovvero viene associato all'oggetto risposta (OUTPUT), quindi per creare un cookie si utilizza il metodo addCookies() dell'interfaccia HttpServletResponse, che prende in ingresso un oggetto Cookie.
A ogni cookie viene associato un nome e mediante il costruttore è possibile assegnargli un valore:

- il nome deve rispettare le regole dettate in RFC 2109, cioè può contenere solo caratteri ASCII alfanumerici esclusi virgole, punti e virgola e spazi, inoltre non può iniziare col simbolo "$"; nelle API Java, il nome del cookie non può essere cambiato dopo la sua creazione;
- il valore del cookie non ha vincoli sul contenuto.

## ESEMPIO

Vediamo un semplice esempio di creazione di un cookie che memorizza il colore preferito dell'utente:

```
9   Cookie mioCookie = null;                    // definisco un cookie
10  mioCookie = new Cookie("colore", "blue");   // creo l'oggetto e lo inizializzo
11  response.addCookie(mioCookie);
```

Un cookie viene trasmesso dal client al server come oggetto di input, ovvero viene associato all'oggetto richiesta (INPUT), quindi per leggere i cookie si utilizza il metodo getCookies() dell'oggetto HttpServletRequest, che restituisce un array di oggetti Cookie.

Non è possibile accedere direttamente al singolo cookie, ma il metodo getCookies() ritorna tutti i cookie. Li mettiamo in un vettore e lo scorriamo fino a individuare il cookie desiderato con questo segmento di codice:

```
Cookie[] cookies = request.getCookies();
for(int i = 0; i < cookies.length; i++)
  out.println(cookies[i].getName() + " " + cookies[i].getValue()+"<br>");
```

**AREA DIGITALE**

**Metodi principali classe Cookie**

## PER SAPERNE DI PIÙ

**PROPRIETÀ DEI COOKIE**

È possibile aggiungere degli attributi ai cookie per stabilirne la validità, sia in termini di tempo sia in termini di dominio o di percorso, o semplicemente per associargli un commento.
Alcuni metodi sono i seguenti:
void setComment (String s) e String getComment()
void setDomain (String s) e String getDomain()
void setMaxAge (int secondi) e int getMaxAge()

Per esempio:

```
13  mioCookie.setComment("colore di sfondo");   // commento
14  mioCookie.setDomain (".hoepli.com");        // dominio
15  mioCookie.setPath ("/");                    // percorso
16  mioCookie.setMaxAge(900);                   // durata in secondi (15 minuti)
```

Vediamo ora un secondo esempio dove effettuiamo l'inserimento dei dati dal client e la registrazione dei cookie da parte del server.

# 6 ESERCIZI IN LABORATORIO

## ESEMPIO

Scriviamo una servlet che visualizza una semplice pagina HTML che richiede all'utente due valori, il primo che verrà utilizzato come nome del nuovo cookie, e il secondo che ne costituirà il valore.

```
out.print("<form action=\"");
out.println("Cookie1\" method=POST>");
out.print("Nome  Cookie: ");
out.println("<input type=text length=20 name=nomeCookie><br>");
out.print("Valore Cookie: ");
out.println("<input type=text length=20 name=valoreCookie><br>");
out.println("<input type=submit></form>");
```

La pagina generata è la seguente:

La servlet legge i due parametri nel metodo doPost() che richiama getParameter() con le seguenti istruzioni e definisce un oggetto del tipo Cookie:

```
String nomeCookie = request.getParameter("nomeCookie");
String valoreCookie = request.getParameter("valoreCookie");

Cookie aCookie = null;
```

In questo oggetto andiamo a memorizzare i valori ricevuti come parametro e lo aggiungiamo all'oggetto response in modo da settare il nuovo cookie:

```
// controlla i parametri passati dal metodo doPost e doGet
if(nomeCookie != null && valoreCookie != null){
  aCookie = new Cookie(nomeCookie, valoreCookie);
  response.addCookie(aCookie);
}
```

Dall'oggetto request, con il metodo getCookies(); andiamo a "caricare" i cookie che sono stati definiti nel client e, nel caso ce ne fosse qualcuno settato, li visualizziamo con un ciclo:

```
Cookie[] cookies = request.getCookies();
if((cookies != null) && (cookies.length > 0)){
  out.println("Il tuo browser ha settato i seguenti cookies <br><br>");
  for (int i = 0; i < cookies.length; i++){
    Cookie cookie = cookies[i];
    out.print  ("Nome  Cookie : " + (cookie.getName())+ "<br>");
    out.println("Valore Cookie : "+ (cookie.getValue())+ "<br><br>");
  }
}else{
  out.println("non ci sono cookies settati nel tuo browser");
}
```

# ESERCIZI IN LABORATORIO 6

Ai quali aggiungiamo l'ultimo cookie settato, quello che abbiamo appena ricevuto dal form:

```
71      if(aCookie != null){
72          out.println("<P>");
73          out.println("Questo è l'ultimo cookie che hai settato:<br><br>");
74          out.print("Nome  Cookie : " + (nomeCookie) + "<br>");
75          out.print("Valore Cookie : " + (valoreCookie));
76      }
```

Mandiamo in esecuzione la servlet, inseriamo per esempio come nome cookie "colore" e come valore "rosso" e inviamo la query; a schermo avremo il seguente risultato:

Aggiungiamo ora un secondo cookie (a piacere) e confermiamo con Invia query ottenendo la seguente schermata, dove sono visibili entrambi i cookie che abbiamo settato:

## METTITI ALLA PROVA

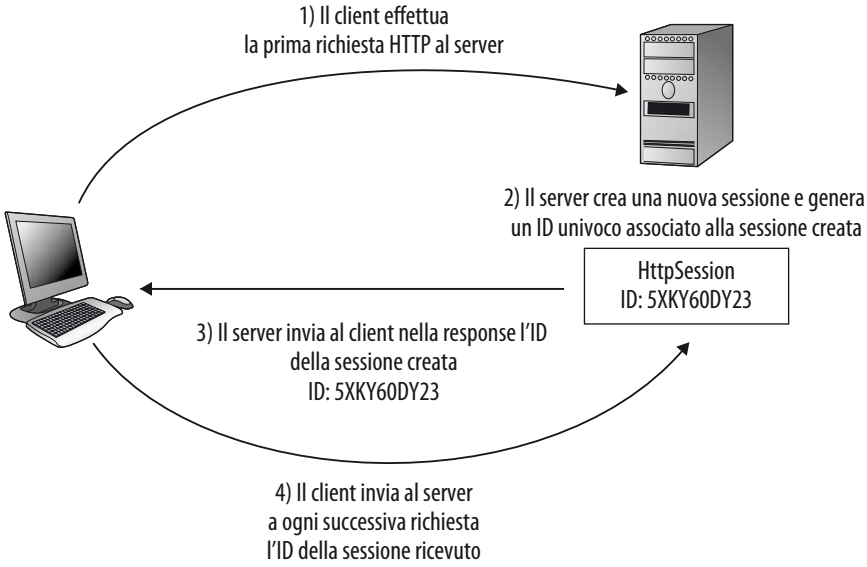
• Creazione dei cookies • Utilizzo dei cookies

Aggiungi gli opportuni controlli sull'input in modo da evitare gli errori dovuti a valori non corretti inseriti nei campi da parte dell'utente. Quindi aggiungi una list box contenente il nome dei possibili colori che l'utente può selezionare come sfondo: memorizza il colore come cookie e utilizzalo come sfondo per le successive pagine che verranno trasmesse al client.

# 7 ESERCIZI IN LABORATORIO

## LA PERMANENZA DEI DATI CON LE SERVLET: LE SESSIONI

Oltre ai cookies, un secondo metodo per effettuare la "permanenza dei dati" tra client e server è quello delle sessioni. La prima differenza sostanziale tra cookies e sessioni è che con le sessioni i dati sono memorizzati sul server e non sul client, mentre sul client viene memorizzato solo un ID di sessione (sempre attraverso un cookie).
Il funzionamento è il seguente: al primo collegamento da parte di un utente a una servlet che supporta le sessioni viene creato un oggetto in cui è possibile memorizzare le informazioni di interesse e gli viene assegnato un identificatore unico che viene inviato al client e memorizzato in un cookie.




A ogni successiva richiesta che viene effettuata dal client questo trasmette al server anche l'ID della sessione in modo che il server possa riconoscere l'utente e recuperarne i dati che ha precedentemente salvato.

> Il client invia solo l'ID di sessione, quindi si riducono le informazioni scambiate fra client e server, si riduce il traffico in rete con benefici sia per le prestazioni che per la sicurezza.

Se il browser non supporta il cookie, il numero di sessione deve essere registrato nelle query string al momento della creazione del codice della pagina, come nell'esempio seguente:

```
http://localhost/servlet/miaServlet?sessionID=69163392288624
```

### Sessioni Java

Le API di Java mettono a disposizione la classe javax.servlet.http.HttpSession per definire le sessioni: in particolare il metodo getSession() permette di creare e restituire il contenuto della sessione.
Quando viene invocato questo metodo, per esempio con

**AREA DIGITALE**
**Elenco dei metodi dell'oggetto HttpSession**

```
HttpSession sessione = request.getSession();
```

abbiamo due possibili situazioni:

1. se non esiste una sessione corrente per questo client, questa viene creata e il server genera un session ID univoco per quel particolare client;
2. se già presente l'ID di sessione, vengono caricati i dati relativi a quel client.

# ESERCIZI IN LABORATORIO 7

Una volta ottenuto l'oggetto session, questo può essere utilizzato tramite i suoi metodi per memorizzare qualsiasi tipo di informazione: in pratica la creazione di una sessione comporta la predisposizione sul server di un'area di memoria per la gestione di tutte le informazioni.

Per gestire le sessioni sono disponibili i seguenti due gruppi di metodi:
**A** per avere informazioni sulla sessione (ID della sessione, tempo trascorso, tempo di inattività)

```
String getID()                                  // ritorna l'ID della sessione
long getCreationTime()                          // ritorna il time stamp della creazione della sessione
long getLastAccessedTime()                      // ritorna il time stamp dell'ultimo accesso alla sessione
void invalidate()                               // interrompe la validità della sessione
```

**B** per memorizzare dati di sessione nella forma di coppia chiave = valore, dove per evitare conflitti si segue la convenzione di dare un nome agli oggetti secondo lo schema:
<nome-applicazione>.<nome-oggetto>

```
void setAttribute(String chiave, Object valore)   // aggiunge un oggetto
Object getAttribute(String chiave)                // recupera un oggetto dalla sessione
void removeValue(String chiave)                   // elimina un oggetto dalla sessione
```

Dato che le informazioni sono memorizzate sul server e che il numero dei client è numeroso, è necessario introdurre un meccanismo che permetta al server di liberare spazio quando la sessione non è più utilizzata; è possibile utilizzare un timer alla cui scadenza il container la elimina e per impostare il timeout si utilizza il seguente metodo:

```
public void setMaxInactiveInterval (int timeout)
```

oppure definendolo direttamente nel descrittore di deployment web.xml:

```
10  <session-config>
11    <session-timeout>15</session-timeout> <!-- in minuti -->
12  </session-config>
```

In questo modo la sessione scade dopo 15 minuti.
Si può anche invalidare esplicitamente la sessione richiamando dalla servlet il metodo invalidate().

## ESEMPIO

Come primo esempio inseriamo in una servlet un contatore di richieste, cioè una variabile che conta quante volte la servlet è stata mandata in esecuzione. Creiamo la sessione alla prima esecuzione della servlet (o la ricarichiamo alle successive esecuzioni) e "leggiamo dalla sessione" l'oggetto desiderato, per esempio Sessioni1.mioContatore.

```
// crea una sessione se questa non esiste
HttpSession sessione = request.getSession();
// recupera e setta il valore del contatore
Integer tanti = (Integer) sessione.getAttribute("Sessioni1.mioContatore");
```

Rispettiamo la convenzione per il naming della variabile di attribuire un nome composto dal nome della classe (Sessioni1) seguito dal "." e dal nome della variabile (mioContatore).

# 7 ESERCIZI IN LABORATORIO

Analizziamo il valore dell'oggetto tanti per incrementarne il valore; lo confrontiamo con null, che è il valore che contiene nel caso in cui la servlet viene eseguita per la prima volta e l'oggetto non è stato ancora definito, altrimenti ne incrementiamo il valore:

```
// alla fine il nuovo valore viene salvato
if (tanti ==null)                          // se non esiste
{
  tanti = new Integer(1);                  // ne crea uno nuovo
  output.println("E' la prima volta che viene eseguita la servlet</B>" );
}
else                                       // altrimenti
{
  tanti = new Integer(tanti.intValue()+1);  // lo incrementa
  output.print("La servlet e' stata eseguita nr. volte : " );
  output.println(tanti);
}
sessione.setAttribute("Sessioni1.mioContatore", tanti);
```

Con l'istruzione di riga 61 salviamo il valore dell'oggetto tanti nella sessione prima creata.
Mandando in esecuzione la servlet e richiamandola alcune volte otteniamo la seguente situazione:

Per vedere il nome del cookie che viene memorizzato sul client e l'identificatore della sessione appena creata possiamo utilizzare la servlet realizzata nella esercitazione precedente:

## METTITI ALLA PROVA

• Creazione sessioni • Utilizzo dati di sessione

**1** Completa l'esempio modificando la servlet Sessioni1.java aggiungendo la data del primo accesso (e quindi di creazione della sessione) e modificando a 3 minuti il tempo di "expired". Verificane il funzionamento richiamando la servlet dopo 5 minuti.

# ESERCIZI IN LABORATORIO 7

**2** Realizza una servlet che visualizzi quante volte un utente, durante la sua sessione di lavoro, ha avuto accesso alla servlet, modificando in base a essa la scritta di saluto, visualizzando per esempio:

**a** **"Benvenuto"** alla sua creazione e, quindi, al primo accesso;

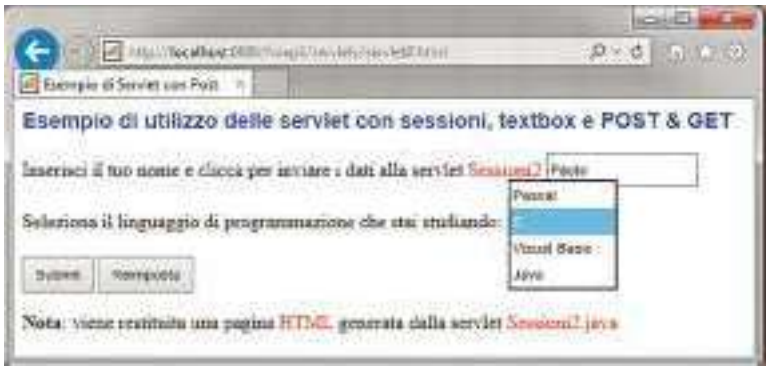
**b** **"Bentornato, vecchio amico"** ai successivi accessi;

**c** **"Sarebbe ora di cambiare!"** dopo il decimo accesso.

Una possibile codifica la puoi trovare nel file MostraSessione.java

## Un esempio completo: memorizziamo il linguaggio di programmazione

Vogliamo adesso realizzare una servlet più complessa, dove l'utente inserisce il proprio nome e seleziona da una list box un linguaggio di programmazione, come mostrato nella seguente pagina HTML:

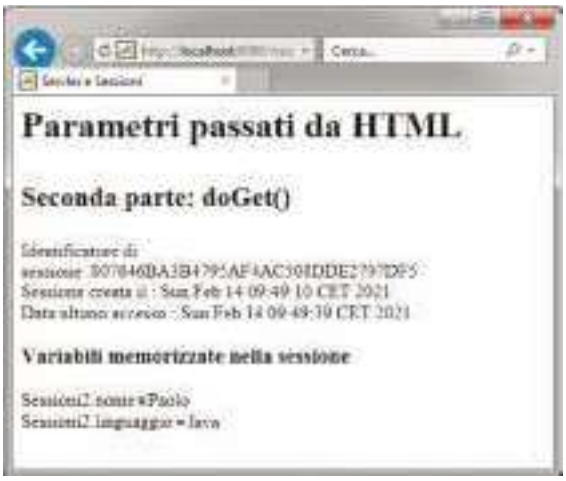


# 7 ESERCIZI IN LABORATORIO

Con il metodo POST inviamo il form alla servlet che memorizza le due variabili in una sessione:

```
// aggiungiamo i parametri alla sessione
session.setAttribute("Sessioni2.linguaggio", linguaggio);
session.setAttribute("Sessioni2.nome", nome);
```

Viene visualizzata una prima schermata in cui vengono evidenziate in un form le scelte dell'utente che vengono memorizzate in una sessione: mediante il postback il form richiama se stesso con il metodo GET e visualizza le informazioni sulla sessione e sulle variabili.




## METTITI ALLA PROVA

- Memorizzare il linguaggio di programmazione

1 Completa l'esempio aggiungendo altre due caselle di testo e inserendo un ciclo che visualizza tutte le variabili di sessione mediante un enumeratore; quindi predisponi due stringhe come quelle riportate di seguito:

```
private final static String names[] ={"Pascal", "C", "Java", "Visual Basic" };
private final static String isbn[] = {"978-88-203-48229","978-88-203-48243","978-88-203-48236","978-88-203-34727"};
```

2 A ogni linguaggio che viene selezionato visualizza il corrispondente codice ISBN del libro suggerito, come nella seguente schermata:

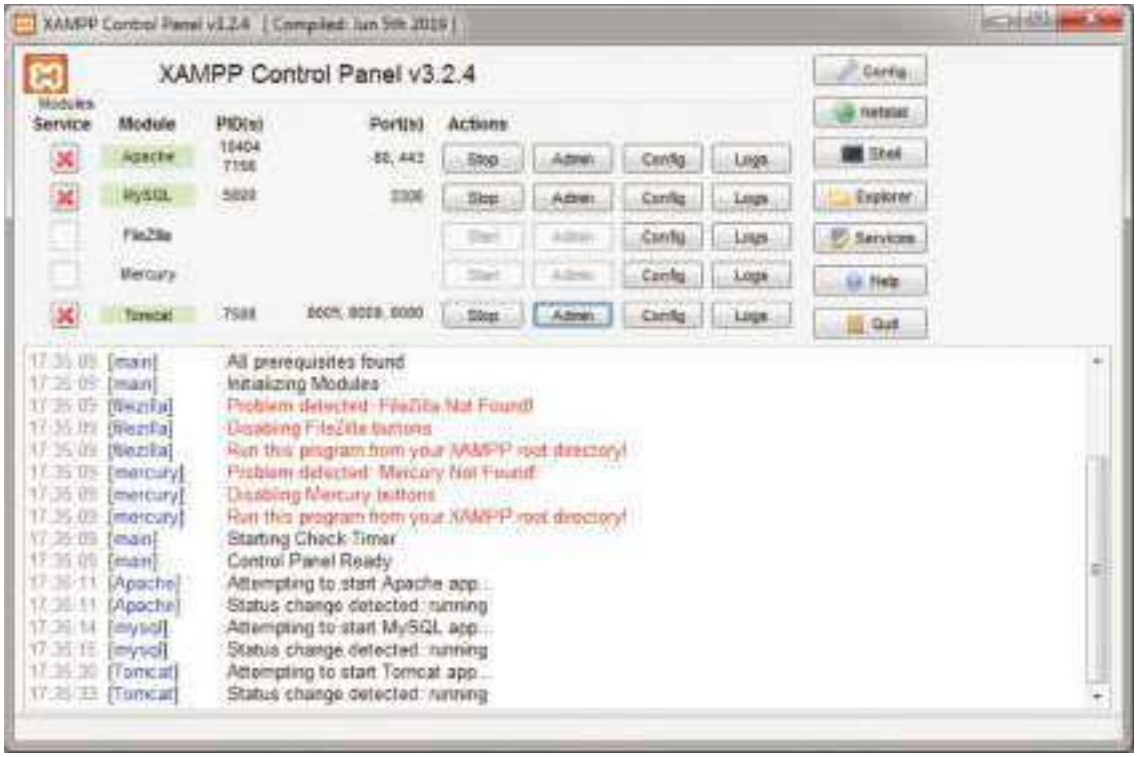
# ESERCIZI IN LABORATORIO 8

## JDBC E MySQL

### Il driver JDBC

Per poter connettere una servlet a un database è necessario per prima cosa caricare il driver specifico per il database che si desidera utilizzare; in questa esercitazione effettueremo la connessione a MySQL, che dispone sia di un driver JDBC sia di un driver ODBC.

MySQL è sicuramente il più famoso e diffuso DBMS open source ed è disponibile per tutte le piattaforme. Per poter effettuare le nostre prove è necessario che MySQL sia presente sul nostro PC: è già compreso nel pacchetto XAMP/LAMP, così come è anche presente nel pacchetto EasyPHP un secondo prodotto open source che contiene l'intero ambiente di sviluppo costituito da Apache, MySQL, PHP.




In alternativa il download può essere eseguito gratuitamente partendo dall'indirizzo http://www.mysql.com/, ma in tal caso per connettere Java al database MySQL è necessario scaricare il driver JDBC denominato Connector/J direttamente dalla pagina http://dev.mysql.com/downloads/connector/j/.

**AREA DIGITALE**

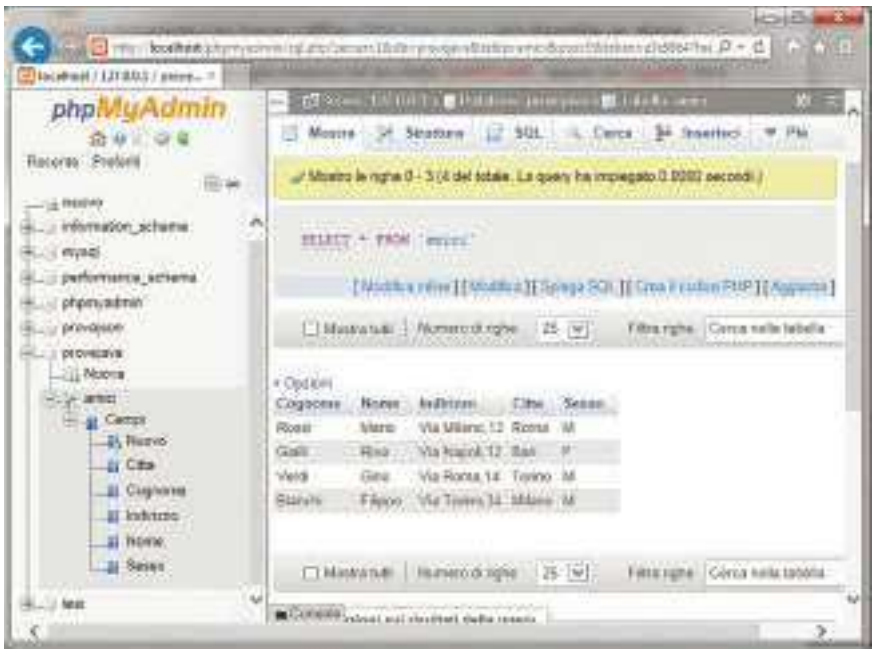
**Configurazione manuale di JDBC**

### Predisposizione dell'archivio MySQL

La connessione a un database richiede necessariamente la presenza del database MySQL per poter effettuare il test del codice Java. Per creare un archivio di prova ci si può connettere con il DBMS dalla linea di comando e digitare manualmente le istruzioni per la creazione del database, oppure utilizzare la comoda interfaccia offerta da XAMPP, che utilizza phpMyAdmin, o, ancora, installare un pacchetto specifico per la gestione e l'amministrazione dei database: HeidiSQL, che è un prodotto open source, può essere scaricato direttamente dal sito http://www.heidisql.com/.

# 8 ESERCIZI IN LABORATORIO

Creiamo un database provejava e all'interno di esso la tabella amici come nella figura:




La tabella può essere creata manualmente oppure mediante la seguente query:

```
1 CREATE TABLE `amici` (
2   `Cognome` CHAR(50) NULL DEFAULT NULL,
3   `Nome` CHAR(50) NULL DEFAULT NULL,
4   `Indirizzo` CHAR(50) NULL DEFAULT NULL,
5   `Citta` CHAR(50) NULL DEFAULT NULL,
6   `Sesso` CHAR(1) NULL DEFAULT NULL
7 )
```

Analogamente la tabella può essere popolata manualmente o mediante la query:

```
INSERT INTO `amici` (`Cognome`, `Nome`, `Indirizzo`, `Citta`, `Sesso`,
        VALUES ('Gialli', 'Rina', 'Via Napoli, 23', 'Bari', 'F');
```

## Connessione al database

Scriviamo ora una prima servlet che visualizza il contenuto della tabella amici che abbiamo appena creato nel database provejava. Estraiamo i dati con la semplice query seguente:

```
// definizione della query
String miaQuery = "SELECT Cognome, Nome, Indirizzo, Citta FROM amici";
```

Si vuole ottenere sullo schermo qualcosa di simile alla seguente schermata:

# ESERCIZI IN LABORATORIO 8

Inseriamo nella servlet un insieme di costanti necessarie per definire i parametri di connessione alla tabella, a partire dall'URL fino alla password che ci permette di accedere ai dati:

```
private final String url         = "jdbc:mysql://";
private final String serverName  = "localhost";
private final String portNumber  = ":3306/";
private final String databaseName= "provejava";
private final String userName    = "root";
private final String password    = "";
// riferimento al database: connessione Mysql
private final String URL_mioDB = url + serverName + portNumber + databaseName;
// definizione del driver per la connessione al DB MySQL
private final String DRIVER = "com.mysql.jdbc.Driver";
```

Dopo aver creato l'oggetto che ci permette di effettuare la connessione, eseguiamo di seguito:

```
// apro la connessione verso il database.
connessione = DriverManager.getConnection(URL_mioDB, userName, password);
// ottengo lo Statement per interagire con il database
Statement statement = connessione.createStatement();
// interrogo il DBMS mediante una query SQL
ResultSet resultSet = statement.executeQuery(miaQuery);
```

Se tutto è andato a buon fine i dati sono ora presenti nel ResultSet e con un semplice ciclo li visualizziamo sullo schermo ottenendo quanto ci eravamo proposti:

```
out.println("<b>cognome"+"&#9;"+"nome"+"&#9;"+"indirizzo"+"&#9;"+"citta</b>");
while (resultSet.next())
{
  String cognome = resultSet.getString(1);
  String nome = resultSet.getString(2);
  String indirizzo = resultSet.getString(3);
  String citta = resultSet.getString(4);
  out.println(cognome+"&#9;"+nome+"&#9;"+indirizzo+"&#9;"+citta);
}
```

Il codice completo è riportato nel file ServletBD2.java.

## Interrogazione del database

Vediamo ora una servlet per interrogare il database estraendo i nominativi in base al sesso; il parametro da passare alla servlet viene selezionato da una pagina HTML simile alla seguente:

# 8 ESERCIZI IN LABORATORIO

In base al valore del parametro scelto viene definita la stringa di query:

```
// predisponiamo la query in base al parametro scelto
String scelta = request.getParameter( "scelta" );
String query = "SELECT * FROM Amici where sesso='"+scelta+"'";
```

Selezionando ***maschi*** si ottiene:

Selezionando ***femmine*** si ottiene:

## METTITI ALLA PROVA

• Connessione a MySQL • Interrogazione mediante query

1 Scrivi una pagina HTML che selezioni da una list box una città ed effettui la richiesta a una servlet che interroghi il database prima creato selezionando gli amici che abitano in quella città.

# ESERCIZI IN LABORATORIO 8

Una possibile esecuzione è la seguente:

**2** Successivamente aggiungi alla scelta della città di residenza anche la possibilità di selezionare solo maschi o femmine.

**3** Come ultima applicazione, aggiungi un form che permetta di inserire i dati di nuovi amici e scrivi una servlet che effettui l'inserimento dei dati nella tabella amici popolando la tabella con almeno 15 record.

**4** Quindi completa la servlet di ricerca scritta in precedenza facendo in modo che, nel caso in cui non fossero presenti nell'archivio record che soddisfino la ricerca, venga richiamata la pagina che ne permette l'inserimento.

Un backup degli archivi popolati con gli esempi descritti nel libro è presente nella cartella:

È sufficiente effettuare l'importazione all'interno di MySQL con l'operazione Importa, selezionando il file opportuno tramite Sfoglia:

# 9 ESERCIZI IN LABORATORIO

## SERVLET E DATABASE MDB CON PARAMETRI

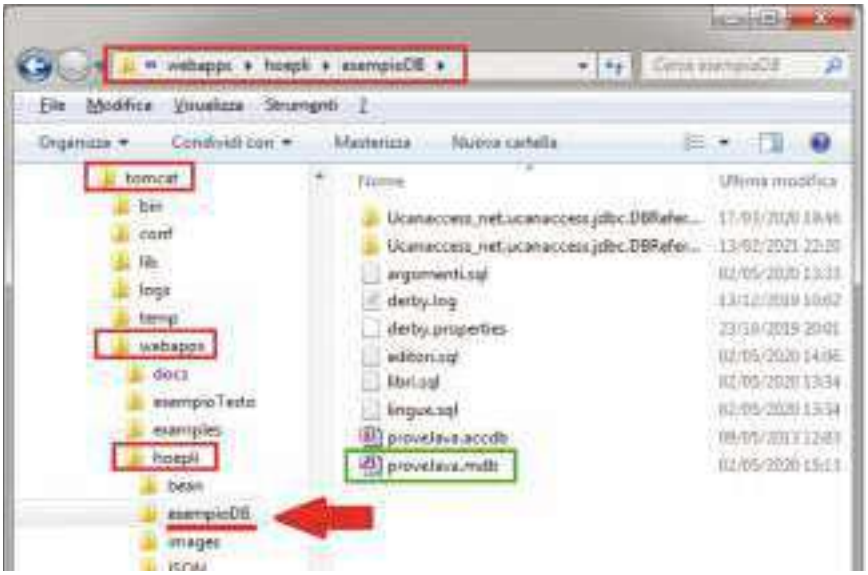
### Premessa

In questa esercitazione replicheremo quanto fatto per MySQL nell'Esercizio precedente realizzando con Access un file proveJava.mdb che contenga una tabella , che andiamo a realizzare e popolare con dati simili a quelli riportati nella figura:

Amici

| Cognome | Nome | Indirizzo | Città | Sesso | Età | Sposato |
|---|---|---|---|---|---|---|
| Bianca | Maria | via Como, 4 | Bari | F | 15 | ☐ |
| Bianchi | Filippo | Via Torino, 32 | Milano | M | 20 | ☐ |
| Gialli | Pina | Via Napoli, 23 | Roma | F | 25 | ☐ |
| Rossi | Mario | Via Milano, 12 | Roma | M | 30 | ☑ |
| Verdi | Gino | Via Roma, 14 | Torino | M | 35 | ☑ |
| Bruni | Alice | Via Bari, 3 | Napoli | F | 27 | ☑ |
| Neri | Anna | Via Rodi, 43 | Como | F | 50 | ☐ |
| Otto | Ciro | Via Priva,32 | Ancona | M | 28 | ☑ |

Memorizziamo il file in un'apposita cartella del Web server, per esempio come nella figura:




Se stiamo utilizzando una versione di Java inferiore alla 8.0 possiamo utilizzare ODBC come meccanismo di connessione: a tal fine è necessario registrare il database con l'apposita utility disponibile in Windows, solitamente nominata Origine Dati (ODBC).

**AREA DIGITALE**

**Registrazione database in Windows mediante ODBC**

Nella nostra trattazione per realizzare la connessione ad Access utilizzeremo le librerie UCanAaccess (che sono un'evoluzione/un completamento delle librerie jackcess) che, di fatto, hanno sostituito universalmente il meccanismo basato su ODBC.

> Inoltre le librerie UCanAccess consentono di "svincolarsi" dal meccanismo JDBC-ODBC che, se da una parte agli inizi sembrava molto comodo, con macchine a 64 bit, con Java 8 installato oppure con la connessione a DBMS diversi, ha portato molteplici problemi soprattutto con recordset di grosse dimensioni.

La loro installazione è immediata: basta copiare i file nella directory lib delle librerie Java.

# ESERCIZI IN LABORATORIO 9

## Passaggio dei parametri da HTML

Scriviamo una servlet che interroga il database selezionando, per esempio, i nominativi presenti in base al sesso: il parametro da passare alla servlet viene selezionato da una pagina HTML, già scritta a pagina 309.
Come accennato nella Lezione 2 definiamo nelle prime righe della servlet le stringhe necessarie alla connessione al database:

```
7  public class ServletDB3 extends HttpServlet
8  {
9    private final String protocollo = "jdbc:ucanaccess://";          // libreria
10   private final String mdbpath = "tomcat/webapps/hoepli/esempioDB/"; // percorso
11   private final String mdbName = "prove.Java.mdb;memory=false";     // database
12   private final String user    = "";                               // nome utente
13   private final String psw     = "";                               // password
14   // riferimento al database: connessione ODBC
15   private final String URL_mioDB = protocollo + mdbpath + mdbName;
16   // definizione del driver per la connessione al DB Access
17   private final String DRIVER = "net.ucanaccess.jdbc.UcanaccessDriver";
18
```

Cerchiamo di effettuare la connessione che renderà attivo all'interno del nostro programma l'apposito driver in un costrutto try … catch:

```
22     // carico il driver per la connessione al DB Access
23     try{
24       Class.forName(DRIVER);
25     }catch (ClassNotFoundException e1)  // il driver è caricato
26     {
27       System.out.println("Driver non trovato...");
28       System.exit(1);
29     }
```

Analogamente proviamo ad effettuare la connessione al database Access:

```
47     // definisco un oggetto per la connessione
48     Connection connessione = null;
49     try{  // apro la connessione verso il database.
50       connessione = DriverManager.getConnection(URL_mioDB);
51     }catch (Exception e){      // gestione dell'errore
52       System.out.println("Connessione al database non riuscita! ");
53       System.exit(1);
54     }
```

La servlet legge ora il parametro "scelta" col metodo doGet() e definisce la stringa di interrogazione:

```
56     String scelta = request.getParameter( "scelta" );
57     out.println("<h4>Sesso selezionato = "+scelta+"</h4>");
58     // definizione della query
59     String query = "SELECT * FROM Amici where sesso='"+scelta+"'";
60     if (scelta.equals("X"))
61       query = "SELECT Cognome, Nome, Indirizzo, Citta FROM Amici";
```

La query viene eseguita con la seguente istruzione, in modo da averne i risultati nell'oggetto resultSet:

```
63     try{
64       // ottengo lo Statement per interagire con il database
65       Statement statement = connessione.createStatement();
66       // interrogo il DBMS mediante una query SQL
67       ResultSet resultSet = statement.executeQuery(query);
```

# 9 ESERCIZI IN LABORATORIO

Il risultato ritornato dalla executeQuery è tipo una Enumeration, i cui vari elementi sono le righe della tabella come ritornate dalla query, accessibili via via tramite next e quindi con un semplice ciclo li visualizziamo sullo schermo:

```
71      out.println("<b>cognome &#9; nome &#9; indirizzo &#9; citta</b>");
72      while (resultSet.next()) {
73        String cognome = resultSet.getString(1);
74        String nome = resultSet.getString(2);
75        String indirizzo = resultSet.getString(3);
76        String citta = resultSet.getString(4);
77
78        out.println(cognome+"&#9;"+nome+"&#9;"+indirizzo+"&#9;"+citta);
79      }
```

Se per esempio nella pagina HTML selezioniamo ***femmine***, si ottiene:

## METTITI ALLA PROVA

• Connessione ad Access • Interrogazione mediante query

1 Scrivi una pagina HTML che selezioni da una list box una città ed effettui la richiesta a una servlet che interroga il database prima creato selezionando e visualizzando gli amici che abitano in quella città.

2 Successivamente aggiungi alla scelta della città di residenza anche la possibilità di selezionare solo maschi o femmine.

# ESERCIZI IN LABORATORIO 9

**3** Scrivi una nuova servlet che seleziona il campo che si vuole visualizzare combinato con l'età, come nell'esempio riportato di seguito:

Una possibile esecuzione è la seguente:

**4** Come ulteriore applicazione, aggiungi un form che permetta di inserire i dati di nuovi amici e scrivi una servlet che effettui l'inserimento dei dati nella tabella amici popolando la tabella con almeno 15 record.

**5** Quindi completa la servlet di ricerca scritta in precedenza facendo in modo che nel caso in cui non fossero presenti nell'archivio record che soddisfino la ricerca venga richiamata la pagina che ne permette l'inserimento.

# 10 ESERCIZI IN LABORATORIO

## SERVLET CON JAVADB DERBY

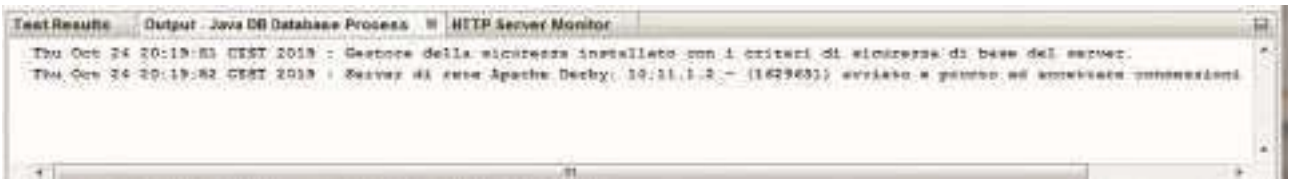
### Creazione di una servlet con accesso a JavaDB

Java DB è un sistema di gestione di database relazionali basato sul linguaggio di programmazione Java e SQL ed è incluso in Java a partire dalla SE 7 SDK.
Creiamo un database con una tabella alunni, nella quale inseriamo alcune istanze.

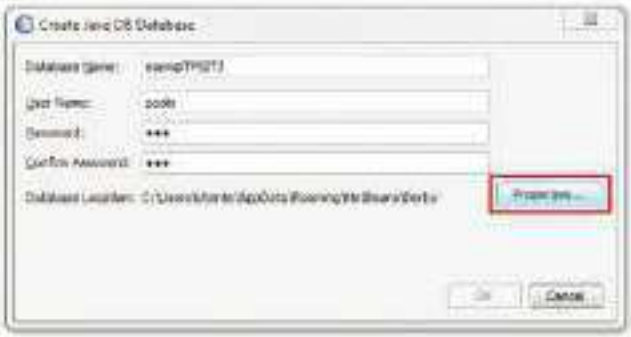
#### Creazione di un database con NetBeans

Avviamo NetBeans e ci posizioniamo nella scheda dei Services: la prima opzione è proprio Databases e, dopo averla selezionata con il tasto destro, ci appare il menu con le seguenti opzioni:
**a** avviamo il database con Start Server;
**b** creiamo il database con Create Database.

Se il server si avvia regolarmente otteniamo il seguente messaggio:




Procediamo con la creazione del database: dopo averne inserito il nome (esempiTPSIT3), il nome dell'utente (paolo) e una password (per esempio psw), modifichiamo la directory posizionando il database nel nostro Web server:

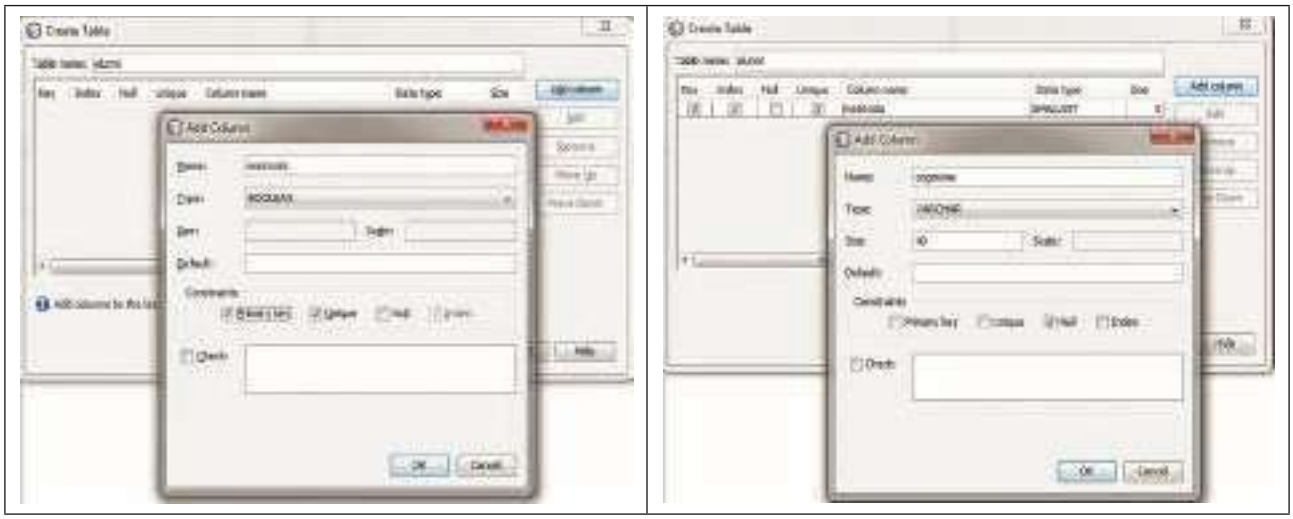


# ESERCIZI IN LABORATORIO 10

Possiamo ora connetterci al database cliccando con il tasto destro e inserendo le credenziali:

Possiamo ora definire la tabella alunni cliccando con il tasto destro sull'apposita opzione di Tables:

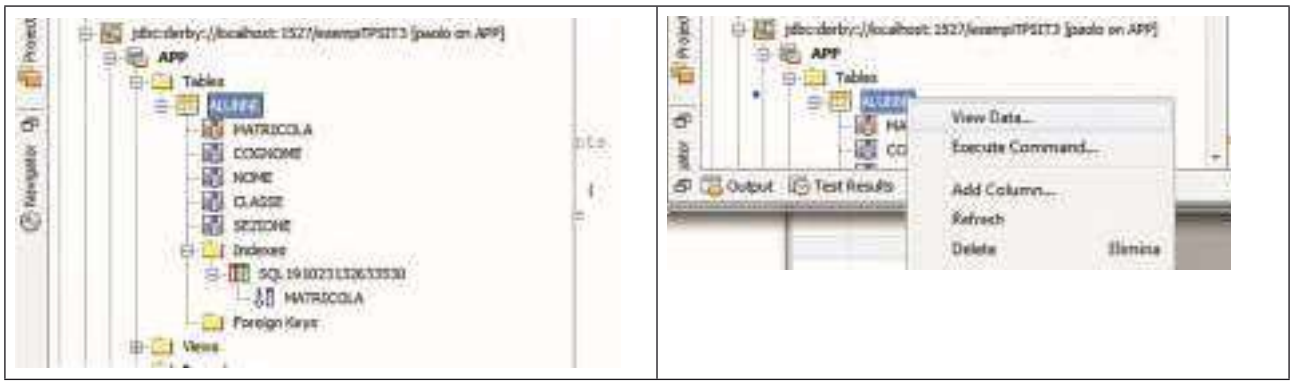
Diamo nome alunni alla tabella e iniziamo ad aggiungere alcuni attributi: matricola, cognome, nome, classe, sezione, sesso.



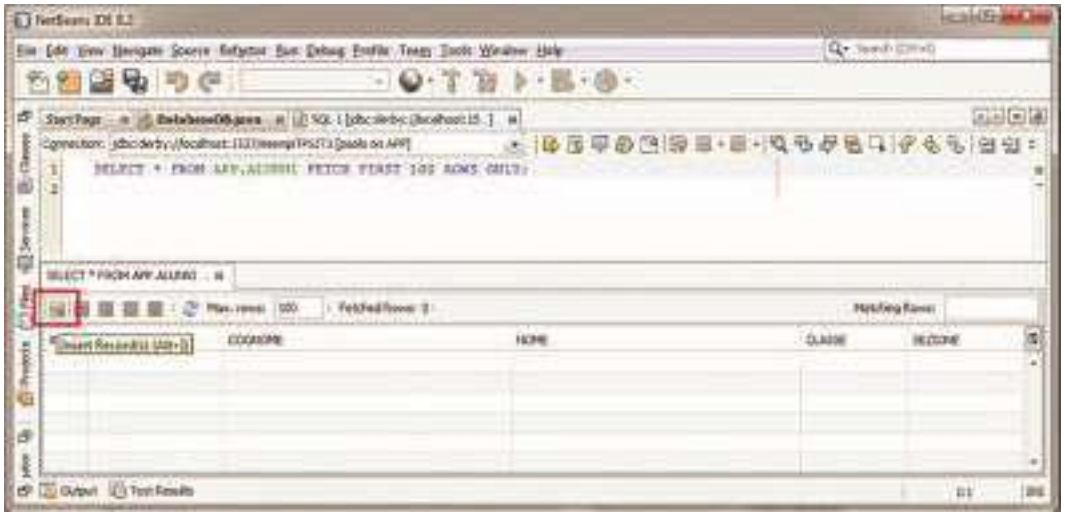
# 10 ESERCIZI IN LABORATORIO

Lo schema finale è il seguente:

Possiamo ora popolare manualmente la tabella con alcuni dati di prova, che utilizzeremo nei prossimi esempi: ci si deve posizionare sul nome della tabella, cliccare sempre con il tasto destro per visualizzare il menu e in questo cliccare sull'opzione View Data:




Ci viene visualizzato il contenuto della tabella che, naturalmente, è vuoto; clicchiamo sull'apposita icona e inseriamo alcuni record:




- creazione, eliminazione, modifica di tabelle;
- popolare le tabelle con i dati;
- visualizzazione di dati tabulari;
- esecuzione di istruzioni e query SQL.

# ESERCIZI IN LABORATORIO 10

## Creazione di un'applicazione Java

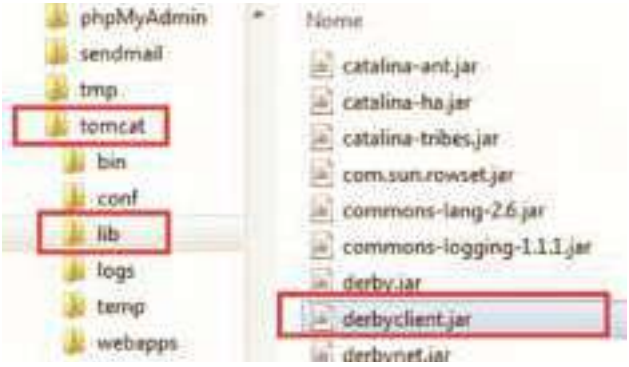
La realizzazione di un'applicazione è composta da due operazioni:
**1** la configurazione della risorsa origine dati su un Web server;
**2** il programma che accede al database da un servlet utilizzando l'origine dati configurata.

Il nostro esempio definisce la configurazione per il Web server Apache Tomcat.

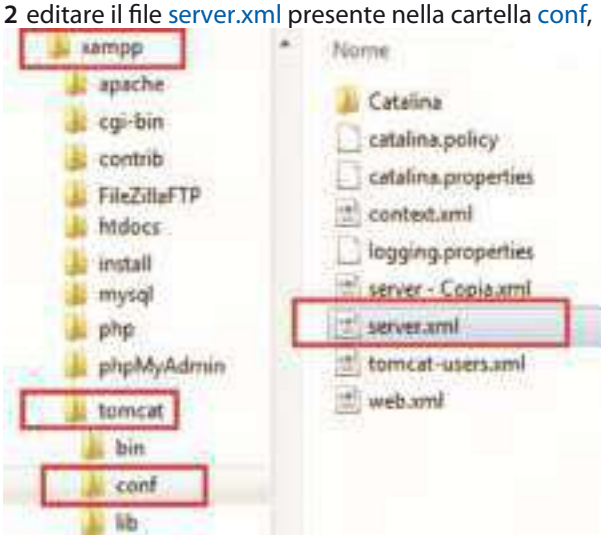
### Configurare la risorsa origine dati sul Web server

Per configurare il Web server bisogna innanzitutto effettuare il download della libreria dal sito http://db.apache.org/derby/, quindi:

**1** estrarre i file derby.jar e derbyclient.jar nella cartella ${TOMCAT_HOME}/lib,




**2** editare il file server.xml presente nella cartella conf,




**3** aggiungendo le seguenti righe nella sezione GlobalNamingResources:

```
-->
<GlobalNamingResources>
  <!-- Editable user database that can also be used by
       UserDatabaseRealm to authenticate users
  -->
  <Resource name="UserDatabase" auth="Container"
            type="org.apache.catalina.UserDatabase"
            description="User database that can be updated and saved"
            factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
            pathname="conf/tomcat-users.xml" />

<Resource auth="Container"
          driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
          maxActive="8" maxIdle="4"
          name="jdbc/javaDB" type="javax.sql.DataSource"
          url="jdbc:derby:esempiTPSIT3;create=true"
          username="paolo" password="123" />

</GlobalNamingResources>
```

**4** Per rendere operative queste opzioni è necessario riavviare Tomcat.

# 10 ESERCIZI IN LABORATORIO

Scriviamo ora la servlet: osserviamo come la sezione di connessione al database e le successive operazioni (query ecc.) siano simili a quelle già utilizzate per la connessione ai database precedentemente descritti (Access, MySQL).

```
public class ServletDerby1 extends HttpServlet
{

  private final String DRIVER     = "org.apache.derby.jdbc.EmbeddedDriver";    // stand alone
  private final String protocollo = "jdbc:derby:";                             // connessione alla libreria
  private final String dbpath     = "/xampp/tomcat/webapps/hcepli/esempioDB/"; // assoluto !=== funziona
  private final String dbName     = "esempiIPSIT3;memory=false";               // nome database
  private final String user       = "";                                        // nome utente
  private final String psw        = "";                                        // password

  // riferimento al database
  private final String URL_mioDB = protocollo + dbpath + dbName;
  // definizione del driver per la connessione al DB Access
```

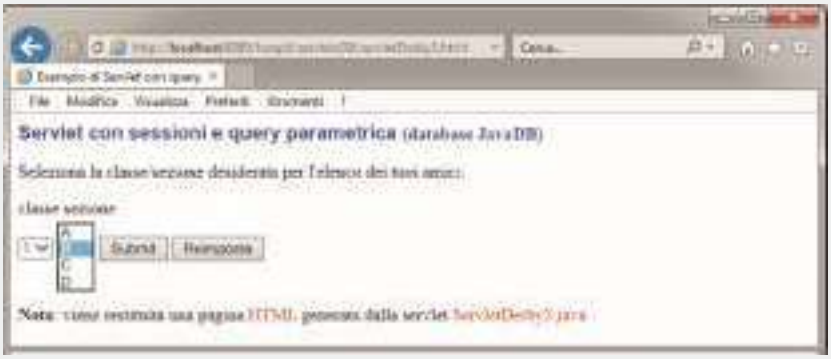
Il resto del codice è riportato nel file ServletDerby1.java.

## METTITI ALLA PROVA

• Creazione di una servlet

Utilizzando il database JavaDB precedentemente definito ti si chiede di:

**a** scrivere una servlet che seleziona mediante due combo box i nominativi presenti in una classe/sezione leggendoli dalla tabella alunni;




**b** scrivere una servlet che seleziona gli alunni di una città scelta da un combo box;

Realizza queste servlet utilizzando come strumento di sviluppo dapprima BlueJ e successivamente NetBeans, in modo da confrontare l'operatività dei due strumenti.

# Applicazioni a codice embedded in HTML con JSP

UNITÀ 5

**LEZIONE 1**
JSP: Java Server Page

**Lezione 2**
Java Server Pages e JavaBean

**Esercizi in Laboratorio**

- L1 JSP: primi esempi di programmazione
- L2 JSP con parametri da HTML
- L3 JSP e database MySQL
- L4 JSP e MDB
- L5 JSP e database Derby
- L6 JSP Bean e database Access, MySQL e Derby

## CONOSCENZE

- Conoscere le caratteristiche delle pagine JSP
- Conoscere le caratteristiche dei Bean
- Acquisire le caratteristiche dell'interfaccia JDBC
- Conoscere i tipi di driver per la connessione ai database

## COMPETENZE

- Realizzare un'applicazione Web
- Riconoscere i componenti di una pagina JSP
- Saper realizzare un'applicazione con JSP e Bean
- Generare un file.WAR

## ABILITÀ

- Realizzare un'applicazione WEB dinamica con pagine JSP
- Richiamare un Bean in una pagina JSP
- Connettere le applicazioni Web JSP con MySQL, Access e Derby
- Connettere i JSP Bean ai database

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

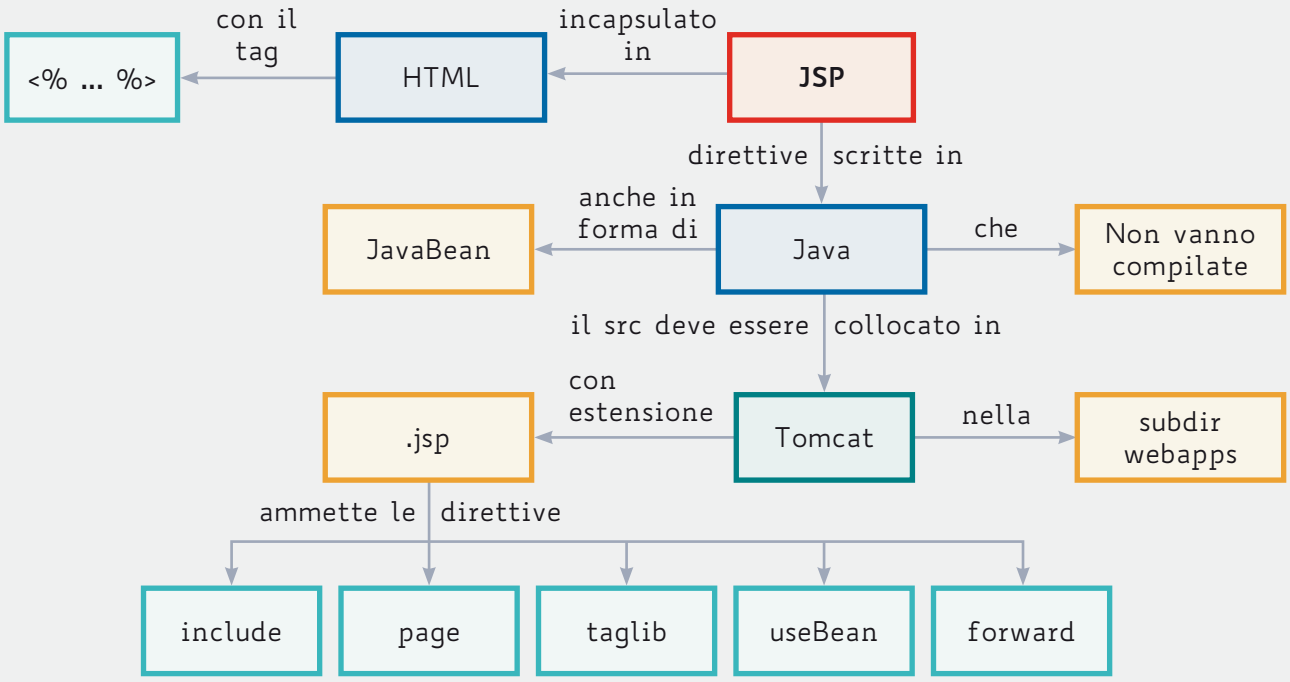
# 1 JSP: Java Server Pages

**IN QUESTA LEZIONE IMPAREREMO...**

- le caratteristiche di una pagina JSP
- i componenti di una pagina JSP
- a realizzare una applicazione Web dinamica con pagine JSP

## MAPPA CONCETTUALE

didattica inclusiva




## Le Java Server Pages (JSP)

La tecnologia Java Server Pages (JSP) è basata su Java e permette lo sviluppo di applicazioni web con contenuti dinamici senza dover essere legati a un ambiente in particolare.
Come con le servlet, anche con le JSP è possibile avere una netta separazione dei ruoli tra la presentazione grafica della pagina e la logica di gestione scritta appunto in Java.
Una pagina JSP è rappresentata da codice HTML con incapsulate al suo interno delle direttive JSP, racchiuse in un particolare tag <%...%>, rappresentanti codice Java: a differenza delle servlet, dove è necessario compilare il codice Java e collocare il file.class nella directory classes prima di iniziare il caricamento delle pagine della wap, con le pagine JSP la compilazione *avviene in automatico*, al momento del caricamento della pagina stessa che deve avere estensione .jsp.

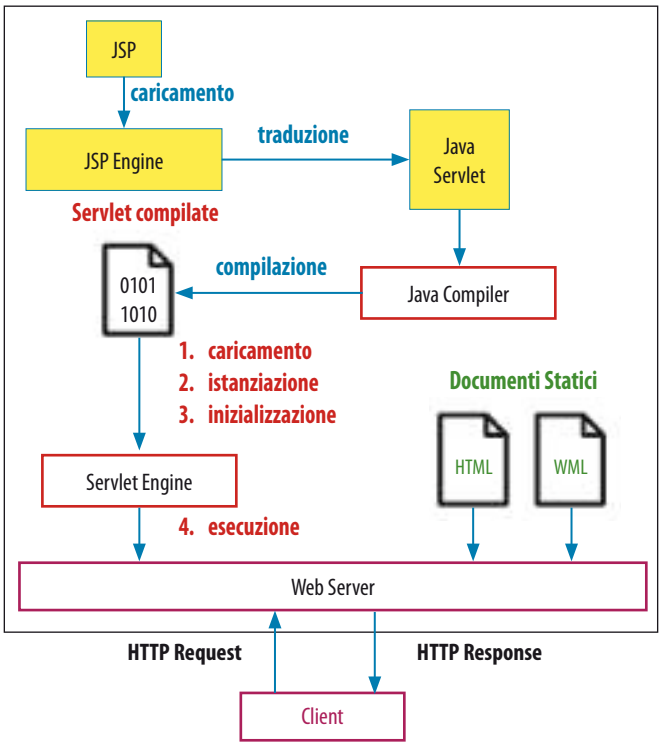
Infatti, alla prima richiesta di una pagina JSP il server effettuerà i seguenti passi:

- compila il codice JSP per ottenere il codice Java rappresentante una servlet; in pratica estrae i comandi Java e ne ricava il codice della classe;
- invia alla nuova classe la richiesta Http per ottenerne la risposta desiderata servendosi di un programma pre-processore che elabora la risorsa rimpiazzando i tag con codice HTML;
- spedisce infine la pagina risultante al browser del client.

Questo procedimento avviene solo alla prima richiesta di caricamento di una JSP dato che la classe prodotta viene mantenuta in cache dal servlet engine così da permettere una rapida risposta alle richieste successive.

La differenza sostanziale con le servlet è che queste sono una classe Java che utilizza il protocollo HTML per ricevere e inviare delle richieste tra client e server, mentre una pagina JSP è una pagina che racchiude sia del codice statico (HTML) che del codice dinamico (Java) che verrà eseguito sul server, secondo lo schema operativo delle pagine PHP o ASP.

Lo schema completo che riporta il meccanismo di funzionamento di una JSP è il seguente:



Naturalmente il server deve avere a disposizione il compilatore Java (javac.exe) per poter generare le classi: non basta il JRE, sufficiente per eseguire le servlet.

## Caratteristiche delle pagine JSP

La programmazione di pagine JSP è una "fusione" tra la programmazione dinamica in linguaggio PHP e quella effettuata mediante la realizzazione delle servlet: il programmatore JSP "pensa" come il programmatore PHP di inserire "frammenti" di codice di linguaggio di programmazione "immersi" in tag di linguaggio nell'HTML ma è consapevole che alla sua prima esecuzione la pagina verrà trasformata in una servlet equivalente, e tale servlet può anche essere testata a parte.

È agevole lo sviluppo e la prototipazione rapida in quanto, essendo la compilazione automatica, basta apporre le modifiche e richiamare la pagina direttamente dal browser per vederne il risultato.

Inizialmente le Web Application si basavano solo sulle servlet, ma erano una tecnologia alla portata dei programmatori più esperti dato che in una classe Java bisognava inserire sia il codice gestionale che il codice che curava il layout della pagina Web; l'introduzione delle JSP è stata fatta per semplificare il lavoro dei Web designer che potevano così dedicarsi al layout della pagina senza entrare in merito della logica di gestione dell'applicazione.
Con la tecnologia JSP è possibile utilizzare strumenti come i **JavaBean** per "spostare" tutta la logica al loro interno in modo da separarla dalla grafica e da ridurre al minimo l'utilizzo delle scriptlet <% … %>; in tal modo, per apportare le successive modifiche si procede in due direzioni diverse:

1 in caso di modifiche estetiche viene aggiornato il template HTML dell'applicazione;
2 in caso di modifiche logiche si modificano le righe JSP o si riscrivono i Bean.

**JavaBean**
Lo strumento JavaBean è un componente Java al 100% che opera all'interno di una qualsiasi macchina virtuale; è una classe Java che implementa l'interfaccia java.io.Serializable e utilizza metodi pubblici get/set per esporre gli attributi. Tali metodi verranno descritti nella prossima Lezione.

Utilizzando questa tecnologia, una volta creata una pagina JSP, questa può essere mantenuta o modificata da chiunque conosca anche solo l'HTML.

## Tag in una pagina JSP

In un pagina JSP le parti variabili sono contenute all'interno di tag speciali che possono avere due tipi diversi di sintassi:

- Scripting-oriented tag;
- XML-oriented tag.

### Scripting-oriented tag

Le scripting-oriented tag sono definite da delimitatori entro cui è presente lo scripting (*self-contained*) e possono essere così classificati:

- commenti JSP: hanno sintassi: <%-- commento --%> e vengono eliminati dal container in fase di compilazione/traduzione della pagina; i commenti HTML sono considerati come normale elemento di testo;
- elementi di scripting: codice Java (o altro *scripting language*) da inserire nella servlet risultante:
  - scriptlet: con sintassi <% code %>; possono contenere variabili locali valide solo per una singola esecuzione della servlet (prodotta in seguito alla compilazione della JSP);
  - dichiarazioni: con sintassi <%! dichiarazioni [dichiarazioni]...%>; sono metodi o variabili la cui durata è quella della servlet (conservano il loro valore per tutte le esecuzioni della servlet);
  - espressioni: con sintassi <%= espressioni %>; l'espressione viene valutata e scritta nella pagina di risposta della JSP;
- direttive JSP: con sintassi <%@ direttiva %>; controllano la struttura della servlet risultante.

Inoltre sono presenti altri due tipi di istruzioni:

- template text: tutte le parti di testo che non sono definite come elementi JSP sono copiate nella pagina di risposta;
- azioni JSP: istruzioni specifiche che controllano il comportamento del motore JSP.

### XML-oriented tag

Le XML-oriented tag utilizzano come sintassi la notazione tipica delle pagine XML e permettono di definire i medesimi tipi di comandi dei tag scripting-oriented, e sono anche chiamate Actions o Standard Actions. Anch'esse contemplano i quattro tipi precedentemente elencati:

- <jsp:declaration> dichiarazione </jsp:declaration>
- <jsp:expression> espressioni </jsp: expression>
- <jsp:scriplet> code </jsp: scriplet>
- <jsp:directive.dir_type direttiva/>

# Tag scripting-oriented

## Espressioni

Vediamo ora un esempio di pagina JSP con espressioni: per valutare le espressioni Java si usano i delimitatori <%= e %>: il risultato viene convertito in stringa e inserito nella pagina al posto del tag.

Scriviamo una semplice JSP che visualizzi data e ora correnti:

```
oggi.jsp
1  <HTML>
2  <BODY>
3  <TITLE> Primo codice JSP </TITLE>
4  Ciao, la data di oggi &egrave;
5  <%= new java.util.Date().toString() %>  <!--espressione-->
6  </BODY>
7  </HTML>
```

Tutti i nostri codici sorgenti verranno collocati nella cartella: hoepli/jsp.



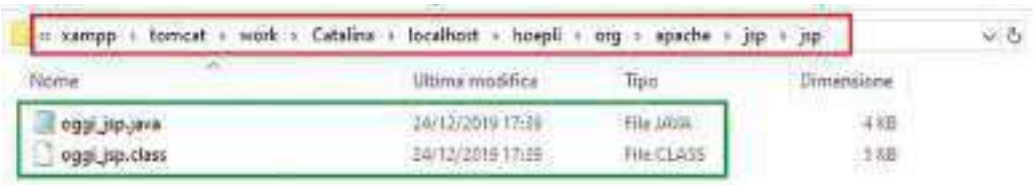

Nella figura è possibile vedere la struttura completa dell'albero a partire del Web server:

Lo mandiamo in esecuzione digitando nel browser semplicemente il nome della pagina (con il suo percorso completo): nel browser avremo un output come quello della seguente figura:

**AREA DIGITALE**

**Esempio di codice Java generato da Tomcat: la classe oggi_jsp.java**

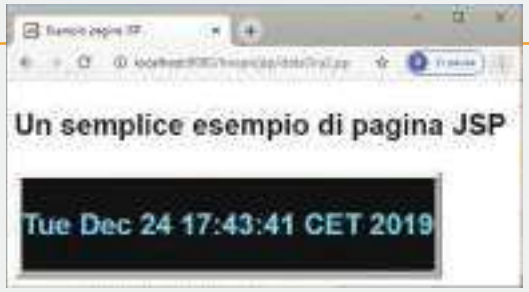
Come detto in precedenza, il container Tomcat ha convertito la pagina JSP in una servlet (classe oggi_jsp.java), ha compilato la classe generando il bytecode e lo ha mandato in esecuzione. memorizzando il codice sorgente e la classe generata al seguente indirizzo:



## METTITI ALLA PROVA

• Scrittura codice JSP • Posizionamento file nel Web server

Realizza la tua prima JSP modificando lo script di pagina precedente e inserendo nello stesso segmento di codice una parte di codice che migliora graficamente l'output, come per esempio quello riportato nella figura, magari definendo un foglio di stile. Confronta la tua soluzione con quella nel file dataOra2Sol.jsp e visualizza la classe dataOra2_jsp.java generata da Tomcat.




## Dichiarazioni

Per dichiarare variabili e metodi si utilizzano i delimitatori <%! e %> che vengono poi referenziati in qualsiasi punto del codice JSP, richiamandoli all'interno di una espressione con <%= e %>.

La pagina JSP è trasformata in una classe servlet e i metodi così dichiarati diventano metodi della servlet: il container ne crea un'istanza e a ogni richiesta genera in corrispondenza un thread.

## ESEMPIO

Integriamo l'esempio precedente, su data e ora, con due dichiarazioni, rispettivamente di una stringa (*utente*) e di un metodo (getTotale()) e salviamolo nella pagina acquisti.jsp:

```
acquisti.jsp
1  <HTML>
2  <BODY>
3  <TITLE> Esempio di codice JSP </TITLE>
4  <!--  dichiarazioni          -->
5  <%!
6   String utente = "Anna Maria";
7   double[] prezzi = {11.5, 73.8, 121.5};
8   double getTotale(){
9     double totale = 0.0;
10    for (int x = 0; x < prezzi.length; x++)
11      totale += prezzi[x];
12    return totale;
13  }
14 %>
15 <!--  espressioni          -->
16 <P>Gentile cliente <%=utente%>, oggi <%= new java.util.Date().toString()%></P>
17 <P>ha effettuato acquisti per Euro: <%=getTotale()%>.</P>
18 </HTML>
19 </BODY>
```

## Scriptlet

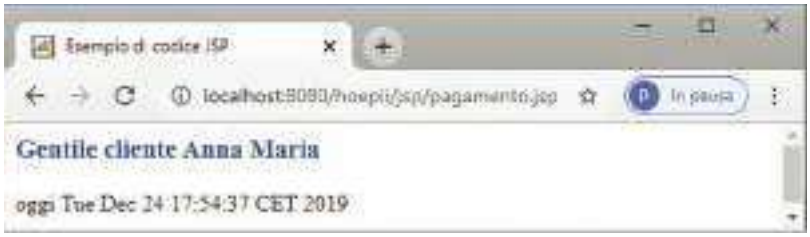
Nella pagina JSP è possibile inserire frammenti di codice Java, delimitati dal tab <% e %>, che prendono il nome di scriptlet e possono comprendere le istruzioni di controllo per la gestione del flusso di esecuzione del codice, quali la selezione e l'iterazione, tipiche dei linguaggi di programmazione: quando la pagina viene tradotta all'atto della sua esecuzione tutti gli scriptlet divengono un unico blocco di codice Java.
Vedremo che è anche possibile indicare con la direttiva page il linguaggio di programmazione per le scriptlet, che di default è Java ma potrebbe anche essere un qualunque linguaggio di scripting.

### ESEMPIO

Sempre nell'esempio precedente aggiungiamo una condizione sull'ammontare totale dell'importo che deve essere corrisposto per gli acquisti, permettendo un pagamento rateale se questo è superiore a 1000 Euro (file pagamento.jsp).

```
pagamento.jsp
1  <HTML>
2  <BODY>
3  <TITLE> Esempio di codice JSP </TITLE>
4  <!--  dichiarazioni                  -->
5  <%!
6    String utente = "Anna Maria";
7    double totale = 1230.50;
8  %>
9  <!--  espressioni                    -->
10 <P><FONT COLOR="#0000FF"><H3>Gentile cliente <%=utente%></H3></FONT>
11 oggi <%= new java.util.Date().toString() %> </P>
12 <P>ha effettuato acquisti per Euro: <%= totale %>;</P>
13 <!--  scriptlet                      -->
14 <% if (totale > 1000){ %>
15    <H3>puo' effettuare un pagamento rateizzato </H3>
16 <% } else { %>
17    <H3>devi pagare in una sola soluzione.</H3>
18 <% } %>
19 </HTML>
20 </BODY>
```

Mandandolo in esecuzione avremo il seguente output:




## Oggetti impliciti

Il blocco di codice specificato dalla scriptlet viene inserito all'interno del metodo service() della servlet corrispondentemente generata alla richiesta di esecuzione della pagina JSP (più precisamente all'interno del metodo _jspService(), richiamato da service()). Avremo quindi implicitamente a disposizione i seguenti nove oggetti:

| OGGETTO | CLASSE/INTERFACCIA |
|---|---|
| page | javax.servlet.jsp.HttpJspPage |
| config | javax.servlet.ServletConfig |
| request | javax.servlet.http.HttpServletRequest |
| response | javax.servlet.http.HttpServletResponse |
| out | javax.servlet.jsp.JspWriter |
| session | javax.servlet.http.HttpSession |
| application | javax.servlet.ServletContext |
| pageContext | javax.servlet.jsp.PageContext |
| exception | Java.lang.Throwable |

Descriviamoli sinteticamente uno per uno:

## page

L'oggetto page rappresenta l'istanza corrente della servlet: ha come tipo l'interfaccia HttpJspPage che discende da JSPpage, la quale a sua volta estende servlet e quindi può essere utilizzata per accedere a tutti i metodi definiti nelle servlet.

## config

Contiene la configurazione della servlet (parametri di inizializzazione) e generalmente è lasciato inutilizzato, visto che le informazioni messe a disposizione da questo oggetto implicito sono generalmente settate e rilevate in automatico. Mette a disposizione i seguenti due metodi:

- getInitParameterName(): restituisce tutti i nomi dei parametri di inizializzazione;
- getInitParameter(name): restituisce il valore del parametro passato per nome.

## request

È l'oggetto di classe HttpServletRequest che rappresenta la richiesta Http che ha portato all'attivazione della pagina JSP/servlet: fornisce i metodi di accesso alla request Http corrente.

## response

È l'oggetto di classe HttpServletResponse che rappresenta la risposta Http da inviare al client.

## out

È un oggetto di classe JSPWriter.out e rappresenta il flusso di output su cui viene prodotta la pagina Web: spesso il suo uso è implicito, ma all'occorrenza si può fare riferimento a questa variabile invocandone i metodi nella tabella che segue.

| METODI | DESCRIZIONE |
|---|---|
| `isAutoFlush()` | Ritorna true se il buffer di output è automaticamente svuotato quando è pieno, false se viene generata un'eccezione. |
| `getBuff erSize()` | Restituisce la dimensione (in byte) del buffer di output. |
| `getRemaining()` | Restituisce la dimensione (in byte) della parte inutilizzata del buffer di output. |
| `clearBuff er()` | Cancella i contenuti del buffer di output, scartandoli. |
| `clear()` | Cancella il contenuto del buffer di output, segnalando un errore se il buffer è stato precedentemente svuotato |
| `newLine()` | Scrive un separatore di riga (specifico per la piattaforma) nel buffer di output. |
| `flush()` | Svuota il buffer di output, quindi il flusso di output. |
| `close()` | Chiude il flusso di output, perdendo qualsiasi contenuto. |

## session

È l'oggetto di classe HttpSession che rappresenta la sessione Http all'interno della quale è stata invocata la pagina JSP: fornisce le informazioni sul contesto di esecuzione della JSP, cioè della sessione corrente di esecuzione per l'utente.

## application

È un oggetto che fornisce informazioni sul contesto di esecuzione della JSP e permette di accedere e di memorizzare gli oggetti per renderli accessibili da qualsiasi utente e modificabili da ogni pagina.

## pageContext

È un oggetto di classe PageContext, che rappresenta l'insieme degli oggetti impliciti associati all'intera pagina: l'oggetto può essere trasferito da una pagina JSP a un'altra ma viene poco utilizzato per lo scripting.

## exception

È un oggetto connesso alla gestione degli errori che viene utilizzato nelle Error Page, quelle che sono appositamente dichiarate con l'attributo errorPage impostato a true, e rappresenta l'eccezione che non viene gestita da nessun blocco catch.

## ESEMPIO

Realizziamo un semplice contatore di accessi gestito con una sessione:

```
<%! Integer x = new Integer(0); %>
<%
 if(session.isNew())                          // se non c'e' una sessione la crea
 {
   session.putValue("contatore", x);                // mette x nella sessione
 }
 else
 {
   x = (Integer)session.getValue("contatore");        // legge  dalla sessione
   if(x == null)                                      // se x non /presente
     session.putValue("contatore", new Integer(1));   // lo inserisce con valore 1
   else                                               // x la incrementa e la salva
     session.putValue("contatore", new Integer(x.intValue() + 1));
 }
%>
<HTML>
<BODY>
<P>Numero di accessi: <%= x.intValue() %> </P>
<P><a href="sessioni.jsp">richiama la JSP</a></P>
</BODY>
```

Una sua esecuzione, dopo che viene richiamata la JSP alcune volte, ha il seguente output:

## Direttive

Sono comandi JSP valutati a tempo di compilazione e quelle più utilizzate sono:

- include: permette di includere un altro documento;
- page: permette di importare package, dichiarare pagine d'errore, definire il modello di esecuzione della JSP relativamente alla concorrenza ecc.;
- taglib: carica una libreria di custom tag implementata dallo sviluppatore.

**AREA DIGITALE**

**XML-oriented tag o JSP Standard Actions**

## Direttiva include

La sintassi della direttiva include è la seguente:

```
<%@include file="nomeFile.xxx"%>
```

Permette di includere sia file statici (HTML) che altre JSP.

## ESEMPIO

Di seguito è riportato un semplice esempio che utilizza la direttiva include per inserire una intestazione nella pagina includi.jsp richiamando una pagina HTML dove è presente una immagine:

```
<HTML>
<BODY>
<TITLE>JSP con direttiva include </TITLE>
<!--  dichiarazioni                  -->
<%!
String utente = "Anna Maria";
%>
<!--  direttiva include        -->
<%@ include file = "intestazione.html"%>
<!--  espressioni                -->
<P>Gentile cliente <%= utente %></P>
<P>La ringraziamo per aver visitato il nostro catalogo.</P>
</HTML>
</BODY>
```

Mandando in esecuzione la pagina JSP avremo il seguente output:

La direttiva include diventa indispensabile quando vogliamo creare dei metodi da utilizzare in più pagine, cioè per crearci una libreria di metodi/funzioni da utilizzare in ogni nostra pagina JSP.
Riassumendo possiamo dire che con la direttiva include, è come copiare il testo (sia statico sia codice JSP) all'interno della pagina stessa. È un metodo molto comodo perché evita la ridondanza di codice senza avere alcun effetto sulle prestazioni.

## METTITI ALLA PROVA

- Utilizzo Direttiva include

1. In un file (funzioni.jsp) scrivi due metodi: il primo che riceve in input un testo e restituisce una stringa compatibile nel linguaggio SQL e il secondo metodo che dato un oggetto ci restituirà la sua trasformazione in stringa o nel caso di oggetto nullo una stringa vuota.
2. Scrivi poi la pagina che includerà e utilizzerà tale file (postback.jsp): questa deve permettere l'inserimento del nome e del cognome di un utente e quindi di inviare a se stessa tali dati dopo averli trasformati richiamando entrambe le funzioni prima definite.

## Direttiva page

La sintassi della direttiva page è la seguente:

```
<%@page attributo="valore"%>
```

**AREA DIGITALE**

**Elenco attributi**

Essa permette di specificare alcune proprietà della pagina JSP che possono essere usate sia al momento della compilazione per costruire il corpo di alcuni metodi della servlet oppure in fase di runtime.

## ESEMPIO

Per esempio, attribuiamo all'attributo info un valore:

```
<HTML>
<BODY>
<%@ page info = "Esempio di uso direttiva page." %>
<P>
  Informazione sulla pagina: <%= getServletInfo()%>
</P>
</BODY>
</HTML>
```

L'esecuzione della JSP produce il seguente output:

## Direttiva taglib

Nelle JSP è possibile definire dei tag addizionali che possono essere usati nella pagina nell'HTML definendone il loro comportamento in apposite librerie, le tag library, che vengono importate con la direttiva taglib.

Sono disponibili tag library standard prodotte da terze parti e messe a disposizione della comunità Java oppure potete realizzare voi stessi le vostre librerie e riutilizzarle in tutte le vostre applicazioni.

La sintassi della direttiva taglib per includere una tag library è:

```
<%@taglib uri="libreria" prefix="prefisso" %>
```

Il prefisso indicato è quello che verrà utilizzato all'interno della pagina per specificare i tag presenti nella libreria (sarà il prefisso di ogni tag presente nella libreria).

### ESEMPIO

Se in una pagina JSP utilizziamo la taglib definita in miaTlib.tld che contiene per esempio la classe SfondoBlu, la definisco e la utilizzo nel modo seguente:

```
1 <%@ taglib uri="/miaTlib.tld" prefix="miaTag" %>
2 <HTML>
3 <HEAD><TITLE> Esempi di TAGLIB </TITLE></HEAD>
4 <BODY>
5 <miaTAG:SfondoBlu/>
6 </BODY>
7 </HTML>
```

Avviando l'esecuzione dello script, quando viene individuato questo tag, Tomcat richiama gli opportuni metodi dalla classe associata al nome SfondoBlu nel file miaTlib.tld.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 L'acronimo WAR significa:**
- **a** Web Application Repository
- **b** Web Archives Repository
- **c** Web Application Root
- **d** Web Archives Root

**2 Quale tra i seguenti oggetti NON è implicito direttamente in service()?**
- **a** page
- **b** config
- **c** request
- **d** response
- **e** in
- **f** out
- **g** session
- **h** application
- **i** pageContext
- **j** exception

**3 Quale tra le seguenti non è una direttiva?**
- **a** include
- **b** session
- **c** taglib
- **d** page

**4 Quale tra i seguenti non è un attributo della direttiva page?**
- **a** language
- **b** import
- **c** buffer
- **d** session
- **e** include
- **f** info

## ASSOCIAZIONE

**Associa a ogni TAG il corrispondente significato.**

| | |
|---|---|
| **1** ..... <%= … %> | **a)** dichiarazione |
| **2** ..... <%! … %> | **b)** direttiva |
| **3** ..... <% … %> | **c)** scriptlet |
| **4** ..... <%@ … %> | **d)** espressione |

## VERO/FALSO

1. Una applicazione Web viene impacchettata come file WAR. V F
2. Una JSP è un'astrazione di alto livello per rappresentare delle servlet. V F
3. Una pagina JSP è formata da codice HTML con incapsulato al suo interno delle direttive JSP. V F
4. Una pagina JSP prima viene compilata e quindi viene copiata nella cartella classes. V F
5. Come per le servlet il server deve avere a disposizione l'ambiente JRE. V F
6. Le scripting-oriented tag sono definite da delimitatori entro cui è presente lo scripting. V F
7. Le XML-oriented tag permettono di definire i medesimi tipi di comandi dei tag scripting-oriented. V F
8. A ogni chiamata di una pagina JSP il container ne crea un'istanza (genera un thread). V F
9. Le scriptlet in una pagina JSP devono necessariamente essere scritte in Java. V F
10. Le direttive non producono nessun output visibile. V F

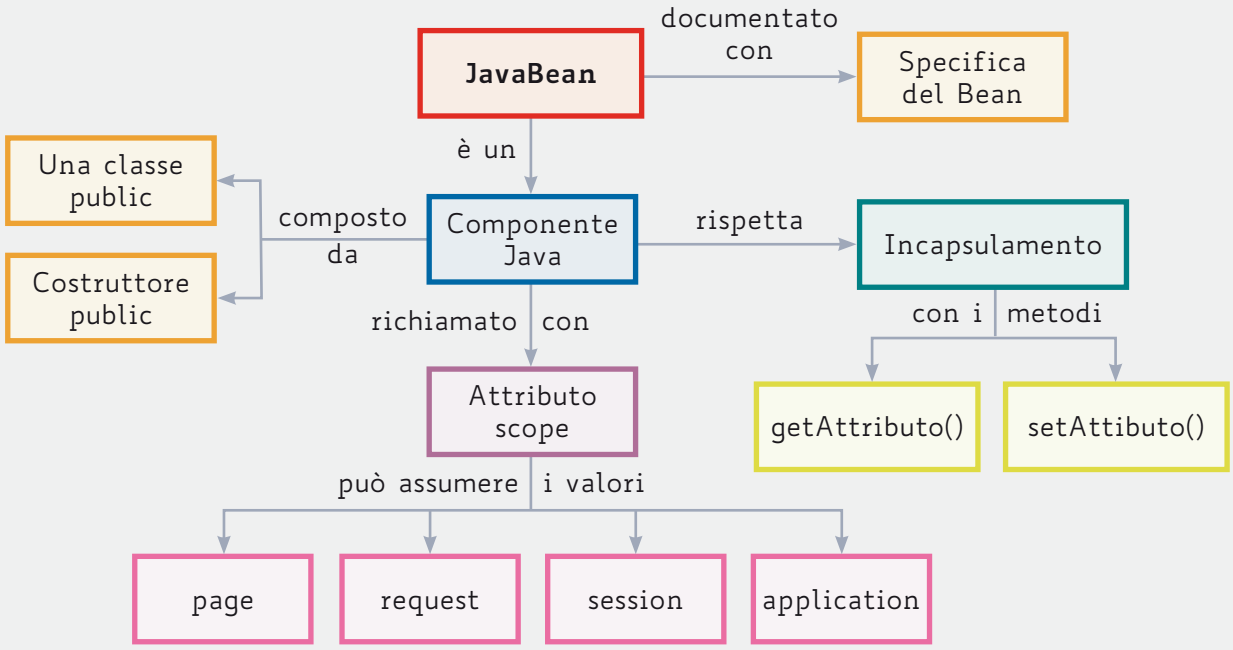
# 2 Java Server Pages e JavaBean

**IN QUESTA LEZIONE IMPAREREMO...**

- le caratteristiche dei Bean
- a richiamare un Bean in una pagina JSP
- a realizzare una applicazione con JSP e Bean

## MAPPA CONCETTUALE

didattica inclusiva

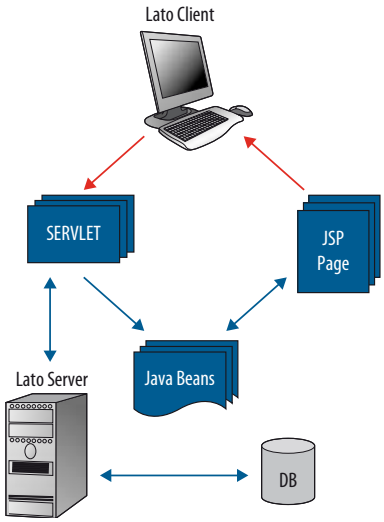


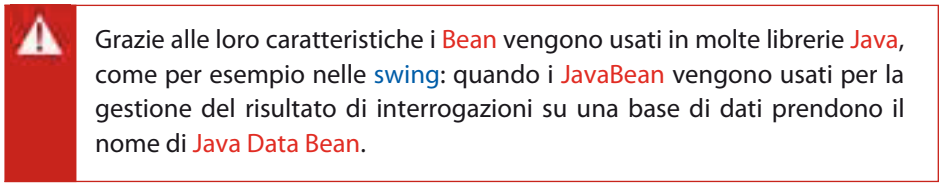

## JavaBean

All'interno delle pagine JSP è possibile utilizzare particolari classi Java con un'interfaccia molto semplice: sono i JavaBean (letteralmente "chicchi di Java", cioè "chicchi di caffè").
Un JavaBean è un componente nella tecnologia Java che può essere utilizzato in modo standard in più applicazioni.

Lo scopo dei componenti è quello di essere *riutilizzati* in contesti diversi e permettere di separare la logica dell'applicazione dalla parte di controllo e di presentazione delle informazioni.

Un JavaBean è quindi una classe particolarmente semplice che risponde sostanzialmente ai seguenti requisiti:
- è una classe public;
- ha un costruttore public di default con zero argomenti;
- soddisfa una serie di direttive relative ai suoi metodi e ai suoi attributi (detti proprietà).

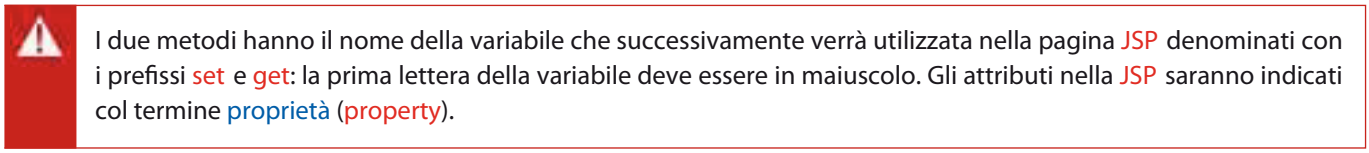


Grazie alle loro caratteristiche i Bean vengono usati in molte librerie Java, come per esempio nelle swing: quando i JavaBean vengono usati per la gestione del risultato di interrogazioni su una base di dati prendono il nome di Java Data Bean.

## Incapsulamento degli attributi

I JavaBean rispettano le direttive che realizzano l'incapsulamento, cioè che tutti gli attributi devono essere di tipo privato e a essi si accede solo con i metodi *setter* e *getter*.
In altre parole per ogni attributo sono definiti due metodi:
- getAttributo(): metodo che permette la lettura dell'attributo;
- setAttributo(): metodo che permette la modifica dell'attributo.



I due metodi hanno il nome della variabile che successivamente verrà utilizzata nella pagina JSP denominati con i prefissi set e get: la prima lettera della variabile deve essere in maiuscolo. Gli attributi nella JSP saranno indicati col termine proprietà (property).

Le proprietà di tipo boolean seguono una regola leggermente diversa: il metodo di lettura ha la forma isAttributo() anziché getAttributo().
Vediamo un esempio, che ci servirà successivamente per realizzare il primo "collegamento" tra una pagina JSP e una classe Java, dove la variabile che vogliamo utilizzare è la stringa testo:
Definiamo la classe e il suo costruttore:

package che raggruppa i nostri Bean

per poter essere utilizzate da uno strumento esterno alla classe

```
1  package mieiBean;
2  import java.util.*;
3  public class CiaoBean implements java.io.Serializable
4  {
5    private String testo;
6    public CiaoBean() {
7      testo = " ";
8    }
```

Questo semplice esempio ha un solo attributo, testo, che viene inizializzato nell'unico metodo costruttore che deve essere privo di parametri.

```
10   public String getTesto() {
11     GregorianCalendar dataAttuale=new GregorianCalendar();
12     int ore = dataAttuale.get(GregorianCalendar.HOUR_OF_DAY);
13     if ( ore < 12 ) {
14       testo = "Buongiorno: e' mattina!";
15     } else if ( ore < 17 ) {
16       testo = "Buon pomeriggio!";
17     } else {
18       testo = "Buona sera !";
19     }
20     return testo;
21   }
```

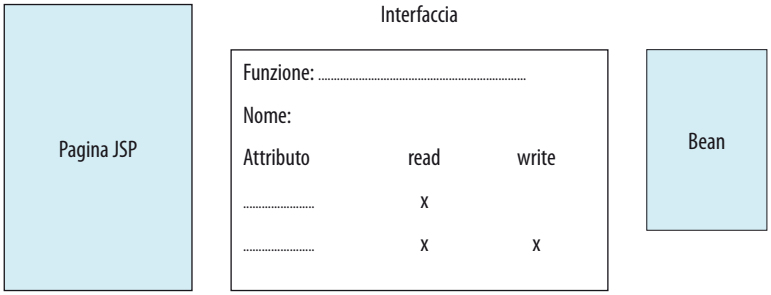
Non esiste alcun vincolo di obbligatorietà tra il nome dei metodi e il nome dell'attributo che gestiscono: per esempio avremmo potuto chiamare il metodo che modifica l'attributo testo col nome getParola() oppure getSaluto(). Vedremo che le variabili di sessione che usano la pagina JSP come collegamento al Bean sfruttano il nome dei metodi e non quello dell'attributo: per comodità (e per non creare confusione) si consiglia di denominarli allo stesso modo.

## Specifica del Bean

Ai Bean è comodo associare nella documentazione la cosiddetta scheda di "specifica del Bean" in modo da semplificarne l'utilizzo nelle JSP (e anche nelle servlet!).
La scheda di "specifica del Bean" descrive la funzione svolta dal Bean, il nome della classe e degli attributi indicandone i possibili utilizzi, cioè se di sola lettura, di scrittura o di lettura/scrittura.
Per l'esempio precedente possiamo avere:

| | Specifica del Bean | | | |
|---|---|---|---|---|
| Funzione | In base all'ora del giorno predispone un saluto diverso | | | |
| Nome | CiaoBean | | | |
| | | | | |
| Proprietà | Tipo | Read (get) | Write (set) | Significato |
| testo | String | X | | Messaggio di saluto |

Come vedremo la presenza di questa scheda permette di utilizzare il Bean senza conoscerne il contenuto: di fatto si realizza quella che è la filosofia della programmazione a oggetti, dove le classi sono delle black box che vengono utilizzate esclusivamente mediante la loro interfaccia pubblica.

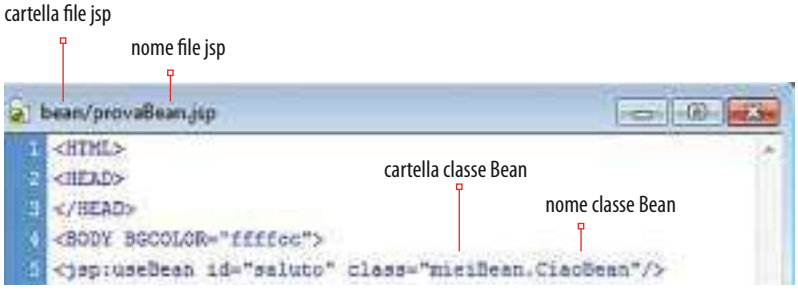



# Uso di JavaBean

Java permette di automatizzare la chiamata a un metodo di un Bean da parte di una pagina JSP mediante un insieme di elementi di azione standard, denotati del tag <jsp: … >.

È quindi possibile utilizzare una classe separando la realizzazione del codice Java dalla realizzazione delle pagine JSP.

Per utilizzare un Bean in una pagina JSP si utilizza la sintassi della riga 5:



Con questa istruzione si inizializza un'istanza della classe Bean CiaoBean in un oggetto di nome (o id) saluto: da ora in poi nella nostra pagina JSP possiamo fare riferimento ai metodi della classe Bean richiamandoli sull'oggetto di riferimento appena creato, cioè saluto.
La sintassi completa dell'istruzione è la seguente:

```
<jsp:useBean id="saluto" class="CiaoBean" scope="page|request|session|application"/>
```

l'attributo scope è opzionale e permette di definire l'ambito di accessibilità e il tempo di vita dell'oggetto (il valore di default è page). I possibili valori sono riportati nella seguente tabella:

| SCOPE | ACCESSIBILITÀ | TEMPO DI VITA |
|---|---|---|
| `page` | Solo la pagina corrente. | Fino a quando la pagina viene completata o fino al forward. |
| `request` | La pagina corrente, quelle incluse e quelle a cui si fa il forward. | Fino alla fine dell'elaborazione della richiesta e alla retribuzione della risposta. |
| `session` | Richiesta corrente e tutte le altre richieste dello stesso client. | Tempo di vita della sessione. |
| `application` | Richiesta corrente e ogni altra richiesta che fa parte della stessa applicazione. | Tempo di vita dell'applicazione. |

Per accedere e modificare gli attributi della classe Bean, qui chiamati property, si utilizzano rispettivamente i seguenti tag: <jsp:getProperty> e <jsp:setProperty>.
Leggiamo per esempio il contenuto dell'attributo testo con la seguente istruzione:

```
11 <jsp:getProperty name="saluto" property="testo"/>
```

Operazione — Oggetto — Attributo

che equivale a dire: "dell'oggetto di nome saluto prendi il risultato del metodo get sulla proprietà testo", cioè esegui il metodo getTesto().
Un tag simile ci permette di scrivere all'interno del Bean, modificando il valore dell'attributo desiderato:

```
19 <jsp:setProperty name="saluto" property="testo" value= "ciao"/>
```

Mandiamo in esecuzione il nostro programma e otteniamo:

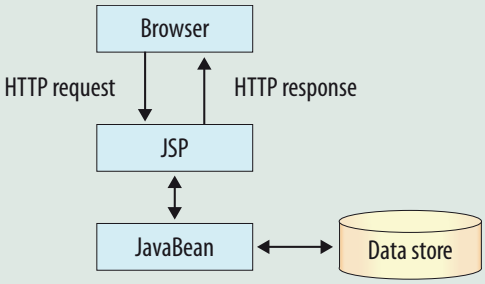
I codici completi sono disponibili nei file provaBean.jsp e CiaoBean.java nella cartella MATERIALI.

## PER SAPERNE DI PIÙ

**MODEL 1**

L'architettura J2EE a due livelli costituita da:
- JSP per il livello di presentazione;
- JavaBean per il livello di business logic;

viene denominata Model 1 e può essere schematizzata come mostrato nel disegno a lato.



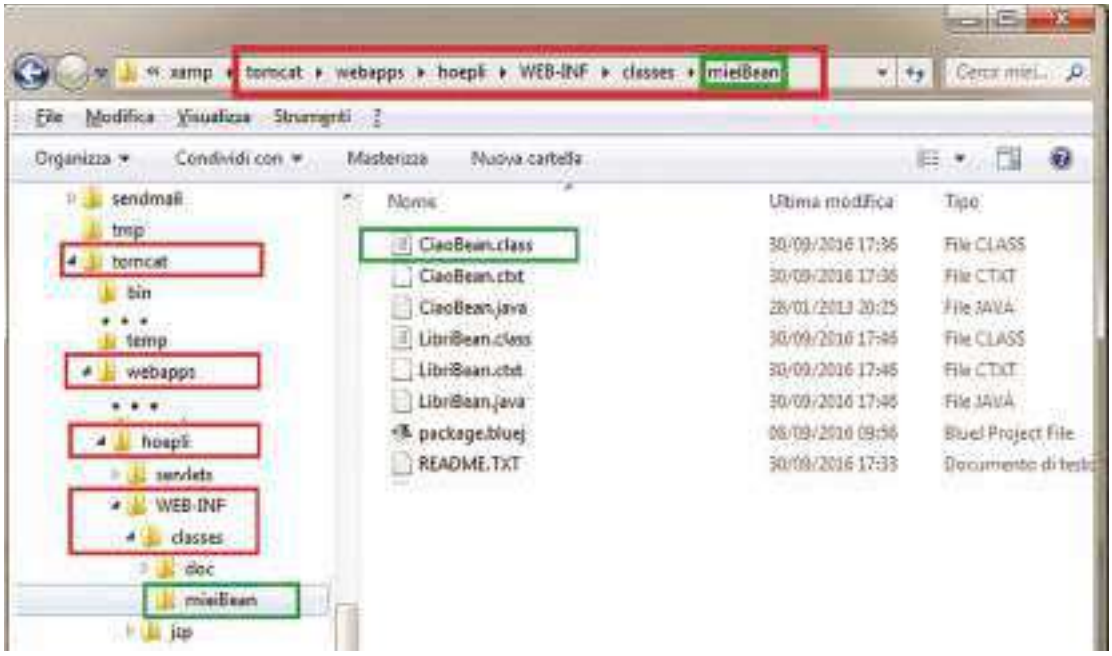
## Configurazione dell'applicazione

Un'applicazione Web che utilizza Bean deve essere opportunamente configurata e i diversi file vanno posizionati accuratamente nelle diverse cartelle.
Come primo esempio disponiamo i file che abbiamo realizzato sino a ora:
**A** file del Bean, che contiene classe CiaoBean.class;
**B** file ProvaBean.jsp che contiene la nostra pagina JSP.

È consigliabile mettere tutti i Bean in un package altrimenti il server potrebbe non essere in grado di trovare la classe e potrebbe essere necessario inserire la riga di comando che li importa direttamente:

```
<%@ page import = "nomeclasse" %>
```

Metteremo i nostri esempi nel package mieiBean che collocheremo nella sottocartella mieiBean della cartella classes di Tomcat.



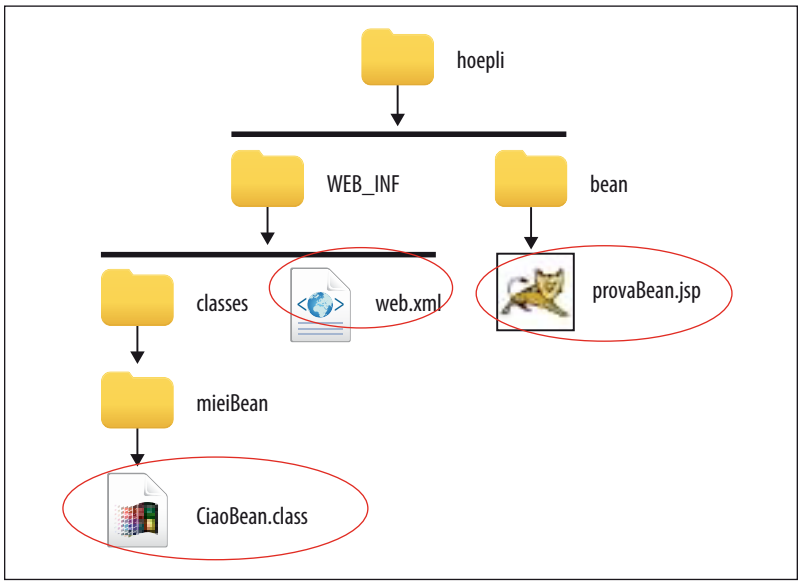

Per permettere la visibilità di un Bean si deve inoltre inserire nel file web.xml il CLASSPATH relativo all'applicazione Web che lo realizza.

```
276   <servlet>
277     <servlet-name>CiaoBean</servlet-name>
278     <servlet-class>CiaoBean</servlet-class>
279   </servlet>
280   <servlet-mapping>
281     <servlet-name>CiaoBean</servlet-name>
282     <url-pattern>/servlets/servlet/mieiBean/CiaoBean</url-pattern>
283   </servlet-mapping>
```

cartella

classe

Il file provaBean.jsp può essere posizionato in una "cartella a piacere", per esempio in Bean: di seguito riportiamo l'albero completo dove inserire/modificare i file della nostra applicazione.




## METTITI ALLA PROVA

- Realizzare e utilizzare i JavaBean

Realizza un Bean che analizzando la data corrente visualizza sullo schermo il nome della stagione e quanti giorni mancano per il veglione di S. Silvestro, che si tiene il 31 dicembre.

## Passaggio parametri al Bean

Aggiungiamo al nostro esempio una pagina HTML che richiama la pagina JSP passandogli un valore come parametro che andrà successivamente a modificare una proprietà del Bean.
Realizziamo un'applicazione dove un utente seleziona una categoria di libri che ha la seguente interfaccia:

| | Specifica del Bean | | | |
|---|---|---|---|---|
| Funzione | Visualizzi i libri disponibili per una categoria richiesta | | | |
| Nome | LibriBean | | | |
| | | | | |
| Proprietà | Tipo | Read | Write | Significato |
| categoria | String | X | X | Selettore per la categoria |
| elenco | String | X | | Elenco dei libri disponibili |

Sullo schermo, come risposta, viene visualizzato l'elenco dei testi disponibili.

Il Bean ha nome LibriBean e viene aggiunto al file web.xml:

```
286     <servlet>
287       <servlet-name>LibriBean</servlet-name>
288       <servlet-class>LibriBean</servlet-class>
289     </servlet>
290     <servlet-mapping>
291       <servlet-name>LibriBean</servlet-name>
292       <url-pattern>/servlets/servlet/mieiBean/LibriBean</url-pattern>
293     </servlet-mapping>
```

Il codice della classe non necessita di spiegazioni: il Bean viene inserito nel package mieiBean e nel costruttore inizializziamo le stringhe contenenti i titoli dei libri disponibili, non accessibili dall'esterno né in lettura e né in scrittura.

```
1  package mieiBean;
2  import java.util.*;
3  public class LibriBean implements java.io.Serializable
4  {
5    private String informatica, linguaggi, poesie;
6    private String categoria;
7
8    public LibriBean(){
9      inizializza();
10   }
11
12   private void inizializza(){
13     informatica = "Algoritmi 1 ISBN XXX-YY1 <br> Tecnologie Informatiche ISBN XXX-YY2<br>
14     linguaggi = "Il linguaggio C ISBN XXX-ZZ1 <br> Java per tutti ISBN XXX-ZZ2 <br> Prog
15     poesie = "Poesie d'amore e di disperazione ISBN XXX-ZZ1 <br> Vita nuova ISBN XXX-ZZ2 <
16   }
```

Naturalmente questo esempio è di solo scopo didattico: al posto delle stringhe sarebbe opportuno interrogare il database e "ritornare" all'utente il risultato della query: questa modifica verrà aggiunta al progetto dopo che avremo affrontato lo studio della connessione delle servlet e delle JSP con i database MySQL e Access.

Per la proprietà categoria inseriamo entrambi i metodi set e get:

```
18   // metodi getter e setter
19   public void setCategoria(String categoria){
20     this.categoria = categoria;
21   }
22   public String getCategoria(){
23     return categoria;
24   }
```

Per la proprietà elenco l'unico metodo definito è getElenco() che ci ritorna in una variabile l'elenco dei libri della categoria prescelta:

```
26   public String getElenco() {
27     String msg = "Nessuna categoria scelta";
28     if ("informatica".equals(categoria)){
29      msg = informatica;
30     }
31     else if ("linguaggi".equals(categoria)){
32      msg = linguaggi;
33     }
34     else if ("poesie".equals(categoria)){
35      msg = poesie;
36     }
37     return msg;
38   }
```

È doveroso osservare che non è presente un attributo con il nome di elenco; elenco è una proprietà che viene "costruita" in questo metodo: la pagina JSP richiama il metodo e riceve un valore, non una variabile!

La pagina HTML che richiede all'utente di selezionare una categoria di libri è la seguente:

e ha la codifica riportata di seguito:

```
JSP e Bean con query (bean/libriBean2.html)
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2  <HTML>
3  <HEAD><TITLE> JSP e Bean con query</TITLE></HEAD>
4  <BODY>
5  <FORM ACTION="libriBean1.jsp" METHOD="GET">
6    <P><STRONG><font color="#0000FF" size="4" face="Arial, Helvetica, sans-serif">JSP
7      con Bean e selezione da HTML</font></STRONG></P>
8    <P>Seleziona la categoria di libri che ti interessano:</P>
9    <P>
10     <SELECT NAME="scelta">
11       <OPTION VALUE="informatica">informatica</OPTION>
12       <OPTION VALUE="linguaggi">linguaggi</OPTION>
13       <OPTION VALUE="poesie">poesie</OPTION>
14     </SELECT>
15     <INPUT TYPE="submit" VALUE="Submit">
16     <INPUT TYPE="reset">
17   </P>
18   <P><strong>Nota</strong>: viene restituita una pagina <font color="#FF0000">HTML</font>
19     generata da <font color="#FF0000">libriBean1.jsp </font></P>
20 </FORM>
21 </BODY>
22 </HTML>
```

dove come ACTION verrà richiamata la seguente pagina libriBean1.jsp che nella riga 4 crea un oggetto della classe LibriBean del package mieiLibri assegnandogli libri come identificatore.

```
bean/libriBean1.jsp
1  <HTML>
2  <HEAD></HEAD>
3  <BODY BGCOLOR="ffffcc">
4  <jsp:useBean id="libri" class="mieiBean.LibriBean" />
5  <H1><img src="images/logoeditore.gif" width="364" height="56"></H1>
6  <h2><font color="#FF0000" face="Arial, Helvetica, sans-serif">Benvenuto nella
7    nostra libreria</font></h2>
8  <HR>
9  <H2><font color="#0000FF">Ti suggerisco i seguenti libri di:
10 <% out.println( request.getParameter("scelta") );%>
11 </font></H2>
```

Con l'istruzione della riga 10 viene letto il parametro scelta passato dalla pagina HTML che viene nuovamente letto nel tag della riga 14 e assegnato come valore alla proprietà categoria del Bean.

```
13 <!-- setto nel Bean la variabile categoria con la scelta dell'utente -->
14 <jsp:setProperty name="libri" property="categoria"
15                  value="<%= request.getParameter(\"scelta\") %>" />
```

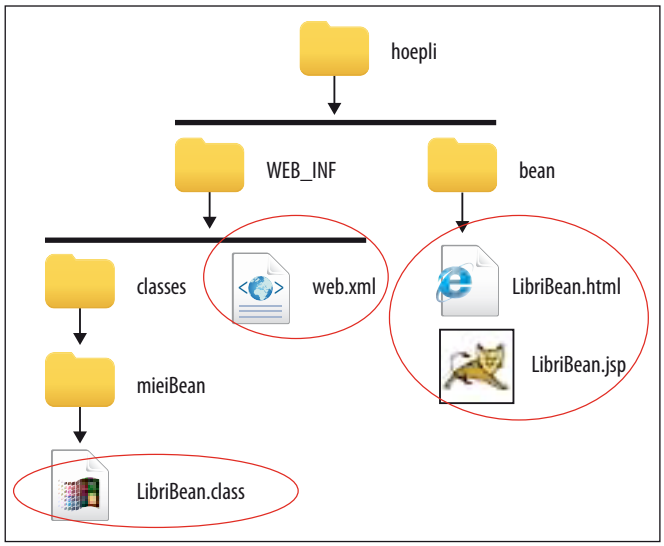
Con l'istruzione della riga 17 viene letta la proprietà elenco che direttamente visualizza sullo schermo il risultato della get:

```
16 <!-- ricevo dal Bean la stringa di risposta -->
17 <jsp:getProperty name="libri" property="elenco"/>
18 <HR></BODY>
19 </HTML>
```

I tag di formattazione dell'output devono essere messi nel Bean e non nella pagina JSP che visualizza semplicemente la stringa che riceve dall'oggetto della classe Bean.

Un'esecuzione ha il seguente output:

I singoli file vengono collocati nelle cartelle di seguito schematizzate:




## Passaggio diretto

Una possibile alternativa alla procedura appena realizzata è quella di effettuare direttamente il passaggio del parametro letto dalla pagina HTML, indicandolo con la parola chiave param, nel tag della riga 14:

```
13 <!-- setto nel Bean la variabile categoria con la scelta dell'utente -->
14 <jsp:setProperty name="libri" property="categoria" param="scelta" />
```

Questa istruzione imposta il valore della proprietà di un Bean al valore di un parametro della richiesta.

> Se nel tag <jsp:setProperty> l'attributo value non viene specificato, di default viene utilizzato il parametro della richiesta HTTP con lo stesso nome: con questa modalità è possibile elaborare molto semplicemente dati in GET o in POST.
> Sempre in <jsp:setProperty>, se l'attributo property è impostato a "*", verranno impostate tutte le proprietà del Bean con nome uguale a un parametro in ingresso.

# Conclusioni

Oltre al notevole vantaggio di riutilizzare i componenti in contesti diversi, nell'ultimo esempio si è visto come è possibile gestire semplicemente i dati provenienti da un form e passarli velocemente al Bean dove implementare anche una logica complessa per effettuare controlli sui campi inseriti dall'utente. Così facendo si mantiene l'isolamento della logica del programma (nel codice Java) dalla pagina Web stessa (pagina JSP).

## METTITI ALLA PROVA

• Passaggio parametri ai Bean

1 Modifica i file dell'esempio precedente in libriBean1.html e libriBean1.jsp introducendo il passaggio diretto del parametro dalla pagina HTML al Bean come indicato nella riga 14 della figura.

2 Successivamente aggiungi anche la possibilità di selezionare la lingua nella quale sono stati scritti i libri semplicemente passando i parametri con:

```
16 <jsp:setProperty name="libri" property="categoria" param="scelta" />
17 <jsp:setProperty name="libri" property="lingua" />
```

Una soluzione è riportata in libriBean2Sol.html e libriBean2Sol.jsp: inoltre è stata modificata anche la classe LibriBean.java.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Quale affermazione è errata per un JavaBean?**
**a** È una classe public
**b** Ha due metodi definiti di default
**c** Ha un costruttore public di default
**d** Il costruttore ha zero argomenti

**2 Che cosa non descrive la scheda di "specifica del Bean"?**
**a** La funzione svolta dal Bean
**b** Il nome della classe
**c** Il nome degli attributi
**d** Gli accessi agli attributi
**e** Il tipo del costruttore

**3 Per utilizzare un Bean in una pagina JSP si utilizza la sintassi:**
**a** <useBean id="saluto" class="mieiBean.CiaoBean.java"/>
**b** <useBean:jsp id="saluto" class="mieiBean.CiaoBean"/>
**c** <jsp:useBean id="saluto" class="mieiBean.CiaoBean"/>
**d** <jsp:useBean id="saluto" class="mieiBean.CiaoBean.java"/>

**4 Quale tra le seguenti opzioni non è prevista per l'attributo scope dell'istruzione <jsp:useBean..>=?**
**a** Page
**b** Request
**c** Response
**d** Session
**e** Application

**5 L'architettura J2EE Model 1 è costituita da:**
**a** Un livello
**b** Due livelli
**c** Tre livelli
**d** Numero differente tra JSP e servlet

**6 Se xxx è la root del mio sito, il package CiaoBean può avere il seguente <url-pattern>:**
**a** xxx/servlet/mieiBean/CiaoBean
**b** servlets/xxx/mieiBean/CiaoBean
**c** servlets/servlet/mieiBean/CiaoBean
**d** xxx/CiaoBean/mieiBean

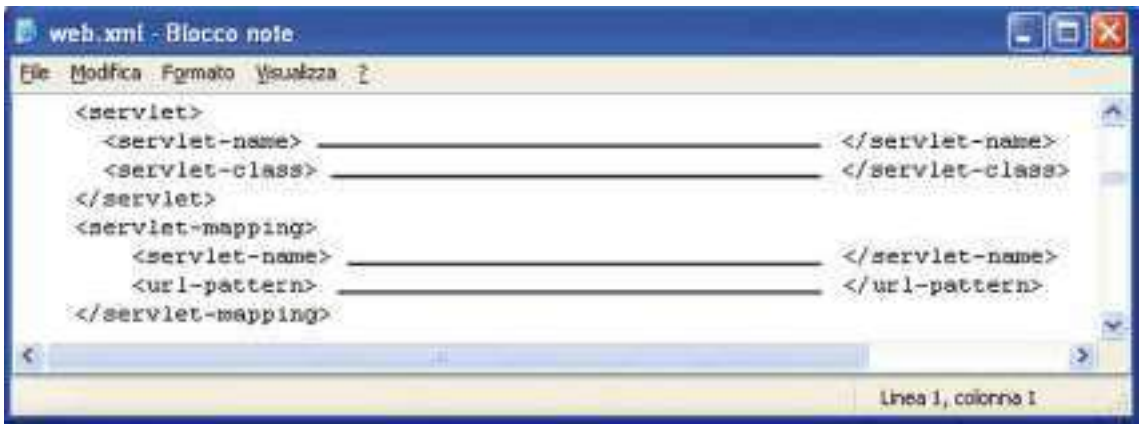
## VERO/FALSO

1 Un JavaBean è un componente nella tecnologia Java, cioè una classe. V F
2 I Java Data Bean sono particolari Bean che gestiscono la data. V F
3 I JavaBean rispettano le direttive che realizzano l'incapsulamento. V F
4 Nei JavaBean tutti gli attributi devono essere di tipo privato. V F
5 Gli attributi nella JSP saranno indicati col termine proprietà (property). V F
6 Le proprietà di tipo boolean sono: getAttributo () e setAttributo (). V F
7 Le classi Bean devono implementare la classe Serializable. V F
8 Il nome dei metodi e il nome dell'attributo che gestiscono devono essere uguali. V F
9 La chiamata a un metodo di un Bean da parte di una pagina JSP avviene con il tag <jsp: … >. V F
10 Per modificare gli attributi della classe Bean si utilizza <isp:getProperty>. V F
11 Per permettere la visibilità di un Bean deve essere modificato il file web.xml. V F
12 La pagina JSP richiama il metodo e riceve una variabile. V F

# VERIFICA... le competenze

## ESERCIZI

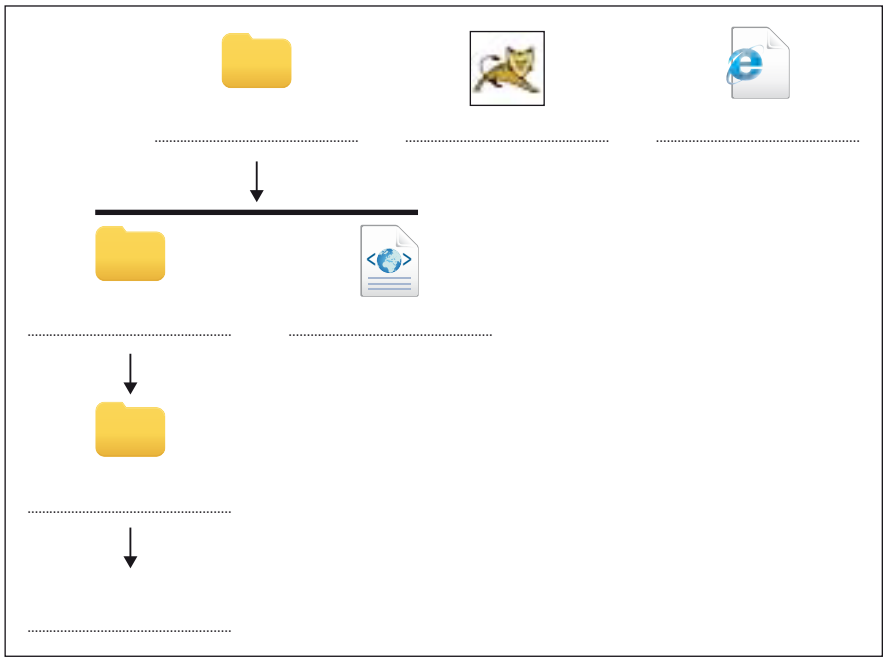
**1** Modifica il file web.xml aggiungendo un Bean di nome HelloBean.




**2** Disponi correttamente i nomi delle directory e dei file di una generica applicazione "Ciao Utente".

**Directiory**: mieiBean – Web-inf – classes

**File**: Web.xml – HelloBean.class – HelloBean.html – HelloBean.jsp



**3** Realizza un'applicazione Hello.xxx che riceva dall'utente il nome e il cognome e controlli se è presente in un elenco (precaricato in un array): in questo caso ne aggiorni la città di nascita che è presente nella pagina HTML.

# 1 ESERCIZI IN LABORATORIO

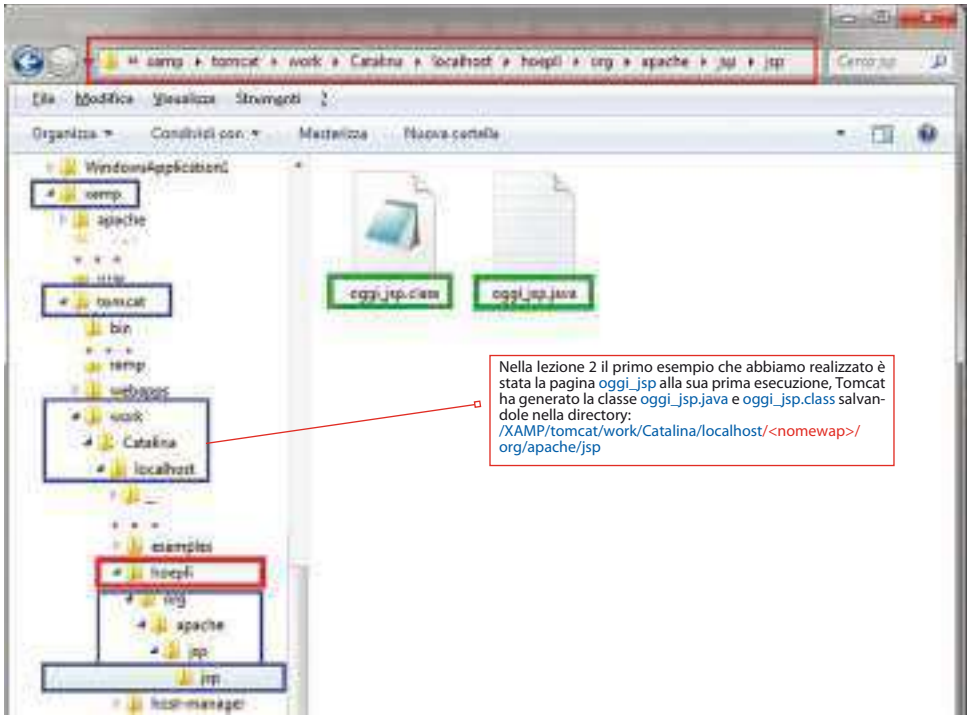
## JSP: PRIMI ESEMPI DI PROGRAMMAZIONE

### Codice sorgente Java delle pagine JSP

La prima volta che al server viene richiesta una pagina JSP questa viene "trasformata" dal JSP compiler in una servlet, cioè in un file sorgente .java: la servlet viene a sua volta compilata e produce un file eseguibile .class che viene caricato nel server, viene eseguito e tenuto pronto per eventuali altre richieste successive.

1 Il client richiede la pagina "oggi"
2 Il server legge la pagina oggi.jsp
3 Il server genera la servlet oggi_jsp.java
4 Il server compila la oggi_jsp.java ottenendo una classe eseguibile oggi_jsp.class
5 Il server esegue la servlet
6 La risposta viene inviata al client

Individuiamo nel nostro server la posizione dei due file generati automaticamente:




### Java Server Pages: inclusione di file

La direttiva include diventa indispensabile quando vogliamo creare dei metodi da utilizzare in più pagine, cioè per crearci una libreria di metodi/funzioni da utilizzare in ogni nostra pagina JSP.
Includiamo per esempio segmenti di codice HTML condivisi da tutte le pagine del nostro sito:

```
10  <body>
11  <%@ include file="header.html" %>
12  <div id="corpo">
13  <%@ include file="menu.html" %>
```

# ESERCIZI IN LABORATORIO 1

La utilizziamo per esempio per realizzare una semplice pagina come la seguente:

## Java Server Pages: iterazioni

Nella pagina JSP è possibile inserire le istruzioni di iterazione, per esempio con il *ciclo for*, molto utilizzato per la costruzione di tabelle che visualizzano i dati generati dalle funzioni Java.
Realizziamo una semplice pagina che visualizzi il fattoriale dei primi 10 numeri utilizzando una funzione ricorsiva fact() così codificata:

```
<%! public long fact (long x) {
  if (x == 0)
    return 1;
  else
    return x * fact(x-1);
}
%>
```

La visualizzazione dei risultati viene effettuata con un segmento di codice "misto" tra HTML e Java, che per ogni iterazione genera una riga a video richiamando la funzione che esegue il calcolo:

```
<HTML>
<HEAD><TITLE>Fattoriale ricorsivo</TITLE></HEAD>
<BODY>
<H2 ALIGN="CENTER">Fattoriale ricorsivo con ciclo FOR</H2>
<TABLE ALIGN="CENTER" BORDER="1">
<TR>                                        <!-- intestazione tabella -->
  <TD><I>n</I></TD>
  <TD><I>n</I>!</TD>
</TR>
<!-- istruzioni Java si alternano con tag HTML per comporre la riga a schermo -->
<% for (long x = 0L; x <= 10L; ++x) { %>        <!-- per tutte le righe -->
  <TR><TD><b> x %></TD><TD><b> fact(x) %></TD></TR>  <!-- genera un valore -->
<% } %>
</TABLE>
</BODY>
</HTML>
```

Mandandola in esecuzione otteniamo il risultato riportato a lato.

### METTITI ALLA PROVA

- Iterazione in JSP

Scrivi una JSP che generi la sequenza di Fibonacci per i primi 10 numeri visualizzandoli in una tabella simile a quella del fattoriale, ma alternando il colore di sfondo per le righe di indice pari con quelle di indice dispari.

# 1 ESERCIZI IN LABORATORIO

## Variabili locali e globali

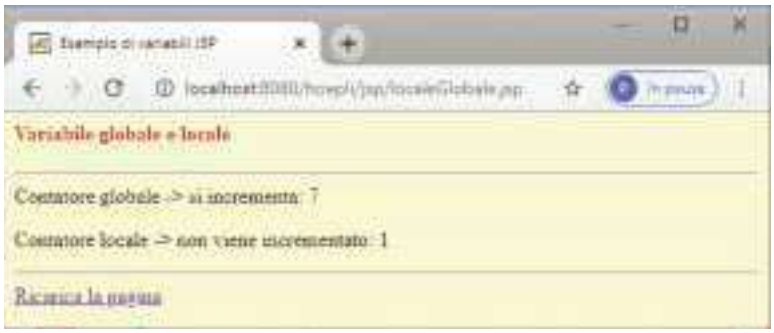
È necessario tener ben presente che:

- una variabile dichiarata nelle righe iniziali con <%! è una variabile di istanza condivisa da tutti i thread e quindi sono possibili problemi dovuti all'accesso concorrente;
- una variabile dichiarata in una scriptlet (che vedremo subito di seguito) è locale al metodo di elaborazione della richiesta e quindi ogni thread ne ha la propria copia in utilizzo esclusivo.

Scriviamo un segmento di codice dove siano presenti due variabili, una locale e una globale, e verifichiamone il funzionamento richiamando più volte la stessa pagina dal browser, cioè mandandone in esecuzione più istanze:

```
localeGlobale.jsp
1  <!--  dichiarazione globale                    -->
2  <%!
3    int contatoreGlobale = 0;
4  %>
5  <HTML>
6  <BODY BGCOLOR="FFFFCC">
7  <TITLE> Esempio di variabili JSP </TITLE>
8  <H4><FONT COLOR="0F7000">Variabile globale e locale </FONT></H4>
9  <HR>
10 Contatore globale -> si incrementa: <%= ++contatoreGlobale %>
11 <!--  variabili dichiarazione locale            -->
12 <%
13   int contatoreLocale = 0;
14 %>
15 <P>Contatore locale -> non viene incrementato: <%= ++contatoreLocale %> </P>
16 <HR>
17 <a href="localeGlobale.jsp">Ricarica la pagina</a></P>
18 </BODY>
19 </HTML>
```

Se ora mandiamo in esecuzione la pagina e la ricarichiamo più volte osservarviamo come il contatore globale viene incrementato mentre quello locale, a ogni "ricarica", ha sempre il valore iniziale, cioè 1, come mostrato nella seguente schermata:




## Cookie e sessioni

La direttiva page permette di includere l'utilizzo delle librerie di Java e quindi "apre le porte" a un insieme pressoché infinito di possibilità.
Vediamo per esempio come è possibile definire e utilizzare i cookie: scriviamo due segmenti di codice rispettivamente il primo per scrivere un cookie e il secondo per leggerlo.

```
1  <%@ page language = "java" %>
2  <%! Cookie mioCookie = new Cookie("provacookie","paolo"); %>
3  <%
4    mioCookie.setMaxAge(3600);     // permanenza del cookie in secondi
5    mioCookie.setSecure(false);    // trasmesso anche con protocolli non sicuri come HTTP
6    response.addCookie(mioCookie);
7  %>
8  Ho scritto un cookies di nome provacookie,, <a href="cookieLeggi.jsp">guardalo!</a>
```

# ESERCIZI IN LABORATORIO 1

Come prime istruzioni del codice JSP dobbiamo indicare il linguaggio utilizzato in modo da poter definire un oggetto della classe Cookie e utilizzare su di esso i metodi disponibili.

```
<%@ page language="java" %>
<%! Cookie mioCookie = null;                    // oggetto classe Cookie %>
<%! int indice = 0; %>
<%! Cookie[] cookiesUtente; %>
```

Il resto del codice non presenta difficoltà: leggiamo i cookie dall'oggetto implicito request e lo analizziamo alla ricerche del cookie dal nome desiderato (nel nostro caso provacookie).

```
<p>Sto leggendo i cookies presenti ... <br>
<% cookiesUtente = request.getCookies();
 while(indice < cookiesUtente.length){        // scorre tutti i cookie
   if((cookiesUtente[indice].getName()).equals("provacookie"))
     break;                                   // trovato il mio cookie
   else
     indice++;                                // non trovato prosegue
 }
 if (indice < cookiesUtente.length%)          // se trovato prima della fine
   mioCookie = cookiesUtente[indice] ;        // allora era quello che cercavo
 else
   mioCookie = null;                          // altrimenti non è presente
 if (mioCookie != null){
   out.println("Cookie con valore provacookie contiene: "+ mioCookie.getValue());
 }else
   out.println("Cookie di valore provacookie non settato" );
%>
<br>
 Clicca per scrivere il cookie <span class="style1">"provacookie"</span> <a href="
 cookieScrivi.jsp">scrivi un cookie </a>
```

Richiamiamo tra loro le due pagine in modo da visualizzare prima e dopo il contenuto della memoria:

**1** chiamo cookieLeggi.jsp inizialmente: nessun cookie settato;

**2** chiamo cookieScrivi.jsp per settare il mio cookie provacookie;

**3** chiamo nuovamente cookieLeggi.jsp e visualizzo il contenuto del cookie provacookie.

# 1 ESERCIZI IN LABORATORIO

## METTITI ALLA PROVA

• Variabili di sessione

**1** Scrivi tre segmenti di codice per definire e visualizzare il contenuto di una sessione.
Il primo segmento sessioniA1.jsp è semplice codice JSP che definisce tre variabili.
Il secondo segmento è sessioniA2.jsp che visualizza le tre variabili settate.

Puoi confrontare la tua soluzione con quella presente nei corrispondenti file.

**2** Scrivi un segmento di codice dove cliccando su un link viene simulato il lancio di una moneta e viene visualizzato quante volte è uscita ogni faccia: quando viene raggiunto il valore 5 lo script termina annunciando come vincitore la faccia che ha raggiunto tale valore.

# ESERCIZI IN LABORATORIO 2

## JSP CON PARAMETRI DA HTML

### Gestione dell'input

In questa esercitazione vedremo come passare i dati da una pagina HTML a una JSP: scriviamo una semplice pagina ciaoUtente.html che legge e invia il parametro nome alla pagina ciaoMondo.jsp.

Il form richiama la pagina JSP mediante il metodo GET: nella pagina JSP leggiamo il parametro in modo identico a quello visto per le servlet.

```
ciaoMondo.jsp
1  <HTML><BODY>
2  <% String visitor = request.getParameter("nome");
3  if (visitor == null) visitor = "Mondo"; %>
4    Ciao, <%= visitor %>!
5  </BODY></HTML>
```

L'oggetto implicito request della classe HttpServletRequest viene creato automaticamente e su di esso possiamo utilizzare i metodi già visti per le servlet: il risultato è il seguente, dove abbiamo evidenziato la stringa di query.

### METTITI ALLA PROVA

• Realizzare una pagina JSP

1 Scrivi una versione della classica pagina "Ciao Mondo" in modo che se riceve un parametro "nome" e "lingua" visualizza un saluto personalizzato:
http://localhost:8080/hoepli/jsp/ciaoMondoNome.jsp produce Hello World
http://localhost:8080/hoepli/jsp/ciaoMondoNome.jsp?nome=Ciro produce Hello Ciro
http://localhost:8080/hoepli/jsp/ciaoMondoNome.jsp?nome=Ciro&lingua=it produce Ciao Ciro

2 Analizza il codice generato da Tomcat: quali sono le principali osservazioni che puoi fare?

# 2 ESERCIZI IN LABORATORIO

## PER SAPERNE DI PIÙ

**METODI HTTPSERVLETREQUEST**

La tabella che segue riporta alcuni metodi disponibili per la classe HttpServletRequest.

| | |
|---|---|
| public String getParameter(String name) | Restituisce il valore di un parametro di richiesta di nome *name* come *String*, oppure il valore *null* se il parametro non esiste. |
| public String[] getParameterValues(String name) | Restituisce un array di oggetti String che contengono tutti i valori posseduti da un determinato parametro della richiesta, (cfr. checkbox) oppure il valore *null* se il parametro non esiste. |
| public String getMethod() | Restituisce il nome del metodo Http usato per effettuare la richiesta (GET, POST). |
| public String getProtocol() | Restituisce il nome e la versione del protocollo utilizzato per effettuare la richiesta. |
| public String getRequestURI() | Restituisce l'URL della richiesta corrente, esclusa la stringa di query. |
| public String getQueryString() | Restituisce la stringa di query contenuta nella richiesta |

Nel secondo esempio introduciamo una list box contenente il nome di alcune città:

Il form richiama la pagina JSP mediante il metodo GET e visualizza questa schermata:

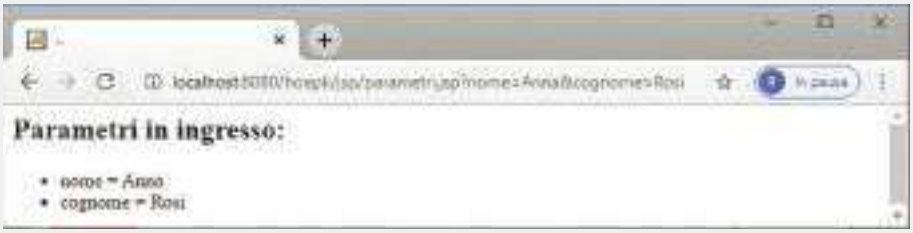
Oltre al parametro selezionato visualizziamo a titolo didattico anche il risultato di alcuni metodi dell'oggetto request della classe HttpServletRequest.

Il codice lo puoi trovare nel file parametro1.jsp.

## METTITI ALLA PROVA

• JSP con parametri

1 Scrivi una semplice pagina HTML dove vengono inseriti nome e cognome di un utente che vengono successivamente visualizzati in una pagina simile alla seguente:



# ESERCIZI IN LABORATORIO 2

**2** Modifica lo script che esegue il calcolo del fattoriale passandogli come parametro il numero per il quale deve effettuare il calcolo:

**3** Scrivi una pagina HTML che permetta di selezionare da una list box una categoria e un bottone che una volta cliccato reindirizzi il nome alla pagina JSP (sezioni.jsp) che si occuperà di scrivere Sezione XXX, dove XXX rappresenta la categoria presa in input. Le categorie possono essere per esempio finanza, politica, sport.

## Elaborazione del codice

Naturalmente è possibile elaborare la variabile ricevuta come parametro per inserirla all'interno di codice scriplet: scriviamo una applicazione con la quale un utente seleziona da una list box un colore che successivamente viene utilizzato come sfondo della pagina di risposta.
La pagina JSP analizza il parametro "scelta" passato col metodo GET dal form iniziale: se è presente un parametro viene utilizzato come sfondo altrimenti viene definito il bianco come colore di default.

```
parametro1.jsp
1  <HTML>
2  <%
3   String bgColor = request.getParameter("scelta");
4   boolean coloreScelto;
5   if(bgColor != null){              // analisi del parametro
6     coloreScelto = true;
7   }else{
8     coloreScelto = false;
9     bgColor = "WHITE";
10  }
11 %>
12 <BODY BGCOLOR="<%= bgColor %>">   <!-- setto colore di sfondo-->
13 <H2>Colore come parametro</H2><B>
14 <%
15  if(coloreScelto){                // visualizza la scelta fatta
16    out.println("Hai scelto un bel colore: " + bgColor + " ");
17  }else{
18    out.println("Colore sfondo non scelto, WHITE di default.");
19  }
20 %>
21 </B></BODY>
22 </HTML>
```

Nella parte finale visualizza sullo schermo una scritta "costruita" in base alla scelta del colore.

# 2 ESERCIZI IN LABORATORIO

Una possibile esecuzione è la seguente:

Confermando il giallo come colore preferito viene prodotta la seguente pagina JSP:

## METTITI ALLA PROVA

→ • JSP con parametri • Interazione HTML e JSP

**1** Modifica la pagina HTML precedente trasformandola in una pagina di login: in caso di mancanza di un dato indica all'utente l'errore e richiedine l'inserimento richiamando la pagina di input ed evidenziando in essa il dato mancante, per esempio colorando lo sfondo del campo in giallo.
Dopo che l'utente ha effettuato il login, presentagli una pagina dove sono presenti tre oggetti:
- una list box, contenente per esempio le materie d'esame di maturità;
- un gruppo di radio button, per esempio per selezionare se scritta od orale;
- un gruppo di check box, per esempio per indicare i giorni della settimana nei quali la materia selezionata verrà studiata.

Dopo che l'utente ha compiuto la scelta ed effettuato la conferma, realizza una pagina JSP che ne sintetizzi a video i dati prescelti per tutte e sei le materie d'esame.

**2** Scrivi una pagina HTML che richieda all'utente il nome e una password e, sfruttando le variabili locali, globali e le sessioni, memorizzi i primi due utenti che si collegano e successivamente conti quante pagine sono visualizzate per ciascuno di essi.
Nel caso di accesso di un ulteriore utente, l'applicazione segnala l'impossibilità del servizio in quanto, per poter accedere, il terzo utente deve attendere che uno dei due precedenti utilizzatori abbia visualizzato almeno 5 pagine e, solo in tal caso, il nuovo utente sostituisce quello che ha già collezionato il maggior numero di accessi.

# ESERCIZI IN LABORATORIO 2

## Esercizi proposti

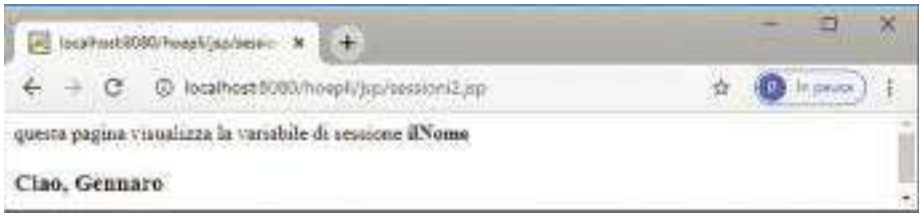
**Verifica le tue competenze eseguendo gli esercizi proposti**

**1** Scrivi tre segmenti di codice per definire e visualizzare il contenuto di una sessione.

1. Il primo segmento è un semplice codice HTML che legge un dato e lo invia come parametro.

2. Il secondo segmento è sessioni1.jsp che legge il parametro e setta l'attributo di sessione.

3. Il terzo segmento sessioni2.jsp visualizza il contenuto della variabile di sessione.




Puoi confrontare la tua soluzione con quella presente nei corrispondenti file.

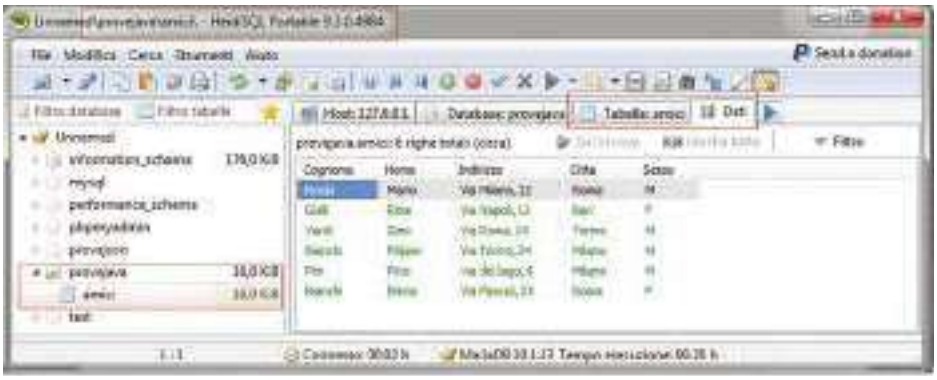
**2** Modifica i tre segmenti di codice scritti in precedenza in modo che l'utente inserisca, oltre al nome, anche l'indirizzo di e-mail e la password e, sfruttando le variabili locali, globali e le sessioni, se si ricollega in un secondo tempo non gli richieda nuovamente i dati, ma visualizzi una nuova pagina di benvenuto personalizzata, predisponendo il suo indirizzo e-mail in un form che presenta i dati per l'accesso a un'area riservata del sito che sta visitando.

# 3 ESERCIZI IN LABORATORIO

## JSP E DATABASE MySQL

### Predisposizione dell'archivio MySQL

L'archivio che utilizziamo in questa esercitazione è lo stesso che abbiamo già descritto negli Esercizi in Laboratorio dell'Unità di Apprendimento 3. Come "novità" in questa esercitazione utilizziamo HeidiSQL come "interfaccia" del database, che è prodotto open source scaricabile da http://www.heidisql.com/ (HeidiSQL_9.3_Portable).
Automaticamente Heidi stabilisce la connessione con MySQL visualizzando una schermata simile alla seguente, dove abbiamo individuato il nostro archivio e cliccato sulla tabella amici.



### Connessione con JSP

Per poter connettere una JSP a un database, come già visto per le servlet nella esercitazione 6 carichiamo il driver specifico per il database che si desidera utilizzare, nel nostro caso MySQL: è lo stesso driver JDBC Connector/J già utilizzato per le servlet!

Il primo passo è quello di includere la libreria Java con i metodi necessari alle operazioni SQL:

```
1  <HTML>
2  <BODY><TITLE>JSP e database MySQL</TITLE></BODY>
3  <H2>JSP e database MySQL</H2>
4  <%@ page import="java.sql.*" %>
```

Definiamo ora le stringhe con il nome del driver, l'indirizzo del database e la query di selezione:

```
7   // carico il driver per la connessione al DB MySQL
8   String DRIVER = "com.mysql.jdbc.Driver";
9   // riferimento al database: connessione Mysql
10  String URL_mioDB = "jdbc:mysql://localhost:3306/provajava";     //3306 di default
11  // definizione della query
12  String query = "SELECT Cognome, Nome, Indirizzo, Citta FROM Amici";
```

La connessione al DRIVER la realizziamo in un costrutto try.. catch:

```
14  try{
15    Class.forName(DRIVER);
16  }
17  catch (ClassNotFoundException e) {
18    System.err.println("Driver non trovato" + e);
19  }
```

# ESERCIZI IN LABORATORIO 3

Segue l'istruzione per la connessione al database con nome utente root e password assente.

```
21    Connection connessione = null;
22    try{                          // apro la connessione verso il database
23      connessione = DriverManager.getConnection(URL_mioDB,"root","");
24    }
25    catch (Exception e){
26      System.err.println("Errore nella connessione col database: " + e);
27    }
```

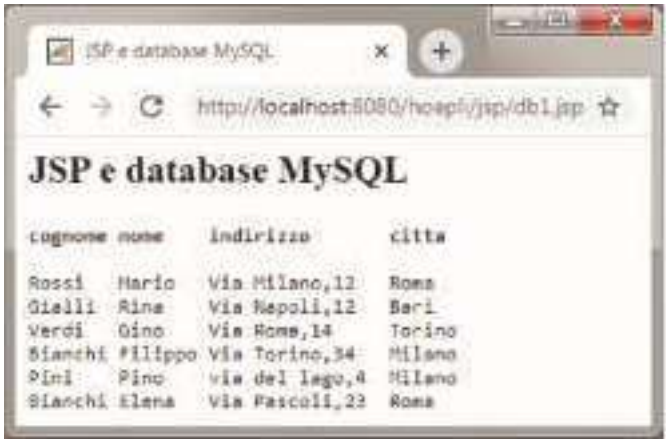
Analogamente a quanto già visto per le servlet, l'esecuzione della query ritorna un oggetto di tipo ResultSet che viene analizzato all'interno di un ciclo per visualizzare sullo schermo i dati che contiene in un segmento HTML preformattato:

```
29    try{
30      // ottengo lo Statement per interagire con il database
31      Statement statement = connessione.createStatement();
32      // interrogo il DBMS mediante una query SQL
33      ResultSet resultSet = statement.executeQuery(query);
34      // Scorro e mostro i risultati
35      out.println("<PRE>");
36      out.println("<B>cognome"+"&#9;"+"nome"+"&#9;"+"indirizzo"+"&#9;"+"citta</B><BR>");
37      while (resultSet.next()) {
38        String cognome = resultSet.getString(1);
39        String nome = resultSet.getString(2);
40        String indirizzo = resultSet.getString(3);
41        String citta = resultSet.getString(4);
42        out.println( cognome+"&#9;"+nome+"&#9;"+indirizzo+"&#9;"+citta);
43      }
44      out.println("</PRE>");
45    }
```

Termina la pagina JSP il ramo catch e la chiusura della connessione:

```
49    try{                          // chiusura connessione
50      connessione.close();
51    }
52    catch (Exception e) {
53      System.err.println(e);
54    }
55    out.close();                  // chiusura oggetto di output
56  %>
```

Un'esecuzione produce il seguente risultato:




Il codice completo è disponibile nel file db1.jsp.

## METTITI ALLA PROVA

• Connessione a MySQL

1 Modifica la JSP e adatta le pagine HTML già realizzate per le esercitazioni sulla connessione al database fatte con le servlet per realizzare le medesime applicazioni, che si riassumono in:
- una pagina con form che permette l'inserimento dei dati nell'archivio;
- una pagina che permette di effettuare la ricerca per città o per sesso.

2 Aggiungi inoltre una pagina che permette di cancellare un dato presente e una pagina che ne permette la rettifica in caso di errori di inserimento.

# 4 ESERCIZI IN LABORATORIO

## JSP E MDB

### Connessione tra JSP e MDB

Il codice che effettua la connessione tra un pagina JSP e il database MBD è simile a quello già descritto per le servlet nell'Esercizio in Laboratorio 8 dell'Unità precedente. Utilizziamo anche in JSP le librerie UCanAccess.

Per prima cosa importiamo la libreria Java:

```
mb2.jsp
1  <HTML>
2  <BODY><TITLE>JSP, UCanAccess e database MDB</TITLE></BODY>
3  <H1>JDBC e database MDB</H1>
4  <%@ page import="java.sql.*" %>
```

Definiamo le stringhe con il nome del driver, l'indirizzo del database e la query di selezione:

```
7   // stringhe di inizializzazione del driver
8   String DRIVER = "net.ucanaccess.jdbc.UcanaccessDriver";
9   // riferimento al database: connessione diretta
10  Steing protocollo = "jdbc:ucanaccess://";            // connessione libreria
11  String mdbpath = "tomcat/webapps/hoepli/esempio08/"; // percorso relativo
12  String mdbName = "proveJava.mdb;memory=false";       // nome database
13  // riferimento al database
14  String URL_mioDB = protocollo + mdbpath + mdbName;
15  // definizione della query
16  String query = "SELECT Cognome, Nome, Indirizzo, Citta FROM Amici";
```

Carichiamo quindi il DRIVER con la seguente istruzione:

```
18  try{
19    Class.forName(DRIVER);
20  }
21  catch (ClassNotFoundException e) {
22    System.err.println("Driver non trovato" + e);
23  }
```

ed effettuiamo la connessione al database:

```
24  // apro la connessione verso il database
25  Connection connessione = null;
26  try{
27    connessione = DriverManager.getConnection(URL_mioDB);
28  }
29  catch (Exception e){
30    System.err.println(e);
31  }
```

Procediamo con l'interrogazione al database definendo dapprima l'oggetto ResultSet:

```
32  // interrogazione del database
33  try{
34    // ottengo lo Statement per interagire con il database
35    Statement statement = connessione.createStatement();
36    // interrogo il DBMS mediante una query SQL
37    ResultSet resultSet = statement.executeQuery(query);
```

# ESERCIZI IN LABORATORIO 4

e successivamente con la visualizzazione dei dati in codice HTML preformattato:

```
38      // scorro il resultSet e mostro i risultati
39      out.println("<PRE>");
40      out.println("<B>cognome"+"&#9;"+"nome"+"&#9;"+"indirizzo"+"&#9;"+"citta</B>");
41      while (resultSet.next()) {
42        String cognome = resultSet.getString(1);
43        String nome = resultSet.getString(2);
44        String indirizzo = resultSet.getString(3);
45        String citta = resultSet.getString(4);
46        out.println(cognome+"&#9;"+nome+"&#9;"+indirizzo+"&#9;"+citta);
47      }
48      out.println("</PRE");
```

Lo script prosegue con il ramo catch per visualizzare eventuali errori nell'interrogazione:

```
50    catch (SQLException e){    // visualizzazione dell'errore
51      out.println("Errore nella esecuzione della query:<br>"+query);
52      out.println(e);
53    }
```

Il programma termina con la chiusura della connessione:

```
54    try{                       // chiusura connessione
55      connessione.close();
56    }
57    catch (SQLException e){    // gestione dell'errore
58      out.println(e);
59    }
60    out.close();
61  %>
```

## METTITI ALLA PROVA

→ • Connessione tra JSP e MDB

1 Modifica la JSP e adatta le pagine HTML già realizzate per le esercitazioni sulla connessione al database MySQL per realizzare le medesime applicazioni, che si riassumono in:
   - una pagina con form che permetta l'inserimento dei dati nell'archivio;
   - una pagina che permetta di effettuare la ricerca per città o per sesso.

2 Aggiungi inoltre una pagina che permetta di cancellare un dato presente e una pagina che ne permetta la rettifica in caso di errori di inserimento.

# 5 ESERCIZI IN LABORATORIO

## JSP E DATABASE DERBY

### Connessione tra JSP e Derby

Il codice che effettua la connessione tra un pagina JSP e il database Derby è simile a quello già descritto per le servlet nell'Esercizio in Laboratorio 9 dell'Unità precedente.

Per prima cosa importiamo la libreria Java:

```
<HTML>
<BODY><TITLE>JSP e database Derby</TITLE></BODY>
<H2>JSP e database Derby </H2>
<%@ page import="java.sql.*" %>
<TABLE BORDER="1">
```

Definiamo le stringhe con il nome del driver, l'indirizzo del database e la query di selezione:

```
// carico il driver per la connessione ad DB MySQL
String DRIVER = "org.apache.derby.jdbc.EmbeddedDriver";
String protocollo = "jdbc:derby:";                    // connessione alla libreria
String dbpath     = "tomcat/webapps/hompll/esempio08/";   // percorso assoluto
String dbName     = "esempiTPSIT3;create=false";      // nome database
String user       = "";                               // nome utente
String psw        = "";                               // password
String URL_mioDB = protocollo + dbpath + dbName;      // riferimento completo al database
// definizione della query
String  query = "SELECT * FROM alunni";
```

Carichiamo quindi il DRIVER con la seguente istruzione:

```
try{
  Class.forName(DRIVER);
}
catch (ClassNotFoundException e) {
  System.err.println("Driver non trovato" + e);
}
```

ed effettuiamo la connessione al database:

```
Connection connessione = null;
try{                                          // apro la connessione verso il database
  connessione = DriverManager.getConnection(URL_mioDB, user, psw);
}
catch (Exception e){
  System.err.println("Errore nella connessione col database: " + e);
}
```

Procediamo con l'interrogazione al database definendo dapprima l'oggetto ResultSet:

```
try{                                          // esecuzione della query
  Statement statement= connessione.createStatement();
  ResultSet rs = statement.executeQuery(query);
```

e successivamente con la visualizzazione dei dati in codice HTML preformattato:

```
    out.println("<pre>");
    out.println("<b>Mat."+"&#9;"+"cognome"+"&#9;"+"nome"+"&#9;"+"classe"+"&#9;"+"sesso</b>");
    while (rs.next()) {                       // scorro il resultset e mostro i risultati
      int mat = rs.getInt(1);
      String cog = rs.getString("cognome");
      String nom = rs.getString(3);
      int classe = rs.getInt("classe");
      String sez = rs.getString("sezione");
      String sex = rs.getString("sesso");
      out.println(mat+"&#9;"+cog+"&#9;"+nom+"&#9;"+classe+sez+"&#9;"+sex);
    }
    out.println("</pre>");
```

# ESERCIZI IN LABORATORIO 5

Lo script prosegue con il ramo catch per visualizzare eventuali errori nell'interrogazione:

```
48    }
49    catch (Exception e){
50      System.err.println(e);
51    }
```

Il programma termina con la chiusura della connessione:

```
52    try{                                        // chiusura connessione
53      connessione.close();
54    }
55    catch (Exception e) {
56      System.err.println(e);
57    }
58    out.close();                    // chiusura oggetto di output
59  %>
```

Lo script termina con la chiusura dei tag HTML

```
60  </TABLE>
61  </BODY>
62  </HTML>
```

## METTITI ALLA PROVA

• Connessione tra JSP e Derby

1 Modifica la JSP e adatta le pagine HTML già realizzate per le esercitazioni sulla connessione al database Derby per realizzare le medesime applicazioni, che si riassumono in:
   - una pagina con form che permetta l'inserimento dei dati nell'archivio;
   - una pagina che permetta di effettuare la ricerca per città o per sesso.

2 Aggiungi inoltre una pagina che permetta di cancellare un dato presente e una pagina che ne permetta la rettifica in caso di errori di inserimento.

# 6 ESERCIZI IN LABORATORIO

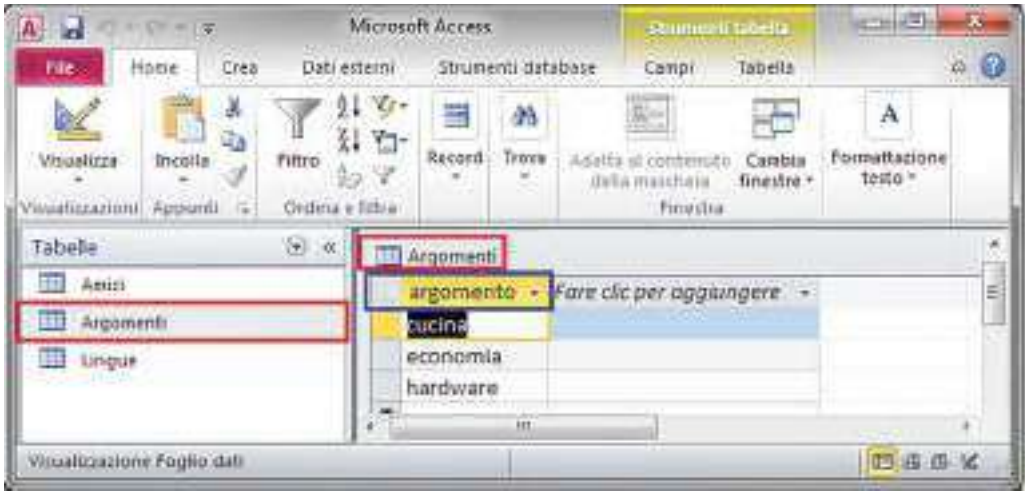
## JSP BEAN E DATABASE ACCESS, MySQL E DERBY

### Connessione tra JavaBean e Access

Vediamo un primo esercizio che ci mostra come realizzare l'interazione tra JavaBean e database Access: utilizziamo le librerie UCanAccess per realizzare la connessione al database Access.

A tal fine è necessario:

**A** modificare il database aggiungendo la tabella Argomenti con il campo argomento:




**B** Creare una classe Java ScelteDBAccess che:

**1** si connette al database proveJava.mdb;

```
// dati per la connessione al database
private final String protocollo = "jdbc:ucanaccess://";              // connessione libreria
private final String mdbpath    = "tomcat/webapps/hoepli/esempioDB/"; // percorso relativo
private final String mdbName    = "proveJava.mdb;memory=false";       // nome database
// stringa completa di indirizzo per il database access
String URL_micDB = protocollo + mdbpath + mdbName;
```

**2** estrae dalla tabella Argomenti l'elenco dei dati che contiene:

```
// definizione della query
String query = "SELECT argomento FROM argomenti";
```

**3** predispone il combo box da proporre all'utente affinché faccia la sua scelta:

```
// scorro e creo la stringa HTML che genera il combobox
elenco="<select name=\"argomentoScelto\">";
while (resultSet.next()){
  String argomento = resultSet.getString(1);
  opzione = "<option value = \"" + argomento + "\">"+argomento+"</option>";
  elenco = elenco + opzione;
}
elenco = elenco +"</select>";
```

# ESERCIZI IN LABORATORIO 6

**C** registrare la classe ScelteDBAccess nel file web.xml;

```
<servlet>
  <servlet-name>ScelteDBAccess</servlet-name>
  <servlet-class>ScelteDBAccess</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ScelteDBAccess</servlet-name>
  <url-pattern>/servlets/servlet/miei8ean/ScelteDBAccess</url-pattern>
</servlet-mapping>
```

**D** creare la pagina JSP (per esempio scegliDBBeanAccess.jsp) che carica l'elenco degli argomenti che gli viene passato dal Bean, che interroga il database Access, estrae i dati, li organizza in un combo box per "ritornargli" la stringa HTML che può essere visualizzata nel browser:

```
<HTML>
<BODY BGCOLOR="ffffcc"><h3>Connessione a database Access </h3>
<h3><font colur="#FF0000" face="Arial, Helvetica, sans-serif">Scegli l'argomento di tuo interesse </font></h3>
<FORM ACTION="scegliDBBean.jsp" METHOD="GET">
<jsp:useBean id="elenco" class="miei8ean.ScelteDBAccess" />
<H4><font color="#0000FF">
  Scegli tra questi argomenti
  <!-- aggiungo del Bean un combobox con risultato del metodo getArgomenti della classe ScelteDBAccess -->
  <jsp:getProperty name="elenco" property="argomenti"/>
  <INPUT TYPE="submit" VALUE="Invia argomento">
</font></H4>
<p><strong>Nota</strong>: viene restituita una pagina <font color="#FF0000">scegliDBBean.jsp </font>con la
scelta effettuata<font color="#FF0000"> </font></p>
</FORM></BODY>
</HTML>
```

La sua esecuzione è la seguente, dove possiamo vedere i dati presenti nel combo box:

## METTITI ALLA PROVA

• Connessione a Access

1 Dopo aver aggiunto le tabelle comuni, province e regioni scrivi un'applicazione che visualizzi l'elenco delle regioni in modo che l'utente ne possa selezionare una: dalla scelte dell'utente si deve presentare un nuovo combo box contenente l'elenco delle province presenti nella regione indicata.

2 Alla successiva scelta dell'utente si carichino in un ulteriore combo box i comuni presenti nella provincia desiderata in modo che anche tra questi l'utente possa fare la sua scelta.

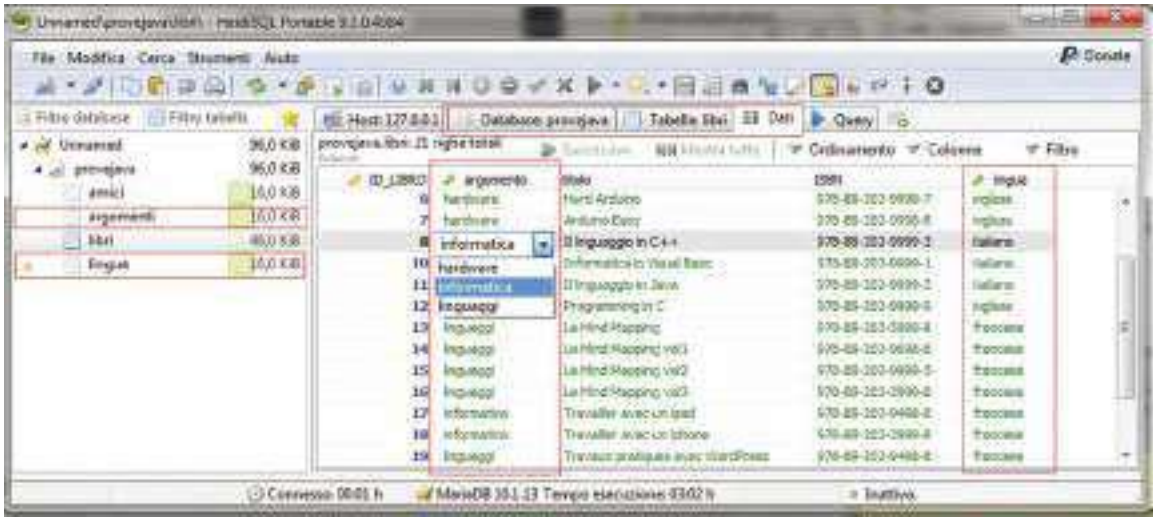
3 Si visualizzino infine i dati del comune prescelto: in particolare il codice comune, la sua longitudine e latitudine.

# 6 ESERCIZI IN LABORATORIO

## Connessione tra JavaBean e MySQL

Come esercizio riepilogativo sui JavaBean realizziamo un'applicazione che seleziona da un database i libri presenti in archivio in base alle scelte operate da un utente sull'argomento e la lingua desiderata.

Come prima operazione aggiungiamo queste due tabelle al database provejava:



**A** Creare una classe Java LibriBeanDB che:

1. si connette alla tabella libri del database provejava;
2. estrae l'elenco lingue e setta un attributo lingua dopo che viene scelto dall'utente;
3. estrae elenco argomenti e setta un attributo argomento dopo che viene scelto dall'utente;
4. estrae l'elenco libri dalla tabella libri utilizzando una query con filtro;

**B** registrare la classe LibriBeanDB nel file web.xml;

**C** creare due pagine JSP:

1. la prima, libriDBBean.jsp, che carica l'elenco delle lingue e degli argomenti in due combo box;
2. la seconda, caricaLibri.jsp, che riceve i parametri scelti dal combo box e interroga il database visualizzando l'elenco dei libri che soddisfano le condizioni impostate dall'utente.

Dopo aver definito le stringhe con il nome del driver e l'indirizzo del database impostiamo la lingua di default e carichiamo il DRIVER nel metodo inizializza() richiamato dal costruttore della classe:

```
24  private void inizializza(){
25    lingua ="italiano";
26    // carico il driver
27    try{
28      Class.forName(DRIVER);
29    }
30    catch (ClassNotFoundException e) {
31      System.err.println("Driver non trovato" + e);
32    }
33  }
```

Il metodo getElenco() restituisce l'elenco dei libri ottenuto dall'interrogazione SQL costruita prendendo come parametri i valori dei due attributi lingua e argomento:

```
30  public String getElenco(){
31    String elenco = "";
32    String opzione = "";
33    // definizione della query
34    String query = "SELECT * FROM libri WHERE lingua = '" + lingua + "'";
35    if (argomento != null)
36        query = query + "AND argomento = '" + argomento + "'";
```

# ESERCIZI IN LABORATORIO 6

I valori di questi due attributi vengono scelti nella prima pagina JSP della nostra applicazione, libriDBBean.jsp, tramite due combo box realizzati con le seguenti istruzioni:

```
<FORM ACTION="caricaLibri.jsp" METHOD="GET">
<jsp:useBean id="libri" class="mielBean.LibriBeanDB" />
<HR>
<H4><font color="#000000">
  Scegli tra questi argomenti:
  <!-- dispongo nel Bean un list box come risultato del metodo getArgomenti della classe LibriDBBean -->
  <jsp:getProperty name="libri" property="argomenti"/>
  <P>
  Scegli tra queste lingue:
  <!-- dispongo nel Bean un list box come risultato del metodo getLingue della classe LibriDBBean -->
  <jsp:getProperty name="libri" property="lingua"/>
</font></H4>
</p>
<HR>
<font color="#FF0000" face="Arial, Helvetica, sans-serif">ricerca nel database</font>
<INPUT TYPE="submit" VALUE="Submit">
<p><strong>Nota</strong>: viene restituita una pagina <font color="#FF0000">HTML</font>
   generata da <font color="#FF0000">caricaLibri.jsp</font></p>
```

Sappiamo che il contenuto del combo box viene "ritornato" dai due metodi getArgomenti() e getLibri(), come già descritto nel paragrafo precedente per il database Access.

Quando l'utente sceglie e conferma le sue preferenze nella seguente videata,

viene richiamata la pagina caricaLibri.jsp che come prima cosa andrà a settare i due attributi richiamando i metodi:

```
public void setArgomento(String argomento){
  this.argomento = argomento;
}
```

```
public void setLingua(String lingua){
  this.lingua = lingua;
}
```

tramite le seguenti istruzioni che leggono i parametri passati dalla pagina precedente:

```
<!-- setto nel Bean la variabile argomento con la scelta dell'utente -->
<jsp:setProperty name="libri" property="argomento"
                              value= "<%= request.getParameter(\"argomentoScelto\") %>" />
<!-- setto nel Bean la variabile lingua con la scelta dell'utente -->
<jsp:setProperty name="libri" property="lingua"
                              value= "<%= request.getParameter(\"linguaScelta\") %>" />
```

# 6 ESERCIZI IN LABORATORIO

Il metodo getElenco(), dopo aver eseguito la query SQL e ottenuto il ResultSet, predispone l'elenco con tutti i libri individuati che passa come parametro di ritorno:

```
// apro la connessione verso il database
connessione = DriverManager.getConnection(URL_mio30,userName,password);
// ottengo lo Statement per interagire con il database
Statement statement = connessione.createStatement();
// interrogo il DBMS mediante una query SQL
ResultSet resultSet = statement.executeQuery(query);
// accedo e predispongo l'elenco con i risultati
elenco="<PRE><B>TITOLO&#9;&#9;&#9;ISBN</B><BR>";
while (resultSet.next()){
  String id_libro = resultSet.getString(1);
  String argomento = resultSet.getString(2);
  String titolo = resultSet.getString(3);
  String ISBN = resultSet.getString(4);
  String lingua = resultSet.getString(5);
  opzione =  titolo +"&#9;"+ISBN+"&#9;<BR>";
  elenco = elenco + opzione;
}
elenco = elenco +"</PRE><BR>";
```

che viene richiamato nella pagina JSP dalla seguente istruzione:

```
27    <!-- ricavo dal Bean la stringa di risposta con l'elenco dei libri -->
28    <jsp:getProperty name="libri" property="elenco"/>
29  <HR>
```

che contemporaneamente visualizza l'attributo elenco sullo schermo:

Il codice completo è disponibile nei file libriDBBean.jsp e caricaLibri.jsp.

## METTITI ALLA PROVA

• Connessione a MySQL

Modifica l'archivio aggiungendo la tabella Autori e collegala alla tabella Libri; quindi aggiungi la possibilità di selezionare i libri anche per autore, in modo da avere le seguenti alternative:

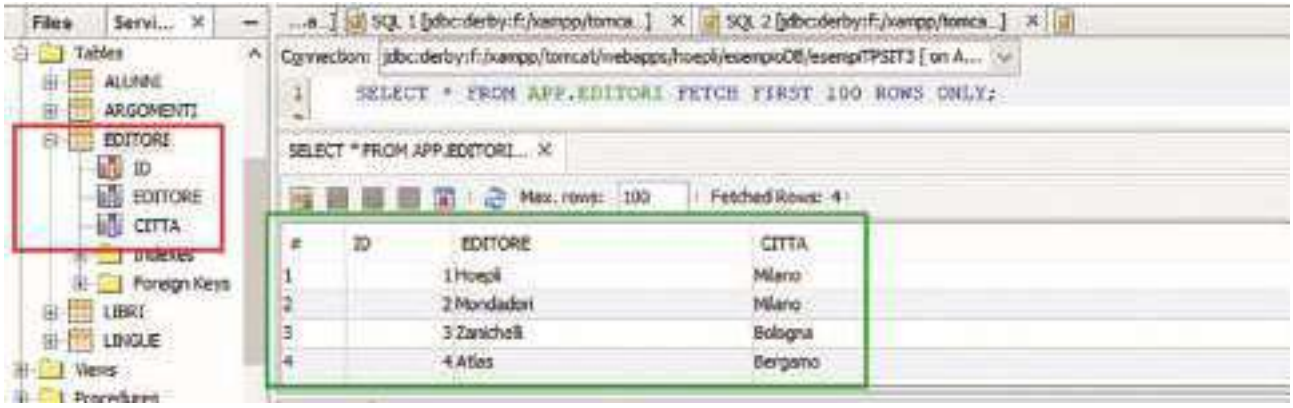
- libri di un autore di qualunque argomento;
- libri di un autore di un argomento particolare;
- libri di un autore in una lingua prescelta;
- libri di tutti gli autori in una lingua e/o argomento prescelto.

# ESERCIZI IN LABORATORIO 6

## Connessione tra JavaBean e Derby

Come ultimo esercizio sui JavaBean realizziamo un'applicazione che seleziona da un database i libri presenti in archivio in base alle scelte operate da un utente sull'argomento, la lingua desiderata e l'editore.

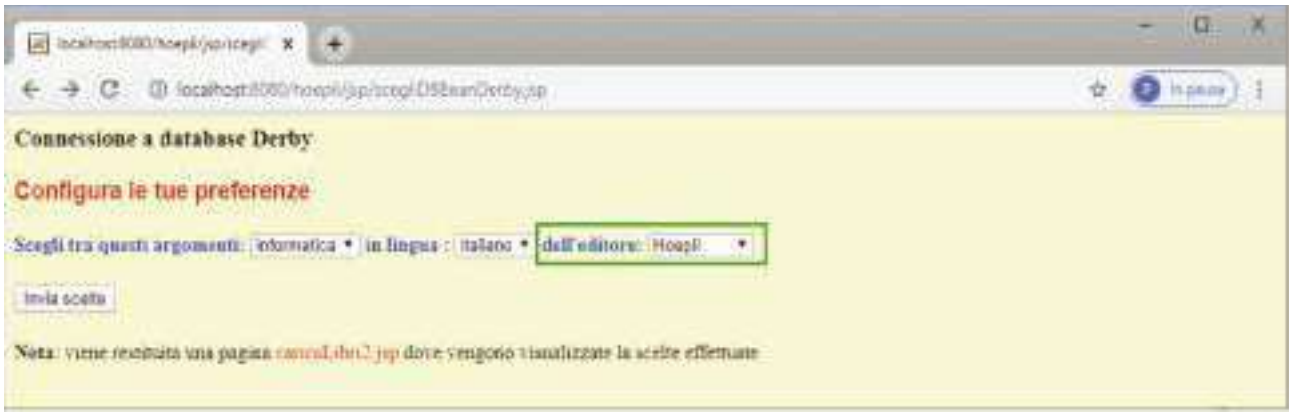
Come prima operazione aggiungiamo queste tre tabelle al database Derby esempiTPSIT3 utilizzando NetBeans, come nella figura seguente, oppure DBeaver.




Scriviamo uno script simile a libriDBBean.jsp, dove collochiamo tre combo box precaricati con le opzioni lette nelle tre tabelle argomenti, lingue ed editori tramite il Bean creato dalla classe ScelteDBDerby.class.

```
5   <FORM ACTION="caricaLibri2.jsp" METHOD="GET">
6   <jsp:useBean id="elenco" class="mieiBean.ScelteDBDerby" />
7   <H4><font color="#0000FF">
8     Scegli tra questi argomenti:
9     <!-- dispongo nel Bean un list box come risultato del metodo getArgomenti della classe ScelteDBDerby -->
10    <jsp:getProperty name="elenco" property="argomenti"/>
11     in lingua :
12    <!-- dispongo nel Bean un list box come risultato del metodo getArgomenti della classe ScelteDBDerby -->
13    <jsp:getProperty name="elenco" property="lingue"/>
14    dell'editore:
15    <!-- dispongo nel Bean un list box come risultato del metodo getEditori della classe ScelteDBDerby -->
16    <jsp:getProperty name="elenco" property="editori"/>
17    </font></H4>
```

avviando l'esecuzione, otteniamo questa videata:



# 6 ESERCIZI IN LABORATORIO

Effettuate le scelte, la pagina di risposta caricaLibri2.jsp si presenta come quella della seguente figura:

Il codice completo è disponibile nei file scegliDBBeanDerby.jsp e caricaLibri2.jsp.

## METTITI ALLA PROVA

• Connessione a Derby

1 Aggiungi alla pagina finale la possibilità di visualizzare l'elenco dei libri ordinato, scegliendo tramite una coppia di radio button il criterio di ordinamento, cioè o per titolo o per codice ISBN.

2 Successivamente trasforma ogni risultato in un link che apra un'ulteriore pagina che riporti la scheda completa con tutti i dati relativi al libro scelto.

3 Aggiungi infine un carrello nel quale memorizzare i libri che hai deciso di acquistare: potrai vedere l'importo totale e il dettaglio della tua spesa quando clicchi sull'icona che ti porta alla cassa.

# I Web Service e le API di Google

UNITÀ 6

**LEZIONE 1**
Introduzione ai Web Service: protocolli SOAP e REST

**LEZIONE 2**
La geolocalizzazione con le API di Google e LeafletJS

**Esercizi in Laboratorio**

L1 Strumenti per lo sviluppo e il testing di Web Service: framework e SDK (Software Development Kit)

L2 Realizzazione di un Web Service SOAP in Java

L3 Realizzazione di un Web Service REST in Java

L4 Realizzazione di un Web Service REST in PHP

L5 Integrazione dei servizi API nelle applicazioni PHP

L6 Utilizzo avanzato delle API di Google in PHP

L7 Le API di Google: interazione tra JavaScript, AJAX, PHP e SQL


## CONOSCENZE

- Avere chiaro il concetto di servizio di rete
- Conoscere il concetto di middleware
- L'evoluzione del modello client-server
- Le caratteristiche del modello SOAP
- Le caratteristiche del modello REST

## COMPETENZE

- Individuare i benefici delle tecnologie Web Service
- Utilizzare NetBeans come ambiente di sviluppo di Web Service
- Saper scrivere un servizio REST in Java
- Saper scrivere un servizio REST in PHP
- Saper testare un servizio REST con Wizdler
- Saper testare un servizio REST con Postman

## ABILITÀ

- Scegliere i servizi più per le applicazioni di rete
- Definire e realizzare servizi SOAP
- Definire e realizzare servizi REST
- Utilizzare API all'interno dei propri programmi
- Definire e realizzare servizi SOAP
- Saper effettuare il deployment di un servizio su un Web Server
- Saper integrare la connessione ai database nei propri servizi REST

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

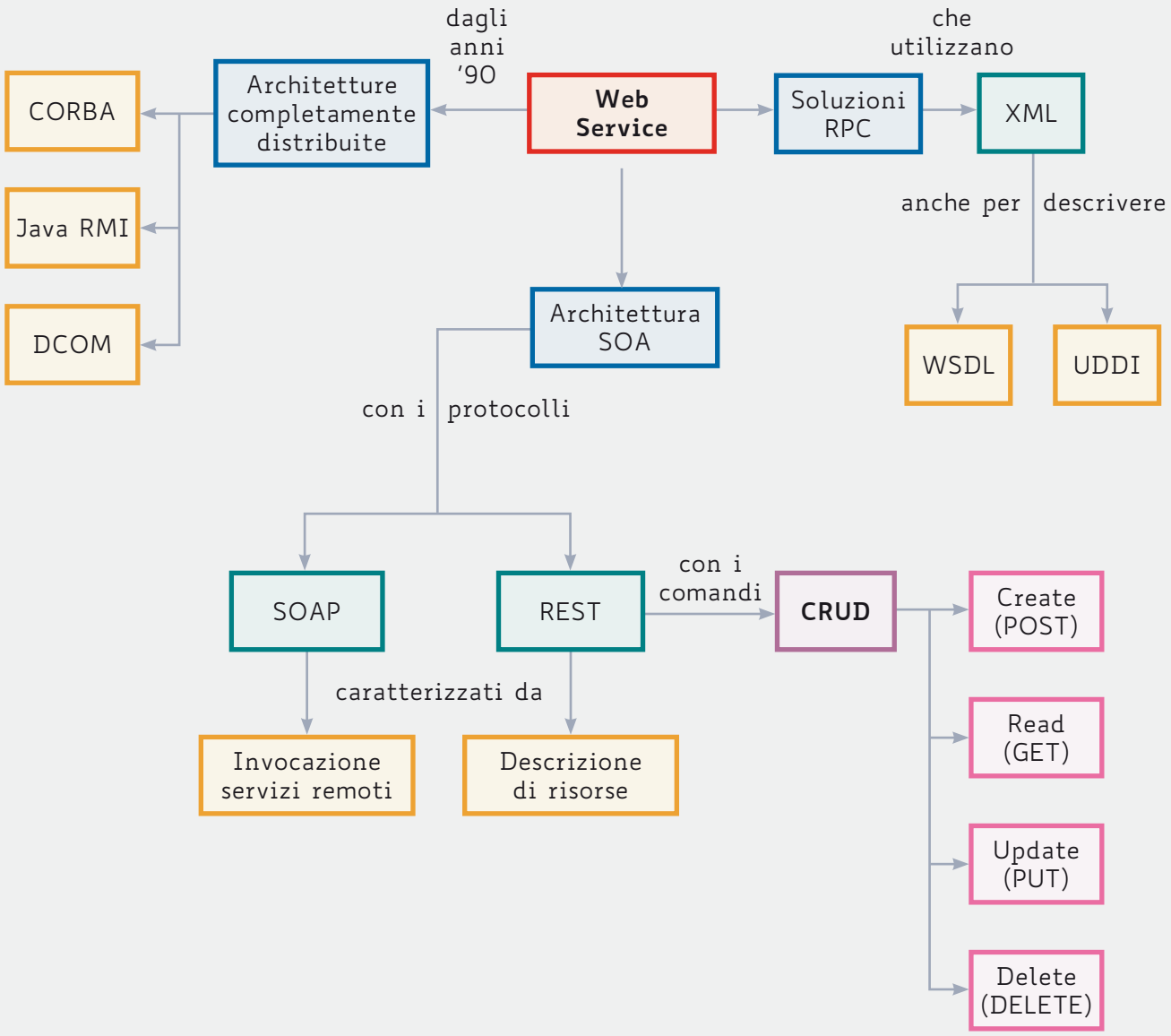
# 1 Introduzione ai Web Service: protocolli SOAP e REST

**IN QUESTA LEZIONE IMPAREREMO...**

- a conoscere i Web Service
- i principi dell'architettura SOA e i protocolli SOAP e REST
- i RESTful Web Service

## MAPPA CONCETTUALE

didattica inclusiva



**Web**
Il World Wide Web (www), denominato anche semplicemente "Web" nasce nel 1991 da un'idea di Tim Berners-Lee. Sempre a Berners-Lee si deve il primo server per il Web (Httpd) e il primo client (un browser e un editor), le specifiche URL e la prima versione del linguaggio per documenti ipertestuali (HTML). Il primo sito Web venne pubblicano nel 1991.

## Introduzione

Il **Web** rappresenta un'enorme piattaforma per la condivisione di documenti in formato ipertestuale interconnessi mediante parole chiave, indipendentemente dalla piattaforma hardware o software utilizzata.
Inoltre il web si è evoluto grazie alla standardizzazione di alcuni semplici concetti, come:

- URI: il meccanismo per individuare risorse in una rete;
- Http: il protocollo semplice e leggero per richiedere una risorsa a una macchina;
- HTML: il linguaggio per la rappresentazione dei contenuti.

**AREA DIGITALE**
**W3C - Standardizzazione**

La necessità di definire regole comuni porta alla standardizzazione del Web grazie all'avvento nel 1994 del W3C (World Wide Web Consortium): esso riveste un'enorme importanza sia da un punto di vista delle dimensioni che degli investimenti e ciascun standard definito dal W3C, chiamato W3C Recommendation, viene sviluppato sulla base delle indicazioni e delle necessità individuate da altre organizzazioni e aziende.
Nel corso degli anni la semplice condivisione di documenti ha subito un'evoluzione che è davanti agli occhi di tutti noi, sia dal punto di vista concettuale che delle modalità attraverso le quali vengono condivisi i documenti stessi.
Il Web è diventato "una piattaforma applicativa distribuita" dove ai documenti testuali si sono aggiunti contenuti multimediali; alle staticità delle pagine HTML dei primi anni si è passati a un'interazione tra più componenti software e a una dinamicità anche dovuta alla presenza di applicazioni software interoperabili: vennero introdotte lato server inizialmente le CGI e successivamente JSP, PHP, ASP ecc.

Le CGI permettevano di invocare un'applicazione presente sul server in grado di trasmetterne il risultato al client come pagina HTML, implementando cosi il dinamismo nelle pagine.

Ma l'evoluzione ha creato un insieme di problemi, tra i quali ricordiamo che il più importante è quello dell'integrazione:

**a** tra le applicazioni informatiche sviluppate in maniera indipendente nei diversi angoli del mondo;
**b** tra gli applicativi esistenti all'interno della singola azienda, dovuti alla presenza di tecnologie eterogenee esistenti all'interno della stessa organizzazione.

Inoltre l'enorme successo che ha avuto il Web negli anni passati, grazie all'evoluzione tecnologica, ha spinto la rete ad assumere un ruolo più completo, anche a seguito delle nuove esigenze legate alla possibilità di interazioni fra applicazioni in diversi contesti.

**ESEMPIO**

Oggi un gestionale aziendale è in diretta interazione con la pubblica amministrazione, inviando, per esempio, la fattura elettronica e i dati per la compilazione automatica del 730 precompilato, con le banche, con i servizi finanziari e i mercati ecc.

Il modello a documenti del Web non è adatto all'interazione fra applicazioni e per le transazioni economiche e fiscali:

- la modalità di localizzazione dei documenti mediante gli URL è uno strumento troppo rudimentale per esprimere richieste complesse e articolate;
- il linguaggio HTML non è in grado di effettuare una chiara separazione tra fra forma e contenuto e non è in grado di garantirne sicurezza e riservatezza.

Con la diffusione di architetture completamente distribuite c'era la necessità di stabilire regole di "comportamento" tra i diversi attori: negli anni Novanta nascono architetture complesse, vincolate dalla limitatezza

del protocollo di comunicazione e dalle tecnologie coinvolte, che cercano comunque di fornire specifiche per rappresentare gli oggetti che vengono inviati e ricevuti; tra le più famose ricordiamo:
- CORBA (Common Object Request Broker Architecture) standard multipiattaforma per linguaggi orientati agli oggetti (1992);
- Java RMI (Remote Method Invocation) specifico del mondo Java (1996);
- DCOM (Distributed COM) tecnologia proprietaria di Microsoft (1996).

Per superare i limiti del protocollo e per rendere totale l'interoperabilità tra i sistemi e le diverse piattaforme operative, in modo da far comunicare componenti software attraverso la rete indipendentemente dalla loro architettura software/hardware, si propongono le prime soluzioni RPC (Remote Procedure Call) con l'utilizzo di XML; si iniziano a utilizzare meccanismi tali per cui client e server possono essere basati su linguaggi e tecnologie diverse e possono essere implementati praticamente con qualsiasi linguaggio.

Con le RPC un utente/cliente può invocare un servizio remoto utilizzando l'XML come formato di rappresentazione e di ricevere un risultato sempre in tale formato.

Il passo successivo è stato quello di cercare di rendere standard le tecnologie coinvolte: nascono i primi Web Service, che implementano meccanismi tali da consentire di far interagire in maniera trasparente applicazioni sviluppate con linguaggi di programmazione diversi, che girano su sistemi operativi eterogenei.

Sinteticamente un Web Service è quindi un sistema software progettato per supportare un'interazione tra applicazioni, utilizzando le tecnologie e gli standard Web.

**API**
Le API (acronimo di Application Programming Interface, ovvero Interfaccia di programmazione delle applicazioni) sono set di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi.

## Che cos'è un Web Service?

Un Web Service rappresenta la disponibilità di un servizio attraverso il Web, come per esempio le prenotazioni online di un biglietto aereo: le disponibilità del posto vengono restituite grazie a un collegamento in tempo reale, in quanto le compagnie aeree espongono i dati dei voli (orari, posti, tariffe, ecc.) attraverso una **API** che il server interroga con la tecnica del Web Service.

In altre parole un Web Service è una piattaforma indipendente di comunicazione che permette a diverse applicazioni di dialogare tra loro attraverso uno standard di comunicazione; è indipendente dalla piattaforma, trasparente per gli utilizzatori che sono completamente ignari dei dettagli della sua implementazione, che può essere effettuata in Java, Visual Basic, C++ ecc.

Per poter far comunicare il client (Service Consumer) con il server (Service Provider) al quale viene inviata una richiesta, è necessario avere a disposizione:
- un medium, cioè uno strumento di comunicazione, come per esempio il protocollo Http;
- un formato, cioè una lingua interpretabile da diverse applicazioni, come per esempio XML o JSON.

**ESEMPIO**

Si pensi, per esempio, alla necessità di effettuare un pagamento online su una piattaforma di e-commerce, utilizzando come strumento la carta di credito: tutte le operazioni legate alla transazione di denaro, a partire dalla verifica della correttezza dei dati della carta di credito fino allo "spostamento fisico" del denaro può essere demandato a un sevizio Web.

Riportiamo la definizione di servizi Web presente sul sito ufficiale del W3C:

> *"Un WS è un'applicazione software identificata da un URI, le cui interfacce pubbliche e relativi binding sono definiti e descritti in XML. La sua definizione può essere trovata da altre applicazioni software. Questi ultimi possono poi interagire con il WS seguendo le direttive presenti nella definizione del servizio, usando messaggi XML trasportati da protocolli Internet."*
>
> ("Web Services Architecture", https://www.w3.org/TR/ws-arch/)

La tecnologia dei Web Service non è una tecnologia proprietaria e i dettagli implementativi sono nascosti da un'interfaccia in formato XML: tutte le tecnologie abilitanti alla base dei Web Service sono tecnologie aperte e/o standard, come XML, WSDL e UDDI, che descriveremo in seguito.
Un Web Service è quindi un sistema software progettato per l'interazione tra diversi elaboratori in rete; tale caratteristica si ottiene associando all›applicazione un›interfaccia software che espone all'esterno il servizio associato offerto dal server, utilizzando la quale altri sistemi, i client, possono interagire con l'applicazione stessa attivando le operazioni descritte nell'interfaccia, siano essi servizi o richieste di procedure remote, effettuati tramite appositi messaggi di richiesta.
I messaggi di richiesta possono avere un formato particolare e sono sempre incapsulati e trasportati tramite i protocolli del Web (solitamente Http sulla porta 80 per evitare il blocco dei firewall), da cui deriva appunto il nome Web Service.

> C'è una differenza importante tra un servizio Web e qualsiasi altra cosa, come un sito Web: il contenuto di un sito Web è creato per fare in modo che un essere umano possa leggerlo e capirlo, mentre un servizio Web è una comunicazione tra due computer.

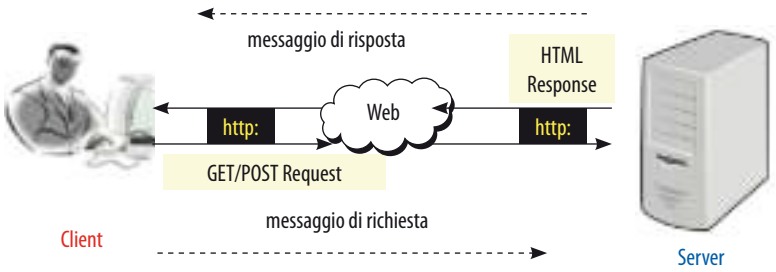
Un Web Service non trasmette testo e grafica, ma soddisfa richieste, che interagiscono direttamente tra due computer spesso senza alcun intervento umano: i computer possono utilizzare i servizi Web per comunicare direttamente tra loro a basso livello, inviare e ricevere dati, condividere funzioni e metodi.

> Un servizio è quindi una risorsa che offre un set di funzionalità astratte mediante un'interfaccia che ne nasconde i dettagli implementativi garantendo indipendenza dalla sua implementazione.

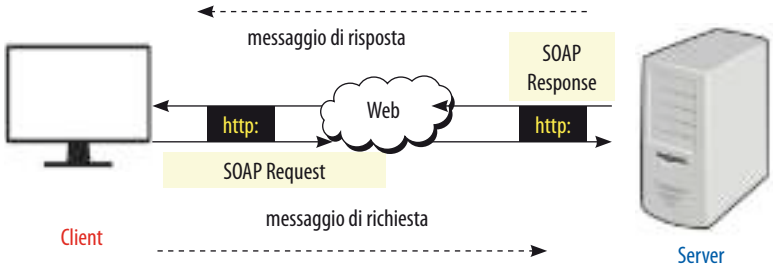
Possiamo azzardare come affermazione che "un servizio Web è il backend del mondo web".

Nella sua forma più semplice, un Web Service è un framework utilizzato per fare conversare due computer tra loro tramite il Web, utilizzando una delle due soluzioni seguenti, che sono le più utilizzate ai giorni nostri:
- Web Service SOAP (medium Http (POST) e formato XML);
- Web Service REST (medium Http (POST, GET, PUT, DELETE) e formato XML, JSON, Text)

Schematicamente, possiamo confrontare lo schema di interazione client-server nel Web tradizionale e in presenza di un Web Service.
**a** Web: un browser richiede una pagina Web tramite http.




**b** Web Service: un client, magari realizzato con un programma Java, invoca un Web Service SOAP (oppure REST) e Http.




Questo meccanismo consente di realizzare porzioni di funzionalità in maniera indipendente e su piattaforme potenzialmente incompatibili facendo interagire i vari pezzi tramite tecnologie Web e creando un'architettura facilmente componibile.
Quando uno sviluppatore crea un Web Service, si deve preoccupare di definire:

1 la logica di funzionamento del servizio, ovvero quello che dovrà fare: questo può spaziare da una semplice classe a un'applicazione molto complessa;
2 il Web Container (Server) su cui verrà installato (per esempio Apache, Tomcat, JBoss, GlassFish) e che ne consentirà l'uso da parte dei client.

## Un nuovo modello basato su XML: l'architettura SOA

Nella definizione dei Web Service si è scelto di adottare un approccio di tipo evolutivo continuando a utilizzare Http come protocollo di trasporto delle informazioni e, quindi, di poter utilizzare l'infrastruttura già esistente, la rete Internet, e di individuare uno strumento più adeguato per descrivere gli elementi del colloquio, cioè la richiesta e la risposta: per questa funzione si è pensato a XML.

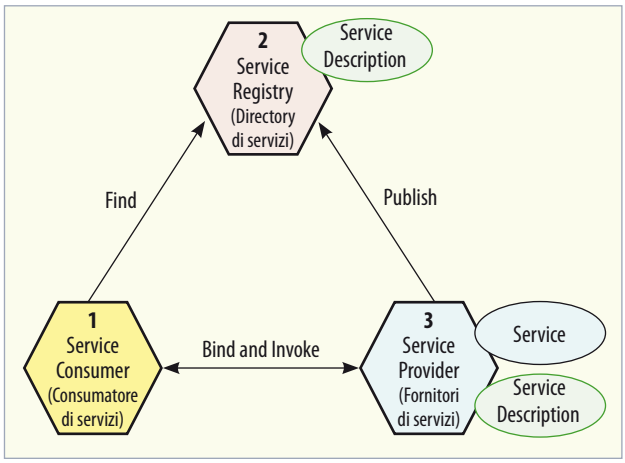
**SOA**
L'approccio SOA è anche noto con il nome di WS-*, con un chiaro richiamo al prefisso utilizzato nei vari standard definiti dal W3C.

Si viene a definire l'architettura orientata ai servizi, la Service Oriented Architecture (**SOA**), con l'obiettivo primario di tagliare i costi e massimizzare l'uso delle tecnologie esistenti e di essere in grado di servire meglio gli utenti e di essere reattivi di fronte all'evoluzione tecnologica in un mercato globalizzato e estremamente competitivo.
L'architettura SOA deve fornire una piattaforma per costruire servizi applicativi aventi le seguenti caratteristiche:

- loosely coupled: l'utente di un servizio non deve preoccuparsi di scegliere con quali servizi comunicare, "ci pensa" l'infrastruttura;
- location transparent: l'infrastruttura deve nascondere quanti più dettagli tecnici possibili, relativi a tecnologie tra loro differenti, al richiedente del servizio;
- protocol independent: deve essere possibile aggiornare in qualunque momento le implementazioni, senza modificare le interfacce.

Gli attori dell'architettura SOA sono sostanzialmente tre:

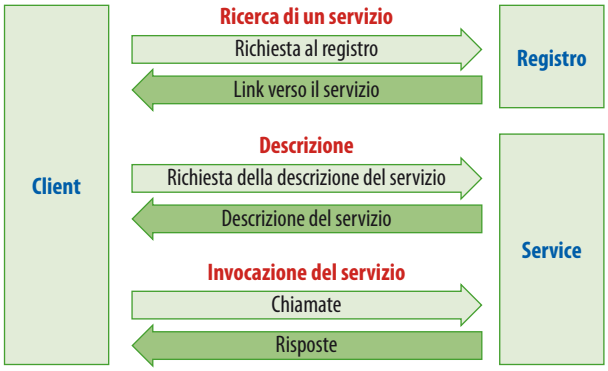
1 la ricerca dei servizi richiesti dall'utente avviene a opera del Service Customer (o Requestor), che è un'entità software che richiede il servizio, agendo come client, interrogando il Service Registry;
2 se i servizi esistono, il Service Registry, che contiene il repository dei servizi, fornisce al consumer l'interface contract e gli endpoint access del Service Provider, fornisce cioè al consumer tutte le informazioni riguardo a quale servizio utilizzare e la sua localizzazione, cioè in quale "percorso" lo può trovare;
3 quando arriva la richiesta al Service Provider, che implementa le specifiche di un servizio, la accetta, rende disponibile il servizio e pubblica il contratto che ne descrive l'interfaccia, tramite un'apposita entità detta broker ed esegue il relativo codice fornendo la risposta.



In sintesi la dinamica avviene in tre fasi:
**a** richiesta del servizio;
**b** richiesta della modalità di utilizzo del servizio (descrizione della interfaccia del servizio);
**c** invocazione del servizio.

**AREA DIGITALE**
**Categorie di elementi SOA**

Naturalmente affinché un servizio sia disponibile deve essere pubblicata (publish) nel Service Registry dal Service Provider una copia della descrizione del servizio (service description): la ricerca del servizio (find) viene effettuata analizzando e individuando proprio tra le descrizioni presenti nel registry quello che soddisfa le aspettative e, una volta trovato, avviene l'effettivo utilizzo tramite le modalità di chiamata anch'essere presenti nella descrizione (bind and invoke).

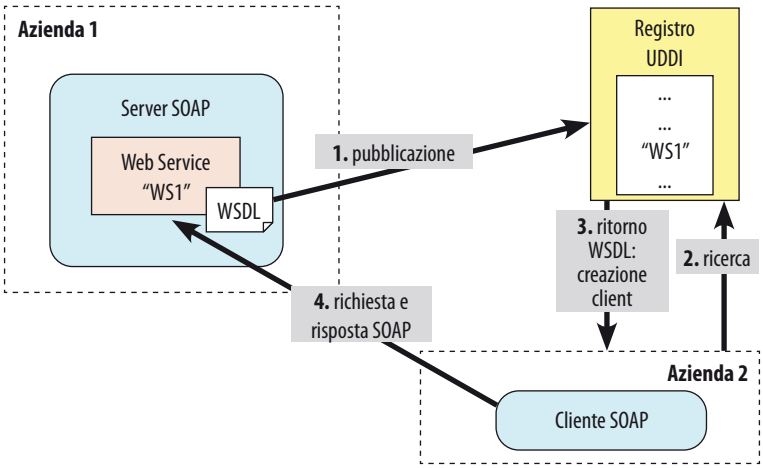



## Protocolli e tecnologie dei Web Service

Allo stato attuale esistono due approcci alla creazione di Web Service:
- il primo è basato sul protocollo standard SOAP (Simple Object Access Protocol), per lo scambio di messaggi per l'invocazione di servizi remoti, e riproduce in ambito Web un approccio a chiamate remote RPC, tipico di protocolli di interoperabilità come CORBA, DCOM e RMI;
- il secondo è ispirato ai principi architetturali tipici del Web e si concentra sulla descrizione di risorse, sul modo di individuarle nel Web e sul modo di trasferirle da una macchina all'altra: questo è l'approccio che oggi trova maggiori consensi e prende il nome di REST (REpresentational State Transfer).

# Il protocollo SOAP

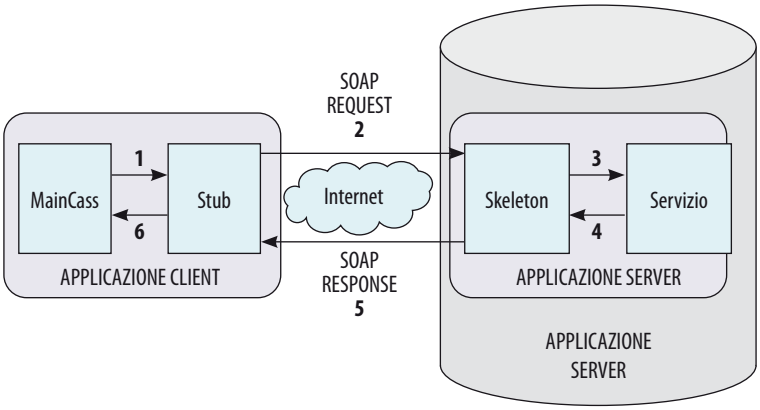
Un protocollo per scambiare messaggi relativi a Web Service è SOAP (Simple Object Access Protocol) che utilizza l'approccio della chiamata remota di metodi esposti da oggetti residenti su altri nodi della rete: è basato su XML per lo scambio d'informazioni in un ambiente distribuito con il quale si definisce un descrittore che indica in un formato comune le modalità per trasmettere dati tra client e service.
Il ciclo di vita di un Web Service SOAP si può riassumere nelle seguenti fasi:
1. il server (provider) crea il servizio la cui descrizione è affidata a un documento WSDL (Web Service Descriptor Language), formato standard XML, che contiene nome dei metodi, parametri attesi, valori di ritorno ecc.); la pubblicazione del servizio viene affidata a un registro conforme allo standard Universal Description Discovery and Integration (UDDI);
2. il client interroga il registro UDDI, per scoprire i servizi di cui ha bisogno, nella fase di chiamata di discovery, che può essere omessa se si conosce già il servizio che si andrà a utilizzare;
3. se la ricerca ha esito positivo, il client chiede al registro il documento WSDL per individuare gli indirizzi (URI) e i modi con cui interrogarlo;



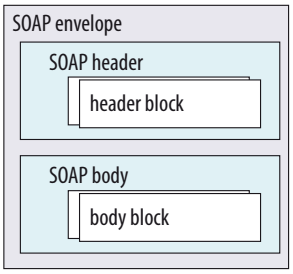
**4.** il client, a partire dal WSDL, crea uno strato software (request agent) che interagisce con lo strato software (provider agent, cioè il servizio vero e proprio) fornitore del servizio; tali componenti, chiamati stub e skeleton (rispettivamente **lato client** e **lato server**), si occupano della gestione dei messaggi SOAP, assemblando e disassemblando messaggi nel formato XML per permettere la comunicazione automatica tra due sistemi informatici.

Nella figura seguente seguiamo in dettaglio l'ultima fase, cioè la comunicazione tra client e server dopo che il client ha analizzato il WSDL:

**1** l'interfaccia **lato client** chiama un metodo del Web Service attraverso lo stub, che si occupa di trasformare la richiesta del client in un messaggio XML conforme al protocollo SOAP;
**2** il messaggio viene spedito al server tramite protocollo SOAP (REQUEST);
**3** lo skeleton riceve il messaggio SOAP, lo trasforma in chiamate al codice del linguaggio in cui è scritto il servizio **lato server** (per esempio in Java);
**4** il servizio **lato server** riceve la chiamata, elabora i dati e, tramite lo skeleton, assembla un messaggio SOAP pronto per essere spedito al client;
**5** la risposta (RESPONSE) viene inviata al client e ricevuta dallo stub;
**6** lo stub estrare la risposta dal messaggio SOAP e la restituisce all'interfaccia che ha chiamato il servizio.

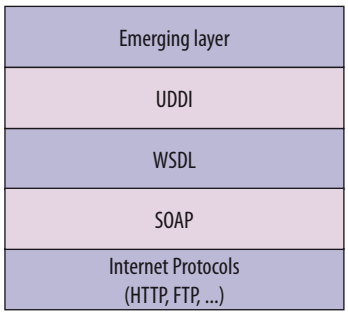



Un messaggio SOAP è formato da una busta (envelope), che contiene un'intestazione e un corpo.
Il corpo (body) racchiude le informazioni che il messaggio vuole effettivamente trasmettere, come per esempio un documento, oppure il nome di un metodo invocato e i relativi parametri. L'intestazione (header) contiene metadati e altre informazioni "non funzionali", come per esempio le credenziali di autenticazione oppure l'ID della transazione.




Possiamo riassume in uno schema la pila protocollare per i Web Service SOAP indicando tecnologie/linguaggi che ogni livello utilizza:

- UDDI (Universal Description, Discovery and Integration): è un registro contenente informazioni utili per la scoperta e l'accesso ai Web Service;
- WSDL (Web Services Description Language): descrive l'interfaccia esterna di un Web Service affinché uno sviluppatore possa creare un client capace di invocarlo;
- SOAP (Simple Object Access Protocol): descrive un protocollo basato su XML che definisce i meccanismi con cui un WS è invocato e il formato dell'input e dell'output (richiesta e risposta); è quindi un protocollo per lo scambio di messaggi;
- XML (eXtensible Markup Language): è un formato per lo scambio dei dati nello strato di trasporto ed è usato per descrivere la struttura dei messaggi scambiati tra servizi Web.



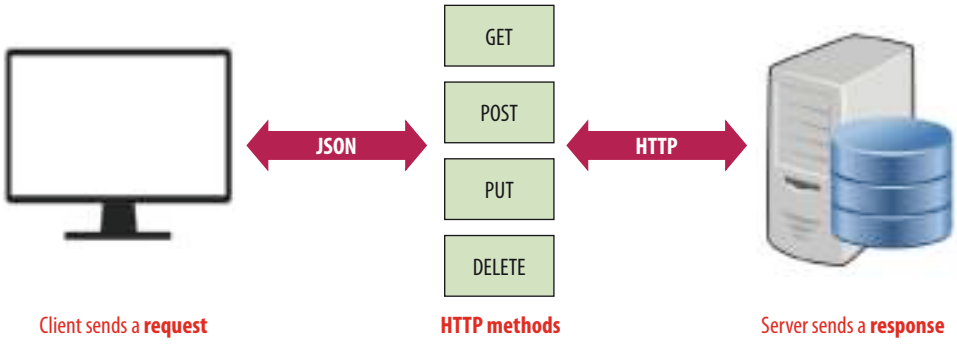

**AREA DIGITALE**

**Caratteristiche e struttura dei componenti pila protocollare SOAP**

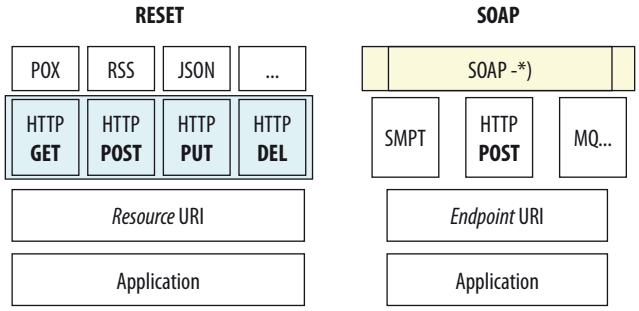
## Il protocollo REST

REST è un approccio alternativo a SOAP per la realizzazione di Web Service e a differenza di SOAP, che si basa sul concetto di chiamata remota e utilizza una complessa infrastruttura di messaggi XML, stub/skeleton e application container, è molto più leggero perché si basa sulla corrispondenza diretta tra:

- oggetti remoti da leggere/scrivere individuati mediante Uniform Resource Identifiers (URI), cioè i comuni link Web;
- comandi Http da applicare agli oggetti (GET, PUT, POST, DELETE).




Possiamo confrontare la struttura protocollare tra REST e SOAP nello schema seguente:




### ESEMPIO

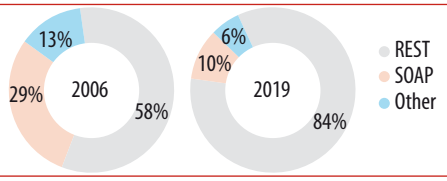
Supponiamo di voler leggere un elenco di prodotti presenti su un catalogo: utilizzeremo semplicemente il metodo Http GET sulla URI che rappresenta il catalogo prodotti e dato che il metodo GET è quello usato per default dai browser Web nell'accesso alle pagine, sarà quindi sufficiente aprire la pagina http://www.nomenegozio.it/catalogo/prodotti.
Il progettista del server farà corrispondere l'URI catalogo/prodotti all'opportuna query sul database e se il server lo supporta, la stessa risorsa può essere richiesta in formati diversi, come per esempio TXT, HTML, XML o JSON.

L'approccio REST tende a conservare e a esaltare le caratteristiche intrinseche del Web evidenziandone la predisposizione a essere una piattaforma per l'elaborazione distribuita e utilizzandone quanto è già esistente sul Web per consentire ad applicazioni remote di interagire.

REST è un modo semplice per organizzare le interazioni tra sistemi indipendenti poiché non prevede esplicitamente nessuna modalità per descrivere come interagire con una risorsa: le operazioni sono implicite nel protocollo Http.

REST assume che le interazioni tra client e server devono avvenire senza memorizzazione di informazioni di stato sul server, cioè le interazioni tra client e server devono essere senza stato (stateless).
REST è un modo semplice per organizzare le interazioni tra sistemi indipendenti e ha iniziato a diffondersi nel Web dal 2005, anche grazie al suo utilizzo nella progettazione di servizi come le API di Twitter.
Il successo di REST rispetto a SOAP è dovuto al fatto che permette un'agile interazione tra client diversi, come telefoni cellulari o altri siti Web e, quindi, non è limitato al Web, e non richiede di imparare nuove convenzioni e protocolli, ma semplicemente sfrutta al meglio l'Http.

Negli ultimi anni REST è emerso come design model predominante, superando di gran lunga l'approccio SOAP, come possiamo notare dal grafico riportato a lato:




## I principi dell'architettura REST

La definizione dell'architettura REST è apparsa per la prima volta nel 2000 nella tesi di dottorato di **Roy Fielding**, *Architectural Styles and the Design of Network-based Software Architectures*, discussa presso l'Università della California, a Irvine. REST definisce un insieme di principi architetturali per la progettazione di un sistema: non si riferisce a un sistema concreto e ben definito né si tratta di uno standard stabilito da un organismo di standardizzazione.

**Roy Fielding**
Roy Fielding è uno dei principali autori delle specifiche dell'HyperText Transfer Protocol (Http) e a lui si deve quindi anche l'espressione "representational state transfer" e il suo acronimo "REST".

Secondo l'approccio REST i principi che rendono il Web adatto a realizzare Web Service si possono riassumere nei seguenti cinque:
**1** identificazione delle risorse con URI in forma di directory (Content Provider);
**2** utilizzo esplicito dei metodi Http;
**3** risorse autodescrittive;
**4** collegamenti tra risorse con interazioni stateless;
**5** comunicazione senza stato in XML, JavaScript Object Notation (JSON) o entrambi.

Per risorsa si intende un qualsiasi elemento e/o oggetto di elaborazione e queste sono gli elementi fondamentali su cui si basano i Web Service REST: ciascuna risorsa deve essere identificata univocamente e in ambito Web il meccanismo più naturale per individuare una risorsa è dato dal concetto di URI.

### ESEMPIO

Sono possibili identificatori di risorse:

```
http://www.miaApplicazione.com/clienti/1111
http://www.miaApplicazione.com/prodotti/2222
http://www.miaApplicazione.com/prodotti?colore=verde
http://www.miaApplicazione.com/ordini/2021/334455
```

Gli URI sono "parlanti": i primi due identificano rispettivamente un determinato cliente e un prodotto, il terzo l'insieme dei prodotti di colore verde mentre il quarto identifica un particolare ordine del 2021.

Il formato utilizzato per mandare e ricevere informazioni con un sistema REST è il JSON; inoltre nell'header della richiesta si possono aggiungere oggetti come token di autenticazione ed eventuali dati che devono essere inviati in richieste PUT o POST possono essere aggiunti come parametri della richiesta o nel body della stessa. Una sessione attiva verrà eliminata dopo un certo lasso di tempo, forzando l'utente a ricreare la sessione di tanto in tanto: tale autenticazione non va effettuata a ogni richiesta dato che l'utente può fornire l'id di sessione nell'header e, quindi, mantenerla aperta.

Riassumiamo le principali differenze tra REST rispetto a SOAP:
**1** un WS REST è incentrato sulla risorsa, mentre un WS SOAP è incentrato sul servizio;
**2** un WS REST gestisce le risorse accessibili con tipiche chiamate Http, mentre un WS SOAP espone dei metodi accessibili in remoto da parte di un client;
**3** in REST si ha una riduzione della parte di configurazione e nei parametri di controllo.

A oggi REST è l'architettura più utilizzata, quasi tutti i Web Service e i provider di API utilizzano REST (+ JSON): è scalabile, veloce, leggera e molto flessibile.

## Mappare le azioni CRUD sui metodi Http

Abbiamo visto come individuare una risorsa, ora descriviamo il meccanismo disponibile per indicare le operazioni che si possono effettuare su di essa: REST utilizza esplicitamente i metodi legati al protocollo Http ovvero i quattro comandi fondamentali che consentono di eseguire le operazioni CRUD: Create, Read, Update e Delete e queste azioni agiscono su oggetti che vengono definiti risorse: in pratica ogni dato è una risorsa che dev'essere presa, caricata, aggiornata, cancellata ecc.

| METODO HTTP | OPERAZIONE CRUD | | DESCRIZIONE |
|---|---|---|---|
| POST | Create | C | Crea una nuova risorsa |
| GET | Read | R | Ottiene una risorsa esistente |
| PUT | Update | U | Aggiorna una risorsa o ne modifica lo stato |
| DELETE | Delete | D | Elimina una risorsa |

### ESEMPIO

Sappiamo che quando inseriamo un URI nella barra degli indirizzi di un browser stiamo in realtà chiedendo al browser di eseguire un metodo Http sulla risorsa individuata dall'URI: implicitamente stiamo eseguendo Il metodo GET, il cui effetto è di accedere a un'istanza della risorsa identificata dall'URI.

Come possiamo osservare questo principio è in netto contrasto con la tendenza generale nell'utilizzazione dei metodi Http: molto spesso il metodo GET viene utilizzato per eseguire qualsiasi tipo di interazione con il server.

### ESEMPIO

Per l'inserimento di un nuovo cliente nell'ambito di un'applicazione Web viene eseguita una richiesta di tipo GET su un URI del tipo:

```
http://www.miaApplicazione.com/aggiungiCliente?cognome=Verdi
```

Questo approccio non è conforme ai principi REST perché il metodo GET serve per accedere alla rappresentazione di una risorsa e non per crearne una nuova.
In pratica questo uso del metodo GET introduce un effetto collaterale che, tra l'altro, può avere delle conseguenze indesiderate se, per esempio, l'URI viene invocato più volte o se la risorsa viene memorizzata in uno dei vari livelli di cache esistenti tra client e server.

Gli sviluppatori che sviluppano un set di API che rispettano la logica imposta da REST creano un set di **RESTful** API.

Un Web Service RESTFull deve garantire l'adesione ai seguenti vincoli tipici dell'architettura REST:

**a** interazione di tipo client-server;

**b** comunicazione di tipo stateless;

**c** possibilità di implementare meccanismi di cache sulle richieste;

**d** interfaccia uniforme;

**e** possibilità di costruire ulteriori layer applicativi in base alle esigenze a partire dal servizio di base.

**RESTful**
Spesso viene utilizzato l'acronimo RESTful nel contesto di REST: REST viene definito con uno "stile architetturale" con delle caratteristiche e principi; RESTful viene generalmente utilizzato per far riferimento a servizi Web che implementano l'architettura REST, ma, in realtà, sono esattamente la stessa cosa.

## Conclusione

L'utilizzo dell'architettura RESTful porta a notevoli vantaggi, di seguito riportiamo i più importanti.

**A** Utilizzo di tecnologie standard: oltre alla semplicità della specifica tutte le tecnologie alla base dei Web Service sono standard, dato che essenzialmente si utilizza XML per la rappresentazione, XML Schema per la validazione e Http per il trasporto; il riuso di tecnologie esistenti, collaudate e adottate da un gran numero di sviluppatori, contribuisce a rendere la tecnologia accessibile.

**B** Separazione tra client e server: il protocollo REST separa completamente l'interfaccia utente dal server e offre alcuni vantaggi in fase di sviluppo:
- permette in grossi progetti di separare nel modo migliore il lavoro tra developer;
- consente l'evoluzione indipendente delle diverse componenti degli sviluppi.

**ESEMPIO**

Supponiamo di aver sviluppato il frontend di una Web Application REST in Angular e che dopo qualche anno venga rilasciato un nuovo framework per il frontend con nuove possibilità "fantastiche": con REST è possibile sostituire il componente frontend senza modificare una riga di codice del backend (e anche il viceversa)!

**C** Portabilità dell'interfaccia ad altri tipi di piattaforme: l'interfaccia REST utilizzata da un frontend può essere riutilizzata a costo zero per sviluppare un'"applicazione mobile".

**D** Visibilità, affidabilità e scalabilità: la separazione tra client e server ha come evidente vantaggio quello di poter scalare facilmente il sistema; client e server possono risiedere su server differenti e con poca difficoltà si può scalare orizzontalmente un componente dell'architettura con nessun impatto per l'altro componente.

**E** Indipendenza da linguaggi e tecnologie specifiche: l'API REST è sempre indipendente dal tipo di piattaforma o linguaggio e si adatta sempre al tipo di linguaggio o piattaforme utilizzate dando libertà nella scelta del linguaggio o framework per l'implementazione di un componente; possiamo avere server PHP, Java, Python o Node.js o qualsiasi linguaggio di programmazione e l'unico vincolo è che le risposte alle richieste vengano sempre eseguite nella lingua utilizzata per lo scambio di informazioni, normalmente JSON.

**F** Ampio consenso da parte di diverse aziende: Microsoft è stata la prima azienda a proporre modalità di comunicazione basate su XML e ha anche avuto la lungimiranza di non mantenere proprietarie le specifiche ma di renderle pubbliche, in modo che anche altri big dell'informatica potessero intervenire in prima persona sia per estenderli sia per supportarli.

**G** Presenza di tool che aiutano alla creazione di nuovi servizi: sono disponibili sia in forma gratuita sia a pagamento alcuni dei numerosi tool adatti a creare e a fruire Web Service per le più svariate piattaforme software e per tutti i più diffusi linguaggi di programmazione.

Esistono moltissimi servizi Web RESTful che utilizziamo regolarmente senza saperlo: alcun sono disponibili gratuitamente mentre altri sono a pagamento; di seguito ne elenchiamo alcuni tra i più famosi:
- Google: fornisce quasi la totalità dei propri servizi come API RESTful, per esempio per Gmail come per Google Translate;
- SoundCloud: la API RESTful del famoso sito per caricare e ascoltare musica permette di disporre di un player per lo streaming integrabile su qualsiasi piattaforma, che generalmente viene interrogato direttamente in JavaScript;
- Vimeo: anche il celebre servizio di video streaming offre una ricca API per gestire video e playlist;
- Mashape: una delle raccolte di Web Service tra le più celebri al mondo, che ha esposto tutti i propri servizi pubblicamente disponibili su PublicAPI.
- Yahoo: offre un enorme database ricercabile di servizi, sia gratuiti sia a pagamento.

### Un esempio di utilizzo delle REST API di Google

L'utilità delle REST API risiede nella possibilità di essere immediatamente fruibili dagli sviluppatori: Google è stato uno dei primi servizi a fare un uso intensivo di REST API sia per gli utenti sia per gli sviluppatori.

Google Maps, per esempio, consente di cercare indicazioni tramite un passaggio di parametri:
- *dir* specifica il tipo di servizio
- *bellagio* è la località di partenza
- *como* è la località di destinazione

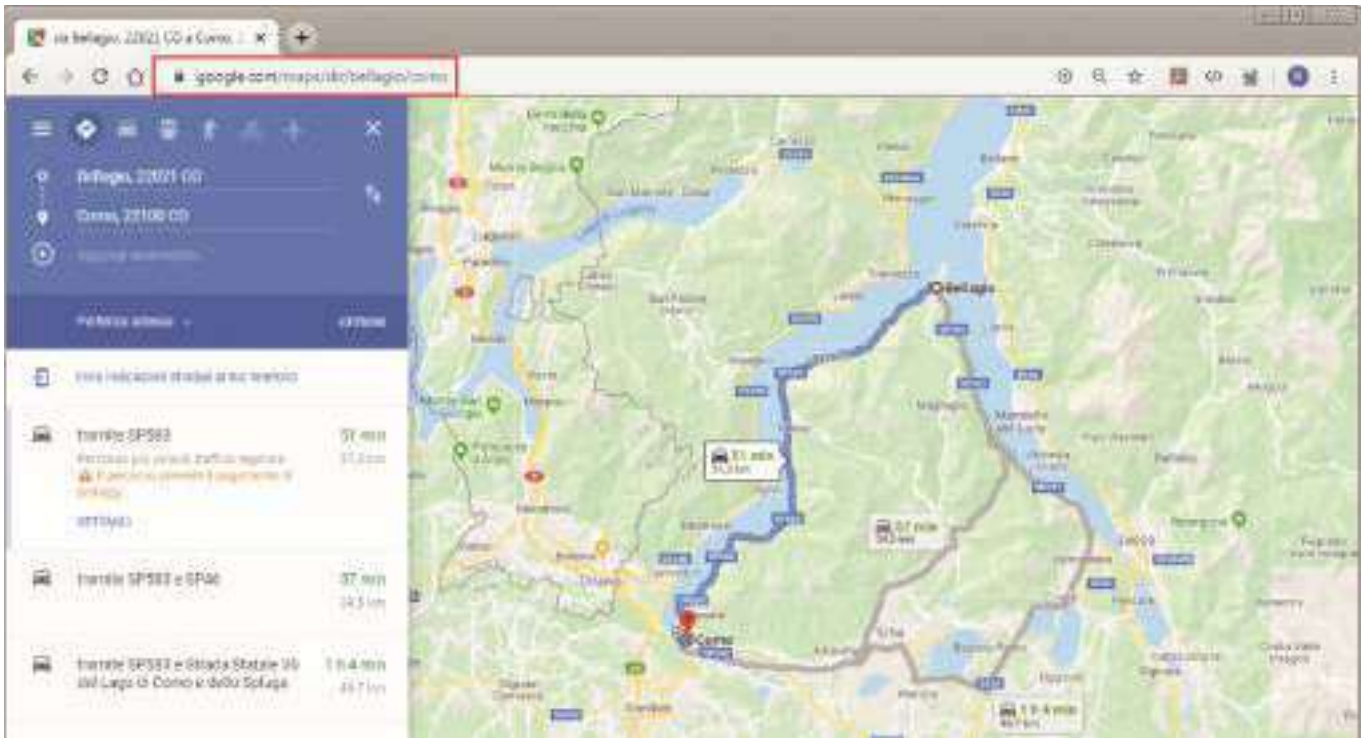
**https://www.google.com/maps/dir/bellagio/como**

**nome della API**

**specifica il tipo di servizio**

**è la località di partenza**

**è la località di destinazione**



La guida completa dei servizi REST API si trova all'indirizzo: https://developers.google.com/fit/rest/v1/get-started.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Quali tra le seguenti non sono architetture complesse nate negli anni Novanta?**
**a** CORBA
**b** SOA
**c** Java RMI
**d** DCOM

**2 Un Web Service è:**
**a** un componente software
**b** un componente hardware
**c** un componente hardware-software
**d** un componente middleware

**3 Per descrivere gli elementi del colloquio con i Web Service si è pensato al linguaggio:**
**a** Http
**b** XML
**c** HTML
**d** JavaScript
**e** C++

**4 L'architettura SOA deve fornire una piattaforma per costruire servizi applicativi aventi le seguenti caratteristiche:** (*più risposte*)
**a** loosely coupled
**b** context free
**c** location transparent
**d** protocol independent

**5 L'acronimo SOAP deriva da:**
**a** Standard Object Access Protocol
**b** State Object Access Protocol
**c** Simple Object Access Protocol
**d** Standard Object Admin Protocol

**6 Il protocollo SOAP:** (*più risposte*)
**a** è basato sul protocollo standard SOAP
**b** è ispirato ai principi architetturali tipici del Web
**c** sfrutta un meccanismo di scambio di messaggi per l'invocazione di servizi remoti
**d** riproduce in ambito Web un approccio a chiamate remote RPC
**e** si concentra sulla descrizione di risorse

**7 L'acronimo REST deriva da:**
**a** REpresentational State Transfer
**b** REmote State Transfer
**c** REpresentational Service Transfer
**d** REmote State Transfer

**8 Il protocollo REST:** (*2 risposte*)
**a** è basato sul protocollo standard SOAP
**b** è ispirato ai principi architetturali tipici del Web
**c** sfrutta un meccanismo di scambio di messaggi per l'invocazione di servizi remoti
**d** riproduce in ambito Web un approccio a chiamate remote RPC
**e** si concentra sulla descrizione di risorse

**9 L'acronimo UDDI deriva da:**
**a** Universal Description, Definition and Integration
**b** Uniform Description, Discovery and Integration
**c** Universal Description, Discovery and Integration
**d** Uniform Description, Definition and Integration

**10 L'acronimo WSDL deriva da:**
**a** Web Services Discovery Language
**b** Web Services Description Language
**c** Web Standard Description Language
**d** Web Standard Discovery Language

**11 La pila protocollare per i Web Service SOAP è costituita da:** (*indica l'intruso*)
**a** UDDI
**b** WSDL
**c** SDK
**d** SOAP
**e** XML

## VERO/FALSO

1 Un Web Service può essere pubblicato, localizzato e consumato attraverso il Web. V F
2 Un Web Service è appositamente progettato per eseguire un compito specifico. V F
3 La tecnologia dei Web Service è una tecnologia proprietaria. V F
4 Nei Web Service si è scelto di utilizzare Http come protocollo di trasporto. V F
5 I servizi vengono memorizzati in un archivio (il Service Registry) che contiene il repository dei servizi. V F
6 Il protocollo REST un primo approccio è basato sul protocollo standard SOAP per lo scambio di messaggi per l'invocazione di servizi remoti. V F
7 SOAP per la pubblicazione del servizio si affida a un registro conforme allo standard UDDI. V F
8 In REST il server crea il servizio la cui descrizione è affidata a un documento WSDL. V F
9 Sulle risorse in REST si eseguono solo operazioni CRUD. V F
10 Un Web Service REST è incentrato sulla risorsa mentre un Web Service SOAP è incentrato sul servizio. V F

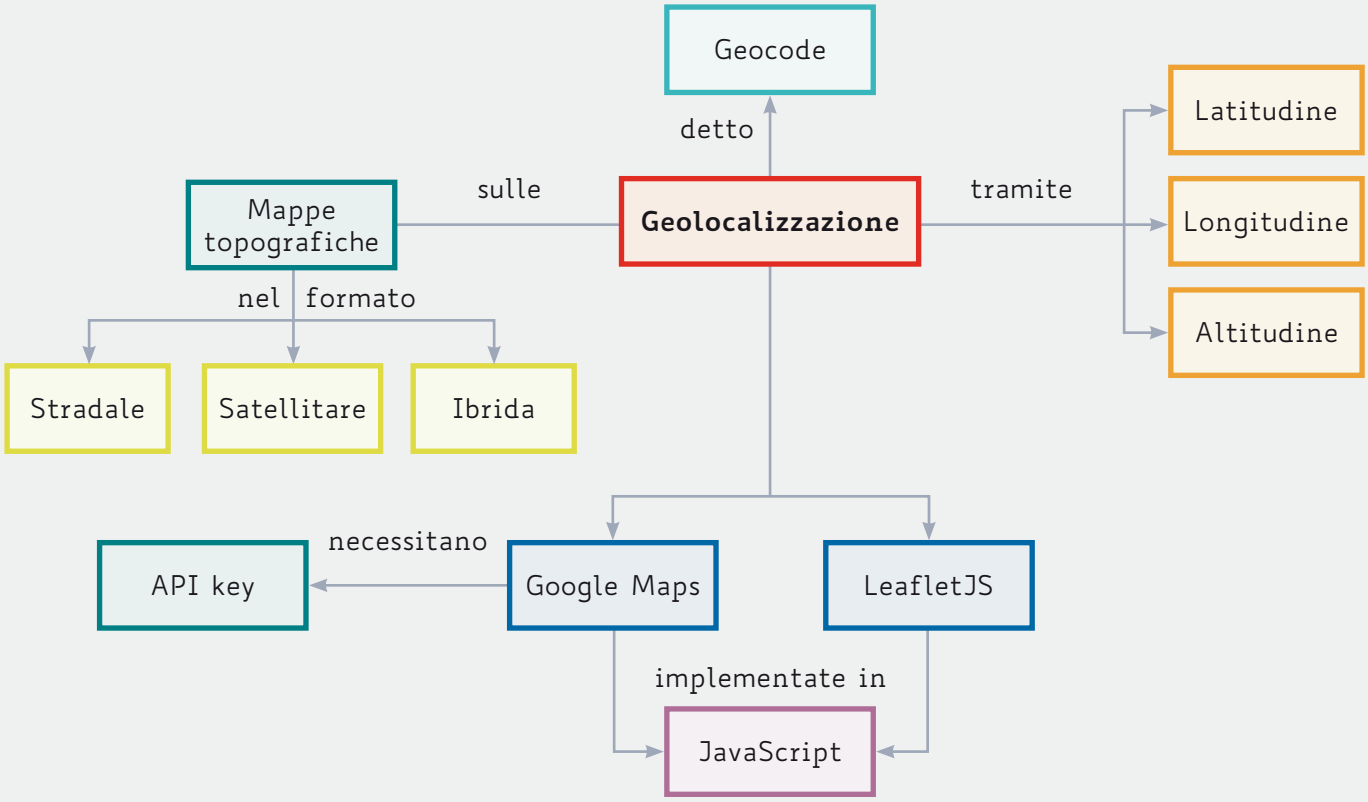
# 2 La geolocalizzazione con le API di Google e LeafletJS

## IN QUESTA LEZIONE IMPAREREMO...

- a conoscere la geolocalizzazione e le API di Google Maps
- a utilizzare le classi di google.maps associando eventi
- a utilizzare la libreria Leaflet per "effetti speciali"

## MAPPA CONCETTUALE

didattica inclusiva




**Topografia**
La parola topografia deriva dal greco τόπος ("luogo") e γραφεῖν ("scrivere") e in senso ampio è la scienza che si occupa di determinare con precisione la posizione di entità sulla superficie terrestre e successivamente di rappresentare tali entità sulla carta.

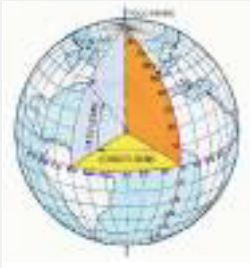
## La geolocalizzazione

La geolocalizzazione è l'identificazione della posizione geografica, mediante le sue coordinate di latitudine e longitudine, di un dato oggetto nel mondo reale secondo varie possibili tecniche. Google Maps (il cui nome originario era Google Local) è un servizio accessibile dal relativo sito Web che consente la ricerca e la visualizzazione di **mappe topografiche** di quasi tutto il pianeta.

**Coordinate geografiche**

Le coordinate geografiche servono a identificare la posizione di un punto sulla superficie terrestre, si dividono in latitudine, longitudine e altitudine:

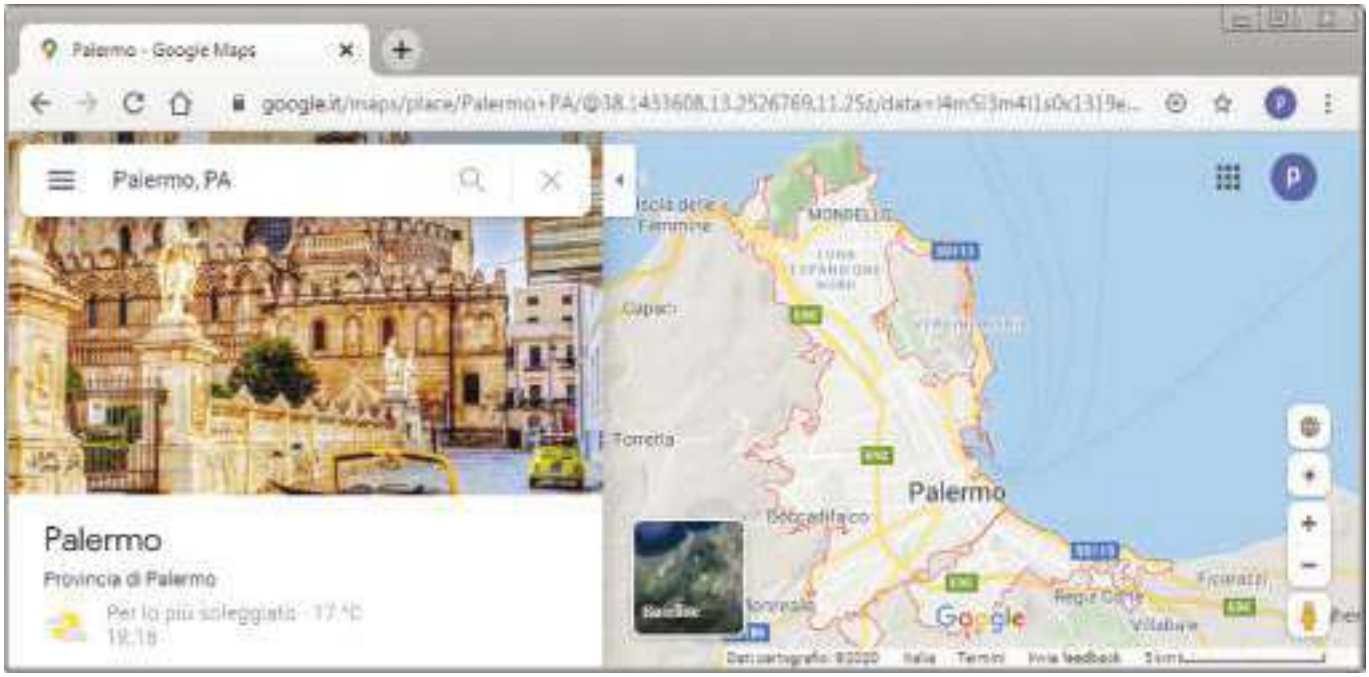
- la latitudine è la distanza angolare del punto dall'Equatore;
- la longitudine è la distanza angolare del punto dal meridiano di riferimento (Greenwich) lungo lo stesso parallelo del luogo;
- l'altitudine è la distanza, misurata lungo la verticale del punto considerato sulla superficie terrestre dal livello del mare.



Con Google Maps è anche possibile ricercare servizi in particolari luoghi, tra cui ristoranti, monumenti, negozi, trovare un possibile percorso stradale tra due punti e visualizzare foto satellitari (statiche) di molte zone con diversi gradi di dettaglio.
Oltre a queste funzioni, Google Maps offre anche una ricerca di attività commerciali sulle stesse mappe e mette a disposizione delle API che consentono di localizzare elementi presenti sulle mappe in base alle **coordinate geografiche**.

Le API di Google Maps utilizzano JavaScript come linguaggio predefinito, necessario per la loro implementazione, come molte altre applicazioni Web di Google.

Le API di Google Maps nascono nel giugno 2005 per consentire agli sviluppatori di integrare l'applicazione Google Maps nei loro siti Web: è un servizio gratuito che attualmente non contiene pubblicità anche se Google, nei suoi termini di utilizzo, si riserva il diritto di visualizzare annunci in futuro. Con le API di Google Maps possiamo incorporare mappe all'interno del nostro sito.
Google Maps consente di visualizzare la mappa di un luogo prescelto e richiedere informazioni di vario tipo, per esempio dove si trova un'azienda o una località, insieme ai dati di contatto e alle indicazioni stradali per raggiungerli.




L'utente può effettuare varie operazioni sulla mappa, come per esempio il trascinamento che mostra altre zone adiacenti senza ricaricare la nuova area, oppure l'ingrandimento o la riduzione dello zoom, oppure ancora la funzionalità Street View che verrà mostrata più avanti.
L'utente può anche scegliere se visualizzare una semplice mappa con le sole indicazioni stradali, oppure una carta satellitare o ibrida.

**Geocode**
Il geocode è la conversione di un indirizzo reale, come per esempio "Via Milano, 132, Roma, Italia", in coordinate geografiche formate da latitudine e longitudine. Google impone che le proprie coordinate siano utilizzate solo su Google Maps e non per altri scopi. È possibile visualizzare un geocode solo su Google Maps.

## La API key

Per poter utilizzare le Google API e i relativi servizi dobbiamo ottenere una API key, una chiave che ci permetta di poter accedere al servizio, univocamente per ogni dominio. Tali servizi vengono offerti gratuitamente per un numero limitato di accessi al sito: esiste un limite di 15.000 generazioni di **geocode** nella pagina con la mappa generata dalle API.

La procedura per ottenere la Google API è molto semplice: basta seguire le indicazioni riportate a partire dall'indirizzo: https://console.developers.google.com.
Per i nostri esempi utilizzeremo le Google Maps JavaScript API in quanto proporremo un codice per dispositivi fissi.

La Google key è una stringa di circa 50 caratteri, come la seguente:

```
AIzaSyCFEZBrASwKXkxsxKoFw_aoSAYeqRf0VQE
```

Per poter utilizzare le API nei nostri esempi è sufficiente indicare l'API key ottenuta all'interno del tag <SCRIPT> della <HEAD> del documento HTML:

```
mappaConKey.html
1 <HTML><HEAD>
2 <LINK HREF="/maps/documentation/javascript/examples/default.css" REL="stylesheet">
3 <SCRIPT TYPE="text/javascript"
4 SRC="https://maps.googleapis.com/maps/api/
js?key=AIzaSyChTchMeTgK-0O-RAdmRVFLBTDMC__5aWk&sensor=false">
5 </SCRIPT>
```

Il parametro sensor indica se l'applicazione usa un sensore per determinare la locazione dell'utente.

Non è più possibile utilizzare le API senza inserire l'API key, modalità che fino allo scorso anno permetteva di effettuare un numero limitato di accessi.

**AREA DIGITALE**
**Come ottenere la API key**

## Usare le API di Google Maps

Incominciamo a vedere come includere una mappa di Google: a tal fine è utile che questa venga collocata all'interno di un <DIV> di HTML, associando un ID a esso (nel nostro esempio con identificatore mappa), come illustrato dal seguente codice:

```
25 <BODY ONLOAD="inizializza();">
26 <DIV ID="mappa" STYLE="position:absolute;left:5px;top:9px;height:250px;width:550px">
   </DIV>
27 </BODY></HTML>
```

Abbiamo inserito una funzione inizializza() che viene eseguita al caricamento della pagina: contiene il codice per visualizzare la nostra mappa con le opzioni che, volta per volta descriveremo, utilizzeranno le due seguenti variabili globali:

- la prima, coordinate, viene inizializzata con la coordinate del punto centrale (coordinate correnti) creando un oggetto di tipo LatLog;
- la seconda, opzioni, è un array associativo necessario per definire alcune proprietà che dovrà avere la nostra mappa:

```
6   <SCRIPT>
7   // definizione coordinate di centratura della mappa
8   var coordinate = new google.maps.LatLng(45.812, 9.0855);
9   // dichiarazione opzioni della mappa
10  var opzioni = {
11    // livello di zoom della mappa mostrata
12    zoom: 10,
13    center: coordinate,
14    // tipo di mappa (strade=ROADMAP, oppure HYBRID, SATELLITE, TERRAIN)
15    mapTypeId: google.maps.MapTypeId.ROADMAP
16  }
```

Questo array deve essere passato come parametro al costruttore dell'oggetto nella funzione inizializza():

```
18  function inizializza(){
19    // creazione dell'oggetto mappa
20    var mappa = new google.maps.Map(document.getElementById('mappa'), opzioni);
21  }
```

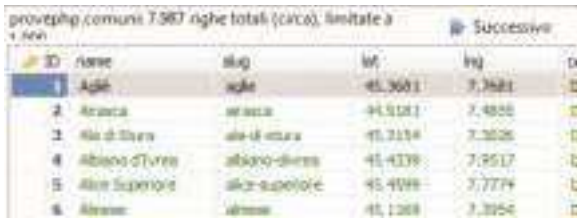
L'esecuzione di questo semplice script ci visualizza la seguente schermata:

## google.maps.LatLng

La classe LatLng rappresenta una coordinata geografica, cioè formata da una longitudine e una latitudine, necessaria per la geolocalizzazione (geocoding).
Il costruttore LatLng() possiede due parametri: latitudine e longitudine

```
7   // definizione coordinate di centratura della mappa
8   var coordinate = new google.maps.LatLng(45.812, 9.0855);
```

Noi utilizzeremo il database localita, presente nella cartella MATERIALI, che contiene la tabella comuni nella quale sono memorizzati i nomi e le geolocalizzazioni di tutti i comuni italiani.




## google.maps.Map

La seguente istruzione rappresenta una singola mappa all'interno di una pagina web:

```
19    // creazione dell'oggetto mappa
20    var mappa = new google.maps.Map(document.getElementById('mappa'), opzioni);
```

Il costruttore possiede due parametri:
**1** un elemento HTML che conterrà la mappa, generalmente un contenitore come per esempio <SPAN> o <DIV>:

```
document.getElementById("mappa")
```

**2** la variabile strutturata opzioni (di tipo object literal), che contiene le proprietà della mappa:

```
// definizione coordinate di centratura della mappa
var coordinate = new google.maps.LatLng(45.812, 9.0855);
// dichiarazione opzioni della mappa
var opzioni = {
  // livello di zoom della mappa mostrata
  zoom: 10,
  // center definisce dove posizionare la mappa con un oggetto della classe LatLng
  center: coordinate
  // mapTypeId definisce il tipo di terreno
  // possibili alternative: strada-ROADMAP, oppure HYBRID, SATELLITE, TERRAIN
  mapTypeId: google.maps.MapTypeId.ROADMAP
}
```

## google.maps.Marker

Questa classe rappresenta un marker che può essere aggiunto alla mappa. Il marker è il segnale che mostra una posizione sulla mappa stessa.
Il costruttore Marker() possiede un solo parametro rappresentato dalle proprietà del marker, racchiuse in oggetto literal.
Il marker di default standard è quello che tutti conosciamo, rosso con la punta che indica la posizione. è anche possibile utilizzare marker personalizzati da definendo una variabile immagine nel modo seguente:

```
var mieImmagine= {                  // nuova immagine per il marker (se vogliamo cambiarla)
    url: 'img/icona.png',                   // file dell'immagine
    size:   new google.maps.Size(32, 32),   // dimensione del marker in pixel
    origin: new google.maps.Point(0, 0),    // origine
    anchor: new google.maps.Point(16, 16)   // punto marker collocato in LatLng
  };
```

Abbiamo diverse possibilità per collocare un marker sulla mappa: vediamo di seguito due metodi alternativi.

**A** creiamo dapprima la mappa e il marker e successivamente lo aggiungiamo sulla mappa mediante la funzione addMarker(), prestando attenzione a rispettare l'ordine nella dichiarazione;

```
function inizializza(){
  map = new google.maps.Map(document.getElementById('map-canvas'), mieOpzioni);
  mioMarker = new google.maps.Marker({
    map: map,                   // la mappa appena creata
    title:"Il castello",        // stringa con descrizione del marker
    icon: mieImmagine,
    position: castello          // posizionamo il marker a queste coordinate
  });
  addMarker(mioMarker, map);    // aggiunge il marker sulla mappa
}
```

**B** posizioniamo direttamente il marker con il metodo di map setMap() (modalità consigliata):

```
  map = new google.maps.Map(document.getElementById('map-canvas'), mieOpzioni);
  mioMarker.setMap(map);      // aggiunge il marker sulla mappa
```

Naturalmente la posizione del marker viene definita tra le variabili iniziali dello script:

```
var napoli   = new google.maps.LatLng(40.8333,14.25);
var castello = new google.maps.LatLng(40.8364,14.2482)
```

Per rimuovere il marker dalla mappa si richiama la funzione setMap(null).

## METTITI ALLA PROVA

- google.maps.Marker

Scrivi un programma che visualizzi su di una mappa due marker di dimensione e tipo diverso.

## google.maps.InfoWindow

La classe InfoWindow rappresenta una finestra di informazioni che può essere aggiunta direttamente sulla mappa oppure su un marker. Il costruttore InfoWindow() possiede un unico parametro: si tratta delle proprietà della InfoWindow, contenute in un object literal.

```
41      // testo html da visualizzare
42      stringa='<br><b>Benvenuti a Como</b><br>Il lago più bello del mondo!';
43      // creazione istanza della InfoWindow
44      infowindow = new google.maps.InfoWindow({
45          content: stringa,          // stringa HTML mostrata nella infowindow
46          position: coordinate       // anche differenti da quelle del marker
47      });
48      // metodo open che mostra la marker con infowindow
49      infowindow.open(map, marker);
50    }
```

Per rendere visibile la InfoWindow abbiamo utilizzato nell'istruzione di riga 49 il metodo open(), a cui passare due parametri: la mappa ed eventualmente il marker a cui ancorarla.

Eseguendo il codice otteniamo:

Il codice completo dello script è riportato nel file infowindow1.html.

# Associare eventi agli oggetti Google Maps

Alcuni oggetti delle Google Maps API sono costruiti in modo tale da rispondere a eventi legati in genere al mouse (per esempio click, mouseover, dblclick ecc.).
La classe google.maps.Map risponde a molti eventi tra i quali citiamo:

| EVENTO | ARGOMENTI | DESCRIZIONE |
|---|---|---|
| click | MouseEvent | Questo evento è scatenato dal click del mouse sulla mappa, ma non dal click su di un marker o su una InfoWindow. |
| dblclick | MouseEvent | Questo evento è scatenato dal doppio click sulla mappa. |
| rightclick | MouseEvent | Questo evento è scatenato dal click con il tasto destro del mouse. |

Per gestire tali eventi occorre registrare un ascoltatore (listener), mediante il metodo addListener() della classe google.maps.event.

```
21    // aggiunta di un ascoltatore di evento per il click
22    google.maps.event.addListener(marker,'click', function(){
23      infowindow.open(map,marker);
24    });
```

Il metodo addListener() possiede 3 parametri:
- un oggetto (per esempio di classe Map, oppure Marker);
- un evento da "catturare" (per esempio 'click');
- una funzione (anonima) da invocare quando l'evento viene catturato (per esempio per mostrare una InfoWindow).

La documentazione completa delle classi di Google Maps, dei relativi metodi e proprietà è consultabile all'indirizzo: https://developers.google.com/maps/documentation/javascript/reference.

Vediamo un esempio dove, cliccando sulla mappa, vengono visualizzate le coordinate del punto selezionato.

## ESEMPIO

In questo esempio vogliamo mostrare una mappa di Google all'interno di un <DIV> di nome identificativo id="mappa" (riga 47). A ogni click sulla mappa viene mostrata la coordinata all'interno di due caselle di testo chiamate lat e lng (righe 45÷46).

```
45  <TABLE><TR><TD>Latitudine<TD><INPUT TYPE="text" ID="lat" SIZE=15>
46  <TR><TD>Longitudine<TD><INPUT TYPE="text" ID="lng" SIZE=15></TABLE>
47  <DIV ID="mappa" STYLE="position:absolute;left:10px;top:70px;height:270px;width:500px"></DIV>
```

Nello script dopo la definizione della API key inizia il codice JavaScript con la dichiarazione una variabile globale di nome map che rappresenta l'oggetto della mappa e opzioni che è un array (oggetto literal) che contiene le proprietà necessarie per definire la mappa.

È utile dichiararla come variabile globale per renderla visibile a tutte le funzioni.

La funzione inizializza() serve per inizializzare la mappa di Google ed è associata all'evento Load del tag <BODY> di HTML.

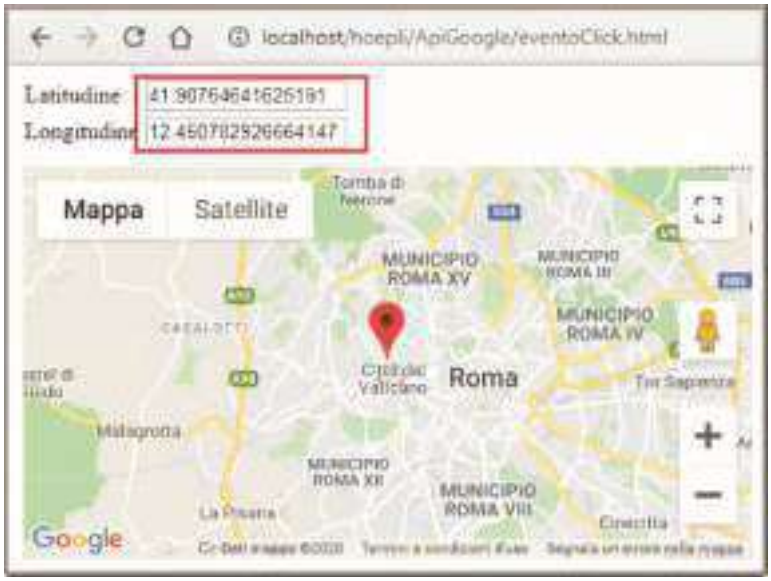
```
16  // funzione richiamata all'apertura del body
17  function inizializza(){
18    // creazione dell'oggetto mappa vero e proprio
19    map = new google.maps.Map(document.getElementById('mappa'),opzioni);
20    // aggiunta di un ascoltatore di evento per il click
21    google.maps.event.addListener(map,'click', function(event){
22      // posizione il marker nel punto cliccato
23      var myLatLng = event.latLng;
24      placeMarkerAndPanTo(myLatLng, map);
25      // scrive nella casella di testo la latitudine utilizzando il metodo lat()
26      document.getElementById('lat').value = myLatLng.lat();
27      // scrive nella casella di testo la longitudine utilizzando il metodo lng()
28      document.getElementById('lng').value = myLatLng.lng();
29    });
```

L'istanza dell'oggetto google.maps.Map possiede due parametri:
- dove collocare la mappa (elemento <DIV> di nome 'mappa');
- le opzioni contenute nell'array che descrivono la tipologia di mappa da visualizzare.

Aggiungiamo l'ascoltatore alla mappa per l'evento 'click' e dichiariamo una funzione anonima che riceve come parametro l'argomento di nome evento di classe MouseEvent (riga 21): all'interno della funzione andiamo a leggere la latitudine e la longitudine del punto in cui è avvenuto il click sulla mappa attraverso i metodi lat() e lng() dell'oggetto myLatLng (righe 26÷28). Aggiungiamo, infine, la funzione placeMarkerAndPanTo() che, richiamata automaticamente al click del mouse, posiziona e centra la mappa nella nuova locazione e vi colloca un marker:

```
30  function placeMarkerAndPanTo(latLng, map) {
31    var marker = new google.maps.Marker({
32      position: latLng,
33      map: map
34    });
35    map.panTo(latLng);
36  };
```

Nell'esecuzione del codice possiamo notare che vengono inserite le coordinate nelle caselle di testo relative al punto dove è avvenuto il click sulla mappa, nel nostro caso sulla Città del Vaticano.




Il codice completo è riportato nel file eventoClick.html.

---

In questo esempio vogliamo inserire un marker all'interno della mappa per poi associarvi un evento che mostri a video una finestra di informazioni (di classe InfoWindow).

## ESEMPIO

Lo script è formato dalla funzione inizializza() che si occupa di mostrare la mappa, il marker e ascoltare l'evento.
Dopo l'intestazione e la API key vengono definite le variabili globali tra cui le coordinate da utilizzare per centrare la mappa (riga 10): come al solito viene creata la variabile literal opzioni Mappa che contiene le opzioni necessarie per la classe Map (righe 11÷15) e un'altra variabile literal image necessaria per la creazione del marker (righe 16÷21).

```
7   var marker;        // dichiarazione oggetto map di tipo globale
8   var map;           // dichiarazione variabile infowindow
9   var infowindow;    // funzione richiamata all'apertura del body
10  var coordinate = new google.maps.LatLng(41.1656,13.7242);
11  var opzioniMappa = {
12     zoom: 10,
13     center: coordinate,
14     mapTypeId: google.maps.MapTypeId.ROADMAP
15  }
16  var image = {
17    url: 'img/welcoR.png',                //simbolo del marker
18    size: new google.maps.Size(32, 32),
19    origin: new google.maps.Point(0, 0),
20    anchor: new google.maps.Point(0, 32)
21  };
```

La parte restante del codice istanzia un oggetto map di classe google.maps.Map (riga 25) e gli associa l'elemento HTML 'mappa' quindi crea un oggetto Marker, in questo caso chiamato (con molta fantasia) marker e contestualmente ne inizializza gli attributi:

- le **coordinate** (proprietà position);
- la **mappa** in cui collocarlo (proprietà map);
- l'**icona** da mostrare (proprietà icon);
- il **nome** che appare passando il mouse sopra di esso (proprietà title) (righe 27÷31).

```
23  function inizializza() {
24    // creazione dell'oggetto mappa vero e proprio
25    map = new google.maps.Map(document.getElementById('mappa'),opzioniMappa);
26    // creazione  del marker
27    marker = new google.maps.Marker({
28      position: coordinate,
29      map: map,
30      icon: image,
31      title: 'Fermo'
32    });
33    // aggiunta ascoltatore sul marker
34    google.maps.event.addListener(marker, 'click', function() {
35      // definizione della stringa HTML da visualizzare
36      var stringa="<br><b>Benvenuti a "+marker.getTitle()+"</b><br>";
37      stringa+="<img src='img/fermo.jpg'><br>Dista circa 5 km dal mare Adriatico, in una zona ad
        alta concentrazione demografica e inclusa in un'area calzaturiera<BR>";
38      // istanza dell'InfoWindow
39      var infowindow = new google.maps.InfoWindow({
40        content: stringa        // stringa HTML mostrata dalla infowindow
41      });
42      // metodo open che mostra la infowindow sul marker
43      infowindow.open(map, marker);
44    });
45  }
46  </SCRIPT>
```

La parte interessante è sicuramente legata all'utilizzo del metodo ascoltatore addListener() della classe google.maps.event (riga 34), che riceve tre parametri:

- l'oggetto marker sul quale intercettare gli eventi;
- l'evento da intercettare (in questo caso il click);
- la funzione anonima di risposta.

La funzione anonima si occupa di mostrare a video una InfoWindow: a tal fine dapprima memorizza in una stringa (stringa) il testo HTML da mostrare successivamente nella InfoWindow (righe 36÷38). Successivamente avviene l'istanza dell'oggetto InfoWindow di classe InfoWindow (riga 39) dove nel costruttore inseriamo un unico argomento (content) che è la stringa da mostrare (riga 40).
La riga 44 conclude la funzione anonima legata all'evento sul marker: mostra la finestra invocando il metodo open(map, marker) sull'oggetto InfoWindow.
L'ultima parte riguarda la sezione HTML (righe 49÷52). Il tag <BODY> richiama subito la funzione inizializza(), quindi mostra un riquadro <DIV> (id='mappa') al quale è associato uno stile che lo identifica in una parte dello schermo.

```
49  <TITLE>Eventi sulla mappa con info</TITLE>
50  <BODY onload='inizializza()'>
51  <DIV ID="mappa" STYLE="position:absolute;left:10px;top:10px;height:350px;width:400px"></div>
52  </BODY></HTML>
```

L'esecuzione mostra la seguente mappa e cliccando sul marker appare una InfoWindow:

Il codice completo è riportato nel file eventoClickInfo.html.

# Le mappe con LeafletJS

Un'alternativa alle Google Map per pubblicare delle mappe all'interno di una pagina Web sono le librerie LeafletJS (o semplicemente Leaflet), che negli ultimi anni sono uno degli strumenti più semplici e più diffusi. Anche questa libreria, scritta in JavaScript e completamente open source, nasce per realizzare mappe interattive per "applicazioni mobile", ma viene subito utilizzata anche per il Web mapping grazie alle sue caratteristiche di leggerezza, versatilità e facilità di utilizzo.

Sul suo sito ufficiale, raggiungibile all'indirizzo leafletjs.com, è possibile trovare tutorial, esempi pratici e una ricca documentazione.

## Primo utilizzo di LeafletJS

Anche l'utilizzo di questa libreria richiede le conoscenze elementari di HTML, JavaScript e fogli di stile (CSS) perché la pagina Web che deve contenere la mappa deve essere editata a mano, con un qualunque IDE adatto per realizzare pagine Web e/o programmare in JavaScript oppure direttamente con un normale text editor come Notepad.
Vediamo ora come procedere.
Come primo passo è necessario scaricare le librerie sul nostro computer direttamente dalla pagina leafletjs.com.

A questo punto scriviamo la pagina Web che conterrà la mappa indicando nell'<head> che utilizziamo:
- la libreria JavaScript di Leaflet (riga 5);
- il foglio di stile CSS di Leaflet (riga 6).

```
2  <head>
3  <title>Primo esempio con la libreria Leaflet</title>
4  <!-- nel caso si con librerie scaricate -->
5  <link rel="stylesheet" href="leaflet/leaflet.css">
6  <script src="leaflet/leaflet.js" ></script>
```

Possiamo anche richiamare (e utilizzare) le librerie in remoto sul server CDN di Leaflet (cdn.leafletjs.com) e, in tal caso, metteremo l'indirizzo completo Http:

```
7  <!-- nel caso di librerie utilizzate on line  -->
8  <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet/v1.6.0/leaflet.css" >
9  <script src="http://cdn.leafletjs.com/leaflet/v1.6.0/leaflet.js"></script>
```

La mappa deve essere posta all'interno del <body>, in uno specifico contenitore: posizioniamo un  e gli assegniamo come id="miaMappa". Inoltre, usando il CSS, nella sezione <head> definiamo almeno le sue dimensioni: larghezza e altezza o almeno una delle due.

```
13  <body>
14  <h2>La mia prima mappa con Leaflet (zoom 14)</h2>
15  <!-- contenitore della mappa -->
16  <div id="miaMappa"> </div>
```

Osserviamo che il blocco  è vuoto: al suo interno non abbiamo scritto nulla.

Aggiungiamo un semplice script necessario a caricare e inizializzare la mappa che vogliamo visualizzare e che, nella sua forma più semplice (con opzioni elementari), sarà:

```
17  <script>
18    //definiamo la mappa: posizione e dimensione
19    var map = L.map('miaMappa').setView([45.8105, 9.0085], 14); // coordinate di Como
20    //carica e inizializza la mappa base
21    var OSM_layer =L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png',
22      {attribution: '&copy; <a href="http://osm.org/copyright"&gt;OpenStreetMap</a>'}).addTo(map);
23  </script>
```

Effettuiamo due visualizzazioni modificando solamente il valore dello zoom, per apprezzare le qualità grafiche di questa libreria:

Possiamo cambiare il tipo di mappa visualizzata: alla pagina https://leaflet-extras.github.io/leaflet-providers/preview/ sono elencati e visualizzabili in preview tutti i **map provider** disponibili per Leaflet (alcuni di essi necessitano di API key, anche se quelli tradizionalmente usati sono OSM e Mapbox).

Inoltre Leaflet non contiene nemmeno una riga di codice specifica di un singolo fornitore, quindi il programmatore è libero di utilizzare anche altri fornitori (consigliamo comunque Mapbox, che sembra essere il migliore tra quelli gratuiti).

## Mappe a più layer

Leaflet permette di gestire agevolmente mappe a più strati (layer sovrapposti). Sfruttando la trasparenza di alcune zone dei diversi layer si "intravvede" il layer sottostante ottenendo come effetto finale la sovrapposizione di più mappe.
La collocazione degli strati avviene semplicemente scrivendo in sequenza l'istruzione:

```
L.tileLayer( … ).addTo(map)
```

Gli stati sono "impilati" rispettando l'ordine sequenziale delle istruzioni.

Riprendiamo l'esempio precedente e aggiungiamo quattro livelli, tre opachi con diverse tipologie di mappe e uno trasparente con i soli nomi/etichette

```
var miaMappa = L.map('miaMappa').setView([45.8009, 9.0885], 13); // coordinate di Como
// mappa primo strato
var OSM_layer =L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {attribution: '&copy; <a
  href="http://osm.org/copyright">OpenStreetMap</a>'
});
// mappa secondo strato
 var Stamen_Terrain = L.tileLayer('https://stamen-tiles-{s}.a.ssl.fastly.net/
   terrain/{z}/{x}/{y}{r}.{ext}', { attribution: 'Map tiles by <a href="http://stamen.com">Stamen
   Design</a>, <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
   &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
   subdomains: 'abcd', minZoom: 0, maxZoom: 18, ext: 'png'
});
// mappa terzo strato
var Stamen_Watercolor = L.tileLayer('https://stamen-tiles-{s}.a.ssl.fastly.net/
  watercolor/{z}/{x}/{y}.{ext}', { attribution: 'Map tiles by <a href="http://stamen.com">Stamen
  Design</a>, <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data
  &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
  subdomains: 'abcd', minZoom: 1, maxZoom: 16, ext: 'jpg'
});
// layer trasparente di etichette
var CartoDB_label = L.tileLayer('http://{s}.basemaps.cartocdn.com/dark_only_labels/{z}/{x}/{y}.png', {
   attribution: '&copy; <a href="http://cartodb.com/attributions">CartoDB</a>'
});
```

e aggiungiamo un menu che abilita/disabilita i singoli livelli:

```
// array delle mappe
var sceltaMappe = {
 "OSM": OSM_layer,
 "Terrain": Stamen_Terrain,
 "Watercolor": Stamen_Watercolor
  };
// array dei layer trasparenti
var sceltaNomi = {
 "Nomi": CartoDB_label
};
```

Possiamo ora caricare la mappa e aggiungere l'oggetto con i selettori di livello

```
// carico la mappa base di default
OSM_layer.addTo(miaMappa);
// attivo il selettore di layers
L.control.layers(sceltaMappe, sceltaNomi).addTo(miaMappa);
```

e alcuni marker, anche personalizzati:

```
// definizione immagine/icona per il marker
var uomo_red = L.icon({
  iconUrl: 'img/omino01.png', // posizione
  iconSize: [32, 32],         // dimensioni
});
var spillo = L.icon({
  iconUrl: 'img/pin2.png',    // posizione
  iconSize: [32, 32],         // dimensioni
});
// definizione e aggiunta dei markers
var marker1 = L.marker([45.8200, 9.0960]).addTo(miaMappa);
var marker2 = L.marker([45.7820, 9.0950], {icon: uomo_red}).addTo(miaMappa);
var marker3 = L.marker([45.8133, 9.0800], {icon: spillo}).addTo(miaMappa);
```

Avviamo l'esecuzione e selezioniamo le diverse opzioni ottenendo la situazione riportata nella seguente figura:

Osserviamo come, all'avvio della pagina, il selettore viene già settato sulla prima opzione del gruppo di radio button.

Il codice completo è riportato nel file lealetf4level.html.

# VERIFICA... le competenze

## ESERCIZI

**Crea gli script che risolvano i problemi proposti.**

**1** Mostra, all'interno di una pagina Web, due <DIV>: entrambi mostrano una mappa, a ogni click sul primo deve comparire nella seconda mappa un marker nella stessa posizione.

**2** Mostra un form e una mappa di Google. Nel form appare una casella a discesa (<SELECT>) che mostra all'utente l'elenco delle regioni. Quando l'utente seleziona una regione mostra, sulla mappa, un elenco di marker su tutti i capoluoghi di provincia di quella regione prelevandoli dal database localita.

**3** Crea una tabella, prendendo spunto dal database localita, che contenga le coordinate delle abitazioni dei tuoi compagni di classe. Mostra sulla mappa un marker per ciascuna abitazione della tua classe, facendo click su di essi deve apparire un InfoWindow con la foto e alcuni dati dell'alunno.

**4** Mostra un form che consenta di effettuare la ricerca di una città dal database localita. Quando la località viene trovata deve apparire in un <DIV> che contiene una mappa di Google la visione della Street View.

**5** Apri il file eventiClickInfo.html:
- **a** modifica il codice dell'esempio in modo che possa mostrare i marker per tutti i capoluoghi della tua regione; a ogni click sul marker devono apparire le coordinate in cui si trova e alcune immagini inserite all'interno di una InfoWindow;
- **b** prova poi a verificare le coordinate di casa tua e aggiungi un marker che mostri la sua foto e un messaggio all'interno di un <DIV>diverso da quello della mappa.

**6** Mostra un form che consenta all'utente di cercare due località dal database localita. Nella mappa posta in un <DIV> devi mostrare il percorso stradale che collega i due luoghi.

**7** Modifica l'esempio precedente cercando il percorso da effettuare a piedi.

**8** Utilizzando Leaflet realizza uno script che visualizza una mappa fisica del territorio dove, eventualmente, è possibile aggiungere i nomi delle località e il confine dei singoli comuni con un layer trasparente.

**9** Utilizzando Leaflet realizza uno script che visualizza una mappa fisica del territorio dove, eventualmente, è possibile aggiungere i percorsi autostradali e le principali strade interurbane.

**10** Utilizzando Leaflet realizza uno script che visualizza una mappa fisica del territorio dove, eventualmente, è possibile aggiungere i percorsi ferroviari con i nomi delle stazioni.

# ESERCIZI IN LABORATORIO 1

## STRUMENTI PER LO SVILUPPO E IL TESTING DI WEB SERVICE: FRAMEWORK E SDK (SOFTWARE DEVELOPMENT KIT)

### La realizzazione di un Web Service

La realizzazione di un Web Service è abbastanza articolata e può essere così "scomposta":
**a** codifica del servizio in linguaggio di programmazione;
**b** mappatura della azioni CRUD sui metodi Http e definizione degli URI che identificheranno le nostre risorse;
**c** definizione dei file di configurazione per il deploy del servizio;
**d** testing e debugging.

La codifica dei servizi in linguaggio di programmazione e la loro mappatura deve naturalmente tener conto dell'interazione tra client e server, e quindi gestire i formati di scambio dei dati tra pagine HTML in formato JSON: il principio dell'uso esplicito dei metodi Http ci indica di sfruttare i metodi (o verbi) predefiniti di questo protocollo, e cioè GET, POST, PUT e DELETE.
Per quanto riguarda le operazioni testing e debugging, è essenziale intercettare le richieste e le risposte scambiate tra client e Web Service per comprendere la causa di malfunzionamenti, ma queste operazioni non sempre si effettuano agevolmente.
Fortunatamente ai programmatori vengono in aiuto appositi pacchetti software, i Software Development Kit (SDK), che rappresentano uno strumento indispensabile nello sviluppo e nel rilascio di software per piattaforme specifiche.

Un SDK è composto da una suite di strumenti utilizzabili per costruire in modo più semplice e immediato, programmi, applicazioni e servizi per le diverse piattaforme operative e per le diverse tipologie di hardware, dai computer agli smartphone.

Gli SDK possono essere molto diversi tra loro sia dal punto di vista delle dimensioni sia delle tecnologie utilizzate e possono essere composti da:
- API;
- documentazione;
- editor;
- librerie;
- ambienti di runtime e di sviluppo;
- compilatori;
- debugger;
- driver;
- protocolli di rete;
- esempi/progetti di test.

Non tutti gli SDK hanno tutti questi componenti, ma sicuramente in tutti sono presenti almeno:
- un compilatore utile a tradurre il codice sorgente in un eseguibile;
- un insieme di API affinché un modulo o un software possa interagire con un altro sviluppato per un'altra piattaforma;
- la documentazione sul linguaggio di programmazione per cui l'SDK stesso è stato sviluppato e le informazioni sulle licenze da utilizzare per distribuire programmi creati con l'SDK.

Gli SDK per la programmazione di Web Service possono utilizzare WSDL e SOAP *per costruire automaticamente* degli oggetti sia lato del servizio sia lato client in modo tale che il programmatore possa codificare la richiesta di servizi usando il linguaggio di programmazione preferito.

# 1 ESERCIZI IN LABORATORIO

## ESEMPIO

Per il linguaggio Java sono disponibili add-on che offrono il supporto Java per lo sviluppo di Web Service:

- JAX-RPC (Java API for XML-RPC): è di fatto Java RMI (Remote Method Invocation) su SOAP e fornisce un'interfaccia remota per lo scambio di messaggi SOAP in stile RPC;
- SAAJ (SOAP API with Attachments for Java): è un'API che modella la struttura di un messaggio SOAP e implementa alcune funzionalità del protocollo per la comunicazione;
- JAXM (Java API for XML Messaging): è simile a JMS (Java Message Service) e fornisce un'infrastruttura di messaging per spedire e ricevere messaggi SOAP;
- JAX-WS (Java API for XML Web Services): è una tecnologia per realizzare Web Service e client in Java che comunicano utilizzando messaggi SOAP-XML.

Per la realizzazione di applicazioni Web, oltre agli SDK, i programmatori hanno a disposizione anche i framework, ovvero "impalcature di codice pronte all'uso", strumenti ancora più completi che hanno al loro interno componenti creati e pronti per essere sfruttati nelle nuove applicazioni in modo che il programmatore non dovrà preoccuparsi di implementare.

Possiamo vedere un framework come "*uno strumento di sviluppo che riunisce in sé un applicativo o un set di modelli che facilitano il processo di sviluppo*", perché permette di utilizzare componenti e/o funzionalità già sviluppate, personalizzabili e integrabili nel nuovo progetto.

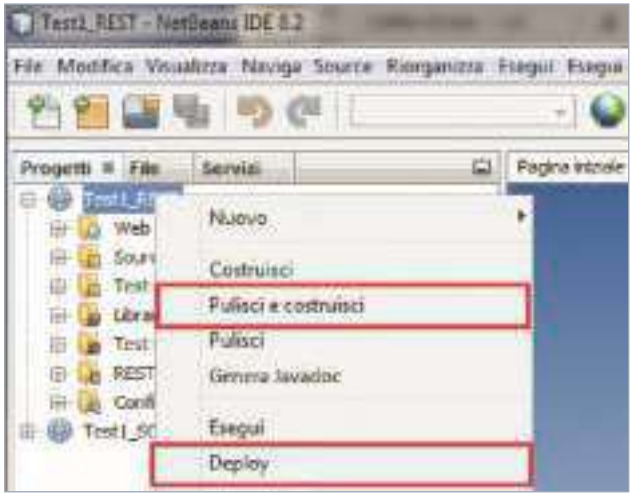
## Utilizzo di NetBeans

Nella nostra trattazione per realizzare i Web Service utilizzeremo NetBeans in quanto risulta essere di semplice utilizzo, "più leggero" rispetto a prodotti equivalenti (come Eclipse) e, quindi, utilizzabile anche su personal computer non eccessivamente performanti (generalmente presenti nei nostri laboratori scolastici!)
Riportiamo la definizione di NetBeans direttamente dal sito ufficiale all'indirizzo https://netbeans.apache.org/:

Apache NetBeans, Fits the Pieces Together: Development Environment, Tooling Platform and Application Framework.

Al momento dell'esecuzione NetBeans genera automaticamente le classi di supporto (artefatti), compila tutte le classi, dispiega il servizio e attiva l'application server: dietro le quinte il Web Service (endpoint) viene generato dal tool wsgen.
Gli artefatti vengono generati automaticamente dal tool wsimport a partire dal WSDL, creando:

- Service Endpoint Interface (SEI);
- Service;
- Exception class a partire da wsdl:fault;
- oggetti generati a partire dagli schemi XML (JAXB...).




Il nostro compito si riduce a quello di:

- avviare NetBeans e creare un nuovo progetto Web;
- creare un nuovo Web Service;
- eseguire Pulisci e costruisci e Deploy;
- eseguire e provare il servizio (Test).

# ESERCIZI IN LABORATORIO 1

## La pubblicazione nel Web di un Web Service

Per offrire un Web Service è inoltre necessario avere un apposito componente nel Web server: per esempio, per Apache abbiamo diverse possibilità a seconda del linguaggio di sviluppo:

- Apache SOAP, Apache Axis (Java);
- SOAP::Lite (Perl);
- PHPSOAP (PHP);
- WhiteMesa SOAP (C++);
- SOAP for ADA;
- Smalltalk Web Services.

### Java

**A** Axis: Apache eXtensible Interaction System (oggi nella versione Axis2) è un SOAP Engine open source di Apache Software Foundation per creare, pubblicare e consumare Web Service in Java; Axis2 fornisce:

**1** strumenti da riga di comando per creare e usare Web Service;
**2** un server SOAP stand-alone;
**3** una Web Application che ospita i Web Service all'interno di un servlet container.

Il sito ufficiale di Axis2 è: http://ws.apache.org/axis2/.

**B** Apache Tomcat è un Web Server open source che implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java. La sua distribuzione standard include anche le funzionalità di Web Server tradizionale, che corrispondono al prodotto Apache.
L'implementazione JAX-WS è un progetto open source e fa parte del progetto GlassFish, un Application Server Java EE ed è integrato nella distribuzione "Metro".
Nelle recenti versioni di Tomcat sono incorporate alcune potenzialità di Java EE e, quindi, per semplici applicazioni Web o applicazioni che utilizzano framework come Spring che non richiedono un server Java EE completo, è sufficiente la versione standard, senza l'aggiunta di librerie.

### PHP

PHP prevede già un'estensione SOAP standard che però necessita della configurazione del modulo PHP del server Apache sul quale gira, cosa che non sempre è fattibile, soprattutto se si utilizzano provider come Aruba, Registrer ecc.
La soluzione è quindi quella di utilizzare una libreria esterna, NuSOAP, che consente di creare e interrogare Web Service velocemente e senza dover "modificare troppo" le configurazioni di Apache o sul file .ini di PHP.
La libreria NuSOAP è una riscrittura della SOAPx4: è composta da un insieme di classi PHP che consente agli sviluppatori di creare e utilizzare servizi Web basati su SOAP 1.1, WSDL 1.1 e Http 1.0 / 1.1, richiamandola semplicemente all'interno dei programmi:

```
require_once('lib/nusoap.php');
```

Il sito ufficiale dove scaricare NuSOAP è https://sourceforge.net/projects/nusoap/.
Nessun accorgimento, invece, per la realizzazione servizi REST: basta rispettare le regole di visibilità.

## Il plug-in Wizdler per il collaudo dei servizi SOAP

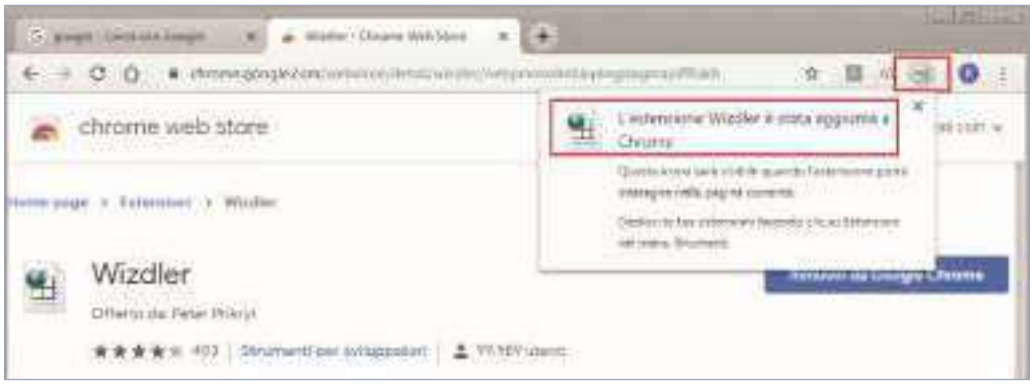
Sono disponibili diverse alternative per collaudare i servizi SOAP e in questa esercitazione utilizzeremo un semplice strumento offerto come plug-in dalla maggio parte dei browser: Wizdler.

Wizdler ci permette di analizzare i file WSDL e generare messaggi SOAP semplicemente conoscendo l'URI del file che descrive il servizio.

# 1 ESERCIZI IN LABORATORIO

Come prima operazione è necessario scaricare il plug-in e installarlo: noi lo faremo per Google Chrome accedendo direttamente all'indirizzo https://chrome.google.com/webstore e digitandone il nome nella casella di ricerca. L'installazione è automatica e immediata: basta cliccare sul pulsante Aggiungi:

e viene aggiunto un pulsante sulla barra delle applicazioni che ci permetterà di visualizzare l'elenco dei servizi disponibili dai vari WS:




Possiamo ora collaudare le operazioni/i metodi messi a disposizione dai Web Service SOAP.

## Un primo esempio

Come primo esempio testiamo le operazioni messe a disposizione dal servizio calculator che è reperibile all'indirizzo www.dneonline.com; digitiamo nella casella degli indirizzi il seguente URI:

```
http://www.dneonline.com/calculator.asmx?wsdl
```

Ci viene mostrata nel browser la pagina XML del documento XSDL che descrive il formato dei comandi.

# ESERCIZI IN LABORATORIO 1

Tra le operazioni elencate scegliamo la moltiplicazione: quindi clicchiamo su Multiply e inseriamo i due operandi nelle due caselle int:

Cliccando sul pulsante Go otteniamo il file XML che contiene il risultato:

## Un secondo esempio

Come secondo esempio interroghiamo un database disponibile con molteplici servizi, che contiene, tra l'altro, le informazioni su tutte le nazioni del mondo, raggiungibile all'indirizzo:

```
http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL
```

Cliccando sull'icona apposita possiamo visualizzare l'elenco delle operazioni disponibili:

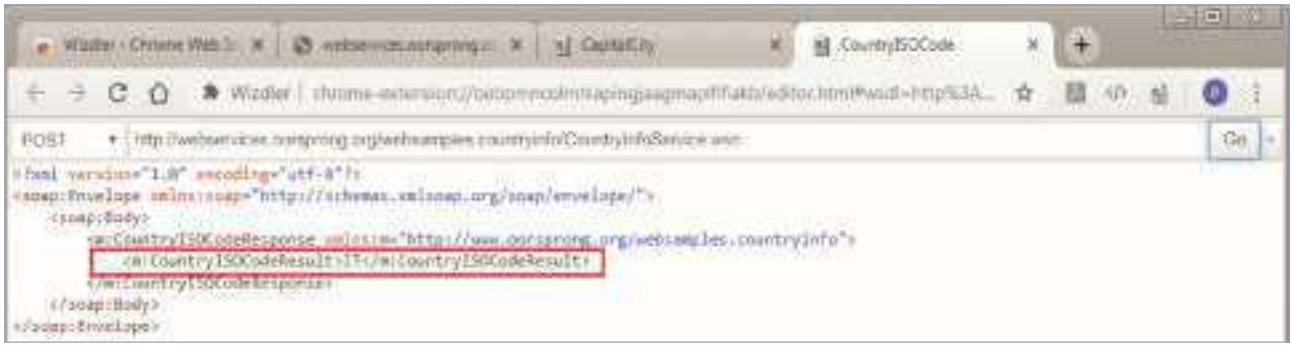
# 1 ESERCIZI IN LABORATORIO

Per esempio, chiediamo di indicarci la capitale degli Stati Uniti inserendo il loro codice internazionale ISO:

Il risultato è naturalmente Washington:

Facciamo ora l'operazione inversa: inseriamo una nazione, per esempio Italy, e mediante l'apposito comando chiediamo di indicarci il suo codice ISO:

Come risposta ci viene indicato "IT":



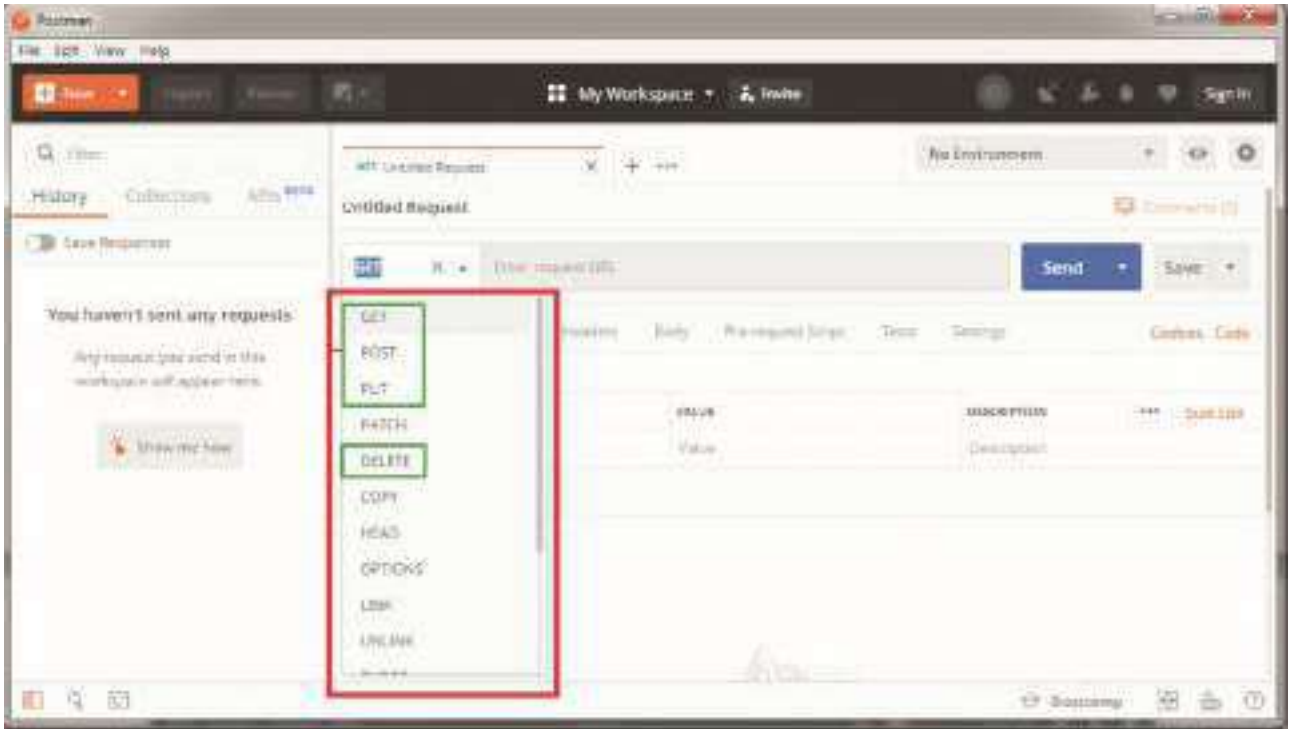
# ESERCIZI IN LABORATORIO 1

## Il programma Postman per il collaudo di servizi REST

Talvolta, per testare il proprio Web Service, realizzare un client di prova è più laborioso della realizzazione del servizio stesso e non sempre efficace: sono disponibili diverse alternative, tra cui veri e propri prodotti creati per questo scopo. Uno di questi è Postman, che dà la possibilità di lanciare richieste Http, selezionando il metodo e aggiungendo l'eventuale body, e di esaminare le risposte ottenute: il programma Postman è scaricabile gratuitamente dall'indirizzo https://www.getpostman.com/downloads dove sono disponibili le versioni per i tre Sistemi Operativi principali (Mac, Windows e Linux) dato che Postman è multipiattaforma.

> Per il testing le varie richieste possono essere raccolte sotto forma di collezioni: tramite Postman è possibile effettuare delle chiamate API senza dover mettere mano al codice dell'applicazione, consentendo di effettuare le chiamate tramite questo plug-in che fornisce un'utile interfaccia grafica.

Le richieste possono essere effettuate sia verso un server locale, sia verso un server online, impostando tutti i dati di una tipica chiamata API, dagli header al body.
Com'è possibile vedere dall'immagine riportata di seguito, tramite l'interfaccia grafica possiamo selezionare facilmente il tipo di chiamata da effettuate (POST, GET, PUT, DELETE ecc.), impostare l'URL, effettuare la chiamata e inviarla.




> Come si può notare, nell'immagine ci sono i metodi classici GET, POST, PUT e DELETE ma ne troviamo tantissimi altri che, però, noi non utilizzeremo.

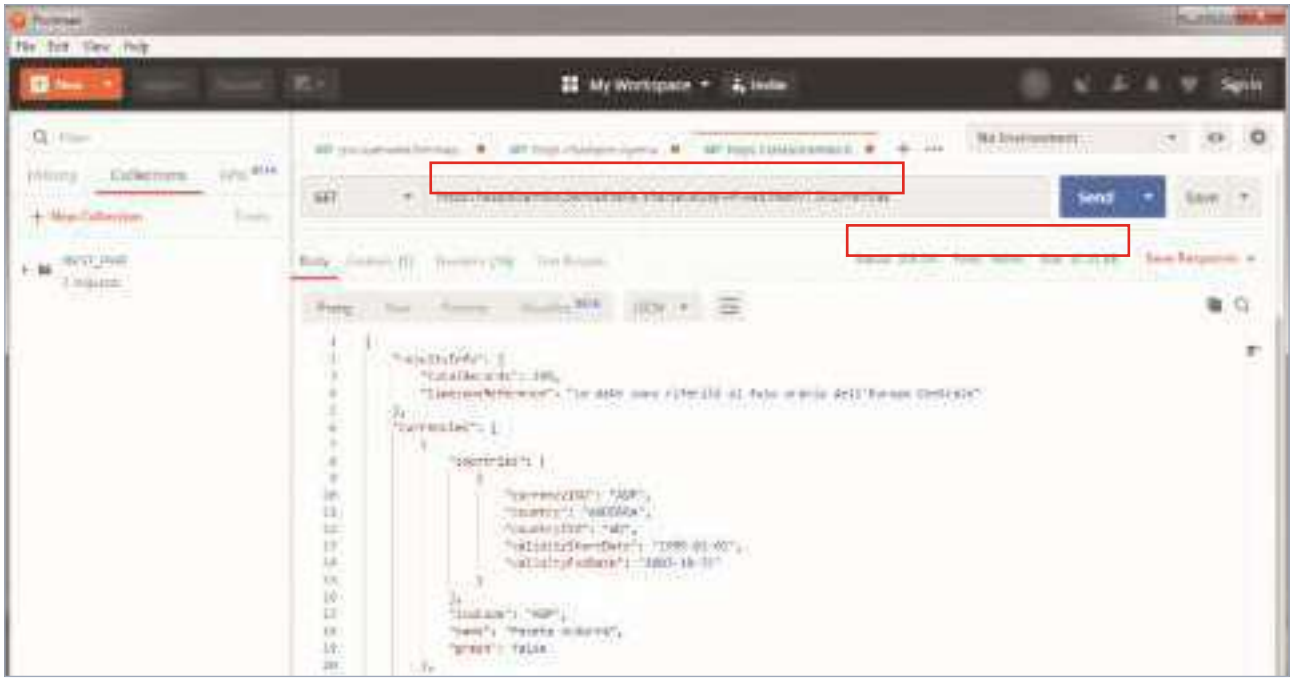
Postman, inoltre, mette a disposizione una serie di tab per poter impostare i dati per l'autorizzazione della chiamata, gli header e il body della chiamata API.
Per il nostro "limitato" utilizzo non è necessario conoscere le complete funzionalità del prodotto, che sono veramente molte: chi le volesse approfondirne può trovare la completa documentazione all'indirizzo https://learning.getpostman.com.

# 1 ESERCIZI IN LABORATORIO

## ESEMPIO

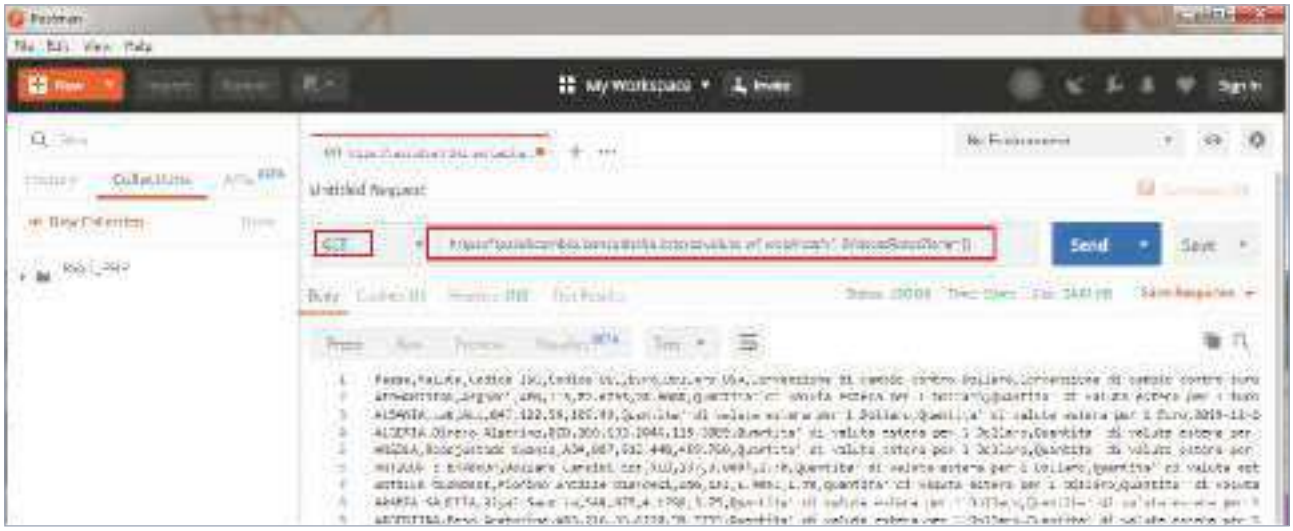
La Banca d'Italia mette a disposizione molteplici servizi, come quello che ci mostra le valute di 208 nazioni, tramite il seguente URI:

```
https://tassidicambio.bancaditalia.it/terzevalute-wf-web/rest/v1.0/currencies
```



Oppure per avere il valore dell'ultimo cambio effettuato:

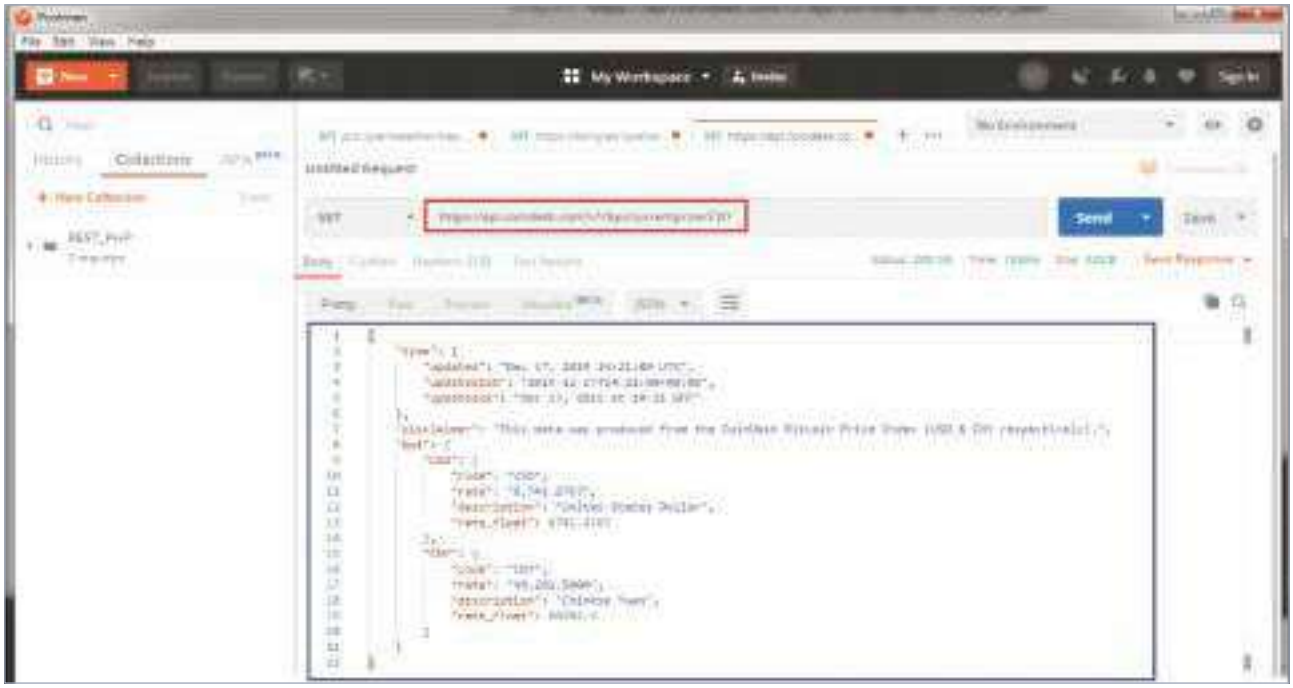
```
https://tassidicambio.bancaditalia.it/terzevalute-wf-web/rest/v1.0/latestRates?lang={}
```



CoinDesk fornisce una semplice API per rendere i dati dell'indice dei prezzi dei bitcoin (BPI) disponibili gratuitamente purché ogni pagina o app che li utilizza includa il link "Powered by CoinDesk", che rimanda alla pagina dove sono presentate le loro proposte commerciali.

# ESERCIZI IN LABORATORIO 1

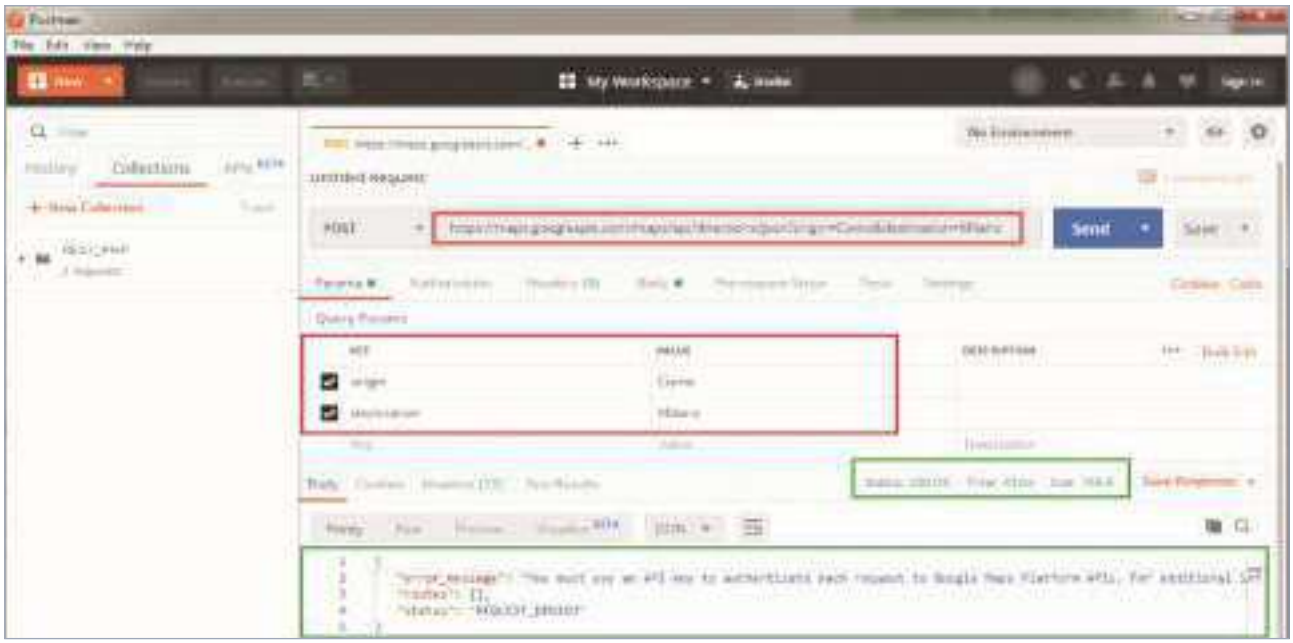
I dati di CoinDesk sono resi disponibili tramite una serie di risorse Http e i dati vengono restituiti in formato JSON: un esempio è riportato di seguito.



Alcuni fornitori offrono i servizi tramite API a pagamento: il sito riportato nell'esempio che segue mette a disposizione molteplici servizi integrabili nelle proprie applicazioni, ma accessibili solo tramite una API key personale che si ottiene versando canone annuale oppure a ogni singola transazione.

## ESEMPIO

Un primo esempio sono le API di Google, che richiedono una API key.



# 1 ESERCIZI IN LABORATORIO

Anche le API messe a disposizione da IBAN.COM per dati finanziari richiedono una API key.

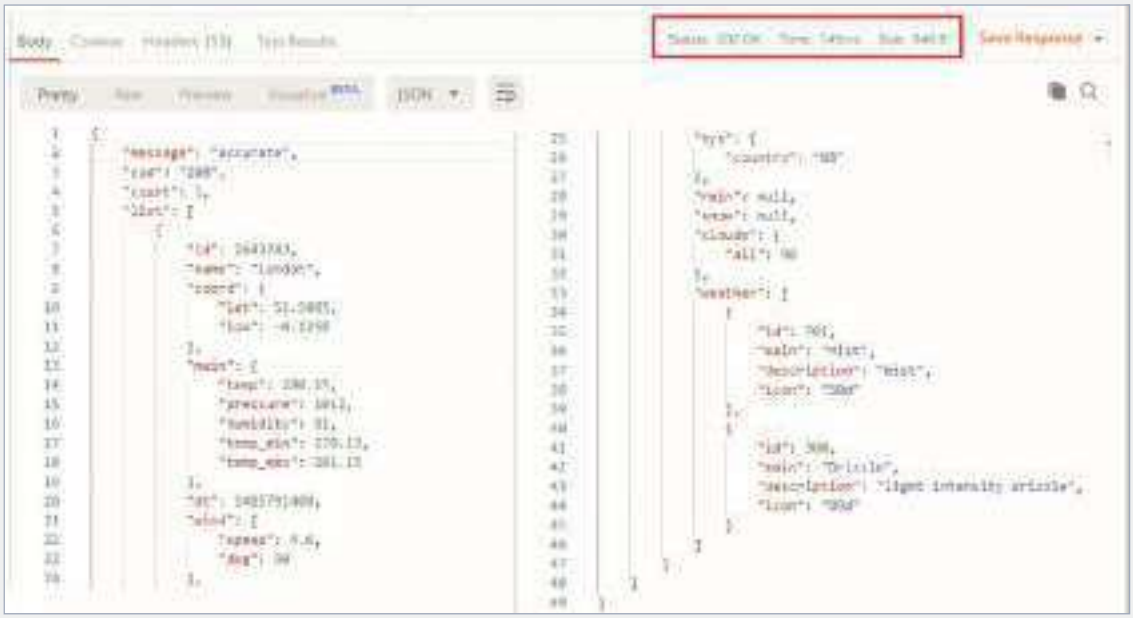
```
https://api.iban.com/clients/api/currency/convert/?from=USD&to=EUR&amount=100&api_key=miakey
```

## METTITI ALLA PROVA

**1 Condizioni meteo**

Il sito https://openweathermap.org/api mette a disposizione un ricco numero di servizi gratuiti dopo una registrazione gratuita all'indirizzo https://home.openweathermap.org/users/sign_up, offrendo però un numero limitato di richieste giornaliere.

Di seguito riportiamo l'esito di una richiesta effettuata sui dati meteorologici di Londra:



**a** Ricerca i dati della tua città e scaricali in un file JSON; quindi scrivi una pagina HTML che esegua la richiesta direttamente e visualizza tali dati sotto forma di tabella.

**b** Modifica la tua applicazione utilizzando le API messe a disposizione da www.wunderground.com/ (l'indirizzo del menu dove sono presentate le API è https://www.wunderground.com/API).

**2 Portafoglio azionario**

**a** Ricerca un gestore di servizi finanziari che metta a disposizione le API per avere in tempo reale la quotazione dei titoli azionari della borsa di Milano.

**b** Quindi realizza un programma dove un investitore può inserire il proprio portafoglio azionario che viene valorizzato in tempo reale e aggiornato a intervalli di tempo prestabiliti (per esempio ogni 5 secondi).

# ESERCIZI IN LABORATORIO 2

## REALIZZAZIONE DI UN WEB SERVICE SOAP IN JAVA

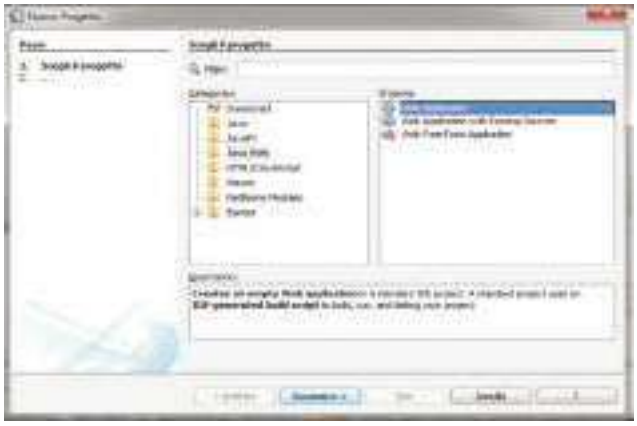
### Come creare un servizio Web Java in NetBeans

**AREA DIGITALE**
Esercizi per il rinforzo e l'approfondimento

In questa esercitazione descriviamo com'è possibile realizzare semplicemente i servizi SOAP in Java utilizzando la l'IDE NetBeans.

#### Passo 1: Creazione del progetto

Dopo aver avviato NetBeans, selezioniamo come nuovo progetto una Web Application:



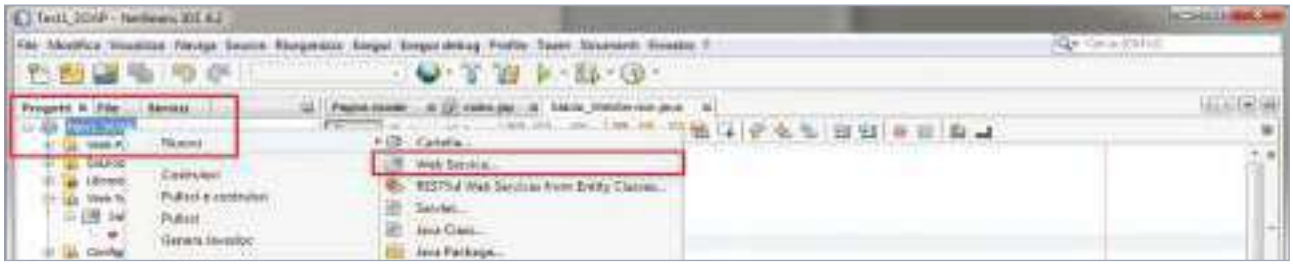
Diamo un nome al progetto, per esempio EsempioCiao-Mondo:

Quindi nella schermata successiva procediamo selezionando il Web server, che per noi è Apache Tomcat:

Nella videata che segue ci viene richiesto di selezionare un Frameworks: per il nostro esempio non è necessario.

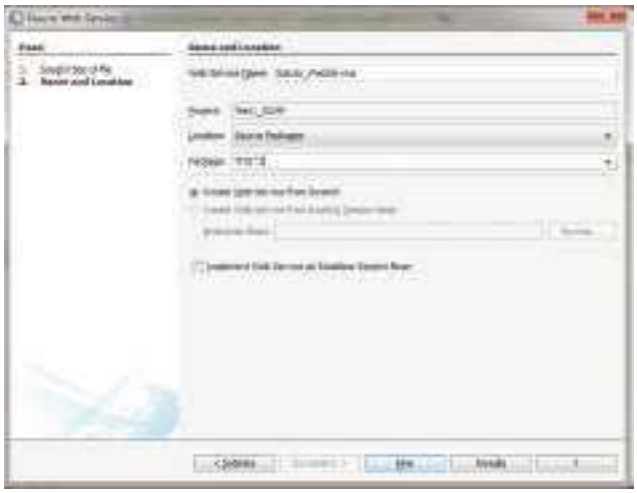
#### Passo 2: Creazione del servizio

Ci posizioniamo ora sull'icona del progetto che abbiamo appena creato e selezioniamo Nuovo per creare in nostro primo servizio:

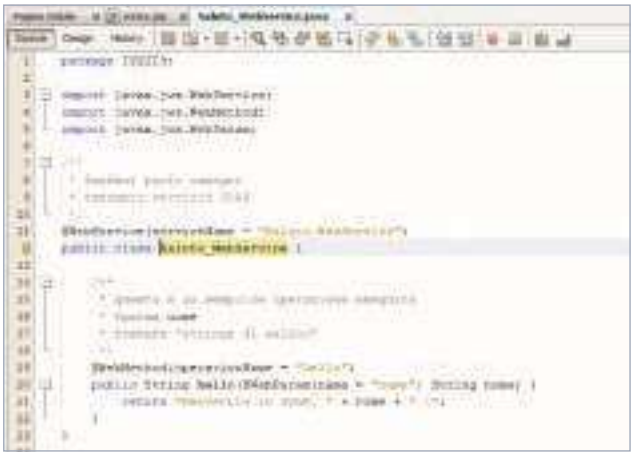


# 2 ESERCIZI IN LABORATORIO

Diamo nome Saluto_WebService al servizio che stiamo realizzando e lo inseriamo nel package TPSIT3:

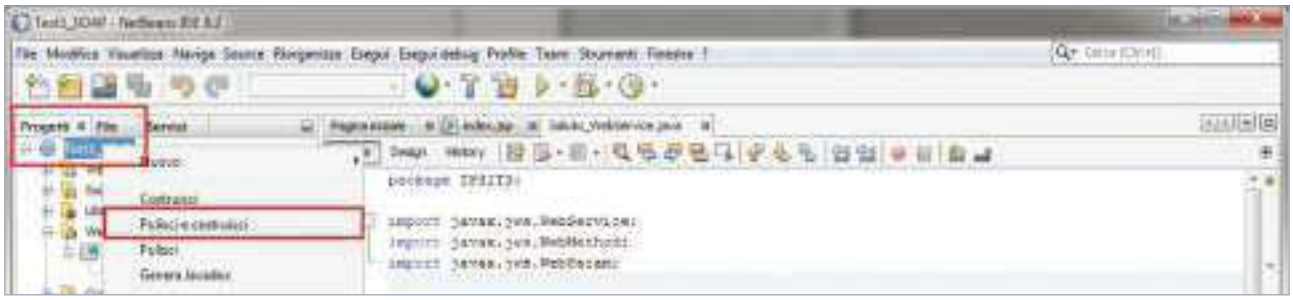


Possiamo ora modificare il codice del metodo che ci viene predisposto: cambiamo le scritte che saranno visualizzate sullo schermo.

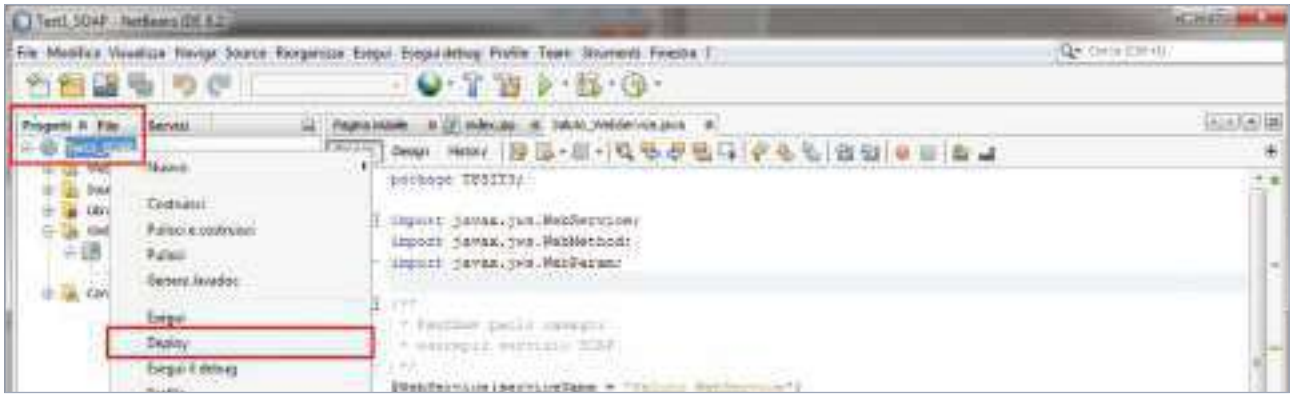


## Passo 3: "Costruiamo" il servizio

Terminata la scrittura del codice, il passo successivo richiede di creare il servizio mediante l'operazione automatica che ci mette a disposizione NetBeans di Pulisci e costruisci (Clean and Build):



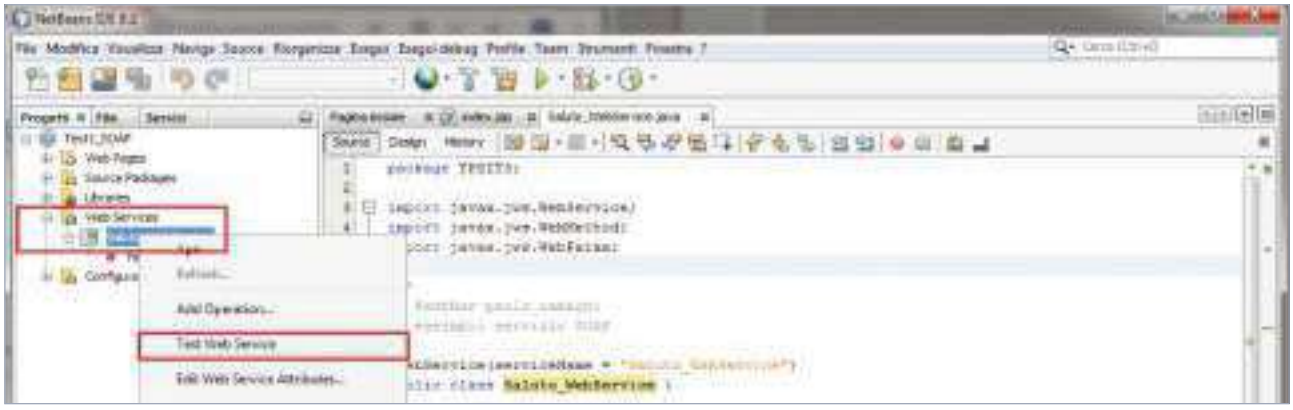
Possiamo ora procedere con il Deploy dell'applicazione che NetBeans effettua sempre in modo automatico.



## Passo 4: Collaudo del servizio

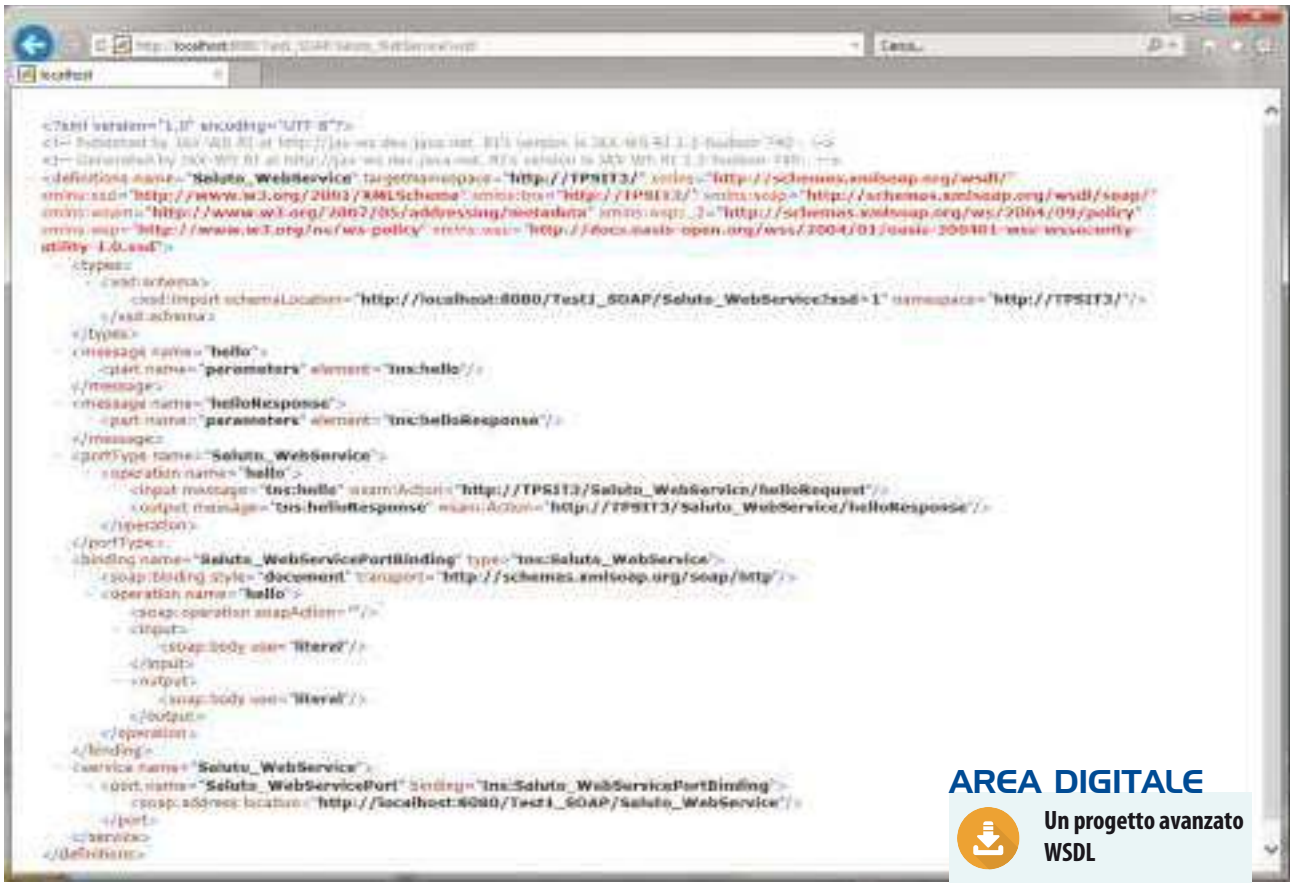
Possiamo ora collaudare il servizio che abbiamo realizzato:

# ESERCIZI IN LABORATORIO 2




Ci vengono visualizzate le informazioni per testare il servizio:

e cliccando sul link possiamo visualizzare il contenuto del file WSDL:



**AREA DIGITALE**

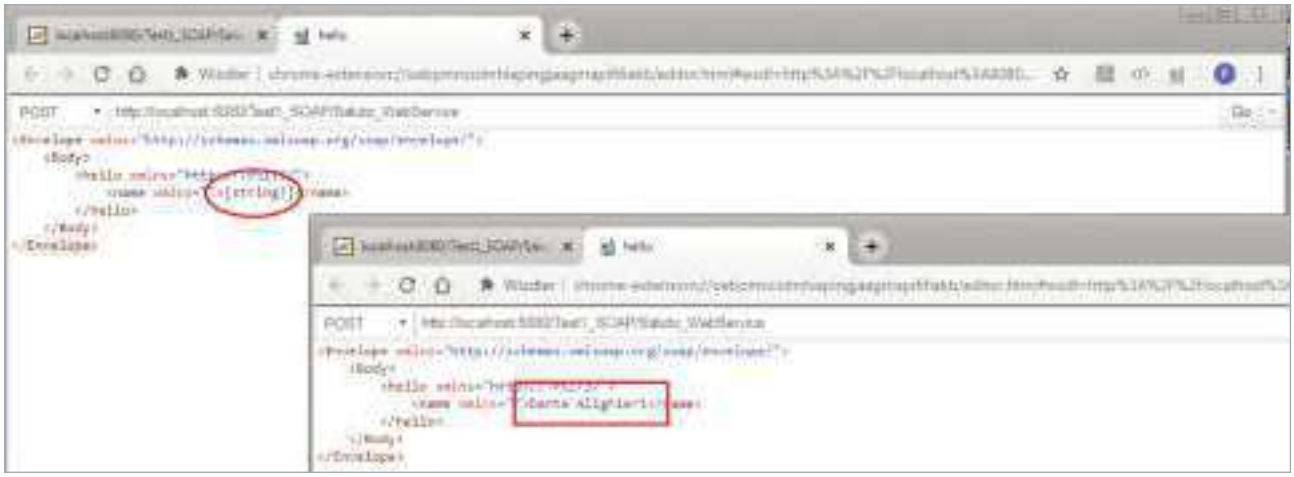
**Un progetto avanzato WSDL**

# 2 ESERCIZI IN LABORATORIO

## Collaudo tramite Wizdler

Testiamo adesso il nostro servizio utilizzando Wizdler: avviamo Chrome, digitiamo l'indirizzo del servizio, cioè l'URI, http://localhost:8080/Test1_SOAP/Saluto_WebService?wsdle e clicchiamo poi sull'icona che visualizza le azioni disponibili:

Clicchiamo ora sull'unico metodo che abbiamo a disposizione, hello, e digitiamo un nome nella casella di testo [string?] che "si aspetta una stringa":



Eseguiamo il servizio, clicchiamo sul pulsante Go e otteniamo la risposta:

# ESERCIZI IN LABORATORIO 3

## REALIZZAZIONE DI UN WEB SERVICE REST IN JAVA

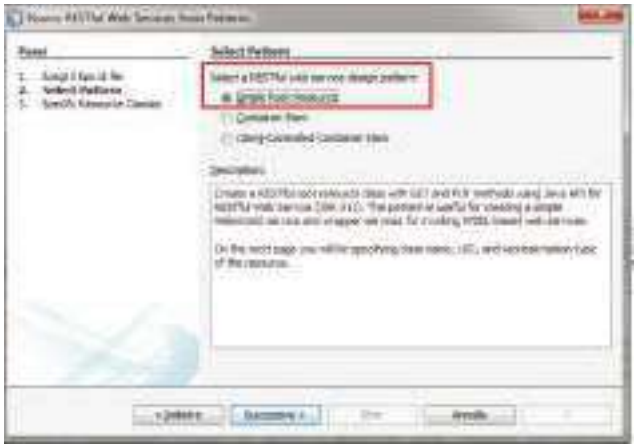
### Creiamo un servizio REST con NetBeans in Java

Mandiamo in esecuzione e clicchiamo su NetBeans > Nuovo Progetto > Java Web > Web Application: assegniamo il nome e la locazione del progetto:

Scegliamo quindi il nome del server (in genere si sceglie Apache Tomcat o TomEE) e la versione di Java EE e, come già visto per il servizio SOAP, non è necessario scegliere alcun framework.

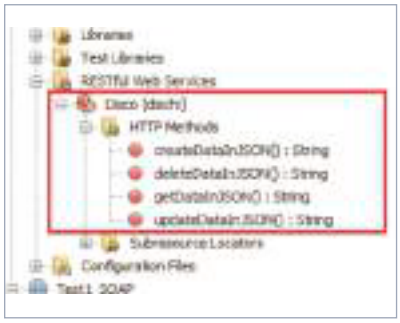
Clicchiamo con il tasto destro sul progetto (icona del mondo) > Nuovo > RESTful Web Services from Patterns... (se non fosse presente nella tendina, lo si può individuare nel menu Altri...)

Selezioniamo il tipo di pattern Simple Root Resource: (figura a sinistra) e scegliamo un nome per il package in cui sono contenute le classi del progetto e il nome della classe che rappresenta i metodi REST (nel nostro caso, TPSIT3_media, dato che realizzeremo servizi per gestire una mediateca di dischi, film ecc.), e chiamiamo Disco la prima classe che realizziamo (figura a destra).



# 3 ESERCIZI IN LABORATORIO

La classe Disco.java contiene i metodi per effettuare le singole operazioni sui contenuti dell'archivio: creiamo quattro metodi, uno per ciascuna delle operazioni CRUD, che saranno automaticamente individuati come metodi Http da NetBeans nel momento del Pulisci e costruisci e Deploy dell'applicazione.




Ogni metodo Http può avere delle clausole:

- nome del metodo Http corrispondente (p.e. @GET);
- il percorso relativo dell'URL: eventuali parametri devono essere posti tra parentesi graffe;
- il tipo di dato restituito: si tratta generalmente di una stringa JSON; la classe MediaType fornisce il tipo di dato restituito (XML, JSON...).

Altre clausole possono essere aggiunte a questi metodi.

I metodi accettano in input dei parametri, interrogano il database e restituiscono una stringa JSON corrispondente al tipo di dato richiesto: descriviamo succintamente la loro codifica.

**1** GET, implementato con il metodo getDataInJSON(), dapprima in una versione che non richiede parametri:

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getDataInJSON() throws Exception
{
  String queryString = "SELECT * FROM disco";
  String JSONDatabase = selectDataFromDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
  return JSONDatabase;
}
```

Quindi in una seconda forma che si aspetta un parametro, che verrà utilizzato nella query per selezionare parzialmente l'archivio:

```
@GET
@Path("{parametro}")
@Produces(MediaType.APPLICATION_JSON)
public String getDataInJSON(@PathParam("parametro") String parametro) throws Exception
{
  String queryString = "SELECT * FROM disco "
    + "WHERE [illegible] LIKE \"%" + parametro + "%\""
    + "OR Genere LIKE \"%" + parametro + "%\""
    + "OR Interprete LIKE \"%" + parametro + "%\""
    + "OR Titolo LIKE \"%" + parametro + "%\""
    + "OR Etichetta  LIKE \"%" + parametro + "%\"";
  String JSONDatabase = selectDataFromDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
  return JSONDatabase;
}
```

**2** POST, implementato con il metodo createDataInJSON():

```
@POST
@Path("{[illegible]}/{Genere}/{Interprete}/{Titolo}/{Etichetta}")
@Produces(MediaType.APPLICATION_JSON)
public String createDataInJSON(@PathParam("[illegible]") String [illegible],
  @PathParam("Genere") String genere,
  @PathParam("Interprete") String interprete,
  @PathParam("Titolo") String titolo,
  @PathParam("Etichetta") String etichetta) throws Exception
```

# ESERCIZI IN LABORATORIO 3

```
    {
      String queryString = "INSERT INTO dischi(NRcatalogo, genere, interprete, titolo, etichetta) "
        + "VALUES (\"" + NRcatalogo + "\",\"" + genere + "\",\"" + interprete + "\",\"" + titolo + "\",\"" + etichetta + "\")"
      String JSONMessage = createDataInDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
      return JSONMessage;
    }
```

Si aspetta come parametri i cinque campi del record, che devono essere inseriti rispettando l'ordine indicato nella figura precedente.

**3** PUT, implementato col metodo updateDataInJSON():

```
    @PUT
    @Path("NRcatalogo/{Genere}/{Interprete}/{Titolo}/{Etichetta}")
    @Produces(MediaType.APPLICATION_JSON)
    public String updateDataInJSON(@PathParam("NRcatalogo") String NRcatalogo,
      @PathParam("genere") String genere,
      @PathParam("interprete") String interprete,
      @PathParam("titolo") String titolo,
      @PathParam("Etichetta") String etichetta) throws Exception
    {
      String queryString = "UPDATE dischi SET genere=\"" + genere + "\", interprete=\"" + interprete + "\","
        + " Etichetta=\"" + etichetta + "\", titolo=\"" + titolo +"\" WHERE NRcatalogo=\"" + NRcatalogo +"\";
      String JSONMessage = updateDataInDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
        return JSONMessage;
    }
```

Come il precedente, anche questo si aspetta come parametri i cinque campi del record, che devono essere inseriti rispettando l'ordine indicato nella figura.

**4** DELETE, implementato con il metodo deleteDataInJSON():

```
    @DELETE
    @Path("NRcatalogo/{}")
    @Produces(MediaType.APPLICATION_JSON)
    public String deleteDataInJSON(@PathParam("NRcatalogo") String NRcatalogo) throws Exception
    {
      String queryString = "DELETE FROM dischi WHERE NRcatalogo=\"" + NRcatalogo + "\";
      String JSONMessage = deleteDataInDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
      return JSONMessage;
    }
```

La cancellazione avviene ricercando il record mediante la chiave primaria NRCatalogo, che è attesa come parametro.

Questi metodi sono contenuti nella classe Disco, che riportiamo di seguito, dopo aver definito il percorso nel quale saranno disponibili le risorse:

```
package IOSIT3_media;

import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.sql.*;
import exception.PersonalException;

@Path("dischi")
public class Disco{
  // stringhe di connessione al database MySql
  private final String CONNECTION_DB = "jdbc:mysql://localhost:3306/";
  private final String CONNECTION_CLASS = "com.mysql.rj.jdbc.Driver";
  private final String DB_NAME = "mediateca";
  private final String DB_USERNAME = "pippo";
  private final String DB_PASSWORD = "123";
  // stringhe di comunicazione esito esecuzione delle query SQL
  private final String SQL_INSERT_CORRECT = "{\"message\": \"Record inserito correttamente.\", \"status\": \"201\"}";
  private final String SQL_UPDATE_CORRECT = "{\"message\": \"Record aggiornato correttamente.\", \"status\": \"200\"}";
  private final String SQL_DELETE_CORRECT = "{\"message\": \"Disco cancellato correttamente.\", \"status\": \"200\"}";
  // definizione gestione personalizzata delle eccezioni
  private final PersonalException perEx;

  public Disco(){
    perEx = new PersonalException();
  }
```

# 3 ESERCIZI IN LABORATORIO

Di seguito riportiamo i metodi Http prima descritti che ricapitoliamo.

```
/** Metodo: getDataInJSON ...9 lines */
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getDataInJSON() throws Exception
{
  String queryString = "SELECT * FROM Dischi";
  String JSONDatabase = selectDataFromDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
  return JSONDatabase;
}

/** Metodo: getDataInJSON ...10 lines */
@GET
@Path("/{parametro}")
@Produces(MediaType.APPLICATION_JSON)
public String getDataInJSON(@PathParam("parametro") String parametro) throws Exception
{...10 lines }

/** Metodo: createDataInJSON ...10 lines */
@POST
@Path("/{Nomecatalogo}/{Genere}/{Interprete}/{Titolo}/{Etichetta}")
@Produces(MediaType.APPLICATION_JSON)
public String createDataInJSON(@PathParam("Nomecatalogo") String Nomecatalogo, @PathParam("Genere") String genere,
  @PathParam("Interprete") String interprete, @PathParam("Titolo") String titolo, @PathParam("Etichetta") String etichetta)
                                                                                                        throws Exception
{
  String queryString = "INSERT INTO dischi (Nomecatalogo, Genere, Interprete, Titolo, Etichetta) "
    + "VALUES (\'" + Nomecatalogo + "\',\'" + genere + "\',\'" + interprete + "\',\'" + titolo + "\',\'" + etichetta + "\')";
  String JSONMessage = createDataIntoDB(DB_NAME, DB_USERNAME, DB_PASSWORD, queryString);
  return JSONMessage;
}

/** Metodo: updateDataInJSON ...10 lines */
@PUT
@Path("/{Nomecatalogo}/{Genere}/{Interprete}/{Titolo}/{Etichetta}")
@Produces(MediaType.APPLICATION_JSON)
public String updateDataInJSON(@PathParam("Nomecatalogo") String Nomecatalogo, @PathParam("Genere") String genere,
  @PathParam("Interprete") String interprete, @PathParam("Titolo") String titolo, @PathParam("Etichetta") String etichetta)
                                                                                                        throws Exception
{...9 lines }

/** Metodo: deleteDataInJSON ...9 lines */
@DELETE
@Path("/{Nomecatalogo}")
@Produces(MediaType.APPLICATION_JSON)
public String deleteDataInJSON(@PathParam("Nomecatalogo") String Nomecatalogo) throws Exception
{...9 lines }
```

Completiamo la classe con la codifica dei metodi privati che servono per eseguire le quattro operazioni Http:

```
// ------------------------------------------------------
// Metodi accessori e privati per le operazioni CRUD
//
// ------------------------------------------------------
// selectDataFromDB -> (READ) operazione di selezione, compressione del JSON e restituzione del risultato.
// createDataIntoDB -> (CREATE) operazione di creazione di un nuovo record.
// updateDataIntoDB -> (UPDATE) operazione di aggiornamento di un record esistente.
// deleteDataIntoDB -> (DELETE) operazione di eliminazione di un record esistente.
// ------------------------------------------------------

// Operazione: READ
private String selectDataFromDB(String nomeDB, String username, String password, String queryString)
{...48 lines }

// Operazione: CREATE
private String createDataIntoDB(String nomeDB, String username, String password, String queryString)
{...38 lines }

// Operazione: UPDATE
private String updateDataIntoDB(String nomeDB, String username, String password, String queryString)
{...38 lines }

// Operazione: DELETE
private String deleteDataIntoDB(String nomeDB, String username, String password, String queryString)
{...38 lines }
}
```

A titolo di esempio, riportiamo il primo metodo, che esegue la query di selezione e prepara la stringa di risposta nella variabile resultString.

# ESERCIZI IN LABORATORIO 3

```
147     // operazione GET
148     private String selectDataFromDB(String nomeDB, String username, String password, String queryString) {
149       ResultSet rs;
150       String connectionString, resultString = "";
151       try {
152         Class.forName(CONNECTION_CLASS);                    // caricamento il driver
153       }
154       catch (ClassNotFoundException e) {
155         resultString = persFac.getClassNotFoundException();
156       }
157       Connection connection = null;
158       try {                                                 // ok connessione al database
159         connectionString = CONNECTION_DB + nomeDB;
160         connection = DriverManager.getConnection(connectionString, username, password);
161         Statement stm = connection.createStatement();
162         rs = stm.executeQuery(queryString);                 // esegguisco la query
163         resultString = "{\"record\":[";                     // prepariamo l'array di risposta
164         while(rs.next()) {
165           resultString += "{\"NMusicalogo\": " + "\"" + rs.getString("NMusicalogo") + "\"
166            + ", \"Genere\": " + "\"" + rs.getString("Genere") + "\" " + ", \"Interprete\": " + "\"" + rs.getString("Interprete") + "\"
167            + ", \"Titolo\": " + "\"" + rs.getString("Titolo") + "\" " + ", \"Elemento\": " + "\"" + rs.getString("Elemento")+ "\"},";
168           }
169           if(resultString.equals("{\"record\":[")) {        // nessun record trovato
170             resultString = persFac.getSQLNoResultFound();
171           }
172           else {
173             resultString = resultString.substring(0, resultString.length() - 1) + "]}";
174           }
175         }
176         catch (SQLException e) {
177           resultString = persFac.getSQLExceptionRequest();  // problemi nella esecuzione
178         }
179         catch (Exception e) {
180           resultString = persFac.getExceptionGeneric();
181         }
182         finally {                                           // chiudiamo connessione
183           try {
184             if (connection != null) connection.close(); }
185           catch (SQLException e) {
186             resultString = persFac.getSQLExceptionClosing();
187           }
188         }
189         return resultString;
190       }
```

## Testiamo il nostro esempio

Il deploy di NetBeans rende automaticamente visibile a Postman i servizi che abbiamo creato: in alternativa possiamo prelevare il file Test1_REST.war presente nella cartella locale:

F:\sviluppo tpsit3\REST\Test1_REST\dist\Test1_REST.war

e copiarlo in xampp/tomcat/webapps: per poterlo utilizzare è necessario che Tomcat sia attivo e la porta di default è 8080 (indirizzo: http://127.0.0.1:8080).
Come ultima opzione possiamo copiare l'intera cartella generata da NetBeans all'interno di webapps:

Per accedere ai vari metodi della nostra classe, l'URL è così composto:

- nome del progetto (nel nostro caso: Test1_REST);
- percorso specificato nel file ApplicationConfig.java (nel nostro caso: /webresources);

# 3 ESERCIZI IN LABORATORIO

```
1   package TPSIT3_media;
2   import java.util.Set;
3   import javax.ws.rs.core.Application;
4   @javax.ws.rs.ApplicationPath("webresources")
5   public class ApplicationConfig extends Application {
6     @Override
7     public Set<Class<?>> getClasses() {
8       Set<Class<?>> resources = new java.util.HashSet<>();
9       addRestResourceClasses(resources);
10      return resources;
11    }
12    private void addRestResourceClasses(Set<Class<?>> resources) {
13      resources.add(TPSIT3_media.Disco.class);
14    }
15  }
```

- percorso specificato nel metodo corrispondente al servizio (nel nostro caso: /dischi) definito come @Path nella dichiarazione della classe.

```
1   package TPSIT3_media;
2
3   import javax.ws.rs.*;
4   import javax.ws.rs.core.*;
5   import java.sql.*;
6   import exception.PersonalException;
7
8   @Path("dischi")
9   public class Disco{
```

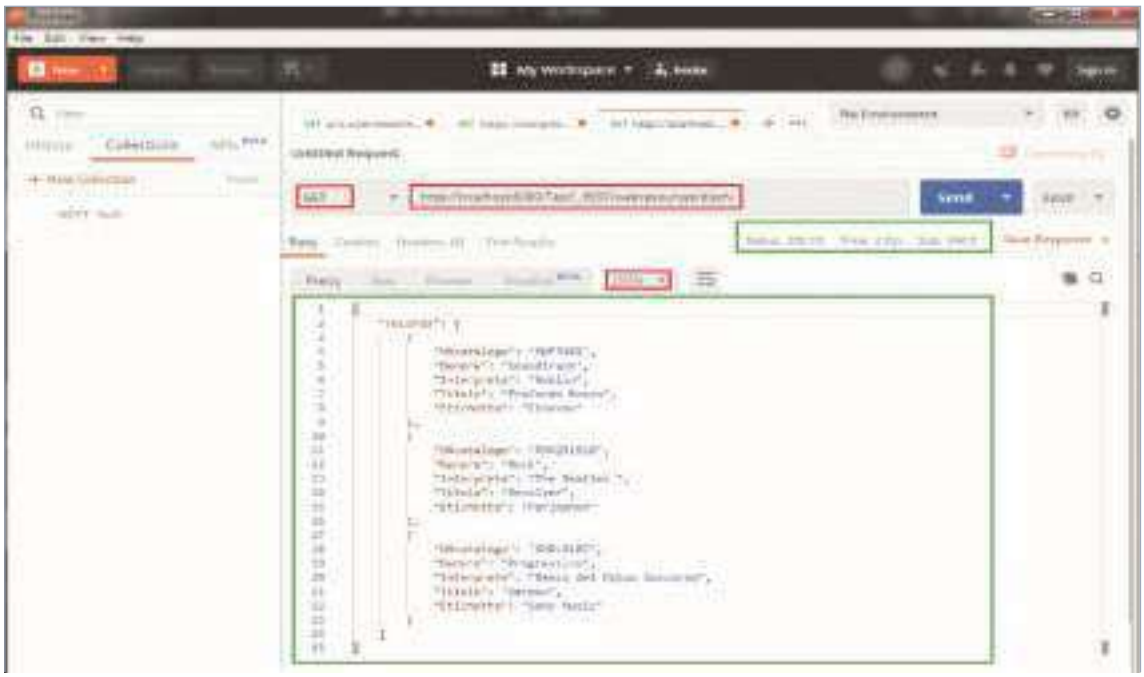
L'indirizzo completo è quindi:

```
http://localhost:8080/Test1_REST/webresources/dischi
```

Come database utilizziamo mediateca, caricabile in MySQL con il file di query mediateca.SQL presente tra le risorse disponibili sul sito Web nel file MySql.jar.

## Metodo GET: lettura di elementi

Avviamo Postman, selezioniamo il metodo GET, e se all'URL, oltre l'indirizzo, non si specificano i parametri viene restituita una stringa JSON con tutti i record presenti nell'archivio.

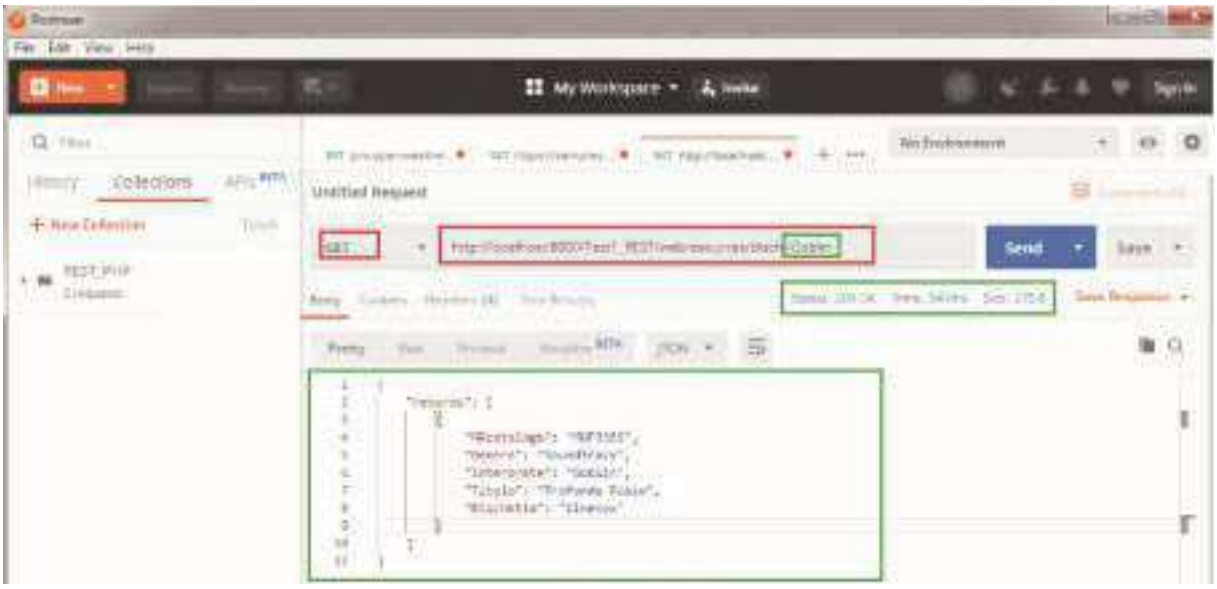


# ESERCIZI IN LABORATORIO 3

Si potrebbe migliorare la ricerca aggiungendo un parametro, che può essere una sottostringa del valore dei campi: aggiungiamo semplicemente Goblin:

```
http://localhost:8080/Test1_REST/webresources/dischi/Goblin
```

e otteniamo come esito 200 e i dati del disco (o dischi) in cui si trova la parola "Goblin" presenti nell'archivio.



## Metodo POST: creazione di una nuova risorsa

I parametri vengono specificati direttamente all'interno dell'URL in maniera ordinata e, in caso di successo, viene restituita una stringa JSON contenente il messaggio con status 201.
Inseriamo per esempio il disco dei Pink Floyd "Ummagomma" che ha il seguente record JSON:

```
{
  "NRcatalogo": "SHVL804",
  "Genere": "Rock",
  "Interprete": "Pink Floyd",
  "Titolo": "Ummagomma ",          // titolo volutamente errato: da correggere con successiva operazione
  "Etichetta": "Harvest"
 }
```

mantenendo l'ordine in cui i dati sono "attesi" dal metodo":

```
String queryString = "INSERT INTO dischi (NRcatalogo, Genere, Interprete, Titolo, Etichetta)"
```

quindi la stringa di comando completa è la seguente:

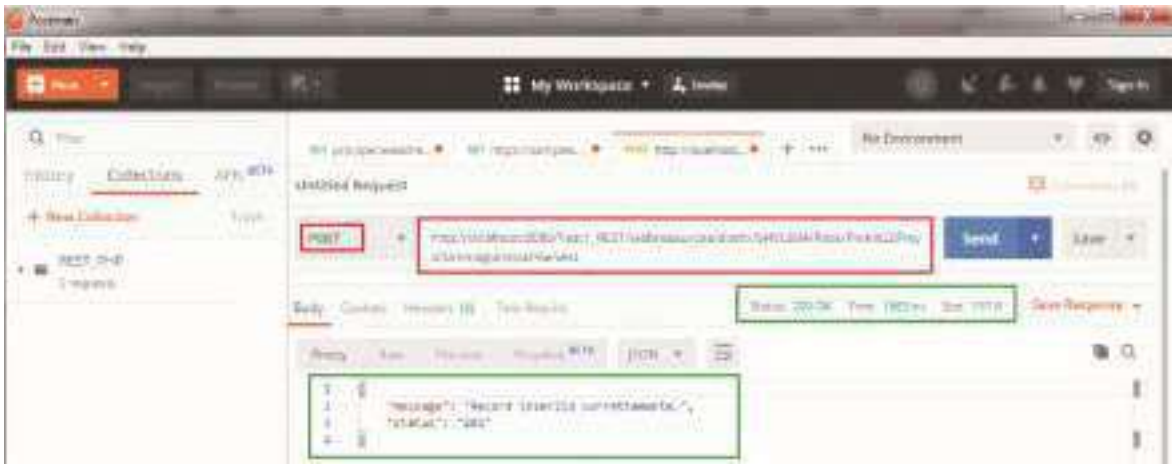
```
http://localhost:8080/Test1_REST/webresources/dischi/SHVL804/Rock/Pink%20Floyd/Ummagomma/Harvest
```

Inseriamo anche un secondo disco dei Goblin che ha il seguente record JSON:

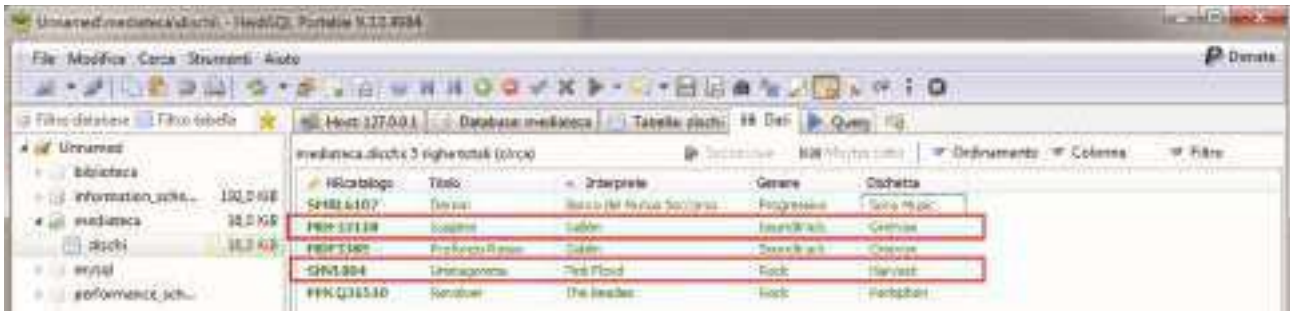
```
http://localhost:8080/Test1_REST/webresources/dischi/MDF33118/Soundtrack/Goblin/Cinevox/Suspiria
```

# 3 ESERCIZI IN LABORATORIO

Com'è possibile vedere nella figura nella scheda Body della risposta, viene restituito l'oggetto JSON contenente in questo caso lo stato Http (in questo caso 201 Created).



Verifichiamo l'esito di queste operazioni utilizzando HeidiSQL:

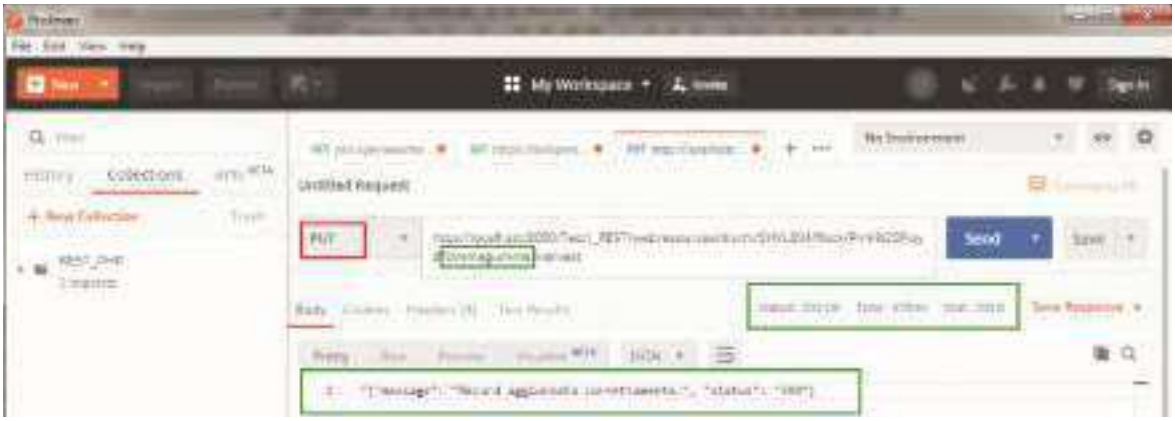


## Metodo PUT: aggiornamento di una risorsa

Anche in questo caso i parametri (sia quelli da cambiare, sia quelli da mantenere intatti) sono contenuti nell'URL: correggiamo il titolo errato del disco appena inserito ("Ummagumma" al posto di "Ummagomma").

```
http://localhost:8080/Test1_REST/webresources/dischi/SHVL804/Rock/Pink%20Floyd/Ummagumma/Harvest
```

In caso di successo, una stringa JSON verrà restituita per confermare il buon esito dell'operazione:



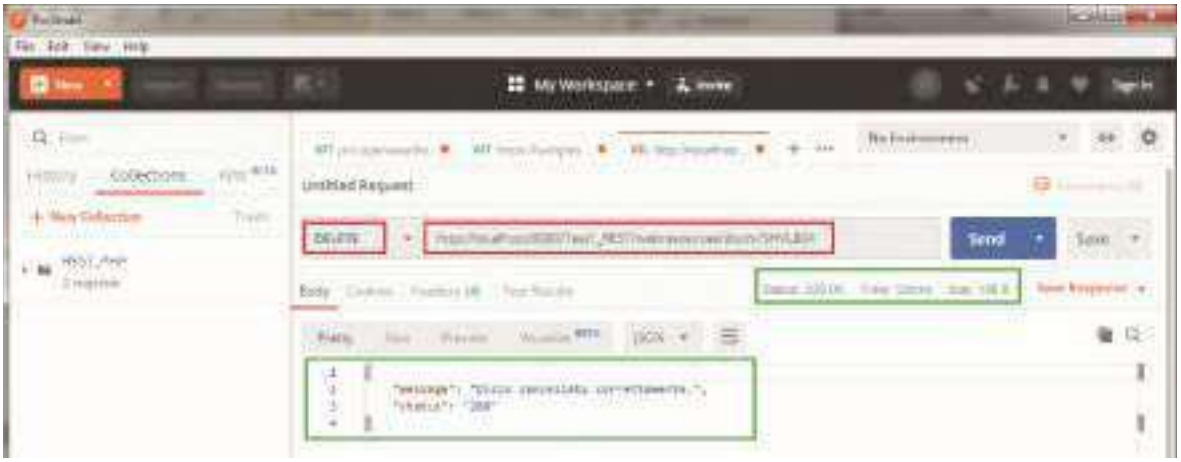
# ESERCIZI IN LABORATORIO 3

## Metodo DELETE: cancellazione di una risorsa

Nell'URL si aggiunge un parametro, cioè l'ISBN del libro da cancellare (essendo chiave primaria, avviene al più una cancellazione per richiesta):

```
http://localhost:8080/Test1_REST/webresources/dischi/SHVL804
```

In caso vada a buon fine, viene restituita una stringa JSON di cancellazione avvenuta:




## METTITI ALLA PROVA

1 Riscrivi il metodo createDataInJSON() in modo che, se si cerca di inserire un disco con NRcatalogo già presente, venga visualizzato un opportuno messaggio di avviso.

2 Riscrivi il metodo getDataInJSON() in modo che si possa passare un parametro per effettuare una ricerca mirata su:
**a** il campo specifico e il valore di ricerca,
oppure
**b** il genere e l'etichetta discografica.

3 Modifica il database aggiungendo i campi edizione_disco data_pubblicazione e aggiorna di conseguenza tutti i metodi.

4 Realizza un nuovo servizio REST per la gestione di un catalogo di vendita automobili, nuove e usate.

# 4 ESERCIZI IN LABORATORIO

## REALIZZAZIONE DI UN WEB SERVICE REST IN PHP

### Creiamo un servizio REST con NetBeans in PHP

In PHP è semplice creare un servizio REST anche senza l'ausilio di alcun framework specifico: lo organizziamo disponendo i file di codice sorgente in una directory (Test2_REST) strutturata in tre sottocartelle:

- config che contiene il file database.php per le istruzioni di connessione al database;
- modelli che contiene il file disco.php, che modellizza in una classe la struttura del database (campo del db -> attributo della classe) e contiene i metodi CRUD veri e propri;
- dischi che contiene gli script in PHP dei metodi CRUD (create.php, read.php, update.php, delete.php) richiamati dal client, in cui vengono delegate le suddette operazioni e vengono gestiti i codici di stato Http.

Vediamo singolarmente i diversi programmi.

Nella cartella config è presente il programma che contiene la classe Database che, a sua volta, annovera la funzione/il metodo che realizza la connessione al database: sono indicate come attributi private le stringhe che includono le credenziali per poter accedere all'archivio.

```
database.php
1  <?php
2  class Database
3  {
4    // Attributi del database: credenziali
5    private $host = "localhost";
6    private $db_name = "mediateca";
7    private $username = "paolo";
8    private $password = "123";
9    public $conn;
10
11   // Funzione di connessione al database
12   public function getConnection()
13   {
14     $this->conn = null;
15     try{
16       $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
17       $this->conn->exec("set names utf8");
18     }
19     catch(PDOException $exception){
20       echo "Errore di connessione: " . $exception->getMessage();
21     }
22     return $this->conn;
23   }
24 }
25 ?>
```

Sempre nella stessa cartella disponiamo di uno script con gli header che verranno richiamati in tutti e quattro i programmi che realizzano le operazioni CRUD.
La prima istruzione rende accessibile le pagine PHP a qualsiasi dominio, la seconda indica di restituire un contenuto di tipo JSON, codificato in UTF-8, le successive rendono accessibile il metodo PUT, definiscono il tempo di refresh, e quali intestazioni Http possono essere utilizzate durante la richiesta effettiva.

```
header.php
1  <?php
2    // Headers
3    header("Access-Control-Allow-Origin: *");
4    header("Content-Type: application/json; charset=UTF-8");
5    header("Access-Control-Allow-Methods: PUT");
6    header("Access-Control-Max-Age: 3600");
7    header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
8  ?>
```

Nella cartella modelli è presente il file disco.php con la descrizione della classe Disco dove, oltre ai metodi getter/setter, sono presenti quattro metodi che eseguono le operazioni CRUD.

# ESERCIZI IN LABORATORIO 4

```
disco.php
1  <?php
2    class Disco
3    {
4      private $conn;
5      private $table_name = "dischi";
6      private $NRcatalogo;
7      private $Genere;
8      private $Interprete;
9      private $Titolo;
10     private $Etichetta;
11
12     public function __construct($db){
13       $this->conn = $db;
14     }
15     // CREAZIONE libro
16     public function create(){ ...
17     }
18     // LETTURA libro
19     public function read(){ ...
20     }
21     // AGGIORNAMENTO libro
22     function update(){ ...
23     }
24     // CANCELLAZIONE libro
25     public function delete(){ ...
26     }
27     // metodi accessori getter e setter
28     public function getNRcatalogo(){
29       return $this->NRcatalogo;
30     }
```

A titolo di esempio riportiamo il metodo create():

```
15     // CREAZIONE libro
16     public function create(){
17       $query = "INSERT INTO ".$this->table_name." SET NRcatalogo=:NRcatalogo, Genere=:Genere, Interprete=:Interprete, Titolo=:Titolo,
             Etichetta=:Etichetta ";
18       $stmt = $this->conn->prepare($query);
19       // legge i dati dall'HTML
20       $this->NRcatalogo = htmlspecialchars(strip_tags($this->NRcatalogo));
21       $this->Genere = htmlspecialchars(strip_tags($this->Genere));
22       $this->Interprete = htmlspecialchars(strip_tags($this->Interprete));
23       $this->Titolo = htmlspecialchars(strip_tags($this->Titolo));
24       $this->Etichetta = htmlspecialchars(strip_tags($this->Etichetta));
25       // Binding
26       $stmt->bindParam(":NRcatalogo", $this->NRcatalogo);
27       $stmt->bindParam(":Genere", $this->Genere);
28       $stmt->bindParam(":Interprete", $this->Interprete);
29       $stmt->bindParam(":Titolo", $this->Titolo);
30       $stmt->bindParam(":Etichetta", $this->Etichetta);
31       // esegue la query
32       if($stmt->execute()){
33         return true;
34       }
35       return false;
36     }
```

Dopo aver predisposto la *query di inserzione*, vengono letti i dati prelevati dalla pagina HTML e analizzati, utilizzando la funzione strip_tag() per rimuovere i tag presenti (PHP e HTML) e la funzione htmlspecialchar() per rimuovere i caratteri speciali HTML.
Una volta che i dati sono sanitize (questo è il termine che indica che sono stati ripuliti da "tutti i caratteri speciali e/o inutili"), si procede con il loro binding alla query predisposta in precedenza.
L'esecuzione della query pone termine alla funzione.

Gli altri tre metodi hanno lo stesso schema di funzionamento, cioè preparazione della query, analisi e pulizia dei dati letti, binding ed esecuzione.

Nella cartella dischi sono presenti i quattro programmi che effettuano le operazioni CRUD: vediamoli in dettaglio uno alla volta.

# 4 ESERCIZI IN LABORATORIO

**1** GET, implementato con il metodo read.php:

```php
<?php
  // Access-Control-Allow-Headers
  include_once '../config/header.php';
  include_once '../config/database.php';
  include_once '../model1/disco.php';

  $database = new Database();          // creiamo un nuovo oggetto database
  $db = $database->getConnection();    // si colleghiamo al nostro database
  $disco = new Disco($db);             // creiamo un nuovo oggetto disco

  $stmt = $disco->read();              // esecuzione della query
  $tanti = $stmt->rowCount();
  if ( $tanti > 0 ){                   // se vengono trovati item nel database
    $arr_dischi = array();             // creazione di un array di dischi
    $arr_dischi["records"] = array();
    while ($row = $stmt->fetch (PDO::FETCH_ASSOC)){
      extract($row);
      $item_disco = array (
        "NRcatalogo" => $NRcatalogo,
        "Genere" => $Genere,
        "Interprete" => $Interprete,
        "Titolo" => $Titolo,
        "Etichetta" => $Etichetta
      );
      array_push($arr_dischi["records"], $item_disco);
    }
    echo json_encode($arr_dischi);
  }
  else{
    echo json_encode( array ( "message" => "Nessun disco trovato." ));
  }
?>
```

Dopo aver richiamato gli header comuni e creato il collegamento al database, viene richiamato il metodo read() che esegue la query di lettura salvando i dischi trovati in un array associativo composto da elementi chiamati records che, una volta pronto, viene convertito in JSON con la funzione json_encode() e ritornato al client.

**2** POST, implementato nel file create.php:

```php
<?php
  // Access-Control-Allow-Headers
  include_once '../config/header.php';
  include_once '../config/database.php';
  include_once '../model1/disco.php';

  $database = new Database();                          // creiamo un nuovo oggetto database
  $db = $database->getConnection();                    // si colleghiamo al nostro database
  $disco = new Disco($db);                             // creiamo un nuovo oggetto disco
  $data = json_decode(file_get_contents("php://input"));  // decodifichiamo il dato JSON ricevuto

  if(!empty($data->NRcatalogo) && !empty($data->Titolo) && !empty($data->Genere) && !empty($data->Interprete) && !empty($data->Etichetta))
  {
    $disco->setNRcatalogo($data->NRcatalogo);
    $disco->setTitolo($data->Titolo);
    $disco->setGenere($data->Genere);
    $disco->setInterprete($data->Interprete);
    $disco->setEtichetta($data->Etichetta);
    $result = $disco->create();
    if($result){                                       // 201 Creazione completata
      http_response_code(201);
      echo json_encode(array("message" => "Disco creato correttamente."));
    }
    else {                                             // 503 Servizio non disponibile
      http_response_code(503);
      echo json_encode(array("message" => "Impossibile creare il disco."));
    }
  }
  else{                                                // 400 Bad request
    http_response_code(400);
    echo json_encode(array("message" => "Impossibile creare il disco, i dati sono incompleti."));
  }
?>
```

# ESERCIZI IN LABORATORIO 4

Dopo aver richiamato gli header comuni e creato il collegamento al database, viene analizzato il file ricevuto e, se sono presenti tutti i dati, questi vengono predisposti per essere inseriti con la query specifica.

**3** PUT, implementato con il metodo update.php:

```php
<?php
  // Access-Control-Allow-Headers
  include_once '../config/header.php';
  include_once '../config/database.php';
  include_once '../model/disco.php';

  $database = new Database();                          // creiamo un nuovo oggetto database
  $db = $database->getConnection();                    // ci colleghiamo al nostro database
  $disco = new Disco($db);                             // creiamo un nuovo oggetto disco
  $data = json_decode(file_get_contents("php://input"));  // decodifichiamo il dato JSON
  ricevuto

  $disco->setNRcatalogo($data->NRcatalogo);            // assegnamo dei dati
  $disco->setTitolo($data->Titolo);
  $disco->setGenere($data->Genere);
  $disco->setInterprete($data->Interprete);
  $disco->setEtichetta($data->Etichetta);
  if($disco->update())
  {
    http_response_code(200);
    echo json_encode(array("risposta" => "disco aggiornato"));
  }
  else
  {                          //503 Servizio non disponibile
    http_response_code(503);
    echo json_encode(array("risposta" => "Impossibile aggiornare il disco"));
  }
?>
```

È molto simile alla funzione precedente ma, naturalmente, non effettua il controllo sulla completezza di tutti i dati, in quanto potrebbe essere aggiornato un unico campo.

**4** DELETE, implementato con il metodo delete.php:

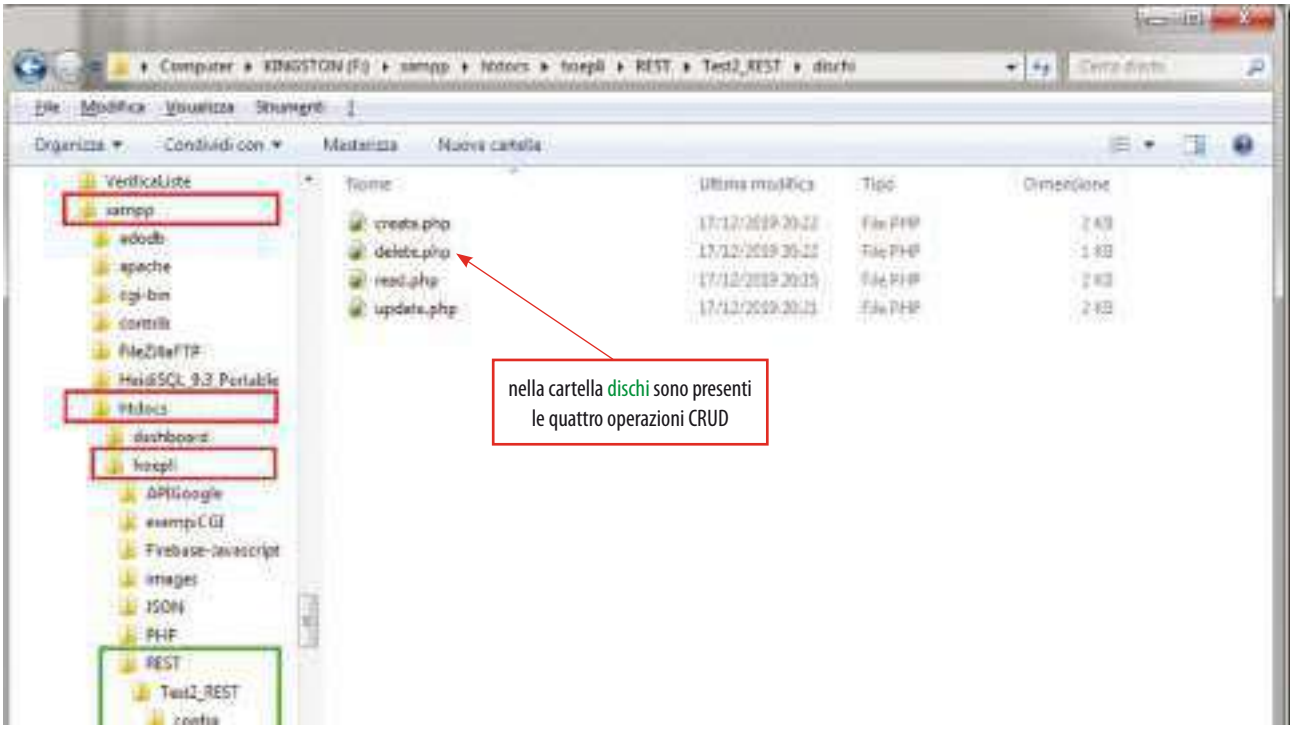
```php
<?php
  // Access-Control-Allow-Headers
  include_once '../config/header.php';
  include_once '../config/database.php';
  include_once '../model/disco.php';

  $database = new Database();                          // creiamo un nuovo oggetto database
  $db = $database->getConnection();                    // ci colleghiamo al nostro database
  $disco = new Disco($db);                             // creiamo un nuovo oggetto disco
  $data = json_decode(file_get_contents("php://input"));  // decodifichiamo il dato JSON
  ricevuto

  $disco->setNRcatalogo($data->NRcatalogo);
  $result = $disco->delete();
  if($result)
  {
    http_response_code(200);
    echo json_encode(array("risposta" => "Il disco e' stato eliminato"));
  }
  else
  {        //503 Servizio non disponibile
    http_response_code(503);
    echo json_encode(array("risposta" => "Impossibile eliminare il disco."));
  }
?>
```

Dopo aver richiamato gli header comuni e creato il collegamento al database, viene analizzato il file ricevuto per individuare la chiave del record da cancellare: quindi viene eseguita la query specifica inviando al client la risposta in base all'esito dell'operazione.

# 4 ESERCIZI IN LABORATORIO

## Testiamo il nostro progetto

Per poter utilizzare il Web Service è necessario che questo risieda in un Web server apposito, che per noi è xampp: collochiamo la nostra cartella all'interno dello spazio già dedicato ai codici descritti in questo volume, cioè la directory hoepli, e come database utilizziamo quello già utilizzato nell'esempio di REST scritto in Java, cioè mediateca (caricabile in MySQL con il file di query mediateca.SQL presente tra le risorse disponibili sul sito Web nel file MySql.jar).
Copiamo quindi la cartella all'interno di questa directory:




Per accedere ai vari metodi della nostra classe, l'URL è così composto:

- nome della directory padre (nel nostro esempio: hoepli/REST);
- nome del progetto (nel nostro esempio: Test2_REST);

L'indirizzo completo è quindi:

```
http://localhost/hoepli/REST/Test2_REST
```

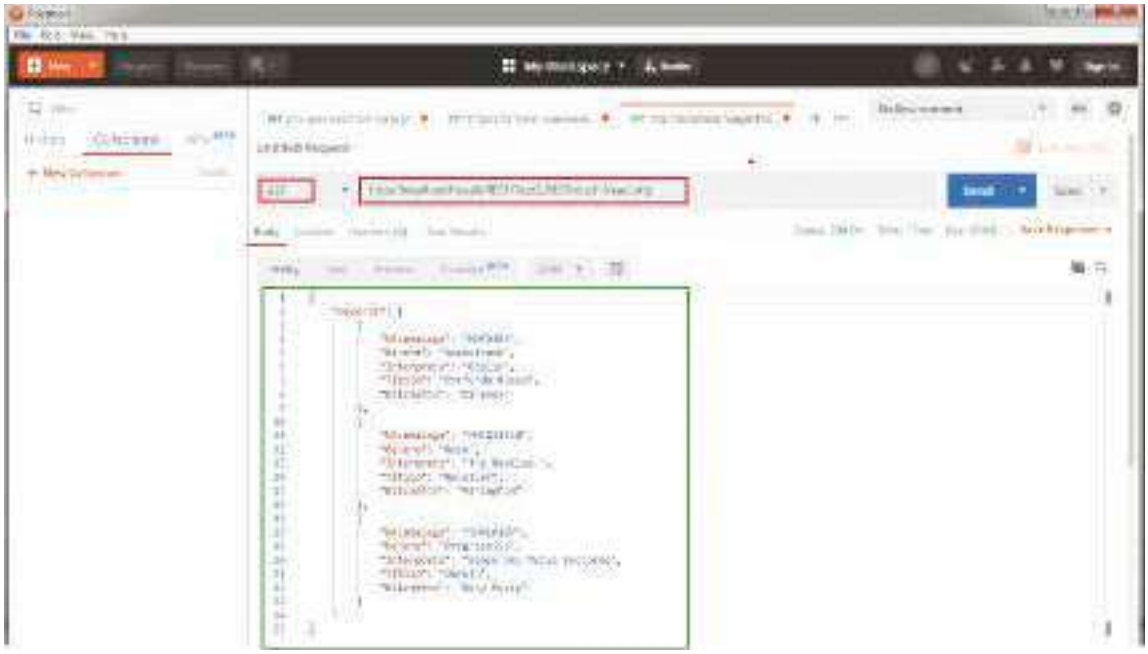
Per testare il nostro Web Service utilizziamo Postman, descritto nell'esercizio precedente e già utilizzato per il collaudo del servizio REST realizzato in linguaggio Java.

## Metodo GET: lettura di elementi con read.php

Avviamo Postman, selezioniamo il metodo GET, e se all'URL non si specificano parametri oltre l'indirizzo, viene restituito una stringa JSON con tutti i record presenti nell'archivio.

```
http://localhost/hoepli/REST/Test2_REST/dischi/read.php
```

# ESERCIZI IN LABORATORIO 4



## Metodo POST: creazione di una nuova risorsa con create.php

Il metodo viene richiamato scrivendo il nome dell'"URL senza che siano specificati parametri oltre:

```
http://localhost/hoepli/REST/Test2_REST/dischi/create.php
```

Inseriamo per esempio il disco dei Pink Floyd "The Dark Side Of The Mooon" scrivendo i dati del record da inserire nel Body, opzione raw, in formato JSON:

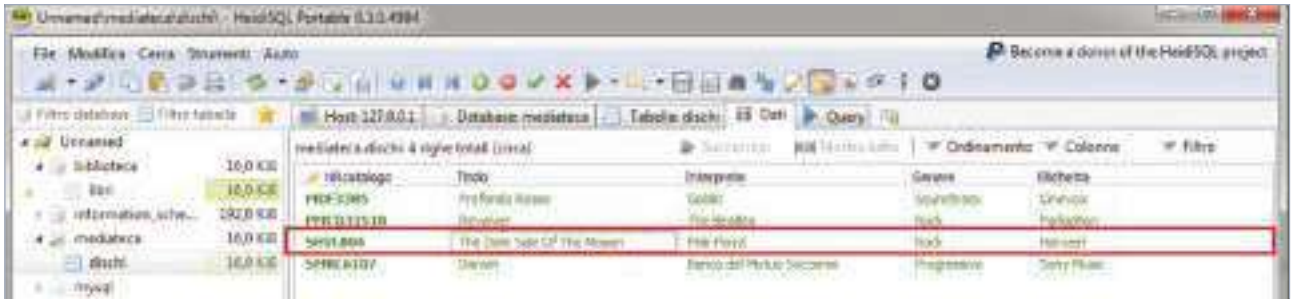
```
{
  "NRcatalogo": "SHVL804",
  "Genere": "Rock",
  "Interprete": "Pink Floyd",
  "Titolo": "The Dark Side Of The Mooon",      // titolo volutamente errato: da correggere con successiva operazione
  "Etichetta": "Harvest"
 }
```

Quindi selezioniamo il metodo POST e inviamo la richiesta:

# 4 ESERCIZI IN LABORATORIO

Com'è possibile vedere nella figura, nella scheda Body della risposta viene restituito l'oggetto JSON contenente in questo caso il messaggio da noi predefinito per l'esito positivo (status: 201 Created).

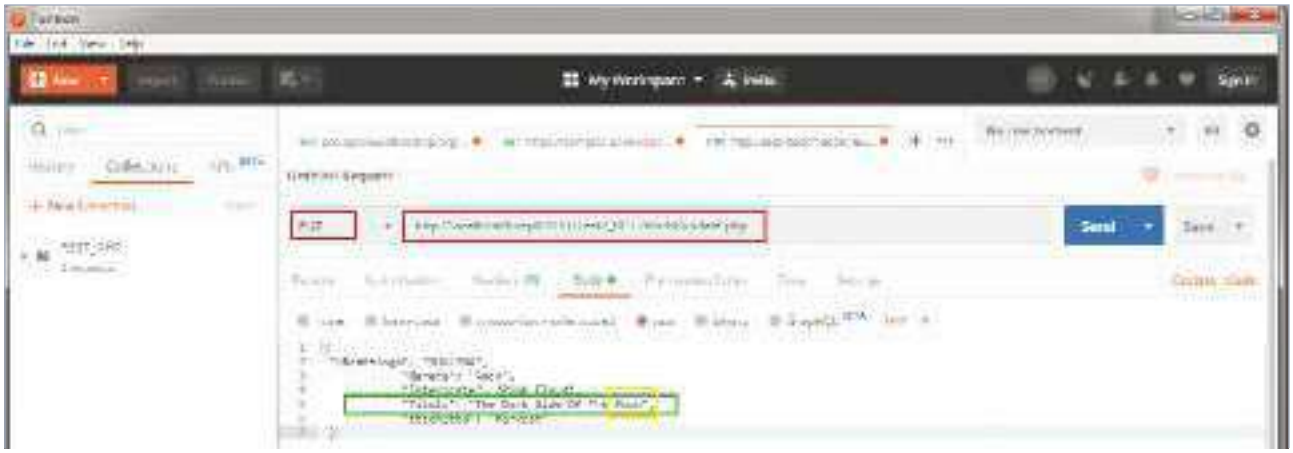
Verifichiamo l'esito di questa operazione tramite HeidISQL:



## Metodo PUT: aggiornamento di una risorsa con update.php

Anche in questo caso i parametri (sia quelli da cambiare, sia quelli da mantenere intatti) sono contenuti nell'URL: correggiamo l'errore presente nel titolo del disco appena inserito ("Moon" al posto di "Mooon").

```
http://localhost/hoepli/REST/Test2_REST/dischi/update.php
```



In caso di successo, una stringa JSON verrà restituita per confermare il buon esito dell'operazione:

# ESERCIZI IN LABORATORIO 4

## Metodo DELETE: cancellazione di una risorsa delete.php

Nel Body si inserisce una stringa JSON contenente solo il NRcatalogo del disco da cancellare (che è un campo definito come chiave primaria, quindi avviene al più una cancellazione per richiesta):

```
http://localhost/hoepli/REST/Test2_REST/dischi/delete.php
```

In caso di successo viene restituita una stringa JSON di cancellazione avvenuta:

Una seconda possibilità è quella di aggiungere il NRcatalogo del disco da cancellare come parametro nell'URL: per esempio cancelliamo il disco con numero di catalogo SHVL804.

```
http://localhost/hoepli/REST/Test2_REST/dischi/delete.php?NRcatalogo=SHVL804
```

Naturalmente è necessario completare il codice php del metodo delete() in modo che il parametro venga riconosciuto e interpretato correttamente.

# 4 ESERCIZI IN LABORATORIO

## METTITI ALLA PROVA

**1** Riscrivi il metodo create.php in modo che, se si cerca di inserire un disco con NRcatalogo già presente, venga visualizzato un opportuno messaggio di avviso.

**2** Riscrivi il metodo read.php in modo che si possa passare un parametro per effettuare una ricerca mirata su:
**a** il codice NRcatalogo,

oppure
**b** l'interprete del disco,
infine
**c** il titolo del disco, e in caso di insuccesso, che vengano visualizzati i dischi che hanno i primi 10 caratteri uguali alla stringa di ricerca.

**3** Modifica il database aggiungendo i campi edizione_disco data_pubblicazione e aggiorna di conseguenza tutti i metodi.

**4** Realizza un nuovo servizio REST per la gestione di un catalogo di vendita automobili, nuove e usate.

# ESERCIZI IN LABORATORIO 5

**AREA DIGITALE**

**ESERCIZI IN LABORATORIO 6**
**Utilizzo avanzato delle API di Google in PHP**

## INTEGRAZIONE DEI SERVIZI API NELLE APPLICAZIONI PHP

### Un'applicazione finanziaria: il cambio delle valute

Come primo esempio realizziamo un'applicazione che in tempo reale ci fornisce il cambio di alcune valute rispetto all'euro. Intestiamo la pagina html "minimalista", senza cioè preoccuparci dell'aspetto grafico.

```
1  <!DOCTYPE html>
2  <html lang="en" dir="ltr">
3      <head>
4          <meta charset="utf-8">
5          <title>Cambiavalute</title>
6      </head>
7      <body>
```

Usiamo la tecnica PostBack, che richiama ricorsivamente la pagina stessa, partendo da un semplice form che visualizza in un combo box l'elenco delle valute disponibili e che riporta come valore di ogni option il codice internazionale della valuta (isoCode):

```
34          else {
35          ?>
36          <form action="" method="post">
37              Valuta di conversione:
38              <select name="valuta">
39                  <option value="USD">Dollaro statunitense</option>
40                  <option value="CHF">Franco svizzero</option>
41                  <option value="GBP">Sterline britanniche</option>
42              </select><br><br>
43              <input type="submit" name="submit" value="Invia">
44          </form>
```

Selezionata e confermata la valuta desiderata, utilizziamo le API messe a disposizione dalla Banca d'Italia: le modalità di utilizzo sono descritte in un documento presente alla pagina https://www.bancaditalia.it/compiti/operazioni-cambi/ al link Estrazioni Automatiche:

# 5 ESERCIZI IN LABORATORIO

Abbiamo scaricato il documento Nuove_Istruzioni_tecnico-operative.pdf dove sono descritti i servizi offerti e le modalità per usufruirne: noi utilizzeremo la modalità di scambio dati in formato JSON, dove i singoli record di ciascuna valuta hanno il seguente formato:

```
"latestRates":
[
    {
        "country": "AFGHANISTAN",
        "currency": "Afghani",
        "isoCode": "AFN",
        "uicCode": "115",
        "eurRate": "1.0000",
        "usdRate": "1.0000",
        "usdExchangeConvention": "Quantita' di valuta estera per 1 Dollaro",
        "usdExchangeConventionCode": "C",
        "referenceDate": "2017-08-18"
    },
```

Settiamo quindi le variabili che ci permettono di specificare le opzioni desiderate:

```
8   <?php
9   if (isset($_POST["valuta"])) {
10      $apiURL = "https://tassidicambio.bancaditalia.it/terzevalute-wf-web/rest/v1.0/latestRates";
11      // Opzioni HTTP (ad esempio, il formato dei dati)
12      $options = [
13          "http" => [
14              "method" => "GET",
15              "header" => "Content-Type: application/x-www-form-urlencoded \r\n",
16                          "Accept: application/json \r\n"     // Richiedi i dati in formato JSON
17          ]
18      ];
```

Successivamente effettuiamo la richiesta di ricezione del file con i dati desiderati:

```
19      // Creo il contesto dell'header e ricevi la risposta
20      $context = stream_context_create($options);
21      $response = file_get_contents($apiURL, false, $context);
```

Il file ricevuto viene quindi convertito dal formato JSON per poter elaborare i singoli record:

```
23      // Decodifica la risposta in formato JSON
24      $json = json_decode($response, true);
25      $latestRates = $json["latestRates"];
```

Analizziamo i dati ricevuti individuando, tramite il codice, il valore in tempo reale del cambio per la divisa desiderata:

```
27      // Mostra il valore di cambio della valuta richiesta
28      foreach ($latestRates as $rate) {
29          if ($rate["isoCode"] == $_POST["valuta"]) {
30              echo "1 Euro = ".$rate["eurRate"]." ".$rate["currency"]."<br><br>";
31          }
32      }
```

Un'esecuzione viene riportata di seguito:

Il codice completo è riportato nel file cambiavalute.php.

# ESERCIZI IN LABORATORIO 5

## METTITI ALLA PROVA

**1** Scrivi un'applicazione che effettua il cambio di un importo inserito da un utente in Euro in una valuta selezionata da un combo box.

**2** Modifica il programma precedente visualizzando il valore equivalente in 3 diverse valute, sempre selezionate dall'utente.

**3** Scrivi un'applicazione che effettua il cambio di un importo inserito da un utente selezionando la valuta di partenza e quella d'arrivo.

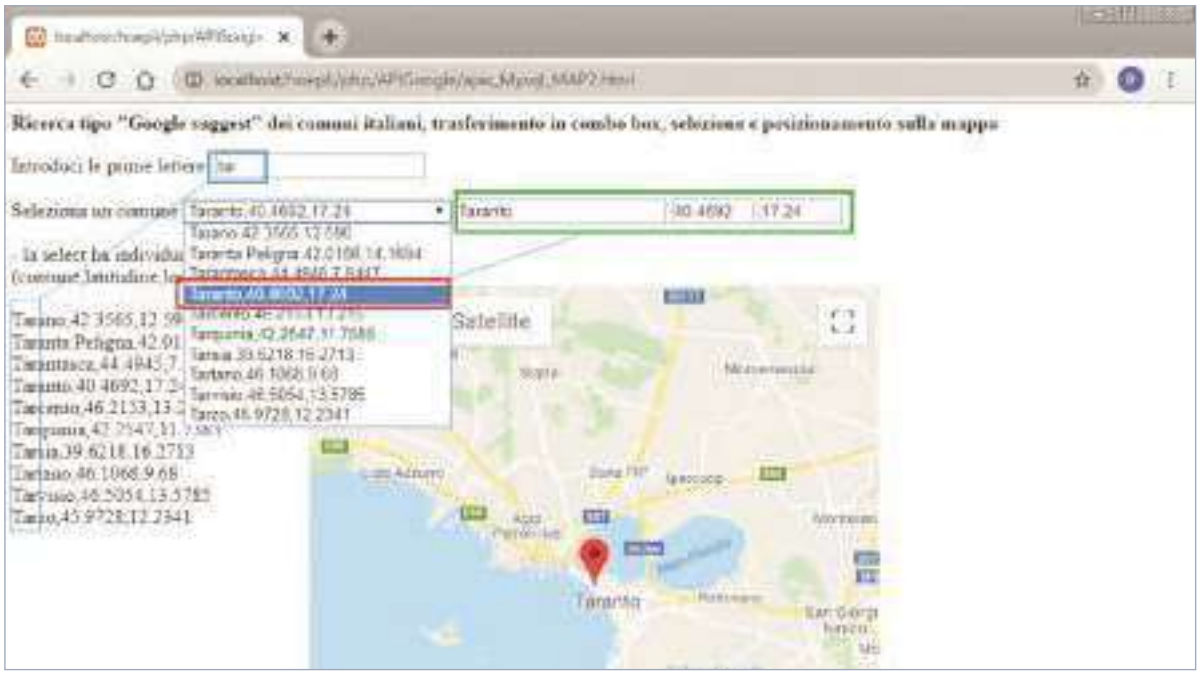
**4** Scrivi un'applicazione che visualizza l'andamento del cambio di una valuta per un periodo di tempo indicato dall'utente, selezionando anche l'intervallo tra due lettura.
Per esempio, proponi la richiesta della data iniziale e quella finale tramite un calendario e la scelta dell'intervallo mediante un group box che propone tre alternative:
- giornaliero,
- settimanale,
- mensile.

**5** Utilizzando le Api messe a disposizione da Yahoo per ottenere in tempo reale la quotazione dei titoli sui principali mercati (Yahoo! Finance APIs), scrivi un'applicazione che in base a una richiesta dell'utente:
- **a** visualizza la quotazione dell'azione desiderata in tempo reale indicando la % di guadagno/perdita rispetto all'apertura giornaliera;
- **b** visualizza la quotazione di apertura giornaliera dell'azione desiderata;
- **c** visualizza la quotazione minima/massima annuale;
- **d** consente di gestire un "portafoglio azionario" in cui l'utente inserisce i propri titoli e il programma visualizza in contemporanea tutte le quotazioni in tempo reale;
- **e** permette all'utente stesso di inserire nel portafoglio azionario anche le quantità e il prezzo medio di carico delle azioni, in modo da poter effettuare in tempo reale la loro valorizzazione indicando per ciascun titolo guadagni/perdite sia in termini di valore che di %.

# 7 ESERCIZI IN LABORATORIO

## LE API DI GOOGLE: INTERAZIONE TRA JAVASCRIPT, AJAX, PHP E SQL

In questa esercitazione vedremo come far interagire le mappe di Google con PHP in modo da estrarre informazioni da un archivio per visualizzarle sulla cartina geografica.
In particolare realizziamo uno script dove l'utente viene assistito da un meccanismo simile a Google suggest per ricercare una località all'interno della tabella comuni del database provephp: i risultati della ricerca vengono presentati in una casella di selezione e quando l'utente individua e clicca su quello desiderato, vengono visualizzate le sue coordinate geografiche e viene localizzato sulla mappa mediante un marker.
Riportiamo un esempio di funzionamento nel caso in cui si ricerchi la città di Taranto: dopo la digitazione della terza lettera vengono individuati 10 possibili alternative di comuni che iniziano con "tar" e vengono caricati nel combo box: selezionando quello desiderato, i suoi dati vengono messi in tre caselle di testo e la sua posizione viene individuata e visualizzata sulla mappa.




Per la realizzazione di queste funzionalità la pagina utilizza i seguenti linguaggi:
- HTML, JavaScript, PHP, SQL,

e le seguenti librerie:
- AJAX, API Google Maps.

L'esempio è formato da due file:
**A** il primo, ajax_Mysql_MAP2.html, lato client, che ha il compito di:
- leggere il nome del comune da cercare con una casella di testo e trasferirlo alla pagina elabora.php;
- ricevere l'elenco di comuni trovati e collocare i nomi restituiti dal database in un riquadro (chiamato box) posto all'interno di un <DIV> della pagina HTML che viene formattato con un sfondo definito con uno stile CSS;
- mostrare il risultato in un combo box e, se selezionato, visualizzarne la posizione sulla mappa;

**B** il secondo, elabora.php, lato server, già descritto nella Lezione 2 dell'Unità di Apprendimento 3, che ha il compito di:
- ricevere la stringa da cercare;
- effettuare un'interrogazione alla tabella comuni del database provephp;
- restituire l'elenco di comuni trovati all'interno di una stringa alla pagina HTML.

# ESERCIZI IN LABORATORIO 7

Iniziamo a vedere il codice della prima pagina ajax_Mysql_MAP2.html.
Prima di tutto colleghiamo i fogli di stile necessari alla formattazione dei due <DIV> chiamati 'box' e 'mappa' che conterranno rispettivamente la mappa e l'elenco dei comuni e inseriamo la Google key.

```
1  <HTML><HEAD>
2  <LINK HREF="/maps/documentation/javascript/examples/default.css" REL="stylesheet">
3  <script TYPE="text/javascript"
4  SRC="https://maps.googleapis.com/maps/api/
   js?key=AIzaSyCATzhGaTgFrBZcRJdmPWFLBTDNG_5oWA&sensor=false">
5  </script>
```

La pagina HTML può essere divisa in 3 sezioni:
**1** la sezione in cui vengono utilizzate le API di Google Map;
**2** la sezione in cui utilizziamo AJAX per collegarci con la pagina elabora.php;
**3** la sezione di HTML che visualizza i risultati.

Nella prima parte vengono dichiarate alcune variabili globali: la matrice posti[], che conterrà le posizioni dei marker, l'oggetto map di classe Maps e l'oggetto marker di classe Marker.
La funzione inizializza() viene chiamata all'avvio della pagina e definisce le opzioni della mappa e la sua posizione iniziale; quindi istanzia l'oggetto map (riga 17).

```
7  <script>
8  var map; // dichiarazione oggetto map di tipo globale
9  function inizializza() {          // funzione richiamata all'apertura del body
10     var opzioni = {               // dichiarazione opzioni della mappa
11         zoom: 11,                 // livello di zoom della mappa mostrata
12         // inizialmente la mappa e' centrata sulla citta' di Roma
13         center: new google.maps.LatLng(41.8184,9.8885),
14         // tipo di mappa (strade=ROADMAP, mappa HYBRID, SATELLITE, TERRAIN)
15         mapTypeId: google.maps.MapTypeId.ROADMAP
16     }
17     map = new google.maps.Map(document.getElementById('mappa'),opzioni); // oggetto mappa
18 }
19 </script>
```

La funzione posiziona() viene chiamata quando si effettua una selezione all'interno del combo box: dall'opzione selezionata vengono ricavate le coordinate di geolocalizzazione (righe 24÷33), viene centrata la mappa (riga 35), quindi definito e posizionato il marker (righe 36÷42).

```
31 <script type="text/javascript">
32 function posiziona(sel){       // centra la mappa e colloca il marker
33   var f = document.frm;
34   // dati selezionati dal combo
35   f.sel_value.value = sel.options[sel.selectedIndex].value;
36   f.sel_text.value = sel.options[sel.selectedIndex].text;
37   // elaborazione dati selezionati
38   stringa = f.sel_value.value;
39   st3 = stringa.split(",");   // separa la stringa in tre campi di un array
40   // visualizzo i dati nelle caselle di testo
41   f.citta.value = st3[0];     // nome comune
42   f.lat .value  = st3[1];     // latitudine
43   f.lng .value  = st3[2];     // longitudine
44   // centro la mappa e posiziono il cursore
45   map.setCenter(new google.maps.LatLng(f.lat.value,f.lng.value));
46   var posCur = new google.maps.LatLng(f.lat.value,f.lng.value);  // poniamo il cursore attento
47   marker = new google.maps.Marker({
48     map:map,
49     draggable:true,
50     animation:google.maps.Animation.DROP,
51     position: posCur
52   });
53 }
54 </script>
```

# 7 ESERCIZI IN LABORATORIO

Descriviamo la sezione di AJAX che è "racchiusa" nella funzione mostra(): questa funzione viene chiamata dalla sezione HTML (riga 106) ogni volta che l'utente digita un carattere nella casella di testo ricevendo come parametro str la stringa digitata fino a quel momento.

L'istruzione di riga 49 valuta il valore della stringa e se ha lunghezza nulla si procede con la creazione di un oggetto xmlhttp: la sua proprietà onreadystatechange che contiene la funzione anonima necessaria per elaborare i dati in provenienza dal server (riga 55):

- la proprietà readyState contiene lo stato della response: se è 4 significa che la richiesta è completa;
- la proprietà status, invece, contiene lo stato della risposta: se 200 significa che la risposta è corretta, mentre se vale 404 indica "Not found".

Nella riga 57 viene verificato il valore di questi attributi e, in caso di esito favorevole, viene collocata la proprietà responseText nel riquadro riservato al testo (riga 59), cioè viene inserito l'elenco dei nomi ricevuti dal server all'interno del <DIV> 'risposta'.

```
46  <script>
47  // funzione attivata a ogni pressione di un tasto sulla casella di testo
48  function mostra(str){
49    if (str.length == 0){  // se stringa testo vuota viene inviato risposta bianca
50      document.getElementById("risposta").innerHTML = "";
51      return;
52    }
53    ajax = new XMLHttpRequest();              // creazione oggetto AJAX/HttpRequest
54    // onreadystatechange contiene la funzione che elabora i dati ricevuti dal server
55    ajax.onreadystatechange = function(){
56      // readyState = 4 richiesta completa e status = 200 OK
57      if (ajax.readyState == 4 && ajax.status == 200){
58        // i dati ricevuti sono ricevuti da responseText di tipo di stringa
59        document.getElementById("risposta").innerHTML = ajax.responseText;
60        // le righe sono composte da [nome, lat, long ecc]
```

L'ultima parte della sezione di AJAX è quella che "riempie" il combo box a partire dai dati ricevuti dal server: come prima operazione viene "svuotato" il combo box dai dati precedentemente caricati (righe 64÷66) e viene dichiarata una variabile stringa che conterrà l'elenco così come ricevuto dal server (riga 67).

Come si vede dal "pezzo" riportato come esempio, la stringa è formata da diverse righe, ciascuna delle quali termina con il tag <BR> necessario per andare a capo:

```
Millesimo,44.3641,8.2059<BR>Milano,45.4612,9.1878<BR>Milzano,45.2745,10.2003<BR>Milazzo,
38.2194,15.2404<BR>Militello Rosmarino,38.0472,14.6778<BR>Milena,37.4708,13.7361...
```

Il metodo split("<BR>") di riga 69 elabora riga per riga estraendo i singoli campi e aggiungendoli all'interno delle opzioni del combo box.

```
61        // i campi sono individuati, separati e messi in un combo box
62        x=document.getElementById("elencoCombo");
63        // svuoto combo da vecchi dati
64        for(k = x.options.length-1; k>=0 ;k--){
65          x.remove(k);
66        }
67        stringa = ajax.responseText;
68        // estrae singole righe per riempire il combo
69        s = stringa.split("<BR>");
70        for(k = s.length-2; k >= 1; k--){
71          var o = document.createElement("option");
72          o.text  = s[k];
73          o.value = s[k];
74          x.add(o,0);
75        }
```

# ESERCIZI IN LABORATORIO 7

Al termine dell'inserimento si controlla se è presente un solo elemento nel combo box e, in tale caso, viene direttamente posizionato sulla mappa il marker che ne indica la sua posizione.

```
76        // se e' presente un solo elemento nel combo box lo visualizziamo direttamente
77        if (s.length == 5){
78          st = s[3].split(",");
79          var f = document.frm;
80          f.citta.value = st[0];
81          f.lng .value  = st[2];
82          f.lat .value  = st[1];
83          map.setCenter(new google.maps.LatLng(f.lat.value,f.lng.value));
84          // posizioniamo il marker nella posizione del cursore
85          var posCur = new google.maps.LatLng(f.lat.value,f.lng.value);
86          marker = new google.maps.Marker({
87            map:map,
88            draggable:true,
89            animation:google.maps.Animation.DROP,
90            position: posCur
91          });
92        }
```

Il metodo open() (riga 96) invia la richiesta alla pagina elabora.php del server e infine il metodo send() trasmette effettivamente la richiesta al server (riga 97).

```
95    // apriamo la connessione con il server e inoltriamo con GET i parametri
96    ajax.open("GET", "elabora.php?stringa=" + str, true);
97    ajax.send();  // inoltra della richiesta al server
98  }
99  </script>
```

La parte finale del codice è rappresentata dal codice HTML che mostra la casella di testo in cui digitare il comune da ricercare (riga 106) alla quale è associato l'evento onkeyup che richiama la funzione mostra(this.value) di cui abbiamo già parlato in precedenza.
Di seguito viene mostrato il combo box che contiene l'elenco dei comuni "filtrati" dalle iniziali della parola che si sta digitando e, se una opzione viene selezionata, si richiama la funzione posiziona(this) che visualizza il comune scelto sulla mappa.
Vengono poi mostrati due <DIV> per la risposta e per la mappa (righe 120 e 121):

```
101  </HEAD>
102  <BODY onload="inizializza()">
103  <P><B>Ricerca tipo &quot;Google suggest&quot; dei comuni Italiani, trasferimento in combo
     box, selezione e posizionamento sulla mappa </B></P>
104  <FORM>
105  Introduci le prime lettere:
106    <INPUT TYPE="text" onkeyup="mostra(this.value)">
107  </FORM>
108  <FORM name="frm">
109    Seleziona un comune
110    <SELECT name="elencoComuni" size="1" id="elencoCombo" onchange="posiziona(this)">
111      <OPTION>&lt;- elenco comuni individuati&gt;</OPTION>
112    </SELECT>
113    <INPUT type="hidden" name="sel_value">
114    <INPUT type="hidden" name="sel_text">
115    <INPUT name="citta" size="20">
116    <INPUT name="lat" size="5">
117    <INPUT name="lng" size="5">
118    <BR>
119  </FORM>
120  <DIV ID="risposta"></DIV></P>
121  <DIV ID="mappa" STYLE="position:absolute;left:25%;top:19%;height:500px;width:400px"></div>
122  </BODY>
```

La pagina elabora.php, come detto in precedenza, è stata descritta nella Lezione 2 dell'Unità di Apprendimento 3 e tratta l'interrogazione del database e la restituzione dei dati richiesti.

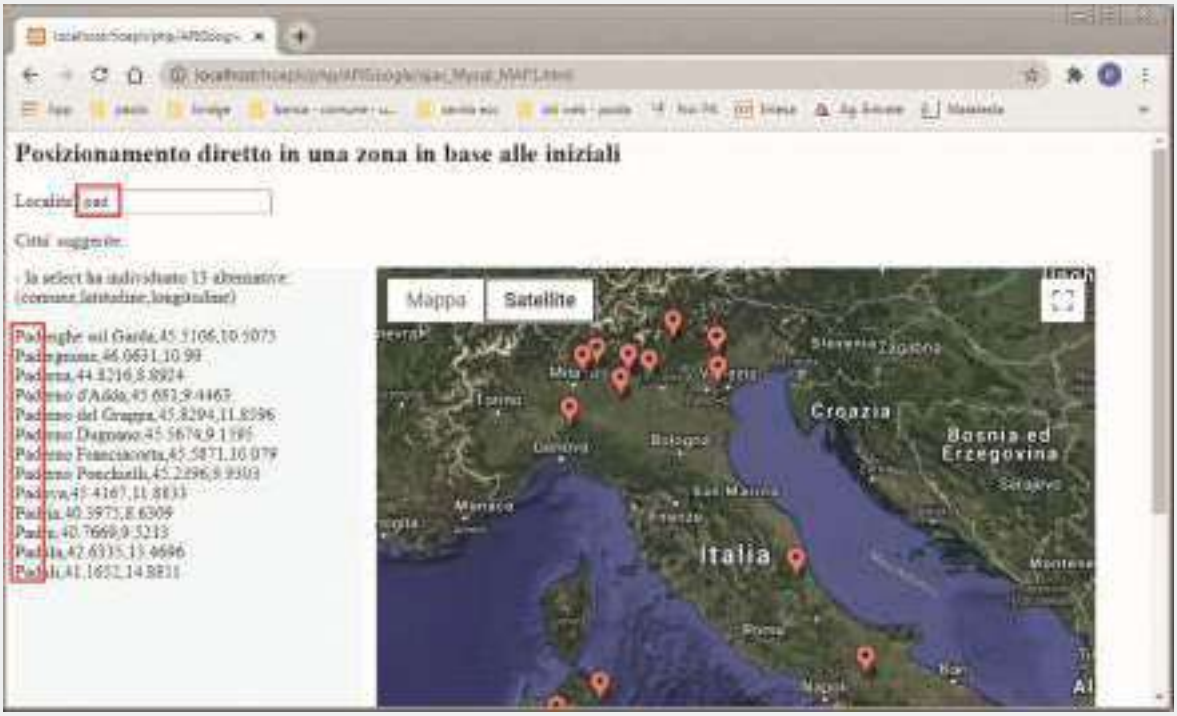
# 7 ESERCIZI IN LABORATORIO

## METTITI ALLA PROVA

• Utilizzare le classi Map, Marker, event, InfoWindow • Utilizzare gli ascoltatori di evento • Utilizzare PHP • Utilizzare il codice SQL di MySQL

**Apri i file ajax_Mysqli.html e elabora.php**

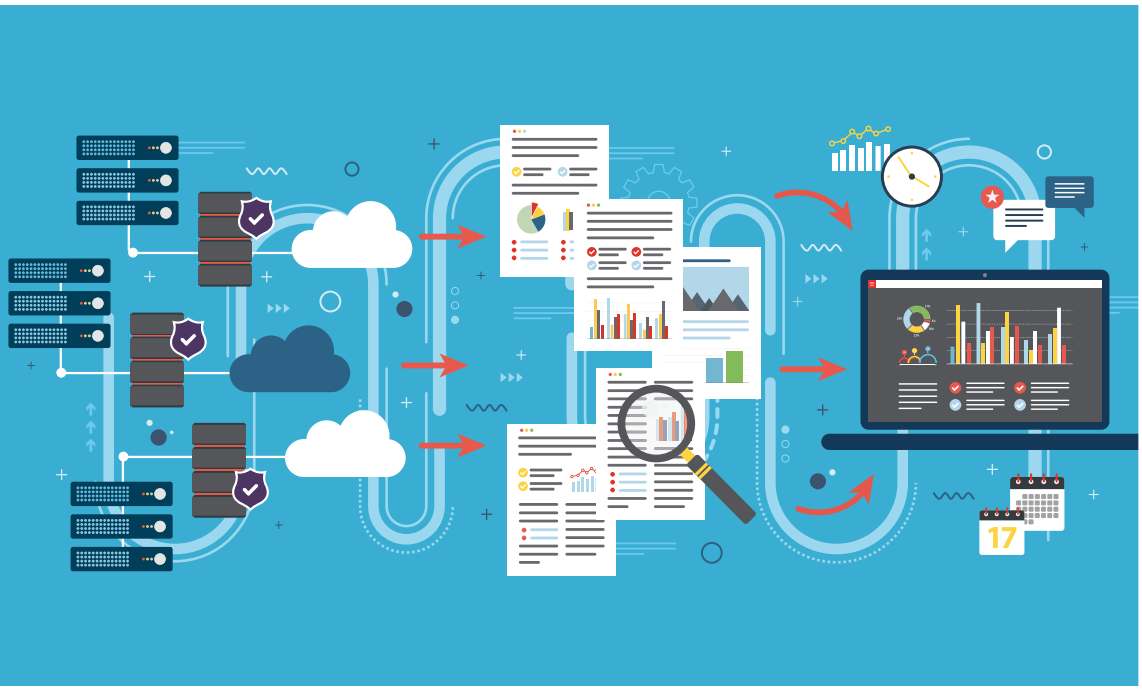
**1** Completa lo script descritto nell'Esercizio di Laboratorio 2 dell'Unità di Apprendimento 3 dove l'utente viene assistito da un meccanismo simile a Google suggest per ricercare una località all'interno della tabella comuni del database provephp: integra quel programma aggiungendo la visualizzazione delle località selezionate sulla cartina geografica utilizzando Google Maps.




**2** Successivamente modifica il codice in modo che facendo click sulla mappa vengano lette le coordinate geografiche del punto selezionato e venga mostrato in una InfoWindow il nome del comune che comprende tale posizione.

# Nuove tipologie di database: i database NoSQL

UNITÀ 7




**LEZIONE 1**
**NoSQL: una nuova proposta di database**

**Esercizi in Laboratorio**

- **L1** Firebase: caratteristiche di base e creazione degli archivi
- **L2** Utilizzare Firebase in JSP

## CONOSCENZE

- Avere chiaro il concetto di database NoSQL
- Conoscere il concetto albero/documento
- L'evoluzione del modello SQL
- Le caratteristiche di MongoDB
- Le caratteristiche di Cassandra
- Le caratteristiche di Firebase

## COMPETENZE

- Individuare i benefici delle tecnologie NoSQL
- Saper scegliere la tipologia di database opportuna alle diverse applicazioni
- Saper creare un database Firebase
- Importare/esportare dati da/in un file JSON in Firebase
- Realizzare script JavaScript per utilizzare Firebase

## ABILITÀ

- Creare e definire un database ad albero
- Utilizzare i dati di Firebase all'interno delle proprie applicazioni
- Realizzare applicazioni con JavaScript per gestire archivi Firebase
- Saper effettuare il deployment di un'applicazione JavaScript che utilizza Firebase
- Saper gestire la connessione ai database con operazioni CRUD

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

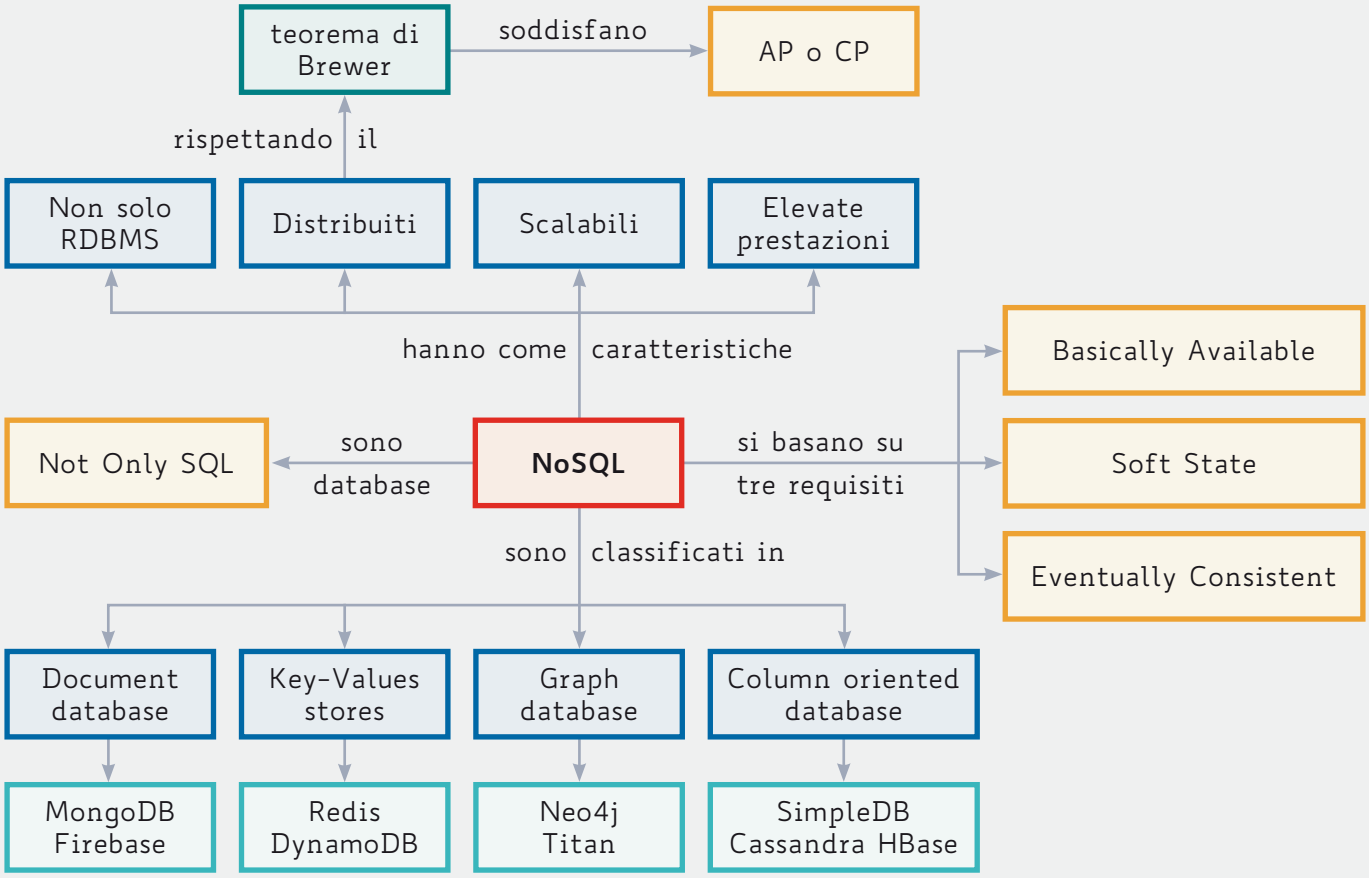
# 1 NoSQL: una nuova proposta di database

**IN QUESTA LEZIONE IMPAREREMO...**

- i database NoSQL: una nuova proposta
- confronto tra SQL vs NoSQL
- le tipologie di database NoSQL
- alcuni esempi: MongoDB, Cassandra, Firebase

## MAPPA CONCETTUALE

didattica inclusiva




## Introduzione

Per far fronte alle nuove tendenze dell'informatica, dominate in questi ultimi anni dai Social Network e dal Cloud Computing, e per superare i limiti del modello relazionale sono state fatte alcune proposte alternative, e tra di esse quelle che stanno riscuotendo i maggiori consensi appartengono alla famiglia chiamata NoSQL.

NoSQL significa "**N**ot **o**nly **S**QL" in quanto i DBMS progettati sotto questo "paradigma" non si oppongono a SQL, anzi, "prelevano" da SQL alcune funzionalità integrandole nel nuovo database che si propone di superare "precedenti vecchi limiti" in termini di elasticità e scalabilità.

Sappiamo che oggi connessi a Internet non ci sono solo PC e smartphone: sono oltre 14 milioni di "cose" connesse, tra fabbriche, aziende agricole, ospedali e warehouses (Internet of Things): le auto parlano con Internet, si inviano/ricevono continuamente dati su ambiente, posizioni geografiche, spostamenti, temperature, meteo da oltre 50 bilioni di sensori, e per questi dati, che devono essere letti in real-time, il modello relazionale di database "va un po' stretto".

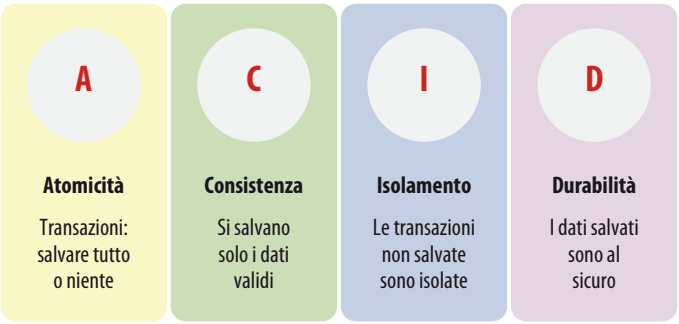
## SQL vs NoSQL

SQL è l'acronimo di Structured Query Language, il linguaggio di programmazione usato per l'interrogazione e la gestione dei database relazionali (RDBMS), quelli cioè che utilizzano il modello logico basato su tabelle e relazioni: per questo motivo, spesso i due termini database relazionali e database SQL, vengono usati come sinonimi. I database relazionali si contraddistinguono quindi per quanto segue:

- l'uso di tabelle e campi per memorizzare i dati;
- le tabelle hanno uno schema fisso: un nome, un elenco di campi, ognuno con i rispettivi tipi (stringhe di caratteri, numeri, date e dati binari);
- le tabelle possono avere una chiave primaria, ossia un campo speciale che identifica univocamente, senza ambiguità, una riga della tabella (per esempio il codice fiscale di una persona);
- i campi di una tabella possono avere dei vincoli, ossia delle condizioni che devono soddisfare (per esempio l'età di una persona non può essere minore di 0);
- tra due o più campi di tabelle (chiavi esterne) può esserci una relazione, ossia una condizione che lega tra loro le rispettive righe a cui i campi appartengono (per esempio tra il codice del comune di nascita di una persona e il codice del comune nella tabella dei comuni); la validità del legame è garantito dai vincoli di integrità referenziale;
- l'accesso ai dati (transazione) viene garantito con le proprietà ACID (Atomicità, Consistenza, Isolamento e Durabilità); questo significa che:
  1. ogni transazione non può essere eseguita parzialmente, ma o solo totalmente con successo (commit) o deve fallire totalmente senza modificare il database (rollback);

     *per esempio se la mia transazione prevede l'inserimento di una riga con 7 campi nel database, alla fine o avrò tutti e 7 i campi inseriti nella riga o nessuna nuova riga;*

  2. alla fine della transazione il database si trova in uno stato consistente, ossia tutti i vincoli dello schema del database sono rispettati, altrimenti la transazione avrebbe fallito: se interrogo il database dopo la transazione vedo che le modifiche sono entrate e sono consistenti;
  3. se eseguo due transazioni in contemporanea l'effetto finale è lo stesso che avrei avuto se avessi eseguito le transazioni una di seguito all'altra, ossia l'effetto temporaneo di una transazione non ancora completata non è visibile alle altre transazioni;
  4. una volta che una transazione è terminata con successo, le modifiche sono diventate persistenti anche in caso di crash del sistema subito dopo la transazione.



NoSQL non è uno specifico linguaggio, ma è il termine universalmente accettato per raggruppare un insieme di tecnologie per la persistenza dei dati che funzionano in modo sostanzialmente diverso dai database relazionali, quindi che non rispettano una o alcune delle caratteristiche indicate sopra.
Nella "famiglia" NoSQL rientrano tipologie di database molto differenti tra loro, con caratteristiche più disparate, che possono magari avere come base le tabelle del modello relazionale ma che permettono di violare i vincoli di integrità referenziale, oppure che hanno campi con sotto-campi o schemi variabili record per record, oppure che consentono di non rispettare le regole per le transazioni, e altro ancora.

Nei database NoSQL la rappresentazione dei dati viene effettuata senza uno schema fisso, cioè vengono implementati in modalità schema-less, dove non è necessario definire i dati in anticipo e che quindi possono continuare a cambiare strutturazione nel tempo.

In sintesi i database NoSQL hanno un modello più morbido e flessibile, che si basa sui seguenti tre principali requisiti:
- Basically Available: l'obiettivo dei database NoSQL è quello di garantire la disponibilità dei dati anche in caso di fallimenti multipli in sistemi distribuiti: i nodi possono essere soggetti a guasti, ma il servizio è sempre disponibile se si adotta un approccio fortemente distribuito;
- Soft State: in alcuni database NoSQL non è garantita la consistenza dei dati in ogni istante: viene lasciato al programmatore l'onere di gestirla;
- Eventually Consistent: viene però garantito il fatto che dopo un certo intervallo di tempo, se le attività di modifica dei dati cessano, il sistema converge a uno stato consistente.

Questa tipologia di database si adatta a molti contesti soprattutto quando si manipolano grandi quantità di dati eterogenei e senza uno schema, mentre non sono adatti nei casi in cui si devono gestire molte relazioni tra i dati e si vuole garantire l'integrità referenziale: in questi casi si utilizza il "buon vecchio" modello relazionale, come nella applicazioni ERP.

Sinteticamente, principali caratteristiche dei database NoSQL:
- Non relazionali: essendo i database NoSQL schema-less consentono di memorizzare "on the fly" attributi, anche senza averli definiti a priori: sono quindi particolarmente adatti a memorizzare dati di sistemi fortemente dinamici.
- Flessibilità: i database NoSQL sono la soluzione ideale per archiviare dati semistrutturati e non strutturati in quanto hanno schemi flessibili che permettono uno sviluppo più veloce e iterativo.
- Distribuiti: grazie alla loro flessibilità è possibile distribuire su più nodi lo storage in modo da realizzare potenti sistemi "fault tolerant".
- Scalabili orizzontalmente: i database NoSQL in genere sono progettati per il dimensionamento orizzontalmente con architetture che consentono di memorizzare e gestire una grande quantità di informazioni usando cluster distribuiti di hardware.
- Elevate prestazioni: il progettista sceglie il tipo di database NoSQL specifico per l'applicazione che deve realizzare, in modo da avere sistemi ottimizzati con modelli di dati specifici e schemi di accesso che consentono prestazioni più elevate rispetto all'utilizzo di database relazionali.

I database NoSQL sono una soluzione ideale per molte applicazioni moderne, quali dispositivi mobili, Web e videogiochi che richiedono database flessibili, scalabili, con prestazioni elevate e altamente funzionali per offrire un'esperienza utente eccezionale.

Inoltre sposano tutti la filosofia open source, che è alla base del movimento NoSQL.

## Il teorema di Brewer e i database NoSQL

Ricordiamo il teorema di Eric Brewer sui database distribuiti, noto come CAP Theorem.

**CAP Theorem**
È impossibile per un sistema informatico distribuito fornire contemporaneamente tutte e tre le caratteristiche CAP. (*Eric Brewer*).

In questo teorema Brewer sostiene che un sistema distribuito è in grado di supportare solamente due tra le seguenti caratteristiche:

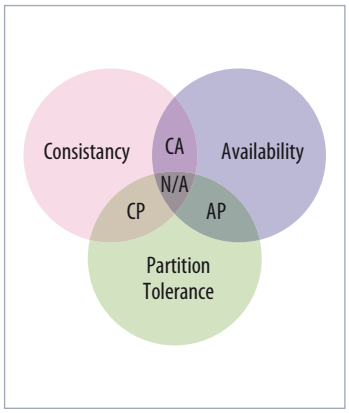
- Consistency (Coerenza): tutti i nodi vedono lo stesso dato allo stesso tempo;
- Availability (Disponibilità): ogni operazione deve sempre ricevere una risposta;
- Partition Tolerance (Tolleranza di partizione): capacità di un sistema di essere tollerante a un'aggiunta, una rimozione di un nodo nel sistema distribuito o alla non disponibilità di un componente singolo.

Vediamo le possibili combinazioni:

**CASO 1:** rispetto Consistency + Availability (CA)
violazione Partition Tolerance
- come anomalia avremo che il sistema non funziona correttamente in caso di perdita di messaggi: è il caso dei database RDBMS.

**CASO 2:** rispetto Availability + Partition Tolerance (AP)
violazione Consistency
- come anomalia avremo che il sistema avrà repliche del dato non aggiornate (Dynamo, Voldemort, CouchDB, Riak, Cassandra).

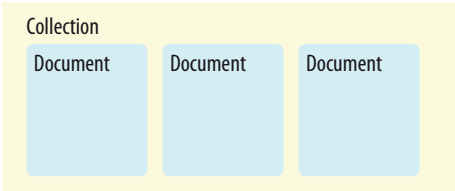
**CASO 3:** Consistency + Partition Tolerance (CP)
violazione Availability
- come anomalia avremo che una query potrebbe non produrre risposta (Redis, MongoDB, Hypertable, HBase, BigTable).



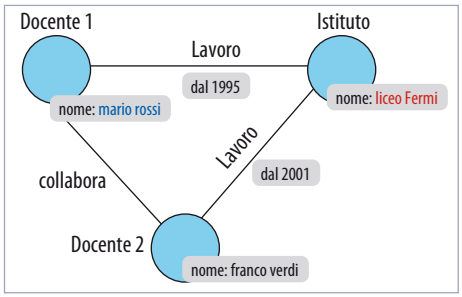

## Tipologie di database NoSQL

I database NoSQL sono classificati in base al tipo di modello che utilizzano per la memorizzazione dei dati; le principali famiglie alternative al modello relazionale sono riportate di seguito.

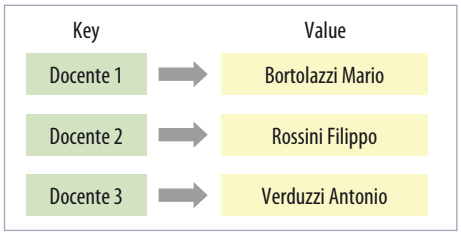
- Database a documenti (Document database): memorizzano i dati in documenti, ossia in collezioni di oggetti complessi, codificati in qualche modo e senza uno schema rigido, cioè ognuno dei quali richiede una struttura fissa e può essere costituito da più campi che possono essere anche differenti l'uno dall'altro nell'ambito della stessa collezione.
  I database di documenti semplificano agli sviluppatori la ricerca e la memorizzazione di dati, usando lo stesso formato di modello di documento che usano nel codice dell'applicazione.
  MongoDB è basato proprio su questo modello e, secondo le stime, è uno dei database NoSQL più diffusi al mondo; anche Firebase ne è un esempio, dove tutti i dati sono inseriti in oggetti proprio denominati documenti (*document*) che a loro volta possono contenere documenti *(quindi hanno una struttura ricorsiva),* che sono raggruppati in raccolte *(collection) che* costituiscono ciascuna un nodo di un albero JSON.

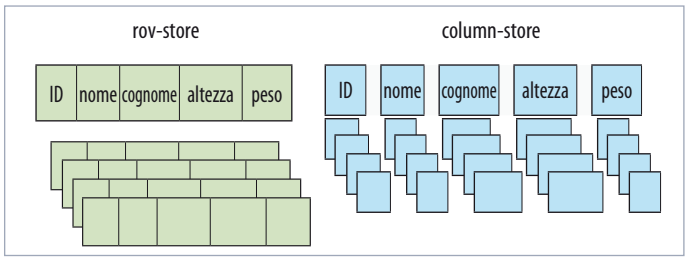



- Database a grafo (Graph database): lo scopo di un database a grafo è quello di facilitare la creazione e l'esecuzione delle applicazioni che operano con set di dati a elevata connessione usando strutture a grafo, che è una delle strutture dati più duttili dell'informatica, con relazioni libere tra nodi stessi e, quindi, non prefissate come nel caso dei database relazionali. I due elementi principali della struttura sono i vertici e gli archi: i primi contengono le informazioni, i secondi stabiliscono relazioni e ne sono un esempio Neo4j e Titan.




- Database chiave-valore (Key-Values stores): utilizzano il modello dell'array associativo e sono ispirati dalle strutture dati conosciute come mappe o dizionari, immagazzinano oggetti associando loro una chiave univoca che ne permetterà un agevole recupero. Redis, Memcached e DynamoDB sono i prodotti più noti di questo tipo. Il modello di dati chiave-valore è particolarmente adatto per casi d'uso quali i videogiochi, le tecnologie pubblicitarie e l'IoT.






- Database orientati alle colonne (Column-oriented database): sono database che immagazzinano dati in sezioni di colonne offrendo una rapida di aggregazione che è necessaria sia ai produttori di motori di ricerca sia ai social network: colossi come Facebook, Google e Amazon sfruttano infatti soluzioni di questo tipo utilizzando prodotti come Big Table, SimpleDB, Cassandra e HBase: questi ultimi offrono modelli ancora più complessi in quanto fanno corrispondere alle chiavi, valori formati da famiglie di colonne anch'esse indicizzate.
  Nei database a righe ogni riga contiene una chiave primaria, mentre nei database a colonne ogni colonna è scritta in un file diverso, come mostrato nella figura.

## Vantaggi e svantaggi dei due modelli

Non è facile valutare vantaggi e svantaggi dei database NoSQL rispetto ai RDBMS, proprio perché ogni database NoSQL merita un discorso a parte. Inoltre, poiché il database NoSQL viene scelto volta per volta in base alle esigenze e al contesto dell'applicazione che si deve implementare, verrà individuato quello più vicino al modello dei dati dell'applicazione, in modo da garantire migliori prestazioni del modello relazionale.

### ESEMPIO

Se dobbiamo implementare un'applicazione che dovrà gestire molte relazioni tra oggetti di vario tipo, un modello logico di dati come quello a grafo può rendere l'applicazione molto più semplice da sviluppare.

Tra gli svantaggi più significativi dei database NoSQL possiamo riportare la carenza di tool: per una tecnologia consolidata da molto tempo come SQL, esistono tantissimi strumenti di gestione e sviluppo, a differenza di quanto accade per la maggior parte dei database NoSQL.
Nell'approccio NoSQL come vantaggi principali dobbiamo evidenziare: la strutturazione dei dati "schema-less" che conferisce una maggiore centralità alle informazioni e alla loro varietà, la scalabilità, dato che i NoSQL sono generalmente basati su strutture fisiche che meglio si prestano alla distribuzione dei dati su più nodi di una rete, le migliori prestazioni, anche grazie alla maggiore distribuibilità dei dati sulle reti, la flessibilità nella progettazione, che lascia liberi i programmatori di risolvere eventuali casi particolari direttamente in fase di sviluppo dell'applicazione.
Riportiamo una tabella che confronta i database relazionali con quelli NoSQL.

| | DATABASE RELAZIONALI | DATABASE NoSQL |
|---|---|---|
| **Carichi di lavoro ottimali** | I database relazionali sono progettati per le applicazioni. | I database NoSQL sono progettati per molti schemi di accesso ai dati che includono le applicazioni a bassa latenza. |
| **Modello di dati** | Il modello relazionale normalizza i dati in tabelle che sono costituite da righe e colonne. Il database attua l'integrità referenziale nelle relazioni tra le tabelle. | I database NoSQL offrono una varietà di modelli di dati come chiave-valore, documento e grafo, ottimizzati per prestazioni e scalabilità. |
| **Proprietà ACID** | Gli RDBMS tradizionali supportano le proprietà ACID di un database relazionale: atomicità, coerenza, isolamento e durabilità. | I database NoSQL spesso rinunciano ad alcune proprietà ACID dei database relazionali a favore di un modello di dati più flessibile e scalabile orizzontalmente, rendendo i database NoSQL favoriti per i casi d'uso a throughput elevato e bassa latenza che richiedono scalabilità orizzontale. |
| **Prestazioni** | Le prestazioni dipendono generalmente dal sottosistema di dischi. Per raggiungere il massimo livello di prestazioni, spesso è necessario ottimizzare query, indici e struttura delle tabelle. | Le prestazioni si calcolano generalmente in funzione delle dimensioni del cluster dell'hardware, della latenza di rete e dell'applicazione di chiamata. |

| | DATABASE RELAZIONALI | DATABASE NoSQL |
|---|---|---|
| **Scalabilità** | La scalabilità, nei database relazionali, avviene tipicamente in verticale, incrementando le capacità di elaborazione dell'hardware, oppure in orizzontale aggiungendo repliche per i carichi di lavoro di sola lettura. | I database NoSQL sono in genere partizionabili perché gli schemi di accesso sono in grado di ridimensionarsi in orizzontale usando l'architettura distribuita per aumentare il throughput che offre prestazioni coerenti quasi senza alcun limite di scalabilità. |
| **API** | Le richieste di archiviazione e ripristino dati vengono comunicate tramite query conformi al linguaggio SQL analizzate ed eseguite dal database relazionale. | Le API basate su oggetti consentono agli sviluppatori di app di archiviare e ripristinare facilmente strutture di dati: le chiavi di partizione permettono di cercare coppie chiave-valore, serie di colonne o documenti contenenti oggetti e attributi. |
| (*fonte:* Amazon) | | |

## Alcuni esempi di database NoSQL

Descriviamo ora sinteticamente alcuni tra i database NoSQL più utilizzati.

### MongoDB: caratteristiche di base

MongoDB è un database orientato ai documenti, ognuno dei quali è memorizzato nel formato JSON; il documento è sostanzialmente un albero che può contenere molti dati, anche annidati.
Un esempio di documento è quello riportato di seguito, dove riportiamo i dati e le opere di Dante Alighieri.

```
[
  "nome": "Dante", "cognome": "Alighieri",
  "nato": 1265,  "morto": 1321,
  "lingue": ["italiano", "latino" ],
  "opere": [
      { "titolo": "Divina Commedia",
        "iniziata": 1304,
        "lingua": "volgare fiorentino",
        "genere": "poema",
        "forma": "poesia",
        "versi": "endecasillabi",
        "canti": 100,
        "cantiche": ["Inferno", "Purgatorio", "Paradiso"]
      },
      { "titolo": "Convivio",
      }
  ]
]
```

Possiamo vedere come non c'è uno schema fisso per i documenti, che sono raggruppati in collezioni che, a loro volta, possono essere anche eterogenee.

Sinteticamente:
- *non c'è uno schema fisso per i documenti;*
- *non esiste un concetto analogo al vincolo di integrità referenziale.*

Caratteristiche principali di MongoDB:
- è stato concepito in origine come database distribuito, l'alta disponibilità, la scalabilità orizzontale e la distribuzione geografica sono quindi native e facili da usare;
- consente servizi di alta disponibilità, dal momento che la replicazione di un database (Replica Set) può avvenire in modo molto semplice;
- garantisce la scalabilità automatica, ossia la possibilità di distribuire (Sharding) le collezioni in cluster di nodi, in modo da supportare grandi quantità di dati senza influire pesantemente sulle performance;
- le query ad hoc, l'indicizzazione e l'aggregazione in tempo reale offrono efficaci modalità di accesso e analisi dei dati.

MongoDB è utilizzabile gratuitamente: le versioni rilasciate prima del 16 ottobre 2018 sono pubblicate sotto licenza AGPL, mentre quelle successive, sempre libere, sono rilasciate sotto licenza Server Side Public License (SSPL).

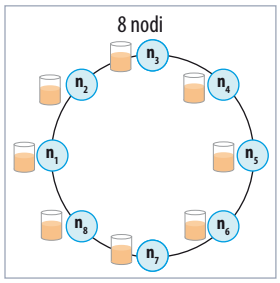
Tra gli svantaggi possiamo indicare che:
- essendo un database a documenti che possono essere incorporati o collegati senza vincoli di integrità, non esistono relazioni tra le tabelle e viene quindi a mancare la coerenza dei dati;
- mentre nei database come MySQL possiamo delegare al DBMS la validazione dei dati, come per esempio il tipo di un dato, la sua lunghezza, la sua validità (si pensi al tipo DATE ecc.) in MongoDB ciò non è possibile e tutte le operazioni di validazione devono essere fatte dall'applicazione, prima di procedere al salvataggio del dato;
- non è sempre di facile attuazione la migrazione tra un sistema esistente e un nuovo progetto che utilizza MongoDB.

## Cassandra: Column Family Stores

Un esempio della famiglia dei Column Family è Cassandra: in questo database i dati vengono memorizzati in modo ottimizzato per effettuare la ricerca su colonne e sono progettati per gestire enormi quantità di dati distribuiti su più server.

Cassandra è stato sviluppato nel 2008 all'interno di Facebook, per la gestione della messaggistica, e l'anno successivo è entrato a far parte del progetto di Apache Software Foundation.

Tra le sue caratteristiche quella più importante è, senza dubbio, che ha un sistema scalabile in orizzontale e che è in grado di offrire la possibilità di gestire grandi quantità di dati con prestazioni eccezionali; infatti, l'architettura si basa su un design dove tutti i nodi partecipanti sono identici e comunicano tra loro come peer: non avendo un'architettura master-slave nessun nodo prevale sull'altro e questo fatto favorisce la scalabilità con l'aggiunta incrementale di nodi senza necessità di riconfigurazione.




I cluster in Cassandra sono anche detti ring per l'architettura ad anello che li caratterizza.

Cassandra è basato su quattro concetti chiave.

1 Cluster: è una raccolta di nodi disposti in un'architettura ad anello; è anche detto data-center e deve avere un nome unico, che verrà successivamente utilizzato dai diversi nodi; il cluster può contenere una o più entità di livello inferiore, i keyspace.

2 Keyspace: un DB relazionale è una collezione di tabelle, e in Cassandra l'equivalente è il keyspace, alla sua creazione viene definita anche una strategia di replica dei dati:
– cluster unico: SimpleStrategy;
– più cluster: NetworkTopologyStrategy.

Nel primo caso non è necessario assegnare il nome del data-center in quanto è unico.
Il numero n indica la molteplicità di replica dei dati presenti nel cluster.

```
Create Keyspace <nome> With Replication = {'class' : <strategiaReplica>, [nome_data-center : n };
```

Il keyspace è un namespace che definisce come i dati vengono replicati nei nodi.

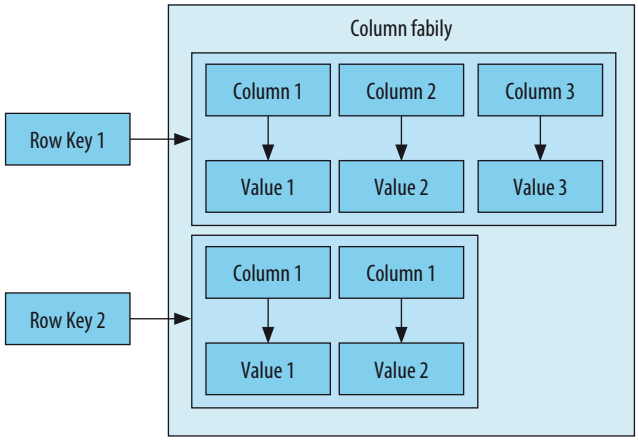
3 Column Family: è il contenitore delle colonne, anche chiamato supercolum, ed equivale alla tabella nei database relazionali: ogni famiglia viene salvata in un file separato e contiene una raccolta di righe rappresentate da una mappa:

```
<RowKey, SortedMap <ColumnKey, ColumnValue >>.
```

Attraverso la chiave si accede all'insieme dei dati correlati.

**4** Column: una colonna è una struttura dati composta da righe, definita attraverso un nome, per ciascuna riga contiene:
– un nome di colonna, che consiste in una key (Row + Column Family);
– un valore;
– un timestamp.

A differenza del modello relazionale, in ciascuna riga può variare il numero di colonne.




Mettiamo a confronto le quattro entità di Cassandra con gli equivalenti del modello relazionale.

| NOME | DEFINIZIONE BREVE | ANALOGIA RDBMS |
|---|---|---|
| Keyspace | Una collezione di Column Family | Database |
| Column Family | Un insieme di Row | Tabella |
| Row | Un insieme ordinato di Column | Riga |
| Column | Una coppia chiave-valore con timestamp associato | Colonna |

Vediamo un semplice esempio.

```
Create table PlaylistMusicale
(
  IDCanzone int,
  NomeCanzone text,
  Anno int,
  Cantanter text,
  Primary key((IDCanzone, Anno), NomeCanzone)
);
```

Viene creata la tabella PlaylistMusicale, dove IDCanzone e Anno sono la chiave di partizione (ColumnKey) e NomeCanzone è la colonna del clustering (ColumnValue).
I dati saranno raggruppati sulla base di NomeCanzone.
In questa tabella, ogni anno, verrà creata una nuova partizione in modo che tutte le canzoni dell'anno saranno sullo stesso nodo e la chiave primaria composta favorisce un recupero dei dati molto veloce.

## Firebase: un database e un backend

Firebase è un BaaS (BackEnd as a Service) che fornisce a sviluppatori di app Web o mobile un modo per collegare le loro applicazioni a un backend cloud storage e API esposte da applicazioni backend.

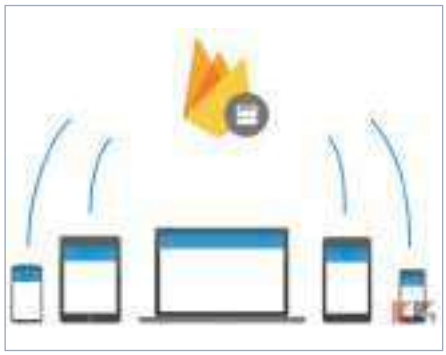
I backend sono applicazioni server in grado di rendere disponibili, tramite API, servizi come l'autenticazione degli utenti, lo storage dei dati, le push notification, la comunicazione tra utenti ecc.

È complesso realizzare un "backend fai da te" anche perché sono necessarie competenze ed esperienza in ambito server che non tutti i programmatori posseggono, e, inoltre, è complesso organizzare e sincronizzare lo sviluppo coordinato di componenti server e client: è meglio utilizzare backend presenti in commercio, come per esempio Firebase.
Firebase è una piattaforma di sviluppo Web e mobile nata nel 2011 e successivamente acquisita da Google nel 2014: offre la possibilità di semplificare numerose operazioni e mette a disposizione molteplici servizi facilmente utilizzabili in qualunque progetto, dal sito Web alle applicazioni Android o iOS.

Essendo un servizio online, permette di salvare e sincronizzare i dati elaborati da applicazioni Web e mobile rendendoli disponibili in tempo reale.

Mette inoltre a disposizione un'importante opzione di analisi dati per "studiare" i dati in modo da trarne le informazioni necessarie, come, per esempio, veicolare le pubblicità personalizzate e massimizzare i profitti.
In Firebase sono presenti due versioni di un database NoSQL a documenti:
- Cloud Firestore;
- Realtime Database.

Realtime Database è un gigantesco albero JSON, mentre Cloud Firestore è un po' più strutturato con i dati costituiti da:
- documenti, composti da campi strutturati nella coppia valore-chiave;
- collezioni, che sono raccolte di documenti.



I documenti spesso rimandano a sotto-raccolte, che contengono altri documenti, che a loro volta possono contenere altri documenti e così via.
L'esempio che segue riporta la corrispondenza tra un file JSON e un archivio Firebase dove abbiamo memorizzato libri raccolti per casa editrice, che a loro volta sono composti da raccolte di documenti per singolo autore, in cui sono raccolti i relativi libri ecc.

**ALBERO JSON**

```
1  {
2    "libri" : {
3      "AdrianoSalaniEditore" : {
4        "J K Rowling " : {
5          "Harry Potter e la pietra filosofale" : {
6            "genere" : "fantasy",
7            "prezzo" : 18
8          }
9        }
10     },
11     "hoepli" : {
12       "Camagni Paolo" : {
13         "Algoritmi I" : {
14           "anno" : 2006,
15           "argomento" : "informatica",
16           "prezzo" : 22
17         },
18         "Tesi di laurea" : {
19           "anno " : 1984,
20           "argomento" : "robotica industriale"
21         }
22       }
23     }
24   }
25 }
```

**DATABASE FIREBASE**

```
tpoit-16d0b
- libri
  - AdrianoSalaniEditore
    - "J K Rowling "
      - Harry Potter e la pietra filosofale
        - genere: "fantasy
        - prezzo: 18
  - hoepli
    - Camagni Paolo
      - Algoritmi I
        - anno: 2006
        - argomento: "informatica
        - prezzo: 22
      - Tesi di laurea
        - "anno ": 1984
        - argomento: "robotica industrial
```

Realtime Database usufruisce della sincronizzazione dati ogni millisecondo e garantisce così assoluta velocità e altissime performance: con esso è quindi possibile realizzare con facilità un servizio di messaggistica istantanea esattamente come quello di WhatsApp, accessibile sia da mobile che da Web.
Negli Esercizi in Laboratorio che seguono viene descritto come creare un database Firebase e come utilizzarlo in una semplice applicazione CRUD scritta in JavaScript.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Il nome NoSQL significa:**
**a** Not SQL
**b** Not only SQL
**c** Near only SQL
**d** Not or SQL

**2 L'accesso ai dati viene garantito con le proprietà ACID:**
**a** solo nei RDBMS
**b** solo nei db NoSQL
**c** sempre in entrambi
**d** in nessuno è obbligatorio

**3 Il tre principali requisiti dei db NoSQL:**
**a** Schema-less X
**b** Basically Available
**c** Soft state
**d** Eventually Consistent

**4 Le tre le caratteristiche del teorema di Brewer sono:**
**a** **C**omputing, Availability, **P**artition Tolerance
**b** **C**onsistency , Availability, **P**artition Tolerance
**c** **C**onnection, Availability, **P**rojection Tolerance
**d** **C**onsistency , Analitics, **P**rojection Tolerance

**5 Tra le tipologie di database NoSQL troviamo:**
**a** Document database
**b** Graph database
**c** Tree database
**d** Column-oriented database
**e** Key-Values stores
**f** Row-oriented database

**6 I principali svantaggi dei database NoSQL sono:**
**a** il costo del prodotto
**b** la carenza di tool
**c** la mancanza delle operazioni di query
**d** la poca flessibilità nelle applicazioni distribuite

## VERO/FALSO

**1** NoSQL è anche uno specifico linguaggio per la gestione dei dati. V F
**2** I database NoSQL sposano tutti la filosofia open source. V F
**3** I database NoSQL sono difficilmente scalabili. V F
**4** I database NoSQL sono particolarmente indicati nei sistemi distribuiti sui dispositivi mobili. V F
**5** I database NoSQL non soddisfano la caratteristica di essere consistenti. V F
**6** Casandra rientra tra i NoSQL Column-oriented database. V F
**7** Firebase rientra tra i NoSQL Column-oriented database. V F

# 1 ESERCIZI IN LABORATORIO

## FIREBASE: CARATTERISTICHE DI BASE E CREAZIONE DEGLI ARCHIVI

Per poter utilizzare i servizi di Google è necessario avere un account, che può essere creato direttamente all'indirizzo https://accounts.google.com: è sufficiente attivare gratuitamente una casella di posta elettronica con Gmail.
Successivamente, per prima cosa, è necessario avere un account a Firebase che si può ottenere all'indirizzo https://console.firebase.google.com/: per la sperimentazione è sufficiente attivare un Free Plan, totalmente gratuito, che ci permette di imparare a interagire con il database.

Dopo esserci autenticati, creiamo il nostro progetto in tre passaggi caricando la pagina https://console.firebase.google.com/u/0/:

1. Come primo passo diamo un nome al progetto, nel nostro caso TPSIT, confermiamo l'accettazione dei termini e procediamo cliccando su Continua:

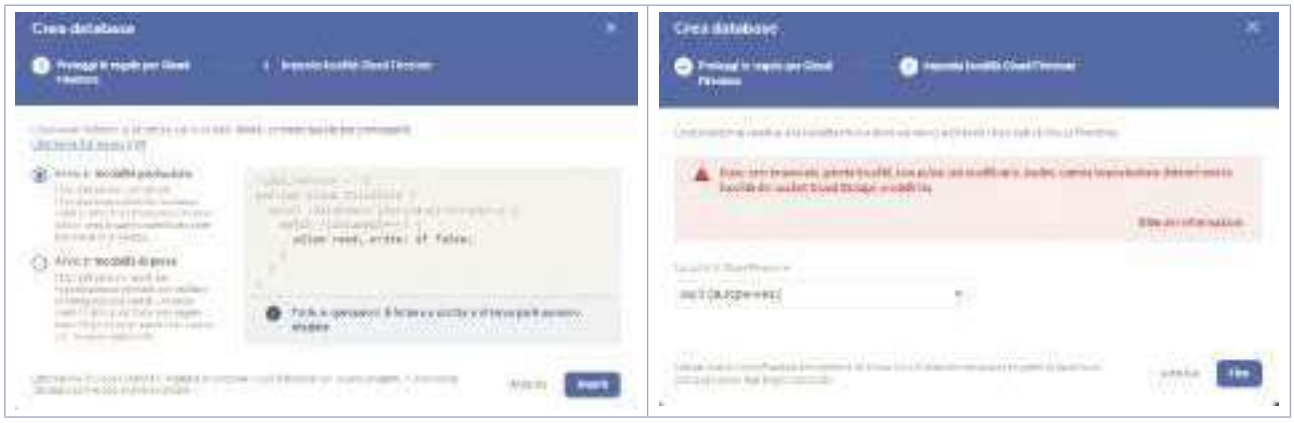
All'account creato sarà associato un indirizzo, nel formato *<username>.firebaseio.com* ed è quello che noi utilizzeremo nel seguito degli esempi: **tpsit-15d0b**.*firebaseio.com*.
Questo URL viene utilizzato all'interno del codice come riferimento alla base di dati remota e, tramite browser, per visualizzare un pannello che permette l'accesso visuale.

# ESERCIZI IN LABORATORIO 1

2. Il secondo passo consente di abilitare Google Analytics, un servizio *"di analisi gratuita e illimitata che abilita il targeting, il reporting e molto altro"*, che per ora a noi non serve, ma ne confermiamo l'abilitazione.
3. Anche il terzo passo è relativo alla configurazione di Google Analytics: selezioniamo la nazione di appartenenza e procediamo con la creazione del progetto.
   Aspettiamo qualche istante, in modo che venga definito il nostro repository, e alla sua terminazione viene visualizzato il messaggio mostrato nella figura a lato.

All'interno del nostro progetto ora possiamo definire il database:

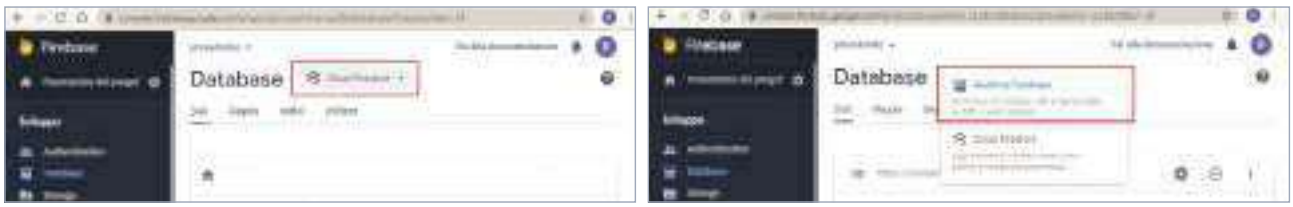
Nella schermata successiva confermiamo la prima opzione e indichiamo la regione del mondo dove far fisicamente memorizzare i nostri dati:



Firebase mette a disposizione due possibili alternative/tipologie di database:

- Cloud Firestore;
- Realtime Database.

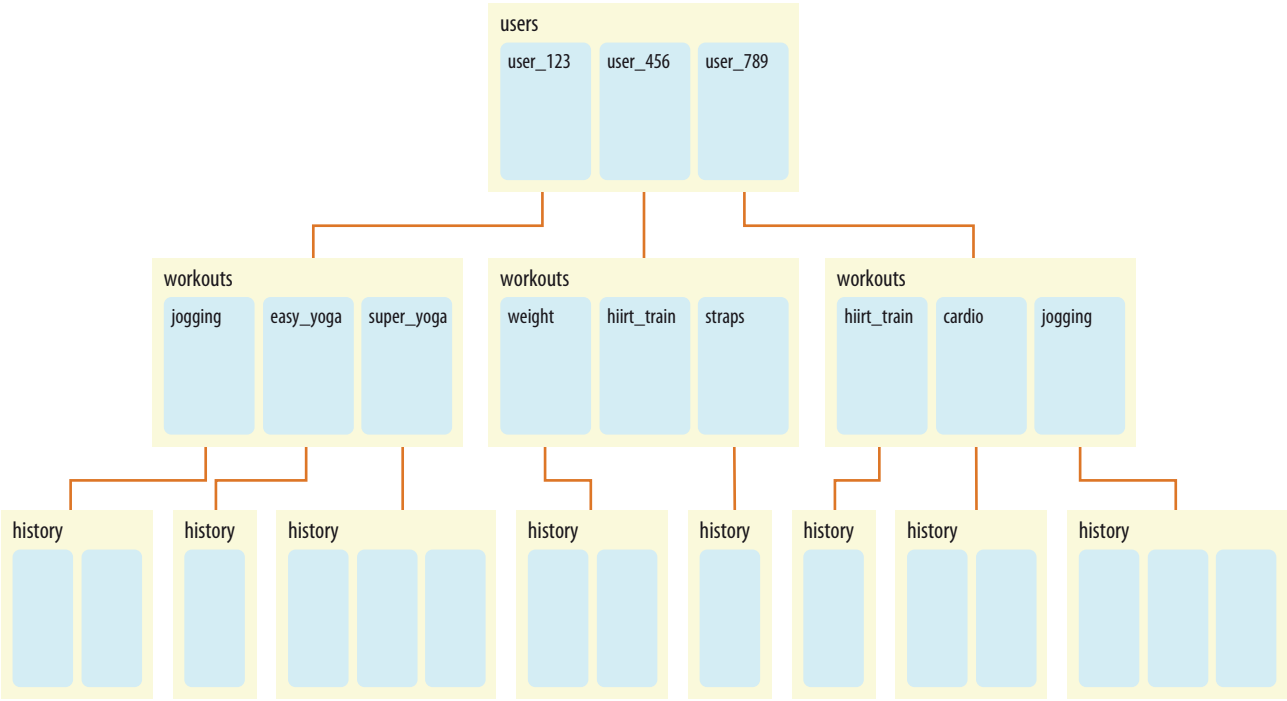
La scelta viene effettuata dall'apposita casella di selezione:



# 1 ESERCIZI IN LABORATORIO

Realtime Database è un gigantesco albero JSON, mentre Cloud Firestore è un po' più strutturato con i dati costituiti da:
- documenti, composti da campi strutturati nella coppia valore-chiave;
- collezioni, che sono raccolte di documenti.

I documenti spesso rimandano a sotto-raccolte, che contengono altri documenti, che a loro volta possono contenere altri documenti e così via.
La strutturazione dei dati permette di effettuare "query superficiali", cioè di effettuare la richiesta di un documento senza necessariamente "prendere" anche tutti i dati sottostanti: questa possibilità offre l'opportunità di realizzare archiviazioni gerarchiche nel modo più aderente alla loro strutturazione logica, senza doversi preoccupare di mantenere il database poco profondo e di poter effettuare query più potenti.




Cloud Firestore è maggiormente scalabile rispetto al Realtime Database: osserviamo come le query vengono ridimensionate in base alle dimensioni del set di risultati, non al set di dati, in modo da mantenere velocità nelle ricerche, indipendentemente dalla grandezza del set di dati.
Anche il recupero manuale dei dati è più semplice e ha il supporto multiregione garantendo una maggiore affidabilità, in quanto i dati vengono condivisi su più data-center contemporaneamente.

Per l'utilizzo commerciale sono previsti due modelli diversi per la determinazione del prezzo:
**1** Realtime Database si basa principalmente sulla memoria o sulla larghezza di banda della rete;
**2** Cloud Firestore si basa principalmente sul numero di operazioni che si esegue.

Naturalmente la scelta deve essere effettuata situazione per situazione, in base al progetto che ci si appresta a realizzare:
- Cloud Firestore sarà favorevole, dal punto di vista dei prezzi, nei progetti di app di notizie, di giochi multiplayer a turni o community di sviluppatori che dialogano tra loro;
- Realtime Database sarà favorevole per app con interazioni di gruppo in tempo reale in cui sono richiesti invii e aggiornamenti a più persone.

# ESERCIZI IN LABORATORIO 1

La scelta di utilizzare Realtime Database viene quindi fatta per i seguenti motivi:

**a** è più economico per le app che effettuano molti e frequenti aggiornamenti;

**b** è più stabile perché è circolazione da molto tempo, mentre ad oggi (2021) Cloud Firestore è appena uscito dalla (versione) beta;

**c** ha una latenza migliore ed è più affidabile per applicazioni in tempo reale.

Vediamo come realizzare un archivio con entrambi.

## Realtime Database

Clicchiamo ora su Realtime Database, che sarà il database aggiornato in tempo reale e condiviso dalle nostre applicazioni; abbiamo tre possibilità di creazione del nostro database:

- manuale;
- importandolo da un file JSON;
- attraverso le applicazioni (in JS, Java, Phyton, php, Android ecc.).

Descriviamo dapprima come avviene la sua creazione manuale e successivamente importiamo un file contenente alcuni dati: in seguito vedremo come manipolare dati con i singoli linguaggi di programmazione.

### Creazione manuale di un archivio

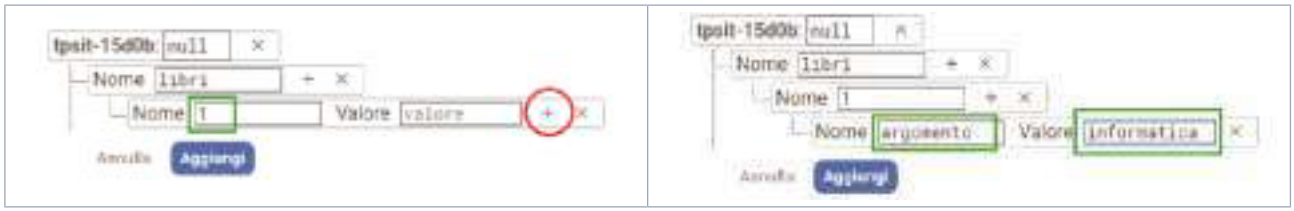
Clicchiamo sull'opzione Realtime Database dopo aver selezionato l'icona che indica il database.

Osserviamo (e memorizziamo) l'indirizzo completo https://tpsit-15d0b.firebaseio.com, che verrà successivamente utilizzato nei programmi per accedere a tale archivio.

Come detto in precedenza, in Realtime Database la struttura ha un formato ad albero: creiamo un archivio che memorizza dati riguardanti libri, partendo quindi dalla definizione del nodo radice libri:

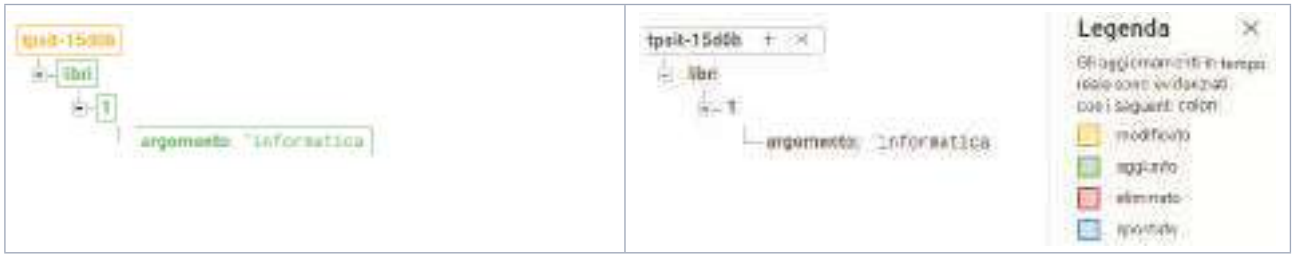
Per definire un nodo è necessario lasciare vuoto il campo Valore.

Inseriamo i dati del primo volume, al quale assegniamo come nodo il valore 1 (che corrisponde all'ID) e, quindi, aggiungendo i singoli campi definiti come coppia nome-valore:



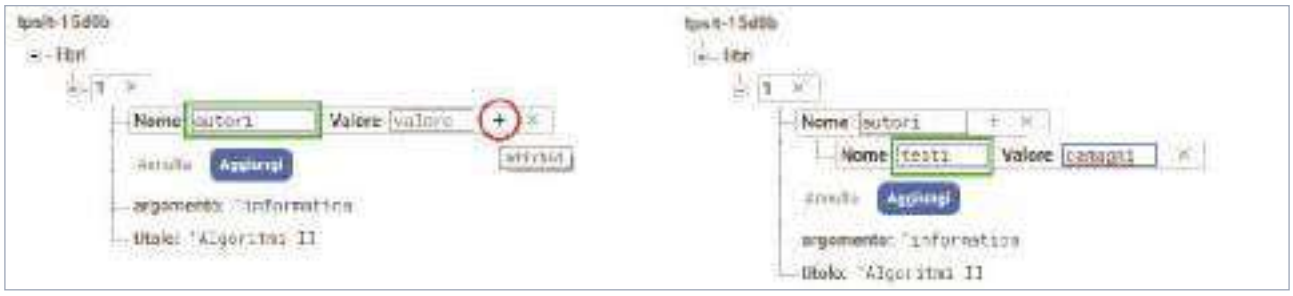
# 1 ESERCIZI IN LABORATORIO

Confermando il primo dato, viene visualizzato "a colori" l'albero che abbiamo creato: riportiamo la legenda dei colori, che ci viene sempre mostrata ogniqualvolta si modifica la struttura e/o il contenuto dell'albero.

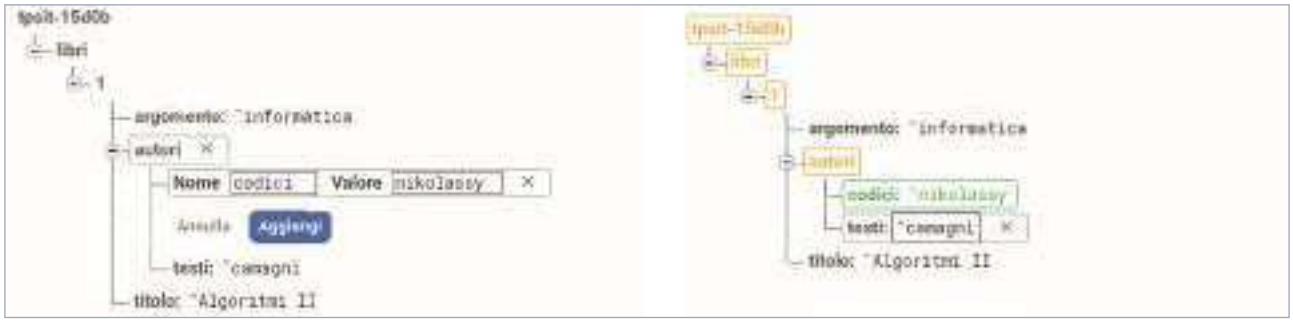



Proseguiamo a inserire gli attributi del primo libro, aggiungendo il titolo che, appena confermato, viene visualizzato in verde, come "aggiunto", e colorando di arancione il resto dell'albero, come "modificato":

Completiamo i dati del primo libro aggiungendo gli autori: creiamo quindi un nuovo livello, dato che gli autori possono essere più di uno, come nel nostro caso, ne inseriamo il primo:



quindi ne aggiungiamo un secondo:



A questo punto abbiamo descritto i passi essenziali che ci permettono di realizzare qualunque archivio.

## METTITI ALLA PROVA

- Creazione manuale di un archivio

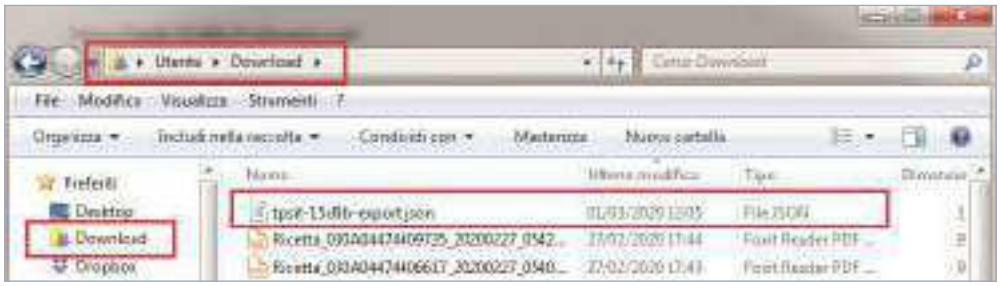
Crea un tuo account e definisci un tuo progetto: quindi replica la struttura appena descritta, inserisci altri due volumi, completando la struttura in modo da aggiungere anche l'editore, con i suoi dati anagrafici e l'anno di edizione.

# ESERCIZI IN LABORATORIO 1

## Backup del database

Prima di procedere, effettuiamo un backup in un file JSON del nostro archivio:

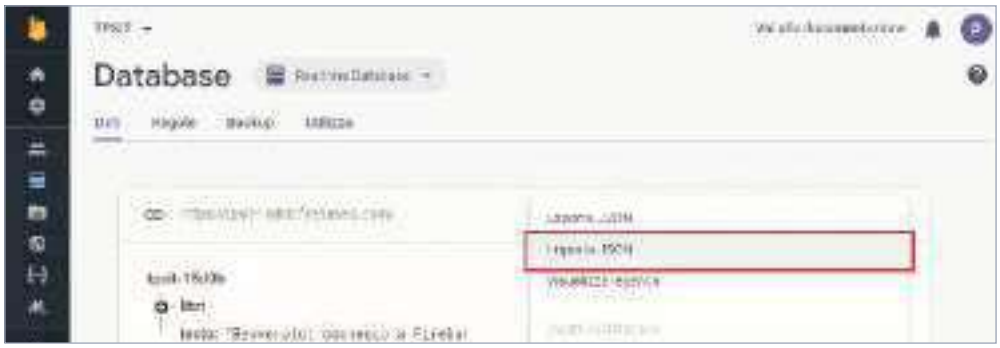
Viene generato il file tpsit-15d0b-export.json e memorizzato nella cartella Download del PC:




che possiamo salvare in una cartella del nostro disco come backup.

## Importazione di un file JSON

Un buon modo per memorizzare i dati dell'utente è creare un nodo per conservare tutti i dati degli utenti e proteggerli usando le tue regole di sicurezza: importiamo questi utenti dal file utenti.json:




Creiamo dapprima un nodo dove inserire l'archivio che verrà letto da file, per esempio permessi:

Se l'importazione va a buon fine ci troviamo nuovi dati aggiunti al nostro nodo permessi:

# 1 ESERCIZI IN LABORATORIO

In questo caso un nodo utenti con due figli, uid1 e uid2.
L'albero completo del nostro database risulta ora essere il seguente:

L'aggiornamento di un nodo comporta la sostituzione del nodo presente con il nuovo nodo importato dal file.

## Cloud Firestore

È la nuova proposta di Google per potenziare Firebase poiché la versione Realtime Dtabase è sufficiente per le applicazioni di base, ma non è abbastanza potente per gestire requisiti complessi.
Sinteticamente riportiamo i cambianti e/o le potenzialità presenti (o che sono previste) nella nuova proposta:

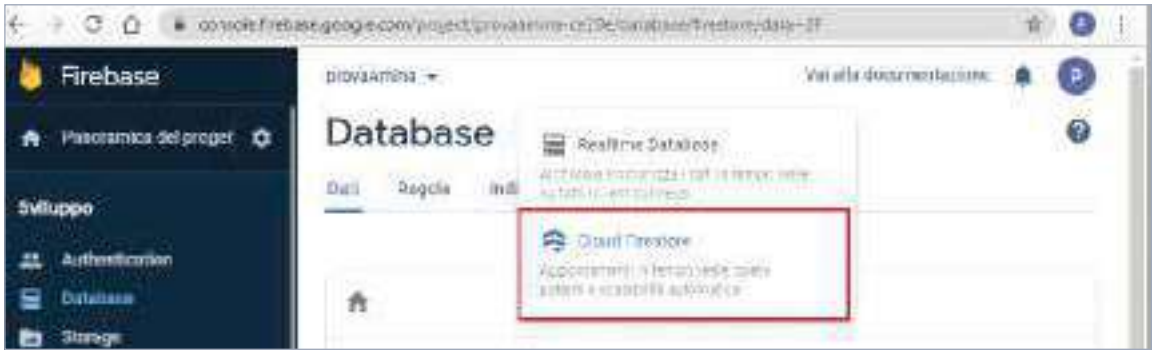
- la struttura di base dei file è migliorata;
- supporto offline per il client Web;
- supporta **query più avanzate**;
- le operazioni di scrittura e transazione sono atomiche;
- affidabilità e miglioramenti delle prestazioni;
- il **ridimensionamento** sarà automatico;
- sarà più sicuro.

### Fb menu

Cliccando sulla prima opzione del menu, dalla schermata successiva possiamo iniziare a definire il nostro database, selezionando Sviluppo e Database:

Le successive due schermate che ci vengono proposte ci chiedono la conferma della condivisione dei dati e la loro localizzazione geografica, al fine della loro sicurezza; confermiamo quanto proposto e, finalmente, possiamo iniziare a definire il nostro database.
Innanzitutto selezioniamo il tipo Cloud Firestore, cliccando sulla seconda opzione:

# ESERCIZI IN LABORATORIO 1




Possiamo ora procedere con la creazione della struttura dell'archivio.

Come possiamo vedere, non abbiamo la classica opzione dei database relazione di "creazione tabelle", in quanto in Firebase NON sono presenti tabelle!

> Tutti i dati inseriti in Firebase vanno a costituire un albero JSON, descritto nell'Esercizio in Laboratorio 5 dell'Unità di Apprendimento 1, strutturato in collezioni, composte da documenti che a loro volta possono essere o campi coppia valore-chiave o collezioni.

Ogni attributo può essere strutturato o semplice: cioè può essere una mappa, una lista, una stringa, un tipo numerico o booleano.

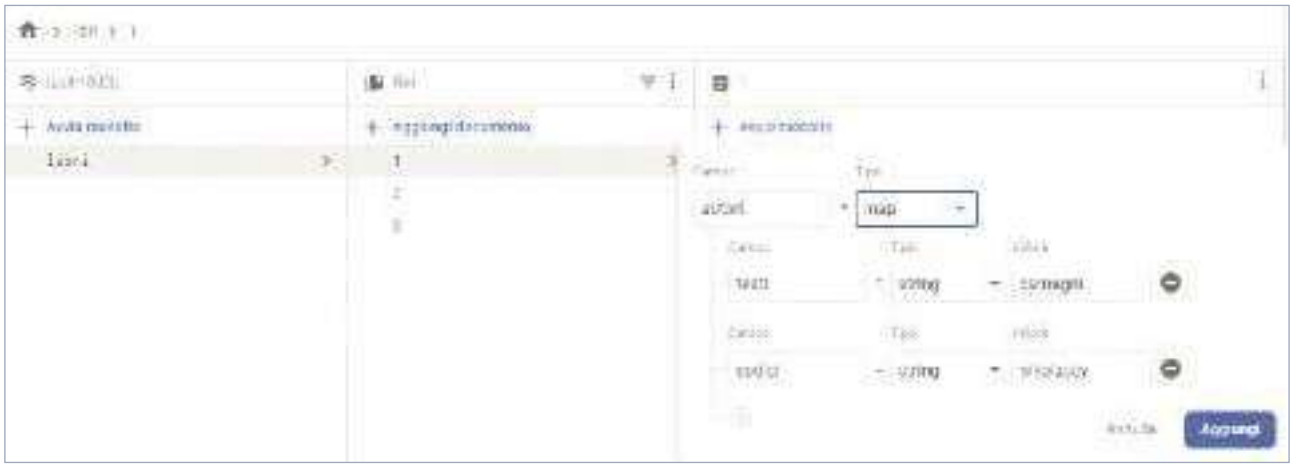
Creiamo ora una raccolta di dati e le diamo come nome libri: per la sua realizzazione utilizziamo direttamente il pannello di controllo, senza scrivere righe di codice.

Definiamo ora tre libri, ciascuno caratterizzato da un autore, un titolo, un argomento e un prezzo.

# 1 ESERCIZI IN LABORATORIO

I dati aggiunti saranno usati come chiavi e potranno essere utilizzati per recuperare i valori eventualmente a essi associati; la situazione al termine dell'inserimento è la seguente:

Modifichiamo il campo autore, sostituendolo con un oggetto composto:



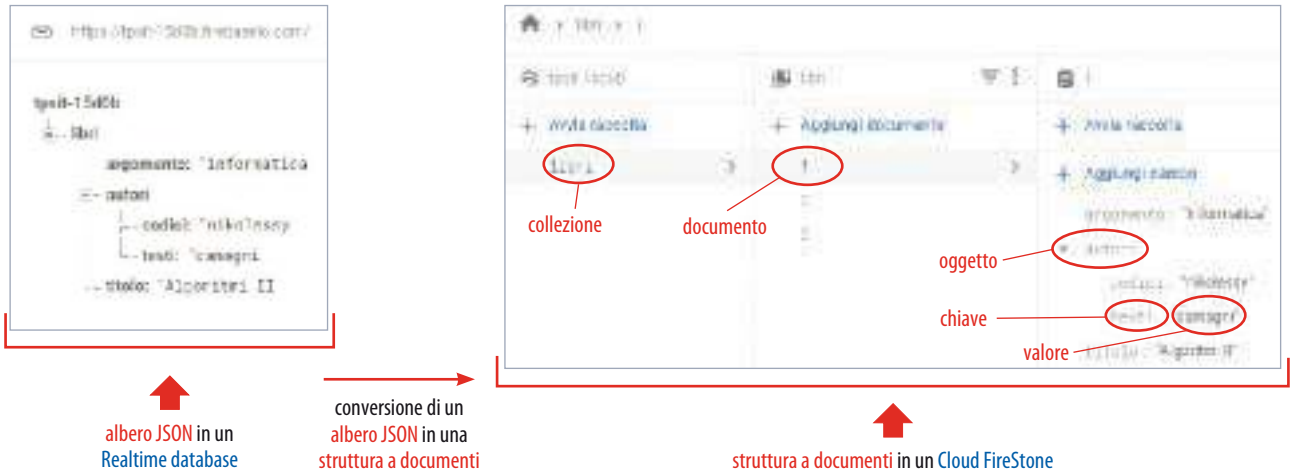
in cui possiamo notare che i singoli campi sono stati ordinati alfabeticamente in modo automatico.

# ESERCIZI IN LABORATORIO 1

## Migrazione da Realtime a Cloud

Molto probabilmente nei prossimi anni la maggior parte delle nuove app utilizzeranno Cloud Firestore, ma allo stato attuale è consigliabile utilizzare Realtime Database.

A titolo di esempio riportiamo uno schema di conversione/corrispondenza tra i due alberi di archiviazione.




Sul sito dedicato a Firebase, all'indirizzo https://firebase.google.com/docs, è disponibile una documentazione chiara e completa.

### METTITI ALLA PROVA

• Cloud Firestore

Crea un archivio di tipo Cloud Firestore per archiviare:

- un catalogo di libri,
- una collezione di dischi,
- un catalogo di automobili,
- una discussione in una chat

e per ciascuno di essi inserisci almeno tre elementi.

# 2 ESERCIZI IN LABORATORIO

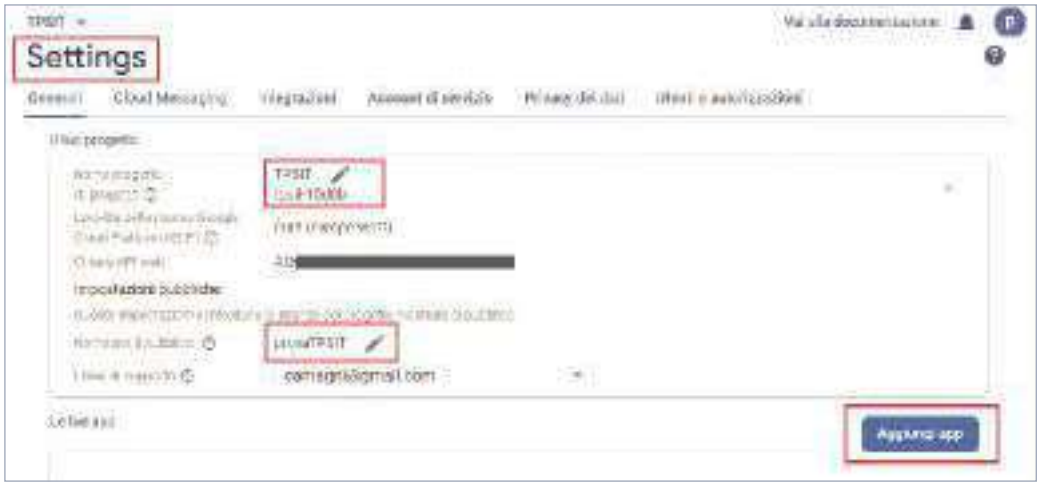
## UTILIZZARE FIREBASE IN JSP

### Connessione a Firebase

Per poter effettuare la connessione a Firebase è necessario predisporre le stringhe con le credenziali necessarie per potersi connettere al database: dal menu di scelta selezioniamo una applicazione per il Web.

Diamo TPSIT come nome al progetto, lo confermiamo e in pochi secondi ci viene segnalato che è pronto per essere realizzato:

Entriamo nel progetto e creiamo una nuova app:



# ESERCIZI IN LABORATORIO 2

Viene assegnato un ID all'applicazione:

ID app

1:1019912647899:web:80027055b67346a5e908fe

e predisposto uno snipped con i parametri e le chiavi per accedere al database:

CDN  Configurazione

Copia e incolla gli script in fondo al tag <body>, ma prima di utilizzare qualsiasi servizio Firebase:

```
<!-- The core Firebase JS SDK is always required and must be listed
<script src="https://www.gstatic.com/firebasejs/7.9.3/firebase-app.j
<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyDUM4h9TpNJbkaoTLdzn3VIk_1S_qZ_654",
    authDomain: "tpsit-15d0b.firebaseapp.com",
    databaseURL: "https://tpsit-15d0b.firebaseio.com",
    projectId: "tpsit-15d0b",
    storageBucket: "tpsit-15d0b.appspot.com",
    messagingSenderId: "1019912647899",
    appId: "1:1019912647899:web:80027055b67346a5e908fe",
    measurementId: "G-2Z8QHYMSJY"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

Copiamo tali dati in un file per poi utilizzarli in tutti i nostri script che necessitano di connettersi al database.

## Operazioni CRUD: recupero dati (Read-R)

Gli script JavaScript devono essere quindi strutturati in modo da avere tra le prime istruzioni il tag HTML che ci consente di inserire la libreria, dove prestiamo attenzione a indicare correttamente il numero di versione presente nella prima riga dello snipped realizzato in precedenza per il progetto.

```
<script src="https://www.gstatic.com/firebasejs/7.9.3/firebase.js"> </script>
```

Iniziamo a scrivere il nostro programma con queste istruzioni:

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <META CHARSET="UTF-8"/>
    <TITLE>Esempi con Firebase </TITLE>
    <!-- 1) connessione ai driver/funzioni Firebase -->
    <script
     src="https://www.gstatic.com/firebasejs/7.9.3/firebase.js">
    </script>
  </HEAD>
```

Inseriamo il tab per definire la variabile JSON che conterrà le diverse chiavi create da Firebase e necessarie per poter effettuare la connessione:

# 2 ESERCIZI IN LABORATORIO

```
11    <BODY>
12      <P>Esempio 1: connessione a Firebase </P>
13      <H3 id="id01"></H3>
14      <script>
15        // dati della nostra configurazione Firebase
16        var firebaseConfig = {
17          apiKey: "AIzaSyDQM4h9TpNJbksoTLdzn3V1k_1S_gZ_6S4",
18          authDomain: "tpsit-15d0b.firebaseapp.com",
19          databaseURL: "https://tpsit-15d0b.firebaseio.com",
20          projectId: "tpsit-15d0b",
21          storageBucket: "tpsit-15d0b.appspot.com",
22          messagingSenderId: "1019912647899",
23          appId: "1:1019912647899:web:80027055b67346a5e988fe",
24          measurementId: "G-2Z8QHYM57Y"
25        };
```

Quindi inizializziamo la nostra applicazione con la seguente istruzione, alla quale passiamo come parametro il record con i dati della nostra connessione dati:

```
26        // 2) inizializzazione di Firebase
27        firebase.initializeApp(firebaseConfig);
```

Indichiamo il percorso del nodo dove vogliamo leggere il contenuto dell'attributo testo:

```
28        // 3) visualizza un singolo elemento del database
29        var output = document.getElementById('id01');
30        var dbRef = firebase.database().ref().child('testo');
31        dbRef.on('value', snap => output.innerText = snap.val());
32      </script>
33    </BODY>
34  </HTML>
```

Avviamo l'esecuzione e sullo schermo otteniamo l'output riportato a sinistra (nella figura a destra, invece, mostriamo la struttura del nostro albero-archivio):

I dati recuperati dal database, nella terminologia di Firebase, prendono il nome di DataSnapshot e rappresentano l'immagine del contenuto di un nodo in un dato istante: è una copia immutabile, generata in modo efficiente, dei dati in una posizione del database.

Analizziamo le istruzioni presenti nelle righe 30 e 31:

- il metodo database() restituisce il database corrente;
- il metodo ref() crea un riferimento al nodo Firebase restituendo un DataSnapshots snap;
- il metodo child(<campo>) specifica l'attributo desiderato (testo nel nostro caso).

Ogni volta che leggi i dati dal database, questi sono ricevuti come oggetto snap.
L'uso più comune degli eventi ref() di Firebase è ascoltare l'evento value per il caricamento iniziale dei dati eseguito alla riga 31 con il metodo on('value', <opzioni>); dobbiamo sottolineare che l'evento value viene generato sia al caricamento iniziale dei dati sia ogni volta che un dato cambia, generando un callback che effettua il refresh della pagina

# ESERCIZI IN LABORATORIO 2

JavaScript: questa caratteristica, se da una parte è particolarmente utile, per esempio in applicazioni con chat e/o messaggistica in tempo reale, a volte può essere fastidiosa e, in tal caso però, può essere disabilitata settando off() sul listener non appena i dati sono stati caricati oppure, in alternativa, utilizzando a on() il metodo once(), che disabiliterà automaticamente il refresh.
Quindi i dati sono passati ai callback dell'evento collegati all'oggetto dbRef con i metodi on() o once():

```
dbRef.on ('value', callback);          // riceve ogni modifica del reference
dbRef.once ('value', callback);        // riceve solo la prima modifica/richiesta del reference
```

Ricapitolando, per disabilitare il push si hanno due possibilità:

**A** con il seguente codice che cancella tutti i metodi di callback assegnati all'evento value del riferimento ref:

```
dbRef.off('value');
```

**B** con il seguente codice che cancella il push subito dopo avere ottenuto lo snapshot dei dati:

```
dbRef.once ('value', function(snapshot))
{
// operazioni da eseguire
});
```

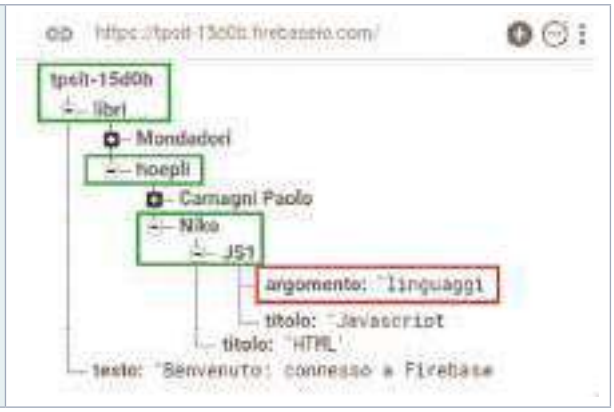
Si consiglia quindi l'utilizzo di dbRef.once ('value', callback) perché restituisce i figli e disattiva la ricezione in un'unica operazione.
L'utilizzo di dbRef.on ('value', callback) può essere necessario quando si vuole mantenere un oggetto in memoria sincronizzato con Firebase.

Modifichiamo ora il codice memorizzando nel file datiConnessione.json le variabili di configurazione:

Modifichiamo il codice per visualizzare gli attributi di un'altra variabile dell'albero, dopo averne scritto il percorso completo in una variabile di comodo:

# 2 ESERCIZI IN LABORATORIO

```
<script>
  // 3) visualizza un singolo elemento del database
  var output = document.getElementById('id01');
  var path = 'libri/hoepli/Niko/JS1/argomento';
  document.write("percorso Firebase:"+path);
  var dbRef = firebase.database().ref().child(path);
  dbRef.on('value', snap => output.innerText = snap.val());
</script>
```




L'esecuzione visualizza il seguente output:

Possiamo anche visualizzare l'albero completo in formato JSON:

```
<script>
  // visualizza albero JSON
  var output = document.getElementById("id01");
  var ref = firebase.database().ref();
  ref.on("value", function(snapshot){
    output.innerHTML = JSON.stringify(snapshot.val(), null, 2);
  });
</script>
```

Ricapitoliamo un insieme di metodi disponibili per l'oggetto snap che è snapshot dei dati:

- snap.val(): estrae il contenuto dallo snapshot come oggetto JavaScript;
- snap.exportVal(): ottiene anche il valore (tipo di duplicativo);
- snap.child(): passa a un riferimento figlio e restituisce snapshot secondari;
- snap.hasChild(): verifica se esiste almeno un figlio;
- snap.forEach(callback): scorre ciclicamente gli snapshot secondari con un callback;
- snap.hasChildren(): verifica se ci sono più figli;
- snap.ref(): è un attributo che contiene il riferimento dell'istantanea ed è possibile utilizzarlo anche con snapshot secondari all'interno di uno snap.forEach();
- snap.key: è un attributo che contiene la chiave di riferimento;
- snap.numChildren(): ritorna il numero di figli del riferimento.

Oltre a value, sono disponibili altri eventi che effettuano contemporaneamente la lettura di uno snapshot e la sincronizzazione automatica, a fronte di modifiche intervenute sui dati remoti.

# ESERCIZI IN LABORATORIO 2

Si tratta di:

- child_added: utilizzato per le liste di oggetti, viene invocato per ognuno degli elementi che le compongono e per ogni nuovo elemento che vi viene aggiunto;
- child_changed: viene attivato quando il nodo puntato è modificato e nel metodo di callback sarà disponibile il nuovo dato aggiunto in remoto;
- child_removed: viene invocato alla rimozione del nodo.

Tipicamente i tre eventi vengono configurati insieme in uno script, per poter fornire alla pagina un flusso continuo di aggiornamenti dei dati ottenuti.

Possiamo rivedere gli script che recuperano i dati presenti nel database definendoli sotto forma di oggetti e individuandone i singoli attributi mediante le apposite funzioni disponibili:

```
16      <script>
17        var output = document.getElementById("id01");
18        // definizione dato del nodo
19        var path = 'libri/hoepli/Nikos/';                    // percorso del nodo
20        var nodo = '351';                                    // nome del nodo
21        document.write("percorso Firebase:" + path);
22
23        var nodoRef = firebase.database().ref(path+nodo);
24        nodoRef.once("value").then(function(snap){
25          var key = snap.key;                                // 351
26          var childKey = snap.child('titolo').key;           // titolo
27          var childTitolo = snap.child("titolo").val();      // Javascript
28          output.innerText = childTitolo;
29        });
30      </script>
```

Passare un percorso relativo al metodo child() di un DataSnapshot restituisce un altro DataSnapshot per la posizione nel percorso relativo specificato: quindi è possibile, a partire da quel riferimento, recuperare un attributo semplicemente indicando il nome (per esempio "titolo") oppure, come nel successivo esempio, indicando semplicemente il nome del figlio (per esempio "/argomento") o un percorso più profondo e separato da barra (per esempio nodo +"/titolo").

Se l'ubicazione figlio non contiene dati, viene restituito un DataSnapshot vuoto, cioè un DataSnapshot il cui valore è null.

```
33        // esempio 2
34        var nodoRef = firebase.database().ref(path);
35        nodoRef.once("value").then(function(snap) {
36          var volume = snap.child(nodo).val();                          // oggetto
37          var childTitolo =  snap.child(nodo+"/titolo").val();          // "Javascript"
38          var childArgomento= snap.child(nodo).child("/argomento").val();  // "linguaggi"
39          var childPrezzo = snap.child("prezzo").val();                 // null
40          output.innerText = childTitolo +  " + childArgomento +" " + childPrezzo;
41        });
```

## METTITI ALLA PROVA

→ • Operazioni CRUD: recupero dati (Read-R)

Crea un database automobili dove inserire i dati di un catalogo di automobili usate; quindi:

**a** visualizza le marche delle automobili presenti;
**b** visualizza il numero di auto presenti;
**c** visualizza l'albero del database;
**d** per un particolare nodo visualizza tutti gli attributi;
**e** effettua una visita in profondità dell'albero;
**f** effettua una visita per livelli dell'albero.

# 2 ESERCIZI IN LABORATORIO

## Operazioni CRUD: update (U)

Modifichiamo un nodo aggiornando un figlio, cioè sostituendo un figlio con un nuovo documento, cioè un nuovo figlio, composto da due attributi:

```
16  <script>
17    // 2) sostituzione di un nodo: tutto il ramo viene sostituito
18    var path ='libri/hoepli/Niko/';
19    var postData =  {
20      "JS1":{
21        argomento: "linguaggi",
22        prezzo: 23
23    }};
24    var updates = {};
25    updates[path] = postData;
26    firebase.database().ref().update(updates);
27
28    // 3) visualizzo il nodo modificato
29    document.write("percorso Firebase:"+path);
30    var output = document.getElementById("id01");
31    var ref = firebase.database().ref().child(path);
32    ref.on("value", function(snapshot){
33      output.innerHTML = JSON.stringify(snapshot.val(), null, 2);
34    });
35  </script>
```

L'aggiornamento viene effettuato alla riga 26 utilizzando la funzione update(): riportiamo l'output e l'albero del database, dove vediamo, in arancione, i dati che vengono modificati con tutti i nodi che ne sono coinvolti:

Scriviamo ora il codice per modificare il singolo attributo prezzo di un nodo:

```
<script>
// 2) aggiorno il singolo attributo di uno specifico nodo
var updates = {};
var path ='libri/hoepli/Niko/JS1/';
var attributo = 'prezzo';
updates[path+attributo] = 1/;
firebase.database().ref().update(updates);
```

Sempre utilizzando update() è anche possibile effettuare più modifiche contemporaneamente: è sufficiente inserire i Path da aggiornare con, tra parentesi, le coppie attributo-valore che si vogliono modificare:

```
// sostituisce attributi in nodi differenti
var postData = {
    "libri/hoepli/Niko/JS1":{            // nodo 1
        titolo: "Il programmo in JS",
        argomento: "informatica",
        prezzo: 23
    },

    "testo" :"Aggiornamento con update"   // nodo 2
};
firebase.database().ref().update(postData);
```

# ESERCIZI IN LABORATORIO 2

## METTITI ALLA PROVA

• Operazioni CRUD: update (U)

Sempre con riferimento all'archivio automobili precedentemente creato:

**a** modifica il nome della marca Fiat con FCA group in ogni occorrenza (se non presente, aggiungine una);

**b** conta il numero di nodi che hai modificato e visualizzalo;

**c** scrivi un programma che visualizzi il tuo albero e chieda all'utente quale nodo modificare, indicando il percorso: quindi esegui l'operazione visualizzando il risultato.

## Operazioni CRUD: creazione (C)

Inseriamo ora un nuovo nodo all'interno del database, per esempio, aggiungiamo un libro al nodo con percorso:

```
var path ='libri/hoepli/Niko/'
```

Definiamo il nome del nodo e l'elenco degli attributi in due variabili:

```
<script>
  // 2) definizione dato del nuovo nodo
  var path ='libri/hoepli/Niko/';
  var newNodo = 'Java';                    // nome del nodo
  var attributi = {                        // attributi del nodo
    titolo: "Linguaggio Java",
    argomento: "linguaggi",
    prezzo: 12
  };
  // 3) aggiornamento database
  var updates = {};
  updates[path+newNodo] = attributi;
  firebase.database().ref().update(updates);
```

È possibile aggiungerlo al percorso indicato utilizzando sempre la funzione update(), già usata per l'aggiornamento dei dati.

Esiste anche una funzione specifica per creare un nuovo ref e collocarlo nell'albero:

```
ref.push(obj);          // crea un nuovo ref con una nuova "push key"
ref.set(obj);           // aggiorna i singoli dati
```

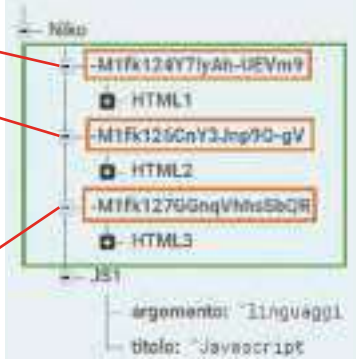
La caratteristica fondamentale di questa funzione è che genera una chiave unica per ciascun figlio, particolarmente utile nelle applicazioni concorrenti.

# 2 ESERCIZI IN LABORATORIO

Vediamo un esempio con tre modalità di utilizzo di tale funzione.

```
16      <script>
17        // 2) definizione del nuovo nodo
18        var path ='libri/hoepli/Niko/';
19        var newNodo = 'HTML1';                  // nome del nodo
20        var attributi = {                       // attributi del nodo
21          titolo: "HTML1 versione1",
22          argomento: "linguaggi"
23        };
24
25        // 3a) aggiungo nodo prima definito con push()
26        var nuovoNodo1 = {};
27        nuovoNodo1[newNodo] = attributi;
28        firebase.database().ref(path).push(nuovoNodo1);
29
30        // 3b) aggiungo nodo con push() e successive set() per gli attributi
31        var nodo = firebase.database().ref(path);
32        var nuovoNodo2 = nodo.push();
33        nuovoNodo2.set({'HTML2':{                // nodo e attributi
34          titolo: "HTML1 versione2",
35          argomento: "linguaggi"
36        }});
37
38        // 3c)aggiungo nodo con push() e contestuale set() per gli attributi
39        var nuovoNodo3 = nodo.push().set({'HTML3':{
40          titolo: "HTML1 versione3",
41          argomento: "linguaggi"
42        }});
43
44        // 4) visualizza albero JSON
45        document.write("percorso Firebase: "+path);
46        var output = document.getElementById("id01");
47        var ref = firebase.database().ref().child(path);
48        ref.on("value", function(snapshot){
49          output.innerHTML = JSON.stringify(snapshot.val(), null, 2);
50        });
51      </script>
```

Osserviamo ora il nostro archivio, dove troviamo tre nuovi nodi, ma inseriti con una chiave casuale:




## Operazioni CRUD: delete (D)

Per eliminare un nodo o parte di esso abbiamo due possibili alternative:

1 la prima è quella di utilizzare la funzione remove() che, una volta assegnato il percorso e il nome del nodo, provvede alla sua rimozione:

```
16      <script>
17        // definizione dato del nodo
18        var path ='libri/hoepli/Niko/';
19        var nodo = 'Java';                  // nome del nodo
20        // aggiornamento database
21        const userRef = dbRef.child(path + nodo);
22        userRef.remove();
```

2 la seconda è quella di sostituire il nodo da cancellare con un nodo vuoto: automaticamente Firebase si accorge che tale nodo non ha più attributi e lo elimina dall'archivio.

```
17        // definizione nodo da cancellare
18        var path = 'libri/hoepli/Niko/';
19        var nodo = 'Java';                  // nome del nodo
20        var postData = {                    // nodo senza attributi
21        };
22        // aggiornamento database
23        var updates = {};
24        updates[path+nodo] = postData;
25        firebase.database().ref().update(updates);
26
```

# ESERCIZI IN LABORATORIO 2

La figura che segue riporta l'esecuzione di uno dei due codici sopra riportato dove si evidenzia il nodo rimosso.

La stessa operazione di rimozione può essere anche effettuata, in modo più semplice, con la scrittura che segue che ci permette di eliminare anche più nodi contemporaneamente:

```
18      // cancellazione di un nodo
19      var postData = {
20        "libri/hoepli/Niko/J52":          // percorso nodo da cancellare
21        { },                              // nodo vuoto
22        "libri/hoepli/Camagni Paolo/TPSIT3": // percorso nodo da cancellare
23        { },                              // nodo vuoto
24      };
25      firebase.database().ref().update(postData);
```

## Conclusioni

Abbiamo descritto solo le operazioni fondamentali che possono essere fatte con Firebase.
Il recupero dei dati può essere arricchito con funzionalità "da query", principalmente, ordinamento e limitazione del numero di risultati.
I metodi che svolgono questi compiti sono:

- ordinamento: orderByChild(), orderByKey() e orderByPriority():
  - il primo ordinerà i nodi in base a una child key, ossia una chiave contenuta in essi;
  - il secondo ordinerà i dati in base alle chiavi;
  - l'ultimo li ordinerà in base alla priorità del nodo;
- selezione del numero: limitToFirst(), limitToLast(), startAt(), endAt() e equalTo().

Lo scopo di questi metodi essenzialmente è quello di prelevare una porzione dei dati ottenuti.

Con le funzioni limitToFirst e limitToLast si preleveranno rispettivamente all'inizio e alla fine della lista, mentre startAt ed endAt permetteranno di ottenere una porzione centrale.

### ESEMPIO

Si può eseguire una query per recuperare i due oggetti con il prezzo minore:

```
47      ref.orderByChild("prezzo").limitToFirst(2).on("hoepli", function(snapshot)
48      {
49        // operazioni da svolgere
50
51      });
```

# 2 ESERCIZI IN LABORATORIO

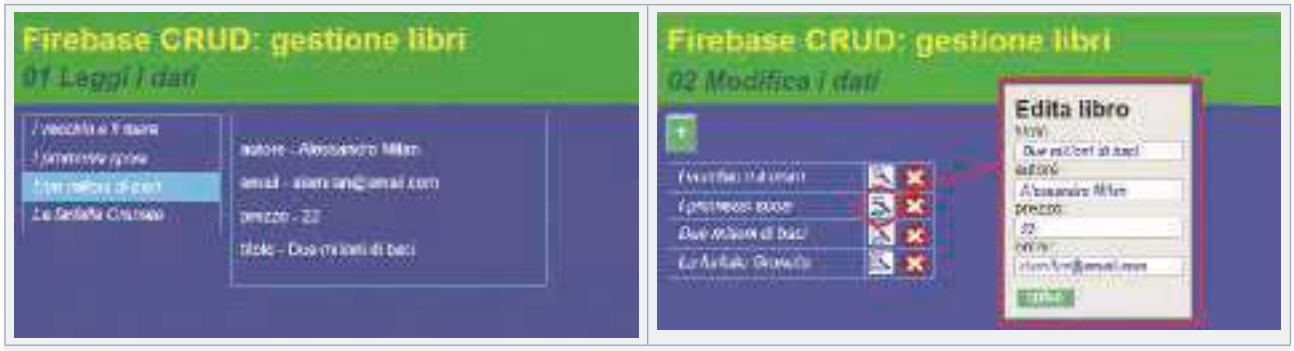
## METTITI ALLA PROVA

• Operazioni CRUD

### Progetto Libri-CRUD

Importa il database libri.json e realizza un'applicazione che offra le opzioni CRUD per la sua manutenzione: predisponi una pagina che visualizzi il contenuto dell'archivio e presenti i pulsanti per effettuare le operazioni.

1 Leggere i dati (Read): cliccando direttamente sul titolo del libro si visualizza la sua scheda.
2 Modificare i dati (Update): cliccando sull'apposita icona si apre il form che ne permette la modifica.




3 Inserire nuovi libri (Create): cliccando sul pulsante "+" si visualizza il form per l'immissione di un libro.
4 Cancellare un libro (Delete): cliccando sull'apposita icona "x" si chiede la conferma della sua cancellazione.

### Progetto Libri completo

Utilizzando sempre il database libri.json realizza un'applicazione più completa che offra un insieme di operazioni tramite una home page simile a quella della figura, dove ogni bottone richiama una specifica funzione.

# Android e i dispositivi mobili

UNITÀ 8

**LEZIONE 1**
Dispositivi e reti mobili

**LEZIONE 2**
Android: un sistema operativo per applicazioni mobili

**Esercizi in Laboratorio**

L1 Android Studio: installazione e configurazione

L2 L'interfaccia grafica di Android Studio

L3 Utilizzare i widget nelle app Android

L4 Un'app completa: la calcolatrice

L5 Utilizzare i sensori nelle app


## CONOSCENZE

- Comprendere il ruolo del Sistema Operativo Android
- Conoscere il ciclo di vita di una Activity
- Riconoscere il campo di applicazione di una Activity e di un Service
- Riconoscere i diversi widget utilizzabili nell'interfaccia grafica Android
- Riconoscere il ruolo dell'annotazione @Override

## COMPETENZE

- Riconoscere gli elementi di un'applicazione Android
- Realizzare una applicazione di prova
- Utilizzare Android Monitor
- Saper collocare breakpoint
- Modificare le proprietà dei widget in ambiente grafico e nel file activity_main.xml
- Mostrare a video messaggi a tempo (Toast)

## ABILITÀ

- Scaricare, installare e configurare Android Studio
- Installare Android SDK e AVD Manager
- Effettuare il debug con emulatore Android e collegando il dispositivo mediante USB
- Collocare i widget disponibili nel layout
- Utilizzare l'evento OnCreate e OnClick

I file relativi a esempi, esercizi, esercizi in laboratorio e a eventuali soluzioni di questa Unità sono scaricabili dal sito hoepliscuola.it.

LEZIONE 1

# 1 Dispositivi e reti mobili

## IN QUESTA LEZIONE IMPAREREMO...

- i tipi di reti mobili
- i dispositivi mobili
- il software per dispositivi mobili

## AREA DIGITALE

**LEZIONE 2**
Android: un sistema operativo per applicazioni mobili

**ESERCIZI IN LABORATORIO 1**
Android Studio: installazione e configurazione

**ESERCIZI IN LABORATORIO 2**
L'interfaccia grafica di Android Studio

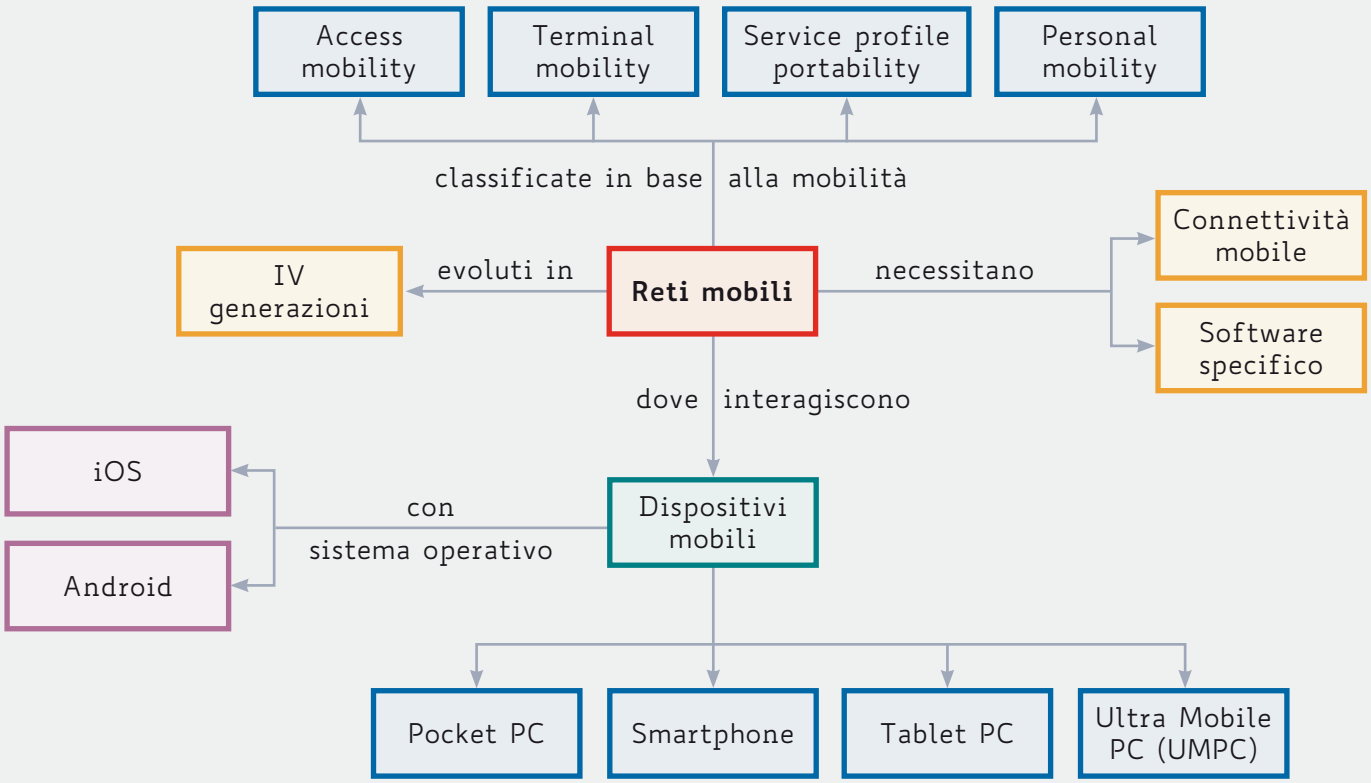
**ESERCIZI IN LABORATORIO 3**
Utilizzare i widget nelle app Android

**ESERCIZI IN LABORATORIO 4**
Un'app completa: la calcolatrice

**ESERCIZI IN LABORATORIO 5**
Utilizzare i sensori nelle app

## MAPPA CONCETTUALE

didattica inclusiva




## Premessa

È uso comune parlare indifferentemente di reti mobili e reti wireless associandole per il fatto che in entrambi i casi i dispositivi si scambiano informazioni senza fili, ma mentre le ultime sono spesso la soluzione di reti locali aziendali realizzate senza dover cablare gli edifici, composte da dispositivi per lo più con postazioni fisse, le prime sono caratterizzate da terminali che possono muoversi continuando a mantenere attiva la connessione, con la innegabile comodità di essere rintracciati anche quando sono in movimento a chilometri di distanza.

La mobilità è la caratteristica fondamentale di queste reti.

Oggi ogni utente ha sempre con sé uno (o più) dispositivi in grado di connettersi con gli altri: telefoni cellulari, smartphone e tablet hanno processori con potenze di calcolo che non hanno nulla da invidiare alle postazioni fisse.
Per le loro caratteristiche fisiche, soprattutto le dimensioni dello schermo, questi necessitano di meccanismi di interfacciamento diversi rispetto alle postazioni fisse, primo fra tutti il meccanismo touch che raramente viene implementato sui PC.
Possiamo quindi individuare due elementi essenziali caratterizzanti i dispositivi mobili:
**a** meccanismi che permettono la connettività mobile;
**b** software specifico per ogni singolo tipo di dispositivo.

# Reti mobili

La connettività tra dispositivi mobili viene realizzata con le reti mobili che possono essere classificate sia in base ai tipi di mobilità che offrono, sia in base alle tecnologie utilizzate.

## Classificazione delle reti mobili in base ai tipi di mobilità

In base ai tipi di mobilità possiamo distinguere quattro diverse classi.

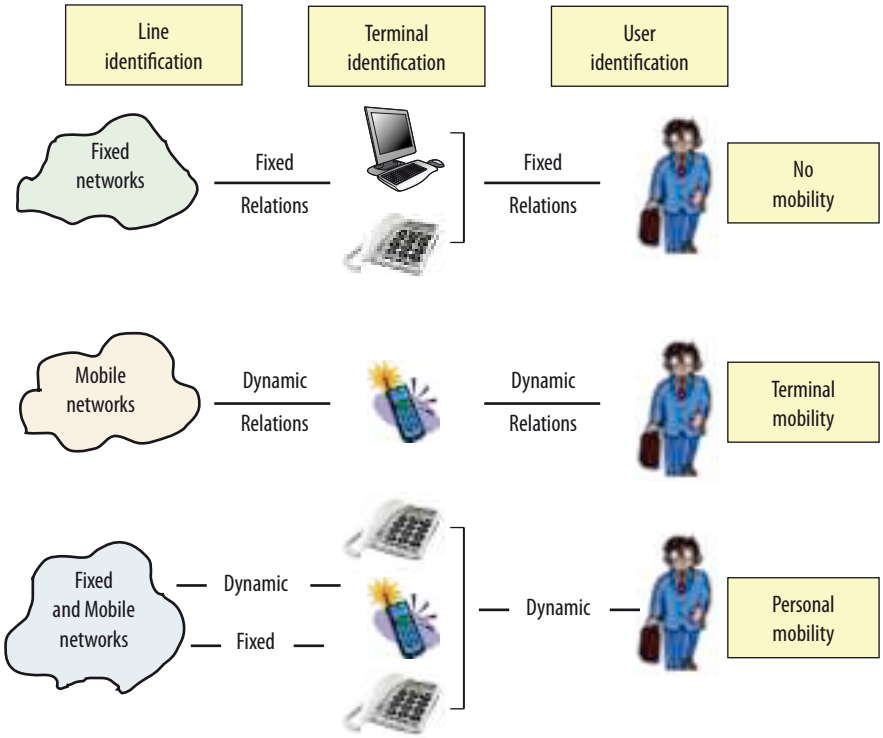
**3** Access mobility – Sono i sistemi tipicamente indoor con raggio di azione inferiore ai 500 metri, come i sistemi cordless telefonici, che tramite un access point connettono alla rete fissa di telecomunicazione.

**4** Terminal mobility – Sono i sistemi nei quali il terminale è in movimento e può accedere, rimanere connesso e identificato in una rete di telecomunicazione.

**5** Service profile portability – In questo sistema un terminale può connettersi non solo alla propria rete ma anche ad altre reti di telecomunicazione per poter usufruire dei servizi associati al proprio profilo; esiste quindi la portabilità del proprio profilo su provider diversi, in modo trasparente per l'utente.

**6** Personal mobility – In questo sistema è l'utente che può accedere ai servizi di telecomunicazione tramite qualsiasi terminale: la rete riconosce l'utente e non il terminale; questo permette un'offerta completa di servizi e in questo caso si parla di Personal Communications Services (PCS).

Possiamo distinguere meglio le diverse situazioni utilizzando la figura riportata di seguito:



- Nel caso di Access mobility esiste una relazione fissa tra l'utente e il proprio terminale e tra il terminale e la rete fissa e, quindi, siamo praticamente in assenza di mobilità (no mobility).
- Nel caso di Terminal mobility il terminale, tramite la sua SIM card, viene identificato e connesso alla rete; inoltre può connettersi a reti diverse in situazioni di Service profile portability.
- Nelle situazioni di Personal mobility abbiamo la completa dinamicità in quanto l'utente può diventare dinamico, poiché può essere contemporaneamente presente su più terminali fissi o mobili.

## Classificazione delle reti mobili in base alla tecnologia utilizzata

La classificazione delle reti in base alla tecnologia utilizzata introduce il concetto di generazione, in particolare di quattro generazioni, con la "quinta alle porte":

**I generazione** – La prima generazione è quella che, nei "lontani" anni Ottanta, venne identificata dalla sigla 1G: i sistemi di trasmissione sono di tipo analogico, in grado di gestire solo il traffico voce. Non esisteva l'interoperabilità a livello europeo e tra i telefoni cellulari di prima generazione ricordiamo il sistema standardizzato TACS (Total Access Communication System), con apparecchiature decisamente voluminose, scarsa qualità audio con frequenti interruzioni, limiti legati alla tipologia di segnale.

**II generazione** – Nel 2006 nacque la prima rete di trasmissione digitale standardizzata e interoperante a livello europeo: si introsse lo standard GSM (Global System for Mobile communications). Con l'avvento del digitale si iniziarono ad avere i primi servizi di trasmissione dati, sotto forma di messaggi di testo (SMS, Short Message Service), messaggi multimediali (MMS, Multimedia Message Service).
In questa generazione venne sviluppato il protocollo WAP (Wireless Application Protocol) in grado di permettere l'accesso da parte dei telefoni a particolari contenuti Internet, realizzando con la tecnologia GPRS quella che è conosciuta come generazione 2.5 G, una via di mezzo fra la seconda e la terza generazione.
La tecnologia GPRS, (General Packet Radio Service) consiste in un tipo di trasferimento dei dati detto commutazione a pacchetto, così chiamato perché i dati vengono divisi per essere spediti separatamente per poi essere ricongiunti a destinazione.
A questa tecnologia è seguita la tecnologia EDGE (Enhanced Data rates for Global Evolution) portando un notevole incremento della velocità di connessione.

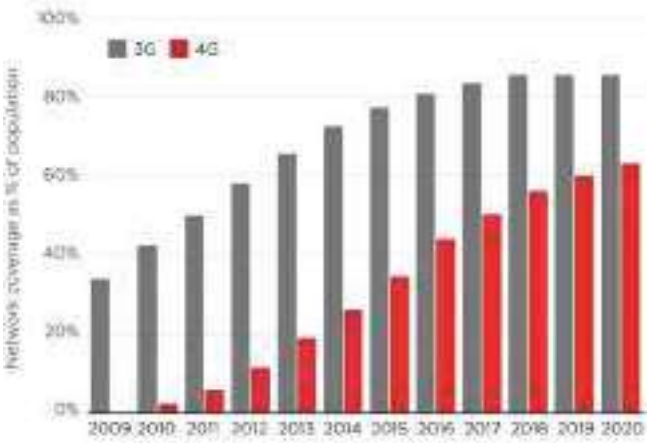
**III generazione** – Con l'inizio del nuovo secolo iniziarono a diffondersi sistemi multimediali, globali, che integrarono più servizi, accessibili da ogni terminale a disposizione dell'utente, così da realizzare la Personal mobility: lo standard UMTS (Universal Mobile Telecommunications System), tutt'ora attuale e il più utilizzato in Europa, è un'evoluzione del GSM, e si impone come standard internazionale di telefonia mobile 3G.
L'UMTS sfrutta il protocollo W-CDMA (Wideband Code Division Multiple Access), una particolare tecnologia di accesso multiplo al canale radio a divisione di codice CDMA (Code Division Multiple Access), che permette di raggiungere fino a 2 Mbps (teorici) di velocità di trasmissione.
Con il successivo protocollo HSPA (High-Speed Packet Access) l'offerta di servizi multimediali venne completata con, per esempio, le videochiamate "finalmente" fluide e senza "scatti" e la navigazione in Internet da terminali mobili, caricando con buona velocità anche le pagine ricche di contenuti, script e immagini.

**IV generazione** – La sigla 4G identifica la quarta, e attuale, generazione dei servizi di telefonia mobile. Il protocollo HSPA del 3G evolve nell'HSDPA (High-Speed Downlink Packet Access), un protocollo di trasmissione che migliora ulteriormente le prestazioni consentendo di raggiungere la velocità teorica di 14.4 Mbps in download e 2 Mbps in upload, cioè prestazioni simili alle connessioni cablate ADSL, e si pone come passaggio intermedio tra l'UMTS delle reti 3G e le reti LTE (Long Term Evolution) della quarta generazione.

Con la tecnologia LTE e la sua più recente evoluzione LTE Advanced (LTE-A) si inviano e ricevono dati a un'ottima velocità di connessione, quasi la stessa delle connessioni domestiche, e quindi si può navigare e accedere a cloud, servizi di streaming e video conferenza in alta definizione senza rallentamenti o interruzioni.

In Italia la rete 4G ha una velocità di connessione di 15 Mbps che ci colloca tra i primi paesi al mondo come qualità del servizio ma, purtroppo, non abbiamo ancora una buona copertura anche se tutti i nuovi dispositivi, sia smartphone che tablet, sono in grado di utilizzarla.

L'obiettivo dell'Agenda digitale per l'Italia prevedeva la copertura universale a 30 Mbit/s per la banda larga dell'UE entro il 2020; il grafico a lato riporta l'andamento della crescita della rete 4G rispetto alla 3G per il decennio appena trascorso.




**Mobile broadband coverage reach, 2009-2020**
(*fonte:* GSMA Intelligence)

> Oltre alle tecnologie LTE e LTE_A, nel 2010 l'Unione Internazionale delle Telecomunicazioni (ITU) ha autorizzato anche la tecnologia Wimax (Worldwide Interoperability for Microwave Access) che consente l'accesso di tipo wireless a reti di telecomunicazioni a banda larga BWA (Broadband Wireless Access).

### 5G: la nuova frontiera della tecnologia mobile

Ma gli operatori del settore telecomunicazioni sono agguerriti e stanno già studiando il "dopo 4G", e l'ITU sta parallelamente delineando gli standard del 5G con la definizione delle linee guida IMT-2020, che porteranno la velocità massima teorica di navigazione con smarthphone e tablet a 20 Gbps, che sarebbe 20 volte più veloce della velocità massima teorizzata per il 4G che è di 1 Gbps!
Con la generazione 5G si potrà connettere un altissimo numero di dispositivi per km$^2$, mantenere la connessione anche viaggiando ad altissime velocità e offrire tempi di latenza ridottissimi.
La sfida è quella di connettere sempre più dispositivi e nuove tipologie di dispositivi, da quelli a bordo delle autovetture e dei mezzi di trasporto a quelle werable delle body network, per scopi ludici (gaming pervasivo), sanitari o di realtà aumentata, dove anche gli oggetti avranno "dispositivi identificativi", cioè saranno dotati di un'identità elettronica e potranno dialogare in rete ed essere controllabili a distanza.

> L'Unione Europea aveva stanziato quasi 80 miliardi di euro di fondi disponibili in 7 anni (dal 2014 al 2020) per il programma di ricerca e innovazione Orizzonte 2020: gli obiettivi del programma, molto ambiziosi, erano innovazioni e primati mondiali, tra i quali il raggiungimento del 5G, con un insieme di progetti e applicazioni elencati all'indirizzo https://ec.europa.eu/programmes/horizon2020.

## Software per dispositivi mobili

La capillare distribuzione di dispositivi mobili, dovuta anche al loro vertiginoso calo dei costi e alla crescita della capacità di elaborazione, impone agli sviluppatori una profonda riflessione in quanto sia le aziende che i privati utilizzano questi strumenti, oltre che per attività ricreative, anche per svolgere le normali funzioni lavorative.

**ESEMPIO**

È abbastanza normale vedere agenti e rappresentanti inserire ordini e/o consultare i dati dei listini o del magazzino direttamente seduti al tavolo del bar oppure nello scompartimento di un treno: con questi strumenti i tempi di lavoro vengono drasticamente ridotti aumentando quindi potenzialmente il volume d'affari.

È necessario scrivere quindi software adeguati a questi dispositivi che presentano meccanismi di interfacciamento diversi dai normali PC dotati di keyboard e monitor: come primo passo individuiamo le caratteristiche di questi dispositivi.

## I dispositivi mobili

Esistono sostanzialmente quattro tipologie di dispositivi mobili, ognuna dotata di caratteristiche particolari da tenere in considerazione quando progettiamo le applicazioni a loro destinate.

1 Pocket PC: sono i cosiddetti "palmari", con display di tipo touch screen, molto compatti ma con una discreta possibilità di espansione e una discreta potenza di calcolo; possiamo individuare due sottofamiglie:
   - Standard: non hanno la possibilità di effettuare chiamate vocali;
   - Phone Edition: hanno in aggiunta le funzionalità telefoniche.

2 Smartphone: sono dispositivi prettamente telefonici, meno potenti dei Pocket PC e non tutti dotati di display touch screen, ma in continua crescita di dimensioni e funzionalità.

3 Tablet PC: sono dispositivi molto simili ai normali Notebook ma dotati di touch screen e di funzionalità di riconoscimento della scrittura.

4 UMPC (Ultra Mobile PC): sono una via di mezzo tra il Pocket PC e il Notebook, più orientati verso il mercato consumer che business.

## Sistema operativo e applicazioni

Come in tutti i calcolatori, anche nei dispositivi mobili abbiamo due categorie di software:

1 software di base, cioè il sistema operativo (SO);

2 applicazioni, anche chiamate app.

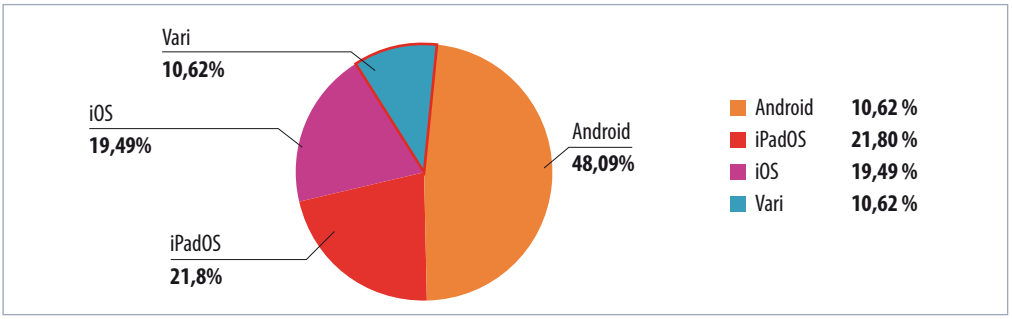
Nel seguito della trattazione faremo una "panoramica" sui principali sistemi operativi oggi "a bordo" degli attuali smartphone e per ciascuno di essi descriveremo un ambiente di sviluppo specifico per la realizzazione delle applicazioni.

Essendo un mercato giovane e in gran fermento, sia i sistemi operativi sia gli ambienti di sviluppo sono in perenne evoluzione e nuove soluzioni vengono proposte "pressoché in continuazione": perciò non entreremo nel dettaglio di ogni prodotto (ma ne descriveremo solo le principali caratteristiche) in quanto potrebbe essere obsoleto già al momento della stampa di questo volume; invitiamo quindi il lettore a mantenersi costantemente aggiornato tramite il Web.

# Sistemi operativi per dispositivi mobili

Nei dispositivi portatili, a differenza dei sistemi fissi, non è l'utente che decide il sistema operativo che vuole utilizzare, ma il produttore dell'hardware, che lo fornisce "imponendolo" assieme al dispositivo. A volte la scelta dell'utente cade su dispositivi che "hanno a bordo" un sistema operativo già conosciuto, cioè il medesimo sistema operativo che egli utilizza su altre tipologie di hardware, ma nel settore dei dispositivi portatili questo aspetto si traduce nell'esistenza di poche principali alternative.

Sono molti i SO per dispositivi mobili e noi ne descriveremo sinteticamente solo alcuni di essi, tra i più importanti, individuati mediante un "semplice" criterio operativo, cioè analizzando le visite effettuate a un sito primario di e-commerce e riportando, nella figura che segue, la classificazione percentuale dei diversi SO utilizzati dai navigatori con dispositivi mobili.




**AREA DIGITALE**

**I sistemi operativi Palm OS, Symbian e Linux**

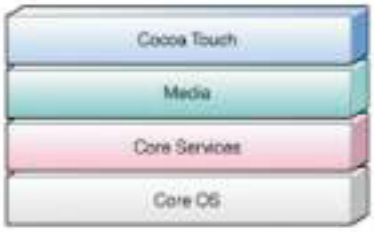
Individuiamo due sistemi operativi che coprono circa il 90% delle visite: Android e OS; per ciascuno di essi illustriamo quali sono i punti di forza e gli "immancabili difetti".

## OS per iPhone e iPad

iOS (fino a giugno 2010 iPhone OS) è il sistema operativo proprietario sviluppato da Apple, rilasciato per la prima volta nel mercato degli Stati Uniti nel giugno 2007. Sviluppato originariamente per iPhone è stato anche utilizzato per gli altri dispositivi Apple (iPad, iPod e Apple TV) fino al settembre 2019, data in cui è stata rilasciato iPadOS uno specifico sistema operativo che sostituisce iOS sui dispositivi iPad.

L'ottima qualità dei prodotti della casa di Cupertino è anche dovuta al fatto che hanno una perfetta integrazione hardware-software dato che chi programma il sistema operativo è lo stesso che ha progettato la componente elettronica.

Il sistema operativo iOS occupa meno di mezzo gigabyte della memoria interna del dispositivo ed è strutturato in quattro livelli di astrazione:
- il Core OS layer;
- il Core Services layer;
- il Media layer;
- il Cocoa Touch layer.




Gli strati più in alto forniscono le astrazioni dei servizi dei livelli più in basso facilitando lo sviluppo delle applicazioni: come in tutti i sistemi organizzati gerarchicamente, ogni livello definisce e implementa nuove funzionalità beneficiando dei servizi offerti dal livello inferiore, indipendentemente dai dettagli implementativi di quest'ultimo. Il Core Services e il Core OS si occupano di offrire i servizi fondamentali del sistema operativo:
- nel livello più basso (Core OS) troviamo lo stack, le funzioni per l'I/O, il threading, la gestione processi ecc. (libreria LibSystem);
- a partire dal livello Core Services in poi troviamo tecnologie sviluppate da Apple, in particolare in questo livello sono presenti i framework di base su cui sono sviluppate molte delle API di livello più alto (framework Open GLES, Quartz Core, Core Animation ecc.);
- il livello Media fornisce principalmente strumenti per lo sviluppo di grafica, per la gestione dell'audio e per la gestione del video;
- al livello più alto troviamo lo strato Cocoa Touch che si compone dei framework UIKit e foundation che rendono disponibili gli strumenti necessari per la realizzazione della grafica, la visualizzazione e l'interazione con l'utente, comprese le classi legate alla gestione dei dati.

L'accesso da parte degli sviluppatori ai livelli inferiori resta comunque garantito.

Tra i punti deboli dei dispositivi Apple, oltre al prezzo decisamente elevato, c'è che è un "sistema chiuso": questo punto, però, è anche un fattore di forza, dato che permette al dispositivo di essere molto performante, pur limitando la libertà dell'utente che non riesce neppure a cambiare le icone oppure il tema degli sfondi.

## Android

Android è un sistema operativo distribuito gratuitamente da Google nel suo codice sorgente: chiunque può prenderlo, modificarlo e metterlo sul suo dispositivo da proporre al mercato.

La filosofia di Android è diametralmente opposta a quella di Apple, decisamente aperta: venne realizzato con l'obiettivo di essere il primo sistema operativo completo di tipo aperto, creato specificamente per i dispositivi mobili e distribuito gratuitamente.

Inizialmente sviluppato da una start-up, Android Inc., venne acquistato da Google nel 2005, si formò un Consorzio di aziende chiamato Open Handset Alliance e nel settembre del 2008 venne rilasciata la sua prima versione, Android 1.0.

## PER SAPERNE DI PIÙ

**OPEN HANDSET ALLIANCE**

Oggi il Consorzio è costituito da molte decine di membri tra i quali:
- operatori telefonici come Vodafone, T-Mobile, Telecom Italia;
- produttori di dispositivi mobili come Motorola, Samsung, Marvell e altri;
- produttori di semiconduttori come Intel, Texas Instruments;

oltre a compagnie di sviluppo e di commercializzazione software.

Per poter essere completamente open fu scelto come kernel del sistema operativo il kernel di Linux, sistema operativo open per antonomasia, e proprio per questa caratteristica di apertura consente la portabilità di Android su differenti piattaforme hardware e si presta a essere modificato e personalizzato in modo da avere una maggiore integrazione con l'hardware.

Queste sono alcune delle motivazioni che hanno spinto grandi produttori di elettronica come LG, Sony, Samsung ecc. a sceglierlo per i loro smartphone.

Anche l'architettura di Android è strutturata in quattro livelli e verrà descritta nella prossima Lezione; in questa Unità di Approfondimento sarà anche illustrato un ambiente di sviluppo per realizzare applicazioni per questo sistema operativo.

Abbiamo scelto di sviluppare app per Android dato che copre circa il 50% del mercato.

# Ambienti di sviluppo per dispositivi mobili

Per i sistemi operativi sopra descritti riportiamo di seguito le principali caratteristiche dei più utilizzati ambienti di sviluppo per il software applicativo.

## SDK per iOS

Il primo Software Development Kit (SDK), messo a disposizione da Apple per gli sviluppatori di terze parti, fu annunciato da Steve Jobs in persona il 17 ottobre 2007 ed è stato rilasciato il 6 marzo 2008 col nome di Xcode: questo software permette agli sviluppatori di creare applicazioni per iPhone e iPad touch e testarle in un simulatore di iPhone (iPhone Simulator).

Unico inconveniente è che per poter caricare un'applicazione nei dispositivi è necessario iscriversi (a pagamento) all'iPhone Developer Program: il prezzo delle applicazioni è stabilito dal programmatore ma l'unico canale di vendita è App Store, e Apple si trattiene per questo servizio il 30% del prezzo richiesto. Tuttavia i programmatori possono anche optare per rilasciare l'applicazione gratis: in tale caso non è previsto nessun costo di rilascio o distribuzione.

**AREA DIGITALE**

**Set che compongono Xcode**

Un vantaggio dell'ambiente di sviluppo Xcode è che consente lo sviluppo di applicazioni per macOS e come linguaggi di programmazione supporta l'Objective C, l'AppleScript, il C++, l'Objective C++ e Java.
Forse la caratteristica più significativa di Xcode è che supporta la distribuzione in rete del lavoro di compilazione tramite Bonjour e Xgrid: è possibile compilare un progetto su più computer riducendo i tempi e, inoltre, la compilazione è di tipo incrementale, cioè il codice viene compilato mentre viene scritto, ottimizzando i tempi di compilazione.
L'ambiente Cocoa, con la sua suite integrata di componenti software orientate agli oggetti che permette di creare rapidamente software ricchi di caratteristiche, può anche essere utilizzato con altri ambienti di sviluppo che supportano linguaggi, come Perl, Python (mediante il bridge PyObjC) e Ruby (tramite RubyCocoa).

Le classi Cocoa sono ricche di funzionalità, coprono praticamente tutte le necessità di ogni sviluppatore e permettono a terze parti di ottenere uniformità tra le applicazioni sviluppate, soprattutto per quanto riguarda l'approccio alle interfacce grafiche.

## Java per Android

Android mette a disposizione degli sviluppatori il Software Development Kit (SDK), che è un insieme di tool di programmazione che contiene al suo interno le librerie relative alle varie versioni della piattaforma Android; inoltre ha una propria macchina virtuale, in modo da eseguire il codice indipendentemente dalle altre applicazioni che sono in esecuzione sulla macchina dello sviluppatore.
Le applicazioni per i dispositivi che utilizzano Android come sistema operativo sono scritte in Java e, sostanzialmente, vengono realizzate in due ambienti di sviluppo:

1. Eclipse: è stato il primo ed è il più diffuso prodotto utilizzato nel mondo Java; è caratterizzato dal fatto di avere a disposizione molteplici plug-in, cioè componenti software ideati per specifici scopi, che rendono possibile l'ampliamento e la personalizzazione di un generico programma da parte di terzi (ricordiamo ADT, Android Development Tools che contiene tutte le classi e le librerie necessarie per poter sviluppare applicazioni Android).
2. Android Studio: è il più recente ambiente di sviluppo integrato (IDE) e sta diventando una soluzione sempre più popolare in quanto offre diversi vantaggi rispetto a Eclipse, per esempio quello di essere più semplice; inoltre, a breve, diventerà l'unico ambiente di sviluppo ufficialmente supportato da parte di Google.

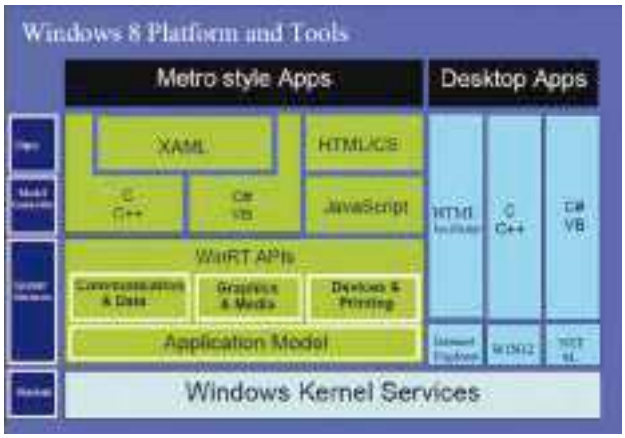
> Dopo averli provati entrambi la nostra scelta è stata quella di Android Studio, sia per motivi di performance, in quanto riduce drasticamente i tempi di progettazione e di testing, sia per portatilità, dato che è compatibile con tutte le piattaforme più utilizzate, e cioè Windows, macOS, Linux.

Nella sezione dedicata all'attività laboratoriale sono disponibili alcuni testi introduttivi che ci permetteranno di sviluppare applicazioni complete.

## App per Windows

Con il rilascio di Windows 10 il colosso informatico di Redmond ha realizzato il concetto di "One Windows", cioè lo sviluppo di un sistema operativo in grado di funzionare allo stesso modo su dispositivi completamente diversi l'uno dall'altro: dai tradizionali desktop e notebook fino ad arrivare ai tablet.

> La piattaforma UWP (Universal Windows Platform) è stata introdotta con Windows 8, ma solo con Windows 10 è disponibile per tutte le famiglie di dispositivi: è possibile realizzare app che "girano" perfettamente in tutti i dispositivi con un solo set di API (app universali distribuite in un pacchetto con estensione Appx).




L'architettura runtime di Windows 10 è la stessa di Windows 8.
Un'app universale è legata alla piattaforma tramite Metro che nella versione 10 prende il nome di Apps Store: è gestito da uno strato di esecuzione chiamato Windows Runtime o WinRT.
Apps Store supporta quattro metodologie di sviluppo già note agli utenti di .NET: ai linguaggi C#, Visual Basic o C++ è possibile integrare HTML, JavaScript o programmazione grafica accelerata usando DirectX.
Microsoft Visual Studio 2019 fornisce un modello di app UWP per ogni linguaggio e, una volta completato il progetto, permette di generare un pacchetto dell'app e inviarlo direttamente a Windows Store per rendere la app disponibile pubblicamente in modo che possa essere installata in qualsiasi dispositivo Windows 10.

## Xamarin

La continua evoluzione dei dispositivi mobili ha spinto gli sviluppatori di app a studiare i meccanismi dei principali sistemi operativi mobili per poter distribuire le proprie app su ogni market e a realizzare framework per lo sviluppo di app cross-platform.
Per risolvere questo problema nasce Xamarin con l'idea di offrire agli sviluppatori un modo semplice e veloce per creare app cross-platform basate su un unico linguaggio orientato agli oggetti attraverso ambienti di sviluppo prestanti: il linguaggio C#.
Nel 2016 Xamarin è stato acquisito da Microsoft dando inizio a una nuova era per lo sviluppo di app cross-platform; inoltre, nonostante il "suo proprietario", Xamarin è ufficialmente open source e gratuito per sviluppatori indipendenti e piccoli team, che possono per utilizzarlo nei propri progetti.

Xamarin è un framework per lo sviluppo di app native e cross-platform con C# che offre pieno supporto alle piattaforme Android e iOS.

Xamarin si compone di tre componenti principali: Xamarin.Android, Xamarin.iOS e Xamarin.Forms; grazie a queste componenti è possibile gestire in C# tutte le caratteristiche di Android e iOS dall'interfaccia utente alle risorse hardware del dispositivo.

## React Native

Negli ultimi anni si è affermata la nuova tecnologia React Native, che permette lo sviluppo di applicazioni native usando JavaScript: questo è un framework che consente di unificare l'esperienza di sviluppo su diversi sistemi operativi.

Quindi il vantaggio principale di React Native è la notevole semplificazione dei processi di sviluppo, poiché consente di elaborare applicazioni per Android e iOS usando un unico codice.

**React**
React è una libreria per JavaScript per realizzare interfacce utente.

React Native è un progetto basato su **ReactJS**, che adotta un nuovo approccio allo sviluppo Web: ReactJS si basa sul concetto di componente che è un qualcosa che ha uno stato, che può accettare delle proprietà e che ha un suo ciclo di vita. Questo approccio di sviluppo **"atomico"** risulta perfetto per effettuare Unit Test o per creare componenti più complessi includendone altri più semplici.
Il suo promotore principale è Facebook; inoltre società come Instagram, Airbnb, Skype, Tesla o Uber stanno già puntando su questo **framework** per le loro app.

## Unity

Con lo sviluppo delle app si è anche rivoluzionato il mercato dei videogame e gli smartphone di nuova generazione si sono trasformati in vere e proprie console spesso in grado di riprodurre versioni di giochi disponibili anche per le comuni PlayStation o Nintendo.

La maggior parte dei videogiochi viene realizzata con dei motori definiti "Videogame Engine" e il più potente e diffuso tra questi è Unity 3D.

L'ambiente di sviluppo di Unity è composto da un motore grafico, un motore fisico molto potente e un live game preview che permette di visualizzare in tempo reale le modifiche apportate al gioco durante le operazioni di programmazione. Con Unity 3D si posso realizzare videogame 3D o altri contenuti interattivi, quali visualizzazioni architettoniche, ambientazioni tridimensionali, short film e piccoli video tridimensionali con costruzioni in tempo reale e animazioni 3D.

# VERIFICA... le conoscenze

## SCELTA MULTIPLA

**1 Le reti mobili in base ai tipi di mobilità possono essere di classe:**
**a** access mobility
**b** terminal mobility
**c** personal mobility
**d** wireless mobility

**2 Nel caso di access mobility esiste una relazione fissa:**
**a** tra l'utente e il proprio terminale
**b** tra l'utente e la propria SIM
**c** tra il terminale e la rete fissa
**d** tra la SIM e l'utente

**3 Nel caso di terminal mobility esiste una relazione fissa:**
**a** tra l'utente e il proprio terminale
**b** tra l'utente e la propria SIM
**c** tra il terminale e la sua SIM
**d** tra la SIM e la rete dell'utente

**4 Nel caso di personal mobility esiste una relazione fissa:**
**a** tra l'utente e il proprio terminale
**b** tra l'utente e la propria SIM
**c** tra il terminale e la rete fissa
**d** tra la SIM e la rete dell'utente

**5 Nei sistemi personal mobility l'acronimo PCS sta a indicare:**
**a** Personal Communications System
**b** Personal Computing Services
**c** Personal Computing System
**d** Personal Communications Services

**6 La trasmissione dei dati digitali avviene a partire dalla:**
**a** 1ª generazione
**b** 2ª generazione
**c** 2,5ª generazione
**d** 3ª generazione

**7 Esistono sostanzialmente quattro tipologie di dispositivi mobili:**
**a** Pocket PC
**b** Tablet PC
**c** Phone PC
**d** Smartphone
**e** UMPC (Ultra Mobile PC)

**8 Il sistema operativo iOS è strutturato in quattro livelli di astrazione:**
**a** il Core OS layer
**b** il Core Services layer
**c** il Touch layer
**d** il Cocoa Touch layer

**9 Chi è stato il promotore principale di React Native?**
**a** Microsoft
**b** Facebook
**c** Google
**d** Yahoo
**e** Amazon
**f** Instagram

## VERO/FALSO

**1** Reti mobili e reti wireless sono sinonimi. V F
**2** I sistemi access mobility sono tipicamente indoor con raggio di azione inferiore ai 500 m. V F
**3** A partire dalla III generazione inizia la diffusione del sistema UMTS. V F
**4** Nel nostro paese la rete 4G ha una velocità di connessione di 1,5 Mbps. V F
**5** La 5G porterà la velocità massima teorica di navigazione con smarthphone e tablet a 20 Gbps. V F
**6** iOS oltre che per iPhone e ora è utilizzato anche negli iPod, iPad e Apple TV. V F
**7** Il Cocoa Touch layer è uno dei quattro livelli di astrazione di Android. V F
**8** Il primo SDK messo a disposizione da Apple fu annunciato nel 2010 da Steve Jobs. V F
**9** Eclipse è stato il primo ed è il più diffuso prodotto utilizzato per il mondo Java. V F
**10** La piattaforma UWP (Universal Windows Platform) è stata introdotta con Windows 7. V F
**11** React Native consente di elaborare applicazioni per Android e iOs usando un unico codice. V F
**12** React è una libreria per JavaScript per realizzare interfacce utente. V F
**13** Unity è una libreria per JavaScript per realizzare interfacce utente. V F
**14** Unity ha un live game preview che permette di visualizzare in tempo reale le modifiche. V F

LIBRO DIGITALE

VERSIONE
SCARICABILE
**EBOOK**

e-ISBN 978-88-360-0472-0

**www.hoepliscuola.it**
Ulrico Hoepli Editore S.p.A.
via Hoepli, 5 - 20121 Milano
e-mail hoepliscuola@hoepli.it