

# Appunti integrativi di Tecnologie di progettazione di sistemi informatici e di telecomunicazione

Classe 5<sup>a</sup>

*Maria Grazia Maffucci  
Giuliano Bellucci  
dal 2016 ad oggi*



*This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/4.0/>.*

*Contact [mariagrazia@maffucci.cc](mailto:mariagrazia@maffucci.cc)*

*I am not the owner of these ideas, I only made a pedagogical transposition.  
Teaching is care.*

# Indice

<b>Indice</b>	<b>2</b>
<b>I socket e la comunicazione in rete</b>	<b>9</b>
Le applicazioni di rete e i socket [LIBRO]	10
CLIL Reading - Custom Networking	10
Esempi di socket TCP	11
ToUpper - Server mono-thread	11
Daytime - Server multi-thread con ExecutorService	15
La connessione tramite socket [LIBRO]	21
Esempi di socket UDP	21
ToUpper UDP	21
<b>Progettazione di protocolli di comunicazione</b>	<b>28</b>
Protocolli di rete	29
Interfacce e servizi	29
Implementazione del servizio	31
CLIL Listening and Writing - Protocols and Layers	33
Progettazione di un protocollo	33
Indirizzamento	34
Frammentazione e riassemblaggio	35
Incapsulamento	35
Controllo della connessione	35
Fasi di una connessione	36
Servizio affidabile	36
Controllo degli errori	37
Controllo del flusso	37
Multiplexing e demultiplexing	37
Servizi di trasmissione	38
Esempi di protocolli applicativi di rete	38
Esempi di progettazione di protocolli applicativi	39
Monitoraggio Azioni	39
TFTP	40
Protocollo di gestione dei dati del vento	42
Testo completo	42
Monitoraggio Vento	43
Richiesta	43
Soluzione	44
Vento in Cloud	50
Richiesta	50
Soluzione	50
Vento nel Mondo	54
Richiesta	54
Soluzione	54

Protocolli applicativi di rete esistenti	58
Protocolli applicativi di rete implementabili	59
Esempi di protocolli applicativi di rete implementabili	59
Un protocollo di spedizione all'Esame di Stato	60
Protocols - CLIL	60
CLIL Listening and Writing - Protocols	60
<b>I sistemi distribuiti: modelli architetturali hardware e software</b>	<b>61</b>
I sistemi distribuiti e loro evoluzione [LIBRO]	62
CLIL Listening and Speaking - Quantum Computers	62
CLIL Listening and Speaking - Flynn Taxonomy and Cluster Architecture	62
CLIL Listening and Speaking - Difference Between Software Architecture and Software Design	63
Il middleware	63
CLIL Listening and Speaking - Middleware	64
CLIL Listening and Speaking - n-Tier Architecture	65
<b>Linguaggi per lo scambio dei dati in rete con Java</b>	<b>66</b>
I linguaggi XML e JSON [LIBRO]	67
CLIL Comprehension - What JSON is	67
Creazione di un XML e di un JSON usando Java	67
JAXB	71
DOM - Document Object Model	75
Il parsing di un documento XML [LIBRO]	77
API per la gestione di documenti XML con il linguaggio Java	78
Parsing di un documento XML con DOM	79
Esempi di parsing di un documento XML con DOM	81
Esempio 1	81
Esempio 2	86
Esempio 3	88
<b>Web e HTTP</b>	<b>92</b>
Comunicazione Web e HTTP [Mdn Web Docs]	93
Web e siti	93
Sito Web statico	93
Sito Web dinamico	94
Web server e Application server	95
HTTP (HyperText Transfer Protocol)	96
I metodi HTTP	97
GET request	98
POST request	99
Content Type	101
HTTP status code	101
<b>Applicazioni lato server in Java: servlet</b>	<b>102</b>
Applicazioni lato server con codice separato: CGI e servlet [LIBRO]	103
<b>Web Service REST - Concetti teorici</b>	<b>105</b>
Web Service [LIBRO]	106
Introduzione ai Web Service	106

Ragioni dell'uso dei Web service	108
REST e SOAP	109
CLIL Listening and Speaking - What REST is	111
I principi dell'architettura RESTful	111
CLIL Listening and Writing - Introduction to Web Services REST	112
Identificazione delle risorse	112
CLIL Listening and Writing - REST and HTTP	113
CLIL Listening and Writing - Resource URIs	113
CLIL Listening and Writing - Collection URIs	113
Utilizzo esplicito dei metodi HTTP	113
CLIL Listening and Writing - HTTP Methods	116
Idempotenza dei metodi HTTP	116
CLIL Listening and Writing - Method Idempotence	117
Risorse autodescrittive	117
CLIL Listening and Writing - REST Response	118
Collegamenti tra risorse e il principio HATEOAS	118
CLIL Listening and Writing - HATEOAS	121
Comunicazione senza stato	121
Stato delle risorse e dell'applicazione	122
CLIL Listening - The Richardson Maturity Model	123
REST e sicurezza HTTP (cenni)	124
Sicurezza con "sessioni REST" e tramite servizi di terze parti (cenni)	125
Lo stato come risorsa	125
OpenID e OAuth: gestione esterna della sicurezza	126
<b>Web Service REST usando Java Servlet</b>	<b>127</b>
Esempi di Web service di tipo REST in linguaggio Java	128
Realizzazione di Web service di tipo REST mediante servlet	128
Le applicazioni lato server [LIBRO]	129
Web service REST - CalcolatriceAPI	129
Web service per operazioni CRUD su database	143
Servlet e database [LIBRO]	143
Countries REST su MySQL	144
ShowRoom REST su MySQL	160
Database productsales	163
Server di connessione al database productsales	169
Web service REST SimpleRestDb	182
Client RestClient	192
Rubrica telefonica REST su database (da finire)	195
<b>Web Service REST in Java e l'uso delle annotazioni</b>	<b>200</b>
Web Service REST, Jersey e JAX-RS	201
UnitConverter - Esempio di Web Service REST usando JAX-RS	201
UnitConverter - Creazione di un progetto su NetBeans	201
Progetto Maven	201
Progetto Java Web	205

UnitConverter - Installazione di un Client REST	207
UnitConverter - Sviluppo dell'applicazione	207
CLIL Comprehension - Developing RESTful APIs with JAX-RS	211
Calculator - Esempio di Web Service REST usando JAX-RS	212
Web service per operazioni CRUD usando una struttura dati	215
CountryPopulation - Esempio di Web Service REST usando JAX-RS	216
CLIL Comprehension - Developing RESTful APIs with JAX-RS	225
CLIL Comprehension - Developing RESTful APIs with JAX-RS	225
Web service per operazioni CRUD su database [da finire]	227
JDBC: Java DataBase Connectivity	227
ShowRoom REST su database	227
Database EsempioREST	230
<b>Esercizi</b>	<b>238</b>
Esercizi sui Thread	239
Esercizi sui protocolli applicativi di rete	239
Socket TCP e UDP	239
Aspetti progettuali di un protocollo di comunicazione	255
Protocols - CLIL	263
Esercizi sul XML e JSON	264
Esercitazione di laboratorio	269
Protocolli applicativi di rete - XML - Parser Java - Database	269
Esercizi su Servlet	272
Esercizi di progettazione di architetture distribuite	280
Esercizi sui protocolli	280
Esercizi sui Web service	282
Voli low cost	283
MyShow	285
Realizzazione	286
Sviluppi futuri	286
Centro assistenza	287
Realizzazione	288
GTT (trasporre)	289
Dottorato (trasporre)	290
Olimpiadi (trasporre)	291
Grande Fratello (trasporre)	292
UVI (trasporre)	293
Colloquio	294
<b>Approfondimenti</b>	<b>337</b>
<b>A - Thread e concorrenza in Java - Propedeutica</b>	<b>338</b>
I Thread	339
CLIL Reading - Concurrency	339
Processi e Thread	339
CLIL Reading - Java Concurrency / Multithreading Basics	343
Definizione di thread in Java	343

Alcuni metodi della classe Thread	346
CLIL Reading - Java Thread and Runnable Tutorial	348
Esempi d'uso dei thread	349
Nomi thread e priorità	349
Settenani	350
Runnable e Thread	351
Terminazione di un'applicazione java - Shutdown Hook	353
Esercizi svolti sui thread	355
Moltiplicazione tra matrici	355
Classi e metodi comuni	355
Versione seriale	357
Prima versione concorrente - Un thread per ogni elemento	358
Seconda versione concorrente - Un thread per ogni riga	360
Terza versione concorrente - Un thread per processore	361
La sincronizzazione dei thread	364
Race condition	364
Sezione critica e sincronizzazione	364
InventoryCounter	364
EvenOdd	366
BankAccount	367
Operazioni atomiche e l'istruzione volatile	368
Esempio d'uso di volatile	369
Data race	372
Strategie di locking delle risorse e deadlock [DA FINIRE]	375
ExecutorService	377
CLIL Reading - Executor Service & Thread Pool	377
Esempi d'uso degli Executor	378
Moltiplicazione tra matrici	378
<b>B - Design Patterns - Propedeutica</b>	<b>380</b>
Design Patterns	381
Observer Pattern	381
Esempi visti a lezione	381
Observer	381
Esercizio grafica observer	384
Roulette	388
Bank Account	390
<b>C - CLIL - Introduction to Distributed system</b>	<b>394</b>
Unit 1 - Transport Protocol	395
Vocabulary	395
Language for thinking: defining	396
Anticipation guide	397
Comparison of Transport Protocols: UDP and TCP	398
Activity: Comparison of Transport Protocols	400
Unit 2 - Interprocess communication - Sockets	401

Vocabulary	401
Language for thinking: linking words	403
Interprocess communication characteristics	405
Synchronous and asynchronous communication	405
Transport layer and interprocess communication	407
Activity: Interprocess communication and Transport layer	410
TCP stream communication	412
Activity: TCP stream communication	416
UDP datagram communication	419
Activity: UDP datagram communication	422
Activity: UDP and TCP Sockets	425
CLIL Writing and Speaking - Networking, TCP and UDP protocols	426
CLIL Writing and Speaking - Java sockets	426
<b>D - I sistemi distribuiti: modelli architetturali hardware e software</b>	<b>427</b>
I sistemi distribuiti	428
Caratteristiche dei sistemi distribuiti	429
Affidabilità	429
Integrazione	430
Trasparenza	431
Economicità	431
Complessità	432
Storia dei sistemi distribuiti e modelli architetturali	433
CLIL Listening and Speaking - Quantum Computers	433
Architetture distribuite hardware	434
SISD	434
SIMD	435
MISD	435
MIMD	435
MIMD a memoria fisica condivisa	436
MIMD a memoria fisica distribuita	436
Cluster di PC	437
CLIL Listening and Speaking - Flynn Taxonomy and Cluster Architecture	438
Architetture distribuite software	438
Architettura centralizzata	438
Architettura client-server	439
Architettura multi-tier	439
CLIL Listening and Speaking - n-Tier Architecture	441
Middleware	442
CLIL Listening and Speaking - Middleware	443
<b>E - Vecchi esempi di socket TCP e UDP</b>	<b>448</b>
Daytime - Server iterativo	449
Echo - Server concorrente	455
TFTP	457
<b>F - Android e app</b>	<b>473</b>

Android e dispositivi mobili [LIBRO]	474
CLIL - Developing Android Apps	474
<b>G - Web Service REST usando PHP</b>	<b>475</b>
Creare un client REST	476
XML over HTTP	476
Client REST usando le API di Google	477
PHP REST API	483
CLIL Comprehension - What a RESTful API is	483
CLIL Reading - RESTful API using PHP	483
CLIL Comprehension - PHP REST API	483
<b>H - Vecchi esempi di Web service</b>	<b>485</b>
Rubrica telefonica REST su file	486
<b>I - Immagini colloquio non utilizzabili</b>	<b>508</b>
<b>Bibliografia e sitografia</b>	<b>512</b>
Thread e concorrenza in Java - Propedeutica	513
Design Patterns - Propedeutica	514
I socket e la comunicazione in rete	515
CLIL - Introduction to Distributed system	516
Java e XML	517
HTTP e Servlet	517
I sistemi distribuiti: modelli architetturali hardware e software	518
Web Service REST	518

# I socket e la comunicazione in rete

# Le applicazioni di rete e i socket [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 1 - Architettura di rete e formati per lo scambio dei dati

- **1.5 Le applicazioni di rete pp.44-52**
  - Il modello ISO/OSI e le applicazioni
  - Applicazioni di rete
  - Scelta della architettura per l'applicazione di rete
  - Servizi offerti dallo strato di trasporto alle applicazioni

[CLIL Activity: Unit 1 - Transport Protocol](#)

## Unità 2 - Il socket e la comunicazione con i protocolli TCP/UDP

- **2.1 I socket e i protocolli per la comunicazione di rete pp.106-112**
  - Generalità
  - Le porte di comunicazione e i socket
- **2.2 La connessione tramite socket pp.113-143**
  - Generalità
  - Famiglie e tipi di socket
  - Trasmissione unicast e multicast
  - Lab 1 Java socket
  - Lab 2 Java socket: realizzazione di un server TCP
  - Lab 3 Realizzazione di un server multiplo in Java
  - Lab 4 Java socket: un'animazione client-server

[CLIL Activity: Unit 2 - Interprocess communication - Socket](#)

## Unità 1 - Architettura di rete e formati per lo scambio dei dati

- **1.4 Le applicazioni Web e il modello client-server pp.36-43**
  - Applicazioni Web: generalità
  - Il modello client-server
  - Distinzione tra server e client
  - Livelli e strati

# CLIL Reading - Custom Networking

Read the Oracle' guide "[Custom Networking](#)", it is a simple and complete guide about networking programming.

# Esempi di socket TCP

## ToUpper - Server mono-thread

La seguente classe<sup>1</sup> implementa un semplice server TCP in linguaggio Java che converte in maiuscolo qualsiasi stringa gli venga inviata da un client. Il server è mono-thread ed è in grado di servire un solo client alla volta; l'unico thread attivo è il main thread. Chiaramente questa modalità di progettazione del server è fortemente sconsigliata in quanto un server deve essere in grado di rispondere a più client, senza attendere la terminazione del servizio offerto ad un client prima di servirne uno nuovo.

### ServerTCP.java

```
import java.io.*;
import java.net.*;

/**
 *
 * @author mgm
 */
public class ServerTCP {

    private final static int PORTA_SERVER = 6789; // porta del server
    private ServerSocket serverSocket; // socket del server
    private Socket clientSocket; // socket del client
    private String stringaRicevuta; // stringa ricevuta dal client
    private String stringaInviata; // stringa inviata dal server
    private BufferedReader in; // stream di input dal client
    private PrintWriter out; // stream di output verso il client

    public ServerTCP() {
        try {
            // creazione del socket del server sulla porta 6789
            serverSocket = new ServerSocket(PORTA_SERVER);
        } catch (IOException ex) {
            System.err.println("Errore interno del server " + ex.getMessage());
        }
    }

    public void attendi() {
        System.out.println("1) SERVER: in esecuzione");
        try {
            // attesa della connessione da parte del client
            clientSocket = serverSocket.accept();
            serverSocket.close(); // chiusura del socket per inibire richieste da parte
di altri client
        }
    }
}
```

<sup>1</sup> La classe è quella presente sul libro di testo P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2017, a p.138 con alcune modifiche sul codice proposto.

```
in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
out = new PrintWriter(clientSocket.getOutputStream());
} catch (IOException ex) {
    System.err.println("Errore di connessione del client " + ex.getMessage());
}
}

public void comunica() {
    try {
        out.println("3) SERVER: scrivi una frase da trasformare");
        out.flush();
        stringaRicevuta = in.readLine(); // attesa della stringa dal client
        System.out.println("6) SERVER: ricevuta la stringa dal client " +
stringaRicevuta);

        // modifica della stringa
        stringaInviata = stringaRicevuta.toUpperCase();
        System.out.println("7) SERVER: invio della stringa al client: " +
stringaInviata);
        out.println(stringaInviata);
        out.flush();
        // terminazione elaborazione sul server e chiusura della connessione con il
client
        System.out.println("9) SERVER: fine elaborazione e chiusura connessione");
        clientSocket.close();
    } catch (IOException ex) {
        System.err.println("Errore di comunicazione " + ex.getMessage());
    }
}

public static void main(String[] args) {
    ServerTCP server;

    server = new ServerTCP();
    server.attendi();
    server.comunica();
}
}
```

Nel server il metodo `main()` esemplifica l'uso della classe `ServerTCP` dove vengono eseguite tutte le fasi di creazione del server, attesa di connessione da parte di un client e comunicazione della stringa modificata.

La classe seguente<sup>2</sup> invece rappresenta un semplice client che invia al server una stringa da convertire in maiuscolo. Si osservi che le singole fasi sono state evidenziate con delle frasi

---

<sup>2</sup> La classe è quella presente sul libro di testo P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2017, a p.135 con alcune migliorie sul codice proposto.

numerate cronologicamente in modo da riconoscere i diversi momenti di azione e di attesa, sia del server, sia del client.

### ClientTCP.java

```
import java.io.*;
import java.net.*;

public class ClientTCP {

    // indirizzo del server
    private final static String NOME_SERVER = "127.0.0.1";
    private final static int PORTA_SERVER = 6789;      // porta del server
    private Socket clientSocket;           // socket del client
    private BufferedReader tastiera;       // buffer per l'input da tastiera
    private String stringaUtente;         // stringa inserita dall'utente
    private String stringaServer;         // stringa ricevuta dal server
    private PrintWriter out;              // stream di output verso il server
    private BufferedReader in;             // stream di input dal server

    public ClientTCP() throws IOException, UnknownHostException {
        // creazione del socket
        clientSocket = new Socket(NOME_SERVER, PORTA_SERVER);
        // creazione dello stream di output
        out = new PrintWriter(clientSocket.getOutputStream());
        // creazione dello stream di input
        in = new BufferedReader(new
                               InputStreamReader(clientSocket.getInputStream()));
    }

    public void connetti() {
        System.out.println("2) CLIENT: partito in esecuzione e creazione della
conessione");
        // istanziazione dell'oggetto che gestisce l'input da tastiera
        tastiera = new BufferedReader(new InputStreamReader(System.in));
    }

    public void comunica() {
        try {
            stringaUtente = in.readLine();
            System.out.println(stringaUtente);
            System.out.println("4) CLIENT: inserire una stringa da trasmettere al
server");
            stringaUtente = tastiera.readLine(); // lettura di una riga da tastiera
            System.out.println("5) CLINET: invio della riga al server e attesa della
risposta");
            out.println(stringaUtente); // invio della stringa verso il server
            out.flush();
            stringaServer = in.readLine(); // lettura della risposta del server
        }
    }
}
```

```
        System.out.println("8) CLIENT: risposta del server\n" + stringaServer);
        System.out.println("9) CLIENT: termina elaborazione e chiusura della
connessoione");
        clientSocket.close();
    } catch (IOException ex) {
        System.err.println("Errore di comunicazione in fase di scambio");
    }
}

public static void main(String[] args) {
    try {
        ClientTCP client = new ClientTCP();
        client.connettiti();
        client.comunica();
    } catch (UnknownHostException ex) { // generabile dal metodo Socket()
        System.err.println("Host sconosciuto " + ex.getMessage());
    } catch (IOException ex) { // generabile dal metodo getOutputStream()
        System.err.println("Errore di comunicazione in fase di connessione " +
ex.getMessage());
    }
}
```

# Daytime - Server multi-thread con ExecutorService<sup>3</sup>

La seguente classe<sup>4</sup> implementa un semplice server TCP in linguaggio Java. Il server restituisce al client, che effettua la connessione, una stringa di caratteri ASCII riportante l'indicazione della data e ora correnti. Il server usa un Executor per gestire le richieste dei client in modo concorrente e il pool di thread gestito dallo Executor è pari al numero di processori disponibili alla JVM.

La classe **Main.java** crea un oggetto server e lo avvia come thread indipendente. All'interno del main thread sarà possibile interrompere il server premendo un qualsiasi tasto nella console.

Viene usato il metodo **join()** per fare in modo che il main thread non termini prima della conclusione dell'oggetto server.

## Main.java

```
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        int c;
        Thread thread;

        try {
            DayTimeServer daytimeServer = new DayTimeServer(4444);
            thread = new Thread(daytimeServer);
            thread.start();
            // the server will stop pressing any button on the console
            c = System.in.read();
            // the server is blocked
            daytimeServer.shutdownServer();
            // the main server waits the termination of child threads
            thread.join();
            System.out.println("DAYTIME SERVER - The main thread is ended");
        } catch(IOException ex) {
            System.err.println("Error " + ex.getMessage());
        } catch(InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

La classe **DayTimeServer.java** è deputata alla creazione dell'oggetto Executor al quale verranno passati tutti i task da eseguire per servire client diversi. All'interno della classe è anche gestito un attributo di tipo [volatile](#)<sup>5</sup> usato per permettere l'interruzione del server dal main thread. La keyword **volatile** è una delle tante tecniche per permettere la sincronizzazione tra thread nella programmazione concorrente.

<sup>3</sup> [ExecutorService](#) spiegati negli appunti.

<sup>4</sup> La versione precedente di questo programma è possibile trovarla negli Approfondimenti.

<sup>5</sup> [Java Concurrency and Multithreading Tutorial](#)

## DayTimeServer.java

```
import java.io.*;
import java.net.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class DayTimeServer implements Runnable {
    // the final keyword is used to define an entity that can only be assigned once
    private final ServerSocket serverSocket;      // server socket
    private final ExecutorService pool;           // Executor managing the pool thread

    // The stopped Boolean variable is declared as volatile because it will
    // be changed from another thread. The Java volatile keyword is used to
    // mark a Java variable as "being stored in main memory". More precisely
    // that means, that every read of a volatile variable will be read from
    // the computer's main memory, and not from the CPU cache, and that
    // every write to a volatile variable will be written to main memory,
    // and not just to the CPU cache. It is enough the use of volatile
    // keyword because only the main thread can stop the server, so no race
    // conditions happen.
    private static volatile boolean stopped = false;

    /**
     * Constructor that accepts network connection to any NIC.
     * @param port server port number
     * @throws IOException
     */
    public DayTimeServer(int port) throws IOException {
        // A null address specify that the server accepts network connection to
        // any NIC.
        serverSocket = new ServerSocket(port);
        // Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
        // With this option set to a non-zero timeout, a read() call on the
        // InputStream associated with this Socket will block for only this
        // amount of time. If the timeout expires, a
        // java.net.SocketTimeoutException is raised, though the Socket is still
        // valid. The option must be enabled prior to entering the blocking
        // operation to have effect. The timeout must be greater than 0.
        // A timeout of zero is interpreted as an infinite timeout.
        serverSocket.setSoTimeout(10000);
        // It is created an Executor that uses a number of thread that is equal to the
        // number of available processors. You can decide a different number.
        pool = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
        System.out.println("DAYTIME SERVER - Initialization completed.");
    }

    /**
     * Start a thread to serve a client
     */
    @Override
    public void run() {
        // The server is going to offer continuously the service needed, until
        // it is stopped. When it receives a stop command from the main thread,
        // it needs to shutting down itself, but the server is waiting in the
        // accept() method of the serverSocket object. To force the server to
        // leave that method, we need to explicitly close the server
    }
}
```

```
// (we'll do that in the shutdownServer() method).
do {
    try {
        System.out.println("DAYTIME SERVER - Accepting...");
        // the server will wait for a client connection for 100 seconds
        Socket clientSocket = serverSocket.accept();
        System.out.println("DAYTIME SERVER - Accepted");
        // this object is the one that offer the server service
        RequestedTask task = new RequestedTask(clientSocket);
        // the new task is submitted to the Executor
        pool.submit(task);
    } catch (IOException e) {
        System.out.println("DAYTIME SERVER - Something happen...");
    }
} while (!stopped); // this thread will continue until it is not stopped
}

public void shutdownServer() {
    stopped = true;
    System.out.println("DAYTIME SERVER - Shutting down the server...");
    System.out.println("DAYTIME SERVER - Shutting down executor...");
    // The executor stops accepting new tasks, waits for previously submitted
    // tasks to execute, and then terminates the executor.
    pool.shutdown();
    // When the server has finished its execution (leaving the loop), it has to
    // wait for the finalization of the executor using the awaitTermination()
    // method. This method will block this thread until the executor has finished
    // its execution() method.
    try {
        pool.awaitTermination(5, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("DAYTIME SERVER - Executor closed");
    System.out.println("DAYTIME SERVER - Closing socket...");
    try {
        // close server socket
        serverSocket.close();
        System.out.println("DAYTIME SERVER - Socket closed");
    } catch (IOException e) {
        // shutdownNow() method interrupts the running task and shuts down
        // the executor immediately
        pool.shutdownNow();
        e.printStackTrace();
    }
}
}
```

Infine la classe **RequestedTask.java** implementa l'effettivo servizio offerto dal server, in questo caso l'invio della data e dell'ora del server. Ovviamente il task deve essere gestito come un thread e quindi la classe implementerà l'interfaccia **Runnable** e il metodo **run()** che all'interno avrà tutte le istruzioni per la gestione del servizio.

### RequestedTask.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;
```

```
import java.util.Date;

public class RequestedTask implements Runnable {
    close client socket
        clientSocket.close();
    Sys// client socket
private final Socket clientSocket;

public RequestedTask(Socket client) {
    this.clientSocket = client;
}

//Create a task serving a client
@Override
public void run() {
    try {
        // server is printing client IP address and port number
        System.out.println("DAYTIME SERVER - Serving: " +
            clientSocket.getInetAddress().toString() +
            ":" + clientSocket.getPort() + " ...");
        // creation of an output stream
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream());
        // current date and time
        Date now = new Date();
        // server sends to the client a string containing current date and
        // time
        out.println(now.toString() + "\r\n");
        // the flush() method flushes the stream; if the stream has saved any
        // characters from the various println() methods in a buffer, write
        // them immediately to their intended destination
        out.flush();
        // close output stream
        out.close();
        // tem.out.println("DAYTIME SERVER - Served: " +
        //     clientSocket.getInetAddress().toString() +
        //     ":" + clientSocket.getPort());
    } catch (IOException ex) {
        System.err.println("DAYTIME SERVER - Task error" + ex.getMessage() + " " + ex.toString());
        Thread.currentThread().interrupt();
    } finally {
        if (clientSocket != null) {
            try {
                // close client socket
                clientSocket.close();
            } catch (IOException ex) {
                System.err.println("DAYTIME SERVER - Finally error in task");
            }
        }
    }
}
}
```

**Il funzionamento del server può essere verificato impiegando un qualsiasi client Telnet;** la stringa contenente la data e l'ora sarà ricevuta dal client subito dopo aver effettuato la connessione. Immediatamente dopo l'invio della stringa al client il server chiuderà la connessione.

```
mgm@umgm:~$ telnet 127.0.0.1 4444
```

```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Tue Oct 02 12:23:14 CEST 2018
```

Connection closed by foreign host.

mgm@umgm:~\$

La classe **DayTimeClient.java** è invece un esempio di client creato ad hoc per interagire con il server **DayTimeServer.java**.

### DayTimeClient.java<sup>6</sup>

```
import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

public class DayTimeClient {

    private String serverName; // server address
    private int serverPort; // server port

    /**
     * Constructor using the server IP address and port number
     * @param server server IP address
     * @param port server port number
     */
    public DayTimeClient(String server, int port) {
        serverName = server;
        serverPort = port;
    }

    /**
     * Client requests for day-time server
     * @return the actual server day and time
     * @throws java.net.SocketTimeoutException
     * @throws IOException
     */
    public String getDayTime() throws SocketTimeoutException, IOException {
        BufferedReader in; // client input stream
        String answer = ""; // used for server response

        // client socket without any indication of server IP address and port
        // number
        Socket clientSocket = new Socket();
        // creation of an inet socket address using server parameters
        InetSocketAddress serverAddress
            = new InetSocketAddress(serverName, serverPort);
        // Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
        // With this option set to a non-zero timeout, a read() call on the
        // InputStream associated with this Socket will block for only this
        // amount of time. If the timeout expires, a
```

<sup>6</sup> La classe è quella presente sul libro di testo G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni - per Informatica*, vol. 3, ed. Zanichelli, 2017, 2a edizione, p.112 con alcune modifiche sul codice proposto.

```
// java.net.SocketTimeoutException is raised, though the Socket is still
// valid. The option must be enabled prior to entering the blocking
// operation to have effect. The timeout must be greater than 0.
// A timeout of zero is interpreted as an infinite timeout.
clientSocket.setSoTimeout(100000);
// client connection request and it will wait at least 100 seconds
// for server connection
System.out.println("DAYTIME CLIENT - Client connecting...");
clientSocket.connect(serverAddress, 100000); // 100000ms = 100s
// client input stream
in = new BufferedReader(
    new InputStreamReader(clientSocket.getInputStream()));
System.out.println("DAYTIME CLIENT - Client waiting server response...");
answer = in.readLine();
System.out.println("DAYTIME CLIENT - Client closing input stream and socket...");
// client stream closing
in.close();
// client socket closing
clientSocket.close();
System.out.println("DAYTIME CLIENT - Client socket and input stream closed");
return answer;
}

public static void main(String args[]) {
    String server; // server IP address
    int port; // server port number
    String daytime; // server day and time
    DayTimeClient client; // object DayTimeClient

    server = JOptionPane.showInputDialog("Indirizzo del server");
    port = Integer.parseInt(JOptionPane.showInputDialog("Porta del server"));

    try {
        client = new DayTimeClient(server, port);
        daytime = client.getDayTime();
        JOptionPane.showMessageDialog(null, daytime, "Server day and time",
JOptionPane.QUESTION_MESSAGE);
    } catch (SocketTimeoutException ex) {
        System.err.println("Nessuna risposta dal server");
    } catch (IOException ex) {
        System.err.println("Errore di comunicazione");
        Logger.getLogger(DayTimeClient.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}
```

# La connessione tramite socket [LIBRO]

[CLIL Activity: Unit 2 - Interprocess communication - Socket](#)

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 2 - Il socket e la comunicazione con i protocolli TCP/UDP

- **2.2 La connessione tramite socket pp.144-159**

- Lab 5 Il protocollo UDP nel linguaggio Java
- Lab 6 Applicazioni multicast in Java
- Lab 7 Un esempio completo con i Java socket: "La chat"

## Esempi di socket UDP

### ToUpper UDP

Una comunicazione che usa il protocollo UDP si basa su un servizio a livello di trasporto veloce, non connesso e non affidabile; i singoli datagrammi vengono inviati senza alcuna garanzia di consegna e di ordine.

Un server che usa a livello di trasporto i servizi del protocollo UDP non implementa una coda di servizio, quindi i client potranno provare a comunicare con il server, ma in caso di eccessivo carico del server è possibile che i tentativi di comunicazione dei client falliscano.

L'utilizzo dei **socket UDP non fornisce alcuna garanzia di ricezione del datagramma: qualsiasi controllo** si voglia fare sulla ricezione dei dati, sulla eventuale conferma ed eventuale creazione di una connessione, sulla eventuale necessità di frammentare i dati e riassemblarli, sul controllo del flusso e il controllo degli errori **dovrà essere necessariamente fatto dal protocollo a livello applicativo**.

Le seguenti classi implementano un semplice server UDP in linguaggio Java che converte in maiuscolo qualsiasi stringa gli venga inviata da un client. Il server gestisce una comunicazione multi-thread usando il framework ExecutorService.

#### Main.java

```
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        int c;
        Thread thread;

        try {
            ToUpperUDPServer toUpperUDPServer = new ToUpperUDPServer(7777);
            thread = new Thread(toUpperUDPServer);
            thread.start();
        }
```

```
// the server will stop pressing any button on the console
c = System.in.read();
// the server is blocked
toUpperCaseUDPServer.shutdownServer();
// the main server waits the termination of the children threads
thread.join();
System.out.println("TOUPPER SERVER - The main thread is ended");
} catch(IOException ex) {
    System.err.println("Error " + ex.getMessage());
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
}
```

### ToUpperUDPServer.java

```
import java.io.*;
import java.net.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ToUpperUDPServer implements Runnable {
    // the final keyword is used to define an entity that can only be assigned once
    private final DatagramSocket serverSocket;      // server socket
    private final ExecutorService pool;              // Executor managing the pool thread

    // The stopped Boolean variable is declared as volatile because it will
    // be changed from another thread. The Java volatile keyword is used to
    // mark a Java variable as "being stored in main memory". More precisely
    // that means, that every read of a volatile variable will be read from
    // the computer's main memory, and not from the CPU cache, and that
    // every write to a volatile variable will be written to main memory,
    // and not just to the CPU cache. It is enough the use of volatile
    // keyword because only the main thread can stop the server, so no race
    // conditions happen.
    private static volatile boolean stopped = false;

    /**
     * Constructor accepts network connection to any NIC.
     * @param port server port number
     * @throws IOException
     */
    public ToUpperUDPServer(int port) throws IOException {
        serverSocket = new DatagramSocket(port);
        serverSocket.setSoTimeout(10000);
        pool = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
        System.out.println("TOUPPER SERVER - Initialization completed.");
    }

    /**
     * Start a thread to serve a client
     */
    @Override
    public void run() {
```

```
// The server is going to offer continuously the service needed, until
// it is stopped. When it receives a stop command from the main thread,
// it needs to shutting down itself. We explicitly close the server using
// the shutdownServer() method in the main thread.
do {
    try {
        // The input buffer is initialized to the datagram maximum dimension
        // defined in the protocol. In this case the value 8192 is randomly
        // chosen.
        byte[] bufferIN = new byte[8192];    // input buffer
        // Creation of a datagram packet that will be used for the client
        // datagram.
        DatagramPacket clientPacket = new DatagramPacket(bufferIN,
                                                        bufferIN.length);
        System.out.println("TOUPPER SERVER - Waiting...");
        // The server will wait for a client datagram for 100 seconds;
        // this is a blocking instruction.
        serverSocket.receive(clientPacket);
        // This object is the one that offers the server service.
        RequestedTask task = new RequestedTask(serverSocket, clientPacket);
        // The new task is submitted to the Executor
        pool.submit(task);
    } catch (IOException e) {
        System.out.println("TOUPPER SERVER - Something happen...");
    }
} while (!stopped); // this thread will continue until it is not stopped
}

public void shutdownServer() throws IOException {
    stopped = true;
    System.out.println("TOUPPER SERVER - Shutting down the server...");
    System.out.println("TOUPPER SERVER - Shutting down executor...");
    // The executor stops accepting new tasks, waits for previously submitted
    // tasks to execute, and then terminates the executor.
    pool.shutdown();
    // When the server has finished its execution (leaving the loop), it has to
    // wait for the finalization of the executor using the awaitTermination()
    // method. This method will block this thread until the executor has finished
    // its execution() method.
    try {
        pool.awaitTermination(5, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("TOUPPER SERVER - Executor closed");
    System.out.println("TOUPPER SERVER - Closing socket...");
    // Termination of the server socket
    serverSocket.close();
    System.out.println("TOUPPER SERVER - Socket closed");
}
}
```

## RequestedTask.java

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class RequestedTask implements Runnable {
```

```
private DatagramSocket serverSocket;
private DatagramPacket clientPacket;      //  packet received from a client
private DatagramPacket serverPacket;      //  packet that will be sent to a client

public RequestedTask(DatagramSocket serverSocket,
                     DatagramPacket client) {
    this.serverSocket = serverSocket;
    this.clientPacket = client;
}

//Create a task serving a client
@Override
public void run() {
    // The server is printing client IP address and port number
    System.out.println("TOUPPER SERVER - Serving: " +
        clientPacket.getAddress() +
        ":" + clientPacket.getPort() + " ...");
    // Data received are saved into the input buffer for future
    // manipulation.
    byte[] bufferIN = clientPacket.getData();
    // In order to change data they are saved in a String object.
    String received = new String(bufferIN);
    // Saving the actual number of characters received.
    int numChar = clientPacket.getLength();
    // Extraction of the substring starting from first position and
    // ending at numChar position. Exceeding characters are thrown
    // away.
    received = received.substring(0, numChar);
    System.out.println("TOUPPER SERVER - Recived from " + clientPacket.getAddress()
        + ":" + clientPacket.getPort() + "; " + received);
    // All characters are transformed in upper case notation.
    String toSend = received.toUpperCase();
    System.out.println("TOUPPER SERVER - Sending to " + clientPacket.getAddress()
        + ":" + clientPacket.getPort() + "; " + toSend);
    // Manipulated string is converted in an array of byte, as
    // requested by the class DatagramPacket.
    byte[] bufferOUT = toSend.getBytes();
    // Creation of the datagram that will be sent.
    serverPacket = new DatagramPacket(bufferOUT, bufferOUT.length,
        clientPacket.getAddress(),
        clientPacket.getPort());
    try {
        // The datagram is sent to the client.
        serverSocket.send(serverPacket);
    } catch (IOException ex) {
        System.out.println("TOUPPER SERVER - Exception on sending data");
        Thread.currentThread().interrupt();
    }
}
```

Per testare il server utilizzando un client a linea di comando, non sarà più possibile usare **telnet** in quanto questo protocollo applicativo usa TCP a livello di trasporto. Si potrà comunque testare il server utilizzando l'applicativo di rete **netcat**, spesso abbreviato con **nc**. L'utilità di rete **netcat** permette di scrivere e leggere su una connessione di rete, sia usando TCP, sia usando UDP.

```
mgm@umaffu:~$ nc -u localhost 7777
prima prova
PRIMA PROVA
seconda prova
SECONDA PROVA
terza prova
TERZA PROVA
^C
mgm@umaffu:~$
```

La seguente classe implementa un semplice client UDP in linguaggio Java che invia una stringa al server precedente per farla convertire in maiuscolo.

### ToUpperUDPClient.java

```
import java.io.*;
import java.net.*;

public class ToUpperUDPClient {

    private DatagramSocket clientSocket;

    /**
     * Constructor.
     */
    public ToUpperUDPClient() throws SocketException {
        // Client socket creation.
        clientSocket = new DatagramSocket();
        // The receive method will wait no more than 10 second in order to
        // receive a datagram.
        clientSocket.setSoTimeout(10000);
    }

    /**
     * Close the client socket.
     */
    public void closeToUpperUDPClient() {
        clientSocket.close();
    }

    /**
     * Send to the server the string to convert in upper case.
     * @param requestToServer the string that has to be converted
     * @param serverHost the server name/address
     * @param serverPort the server port
     * @throws UnknownHostException
     * @throws IOException
     * @throws SocketTimeoutException
     */
    public void sendToServer(String requestToServer,
                           String serverHost,
                           int serverPort)
            throws UnknownHostException,
            IOException,
            SocketTimeoutException {
        byte[] buffer;                      // output buffer
```

```
DatagramPacket datagram; // datagram used to send data

// Destination IP address.
InetAddress serverAddress = InetAddress.getByName(serverHost);
// Check the client socket
if (clientSocket.isClosed()) {
    throw new IOException();
}
// The string given is transformed in an array of byte, as
// requested by the class DatagramPacket. The charset used is UTF-8
// (ISO-8859-1)
buffer = requestToServer.getBytes("UTF-8");
// creation of the requesting datagram.
datagram = new DatagramPacket(buffer, buffer.length, serverAddress,
    serverPort);
// The datagram is sent.
clientSocket.send(datagram);
}

/**
 * Receive from the server the upper case converted string.
 * @param serverHost the server name/address
 * @param serverPort the server port
 * @return
 * @throws UnknownHostException
 * @throws IOException
 * @throws SocketTimeoutException
 */
public String receiveFromServer(String serverHost,
                                int serverPort)
    throws UnknownHostException,
    IOException,
    SocketTimeoutException {
    DatagramPacket datagram; // datagram used to receive data
    String answerFromServer; // string received from the server

    // Destination IP address.
    InetAddress serverAddress = InetAddress.getByName(serverHost);
    // Check the client socket
    if (clientSocket.isClosed()) {
        throw new IOException();
    }
    byte[] buffer = new byte[8192]; // input buffer
    // Creation of a UDP datagram using the input buffer.
    datagram = new DatagramPacket(buffer, buffer.length);
    // The client is waiting to receive the answer from the serve. Il will
    // wait for 10 seconds.
    clientSocket.receive(datagram);
    // Check the IP address and port number of the received datagram.
    if (datagram.getAddress().equals(serverAddress) &&
        datagram.getPort() == serverPort) {
        // Bytes received are saved in a String object, starting from the 0
        // position and ending at the actual length of the data. The bytes
        // are converted using the charset UTF-8.
        answerFromServer = new String(datagram.getData(), 0,
            datagram.getLength(), "ISO-8859-1");
    } else {
        throw new SocketTimeoutException();
    }
}
```

```
        }
        return answerFromServer;
    }

/***
 * The client that started the request.
 * @param args command line arguments
 */
public static void main(String args[]) {
    String serverIPAddress = "localhost";
    int serverUDPPort = 7777;
    String request, answer;
    ToUpperUDPClient client;
    BufferedReader tastiera;

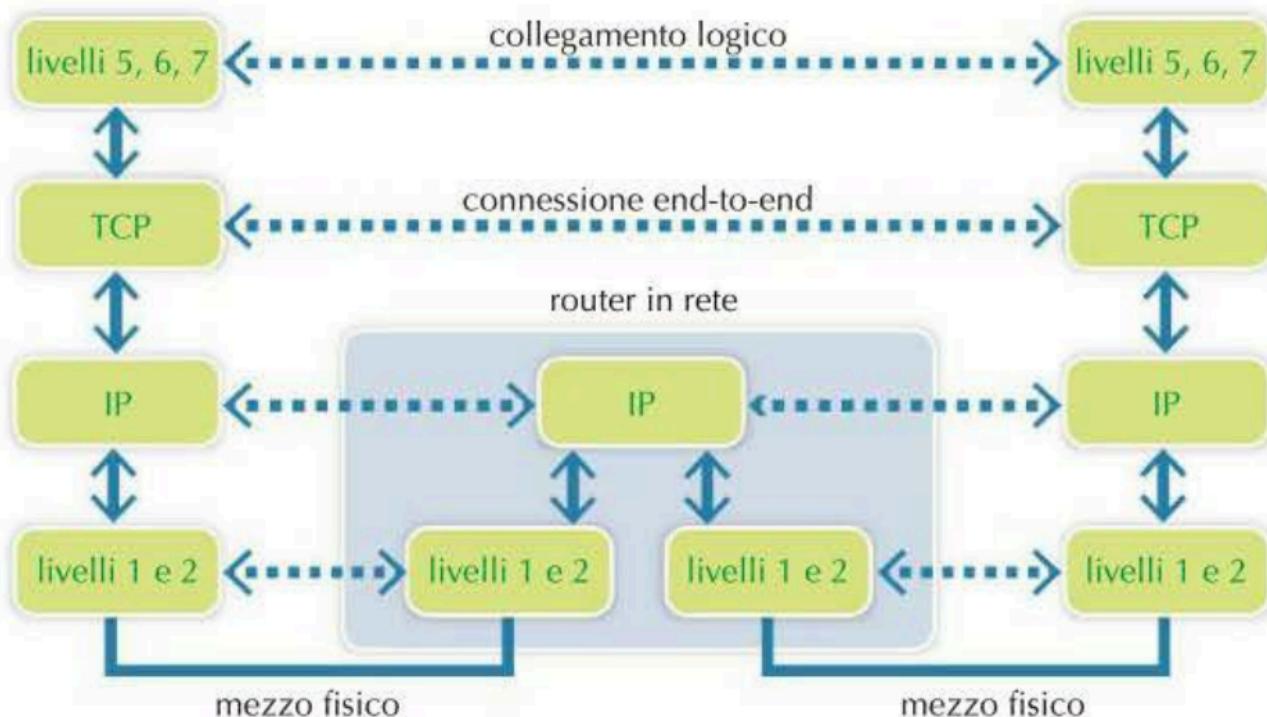
    try {
        tastiera = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("CLIENT: insert a string to convert in upper case: ");
        request = tastiera.readLine();
        client = new ToUpperUDPClient();
        client.sendToServer(request, serverIPAddress, serverUDPPort);
        answer = client.receiveFromServer(serverIPAddress, serverUDPPort);
        System.out.println("The answer received: " + answer);
        client.closeToUpperUDPCClient();
    } catch (SocketException exception) {
        System.err.println("Socket error creation");
    } catch (UnknownHostException exception) {
        System.err.println("IP address not correct");
    } catch (SocketTimeoutException exception) {
        System.err.println("Server not responding");
    } catch (IOException exception) {
        System.err.println("Communication error");
    }
}
}
```

# Progettazione di protocolli di comunicazione

# Protocolli di rete

Un **protocollo** è l'insieme di regole utilizzate da due entità di pari livello per scambiarsi informazioni, specificando cosa deve essere comunicato, in che modo e quando. In pratica, due entità remote potranno colloquiare fra di loro tramite i diversi protocolli usati nei vari livelli della pila protocollare.

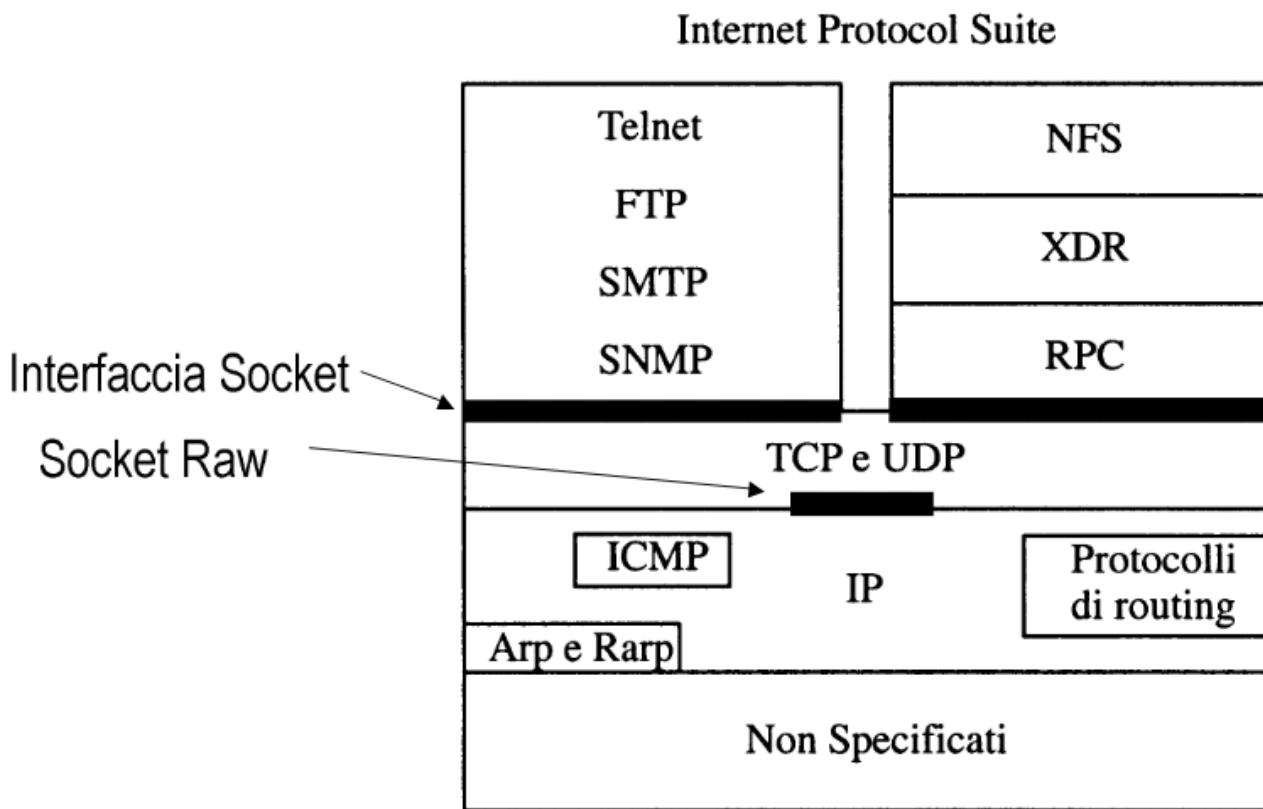
Degli esempi di protocolli sono quelli utilizzati nell'architettura di rete a strati TCP/IP, che a sua volta fornisce i servizi associati ad ogni protocollo del livello applicativo, come ad es. SMTP, FTP, HTTP, ecc.



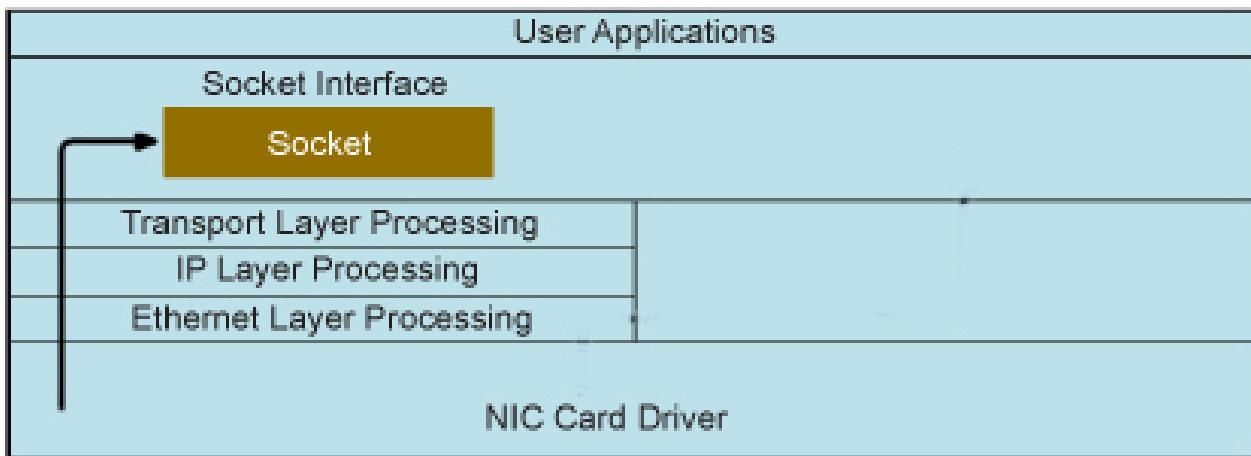
I protocolli del livello applicativo supportano diversi programmi applicativi offrendo determinati servizi, che variano a seconda dell'architettura scelta. Ad esempio, se il servizio è basato su un'architettura di tipo client-server, il protocollo dovrà sempre prevedere la gestione di queste due diverse entità e delle relative comunicazioni. Ciò vuol dire che i processi dovranno essere almeno due, il client e il server.

## Interfacce e servizi

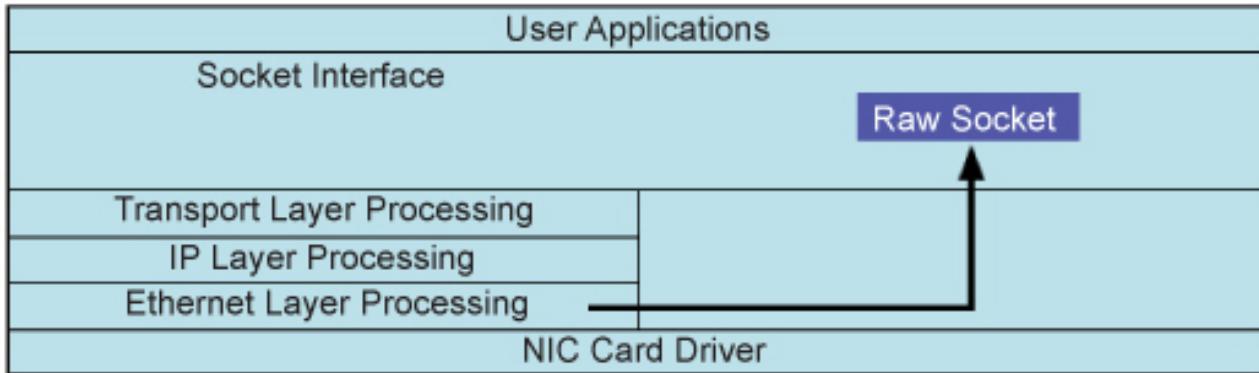
Affinché due protocolli remoti di pari livello possano comunicare devono necessariamente usare i servizi offerti dal livello inferiore. Nella pila protocollare **TCP/IP** la comunicazione tra il livello applicativo e il livello di trasporto avviene attraverso una interfaccia che usa i **socket** e le **API** offerte dal sistema operativo per la programmazione. Il socket permette di identificare univocamente il processo all'interno di un'entità remota.



I socket usati normalmente, come lo **stream socket** (TCP) o il **datagram socket** (UDP), ricevono i dati dal livello di trasporto privati degli header dei livelli sottostanti, quindi ricevono solo il **payload** costituito dai dati finali. Questo vuol dire che non sarà fornita direttamente alcuna informazione relativamente all'indirizzo MAC e IP del sorgente; se la comunicazione avviene tra macchine diverse, viene effettuato solo uno scambio di dati.

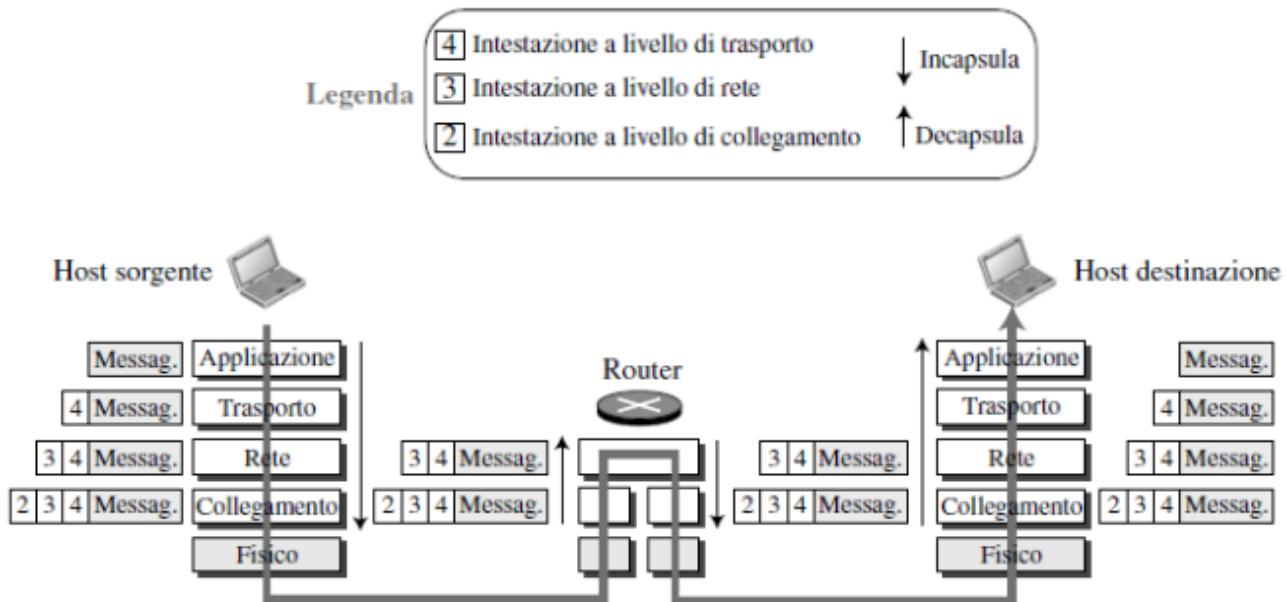


Lo scopo dei **raw socket** è completamente diverso. Un **raw socket** permette all'applicazione ricevente di accedere direttamente ai protocolli di basso livello, permettendo quindi di ricevere un pacchetto non decapsulato, con tutti gli header dei diversi livelli protocollari; infatti per un **raw socket** non serve fornire l'indirizzo MAC e IP.



## Implementazione del servizio

I dati inviati dal livello applicativo saranno **incapsulati** dai diversi livelli protocollari per **aggiungere** negli **header** tutti i dati necessari per permettere la comunicazione tra i protocolli di pari livello. Una volta giunto a destinazione avverrà la fase di **decapsulamento** per passare il **payload** al livello superiore.



A seconda dei protocolli e della capacità della rete da utilizzare, i pacchetti possono essere **frammentati** in modo da generare dei pacchetti di dimensioni adeguatamente piccole per essere inviati nella rete, per poi essere **riassemblati** dal protocollo di livello paritario a destinazione.

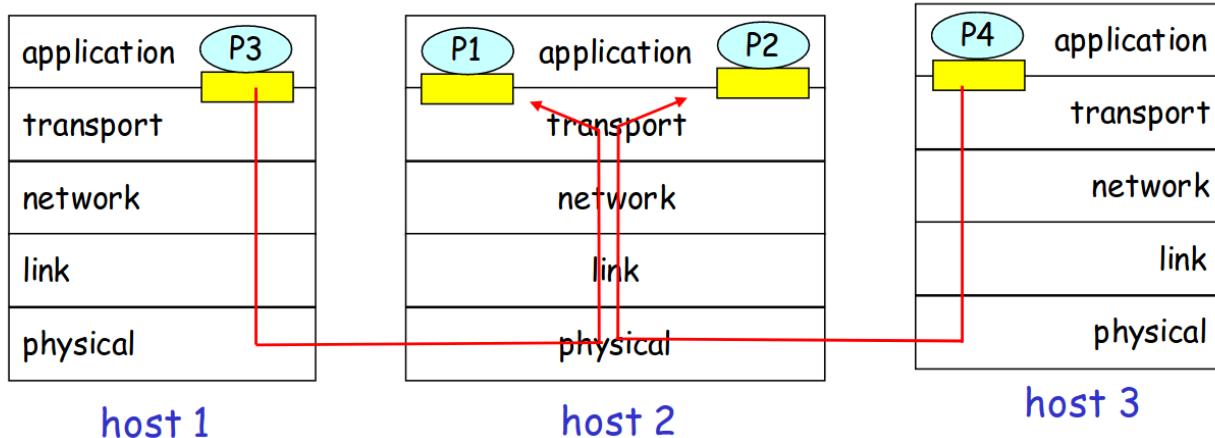
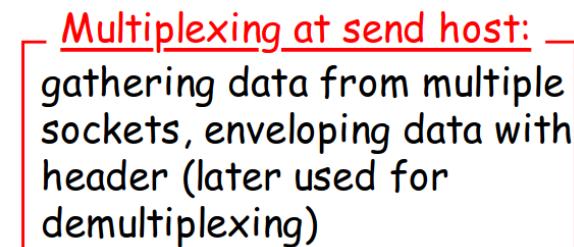
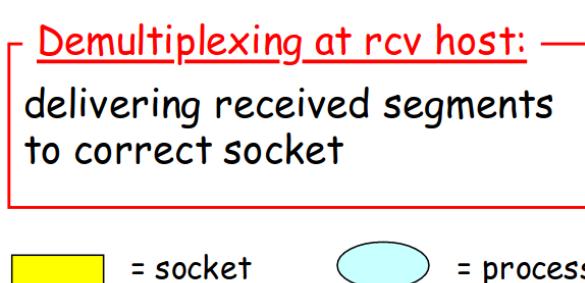
Il pacchetto derivante dal processo di **incapsulamento**, conterrà:

- una intestazione o **header** contenente dati di controllo aggiunti dal protocollo;
- il corpo o **payload** costituito dagli eventuali frammenti;
- una eventuale coda o **tail** (anche **trailer** o **footer**) contenente anch'essa dati di controllo aggiunti dal protocollo.

Lo **header** e il **tail** sono usati dal protocollo per controllare la comunicazione, ad esempio, lo header contiene informazioni utilizzate dall'entità del protocollo come:

- indirizzo del sistema mittente e destinatario;
- numero di sequenza per poter ordinare i pacchetti;
- codice per la individuazione di errori di trasmissione;
- codice identificativo di servizi particolari (es. priorità).

L'**indirizzamento** è una delle informazioni più importanti **presente** praticamente **in qualsiasi layer** della pila protocollare TCP/IP, perché **permette** di effettuare le operazioni di **multiplexing** e **demultiplexing**. I campi **Ethertype**, l'**IP address** e il **Port Number** sono i diversi indirizzi usati rispettivamente dal protocollo *Ethernet*, *IP* e *TCP/UDP*. L'**indirizzamento** può però anche essere **implementato a livello applicativo**, ad esempio specificando nel pacchetto del livello applicativo il nome di un file da manipolare, che identifica la risorsa finale della comunicazione.



I servizi offerti dai vari livelli della pila protocollare possono inoltre essere di due tipi:

- **Orientati alla connessione:** il servizio permette di creare una connessione fra gli estremi della comunicazione, e quindi il protocollo deve prevedere delle istruzioni che permettano di instaurare tale connessione e di rimuoverla una volta conclusa la comunicazione. Di solito i pacchetti sono consegnati in ordine agli strati protocollari di più alto livello, e se persi possono essere ri-trasmessi (**affidabilità**). Solitamente è previsto anche un riscontro della ricezione in modo da prevedere un metodo di ri-trasmissione dei dati nel caso venissero persi (**conferma, acknowledgment**).
- **Senza connessione:** il servizio non prevede l'instaurazione di una connessione, i dati non vengono consegnati in ordine agli strati protocollari di più alto livello, e **può o meno esserci il riscontro della ricezione**. Di conseguenza, nel caso non ci sia il riscontro della ricezione, i dati persi non saranno gestiti in alcun modo.

Il concetto di **conferma della ricezione** è comunque una caratteristica indipendente dalla presenza o meno di una connessione; spesso vengono abbinate, ma la conferma della ricezione può rimanere una caratteristica a sé stante che può essere presente anche in un protocollo non connesso, come ad esempio avviene in una delle possibili implementazioni del protocollo LLC Layer (*Logical Link Control Layer - Standard IEEE 802.2*).

Un altro servizio offerto da diversi protocolli della pila protocollare TCP/IP è il **controllo del flusso**, con il quale è possibile stabilire la quantità di pacchetti che le entità possono scambiarsi senza sovraccaricare le risorse che localmente hanno a disposizione. Una delle **tecniche di controllo del flusso** più semplici è l'implementazione della tecnica [stop-and-wait](#), mentre i protocolli [sliding window](#) usati dal protocollo Data Link Layer e dal protocollo TCP, seppur complessi, risultano molto più efficienti soprattutto in presenza di molto rumore che può disturbare il segnale.

Spesso i protocolli sono anche progettati per effettuare un **controllo della presenza di errori**, come nel caso dell'uso del FCS ([Frame Check Sequence](#)) del frame Ethernet, o verificare che non siano stati persi pacchetti come nel caso del protocollo di conferma, numerazione dei segmenti e timer del protocollo TCP, o che non siano errati i comandi protocollari, come nel caso del [Checksum](#) usato dallo header TCP per rilevare l'eventuale presenza di errori nello header del segmento. Non necessariamente queste caratteristiche sono implementate dai protocolli della pila protocollare, in quanto **il controllo di errori introduce un overhead** che potrebbe essere ritenuto non accettabile; la loro implementazione dipende dal servizio che il protocollo deve offrire.

Decisamente più complessa, ma in costante aumento è infine l'implementazione dei **servizi di trasmissione**, come l'assegnazione di una **priorità ai pacchetti**, la **garanzia di un certo livello di qualità del servizio** e la **gestione della sicurezza**. Questi ultimi aspetti stanno di fatto assumendo un'importanza sempre maggiore, e poco per volta le nuove revisioni dei protocolli che compongono la suite TCP/IP, così come i moderni protocolli del livello applicativo, si stanno sempre più orientando ad una loro gestione.

## CLIL Listening and Writing - Protocols and Layers

Watch the video [Protocols and Layers](#) (on [YouTube](#)) and answer the following questions. Send your answers to your teacher by email.

1. Explain how two peer protocol instances can talk in network communication.
2. Specify which mechanism is used to effect protocol layering, and describe briefly how it works. (*Hint: it is like sending a letter in an envelope*).
3. Describe which main purpose of demultiplexing is.
4. Explain how different layers are able to implement demultiplexing.
5. Describe which advantages and disadvantages of layering are.

## Progettazione di un protocollo

Per **progettare di un nuovo protocollo applicativo** si deve prima di tutto stabilire il **tipo di servizio che è necessario usare al livello di trasporto**, decidendo se il nuovo

protocollo di comunicazione a livello applicativo dovrà appoggiarsi ad un servizio connection-oriented o connectionless; nel primo caso si userà TCP, nel secondo UDP.

Questa scelta dipende da molti fattori in quanto optare per un servizio orientato alla connessione permette di demandare al livello di trasporto tutto ciò che è relativo all'instaurazione di una connessione affidabile e, se non viene richiesta una ulteriore gestione di connessione, la progettazione del protocollo risulterà più semplice.

Chiaramente un servizio a livello trasporto di tipo connection-oriented necessiterà di un maggiore overhead proprio per l'operazione di instaurazione e gestione della connessione stessa, ma se il protocollo che si sta progettando non necessita di un servizio di trasporto complesso come quello offerto da TCP si potrà optare per un servizio non orientato alla connessione e non affidabile, come quello offerto da UDP. In questo caso sarà compito del nuovo protocollo a livello applicativo la gestione di una eventuale connessione, così come di un sistema di conferma di ricezione se fosse necessario anche evitare la perdita di pacchetti, implicando una progettazione molto più complessa.

Si osservi che **la scelta del servizio a livello di trasporto non determina le caratteristiche del nuovo protocollo di comunicazione a livello applicativo**. Ad esempio, se si sceglie a livello di trasporto il protocollo TCP, **non consegue** che il nuovo protocollo sia necessariamente connection-oriented; il nuovo protocollo potrà essere orientato alla connessione o meno a seconda degli aspetti che saranno implementati. Usare un certo servizio a livello di trasporto determina solo i servizi che il nuovo protocollo userà.

Quindi, a seconda dell'obiettivo del nuovo protocollo di comunicazione, la progettazione del protocollo dovrà prevedere la definizione di tutti o parte dei seguenti aspetti, analizzati in dettaglio nel seguito della trattazione:

1. indirizzamento;
2. frammentazione e riassemblaggio;
3. encapsulamento;
4. controllo della connessione;
5. servizio confermato o non confermato;
6. controllo degli errori;
7. controllo del flusso;
8. multiplexing e demultiplexing;
9. servizi di trasmissione.

## Indirizzamento

L'**indirizzamento** permette di **identificare univocamente una entità nella rete**. Esistono diversi livelli di indirizzamento e ogni livello ha una propria visibilità:

- l'indirizzo fisico, come ad esempio l'indirizzo MAC, ha una visibilità locale ad una rete;
- l'indirizzo di rete, come ad esempio l'indirizzo IP, ha una visibilità globale su più reti;
- l'indirizzo del processo, come ad esempio il numero di porta in TCP/IP o il Service Access Point in OSI, ha una visibilità locale al sistema.

Se il protocollo è di tipo client-server il nuovo protocollo di comunicazione dovrà indicare il numero di porta del server, mentre la scelta per il client sarà lasciata al Sistema Operativo.

Questo aspetto riguarda l'indirizzamento a livello di trasporto, a livello applicativo il protocollo che si andrà a progettare potrebbe richiedere una sua propria forma di indirizzamento per identificare la risorsa finale oggetto della comunicazione, come ad esempio il nome di un file.

## Frammentazione e riassemblaggio

Se il protocollo progettato per il livello applicativo lo prevede, i **dati** dovranno essere **frammentati in blocchi più piccoli**, se necessario di dimensione fissata, e ogni blocco sarà successivamente passato al livello di trasporto. Se nel nuovo protocollo viene prevista questa fase, vista la tecnica di commutazione usata in Internet, di tipo packet switching, sarà probabilmente opportuno determinare una modalità per **numerare i frammenti**, per consentire di **riassemblerli** nel giusto ordine una volta raggiunta la destinazione (il protocollo applicativo paritario).

La **frammentazione** e il **riassemblaggio** sono utili in quanto permettono di:

- effettuare il controllo degli errori su ogni blocco;
- utilizzare in modo equo il mezzo trasmissivo;
- ritrasmettere un solo blocco in caso di errori, invece dell'intero messaggio originale;
- subire minori ritardi;
- avere un buffering di dimensione ridotte,

ma al contempo prevedono anche alcuni svantaggi, tra cui:

- una maggiore elaborazione in quanto i frammenti devono essere riassemblati in ordine;
- una minore quantità di dati trasmessi nell'unità di tempo vista la maggiore elaborazione richiesta (minor throughput).

## Incapsulamento

L'**incapsulamento** definisce quali **informazioni di controllo** devono essere aggiunte ai dati e come vengono organizzate nel pacchetto, affinché due entità remote di pari livello possano gestire correttamente il payload.

L'incapsulamento prevede ad esempio l'aggiunta di:

- indirizzi;
- codici per il controllo degli errori;
- comandi di controllo per la gestione del dato;
- ecc.,

organizzandoli in un header ed eventualmente un tail.

## Controllo della connessione

La progettazione di un nuovo protocollo comunicativo a livello applicativo servirà al perseguitamento di uno specifico scopo e, in base a questo si dovrà decidere se il nuovo protocollo dovrà essere orientato alla connessione o non orientato alla connessione, indipendentemente dal servizio scelto al livello di trasporto.

Se il nuovo protocollo è **orientato alla connessione** (*connection-oriented*) deve prevedere le seguenti fasi:

1. creazione della connessione;
2. trasmissione dei dati;
3. chiusura della connessione.

Se invece il protocollo **non è orientato alla connessione** (*connectionless*) queste fasi non sono previste.

## Fasi di una connessione

Nella fase di **creazione della connessione** le entità si accordano su come scambiare i dati:

- a. nel **caso più semplice** il mittente fa una richiesta di connessione e il ricevente la accetta o la rifiuta;
- b. nel **caso più complesso** le due entità dovranno prevedere l'instaurazione di una connessione, oltre alla eventuale fase di negoziazione per garantire un determinato livello del servizio di trasmissione (*priorità, qualità del servizio, sicurezza*).

Qualunque sia la modalità scelta, la creazione di una connessione prevede il mantenimento dei dati della sessione di comunicazione e quindi le due parti dovranno riservare risorse per mantenere memorizzato l'identità dell'entità paritaria della comunicazione, identificata in qualche modo, oltre allo spazio di memoria dedicato al mantenimento dei dati della comunicazione.

Nella fase di **trasferimento dei dati** si dovranno prevedere tutti o solo parte dei seguenti controlli:

- a. **corretta sequenza dei blocchi di dati** prevedendo, ad esempio, nello header un codice che permetta di ricostruire il dato originale, nel caso sia stato frammentato, indipendentemente dall'ordine di arrivo dei pacchetti;
- b. **controllo degli errori**, ad esempio utilizzando un CRC ([\*Cyclic Redundancy Check\*](#));
- c. **controllo del flusso**, ad esempio prevedendo un limite massimo di pacchetti che possono essere trasmessi per motivi di bufferizzazione.

Nella fase di **chiusura della connessione** le due entità concordano in qualche modo di terminare la comunicazione e rilasciano tutte le risorse allocate durante la fase di apertura della connessione.

## Servizio affidabile

A seconda dello scopo per cui viene creato il nuovo protocollo, si può prevedere che esso sia **affidabile** progettando una tecnica di **conferma** dei pacchetto ricevuti, oppure può essere previsto che **non sia affidabile** evitando tale progettazione.

Le modalità di conferma possono essere molto semplici come la conferma di ogni singolo pacchetto appena questo viene ricevuto, oppure prevedere tecniche più raffinate come la conferma cumulativa simile a quella adottata dal protocollo TCP.

## Controllo degli errori

Se il protocollo che si sta progettando richiede la rilevazione degli errori, sarà necessario sviluppare o adottare una tecnica di **controllo degli errori** che permetta di individuare eventuali alterazioni dei dati e/o delle informazioni di controllo.

Le modalità adottabili possono essere le più varie, ad esempio si possono prevedere due fasi dove il destinatario individua gli errori e quindi richiede esplicitamente la ritrasmissione del dato, oppure il destinatario rileva l'errore scartando il pacchetto e demandando al mittente la ritrasmissione del dato che eventualmente non risulta confermato, come avviene in TCP, avvalendosi dell'ausilio di timer appositi.

Inoltre esistono diversi algoritmi che permettono di generare codici in grado solo di individuare l'errore, come i codici di parità, la check-sum o il CRC ([Cyclic Redundancy Check](#)), o addirittura di correggerli, come i codici di parità trasversale e longitudinale o il codice di Hamming.

## Controllo del flusso

Il **controllo del flusso** è utilizzato dal ricevente per limitare o aumentare la velocità con cui la sorgente invia i dati; la sorgente non deve inviare più dati di quanti il ricevente ne possa processare, ma allo stesso tempo, se il ricevente fosse in grado di reggere la velocità di invio del mittente, sarebbe opportuno aumentare la velocità di spedizione.

La progettazione di un nuovo protocollo di comunicazione deve prevedere una modalità di controllo del flusso, in quanto da questa dipenderà anche la gestione delle risorse necessarie per la bufferizzazione dei dati ricevuti. Se questa gestione non viene progettata accuratamente si rischierà di perdere dei dati.

La modalità più semplice di gestione del flusso prevede che ad ogni invio di un dato il mittente debba attendere la conferma di ricezione da parte del destinatario. Chiaramente questa gestione è inefficiente, ma raggiunge perfettamente lo scopo in modo molto semplice.

Tecniche più avanzate possono prevedere la bufferizzazione di più dati e la conferma cumulativa dei segmenti, come avviene per TCP, ma ovviamente la sua implementazione sarà decisamente più complessa rispetto alla precedente.

## Multiplexing e demultiplexing

Nelle reti di computer il **multiplexing** è una tecnica che permette a più comunicazioni di un determinato livello protocollare di confluire in una sola comunicazione a livello inferiore. Il **multiplexing** consente quindi di aggregare i dati e di ottimizzare l'efficienza della trasmissione.

Il ricevente dovrà necessariamente implementare un meccanismo di disaggregazione (**demultiplexing**) dei dati per consegnare i dati ricevuti al corretto destinatario.

Nell'ambito delle reti le tecniche di multiplexing e demultiplexing sono ampiamente usate:

- a livello di trasporto i protocolli TCP e UDP usano i numeri di porta per distinguere tra i diversi protocolli del livello applicativo;
- a livello di rete il protocollo IP prevede il campo *Protocollo* nel suo header per capire a quale protocollo di trasporto consegnare il pacchetto;
- a livello data link il frame Ethernet prevede il campo *EtherType* nel suo header per consegnare i dati al giusto protocollo del livello di rete.

Progettando un nuovo protocollo di comunicazione a livello applicativo, lo sviluppo di tecniche di **multiplexing** e di **demultiplexing** implicano la necessità di aggregare i dati provenienti da livelli superiori, e di disaggregarli una volta ricevuti, e quindi il protocollo oggetto di progettazione sarà utilizzato almeno come trasporto da un altro protocollo, come avviene ad esempio nei Web Service SOAP (*Simple Object Access Protocol*).

## Servizi di trasmissione

Nel caso il nuovo protocollo progettato richiedesse determinate caratteristiche di **servizi di trasmissione**, tali **servizi** dovranno essere **negoziati**, e quindi generalmente il nuovo protocollo sarà connection-oriented in quanto la negoziazione avviene di solito nella fase di creazione della connessione.

I **servizi di trasmissione** possono riguardare:

- la **priorità** dei messaggi, ad esempio rendendo alcuni messaggi di controllo prioritari sui dati, o ammettendo la possibilità di utenti con diversi livelli di priorità in base al tipo di contratto sottoscritto;
- la **qualità del servizio** che può riguardare il minimo ritardo o il minimo throughput accettabile;
- la **sicurezza** imponendo delle restrizioni di accesso al messaggio o all'intera comunicazione.

Ad esempio nello header IPv4 è presente il campo *Type of Service* che potrebbe permettere di impostare dei servizi di trasmissione a livello di rete, così come nello header IPv6 possono essere usati i due campi *Traffic Class* e *Flow Label*.

**La gestione dei servizi di trasmissione** è comunque **particolarmente complessa**, anche se di fatto la tendenza è quella di diversificare sempre più le comunicazioni in base a questo aspetto.

## Esempi di protocolli applicativi di rete

I protocolli applicativi di rete consentono la comunicazione tra processi utente su sistemi diversi e forniscono diversi servizi di comunicazione

# Esempi di progettazione di protocolli applicativi

## Monitoraggio Azioni

Il protocollo Monitoraggio Azioni permette di monitorare le quotazioni di un'azione scelta da un elenco.

La parte server del protocollo prevede che nel momento in cui un client si connette al server, questo gli invii la lista di tutte le azioni monitorate, e una volta che il client avrà comunicato al server l'azione da monitorare, questo invierà le quotazioni del titolo ogni *tot tempo*.

Il server ascolterà sulla porta *nnnnn*.

Il protocollo è completamente testuale con un formato dei comandi del seguente tipo:

COMANDO\$<Dato1>[#<Dato2>][#<Dato3>]

Ogni COMANDO dovrà essere inviato in capital letters. Il comando è terminato dal carattere \$ (section), e i dati inviati possono essere costituiti da un solo elemento o da una lista di elementi separati dal carattere # (hash).

Il protocollo è connectionless, non affidabile in quanto non prevede un meccanismo di conferma di ricezione dei pacchetti e del conseguente recupero nel caso non fossero stati ricevuti, e non prevede un controllo del flusso; il server invia le quotazioni ogni tot tempo, non prevedendo comandi che modifichino la velocità di spedizione.

I dati scambiati non sono frammentati, sebbene il comando TITLE LIST possa prevedere una lista di titoli molto estesa.

### Comandi del server

I comandi descritti di seguito sono quelli inviati dal server al client (server → client).

#### TITLE LIST

TITLE LIST\$<Dato1>[#<Dato2>[#<Dato3>[ . . . ]]

Il comando invia la lista dei titoli azionari.

Esempio: TITLE LIST\$Adidas#First Capital#Unicredit

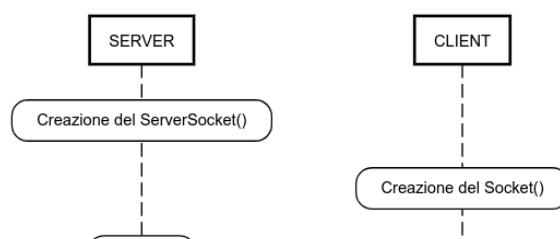
#### VALUE

VALUE\$<n>

Il comando invia la quotazione attuale del titolo azionario scelto.

Esempio: VALUE\$134

### Comandi del client



I comandi descritti di seguito sono quelli inviati dal client al server (client → server)

## TITLE

TITLE§<Titolo>

Il comando indica il titolo di cui si vogliono monitorare le quotazioni.

Esempio: TITLE§First Capital

## Sequenza temporale<sup>7</sup>

Il codice usato su [SequenceDiagram.org](#) per ottenere la sequenza temporale illustrata sopra è il seguente:

```
rbox over SERVER:Creazione del ServerSocket()
rbox over CLIENT:Creazione del Socket()
rbox over SERVER:accept()
activate SERVER
CLIENT->(1)SERVER: connect()
deactivate SERVER
SERVER->(1)CLIENT:TITLE LIST§Titolo 1#Titolo 2#Titolo 3
CLIENT->(1)SERVER:TITLE§Titolo 3
SERVER->(1)CLIENT:VALUE§134
SERVER->(1)CLIENT:VALUE§151
CLIENT->(1)SERVER:TITLE§Titolo 1
SERVER->(1)CLIENT:VALUE§25
SERVER->(1)CLIENT:VALUE§36
SERVER->(1)CLIENT:VALUE§48
box over SERVER,CLIENT: Continua in questo modo fino alla\nchiusura del
SERVER e del CLIENT
```

## TFTP

TFTP (*Trivial File Transfer Protocol*) è un semplice protocollo di trasferimento dei file descritto nello [RFC 1350](#), ed è stato implementato al di sopra di UDP. Il protocollo è stato progettato per essere piccolo e semplice da implementare, di fatti non presenta tutte le caratteristiche di FTP, può solo leggere e scrivere file da e verso un server remoto; non ha alcuna forma di autenticazione e non permette di ottenere la lista di file e directory, come invece fa FTP. I dati scambiati sono quantificati in byte da 8 bit e prevede tre diversi formati di trasferimento dei file, di fatto implementando un livello di presentazione.

Il protocollo applicativo implementa una serie di caratteristiche che sono deducibili anche solo osservando il formato dei suoi messaggi.

TFTP è un **protocollo connesso**, infatti la connessione viene richiesta dal client inviando il seguente messaggio nel quale i primi due byte costituiscono lo header e indicano se il cliente vuole effettuare un'operazione di lettura (01) o di scrittura (02). Il campo successivo, di lunghezza variabile (in byte), indica il nome del file su cui vuole agire il

---

<sup>7</sup> Il sequence diagram è stato creato usando [SequenceDiagramOrg](#)

client; per indicare la fine del nome viene usato un byte posto a 0 (stringa zero terminata). Il nome del file è di fatto una forma di **indirizzamento** a livello applicativo in quanto individua la risorsa su cui si vuole agire. Dopo il nome del file viene indicata la modalità di trasferimento e, anche in questo caso la lunghezza del campo è variabile e quindi la terminazione della stringa viene effettuata tramite un byte posto a zero.

2 bytes	string	1 byte	string	1 byte							
-----											
RRQ/		01/02		Filename		0		Mode		0	
WRQ	-----										

Il server se accetta la richiesta di connessione si comporterà nel modo seguente:

- se il client ha richiesto una connessione per leggere un file (RRQ) il server invierà il primo blocco di dati con un messaggio che ha il seguente formato:

2 bytes	2 bytes	n bytes					
-----							
DATA		03		Block #		Data	
-----							

L'operazione di invio dei dati è identificata nello header dal codice 03, salvato in due byte, i blocchi dei dati sono numerati nello header con numeri progressivi partendo dal valore 01 ed utilizzando due byte, mentre i dati che costituiscono il payload, occupando esattamente 512 byte;

- se invece il client ha richiesto una connessione per scrivere un file (WRQ) il server invierà un messaggio di conferma che ha il seguente formato:

2 bytes	2 bytes				
-----					
ACK		04		Block #	
-----					

Ogni **invio dei dati è confermato** e la conferma è individuata dal codice 04 nello header del messaggio (due byte), seguito dal numero di blocco che viene confermato; nel caso di conferma per l'apertura di connessione in scrittura il numero del blocco assume il valore 00.

La **chiusura della connessione** avviene nel momento in cui viene spedito un messaggio dati contenente un payload di lunghezza inferiore a 512 byte.

Qualunque sia l'operazione richiesta dal client, il trasferimento dei dati, da e verso il server, avviene sempre usando il messaggio dati e la conferma indicati in precedenza. È importante però sottolineare che il protocollo prevede la trasmissione di messaggi con un payload non superiori a 512 byte, che implica di fatto la **frammentazione e il riassemblaggio** del file.

Inoltre prima di poter effettuare la trasmissione di un nuovo messaggio deve essere stata ricevuta la conferma di ricezione del messaggio precedente, quindi il protocollo **esegue un controllo di flusso** di tipo stop-and-wait.

Il protocollo effettua anche un **controllo errori** inviando un messaggio con il seguente formato.

	2 bytes	2 bytes	string	1 byte
ERROR	05   ErrorCode   ErrMsg   0			

Un messaggio di errore non è mai confermato, quindi di fatto è una pura cortesia in quanto potrebbe andare perso durante la trasmissione. Nello header viene posto il codice 05 nei primi due byte, seguono due byte che riportano uno dei codici di errore elencati di seguito, e infine una stringa di messaggio di errore di lunghezza variabile e terminata da un byte posto a 0; il messaggio di errore è destinato ad un utente umano, quindi dovrà essere comprensibile.

#### Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

Da quanto visto finora **il protocollo effettua l'incapsulamento** dei messaggi in quanto in ogni tipologia di messaggio **è presente uno header** che indica, tramite un codice, lo scopo del messaggio stesso. Questo codice è ovviamente comprensibile solo alle due entità che implementano il protocollo TFTP, e quindi solo alle due entità a livello applicativo.

Il protocollo non prevede la gestione del **multiplexing** e del **demultiplexing** in quanto non offre un servizio ad un protocollo di livello superiore per il quale effettuare aggregazione e disaggregazione dei dati trasmessi.

Il protocollo non prevede **servizi di trasmissione**, come la **priorità dei pacchetti**, la **garanzia di un certo livello di qualità del servizio** e la **gestione della sicurezza**.

## Protocollo di gestione dei dati del vento

Di seguito viene mostrata una possibile soluzione di un protocollo del livello applicativo per la gestione dei dati di rilevazione del vento fatta da privati e la relativa memorizzazione in Cloud. Viene fornito prima il testo integrale e poi viene mostrata una possibile soluzione per ognuna delle parti richieste.

### Testo completo

Un produttore di sensori offre un dispositivo per interfacciare un sensore in grado di misurare la velocità (km/h) e la direzione (gradi) del vento con un programma installato

localmente sul computer del cliente. Il produttore rende disponibile un driver che permette di acquisire i valori misurati ogni 5 secondi, in modo tale da permettere la memorizzazione in locale dei dati per poi poterli usare in base alle necessità dell'utilizzatore del sensore. Il sensore e il programma installato localmente usano un protocollo di comunicazione per la raccolta dei dati sul vento.

Inoltre la società produttrice del sensore offre anche un servizio di raccolta in cloud dei dati rilevati dai sensori dei vari utilizzatori privati, e anche in questo caso i client privati comunicheranno con il server di raccolta dei dati tramite un protocollo comunicativo.

Tutti i dati raccolti dalla società produttrice vengono poi resi disponibili gratuitamente a scopo pubblicitario, in modo tale che chiunque ne necessiti possa recuperare le informazioni sui venti in un intervallo di tempo indicato nella richiesta del client. Anche in questo caso viene usato un protocollo comunicativo per lo scambio di questi dati.

Dato che i tre protocolli sono implementati dalla stessa società e sono relativi al medesimo progetto, utilizzano una struttura dei comandi simile. I tre protocolli presentano comunque caratteristiche diverse in quanto in una determinata fase potrebbe essere più efficace un protocollo essenziale e veloce, ammettendo anche eventuali perdite di dati, mentre in un'altra fase potrebbe essere opportuno assicurare un servizio che garantisca un completo scambio dei dati anche a discapito della velocità.

Il candidato, fatte le opportune ipotesi aggiuntive che dovranno essere adeguatamente giustificate, progetti i seguenti protocolli comunicativi:

1. il protocollo di comunicazione tra il sensore che misura la velocità e la direzione del vento e il programma installato localmente sul computer del client;
2. il protocollo di comunicazione che permette la raccolta dei dati su un server in cloud rilevati dai vari utilizzatori privati;
3. il protocollo di comunicazione che permette la diffusione dei dati raccolti in cloud a tutti i client che vogliono recuperare i dati sui venti in un determinato intervallo di tempo.

## Monitoraggio Vento

### Richiesta

Un produttore di sensori offre un dispositivo per interfacciare un sensore in grado di misurare la velocità (km/h) e la direzione (gradi) del vento con un programma installato localmente sul computer del cliente. Il produttore rende disponibile un driver che permette di acquisire i valori misurati ogni 5 secondi, in modo tale da permettere la memorizzazione in locale dei dati per poi poterli usare in base alle necessità dell'utilizzatore del sensore. Il sensore e il programma installato localmente usano un protocollo di comunicazione per la raccolta dei dati sul vento.

Il candidato, fatte le opportune ipotesi aggiuntive che dovranno essere adeguatamente giustificate, progetti il protocollo di comunicazione tra il sensore che misura la velocità e la direzione del vento e il programma installato localmente sul computer del client.

## Soluzione

5A Informatica  
Request For Comment: 1

M. G. Maffucci  
I.I.S. A. Avogadro - Torino  
Gennaio 2022

### Protocollo Monitoraggio Vento

#### Stato di questo documento

Questo RFC è una possibile soluzione del 1° protocollo della verifica del 09/12/2022 ed è attualmente in stato di lavorazione.

#### Riepilogo

Il protocollo **Monitoraggio Vento** permette di monitorare le misurazioni della velocità e della direzione del vento rilevate tramite un sensore presso la sede del cliente. Questo documento descrive il protocollo e i pacchetti usati per la comunicazione. Inoltre spiega anche le ragioni delle scelte fatte nella sua progettazione.

#### Ringraziamenti

Si ringrazia tutta la comunità che condivide su Internet il proprio lavoro, la vera conoscenza è possibile solo tramite una comunione di menti.

#### Panoramica del protocollo

Il protocollo prevede la comunicazione tra un sensore (lato client) e l'applicazione che colleziona le rilevazioni (lato server). Il sensore rileva i dati del vento ogni 5 secondi, per un totale di 17280 rilevazioni continuative al giorno. Vista la considerevole quantità di dati raccolta giornalmente, il protocollo è progettato in modo tale da non appesantire la comunicazione tra il client e il server, preferendo l'eventuale perdita di alcuni dati rispetto alla certezza del loro ricevimento. Per questa ragione si decide di **usare a livello trasporto il protocollo UDP** che non appesantisce la comunicazione con ulteriori controlli, demandando al protocollo **Monitoraggio Vento** qualsiasi caratteristica comunicativa aggiuntiva.

Il protocollo **prevede l'indirizzamento** sia per il server, sia per il client. Gli identificatori sono forniti dall'azienda produttrice prevedendo 2 byte per l'identificatore del server e 2 byte per l'identificatore del sensore client. Al momento del primo avvio il server e il client si associeranno creando un unico codice a 4 byte, formato unendo il codice del server e il codice del client:

2 byte	2 byte
----- -----	
<id-server>	<id-client>
----- -----	

## Figura 1

Questa operazione viene effettuata solo al primo avvio del sensore, successivamente il server contatterà il sensore usando il codice identificativo completo aggiungendolo ai propri comandi, e il sensore farà altrettanto quando invia i propri dati. In questo modo è possibile pensare ad una espansione futura del protocollo per la gestione contemporanea di più sensori, oltre ad evitare problemi di interferenze da parte di altri sensori nelle vicinanze. Il sensore quando sarà svegliato dal server risponderà con un messaggio di conferma e subito dopo inizierà a rilevare ed inviare i dati, aggiungendo ad ogni trasmissione il proprio identificatore.

Il protocollo **non prevede la frammentazione e il riassemblaggio** in quanto i dati trasmessi sono costituiti da pochi byte in formato testuale, prevedendo nello header un comando e l'identificatore, mentre l'eventuale payload sarà costituito anch'esso da pochi dati di rilevazione del vento.

Il protocollo **prevede l'incapsulamento** in quanto tutti i messaggi scambiati tra il client ed il server prevedono un header contenente un comando e l'identificatore.

Il protocollo **è connesso** in quanto al primo avvio avviene una fase iniziale di associazione tra il server e il sensore client creando l'identificatore univoco, associazione che può essere terminata chiudendo la connessione ed eliminando il sensore dall'applicazione. Le fasi di terminazione e ri-avvio della rilevazione dei dati del vento, prevedono comunque uno scambio di messaggi di associazione tra il client e il server, ma non prevedono la chiusura della connessione iniziale.

Il protocollo **non è confermato durante la spedizione delle rilevazioni** per non creare traffico e perché per il sensore client non è necessario ricevere la conferma di ricezione dei dati inviati al server. È possibile che alcuni pacchetti vengano persi per interferenze momentanee, ma vista la rilevante quantità di dati inviati è accettabile questa perdita.

Il protocollo **prevede una conferma per ogni comando di controllo** per la gestione del sensore. Quest'ultimo invia un messaggio di conferma per ogni comando relativo all'associazione o alla chiusura dell'associazione, definitiva o temporanea, con il server.

La **rilevazione degli errori è prevista** su diversi livelli. Il funzionamento regolare del sensore viene garantito dalle spedizioni dei dati ogni 5 secondi. Nel caso il sensore non effettuasse più le spedizioni si assumerà un malfunzionamento che dovrà essere gestito manualmente dall'utente. L'eventuale mancata associazione tra sensore e server, sia al primo avvio, sia ai successivi, sarà da considerare come un malfunzionamento da gestire manualmente dall'utente.

Il **controllo del flusso è previsto** intrinsecamente nel protocollo in quanto il sensore client invierà i dati rilevati ogni 5 secondi.

**Non è previsto il multiplexing** in quanto non è prevista aggregazione di dati.

Il protocollo **non prevede priorità dei pacchetti, non offre qualità del servizio e non gestisce la sicurezza**. Per quest'ultimo aspetto, se necessario, dovranno essere previste misure di sicurezza aggiuntive usando protocolli appositi a corredo di questo.

Il server ascolterà sulla porta 55555.

Il protocollo è completamente testuale con un formato dei comandi del seguente tipo:

2 byte	2-4 byte	0-25 byte
-----		
COMANDO	<id>	[<payload>]
-----		

Figura 2

Il campo COMANDO è costituito da un codice numerico che individua il tipo di operazione.

Il campo <id> identifica l'identificatore univoco del server in fase di prima connessione (2 byte) o l'identificatore univoco del sensore creato dall'unione dell'identificatore del server e del sensore client (4 byte).

Il campo payload può o meno essere presente a seconda del tipo di comando ed è adibito al contenimento dei dati sul vento.

### Comandi del server

I comandi descritti di seguito sono quelli inviati dal server al sensore client (server → client).

2 byte	2 byte
-----	
01	<id-server>
-----	

Figura 3

Il comando serve per effettuare la prima connessione con il sensore client. L'identificatore univoco <id> è l'identificatore del server.

2 byte	4 byte
-----	
03	<id>
-----	

Figura 4

Il comando serve per indicare al sensore client di terminare temporaneamente la rilevazione dei dati del vento. L'identificatore univoco <id> è l'identificatore del sensore.

```
2 byte 4 byte
-----
| 05 | <id> |
-----
```

Figura 5

Il comando serve per indicare al sensore client di riprendere la rilevazione dei dati del vento. L'identificatore univoco <id> è l'identificatore del sensore.

```
2 byte 4 byte
-----
| 07 | <id> |
-----
```

Figura 6

Il comando serve per terminare l'associazione tra il server e il sensore client, in modo tale che quest'ultimo possa essere sostituito o associato ad un altro server. L'identificatore univoco <id> è l'identificatore del sensore.

### Comandi del client

I comandi descritti di seguito sono quelli inviati dal client al server (client → server)

```
2 byte 4 byte
-----
| 02 | <id> |
-----
```

Figura 7

Il comando serve per effettuare la prima connessione con il server. L'identificatore univoco <id> è l'identificatore generato dall'unione dell'identificatore del server seguito dall'identificatore del sensore client. Questo identificatore servirà per individuare univocamente un sensore.

```
2 byte 4 byte
-----
| 04 | <id> |
-----
```

Figura 8

Il comando serve per segnalare che il sensore smette di rilevare i dati del vento avendo ricevuto il comando dal server. L'identificatore univoco <id> è l'identificatore del sensore.

2 byte	4 byte
-----	
06   <id>	
-----	

Figura 9

Il comando serve per segnalare che il sensore riprende la rilevazione dei dati del vento dopo aver ricevuto il relativo comando dal server. L'identificatore univoco <id> è l'identificatore del sensore.

2 byte	4 byte
-----	
08   <id>	
-----	

Figura 10

Il comando serve per chiudere l'associazione tra il sensore client e il server. L'identificatore univoco <id> è l'identificatore del sensore.

2 byte	4 byte	19 byte	3 byte	3 byte
-----				
10   <id>   <timestamp>   <velocità>   <direzione>				
-----				

Figura 11

Il comando è usato per l'invio dei dati del vento ogni 5 secondi.

L'identificatore univoco <id> è l'identificatore generato dall'unione dell'identificatore del server seguito dall'identificatore del sensore client. Questo identificatore servirà per individuare univocamente un sensore.

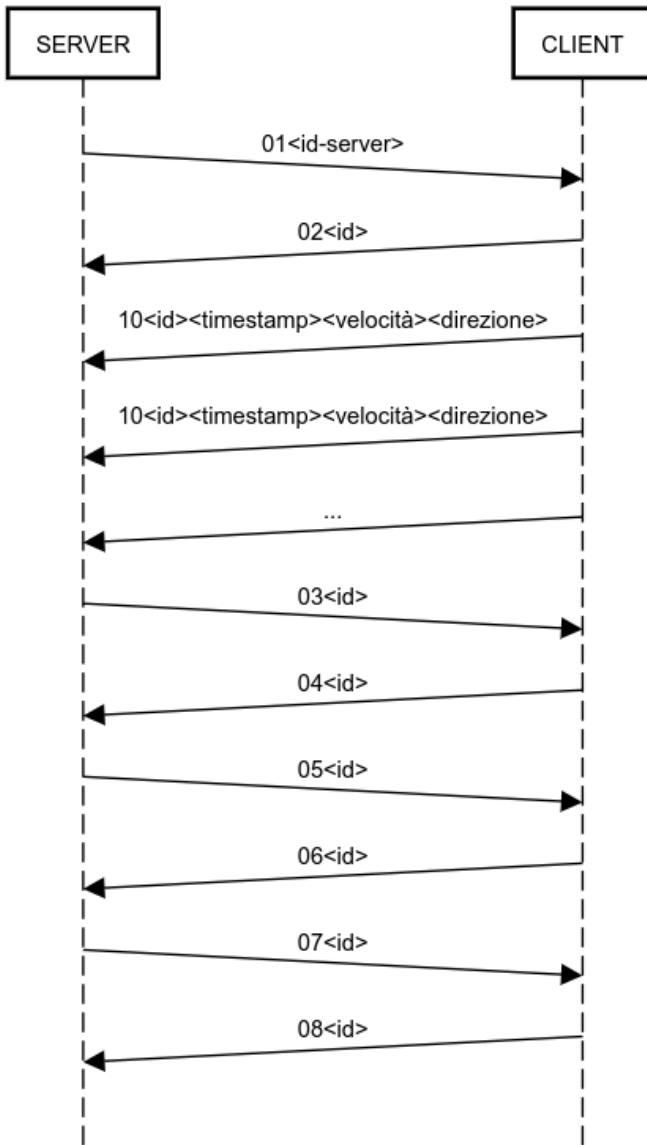
Il formato del timestamp è yyyy-mm-dd hh.mm.ss costituito da cifre decimali in formato testuale.

Il formato della velocità in km/h è nnn, tre cifre decimali in formato testuale.

Il formato della direzione in gradi è nnn, tre cifre decimali in formato testuale.

## Sequenza temporale<sup>8</sup>

### Protocollo Monitoraggio Vento



Il codice usato su [SequenceDiagram.org](#) per ottenere la sequenza temporale illustrata sopra è il seguente:

```
title Protocollo Monitoraggio Vento
```

```
SERVER->(1)CLIENT: 01<id-server>
CLIENT->(1)SERVER: 02<id>
CLIENT->(1)SERVER:10<id><timestamp><velocità><direzione>
CLIENT->(1)SERVER:10<id><timestamp><velocità><direzione>
CLIENT->(1)SERVER:...
SERVER->(1)CLIENT:03<id>
CLIENT->(1)SERVER:04<id>
SERVER->(1)CLIENT:05<id>
```

<sup>8</sup> Il sequence diagram è stato creato usando [SequenceDiagramOrg](#)

```
CLIENT->(1)SERVER:06<id>
SERVER->(1)CLIENT:07<id>
CLIENT->(1)SERVER:08<id>
```

## Vento in Cloud

### Richiesta

Inoltre la società produttrice del sensore offre anche un servizio di raccolta in cloud dei dati rilevati dai sensori dei vari utilizzatori privati, e anche in questo caso i client privati comunicheranno con il server di raccolta dei dati tramite un protocollo comunicativo.

Il candidato, fatte le opportune ipotesi aggiuntive che dovranno essere adeguatamente giustificate, progetti il protocollo di comunicazione che permette la raccolta dei dati su un server in cloud rilevati dai vari utilizzatori privati.

### Soluzione

5A Informatica  
Request For Comment: 2

M. G. Maffucci  
I.I.S. A. Avogadro - Torino  
Gennaio 2022

### Protocollo Vento in Cloud

#### Stato di questo documento

Questo RFC è una possibile soluzione del 2° protocollo della verifica del 09/12/2022 ed è attualmente in stato di lavorazione.

#### Riepilogo

Il protocollo **Vento in Cloud** permette di raccogliere le misurazioni della velocità e della direzione del vento rilevate tramite un sensore presso la sede dei vari cliente. Questo documento descrive il protocollo e i pacchetti usati per la comunicazione. Inoltre spiega anche le ragioni delle scelte fatte nella sua progettazione.

#### Ringraziamenti

Si ringrazia tutta la comunità che condivide su Internet il proprio lavoro, la vera conoscenza è possibile solo tramite una comunione di menti.

#### Panoramica del protocollo

Il protocollo prevede la comunicazione tra un server in cloud che offre il servizio di raccolta dati, e i client che inviano i dati raccolti sulle rilevazioni di velocità e direzione del vento. Dato che i sensori presso le sedi locali dei client rilevano i dati del vento ogni 5 secondi, per un totale di 17280 rilevazioni continuative al giorno, la quantità di dati inviata da un client può essere considerevole, e per essere certi che tutti i dati inviati al server siano ricevuti si decide di **usare a livello trasporto il protocollo TCP** che implementa già tutte le caratteristiche di

affidabilità e mantenimento della sessione necessarie per il trasferimento di file anche di grandi dimensioni. Inoltre il protocollo TCP effettua autonomamente la frammentazione e il riassemblaggio del messaggio nel caso fosse necessario, usando una modalità che risulta adatta per la tipologia di trasferimento richiesto. Le uniche caratteristiche che il protocollo **Vento in Cloud** implementa sono l'identificazione dell'utente tramite un identificatore di 2 byte fornito direttamente dall'azienda produttrice, l'identificazione del tempo delle rilevazioni usando un timestamp di inizio e uno di fine, e l'identificazione del luogo di rilevazione dei dati usando la latitudine, la longitudine e l'altitudine dal livello del mare del punto di rilevazione.

Il protocollo **prevede l'indirizzamento** per il client utilizzando un identificatore di 2 byte dall'azienda produttrice che sarà aggiunto ad ogni messaggio inviato dal client.

Il protocollo **non prevede la frammentazione e il riassemblaggio** in quanto questa funzionalità è demandata al protocollo TCP.

Il protocollo **prevede l'incapsulamento** in quanto tutti i messaggi scambiati tra il client ed il server prevedono un header contenente un comando e l'identificatore del client.

Il protocollo **non è connesso** in quanto la sessione viene mantenuta solo dal protocollo TCP.

Il protocollo **è confermato** prevedendo l'invio di una conferma di ricezione del file una volta conclusa la spedizione da parte del client.

Il protocollo **prevede la rilevazione di un errore**, cioè il non riconoscimento dell'identificatore del client da parte del server. Inoltre indirettamente la mancata ricezione della conferma di ricezione del file da parte del client rappresenta intrinsecamente il rilevamento di un problema di spedizione o di ricezione. Tutti gli altri errori sono demandati completamente al protocollo TCP.

Il protocollo **non prevede controllo del flusso** demandandolo completamente al protocollo TCP.

**Non è previsto il multiplexing** in quanto non è prevista aggregazione di dati.

Il protocollo **non prevede priorità dei pacchetti, non offre qualità del servizio e non gestisce la sicurezza**. Per quest'ultimo aspetto, se necessario, dovranno essere previste misure di sicurezza aggiuntive usando protocolli appositi a corredo di questo.

Il server ascolterà sulla porta 44444.

Il protocollo è completamente testuale con un formato dei comandi del seguente tipo:

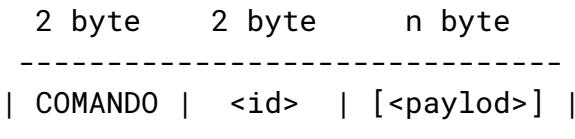


Figura 1

Il campo COMANDO è costituito da un codice numerico che individua il tipo di operazione.

Il campo <id> identifica l'identificatore univoco del client usato sia in fase di spedizione del messaggio, sia in fase di conferma da parte del server.

Il campo payload può o meno essere presente a seconda del tipo di comando ed è adibito al contenimento dei dati relativi al periodo di rilevazione, all'esatta localizzazione del punto di rilevazione e ai dati del vento.

### Comandi del client

Il comando descritto di seguito è inviato dal client al server (client → server).

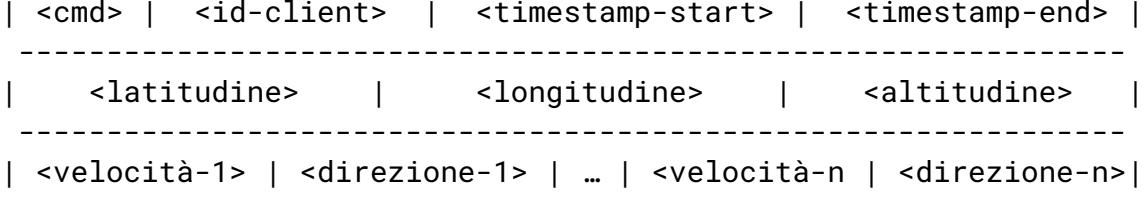


Figura 2

Il comando serve per effettuare la trasmissione dei dati del vento rilevati in una precisa posizione identificata da <latitudine>, <longitudine> e <altitudine>, all'interno di due date indicate da <timestamp-start> e <timestamp-end>.

Il significato e la lunghezza in byte dei campi sono i seguenti:

- <cmd>: lunghezza 2 byte, valore pari a 12 che indica la spedizione di un messaggio;
- <id-client>: lunghezza 2 byte, è l'identificativo dell'utente assegnato dall'azienda produttrice;
- <timestamp-start>: lunghezza 19 byte, timestamp di inizio delle rilevazioni inviate, il formato è 'YYYY-MM-DD hh:mm:ss';
- <timestamp-end>: lunghezza 19 byte, timestamp di fine delle rilevazioni inviate, il formato è 'YYYY-MM-DD hh:mm:ss';
- <latitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit della latitudine;
- <longitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit della longitudine;

- <altitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit dell'altitudine rispetto al livello del mare;
- velocità-n: lunghezza 3 byte, velocità del vento rilevata, i valori sono molteplici;
- direzione-n: lunghezza 3 byte, direzione del vento rilevata, i valori sono molteplici.

### Comandi del server

I comandi descritti di seguito sono quelli inviati dal server al client (server → client)

2 byte	2 byte
-----	
11   <id>	
-----	

Figura 3

Il comando serve per confermare la ricezione di un messaggio contenente i dati inviati da un client identificato dall'identificatore <id>.

### Segnalazione di errore

Nel caso in cui il server non riconosca l'identificatore del client, spedirà un messaggio di errore con il seguente formato.

2 byte	2 byte
-----	
13   <id>	
-----	

Figura 4

Il significato e la lunghezza in byte dei campi sono i seguenti:

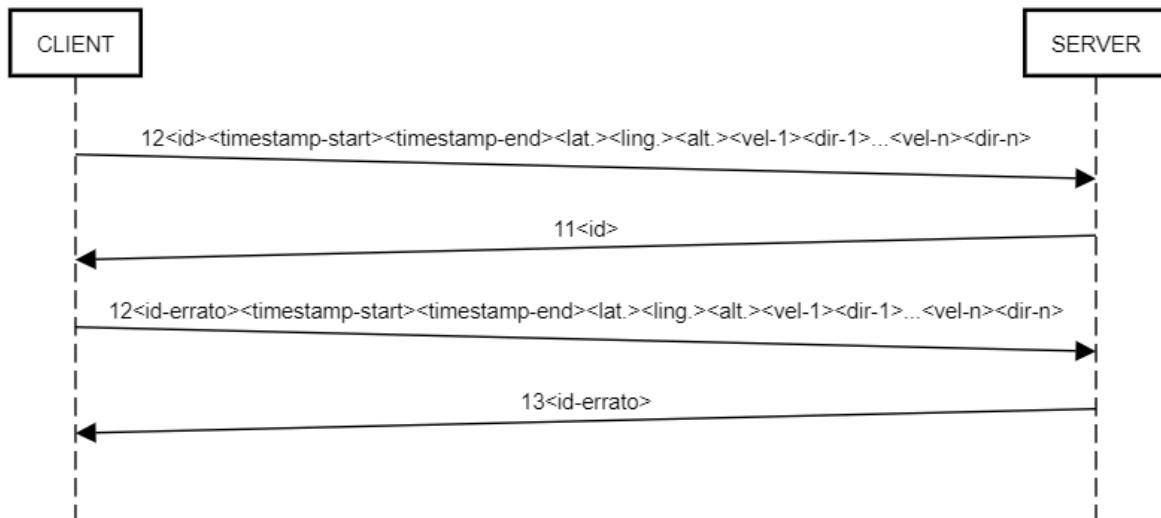
- 13 è il codice del comando di errore e occupa 2 byte;
- <id>: lunghezza 2 byte, rappresenta l'identificatore errato inviato dal client;

### Sequenza temporale<sup>9</sup>

---

<sup>9</sup> Il sequence diagram è stato creato usando [SequenceDiagramOrg](#)

## Protocollo Vento in Cloud



Il codice usato su [SequenceDiagram.org](http://SequenceDiagram.org) per ottenere la sequenza temporale illustrata sopra è il seguente:

title Protocollo Vento in Cloud

CLIENT->(1)SERVER:

12<id><timestamp-start><timestamp-end><lat.><ling.><alt.><vel-1><dir-1>...<vel-n><dir-n>

SERVER->(1)CLIENT:11<id>

CLIENT->(1)SERVER:

12<id-errato><timestamp-start><timestamp-end><lat.><ling.><alt.><vel-1><dir-1>...<vel-n><dir-n>

SERVER->(1)CLIENT:13<id-errato>

## Vento nel Mondo

### Richiesta

Tutti i dati raccolti dalla società produttrice vengono poi resi disponibili gratuitamente a scopo pubblicitario, in modo tale che chiunque ne necessiti possa recuperare le informazioni sui venti in un intervallo di tempo indicato nella richiesta del client. Anche in questo caso viene usato un protocollo comunicativo per lo scambio di questi dati.

Il candidato, fatte le opportune ipotesi aggiuntive che dovranno essere adeguatamente giustificate, progetti il protocollo di comunicazione che permette la diffusione dei dati raccolti in cloud a tutti i client che vogliono recuperare i dati sui venti in un determinato intervallo di tempo.

### Soluzione

5A Informatica

Request For Comment: 3

M. G. Maffucci  
I.I.S. A. Avogadro - Torino  
Gennaio 2022

## Protocollo Vento nel Mondo

### Stato di questo documento

Questo RFC è una possibile soluzione del 3° protocollo della verifica del 09/12/2022 ed è attualmente in stato di lavorazione.

### Riepilogo

Il protocollo **Vento nel Mondo** permette di recuperare le misurazioni della velocità e della direzione del vento raccolte in cloud grazie all'invio dei dati dei clienti che le hanno raccolte tramite i propri sensori. Questo documento descrive il protocollo e i pacchetti usati per la comunicazione. Inoltre spiega anche le ragioni delle scelte fatte nella sua progettazione.

### Ringraziamenti

Si ringrazia tutta la comunità che condivide su Internet il proprio lavoro, la vera conoscenza è possibile solo tramite una comunione di menti.

### Panoramica del protocollo

Il protocollo prevede la comunicazione tra un server in cloud che offre il servizio di diffusione dei dati sul venti raccolti in diverse parti del mondo, e i client che chiedono di scaricare tali dati. Si decide di **usare a livello trasporto il protocollo TCP** in quanto la quantità di dati raccolta dal server in cloud è probabilmente considerevole, e inoltre si vuole avere la certezza che tutti i dati inviati dal server siano ricevuti dal client che li richiede. Il protocollo TCP, implementando le caratteristiche di affidabilità e mantenimento della sessione necessarie per il trasferimento di file anche di grandi dimensioni, garantisce questo tipo di gestione. Inoltre il protocollo TCP effettua autonomamente la frammentazione e il riassemblaggio del messaggio nel caso fosse necessario, usando una modalità che risulta adatta per la tipologia di trasferimento richiesto. Le uniche caratteristiche che il protocollo **Vento nel Mondo** implementa sono l'identificazione del tempo delle rilevazioni che si vogliono scaricare usando un timestamp di inizio e uno di fine, e l'identificazione del luogo di rilevazione dei dati usando la latitudine, la longitudine e l'altitudine dal livello del mare del punto di rilevazione.

Il protocollo **non prevede l'indirizzamento** per il client in quanto i dati sono di pubblico dominio e scaricabili liberamente.

Il protocollo **non prevede la frammentazione e il riassemblaggio** in quanto questa funzionalità è demandata al protocollo TCP.

Il protocollo **non prevede l'incapsulamento** in quanto tutti i messaggi scambiati a livello applicativo tra il client ed il server contengono solo dati, senza bisogno di comandi particolari, in quanto il protocollo ha il solo scopo di inviare i dati del vento a qualsiasi client li richieda.

Il protocollo **non è connesso** in quanto la sessione viene mantenuta solo dal protocollo TCP.

Il protocollo **non è confermato** per non appesantire la comunicazione del server con i diversi client, sarà il protocollo TCP a fornire un servizio affidabile.

Il protocollo **non prevede la rilevazione di errori** per non appesantire la comunicazione del server con i diversi client.

Il protocollo **non prevede controllo del flusso** demandandolo completamente al protocollo TCP.

**Non è previsto il multiplexing** in quanto non è prevista aggregazione di dati.

Il protocollo **non prevede priorità dei pacchetti, non offre qualità del servizio e non gestisce la sicurezza**. Per quest'ultimo aspetto, se necessario, dovranno essere previste misure di sicurezza aggiuntive usando protocolli appositi a corredo di questo.

Il server ascolterà sulla porta 54321.

### Comandi del client

Il comando descritto di seguito è inviato dal client al server (client → server).

```
-----  
|      <timestamp-start>      |      <timestamp-end>      |  
-----  
|  <latitudine>    |  <longitudine>    |  <altitudine>    |  
-----
```

Figura 1

Il comando serve per chiedere la trasmissione dei dati del vento rilevati in una precisa posizione identificata da <latitudine>, <longitudine> e <altitudine>, all'interno di due date indicate da <timestamp-start> e <timestamp-end>.

Il significato e la lunghezza in byte dei campi sono i seguenti:

- <timestamp-start>: lunghezza 19 byte, timestamp di inizio delle rilevazioni inviate, il formato è 'YYYY-MM-DD hh:mm:ss';
- <timestamp-end>: lunghezza 19 byte, timestamp di fine delle rilevazioni inviate, il formato è 'YYYY-MM-DD hh:mm:ss';
- <latitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit della latitudine;
- <longitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit della longitudine;
- <altitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit dell'altitudine rispetto al livello del mare;

## Comandi del server

I comandi descritti di seguito sono quelli inviati dal server al client (server → client)

<timestamp-start>		<timestamp-end>	
<latitudine>	<longitudine>	<altitudine>	
<velocità-1>   <direzione-1>   ...   <velocità-n>   <direzione-n>			

Figura 2

Il comando serve inviare i dati del vento rilevati in una precisa posizione identificata da <latitudine>, <longitudine> e <altitudine>, all'interno di due date indicate da <timestamp-start> e <timestamp-end>.

Il significato e la lunghezza in byte dei campi sono i seguenti:

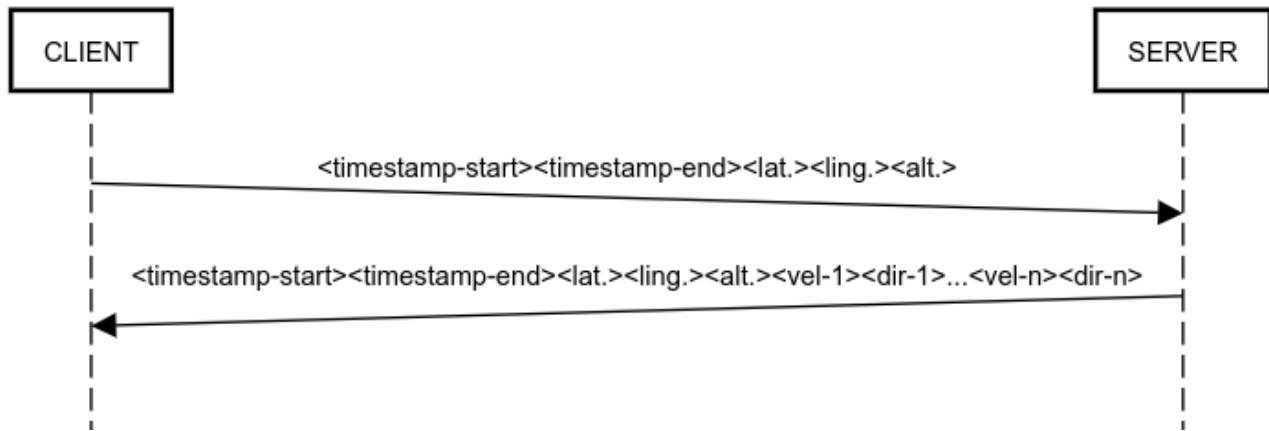
- <timestamp-start>: lunghezza 19 byte, timestamp di inizio delle rilevazioni inviate, il formato è 'YYYY-MM-DD hh:mm:ss';
- <timestamp-end>: lunghezza 19 byte, timestamp di fine delle rilevazioni inviate, il formato è 'YYYY-MM-DD hh:mm:ss';
- <latitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit della latitudine;
- <longitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit della longitudine;
- <altitudine>: lunghezza 4 byte, rappresentazione in codifica IEEE-754 a 32 bit dell'altitudine rispetto al livello del mare;
- velocità-n: lunghezza 3 byte, velocità del vento rilevata, i valori sono molteplici;
- direzione-n: lunghezza 3 byte, direzione del vento rilevata, i valori sono molteplici.

## Sequenza temporale<sup>10</sup>

---

<sup>10</sup> Il sequence diagram è stato creato usando [Sequence Diagram Org](#)

## Protocollo Vento nel Mondo



Il codice usato su [SequenceDiagram.org](#) per ottenere la sequenza temporale illustrata sopra è il seguente:

```
title Protocollo Vento nel Mondo
```

```
CLIENT->(1)SERVER: <timestamp-start><timestamp-end><lat.><ling.><alt.>
SERVER->(1)CLIENT:
<timestamp-start><timestamp-end><lat.><ling.><alt.><vel-1><dir-1>...<vel-n>
<dir-n>
```

## Protocolli applicativi di rete esistenti

Di seguito sono elencati alcuni protocolli applicativi di rete comunemente usati su Internet:

- [TELNET](#): terminale remoto;
- [FTP](#): trasferimento di file;
- [SMTP](#): invio della posta elettronica;
- [HTTP](#): trasferimento di dati ipertestuali;
- [BGP](#): protocollo di gestione del routing fra autonomous system diversi;
- [SNMP](#): protocollo che consente la configurazione, la gestione e la supervisione di apparati collegati in una rete (dispositivi di rete o terminali utente), riguardo a tutti gli aspetti che richiedono azioni di tipo amministrativo;
- [DNS](#): protocollo per la risoluzione dei nomi di dominio utilizzando un database distribuito;
- [DHCP](#): protocollo che permette ai dispositivi di una rete locale di ricevere automaticamente la configurazione di rete necessaria per stabilire una connessione nella LAN;
- [SSH](#): protocollo che utilizza tecniche crittografiche per effettuare operazioni di rete in modo sicuro su una rete non sicura;
- [NTP](#): protocollo usato per la sincronizzazione degli orologi tra sistemi di computer;
- [POP](#): protocollo per il recupero da parte di un client delle e-mail depositate in un server remoto;
- [IMAP](#): protocollo per il recupero da parte di un client delle e-mail depositate in un server remoto;

- [SFTP](#): protocollo per il trasferimento non sicuro di file, con un livello di complessità intermedio tra TFTP e FTP;
- [TFTP](#): protocollo di trasferimento file molto semplice, che veniva utilizzato ad es. per l'avvio di computer che non hanno dispositivi di memoria di massa, come i router;
- [ECHO](#): protocollo originariamente proposto per testare e misurare il [round-trip time](#) nelle reti IP.
- [DAYTIME](#): protocollo di spedizione della data e dell'ora fornita da un server;
- [TIME](#): protocollo di spedizione della data e dell'ora fornita da un server come numero di secondi a partire dal 1 gennaio 1900;
- [DISCARD](#): protocollo che butta via qualsiasi dato riceva;
- [CHARACTER GENERATOR](#): protocollo che genera e invia dati indipendentemente dall'input;
- [QOTD](#): protocollo che invia un breve messaggio (*Quote Of The Day*);
- [MESSAGE SEND PROTOCOL](#): protocollo per l'invio di brevi messaggi fra nodi di una rete;
- [WHOIS](#): protocollo per il recupero delle informazioni di utenti Internet.

## Protocolli applicativi di rete implementabili

Oltre ai normali protocolli applicativi di rete già standardizzati, è possibile creare dei propri, come ad esempio:

- SOMMA: il protocollo prevede che un client invii due valori numerici al server il quale ne determinerà la somma e la invierà al client;
- CALCOLATRICE: protocollo che prevede l'invio da parte di un client ad un server di due valori numerici e di un operatore aritmetico, ricevendo dal server il risultato dell'operazione aritmetica;
- CHAT: protocollo che permette di implementare una semplice chat unicast;
- CONVERTI: protocollo che prevede l'invio da parte di un client ad un server di una stringa che la convertirà in maiuscolo rispedendogliela;
- SPAM: protocollo che preveda lo spam di una frase inviata da un client a tutti i client connessi al server che la riceve.

## Esempi di protocolli applicativi di rete implementabili

Esempi di descrizioni di protocolli applicativi di rete li potrete trovare nelle seguenti specifiche di progetto, per l'avvio della relativa progettazione:

- [CHAT](#)
- [BATTAGLIA NAVALE v.1](#)
- [TRIS](#)
- [MASTERMIND](#)
- [FORZA 4](#)
- [COMBINAZIONE SEGRETA](#)
- [BATTAGLIA NAVALE v.2](#)

# Un protocollo di spedizione all'Esame di Stato

L'[Esame di Stato del 2018](#) per gli Istituti Tecnici Industriali di Informatica e Telecomunicazioni - Articolazione "Informatica" ha previsto una prova che richiede l'implementazione di un protocollo per la gestione delle spedizioni di pacchi, dal mittente al destinatario. Anche se il protocollo richiesto non è di tipo informatico, le competenze richieste sono fortemente sviluppate durante il corso di TPSI che nel quinto anno è completamente incentrata allo sviluppo di protocolli di reti.

Di seguito sono proposte alcune soluzioni dell'Esame per consentirne un'analisi critica da parte dello studente:

- [Soluzione Sistemi e Reti Esame di Stato 2018, Prof. Mauro De Brnardis](#);
- [Prova scritta esame di stato 2018 - Informatica ITIA, Prof.ssa Giselda De Vita](#);
- [Esame di Stato 2018 ITIA - Soluzione](#), Zanichelli online per la scuola.

## Protocols - CLIL

### CLIL Listening and Writing - Protocols

The following video give an overview about what Internet protocols are. Listen each video and answer to the questions. At the end of the work send the answer to your teacher.

1. What is a protocol? [What is a protocol?](#)
2. Why is so important that protocols are written? [What is a protocol?](#)
3. Which are the two things that a protocol has to do? [What is a protocol?](#)
4. Why are protocols so highly structured? [Why are protocols so highly structured?](#)
5. What are clients and servers and which is the difference between them? [What is a client? What is a server?](#)
6. What is the robustness principle? [What is the robustness principle?](#)
7. Regarding Internet protocol stack, explain what the two goals "Separation of concern" and "Reuse of good ideas" are. [What is the internet protocol stack?](#)
8. How can UDP be used other then its normal purpose? [Can you still deploy new transport protocols?](#)
9. What is a peer-to-peer infrastructure? [What is a peer to peer system?](#)

# I sistemi distribuiti: modelli architetturali hardware e software

# I sistemi distribuiti e loro evoluzione [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 1 - I sistemi distribuiti

- **1.1 I sistemi distribuiti pp.2-9**
  - I sistemi distribuiti
  - Classificazione dei sistemi distribuiti
  - Benefici legati alla distribuzione
  - Svantaggi legati alla distribuzione
- **1.2 Evoluzione dei sistemi distribuiti e dei modelli architetturali pp.10-21**
  - Premessa - [CLIL Listening and Speaking - Quantum Computers](#)
  - Architetture distribuite hardware: dalle SISD al cluster di PC - [CLIL Listening and Speaking - Flynn Taxonomy and Cluster Architecture](#) - [Tassonomia di Flynn](#)
  - Architetture distribuite software: dai terminali remoti ai sistemi completamente distribuiti - [CLIL Listening and Speaking - Difference Between Software Architecture and Software Design](#)
  - Architetture a livelli - [Il middleware](#), [CLIL Listening and Speaking - Middleware](#), [Multitier architecture](#)
  - Conclusioni

## CLIL Listening and Speaking - Quantum Computers

In these videos you will find how normal computer evolution has taken a strange path due to quantum mechanics. Watch the three videos and discuss the topic with your classmate.

1. [Transistors & The End of Moore's Law](#)
2. [Quantum Computers Explained – Limits of Human Technology](#)
3. [How Does a Quantum Computer Work?](#)

## CLIL Listening and Speaking - Flynn Taxonomy and Cluster Architecture

In these videos you will understand Flynn's taxonomy and why cluster architecture is widely used in big enterprises. Watch the videos and discuss the topics with your classmate.

4. [Flynn's Taxonomy of Parallel Machines - Georgia Tech - HPCA: Part 5](#)

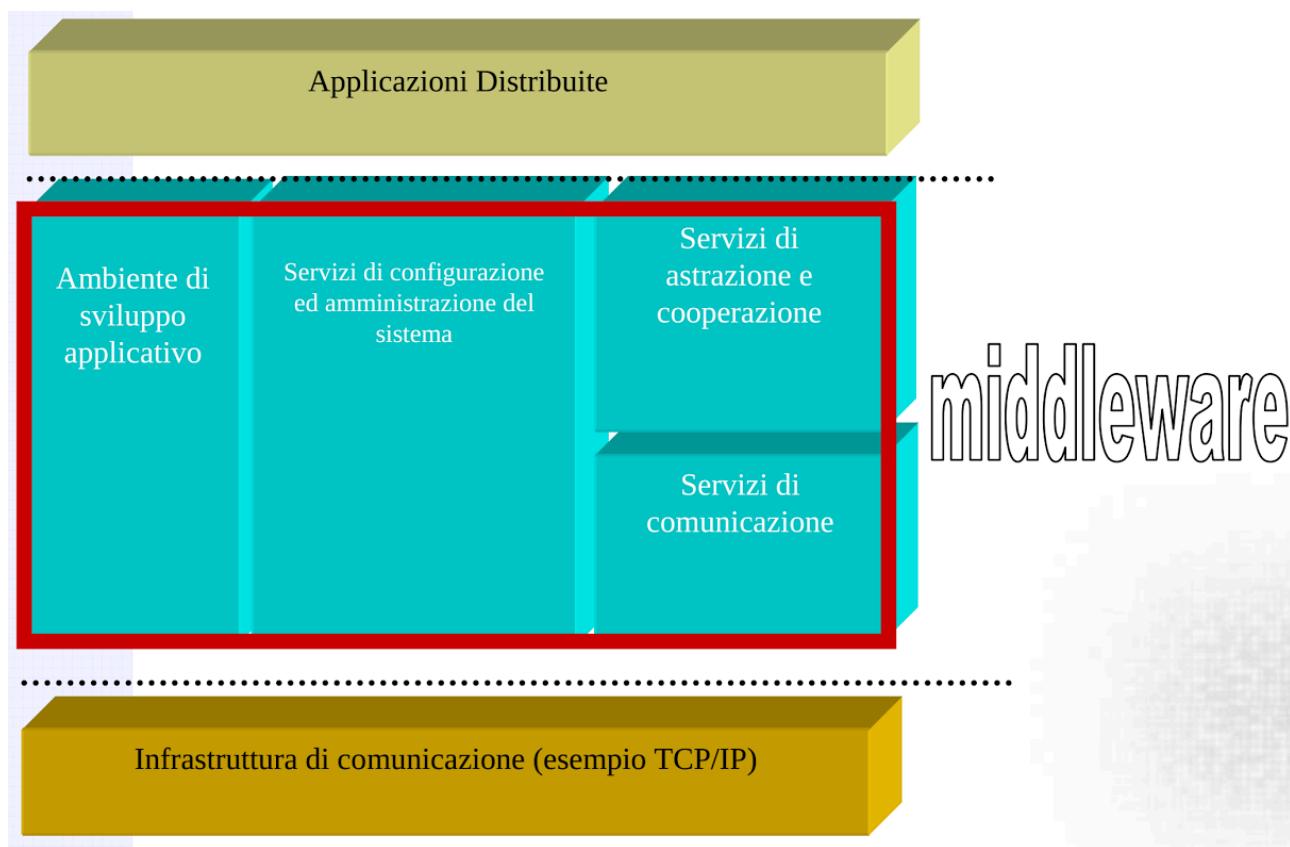
# CLIL Listening and Speaking - Difference Between Software Architecture and Software Design

In these videos you will understand the difference between software architecture and software design, in order to catch how to collocate the distributed software architecture. Watch the videos and discuss the topics with your classmate.

## 5. [Difference Between Software Architecture and Software Design](#)

### Il middleware

Il middleware è un software di connessione che consiste di un insieme di servizi e/o di ambienti di sviluppo di applicazioni distribuite che permettono a più entità (processi, oggetti, ecc.) residenti su uno o più elaboratori, di interagire attraverso una rete di interconnessione a dispetto di differenze dei protocolli di comunicazione, delle architetture dei sistemi locali, dei sistemi operativi, ecc.



I servizi o funzionalità offerti dal middleware sono:

- **servizi di astrazione e cooperazione:** rappresentano il cuore del middleware e comprendono:
  - **directory service:** provvede alla identificazione e alla localizzazione delle entità, rendendo le applicazioni al di sopra del middleware indipendenti dal sottosistema di instradamento dei messaggi (**trasparenza di locazione**);

- **security service:** finalizzato alla protezione degli elementi critici, come i dati e i servizi applicativi, utilizzando tecniche di autenticazione, autorizzazione e crittografia;
  - **time service:** assicura che tutti i clock interni tra client e server siano sincronizzati entro un accettabile livello di varianza.
- **servizi per le applicazioni:** permettono alle applicazioni di avere un accesso diretto con la rete e con i servizi offerti dal middleware, ad esempio, un servizio orientato alle transazioni può fornire un supporto per un accesso transazionale a basi di dati eterogenee;
  - **servizi di amministrazione del sistema:** servono per effettuare monitoraggio, configurazione e pianificazione di interventi sul sistema;
  - **servizio di comunicazione:** questo servizio offre delle API che permettono all'applicazione distribuita di scambiare informazioni tra tutte le sue componenti residenti su altri elaboratori aventi anche caratteristiche hardware o software diverse. Lo scopo di questo servizio è di nascondere le disomogeneità dovute alla rappresentazione dei dati usata dai vari elaboratori, dai sistemi operativi locali e dalle diverse reti che costituiscono l'infrastruttura della piattaforma. I paradigmi di comunicazione possono essere basati su RPC, messaggistica applicativa, o altre;
  - **ambiente di sviluppo applicativo:** offre diversi tool, come strumenti di aiuto alla scrittura di programmi, debugging, gestione di applicativi sia ad oggetti che a processi, *Interface Definition Language* che permette l'interconnessione tra moduli scritti in diversi linguaggi di programmazione e residenti su elaboratori distinti.

Il middleware offre molti servizi, anche più di quanti in realtà potrebbero essere necessari per un'applicazione, e quindi a livello progettuale si dovrà mediare tra le prestazioni dell'applicazione e l'efficienza esecutiva, cercando di includere solo i servizi necessari per non appesantire troppo il sistema finale.

## CLIL Listening and Speaking - Middleware

In this video you will understand what middleware is. Watch the videos and discuss the topics with your classmate.

### 6. [Middleware Concepts](#)

## CLIL Listening and Speaking - n-Tier Architecture

In these videos you will understand how n-tier architecture works. Watch the videos and discuss the topics with your classmate.

7. [3 Tier Client Server Architecture](#)
8. [n-Tier Architecture Explained](#)
9. [What Is n-Tier Architecture?](#)
10. [N-Tier Architecture for kids](#)
11. [Middleware Architecture](#)

# Linguaggi per lo scambio dei dati in rete con Java

# I linguaggi XML e JSON [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 1 - Architettura di rete e formati per lo scambio dei dati

- **1.5 Le applicazioni di rete**
  - [Lab 1](#) Il linguaggio XML pp.53-61
  - [Lab 5](#) Il linguaggio JSON pp.93-101

## CLIL Comprehension - What JSON is

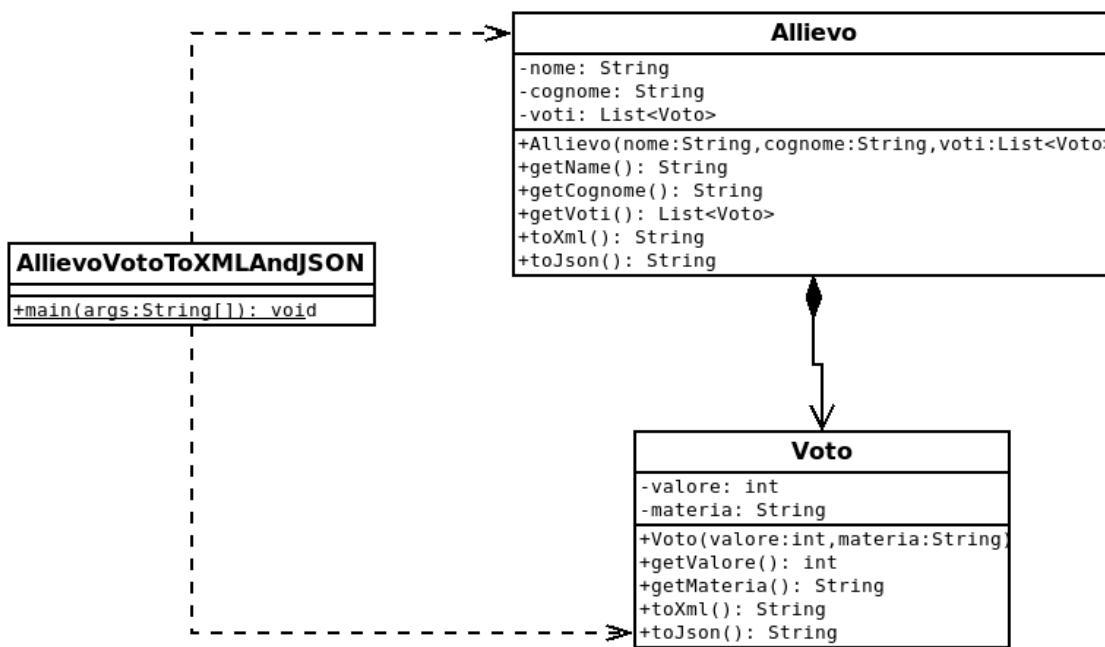
Watch the following video in order to understand what JSON is, in which cases is used and which is the structure of a JSON document.

[JSON Crash Course](#)

## Creazione di un XML e di un JSON usando Java

L'applicazione presentata nel paragrafo crea delle stringhe XML e JSON partendo dai dati degli allievi di una classe e dei voti conseguiti nelle diverse materie. Gli studenti e i voti sono collezionati in ArrayList, ma potrebbero essere recuperati tramite l'interrogazione di un database. Le stringhe XML e JSON sono direttamente visualizzate, ma possono essere salvate in un file o inviate remotamente tramite una API. L'estrema versatilità di questi linguaggi di scambio dei dati rende possibile la realizzazione di diversi scenari a seconda delle necessità dell'applicazione sviluppata.

L'UML dell'applicazione è il seguente e, il codice<sup>11</sup> è mostrato subito dopo.



<sup>11</sup> Il codice è stato sviluppato dal Prof. Bellucci.

**AllievoVotoToXMLAndJSON.java**

```
import java.util.ArrayList;
import java.util.List;

public class AllievoVotoToXMLAndJSON {
    public static void main(String[] args) {
        // Istanziazione della lista dei voti dell'allievo
        List<Voto> list = new ArrayList<>();
        // Aggiunta dei voti nella lista dei voti
        list.add(new Voto(4,"tps"));
        list.add(new Voto(8,"info"));
        list.add(new Voto(7,"italiano"));
        // Istanziazione dell'allievo
        Allievo a=new Allievo("pippo", "pluto", list);
        System.out.println(list.get(0).toXml());
        System.out.println(a.toXml());
        System.out.println(list.get(0).toJson());
        System.out.println(a.toJson());
    }
}
```

**Allievo.java**

```
import java.util.List;

public class Allievo {
    private String nome;
    private String cognome;
    private List<Voto> voti;

    public Allievo(String nome, String cognome, List<Voto> voti) {
        this.nome = nome;
        this.cognome = cognome;
        this.voti = voti;
    }

    public String getNome() {
        return nome;
    }

    public String getCognome() {
        return cognome;
    }

    public List<Voto> getVoti() {
        return voti;
    }

    // Creazione dell'XML usando i valori delle variabili di istanza
    // dell'allievo e dei voti
    public String toXml(){
        String xml = "<Allievo>";

```

```
xml += "\t<nome>";
xml += nome;
xml += "</nome>\n";
xml += "\t<cognome>";
xml += cognome;
xml += "</cognome>\n";
xml += "\t<voti>\n";
for (Voto voto : voti) {
    xml += voto.toXml();
}
xml += "\t</voti>\n";
xml += "</Allievo>\n";
return xml;
}

// Creazione del JSON usando i valori delle variabili di istanza
// dell'allievo e dei voti
public String toJson() {
    String json = "{";
    json += "\n\t\"nome\":";
    json += "\"\"" + nome + "\",\n";
    json += "\t\"cognome\":";
    json += "\"\"" + cognome + "\",\n";
    json += "\t\"voti\": [";
    for (Voto voto : voti) {
        json += voto.toJson();
        json += ",";
    }
    json= json.substring(0, json.length()-1);
    json += "\n\t]\n";
    json += "}\n";
    return json;
}
}
```

## Voto.java

```
public class Voto {
    int valore;
    String materia;

    public Voto(int valore, String materia) {
        this.valore = valore;
        this.materia = materia;
    }

    public int getValore() {
        return valore;
    }

    public String.getMateria() {
        return materia;
    }
}
```

```
public String toXml() {
    String xml = "\t\t<Voto>\n";
    xml += "\t\t\t<valore>";
    xml += valore;
    xml += "</valore>\n";
    xml += "\t\t\t<materia>";
    xml += materia;
    xml += "</materia>\n";
    xml += "\t\t</Voto>\n";
    return xml;
}

public String toJson() {
    String json = "\n\t\t{\n";
    json += "\t\t\t\"valore\":";
    json += valore;
    json += ",\n";
    json += "\t\t\t\"materia\":";
    json += "\"";
    json += materia;
    json += "\"";
    json += "\n\t\t}";
    return json;
}
}
```

L'output dell'applicazione sarà il seguente.

```
<Voto>
    <valore>4</valore>
    <materia>tps</materia>
</Voto>

<Allievo>  <nome>pippo</nome>
    <cognome>pluto</cognome>
    <voti>
        <Voto>
            <valore>4</valore>
            <materia>tps</materia>
        </Voto>
        <Voto>
            <valore>8</valore>
            <materia>info</materia>
        </Voto>
        <Voto>
            <valore>7</valore>
            <materia>italiano</materia>
        </Voto>
    </voti>
</Allievo>
```

```
        {
            "valore":4,
            "materia":"tps"
        }
    {
        "nome":"pippo",
        "cognome":"pluto",
        "voti": [
            {
                "valore":4,
                "materia":"tps"
            },
            {
                "valore":8,
                "materia":"info"
            },
            {
                "valore":7,
                "materia":"italiano"
            }
        ]
}
```

## JAXB

Per manipolare il formato XML da Java è possibile usare l'API JAXB (*Java Architecture for XML Binding*), usando delle semplici annotazioni. Nelle versioni recenti di Java è necessario usare un progetto Gradle o Maven. Si veda il seguenti link per avere indicazioni su come usare l'API.

- [JAXB \(with Java 11\) - Tutorial<sup>12</sup>](#)

L'esercizio precedente può essere trasformato in un progetto Gradle in grado di fare il marshalling e l'unmarshalling come mostrato di seguito.

Creare un nuovo progetto Gradle e, per prima cosa bisogna aggiungere i plugins e le dipendenze necessarie nel file build.gradle per consentire il corretto uso di JAXB, oltre a sistemare il nome della classe main.

### **build.gradle**

```
plugins {
    // Apply the application plugin to add support for building a CLI
    // application in Java.
    id 'application'
    id 'java-library' // ← aggiungere
}

repositories {
```

<sup>12</sup> [\[Solved\]: Exception in thread “main” com.sun.xml.internal.bind.v2.runtime.IllegalAnnotationsException: 3 counts of IllegalAnnotationExceptions](#) può essere utile per gli errori

```
// Use Maven Central for resolving dependencies.
mavenCentral()
}

dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation 'org.junit.jupiter:junit-jupiter:5.9.1'

    // This dependency is used by the application.
    implementation 'com.google.guava:guava:31.1-jre'
    implementation 'com.sun.xml.bind:jaxb-impl:2.3.3' // ← aggiungere
}

application {
    // Define the main class for the application.
    mainClass = 'allievo.AllievoVotoToXMLAndJSON' // ← modificare
}

tasks.named('test') {
    // Use JUnit Platform for unit tests.
    useJUnitPlatform()
}
```

### AllievoVotoToXMLAndJSON.java

```
package allievo;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

public class AllievoVotoToXMLAndJSON {

    private static final String VOTI_XML = "./voti.xml";

    public static void main(String[] args)
        throws JAXBException, FileNotFoundException {
        // Istanziazione della lista dei voti dell'allievo
        List<Voto> list = new ArrayList<>();
        // Aggiunta dei voti nella lista dei voti
        list.add(new Voto(4, "tps"));
        list.add(new Voto(8, "info"));
        list.add(new Voto(7, "italiano"));
        // Istanziazione dell'allievo
```

```
Allievo allievo = new Allievo("pippo", "pippoli", list);

// create JAXB context and instantiate marshaller
JAXBContext context = JAXBContext.newInstance(Allievo.class);
Marshaller m = context.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

// Write to System.out
m.marshal(allievo, System.out);

// Write to File
m.marshal(allievo, new File(VOTI_XML));

// get variables from our xml file, created before
System.out.println();
System.out.println("Output from our XML File: ");
Unmarshaller um = context.createUnmarshaller();
Allievo allievo2 = (Allievo) um.unmarshal(new FileReader(
    VOTI_XML));

System.out.println("Allievo: " + allievo2.getNome() + ", " +
    allievo2.getCognome());
List<Voto> voti = allievo2.getVoti();
for (Voto voto : voti) {
    System.out.println(voto.getMateria() + ": " +
        voto.getVoto());
}
}
```

### Allievo.java

```
package allievo;

import java.util.List;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

// Sets the namespace for XML documents
@XmlRootElement(namespace = "allievo")
// Allows to define the order in which the fields are written in
// the XML file
@XmlType(propOrder = {"nome", "cognome", "voti"})
public class Allievo {

    // Define the XML element which will be used.
    @XmlElement
    private String nome;
```

```
@XmlElement
private String cognome;
// Generates a wrapper element around XML elements
@XmlElementWrapper(name = "voti")
// Define the XML element which will be used with a new name
@XmlElement(name = "valutazione")
private List<Voto> voti;

public Allievo() {
}

public Allievo(String nome, String cognome, List<Voto> voti) {
    this.nome = nome;
    this.cognome = cognome;
    this.voti = voti;
}

public String getNome() {
    return nome;
}

public String getCognome() {
    return cognome;
}

public List<Voto> getVoti() {
    return voti;
}
}
```

**Voto.java**

```
package allievo;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

// Define the root element for an XML tree
@XmlRootElement(name = "voto")
// Allows to define the order in which the fields are written in the XML file
@XmlType(propOrder = {"voto", "materia"})
public class Voto {

    @XmlElement
    private int voto;
    @XmlElement
    private String materia;

    public Voto() {
```

```
}

public Voto(int valutazione, String materia) {
    this.voto = valutazione;
    this.materia = materia;
}

public int getVoto() {
    return voto;
}

public String getMateria() {
    return materia;
}
}
```

## DOM - Document Object Model

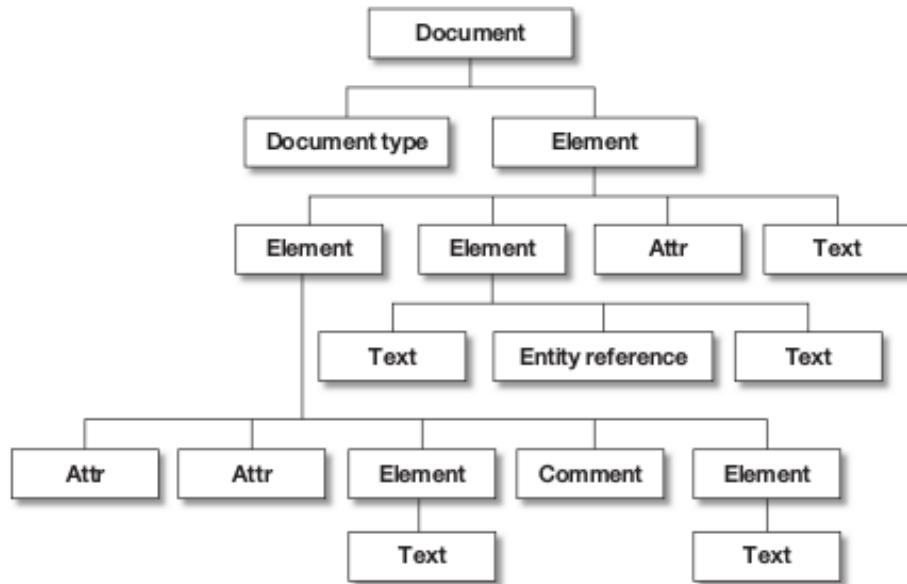
**DOM** (*Document Object Model*) è una delle modalità fondamentali per effettuare il **parsing** di un documento XML. In informatica, il *parsing*, *analisi sintattica* o *parsificazione* è un processo che analizza un flusso continuo di dati in ingresso (letti per esempio da un file o una tastiera) in modo da determinare la sua struttura grazie ad una data grammatica formale. Un **parser** è un programma che esegue questo compito.

**DOM** è uno standard, come XML, che vede la sua origine all'interno del *World Wide Web Consortium* (W3C, <http://www.w3.org>).

**DOM** è stato progettato per rappresentare il contenuto e il modello di un documento XML (o comunque strutturato, come ad esempio HTML) all'interno di un linguaggio di programmazione, fornendo una rappresentazione dei documenti strutturati come modello orientato agli oggetti.

Molti linguaggi di programmazione e tecnologie (JavaScript, Java, CORBA, ecc.) hanno API predefinite per accedere al **DOM** di un documento XML, rendendo di fatto **DOM** una specifica indipendente dalla piattaforma e dal linguaggio utilizzato.

Un tipico documento XML potrebbe avere una struttura **DOM** analoga a quella mostrata nella figura seguente.



**DOM** permette di creare in memoria una rappresentazione ad albero del documento XML e consente di accedere velocemente a qualsiasi sua parte utilizzando l'interfaccia `org.w3c.dom.Node`. Questa interfaccia **DOM** permetterà di ereditare diverse interfacce specifiche per XML, come `Element`, `Document`, `Attr` e `Text`.

Si osservi nell'immagine precedente come la struttura ad albero sia rigorosamente implementata. Ad esempio, il nodo `Element` contiene dei valori testuali (uno di loro potrebbe essere un titolo), al cui valore non sarà comunque consentito accedere direttamente tramite un metodo del nodo `Element`, si dovrà ancora accedere al nodo figlio di tipo `Text` per poter ottenere il testo associato. Sebbene questa tecnica possa sembrare strana, permette però di preservare completamente il modello **DOM** ad albero, rendendo gli algoritmi di attraversamento dell'albero molto semplici, con pochi casi particolari e quindi facilmente automatizzabili.

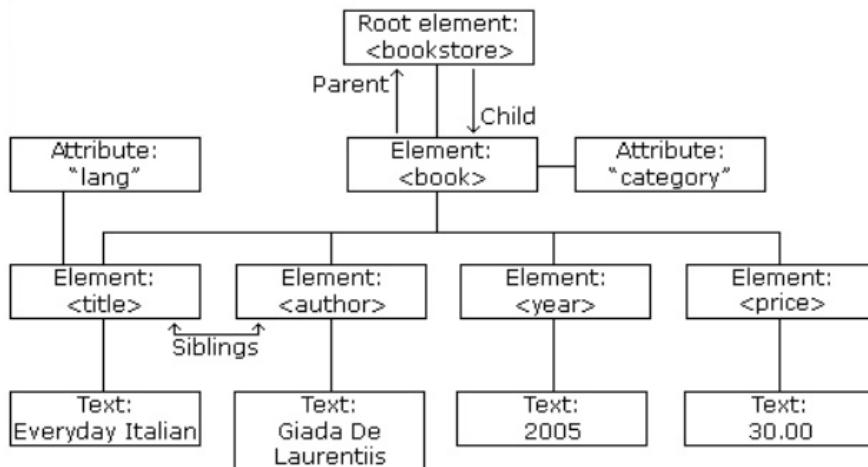
La rappresentazione di un documento XML tramite **DOM** permette di individuare esattamente la gerarchia ad albero del documento.

Se ad esempio abbiamo il seguente documento XML:

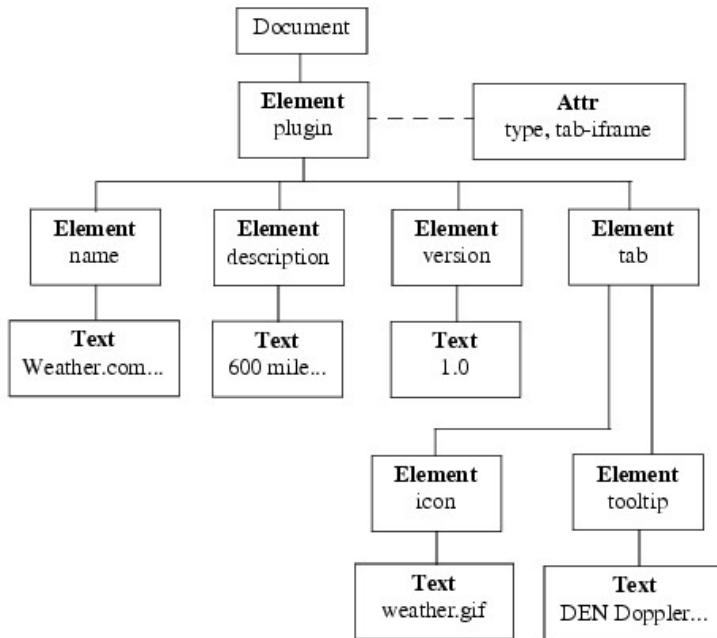
```
<bookstore>
    <book category="narrative">
        <title lang="it">
            Everyday Italian
        </title>
        <author>
            Giada De Laurentiis
        </author>
        <year>
            2005
        </year>
        <price>
            30.00
        </price>
```

```
</book>  
</bookstore>
```

Il **DOM** corrispondente sarà il seguente



Dato il seguente **DOM**, si provi a scrivere il corrispondente file XML.



## Il parsing di un documento XML [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

### Unità 1 - Architettura di rete e formati per lo scambio dei dati

- **1.5 Le applicazioni di rete pp.53-61 e pp.81-101**
  - **Lab 1** Il linguaggio XML pp.53-61
  - **Lab 4** Il parsing XML con Java: le specifiche JAXP pp.81-92

- Lab 5 Il linguaggio JSON pp.93-101

## API per la gestione di documenti XML con il linguaggio Java

Il linguaggio di programmazione Java prevede una **API** (*Application Program Interface*) standard per la gestione di documenti XML denominata **JAXP** (*Java API for XML Processing*). Tra le elaborazioni che le classi dei package che costituiscono JAXP consentono di effettuare sono fondamentali le seguenti:

- validazione di un documento XML rispetto a uno schema XSD;
- produzione di un nuovo documento XML ed eventuale salvataggio in un file di testo;
- lettura di un documento XML contenuto in un file di testo o reso disponibile come risorsa in rete o come stream di byte in memoria ed eventuale verifica della correttezza;
- trasformazione di un documento XML.

La lettura di un documento XML in una struttura dati che è possibile gestire da parte del codice di un programma, denominato **parser**, è tecnicamente definita **parsing** e, come abbiamo già detto, rappresenta un processo che analizza un flusso continuo di dati in ingresso in modo da determinare la sua struttura grazie ad una data grammatica formale.

Storicamente il parsing di un documento XML da parte di un programma si basa su una delle due seguenti modalità fondamentali:

- **DOM** (*Document Object Model*): il documento viene interamente «mappato» in una struttura ad albero che rappresenta la struttura gerarchica degli elementi del documento XML e il codice può liberamente «navigare» attraverso i nodi dell'albero. La modalità **DOM** consente di modificare il documento XML originale e anche di crearne dinamicamente uno completamente nuovo. La tecnica di parsing **DOM** richiede la memorizzazione in strutture dati del programma dell'intero albero che rappresenta il documento, cosa che, nel caso di documenti voluminosi, può richiedere una grande quantità di memoria.
- **SAX** (*Simple API for XML*): la lettura del documento è condotta in modo autonomo dal parser e avviata dal codice che istanzia una classe di gestione degli «eventi» (*event handler*), i cui metodi sono invocati automaticamente in corrispondenza di specifiche tipologie di elementi presenti nel documento. L'ordine in cui gli elementi sono analizzati dal parser non è prevedibile a priori, inoltre **questa modalità non consente di modificare il documento XML originale**. La modalità di parsing **SAX** è estremamente efficiente in quanto il documento XML viene letto in modo sequenziale e solo gli elementi intercettati dal gestore degli eventi sono presi in considerazione. Il costo dell'efficienza è rappresentato dalla relativa rigidità della tecnica.

I vari package che costituiscono JAXP permettono di effettuare sia il parsing di tipo DOM, sia quello in modalità SAX. In questo documento ci occuperemo solo del parsing di tipo DOM.

I package principali di JAXP sono i seguenti:

- **java.xml.parsers**: interfaccia comune per i parser dei vari produttori;
- **org.w3c.dom**: API per il parsing di tipo DOM;
- **org.xml.sax**: API per il parsing di tipo SAX;
- **javax.xml.transform**: API per la gestione delle regole di trasformazione di un documento XML (comprende le API per il parsing di tipo StAX);
- **javax.xml.stream**: API per la gestione della lettura e della scrittura dei file in formato XML.

## Parsing di un documento XML con DOM

Per poter effettuare il parsing di un documento XML è necessario seguire una serie di passaggi precisi:

1. Ottenere un parser DocumentBuilderFactory per produrre un oggetto albero DOM.
2. Creare un DocumentBuilder per generare un oggetto Document
3. Creare un Document da un file o uno stream.
4. Estrarre l'elemento Element radice del documento.
5. Esaminare gli attributi.
6. Esaminare gli elementi figli.

La API DocumentBuilderFactory è una classe astratta che ci permetterà di avviare la parsificazione del documento XML. È un API di tipo factory.

La classe DocumentBuilderFactory presenta, tra gli altri, i seguenti metodi specifici:

- newInstance - Utilizzato per ottenere una nuova istanza di un parser DocumentBuilderFactory
- newDocumentBuilder - Crea una nuova istanza di un oggetto DocumentBuilder necessaria per creare un documento DOM

La API DocumentBuilder è una classe astratta che ci permetterà di creare l'istanza di un documento DOM da un documento XML.

La classe DocumentBuilder presenta, tra gli altri, il seguente metodo specifico:

- parse - Parsifica il contenuto del file XML o dello stream di byte in memoria e restituisce un nuovo oggetto di tipo Document DOM, cioè costruisce l'albero DOM del file XML

DOM utilizza un insieme di interfacce a cui fare riferimento durante la programmazione e il parsing di un documento XML in modalità DOM comporta quindi la costruzione dell'albero i cui nodi sono istanze di classi che implementano l'interfaccia Node. Node è il tipo di base di un DOM.

Oltre all'interfaccia Node può essere utilizzata anche una sua interfaccia derivata, come quelle elencate di seguito e definite nel package org.w3c.dom:

- Document - Intero albero del documento XML, è l'oggetto albero DOM
- Element - Elemento di un documento XML, rappresenta la maggior parte degli elementi da gestire

- Attr - Attributo di un elemento
- Text - Contenuto testuale di un elemento

La struttura DOM può essere gestita sia utilizzando il tipo generico Node, sia attraverso tipi specifici (Element, Attr, ecc.). Molti dei metodi di navigazione dell'albero DOM, come getParent() e getChildren() che vedremo più avanti, lavorano attraverso l'interfaccia di base Node, quindi l'albero può essere attraversato senza preoccuparsi della specifica struttura dei tipi.

L'interfaccia Node definisce, fra gli altri, i seguenti metodi per la navigazione dell'albero DOM.

- appendChild - Aggiunge un nuovo nodo figlio dopo i figli esistenti
- getAttributes - Restituisce la collezione degli attributi
- getChildNodes - Restituisce la collezione dei nodi figli
- getFirstChild - Restituisce il primo nodo figlio
- getLastChild - Restituisce l'ultimo nodo figlio
- getNextSibling - Restituisce il prossimo nodo fratello
- getNodeName - Restituisce il nome del nodo
- getNodeValue - Restituisce il valore del contenuto del nodo
- getNodeType - Restituisce il tipo di nodo
- getParentNode - Restituisce il nodo padre
- getPreviousSibling - Restituisce il nodo fratello precedente
- hasAttributes - Restituisce vero se sono presenti attributi, falso altrimenti
- hasChildren - Restituisce vero se esistono nodi figli, falso altrimenti
- insertBefore - Aggiunge un nuovo nodo figlio prima del nodo figlio specificato
- removeChild - Rimuove il nodo figlio specificato
- replaceChild - Sostituisce il nodo figlio specificato con un nuovo nodo figlio
- setNodeValue - Imposta il valore del contenuto del nodo

L'interfaccia Document derivata da Node aggiunge, tra gli altri, i seguenti metodi specifici:

- createAttribute - Crea un nodo attributo
- createElement - Crea un nodo element
- createTextNode - Crea un nodo testuale
- getDocumentElement - Restituisce il nodo radice del documento
- getElementsByTagName - Restituisce la collezione degli elementi che hanno il nome specificato

- `getXmlEncoding` - Restituisce la codifica del documento XML
- `getXmlVersion` - Restituisce la versione XML del documento

L'interfaccia `Element` derivata da `Node` aggiunge, tra gli altri, i seguenti metodi specifici:

- `getAttribute` - Restituisce il valore dell'attributo specificato
- `getElementsByTagName` - Restituisce la collezione degli elementi che hanno il nome specificato
- `getTagName` - Restituisce il nome dell'elemento
- `hasAttribute` - Restituisce vero se l'elemento ha l'attributo specificato, falso altrimenti
- `removeAttribute` - Rimuove dall'elemento l'attributo specificato
- `setAttribute` - Aggiunge l'attributo specificato impostandone il valore

L'interfaccia `Attr` derivata da `Node` aggiunge, tra gli altri, i seguenti metodi specifici:

- `getName` - Restituisce il nome dell'attributo
- `getOwnerElement` - Restituisce l'elemento a cui appartiene l'attributo
- `getValue` - Restituisce il valore dell'attributo
- `setValue` - Imposta il valore dell'attributo

L'interfaccia `Text` derivata da `Node` aggiunge, tra gli altri, i seguenti metodi specifici:

- `getWholeText` - Restituisce il contenuto testuale
- `replaceWholeText` - Sostituisce il contenuto testuale

L'interfaccia `NodeList` rappresenta una collezione ordinata di oggetti di tipo `Node` e definisce i seguenti metodi:

- `getLength` - Restituisce il numero di nodi della collezione
- `item` - Restituisce un nodo della collezione a partire dal suo indice di posizione

## Esempi di parsing di un documento XML con DOM

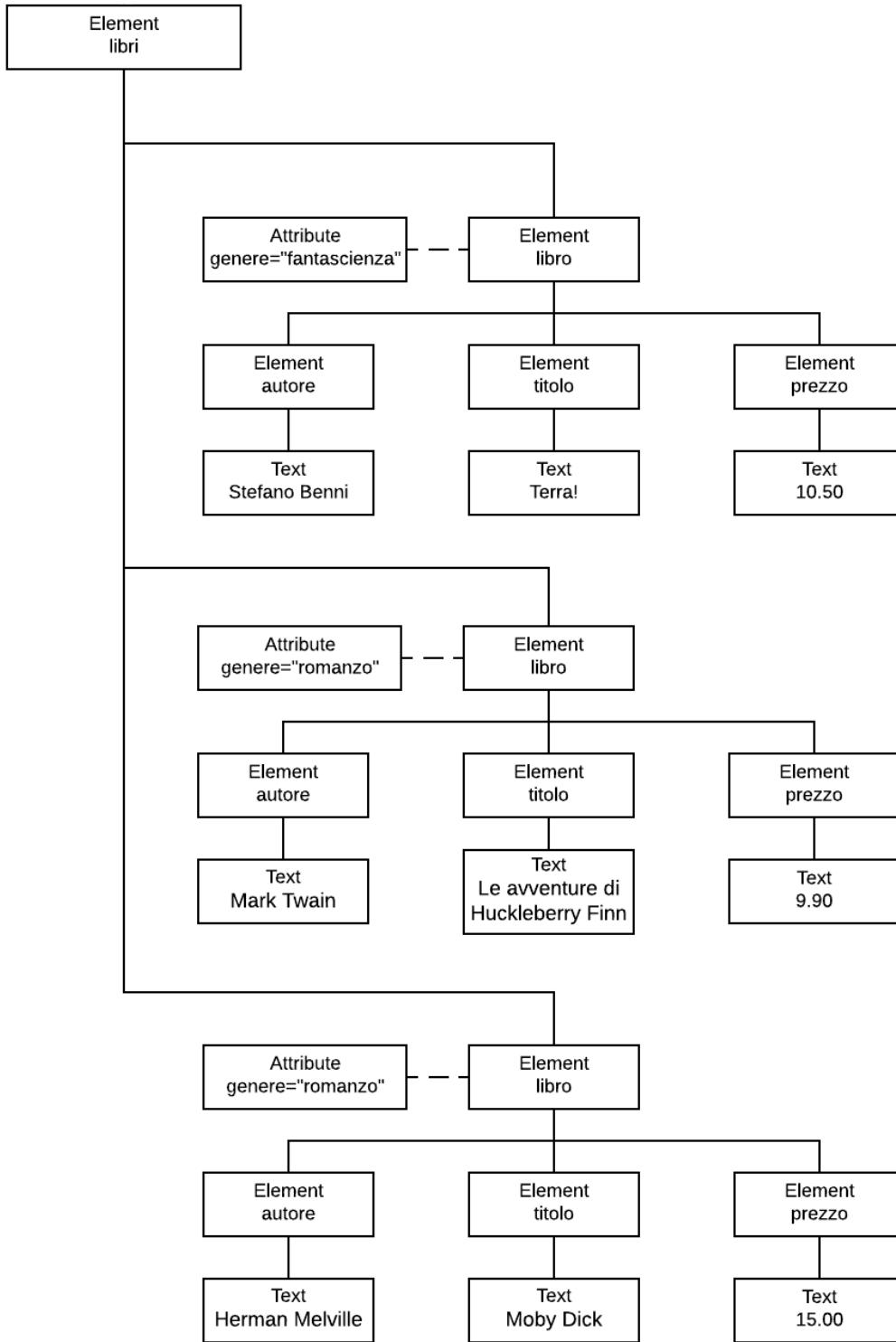
### Esempio 1

Consideriamo il seguente documento XML che utilizzeremo per fare un esempio di parsing.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libri>
    <libro genere="fantascienza">
        <autore>Stefano Benni</autore>
        <titolo>Terra!</titolo>
        <prezzo>10.50</prezzo>
    </libro>
    <libro genere="romanzo">
```

```
<autore>Mark Twain</autore>
<titolo>Le avventure di Huckleberry Finn</titolo>
<prezzo>9.90</prezzo>
</libro>
<libro genere="romanzo">
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <prezzo>15.00</prezzo>
</libro>
</libri>
```

Il DOM del documento XML sarà il seguente.



Di seguito viene fornito un esempio di un semplice parser del file XML mostrato in precedenza, presupponendo di conoscere i nomi dei nodi attraversati.

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class Parser {
```

```
public void parseDocument(String filename)
    throws ParserConfigurationException, SAXException, IOException {
/*
 * DocumentBuilderFactory è una API di tipo factory necessaria per
 * ottenere un parser per la produzione di un oggetto albero DOM da un
 * documento XML.
 * Istanziamo un DocumentBuilderFactory per la generazione di un albero
 * DOM da un documento XML.
 */
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
/*
 * DocumentBuilder è una API per ottenere una interfaccia di tipo
 * "Document" DOM da un documento XML.
 * Istanziamo un DocumentBuilder per generare un oggetto di tipo
 * "Document".
 */
DocumentBuilder builder = factory.newDocumentBuilder();
/*
 * L'interfaccia "Document" rappresenta un intero documento XML (o HTML).
 * Concettualmente è la radice dell'albero del documento e fornisce
 * l'accesso principale ai dati del documento.
 * Il metodo parse(File f) parsifica il contenuto del file XML e
 * restituisce un nuovo oggetto "Document" DOM, cioè costruiamo l'albero
 * DOM del file XML.
 */
Document document = builder.parse(filename);
/*
 * L'interfaccia Element rappresenta un elemento di un documento XML
 * (o HTML).
 * Estraiamo l'elemento radice del documento.
 */
Element root = document.getDocumentElement();
// L'oggetto di tipo NodeList conterrà l'insieme degli elementi estratti
// dall'albero DOM a partire dalla posizione indicata.
NodeList nodelist;
// L'oggetto di tipo Element servirà per estrarre i singoli elementi
// recuperati tramite un oggetto NodeList.
Element element;

// Estrazione di tutti gli elementi figli il cui nome è "libro".
nodelist = root.getElementsByTagName("libro");
if (nodelist != null && nodelist.getLength() > 0) {
    for (int i = 0; i < nodelist.getLength(); i++) {
        // Scorriamo ogni singolo elemento "libro".
        element = (Element)nodelist.item(i);
        // Recupero del valore dell'attributo specificato.
        System.out.println("Genere: " + element.getAttribute("genere"));
        // Per recuperare i valori degli elementi figli bisogna
        // istanziare un nuovo oggetto NodeList. Utilizzeremo un metodo
        // creato proprio per questo scopo, getTextValue().
        String titolo = getTextValue(element, "titolo");
        String autore = getTextValue(element, "autore");
        float prezzo = Float.parseFloat(getTextValue(element, "prezzo"));
    }
}
```

```
        System.out.println("Titolo: " + titolo);
        System.out.println("Autore: " + autore);
        System.out.println("Prezzo: " + prezzo);
    }
}
}

private String getTextValue(Element element, String tag) {
    String value = null;
    NodeList nl;

    nl = element.getElementsByTagName(tag);
    if (nl != null && nl.getLength() > 0) {
        // Recupero del valore del nodo - getNodeValue() - del primo figlio
        // - getChild() - dell'elemento nella posizione 0 della
        // collezione dell'oggetto NodeList - item(0) -.
        value = nl.item(0).getFirstChild().getNodeValue();
    }

    return value;
}

public static void main(String[] args)
    throws ParserConfigurationException, SAXException, IOException {
    Parser parser = new Parser();

    parser.parseDocument("libri.xml");
    /**
     * Avendo il documento xml in una stringa (nell'esempio di nome s) è
     * possibile utilizzare uno stream di byte direttamente in memoria:
     * ByteArrayInputStream stream = new ByteArrayInputStream(s.getBytes());
     * parser.parseDocument(stream);
     */
}
}
```

L'esecuzione del parser fornirà il seguente output.

```
Genere: fantascienza
Titolo: Terra!
Autore: Stefano Benni
Prezzo: 10.5
Genere: romanzo
Titolo: Le avventure di Huckleberry Finn
Autore: Mark Twain
Prezzo: 9.9
Genere: romanzo
Titolo: Moby Dick
Autore: Herman Melville
Prezzo: 15.0
```

## Esempio 2

Di seguito viene fornito un esempio alternativo in cui i nodi del DOM vengono visualizzati tramite un ciclo, presupponendo di non conoscerne il nome.

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class Parser {

    public void parseDocument(String filename)
        throws ParserConfigurationException, SAXException, IOException {
        /**
         * DocumentBuilderFactory è una API di tipo factory necessaria per
         * ottenere un parser per la produzione di un oggetto albero DOM da un
         * documento XML.
         * Istanziamo un DocumentBuilderFactory per la generazione di un albero
         * DOM da un documento XML.
         */
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        /**
         * DocumentBuilder è una API per ottenere una interfaccia di tipo
         * "Document" DOM da un documento XML.
         * Istanziamo un DocumentBuilder per generare un oggetto di tipo
         * "Document".
         */
        DocumentBuilder builder = factory.newDocumentBuilder();
        /**
         * L'interfaccia "Document" rappresenta un intero documento XML (o HTML).
         * Concettualmente è la radice dell'albero del documento e fornisce
         * l'accesso principale ai dati del documento.
         * Il metodo parse(File f) parsifica il contenuto del file XML e
         * restituisce un nuovo oggetto "Document" DOM, cioè costruiamo l'albero
         * DOM del file XML.
         */
        Document document = builder.parse(filename);
        /**
         * L'interfaccia Element rappresenta un elemento di un documento XML
         * (o HTML).
         * Estraiamo l'elemento radice del documento.
         */
        Element root = document.getDocumentElement();
        // L'oggetto di tipo NodeList conterrà l'insieme degli elementi estratti
        // dall'albero DOM a partire dalla posizione indicata.
        NodeList nodelist;
        // L'oggetto di tipo Element servirà per estrarre i singoli elementi
        // recuperati tramite un oggetto NodeList.
        Element element;
        // Estrazione di tutti gli elementi figli il cui nome è "libro".
        nodelist = root.getElementsByTagName("libro");
        if (nodelist != null && nodelist.getLength() > 0) {
```

```
for (int i = 0; i < nodelist.getLength(); i++) {
    // Scorriamo ogni singolo elemento "libro".
    element = (Element)nodelist.item(i);
    // Recupero del valore dell'attributo specificato.
    System.out.println("Genere: " + element.getAttribute("genere"));
    // Per recuperare i valori degli elementi figli bisogna
    // istanziare un nuovo oggetto NodeList.
    NodeList nl;
    // Si recuperano i nodi figli del nuovo NodeList.
    nl = element.getChildNodes();
    if (nl != null && nl.getLength() > 0) {
        for (int j = 0; j < nl.getLength(); j++) {
            // Si scorrono i singoli elementi del NodeList.
            Node n = nl.item(j);
            // Si controlla che il nodo estratto sia proprio un
            // elemento. Allo scopo viene utilizzata la costante
            // ELEMENT_NODE.
            if (n.getNodeType() == Node.ELEMENT_NODE) {
                // Recupero del nome dell'elemento.
                String nodeName = n.getNodeName();
                // Recupero del valore testuale memorizzato nel
                // nodo.
                String value = n.getTextContent();
                System.out.println(nodeName + ": " + value);
            }
        }
    }
}
}

public static void main(String[] args)
    throws ParserConfigurationException, SAXException, IOException {
Parser parser = new Parser();

parser.parseDocument("libri.xml");
/**
 * Avendo il documento xml in una stringa (nell'esempio di nome s) è
 * possibile utilizzare uno stream di byte direttamente in memoria:
 * ByteArrayInputStream stream = new ByteArrayInputStream(s.getBytes());
 * parser.parseDocument(stream);
 */
}
}
```

L'esecuzione del parser fornirà il seguente output.

```
Genere: fantascienza
autore: Stefano Benni
titolo: Terra!
prezzo: 10.50
Genere: romanzo
autore: Mark Twain
titolo: Le avventure di Huckleberry Finn
```

```
prezzo: 9.90
Genere: romanzo
autore: Herman Melville
titolo: Moby Dick
prezzo: 15.00
```

## Esempio 3

Di seguito viene fornito un esempio alternativo in cui i nodi del DOM vengono visualizzati tramite un ciclo, presupponendo di non conoscerne il nome. La visualizzazione riflette la struttura del documento XML. Il codice è più complesso dei precedenti in quanto utilizza, fra le altre cose, il concetto di ricorsione.

```
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.DOMException;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class Parser {
    public static void main(String[] args) throws SAXException, IOException,
ParserConfigurationException {
        /**
         * DocumentBuilderFactory è una API di tipo factory necessaria per
         * ottenere un parser per la produzione di un oggetto albero DOM da un
         * documento XML.
         * Istanziamo un DocumentBuilderFactory per la generazione di un albero
         * DOM da un documento XML.
         */
        DocumentBuilderFactory documentBuilderFactory =
            DocumentBuilderFactory.newInstance();
        /**
         * Il metodo setNamespaceAware() specifica che il parser prodotto
         * da questo codice supporterà i namespace XML.
         */
        documentBuilderFactory.setNamespaceAware(true);
        /**
         * DocumentBuilder è una API per ottenere una interfaccia di tipo
         * "Document" DOM da un documento XML.
         * Istanziamo un DocumentBuilder per generare un oggetto di tipo
         * "Document".
         */
        DocumentBuilder documentBuilder =
            documentBuilderFactory.newDocumentBuilder();
        /**
         * L'interfaccia "Document" rappresenta un intero documento XML (o HTML).
         * Concettualmente è la radice dell'albero del documento e fornisce
         * l'accesso principale ai dati del documento.
         * Il metodo parse(File f) parsifica il contenuto del file XML e
```

```
* restituisce un nuovo oggetto "Document" DOM, cioè costruiamo l'albero
* DOM del file XML.
*/
Document doc = documentBuilder.parse("books.xml");
/***
 * L'interfaccia Node è il tipo fondamentale di un DOM e rappresenta un
 * singolo nodo nell'albero del documento.
 * Il metodo getDocumentElement() permette di accedere direttamente al
 * nodo figlio del documento, che in questo caso è la radice del
 * documento.
*/
Node root = doc.getDocumentElement();
// Il metodo printNodes() elabora solo gli elementi del documento DOM,
// evitando gli spazi e le tabulazioni inserite per chiarezza di
// lettura.
// Il valore restituito da printNodes() non viene prelevato in quanto
// ora non serve, essendo la prima chiamata del metodo ricorsivo.
printNodes(root, 0);
}

/***
 * Il metodo printNodes() elabora solo gli elementi del documento DOM,
 * evitando gli spazi e le tabulazioni inserite per chiarezza di lettura.
 * È un metodo ricorsivo, cioè richiama se stesso.
*/
private static int printNodes(Node node, int level) throws DOMException {
    // Il nodo viene elaborato solo se è un elemento, cioè se non è un
    // nodo formato da soli return o tabulazioni inseriti nell'XML per
    // chiarezza di lettura.
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        String name = node.getNodeName();
        // Si stampano degli spazi per ottenere la stampa indentata
        // secondo il livello del nodo. Il metodo printSpaces() stampa
        // tanti spazi quanto indicato nel parametro attuale, moltiplicati
        // per 7.
        printSpaces(level);
        System.out.print(name + " : ");
        String value = null;
        // Prelevo il valore del nodo che viene rappresentato come
        // un nodo figlio.
        if (node.getChildNodes() != null
            && node.getChildNodes().getLength() > 0) {
            value = node.getFirstChild().getNodeValue();
        }
        // La stampa viene effettuata solo se il valore contiene
        // informazioni significative.
        if (value != null) {
            value = value.trim();
        } else {
            value = "";
        }
        System.out.println(value);
        // Un oggetto che implementa l'interfaccia NameNodeMap viene usato
        // per rappresentare una collezione di nodi a cui si accede
    }
}
```

```
// tramite nome. In questo caso viene usato per collezionare tutti
// gli attributi del nodo di riferimento.
NamedNodeMap attributes = node.getAttributes();
if (attributes != null && attributes.getLength() > 0) {
    for (int i = 0; i < attributes.getLength(); i++) {
        // Si accede alla collezione di attributi tramite indice.
        Node n = attributes.item(i);
        printSpaces(level);
        System.out.println("A - " + n.getNodeName() + " : " +
                           n.getNodeValue().trim());
    }
}
// Per ogni nodo figlio viene invocato ricorsivamente il metodo
// aumentando di 1 il livello di indentazione.
NodeList list = node.getChildNodes();
if (list != null && list.getLength() > 0) {
    for (int i = 0; i < list.getLength(); i++) {
        level = printNodes(list.item(i), level + 1);
    }
}
// Avendo richiamato ricorsivamente il metodo, ogni richiamo che termina
// decrementerà di uno il livello di indentazione della stampa, proprio
// perché "si torna indietro".
return level - 1;
}

// Il metodo printSpaces() stampa tanti spazi quanto sono quelli indicati
// nel parametro formale moltiplicato per 7.
private static void printSpaces(int level) {
    for (int i = 0; i < level; i++) {
        System.out.print("\t");
    }
}
```

L'esecuzione del parser fornirà il seguente output.

```
libri :
libro :
A - genere : fantascienza
    autore : Stefano Benni
    titolo : Terra!
    prezzo : 10.50
libro :
A - genere : romanzo
    autore : Mark Twain
    titolo : Le avventure di Huckleberry Finn
    prezzo : 9.90
libro :
A - genere : romanzo
    autore : Herman Melville
    titolo : Moby Dick
```

[Vai all'indice](#)

prezzo : 15.00

# Web e HTTP

# Comunicazione Web e HTTP [Mdn Web Docs]

**STUDIARE:** sul sito [Mdn Web Docs](#)

1. [An overview of HTTP](#)
2. [Evolution of HTTP](#)
3. [HTTP Messages](#)
4. [A typical HTTP session](#)
5. [Connection management in HTTP/1.x](#)
6. [Identifying resources on the Web](#)
7. [MIME types \(IANA media types\)](#)
8. [HTTP request methods](#)
9. [Safe \(HTTP Methods\)](#)
10. [Idempotent](#)
11. [Cacheable](#)
12. [HTTP response status codes](#)

Watch the following video by [ByteByteGo](#).

[HTTP/1 to HTTP/2 to HTTP/3](#)

## Web e siti

Un sito Web è una collezione di pagine web che possono contenere testo, immagini, audio e video, la cui prima pagina viene definita home page.

Tutti i siti Web sono identificabili tramite uno specifico indirizzo Internet denominato **URL** (*Uniform Resource Locator*) che deve essere inserito in un browser per poter accedere ai contenuti del sito Web.

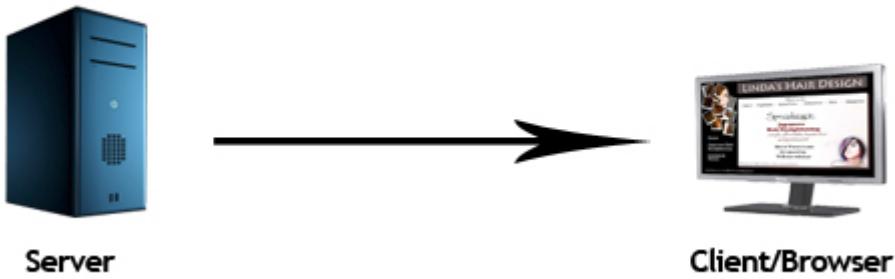
I siti Web possono essere di due tipi, **statici** o **dinamici**.

## Sito Web statico

Un sito Web statico è costituito essenzialmente da pagine HTML, fogli di stile CSS o programmi scritti in un linguaggio di scripting che agiscono solo sull'interfaccia del lato client. Il codice delle pagine è quasi del tutto statico come le informazioni contenute in esse.

L'interazione avviene solo tra il server che fornisce le pagine HTML e il client che le richiede.

## Static Website



## Sito Web dinamico

Un sito Web dinamico è costituito da pagine il cui contenuto cambia dinamicamente, recuperandolo da un database o da un **CMS** (*Content Management System*). Aggiornando il contenuto del database si aggiorna di conseguenza il contenuto del sito Web.

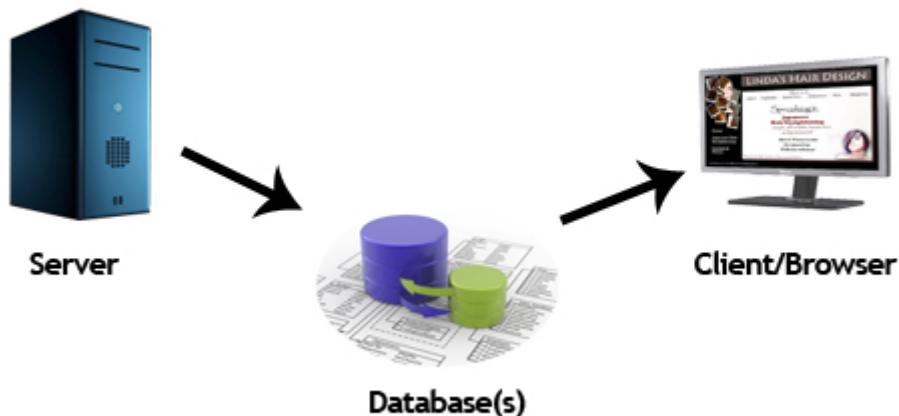
Il contenuto dinamico viene generato utilizzando degli **scripting client side** e/o **server side**.

Gli **script client side** generano il contenuto sulla base dell'input fornito dall'utente; il browser scarica la pagina Web dal server ed esegue il codice presente in essa per fornire le informazioni all'utente.

Gli **script server side** sono delle applicazioni che operano sul server generando in esso le pagine che saranno effettivamente inviate al client.

In entrambi i casi vengono utilizzate altre tecnologie, come un database, per derivare dinamicamente i contenuti delle pagine.

## Dynamic Website



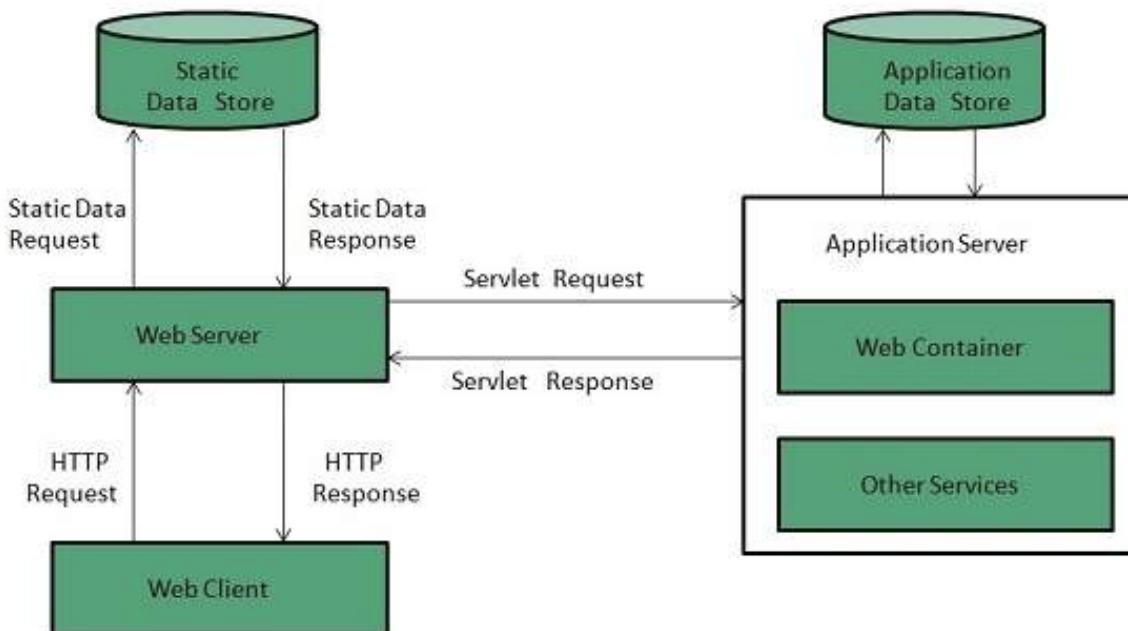
# Web server e Application server

I **server** sono dei programmi che accettano richieste fatte da altri programmi, detti **client**, fornendo una risposta adeguata alla richiesta. Sono usati per **gestire le risorse** presenti sulla rete e per **fornire dei servizi**.

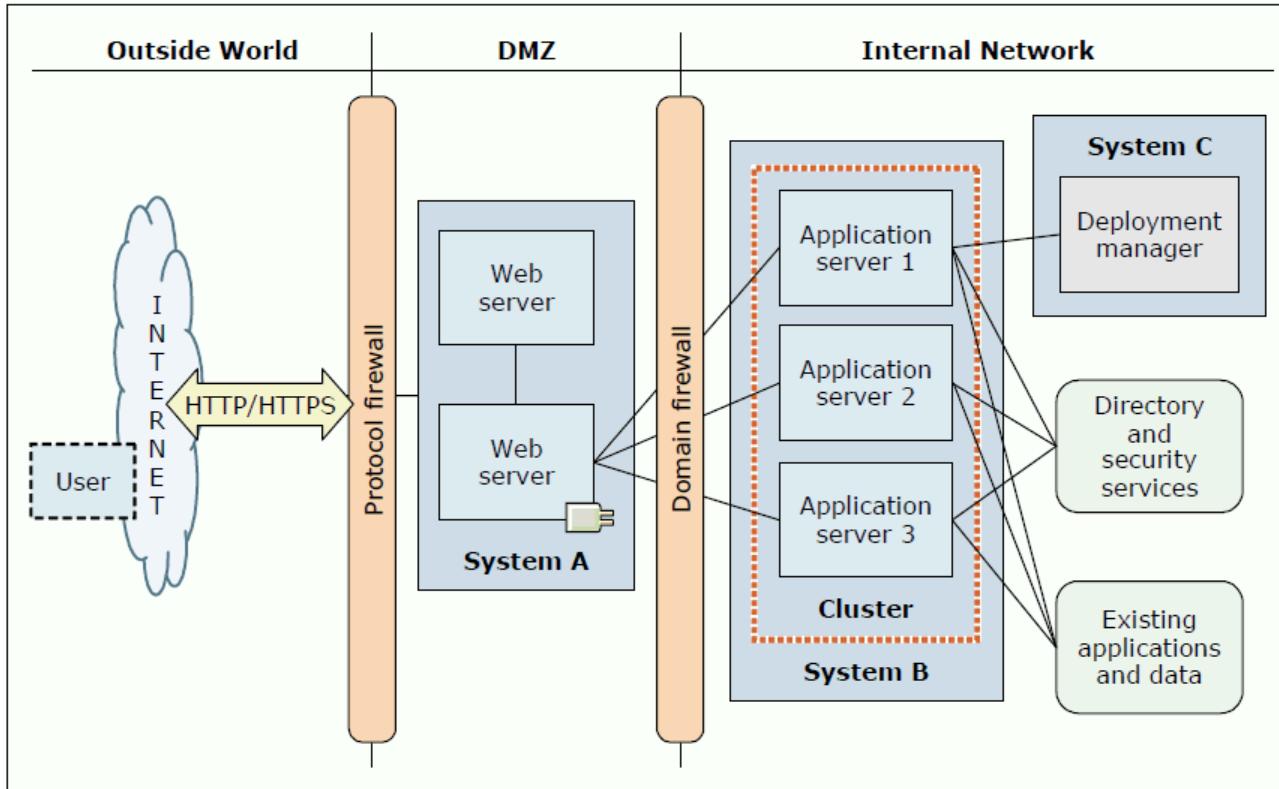
I server si distinguono in due categorie, **Web server** e **Application server**.

Un **Web server**, come ad es. **Apache Tomcat**, gestisce di solito il contenuto Web di un sito. Quando un client **richiede una pagina** al Web server, se questa viene trovata sarà spedita tramite una risposta HTTP al client, altrimenti verrà inviata una risposta *HTTP status code* di tipo "Error 404 not found".

Se il client dovesse richiedere un **qualsiasi altro tipo di risorsa** il Web server dovrà contattare un **Application server** per recuperare i dati necessari per la costruzione della risposta HTTP.



Un **Application server**, come ad esempio **Glassfish** o **JBoss**, è un prodotto costituito solitamente da più componenti che risiedono nel **middle-tier** di una **architettura server centric**. Un **Application server** fornisce **servizi middleware** come **sicurezza** e **gestione** del **sistema, accesso e persistenza** dei **dati**. Un **Application server** è un tipo di server progettato per operare e offrire servizi e applicazioni a utenti finali e organizzazioni.

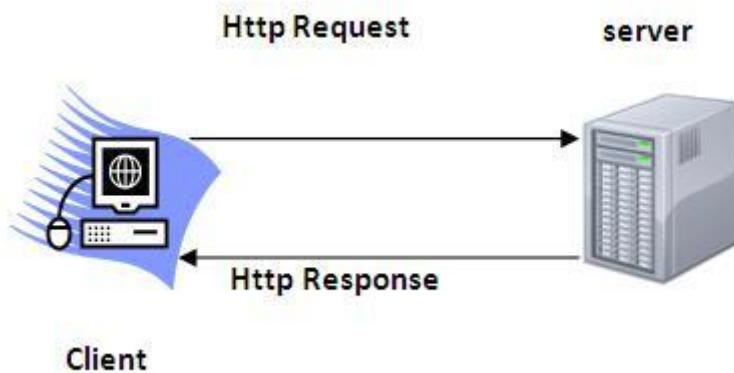


## HTTP (HyperText Transfer Protocol)

**HTTP** è un protocollo applicativo di rete di tipo client-server, utilizzato per la distribuzione collaborativa di informazioni ipertestuali.

HTTP si basa sulla suite TCP/IP, utilizzata per il trasferimento sul WWW (*World Wide Web*) dei dati come immagini, file, interrogazioni, file HTML, ecc. Tramite questo protocollo applicativo di rete un Web server e un browser (client) possono scambiarsi dei dati attraverso il Web utilizzando una connessione TCP, con il server che risponde di default sulla porta 80.

HTTP è un protocollo di tipo **request-response** cioè il client invia una richiesta al server per poter accedere ad una risorsa o ad un servizio di interesse, e il server risponderà fornendo l'accesso alla risorsa o il servizio. A meno che la comunicazione non venga interrotta in modo anomalo, un server fornirà sempre una risposta, anche solo per comunicare la non esistenza della risorsa richiesta attraverso una coppia di valori *HTTP status-code*.



Il protocollo applicativo di rete HTTP è **media independent**, cioè attraverso HTTP è possibile inviare qualsiasi tipo di contenuto; la gestione spetterà al client e al server.

Inoltre HTTP è fondamentalmente un protocollo **stateless**, cioè ogni richiesta è indipendente da quelle precedenti e la connessione TCP si conclude al momento della consegna della pagina richiesta al client. La versione 1.1 di HTTP permette comunque di mantenere attiva la connessione TCP fino allo scadere di un timeout di inattività della connessione.

## I metodi HTTP

I metodi HTTP più utilizzati sono i seguenti:

HTTP Request	Descrizione
<b>GET</b>	Richiede la risorsa individuata nell'URL
<b>POST</b>	Chiede al server di accettare il body allegato alla richiesta. È simile alla GET con la differenza che la richiesta trasporta più informazioni
<b>HEAD</b>	Analoga alla GET, ma restituisce solo l'header, tralasciando il body
<b>PUT</b>	Chiede di caricare il contenuto del body nell'URL richiesto
<b>DELETE</b>	Chiede di cancellare la risorsa identificata nell'URL della richiesta
<b>OPTION</b>	Chiede la lista dei metodi HTTP a cui risponde la risorsa corrispondente all'URL

## GET request

Nel metodo **GET** la coppia nome/valore, detta **query string**, è inviata all'interno dell'URL della richiesta GET. In una richiesta GET non esistendo il body, i dati necessari per la richiesta vengono trasportati direttamente come parametri nell'URL.

```
http://localhost:8084/Dati/Utente?nome=Maria+Grazia&cognome=Maffucci
```

Un **header** possibile della **richiesta GET** è il seguente:

```
Host: localhost:8084
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101
Firefox/50.0
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8084/Dati/
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Un possibile **header** della **risposta** del server è il seguente:

```
Content-Length: 251
Content-Type: text/html; charset=UTF-8
Date: Sat, 07 Jan 2017 21:28:50 GMT
Server: Apache-Coyote/1.1
```

Mentre il **body** della **risposta** del server potrebbe essere la pagina HTML seguente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dati</title>
    <style>
      p.spesso {
        font-weight: bold;
      }
    </style>
  </head>
  <body>
    <p>Ciao Maria Grazia Maffucci</p>
    <p class="spesso">Sat Jan 07 22:28:50 CET 2017</p>
  </body>
</html>
```

Un altro esempio:

The HTTP Method	Path to the source on Web Server	Parameters to the server	Protocol Version Browser supports
GET /RegisterDao.jsp?user=ravi&pass=java	/RegisterDao.jsp	?user=ravi&pass=java	HTTP/1.1
The Request Headers	Host: www.javatpoint.com User-Agent: Mozilla/5.0 Accept-text/xml,text/html,text/plain,image/jpeg Accept-Language: en-us,en Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8 Keep-Alive: 300 Connection: keep-alive		

Di seguito sono elencate alcune caratteristiche delle richieste GET:

- può essere salvata nella cache;
- viene salvata nella cronologia del browser;
- può essere salvata nei bookmark;
- non dovrebbe mai essere usata con dati sensibili;
- ha dei limiti di lunghezza;
- dovrebbe essere usato solo per recuperare dei dati.

## POST request

Nel metodo **POST** la coppia nome/valore, detta **query string**, è inviata all'interno del body del messaggio HTTP della richiesta POST.

```
http://localhost:8084/Dati/Utente
Content-Type: application/x-www-form-urlencoded
Content-Length: 34

nome=Maria+Grazia&cognome=Maffucci
```

Un **header** possibile della **richiesta POST** è il seguente:

```
Host: localhost:8084
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101
Firefox/50.0
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8084/Dati/
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Un possibile **header** della **risposta** del server è il seguente:

Content-Length: 251  
Content-Type: text/html; charset=UTF-8  
Date: Sat, 07 Jan 2017 21:54:49 GMT  
Server: Apache-Coyote/1.1

Mentre il **body** della **risposta** del server potrebbe essere la pagina HTML seguente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dati</title>
    <style>
      p.spesso {
        font-weight: bold;
      }
    </style>
  </head>
  <body>
    <p>Ciao Maria Grazia Maffucci</p>
    <p class="spesso">Sat Jan 07 22:28:50 CET 2017</p>
  </body>
</html>
```

Un altro esempio:

The HTTP Method	Path to the source on Web Server	Protocol Version Browser supports
	Post /RegisterDao.jsp	HTTP/1.1
The Request Headers	Host: www.javatpoint.com User-Agent: Mozilla/5.0 Accept: text/xml,text/html,text/plain,image/jpeg Accept-Language: en-us,en Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8 Keep-Alive:300 Connection:keep-alive User=ravi&pass=java	} Message body

Di seguito sono elencate alcune caratteristiche delle richieste POST:

- non può essere salvata nella cache;
- non viene salvata nella cronologia del browser;
- non può essere salvata nei bookmark;
- non ha dei limiti di lunghezza;

## Content Type

Il **content type**, anche conosciuto come tipo **MIME** (*Multipurpose Internet Mail Extension*) è un **header HTTP** che specifica cosa si sta inviando attraverso un browser.

**MIME** è uno standard Internet usato per estendere le limitate capacità delle e-mail, permettendo di inserirvi i caratteri non-ASCII, gli attachment multipli ad un singolo messaggio e gli attachment costituiti da suoni, immagini e video, e supportando una lunghezza illimitata dei messaggi.

I **content type** sono di diversi tipi, fra i più utilizzati ci sono:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- images/png
- images/gif
- audio/mp3
- video/mp4
- video/quicktime

## HTTP status code

Le coppie di valori *status-code* di HTTP sono delle risposte standard restituiti dai Web server. Questi codici aiutano ad identificare la causa di un problema nel caso non si riuscisse a trovare la pagina Web o la risorsa richiesta.

Gli *status-code* HTTP includono sia un codice numerico, sia una frase esplicativa. Ad esempio, la linea di stato HTTP **500: Internal Server Error** segnala l'impossibilità del server di soddisfare la richiesta.

Un elenco degli HTTP status code è possibile trovarlo a questo [link](#).

# Applicazioni lato server in Java: servlet

# Applicazioni lato server con codice separato: CGI e servlet [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 4 - Applicazioni lato server con codice separato: CGI e servlet

- **4.1 Le applicazioni lato server pp.236-250**
  - La programmazione server-side
  - Modello a codice separato
    - [What is a Servlet?](#)
  - Struttura di una servlet
    - [Servlet Container](#)
  - La classe HttpServlet
    - [Servlet API](#)
    - [Servlet Interface \(approfondimento\)](#)
    - [GenericServlet class \(approfondimento\)](#)
    - [HttpServlet class](#)
  - Ciclo di vita di una servlet
    - [Life Cycle of a Servlet \(Servlet Life Cycle\)](#)
  - Output sul client
  - Deployment di un'applicazione Web
    - [Steps to create a servlet example](#)
    - [War File](#)
  - Il Context XML descriptor o Deployment descriptor
    - [welcome-file-list in web.xml \(approfondimento\)](#)
    - [load on startup in web.xml \(approfondimento\)](#)
  - Esecuzione di una servlet
    - [How Servlet works?](#)
  - Servlet concorrenti
  - Vantaggi e svantaggi delle servlet
- **4.2 Servlet e database pp.251-260**
  - La connessione ai database
  - La connessione con JDBC Java Database Connectivity
  - Tipi di driver JDBC
  - Utilizzare JDBC standalone
  - Servlet con connessione a MySQL
- **4.3 Servlet con database embedded pp.261-268**
  - I database embedded
  - Uno strumento comune per l'amministrazione dei database
  - Utilizzare Derby
- **Esercizi di Laboratorio pp.274-321**
  - Lab 2 XAMPP e il server Engine Tomcat
  - Lab 3 L'inizializzazione delle servlet

- [ServletConfig Interface](#)
- [ServletContext Interface](#) (*approfondimento*)
- [Attribute in Servlet](#) (*approfondimento*)
- Lab 4 L'interazione get/post tra client e servlet
  - [War File](#)
  - Servlet Collaboration
    - [RequestDispatcher in Servlet](#)
    - [SendRedirect in servlet](#)
  - Session Tracking in Servlets
    - [Session Tracking in Servlets](#)
- Lab 5 L'interazione con client AJAX e lo scambio dati in formato JSON
- Lab 6 La permanenza dei dati con le servlet: i cookie
  - [Cookies in Servlet](#)
  - [Servlet Login and Logout Example using Cookies](#)
  - Hidden Form Field
    - [Hidden Form Field](#)
  - URL Rewriting
    - [URL Rewriting](#)
- Lab 7 La permanenza dei dati con le servlet: le sessioni
  - [HttpSession interface](#)
  - [Servlet HttpSession Login and Logout Example](#)
- Lab 8 JDBC e MySQL
- Lab 9 Servlet e database MDB con parametri
- Lab 10 Servlet con JavaDB Derby

# Web Service REST - Concetti teorici

# Web Service [LIBRO]

**DA VEDERE:** [REST API Design](#)

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 6 - I Web Service e le API di Google

- **6.1 Introduzione ai Web Service: protocolli SOAP e REST pp.368-380**
  - Introduzione
  - Che cos'è un Web Service
  - Un nuovo modello basato su XML: l'architettura SOA
  - Il protocollo SOAP
  - Il protocollo REST
  - Conclusione
- **Esercizi di Laboratorio pp.395-417**
  - Lab 1 Strumenti per lo sviluppo e il testing di Web Service: framework e SDK (Software Development Kit)
  - Lab 2 Realizzazione di un Web Service SOAP in Java
  - Lab 3 Realizzazione di un Web Service REST in Java

## Introduzione ai Web Service

Il Web è nato negli anni '90 del secolo scorso come una piattaforma per la condivisione di documenti distribuiti su diverse macchine tra loro interconnessi. L'evoluzione del Web è avvenuta grazie alla standardizzazione di alcuni concetti che si dimostrarono essere fondamentali:

- il linguaggio **HTML** (*HyperText Markup Language*) che è il linguaggio di markup solitamente usato per la formattazione e impaginazione di documenti ipertestuali disponibili nel World Wide Web sotto forma di pagine web.
- il protocollo **HTTP** (*HyperText Transfer Protocol*) che è un protocollo del livello applicativo usato come principale sistema per la trasmissione di informazioni sul web, in un'architettura client-server.
- ([\*The Real Difference Between a URL and a URI\*](#)) un metodo di indirizzamento delle risorse tramite **URI** (*Uniform Resource Identifier*) composto da una stringa che identifica univocamente una risorsa generica. La risorsa può essere un elemento raggiungibile tramite un indirizzo Web, il nome di un file, un indirizzo di posta elettronica o altro. Un **URI** implementa il concetto di **trasparenza di locazione** già visto durante l'introduzione dei sistemi distribuiti. Un **URI** può essere usato per identificare tipologie diverse di risorse e tra le principali ricordiamo lo **URL** e lo **URN**.

Un **URL** (*Uniform Resource Locator*) è un URI che, oltre a identificare una risorsa, fornisce anche i mezzi per ottenere una sua rappresentazione descrivendo la sua "ubicazione" ("location") in una rete. Per esempio, l'URL <https://www.example.com/index.html> è un URI che identifica una risorsa

(l'homepage `index.html` di `example.com`) e lascia intendere che una rappresentazione di tale risorsa (il codice HTML della versione corrente di tale homepage) è ottenibile via HTTP da un host di rete identificato da `www.example.com`.

Un **URN** (*Uniform Resource Name*) è un URI che identifica una risorsa mediante un "nome" in un particolare dominio di nomi (*namespace*). Un URN può essere usato per identificare una risorsa senza fornire informazioni sulla sua ubicazione o sulla sua rappresentazione. Per esempio l'URN `urn:isbn:0-395-36341-1` è un URI che consente di individuare univocamente un libro mediante il suo identificatore `0-395-36341-1` nel *namespace* dei codici ISBN, ma non suggerisce dove e come possiamo ottenere una copia di tale libro.

L'uso e l'evoluzione di queste tecnologie ha permesso al Web di evolvere nel corso degli anni nel modo di intenderlo e di utilizzarlo. Poco per volta i dati recuperabili dal Web sono passati dal semplice testo a contenuti multimediali, i documenti, inizialmente statici, hanno iniziato ad essere generati dinamicamente, e **il Web**, inizialmente usato solo come contenitore di documenti ipertestuali, è divenuto un contenitore di applicazioni software interoperabili **divenendo a tutti gli effetti una piattaforma applicativa distribuita**.

Questa nuova prospettiva ha dato origine, intorno all'anno 2000, al concetto di **Web Service** che rappresenta un sistema software progettato per supportare una **interazione tra applicazioni, utilizzando le tecnologie e gli standard Web** illustrati precedentemente.

Il meccanismo dei **Web Service consente di far interagire in maniera trasparente applicazioni sviluppate con linguaggi di programmazione diversi, che girano su sistemi operativi eterogenei**.

Questo meccanismo consente di realizzare porzioni di funzionalità in maniera indipendente, su piattaforme potenzialmente incompatibili, facendo interagire i vari pezzi tramite tecnologie Web, creando di fatto un'architettura facilmente componibile.

Un esempio di Web service è il servizio offerto da **GitHub Developer** che permette, ad esempio, di recuperare l'elenco degli utenti con una serie di informazioni aggiuntive per ogni utente. I dati restituiti sono in formato JSON. Il programma usato per richiedere i dati è [RESTED](#), un client REST installabile come estensione del browser.

I **Web service** disponibili nel Web, liberi o a pagamento, sono numerosi; un elenco aggiornato e documentato è mantenuto dal sito Web <https://www.programmableweb.com/> o <https://rapidapi.com/> o <https://apilist.fun/>

# /> RESTED

The screenshot shows a REST client interface with the following details:

- Request:** GET https://api.github.com/users
- Headers:** Accept: application/json
- Response (0.336s) - https://api.github.com/users:**

```
[{"login": "mojombo", "id": 1, "node_id": "MDQ6VXNlcjE=", "avatar_url": "https://avatars0.githubusercontent.com/u/1?v=4", "gravatar_id": "", "url": "https://api.github.com/users/mojombo", "html_url": "https://github.com/mojombo", "followers_url": "https://api.github.com/users/mojombo/followers", "following_url": "https://api.github.com/users/mojombo/following{/other_user}", "gists_url": "https://api.github.com/users/mojombo/gists{/gist_id}", "starred_url": "https://api.github.com/users/mojombo/starred{/owner}{/repo}", "subscriptions_url": "https://api.github.com/users/mojombo/subscriptions", "organizations_url": "https://api.github.com/users/mojombo/orgs",}
```

## Ragioni dell'uso dei Web service

I Web Service hanno avuto un'ampia diffusione per una serie di vantaggi che hanno apportato permettendo di ampliare l'interoperabilità di sistemi remoti, l'integrazione di vecchi sistemi e la possibilità di esporre sul Web dati di interesse pubblico.

I Web service permettono di **integrare** il vecchio software di un'azienda a quello nuovo perché implementano le funzionalità che permettono l'interfacciamento con i nuovi software. Questo utilizzo dei Web service consente l'**integrazione** dei sistemi *legacy*, vecchi programmi che non è possibile ancora sostituire ma che devono essere integrati con le nuove applicazioni effettuando uno scambio nettamente isolato dei dati.

Inoltre i Web service permettono di esporre sul Web il *Business logic layer* di molti sistemi usando API specifiche, fornendo alle applicazioni client remote la possibilità di scegliere il Web service di cui necessitano. In questo modo, sul lato client potranno essere aggiunti i servizi che si vogliono, sviluppati utilizzando i linguaggi e i tool preferiti, implementando il concetto di **usabilità**.

I Web service offrono anche una soluzione non proprietaria per la risoluzione di problemi, proprio perché sono spesso offerti al di fuori di una rete privata. Consentono agli sviluppatori di usare il linguaggio di programmazione che preferiscono e sono virtualmente

indipendenti dalla piattaforma proprio per l'uso di metodi di comunicazione basati su standard riconosciuti, implementando il concetto di **interoperabilità**.

Ogni servizio offerto da un Web service esiste indipendentemente dagli altri servizi che insieme costituiscono l'applicazione. Ciò permette di modificare parti dell'applicazione senza impattare in aree che non sono correlate con la parte modificata (**loosely coupled**).

I Web service infine sono messi in opera utilizzando delle tecnologie standard su Internet. Questo permette di effettuare il **deploy** dei Web service anche remotamente su Internet, e di utilizzare sistemi di sicurezza integrati e comunemente accettati.

## REST e SOAP

Attualmente esistono due diversi **approcci alla creazione di Web Service**.

La tecnica iniziale che fu adottata si basava, e si basa ancora oggi, sul **protocollo standard SOAP** (*Simple Object Access Protocol*), **incentrato sullo scambio di messaggi per l'invocazione di servizi remoti, con lo scopo di riprodurre in ambito Web un approccio a chiamate remote di metodi o procedure**, (**RPC** - *Remote Procedure Call*) tipico di protocolli di interoperabilità come **CORBA**, **DCOM** e **RMI**. Le regole di interazione tra client e server sono descritte in un documento apposito (*WSDL - Web Service Description Language*) e per questa ragione risultano difficili da manutenere.

A partire dagli anni 2000 è stata sviluppata una tecnica più flessibile identificata con il nome **REST** (*REpresentational State Transfer*), ispirata ai **principi architetturali tipici del Web**. Questa architettura si concentra sulla descrizione delle risorse e sul modo di individuarle nel Web e trasferirle da una macchina all'altra.

L'architettura **REST** è una **architettura client-server stateless nella quale i Web service sono visti come delle risorse identificate dal loro URI**, usato dal client per accedere alla risorsa. I Web service REST supportano molteplici formati per lo scambio dei dati (XML, Json, testo, ecc.) e sono facilmente manutenibili.

Anche se l'obiettivo dei due approcci è pressoché identico ed entrambi adottano il Web come piattaforma di elaborazione, le visioni sono totalmente differenti.

La prima evidente differenza tra i due tipi di Web Service è la **visione del Web** proposta **come piattaforma di elaborazione**. **REST** propone una visione del **Web** incentrata sul concetto di **risorsa** mentre **SOAP** mette in risalto il concetto di **servizio**.

- Un **Web Service REST** è custode di un insieme di risorse sulle quali un client può chiedere le operazioni canoniche usando il protocollo HTTP.
- Un **Web Service** basato su **SOAP** espone un insieme di metodi richiamabili da remoto da parte di un client.

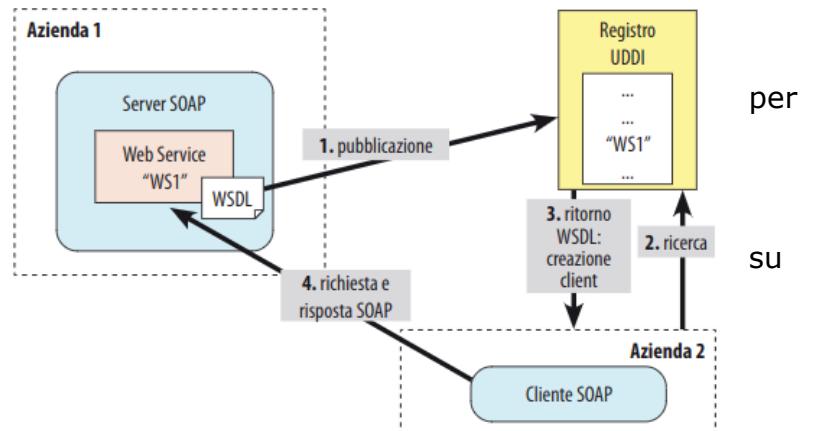
Il protocollo **SOAP** (*Simple Object Access Protocol*) **utilizza HTTP come protocollo per il trasporto dei propri comandi**, ma non è limitato né vincolato ad esso, dal momento che **può benissimo usare altri protocolli per il trasporto dei suoi comandi**. A differenza di HTTP, le specifiche di SOAP non affrontano argomenti come la sicurezza o l'indirizzamento, per i quali sono stati definiti standard a parte. Quindi **SOAP non sfrutta completamente il protocollo HTTP, utilizzandolo come semplice protocollo per il trasporto dei propri comandi**.

**REST** invece **sfrutta HTTP come un protocollo di livello applicativo**, e ne **utilizza a pieno le sue potenzialità**.

È evidente che l'**approccio** adottato dai **Web Service** basati su **SOAP** è derivato dalle tecnologie di interoperabilità esistenti al di fuori del Web, basandosi essenzialmente su chiamate di procedura remota. In sostanza questo approccio può essere visto come una sorta di adattamento di queste tecnologie al Web.

**L'approccio REST**, invece, **tende a conservare e ad esaltare le caratteristiche intrinseche del Web evidenziando la predisposizione ad essere una piattaforma per l'elaborazione distribuita**. Quindi, non è necessario aggiungere nulla a quanto è già esistente sul Web per consentire ad applicazioni remote di interagire.

Inoltre i Web Service basati su **SOAP** prevedono lo standard **WSDL** (*Web Service Description Language*) definire l'interfaccia di un servizio. Questa è un'ulteriore evidenza del tentativo di adattare al Web l'approccio di interoperabilità basato su chiamate remote. Infatti il **WSDL** non è altro che un **IDL** (*Interface Description Language*) per un componente software. Da un lato l'esistenza di **WSDL** favorisce l'uso di tool per creare automaticamente client in un determinato linguaggio di programmazione, ma allo stesso tempo **induce a creare una forte dipendenza tra client e server**.



**REST non prevede esplicitamente nessuna modalità per descrivere come interagire con una risorsa**, in quanto per questa architettura **le operazioni sono implicite nel protocollo HTTP**. I **Web service REST** presuppongono una **visione molto dinamica e un debole accoppiamento tra client e server**, differenziandosi completamente dalla visione SOAP. Questo è possibile grazie ad uno dei principi cardine dei **Web service REST**, il **principio HATEOAS** (*Hypermedia As The Engine Of Application State*), che interagisce con un'applicazione web esclusivamente attraverso gli ipermedia forniti dinamicamente dai server dell'applicazione stessa. In sostanza un client REST non necessita di nessuna conoscenza a priori per interagire con una particolare applicazione o server che vada oltre la normale comprensione degli ipermedia. Al contrario, nello standard SOAP i client e i server interagiscono tra loro tramite un'interfaccia specifica stabilita tramite documentazione o tramite un linguaggio di descrizione di interfaccia (WSDL). **Il vincolo HATEOAS disaccoppia client e server in modo tale da permettere alle funzionalità server di evolvere indipendentemente**.

In conclusione, mentre i Web service basati su SOAP costruiscono una infrastruttura prolissa e complessa al di sopra del Web per fare cose che il Web è già in grado di fare, i Web service REST invece usano il Web come architettura per la programmazione distribuita, senza aggiungere sovrastrutture non necessarie.

I Web service SOAP consentono però di definire uno standard indipendente dal Web, permettendo all'infrastruttura di basarsi anche su protocolli diversi da HTTP.

## CLIL Listening and Speaking - What REST is

This video gives a brief overview of what REST web services are. Watch the video and discuss the topics with your classmate.

1. [What are RESTful Services \(RESTful APIs\)? | Mosh](#)

## I principi dell'architettura RESTful

**REST** definisce un **insieme di principi architetturali per la progettazione di un sistema** in quanto è uno **stile architettonico** che specifica come dovrebbe essere scritto un software in un sistema distribuito che risponda a determinati criteri e principi di funzionamento. **REST** non fa riferimento ad un sistema concreto e ben definito, e **non è uno standard** stabilito da un organismo di standardizzazione.

La sua definizione è apparsa per la prima volta nel 2000 nella tesi di **Roy Fielding**, [Architectural Styles and the Design of Network-based Software Architectures](#), discussa presso l'Università della California, ad Irvine. In questa tesi venivano analizzati alcuni principi alla base di diverse architetture software, tra cui appunto i principi di un'architettura software che consentisse di vedere il Web come una piattaforma per l'elaborazione distribuita.

È bene precisare che **i principi REST non sono necessariamente legati al Web**, nel senso che si tratta di principi astratti di cui il **World Wide Web** ne risulta essere un esempio concreto.

All'epoca questa visione del Web non fu adeguatamente considerata, ma negli ultimi anni l'approccio **REST** è venuto alla ribalta come metodo per la realizzazione di **Web Service** altamente efficienti e scalabili ed ha al suo attivo un significativo numero di applicazioni.

La ragione per questa inversione di tendenza deriva dal fatto che **il Web ha tutto quello che serve per essere considerata una piattaforma di elaborazione distribuita secondo i principi REST**, e non sono necessarie altre sovrastrutture per realizzare quello che è il Web programmabile, idea palasemente in conflitto con i **Web Service** basati su **SOAP**.

I **principi** che rendono il Web adatto a realizzare **Web Service** secondo l'approccio **REST** possono essere riassunti nei seguenti cinque punti:

1. identificazione delle risorse
2. utilizzo esplicito dei metodi HTTP
3. risorse autodescrittive
4. collegamenti tra risorse
5. comunicazione senza stato

Questi principi rappresentano in realtà concetti ben noti perché ormai insiti nel Web che conosciamo. Li analizzeremo tuttavia sotto una prospettiva diversa, quella della realizzazione di **Web Service**.

Si osservi che è comunque possibile realizzare Web service utilizzando il protocollo HTTP senza rispettare tutti i principi esposti in precedenza. I **Web service che però implementano l'architettura REST in modo integrale sono definiti RESTful**.

## CLIL Listening and Writing - Introduction to Web Services REST

Watch the video [REST Web Services 01 - Introduction](#) and answer the following questions. Send by email your answers to your teacher.

1. Explain what a Web Service is.
2. Make a list of the different formats used by a REST Web Service in order to return a result.
3. Specify which application protocol is used by REST Web Services in order to exchange data.
4. Explain how Web Services communication appen.
5. Explain why a Web Service does not need a service definition.

## Identificazione delle risorse

Le **risorse** sono gli **elementi fondamentali** su cui si basano i **Web Service REST**, in netta contrapposizione ai *Web Service SOAP-oriented* che sono basati sul concetto di *chiamata remota*.

Per **risorsa** si intende **un qualsiasi elemento oggetto di elaborazione**. Per fare qualche esempio concreto, una risorsa può essere un cliente, un libro, un articolo, un qualsiasi oggetto su cui è possibile effettuare operazioni. Per fare un parallelo con la programmazione ad oggetti possiamo dire che una risorsa può essere assimilata ad una istanza di una classe.

Il principio che stiamo analizzando stabilisce che **ciascuna risorsa deve essere identificata univocamente**. Dato che le interazioni avvengono in ambito Web, il **meccanismo più naturale per individuare una risorsa è dato dal concetto di URI**.

Il principale beneficio nell'adottare lo schema **URI** per identificare le risorse consiste nel fatto che esiste già, è ben definito e collaudato e non occorre pertanto inventarne uno nuovo.

I seguenti sono esempi di possibili identificatori di risorse:

`http://www.myapp.com/clienti/1234`  
`http://www.myapp.com/ordini/2016/98765`  
`http://www.myapp.com/prodotti/7654`  
`http://www.myapp.com/ordini/2016`  
`http://www.myapp.com/prodotti/rosso`

Gli **URI** sono abbastanza **autoesplicativi**: il primo identifica un determinato cliente, il secondo un ordine del 2016 e il terzo un prodotto. Il quarto URI identifica l'insieme degli ordini del 2016, mentre l'ultimo URI identifica l'insieme dei prodotti di colore rosso.

L'interpretazione che abbiamo dato a questi URI è però desunta dalla semantica delle parole contenute nelle sue parti. Dal punto di vista di un **Web service** un **URI** è soltanto una **stringa che identifica una risorsa**, per cui anche <http://www.myapp.com/tgw34/2099ww> può essere un URI valido per identificare un cliente.

## CLIL Listening and Writing - REST and HTTP

Watch the video [REST Web Services 02 - REST and HTTP](#) and answer the following questions. Send by email your answers to your teacher.

6. Describe the main purpose of HTTP.
7. Explain which Hypertext characteristic is.
8. Explain what is identified by an URI in a RESTful architecture.
9. Describe how HTTP methods are used by a RESTful API.
10. Explain how status codes are used in RESTful architecture.
11. Specify which header is used to specify the format of data.

## CLIL Listening and Writing - Resource URIs

Watch the video [REST Web Services 03 - Resource URIs](#) and answer the following questions. Send by email your answers to your teacher.

12. Specify which the first thing to design a RESTful API is.
13. Make an example of a RESTful URI to identify a resource instance.
14. Specify which is a correct identifier in RESTful URI among nouns and verbs.
15. Explain why plural is used in RESTful URI.
16. Explain what "resource relations" represents and give a simple example to manage it.

## CLIL Listening and Writing - Collection URIs

Watch the video [REST Web Services 04 - Collection URIs](#) and answer to the following questions. Send by email your answers to your teacher.

17. Make an example of a RESTful URI to identify a collection of resources.
18. Explain how create a query parameters URI.

## Utilizzo esplicito dei metodi HTTP

Una volta spiegato come individuare una risorsa abbiamo bisogno di un **meccanismo per indicare quali operazioni effettuare** su di esse. Il principio dell'**uso esplicito dei metodi HTTP** ci indica di usare i metodi predefiniti di questo protocollo, e cioè **GET, POST, PUT e DELETE**.

Facciamo un semplice esempio per provare a chiarire questo concetto. 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. Il metodo che implicitamente stiamo eseguendo è GET, il cui effetto è l'accesso ad una rappresentazione della risorsa identificata dall'URI.

Dal punto di vista del codice, non si avrà bisogno di un metodo del tipo `getCliente(1234)` per ottenere la rappresentazione del cliente con codice 1234, sarà infatti sufficiente sfruttare il metodo standard **GET** del protocollo HTTP sull'URI che identifica quel determinato cliente.

Questo **rende uniforme l'invocazione di operazioni sulle risorse**, cioè **il client non ha bisogno di sapere qual è la specifica interfaccia da utilizzare per invocare il metodo che consente di ottenere la rappresentazione di una risorsa**. In un contesto non RESTful potremmo avere metodi come `getCliente()` o `getCustomer()` o altra specifica dipendente dalle scelte di chi ha sviluppato il Web Service, ma in ambito **REST** per ottenere lo stato della risorsa basterà usare il metodo **GET** applicato all'**URI** che identifica la risorsa stessa.

Quindi **in un contesto RESTful l'accesso alla risorsa viene effettuata direttamente con i metodi HTTP**, mappando uno a uno le tipiche operazioni **CRUD** e i **metodi HTTP**:

Metodo HTTP	Operazione CRUD	Descrizione
POST	Create	Crea una nuova risorsa
GET	Read	Ottiene lo stato di una risorsa esistente
PUT	Update	Aggiorna una risorsa o ne modifica lo stato
DELETE	Delete	Elimina una risorsa

È opportuno notare che questo principio è in contrasto con quella che è la tendenza generale nell'utilizzo dei metodi HTTP. Infatti, molto spesso viene utilizzato il metodo GET per eseguire qualsiasi tipo di interazione con il server. Ad esempio, spesso per l'inserimento di un nuovo cliente nell'ambito di un'applicazione Web viene eseguita una richiesta di tipo GET su un URI del seguente tipo:

```
GET http://www.myapp.com/api/addCustomer?name=Rossi
```

**Questo approccio non è conforme ai principi REST**, perché secondo questo stile architettonico il metodo **GET serve per accedere alla rappresentazione di una risorsa, e non per crearne una nuova**. Nella pratica questo uso del metodo GET introduce un effetto collaterale che può avere delle conseguenze indesiderate se, ad esempio, lo URI viene invocato più volte o se la risorsa viene memorizzata in uno dei vari livelli di cache esistenti tra client e server.

Come principio generale, nel progettare un **Web service** in modalità **REST** è utile evitare l'uso di verbi negli URI e limitarsi ad utilizzare nomi, ricordandosi che **un URI identifica una risorsa o una collezione di risorse**.

Allo stesso modo è opportuno sottolineare che il corpo di una richiesta HTTP con metodi **PUT** e **POST** è pensato per il **trasferimento della rappresentazione di una risorsa** e non per eseguire chiamate remote o altre attività simili.

Per comprendere il principio dell'architettura **RESTful** per l'accesso alle risorse, si consideri la gestione di una rubrica telefonica. Le operazioni **CRUD** effettuabili sulla rubrica potrebbero essere semplificate nel modo seguente.

Creazione (**CREATE**) di una nuova risorsa *Rossi* nella rubrica telefonica:

*In XML*

```
POST http://rubrica.it/api/contatti
<VoceRubrica>
  <nominativo>Rossi</nominativo>
  <telefono>444-555</telefono>
</VoceRubrica>
```

*In JSON*

```
POST http://rubrica.it/api/contatti
{
  "VoceRubrica": {
    "nominativo": "Rossi",
    "telefono": "444-555"
  }
}
```

e in questo caso il server invierà come risposta al client l'identificativo della nuova voce in rubrica.

Acquisizione (**READ**) della rappresentazione di una risorsa *Rossi* dalla rubrica telefonica:

```
GET http://rubrica.it/api/contatti/Rossi
```

che restituirà:

*In XML*

```
<VoceRubrica>
  <nominativo>Rossi</nominativo>
  <telefono>444-555</telefono>
</VoceRubrica>
```

*In JSON*

```
{
  "VoceRubrica": {
    "nominativo": "Rossi",
    "telefono": "444-555"
  }
}
```

Modifica (**UPDATE**) di una risorsa *Rossi* nella rubrica telefonica:

*In XML*

```
PUT http://rubrica.it/api/contatti/Rossi
<VoceRubrica>
  <nominativo>Rossi</nominativo>
```

```
<telefono>444-333 </telefono>
</VoceRubrica>
```

In JSON

```
PUT http://rubrica.it/api/contatti/Rossi
{
  "VoceRubrica": {
    "nominativo": "Rossi",
    "telefono": "444-333"
  }
}
```

Cancellazione (**DELETE**) di una risorsa *Rossi* dalla rubrica telefonica:

```
DELETE http://rubrica.it/api/contatti/Rossi
```

## CLIL Listening and Writing - HTTP Methods

Watch the video [REST Web Services 05 - HTTP Methods](#) and answer the following questions. Send by email your answers to your teacher.

19. Give an example of an action based URI and an example of a resource based URI.
20. Describe how it is possible to use the same URI for each CRUD operation in REST Web services.
21. Give an example for each CRUD operation using the correct HTTP method and an URI to identify a single resource.
22. Give an example for each CRUD operation using the correct HTTP method and an URI to identify a collection of resources.

## Idempotenza dei metodi HTTP

La ragione per cui vengono utilizzati due metodi distinti per effettuare la modifica (**PUT**) e l'aggiunta (**POST**) di una risorsa, risiede nel concetto di **idempotenza** dei metodi HTTP.

Generalmente è possibile **classificare i metodi HTTP** utilizzati per effettuare **operazioni CRUD** in due gruppi distinti:

1. **metodi di sola lettura:** GET
2. **metodi di scrittura:** POST, PUT e DELETE

Alla luce di questa prima classificazione si può dire che il **metodo GET**, essendo un **metodo di sola lettura**, può essere ritenuto **sicuro**; è possibile effettuare più volte la stessa richiesta GET sul server senza nessun effetto collaterale, in quanto nulla verrà modificato sul server. Detto in altre parole, **il metodo GET è un metodo idempotente**, e quindi **ripetibile**.

L'**idempotenza** risiede esattamente su questo concetto: **un metodo è ritenuto idempotente se è possibile ripetere più volte la stessa richiesta al server senza che vi siano effetti collaterali, tranne eventualmente la prima volta che è stato invocato il metodo.**

Dalla definizione di **idempotenza** segue che anche **il metodo PUT è idempotente** in quanto, dopo il primo aggiornamento della risorsa, la ripetizione del medesimo

aggiornamento non avrà alcun effetto collaterale; eventualmente verrà riscritto più volte il medesimo valore.

Analogamente, il **metodo DELETE risulta essere idempotente** in quanto, dopo aver cancellato la risorsa identificata da un particolare URI, univoco per ogni risorsa, la ripetizione della medesima cancellazione, usando sempre lo stesso URI, non avrà alcun effetto collaterale.

Invece il **metodo POST non è idempotente** in quanto la ripetizione della creazione di una risorsa, anche se identica nei contenuti, provocherà l'aggiunta successiva di risorse identificate da URI diversi.

Seguendo il concetto di **idempotenza** è quindi possibile **riclassificare i metodi HTTP** utilizzati per effettuare **operazioni CRUD** nei seguenti due gruppi:

1. **idempotenti** (ripetibili in modo sicuro): GET, PUT e DELETE
2. **non idempotenti** (non ripetibili in modo sicuro): POST

**L'architettura REST richiede esplicitamente che i metodi GET, PUT e DELETE siano idempotenti, mentre il metodo POST deve essere non idempotente.**

Il fatto che il metodo GET sia idempotente permette di poter memorizzare nella cache una risposta da parte del server originata da una richiesta GET, e il reinvio ripetuto tramite il refresh del browser non deve avere effetti collaterali. Con il metodo POST, che non è idempotente, si dovrebbe evitare di memorizzare in cache la risposta del server, in quanto il reinvio ripetuto tramite refresh del browser provocherebbe la successiva creazione di nuove risorse sul server.

## CLIL Listening and Writing - Method Idempotence

Watch the video [REST Web Services 06 - Method Idempotence](#) and answer to the following questions. Send by email your answers to your teacher.

23. Specify which HTTP methods are read-only and which ones are write methods.
24. Specify which HTTP methods are safely repeatable and which ones cannot be repeated safely.
25. Given the previous answers, specify which HTTP methods are safely cacheable and which ones are not safely cacheable.

## Risorse autodescrittive

Il concetto di **risorse autodescrittive** prevede che queste siano **concettualmente separate dalle rappresentazioni restituite al client**. Ad esempio, un Web Service non invia al client direttamente un record del suo database, ma una sua rappresentazione in una codifica dipendente dalla richiesta del client e/o dall'implementazione del servizio, ad esempio **in XML o in JSON**.

**I principi REST non pongono nessun vincolo sulle modalità di rappresentazione di una risorsa.** Virtualmente è possibile utilizzare il formato che si preferisce senza essere obbligati a seguire uno standard. Di fatto, però, è opportuno utilizzare formati il più possibile standard in modo da semplificare l'interazione con i client. Inoltre, sarebbe opportuno prevedere rappresentazioni multiple di una risorsa, per soddisfare client di tipo diverso, ad esempio sia in XML, sia in JSON.

**Il tipo di rappresentazione** inviata dal Web service al client è **indicato nella stessa risposta HTTP tramite un tipo MIME**, così come avviene nella classica comunicazione tra Web server e browser, utilizzando lo *header* HTTP *Content-type*.

Un client a sua volta ha la possibilità di richiedere una risorsa in uno specifico formato sfruttando l'attributo *Accept* di una richiesta HTTP, come mostrato nel seguente esempio:

```
GET /api/clienti/1234
HTTP/1.1
Host: www.myapp.com
Accept: text/xml
```

In questo caso il client richiede la rappresentazione del cliente identificato dal codice 1234 in **formato XML**. Ovviamente, se il Web service supportasse ulteriori formati, la stessa risorsa potrebbe essere richiesta in formati diversi, come ad esempio in JSON o in HTML.

La possibilità di rappresentazioni multiple produce alcuni benefici pratici: ad esempio, se abbiamo un output sia in HTML, sia in XML, possiamo consumare il servizio sia con un'applicazione sia con un comune browser. In altre parole, **seguendo i principi REST** nel progettare un'applicazione Web è possibile **costruire sia una Web API che una Web UI**.

## CLIL Listening and Writing - REST Response

Watch the video [REST Web Services 07 - REST Response](#) and answer the following questions. Send by email your answers to your teacher.

26. Explain the reason for which REST Web services use different representations of resources.
27. Describe for which purpose is used the HTTP header Content Type in a HTTP response.
28. Describe the purpose of a status code in a HTTP response.
29. List all the status code groups giving a brief explanation for each of them.
30. Give an example for each CRUD operations of a suitable status code and explaining string sent by the server in its response.

## Collegamenti tra risorse e il principio HATEOAS

Un altro **vincolo dei principi REST** consiste nella necessità che le **risorse** siano tra loro **messe in relazione tramite link ipertestuali**. Questo principio è anche noto come **HATEOAS**, dall'acronimo di **Hypermedia As The Engine Of Application State**, e pone l'accento sulle modalità di **gestione dello stato dell'applicazione**.

In sostanza questo principio afferma che **tutto quello che un client deve sapere su una risorsa, e sulle risorse ad essa correlate, deve essere contenuto nella rappresentazione della risorsa stessa o deve essere accessibile tramite collegamenti ipertestuali**.

Ad esempio, la rappresentazione di un ordine in un linguaggio XML-based deve contenere gli eventuali collegamenti al cliente e agli articoli correlati:

In XML

```
<ordine>
  <numero>12345678</numero>
  <data>01/07/2011</data>
  <cliente rif="http://www.myapp.com/api/clienti/1234" />
  <articoli>
    <articolo rif="http://www.myapp.com/api/prodotti/98765" />
    <articolo rif="http://www.myapp.com/api/prodotti/43210" />
  </articoli>
</ordine>
```

In questo modo il client può accedere alle risorse correlate seguendo semplicemente i collegamenti contenuti nella rappresentazione XML della risorsa corrente.

**Il fatto di utilizzare un URI come identificatore di una risorsa**, quindi un meccanismo standard e consolidato, **consente al client di accedere anche a risorse messe a disposizione da altre applicazioni**, che potrebbero trovarsi eventualmente su altri server.

L'architettura REST (*REpresentational State Transfer*) enfatizza proprio il concetto di **trasferimento di stato**, cioè il fatto che un'applicazione possa passare velocemente da uno stato all'altro. Nell'**architettura REST lo stato di una applicazione è rappresentato dallo stato delle sue risorse** e, grazie al principio **HATEOAS**, la relativa **transazione di stato delle risorse** viene **attivata attraverso** l'uso di semplici **collegamenti ipermediali**.

Nella visione REST l'esecuzione di un'applicazione può quindi essere rappresentata da una rete di risorse fra cui un client "naviga" seguendo i collegamenti ipermediali ammessi tra una risorsa e l'altra. Questa visione richiama esattamente la navigazione tra pagine Web diverse, visitabili tramite link.

Il principio **HATEOAS** cerca di incoraggiare l'uso di **collegamenti ipermediali** sia per **rappresentare risorse composte**, sia per **definire qualsiasi altra relazione tra le risorse** e per **controllare le transizioni ammissibili** tra uno stato e l'altro dell'applicazione.

Per fare un esempio, consideriamo la seguente rappresentazione di un ordine in un ipotetico negozio online :

In XML

```
<ordine id="123" data="10/03/2017">
  <cliente>Mario Rossi</cliente>
  <stato>EVASO</stato>
  <dettagli>
    <dettaglio>
      <articolo>Libro</articolo>
      <quantita>1</quantita>
    </dettaglio>
    <dettaglio>
      <articolo>CD</articolo>
```

```
<quantita>5</quantita>
</dettaglio>
</dettagli>
</ordine>
```

In esso non è riportato alcun collegamento con altre risorse gestite dal sistema. Affinché un cliente possa risalire alla fattura associata a questo ordine dovrà effettuare un'apposita richiesta al Web Service, utilizzando l'URI della risorsa *fattura* che il cliente dovrebbe procurarsi in qualche modo. Nell'ottica del principio HATEOAS sarebbe opportuno includere un collegamento direttamente nella rappresentazione dell'ordine:

*In XML*

```
<ordine id="123" data="10/03/2017">
  <cliente>Mario Rossi</cliente>
  <stato>EVASO</stato>
  <dettagli>
    <dettaglio>
      <articolo>Libro</articolo>
      <quantita>1</quantita>
    </dettaglio>
    <dettaglio>
      <articolo>CD</articolo>
      <quantita>5</quantita>
    </dettaglio>
  </dettagli>
  <links>
    <link rel="self" mediaType="application/xml"
          href="http://www.mionegozio.com/api/ordini/123" />
    <link rel="fattura" mediaType="application/xml"
          href="http://www.mionegozio.com/api/fatture/789" />
  </links>
</ordine>
```

In questo modo il client ha già nella rappresentazione dell'ordine tutte le informazioni necessarie per accedere sia all'ordine (**self**), sia alla fattura associata. Inoltre, la presenza o meno del collegamento alla fattura associata all'ordine fornisce implicitamente un'altra informazione al client: se il collegamento non è presente nella rappresentazione dell'ordine vuol dire che la fattura non è ancora stata emessa, se invece è presente vuol dire che è stata emessa ed è accessibile. In questo modo si avrà un maggior controllo della transizione da uno stato all'altro dell'applicazione, evitando transizioni non ammissibili in un dato momento.

**Sfruttando pienamente il principio HATEOAS è possibile creare servizi Web con accoppiamento debole tra client e server.** Infatti, se il server riorganizza le relazioni tra le risorse, il client è in grado di trovare tutto ciò che serve nelle rappresentazioni ricevute. Potenzialmente tutto quello che servirebbe ad un client è solo l'URI della risorsa iniziale, ad esempio quello dell'ordine. Come avanzare tra uno stato e l'altro dell'applicazione verrà indicato man mano che si seguono i collegamenti incorporati nelle successive rappresentazioni delle risorse.

**Il principio HATEOAS**, che sancisce il legame tra transizioni di stato e collegamenti tra risorse, è purtroppo **scarsamente sfruttato nelle attuali implementazioni di Web service di tipo REST**. La maggior parte di queste implementazioni si limita a definire le rappresentazioni delle risorse e le modalità di interazione tramite i metodi HTTP, non sfruttando a pieno le potenzialità dei principi REST.

## CLIL Listening and Writing - HATEOAS

Watch the video [REST Web Services 08 - HATEOAS](#) and answer the following questions. Send by email your answers to your teacher.

31. From the client point of view, explain which is the purpose of using an URI inside an XML file received from a server.
32. Specify which kind of element can be used in a XML file to identify a resource URI.
33. Explain which is the purpose of the href and rel attributes.
34. Suppose it is necessary to create an XML file to describe the books in a library. For each book it is important to store in the XML file the library identification, the title, the author's name, the publisher and the year when the book was published. Create an example of the XML file using the HATEOAS principle.

## Comunicazione senza stato

Il principio della **comunicazione stateless** è ben noto a chi lavora con il Web. Questa è infatti una delle caratteristiche principali del protocollo HTTP, cioè **ciascuna richiesta non ha alcuna relazione con le richieste precedenti e successive**.

Lo stesso principio si applica ad un **Web service REST**, cioè **le interazioni tra client e server devono essere senza stato**, rendendo ciascuna richiesta indipendente dalle altre.

**La comunicazione stateless prevede che una richiesta effettuata dal client non richieda al server il recupero dello stato dell'applicazione o di un suo contesto.** Il Web service o il client REST devono includere nelle intestazioni o nel corpo HTTP tutti i parametri, il contesto e i dati necessari al server per generare una risposta.

È importante sottolineare che sebbene **REST preveda la comunicazione stateless, non vuol dire che un'applicazione non debba avere uno stato**. Ciò che si richiede è che **la responsabilità della gestione dello stato dell'applicazione non debba essere conferita al server, ma deve rientrare nei compiti del client**.

**La principale ragione di questa scelta è la scalabilità:** mantenere lo stato di una sessione ha un costo in termini di risorse sul server e, all'aumentare del numero di client, tale costo può diventare insostenibile.

Inoltre, **con una comunicazione senza stato è possibile creare cluster di server** che possono rispondere ai client senza vincoli sulla sessione corrente, ottimizzando le prestazioni globali dell'applicazione.

Infine **una comunicazione senza stato consente di ottimizzare le prestazioni del Web service e di semplificare la progettazione e l'implementazione lato server**, dato che l'assenza della gestione dello stato rimuove la necessità di sincronizzare i dati di sessione con una applicazione esterna.

Il protocollo HTTP è intrinsecamente senza stato, ma molto spesso, nello sviluppo di applicazioni Web, vengono artificialmente ricreate le condizioni per una comunicazione con stato. Ad esempio, l'uso di **chiavi di sessione**, utilizzate in diversi framework di sviluppo per il Web, non fa altro che introdurre lo stato della comunicazione al di sopra del protocollo HTTP, in genere sfruttando i **cookie**.

In una **architettura REST** l'uso di comunicazioni con stato sarebbe vietato; il server non dovrebbe tenere traccia delle relazioni tra le diverse richieste. Se fosse necessario gestire uno stato della comunicazione, questo compito spetterebbe completamente al client.

## Stato delle risorse e dell'applicazione

Il fatto che i principi REST escludano la gestione dello stato della comunicazione non deve però far pensare che i Web Service REST siano senza stato. L'acronimo **REST** sta per *REpresentational State Transfer*, sottolineando proprio la **centralità della gestione dello stato in un sistema distribuito**. Lo **stato che REST prende in considerazione** è però quello **delle risorse e dell'intera applicazione**, come enfatizzato in precedenza relativamente al **principio HATEOAS**.

Lo **stato delle risorse** è dato dall'insieme dei valori che caratterizzano una risorsa in un dato momento. Un Web Service è responsabile della gestione dello stato delle risorse. Un client può accedere allo stato di una risorsa tramite le diverse rappresentazioni della risorsa stessa utilizzando il metodo GET di HTTP, e contribuire a modificarlo per mezzo dei metodi PUT, POST e DELETE di HTTP.

Lo **stato del client** è rappresentato dall'insieme del suo contesto e delle risorse ottenute in uno specifico momento. Il server può influenzare le transizioni di stato del client inviando differenti rappresentazioni delle risorse in risposta alle sue richieste.

Lo **stato dell'applicazione**, cioè del risultato dell'interazione tra client e server, è dato dallo stato del client e delle risorse gestite dal server, e determina le modalità di modifica dello stato delle risorse e del client. A differenza di quanto avviene in buona parte delle applicazioni Web, dove lo stato dell'applicazione viene spesso mantenuto dal server insieme allo stato della comunicazione, lo **stato dell'applicazione in un'architettura REST è il frutto della collaborazione di client e server**, ciascuno con i propri ruoli e responsabilità. Nella progettazione di un Web Service REST questa separazione dei ruoli deve essere chiara e spesso occorre ripensare opportunamente la gestione dello stato in termini di gestione di risorse.

Ad esempio, per gestire un *carrello* in una comune applicazione Web di e-commerce, si ricorre tipicamente ad una sua rappresentazione lato server associata alla sessione corrente dell'utente. Per quanto abbiamo detto finora, questo approccio viola i principi REST dal momento che richiede il mantenimento dello stato della sessione. Per rimanere nell'ambito dei principi REST abbiamo a disposizione due possibili soluzioni: demandare al client il compito di gestire il carrello, o gestire il carrello sul server come una risorsa.

Nel primo caso il client avrà una struttura dati in cui terrà traccia degli articoli a cui l'utente è interessato. Nel momento della generazione della richiesta di un nuovo ordine da inviare al server, gli articoli annotati nel carrello verranno riportati nell'ordine.

In alternativa è possibile che il nostro Web Service preveda una **risorsa carrello** dedicata al mantenimento degli articoli scelti dall'utente durante la fase di acquisto. È da sottolineare il fatto che il carrello è una risorsa come le altre e non è associata alla sessione corrente. È quindi accessibile tramite URI in qualsiasi momento, è persistente ed è gestibile tramite i metodi HTTP.

La trasformazione in risorse di elementi tradizionalmente gestiti tramite lo stato della sessione, è un approccio da tenere presente durante la progettazione di un Web Service REST.

## CLIL Listening - The Richardson Maturity Model

Watch the video [REST Web Services 09 - The Richardson Maturity Model](#) to complete your theoretical preparation about Web Service RESTful.

# REST e sicurezza HTTP (cenni)

Uno degli aspetti cruciali di qualsiasi applicazione Web è la **gestione della sicurezza**. I Web service REST, rendendo accessibili le risorse tramite i meccanismi del Web, non possono sottrarsi a questa necessità.

In linea di massima la sicurezza di un sistema coinvolge almeno i seguenti aspetti:

- l'**autenticazione**, cioè la capacità di identificare le parti coinvolte in una comunicazione
- l'**autorizzazione**, cioè la concessione dei diritti di accesso ad una risorsa in base all'identità
- la **fiducia**, cioè la capacità di confidare nel risultato di un'operazione, come ad esempio l'autenticazione o l'autorizzazione, eseguita da terze parti
- la **riservatezza**, cioè la capacità di mantenere l'informazione privata durante la trasmissione o la memorizzazione
- l'**integrità**, cioè la capacità di prevenire che l'informazione possa essere modificata da terzi

Nel Web tutti questi aspetti sono stati efficacemente affrontati da diverso tempo, rendendo il modello sufficientemente maturo per decidere di **gestire la sicurezza dei Web service REST basandosi largamente sul protocollo HTTP**, ereditando le caratteristiche ormai ampiamente collaudate.

In sostanza, per gestire l'identità nell'ambito di un Web Service REST possono essere utilizzati i meccanismi propri del protocollo HTTP, come ad esempio [\*\*HTTP Basic Authentication\*\*](#) o [\*\*HTTP Digest Authentication\*\*](#).

Ad esempio, se un client richiede l'accesso ad una risorsa protetta, il server può richiedere l'autenticazione con una risposta HTTP analoga alla seguente:

```
401 Unauthorized
WWW-Authenticate:
Basic realm="risorsaRiservata"
```

La risposta richiede di autenticarsi utilizzando [\*\*HTTP Basic Authentication\*\*](#). Il client invierà una risposta di questo tipo:

```
GET /ordini/?123 HTTP/1.1
Host: www.mionegozio.com
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

dove le **credenziali di autenticazione vengono passate sotto forma di stringa codificata in [Base64](#)**. Da questo momento in poi, rispettando le caratteristiche dell'architettura REST, ciascuna richiesta successiva inviata al server conterrà queste credenziali, mantenendo la natura **stateless** delle comunicazioni HTTP. Naturalmente **l'invio delle credenziali con la semplice codifica [Base64](#) non è sufficiente garanzia di riservatezza**, infatti viene usata solo congiuntamente a [TLS](#).

L'uso di [\*\*HTTP Digest Authentication\*\*](#) è sicuramente un passo avanti per **evitare la trasmissione in chiaro della password**, ma non è esente da possibili attacchi. Questa tecnica permette la negoziazione delle credenziali di autenticazione (username e password) usando delle **funzioni hash** su di esse prima di inviarli in rete, ad esempio applicando [MD5](#) con l'uso di un valore [nonce](#) per prevenire [attacchi replay](#).

La **soluzione che** in genere **andrebbe adottata** consiste nella creazione di un canale di trasmissione sicuro, come può essere [\*\*HTTPs\*\*](#), cioè **HTTP over TLS**. L'adozione di [\*\*HTTPs\*\*](#) garantisce la riservatezza e l'integrità nella trasmissione delle informazioni, coprendo gli altri aspetti di sicurezza.

## Sicurezza con “sessioni REST” e tramite servizi di terze parti (cenni)

Nel caso in cui i meccanismi di sicurezza propri del protocollo HTTP non siano sufficienti o adeguati per il tipo di Web Service da sviluppare, è sempre possibile **gestire la sicurezza in modo personalizzato**. In genere questo accade per la gestione dell'autenticazione e dell'autorizzazione dei client, dove i criteri possono essere diversi in base allo specifico dominio del problema.

Non c'è un obbligo nell'utilizzare i meccanismi standard del protocollo HTTP, ma è bene sottolineare che ogni eventuale soluzione personalizzata dovrebbe rispettare i principi REST. Infatti il **rischio di cadere in una soluzione non REST** per gestire l'autenticazione e l'autorizzazione dei client è alto, dal momento che di solito si tenterebbe di introdurre nuovamente il concetto di sessione associata al client appena autenticato.

In realtà il problema non sta nel concetto di sessione, ma nella modalità di gestione. Dal momento che **REST non ammette il mantenimento dello stato della sessione tra una richiesta e le altre**, occorre che **il client e il server si scambino tutte le informazioni necessarie per ricreare la sessione ad ogni interazione**. Questo vuol dire **rigenerare la sessione ad ogni richiesta** utilizzando le credenziali fornite dal client, con potenziale **perdita di prestazioni**, anche se i criteri di autenticazione ed autorizzazione non sono molto complessi.

## Lo stato come risorsa

Un'alternativa potrebbe essere **trasformare la sessione in risorsa**, utilizzando un approccio analogo a quello adottato per l'implementazione del carrello del negozio online visto in precedenza. L'autenticazione di un client corrisponde alla richiesta di creazione di una risorsa di tipo sessione: se le credenziali fornite dal client sono valide, allora **viene creata la risorsa ed inviato il corrispondente URI al client**, che da questo momento in poi **lo invierà al server con ogni richiesta**.

Ad ogni richiesta il server recupera la sessione per effettuare le necessarie verifiche e in caso positivo soddisfa le richieste del client.

Il server avrà la responsabilità di gestire opportunamente la sessione, come ad esempio eliminarla dopo un determinato periodo di inattività, ma non la manterrà in memoria, la gestirà come una normale risorsa persistente.

## OpenID e OAuth: gestione esterna della sicurezza

Come è possibile immaginare, la gestione dell'autenticazione e dell'autorizzazione dei client rischia di complicare la progettazione e l'implementazione di un Web Service. Una soluzione potrebbe essere quella di **delegare ad un servizio esterno** il compito di **gestire i client, utilizzando un Web Service specializzato nella gestione dell'autenticazione e/o dell'autorizzazione**.

Negli ultimi tempi sono venuti alla ribalta servizi aperti che si pongono proprio l'obiettivo di decentralizzare la gestione di alcuni aspetti della sicurezza. Tra i servizi di questo tipo i più noti e diffusi nell'ambito della realizzazione di Web Service REST sono [OpenID](#) e [OAuth](#).

**OpenID è un protocollo per la gestione dell'identità online.** Esso consente ad un client di presentare ad un Web Service o ad un sito Web un URI come dichiarazione della propria identità. L'applicazione Web chiede conferma della validità al relativo servizio di autenticazione (*OpenID provider*) il quale, dopo aver interagito con il client, conferma o smentisce l'autenticità dell'identità dichiarata.

**OAuth è un protocollo per la concessione a terze parti di autorizzazioni per l'accesso a risorse protette.** In pratica, un client può consentire ad un Web Service l'accesso ad una risorsa protetta, ospitata da un *service provider*, senza fornire le proprie credenziali.

È naturale che l'**utilizzo di servizi esterni per la gestione** di alcuni aspetti **della sicurezza presuppone** che il Web Service riponga **piena fiducia nel fornitore del servizio**, senza la quale questo tipo di servizi non avrebbero senso.

## Web Service REST usando Java Servlet

# Esempi di Web service di tipo REST in linguaggio Java

Il linguaggio di programmazione Java dispone di una estensione standard denominata **Servlet** che consente di scrivere con facilità il codice di gestione delle richieste HTTP ricevute da un server. Lo sviluppo di una servlet, cioè di una classe che deriva dalla classe astratta **HttpServlet**, è un modo semplice per realizzare un Web service di tipo **REST in Java**.

## Realizzazione di Web service di tipo REST mediante servlet

La realizzazione di **Web service** di tipo **REST** mediante servlet Java è basata sulle classi astratte e interfacce fondamentali del package `jakarta.servlet.http` indicate nella tabella seguente:

Classe	Descrizione
HttpServlet	È una classe astratta che estende la classe <i>GenericServlet</i> . Una servlet deve derivare dalla classe <i>HttpServlet</i> e sovrascrivere i metodi astratti <i>doGet</i> , <i>doPost</i> , <i>doPut</i> , <i>doDelete</i> , <i>doHead</i> , <i>doOptions</i> e <i>doTrace</i>
HttpServletRequest	È un'interfaccia che deriva dall'interfaccia <i>ServletRequest</i> e rappresenta una richiesta del protocollo HTTP. Permette di accedere agli <i>header</i> e di acquisirne il <i>body</i> della richiesta. È sempre un parametro dei metodi <i>doGet</i> , <i>doPost</i> , <i>doPut</i> , <i>doDelete</i> , <i>doHead</i> , <i>doOptions</i> e <i>doTrace</i>
HttpServletResponse	È un'interfaccia che deriva dall'interfaccia <i>ServletResponse</i> e rappresenta una risposta del protocollo HTTP. permette di impostare lo <i>status code</i> , gli <i>header</i> e di definire il <i>body</i> . È sempre un parametro dei metodi <i>doGet</i> , <i>doPost</i> , <i>doPut</i> , <i>doDelete</i> , <i>doHead</i> , <i>doOptions</i> e <i>doTrace</i>

La classe **HttpServlet**, che estende *GenericServlet*, espone i metodi fondamentali riportati nella tabella seguente:

Metodo	Descrizione
init	Metodo invocato dall'ambiente di esecuzione nel momento in cui la <i>servlet</i> viene attivata. Il metodo viene eseguito solo alla prima attivazione della <i>servlet</i>
service	Metodo invocato dall'ambiente di esecuzione al momento in cui riceve una richiesta. Se il metodo non è sovrascritto, invoca i metodi <i>doGet</i> , <i>doPost</i> , <i>doPut</i> , <i>doDelete</i> , <i>doHead</i> ,

	<i>doOptions</i> o <i>doTrace</i> , in funzione del tipo di richiesta ricevuta
destroy	Metodo invocato dall'ambiente di esecuzione quando la <i>servlet</i> viene disattivata, permettendo di rilasciare le risorse occupate
doGet, doPost, doPut, doDelete, doHead, doOptions	Metodi da sovrascrivere per definire il comportamento in risposta alle relative richieste del protocollo HTTP. Questi metodi sono invocati dall'ambiente di esecuzione quando il server riceve una richiesta specifica

Una **servlet** è una classe Java derivata da **HttpServlet**. La ridefinizione dei metodi *doGet*, *doPost*, *doPut*, *doDelete*, *doHead*, *doOptions* e *doTrace* consente di gestire le corrispondenti richieste del protocollo HTTP realizzando un Web service di tipo REST. Ognuno di questi metodi ha come parametri un oggetto che implementa l'interfaccia **HttpServletRequest** che rappresenta la **richiesta** ricevuta, e un oggetto che implementa l'interfaccia **HttpServletResponse** che rappresenta la **risposta** da restituire al client.

L'interfaccia **HttpServletRequest**, che deriva da *ServletRequest*, prevede i metodi fondamentali descritti nel seguente [link](#).

L'interfaccia **HttpServletResponse**, che deriva da *ServletResponse*, prevede i metodi fondamentali elencati nel seguente [link](#).

## Le applicazioni lato server [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021, **pp.236-250**.

- La programmazione server-side
- Modello a codice separato
- Struttura di una servlet
- La classe HttpServlet
- Ciclo di vita di una servlet
- Output sul client
- Deployment di un'applicazione Web
- Il Context XML descriptor o Deployment descriptor
- Esecuzione di una servlet
- Servlet concorrenti
- Vantaggi e svantaggi delle servlet

## Web service REST - CalcolatriceAPI

La seguente **servlet** Java realizza un semplice Web service che implementa una calcolatrice che restituisce il risultato di una delle quattro operazioni aritmetiche fondamentali. La richiesta avviene mediante URL che termina con una *query string* costituita da quattro valori: l'operazione effettuata, il formato di restituzione e i due operandi.

La *query string* viene costruita automaticamente compilando un form iniziale.

The screenshot shows a web browser window titled "Calcolatrice". The address bar displays "localhost:9090/CalculatorAPI/". The main content area contains a form with the following elements:

- Radio buttons for operation: ADD (selected), SUB, MUL, DIV.
- Radio buttons for format: XML (selected), JSON.
- Two input fields containing the numbers 8 and 9.
- Buttons for "Invia" and "Cancella".

Il codice per generare il form è il seguente. L'attributo action del form deve assumere come valore il path richiesto dalla servlet, che sarà illustrata successivamente.

### index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Calcolatrice</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <form action="calculator" method="GET">
            <input type="radio" name="operation" value="adder" checked/>ADD
            <input type="radio" name="operation" value="subtractor" />SUB
            <input type="radio" name="operation" value="multiplier" />MUL
            <input type="radio" name="operation" value="divisor" />DIV <br />
            <input type="radio" name="format" value="xml" />XML
            <input type="radio" name="format" value="json" />JSON <br />
            <input type="number" name="op1"/><br />
            <input type="number" name="op2"/><br />
            <input type="submit" value="Invia"/>
            <input type="reset" value="Cancella"/>
        </form>
    </body>
</html>
```

Selezionando l'operazione *ADD* e il formato *XML* viene generato il seguente URL di richiesta:

<http://localhost:9090/Calcolatrice/calculator?operation=adder&format=xml&op1=8&op2=9>

ottenendo il seguente risultato XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <statuscode>200</statuscode>
    <op1>8</op1>
    <op2>9</op2>
```

```
<operation>adder</operation>
<value>17</value>
</response>
```

Selezionando l'operazione *add* e il formato *JSON* viene generato il seguente URL di richiesta:

`http://localhost:9090/Calcolatrice/calcolatrice?operation=adder&format=json&op1=8&op2=9`

ottenendo il seguente risultato *JSON*.

```
{
  "response": {
    "statuscode": 200,
    "op1": 8,
    "op2": 9,
    "operation": "adder",
    "value": 17
  }
}
```

Gli URL generati in corrispondenza delle altre possibili richieste, usando il formato *XML*, corrispondono ai seguenti. Per il formato *JSON* gli URL sono analoghi, con la sola sostituzione del formato con *json*.

`http://localhost:9090/Calcolatrice/calcolatrice?operation=subtracter&format=xml&op1=8&op2=9`

`http://localhost:9090/Calcolatrice/calcolatrice?operation=multiplier&format=xml&op1=8&op2=9`

`http://localhost:9090/Calcolatrice/calcolatrice?operation=divisor&format=xml&op1=8&op2=9`

L'identificazione delle risorse utilizzate negli URI di questo Web service potrebbero apparire in contrasto con i principi RESTful. La **presenza di parametri nello URI** potrebbe far pensare ad una chiamata di procedura piuttosto che ad un identificatore di risorsa.

In realtà l'approccio è comunque **legittimo nell'ambito dell'architettura REST in quanto la stringa è univoca ed identifica esattamente la risorsa che vogliamo ottenere senza alcun effetto collaterale**. La presenza dei parametri può sembrare una chiamata di procedura, ma da un punto di vista REST questi sono visti come una sequenza di caratteri per l'identificazione di una risorsa.

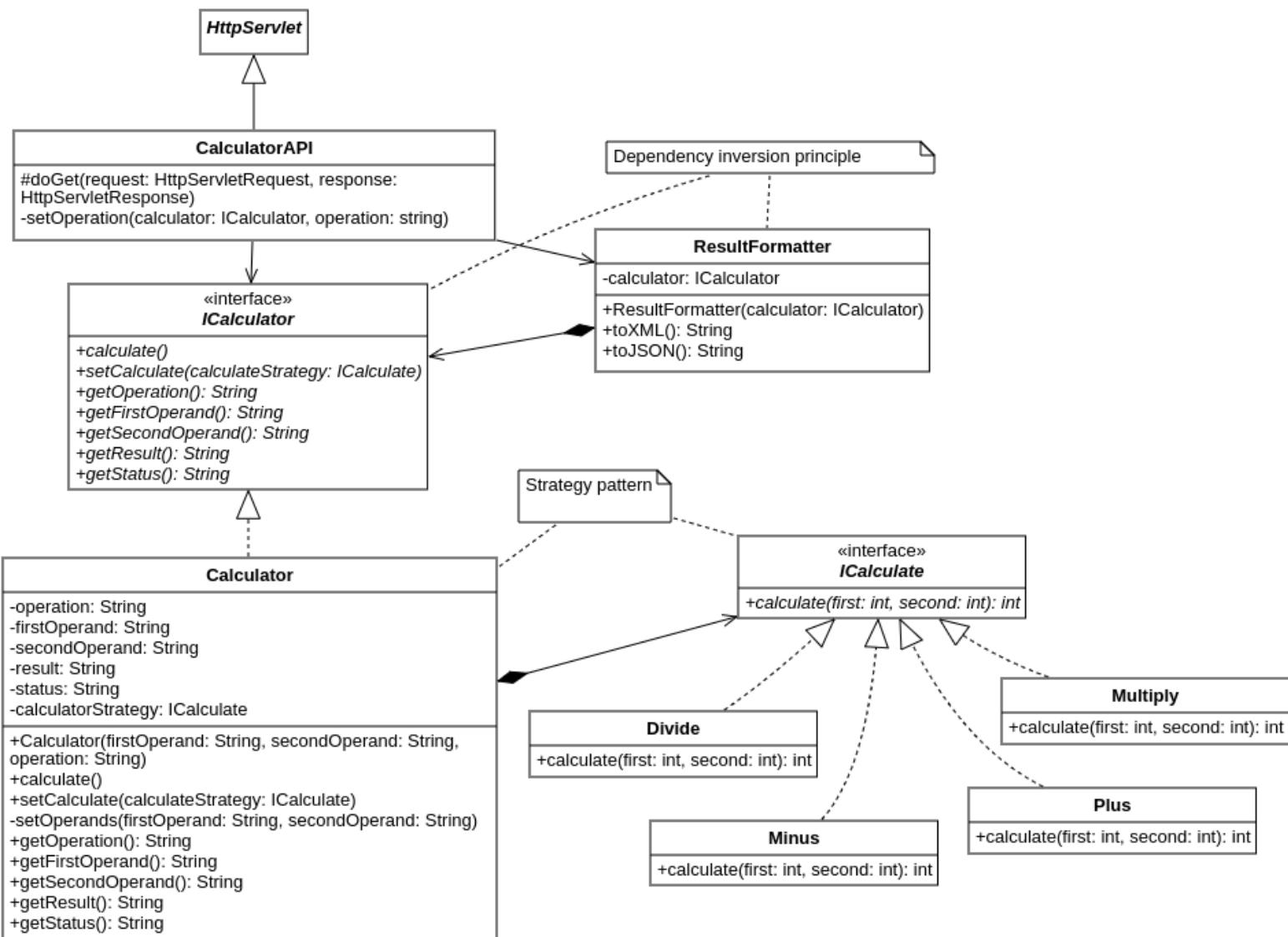
La logica dell'API è mostrata dal seguente diagramma delle classi che mostra la presenza della servlet nella classe **CalculatorAPI** che gestisce il solo metodo GET interfacciato al metodo `doGet()`. Il metodo `setOperation()` viene usato per impostare la strategia di calcolo in base alla scelta effettuata nel form della pagina `index.html`.

L'interfaccia **ICalculator** viene usata rendere debolmente accoppiata la classe contenente la servlet dalle classi che svolgono il ruolo della calcolatrice, **Calculator**, e la classe che si occupa di rappresentare l'operazione svolta in formato XML e JSON, **ResultFormatter**.

La servlet usa la classe **Calculator** tramite il metodo `setCalculate()` per impostare la strategia di calcolo e il metodo `calculate()` per effettuare il calcolo effettivo.

La classe **ResultFormatter** viene invece usata dalla servlet per rappresentare l'operazione svolta in formato XML e JSON, rispettivamente usando i metodi `toXML()` e `toJSON()`, invocati per permettere al Container di inviare la risposta al client nel formato richiesto.

L'interfaccia **ICalculate** viene invece usata dalla classe **Calculator** per delegare l'operazione di calcolo alle classi concrete che implementano le quattro operazioni fondamentali, **Divide**, **Minus**, **Plus** e **Multiply**.



Le classi coinvolte nella logica della calcolatrice usano uno *Strategy pattern*<sup>13</sup> per gestire le diverse operazioni aritmetiche. Viene invece applicato il principio *Dependency inversion*<sup>14</sup> per gestire le diverse modalità di restituzione del risultato all'API. Anche se questa progettazione può risultare eccessivamente complicata, l'uso dello *Strategy pattern* e l'applicazione del principio *Dependency inversion* permettono di aggiungere in modo molto semplice ulteriori operazioni e diverse modalità di restituzione del risultato all'API, senza richiedere eccessive modifiche al resto dell'applicazione. Per testare la semplicità di modifica dell'applicazione grazie all'uso dello *Strategy pattern* e del principio *Dependency*

<sup>13</sup> "Strategy - Refactoring Guru." <https://refactoring.guru/design-patterns/strategy>. Ultimo accesso: 12 apr. 2023.

[The Strategy Pattern Explained and Implemented in Java | Behavioral Design Patterns | Geekific](#) by Geekific

<sup>14</sup> [Learn SOLID Principles with CLEAN CODE Examples - Dependency Inversion Principle](#) by Amigoscode

*inversion* si propone di aggiungere l'operazione di calcolo del resto tra interi e la rappresentazione in formato CSV del risultato delle operazioni.

L'implementazione del Web service che genera la risposta XML/JSON con il risultato dell'operazione aritmetica, è realizzata utilizzando la **servlet** mostrata di seguito. Nella servlet è stato utilizzato il *deployment descriptor web.xml* per configurare la servlet, anziché le **annotazioni di Java**, che sono comunque riportate in modo commentato. Al riguardo, l'*annotazione @WebServlet* permette di impostare il nome della servlet attraverso l'*elemento name* (*CalculatorAPI*), mentre l'URL di invocazione della servlet viene impostato attraverso l'*elemento urlPatterns* (*/calculator*) che deve corrispondere al valore dell'attributo action del form nella pagina *index.html*. Se in quest'ultimo elemento fosse stato impostato il valore */\** si sarebbe potuto associare all'invocazione della servlet una qualsiasi sezione finale dell'URL, permettendo al codice della servlet stessa di modificare la propria risposta in base all'analisi della struttura dello URL.

Il *deployment descriptor web.xml* usato per configurare la **servlet** è riportato di seguito. In esso il tag `<web-app>` definisce il tag radice all'interno del quale vengono configurate tutte le servlet usate nell'applicazione.

Il tag `<servlet>` permette di definire le servlet, indicando il nome con il tag `<servlet-name>` e la classe ad essa associata con il tag `<servlet-class>`.

Con il tag `<servlet-mapping>` viene specificata l'associazione tra una servlet e l'URL con cui viene invocata. L'associazione viene creata riprendendo il nome della servlet già definita nella sezione `<servlet>` con il tag `<servlet-name>`, e identificata con lo stesso tag anche in `<servlet-mapping>`, e l'URL indicato nel tag `<url-pattern>`. Anche in questo caso l'elemento del tag `<url-pattern>` sarebbe potuto essere il valore */\** invece di */calculator*.

Con il tag `<welcome-file-list>` vengono elencate le eventuali pagine da visualizzare quando viene invocata la servlet tramite l'URL indicato nel tag `<url-pattern>`.

### **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="5.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
  https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd">
  <servlet>
    <servlet-name>CalculatorAPI</servlet-name>
    <servlet-class>servlet.CalculatorAPI</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CalculatorAPI</servlet-name>
    <url-pattern>/calculator</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

La classe **CalculatorAPI** contiene la servlet invocata dal *Container* ogni volta che un client invia una richiesta di calcolo. Il *Container* avvia un thread per ogni richiesta.

Per il corretto funzionamento delle servlet in Java è necessario importare la libreria Jakarta EE 9 API Library o superiori.

### CalculatorAPI.java<sup>15</sup>

```
package servlet;

import application.Calculator;
import application.behavior.ICalculator;
import application.ResultFormatter;
import application.concrete.*;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
//import jakarta.servlet.annotation.WebServlet;

/**
 * Calculator API
 * @author mgm
 */
//@WebServlet(name = "CalculatorAPI", urlPatterns = {"calculator"})
public class CalculatorAPI extends HttpServlet {

    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
        throws ServletException, IOException {
        try (PrintWriter out = response.getWriter()) {
            String op1 = request.getParameter("op1");
            String op2 = request.getParameter("op2");
            String format = request.getParameter("format");
            String operation = request.getParameter("operation");
            ICalculator calculator = new Calculator(op1, op2, operation);
            ResultFormatter resultFormatter = new ResultFormatter(calculator);
            if ("200".equals(calculator.getStatus())) {
```

---

<sup>15</sup> [How can I create a RESTful calculator?, stackoverflow,](#)

```
        setOperation(calculator, operation);
        calculator.calculate();
    }
    if ("xml".equals(format)) {
        response.setContentType("application/xml; charset=UTF-8");
        out.println(resultFormatter.toXML());
    } else {
        response.setContentType("application/json; charset=UTF-8");
        out.println(resultFormatter.toJSON());
    }
}
}

/**
 * Sets the operation strategy
 * @param calculator calculator object
 * @param operation operation requested
 */
private void setOperation(ICalculator calculator, String operation) {
    switch (operation) {
        case "adder" -> calculator.setCalculate(new Plus());
        case "subtractor" -> calculator.setCalculate(new Minus());
        case "multiplier" -> calculator.setCalculate(new Multiply());
        case "divisor" -> calculator.setCalculate(new Divide());
    }
}
}
```

L'interfaccia **ICalculator** permette di far dipendere la servlet dall'astrazione rappresentata dall'interfaccia stessa, e non dalla classe concreta **Calculator**, rendendo così le classi debolmente accoppiate e permettendo di applicare il principio di *Dependency inversion* nella classe **ResultFormatter** che si occupa di gestire la formattazione del risultato da restituire al richiedente.

### **ICalculator.java**

```
package application.behavior;

import application.behavior.ICalculate;

public interface ICalculator {

    /**
     * Executes the calculation that was set as strategy
     */
    void calculate();

    /**
     * Sets the calculation strategy
     * @param calculatorStrategy one of the basic arithmetic operations
    }
```

```
/*
void setCalculate(ICalculate calculatorStrategy);

/**
 * Gets the operation
 * @return the operation
 */
public String getOperation();

/**
 * Gets the first operand
 * @return the first operand
 */
public String getFirstOperand();

/**
 * Gets the second operand
 * @return the second operand
 */
public String getSecondOperand();

/**
 * Gets the result
 * @return the result
 */
public String getResult();

/**
 * Gets the status code
 * @return status code
 */
String getStatus();
}
```

La classe **ResultFormatter** che si occupa di gestire la formattazione del risultato da restituire al richiedente. La classe risulta debolmente accoppiata alla classe **Calculator** dato che dipende solo dall'interfaccia **ICalculate** e non dalla classe concreta **Calculator**. Questo rende l'applicazione flessibile e permette di aggiungere modalità di formattazione del risultato senza richiedere particolari modifiche alla classe che implementa la calcolatrice.

### **ResultFormatter.java**

```
package application;

import application.behavior.ICalculate;

public class ResultFormatter {
    private ICalculate calculator;
```

```
public ResultFormatter(ICalculator calculator) {
    this.calculator = calculator;
}

/**
 * Creates an XML string
 * @return the XML string
 */
public String toXML() {
    return """
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
            <statuscode>%s</statuscode>
            <op1>%s</op1>
            <op2>%s</op2>
            <operation>%s</operation>
            <value>%s</value>
        </response>
        """.formatted(calculator.getStatus(),
            calculator.getFirstOperand(),
            calculator.getSecondOperand(),
            calculator.getOperation(),
            calculator.getResult());
}

/**
 * Creates a JSON string
 * @return the JSON string
 */
public String toJSON() {
    String first = ("400".equals(calculator.getStatus())) ?
        "\\" + calculator.getFirstOperand() + "\\" :
        calculator.getFirstOperand();
    String second = ("400".equals(calculator.getStatus())) ?
        "\\" + calculator.getSecondOperand() + "\\" :
        calculator.getSecondOperand();
    String res = ("400".equals(calculator.getStatus())) ?
        "\\" + calculator.getResult() + "\\" :
        calculator.getResult();
    String json = """
        {
            "response": {
                "statuscode": "%s",
                "op1": %s,
                "op2": %s,
                "operation": "%s",
                "value": %s
            }
        }
    """
}
```

```
        """ .formatted(calculator.getStatus(), first, second,
                    calculator.getOperation(), res);
    return json;
}
}
```

La classe **Calculator** effettua il calcolo richiesto dal client usando la strategia impostata nella servlet.

### Calculator.java

```
package application;

import application.behavior.ICalculator;
import application.behavior.ICalculate;

/**
 * A simple calculator that uses Strategy pattern
 * @author mgm
 */
public class Calculator implements ICalculator {
    private String operation; // operation requested
    private String firstOperand; // 1st operand
    private String secondOperand; // 2nd operand
    private String result; // operation result
    private String status; // status code
    private ICalculate calculatorStrategy; // operation strategy

    /**
     * Calculator constructor
     * @param firstOperand 1st operand
     * @param secondOperand 2nd operand
     * @param operation operation that is applied
     */
    public Calculator(String firstOperand, String secondOperand, String operation) {
        setOperands(firstOperand, secondOperand);
        this.operation = operation;
    }

    /**
     * Executes the calculation that was set as strategy
     */
    @Override
    public void calculate() {
        int res = calculatorStrategy.calculate(Integer.parseInt(firstOperand),
                                              Integer.parseInt(secondOperand));
        result = ((Integer) res).toString();
        status = "200";
    }
}
```

```
/**  
 * Sets the calculation strategy  
 * @param calculatorStrategy one of the basic arithmetic operations  
 */  
@Override  
public void setCalculate(ICalculate calculatorStrategy) {  
    this.calculatorStrategy = calculatorStrategy;  
}  
  
/**  
 * Checks the two operands, and assigns a corresponding value to each one  
 * @param firstOperand the 1st operand  
 * @param secondOperand the 2nd operand  
 */  
private void setOperands(String firstOperand, String secondOperand) {  
    if (firstOperand == null || firstOperand.trim().isEmpty()  
        || secondOperand == null || secondOperand.isEmpty()) {  
        result = "ERRORE";  
        status = "400";  
        this.firstOperand = "-";  
        this.secondOperand = "-";  
    } else {  
        status = "200";  
        this.firstOperand = firstOperand;  
        this.secondOperand = secondOperand;  
    }  
}  
  
/**  
 * Gets the operation  
 * @return the operation  
 */  
@Override  
public String getOperation() {  
    return operation;  
}  
  
/**  
 * Gets the first operand  
 * @return the first operand  
 */  
@Override  
public String getFirstOperand() {  
    return firstOperand;  
}  
  
/**  
 * Gets the second operand  
 * @return the second operand  
 */
```

```
/*
@Override
public String getSecondOperand() {
    return secondOperand;
}

/***
 * Gets the result
 * @return the result
 */
@Override
public String getResult() {
    return result;
}

/***
 * Gets the status code
 * @return status code
 */
@Override
public String getStatus() {
    return status;
}
}
```

L'interfaccia **ICalculate** viene usata come supertipo per la strategia di calcolo dell'attributo calculatorStrategy presente nella classe **Calculator**. Questa interfaccia viene concretizzata dalle classi che implementano le quattro operazioni aritmetiche, permettendo di implementare lo *Strategy pattern*.

### **ICalculate.java**

```
package application.behavior;

/**
 * Interface used to set the calculation strategy
 * @author mgm
 */
public interface ICalculate {
    public int calculate(int first, int second);
}
```

### **Divide.java**

```
package application.concrete;

import application.behavior.ICalculate;

/**
 * Division strategy
 * @author mgm
 */

```

```
public class Divide implements ICalculate {  
  
    @Override  
    public int calculate(int first, int second) {  
        return first / second;  
    }  
}
```

### Minus.java

```
package application.concrete;  
  
import application.behavior.ICalculate;  
  
/**  
 * Subtraction strategy  
 * @author mgm  
 */  
public class Minus implements ICalculate {  
  
    @Override  
    public int calculate(int first, int second) {  
        return first - second;  
    }  
}
```

### Multiply.java

```
package application.concrete;  
  
import application.behavior.ICalculate;  
  
/**  
 * Addition strategy  
 * @author mgm  
 */  
public class Plus implements ICalculate {  
  
    @Override  
    public int calculate(int first, int second) {  
        return first * second;  
    }  
}
```

### Plus.java

```
package application.concrete;  
  
import application.behavior.ICalculate;  
  
/**  
 * Addition strategy
```

```
* @author mgm
*/
public class Plus implements ICalculate {

    @Override
    public int calculate(int first, int second) {
        return first + second;
    }
}
```

# Web service per operazioni CRUD su database

Per rendere persistenti le risorse esposte da un Web service può essere utilizzato un database relazionale gestito da un DBMS, implementando in questo modo una **architettura a tre livelli**:

1. il **Presentation layer** che presenterà al **client** l'interfaccia per poter accedere alla risorsa, utilizzando i metodi offerti dal *gestore della risorsa*;
2. il **Business logic layer** costituito dal **Web server** o **Application server** che riveste il ruolo di *container* ed esegue la servlet che implementa il Web service;
3. il **Resource Management layer** costituito dal **server DBMS** che gestisce i dati memorizzati in modo persistente.

Un'architettura di questo tipo, definita **three-tier**, garantisce un ampio disaccoppiamento tra i vari elementi del sistema software, permettendo una diversa gestione della sicurezza sui diversi layer, rendendoli facilmente scalabili e manutenibili e permettendo aggiornamenti mirati per ogni tier.

## Servlet e database [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021, **pp.251-258**.

- La connessione ai database
- La connessione con JDC Java Database Connectivity
- Tipi di driver JDBC
- Utilizzare JDBC standalone
- Servlet con connessione a MySQL
  - [CRUD in Servlet](#)
  - [Interface PreparedStatement](#)
  - [Method executeQuery](#)
  - [Method executeUpdate](#)
  - [The finally Block](#)

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021, **pp.274-291**.

- Lab. 2 - XAMPP e il server engine Tomcat
- Lab. 3 - L'inizializzazione delle servlet
- Lab. 4 - L'interazione GET/POST tra client e servlet
- Lab. 6 - JDBC e MySQL

**LEGGERE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021, **pp.292-306**.

- Lab. 5 - L'interazione con client Ajax e lo scambio dati in formato JSON
- Lab. 6 - La permanenza dei dati con le servlet: i cookie
- Lab. 7 - La permanenza dei dati con le servlet: le sessioni

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021, **pp.307-311.**

- Lab. 8 - JDBC e MySQL

## Countries REST su MySQL

In questo paragrafo viene mostrata un'applicazione REST completa che permette di eseguire le operazioni CRUD in un database MySQL.

I dati gestiti riguardano i nomi di diverse nazioni con il corrispondente codice Alpha-2 definito in ISO 3166<sup>16</sup> e una primary key autogenerata.

L'API permette di gestire i dati delle nazioni con un interscambio in formato JSON, consentendo di ottenere tutti i dati delle nazioni o di una specifica nazione tramite il suo id, aggiungerne una nuova, modificare i dati di una nazione fornendo il suo id ed eliminare una nazione sempre tramite il suo id. Le diverse operazioni sono mostrate dalle immagini seguenti usando come client l'estensione RESTED su Firefox.

Nella *figura 1* viene mostrata la richiesta per ottenere l'elenco di tutte le nazioni. Come indicato dalle regole dell'architettura REST il metodo HTTP usato è GET e l'URL della richiesta REST identifica l'insieme di tutte le nazioni.

Nella *figura 2* viene mostrata la richiesta per ottenere lo stato di una nazione fornendo nell'URL il suo identificativo. Il metodo HTTP usato è sempre GET, ma l'URL della richiesta REST in questo caso identifica una risorsa precisa.

La *figura 3* mostra invece l'aggiunta di una nuova nazione. Il metodo HTTP usato è POST, l'URL è quello generale, ma nel body vengono inseriti i dati della nuova nazione. Viene previsto anche lo header Content-Type con il valore application/json affinché la richiesta vada a buon fine. Per verificare l'aggiunta della nuova nazione viene mostrata nella *figura 4* l'operazione di lettura dello stato della nazione appena inserita.

Nella *figura 5* viene invece mostrato l'aggiornamento dei dati di una nazione; per semplicità viene modificata la nazione appena inserita. Il metodo HTTP usato è PUT, l'URL riporta l'id della risorsa da variare e nel body del metodo PUT vengono inseriti i nuovi valori, includendo anche lo header Content-Type con il valore application/json affinché la richiesta vada a buon fine. Nella *figura 6* viene effettuata l'operazione di lettura dello stato della risorsa per verificare la modifica.

Infine nella *figura 7* viene eliminata una risorsa usando il metodo HTTP DELETE e inserendo nello URL il suo id. La *figura 8* riporta l'elenco delle nazioni nel quale manca la risorsa con id pari a 2.

---

<sup>16</sup> "List of country codes by alpha-2, alpha-3 code (ISO 3166) - IBAN." <https://www.iban.com/country-codes>. Ultimo accesso: 7 mag. 2023.

## Figura 1

Request

GET <http://localhost:8080/DAOCountryCRUD/countries>

Headers >

Basic auth >

Response (0.147s) - http://localhost:8080/DAOCountryCRUD/countries

200

Headers >

```
[  
  {  
    "id": "1",  
    "name": "France",  
    "iso2": "FR"  
  },  
  {  
    "id": "2",  
    "name": "Germany",  
    "iso2": "DE"  
  },  
  {  
    "id": "3",  
    "name": "England",  
    "iso2": "UK"  
  },  
  {  
    "id": "4",  
    "name": "Spain",  
    "iso2": "SP"  
  },  
  {  
    "id": "5",  
    "name": "Belgium",  
    "iso2": "BE"  
  },  
  {  
    "id": "6",  
    "name": "Italy",  
    "iso2": "IT"  
  },  
  {  
    "id": "7",  
    "name": "Portugal",  
    "iso2": "PT"  
  },  
  {  
    "id": "8",  
    "name": "Austria",  
    "iso2": "AT"  
  }]
```

Figura 2

Request

GET <http://localhost:8080/DAOCountryCRUD/countries/6>

Headers >

Basic auth >

Response (0.014s) - http://localhost:8080/DAOCountryCRUD/countries/6

200

Headers >

```
{  
  "id": "6",  
  "name": "Italy",  
  "iso2": "IT"  
}
```

Figura 3

Request

POST <http://localhost:8080/DAOCountryCRUD/countries>

Headers ▾

Content-Type application/json

+Add header

Basic auth >

Request body ▾

Type JSON

name Rwanda

iso2 RW

+Add parameter

Response (0.018s) - http://localhost:8080/DAOCountryCRUD/countries

200

Headers >

Figura 4

Request

GET  http://localhost:8080/DAOCountryCRUD/countries/14

Headers   
+Add header  
Basic auth >

Response (0.017s) - http://localhost:8080/DAOCountryCRUD/countries/14

200

Headers >

```
{  
  "id": "14",  
  "name": "Rwanda",  
  "iso2": "RW"  
}
```

Figura 5

Request

PUT  http://localhost:8080/DAOCountryCRUD/countries/14

Headers   
Content-Type   
+Add header  
Basic auth >

Request body   
Type    

<input type="text" value="name"/>	<input type="text" value="South Africa"/>
<input type="text" value="iso2"/>	<input type="text" value="ZA"/>

  
+Add parameter

Response (0.016s) - http://localhost:8080/DAOCountryCRUD/countries/14

200

Headers >

*Figura 6*

Request

GET  http://localhost:8080/DAOCountryCRUD/countries/14

Headers ▾  
+ Add header  
Basic auth >

---

Response (0.011s) - http://localhost:8080/DAOCountryCRUD/countries/14

**200**

Headers >

```
{  
    "id": "14",  
    "name": "South Africa",  
    "iso2": "ZA"  
}
```

*Figura 7*

Request

DELETE  http://localhost:8080/DAOCountryCRUD/countries/2

Headers ▾  
+ Add header  
Basic auth >  
Request body >

---

Response (0.021s) - http://localhost:8080/DAOCountryCRUD/countries/2

**200**

Headers >

## Figura 8

Request

GET <http://localhost:8080/DAOCountryCRUD/countries>

Headers ▾  
+Add header  
Basic auth >

Response (0.016s) - http://localhost:8080/DAOCountryCRUD/countries

200

Headers >

```
[{"id": "3", "name": "England", "iso2": "UK"}, {"id": "4", "name": "Spain", "iso2": "SP"}, {"id": "5", "name": "Belgium", "iso2": "BE"}, {"id": "6", "name": "Italy", "iso2": "IT"}, {"id": "7", "name": "Portugal", "iso2": "PT"}, {"id": "8", "name": "Ireland", "iso2": "IE"}, {"id": "9", "name": "Luxembourg", "iso2": "LU"}]
```

Il database countriesdb è costituito da una sola tabella, countries, di seguito viene fornito lo script SQL per la sua creazione.

### **countriesdb.sql**

```
-- phpMyAdmin SQL Dump
-- version 5.2.1deb1
```

```
-- https://www.phpmyadmin.net/
--
-- Host: localhost:3306
-- Generation Time: May 07, 2023 at 11:45 PM
-- Server version: 8.0.32-0ubuntu4
-- PHP Version: 8.1.12-1ubuntu4

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `countriesdb`
--

CREATE DATABASE IF NOT EXISTS `countriesdb` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
USE `countriesdb`;

-----

-- Table structure for table `countries`
--

CREATE TABLE IF NOT EXISTS `countries` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL,
  `iso2` char(2) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Dumping data for table `countries`
--

INSERT INTO `countries` (`id`, `name`, `iso2`) VALUES
(3, 'England', 'UK'),
(4, 'Spain', 'SP'),
(5, 'Belgium', 'BE'),
(6, 'Italy', 'IT'),
(7, 'Portugal', 'PT'),
(8, 'Ireland', 'IE'),
(9, 'Luxembourg', 'LU'),
(10, 'Brazil', 'BR'),
(12, 'Oman', 'OM'),
(13, 'Norfolk Island', 'NF'),
(14, 'South Africa', 'ZA');
COMMIT;
```

```
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

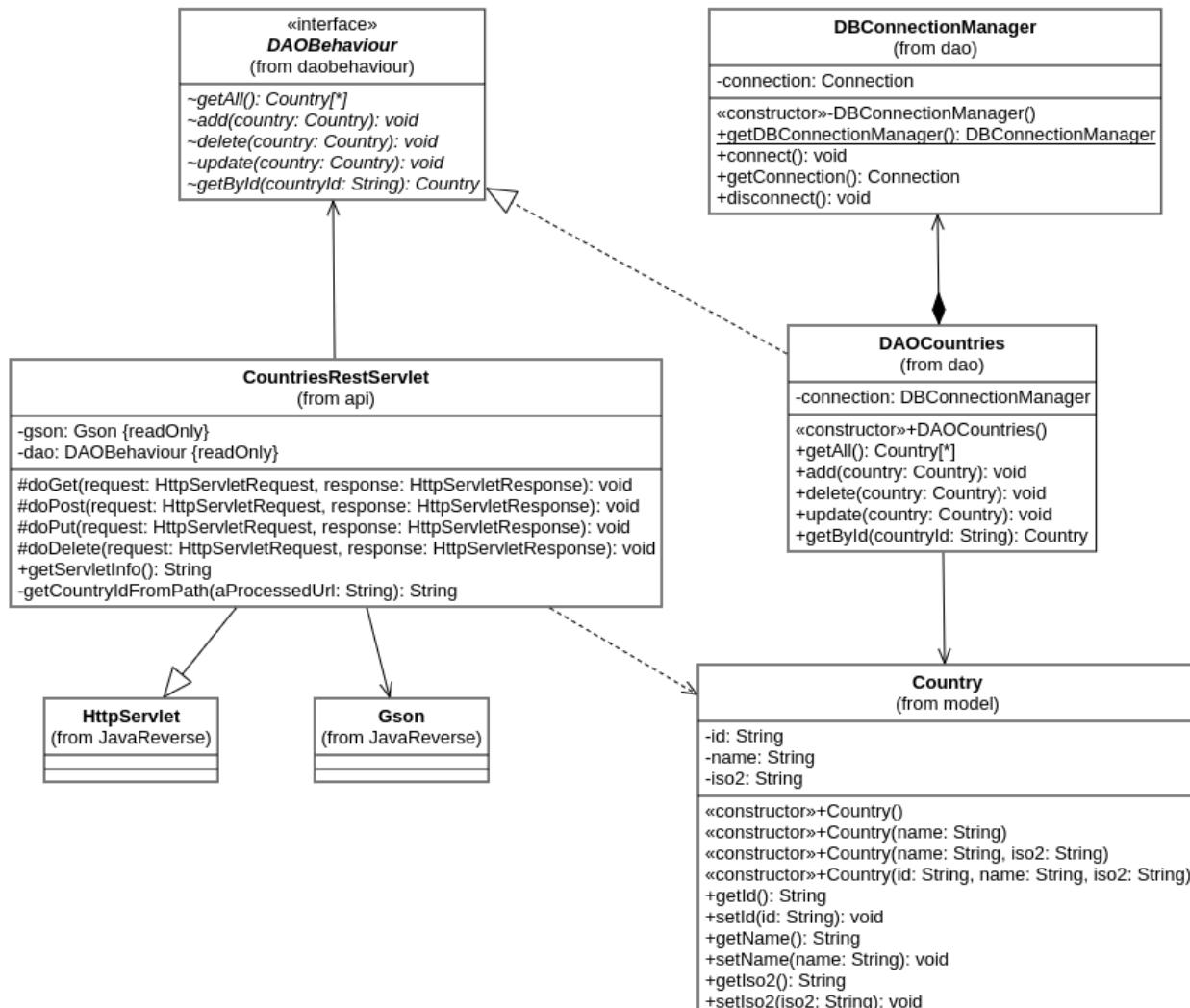
L'API prevede l'uso del server [Apache TomEE](#), della libreria [gson-2.10.1.jar](#) per la serializzazione dei dati in formato JSON e la libreria [mysql-connector-j-8.0.33.jar](#) per la connessione al database. I link fanno riferimento alle versioni disponibili a maggio 2023.

Le classi che costituiscono l'API sono:

- la classe CountriesRestServlet che contiene la servlet, implementa i metodi doGet(), doPost(), doPut() e doDelete() e tramite la libreria gson-2.10.1.jar serializza e deserializza gli elementi JSON;
- la classe Country che è la classe POJO e che quindi rappresenta la Business logic;
- la classe DBConnectionManager che si occupa della connessione con il database;
- la classe DAOCountries che si occupa della logica di interrogazione del database;
- l'interfaccia DAOBehaviour che definisce il comportamento di interrogazione del database.

Infine il deployment descriptor web.xml effettua il mapping della servlet con un URL generale che permette di variare la parte finale a seconda delle necessità, usando il simbolo /\*.

Di seguito viene mostrato il diagramma delle classi dell'API e successivamente viene fornito il codice.



## web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="5.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd">
    <servlet>
        <servlet-name>CountriesRestServlet</servlet-name>
        <servlet-class>cc.maffucci.api.CountriesRestServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CountriesRestServlet</servlet-name>
        <url-pattern>/countries/*</url-pattern>
    </servlet-mapping>
</web-app>
```

## CountriesRestServlet.java

```
package cc.maffucci.api;

import cc.maffucci.daobehaviour.DAOBehaviour;
import cc.maffucci.dao.DAOCountries;
import cc.maffucci.model.Country;
import com.google.gson.Gson;
//import com.google.gson.GsonBuilder;
import java.io.IOException;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.PrintWriter;
import java.util.List;

public class CountriesRestServlet extends HttpServlet {
    // GsonBuilder builder constructs a Gson instance setting configuration
    // options other than the default, like setPrettyPrinting. The method
    // create() has to be the last one to be invoked.
// https://www.javadoc.io/doc/com.google.code.gson/2.8.5/com/google/gson/Gson.html
// https://repo1.maven.org/maven2/com/google/code/gson/2.10.1/
    private final Gson gson = new Gson();
    // GsonBuilder builder constructs a Gson instance setting configuration
    // options other than the default, like setPrettyPrinting. The method
    // create() has to be the last one to be invoked.
// https://www.javadoc.io/doc/com.google.code.gson/2.6/com/google/gson/GsonBuilder.html
    //private final Gson gson = GsonBuilder().setPrettyPrinting().create();
    private final DAOBehaviour dao = new DAOCountries();

    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
}
```

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("application/json;utf-8");
    // Difference between getPathInfo() and getRequestURI
// https://stackoverflow.com/questions/4931323/whats-the-difference-between-getrequesturi-and-getpathinfo-methods-in-httpservl
    // In this case getPathInfo() will return the path after /countries (see web.xml)
    String countryId = this.getCountryIdFromPath(request.getPathInfo());
    if (countryId != null) {
        // Gets a single entity by id
        // URL like "/countries/id" where "id" is the country id
        Country country = dao.getById(countryId);
        gson.toJson(country, response.getWriter());
    } else {
        // Gets all countries
        // URL like "/countries"
        List<Country> countryList = dao.getAll();
        gson.toJson(countryList, response.getWriter());
    }
}

/**
 * Handles the HTTP <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("application/json;utf-8");
    // The URL has to be like "/countries"
    // The new country data in JSON format is in the body
    Country newCountry = gson.fromJson(request.getReader(), Country.class);
    dao.add(newCountry);
}

/**
 * Handles the HTTP <code>PUT</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPut(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("application/json;utf-8");
    // The URL has to be like "/countries"
    // The updated country data in JSON format is in the body
```

```
Country newCountry = gson.fromJson(request.getReader(), Country.class);
String countryId = this.getCountryIdFromPath(request.getPathInfo());
if (countryId == null) {
    // DELETE should have a single resource request
// https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html
    response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
} else {
    // Update a single entity by id
    // URL like "/countries/id" where "id" is the country id
    newCountry.setId(countryId);
    dao.update(newCountry);
}

}

/***
 * Handles the HTTP <code>DELETE</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doDelete(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
PrintWriter out = new PrintWriter(response.getWriter());
response.setContentType("application/json;utf-8");
// Difference between getPathInfo() and getRequestURI
// https://stackoverflow.com/questions/4931323/whats-the-difference-between-getrequesturi-and-getpathinfo-methods-in-httpservl
// In this case getPathInfo() will return the path after /countries (see web.xml)
String countryId = this.getCountryIdFromPath(request.getPathInfo());
if (countryId != null) {
    // Deletes a single entity by id
    // URL like "/countries/id" where "id" is the country id
    Country country = dao.getById(countryId);
    dao.delete(country);

} else {
    // DELETE should have a single resource request
// https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html
    response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
}
}

/***
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
}
```

```
}

/** 
 * Returns parameters after "/countries", if there is any
 * @param aProcessedUrl the final part of the URL
 * @return the final part of the URL after "/countries" if there is any
 */
private String getCountryIdFromPath(String aProcessedUrl) {
    // If the URL is null, it returns null
    if (aProcessedUrl == null) return null;
    // Splits the URL in its basic components using the symbol "/" as separator
    String[] split = aProcessedUrl.split("/");
    if (split.length == 0)
        // No other parameters after "/countries"
        return null;
    // The country id is placed after "/countries"
    else return split[1];
}
}
```

## Country.java

```
package cc.maffucci.model;

import java.util.UUID;

public class Country {
    private String id;
    private String name;
    private String iso2;

    public Country() {}

    public Country(String name) {
        this.name = name;
    }

    public Country(String name, String iso2) {
        this.name = name;
        this.iso2 = iso2;
    }

    public Country(String id, String name, String iso2) {
        this.id = id;
        this.name = name;
        this.iso2 = iso2;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
}
```

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getIso2() {
    return iso2;
}

public void setIso2(String iso2) {
    this.iso2 = iso2;
}
}
```

### **DAOBehaviour.java**

```
package cc.maffucci.daobehaviour;

import cc.maffucci.model.Country;
import java.util.List;

public interface DAOBehaviour {
    List<Country> getAll();
    void add(Country country);
    void delete(Country country);
    void update(Country country);
    Country getById(String countryId);
}
```

### **DAOCountries.java**

```
package cc.maffucci.dao;

import cc.maffucci.daobehaviour.DAOBehaviour;
import cc.maffucci.model.Country;
import java.util.ArrayList;
import java.util.List;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DAOCountries implements DAOBehaviour {
    private DBConnectionManager connection; // db connection

    public DAOCountries() {
        connection = DBConnectionManager.getDBConnectionManager();
    }

    @Override
    public List<Country> getAll() {
        // https://alvinalexander.com/java/java-mysql-select-query-example/
        String query = "SELECT * FROM countries";
        List<Country> countries = new ArrayList<>();
    }
}
```

```
connection.connect();
try {
    ResultSet resultSet =
        connection.getConnection().prepareStatement(query).executeQuery();
    while (resultSet.next()) {
        countries.add(new Country(Integer.toString(resultSet.getInt("id")),
            resultSet.getString("name"),
            resultSet.getString("iso2")));
    }
} catch (SQLException ex) {
    Logger.getLogger(DAOCountries.class.getName())
        .log(Level.SEVERE, null, ex);
} finally {
    try {
        connection.disconnect();
    } catch (SQLException ex) {
        Logger.getLogger(DAOCountries.class.getName())
            .log(Level.SEVERE, null, ex);
    }
}
return countries;
}

@Override
public void add(Country country) {
// https://alvinalexander.com/java/java-mysql-insert-example-statement/
String query = "INSERT INTO countries(name, iso2) VALUES (?, ?)";
connection.connect();
try {
    PreparedStatement preparedStm =
        connection.getConnection().prepareStatement(query);
    preparedStm.setString(1, country.getName());
    preparedStm.setString(2, country.getIso2());
    preparedStm.execute();
} catch (SQLException ex) {
    Logger.getLogger(DAOCountries.class.getName())
        .log(Level.SEVERE, null, ex);
} finally {
    try {
        connection.disconnect();
    } catch (SQLException ex) {
        Logger.getLogger(DAOCountries.class.getName())
            .log(Level.SEVERE, null, ex);
    }
}
}

@Override
public void delete(Country country) {
// https://alvinalexander.com/java/java-mysql-delete-query-example/
String query = "DELETE FROM countries WHERE id = ?";
connection.connect();
try {
    PreparedStatement preparedStm =
```

```
        connection.getConnection().prepareStatement(query);
preparedStm.setInt(1, Integer.parseInt(country.getId()));
preparedStm.execute();
} catch (SQLException ex) {
    Logger.getLogger(DAOCountries.class.getName())
        .log(Level.SEVERE, null, ex);
} finally {
    try {
        connection.disconnect();
    } catch (SQLException ex) {
        Logger.getLogger(DAOCountries.class.getName())
            .log(Level.SEVERE, null, ex);
    }
}
}

@Override
public void update(Country country) {
// https://alvinalexander.com/java/java-mysql-update-query-example/
String query = "UPDATE countries SET name = ?, iso2 = ? WHERE id = ?";
connection.connect();
try {
    PreparedStatement preparedStm =
        connection.getConnection().prepareStatement(query);
    preparedStm.setString(1, country.getName());
    preparedStm.setString(2, country.getIso2());
    preparedStm.setString(3, country.getId());
    preparedStm.execute();
} catch (SQLException ex) {
    Logger.getLogger(DAOCountries.class.getName())
        .log(Level.SEVERE, null, ex);
} finally {
    try {
        connection.disconnect();
    } catch (SQLException ex) {
        Logger.getLogger(DAOCountries.class.getName())
            .log(Level.SEVERE, null, ex);
    }
}
}

@Override
public Country getById(String countryId) {
// https://alvinalexander.com/java/java-mysql-select-query-example/
String query = "SELECT * FROM countries WHERE id = " + countryId;
Country country = null;
connection.connect();
try {
    ResultSet resultSet =
        connection.getConnection().prepareStatement(query).executeQuery();
    while (resultSet.next()) {
        country = new Country(Integer.toString(resultSet.getInt("id")),
            resultSet.getString("name"),
            resultSet.getString("iso2"));
    }
}
}
```

```
        }
    } catch (SQLException ex) {
        Logger.getLogger(DAOCountries.class.getName())
            .log(Level.SEVERE, null, ex);
    } finally {
        try {
            connection.disconnect();
        } catch (SQLException ex) {
            Logger.getLogger(DAOCountries.class.getName())
                .log(Level.SEVERE, null, ex);
        }
    }
}
return country;
}
}
```

## DBConnectionManager.java

```
package cc.maffucci.dao;

import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * This class manages the db connection. This class uses a Singleton design
 * pattern in order to create only one db connection instance, and it is
 * returned in all subsequent calls. To create a single db connection instance
 * the class has a private constructor, so clients cannot create any instances,
 * but there is a method which returns an instance of the class,
 * getDBConnectionManager(), that creates an instance only if there was none,
 * otherwise it returns the existing one.
 *
 * @author mgm
 */
public class DBConnectionManager {

    private Connection connection; // connection to the db
    private static DBConnectionManager instance = null; // class instance

    /**
     * Class constructor. The constructor is private so no object can be
     * instantiated.
     */
    private DBConnectionManager() {
    }

    /**
     * Return the single class instance. This method creates an instance if
     * there is none, otherwise it returns the existing one.
     *
     * @return the single instance reference
     */
    public static DBConnectionManager getDBConnectionManager() {
        if (DBConnectionManager.instance == null) {
```

```
DBConnectionManager.instance = new DBConnectionManager();
}
return DBConnectionManager.instance;
}

/**
 * Connects to the db
 *
 * @throws SQLException
 */
public void connect() {
    try {
        String url = "jdbc:mysql://localhost:3306/countriesdb";
        String user = "root";
        String password = "...";
        Class.forName("com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection(url, user, password);
    } catch (ClassNotFoundException | SQLException ex) {
        Logger.getLogger(DBConnectionManager.class.getName()).log(Level.SEVERE, null,
ex);
    }
}

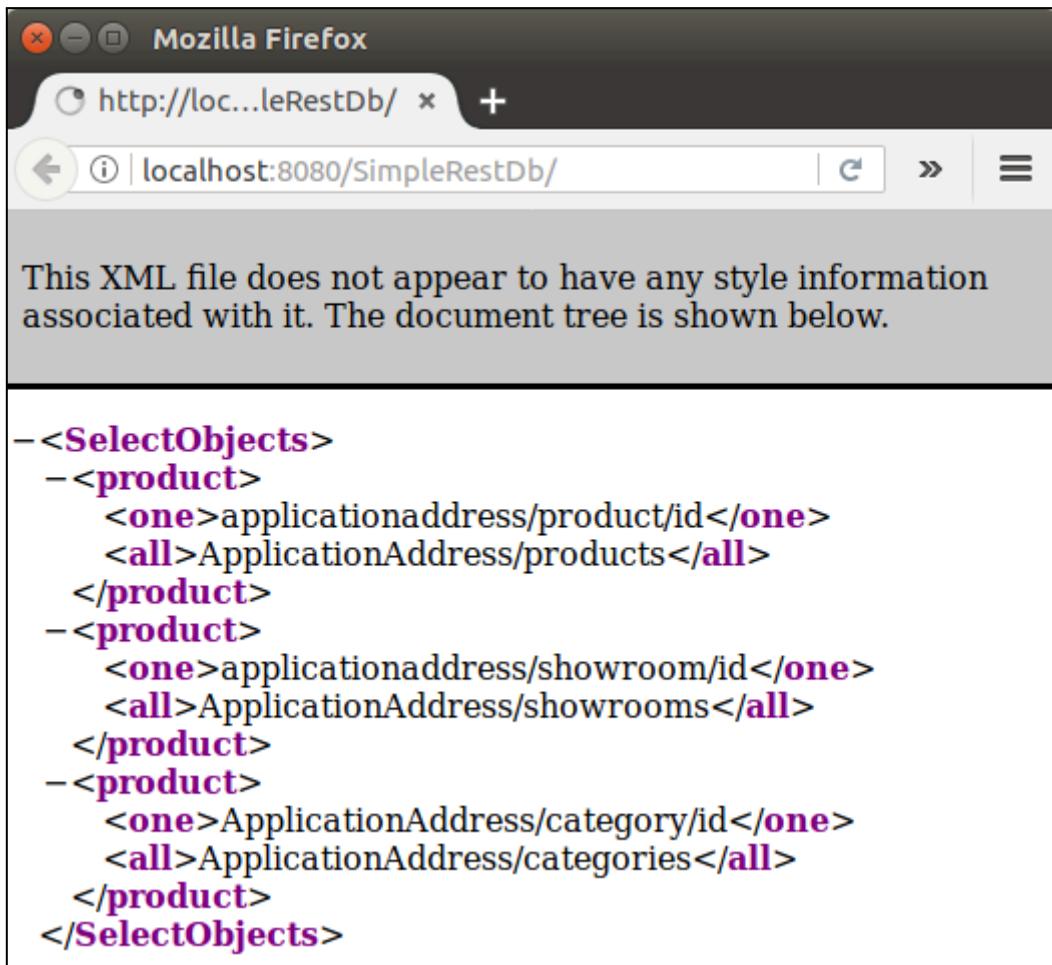
/**
 * Returns the db connection
 *
 * @return db connection
 */
public Connection getConnection() {
    return connection;
}

public void disconnect() throws SQLException {
    connection.close();
}
}
```

## ShowRoom REST su MySQL

Il seguente esempio mostra alcune funzionalità di un Web service REST che manipola i dati di un database di prodotti venduti presso diversi negozi appartenenti ad una catena commerciale e i prodotti sono organizzati per categorie. Il Web service permette di interrogare il database utilizzando esclusivamente i metodi HTTP e un URL che identifica la risorsa, come richiesto dall'architettura REST.

Il **server SimpleRestDb** è stato creato in modo tale da presentare all'avvio le modalità di costruzione degli URL per poter effettuare una richiesta di visualizzazione del contenuto di una tabella del database *productsales*. Questa leggenda, fornita in formato XML, permette di effettuare la visualizzazione dei contenuti delle singole tabelle del database, sia globalmente, sia in modo parametrico usando un identificatore.



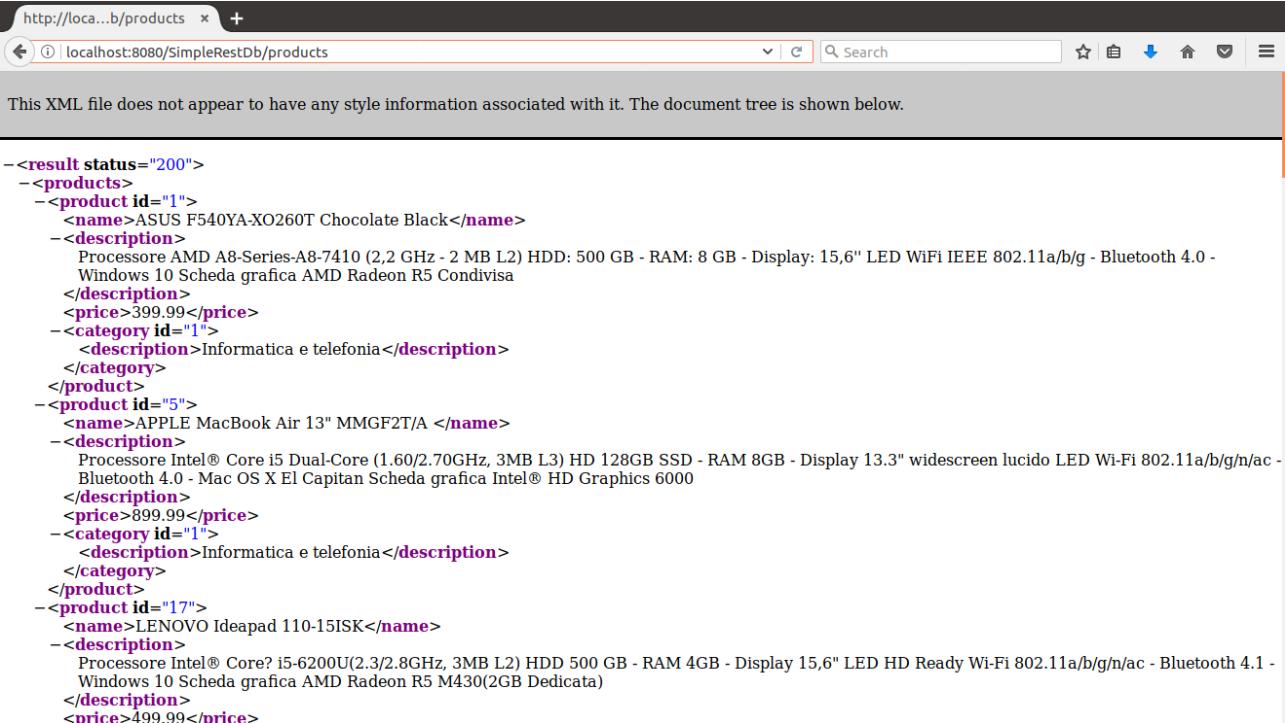
The screenshot shows a Mozilla Firefox browser window. The address bar displays "localhost:8080/SimpleRestDb/". The main content area shows an XML document tree. At the top, a message reads: "This XML file does not appear to have any style information associated with it. The document tree is shown below." Below this message, the XML structure is displayed:

```
--<SelectObjects>
--<product>
  <one>applicationaddress/product/id</one>
  <all>ApplicationAddress/products</all>
</product>
--<product>
  <one>applicationaddress/showroom/id</one>
  <all>ApplicationAddress/showrooms</all>
</product>
--<product>
  <one>ApplicationAddress/category/id</one>
  <all>ApplicationAddress/categories</all>
</product>
</SelectObjects>
```

Ad esempio, per visualizzare tutti i prodotti presenti nella tabella *Products* del database *productsales* si deve utilizzare l'URL:

<http://localhost:8080/SimpleRestDb/products>

Il metodo GET invocato con questo URL viene gestito dal Web service tramite il metodo *doGet()* che, effettuando un'analisi dell'URL, seleziona la visualizzazione di tutti i prodotti. Nell'immagine seguente viene riportata una porzione della risposta XML restituita dal Web service.



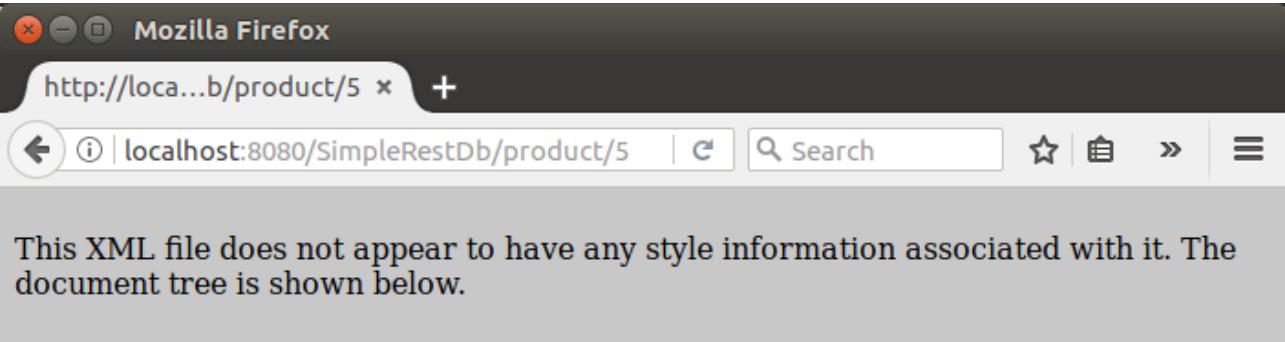
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
--<result status="200">
-<products>
--<product id="1">
<name>ASUS F540YA-XO260T Chocolate Black</name>
--<description>
    Processore AMD A8-Series-A8-7410 (2,2 GHz - 2 MB L2) HDD: 500 GB - RAM: 8 GB - Display: 15,6" LED WiFi IEEE 802.11a/b/g - Bluetooth 4.0 -
    Windows 10 Scheda grafica AMD Radeon R5 Condivisa
</description>
<price>399.99</price>
--<category id="1">
    <description>Informatica e telefonia</description>
</category>
</product>
--<product id="5">
<name>APPLE MacBook Air 13" MMGF2T/A </name>
--<description>
    Processore Intel® Core i5 Dual-Core (1.60/2.70GHz, 3MB L3) HD 128GB SSD - RAM 8GB - Display 13.3" widescreen lucido LED Wi-Fi 802.11a/b/g/n/ac -
    Bluetooth 4.0 - Mac OS X El Capitan Scheda grafica Intel® HD Graphics 6000
</description>
<price>899.99</price>
--<category id="1">
    <description>Informatica e telefonia</description>
</category>
</product>
--<product id="17">
<name>LENOVO Ideapad 110-15ISK</name>
--<description>
    Processore Intel® Core? i5-6200U(2.3/2.8GHz, 3MB L2) HDD 500 GB - RAM 4GB - Display 15,6" LED HD Ready Wi-Fi 802.11a/b/g/n/ac - Bluetooth 4.1 -
    Windows 10 Scheda grafica AMD Radeon R5 M430(2GB Dedicata)
</description>
<price>499.99</price>
```

Invece, effettuando la richiesta di un singolo prodotto tramite il suo *id* utilizzando il seguente URL

<http://localhost:8080/SimpleRestDb/product/5>

si ottiene la visualizzazione del singolo prodotto, sempre in formato XML.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

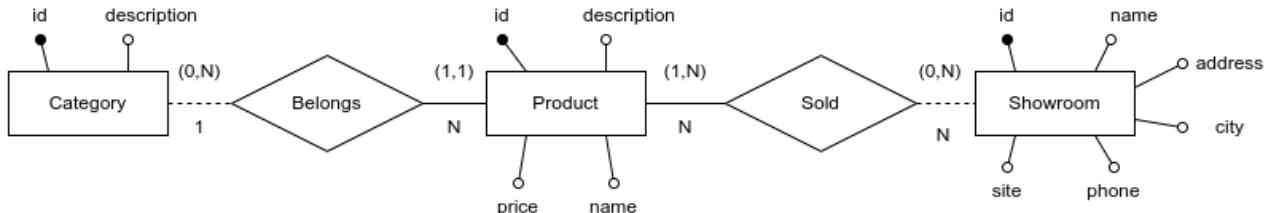
```
--<result status="200">
-<product id="5">
<name>APPLE MacBook Air 13" MMGF2T/A </name>
--<description>
    Processore Intel® Core i5 Dual-Core (1.60/2.70GHz, 3MB L3) HD 128GB SSD -
    - RAM 8GB - Display 13.3" widescreen lucido LED Wi-Fi 802.11a/b/g/n/ac -
    Bluetooth 4.0 - Mac OS X El Capitan Scheda grafica Intel® HD Graphics 6000
</description>
<price>899.99</price>
--<category id="1">
    <description>Informatica e telefonia</description>
</category>
</product>
</result>
```

Per testare le operazioni di aggiunta (CREATE - POST), aggiornamento (UPDATE - PUT) e cancellazione (DELETE - DELETE) di un'istanza, viene usato un semplice **client**, **RestClient**.

Nel seguito della trattazione viene fornita una possibile implementazione del database, del Web service e del client.

## Database productsales

Lo **schema concettuale**, semplificando la realtà gestita, è il seguente:



Analizzando lo schema E-R (concettuale) si deduce:

- un prodotto dell'entità *Product* è presente in una sola categoria dell'entità *Categorie* e l'occorrenza è obbligatoria, cioè un prodotto non può essere privo di categoria;
- una categoria dell'entità *Categorie* può contenere più prodotti, ma vista la cardinalità minima pari a zero di partecipazione all'associazione *Belongs*, una categoria potrebbe anche non avere prodotti, ad esempio quando una categoria è stata appena creata e nessun prodotto è stato ancora associato alla categoria;
- un prodotto dell'entità *Product* è venduto almeno in un negozio dell'entità *Showroom* e l'occorrenza è obbligatoria, cioè un prodotto deve essere venduto da almeno un negozio;
- un negozio dell'entità *Showroom* può vendere più prodotti, ma vista la cardinalità minima pari a zero di partecipazione all'associazione *Exhibited*, un negozio potrebbe anche non avere prodotti in vendita, ad esempio quando un negozio è stato appena aperto e nessun prodotto è stato ancora associato al negozio.

Lo **schema logico** è il seguente:

Categories([id](#), description)

Products([id](#), name, description, price, category\*)

Showrooms([id](#), name, address, city, phone, site)

Exhibited([product](#)\*, [showroom](#)\*)

Il **DDL** e **DML** della relativa **progettazione fisica** per la creazione del database, delle tabelle e della loro popolazione, è il seguente:

```

-- phpMyAdmin SQL Dump
-- version 4.5.4.1deb2ubuntu2
-- http://www.phpmyadmin.net
--
-- Host: localhost
  
```

```
-- Generation Time: Mar 28, 2017 at 09:59 PM
-- Server version: 5.7.17-0ubuntu0.16.04.1
-- PHP Version: 7.0.15-0ubuntu0.16.04.4

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--

-- Database: `productsales`
--

CREATE DATABASE IF NOT EXISTS `productsales` DEFAULT CHARACTER SET latin1 COLLATE
latin1_swedish_ci;
USE `productsales`;

-----


-- Table structure for table `Categories`


CREATE TABLE `Categories` (
  `id` int(11) NOT NULL,
  `description` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- RELATIONS FOR TABLE `Categories`:


-- Dumping data for table `Categories`


INSERT INTO `Categories` (`id`, `description`) VALUES
(1, 'Informatica e telefonia'),
(2, 'TV e audio'),
(3, 'Fotografia, Auto e Navi'),
(4, 'Grandi Elettrodomestici'),
(5, 'Piccoli Elettrodomestici'),
(6, 'Gaming, Musica e Film'),
(7, 'Innovazione'),
(8, 'Tempo Libero'),
(9, 'Outlet');
```

```
--  
-- Table structure for table `Products`  
  
CREATE TABLE `Products` (  
  `id` int(11) NOT NULL,  
  `name` varchar(50) NOT NULL,  
  `description` varchar(500) DEFAULT NULL,  
  `price` float(5,2) NOT NULL,  
  `category` int(11) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
  
--  
-- RELATIONS FOR TABLE `Products`:  
--   `category`  
--   `Categories` -> `id`  
  
--  
  
-- Dumping data for table `Products`  
  
--  
  
INSERT INTO `Products` (`id`, `name`, `description`, `price`, `category`) VALUES  
(1, 'ASUS F540YA-X0260T Chocolate Black', 'Processore AMD A8-Series-A8-7410 (2,2 GHz  
- 2 MB L2)\r\nHDD: 500 GB - RAM: 8 GB - Display: 15,6\'\' LED\r\nWiFi IEEE  
802.11a/b/g - Bluetooth 4.0 - Windows 10\r\nScheda grafica AMD Radeon R5 Condivisa',  
399.99, 1),  
(2, 'EPSON V11H664040 ', 'Videoproiettore con Tecnologia 3LCD\r\nContrasto di  
immagine: 15.000:1\r\nLuminosità 3000 Ansi Lumen\r\nRisoluzione: 1280x800  
pixel\r\nImmagini brillanti in 2D e 3D', 449.99, 2),  
(3, 'HOTPOINT FMG 723 B IT.M ', 'Lavatrice - Carica: Frontale - Tipo installazione:  
Libera installazione (FS) - Classe energetica: A +++ - Capacità max carico: 7 kg -  
Velocità max centrifuga: 1.200 giri/min - Profondità: 52,2 cm', 279.99, 4),  
(4, 'HP 15-AC195NL - PRMG GRADING OOAN - SCONT 10,00% ', 'Processore Intel® Core™  
i7-5500U(2,4/3GHz - 4MB L3)\r\nHDD 1000 GB - RAM 8GB - Display 15,6" WLED HD  
Ready\r\nWi-Fi 802.11b/g/n - Windows 10\r\nScheda grafica AMD Radeon R5 M330(2GB  
Dedicata)\r\nPRODOTTO RICONDIZIONATO - PRMG GRADING OOAN\r\nConfezione originale  
integra (0)\r\nAccessori principali presenti (0)\r\nEstetica prodotto come nuovo  
(A)\r\nStato prodotto funzionante (N)', 719.99, 9),  
(5, 'APPLE MacBook Air 13" MMGF2T/A ', 'Processore Intel® Core i5 Dual-Core  
(1.60/2.70GHz, 3MB L3)\r\nHD 128GB SSD - RAM 8GB - Display 13.3" widescreen lucido  
LED\r\nWi-Fi 802.11a/b/g/n/ac - Bluetooth 4.0 - Mac OS X El Capitan\r\nScheda  
grafica Intel® HD Graphics 6000', 899.99, 1),  
(6, 'BRONDI FX-Compact Sport S ', 'Frequenza PMR - 8 Canali\r\n38 Codici per canali  
- Funzione VOX\r\nFunziona con 3 ministilo alcaline\r\nComunica con tutte le  
ricetrasmettenti PMR446', 19.99, 8),
```

(7, 'MAJESTIC SD 247 RDS USB AX ', 'Digital Media Receiver Mechaless - Potenza 4x30W\r\nLettura MP3, WMA - Display ID3 Tag\r\nSintonizzatore RDS\r\nIngresso AUX - USB - SD/MMC Card', 34.99, 3),  
(8, 'SONY KD43XD8099B ', 'SMART TV LED 43" Ultra HD 4K - Risoluzione: 3840x2160\r\nTecnologia 400Hz - DLNA - Wi-fi\r\nTuner Digitale Terrestre DVB-T2 HEVC e Satellitare DVB-S\r\nProcessore Video 4K X-Reality Pro\r\nClasse efficienza energetica: B\r\nDistribuito da Sony Italia', 749.99, 2),  
(9, 'PANASONIC RP-WF830E-K ', 'Cuffie Stereo Wireless\r\nRisposta in frequenza 18-22.000 Hz\r\nDistanza di trasmissione fino a 100 m', 54.99, 2),  
(16, 'BOSCH PCQ715B90E ', 'Piano cottura - 5 Fuochi gas - Accensione ad 1 mano\r\nTermosicurezza - 2 Griglie in ghisa - Comandi frontali\r\nLarghezza: 70 cm', 299.99, 4),  
(17, 'LENOVO Ideapad 110-15ISK', 'Processore Intel® Core™ i5-6200U(2.3/2.8GHz, 3MB L2)\r\nHDD 500 GB - RAM 4GB - Display 15,6" LED HD Ready\r\nWi-Fi 802.11a/b/g/n/ac - Bluetooth 4.1 - Windows 10\r\nScheda grafica AMD Radeon R5 M430(2GB Dedicata)', 499.99, 1),  
(18, 'CANON EOS 1300D 18-55 IS', 'Fotocamera Reflex digitale - Sensore CMOS da 18 Megapixel\r\nLCD da 3" - Filmati Full HD - Slot SD/SDHC\r\nObiettivo 18-55 IS - Interfaccia USB - Peso: 485 g.\r\nGaranzia ufficiale Canon Italia', 379.99, 3),  
(19, 'KOENIC KSI 270 ', 'Ferro a vapore - Potenza max: 2.700 W\r\nCapacità serbatoio: 350 ml - Regolazione\r\nvapore - Funzione spray - Spegnimento\r\nnautomatico - Piastra: Ceramica rivestita', 25.99, 5),  
(21, 'MT DISTRIBUTION TEKKDRONE VAMPIRE PLUS', 'Minidrone - Camera HD da 2 megapixel\r\nGiroscopio 6 assi\r\nLuci LED per utilizzo notturno\r\nAutonomia 14/16 minuti\r\nDistanza di controllo 100 metri\r\nFunzione hovering - headless mode\r\nTasto one key return ', 89.99, 7),  
(22, 'SAMSUNG UE55KU6000KXZT', 'SMART TV LED 55" Ultra HD 4K - Risoluzione: 3840x2160\r\nTecnologia 1300 PQI - DLNA - WiFi + Ethernet\r\nTuner Digitale Terrestre DVB-T2\r\nClasse efficienza energetica: A\r\nDistribuito da Samsung Italia', 699.99, 2),  
(23, 'SONY PS4 PRO 1TB ', 'Console fissa - 1000 GB HDD\r\nLettore BD/ DVD - Controller wireless\r\nBluetooth - HDMI - USB', 409.99, 6),  
(24, 'BEKO RDSA240K20W', 'Frigorifero Doppia porta\r\nClasse di efficienza energetica: A\r\nCapacità netta frigo: 177 l\r\nCapacità netta congelatore: 46 l\r\nConsumo energia annuo: 226 kWh/anno\r\nSistema di raffreddamento frigo: Statico\r\nColore: Bianco', 259.99, 4),  
(25, 'DE LONGHI EDG 100.W + 48 Capsule ', 'Macchina per caffè - Pressione: 15 bar - Potenza: 1460 W\r\nLeva di erogazione - Stand-by automatico - Sistema\r\nThermoblock - Cassetto raccogligocce regolabile in altezza\r\nFunzionamento esclusivo con cialde Nescafè Dolce Gusto', 49.99, 5);

---

--  
-- Table structure for table `Exhibited`  
--

```
CREATE TABLE `Exhibited` (  
  `product` int(11) NOT NULL,
```

```
`showroom` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- 
-- RELATIONS FOR TABLE `Exhibited`:
--   `product`
--     `Products` -> `id`
--   `showroom`
--     `Showrooms` -> `id`
-- 

-- 
-- Dumping data for table `ProductsShowrooms`
-- 

INSERT INTO `Exhibited` (`product`, `showroom`) VALUES
(1, 1),(5, 1),(6, 1),(7, 1),(9, 1),(1, 2),(2, 2),(6, 2),(7, 2),(9, 2),(1, 3),(2, 3),(3, 3),(7, 3),(9, 3),(2, 4),(3, 4),(4, 4),(8, 4),(3, 5),(4, 5),(5, 5),(8, 5),(4, 6),(5, 6),(6, 6),(8, 6);

-- -----
-- 
-- Table structure for table `Showrooms`
-- 

CREATE TABLE `Showrooms` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `address` varchar(100) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(12) DEFAULT NULL,
  `site` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- 
-- RELATIONS FOR TABLE `Showrooms`:
-- 

-- 
-- Dumping data for table `Showrooms`
-- 

INSERT INTO `Showrooms` (`id`, `name`, `address`, `city`, `phone`, `site`) VALUES
(1, 'show1', 'address1', 'city1', '123456781', 'http://www.show1.com'),
(2, 'show2', 'address2', 'city2', '123456782', 'http://www.show2.com'),
(3, 'show3', 'address3', 'city3', '123456783', 'http://www.show3.com'),
(4, 'show4', 'address4', 'city4', '123456784', NULL),
(5, 'show5', 'address5', 'city5', NULL, 'http://www.show5.com'),
```

```
(6, 'show6', 'address6', 'city6', '123456786', 'http://www.show6.com');

-- 
-- Indexes for dumped tables
-- 

-- 
-- Indexes for table `Categories`
-- 

ALTER TABLE `Categories`
  ADD PRIMARY KEY (`id`);

-- 
-- Indexes for table `Products`
-- 

ALTER TABLE `Products`
  ADD PRIMARY KEY (`id`),
  ADD KEY `id_category` (`category`);

-- 
-- Indexes for table `ProductsShowrooms`
-- 

ALTER TABLE `Exhibited`
  ADD PRIMARY KEY (`product`, `showroom`),
  ADD KEY `showroom` (`showroom`);

-- 
-- Indexes for table `Showrooms`
-- 

ALTER TABLE `Showrooms`
  ADD PRIMARY KEY (`id`);

-- 
-- AUTO_INCREMENT for dumped tables
-- 

-- 
-- AUTO_INCREMENT for table `Categories`
-- 

ALTER TABLE `Categories`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;

-- 
-- AUTO_INCREMENT for table `Products`
-- 

ALTER TABLE `Products`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=26;

-- 
-- AUTO_INCREMENT for table `Showrooms`
-- 
```

```
ALTER TABLE `Showrooms`  
    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;  
--  
-- Constraints for dumped tables  
--  
  
--  
-- Constraints for table `Products`  
--  
ALTER TABLE `Products`  
    ADD CONSTRAINT `Products_ibfk_1` FOREIGN KEY (`category`) REFERENCES `Categories`(`id`);  
  
--  
-- Constraints for table `Exhibited`  
--  
ALTER TABLE `Exhibited`  
    ADD CONSTRAINT `Exhibited_ibfk_1` FOREIGN KEY (`product`) REFERENCES `Products`(`id`),  
    ADD CONSTRAINT `Exhibited_ibfk_2` FOREIGN KEY (`showroom`) REFERENCES `Showrooms`(`id`);  
  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

## Server di connessione al database productsales

Nel server per la gestione della connessione al database viene definita un'interfaccia per specificare i metodi utilizzabili per la conversione dei dati del database in formato XML o JSON (quest'ultimo non è implementato). I due metodi dichiarati saranno poi definiti nelle classi relative alle tre entità del database. Il server implementa diversi metodi per l'interrogazione del database *productsales*.

L'intera implementazione del server di gestione del database viene inserita come libreria nel Web service che offre il servizio, il cui codice è mostrato nel seguito della trattazione.

### **Bointerface.java**

```
package simple.bo;  
  
public interface BoInterface {  
    public String toXml();  
    public String toJson();  
}
```

Di seguito vengono definite le classi per manipolare i dati del database come oggetti.

### **Category.java**

La classe **Category** sarà usata per creare oggetti derivanti dalla tabella **Categories** che contiene le diverse categorie dei prodotti.

```
package simple.bo;

public class Category implements BoInterface {

    private int id = -1;
    private String description;

    public Category(int id, String description) {
        this.id = id;
        this.description = description;
    }

    public Category(String description) {
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 59 * hash + this.id;
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Category other = (Category) obj;
        if (this.id != other.id) {
```

```
        return false;
    }
    return true;
}

// Restituisce un elemento XML
public String toXml() {
    return "<category id=\"" + id + "\"><description>" + description
           + "</description></category>";
}

// Restituisce un elemento Json
public String toJson() {
    return "{\"category\":{\n\"id\": \"" + id + "\",\n\"description\":"
           + description + "\"\n}}\n";
}
}
```

## Product.java

La classe **Product** sarà usata per creare oggetti derivanti dalla tabella **Products** che contiene tutti i prodotti gestiti, ognuno appartenente ad una specifica categoria.

```
package simple.bo;

public class Product implements BoInterface {

    private int id = -1;
    private String name;
    private String description = null;
    private float price;
    private Category category;

    // Costruttore di un prodotto.
    // L'attributo che conterrà la categoria è un riferimento all'oggetto che
    // identifica la categoria stessa, non un identificatore, come avviene nel
    // db. La rappresentazione dei dati usando un linguaggio ad oggetti, non
    // necessariamente deve essere riprodotta esattamente. In questo caso è stato
    // scelto di memorizzare direttamente il riferimento alla categoria invece di
    // salvarne il codice.
    public Product(int id, String name, String description, float price, Category
category) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.description = description;
        this.category = category;
    }
}
```

```
public Product(String name, String description, float price, Category category) {
    this.name = name;
    this.description = description;
    this.price = price;
    this.category = category;
}

public int getId() {
    return id;
}

public String getName() {
    return name;
}

public String getDescription() {
    return description;
}

public float getPrice() {
    return price;
}

public Category getCategory() {
    return category;
}

public void setDescription(String description) {
    this.description = description;
}

public void setPrice(float price) {
    this.price = price;
}

public void setCategory(Category category) {
    this.category = category;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 71 * hash + this.id;
    return hash;
}

@Override
public boolean equals(Object obj) {
```

```
if (obj == null) {
    return false;
}
if (getClass() != obj.getClass()) {
    return false;
}
final Product other = (Product) obj;
if (this.id != other.id) {
    return false;
}
return true;
}

// Restituisce un elemento XML
public String toXml() {
    String s = "<product id=\"" + id + "\"><name>" + name + "</name>";
    s += "<description>" + description + "</description>";
    s += "<price>" + price + "</price>";
    s += category.toXml() + "</product>";
    return s;
}

// Restituisce un elemento Json
public String toJson() {
    String s = "{\n\"product\"\n{id\":\"" + id + "\",\n";
    s += "\"name\":\"" + name + "\",\n";
    s += "\"description\":\"" + description + "\",\n";
    s += "\"price\":\"" + price + "\",\n";
    s += category.toJson();
    s += "\n}\n";
    return s;
}
}
```

## Showroom.java

La classe **Showroom** sarà usata per creare oggetti derivanti dalla tabella **Showrooms** che contiene tutti i negozi della catena.

```
package simple.bo;

public class Showroom implements BoInterface {

    private int id = -1;
    private String name;
    private String address;
    private String city;
    private String phone;
    private String site;
```

```
public Showroom(int id, String name, String address, String city) {  
    this.id = id;  
    this.name = name;  
    this.address = address;  
    this.city = city;  
}  
  
public Showroom(String name, String address, String city, String phone, String  
site) {  
    this.name = name;  
    this.address = address;  
    this.city = city;  
    this.phone = phone;  
    this.site = site;  
}  
  
public int getId() {  
    return id;  
}  
  
public String getName() {  
    return name;  
}  
  
public String getAddress() {  
    return address;  
}  
  
public String getCity() {  
    return city;  
}  
  
public String getPhone() {  
    return phone;  
}  
  
public String getSite() {  
    return site;  
}  
  
public void setPhone(String phone) {  
    this.phone = phone;  
}  
  
public void setSite(String site) {  
    this.site = site;  
}  
  
@Override
```

```
public int hashCode() {
    int hash = 3;
    hash = 79 * hash + this.id;
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Showroom other = (Showroom) obj;
    if (this.id != other.id) {
        return false;
    }
    return true;
}

// Restituisce un elemento XML
public String toXml() {
    String s = "<showroom id=\"" + id + "\"><name>" + name + "</name>";
    s += "<address>" + address + "</address><city>" + city + "</city>";
    if (phone != null) {
        s += "<phone>" + phone + "</phone>";
    }
    if (site != null) {
        s += "<site>" + site + "</site>";
    }
    s += "</showroom>";
    return s;
}

// Restituisce un elemento Json
@Override
public String toJson() {
    throw new UnsupportedOperationException("Not supported yet.");
}
}
```

## Server.java

Nel server mancano le interrogazioni al database:

- prodotto per categoria;
- negozi che vendono un prodotto;
- tutti i prodotti venduti da un negozio.

Provare ad implementarli come esercizio.

```
package simple.db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import simple.bo.Category;
import simple.bo.Product;
import simple.bo.Showroom;

/**
 *
 * @author palma
 * Mancano le interrogazioni al database di:
 * - Prodotto per Categoria
 * - Negozi che vendono un prodotto
 * - Tutti i prodotti venduti da un negozio
 */
public class Server {
    // La classe Connection permette di connettersi ad un database.
    Connection connection;
    // La classe Statement permette di eseguire una query al database.
    Statement statement;
    // La classe ResultSet contiene il risultato della query effettuata sul
    // database.
    ResultSet resultSet;

    /**
     * Il costruttore prende come parametro formale la stringa di connessione
     * al database.
     */
    public Server(String connectionString) {
        try {
            // La classe DriverManager è il servizio di base per la gestione
            // dei driver JDBC, mentre il metodo getConnection() permette di
            // creare una connessione al database identificato dalla stringa
            // fornita come parametro attuale.
            connection = DriverManager.getConnection(connectionString);
            // Il metodo createStatement() crea un oggetto Statement utilizzato
            // per inviare istruzioni SQL al database.
            statement = connection.createStatement();
        } catch (SQLException ex) {
            throw new RuntimeException("Impossibile connettersi al database");
        }
    }
}
```

```
    }

}

/***
 * Il metodo memorizza in un ArrayList l'elenco di tutte le categorie
 * presenti nella tabella Categories.
 * @return elenco di oggetti Category
 */
public List<Category> selectCategories() {
    List<Category> list = new ArrayList();
    try {
        // Il metodo executeQuery() dell'interfaccia Statement esegue una
        // istruzione SQL fornita come parametro attuale, e restituisce un
        // singolo oggetto ResultSet.
        resultSet = statement.executeQuery("select * from Categories");
        while (resultSet.next()) {
            // I metodi getInt() e getString() dell'interfaccia ResultSet
            // recuperano il valore nella colonna passata come parametro
            // attuale ai metodi, dell'attuale riga dell'oggetto ResultSet.
            // Il metodo getInt() trasforma il valore in un intero, mentre
            // il metodo getString() lo trasforma in una stringa.
            // Il parametro passato nei metodi può essere un numero o una
            // stringa. Il primo identifica la posizione della colonna, il
            // secondo l'attributo della tabella nel database.
            list.add((Category) (new Category(resultSet.getInt(1),
                                              resultSet.getString(2))));
        }
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return list;
}

/***
 * Il metodo memorizza in un ArrayList l'elenco di tutti i negozi
 * presenti nella tabella Showrooms.
 * @return elenco di oggetti Showroom
 */
public List<Showroom> selectShowrooms() {
    List<Showroom> list = new ArrayList();
    Showroom ss = null;
    try {
        resultSet = statement.executeQuery("select * from Showrooms");
        while (resultSet.next()) {
            ss = new Showroom(resultSet.getInt(1), resultSet.getString(2),
                             resultSet.getString(3), resultSet.getString(4));
            ss.setPhone(resultSet.getString(5));
            ss.setSite(resultSet.getString(6));
            list.add((Showroom) ss);
        }
    }
```

```
        }
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return list;
}

/** 
 * Il metodo memorizza in un ArrayList l'elenco di tutti i prodotti
 * presenti nella tabella Products, ognuno associato alla relativa
 * categoria.
 * @return elenco di oggetti Product
 */
public List<Product> selectProducts() {
    List<Product> list = new ArrayList();
    Category sc = null;
    Product sp = null;
    try {
        resultSet = statement.executeQuery("select * from Products, "
                + "Categories where id_category=Categories.id");
        while (resultSet.next()) {
            if (sc == null || sc.getId() != resultSet.getInt(6)) {
                sc = new Category(resultSet.getInt(6), resultSet.getString(7));
            }
            sp = new Product(resultSet.getInt(1), resultSet.getString(2),
                    resultSet.getString(3), resultSet.getFloat(4), (Category)
sc);
            list.add((Product) sp);
        }
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return list;
}

/** 
 * Il metodo seleziona dal database l'istanza di Categories identificata
 * dalla chiave primaria passata come parametro al metodo.
 * @param id chiave primaria che identifica l'istanza nella tabella
 * Categories.
 * @return l'oggetto Category con i dati estratti dal database.
 */
public Category selectCategory(int id) {
    Category c = null;
    try {
        resultSet = statement.executeQuery("select * from Categories where id="
+ id);
    }
```

```
        if (resultSet.next()) {
            c = (Category) (new Category(resultSet.getInt(1),
resultSet.getString(2)));
        }
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return c;
}

/***
 * Il metodo seleziona dal database l'istanza di Showroom identificata
 * dalla chiave primaria passata come parametro al metodo.
 * @param id chiave primaria che identifica l'istanza nella tabella
 * Showrooms.
 * @return l'oggetto Showroom con i dati estratti dal database.
 */
public Showroom selectShowroom(int id) {
    Showroom ss = null;
    try {
        resultSet = statement.executeQuery("select * from Showrooms where id=" +
id);
        if (resultSet.next()) {
            ss = new Showroom(resultSet.getInt(1), resultSet.getString(2),
resultSet.getString(3), resultSet.getString(4));
            ss.setPhone(resultSet.getString(5));
            ss.setSite(resultSet.getString(6));
        }
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return (Showroom) ss;
}

/***
 * Il metodo seleziona dal database l'istanza di Products identificata
 * dalla chiave primaria passata come parametro al metodo.
 * @param id chiave primaria che identifica l'istanza nella tabella
 * Products.
 * @return l'oggetto Product con i dati estratti dal database.
 */
public Product selectProduct(int id) {
    Category sc = null;
    Product sp = null;
    try {
        resultSet = statement.executeQuery("select * from Products, Categories "
                + "where id_category=Categories.id and Products.id=" + id);
        if (resultSet.next()) {
            sc = new Category(resultSet.getInt(6), resultSet.getString(7));
        }
    }
```

```
        sp = new Product(resultSet.getInt(1), resultSet.getString(2),
                          resultSet.getString(3), resultSet.getFloat(4), (Category)
sc);
    }
} catch (SQLException ex) {
    throw new RuntimeException("Errore nell'esecuzione della query");
}
return (Product) sp;
}

public int deleteProduct(Product p){
    return deleteProduct(p.getId());
}

public int deleteProduct(int id) {
    int n = 0;
    try {
        n = statement.executeUpdate("delete from Products where id=" + id);
        statement.executeUpdate("delete from ProductsShowrooms where
id_product=" + id);
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return n;
}

public int deleteShowroom(Showroom s){
    return deleteShowroom(s.getId());
}

public int deleteShowroom(int id) {
    int n = 0;
    try {
        n = statement.executeUpdate("delete from Showroom where id=" + id);
        statement.executeUpdate("delete from ProductsShowrooms where
id_showroom=" + id);
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return n;
}

public int deleteCategory(Category c){
    return deleteCategory(c.getId());
}

public int deleteCategory(int id) {
    int n = 0;
    try {
```

```
        statement.executeUpdate("delete from Products "
            + " where id_category=" + id);
        n = statement.executeUpdate("delete from Categories where id=" + id);
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
    return n;
}

public int updateCategory(Category c) {
    try {
        return statement.executeUpdate("update Categories set description='"
            + c.getDescription() + '' where id=" + c.getId());
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
}

public int updateShowroom(Showroom s) {
    try {
        return statement.executeUpdate("update Showrooms set phone='"
            + s.getPhone() + ',', site=''' + s.getSite() + '' where id=" +
s.getId());
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
}

public int updateProduct(Product p) {
    try {
        return statement.executeUpdate("update Products set description='"
            + p.getDescription() + ', price=' + p.getPrice() + ',
id_category='
            + p.getCategory().getId() + " where id=" + p.getId());
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
}

public int insertProduct(Product p) {
    try {
        return statement.executeUpdate("insert into Products (name,description"
            + ",price,id_category) values ('" + p.getName() + "','" +
p.getDescription()
            + "','" + p.getPrice() + "','" + p.getCategory().getId() + ")");
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
}
```

```
public int insertShowroom(Showroom s) {
    try {
        return statement.executeUpdate("insert into Showrooms (name,address"
            + ",city,phone,site) values ('" + s.getName() + "','" +
s.getAddress()
            + "','" + s.getCity() + "','" + s.getPhone() + "','" +
s.getSite() + "')");
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
}

public int insertCategory(Category c) {
    try {
        return statement.executeUpdate("insert into Categories (description)"
            + " values ('" + c.getDescription() + "')");
    } catch (SQLException ex) {
        throw new RuntimeException("Errore nell'esecuzione della query");
    }
}
}
```

## Web service REST SimpleRestDb

### web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
<servlet>
    <servlet-name>Crud</servlet-name>
    <servlet-class>simplerestdb.Crud</servlet-class>
    <init-param>
        <param-name>connectionString</param-name>
        <param-value>
            jdbc:mysql://localhost:3306/EsempioRest?user=root&password=password
        </param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>Crud</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>
```

30

```
</session-timeout>
</session-config>
</web-app>
```

## Crud.java

```
package simplerestdb;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import parser.CategoryParser;
import parser.ProductParser;
import simple.bo.Category;
import simple.bo.Product;
import simple.bo.Showroom;
import simple.db.Server;

/**
 *
 * @author palma
 * Manca il parser degli Showroom, implementatelo per esercizio.
 */
public class Crud extends HttpServlet {

    private Server server;

    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("application/xml;charset=UTF-8");
        String[] parts = request.getRequestURI().split("/");
        if (parts != null && (parts.length == 4 || parts.length == 3
                || parts.length == 2)) {
```

```
if (parts.length == 4) {
    selectOne(parts, out, response);
} else {
    if (parts.length == 3) {
        selectAll(parts, out, response);
    } else {
        instructions(out);
    }
}
} else {
    response.sendError(400);
}
}

@Override
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
insertOrUpdate(request, response, 1);
}

@Override
protected void doPut(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
insertOrUpdate(request, response, 2);
}

@Override
protected void doDelete(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
String[] parts = request.getRequestURI().split("/");
if (parts != null && parts.length == 4) {
    String typeOfObject = parts[2];
    String idObject = parts[3];
    if (idObject != null) {
        int n = -1;
        try {
            PrintWriter out = response.getWriter();
            switch (typeOfObject) {
                case "product":
                    n = server.deleteProduct(Integer.parseInt(idObject));
                    break;
                case "category":
                    n = server.deleteCategory(Integer.parseInt(idObject));
                    break;
                case "showroom":
                    n = server.deleteShowroom(Integer.parseInt(idObject));
                    break;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
```

```
        break;
    }
    out.println("<result status=\"200\">" + n
               + " record</result>");
} catch (IllegalArgumentException ex) {
    response.sendError(400);
}
} else {
    response.sendError(400);
}
}

private void selectAll(String[] parts, PrintWriter out,
                      HttpServletResponse response)
throws IOException, IllegalArgumentException {
String typeOfObject = parts[2];
switch (typeOfObject) {
    case "products":
        List<Product> products = server.selectProducts();
        out.println("<result status=\"200\"");
        out.println("<" + typeOfObject + ">");
        for (Product product : products) {
            out.println(product.toXml());
        }
        out.println("</" + typeOfObject + ">");
        out.println("</result>");
        break;
    case "categories":
        out.println("<result status=\"200\"");
        out.println("<" + typeOfObject + ">");
        List<Category> categories = server.selectCategories();
        for (Category product : categories) {
            out.println(product.toXml());
        }
        out.println("</" + typeOfObject + ">");
        out.println("</result>");
        break;
    case "showrooms":
        out.println("<result status=\"200\"");
        out.println("<" + typeOfObject + ">");
        List<Showroom> showrooms = server.selectShowrooms();
        for (Showroom showroom : showrooms) {
            out.println(showroom.toXml());
        }
        out.println("</" + typeOfObject + ">");
        out.println("</result>");
```

```
        break;
    default:response.sendError(400);
}

}

private void selectOne(String[] parts, PrintWriter out,
    HttpServletResponse response) throws IOException {
String typeOfObject = parts[2];
String idObject = parts[3];
if (idObject != null) {
    try {
        switch (typeOfObject) {
            case "product":
                Product product =
server.selectProduct(Integer.parseInt(idObject));
                if (product != null) {
                    out.println("<result status=\"200\">");
                    out.println(product.toXml());
                    out.println("</result>");
                } else{
                    out.println("<result status=\"200\"/>");
                }
                break;
            case "category":
                Category category =
server.selectCategory(Integer.parseInt(idObject));
                if (category != null) {
                    out.println("<result status=\"200\">");
                    out.println(category.toXml());
                    out.println("</result>");
                }else{
                    out.println("<result status=\"200\"/>");
                }
                break;
            case "showroom":
                Showroom showroom =
server.selectShowroom(Integer.parseInt(idObject));
                if (showroom != null) {
                    out.println("<result status=\"200\">");
                    out.println(showroom.toXml());
                    out.println("</result>");
                }else{
                    out.println("<result status=\"200\"/>");
                }
                break;
            default:response.sendError(400);
        }
    } catch (IllegalArgumentException ex) {
        response.sendError(400);
    }
}
```

```
        }
    } else {
        response.sendError(400);
    }
}

private void insertOrUpdate(HttpServletRequest request,
    HttpServletResponse response, int index) throws IOException {
String url = request.getRequestURI();
String[] parts = url.split("/");
if (parts != null && parts.length == 3) {
    String typeOfObject = parts[2];
    try {
        String input = "";
        BufferedReader br = new BufferedReader(request.getReader());
        String line;
        while ((line = br.readLine()) != null) {
            input += line;
        }
        if (!input.isEmpty()) {
            int n = -1;
            switch (typeOfObject) {
                case "product":
                    ProductParser productParser = new ProductParser(input,
true);
                    Product product = productParser.parse();
                    if (index == 1) {
                        n = server.insertProduct(product);
                    } else {
                        n = server.updateProduct(product);
                    }
                    break;
                case "category":
                    CategoryParser categoryParser = new
CategoryParser(input, true);
                    Category category = categoryParser.parse();
                    if (index == 1) {
                        n = server.insertCategory(category);
                    } else {
                        n = server.updateCategory(category);
                    }
                    break;
                case "showroom":
                    //da fare
                    break;
            }
        }
        PrintWriter out = response.getWriter();
        response.setContentType("application/xml;charset=UTF-8");
        out.println("<result status=\"200\">" + n + " record</result>");
    }
}
```

```
        } else {
            response.sendError(400);
        }
    } catch (Exception ex) {
        response.sendError(500, ex.getLocalizedMessage());
    }
} else {
    response.sendError(400);
}
}

@Override
public void init() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
        server = new Server(getInitParameter("connectionString"));
    } catch (ClassNotFoundException ex) {
        throw new RuntimeException(ex);
    }
}

/**
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
}

private void instructions(PrintWriter out) {
    out.println("<selectObjects>");
    out.println("<product>");
    out.println("<one>applicationAddress/product/id</one>");
    out.println("<all>applicationAddress/products</all>");
    out.println("</product>");
    out.println("<showroom>");
    out.println("<one>applicationaddress/showroom/id</one>");
    out.println("<all>applicationAddress/showrooms</all>");
    out.println("</showroom>");
    out.println("<category>");
    out.println("<one>ApplicationAddress/category/id</one>");
    out.println("<all>ApplicationAddress/categories</all>");
    out.println("</category>");
    out.println("</selectObjects>");
}
}
```

## CategoryParser.java (*commentare*)

```
package parser;

import java.io.ByteArrayInputStream;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import simple.bo.Category;

/**
 *
 * @author palma
 */
public class CategoryParser {

    private final boolean xml;
    private final String s;
    private Element root = null;

    public CategoryParser(String s, boolean xml) throws Exception {
        this.xml = xml;
        this.s = s;
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(new ByteArrayInputStream(s.getBytes()));
        root = document.getDocumentElement();
    }

    public Category parse() throws Exception {
        Category parsed = null;
        if (xml) {
            parsed = parseXml();
        } else {
            parsed = parseJson();
        }
        return parsed;
    }

    private Category parseXml() throws Exception {
        Category c = null;
        int id = -1;
        NodeList list = root.getElementsByTagName("category");
        if (list != null && list.getLength() > 0) {
            String description =

```

```
list.item(0).getFirstChild().getFirstChild().getNodeValue();
    if (description != null && !description.isEmpty()) {
        Node nId = list.item(0).getAttributes().getNamedItem("id");
        String sid = null;
        if (nId != null) {
            sid = nId.getNodeValue();
            try {
                id = Integer.parseInt(sid);
            } catch (NumberFormatException ex) {
                throw new Exception(ex);
            }
            c = new Category(id, description);
        } else {
            c = new Category(description);
        }
    } else {
        throw new Exception("XML error: incorrect description ");
    }
} else {
    if (root.getnodeName().equals("category")) {
        list = root.getElementsByTagName("description");
        if (list != null && list.getLength() > 0) {
            String description =
list.item(0).getFirstChild().getNodeValue();
            String sid = root.getAttribute("id");
            if (sid != null && !sid.isEmpty()) {
                try {
                    id = Integer.parseInt(sid);
                } catch (NumberFormatException ex) {
                    throw new Exception(ex);
                }
                c = new Category(id, description);
            } else {
                c = new Category(description);
            }
        } else {
            throw new Exception("XML error: no description");
        }
    } else {
        throw new Exception("XML error: no category");
    }
}
}

return c;
}

private Category parseJson() {
    throw new UnsupportedOperationException("Not supported yet.");
}
```

}

**ProductParser.java (commentare)**

```
package parser;

import java.io.ByteArrayInputStream;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import simple.bo.Category;
import simple.bo.Product;

/**
 *
 * @author palma
 */
public class ProductParser {

    private final boolean xml;
    private final String s;
    private Element root = null;

    public ProductParser(String s, boolean xml) throws Exception {
        this.xml = xml;
        this.s = s;
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(new
ByteArrayInputStream(s.getBytes()));
        root = document.getDocumentElement();
    }

    public Product parse() throws Exception {
        Product parsed = null;
        if (xml) {
            parsed = parseXml();
        } else {
            parsed = parseJson();
        }
        return parsed;
    }

    private Product parseXml() throws Exception {
        Product p = null;
        String name = null;
```

```
String description = null;
float price = 0;
int id = -1;

NodeList list = root.getElementsByTagName("name");
if (list != null && list.getLength() > 0) {
    name = list.item(0).getFirstChild().getNodeValue().trim();
    list = root.getElementsByTagName("description");
    if (list != null && list.getLength() > 0) {
        description =
list.item(0).getFirstChild().getNodeValue().trim();
    }
    list = root.getElementsByTagName("price");
    if (list != null && list.getLength() > 0) {
        price =
Float.parseFloat(list.item(0).getFirstChild().getNodeValue().trim());
        CategoryParser parser = new CategoryParser(s, true);
        Category category = (Category) parser.parse();
        String sid = root.getAttribute("id");
        if (sid != null && !sid.isEmpty()) {
            id = Integer.parseInt(sid);
            p = new Product(id, name, description, price, category);
        } else {
            p = new Product(name, description, price, category);
        }
        return p;
    } else {
        throw new Exception("L'XML error: incorrect price");
    }
} else {
    throw new Exception("XML error: incorrect name");
}
}

private Product parseJson() {
    throw new UnsupportedOperationException("Not supported yet.");
}
}
```

## ShowroomParser.java

Manca, da implementare per esercizio.

## Client RestClient

### RestClient.java

```
package restclient;
```

```
import java.awt.HeadlessException;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import javax.swing.JOptionPane;

/**
 *
 * @author palma
 */
public class RestClient {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Per lavorare sulle tre tabella
        String product="product";
        String showroom="showroom";
        String category="category";

        //modifica ha l'id
        String modifyProduct=<product id=\"20\"><name>xxxxx</name><description>
            + "cambio description</description><price>23.45</price>
            + "<category id=\"3\"><description>qqqqqq</description>
            + "</category></product>";

        String modifyShowroom=<showroom
id=\"4\"><name>cambiato</name><address>address 8<
            + "</address><city>umpapa</city><phone>12345</phone><site>
            + "www.puntopunto.it</site></showroom>";

        String modifyCategory=<category
id=\"2\"><description>modifica</description>
            + "</category>";

        //insert non ha id che viene messo dal db
        String insertProduct=<product><name>nuovo
prodotto</name><description>cambio<
            + " description</description><price>23.45</price>
            + "<category id=\"3\"><description>qqqqqq</description>
            + "</category></product>";

        String insertShowroom=<showroom><name>show8</name><address>address 8<
            + "</address><city>umpapa</city><phone>12345</phone><site>
            + "www.puntopunto.it</site></showroom>";

        String insertCategory=<category><description>nuova</description>
            + "</category>";
```

```
insertOrModify(category, modifyCategory);
//id dell'oggetto da cancellare
//verificare sul db gli id presenti
//delete(category, "3");
}

private static void insertOrModify(String table,String object) throws HeadlessException {
try{
    URL url = new URL("http://localhost:8080/SimpleRestDb/"+table);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestProperty("Accept", "application/xml");
    //conn.setRequestMethod("POST");
    conn.setRequestMethod("PUT");
    conn.setDoOutput(true);
    conn.setRequestProperty("Content-Type", "text/xml");
    conn.setRequestProperty("Connection", "Keep-Alive");
    conn.setRequestProperty("charset", "utf-8");
    PrintWriter out = new PrintWriter(conn.getOutputStream());
    out.println(object);
    out.flush();
    out.close();
    BufferedReader br = new BufferedReader(new InputStreamReader(
        (conn.getInputStream())));
    String output;
    String s = "";
    while ((output = br.readLine()) != null) {
        s += output;
    }
    System.out.println(s);
    conn.disconnect();

} catch (MalformedURLException ex) {
    JOptionPane.showMessageDialog(null, "Malformed url");
} catch (IOException ex) {
    JOptionPane.showMessageDialog(null, "Connection problem");
}
}

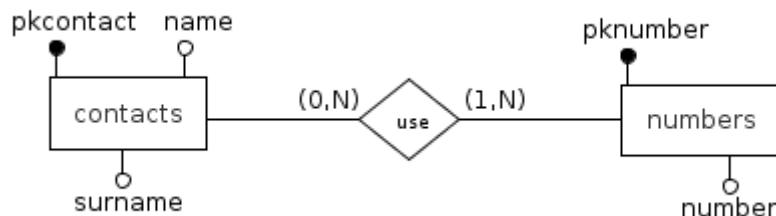
private static void delete(String object,String id) {
try{
    URL url = new URL("http://localhost:8080/RestDb/"+object+"/"+id);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestProperty("Accept", "application/xml");
    conn.setRequestMethod("DELETE");
    conn.setRequestProperty("Content-Type", "text/xml");
    conn.setRequestProperty("Connection", "Keep-Alive");
    conn.setRequestProperty("charset", "utf-8");
    BufferedReader br = new BufferedReader(new InputStreamReader(
```

```
        (conn.getInputStream())));
String output;
String s = "";
while ((output = br.readLine()) != null) {
    s += output;
}
System.out.println(s);
conn.disconnect();

} catch (MalformedURLException ex) {
    JOptionPane.showMessageDialog(null, "Malformed url");
} catch (IOException ex) {
    JOptionPane.showMessageDialog(null, "Connection problem");
}
}
}
```

## Rubrica telefonica REST su database (*da finire*)

I dati della rubrica telefonica possono essere gestiti nel database *phonebook*, il cui [schema concettuale](#), semplificando la realtà da gestire, può essere il seguente.



Dizionario dati:

Entità	Descrizione	Attributi	Identificatore
Contacts	Contatti presenti nella rubrica telefonica	pkcontact, name, surname	pkcontact
Numbers	Numeri di telefono presenti nella rubrica	pknumber, number	pknumber

Relazione	Descrizione	Entità coinvolte	Attributi
Use	Associa ad un contatto i relativi numeri di telefono	Contacts, Numbers	-

L'attributo *pkcontact* identifica univocamente le occorrenze dell'entità *contacts*. L'attributo *pknumber* identifica univocamente le occorrenze dell'entità *numbers*.

Lo [schema logico](#) sarà il seguente.

```
contacts(pkcontact, name, surname)
```

```
numbers(pknumber, number)
```

```
use(fkcontact, fknumber)
```

### DDL e DML

```
-- phpMyAdmin SQL Dump
-- version 4.5.4.1deb2ubuntu2
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Mar 18, 2017 at 12:41 PM
-- Server version: 5.7.17-0ubuntu0.16.04.1
-- PHP Version: 7.0.15-0ubuntu0.16.04.4
```

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
```

```
-- Database: `phonebook`
```

```
--
```

```
-- -----
```

```
-- Table structure for table `contacts`
```

```
--
```

```
CREATE DATABASE `phonebook`;
```

```
USE `phonebook`;
```

```
CREATE TABLE `contacts` (
  `pkcontact` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `surname` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
-- Dumping data for table `contacts`
--
```

```
INSERT INTO `contacts` (`pkcontact`, `name`, `surname`) VALUES
(1, 'Silvio', 'Rossi'),
(2, 'Gianna', 'Bianca'),
(3, 'Romeo', 'Capuleti'),
(4, 'Giulietta', 'Montecchi'),
(5, 'Renzo', 'Tramaglino'),
(6, 'Lucia', 'Mondella');
```

---

```
--  
-- Table structure for table `contacts_numbers`  
--
```

```
CREATE TABLE `use` (
  `fkcontact` int(11) NOT NULL,
  `fknumber` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Dumping data for table `contacts_numbers`  
--
```

```
INSERT INTO `use` (`fkcontact`, `fknumber`) VALUES
(3, 1),
(4, 1),
(2, 2),
(1, 3),
(2, 3),
(5, 4),
(6, 4),
(3, 5),
(4, 6),
(1, 7),
(5, 8),
(6, 9);
```

```
--  
-- Table structure for table `numbers`  
--
```

```
CREATE TABLE `numbers` (
  `pknumber` int(11) NOT NULL,
  `number` varchar(30) NOT NULL
) ENGINE=Innodb DEFAULT CHARSET=latin1;
```

```
--  
-- Dumping data for table `numbers`  
  
INSERT INTO `numbers` (`pknumber`, `number`) VALUES  
(1, '01156461223'),  
(2, '34566889977'),  
(3, '0255112244'),  
(4, '0645612399'),  
(5, '333666555444'),  
(6, '32265897411'),  
(7, '38845123945'),  
(8, '32847812365'),  
(9, '323564879');  
  
--  
-- Indexes for dumped tables  
  
--  
  
--  
-- Indexes for table `contacts`  
  
--  
ALTER TABLE `contacts`  
ADD PRIMARY KEY (`pkcontact`);  
  
--  
-- Indexes for table `contacts_numbers`  
  
--  
ALTER TABLE `use`  
ADD PRIMARY KEY (`fkcontact`, `fknumber`),  
ADD KEY `fk_number` (`fknumber`);  
  
--  
-- Indexes for table `numbers`  
  
--  
ALTER TABLE `numbers`  
ADD PRIMARY KEY (`pknumber`);  
  
--  
-- AUTO_INCREMENT for dumped tables  
  
--  
  
--  
-- AUTO_INCREMENT for table `contacts`  
  
--  
ALTER TABLE `contacts`  
MODIFY `pkcontact` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;
```

```
--  
-- AUTO_INCREMENT for table `numbers`  
--  
ALTER TABLE `numbers`  
  MODIFY `pknumber` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;  
--  
-- Constraints for dumped tables  
--  
  
--  
-- Constraints for table `contacts_numbers`  
--  
ALTER TABLE `use`  
  ADD CONSTRAINT `use_ibfk_1` FOREIGN KEY (`fkcontact`) REFERENCES `contacts`(`pkcontact`),  
  ADD CONSTRAINT `use_ibfk_2` FOREIGN KEY (`fknumber`) REFERENCES `numbers`(`pknumber`);  
  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

# Web Service REST in Java e l'uso delle annotazioni

# Web Service REST, Jersey e JAX-RS

I cinque principi analizzati nelle sezioni precedenti ci consentono di definire un **Web service** aderente ad una architettura in stile **REST**.

Inizialmente creeremo un semplice Web service REST che effettui la conversione da metri a chilometri. Questo Web service sarà anche l'introduzione alla libreria **Jersey**, una delle molteplici librerie disponibili per la scrittura di Web service REST. Si consideri che l'uso di una libreria permette in sostanza di imparare l'uso della maggior parte delle altre librerie, in quanto tutte usano la medesima interfaccia API, cioè **JAX-RS**.

La API **JAX-RS** è costituita da una serie di interfacce e annotazioni<sup>17</sup>, e imparare ad usare le interfacce e le annotazioni è la modalità per scrivere dei Web service REST usando **JAX-RS**. Risulta necessario l'uso di una libreria come **Jersey** in quanto **JAX-RS** non ha alcuna funzionalità, e quindi è necessario avere delle classi che implementino le interfacce e siano in grado di leggere le annotazioni. Quindi sono le librerie come **Jersey** o RESTEasy che fanno effettivamente tutto il lavoro, usando le interfacce e le annotazioni di **JAX-RS**.

Una volta costruita l'applicazione, di questa se ne dovrà fare il [deployment](#) in un Application server, come Tomcat o Glassfish e quindi dovrà essere inclusa la libreria scelta, nel nostro caso **Jersey**, che al suo interno ha già una copia della API **JAX-RS**.

## UnitConverter - Esempio di Web Service REST usando JAX-RS

### UnitConverter - Creazione di un progetto su NetBeans

#### Progetto Maven

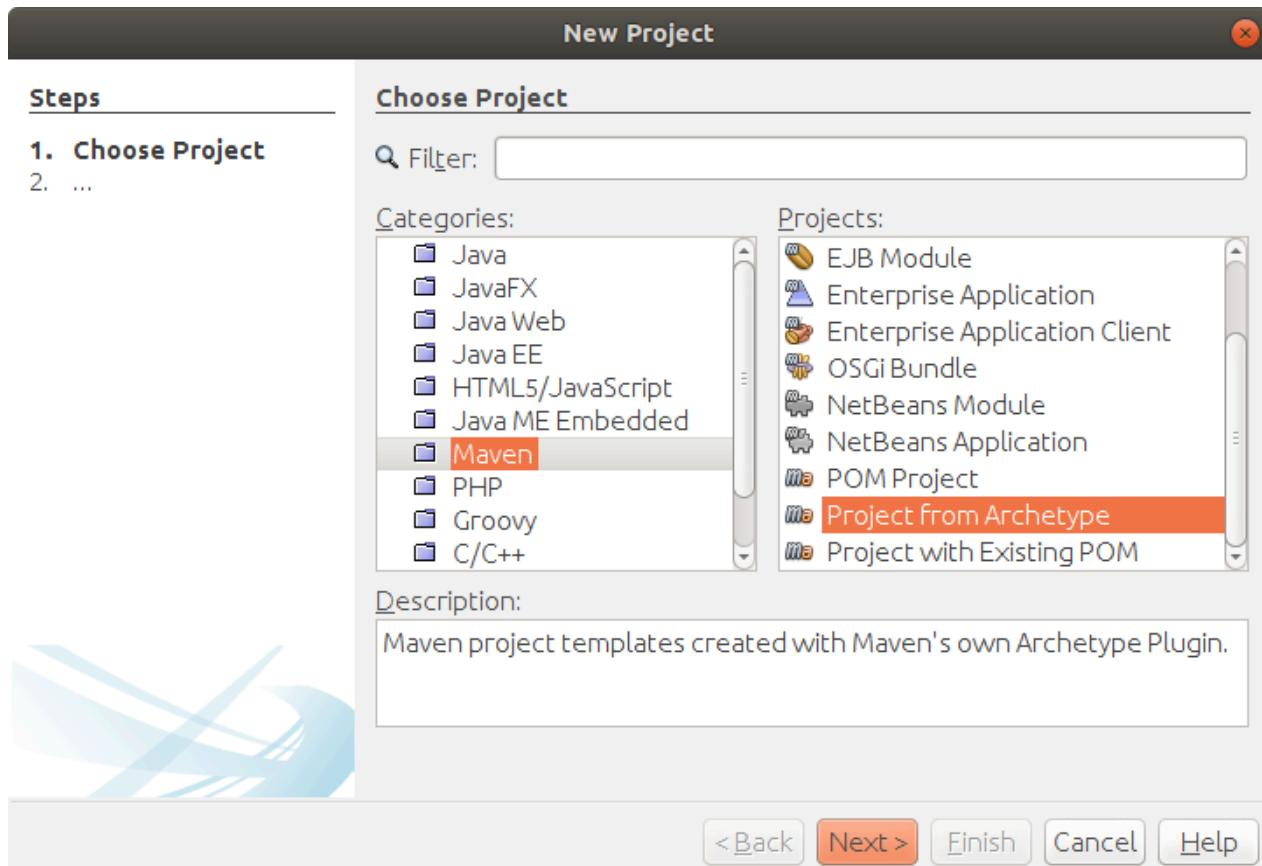
Se fosse necessario scaricare **JAX-RS** si potrà andare nella sezione download del sito di **Jersey**, ma, come è anche riportato dallo stesso sito, **Jersey** può essere usato in modo molto semplice creando un progetto [Maven](#) che è sostanzialmente uno strumento di gestione di progetti software basati su Java e build automation e che scarica automaticamente **JAX-RS**.

Sul proprio computer dovranno essere installati la versione Java EE e una versione completa di NetBeans.

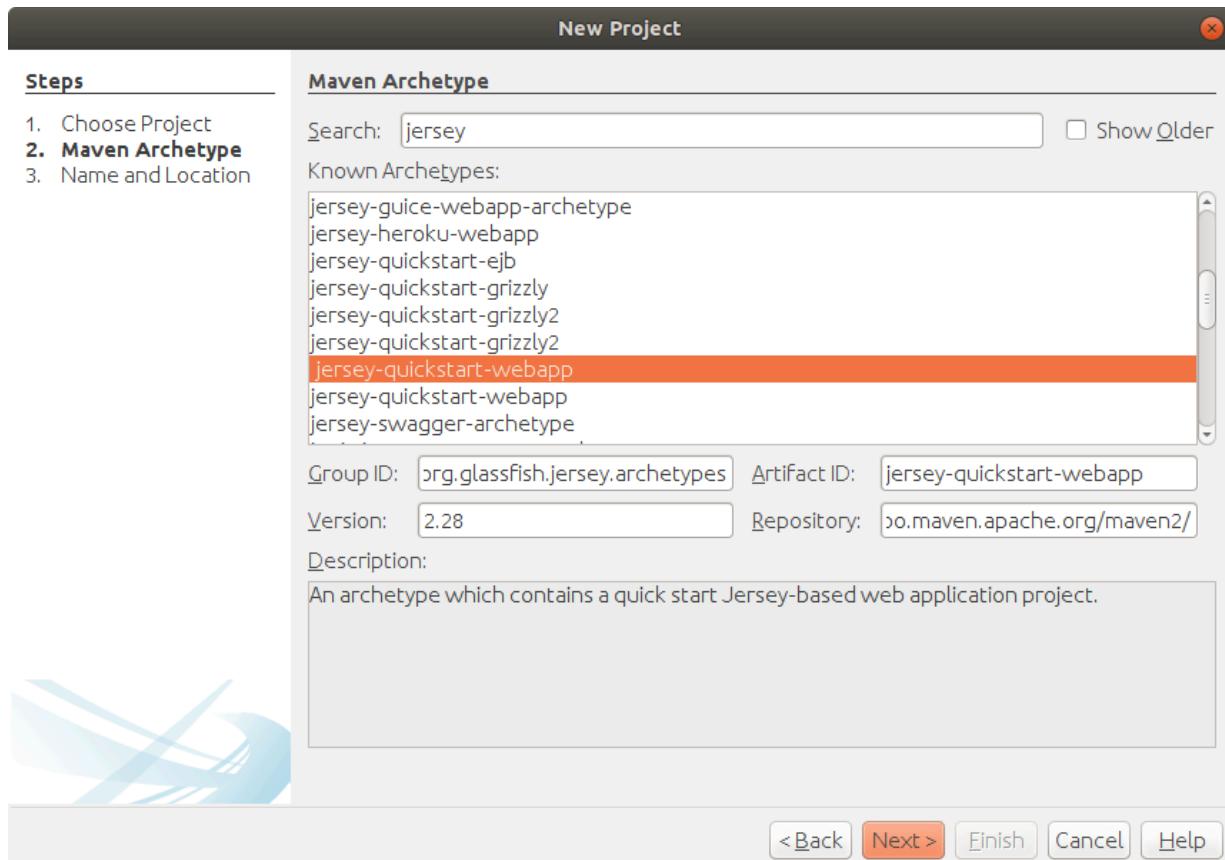
Su NetBeans si crei un nuovo progetto scegliendo in *Categories: Maven* e in *Projects: Project from Archetype*.

---

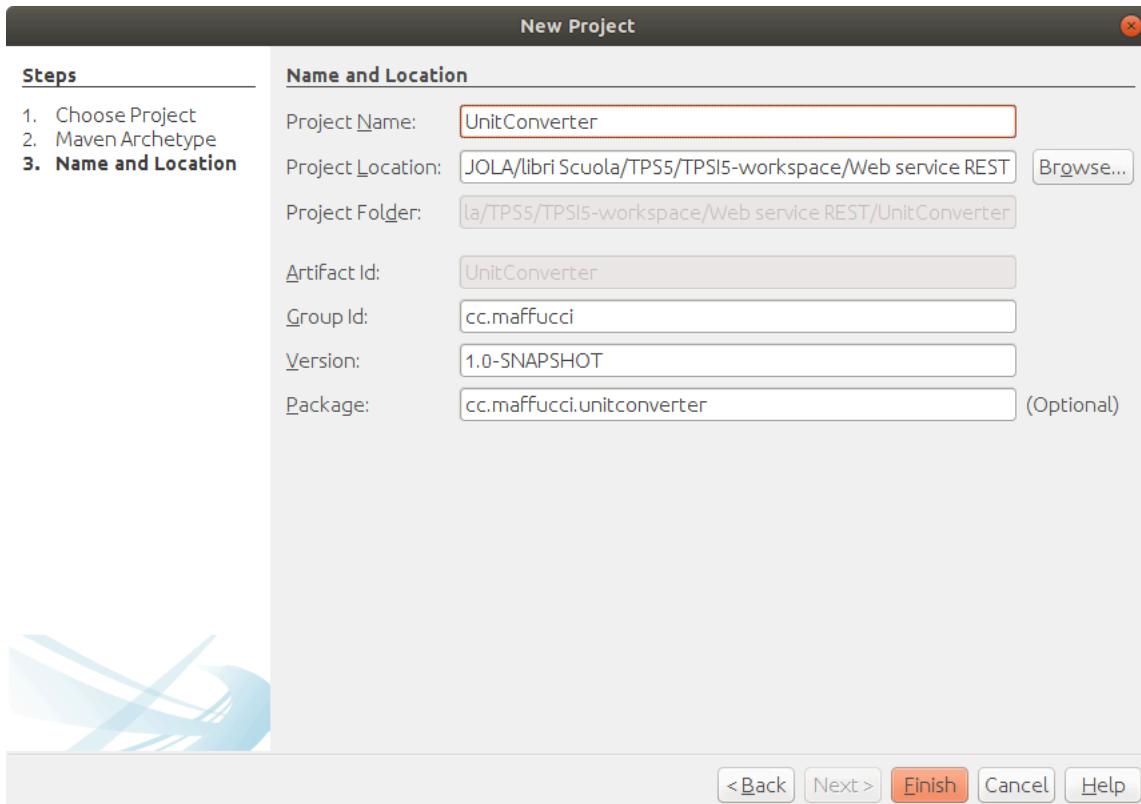
<sup>17</sup> [Annotation - The Java™ Tutorials](#), ORACLE© Java Documentation

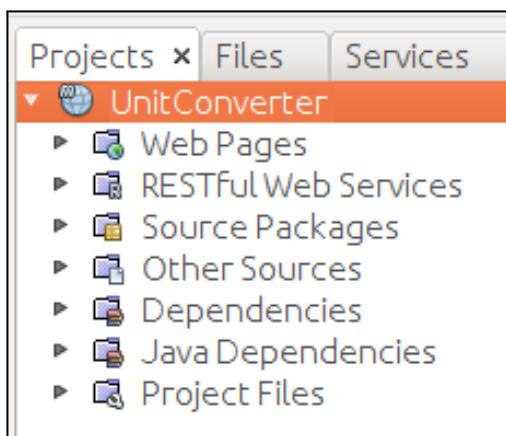


Un *Project from Archetype* è sostanzialmente un progetto già impostato con una precisa struttura per costruire progetti di un certo tipo. Esistono molteplici archetipi per i progetti **Jersey**, e nella finestra successiva si dovrà scegliere quello che si vorrà usare, che per i nostri scopi sarà *jersey-quickstart-webapp* del gruppo *org.glassfish.jersey.archetypes*. Il numero di versione potrebbe differire da quello mostrato in figura.



Continuando con la creazione si dovrà assegnare un nome al progetto, *UnitConverter*, e un nome al package, identificato con *Group Id*, che sarà scelto seguendo la convenzione di usare il nome del sito dell'azienda o del proprietario scritto in ordine inverso, nel mio caso *cc.maffucci*.

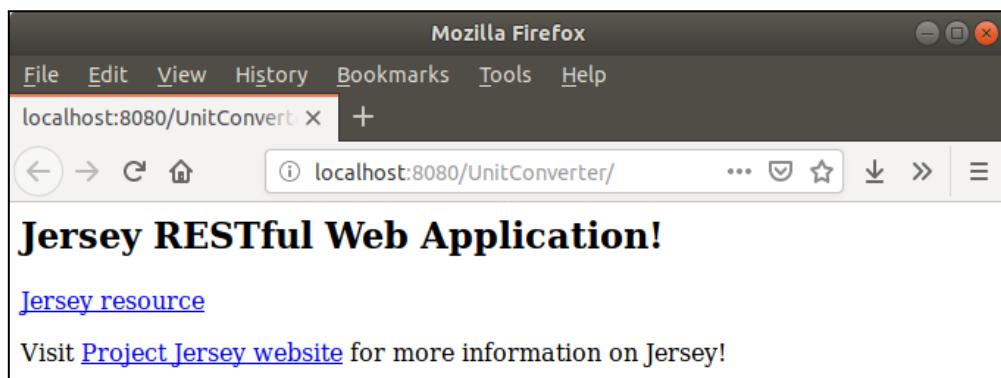




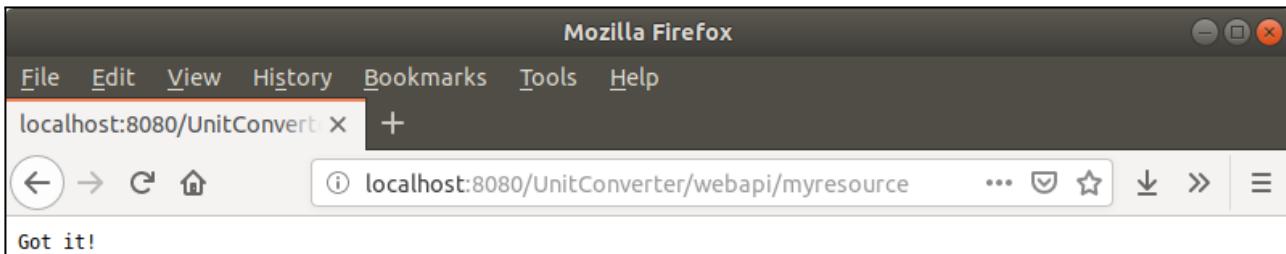
Una volta conclusa la fase di creazione il progetto sarà creato con la struttura dell'archetipo selezionato.

Per creare un progetto Maven usando [Eclipse](#), si vedano i video [10](#) e [11](#) di [CLIL Listening - Developing RESTful APIs with JAX-RS](#), utili anche per imparare ad usare **JAX-RS**.

Una volta creato il progetto questo potrà essere avviato scegliendo, quando richiesto, uno degli Application server presenti, e la pagina presentata sarà la seguente.



Si osservi che il Web service risponde dal localhost sulla porta 8080 mostrando una semplice pagina Web. Il link *Jersey resources* se premuto provocherà l'invocazione di una request al Web service che risponderà con una response costituita dalla visualizzazione di una semplice stringa, come mostrato nell'immagine seguente.



Se dovesse essere visualizzata una pagina di errore, probabilmente si dovrà modificare il *deployment descriptor* dell'applicazione, cioè il file **web.xml**, modificando il valore del tag <url-pattern> inserendo webapi:

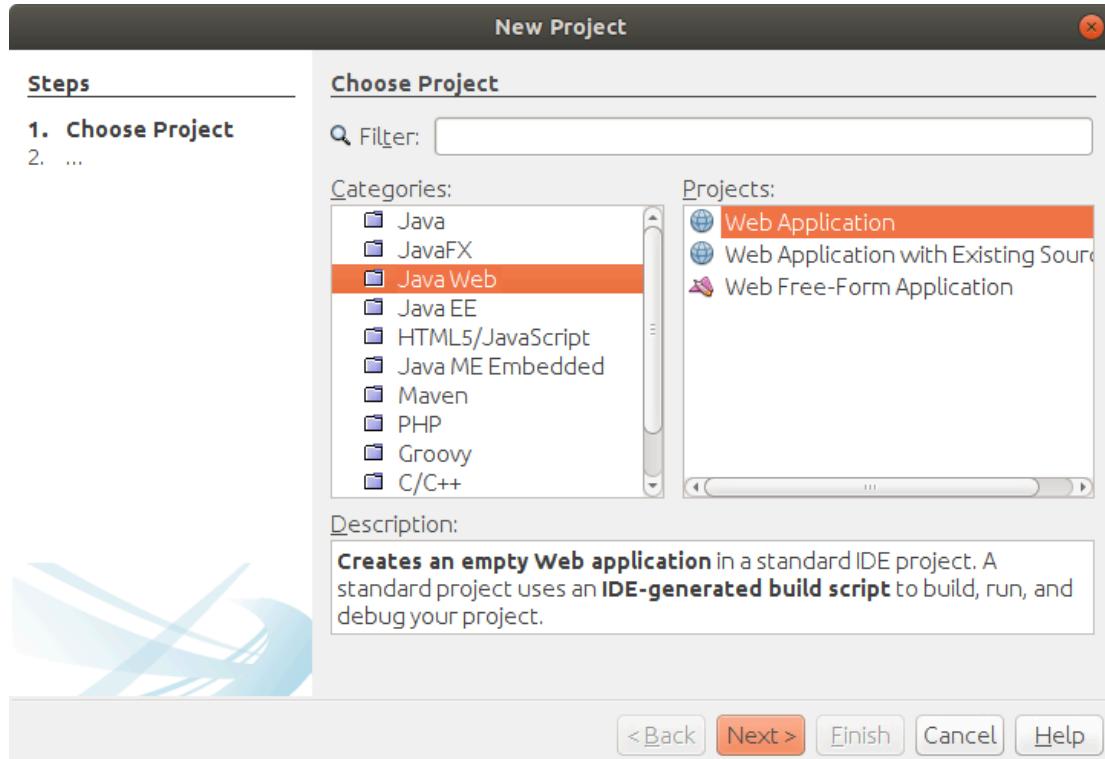
```
<url-pattern>/webapi/*</url-pattern>
```

La libreria **Jersey** sostanzialmente implementa le Servlet necessarie per gestire le richieste e le risposte HTTP del Web service REST, per questa ragione è presente nell'applicazione il *deployment descriptor* **web.xml**.

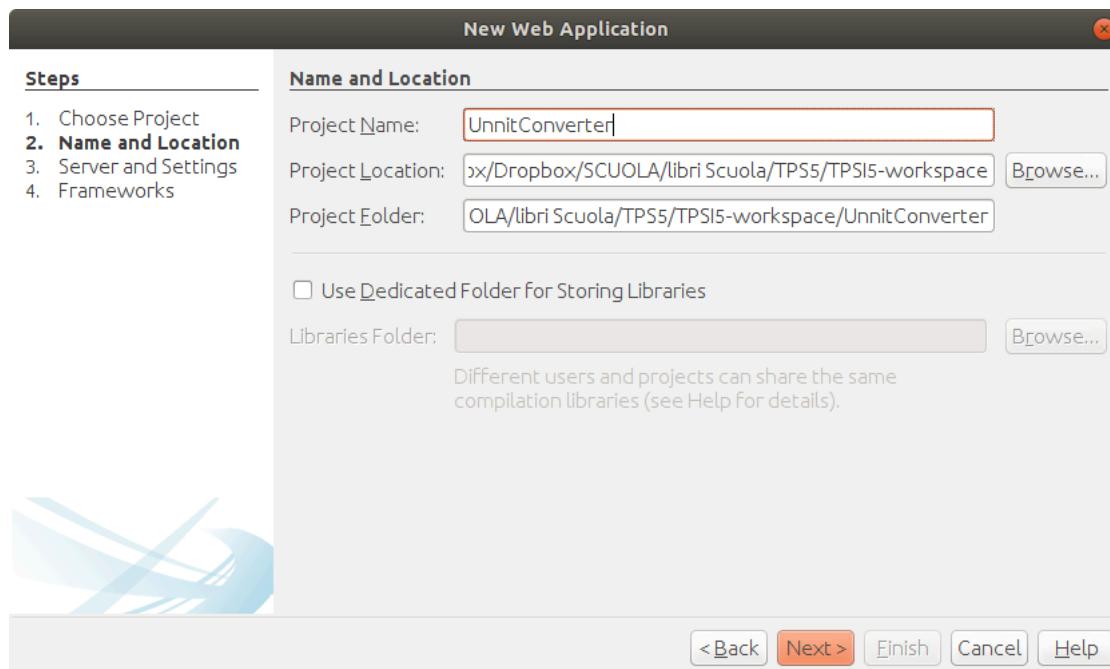
## Progetto Java Web

È possibile creare un Web service REST anche senza usare Maven, creando una *Web Application*, aggiungendo le librerie di **Jersey**, **JAX-RS**, **JAX-WS** e creando il *deployment descriptor* **web.xml**.

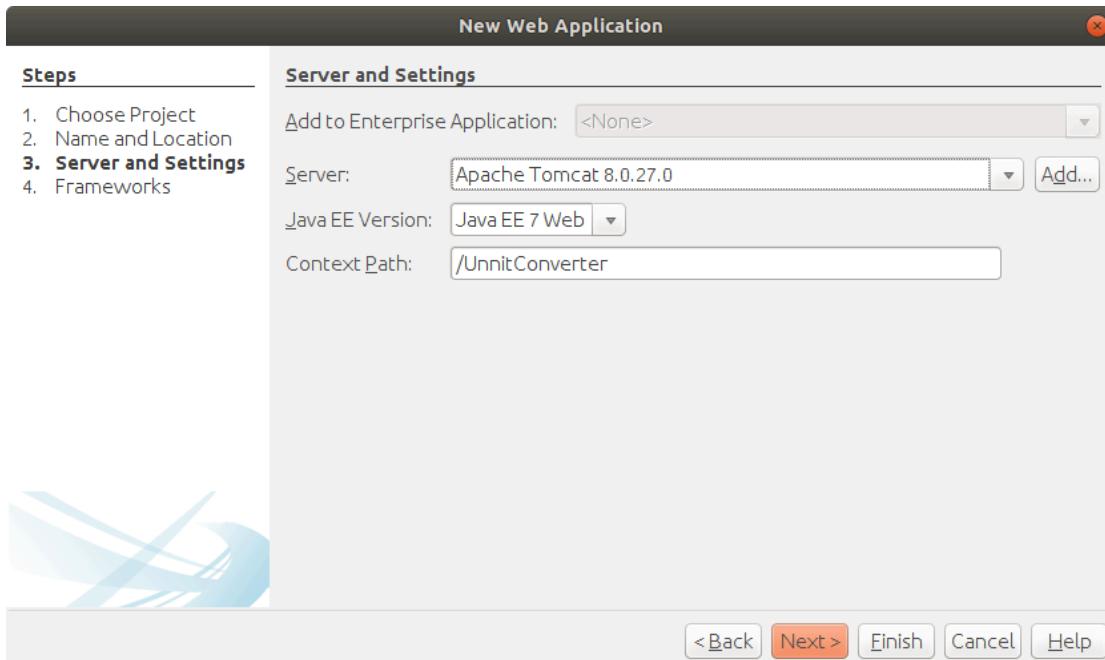
Su NetBeans si crei nuovo progetto scegliendo in *Categories: Java Web* e in *Projects: Web Application*.



Si assegna un nome al progetto.



Si scelga l'Application server desiderato e si prema il tasto *Finish*.



Nella cartella *Libraries* del progetto si aggiungano le librerie **Jersey 2.5.1 (JAX-RS RI)**, **JAX-RS 2.0** e **JAX-WS 2.2.6** (i numeri di versione potrebbero differire). Per l'aggiunta si prema il tasto destro del mouse sulla cartella *Libraries* e si scelga *Add Library...*. Se alcune librerie non fossero presenti, le si aggiunga come plugin.

Si crei nella cartella *Source Packages* il package principale (ad esempio *cc.maffucci.unnitconverter*) in cui saranno salvate le classi del progetto, eventualmente salvate in altri sottopackage. Il package principale è importante perché dovrà essere associato al package di **Jersey** nel *deployment descriptor web.xml*.

Si dovrà creare il *deployment descriptor web.xml* nella cartella *WEB-INF* del progetto, richiedendo la creazione con il menù contestuale sulla cartella. Si scelga dal menù contestuale *New/Other...* e si scelga *Categories: Web* e *File Types: Standard Deployment Descriptor*. Ci si accerti che venga salvato all'interno della cartella *WEB-INF* del progetto.

Si sostituisca il contenuto del file **web.xml** con il seguente codice dove viene indicata la classe di **Jersey** da usare nel tag `<servlet-class>`, viene associato il package di **Jersey** con il package in cui sono memorizzate le classi Java del progetto tramite in tag `<init-param>`, e viene associato il nome della servlet ad un preciso URL nel tag `<servlet-mapping>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer
    </servlet-class>
```

```
<init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>cc.maffucci.unitconverter</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webapi/*</url-pattern>
</servlet-mapping>
</web-app>
```

Infine si proceda con la creazione del Web service.

## UnitConverter - Installazione di un Client REST

Per poter testare il Web Service *UnitConverter* senza dover scrivere un client è possibile installare un'estensione nel browser che ci permetterà di fare i test necessari e verificare il codice mentre si effettua lo sviluppo.

Esistono molti Client REST disponibili per i vari browser ed installabili come semplici estensioni, di seguito ne vengono elencati alcuni:

- Chrome: [Tabbed Postman - REST Client](#)
- Firefox: [RESTED](#)
- Firefox: [RESTClient](#)

Ognuno di questi client REST permettono di inserire l'URL di richiesta della risorsa, di scegliere il metodo HTTP da usare, e nel caso si usasse un metodo che prevede un body, ne consentono l'inserimento, e visualizzano la risposta inviata dal Web service della quale è possibile anche analizzare lo header HTTP.

## UnitConverter - Sviluppo dell'applicazione

Il Web service appena creato risulta subito funzionante. Esso è costituito da un file index.jsp che implementa la semplice pagina Web mostrata in precedenza e che per ora non sarà utilizzata nell'applicazione<sup>18</sup>.

### index.jsp

```
<html>
<body>
    <h2>Jersey RESTful Web Application!</h2>
    <p><a href="webapi/myresource">Jersey resource</a>
    <p>Visit <a href="http://jersey.java.net">Project Jersey website</a>
        for more information on Jersey!
</body>
</html>
```

---

<sup>18</sup> Questo file sarà presente solo se il progetto è un progetto Maven.

Inoltre è presente il seguente *deployment descriptor*<sup>19</sup> che fornisce, con il tag <url-pattern>, la struttura del path dell'applicazione che dovrà essere usato come prefisso per qualsiasi invocazione di risorse dell'applicazione stessa, /webapi/\*. L'asterisco permetterà di inserire qualsiasi sottosequenza di URL che identificheranno le risorse.

## web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>
      org.glassfish.jersey.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>cc.maffucci.unitconverter</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webapi/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Il Web service, una volta che sarà completato, fornirà come risposta un file XML pronto per essere elaborato da un'altra applicazione. Il risultato sarà analogo al seguente:



La prima cosa che sarà implementata sarà una classe che identifica la risorsa, nel nostro caso i dati che identificano i metri e i chilometri. La classe sarà usata come elemento radice per la creazione del file XML grazie alla presenza dell'annotazione **@XmlRootElement**. Risulta fondamentale, per la costruzione del file XML, la presenza di un costruttore senza parametri che può anche risultare vuoto.

<sup>19</sup> Aggiunto automaticamente se è stato creato un progetto Maven, o aggiunto dal programmatore nel caso fosse stata creata una Web Application.

## Conversion.java

```
package cc.maffucci.unitconverter.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement // Identifies the root element to build the XML file
public class Conversion {
    private double meter;
    private double kilometer;

    public Conversion () {}

    public Conversion(double meter, double kilometer){
        this.meter = meter;
        this.kilometer = kilometer;
    }

    public double getMeter() {
        return meter;
    }

    public void setMeter(double meter) {
        this.meter = meter;
    }

    public double getKilometer() {
        return kilometer;
    }

    public void setKilometer(double kilometer) {
        this.kilometer = kilometer;
    }
}
```

Il passo successivo consisterà nella costruzione di una classe che manipola i dati offrendo tutti i servizi necessari per fornire le risposte che potrebbe richiedere il client. Nel caso dell'applicazione *UnitConverter* sarà necessario solo un metodo che effettui la conversione da metri a chilometri.

## ConversionService.java

```
package cc.maffucci.unitconverter.service;

import cc.maffucci.unitconverter.model.Conversion;

public class ConversionService {

    public Conversion getConversion(double meter) {
        double kilometer = meter / 1000;
```

```
    Conversion conversion = new Conversion(meter, kilometer);
    return conversion;
}
}
```

Infine si dovrà gestire la comunicazione con il client, creando una classe che conterrà tutti i metodi Java ai quali sarà associato un preciso metodo HTTP, tramite un'annotazione (**@GET**, **@POST**, **@PUT**, **@DELETE**), che identifica l'azione da svolgere sulla risorsa.

La classe Java è associata ad un URL tramite l'annotazione **@Path**, che è usata anche per mappare i singoli metodi Java alla sottosequenza dell'URL che permette di invocarlo. Questa sottosequenza può contenere l'indicazione di parametri, rinchiusi tra parentesi graffe, che sono associati ai parametri formali del metodo tramite l'annotazione **@PathParam**.

Infine viene specificato il Content-type della risposta utilizzando l'annotazione **@Produces** e l'enumerazione **MediaType**. L'indicazione del Content-type della risposta è fondamentale affinché la risposta del Web service abbia un esito positivo (status-code 2xx).

### ConversionResource.java

```
package cc.maffucci.unitconverter.resources;

import cc.maffucci.unitconverter.model.Conversion;
import cc.maffucci.unitconverter.service.ConversionService;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("conversions") // Mapping an URL to a Java class
public class ConversionResource {

    ConversionService conversionService = new ConversionService();

    @GET // Mapping an HTTP method to a Java method
    @Path("{meter}") // Mapping a subsequent URL path to a Java method
    @Produces(MediaType.APPLICATION_XML) // Defines the response Content-type
    public Conversion getConversion(@PathParam("meter") double meter) {
        return conversionService.getConversion(meter);
    }
}
```

Per testare il Web service se ne faccia il deploy e lo si invochi tramite uno dei client REST indicati in precedenza, oppure usando direttamente il browser dato che viene invocato il metodo HTTP GET, inserendo l'URL seguente:

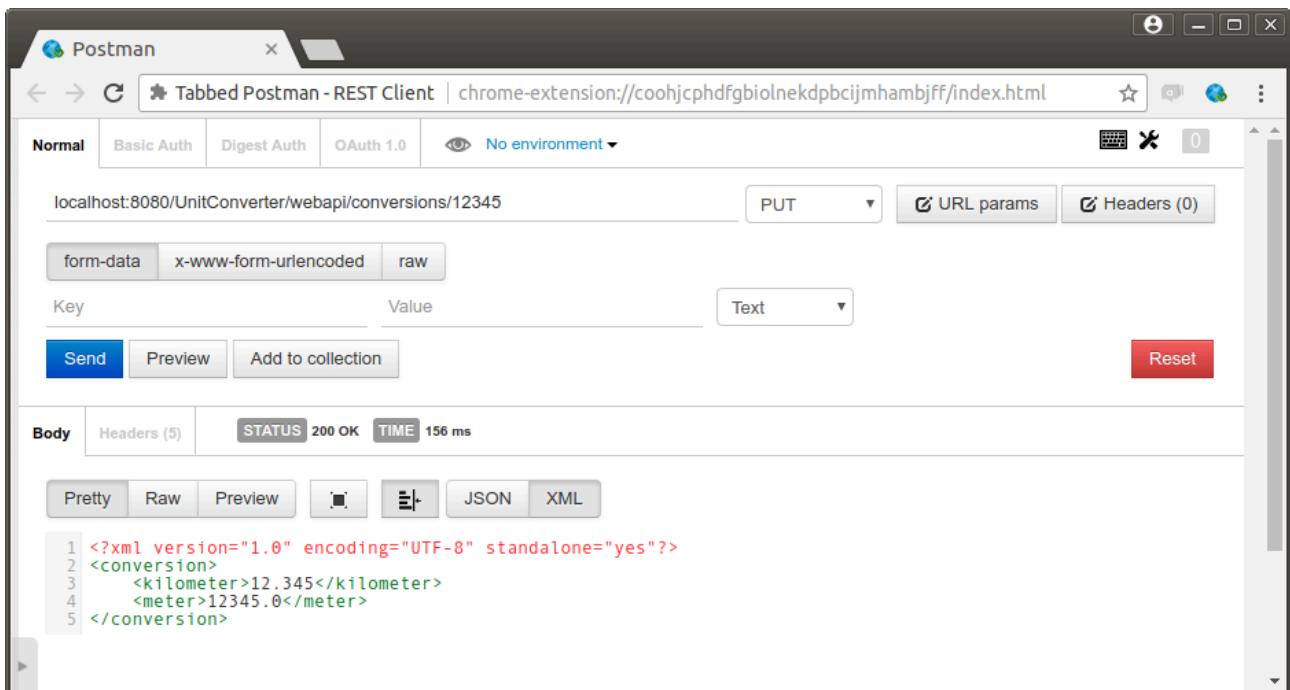
<http://localhost:8080/UnitConverter/webapi/conversions/1750>

Il risultato sarà la visualizzazione del file XML generato automaticamente da **Jersey**:



```
<conversion>
<kilometer>1.75</kilometer>
<meter>1750.0</meter>
</conversion>
```

Secondo i dettami dell'architettura **REST** l'uso del metodo HTTP **GET** non è il metodo adatto per trattare la conversione proposta, che si avvicina più ad una modifica della risorsa piuttosto che ad una lettura, richiedendo quindi l'uso del metodo HTTP **PUT**. Modificando il codice della classe *ConversionResource.java* utilizzando l'annotazione @PUT al posto dell'annotazione @GET, si dovrà testarne il funzionamento con uno dei client REST installati come estensione del browser, ottenendo il seguente risultato. Si osservi la scelta del metodo PUT dall'elenco dei metodi disponibili, e si consideri che in questo caso l'uso del solo browser non va bene dato che non è possibile indicare il metodo HTTP che si desidera usare per la request.



## CLIL Comprehension - Developing RESTful APIs with JAX-RS

Watch the following video to understand how to implement a CRUD REST Web Service. There will be some differences using NetBeans because the speaker uses Eclipse rather than NetBeans and JSON rather than XML, but these differences can easily be overcome.

- [REST Web Services 10 - What Is JAX RS?](#) - The video gives a general introduction of what JAX-RS and Jersey are.
- [REST Web Services 11 - Setting Up](#) - The video shows how to create a Maven project on Eclipse, using a chosen Jersey archetype.

- [REST Web Services 12 - Understanding the Application Structure](#) - The video describes the application structure and the URL structure to access to an application resource. The two structures are strictly defined by the chosen application archetype.
- [REST Web Services 13 - Creating a Resource](#) - The video describes the URLs structure used, in order to create the REST Web service Messenger Application that manages users messages and users profile. The video also describes how to write personal resources and how to map an URL to a Java class and an HTTP method to a Java method. In the package resources there are all the REST methods.
- [REST Web Services 14 - Returning XML Response](#) - In the video is created a class that is used as a mock model in the application, rather than create a database.
- [REST Web Services 15 - Installing a REST API client](#) - The video shows how to install [Tabbed Postman - REST Client Extension](#) into Google Chrome and use it as a REST client. For Firefox there are also REST client such as [RESTED](#) and [RESTClient](#)
- [REST Web Services 16 - Building Service Stubs](#) - The video shows all the services used to manipulate messages. It is also created a class to manage users profile. The database is mocked up by a database class that uses a Map data structure to memorize messages and profiles.
- [REST Web Services 17 - Accessing Path params](#) - The video shows how to access single and multiple path parameters in the URL.
- [REST Web Services 18 - Returning JSON Response](#) - The video shows how to obtain a JSON format response instead We'll now switch the response format of the APIs from XML to JSON

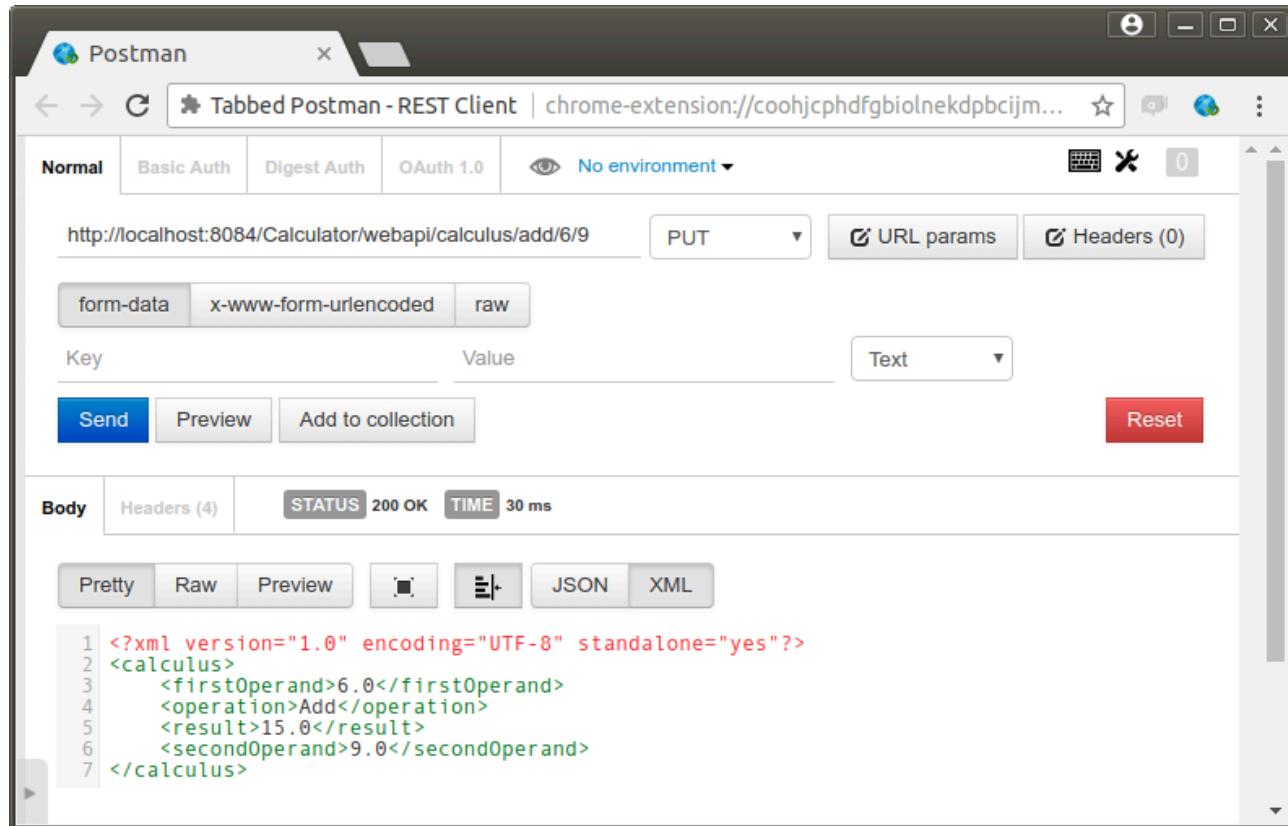
## Calculator - Esempio di Web Service REST usando JAX-RS

Il seguente esempio prevede la gestione di una semplice calcolatrice. È stata sviluppata solo l'operazione di addizione, le altre operazioni vengono lasciate come esercizio.

Il progetto prevede l'inserimento di un URL che identifichi in modo univoco la risorsa, come richiesto dai principi REST, e che in questo caso è costituita dall'operazione e i due operandi:

`http://localhost:8084/Calculator/webapi/calculus/add/6/9`

Il risultato che si ottiene è il seguente:



Anche in questo caso si è creata inizialmente la classe **Calculus.java** che definisce la risorsa utile per la creazione del file XML.

### Calculus.java

```
package cc.maffucci;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Calculus {
    private String operation;
    private float firstOperand;
    private float secondOperand;
    private float result;

    public Calculus(){}
    public Calculus(String operation, float firstOperand, float secondOperand,
                  float result) {
        this.operation = operation;
        this.firstOperand = firstOperand;
        this.secondOperand = secondOperand;
        this.result = result;
    }
    public String getOperation() {
```

```
    return operation;
}

public void setOperation(String operation) {
    this.operation = operation;
}

public float getFirstOperand() {
    return firstOperand;
}

public void setFirstOperand(float firstOperand) {
    this.firstOperand = firstOperand;
}

public float getSecondOperand() {
    return secondOperand;
}

public void setSecondOperand(float secondOperand) {
    this.secondOperand = secondOperand;
}

public float getResult() {
    return result;
}

public void setResult(float result) {
    this.result = result;
}
```

Il passo successivo consisterà nella costruzione di una classe **CalculatorService.java** che manipola i dati offrendo tutti i servizi necessari per fornire le risposte che potrebbe richiedere il client. Nel caso di questa applicazione sarà per ora necessario solo un metodo che effettui la somma dei due operandi.

### **CalculatorService.java**

```
package cc.maffucci;

public class CalculatorService {

    public Calculus addOperation(float firstOperand, float secondOperand) {
        float result = firstOperand + secondOperand;
        Calculus calculus = new Calculus("Add", firstOperand, secondOperand,
                                         result);
        return calculus;
    }
}
```

}

Infine si dovrà gestire la comunicazione con il client, creando la classe **CalculatorResource.java** che conterrà tutti i metodi Java ai quali sarà associato, tramite un'annotazione (@GET, @POST, @PUT, @DELETE), un preciso metodo HTTP che identificherà l'azione da svolgere sulla risorsa. Questa classe sarà quella che intercetterà il metodo e l'URL inviato dal client.

### **CalculatorResource.java**

```
package cc.maffucci;

import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("calculus")
public class CalculatorResource {
    CalculatorService calculatorService = new CalculatorService();

    @PUT
    @Path("add/{firstOp}/{secondOp}")
    @Produces(MediaType.APPLICATION_XML)
    public Calculus add(@PathParam("firstOp") float firstOp,
        @PathParam("secondOp") float secondOp) {
        return calculatorService.addOperation(firstOp, secondOp);
    }
}
```

Si osservi che l'uso di due operandi nell'URL viene risolto semplicemente associando ogni parametro all'argomento corrispondente del metodo add() utilizzando l'annotazione @PathParam.

## Web service per operazioni CRUD usando una struttura dati

Per comprendere il funzionamento delle operazioni CRUD, verrà utilizzato inizialmente un esempio basato su una struttura dati che simulerà un database. In questo modo si svilupperanno prima le competenze di gestione delle annotazioni e dei metodi necessari ad invocare il metodo HTTP atto a svolgere la giusta operazione richiesta sulla struttura dati, senza perdere nei dettagli della connessione al database e la conseguente modifica dei dati. Successivamente queste competenze saranno usate per modificare effettivamente un database.

# CountryPopulation - Esempio di Web Service REST usando JAX-RS

L'esempio illustrato nel paragrafo prevede la manipolazione dei dati sulla popolazione di diversi paesi, che saranno memorizzati all'interno di una *HashMap*<sup>20</sup>.

Il progetto prevede la creazione di un *HashMap* in cui saranno inseriti all'avvio dell'applicazione quattro stati con la relativa popolazione, e verrano usati i quattro metodi HTTP per effettuare tutte le operazioni CRUD sulla struttura dati, usando adeguatamente l'uso degli URL per identificare in modo univoco le diverse risorse, come richiesto dai principi REST.

Per procedere con la creazione del progetto si crei un progetto Maven o una Web Application come illustrato nei paragrafi precedenti, assegnandogli il nome *CountryPopulation* e si proceda con l'inserimento delle seguenti classi.

---

<sup>20</sup> [Come funzionano le \*HashMap\* in Java](#)

La classe **CountryDatabase.java** simula il database utilizzando la struttura dati *HashMap*. Nell'esempio proposto è stato necessario inizializzare il contenuto della *HashMap* usando un **blocco statico** all'interno della classe **CountryDatabase.java**. Un blocco statico esegue le istruzioni in esso contenuto solo una volta, quando la classe è inizializzata, indipendentemente dal numero di oggetti che vengono creati. Questo si è reso necessario perché Jersey crea un'istanza di **CountryController** ad ogni richiesta, istanziando a sua volta la *HashMap* presente in **CountryService** tutte le volte, vanificando le modifiche effettuate con il metodo PUT.

### CountryDatabase.java

```
package cc.maffucci;

import java.util.HashMap;
import java.util.Map;

public class CountryDatabase {

    private static HashMap<Integer, Country> countries = new HashMap<>();

    // Jersey creates a new instance of the CountryController for every request.
    // So the CountryService reference is instantiated time and time again,
    // overwriting changes made by a PUT its constructor. This can be solved by
    // moving the countries.put(...) code inside a 'static' block inside the
    // CountryDatabase class.
    // All the instruction inside a static block are executed once, when the
    // class itself is initialized, no matter how many objects of that type you
    // create.
    static {
        countries.put(1, new Country(1, "India", 10000));
        countries.put(4, new Country(4, "China", 20000));
        countries.put(3, new Country(3, "Nepal", 8000));
        countries.put(2, new Country(2, "Bhutan", 7000));
    }

    public static Map<Integer, Country> getCountries() {
        return countries;
    }
}
```

La classe **Country.java** fornisce il supporto per la creazione dei file XML (o JSON) e riporta tutti gli attributi che dovranno essere presenti nel file XML e che saranno memorizzati nella *HashMap*. Nella classe è stata aggiunta l'annotazione @XmlType con l'elemento propOrder che permettono di stabilire l'ordine dei campi nello schema generato, che nell'esempio proposto è un file XML.

### Country.java

```
package cc.maffucci;
```

```
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement // Identifies the root element to build the XML file
// The annotation @XmlType used with the element propOrder allow you to
// specify the content order in the generated schema type.
@XmlType(propOrder = {"id", "countryName", "population"})

public class Country {

    int id;
    String countryName;
    long population;

    public Country() {}

    public Country(int i, String countryName, long population) {
        this.id = i;
        this.countryName = countryName;
        this.population = population;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCountryName() {
        return countryName;
    }

    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }

    public long getPopulation() {
        return population;
    }

    public void setPopulation(long population) {
        this.population = population;
    }
}
```

La classe **CountryService.java** simulerà tutte le azioni svolte sul database, facendole sulla struttura dati *HashMap*. Inoltre popolerà inizialmente la *HashMap* con i valori delle popolazioni di quattro paesi.

### CountryService.java

```
package cc.maffucci;

import cc.maffucci.CountryDatabase;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CountryService {

    private static Map<Integer, Country> countries =
        CountryDatabase.getCountries();

    public CountryService() {}

    // Get all countries.
    public List<Country> getAllCountries() {
        return new ArrayList<Country>(countries.values());
    }

    // Get the country of a specified id
    public Country getCountry(int id) {
        return countries.get(id);
    }

    // Add a specifies country.
    public Country addCountry(Country country) {
        country.setId(countries.size() + 1);
        countries.put(country.getId(), country);
        return country;
    }

    // Update a country.
    public Country updateCountry(Country country) {
        if (country.getId() <= 0) {
            return null;
        }
        countries.put(country.getId(), country);
        System.out.println(countries.get(country.getId()).getCountryName());
        return country;
    }

    // Delete a country.
```

```
public void deleteCountry(int id) {
    countries.remove(id);
}
```

La classe **CountryController.java** implementa le operazioni CRUD seguendo i dettami dell'architettura REST.

Nella classe è previsto l'uso dell'annotazione @Consumes che indica il tipo di MIME della risorsa che può essere accettata o consumata da un client. Ogni volta che il Web service dovrà produrre una risorsa si dovrà prevedere l'annotazione @Produces e, ogni volta che il Web service dovrà accettare una risorsa inviata con un POST o un PUT, si dovrà prevedere un'annotazione @Consumes.

### CountryController.java

```
package cc.maffucci;

import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

/*
 * The @Consumes annotation is used to specify which MIME media types of
 * representations a resource can accept, or consume, from the client.
 * If @Consumes is applied at the class level, all the response methods accept
 * the specified MIME types by default. If @Consumes is applied at the method
 * level, it overrides any @Consumes annotations applied at the class level.
 * If a resource is unable to consume the MIME type of a client request, the
 * Jersey runtime sends back an HTTP "415 Unsupported Media Type" error.
 */
@Path("/countries")
// This method accepts XML resources.
@Consumes(MediaType.APPLICATION_XML)
// This method produces XML resources.
@Produces(MediaType.APPLICATION_XML)
public class CountryController {

    CountryService countryService = new CountryService();

    @GET
    public List<Country> getCountries() {
```

```
    return countryService.getAllCountries();
}

@GetMapping("/id")
public Country getCountryById(@PathParam("id") int id) {
    return countryService.getCountry(id);
}

@PostMapping
public Country addCountry(Country country) {
    return countryService.addCountry(country);
}

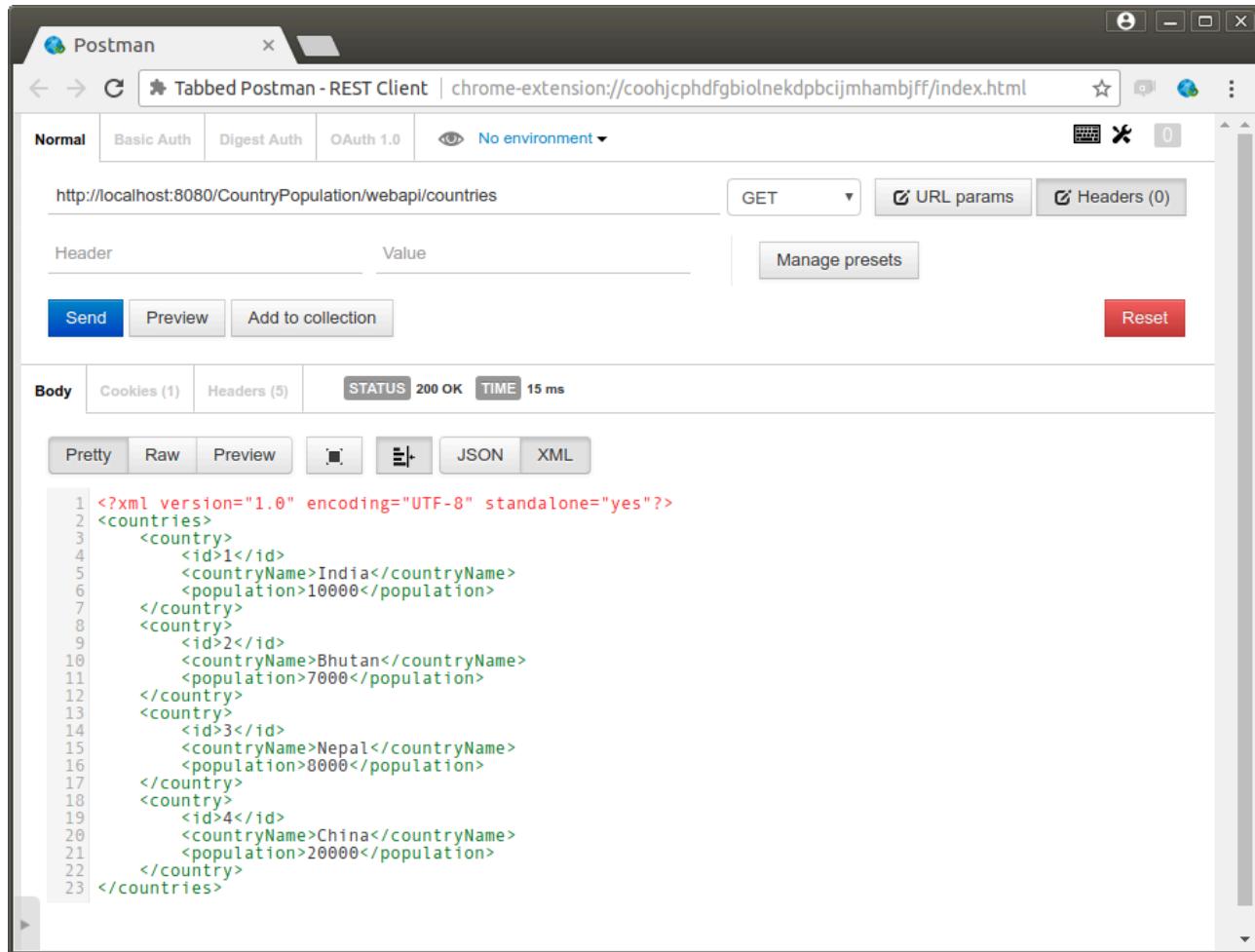
@PutMapping("/id")
public Country updateCountry(@PathParam("id") int id, Country country) {
    country.setId(id);
    return countryService.updateCountry(country);
}

@DeleteMapping("/id")
public void deleteCountry(@PathParam("id") int id) {
    countryService.deleteCountry(id);
}
```

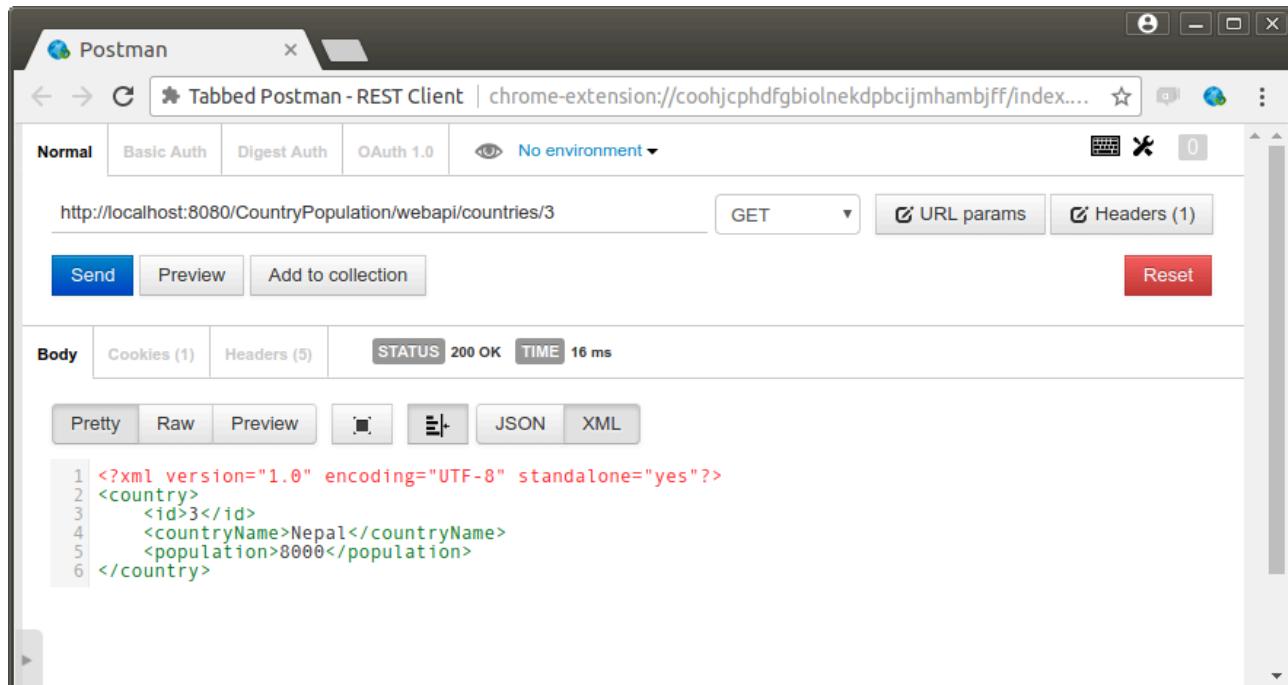
Nella classe sono previste due operazioni di lettura, una per ottenere i dati sulle popolazioni di tutti i paesi, l'altra per ottenere i dati sulla popolazione di un singolo paese, fornito l'identificativo del paese.

Nell'immagine seguente viene mostrato un esempio di lettura per ottenere i dati sulle popolazioni di tutti i paesi, usando il metodo GET e l'URL

`http://localhost:8080/CountryPopulation/webapi/countries`



Nell'immagine seguente viene invece richiesta la popolazione del paese con id pari a 3 usando l'URL `http://localhost:8080/CountryPopulation/webapi/countries/3`



Il metodo POST prevede l'inserimento dello header Content-Type di tipo application/xml e dell'inserimento dei dati in modalità raw sotto forma di file XML ben formato. Nel file XML non viene inserito l'id in quanto è previsto l'inserimento automatico da parte del metodo addcountry() della classe CountryService.java, simulando una chiave AUTO-INCREMENT di un database. L'URL usato è

`http://localhost:8080/CountryPopulation/webapi/countries`

Il server restituirà un file XML che riporta i dati inseriti con l'aggiunta dell'id pari a 5.

Invocando nuovamente una GET, si potrà vedere che nel file XML comparirà il nuovo paese.

The screenshot shows the Postman REST Client interface. In the top bar, the URL is set to `http://localhost:8080/CountryPopulation/webapi/countries`. The method is selected as `POST`. The `Content-Type` header is set to `application/xml`. The body of the request contains the following XML:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <country>
3   <countryName>India</countryName>
4   <population>10000</population>
5 </country>
```

Below the request section, the response status is shown as `200 OK` with a time of `13 ms`. The response body is displayed in XML format, showing the same XML structure as the request but with an additional `<id>5</id>` element added:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <country>
3   <id>5</id>
4   <countryName>India</countryName>
5   <population>10000</population>
6 </country>
```

Per modificare i dati di un paese si dovrà utilizzare il metodo PUT prevedendo l'inserimento dello header Content-Type di tipo application/xml e l'inserimento dei dati in modalità raw sotto forma di file XML ben formato. Nel file XML può essere omesso l'inserimento dell'id in quanto è previsto direttamente nell'URL. L'URL usato è

`http://localhost:8080/CountryPopulation/webapi/countries/1`

Il server restituirà un file XML che riporta i dati aggiornati per il paese con l'id fornito nell'URL.

Invocando nuovamente una GET, si potrà vedere che nel file XML comparirà il paese con i dati aggiornati.

The screenshot shows the Postman REST Client interface. The URL is `http://localhost:8080/CountryPopulation/webapi/countries/1`. The method is set to `PUT`. The request body is an XML document:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <country>
3   <countryName>India</countryName>
4   <population>40000</population>
5 </country>
```

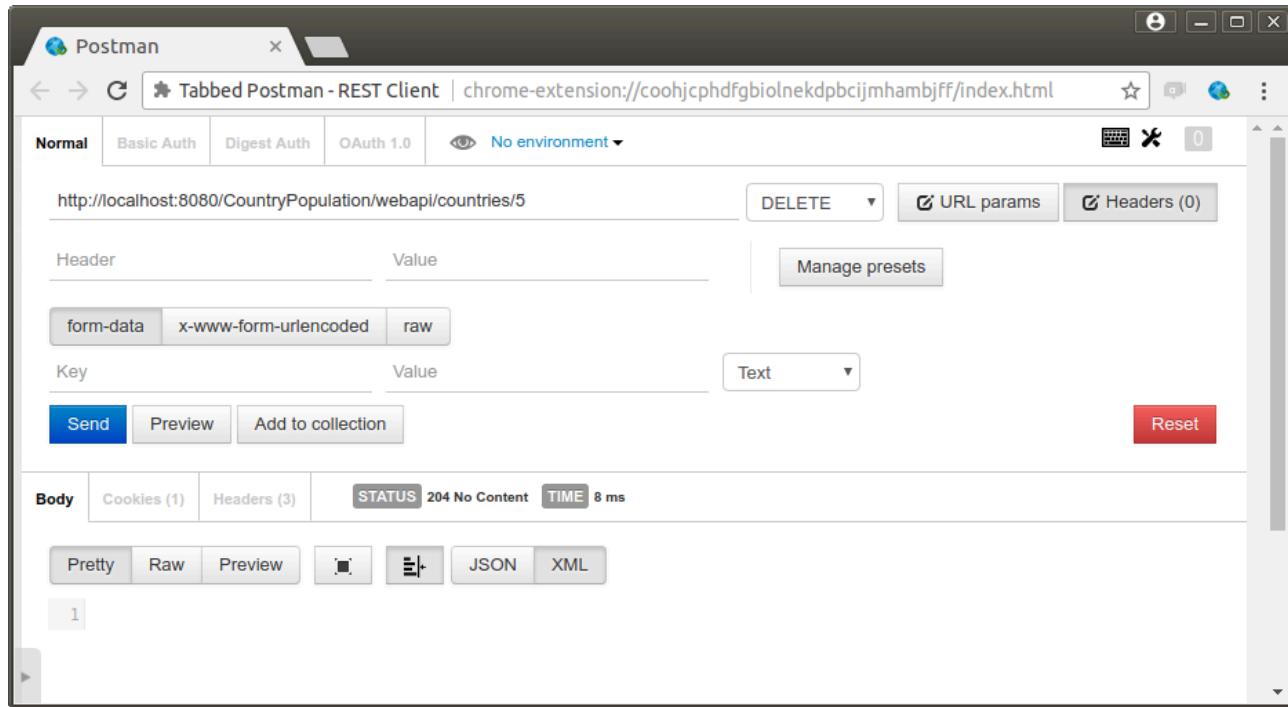
The response shows a `200 OK` status with a time of `28 ms`. The response body is identical to the request body:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <country>
3   <id>1</id>
4   <countryName>India</countryName>
5   <population>40000</population>
6 </country>
```

Per procedere con la cancellazione dei dati di un paese si dovrà utilizzare il metodo `DELETE` indicando nell'URL l'id del paese che si vorrà eliminare, ad esempio

`http://localhost:8080/CountryPopulation/webapi/countries/5`

Anche in questo caso invocando il metodo `GET`, si potrà vedere che nel file XML non comparirà più il paese con id pari a 5.



## CLIL Comprehension - Developing RESTful APIs with JAX-RS

Watch the following video to understand how to implement a CRUD REST Web Service. There will be some differences using NetBeans because the speaker uses Eclipse rather than NetBeans and JSON rather than XML, but these differences can easily be overcome.

- [REST Web Services 19 - Implementing POST Method](#) - The video shows how creating a POST method in order to add a new element into the collection.
- [REST Web Services 20 - Implementing Update and Delete](#) - The video explains how creating PUT and DELETE method.
- [REST Web Services 21 - Implementing ProfileResource](#) - In the video is created a new resource and a new service to manage profiles.

## CLIL Comprehension - Developing RESTful APIs with JAX-RS

Watch the following video to understand how to implement a CRUD REST Web Service. There will be some differences using NetBeans because the speaker uses Eclipse rather than NetBeans and JSON rather than XML, but these differences can easily be overcome.

- [REST Web Services 22 - Pagination and Filtering](#)
- [REST Web Services 23 - The Param Annotations](#)
- [REST Web Services 24 - Using Context and BeanParam annotations](#)
- [REST Web Services 25 - Implementing Subresources](#)
- [REST Web Services 26 - Sending Status Codes and Location Headers](#)
- [REST Web Services 27 - Handling Exceptions](#)
- [REST Web Services 28 - Using WebApplicationException](#)

- [REST Web Services 29 - HATEOAS \(Part 1\)](#)
- [REST Web Services 30 - HATEOAS \(Part 2\)](#)
- [REST Web Services 31 - Content Negotiation](#)

# Web service per operazioni CRUD su database [da finire]

Per rendere persistenti le risorse esposte da un Web service può essere utilizzato un database relazionale gestito da un DBMS, implementando in questo modo una **architettura a tre livelli**:

4. il **Presentation layer** che presenterà al **client** l'interfaccia per poter accedere alla risorsa, utilizzando i metodi offerti dal *gestore della risorsa*;
5. il **Business logic layer** costituito dal **Web server** o **Application server** che riveste il ruolo di *container* ed esegue la servlet che implementa il Web service;
6. il **Resource Management layer** costituito dal **server DBMS** che gestisce i dati memorizzati in modo persistente.

Un'architettura di questo tipo, definita **three-tier**, garantisce un ampio disaccoppiamento tra i vari elementi del sistema software, permettendo una diversa gestione della sicurezza sui diversi layer, rendendoli facilmente scalabili e manutenibili e permettendo aggiornamenti mirati per ogni tier.

## JDBC: Java DataBase Connectivity

**STUDIARE:** P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2017, pp.216-222.

- JDBC
- Utilizzare JDBC standalone
- Servlet con connessione a MySQL
  - [CRUD in Servlet](#)
  - [Interface PreparedStatement](#)
  - [Method executeQuery](#)
  - [Method executeUpdate](#)
  - [The finally Block](#)

**STUDIARE:** P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2017, pp.255-259.

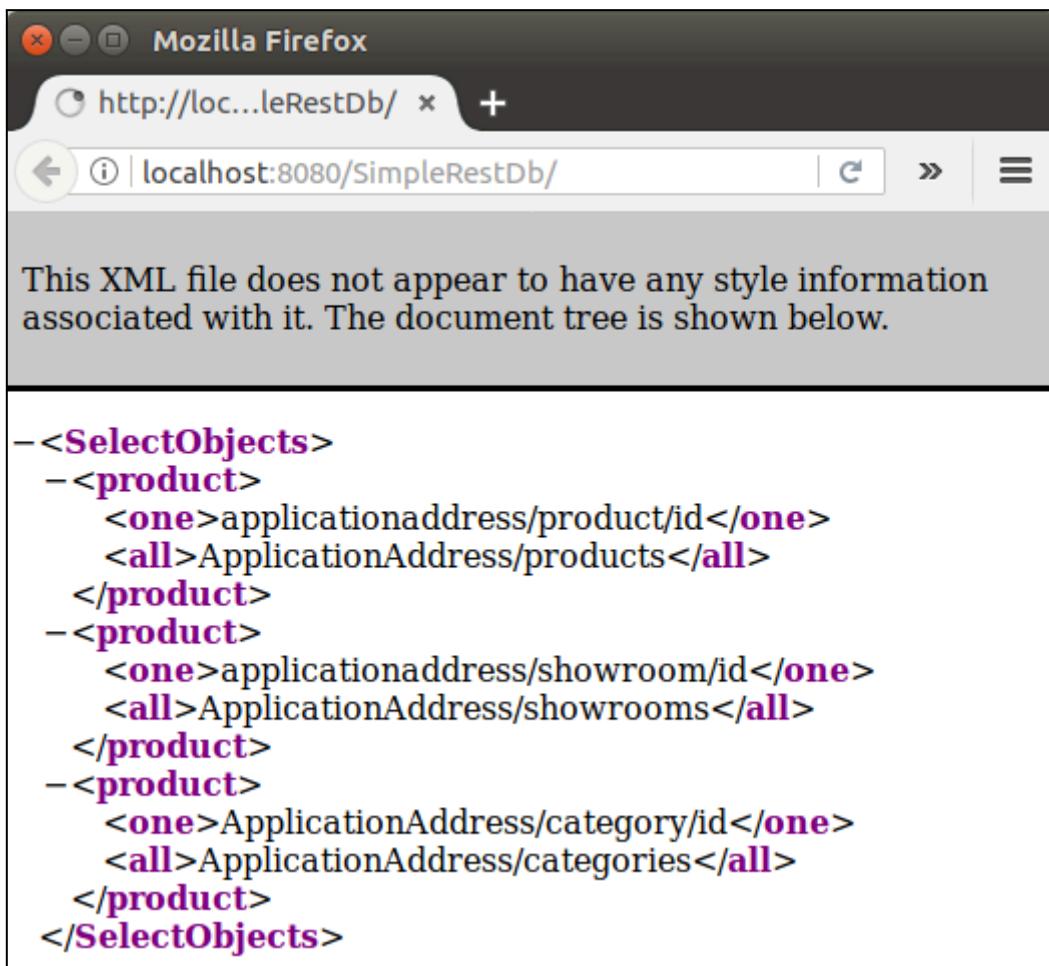
- Lab. 6 - JDBC e MySQL

## ShowRoom REST su database

Il seguente esempio mostra alcune funzionalità di un Web service REST che manipola i dati di un database (*EsempioREST*) di prodotti (*Products*) venduti presso diversi negozi (*Showrooms*) di una catena commerciale. I prodotti sono organizzati per categorie (*Categories*). Il Web service permetterà di interrogare il database utilizzando esclusivamente i metodi HTTP e un URL che identifichi la risorsa, come richiesto dall'architettura REST.

Il **server SimpleRestDb** è stato creato in modo tale da presentare all'avvio le modalità di costruzione degli URL per poter effettuare una richiesta di visualizzazione del contenuto di una tabella del database *EsempioREST*. Questa leggenda, fornita in formato XML,

permetterà di effettuare la visualizzazione dei contenuti delle singole tabelle del database, sia globalmente, sia in modo parametrico usando un identificatore.



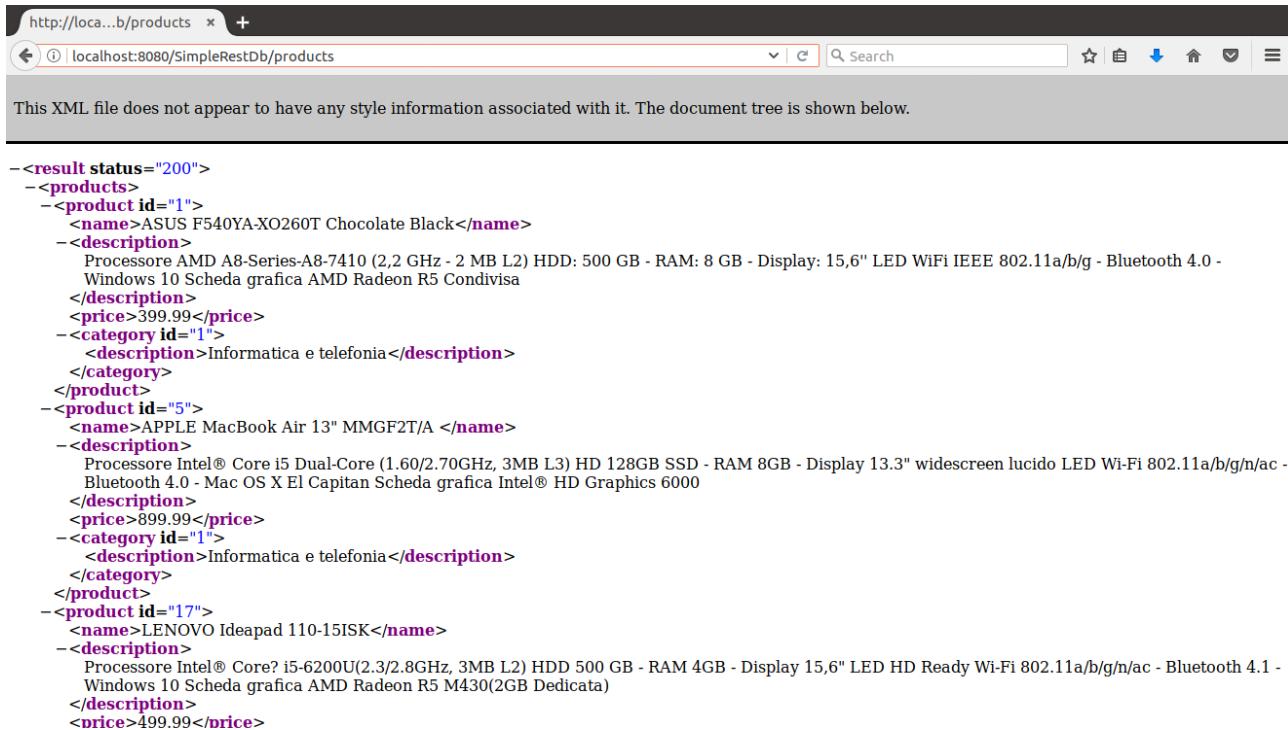
The screenshot shows a Mozilla Firefox browser window. The address bar displays "localhost:8080/SimpleRestDb/". The main content area shows an XML document structure:

```
- <SelectObjects>
  - <product>
    <one>applicationaddress/product/id</one>
    <all>ApplicationAddress/products</all>
  </product>
  - <product>
    <one>applicationaddress/showroom/id</one>
    <all>ApplicationAddress/showrooms</all>
  </product>
  - <product>
    <one>ApplicationAddress/category/id</one>
    <all>ApplicationAddress/categories</all>
  </product>
</SelectObjects>
```

Ad esempio, per visualizzare tutti i prodotti presenti nella tabella *Products* del database *EsempioREST* si dovrà utilizzare lo URL:

`http://localhost:8080/SimpleRestDb/products`

Il metodo GET invocato con questo URL verrà gestito dal Web service tramite il metodo *doGet()* che, effettuando un'analisi dello URL, selezionerà la visualizzazione di tutti i prodotti. Nell'immagine seguente viene riportata una porzione della risposta XML restituita dal Web service.



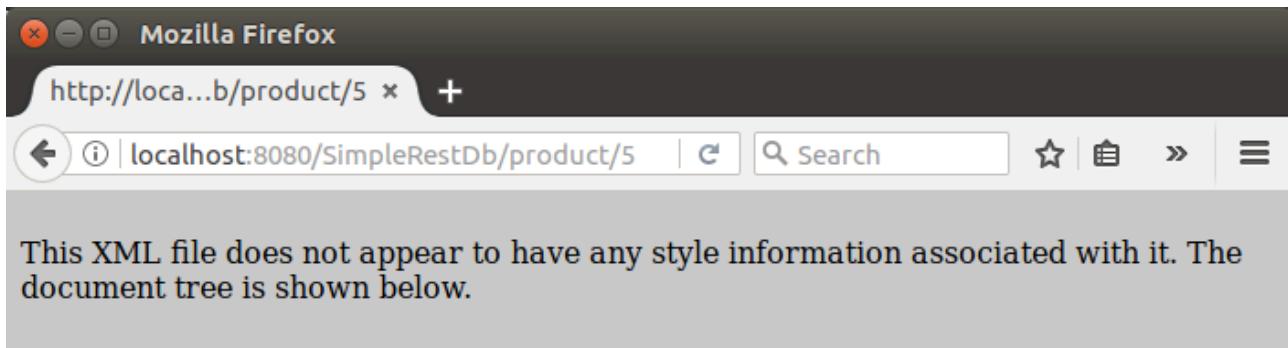
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<result status="200">
<products>
-<product id="1">
<name>ASUS F540YA-XO260T Chocolate Black</name>
-<description>
    Processore AMD A8-Series-A8-7410 (2,2 GHz - 2 MB L2) HDD: 500 GB - RAM: 8 GB - Display: 15,6" LED WiFi IEEE 802.11a/b/g - Bluetooth 4.0 -
    Windows 10 Scheda grafica AMD Radeon R5 Condivisa
-</description>
<price>399.99</price>
-<category id="1">
    <description>Informatica e telefonia</description>
-</category>
-</product>
-<product id="5">
<name>APPLE MacBook Air 13" MMGF2T/A </name>
-<description>
    Processore Intel® Core i5 Dual-Core (1.60/2.70GHz, 3MB L3) HD 128GB SSD - RAM 8GB - Display 13.3" widescreen lucido LED Wi-Fi 802.11a/b/g/n/ac -
    Bluetooth 4.0 - Mac OS X El Capitan Scheda grafica Intel® HD Graphics 6000
-</description>
<price>899.99</price>
-<category id="1">
    <description>Informatica e telefonia</description>
-</category>
-</product>
-<product id="17">
<name>LENOVO Ideapad 110-15ISK</name>
-<description>
    Processore Intel® Core i5-6200U(2.3/2.8GHz, 3MB L2) HDD 500 GB - RAM 4GB - Display 15,6" LED HD Ready Wi-Fi 802.11a/b/g/n/ac - Bluetooth 4.1 -
    Windows 10 Scheda grafica AMD Radeon R5 M430(2GB Dedicata)
-</description>
<price>499.99</price>
```

Invece, effettuando la richiesta di un singolo prodotto tramite il suo *id* utilizzando il seguente URL

<http://localhost:8080/SimpleRestDb/product/5>

si otterrà la visualizzazione del singolo prodotto, sempre in formato XML.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

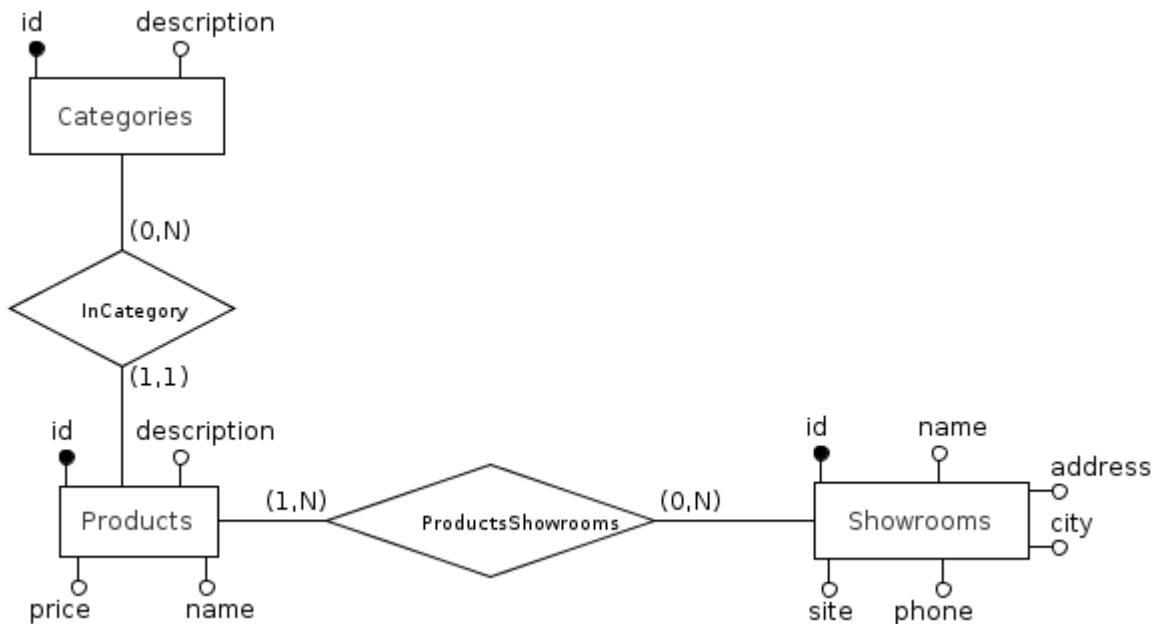
```
<result status="200">
-<product id="5">
<name>APPLE MacBook Air 13" MMGF2T/A </name>
-<description>
    Processore Intel® Core i5 Dual-Core (1.60/2.70GHz, 3MB L3) HD 128GB SSD -
    - RAM 8GB - Display 13.3" widescreen lucido LED Wi-Fi 802.11a/b/g/n/ac -
    - Bluetooth 4.0 - Mac OS X El Capitan Scheda grafica Intel® HD Graphics 6000
-</description>
<price>899.99</price>
-<category id="1">
    <description>Informatica e telefonia</description>
-</category>
-</product>
</result>
```

Per testare le operazioni di aggiunta (CREATE - POST), aggiornamento (UPDATE - PUT) e cancellazione (DELETE - DELETE) di una istanza, verrà utilizzato un semplice **client**, **RestClient**.

Nel seguito della trattazione verrà fornita una possibile implementazione del database, del Web service e del client.

## Database EsempioREST

Lo **schema concettuale**, semplificando la realtà gestita, è il seguente:



Analizzando lo schema E-R (concettuale) si deduce:

- un prodotto dell'entità *Products* è presente in una sola categoria dell'entità *Categories* e l'occorrenza è obbligatoria, cioè un prodotto non può essere privo di categoria;
- una categoria dell'entità *Categories* può contenere più prodotti, ma vista la cardinalità minima pari a zero di partecipazione alla relazione *InCategory*, una categoria potrebbe anche non avere prodotti (ad esempio quando una categoria è stata appena creata e nessun prodotto è stato ancora associato alla categoria);
- un prodotto dell'entità *Products* è venduto da almeno un negozio dell'entità *Showrooms* e l'occorrenza è obbligatoria, cioè un prodotto non può non essere venduto almeno in un negozio;
- un negozio dell'entità *Showrooms* può vendere più prodotti, ma vista la cardinalità minima pari a zero di partecipazione alla relazione *ProductsShowrooms*, un negozio potrebbe anche non avere prodotti in vendita (ad esempio quando un negozio è stato appena aperto e nessun prodotto è stato ancora associato al negozio);

Lo **schema logico** sarà il seguente:

`Categories(id, description)`

`Products(id, name, description, price, id_category)`

Showrooms(id, name, address, city, phone, site)

ProductsShowrooms(id\_products, id\_showroom)

Il **DDL** e **DML** della relativa **progettazione fisica** per la creazione del database, delle tabelle e della relativa popolazione delle stesse, sarà il seguente:

```
-- phpMyAdmin SQL Dump
-- version 4.5.4.1deb2ubuntu2
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Mar 28, 2017 at 09:59 PM
-- Server version: 5.7.17-0ubuntu0.16.04.1
-- PHP Version: 7.0.15-0ubuntu0.16.04.4

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `EsempioRest`
--

CREATE DATABASE IF NOT EXISTS `EsempioRest` DEFAULT CHARACTER SET latin1
COLLATE latin1_swedish_ci;
USE `EsempioRest`;

-----
-- Table structure for table `Categories`
--

CREATE TABLE `Categories` (
  `id` int(11) NOT NULL,
  `description` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- RELATIONS FOR TABLE `Categories`:
--



--
```

```
-- Dumping data for table `Categories`
--


INSERT INTO `Categories` (`id`, `description`) VALUES
(1, 'Informatica e telefonia'),
(2, 'TV e audio'),
(3, 'Fotografia, Auto e Navi'),
(4, 'Grandi Elettrodomestici'),
(5, 'Piccoli Elettrodomestici'),
(6, 'Gaming, Musica e Film'),
(7, 'Innovazione'),
(8, 'Tempo Libero'),
(9, 'Outlet');

-----


-- Table structure for table `Products`
--


CREATE TABLE `Products` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `description` varchar(500) DEFAULT NULL,
  `price` float(5,2) NOT NULL,
  `id_category` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- RELATIONS FOR TABLE `Products`:
--   `id_category`
--   `Categories` -> `id`


-- Dumping data for table `Products`
--


INSERT INTO `Products` (`id`, `name`, `description`, `price`, `id_category`)
VALUES
(1, 'ASUS F540YA-X0260T Chocolate Black', 'Processore AMD A8-Series-A8-7410\n(2,2 GHz - 2 MB L2)\r\nHDD: 500 GB - RAM: 8 GB - Display: 15,6\'\' LED\r\nWiFi IEEE 802.11a/b/g - Bluetooth 4.0 - Windows 10\r\nScheda grafica AMD Radeon R5 Condivisa', 399.99, 1),
(2, 'EPSON V11H664040 ', 'Videoproiettore con Tecnologia 3LCD\r\nContrasto di immagine: 15.000:1\r\nLuminosità 3000 Ansi Lumen\r\nRisoluzione: 1280x800 pixel\r\nImmagini brillanti in 2D e 3D', 449.99, 2),
(3, 'HOTPOINT FMG 723 B IT.M ', 'Lavatrice - Carica: Frontale - Tipo
```

installazione: Libera installazione (FS) - Classe energetica: A +++ - Capacità max carico: 7 kg - Velocità max centrifuga: 1.200 giri/min - Profondità: 52,2 cm', 279.99, 4),  
(4, 'HP 15-AC195NL - PRMG GRADING OOAN - SCONTO 10,00% ', 'Processore Intel® Core™ i7-5500U(2,4/3GHz - 4MB L3)\r\nHDD 1000 GB - RAM 8GB - Display 15,6" WLED HD Ready\r\nWi-Fi 802.11b/g/n - Windows 10\r\nScheda grafica AMD Radeon R5 M330(2GB Dedicata)\r\nPRODOTTO RICONDIZIONATO - PRMG GRADING OOAN\r\nConfezione originale integra (0)\r\nAccessori principali presenti (0)\r\nEstetica prodotto come nuovo (A)\r\nStato prodotto funzionante (N)', 719.99, 9),  
(5, 'APPLE MacBook Air 13" MMGF2T/A ', 'Processore Intel® Core i5 Dual-Core (1.60/2.70GHz, 3MB L3)\r\nHD 128GB SSD - RAM 8GB - Display 13.3" widescreen lucido LED\r\nWi-Fi 802.11a/b/g/n/ac - Bluetooth 4.0 - Mac OS X El Capitan\r\nScheda grafica Intel® HD Graphics 6000', 899.99, 1),  
(6, 'BRONDI FX-Compact Sport S ', 'Frequenza PMR - 8 Canali\r\n38 Codici per canali - Funzione VOX\r\nFunziona con 3 ministilo alcaline\r\nComunica con tutte le ricetrasmettenti PMR446', 19.99, 8),  
(7, 'MAJESTIC SD 247 RDS USB AX ', 'Digital Media Receiver Mechaless - Potenza 4x30W\r\nLettura MP3, WMA - Display ID3 Tag\r\nSintonizzatore RDS\r\nIngresso AUX - USB - SD/MMC Card', 34.99, 3),  
(8, 'SONY KD43XD8099B ', 'SMART TV LED 43" Ultra HD 4K - Risoluzione: 3840x2160\r\nTecnologia 400Hz - DLNA - Wi-fi\r\nTuner Digitale Terrestre DVB-T2 HEVC e Satellitare DVB-S\r\nProcessore Video 4K X-Reality Pro\r\nClasse efficienza energetica: B\r\nDistribuito da Sony Italia', 749.99, 2),  
(9, 'PANASONIC RP-WF830E-K ', 'Cuffie Stereo Wireless\r\nRisposta in frequenza 18-22.000 Hz\r\nDistanza di trasmissione fino a 100 m', 54.99, 2),  
(16, 'BOSCH PCQ715B90E ', 'Piano cottura - 5 Fuochi gas - Accensione ad 1 mano\r\nTermosicurezza - 2 Griglie in ghisa - Comandi frontali\r\nLarghezza: 70 cm', 299.99, 4),  
(17, 'LENOVO Ideapad 110-15ISK', 'Processore Intel® Core™ i5-6200U(2.3/2.8GHz, 3MB L2)\r\nHDD 500 GB - RAM 4GB - Display 15,6" LED HD Ready\r\nWi-Fi 802.11a/b/g/n/ac - Bluetooth 4.1 - Windows 10\r\nScheda grafica AMD Radeon R5 M430(2GB Dedicata)', 499.99, 1),  
(18, 'CANON EOS 1300D 18-55 IS', 'Fotocamera Reflex digitale - Sensore CMOS da 18 Megapixel\r\nLCD da 3" - Filmati Full HD - Slot SD/SDHC\r\nObiettivo 18-55 IS - Interfaccia USB - Peso: 485 g.\r\nGaranzia ufficiale Canon Italia', 379.99, 3),  
(19, 'KOENIC KSI 270 ', 'Ferro a vapore - Potenza max: 2.700 W\r\nCapacità serbatoio: 350 ml - Regolazione\r\nvapore - Funzione spray - Spegnimento\r\nnautomatico - Piastra: Ceramica rivestita', 25.99, 5),  
(21, 'MT DISTRIBUTION TEKKDRONE VAMPIRE PLUS', 'Minidrone - Camera HD da 2 megapixel\r\nGiroscopio 6 assi\r\nLuci LED per utilizzo notturno\r\nAutonomia 14/16 minuti\r\nDistanza di controllo 100 metri\r\nFunzione hovering - headless mode\r\nTasto one key return ', 89.99, 7),  
(22, 'SAMSUNG UE55KU6000KXZT', 'SMART TV LED 55" Ultra HD 4K - Risoluzione: 3840x2160\r\nTecnologia 1300 PQI - DLNA - WiFi + Ethernet\r\nTuner Digitale Terrestre DVB-T2\r\nClasse efficienza energetica: A\r\nDistribuito da Samsung Italia', 699.99, 2),

```
(23, 'SONY PS4 PRO 1TB ', 'Console fissa - 1000 GB HDD\r\nLettore BD/ DVD - Controller wireless\r\nBluetooth - HDMI - USB', 409.99, 6),
(24, 'BEKO RDSA240K20W', 'Frigorifero Doppia porta\r\nClasse di efficienza energetica: A\r\nCapacità netta frigo: 177 l\r\nCapacità netta congelatore: 46 l\r\nConsumo energia annuo: 226 kWh/anno\r\nSistema di raffreddamento frigo: Statico\r\nColore: Bianco', 259.99, 4),
(25, 'DE LONGHI EDG 100.W + 48 Capsule ', 'Macchina per caffè - Pressione: 15 bar - Potenza: 1460 W\r\nLeva di erogazione - Stand-by automatico - Sistema\r\nThermoblock - Cassetto raccoglie gocce regolabile in altezza\r\nFunzionamento esclusivo con cialde Nescafé Dolce Gusto', 49.99, 5);
```

---

```
-- Table structure for table `ProductsShowrooms`
```

```
--
```

```
CREATE TABLE `ProductsShowrooms` (
  `id_product` int(11) NOT NULL,
  `id_showroom` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--
```

```
-- RELATIONS FOR TABLE `ProductsShowrooms`:
```

```
--   `id_product`
--   `Products` -> `id`
--   `id_showroom`
--   `Showrooms` -> `id`
--
```

```
-- Dumping data for table `ProductsShowrooms`
```

```
--
```

```
INSERT INTO `ProductsShowrooms` (`id_product`, `id_showroom`) VALUES
(1, 1),
(5, 1),
(6, 1),
(7, 1),
(9, 1),
(1, 2),
(2, 2),
(6, 2),
(7, 2),
(9, 2),
(1, 3),
(2, 3),
(3, 3),
```

```
(7, 3),
(9, 3),
(2, 4),
(3, 4),
(4, 4),
(8, 4),
(3, 5),
(4, 5),
(5, 5),
(8, 5),
(4, 6),
(5, 6),
(6, 6),
(8, 6);
```

```
-----
```

```
--  
-- Table structure for table `Showrooms`  
--
```

```
CREATE TABLE `Showrooms` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `address` varchar(100) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(12) DEFAULT NULL,
  `site` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- RELATIONS FOR TABLE `Showrooms`:  
--
```

```
--  
-- Dumping data for table `Showrooms`  
--
```

```
INSERT INTO `Showrooms` (`id`, `name`, `address`, `city`, `phone`, `site`)
VALUES
(1, 'show1', 'address1', 'city1', '123456781', 'http://www.show1.com'),
(2, 'show2', 'address2', 'city2', '123456782', 'http://www.show2.com'),
(3, 'show3', 'address3', 'city3', '123456783', 'http://www.show3.com'),
(4, 'show4', 'address4', 'city4', '123456784', NULL),
(5, 'show5', 'address5', 'city5', NULL, 'http://www.show5.com'),
(6, 'show6', 'address6', 'city6', '123456786', 'http://www.show6.com');
```

```
--
```

```
-- Indexes for dumped tables
--
-- 
-- 
-- Indexes for table `Categories`
-- 
ALTER TABLE `Categories`
  ADD PRIMARY KEY (`id`);


-- 
-- Indexes for table `Products`
-- 
ALTER TABLE `Products`
  ADD PRIMARY KEY (`id`),
  ADD KEY `id_category`(`id_category`);


-- 
-- Indexes for table `ProductsShowrooms`
-- 
ALTER TABLE `ProductsShowrooms`
  ADD PRIMARY KEY (`id_product`, `id_showroom`),
  ADD KEY `id_showroom`(`id_showroom`);


-- 
-- Indexes for table `Showrooms`
-- 
ALTER TABLE `Showrooms`
  ADD PRIMARY KEY (`id`);


-- 
-- AUTO_INCREMENT for dumped tables
-- 
-- 
-- AUTO_INCREMENT for table `Categories`
-- 
ALTER TABLE `Categories`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10; 


-- 
-- AUTO_INCREMENT for table `Products`
-- 
ALTER TABLE `Products`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=26; 


-- 
-- AUTO_INCREMENT for table `Showrooms`
-- 
ALTER TABLE `Showrooms`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;
```

```
--  
-- Constraints for dumped tables  
--  
  
--  
-- Constraints for table `Products`  
--  
ALTER TABLE `Products`  
  ADD CONSTRAINT `Products_ibfk_1` FOREIGN KEY (`id_category`) REFERENCES  
`Categories` (`id`);  
  
--  
-- Constraints for table `ProductsShowrooms`  
--  
ALTER TABLE `ProductsShowrooms`  
  ADD CONSTRAINT `ProductsShowrooms_ibfk_1` FOREIGN KEY (`id_product`)  
REFERENCES `Products` (`id`),  
  ADD CONSTRAINT `ProductsShowrooms_ibfk_2` FOREIGN KEY (`id_showroom`)  
REFERENCES `Showrooms` (`id`);  
  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

# Esercizi

## Esercizi sui Thread

1. Sviluppare un progetto che permetta di sommare tutti gli elementi di una matrice di interi di dimensioni MxN. Sviluppare le seguenti versioni del progetto per confrontare le prestazioni delle soluzioni:

- a. una versione seriale che non usi i thread;
- b. una versione concorrente che usi un thread per riga, limitando ad un valore prestabilito il numero di thread in esecuzione;
- c. una versione concorrente che usi un thread per riga limitando il numero di thread al valore dei processori disponibili per la JVM.

Il thread principale deve sincronizzarsi sulla terminazione dei thread figli, tramite dei *join* multipli, e visualizzare poi il risultato finale della somma totale delle righe.

2. Sviluppare un progetto che permetta di cercare un elemento all'interno di una matrice quadrata. Si sviluppino le seguenti versioni del progetto per confrontare le prestazioni delle soluzioni:

- a. una versione seriale che non usi i thread;
- b. una versione concorrente che usi un thread per riga, limitando ad un valore prestabilito il numero di thread in esecuzione;
- c. una versione concorrente che usi un thread per riga limitando il numero di thread al valore dei processori disponibili per la JVM.

Il thread principale deve sincronizzarsi sulla terminazione dei thread figli, tramite dei *join* multipli, prima di terminare.

*[Richiesta la sincronizzazione per la gestione di un flag per segnalare il caso di elemento trovato]*

3. Sviluppare una nuova versione dell'esercizio della moltiplicazione tra matrici, mostrata negli appunti, che usi però un oggetto ExecutorService che generi un pool di thread pari al numero di processori disponibili alla JVM.

## Esercizi sui protocolli applicativi di rete

### Socket TCP e UDP

4. Modificare il server ToUpper presente negli appunti in modo tale che diventi un server concorrente usando il framework ExecutorService.

5. *Realizzare l'esercizio Prova adesso! sul libro di testo a p.140 creando un server concorrente usando il framework ExecutorService.*

6. *Realizzare l'esercizio 2 sul libro di testo a p.140 creando un server concorrente usando un Executor.*

7. *Realizzare l'esercizio 3 sul libro di testo a p.140 creando un server concorrente usando un Executor.*

8. Progettare un protocollo comunicativo che permetta di interrogare un server per ottenere la **data** e l'**ora** del server stesso, considerando la possibilità che possano

essere richiesti in diversi formati. La comunicazione deve avvenire utilizzando il protocollo TCP a livello di trasporto. Del protocollo comunicativo si dovranno definire:

- a. la tipologia di protocollo (*controllo connessione*: connection-oriented o connectionless; *affidabile*: confermato o non confermato; *controllo flusso*);
- b. la porta di ascolto del server (*indirizzamento*);
- c. se i dati devono essere frammentati prima della spedizione o meno (*frammentazione e riassemblaggio*);
- d. i comandi utilizzati per la richiesta e la risposta;
- e. il formato dei messaggi di richiesta e di risposta;
- f. la semantica dei campi di ogni messaggio;
- g. opportuni messaggi di errore inviati dal server.

Si facciano tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate.

9. Si vuole realizzare un programma per la gestione di un **gioco a quiz** online giocabile tramite un client desktop. Il gioco, al momento, non prevede una registrazione da parte dei giocatori, sarà sufficiente fornire solo un nick. Se il nick è già presente si avvisa l'utente e gli si impedisce l'ingresso al gioco. Il gioco inizia quando almeno tre utenti sono connessi, e consiste nel rispondere in modo corretto ad una domanda posta dal gestore. Vince chi per primo risponde in modo corretto. Se nessuno risponde, o non arrivano risposte corrette il gestore, a sua discrezione, chiude la domanda passando ad un'altra.

Viene tenuto il conto della situazione delle vincite per tutti gli utenti che nel tempo partecipano al gioco e, alla chiusura di una domanda i giocatori vengono aggiornati sul numero di vittorie di tutti.

Scrivere il protocollo di comunicazione fra il gestore e gli utenti, indicando i comandi testuali che viaggiano dal gestore ai client e viceversa, facendo per ognuno di essi un esempio. Si facciano tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate e si producano anche i diagrammi di sequenza per mostrare lo scambio temporale dei messaggi tra i client e il server.

Si ricordi di analizzare le seguenti voci: indirizzamento, frammentazione e riassemblaggio, incapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione.

10. Una grande azienda vuole gestire una **rubrica** di numeri telefonici interni tramite un servizio di rete a cui è possibile accedere per richiedere un numero telefonico partendo da un nominativo, o un nominativo fornendo il numero telefonico. Dovrà inoltre essere possibile aggiungere una nuova voce nell'elenco della rubrica, o modificarne uno già esistente.

L'azienda ha deciso, per ragioni interne, di non usare il protocollo HTTP per il trasporto dei dati, preferendo sviluppare in proprio un protocollo comunicativo privato che a livello di trasporto faccia uso del protocollo TCP.

Prescindendo dagli aspetti organizzativi del possibile database con cui si gestirà la rubrica telefonica e dell'applicazione di interfaccia utente che userà il protocollo, si progetti il protocollo comunicativo illustrato relazionando quanto segue:

- a. la descrizione del protocollo spiegando qual è lo scopo del protocollo;
- b. la porta di ascolto del server;
- c. il formato dei messaggi scambiati tra client e server con una breve spiegazione della funzione di ogni messaggio;

- d. la descrizione delle caratteristiche del protocollo: indirizzamento, frammentazione e riassemblaggio, encapsulamento, connection-oriented o connectionless, servizio affidabile o non affidabile, controllo degli errori, controllo del flusso, multiplexing e demultiplexing, servizi di trasmissione (qualità del servizio, priorità, sicurezza);
- e. un diagramma della sequenza temporale di scambio dei messaggi<sup>21</sup>.

11. Il sistema informatico del **magazzino** di un grande **sito web di e-commerce** mantiene per ogni prodotto in vendita, identificato da un codice numerico, la quantità disponibile.

Progettare un protocollo di comunicazione tra client e server, basata su TCP, che debba prevedere le seguenti operazioni:

- a. a richiesta del client il server deve restituire la quantità presente in magazzino di un prodotto;
- b. a richiesta del client il server deve modificare la quantità di un prodotto specificato.

Del protocollo comunicativo si dovranno definire:

- a. la tipologia di protocollo (*controllo connessione*: connection-oriented o connectionless; *affidabile*: confermato o non confermato; *controllo flusso*);
- b. la porta di ascolto del server;
- c. i comandi utilizzati per la richiesta e la risposta;
- d. il formato dei messaggi di richiesta e di risposta;
- e. la semantica dei campi di ogni messaggio.

Si prevedano anche opportuni messaggi di errore inviati dal server.

Si facciano tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate e si producano anche i diagrammi di sequenza per mostrare lo scambio temporale dei messaggi tra i client e il server.

Si ricordi di analizzare le seguenti voci: indirizzamento, frammentazione e riassemblaggio, encapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione.

12. Un piccolo **cinema** dispone di 3 sale con 100 posti ciascuna ed è necessario realizzare il servizio di prenotazione dei posti per i giorni festivi in cui in ciascuna delle sale sono proiettati 3 film, rispettivamente alle ore 17:30 (I spettacolo), alle ore 20:00 (II spettacolo) ed alle ore 22:30 (III spettacolo). Il servizio da realizzare consiste in un server TCP concorrente che risponde sulla porta 12345 alle seguenti richieste da parte dei client:

- a. elenco dei posti liberi per uno spettacolo in una certa sala;
- b. prenotazione di un elenco di posti liberi per uno spettacolo in una certa sala e per un certo spettacolo. Per questo caso il server dovrà restituire due tipologie di messaggi diversi in caso di successo o insuccesso.

Del protocollo comunicativo si dovranno definire:

- a. la tipologia di protocollo (*controllo connessione*: connection-oriented o connectionless; *affidabile*: confermato o non confermato; *controllo flusso*);
- b. la porta di ascolto del server;
- c. i comandi utilizzati per la richiesta e la risposta;

---

<sup>21</sup> Il sequence diagram è stato creato usando [SequenceDiagramOrg](#)

- d. il formato dei messaggi di richiesta e di risposta;
- e. la semantica dei campi di ogni messaggio.

Si prevedano anche opportuni messaggi di errore inviati dal server.

Progettare il protocollo di comunicazione giustificando opportunamente tutte le ipotesi aggiuntive.

### 13. Lavoro di laboratorio - Cambio valuta<sup>22</sup>

Si vuole implementare un server che gestisca il cambio di valuta euro/dollaro americano, che utilizzi a livello di trasporto i servizi offerti dal protocollo TCP. L'architettura deve essere client-server, e il server dovrà accettare le richieste provenienti da un client, utilizzando un protocollo che preveda i comandi elencati di seguito per effettuare le operazioni indicate, inviando la conversione al client. Il server deve gestire il multithreading utilizzando il framework ExecutorService.

#### Livello 1 - Voto massimo 6,5

Il client invierà una sola richiesta e, una volta ricevuta la conversione concluderà la comunicazione.

La parte di protocollo lato client prevede i seguenti comandi:

Comando	Operazione
EUR,num	L'ammontare <i>num</i> espresso in euro deve essere convertito in dollari americani
USD,num	L'ammontare <i>num</i> espresso in dollari americani deve essere convertito in euro

Il server, una volta ricevuta la richiesta dal client invierà la conversione.

La parte di protocollo lato server prevede i seguenti comandi:

Comando	Operazione
USD,num	L'ammontare <i>num</i> espresso in dollari americani è la conversione di un valore in euro richiesto precedentemente dal client
EUR,num	L'ammontare <i>num</i> espresso in euro è la conversione di un valore in dollari americani richiesto precedentemente dal client

#### Livello 2 - Voto massimo 8,5

Il client potrà inviare più richieste aspettando però di ricevere una risposta dal server prima di inviare la richiesta successiva ([stop-and-wait](#)). Il client comunicherà al server la conclusione della comunicazione con un comando specifico.

La parte di protocollo lato client prevede i seguenti comandi:

Comando	Operazione
---------	------------

<sup>22</sup> Laboratorio sviluppato da un'idea del Prof. Marco Sammartino.

EUR, <i>num</i>	L'ammontare <i>num</i> espresso in euro deve essere convertito in dollari americani
USD, <i>num</i>	L'ammontare <i>num</i> espresso in dollari americani deve essere convertito in euro
STP	Il client segnala al server che non vuole più effettuare delle conversioni

Il server, una volta ricevuta la richiesta dal client invierà la conversione, ma in caso di valuta non riconosciuta, segnalerà un errore al client.

La **parte di protocollo lato server** prevede i seguenti comandi:

Comando	Operazione
USD, <i>num</i>	L'ammontare <i>num</i> espresso in dollari americani è la conversione di un valore in euro richiesto precedentemente dal client
EUR, <i>num</i>	L'ammontare <i>num</i> espresso in euro è la conversione di un valore in dollari americani richiesto precedentemente dal client
ERR,VAL	La valuta <i>VAL</i> inviata dal client non corrisponde a quelle conosciute (EUR o USD)

### **Livello 3 - Voto massimo 10**

Il client potrà inviare più richieste aspettando però di ricevere una risposta dal server prima di inviare la richiesta successiva ([stop-and-wait](#)), e comunicherà al server la conclusione della comunicazione con un comando specifico. In questo caso però il cambio di valuta deve essere gestito tra valute diverse (almeno tre), non solo euro/dollaro americano.

Tutte le valute devono essere espresse con tre caratteri, come specificato dallo standard [ISO 4217](#)<sup>23</sup>, e il client dovrà richiedere al server per prima cosa l'elenco delle valute che gestisce, successivamente potrà richiedere il cambio valuta.

La **parte di protocollo lato client** prevede i seguenti comandi:

Comando	Operazione
LST	Richiesta dell'elenco delle valute gestite dal server.
VALout,VALin, <i>num</i>	L'ammontare <i>num</i> espresso in una delle valute gestite dal server (VALin) deve essere convertito in un'altra valuta (VALout).
STP	Il client segnala al server che non vuole più effettuare delle conversioni.

<sup>23</sup> [ISO 4217](#) on Wikipedia, and [Exchange Rates](#).

Il server, una volta ricevuta la richiesta di elenco delle valute che gestisce, dovrà inviarla al client, ponendosi in attesa di ricevere o una richiesta di conversione, o il comando di chiusura della comunicazione dal client. Nel caso di valuta non riconosciuta, segnalerà un errore al client.

La **parte di protocollo lato server** prevede i seguenti comandi:

Comando	Operazione
<i>VAL1,VAL2,VAL3,...,END</i>	Spedizione al client dell'elenco di valute gestite dal server; l'elenco deve essere terminato dal comando END.
<i>VALout,VALin,num</i>	L'ammontare <i>num</i> è espresso nella valuta di conversione <i>VALout</i> , risultato di una richiesta di cambio dalla valuta <i>VALin</i> .
<i>ERR,VAL</i>	La valuta <i>VAL</i> inviata dal client non corrisponde a quelle conosciute.

Il candidato, dopo aver valutato le diverse versioni del protocollo, ne scelga una e proceda con il relativo sviluppo creando due progetti separati, uno per il server e uno per il client. È possibile procedere in modo incrementale, partendo dal *Livello 1* e procedendo verso gli altri livelli. La gestione dell'interfaccia utente, che non fa parte dei protocolli proposti, dovrà essere una scelta implementativa del candidato, opportunamente documentata nell'analisi del progetto. In ogni caso l'interfaccia utente dovrà essere completamente separata dalla gestione della comunicazione tra client e server.

Al termine del lavoro il candidato dovrà consegnare una relazione contenente:

- l'analisi del proprio lavoro giustificando le scelte implementative effettuate;
- il diagramma delle classi dei due progetti del server e del client;
- il diagramma di sequenza dell'interazione tra il client e il server che enfatizzi lo scambio di pacchetti protocollare tra le due entità;
- il codice dei due progetti, server e client, opportunamente commentato.

#### 14. Lavoro di laboratorio - Banca

Si vuole implementare un server che gestisca i saldi dei conti correnti bancari, che utilizzi a livello di trasporto i servizi offerti dal protocollo TCP. L'architettura deve essere client-server, e il server dovrà accettare le richieste provenienti da un client, utilizzando un protocollo che preveda i seguenti comandi per effettuare le operazioni indicate:

Comando	Operazione
<i>U,n</i>	Il conto <i>n</i> diventa quello su cui saranno effettuate le prossime operazioni
<i>V,num</i>	Versamento sul conto in uso della somma <i>num</i>
<i>P,num</i>	Prelievo sul conto in uso della somma <i>num</i>

S	Richiesta di invio del saldo del conto in uso
---	---

Il server risponde con un messaggio di errore se il comando non è fornito secondo la sintassi corretta o non è riconosciuto.

### 15. Lavoro di laboratorio - Calcolatrice TCP<sup>24</sup>

Si vuole implementare una calcolatrice che utilizzi a livello di trasporto i servizi offerti dal protocollo TCP. L'architettura deve essere client-server, e il server dovrà accettare le richieste provenienti da un client, utilizzando un protocollo che preveda i seguenti comandi per effettuare le operazioni indicate:

Comando	Operazione
ADD,X,Y	Addizione: $(X+Y)$
SUB,X,Y	Sottrazione $(X-Y)$
MUL,X,Y	Moltiplicazione $(X * Y)$
DIV,X,Y	Divisione $(X : Y)$
POW,X,Y	Potenza $(X^Y)$

Tab.1

La calcolatrice risponde con un messaggio di errore se il comando non è fornito secondo la sintassi corretta.

#### I Parte Facoltativa:

Il protocollo prevede che il client possa richiedere al server le seguenti funzionalità di memoria:

Comando	Operazione
MEM,M1,X	Assegna: $(M1=X)$
MEM,M2,X	Assegna: $(M2=X)$
STORE,M1	Memorizza in M1 risultato ultima operazione
STORE,M2	Memorizza in M2 risultato ultima operazione
CLEAR,M1	$M1=0$
CLEAR,M2	$M2=0$

Tab.2

Inoltre in tutti i comandi della tabella 1 gli argomenti X e Y possono essere sostituiti con M1 ed M2, p.e. POW,3,M1 , SUB,M1,M2 , MUL,M2,-5 sono tutti comandi accettabili.

<sup>24</sup> Lavoro di laboratorio ideato dal Prof. Massimo Papa

**II Parte Facoltativa:**

Costruire il server progettandolo per gestire più client contemporaneamente utilizzando l'architettura a thread.

**III Parte Facoltativa:**

Il protocollo prevede la gestione del calcolo di un'operazione suddivisa tra due client. Un client potrà richiedere di eseguire un calcolo utilizzando un operando fornito da un altro client. Entrambi i client dovranno appoggiarsi al server.

I comandi previsti dal protocollo che il client utilizzerà per questa ulteriore funzionalità sono elencati in tabella 3:

Comando
ADD,X,IP
SUB,X,IP
MUL,X,IP
DIV,X,IP
POW,X,IP
STORE,M1,IP
STORE,M2,IP

Tab.3

Per il significato dei comandi e del primo operando si rimanda alle specifiche indicate nei precedenti punti.

IP può essere di due tipi:

1. Indirizzo IP di una macchina su cui gira un client;
2. ANY.

Nel primo caso il server si mette in attesa di una richiesta analoga dal client con indirizzo IP o gestirà subito il calcolo se il client con indirizzo IP ha già mandato al server il comando indicando l'IP del primo client.

p.e. Primo client (ip=192.168.1.3): ADD,3,192.168.1.10 - Secondo client(ip=192.168.1.10): ADD,M1,192.168.1.3

In ogni caso il server verifica la correttezza formale dei comandi inviati.

Se il secondo client non risponde si preveda un timeout trascorso il quale il server manderà un messaggio di errore al primo client e eliminerà dalla sua coda il comando di tale client.

Nel secondo caso il server si mette in attesa di una richiesta analoga da un qualsiasi client, o gestisce subito il calcolo se un client ha già mandato al server il comando, indicando nel campo IP ANY.

p.e. Primo client (ip=192.168.1.3) : ADD,3,ANY - Secondo client(ip=192.168.1.10): ADD,M1,ANY

In ogni caso il server verifica la correttezza formale dei comandi inviati.

Se nessun client risponde prevedere un timeout trascorso il quale il server manderà un messaggio di errore al primo client e eliminerà dalla sua coda il comando di tale client.

Porre attenzione alla gestione dei comandi con lo stesso indirizzo IP (più client con lo stesso indirizzo IP che effettuano una richiesta allo stesso server).

I risultati del calcolo devono essere inviati a tutti e due i client coinvolti.

16. Con riferimento al protocollo "Calcolatrice TCP" esposto nell'esercizio precedente, si risponda alle seguenti domande.

- a. Descrivere come viene inserito l'input da parte dell'utente.
- b. Indicare quali tipi di numeri gestisce il protocollo.
- c. Spiegare cosa avviene se l'utente inserisce un'operazione non consentita e chi gestirà l'errore.
- d. Spiegare cosa avviene se l'utente inserisce un operando non consentito e chi gestirà l'errore.
- e. Indicare in quali situazioni un client può non ricevere una risposta dal server.
- f. Indicare quante memorie M1 e M2 sono previste, decidendo se ogni client avrà una coppia di memorie disponibili o se saranno previste solo due memorie per gestire tutti i client.
- g. Descrivere come gestirete le richieste dei client che prevedono un indirizzo IP come operando. Nello specifico si indichi quale struttura dati si pensa di utilizzare per memorizzare tali richieste e cosa verrà memorizzato in essa.
- h. Indicare se il protocollo proposto è connection-oriented o connectionless. Al riguardo si dovrà pensare se il protocollo prevede una fase di gestione della connessione fra il client e il server.
- i. Indicare se il protocollo proposto è confermato o non è confermato. Al riguardo si dovrà pensare se il protocollo prevede una forma di conferma a livello applicativo per ogni PDU inviato dal client o dal server.
- j. Indicare se il protocollo proposto effettua encapsulamento. Al riguardo si dovrà pensare se il protocollo prevede un header/trailer per salvare delle informazioni utili per la comprensione della gestione del payload.
- k. Indicare se il protocollo proposto usa una tecnica di indirizzamento a livello applicativo. Al riguardo si dovrà pensare se il protocollo prevede l'aggiunta di un header in cui viene posta una forma di indirizzamento a livello applicativo.
- l. Indicare se il protocollo proposto effettua controllo del flusso. Al riguardo si dovrà pensare se un client può inviare più richieste di operazioni prima di ricevere il primo risultato, o se invece deve attendere un risultato prima di inviare una nuova richiesta.
- m. Indicare se il protocollo effettua il controllo degli errori. Al riguardo si dovrà pensare se il protocollo prevede una forma di controllo degli errori, sia a livello client, sia a livello server.
- n. Indicare se il protocollo effettua frammentazione e riassemblaggio. Al riguardo si dovrà pensare se il protocollo necessita di frammentare i PDU prima di effettuare la spedizione, e quindi se è necessario riassemblare i PDU.

- o. Indicare se il protocollo effettua multiplexing e demultiplexing. Al riguardo si dovrà pensare se il protocollo usa a livello applicativo una tecnica che permette a più connessioni di confluire in una sola connessione a livello inferiore.
- p. Indicare se il protocollo prevede servizi alla trasmissione, come qualità del servizio, gestione delle priorità o gestione della sicurezza.

## 17. Lavoro di laboratorio - Servizio ToUpLowUDP

### Livello 1 - TuUpperUDP

Modificare il protocollo ToUpper UDP presente negli appunti in modo tale che:

- a. ci sia una fase di connessione tra client e server costituita dall'invio, da parte del client, della prima stringa da convertire, e il server se accetterà la connessione semplicemente invierà la stringa convertita;
- b. il client possa inviare più stringhe prima di chiudere la connessione, ma dovrà aspettare di ricevere la conversione da parte del server prima di inviarne un'altra (stop-and-wait);
- c. quando il client avrà terminato le conversioni necessarie, dovrà chiudere la connessione inviando un messaggio particolare al server, che non possa essere confuso con una stringa da convertire; per non confondere un messaggio di conversione con uno di terminazione della connessione si dovrà prevedere un formato/contenuto dei messaggi che permetta di distinguere i due casi;
- d. se un messaggio non è una richiesta di conversione o la terminazione della connessione il server dovrà inviare una segnalazione di errore al client e chiudere la connessione.

Osservazioni sul nuovo protocollo:

- è confermato in quanto la conversione inviata dal server funge anche da conferma di ricezione;
- effettua anche il controllo di flusso in quanto implementando una tecnica di stop-and-wait per gestire la comunicazione tra client e server, di fatto presuppone che la dimensione del buffer di comunicazione del client e del server sia di un solo messaggio;
- effettua una gestione degli errori in quanto rileva il caso di un tipo di messaggio non interpretabile e agisce di conseguenza, chiudendo la connessione.

Consegnare l'intera cartella del progetto (NetBeans o IntelliJ) con una breve relazione che contenga il diagramma delle classi e spieghi le scelte implementative effettuate, soprattutto relative ai punti (c) e (d).

La relazione dovrà contenere le seguenti parti:

- la descrizione del protocollo spiegando qual è lo scopo del protocollo;
- la porta di ascolto del server;
- una tabella con il formato dei messaggi scambiati tra client e server con una breve spiegazione della funzione di ogni messaggio del client e del server;

- la descrizione delle caratteristiche del protocollo: indirizzamento, frammentazione e riassemblaggio, encapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing e demultiplexing, servizi di trasmissione (qualità del servizio, priorità, sicurezza);
- spiegazione della soluzione trovata relativa alla gestione della connessione e alla gestione dell'errore;
- descrizione della funzione delle diverse classi usate nel client e nel server.

### **Griglia di valutazione.**

Voti nell'intervallo [1 - 10]. Punteggio nell'intervallo [0 - 17]. Punti per la sufficienza 10,20.

Obiettivi valutativi:

- [PUNTI 4] analisi;
- [PUNTI 3] server: thread [0,5], framework Executor Service [0,5], chiusura server [0,5], DatagramSocket e DatagramPacket [0,5], funzionamento generale [1];
- [PUNTI 2] client: thread [0,5], ciclo trasmissione [1], DatagramSocket e DatagramPacket [0,5];
- [PUNTI 8] implementazione del protocollo: a. apertura connessione [2], b. controllo del flusso [2], c. chiusura connessione [2], d. errore segnalato dal server [2].

### **Livello 2 - ToUpLowUDP**

Si preveda di variare il protocollo in modo tale che:

- a. un client possa chiedere non solo la conversione in maiuscolo, ma anche in minuscolo, usando due comandi diversi del protocollo;
- b. un client possa inviare più messaggi da convertire, senza attendere che il server gli invii la conversione;
- c. il formato del messaggio inviato e la relativa risposta dovranno permettere di distinguere i diversi pacchetti di richiesta di conversione attraverso un campo aggiuntivo;
- d. la spedizione del messaggio convertito dovrà ancora fungere da conferma, ma il client dovrà ricavare in qualche modo quale messaggio è stato confermato;
- e. il client e il server dovranno avere un buffer dei messaggi che gli permetta di mantenere in memoria tutti i messaggi ancora da confermare per il client, e ancora da convertire per il server;
- f. se il server dovesse ricevere un numero di messaggi superiore alla capacità del suo buffer dovrà scartare quelli in eccesso, mandare una segnalazione di errore al client e chiudere la connessione.

Osservazioni sul nuovo protocollo:

- Il protocollo dovrà distinguere una richiesta di conversione in maiuscolo da una richiesta di conversione in minuscolo quindi si amplia l'insieme di comandi del protocollo;
- la conferma deve essere mirata ad uno specifico messaggio convertito, individuabile in modo preciso sia dal client, sia dal server;
- si presuppone che il buffer di comunicazione del client e del server non sia più di un solo messaggio, come nella versione del *Livello 1*; in pratica il client dovrà usare una struttura dati in cui memorizzare i messaggi inviati e non ancora confermati e, analogamente il server dovrà usare una struttura dati per gestire tutti i messaggi inviati dal client che ancora non ha convertito;
- la dimensione della struttura dati dei messaggi del client e del server indica il flusso massimo di messaggi gestibili durante una comunicazione, quindi viene modificato il controllo di flusso che ora sarà basato sulla dimensione del buffer;
- il protocollo amplia la gestione degli errori prevedendo la segnalazione di superamento della quota massima di messaggi inviabili da parte del client, con conseguente chiusura della connessione.

Per provare il funzionamento del protocollo potete attivare un ritardo nel server alla ricezione di ogni messaggio da convertire in modo tale che possiate fare più invii con il client. Si veda a questo proposito il metodo statico [sleep\(\)](#)<sup>25</sup> di Java o si usi il framework [ScheduledExecutorService](#) al posto del framework ExecutorService nel server.

Consegnare l'intera cartella del progetto (NetBeans o IntelliJ) con una breve relazione che contenga la spiegazione del protocollo implementato, indicandone la funzione ed elencando e descrivendo i messaggi usati con il loro formato.

La relazione dovrà contenere le seguenti parti:

- la descrizione del protocollo spiegando qual è lo scopo del protocollo;
- la porta di ascolto del server;
- una tabella con il formato dei messaggi scambiati tra client e server con una breve spiegazione della funzione di ogni messaggio del client e del server;
- la descrizione delle caratteristiche del protocollo: indirizzamento, frammentazione e riassemblaggio, encapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing e demultiplexing, servizi di trasmissione (qualità del servizio, priorità, sicurezza);
- spiegazione della soluzione trovata relativa alla gestione del buffer del client e/o del server e delle modalità di invio dei messaggi e delle conferme;
- descrizione della funzione delle diverse classi usate nel client e nel server.

### ***Livello 3***

Si preveda di variare il protocollo in modo tale che:

- a. il client avvii un timer per ogni richiesta inviata al server, che indichi il tempo massimo di attesa per ricevere una conversione/conferma;

---

<sup>25</sup> Vedere anche [Thread.sleep\(\) in Java – Java Thread sleep - JournalDev](#).

- b. allo scadere del timer, se il client non ha ricevuto la risposta da parte del server dovrà inviare nuovamente la stringa da convertire, e questa operazione sarà ripetuta per ogni messaggio al massimo per tre volte;
- c. se tutti i timer dei tre invii di ogni messaggio dovessero scadere senza ricevere una risposta, il client dovrà chiudere la connessione con il server segnalando all'utente un problema di comunicazione.

Osservazioni sul nuovo protocollo:

- è affidabile perché cerca di recuperare un problema di comunicazione permettendo il reinvio di un messaggio eventualmente perso o scartato;
- amplia la gestione degli errori prevedendo la gestione della perdita di un messaggio, di richiesta o di risposta, limitando però i tentativi di comunicazione in modo da non lasciare il client in attesa di una risposta dal server per un tempo indefinito.

Per provare il funzionamento del protocollo potete attivare un ritardo nel server alla ricezione di ogni messaggio in modo tale che mandi in scadenza i timer del client, simulando in questo modo un ritardo nella comunicazione. Si veda a questo proposito il metodo statico [sleep\(\)](#)<sup>26</sup> di Java o si usi il framework [ScheduledExecutorService](#) al posto del framework ExecutorService nel server.

Consegnare l'intera cartella del progetto (NetBeans o IntelliJ) con una breve relazione che contenga il diagramma delle classi e spieghi le scelte implementative effettuate per i punti (a), (b) e (c).

18. Progettare un protocollo comunicativo che permetta di interrogare un server **time** che risponda con un datagram UDP contenente il numero di secondi trascorsi dal 1 gennaio 1970, per qualsiasi richiesta ricevuta da parte di un client, indipendentemente dal contenuto del datagram inviato dal client stesso.

In base a quanto stabilito dal contesto presentato si decida quali elementi di progettazione, fra i seguenti, è necessario considerare:

- a. indirizzamento;
- b. frammentazione e riassemblaggio;
- c. encapsulamento;
- d. controllo della connessione;
- e. servizio confermato o non confermato;
- f. controllo degli errori;
- g. controllo del flusso;
- h. multiplexing;
- i. servizi di trasmissione.

Si facciano tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate.

19. Visto l'avvicinarsi delle vacanze di Natale avete deciso di fare un regalo ai vostri familiari progettando un protocollo comunicativo che vi permetta di condividere libri in formato testuale con qualsiasi componente della famiglia, anche se distante. Il protocollo lo utilizzerete successivamente per creare un'applicazione di scambio di

---

<sup>26</sup> Vedere anche [Thread.sleep\(\) in Java – Java Thread sleep - JournalDev](#).

documenti di cui state ancora progettando l'interfaccia grafica.

Il vostro protocollo, nominato **Simple Text Transfer Protocol (STTP)**, dato che sarà usato solo in un ambiente di cui vi fidate, userà i servizi del protocollo UDP a livello di trasporto e non prevederà alcuna forma di autenticazione, dovrà solo permettere ai client (i vostri familiari) di leggere e scrivere i file nel server (il vostro computer).

Il protocollo **STTP** volete che sia connesso e affidabile, e tutti gli host dovranno gestire un buffer di trasmissione e ricezione di un solo **APDU** (*Application Protocol Data Unit*).

Inoltre, per non consumare eccessivamente la banda trasmittiva con un'unica spedizione, decidete di frammentare il file.

Dato che il protocollo volete che sia semplice, qualsiasi errore dovrà terminare la connessione, e in questo caso il file dovrà essere ri-trasmesso interamente. Il protocollo comunque dovrà segnalare al client la tipologia di errore, quando possibile.

Si progetti il protocollo **STTP** facendo tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate e si producano anche i diagrammi di sequenza per mostrare lo scambio temporale dei messaggi tra i client e il server.

Si ricordi di analizzare le seguenti voci: indirizzamento, frammentazione e riassemblaggio, incapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione.

20. Si analizzi in modo dettagliato il protocollo **TFTP** leggendo il relativo [RFC 1350](#) che fornisce tutte le specifiche.

Si produca un documento di sintesi che riporti la modalità di gestione per ognuna delle seguenti voci:

- a. protocollo utilizzato a livello di trasporto;
- b. indirizzamento;
- c. frammentazione e riassemblaggio;
- d. incapsulamento;
- e. controllo della connessione;
- f. servizio confermato o non confermato;
- g. controllo degli errori;
- h. controllo del flusso;
- i. multiplexing;
- j. servizi di trasmissione.

Il protocollo potrebbe essere oggetto del prossimo laboratorio.

21. Un **circolo nautico** utilizza un dispositivo per interfacciare un **sensore** in grado di **misurare velocità** (Km/h) e **direzione** (°) del **vento**. Il produttore del sensore rende disponibile un driver che permette di acquisire i valori misurati ogni 5 secondi, in modo tale da permettere la memorizzazione dei dati per poi poterli usare in base alle necessità dell'utilizzatore del sensore.

Una società che offre, fra le altre cose, informazioni metereologiche, intende realizzare un **server UDP** attivo sulla **porta 54321** per **rendere disponibili** in rete **i valori misurati dal sensore**.

Il **server** dovrà **fornire le informazioni** sul vento **in un intervallo di tempo indicato nella richiesta del client**.

Il **protocollo** di comunicazione dovrà essere **affidabile** e **orientato alla connessione**.

Si progetti il protocollo facendo tutte le ipotesi aggiuntive ritenute necessarie, purché

giustificate e si producano anche i diagrammi di sequenza per mostrare lo scambio temporale dei messaggi tra i client e il server.

Si ricordi di analizzare le seguenti voci: indirizzamento, frammentazione e riassemblaggio, incapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione.

22.Un drone espone la **propria posizione** rappresentandola attraverso **3 valori floating-point a 32 bit**, che indicano la **latitudine**, la **longitudine** e l'**altitudine**. Per fare ciò il **drone** implementa un **server UDP** che risponde alle richieste ricevute sulla **porta 9999**. Le postazioni **client** a terra che vogliono sapere qual è la posizione del drone, dovranno **inviare una richiesta al drone** che dovrà **rispondere entro un secondo**, altrimenti la richiesta si considera fallita. Si progetti il protocollo facendo tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate e si producano anche i diagrammi di sequenza per mostrare lo scambio temporale dei messaggi tra i client e il server.

Si ricordi di analizzare le seguenti voci: indirizzamento, frammentazione e riassemblaggio, incapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione.

23.I **dispositivi domotici** (punti luce, elementi di condizionamento climatico, prese di alimentazione per elettrodomestici, ecc.) possono essere connessi anche alla rete WiFi e vengono **comandati** mediante **datagrammi UDP** indirizzati alla **porta 23365**.

Tutti i dispositivi possono accettare due tipi di comandi:

- un comando per **accendere** il dispositivo **dopo un certo numero di secondi**;
- un comando per **spegnere** il dispositivo **dopo un certo numero di secondi**.

Per entrambi i comandi si pensi anche ad una modalità per effettuare l'**accensione** o lo **spegnimento immediato** appena ricevuto il relativo comando.

Il **protocollo** di comunicazione deve essere **confermato**, quindi il dispositivo che ha ricevuto uno dei due comandi precedenti dovrà inviare una conferma al programma mittente. Inoltre si dovrà prevedere un **controllo di errore** nel caso il comando sia rifiutato per qualsiasi ragione.

Si progetti il protocollo facendo tutte le ipotesi aggiuntive ritenute necessarie, purché giustificate e si producano anche i diagrammi di sequenza per mostrare lo scambio temporale dei messaggi tra i client e il server.

Si ricordi di analizzare le seguenti voci: indirizzamento, frammentazione e riassemblaggio, incapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione.

Il seguente [link](#) (in inglese [qui](#)) potrà esservi utile per migliorare il protocollo.

## 24.Lavoro di laboratorio - Biblioteca

Si vuole implementare un server che permetta di consultare i titoli dei libri presenti in una biblioteca e il numero di copie disponibili per ogni libro, e il server dovrà utilizzare a livello di trasporto i servizi offerti dal protocollo UDP. L'architettura dovrà essere di tipo

client-server, e il server dovrà accettare le richieste provenienti da un client utilizzando un protocollo connesso e confermato.

### **Esercitazione 1 - Voto massimo 8**

Il server dovrà rispondere alle seguenti richieste del client:

- a. richiesta della lista di tutti i titoli presenti nella biblioteca e del numero di copie per ogni libro;
- b. richiesta del numero di copie presenti in biblioteca di uno specifico libro.

Il server dovrà rispondere con un messaggio di errore se il comando non è fornito secondo la sintassi corretta o non è riconosciuto.

### **Esercitazione 2 facoltativa - Voto massimo 10**

Il server oltre a rispondere alle richieste indicate nell'esercitazione obbligatoria, dovrà anche rispondere alle seguenti richieste del client:

- c. richiesta di prestito di un libro;
- d. richiesta di restituzione di un libro.

Il server dovrà rispondere con un messaggio di errore se il comando non è fornito secondo la sintassi corretta o non è riconosciuto.

[Le richieste dei client dovrebbero essere gestite in modo sincronizzato.]

## **25. Lavoro di laboratorio - Client TFTP<sup>27</sup>**

Al candidato viene richiesta la progettazione e l'implementazione di un client TFTP seguendo le specifiche del protocollo esposte nello [RFC 1350](#).

Il server TFTP (LiTFTPServer) con cui fare le prove, e che dovrà essere analizzato per implementare il client, potrà essere scaricato a questo [link](#).

Il client implementato nell'Esercitazione 1 potrà essere testato anche con altri tipi di server TFTP scaricabili ai seguenti link:

- server TFTP per Ubuntu e derivati: sudo apt-get install tftpd-hpa
- server TFTP per Windows: [link](#)

### **Esercitazione 1 - Voto massimo 8**

Il client TFTP dovrà svolgere le seguenti operazioni:

- download di un file dal server, operazione già presente sugli appunti del docente;
- upload di un file sul server, operazione che il candidato dovrà progettare ed implementare individualmente.

### **Esercitazione 2 facoltativa - Voto massimo 10**

Il server LiTFTPServer è in grado di fornire anche l'elenco dei file presenti nello spazio di archiviazione del server. Progettare ed implementare un client che sia in grado di effettuare la richiesta della lista dei file presenti sul server.

## **Suggerimenti**

---

<sup>27</sup> Il laboratorio si è ispirato ad un lavoro ideato dal Prof. Giuliano Bellucci nell'a.s.2017-2018 e il server TFTP da usare per le prove (LiTFTPServer) è stato ideato dallo studente Oulai Li diplomato nell'a.s.2017-2018 presso l'IIS "Giuseppe Peano" di Torino.

Per la scrittura del file ricevuto si utilizzi la classe `FileOutputStream` ed il metodo `write(byte[] b, int offset, int length)`;

Per la lettura del file da inviare usare `FileInputStream` ed il metodo `read(byte[] b)`;

Nella costruzione del messaggio di connessione per download/upload può essere utile la funzione

```
System.arraycopy(Object src, int srcPos, Object dest, int destPos,  
                 int length)
```

## Aspetti progettuali di un protocollo di comunicazione

26. Il seguente protocollo permette di effettuare dei calcoli aritmetici usando un servizio offerto da un server remoto. Il protocollo può essere implementato sia su TCP, sia su UDP.

Il formato dei messaggi del protocollo applicativo lato client è il seguente:

Comando	Operazione
CALC	Richiesta al server di usare il servizio di calcolo
QUIT	Segnalazione al server di terminazione d'uso del servizio di calcolo
ADD, <i>x,y</i>	Addizione: $x + y$ , con $x$ e $y$ valori numerici
SUB, <i>x,y</i>	Sottrazione: $x - y$ , con $x$ e $y$ valori numerici
MUL, <i>x,y</i>	Moltiplicazione: $x * y$ , con $x$ e $y$ valori numerici
DIV, <i>x,y</i>	Divisione: $x / y$ , con $x$ e $y$ valori numerici

Il formato dei messaggi del protocollo applicativo lato server è il seguente:

Comando	Operazione
OKCALC	Accettazione della richiesta di servizio da parte del client
OKQUIT	Accettazione della richiesta di terminazione del servizio da parte del client
ADD, <i>n</i>	<i>n</i> è il risultato dell'addizione richiesta
SUB, <i>n</i>	<i>n</i> è il risultato della sottrazione richiesta
MUL, <i>n</i>	<i>n</i> è il risultato della moltiplicazione richiesta
DIV, <i>n</i>	<i>n</i> è il risultato della divisione richiesta
ERR, <i>codErr</i>	<i>codErr</i> <ul style="list-style-type: none"><li>• 1 - operazione non riconosciuta</li><li>• 2 - valore numerico non valido</li></ul>

- |  |                          |
|--|--------------------------|
|  | ● 3 - comando non valido |
|--|--------------------------|

Il client può richiedere più operazioni aritmetiche ma solo dopo aver ricevuto la risposta di una richiesta fatta in precedenza.

Il candidato esponga, motivando la risposta, quali caratteristiche presenta il protocollo tra indirizzamento, frammentazione e riassemblaggio, encapsulamento, controllo della connessione, servizio confermato o non confermato, controllo degli errori, controllo del flusso, multiplexing, servizi di trasmissione (qualità del servizio, priorità, sicurezza).

27. Dati i seguenti messaggi scambiati tra un client ed un server che usino il protocollo [TFTP](#) per lo scambio di file, individuare e spiegare quali degli aspetti progettuali di un protocollo di comunicazione sono previsti, prendendo come riferimento i seguenti:

- indirizzamento;
- frammentazione e riassemblaggio;
- encapsulamento;
- controllo della connessione;
- servizio confermato o non confermato;
- controllo degli errori;
- controllo del flusso;
- multiplexing;
- servizi di trasmissione.

#### TFTP Formats

Type	Op #	Format without header				
		2 bytes	string	1 byte	string	1 byte
RRQ / WRQ	01/02	Filename	0	Mode	0	
	-----					
DATA	2 bytes	2 bytes	n bytes			
	-----					
ACK	03	Block #	Data			
	-----					
ERROR	2 bytes	2 bytes	string	1 byte		
	-----					
	05   ErrorCode   ErrMsg   0					
	-----					

#### Initial Connection Protocol for reading a file

- Host A sends a "RRQ" to host B with source= A's TID, destination= 69.

2. Host B sends a "DATA" (with block number= 1) to host A with source= B's TID, destination= A's TID.

28.Considerando il protocollo [TIME](#), individuare e spiegare quali degli aspetti progettuali di un protocollo di comunicazione sono in esso previsti, prendendo come riferimento i seguenti:

- a. indirizzamento;
- b. frammentazione e riassemblaggio;
- c. incapsulamento;
- d. controllo della connessione;
- e. servizio confermato o non confermato;
- f. controllo degli errori;
- g. controllo del flusso;
- h. multiplexing;
- i. servizi di trasmissione.

29.Ripetere l'esercizio precedente scegliendo uno nuovo protocollo tra quelli elencati di seguito:

- [ECHO](#): protocollo originariamente proposto per testare e misurare il round-trip times nelle reti IP.
- [DAYTIME](#): protocollo di spedizione della data e dell'ora fornita da un server;
- [DISCARD](#): protocollo che butta via qualsiasi dato riceva;
- [CHARACTER GENERATOR](#): protocollo che genera e invia dati indipendentemente dall'input;
- [QOTD](#): protocollo che invia un breve messaggio (Quote Of The Day);
- [MESSAGE SEND PROTOCOL](#): protocollo per l'invio di brevi messaggi fra nodi di una rete;
- [WHOIS](#): protocollo per il recupero delle informazioni di utenti Internet.

30.Prendendo come riferimento l'implementazione del protocollo di trasferimento di file presentato di seguito<sup>28</sup>, si realizzi un documento che lo descriva in tutte le sue fasi di progettazione. Le possibili fasi da considerare sono quelle descritte in questo documento ed elencate di seguito, alle quali è stata aggiunta la fase in cui nel documento viene fornita una breve spiegazione dello scopo del protocollo. Si consideri che non tutte le fasi di progettazione indicate potrebbero essere state implementate.

1. Breve descrizione dello scopo del protocollo applicativo
2. Indirizzamento
3. Frammentazione e riassemblaggio
4. Incapsulamento
5. Controllo della connessione
6. Controllo degli errori
7. Controllo del flusso
8. Multiplexing
9. Servizi di trasmissione.

## **SERVER UDP per il trasferimento di file**

### **ServerUDP.java**

---

<sup>28</sup> "SERVER UDP per il trasferimento di file", implementato dal Prof. Giuliano Bellucci.

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

/**
 *
 * @author palma
 */
public class ServerUDP {

    private byte[] buf;
    private DatagramPacket received;
    private InetAddress clientAddress;
    private int clientPort;
    private int clientNumber;
    private Thread thread;

    public void start() throws IOException {
        clientNumber=1;
        DatagramSocket socket = new DatagramSocket(9999);
        int serverPort = socket.getLocalPort();
        System.out.println("Server in ascolto sulla porta: " + serverPort);
        while (true) {
            buf = new byte[100];
            System.out.println("In attesa di chiamate dai Client... ");
            received = new DatagramPacket(buf, buf.length);
            socket.receive(received);
            thread=new Thread(new RequestHandler(received,new String(buf)));
            thread.start();
            clientAddress = received.getAddress();
            clientPort = received.getPort();
            System.out.println("In chiamata Client N° "+clientNumber+" : " +
                               clientAddress + " porta: " + clientPort);
            clientNumber++;
        }
    }

    public static void main(String[] args) throws IOException {
        ServerUDP server = new ServerUDP();
        server.start();
    }
}
```

### RequestHandler.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author palma
 */
class RequestHandler implements Runnable {

    DatagramPacket received;
    String data;

    public RequestHandler(DatagramPacket received, String data) {
        this.received = received;
        this.data = data.trim();
    }

    @Override
    public void run() {
        BufferedReader r = null;
        String line = null;
        DatagramSocket printer;
        DatagramPacket packet = new DatagramPacket("$FOUND$".getBytes(),
                                                    "$FOUND$".getBytes().length,
                                                    received.getAddress(),
                                                    received.getPort());
        try {
            r = new BufferedReader(new InputStreamReader(this.getClass().
                getResourceAsStream("hg.txt")));
            printer = new DatagramSocket();
            printer.send(packet);
            printer.send(new DatagramPacket("$BEGIN$".getBytes(),
                                            "$BEGIN$".getBytes().length,
                                            received.getAddress(),
                                            received.getPort()));
            while ((line = r.readLine()) != null) {
                packet = new DatagramPacket(line.getBytes(),
                                            line.getBytes().length,
                                            received.getAddress(),
                                            received.getPort());
                try {
                    printer.send(packet);
                    printer.receive(packet);
                } catch (SocketException ex) {

```

```
        Logger.getLogger(RequestHandler.class.getName()).
                log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(RequestHandler.class.getName()).
                log(Level.SEVERE, null, ex);
    }
}
printer.send(new DatagramPacket("$END$".getBytes(),
                                "$END$".getBytes().length,
                                received.getAddress(),
                                received.getPort()));
} catch (FileNotFoundException ex) {
    Logger.getLogger(RequestHandler.class.getName()).
            log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    Logger.getLogger(RequestHandler.class.getName()).
            log(Level.SEVERE, null, ex);
} finally {
    try {
        r.close();
    } catch (IOException ex) {
        Logger.getLogger(RequestHandler.class.getName()).
                log(Level.SEVERE, null, ex);
    }
}
}
}
```

## CLIENT UDP per il trasferimento file

### ClientUDP.java

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 *
 * @author palma
 */
public class ClientUDP {

    /**
     *
     */
    public static void main(String[] args) {
        try {
            DatagramSocket dsocket = new DatagramSocket();
            InetAddress host = InetAddress.getByName("127.0.0.1");
            int port = 12345;
            byte[] buffer = new byte[1024];
            String file = JOptionPane.showInputDialog("Inserire nome file");
            File f = new File(file);
            FileInputStream fis = new FileInputStream(f);
            byte[] fileContent = new byte[(int)f.length()];
            fis.read(fileContent);
            fis.close();
            DatagramPacket dp = new DatagramPacket(fileContent, fileContent.length, host, port);
            dsocket.send(dp);
            dsocket.close();
        } catch (IOException ex) {
            Logger.getLogger(ClientUDP.class.getName()).
                    log(Level.SEVERE, null, ex);
        }
    }
}
```

```
* @param args the command line arguments
*/
public static void main(String[] args) {
    //JOptionPane.showInputDialog("Inserire l'indirizzo del server");
    String ip="127.0.0.1";
    //Integer.parseInt(JOptionPane.showInputDialog("Inserire la porta su cui
    //è in ascolto il server"));
    int port=9999;
    //JOptionPane.showInputDialog("Inserire il nome del file");
    String fileName="hg.txt";
    try {
        DatagramPacket packet = new DatagramPacket(fileName.getBytes(),
                                                     fileName.getBytes().length,
                                                     InetAddress.getByName(ip),
                                                     port);
        DatagramSocket socket=new DatagramSocket((int)(Math.random()*
64000+1025));
        socket.send(packet);
        byte [] data = new byte[6];
        packet = new DatagramPacket(data, data.length);
        socket.receive(packet);
        String response=new String(packet.getData());
        if ("$FOUND$".equals(response.trim())) {
            new View(socket);
        } else {
            JOptionPane.showMessageDialog(null, "File non trovato",
                                         "Errore", JOptionPane.ERROR_MESSAGE);
        }
    } catch (UnknownHostException ex) {
        Logger.getLogger(ClientUDP.class.getName()).
            log(Level.SEVERE, null, ex);
    } catch (SocketException ex) {
        Logger.getLogger(ClientUDP.class.getName()).
            log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(ClientUDP.class.getName()).
            log(Level.SEVERE, null, ex);
    }
}
```

## View.java

```
import java.awt.Dimension;
import java.awt.Toolkit;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import javax.swing.JFrame;
```

```
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

/** 
 * 
 * @author palma
 */
public class View extends JFrame {

    private JTextArea msgIn;

    View(DatagramSocket socket) {
        msgIn = new JTextArea(30, 50);
        msgIn.setEditable(false);
        add(new JScrollPane(msgIn));
        pack();
        Thread thread=new Thread(new Reader(socket));
        thread.start();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension dim = toolkit.getScreenSize();
        setLocation(dim.width / 2 - getWidth() / 2, dim.height / 2 -
                    getHeight() / 2);
        setVisible(true);
    }

    class Reader implements Runnable{
        private DatagramSocket socket;
        private DatagramPacket packet;

        public Reader(DatagramSocket socket) {
            this.socket = socket;
        }

        @Override
        public void run() {
            int i=1;
            String sp=null;
            String s="";
            byte [] buffer;
            while(!s.equals("$END$")){
                try {
                    buffer=new byte[5000];
                    packet=new DatagramPacket(buffer, buffer.length);
                    socket.receive(packet);
                    s=new String(packet.getData()).trim();
                    if(!s.equals("$BEGIN$") && !s.equals("$END$")){
                        msgIn.setText(msgIn.getText()+s+'\n');
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```
        }
        socket.send(new DatagramPacket("$OK$".getBytes(),
                                       "$OK$".getBytes().length,
                                       packet.getAddress(),
                                       packet.getPort()));
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
socket.close();
}

}
```

## Protocols - CLIL

31.The following video gives an overview about what Internet protocols are. Listen to each video and answer the question. At the end of the work, send the answer to your teacher.

### Protocols

- a. What is a protocol? [What is a protocol?](#)
- b. Why is it so important that protocols are written? [What is a protocol?](#)
- c. Which are the two things that a protocol has to do? [What is a protocol?](#)
- d. Why are protocols so highly structured? [Why are protocols so highly structured?](#)
- e. What are clients and servers and which is the difference between them? [What is a client? What is a server?](#)
- f. What is the robustness principle? [What is the robustness principle?](#)
- g. Regarding Internet protocol stack, explain what the two goals "Separation of concern" and "Reuse of good ideas" are. [What is the internet protocol stack?](#)
- h. How can UDP be used other than its normal purpose? [Can you still deploy new transport protocols?](#)

32.bla bla

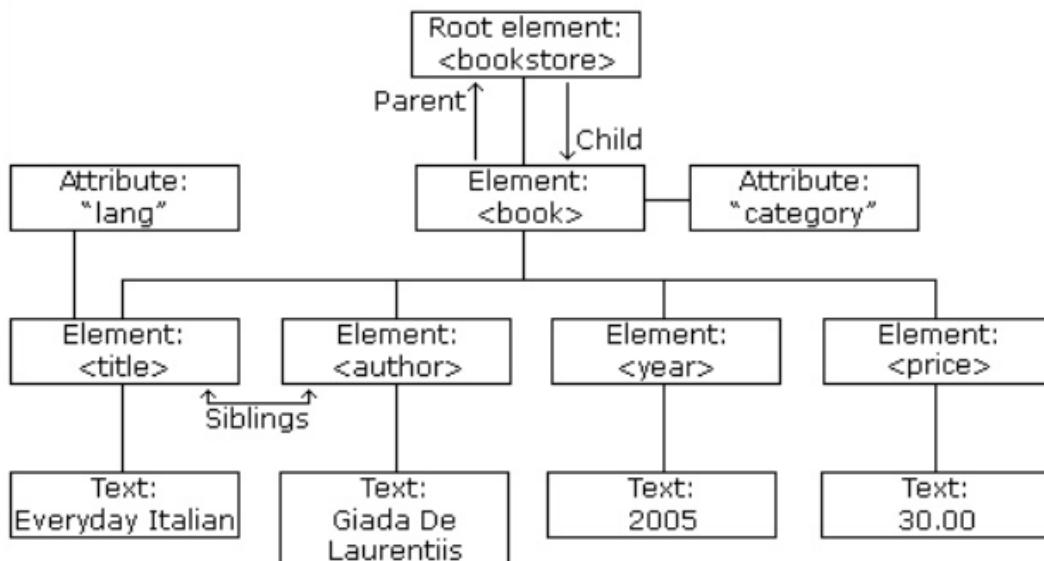
# Esercizi sul XML e JSON

33.Si vuole definire un formato per l'interscambio dati relativo allo **stato dei prestiti dei libri di una biblioteca**. Il file scambiato specifica la lista degli utenti per ciascuno dei quali si indicano il nominativo (string), il numero di tessera (integer) e la lista dei libri che ha in prestito. Per ciascun libro è indicato il codice del catalogo (string) e la data di scadenza del prestito (date). Si propongano le strutture XML e JSON necessarie, mostrando un esempio per ognuna di esse. ~~Si rappresenti l'XML create con l'albero DOM<sup>29</sup> corrispondente.~~

Si svolgono successivamente anche i seguenti punti:

- realizzare lo schema concettuale e lo schema logico del database dal quale possono essere derivati i dati delle stringhe XML e JSON, aggiungendo tutti gli attributi necessari per rendere completo il database;
- scrivere la query usando MySQL per ottenere dal database i dati espressi tramite le stringhe XML e JSON; *Visualizzare per ogni utente che ha preso in prestito almeno un libro, la lista dei libri che ha preso in prestito.*
- creare l'UML per l'applicazione Java in grado di generare le stringhe XML e JSON;
- realizzare l'applicazione Java per derivare le stringhe XML e JSON;
- progettare anche mediante rappresentazioni grafiche l'infrastruttura tecnologica ed informatica necessaria a gestire il servizio nel suo complesso, dettagliando le modalità di comunicazione tra le varie componenti, e le soluzioni hardware e software per garantire la sicurezza perimetrale del sistema.

34.Dato il seguente albero DOM, scrivere il corrispondente file XML.



35.Si vuole definire un formato per l'interscambio dati relativo alle **chiamate di soccorso di una centrale operativa**. Il file scambiato specifica la lista delle chiamate. Ciascuna chiamata è caratterizzata da un timestamp (dateTime), dal numero di telefono da cui è stata ricevuta (string), dal codice dell'operatore (integer). Alla chiamata è inoltre associata una lista di segnalazioni. Per ciascuna segnalazione è specificato il luogo

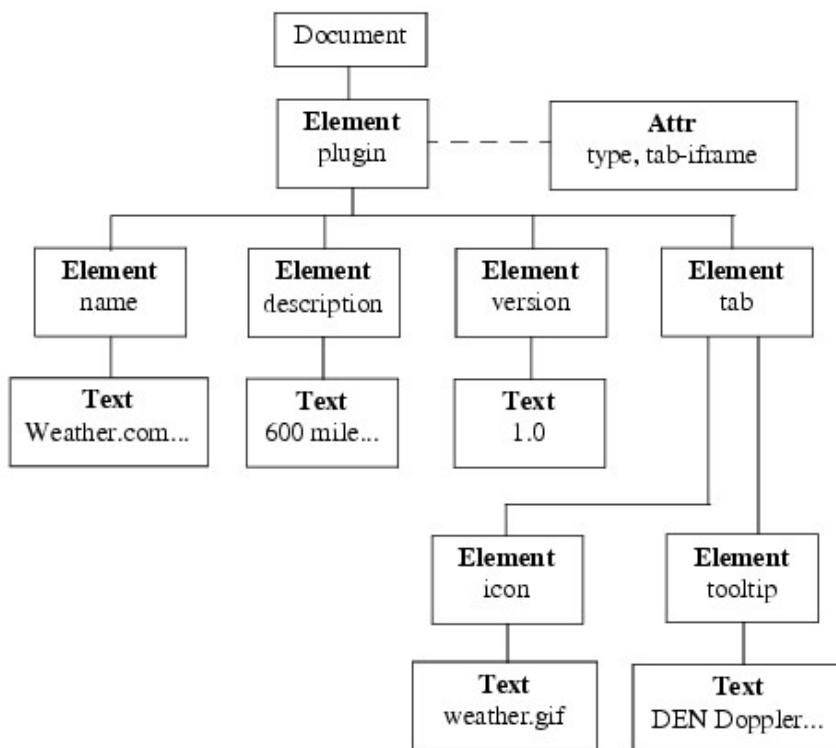
<sup>29</sup> Document Object Model.

(string), la descrizione (string) e il livello di rischio (integer).

Si svolgano i seguenti punti.

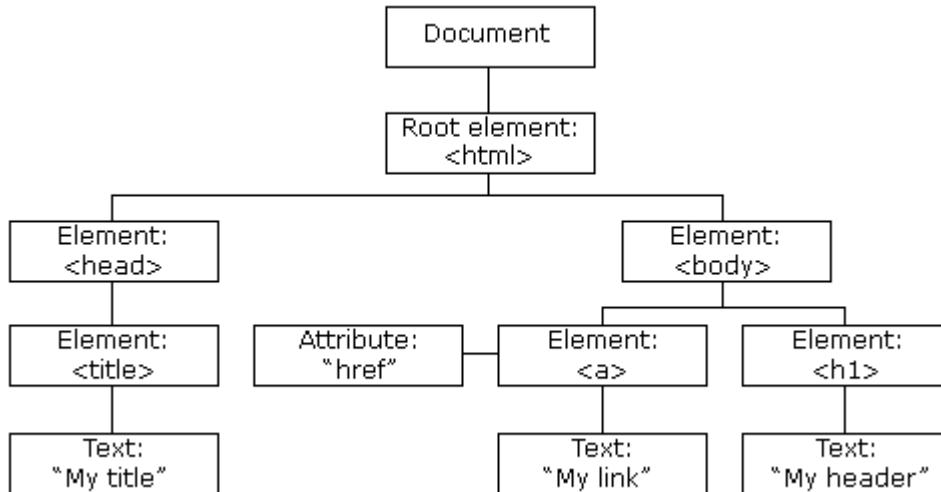
- a. Si propongano le strutture XML e JSON necessarie, mostrando un esempio per ognuna di esse.
- b. Realizzare lo schema concettuale e lo schema logico del database dal quale possono essere derivati i dati delle stringhe XML e JSON, aggiungendo tutti gli attributi necessari per rendere completo il database.
- c. Scrivere la query usando MySQL per ottenere dal database i dati espressi tramite le stringhe XML e JSON.
- d. Creare l'UML per l'applicazione Java in grado di generare le stringhe XML e JSON.
- e. Realizzare l'applicazione Java per derivare le stringhe XML e JSON.
- f. Progettare anche mediante rappresentazioni grafiche l'infrastruttura tecnologica ed informatica necessaria a gestire il servizio nel suo complesso (deployment diagram), dettagliando le modalità di comunicazione tra le varie componenti, e le soluzioni hardware e software per garantire la sicurezza perimetrale del sistema.
- g. Progettare lo use case diagram che rappresenti la modalità di interazione per l'uso dell'applicazione.
- h. h) Progettare, anche mediante rappresentazioni grafiche, l'infrastruttura tecnologica ed informatica necessaria a gestire il servizio nel suo complesso, dettagliando (1) l'architettura della rete e le caratteristiche del o dei sistemi server, motivando anche la scelta dei luoghi in cui installare questi ultimi, (2) le modalità di comunicazione tra server e dispositivi di consultazione dello storico delle chiamate, descrivendo protocolli e servizi software da implementare per gestire la rete e fornire le pagine.
- i. ~~Si rappresenti l'XML creato con l'albero DOM corrispondente.~~

36. Dato il seguente albero DOM, scrivere il corrispondente file XML.



37.Si vuole definire un formato per l'interscambio dati relativo alla gestione di un insieme di **cartelloni pubblicitari elettronici**. Il file scambiato specifica la lista dei cartelloni. Ciascun cartellone è identificato da un codice (integer), dalla città (string), dalla posizione (string), dall'indirizzo IP (string). Ad ogni cartellone è inoltre associata una lista di annunci. Per ciascun annuncio è specificata l'ora di inizio per la visualizzazione (time), l'ora di fine (time) e il testo (string). Si proponga la struttura XML necessaria, mostrando un esempio. Si rappresenti l'XML creato con l'albero DOM corrispondente.

38.Dato il seguente albero DOM, scrivere il corrispondente file XML.



39.Si vuole definire un formato per l'interscambio dati relativo alle **rilevazioni di inquinamento acustico di una città**. Il file scambiato specifica la lista dei punti di misura attivi. Ciascun punto di misura è identificato da un codice numerico (integer) e ha associate le coordinate geografiche (string, es. "43.318423, 11.331361"), la data di installazione del sensore (date), la precisione (decimal), la lista delle misure effettuate e la lista degli interventi di taratura. Per ogni misura si specifica il momento in cui è stata effettuata (dateTime) e il valore misurato (decimal). Per ogni intervento di taratura si indicano la data (date), il nome del tecnico che ha effettuato l'operazione (string) e una eventuale nota (string). Si proponga la struttura XML necessaria, mostrando un esempio. Si rappresenti l'XML creato con l'albero DOM corrispondente.

40.Si vuole definire un formato per l'interscambio dati relativo ai dati di **installazione del software per i computer di un'azienda**. Il file scambiato specifica la lista dei computer. Per ciascun computer si indica l'indirizzo IP (string), la descrizione (string), la data di acquisto (date) e la lista dei software installati. Per ogni software si specificano il nome del software (string), la versione (integer) e la data di installazione (date). Si proponga la struttura XML necessaria mostrando un esempio, e l'albero DOM corrispondente.

41.Si vuole definire un formato per l'interscambio dati relativo alle **misure di qualità dell'acqua di una serie di impianti di potabilizzazione**. Il file scambiato specifica la lista degli impianti. Ciascun impianto è identificato da un nome (string) ed ha associato un indirizzo (string) e una lista dei depositi. Per ogni deposito si specifica il codice (integer), la capienza (decimal), un flag che indica se è in uso (boolean) e la lista delle misure che sono state fatte. Per ogni misura si specifica il tipo (string), il timestamp di

quando è stata fatta (dateTime) e il valore misurato (decimal). Si proponga la struttura XML necessaria mostrando un esempio, e l'albero DOM corrispondente.

- 42.Si vuole definire un formato per l'interscambio dati relativo alla gestione dei dati di un **insieme di stazioni meteorologiche**. Il file scambiato specifica la lista delle stazioni. Ciascuna stazione meteorologica è identificata da un codice numerico (integer) e ha associata la città in cui si trova (string), le sue coordinate geografiche (string, es. "43.313914, 11.338857"), la una lista delle misurazioni effettuate e la lista degli strumenti disponibili. Per ogni misurazione si specifica il momento in cui è stata effettuata (dateTime), il tipo di misurazione (string, es. "pioggia" ) e il valore misurato (decimal). Per ogni strumento di indica il numero di inventario (integer), il nome (string) e la data di acquisto (date). Si proponga la struttura XML necessaria mostrando un esempio, e l'albero DOM corrispondente.
- 43.Si vuole definire un formato per l'interscambio dati relativo alla **misure dell'attività sismica di una regione**. Il file scambiato specifica la lista delle stazioni di misurazione. Ciascuna stazione è identificata da un codice numerico (integer) ed ha associata la località in cui si trova (string), il nome del responsabile (string) e la lista degli eventi sismici rilevati. Per ogni evento sismico si specifica l'istante della prima scossa (dateTime), l'epicentro (string, in coordinate geografiche, es. "43.318264, 11.338220"), la profondità in metri (decimal) e la lista delle scosse. Per ogni scossa si riporta l'intensità in scala Richter (decimal), l'ora (time) e la durata in secondi (integer). Si proponga la struttura XML necessaria mostrando un esempio, e l'albero DOM corrispondente.
- 44.Si vuole definire un formato per l'interscambio dati relativo alla gestione di una **bacheca on-line per la pubblicazione di messaggi**. Il file scambiato specifica la lista degli utenti. Ciascun utente è identificato da un username (string) ed ha associata la data di registrazione (date), l'istante dell'ultimo login (dateTime), l'età (decimal), la lista degli utenti che lo seguono (followers) e la lista dei post. Per follower si specificano l'identificatore (string), la data di iscrizione (date) e il tipo di relazione (integer, un codice interno es. 0→ "amico", 1→"amico di amico",...). Per ogni post si specificano il momento della pubblicazione (dateTime), il testo (string) e la visibilità (integer, un codice es. 0 → "solo amici",...). Si proponga la struttura XML necessaria, mostrando un esempio, e l'albero DOM corrispondente.
- 45.Si vuole definire un formato per l'interscambio dati relativo alla gestione dei **dispositivi condivisi sulla rete di un'azienda**. Il file scambiato specifica la lista dei dispositivi. Ciascun dispositivo è identificato dall'indirizzo IP (string) ed ha associate la sua posizione nell'edificio (string), il tipo (string, es. "stampante", "NAS",..) e la lista degli account. Per ogni account si specifica il codice (integer), la data di scadenza (date), il massimo uso totale consentito per la risorsa (decimal) e la lista dei job inviati. Per ogni job si riporta il timestamp (dateTime), la quota di uso della risorsa (decimal), il codice di esito della richiesta (integer) e un messaggio che descrive l'esito (string, es. "quota exceeded"). Si proponga la struttura XML necessaria, mostrando un esempio, e l'albero DOM corrispondente.
- 46.Si implementi un parser in Java che utilizzi la tecnica DOM per il seguente documento XML, che riporta le informazioni relative ad un file in cui vengono memorizzati i dati relativi ai download effettuati su una serie di server. Il parser dovrà scorrere l'albero

DOM utilizzando i nomi dei tag (Esempio 1) e visualizzare, per ogni server, i dati relativi ai vari download effettuati.

```
<file name="pippo" dimension="20.0" type="pdf" >
  <servers>
    <server ip="1.1.1.1" creationDate="Sun Jan 15 15:19:08 CET 2017"
      totalDownload="2" >
      <downloads>
        <download date="Sun Jan 15 15:19:08 CET 2017"
          ip="110.234.23.124" speed="10.5" result="OK" />
        <download date="Sun Jan 15 15:19:08 CET 2017"
          ip="110.234.23.124" speed="10.5" result="OK" />
      </downloads>
    </server>
    <server ip="22.22.1.1" creationDate="Sun Jan 15 15:19:08 CET 2017"
      totalDownload="1" >
      <downloads>
        <download date="Sun Jan 15 15:19:08 CET 2017"
          ip="110.111.23.124" speed="15.5" result="OK" />
      </downloads>
    </server>
  </servers>
</file>
```

47.Si implementi un parser in Java che utilizzi la tecnica DOM per un qualsiasi documento XML, come ad esempio quello dell'esercizio precedente che riporta le informazioni relative ad un file in cui vengono memorizzati i dati relativi ai download effettuati su una serie di server. Il parser dovrà scorrere l'albero DOM senza utilizzare i nomi dei tag (Esempio 2 o 3) e visualizzare le informazioni in esso contenute.

48.bla bla.

# Esercitazione di laboratorio

## Protocolli applicativi di rete - XML - Parser Java - Database

49. Un cronista di partite di calcio utilizza un programma per inviare alla redazione i fatti salienti delle partite in tempo reale.

Supponendo che le formazioni calcistiche siano già state inviate in precedenza, il programma permetterà di scegliere, tramite un menù, i fatti salienti della partita. Gli eventi invocati dal menù possono o meno richiedere l'inserimento di informazioni aggiuntive. Ogni evento, dopo essere stato inserito dal cronista, viene inviato alla redazione come blocco XML.

Evento	Informazioni aggiuntive
Calcio di inizio	
Ammonizione	<ul style="list-style-type: none"><li>● nome giocatore</li><li>● squadra</li><li>● minuto</li><li>● motivazione</li></ul>
Espulsione	<ul style="list-style-type: none"><li>● nome giocatore</li><li>● squadra</li><li>● minuto</li><li>● motivazione</li></ul>
Goal	<ul style="list-style-type: none"><li>● nome giocatore</li><li>● squadra</li><li>● se fatto su rigore</li><li>● minuto</li></ul>
Calcio di punizione	<ul style="list-style-type: none"><li>● squadra responsabile del fallo</li><li>● autore del fallo</li><li>● giocatore che ha subito il fallo</li><li>● minuto</li></ul>
Calcio di punizione diretto	<ul style="list-style-type: none"><li>● squadra responsabile del fallo</li><li>● autore del fallo</li><li>● giocatore che ha subito il fallo</li><li>● minuto</li></ul>
Rigore	<ul style="list-style-type: none"><li>● squadra responsabile del fallo</li><li>● autore del fallo</li><li>● giocatore che ha subito il fallo</li><li>● minuto</li></ul>
Sostituzione	<ul style="list-style-type: none"><li>● squadra</li><li>● nome giocatore uscente</li><li>● nome giocatore entrante</li></ul>

	<ul style="list-style-type: none"><li>• minuto</li></ul>
Azione importante	<ul style="list-style-type: none"><li>• descrizione libera</li><li>• minuto</li></ul>
Fatto anomalo	<ul style="list-style-type: none"><li>• descrizione libera</li><li>• minuto</li></ul>
Sospensione temporanea	<ul style="list-style-type: none"><li>• descrizione libera</li><li>• minuto iniziale</li><li>• minuto finale</li></ul>
Fine primo tempo	<ul style="list-style-type: none"><li>• descrizione libera</li></ul>
Inizio secondo tempo	<ul style="list-style-type: none"><li>• descrizione libera</li></ul>
Fine partita	<ul style="list-style-type: none"><li>• descrizione libera</li></ul>

Progettare e sviluppare le seguenti fasi:

1. Determinare il formato del blocco XML per inviare gli eventi alla redazione.
2. Progettare un protocollo di comunicazione che preveda l'invio dei blocchi XML con conferma della ricezione, ma senza la gestione degli errori. Il protocollo deve prevedere anche un lato ricevente che confermerà i blocchi ricevuti. Per progettare il protocollo si preveda l'invio dei dati in formato testuale. Come esempio potete prendere in considerazione il protocollo creato dal Prof. Bellucci sul trasferimento dei file e presente nell'area didattica del registro. (Opzionale) Criptare i dati inviati con uno degli algoritmi studiati in Sistemi e reti.
3. Determinare il formato del blocco XML delle formazioni e memorizzato in redazione. Il file dovrà contenere la data della partita e i nomi delle due squadre.
4. Progettare un programma per la redazione che, una volta ricevuto i blocchi XML della cronaca della partita, lo utilizzi congiuntamente con il blocco XML delle formazioni già presente in redazione per costruire un unico file XML da inviare all'archivio. Il nuovo file XML costituirà la cronaca completa della partita. Si progetti al riguardo il formato del nuovo file XML che dovrà essere costruito in automatico dal programma. Per l'invio all'archivio si progetti un protocollo che utilizzi la stessa tecnica del protocollo TFTP, cioè la gestione di dati binari con blocchi di lunghezza fissa e il blocco di avvenuta ricezione del singolo blocco di dati, ma senza la gestione degli errori. (Opzionale) Criptare i dati inviati con uno degli algoritmi studiati in Sistemi e reti.
5. Progettare un programma per l'archivio che, una volta ricevuto il file XML inviato dalla redazione, crei e salvi su disco un nuovo file XML contenente il tabellino della partita. Il nome del file dovrà essere *SquadraDiCasaSquadraOspiteAnno*. Il tabellino della partita dovrà riportare la data della partita, le squadre, le formazioni, il risultato della partita, gli autori dei gol e il minuto, i giocatori ammoniti e quelli espulsi distinguendoli per squadra.

6. (Opzionale) Definire un database per il salvataggio delle informazioni presenti nel tabellino e salvarle.

## Esercizi su Servlet

50. Progettare una servlet in grado di visualizzare una frase di benvenuto. La servlet dovrà essere richiamata tramite un link in una pagina HTML.



51. Progettare una servlet che visualizzi il tuo nome e cognome preventivamente inseriti in un form iniziale, congiuntamente alla date e ora del sistema in cui viene eseguita. Il nome e il cognome dovranno essere passati come parametri alla servlet.

Per svolgere questo esercizio potete trarre spunto dal laboratorio 2 della Uda 3 del libro, pp.234-239.

Il risultato finale dovrà essere simile a quanto mostrato di seguito.

Dati utente

Nome: Maria Grazia

Cognome: Maffucci

Invia Cancella

Dati inseriti:

Nome: Maria Grazia

Cognome: Maffucci

La data di oggi è Mon Mar 21 21:30:44 CET 2016

52. Modificare l'esercizio precedente prevedendo l'inserimento nel file web.xml di due parametri costanti di inizializzazione. Un parametro deve permettere di impostare il titolo della pagina generata dinamicamente dalla servlet (ad es. **Pagina di benvenuto**), mentre l'altro deve essere utilizzato per scrivere un saluto subito dopo l'elenco dei dati (ad es. **Ciao**) riportando il nome e il cognome inseriti nel form iniziale. Provare a gestire i parametri di configurazione nei due seguenti modi, creando due progetti distinti:

- usando il metodo **init()** e il metodo **getInitParameter()** come mostrato sul libro di testo a p.187;
- lavorando direttamente nel metodo che implementa la servlet utilizzando la classe **ServletConfig**, il metodo **getServletConfig()** e infine il metodo **getInitParameter()**, come mostrato in questo [link](#).

Il risultato finale, in entrambi i casi, dovrà essere simile al seguente.

Dati utente - Mozilla Firefox

Dati utente

localhost:8080/03NEWNomeCognome/

Nome: Maria Grazia

Cognome: Maffucci

Invia Cancella

**Dati inseriti:**

Nome: Maria Grazia

Cognome: Maffucci

Ciao Maria Grazia Maffucci

**La data di oggi è Mon Mar 21 22:35:48 CET 2016**

53. Progettare una servlet che, dopo che l'utente ha inserito due valori numerici interi tramite un form, generi la tabellina relativa.

Numero colonne:

Numero righe:

Tabelline - Mozilla Firefox

Tabelline

localhost:8080/Tabelline/Tabelline

Search

Numero colonne:

Numero righe:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

54. Progettare una servlet che permetta di fare un sondaggio culinario, raccogliendo in un file binario le preferenze di alcuni piatti.  
La servlet deve permettere la scelta fra più piatti tramite dei radio button e, una volta inviato il form, dovranno essere visualizzate le attuali percentuali di gradimento.  
Prendere spunto dall'esercizio proposto sul libro di testo a p.238.

Scegli il piatto preferito - Mozilla Firefox

Scegli il piatto prefe... × +

localhost:8084/Piatto/

**Scegli il tuo piatto preferito**

Qual è il tuo piatto preferito?

Linguaggi conosciuti

Pizza  
 Spaghetti  
 Pastina  
 Lasagne

Invia Cancella

Piatti preferiti - Mozilla Firefox

Piatti preferiti × +

localhost:8084/Piatto/Piatto

**Sondaggio gastronomico**

Grazie per aver partecipato al nostro sondaggio  
Hai selezionato il piatto pizza

Risultati:

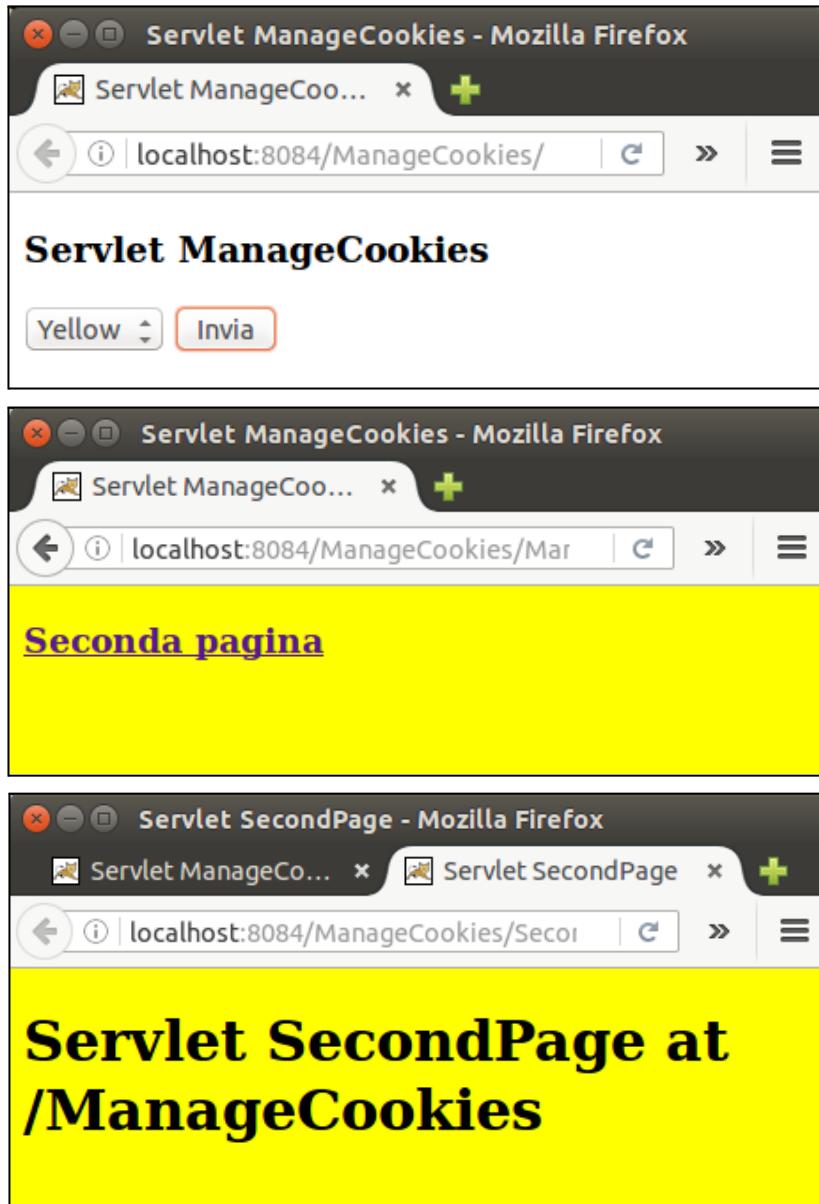
pizza: 100.00% - Risposte: 1  
spaghetti: 0.00% - Risposte: 0  
pastina: 0.00% - Risposte: 0  
lasagne: 0.00% - Risposte: 0

**Totale risposte: 1**

55. Progettare un'applicazione Web che permetta di effettuare un login (elementare) ad un sito. Se la password inserita risulta corretta l'utente deve essere reindirizzato ad una pagina di benvenuto che visualizzi il nome dell'utente (scegliete voi la password). Se invece la password risultasse scorretta, si dovrà segnalare l'errore riproponendo il form per l'inserimento del nome utente e della password. Utilizzare la classe RequestDispatcher per il passaggio dei parametri. Un esercizio analogo lo trovate a questo [link](#).

The figure consists of three vertically stacked screenshots of Mozilla Firefox browser windows. The top window is titled 'Login page - Mozilla Firefox' and shows a login form with fields for 'Nome' (containing 'maffu') and 'Password' (containing '.....'). Below this are two 'Login' and 'Reset' buttons. The middle window is titled 'Welcome - Mozilla Firefox' and displays the error message '**Username or password errati**'. It contains the same login form as the first window. The bottom window is also titled 'Welcome - Mozilla Firefox' and displays the welcome message '**Welcome maffu**'.

56. Progettare un'applicazione Web in grado di chiedere l'inserimento di due valori, il primo sarà il nome di un nuovo cookie, mentre il secondo sarà il valore assunto dal cookie. Una volta inseriti i due valori la pagina Web dovrà visualizzare l'elenco di tutti i cookie inseriti sino a quel momento, fornendo la possibilità di inserirne altri.
57. Progettare un'applicazione Web che preveda inizialmente la scelta di un colore da un listbox. Il server risponderà inviando una pagina con lo sfondo del colore selezionato. Da quest'ultima pagina si dovrà richiedere l'invio di una seconda pagina al server, tramite un link, anch'essa con il colore dello sfondo scelto inizialmente. Usare i cookie per memorizzare il colore scelto.



58. Progettare una servlet in grado di visualizzare il numero di volte che è stata eseguita, evidenziando quando viene eseguita la prima volta. Utilizzare le sessioni per gestire il conteggio.
59. Modificare la servlet precedente visualizzando la data di creazione della servlet e modificando il tempo di terminazione della sessione riducendolo a 3 minuti (o valore inferiore) in modo tale che, provando ad eseguirla dopo il tempo di chiusura della sessione, il contatore ricomincerà nuovamente il conteggio.
60. Progettare una applicazione Web in cui l'utente può inserire il proprio nome e scegliere un linguaggio di programmazione oggetto di studio. Inviando il form il server dovrà visualizzare una nuova pagina in cui verrà visualizzato un saluto all'utente ed un commento sul linguaggio di programmazione. Questa seconda pagina dovrà richiedere una terza pagina al server in cui verranno visualizzati:
- a. l'identificatore della sessione;
  - b. la data di creazione della sessione;
  - c. la data dell'ultimo accesso alla sessione;
  - d. il nome dell'utente;

e. il linguaggio di programmazione selezionato.

Si effettui opportunamente il passaggio dei parametri fra le due servlet utilizzando i parametri di sessione.

# Esercizi di progettazione di architetture distribuite

I seguenti esercizi sono formulati supponendo che il discente abbia già una preparazione completa su tutte le discipline oggetto della seconda prova (Informatica, TPSI, Sistemi e Reti). Il livello di preparazione richiesto è volutamente più elevato rispetto a quello che in effetti potrebbe essere a metà dell'ultimo anno di corso, e di ciò si dovrà tenere conto durante la correzione. Ciò nonostante al discente è richiesto, come parte integrante degli esercizi, di cercare in autonomia qualunque argomento che non sia ancora stato affrontato a lezione, utilizzando qualsiasi mezzo a disposizione per il recupero delle conoscenze mancanti. Lo scopo di questi esercizi non è quello di produrre un elaborato perfetto, bensì quello di cercare quanto ancora non si conosce provando ad integrarlo e rielaborarlo con quanto già appreso. Come spesso capita, la conoscenza è quasi sempre parziale, ma la vera competenza è nel riuscire ad integrarla come si può, e farsela bastare.

## Esercizi sui protocolli

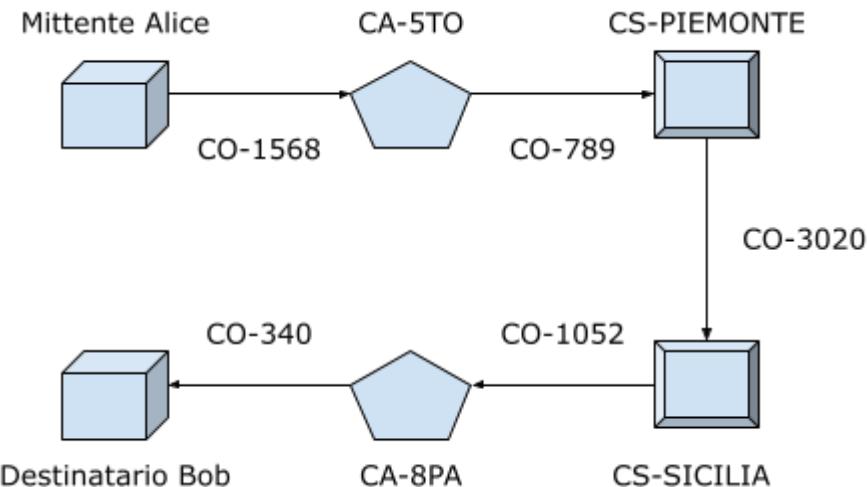
### 61. **WingedFeet - Ditta di trasporto**<sup>30</sup>

Il corriere espresso **WingedFeet** si occupa della spedizione di pacchi su tutto il territorio nazionale, sia per clienti privati, sia per ditte.

**WingedFeet** opera tramite dei **corrieri (CO)** che provvedono al trasporto dei pacchi, ed è organizzata in modo da avere diversi **centri di accettazione (CA)** dislocati nelle città presso i quali vengono inizialmente raccolti i pacchi per la prima fase della spedizione, o per la consegna finale. In ogni regione sono inoltre dislocati i **centri di smistamento (CS)**, uno per regione, nei quali vengono raccolti tutti i pacchi dei CA di una regione per essere smistati sul territorio nazionale. Ad esempio, la spedizione di un pacco da Torino a Palermo prevede un percorso analogo a quello indicato nell'immagine seguente, dove il pacco della Sig.ra Alice viene preso in carico dal CO 1568 che lo consegna presso il CA numero 5 di Torino, poi viene preso in carico dal CO 789 che lo consegna al CS del Piemonte. Il CO 3020 effettuerà la spedizione fino al CS della Sicilia e da qui il CO 1052 consegnerà il pacco al CA numero 8 di Palermo. Infine il CO 340 effettuerà la consegna al Sig. Bob.

---

<sup>30</sup> Ispirato dall'Esame di Stato di Sistemi e Reti del 2018.



**WingedFeet** vuole automatizzare maggiormente il processo di raccolta, smistamento e consegna dei pacchi, identificandoli univocamente in modo da permettere un'agevole tracciamento del percorso sia da parte dei clienti, sia da parte della stessa ditta.

Quando un cliente mittente vuole spedire un pacco effettua una richiesta online presso il sito della ditta, e il CA più vicino genera un codice univoco da assegnare al pacco, sincronizzandosi con il database centrale di **WingedFeet**, e incaricando al contempo un CO di prelevare il pacco per iniziare la spedizione. Il CO incaricato al prelievo attaccherà sul pacco l'etichetta con il codice univoco al momento della presa in carico presso il cliente.

In ciascun passaggio del trasporto viene anche segnato il [timestamp](#) della presa in carico del pacco e della relativa consegna o uscita dal CA o dal CS. Il CO che effettua la consegna finale raccoglie anche la firma del destinatario in formato elettronico.

Il candidato analizzi la realtà di riferimento e, fatte le opportune ipotesi aggiuntive, individui una soluzione che a suo motivato giudizio sia la più idonea a sviluppare i seguenti punti:

- a. illustrare il progetto dell'infrastruttura informatica necessaria per realizzare la gestione automatizzata dei pacchi e consentirne la tracciabilità, dettagliando:
  - i. i dispositivi utilizzati dai CO e dai magazzinieri per lo svolgimento delle proprie attività;
  - ii. le modalità di comunicazione tra i sistemi;
  - iii. l'organizzazione dei server di raccolta dati ed offerta dei servizi informativi, ipotizzando almeno un'ipotesi totalmente interna all'azienda ed una che contempla il ricorso a servizi Cloud;
  - iv. gli aspetti legati alla sicurezza delle strumentazioni, dei dati gestiti e del servizio offerto nel caso in esame, e discuta le misure che ritiene utili per garantire la continuità del servizio (aspetti di business continuity e fault tolerance);
- b. progettare un protocollo di comunicazione che permetta ai dispositivi elettronici usati per la lettura del codice del pacco, di comunicare con il server centrale della ditta memorizzando ad ogni passaggio il timestamp e il codice del CO, CA o CS che sta in quel momento gestendo il pacco, con l'indicazione se il pacco è in entrata o in uscita, dettagliando:
  - i. il protocollo usato a livello di trasporto e la porta di ascolto del server;
  - ii. il formato dei messaggi scambiati tra i dispositivi di lettura del codice e il server centrale, con una breve spiegazione della funzione di ogni messaggio;

- iii. la descrizione delle caratteristiche del protocollo;
- c. progettare il database per la gestione dei pacchi e dei clienti sviluppando:
  - i. la progettazione dello schema concettuale con la descrizione delle entità e degli attributi coinvolti e le eventuali regole aziendali;
  - ii. la progettazione dello schema logico con l'indicazione dei vincoli intrarelazionali e interrelazionali;
  - iii. la definizione dello schema fisico del database e delle tabelle;
- d. sviluppare le query SQL che permettono di effettuare le seguenti interrogazioni:
  - i. recuperare l'attuale stato di spedizione di un pacco (l'elenco dei CA e CS da cui è già transitato e l'eventuale stato di avvenuta consegna); di questa query si progettino anche le pagine web che consentono di ottenere le informazioni richieste, scrivendo in un linguaggio a scelta il codice di una parte significativa;
  - ii. il numero di spedizioni che hanno avuto origine da una specifica regione;
  - iii. il numero di spedizioni che hanno avuto origine da ogni regione del territorio nazionale;
  - iv. l'elenco delle regioni che hanno effettuato un numero di spedizioni superiori alla media nazionale;
  - v. l'elenco dei CA che non hanno mai ricevuto pacchi da una regione specificata.

## Esercizi sui Web service<sup>31</sup>

62. Progettare un Web service in grado di gestire le operazioni CRUD su una struttura dati atta a contenere i dati degli impiegati di un'azienda: codice, nome, cognome.

63. Progettare un Web service in grado di gestire le operazioni CRUD su una struttura dati atta a contenere i dati degli utenti che hanno un account su un sito: username, password, nome, e-mail.

64. Progettare un Web service in grado di gestire le operazioni CRUD su una struttura dati atta a contenere i dati dei libri di un negozio di libri: codice, titolo, prezzo.

65. Facendo riferimento ai video contenuti nel capitolo che tratta i Web service (dal 10 fino almeno al 21), si progettino e si sviluppino i Web service necessari per la gestione delle biblioteche di Torino.

Per ogni biblioteca si dovranno memorizzare i seguenti dati:

- codice;
- nome;
- indirizzo;
- telefono del centralino.

Per ogni libro contenuto in una biblioteca si dovranno memorizzare i seguenti dati:

- codice;

---

<sup>31</sup> Per questi esercizi si ringraziano anche il Prof. Giuliano Bellucci per la fantasia nell'inventarne alcuni e il Prof. Fulvio Corno del Politecnico di Torino che ha reso disponibile sul Web il materiale del corso di [Architetture distribuite per i sistemi informativi aziendali](#) dell'a.a. 2008/2009; i suoi esercizi sono stati trasposti per un corso di scuola secondaria di secondo grado.

- titolo;
- autore;
- casa editrice;
- anno di pubblicazione.

Per gestire le biblioteche e i libri si utilizzino delle strutture dati senza usare un DBMS. Si progettino opportunamente tutte le classi Java necessarie per la gestione delle biblioteche e dei libri.

## Voli low cost

66.Una compagnia aerea ha una serie di voli low cost fra capitali europee. Per ogni tratta sono noti la capitale di partenza, la capitale di arrivo, i giorni della settimana in cui si tiene (ad esempio lunedì, mercoledì e venerdì), la capienza dell'aereo (si suppone che su una tratta ci siano sempre aerei con la stessa capienza), l'orario e il prezzo base. Si progetti il Web service che permetta le operazioni CRUD sulle tratte e il relativo client che lo interroga.

### Esempio di operazioni

*Richiesta di tutte le tratte*

GET http://localhost:8080/Esercitazione2

*Richiesta della tratta con id = 5*

GET http://localhost:8080/Esercitazione2/5

*Inserimento di una nuova tratta*

POST http://localhost:8080/Esercitazione2

*dati inviati dal POST in formato XML*

```
<tratta>
    <partenza>Roma</partenza>
    <arrivo>Vienna</arrivo>
    <giorni>
        <giorno value="1"/>
        <giorno value="3"/>
    </giorni> <!-- 1 è domenica (Calendar di java) -->
    <capienza value="30"/>
    <orario value="14.45"/>
    <prezzo value="35" />
</tratta>
```

Modifica dei dati della tratta con id = 5. I dati modificati sono evidenziati in grassetto.

PUT http://localhost:8080/Esercitazione2

```
<tratta id="5">
    <partenza>Roma</partenza>
    <arrivo>Vienna</arrivo>
```

```
<giorni>
  <giorno value="1"/>
  <giorno value="3"/>
</giorni> <!-- 1 è domenica (Calendar di java) -->
<capienza value="30"/>
<orario value="14.30"/>
<prezzo value="40" />
</tratta>
```

Eliminazione della tratta con id = 5

DELETE http://localhost:8080/Esercitazione2/5

# MyShow

67. La nuova società *MyShow* vuole fornire via Internet un nuovo servizio agli utenti registrati sul suo sito. Un utente registrato, dopo aver fornito le credenziali di accesso al sito, seleziona una città per ricevere la lista dei film proiettati.

Selezionando uno dei cinema elencati, l'utente otterrà la lista dei film proiettati nella giornata con le relative informazioni per ogni proiezione (*titolo del film, orari di proiezione, ecc.*)

Per fornire l'elenco dei cinema, il sito della compagnia *MyShow* utilizza un servizio di terze parti offerto dalla compagnia *MovieDB*, che per **ogni cinema** di una **data città** fornisce il **nome del cinema, l'indirizzo, il numero di telefono, l'e-mail e gli orari di apertura**. Inoltre, relativamente alle **proiezioni** dei **singoli cinema**, il servizio offre **l'elenco di tutti i film** proiettati, e per **ogni film** fornisce il **titolo, la descrizione, il rating, la durata, gli orari di proiezione e il periodo di proiezione**.

Si progetti una architettura distribuita basata sulla tecnologia dei Web service REST utilizzando le servlet, che implementi il servizio offerto dalla compagnia *MovieDB* alla società *MyShow*, e il relativo servizio offerto da quest'ultima ai suoi utenti.

Il progetto dovrà comprendere l'architettura complessiva del sistema suddivisa in *tier*, dei quali si dovrà fornire una rappresentazione grafica e una descrizione di massima dei *tier* coinvolti e della relativa funzione, considerando i punti di seguito indicati.

1. Resource Management Layer (*Data Layer, Back End*):
  - a. progettare il database gestito dalla compagnia *MovieDB* dei cinema nelle relative città e dei film in essi proiettati;
  - b. progettare il database degli utenti gestito dalla compagnia *MyShow*.
2. Presentation Layer (*Front End*):
  - a. progettare l'interfaccia della pagina Web del servizio offerto dalla società *MyShow* che dovrà prevedere l'autenticazione o la registrazione dell'utente;
  - b. progettare l'interfaccia con cui l'utente, una volta effettuato il login, selezionerà una città per ottenere l'elenco dei cinema e dei film con le informazioni associate;
3. Business Logic Layer (*Application Layer, Logic Layer*):
  - a. la descrizione funzionale della servlet utilizzata dalla società *MyShow* per autenticare gli utenti;
  - b. la query SQL utilizzata dalla compagnia *MovieDB* per ottenere l'elenco dei cinema di una città (con tutte le informazioni per raggiungerli) e l'elenco dei film proiettati in ognuno di essi (con tutti i dati associati) da inviare successivamente a *MyShow*;
  - c. la descrizione del Web service offerto dalla compagnia *MovieDB* alla società *MyShow* indicando:
    - i. la descrizione funzionale del Web service REST, implementato tramite servlet, esposto dalla compagnia *MovieDB* per offrire il servizio di ricerca richiesto dalla società *MyShow*;
    - ii. il metodo HTTP utilizzato per invocarlo secondo l'architettura REST;
    - iii. l'URL usato per l'invocazione;
    - iv. un esempio di file XML restituito dal servizio.

Infine si proponga e si illustri un progetto di massima del sistema hardware/software che comporti la installazione del server web:

- presso la compagnia MyShow
- in hosting presso un provider ISP.

Per ciascun scenario si illustrino le tecniche di sicurezza che rendono sicuro l'accesso al server.

Se utilizzato durante le lezioni di informatica, o durante gli anni precedenti, potrete illustrare la collaborazione fra i vari elementi del sistema distribuito utilizzando dei diagrammi UML<sup>32</sup>.

## Realizzazione

Si realizzi un prototipo funzionante del sistema sopra progettato. Se possibile si utilizzino tanti computer quanti sono i sistemi informativi coinvolti.

## Sviluppi futuri

1. Sviluppare una app su Android per l'offerta del servizio di *MyShow*.
2. Permettere di restringere la ricerca del cinema nell'ambito di una distanza stabilita dall'utilizzatore dell'app.

---

<sup>32</sup> [Introduction to the Diagrams of UML 2.X](#)  
[The Unified Modeling Language](#)  
[An introduction to the Unified Modeling Language](#)  
[WebSequenceDiagrams](#)

## Centro assistenza

68. Un'azienda produttrice di automobili ha circa 10000 *centri di assistenza* distribuiti in tutto il mondo. Ogni cliente che acquista un'automobile può richiedere la manutenzione della stessa in garanzia nel caso in cui essa presenti dei problemi di funzionamento. Tutte le spese relative alla gestione della manutenzione dell'automobile sono a totale carico dell'azienda produttrice.

Per consentire una migliore integrazione tra la dirigenza dell'azienda ed i *centri di assistenza*, l'azienda permette ai *centri* di poter accedere rapidamente ed in tempo reale ai dati principali riguardanti il veicolo che necessita assistenza. Tali dati comprendono, ad esempio, i dati del proprietario, la validità del contratto di assistenza, gli interventi di manutenzione precedenti per quel veicolo, i difetti frequenti riscontrati per il modello di veicolo, eccetera.

L'azienda ha quindi **due necessità**:

1. **raccogliere informazioni** sugli interventi effettuati da parte dei *centri di assistenza*;
2. **fornire** a tali *centri*, nel momento in cui si presenta un veicolo, tutte le **informazioni** necessarie.

I *centri di assistenza*, nei vari continenti, possiedono ognuno un portale Internet, indipendente da quello dell'azienda, ma che segue un layout comune fornito da essa, attraverso il quale possono offrire alcuni servizi personalizzati ai propri clienti. Il **portale** è anche **usato** dal *centro di assistenza* per accedere al servizio offerto dall'azienda **per ottenere le informazioni** su un'automobile o per **inviare i dati di un nuovo intervento** effettuato sull'auto di un cliente.

Si progetti il sistema descritto, facendo tutte le ipotesi aggiuntive e prevedendo una architettura *multi-tier*, considerando tutti i punti seguenti.

1. Presentation Layer (*Front End*):
  - a. progettare l'interfaccia delle **pagine Web** di un generico **portale** che permetta ad un *centro di assistenza* di **autenticarsi**, **gestendo** anche gli eventuali **errori**;
  - b. progettare l'interfaccia della **pagina Web** per la **richiesta dei dati di un'automobile** in manutenzione presso il *centro di assistenza*;
  - c. progettare l'interfaccia della **pagina Web** che **visualizzi i dati dell'auto richiesta**;
  - d. progettare l'interfaccia della **pagina Web** che permetta di **inserire ed inviare i dati di manutenzione effettuati su un'auto**.
2. Resource Management Layer (*Data Layer, Back End*) costituito da un database gestito dall'azienda produttrice, che descriva le realtà seguenti:
  - a. i **clienti** che hanno acquistato un'auto dell'azienda produttrice;
  - b. le **automobili** prodotte dall'azienda;
  - c. gli **interventi effettuati** sull'auto di un cliente da parte del *centro di assistenza*;
  - d. i **centri di assistenza** sparsi nel mondo;
3. Business Logic Layer (*Application Layer, Logic Layer*):

- a. la **query SQL** utilizzata dal *centro di assistenza* **per effettuare il login** nel portale dell'azienda;
- b. la **query SQL** utilizzata dal *centro di assistenza* **per ottenere tutti i dati** dell'auto di un cliente;
- c. la **query SQL** utilizzata dal *centro di assistenza* **per aggiungere dati** nel database dell'azienda riguardanti un nuovo intervento effettuato sull'auto di un cliente;
- d. la **descrizione del Web service REST** offerto dall'azienda produttrice ai *centri di assistenza* per poter eseguire tutte le query SQL descritte precedentemente, indicando per ognuna di esse:
  - i. il **metodo HTTP** da utilizzare secondo l'architettura REST;
  - ii. l'**URL** usato per l'invocazione, secondo i principi dell'architettura REST;
  - iii. un esempio di **file XML** restituiti dal servizio o inviati dal *centro di assistenza*.

Se utilizzato durante le lezioni di informatica, o durante gli anni precedenti, potrete illustrare la collaborazione fra i vari elementi del sistema distribuito utilizzando dei diagrammi UML<sup>33</sup>.

## Realizzazione

Si realizzi un prototipo funzionante del sistema sopra progettato. Se possibile si utilizzino tanti computer quanti sono i sistemi informativi coinvolti.

---

<sup>33</sup> [Introduction to the Diagrams of UML 2.X](#)  
[The Unified Modeling Language](#)  
[An introduction to the Unified Modeling Language](#)  
[WebSequenceDiagrams](#)

## GTT (*trasporre*)

69. La società GTT (Gruppo Trasporti Timbuktu) ha recentemente integrato una serie di altre società locali di trasporti dello stato di Mali. Come parte della nuova strategia gestionale, la GTT intende fornire ai propri utenti un servizio centralizzato di consultazione degli orari, di calcolo dei percorsi e di acquisto dei documenti di viaggio (biglietti).

La società GTT costruisce quindi un proprio sistema informativo, accessibile dai propri utenti mediante un portale web, che offre le funzioni citate. Ovviamente le informazioni riguardo i tragitti, gli orari, i prezzi sono di competenza delle singole società locali di trasporti, ciascuna delle quali ha nel tempo sviluppato il proprio sistema informativo.

Descrivere una possibile modalità di integrazione (quale tecnologia usare, quali dati scambiare, dove memorizzare i dati, ...) tra i sistemi informativi della GTT e quelli delle società locali di trasporti, in grado di supportare le funzionalità descritte (consultazione degli orari, calcolo automatico dei percorsi, acquisto dei documenti di viaggio). In particolare si consideri che per il calcolo dei percorsi è necessario integrare i dati delle diverse società locali, proponendo anche percorsi "misti", ossia composti da tratte gestite da società locali distinte.

## Dottorato (*trasporre*)

70. Gli studenti di Dottorato di Ricerca, nella loro formazione specialistica, devono spesso seguire corsi di dottorato che si tengono presso varie università, anche diverse dalla propria università di iscrizione.

Le Università intendono potenziare il proprio sistema informativo per permettere a tali studenti di trattare anche i corsi presso altre università alla stessa stregua dei corsi “interni”. Possiamo supporre che ciascuna università abbia un sistema equivalente al sistema proposto dal Politecnico di Torino, in cui esiste la gestione del carico didattico, prenotazione esami, visione del libretto elettronico, ecc. Tuttavia, possiamo essere certi che le varie università abbiano implementato il sistema in modi anche molto diversi, sia dal punto di vista dell’interfaccia utente che dal punto di vista delle tecnologie coinvolte.

Descrivere una possibile soluzione al problema dell’integrazione tra i sistemi informativi, che permetta ad uno studente iscritto in una qualsiasi università partecipante di visionare i corsi disponibili, inserirli nel proprio carico didattico, comparire negli elenchi ufficiali visibili al docente, prenotarsi all’esame, ed avere registrato il proprio voto.

## Olimpiadi (*trasporre*)

71. In vista delle prossime Olimpiadi, il comitato olimpico del paese X intende offrire a tutti i giornalisti (ed in generale ai media interessati) un servizio in tempo reale con i risultati di tutti gli eventi sportivi in corso.

In particolare, dovranno essere disponibili in modalità distribuita: i calendari delle gare, i risultati di ciascuna gara (in cui per ciascun atleta o squadra partecipante è indicato il punteggio o misura significativa), ed il “medagliere” (le medaglie assegnate ai vari atleti o squadre, divise per nazione e per tipo di medaglia).

Si tenga presente che anche l’Olimpiade stessa è un evento “distribuito”, poiché le gare si svolgono in luoghi geograficamente anche molto lontani; per questo motivo occorre identificare anche la modalità migliore di aggiornamento delle informazioni da parte degli organizzatori e dei giudici.

Si progetti, per conto del comitato olimpico del paese X, il sistema informativo che dovrà gestire la pubblicazione di tali dati e la relativa gestione ed aggiornamento.

## Grande Fratello (*trasporre*)

72. Il Ministero delle Finanze intende investire in un sistema informativo in grado di combattere alla radice il fenomeno dell'evasione fiscale. Il principio guida di tale riforma –denominata Grande Fratello– è che esistono già molti sistemi informativi “parziali”, alcuni pubblici (catasto, anagrafe dei residenti, motorizzazione civile, ...) ed altri privati (banche, assicurazioni, finanziarie, ...). Il Ministero ritiene che “incrociando” tali dati sia possibile rilevare molte “anomalie” o situazioni sospette, grazie al fatto che tutti i sistemi informativi fanno capo comunque all’identificazione mediante codice fiscale (si tralasci il caso degli evasori totali o di coloro che si servono di prestanomi che non verrebbero comunque identificati da questo sistema).

In particolare, il Ministero dovrà poter consultare, in ogni momento, i dati relativi ad un determinato contribuente nelle varie base dati disponibili, ciascuna delle quali ovviamente conterrà dati di natura diversa. Deve inoltre essere possibile, da parte da qualsiasi ente, “segnalare” un determinato contribuente, qualora l’ente abbia il sospetto di comportamenti anomali.

Si progetti, per conto del Ministero delle Finanze, il sistema informativo che dovrà gestire la condivisione di tali dati ed il meccanismo di segnalazione. Le scelte progettuali dovranno mirare alla minimizzazione dei costi per i vari sistemi informativi che dovranno essere adattati ed integrati, mentre non vi sono praticamente vincoli di complessità sul sistema del Ministero.

## UVI (*trasporre*)

73. Una grande catena di supermercati intende realizzare un sistema di videosorveglianza intelligente, in grado di osservare i comportamenti degli utenti nei propri punti vendita e dedurne informazioni di vario tipo (ad esempio, l'interesse relativo ai prodotti esposti, oppure sul rischio di taccheggio).

Il sistema è basato su una serie di Unità Video Intelligenti (UVI) dotate ciascuna di una telecamera e di un computer collegato alla rete. Ciascuna UVI (ve ne possono essere diverse in ciascun punto vendita) analizza il flusso di immagini provenienti dalla telecamera, e ne estrae alcuni **dati sintetici** significativi. Ad esempio, associa ad ogni persona diversa un diverso ID numerico autogenerato, rappresenta tale persona mediante delle caratteristiche biometriche in grado di riconoscere se la stessa persona compare nel campo visivo di più UVI, identifica una serie di "gesti" o "posture" (es: guarda, prende, confronta, ...) per ciascuna persona. In aggiunta, ciascuna UVI salva anche sul proprio hard disk i **dati completi** (tutti i fotogrammi e tutti i dati da essi estratti) relativi alle ultime 48 ore.

L'insieme di tutti gli UVI è collegato con il Centro di Elaborazione della grande catena, il quale è deputato a ricevere le informazioni sintetiche da tutti gli UVI supervisionati ed a compiere delle ulteriori elaborazioni sulle informazioni ricevute. Per alcune di tali elaborazioni (es. tracciamento del percorso di un cliente tra i campi visivi dei vari UVI) sono sufficienti i **dati sintetici** forniti dagli UVI. Per altre elaborazioni (es. invio alla polizia del filmato di un potenziale taccheggiatore) sono invece necessari i **dati completi**.

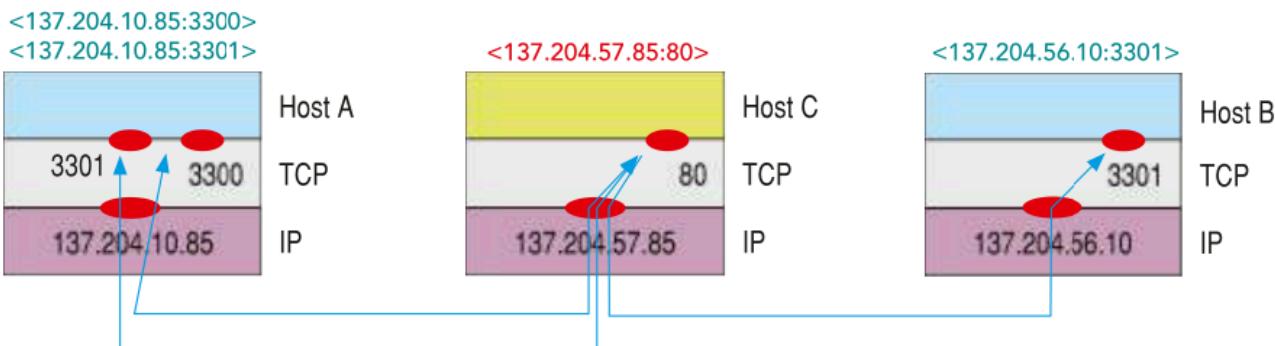
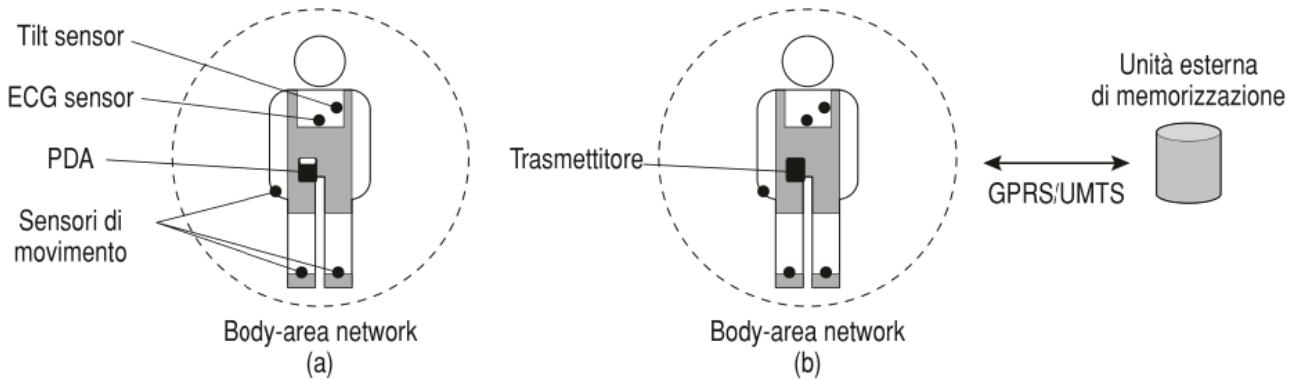
Ovviamente le capacità trasmissive di Internet non permettono in alcun modo la trasmissione continua, completa ed in tempo reale dei **dati completi** da parte di tutti gli UVI contemporaneamente, mentre invece per i **dati sintetici** non vi sono problemi di banda.

Si progetti, per conto della grande catena, il sistema informativo che dovrà gestire la raccolta dei dati dagli UVI ed il meccanismo di comunicazione tra questi ultimi ed il centro di elaborazione. Le scelte progettuali dovranno mirare alla compatibilità con la banda disponibile su Internet, permettendo però una rilevazione tempestiva di eventuali comportamenti anomali.

# Colloquio

Il candidato analizzi e commenti il materiale fornito, facendo i collegamenti necessari anche con altre discipline.

---



---

```
private ServerSocket server = new ServerSocket(40000);
```

---

# SERVIZIO



---

```
private DatagramSocket socket = new DatagramSocket(40000);
```

---

### TFTP Formats

Type Op # Format without header

	2 bytes	string	1 byte	string	1 byte
-----					
RRQ /	01/02	Filename	0	Mode	0
WRQ	-----				
	2 bytes	2 bytes	n bytes		
-----					
DATA	03	Block #		Data	
-----					

If data length < 512 byte it signals the end of the transfer

	2 bytes	2 bytes		
-----				
ACK	04	Block #		
-----				
	2 bytes	2 bytes	string	1 byte
-----				
ERROR	05	ErrorCode	ErrMsg	0
-----				

---

# SAFE - IDEMPOTENT - CACHEABLE

---

Il seguente protocollo permette di gestire i saldi dei conti correnti bancari di una Banca usando un servizio offerto da un server remoto. Il protocollo può essere implementato sia su TCP, sia su UDP. Spiega quali caratteristiche protocollari sono presenti

Formato dei messaggi del protocollo applicativo lato client.

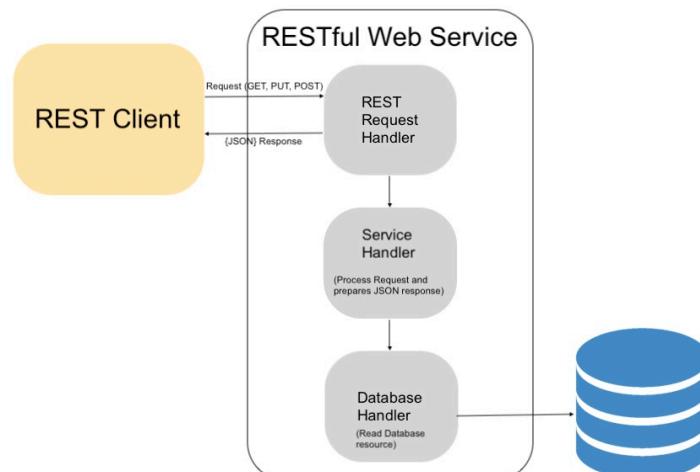
Comando	Operazione
USE,n	Richiesta di effettuare delle operazione sul conto n
VER,num	Versamento sul conto in uso della somma num

PRE,num	Prelievo dal conto in uso della somma num
SLD	Richiesta di invio del saldo del conto in uso
STP,n	Terminazione delle operazione sul conto n

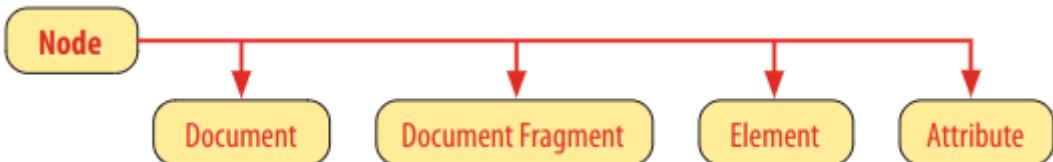
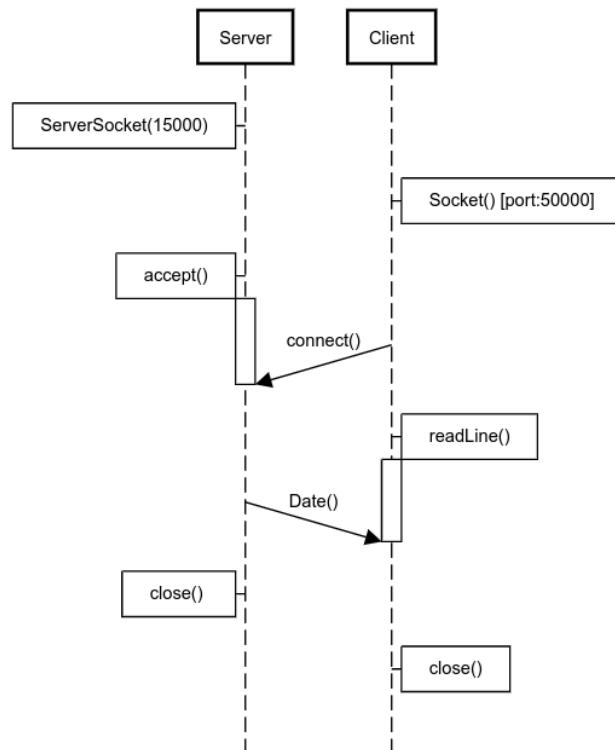
Formato dei messaggi del protocollo applicativo lato server.

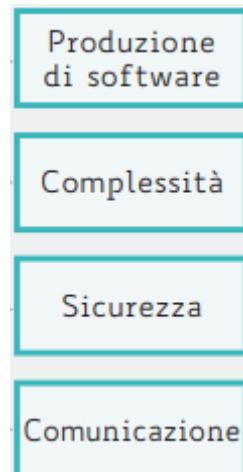
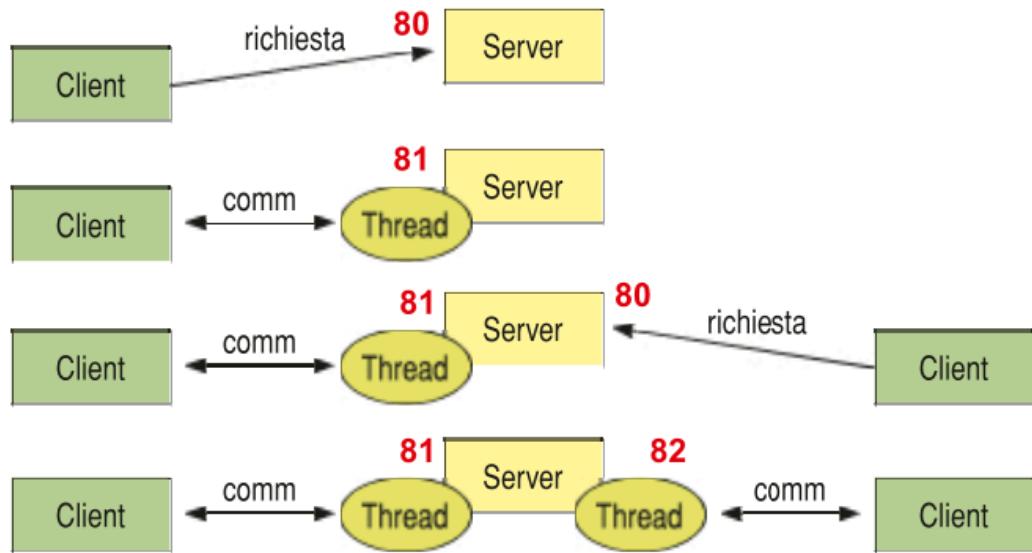
Comando	Operazione
AKUSE,n	Accettazione della richiesta di operare sul conto n
AKVER,num	Segnalazione che il versamento della somma num è stata effettuata
AKPRE,num	Segnalazione che il prelievo della somma num è stato effettuato
SLD,num	Invio del saldo richiesto
AKSTP,n	Segnalazione che sono state chiuse le operazioni sul conto n
ERR,codErr	<p>codErr</p> <ul style="list-style-type: none"><li>• 1 - operazione non riconosciuta</li><li>• 2 - valore numerico non valido</li><li>• 3 - comando non valido</li></ul>

Il client può richiedere più operazioni sul conto ma solo dopo aver ricevuto la risposta di una richiesta fatta in precedenza.

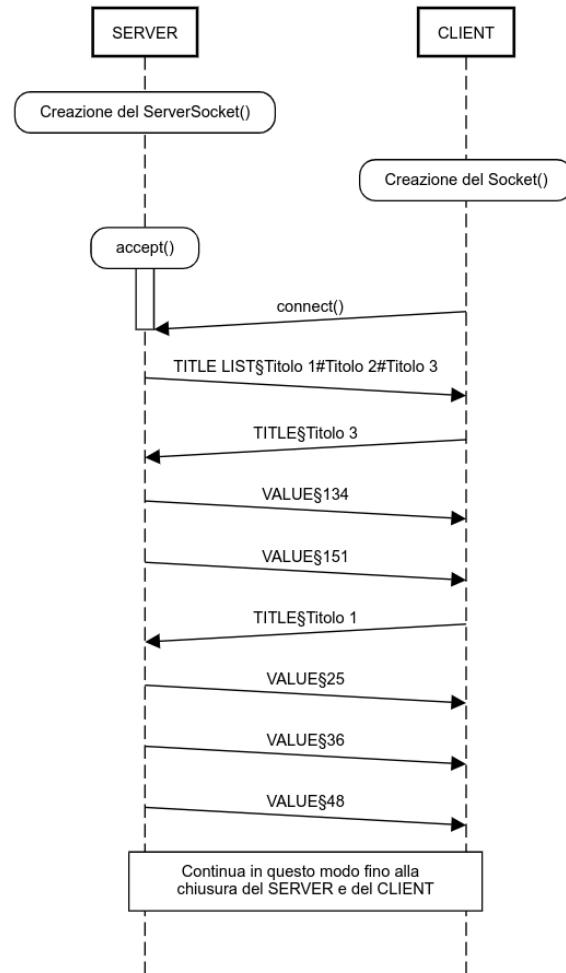


The HTTP Method	Path to the source on Web Server	Protocol Version Browser supports
	Post /RegisterDao.jsp	HTTP/1.1
The Request Headers	Host: www.javatpoint.com	
	User-Agent: Mozilla/5.0	
	Accept: text/xml,text/html,text/plain,image/jpeg	
	Accept-Language: en-us,en	
	Accept-Encoding: gzip,deflate	
	Accept-Charset: ISO-8859-1,utf-8	
	Keep-Alive:300	
	Connection:keep-alive	
	User=ravi&pass=java	Message body





# REST

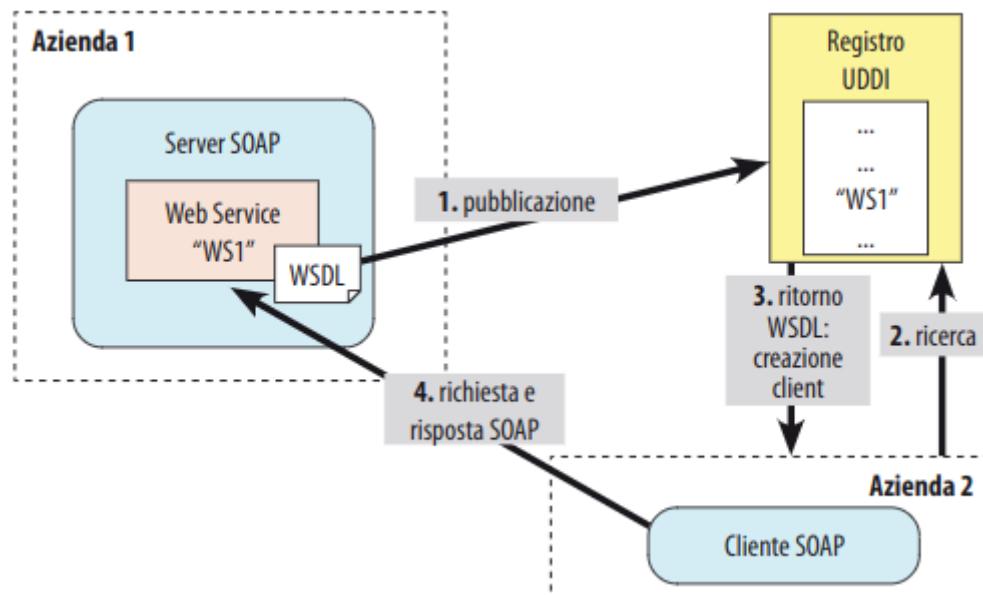


# SERVER CONCORRENTE

Considerando i seguenti messaggi per la gestione di una calcolatrice, spiega quali caratteristiche protocollari sono presenti:

Comando	Operazione
ADD , X , Y	Addizione: (X+Y)
SUB , X , Y	Sottrazione (X-Y)
MUL , X , Y	Moltiplicazione (X * Y)
DIV , X , Y	Divisione (X : Y)
POW , X , Y	Potenza ( $X^Y$ )

La calcolatrice risponde con un messaggio di errore se il comando non è fornito secondo la sintassi corretta.



# PROTOCOLLO

Spiega quali sono le caratteristiche del seguente protocollo.

Network Working Group  
Request for Comments: 862

J. Postel  
ISI  
May 1983

### Echo Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Echo Protocol are expected to adopt and implement this standard.

A very useful debugging and measurement tool is an echo service. An echo service simply sends back to the originating source any data it receives.

#### TCP Based Echo Service

One echo service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 7. Once a connection is established any data received is sent back. This continues until the calling user terminates the connection.

#### UDP Based Echo Service

Another echo service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 7. When a datagram is received, the data from it is sent back in an answering datagram.

---

Sistemi  
di calcolo

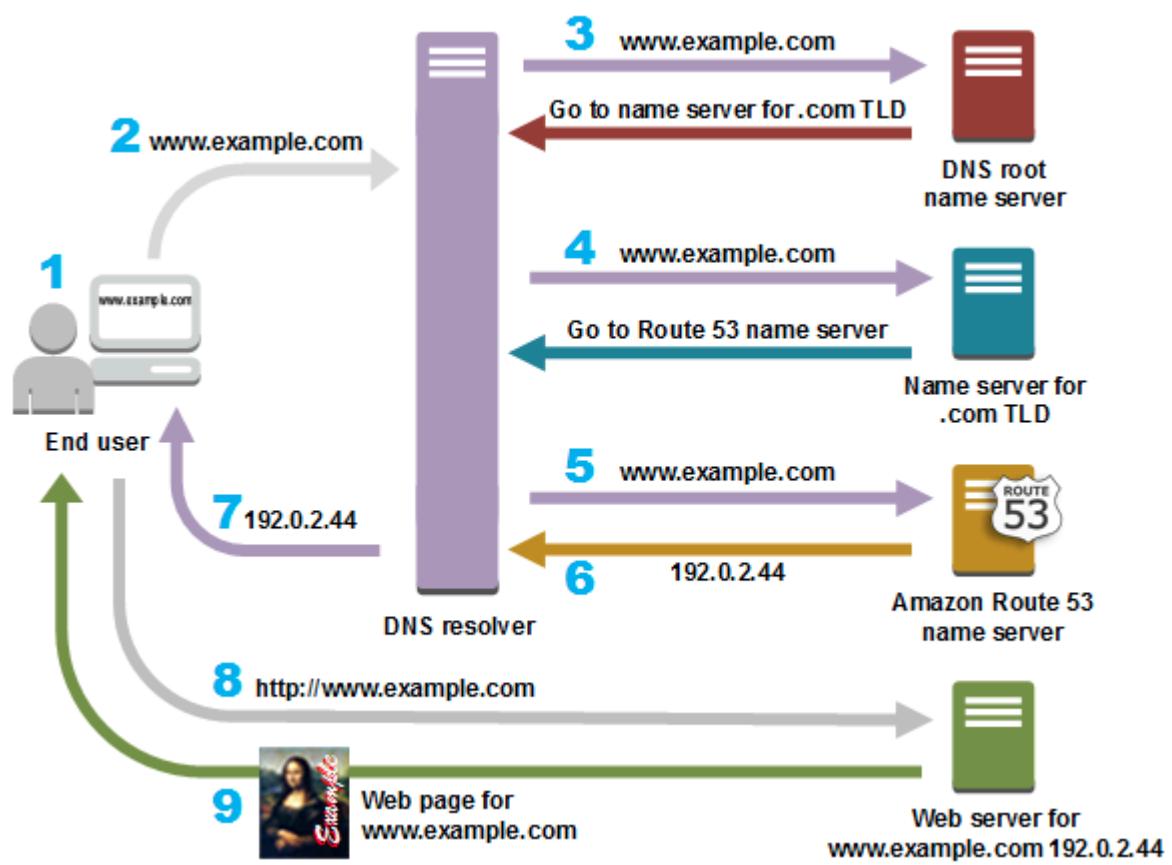
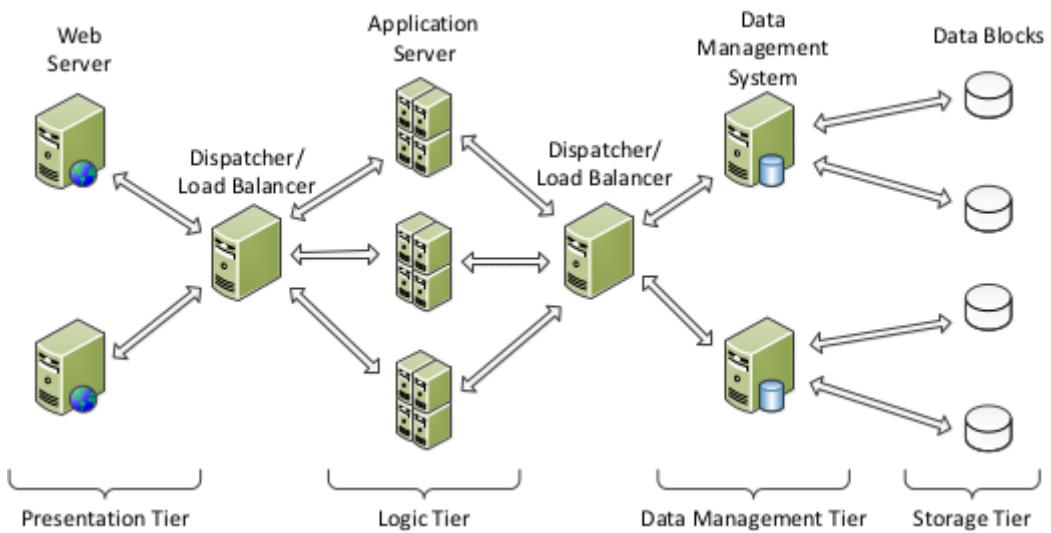
Sistemi  
informativi

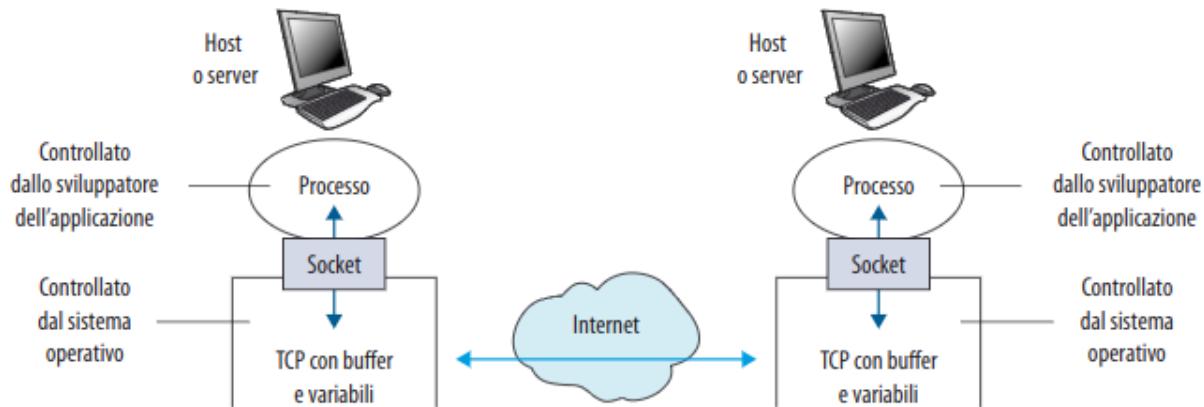
Sistemi  
pervasivi

---

JSON

---





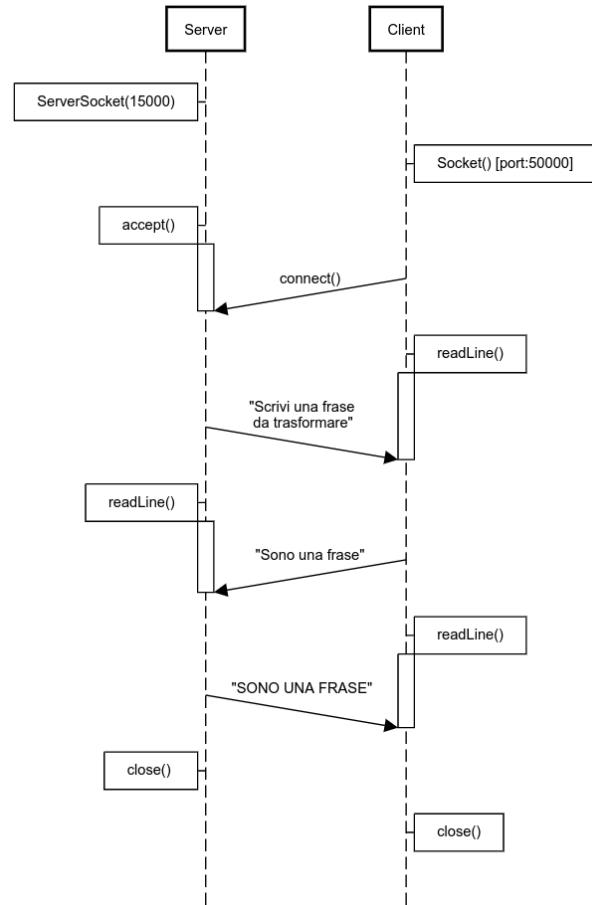
Un impianto di condizionamento centrale di un grande edificio gestisce decine di sensori di temperatura e umidità per regolare le condizioni ambientali delle varie stanze. I sensori sono connessi alla rete LAN dei dispositivi dell'edificio.

L'impianto di condizionamento centrale e i sensori usano il protocollo di trasporto UDP per comunicare, e l'impianto di condizionamento riceve periodicamente le misure dai sensori, distinte tra la misurazione della temperatura e quella dell'umidità.

Tutti i sensori di rilevamento sono individuati in modo inequivocabile da un codice univoco, in modo tale che l'impianto di condizionamento centrale possa distinguere le misurazioni ricevute dai vari sensori.

Descrivi le caratteristiche del protocollo.

The HTTP Method	Path to the source on Web Server	Parameters to the server	Protocol Version Browser supports
			HTTP/1.1
The Request Headers	GET /RegisterDao.jsp?user=ravi&pass=java	Host: www.javatpoint.com User-Agent: Mozilla/5.0 Accept-text/xml,text/html,text/plain,image/jpeg Accept-Language: en-us,en Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8 Keep-Alive: 300 Connection: keep-alive	



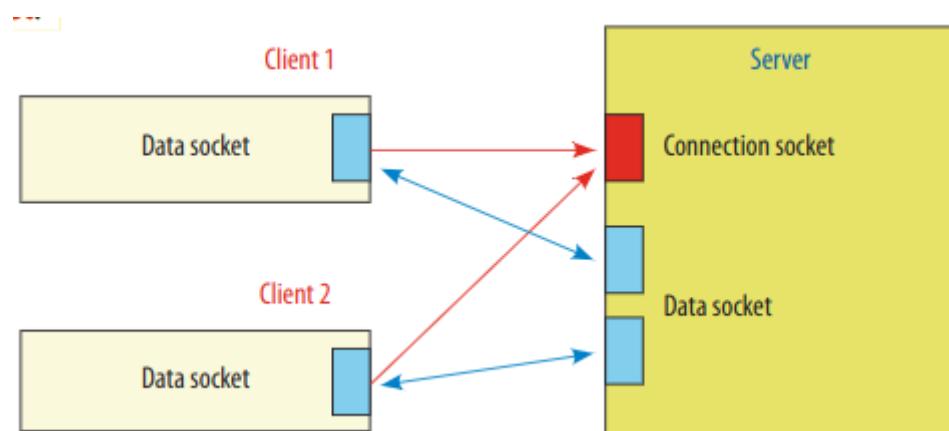
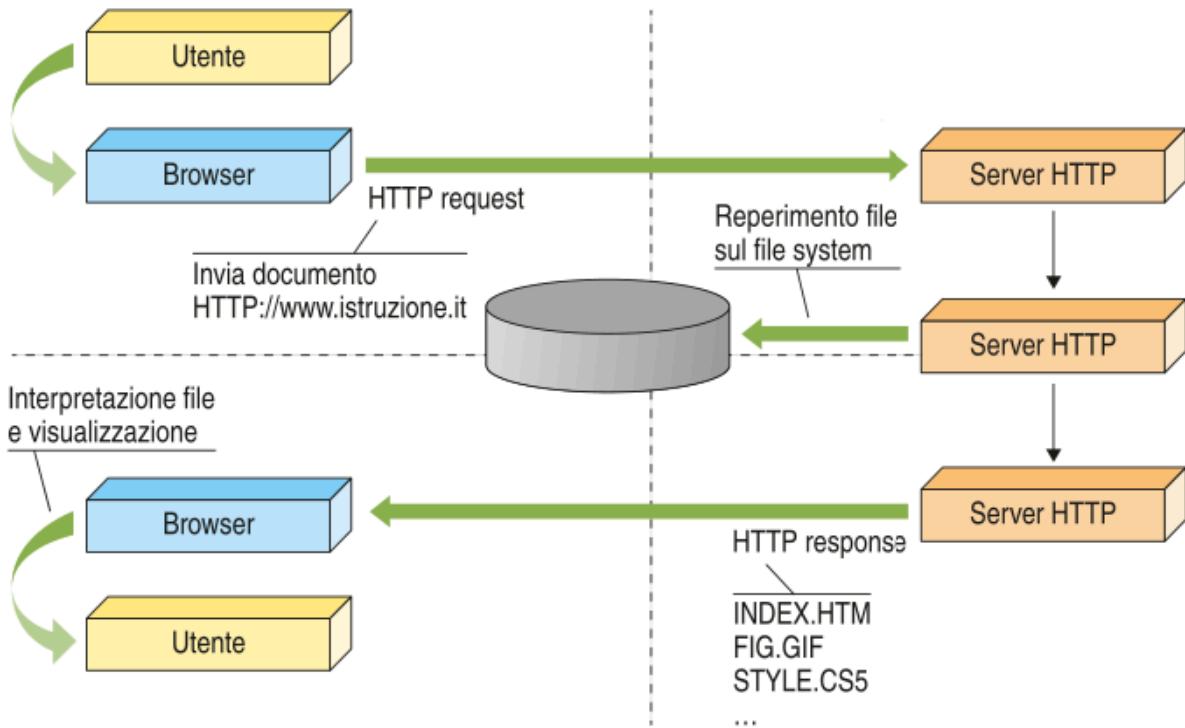
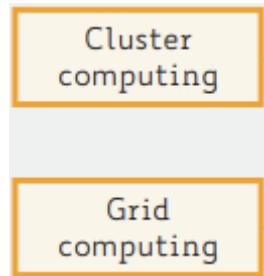
---

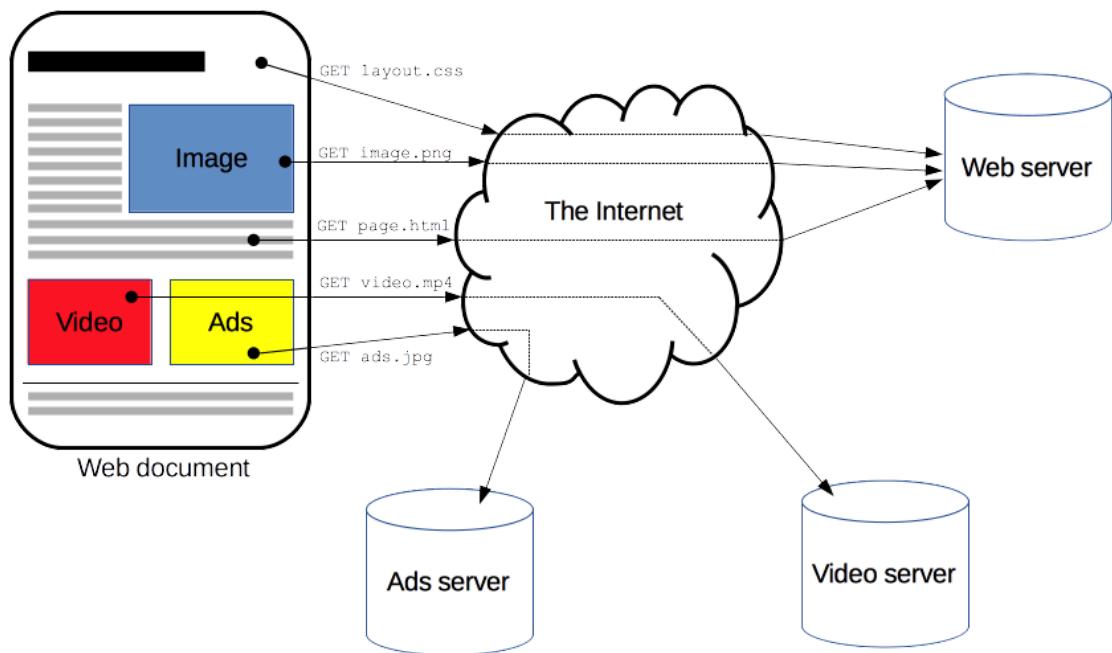
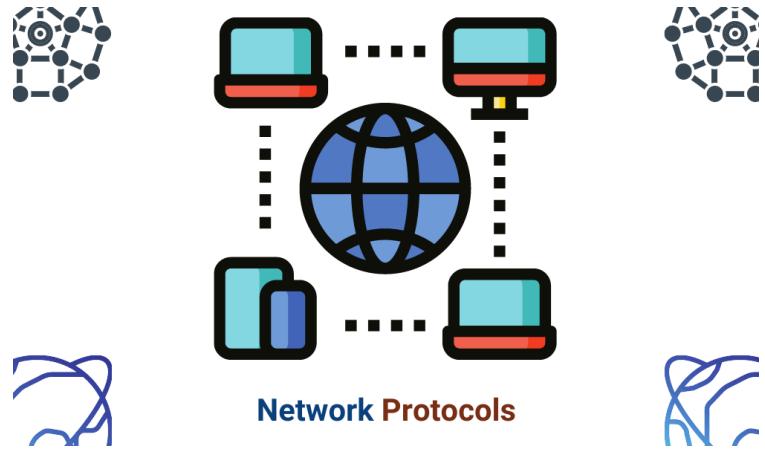
# URI

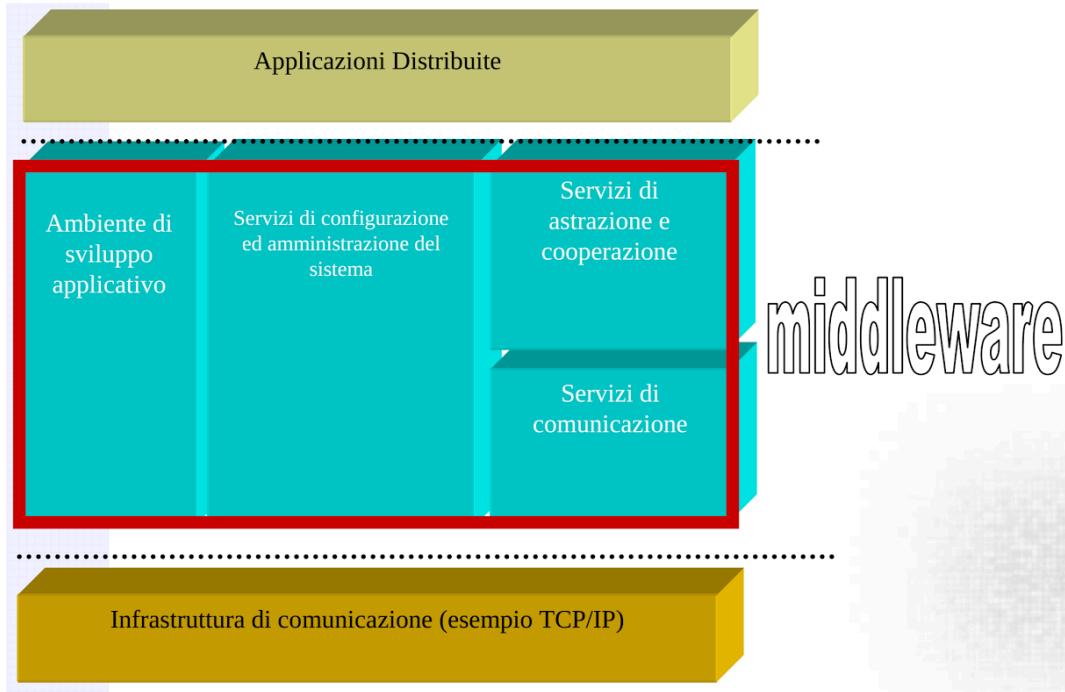
---

# RISORSE

---







```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".QuzeActivity"
    android:background="@color/colorBackground"
    android:padding="16dp">
    <TextView
        android:id="@+id/text_view_score"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="score"/>
    <TextView
        android:id="@+id/text_view_question"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="question"
        android:layout_below="@+id/text_view_hige_score"/>
</RelativeLayout>
```

---

Spiega quali sono le caratteristiche del seguente protocollo.

Network Working Group  
Request for Comments: 864

INTERNET STANDARD  
Errata Exist  
J. Postel  
ISI  
May 1983

#### Character Generator Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on

the ARPA Internet that choose to implement a Character Generator Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a character generator service. A character generator service simply sends data without regard to the input.

#### TCP Based Character Generator Service

One character generator service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 19. Once a connection is established a stream of data is sent out the connection (and any data received is thrown away). This continues until the calling user terminates the connection.

It is fairly likely that users of this service will abruptly decide that they have had enough and abort the TCP connection, instead of carefully closing it. The service should be prepared for either the careful close or the rude abort.

The data flow over the connection is limited by the normal TCP flow control mechanisms, so there is no concern about the service sending data faster than the user can process it.

#### UDP Based Character Generator Service

Another character generator service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 19. When a datagram is received, an answering datagram is sent containing a random number (between 0 and 512) of characters (the data in the received datagram is ignored).

There is no history or state information associated with the UDP version of this service, so there is no continuity of data from one answering datagram to another.

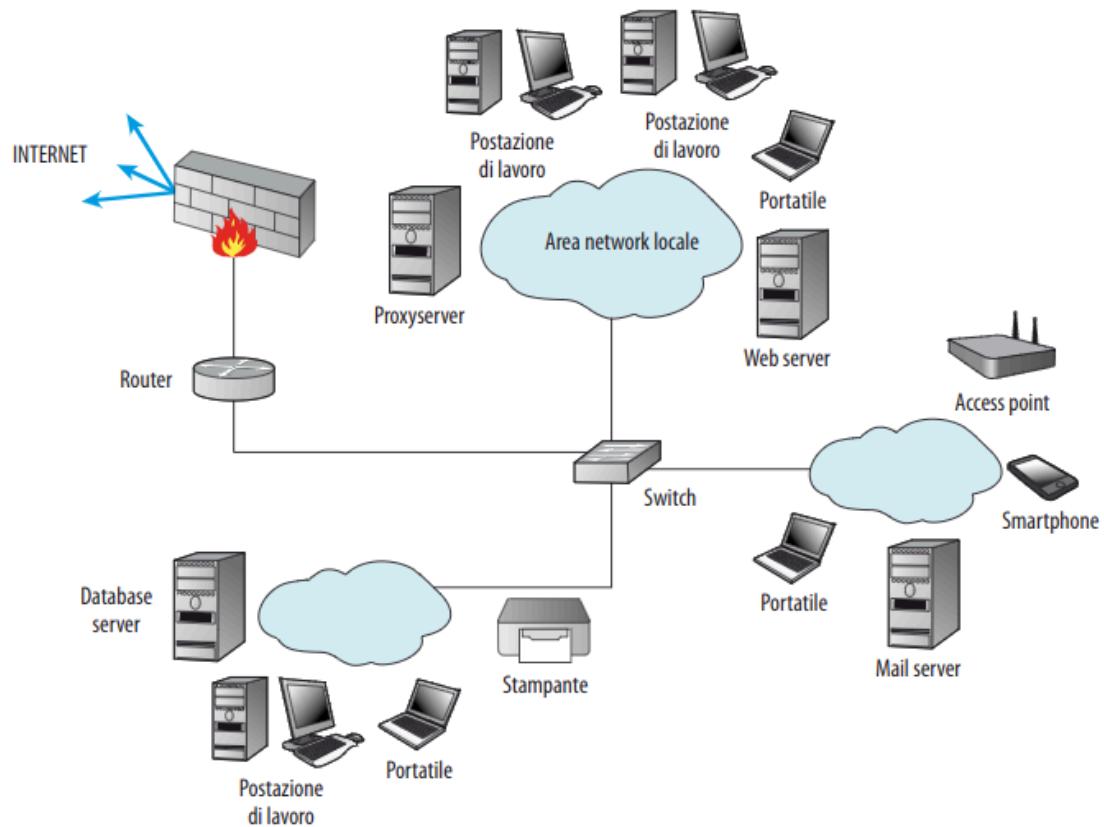
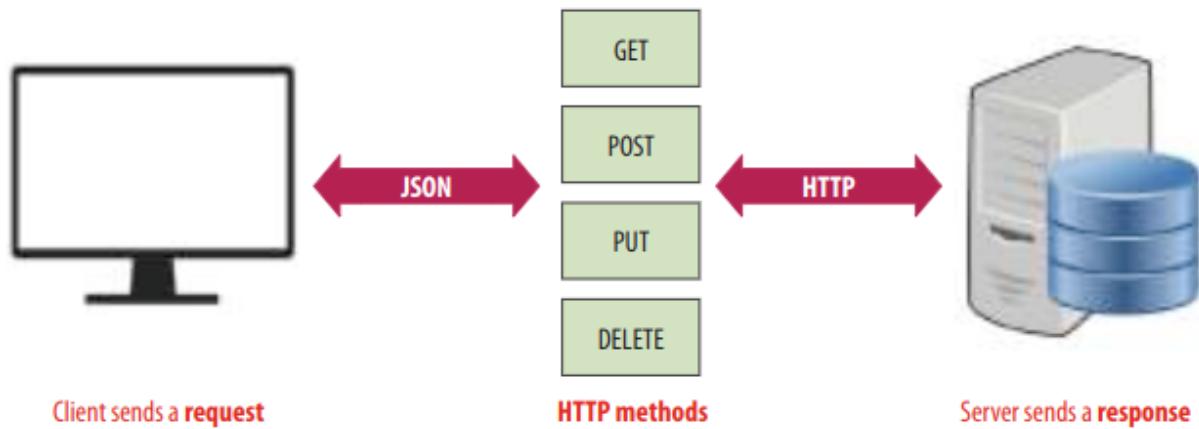
The service only send one datagram in response to each received datagram, so there is no concern about the service sending data faster than the user can process it.

#### Data Syntax

The data may be anything. It is recommended that a recognizable pattern be used in the data.

One popular pattern is 72 character lines of the ASCII printing characters. There are 95 printing characters in the ASCII character set. Sort the characters into an ordered sequence and number the characters from 0 through 94. Think of the sequence as a ring so that character number 0 follows character number 94. On the first line (line 0) put the characters numbered 0 through 71. On the next line (line 1) put the characters numbered 1 through 72. And so on. On line N, put characters ( $0+N \bmod 95$ ) through ( $71+N \bmod 95$ ). End each line with carriage return and line feed.

## Example



HTTP

# GET, POST, PUT, DELETE

---

I nuovi contatori dell'energia elettrica sono dispositivi interrogabili mediante WiFi. La lettura di tutti i contatori di un condominio avviene inviando in broadcast un datagram UDP contenente il codice del condominio (un valore a 32 bit) e scrivendo sul dispositivo di lettura, in un file di testo in formato CSV, l'elenco di tutte le risposte ricevute costituite dal codice del condominio, dalla matricola del contatore e dalla lettura rilevata (tutti valori a 32 bit). Il lettore ha un tempo di attesa limite delle letture da rilevare.

Descrivi le caratteristiche del protocollo.

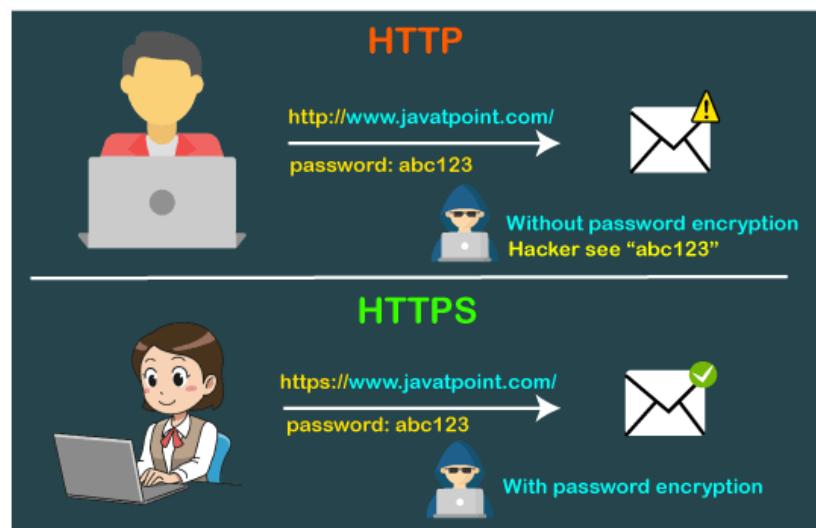
---

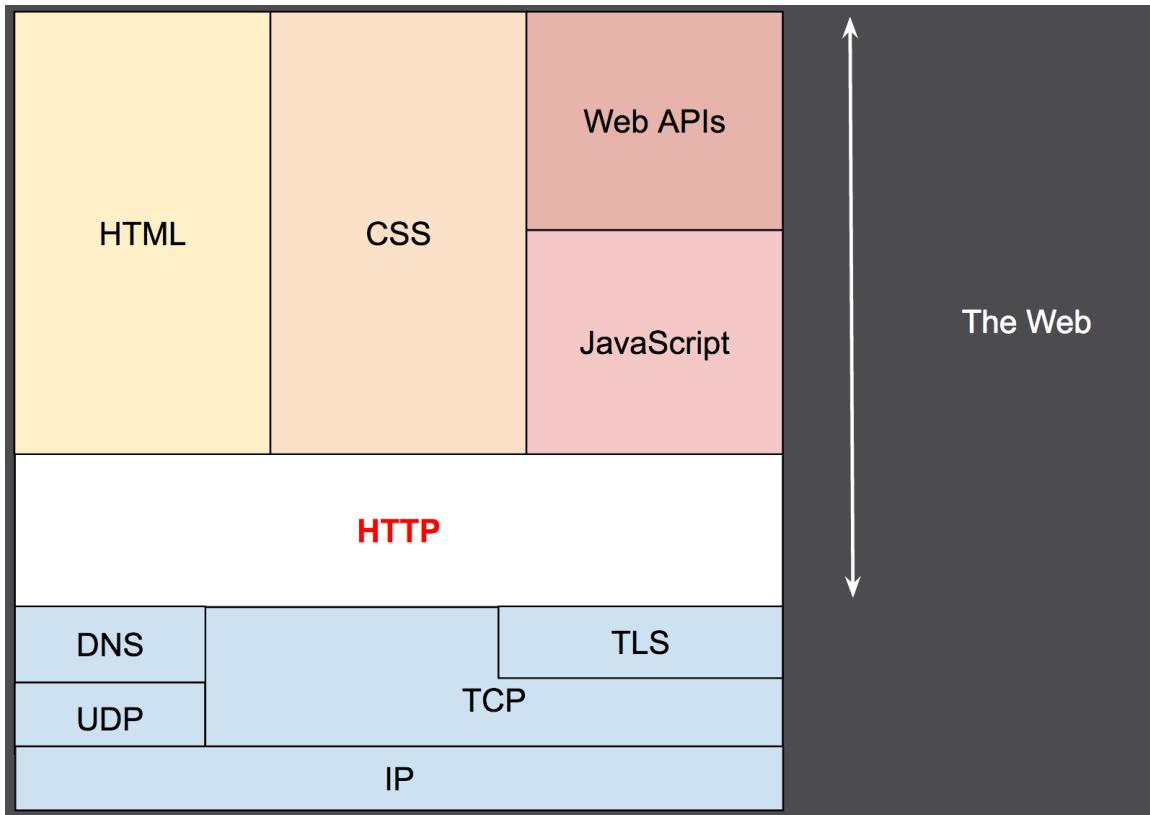
# XML

---

<http://mysite.com/CountryPopulation/webapi/countries>

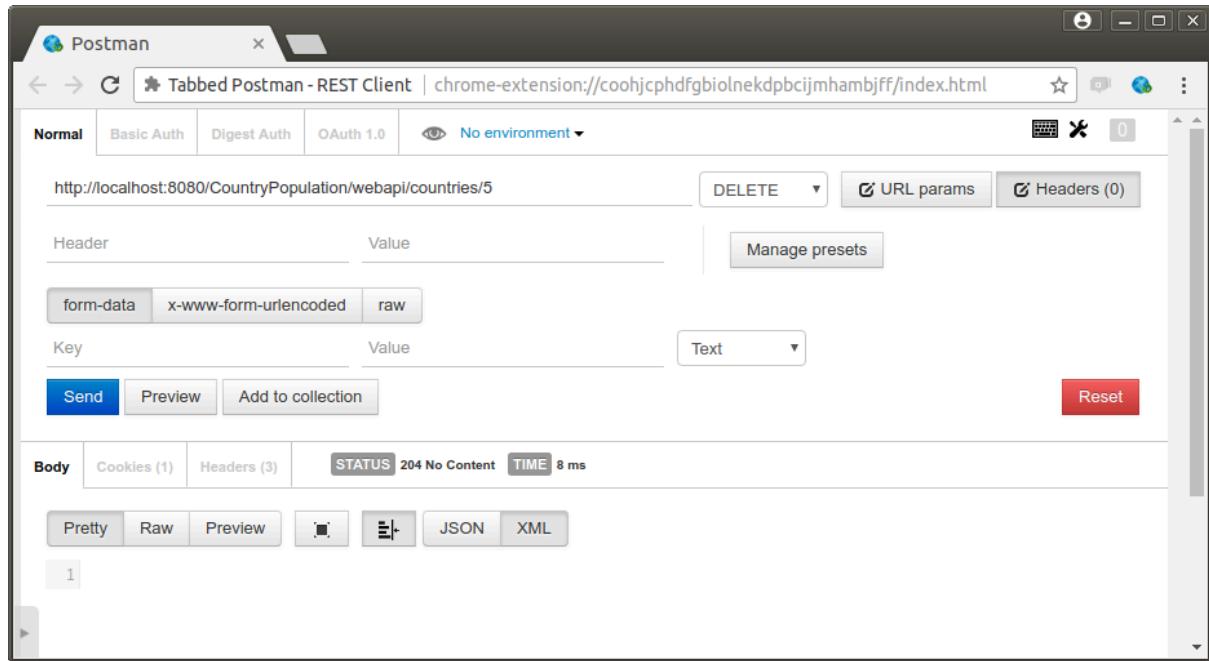
---



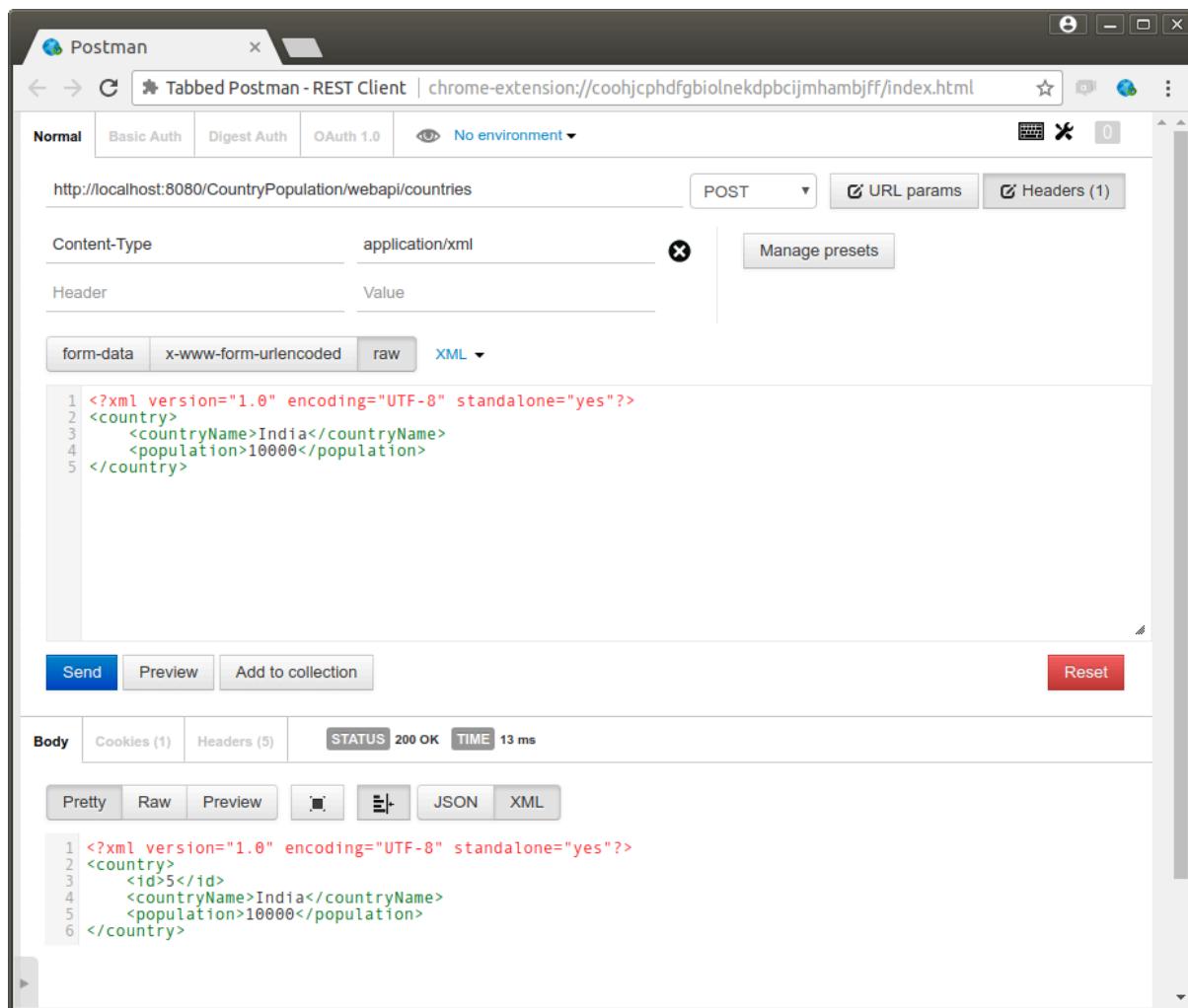
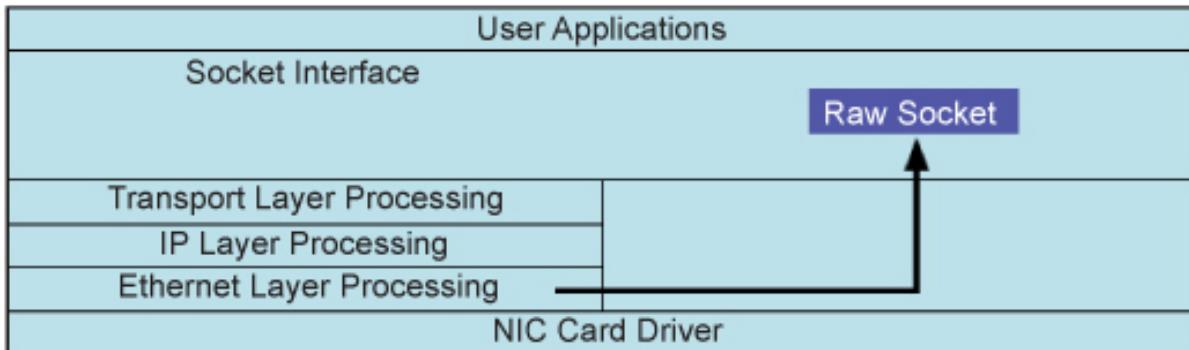


```
<ordine id="123" data="10/03/2017">
  <cliente>Mario Rossi</cliente>
  <stato>EVASO</stato>
  <dettagli>
    <dettaglio>
      <articolo>Libro</articolo>
      <quantita>1</quantita>
    </dettaglio>
    <dettaglio>
      <articolo>CD</articolo>
      <quantita>5</quantita>
    </dettaglio>
  </dettagli>
  <links>
    <link rel="self" mediaType="application/xml"
          href="http://www.mionegozio.com/ordini/123" />
    <link rel="fattura" mediaType="application/xml"
          href="http://www.mionegozio.com/fatture/789" />
  </links>
</ordine>
```

# PROGETTAZIONE DI UN PROTOCOLLO



# DEPLOYMENT DESCRIPTOR



LOOSELY COUPLED

# SISTEMA DISTRIBUITO

---

Spiega quali sono le caratteristiche del seguente protocollo.

INTERNET STANDARD

Network Working Group  
Request for Comments: 867

J. Postel  
ISI  
May 1983

## Daytime Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Daytime Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a daytime service. A daytime service simply sends the current date and time as a character string without regard to the input.

### TCP Based Daytime Service

One daytime service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 13. Once a connection is established the current date and time is sent out the connection as a ascii character string (and any data received is thrown away). The service closes the connection after sending the quote.

### UDP Based Daytime Service

Another daytime service service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 13. When a datagram is received, an answering datagram is sent containing the current date and time as a ASCII character string (the data in the received datagram is ignored).

### Daytime Syntax

There is no specific syntax for the daytime. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. The daytime should be just one line.

One popular syntax is:

Weekday, Month Day, Year Time-Zone

Example:

Tuesday, February 22, 1982 17:37:43-PST

Another popular syntax is that used in SMTP:

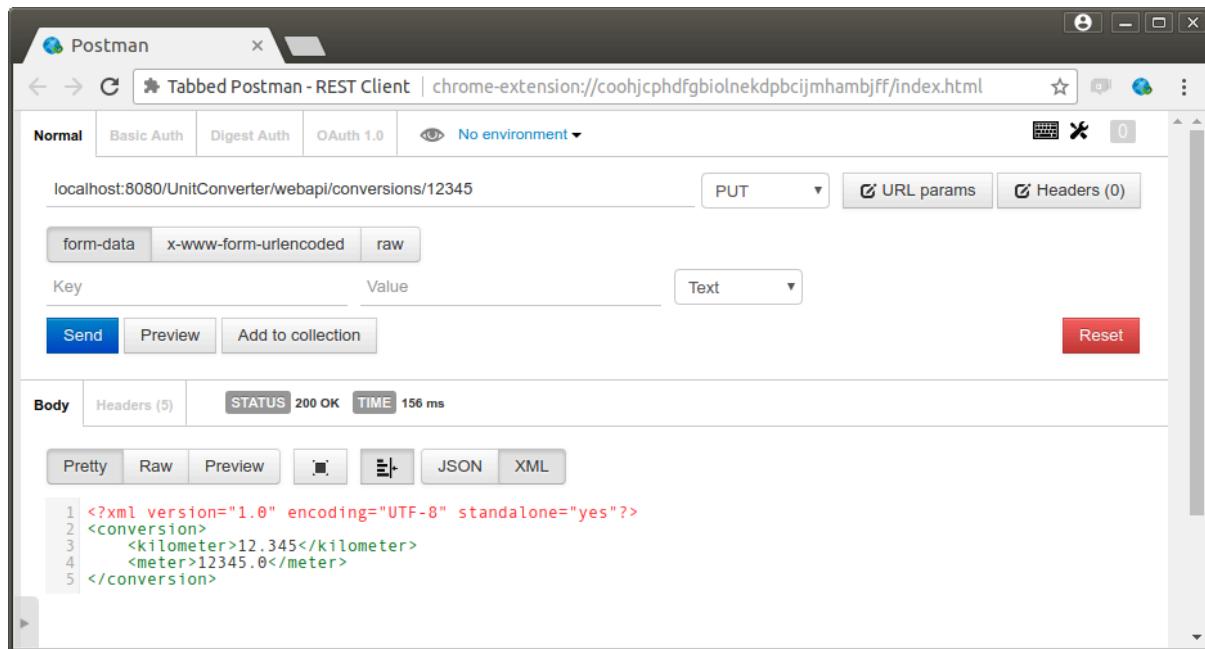
dd mmm yy hh:mm:ss zzz

Example:

02 FEB 82 07:59:01 PST

NOTE: For machine useful time use the Time Protocol (RFC-868).

---



---

GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr

HTTP/1.1 200 OK

Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

ETag: "51142bc1-7449-479b075b2891b"

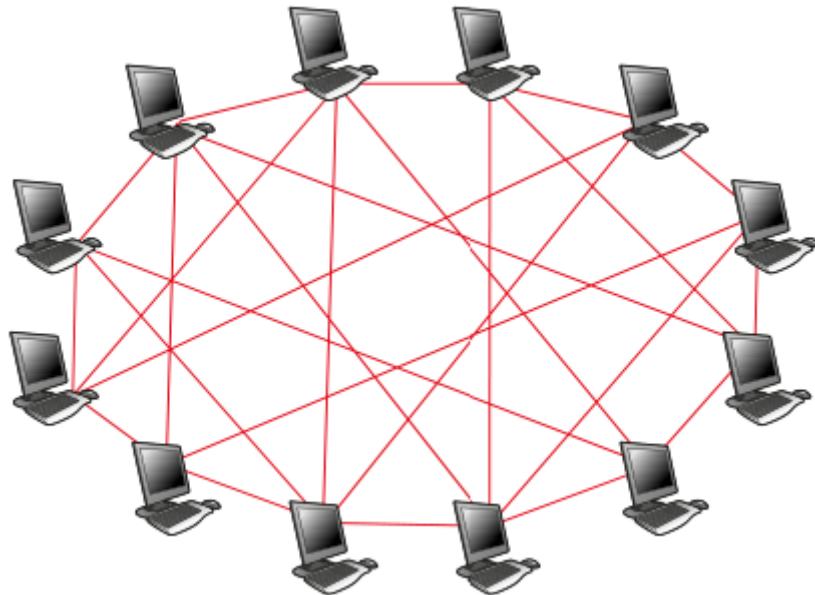
Accept-Ranges: bytes

Content-Length: 29769

Content-Type: text/html

<!DOCTYPE html>... (here come the 29769 bytes of the requested web page)

---



# TRASPARENZA

---

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

# INTEGRAZIONE

---

```
<studentsList>
  <student id="1">
    <firstName>Greg</firstName>
    <lastName>Dean</lastName>
    <certificate>True</certificate>
    <scores>
      <module1>70</module1>
      <module12>80</module12>
      <module3>90</module3>
    </scores>
  </student>
  <student id="2">
    <firstName>Wirt</firstName>
    <lastName>Wood</lastName>
    <certificate>True</certificate>
    <scores>
      <module1>80</module1>
      <module12>80.2</module12>
      <module3>80</module3>
    </scores>
  </student>
</studentsList>
```

---

Spiega quali sono le caratteristiche del seguente protocollo.

Network Working Group  
Request for Comments: 868

INTERNET STANDARD  
*Errata Exist*  
J. Postel - ISI  
K. Harrenstien - SRI  
May 1983

## Time Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Time Protocol are expected to adopt and implement this standard.

This protocol provides a site-independent, machine readable date and time. The Time service sends back to the originating source the time in seconds since midnight on January first 1900.

One motivation arises from the fact that not all systems have a date/time clock, and all are subject to occasional human or machine

error. The use of time-servers makes it possible to quickly confirm or correct a system's idea of the time, by making a brief poll of several independent sites on the network.

This protocol may be used either above the Transmission Control Protocol (TCP) or above the User Datagram Protocol (UDP).

When used via TCP the time service works as follows:

S: Listen on port 37 (45 octal).

U: Connect to port 37.

S: Send the time as a 32 bit binary number.

U: Receive the time.

U: Close the connection.

S: Close the connection.

The server listens for a connection on port 37. When the connection is established, the server returns a 32-bit time value and closes the connection. If the server is unable to determine the time at its site, it should either refuse the connection or close it without sending anything.

When used via UDP the time service works as follows:

S: Listen on port 37 (45 octal).

U: Send an empty datagram to port 37.

S: Receive the empty datagram.

S: Send a datagram containing the time as a 32 bit binary number.

U: Receive the time datagram.

The server listens for a datagram on port 37. When a datagram arrives, the server returns a datagram containing the 32-bit time value. If the server is unable to determine the time at its site, it should discard the arriving datagram and make no reply.

### The Time

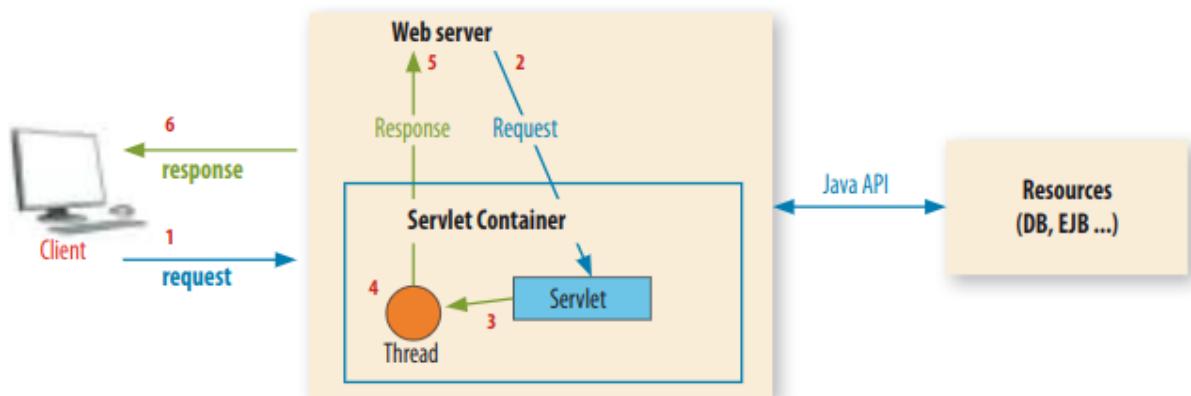
The time is the number of seconds since 00:00 (midnight) 1 January 1900 GMT, such that the time 1 is 12:00:01 am on 1 January 1900 GMT; this base will serve until the year 2036.

For example:

the time 2,208,988,800 corresponds to 00:00 1 Jan 1970 GMT,

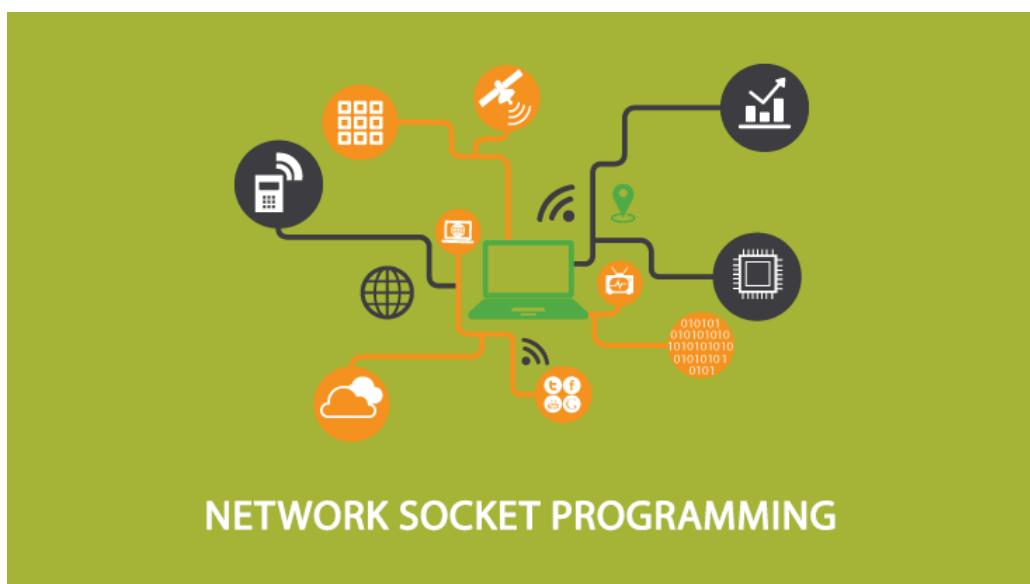
2,398,291,200 corresponds to 00:00 1 Jan 1976 GMT,  
2,524,521,600 corresponds to 00:00 1 Jan 1980 GMT,  
2,629,584,000 corresponds to 00:00 1 May 1983 GMT,  
and -1,297,728,000 corresponds to 00:00 17 Nov 1858 GMT.

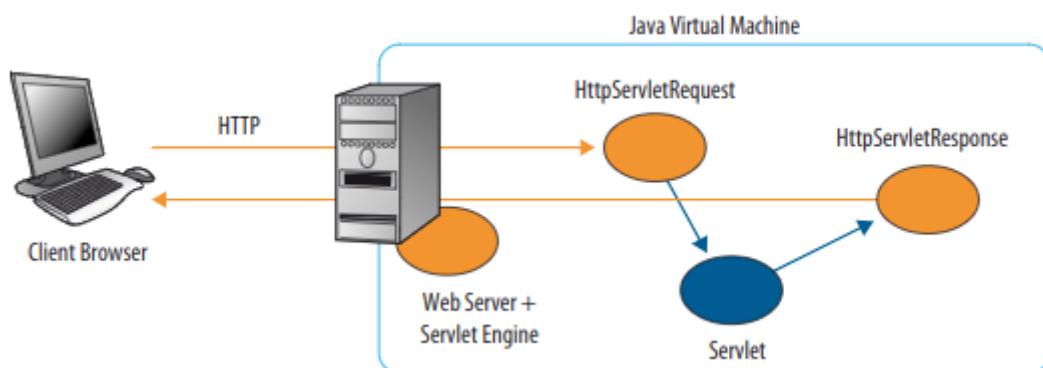
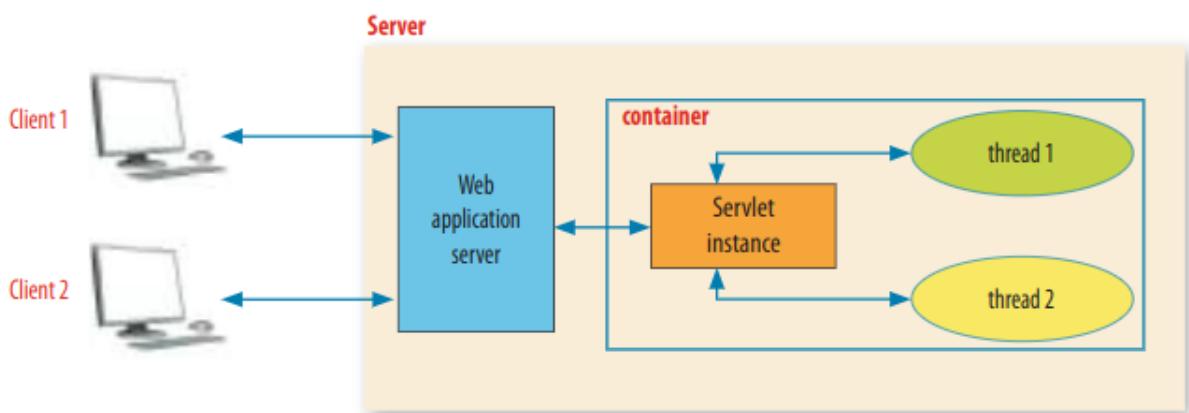
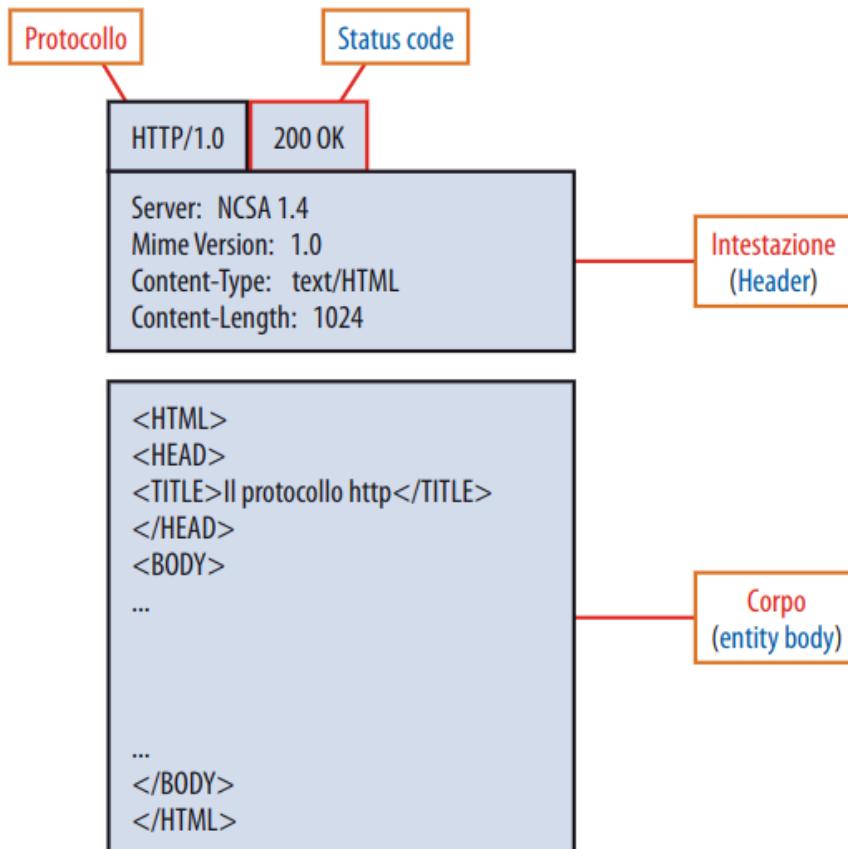
---



## init(), service(), destroy()

---





The screenshot shows the Postman REST Client interface. At the top, it says "Tabbed Postman - REST Client | chrome-extension://coohjcphdfgbiolnekdpbcijmhambjff/index.html". Below that, there are tabs for "Normal", "Basic Auth", "Digest Auth", "OAuth 1.0", and "No environment". The main area shows a GET request to "http://localhost:8080/CountryPopulation/webapi/countries". The response status is "200 OK" and the time taken is "15 ms". The response body is displayed in XML format:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <countries>
3   <country>
4     <id>1</id>
5     <countryName>India</countryName>
6     <population>10000</population>
7   </country>
8   <country>
9     <id>2</id>
10    <countryName>Bhutan</countryName>
11    <population>7000</population>
12  </country>
13  <country>
14    <id>3</id>
15    <countryName>Nepal</countryName>
16    <population>8000</population>
17  </country>
18  <country>
19    <id>4</id>
20    <countryName>China</countryName>
21    <population>20000</population>
22  </country>
23 </countries>
```

# SICUREZZA

# SISTEMA LEGACY

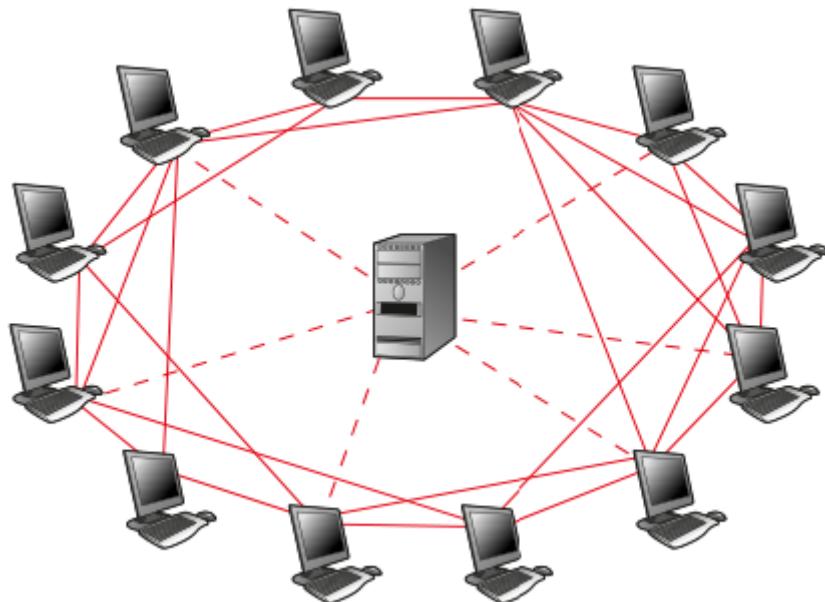
A terminali  
remoti

Client-  
Server

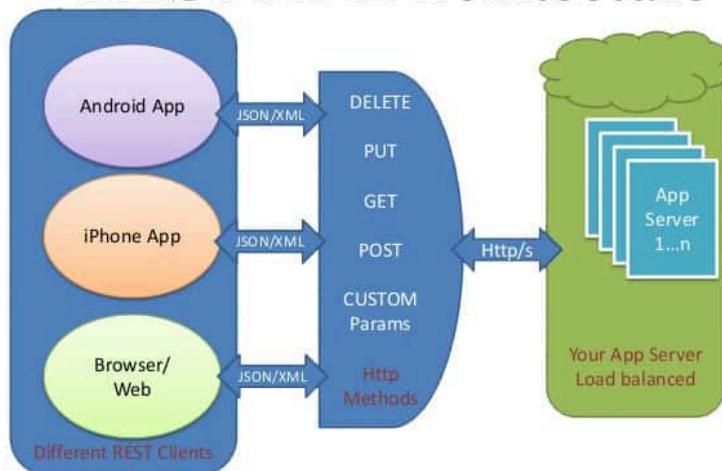
Web-centric

Cooperativa

Completamente  
distribuita



## REST API Architecture

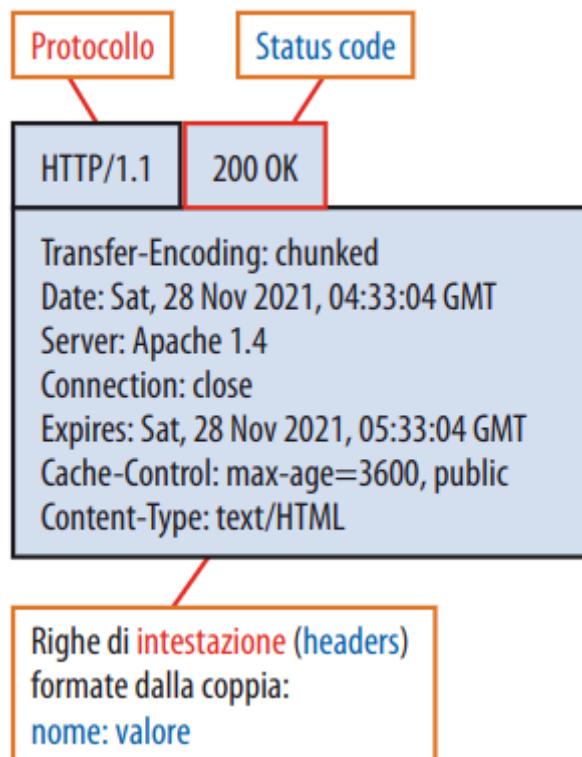
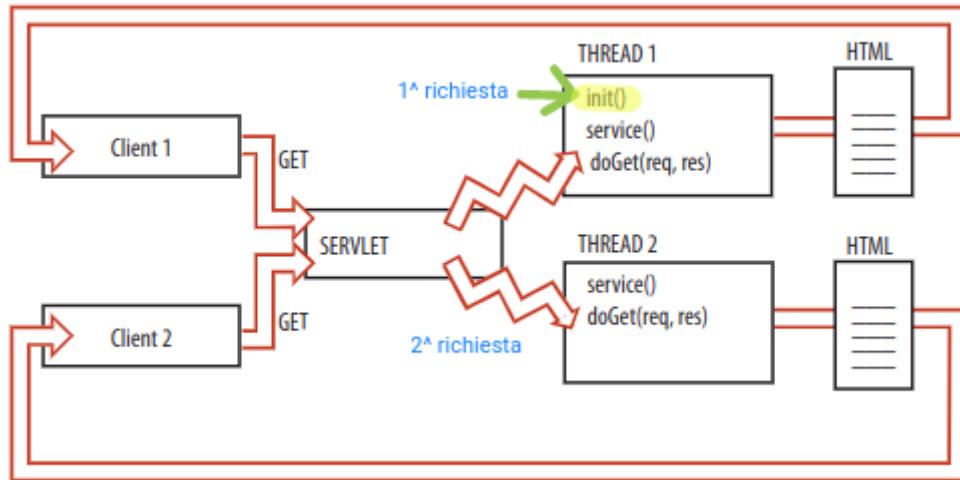


	DATI SINGOLI	DATI MULTIPLI
Istruzioni singole	SISD	SIMD
Istruzioni multiple	MISD	MIMD

Cluster computing

LAN - PAN

Domotica



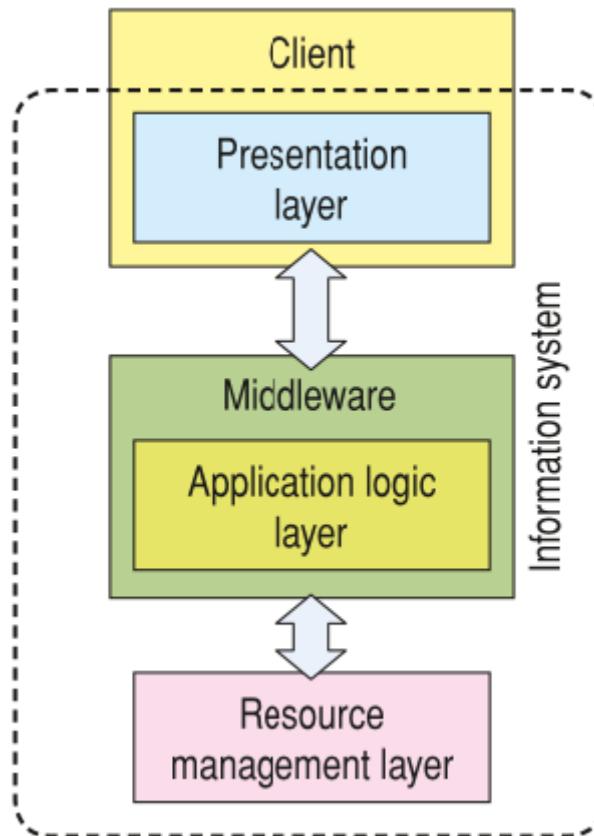
Ogni società per azioni quotata nel mercato borsistico elettronico NASDAQ viene identificata da una stringa di massimo 4 caratteri alfanumerici (esempi: GOOG = Google, AAPL = Apple, MSFT = Microsoft, INTC = Intel, CSCO = Cisco, IBM = IBM, HPQ = Hewlett-Packard,

NVDA = NVIDIA, RHT = Red Hat, KO = Coca-cola, MCD = McDonald's, SBUX = Starbucks, ...).

Il protocollo comunicativo che permette ad un client di richiedere al server del NASDAQ le informazioni sui titoli è basato sul protocollo di trasporto TCP. Un client può chiedere l'attuale quotazione di una società specificata tramite la stringa identificativa e la modifica dell'attuale quotazione in borsa di una società specificata sempre dalla stringa identificativa.

Descrivi le caratteristiche del protocollo.

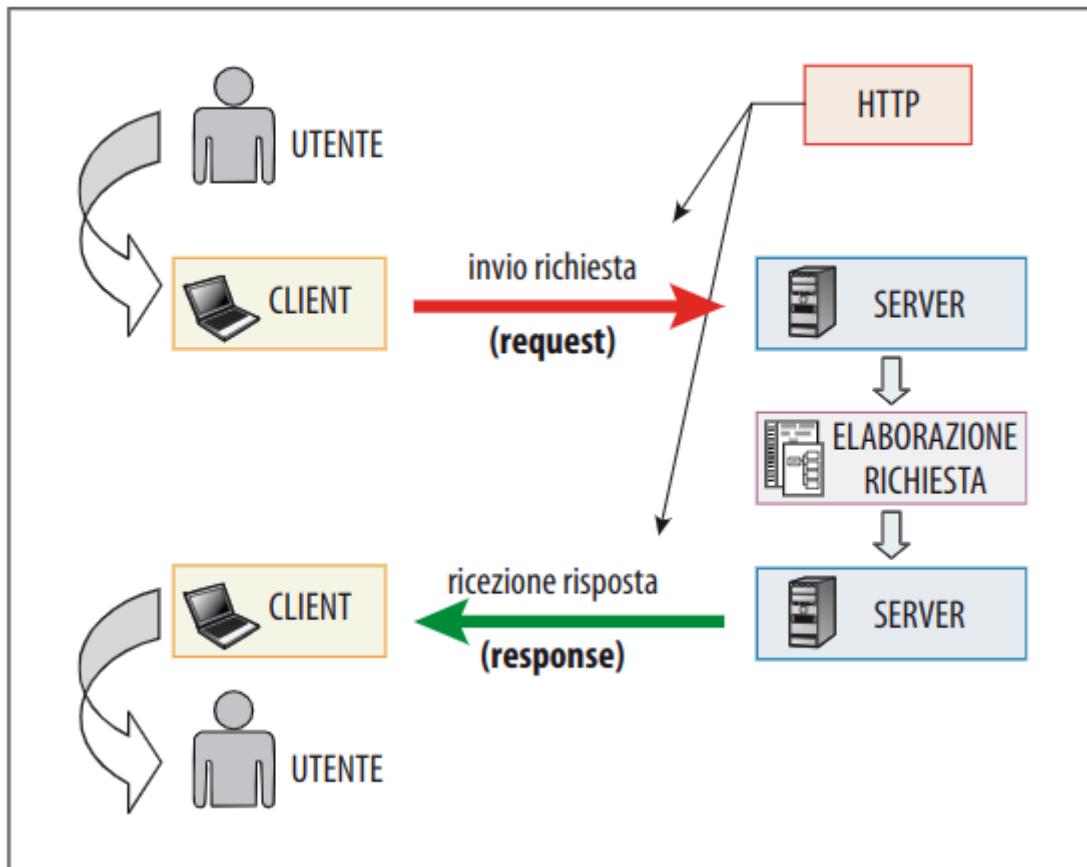
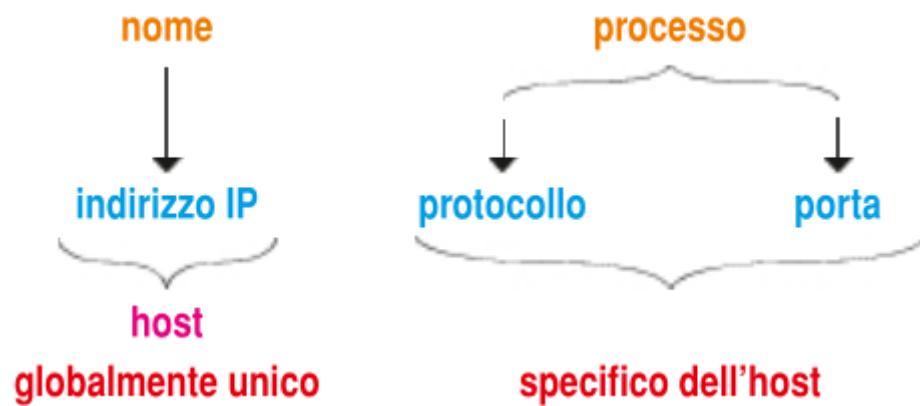
---



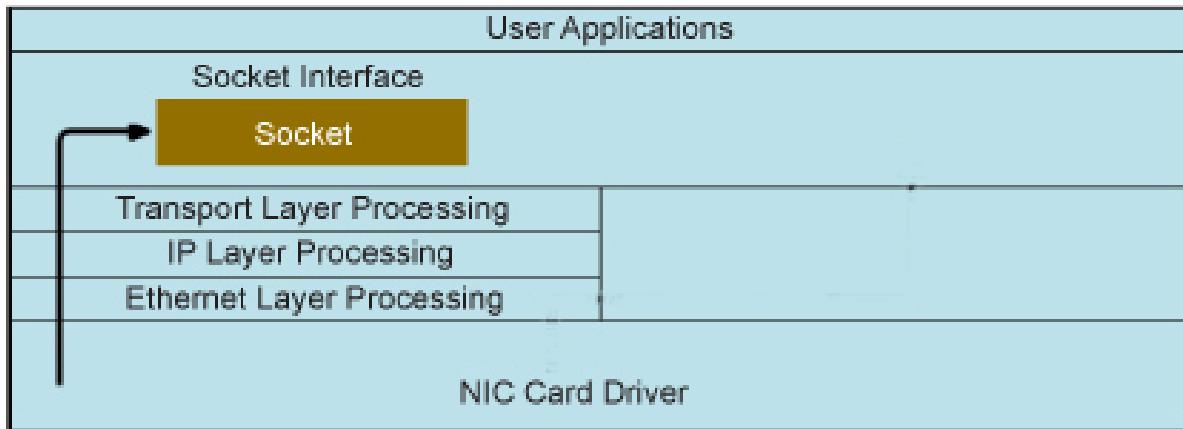
---

# STATELESS

---



# ANDROID

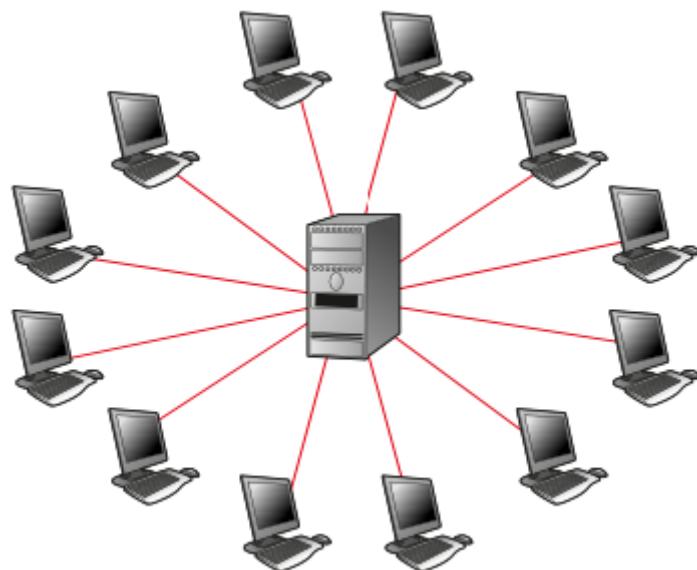


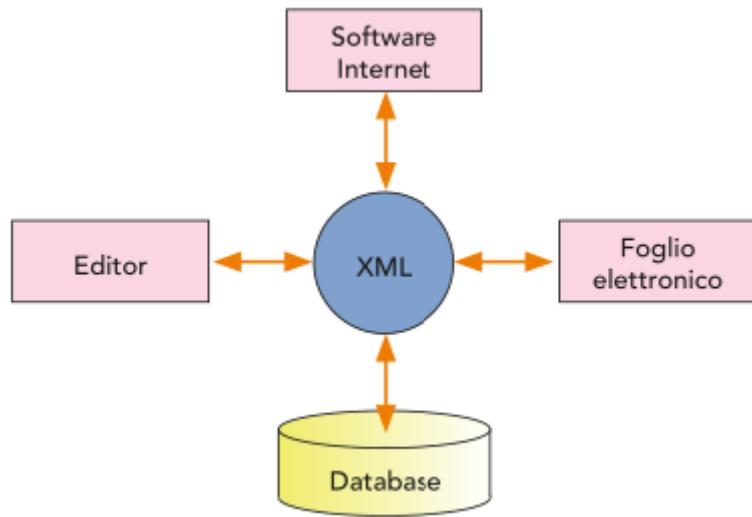
**HTTP/1.1 200 OK**

Date: Sun, 08 Feb xxxx 01:11:12 GMT  
Server: Apache/1.3.29 (Win32)  
Last-Modified: Sat, 07 Feb xxxx  
ETag: "0-23-4024c3a5"  
Accept-Ranges: bytes  
Content-Length: 35  
Connection: close  
Content-Type: text/html

<h1>My Home page</h1>

→ Status Line  
→ Response Headers  
→ A blank line separates header & body  
→ Response Message Body



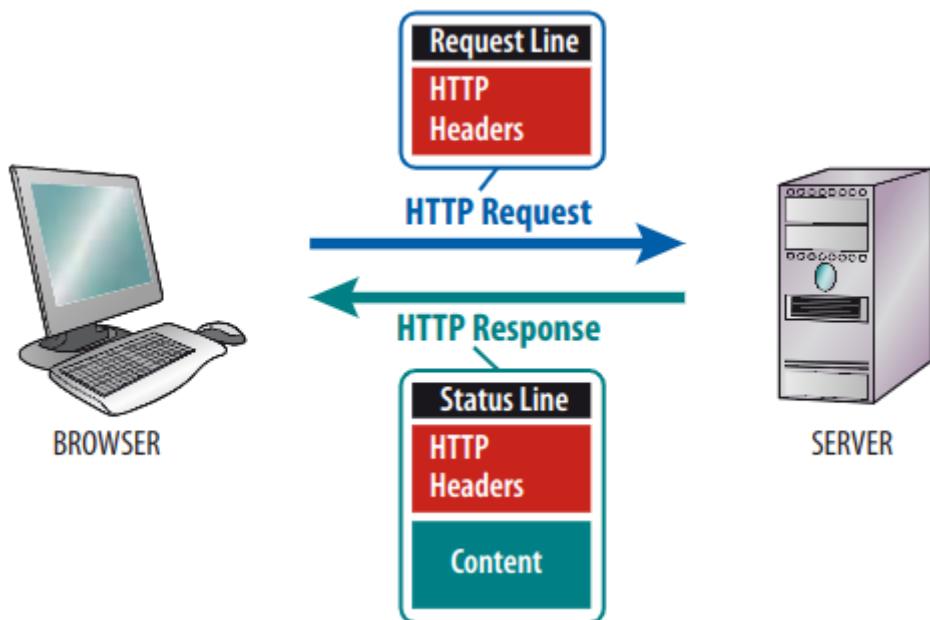


---

# SOCKET

---

Protocollo (TCP, UDP...)	Indirizzo IP (locale)	Indirizzo IP (remoto)
	Porta (locale)	Porta (remota)



---

Considerando i seguenti messaggi per la gestione dei titoli in borsa, spiega quali caratteristiche protocollari sono presenti.

## Comandi del server

I comandi descritti di seguito sono quelli inviati dal server al client (server → client).

### TITLE LIST

TITLE LIST§<Dato1>[#<Dato2>#<Dato3>[ . . . ]]

Il comando invia la lista dei titoli azionari.

Esempio: TITLE LIST§Adidas#First Capital#Unicredit

### VALUE

VALUES§<n>

Il comando invia la quotazione attuale del titolo azionario scelto.

Esempio: VALUE§134

## Comandi del client

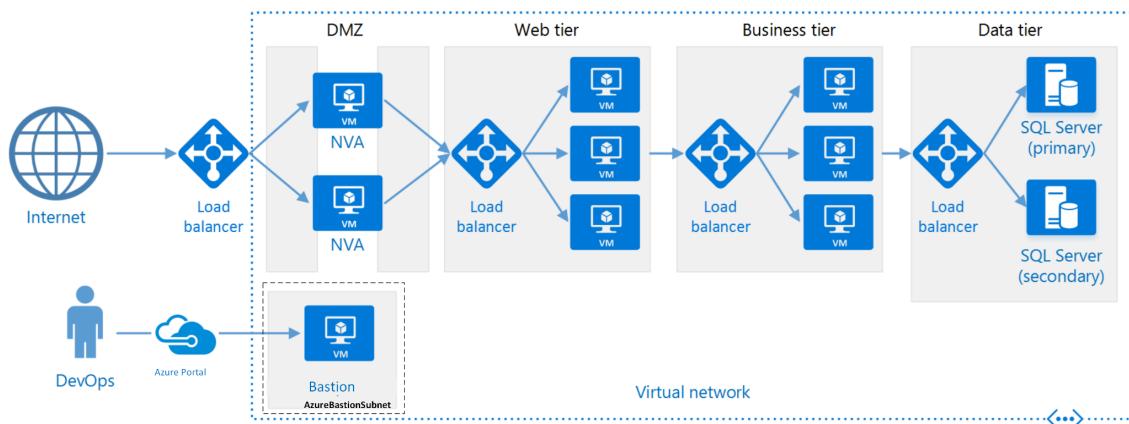
I comandi descritti di seguito sono quelli inviati dal client al server (client → server)

### TITLE

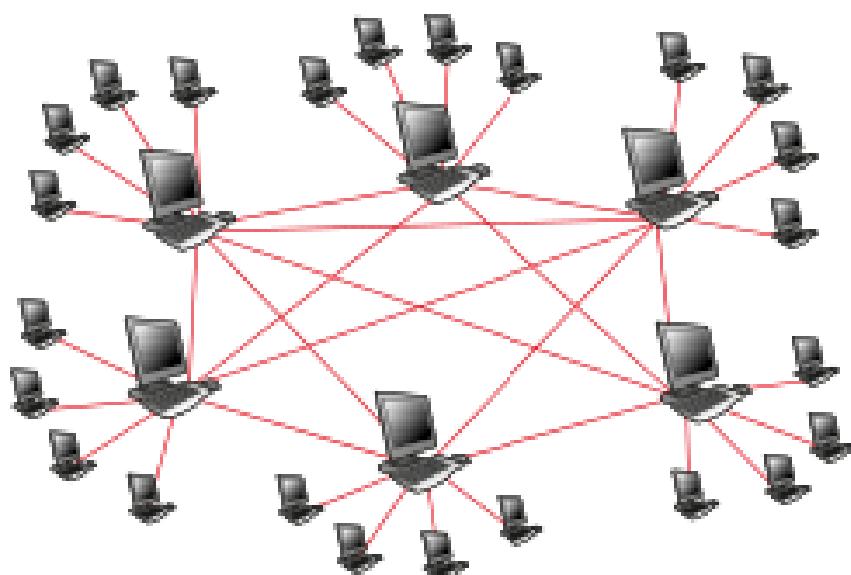
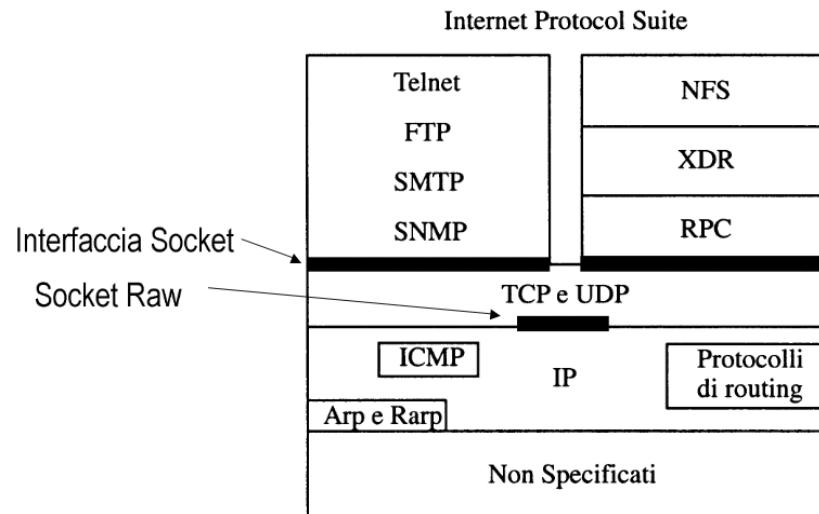
TITLE§<Titolo n>

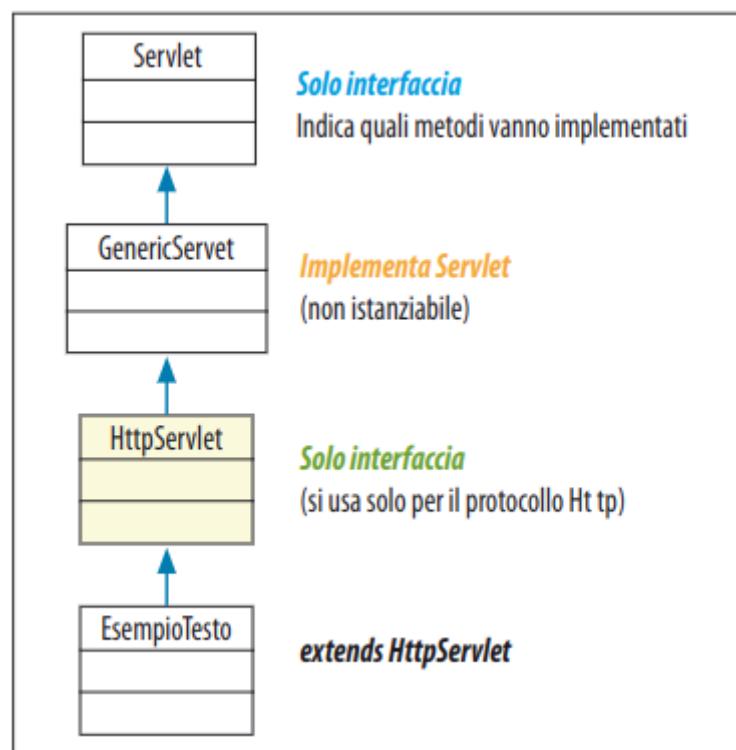
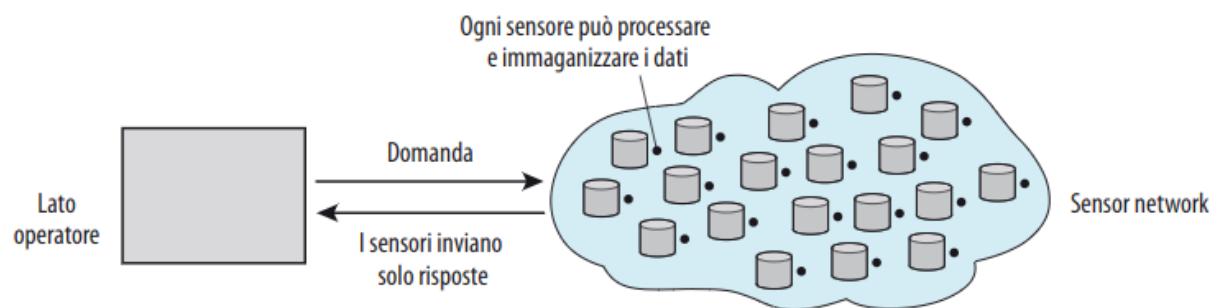
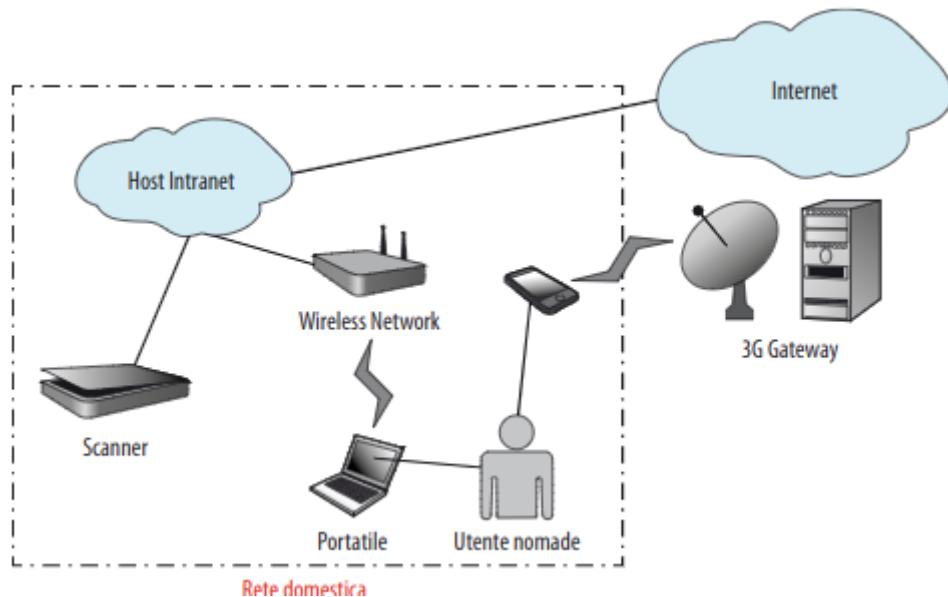
Il comando indica il titolo di cui si vogliono monitorare le quotazioni.

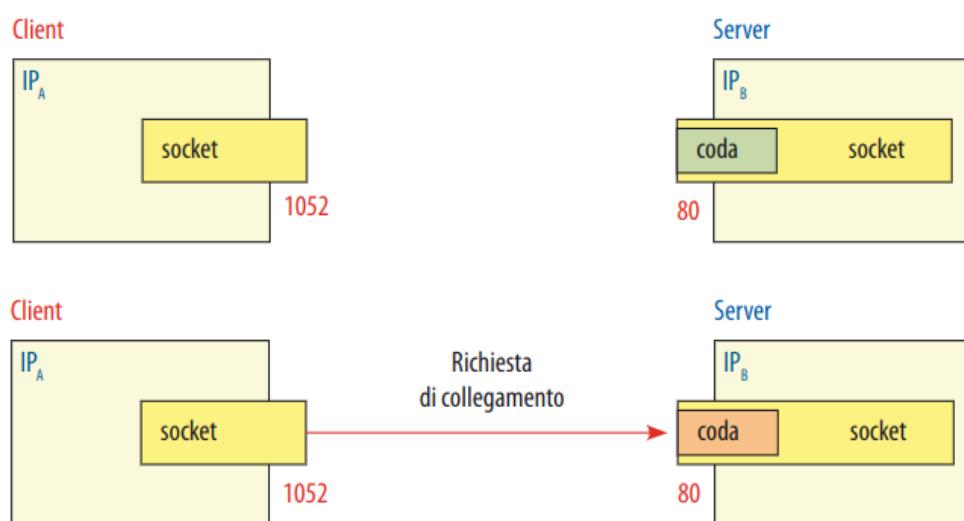
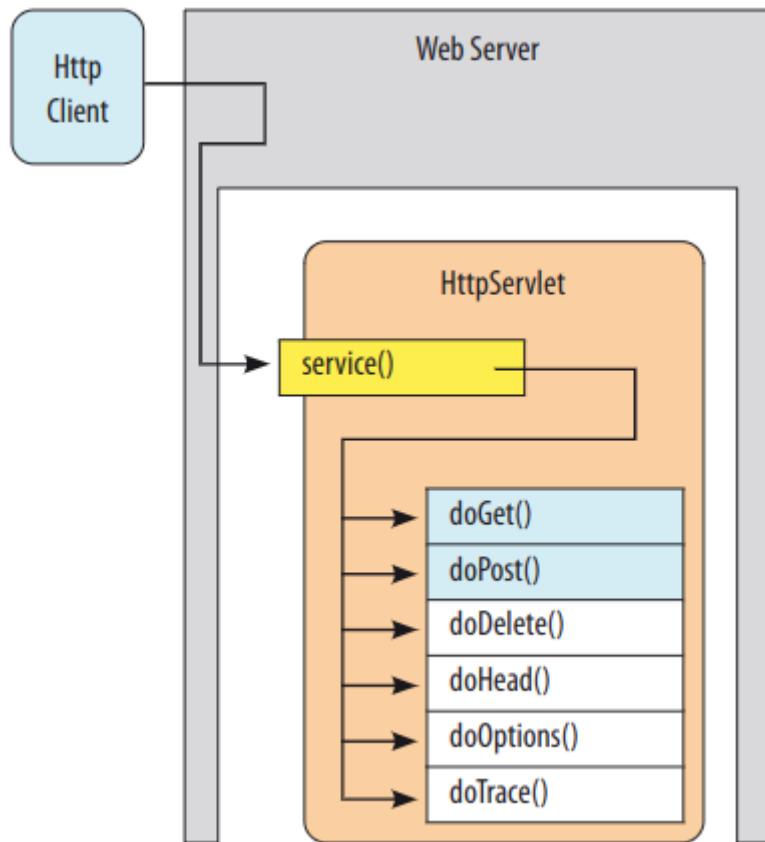
Esempio: TITLE§First Capital

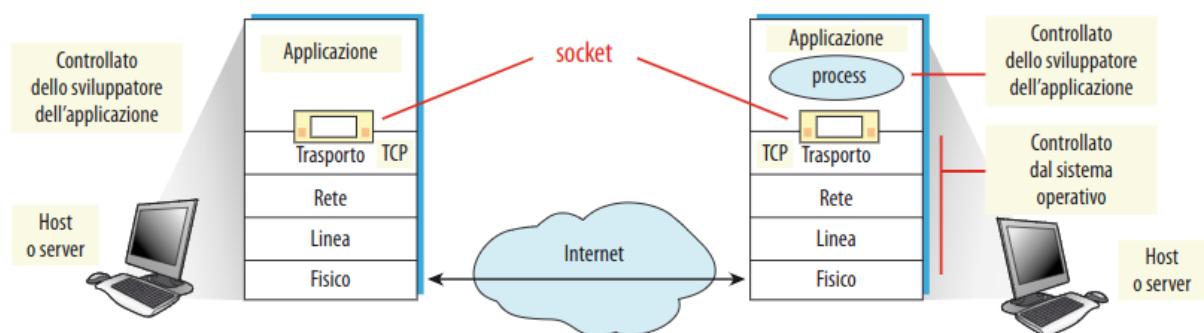
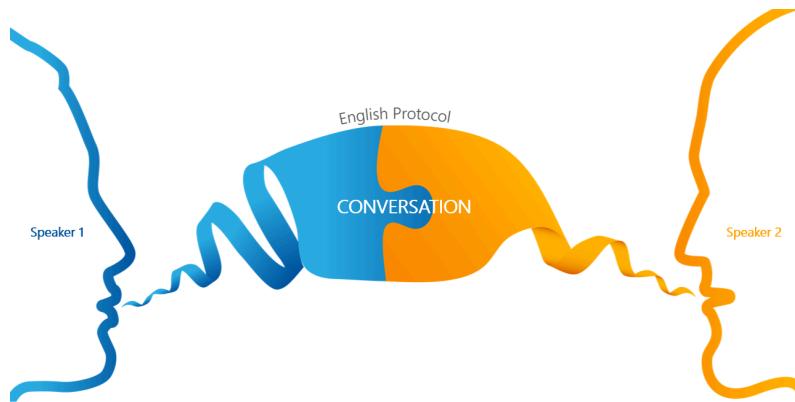
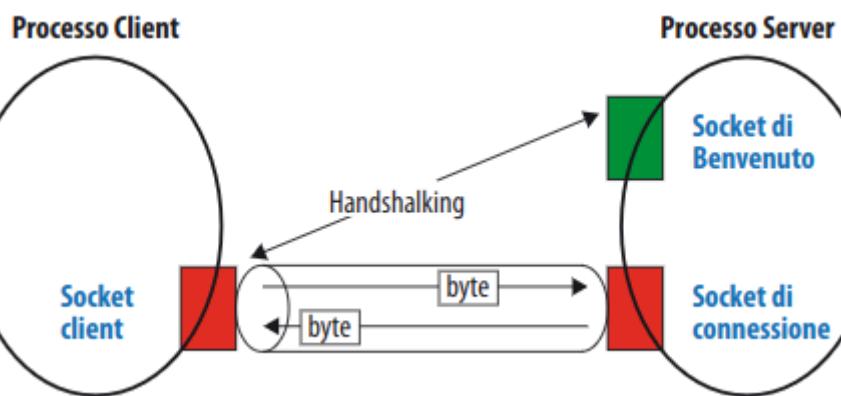
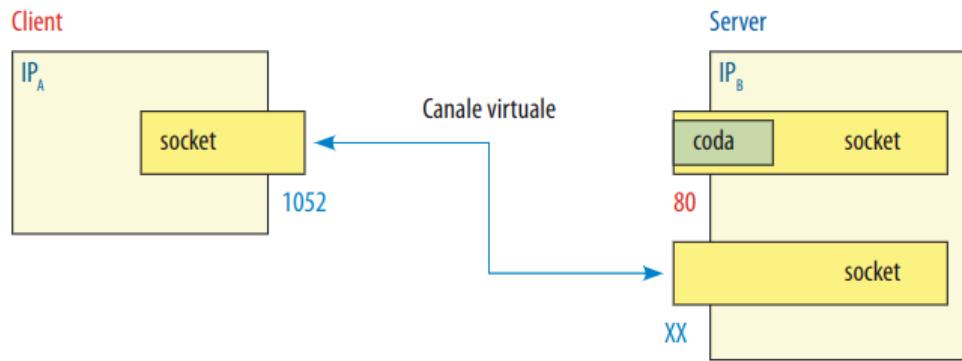


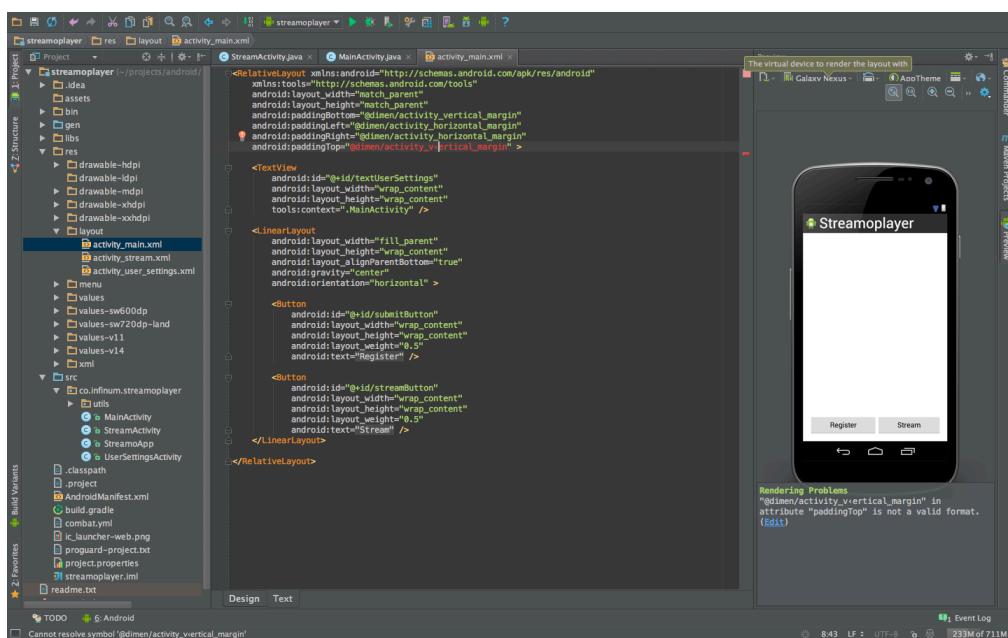
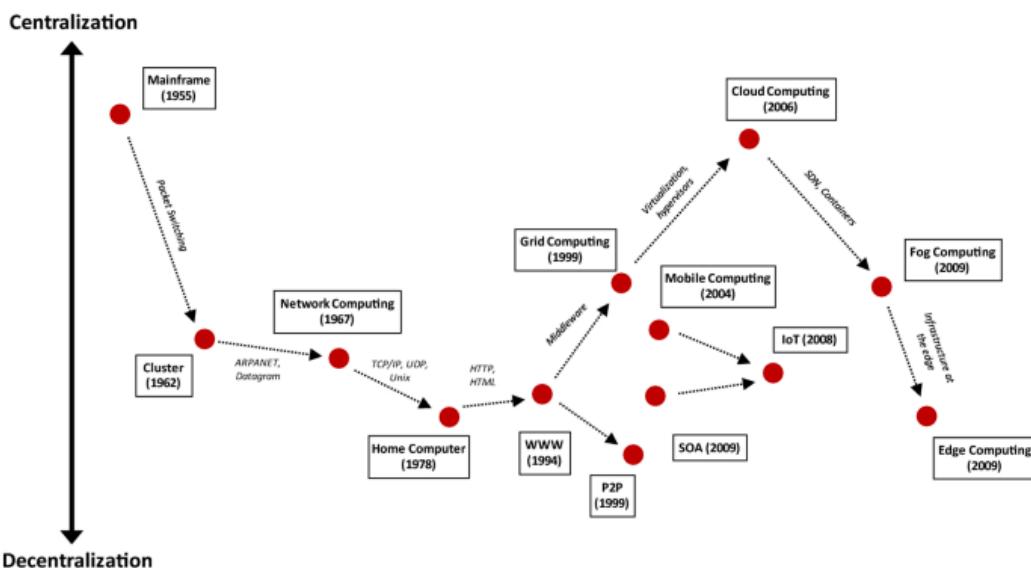
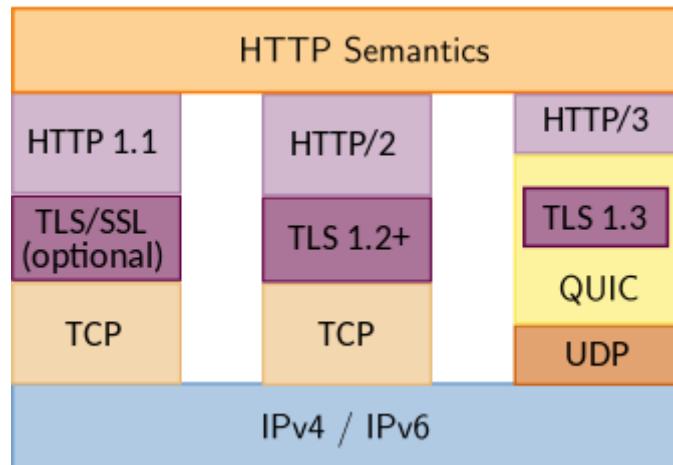
# SERVLET

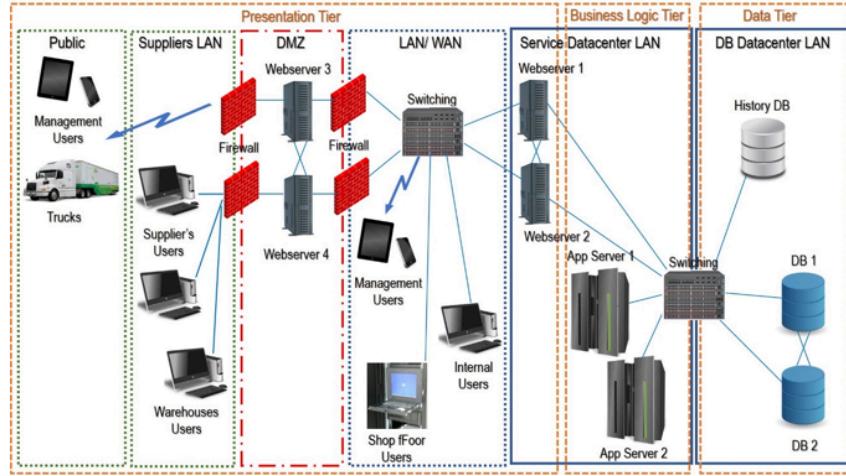




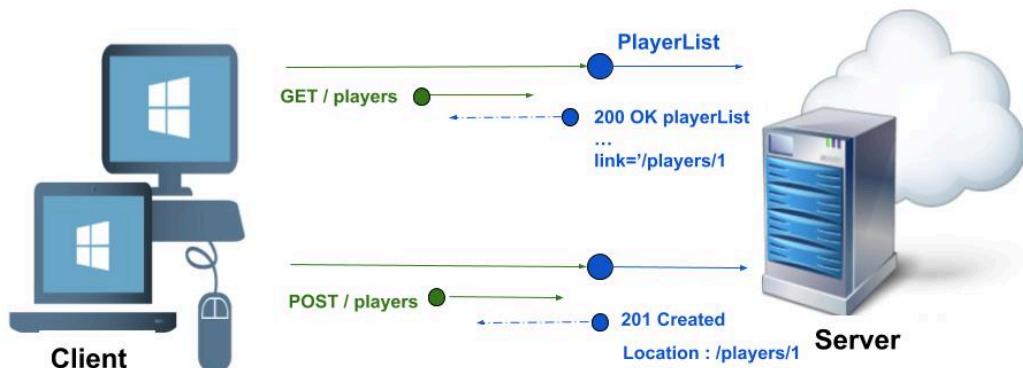


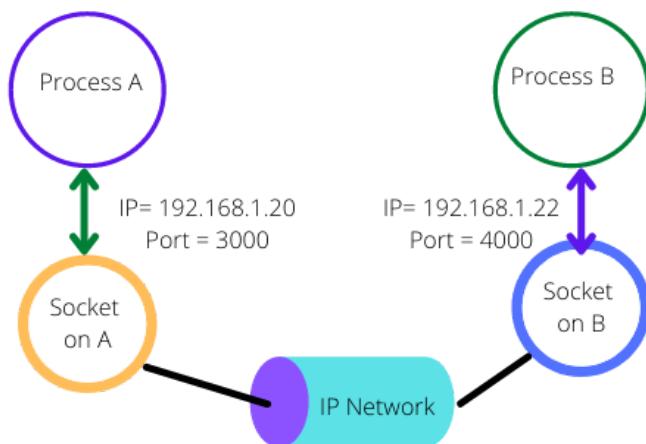
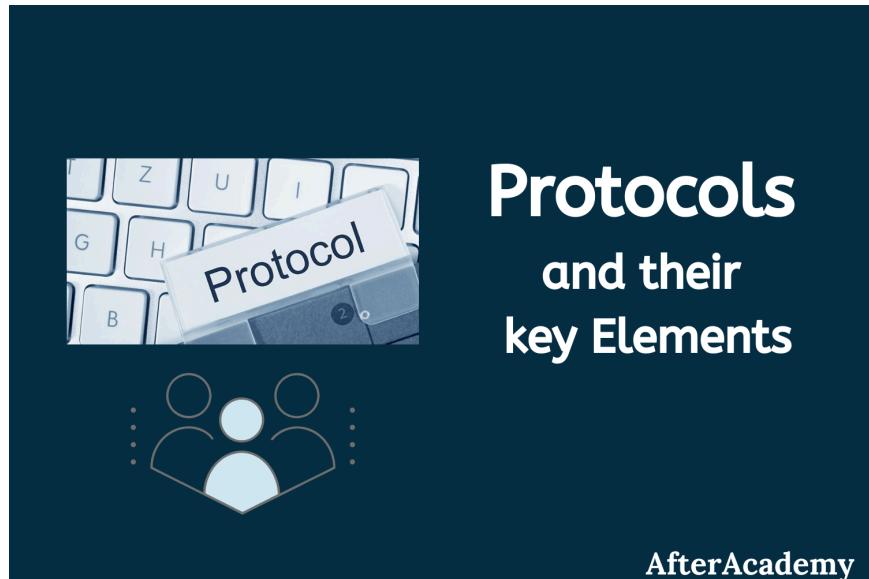






```
1  {
2    "string": "Hi",
3    "number": 2.5,
4    "boolean": true,
5    "null": null,
6    "object": { "name": "Kyle", "age": 24 },
7    "array": [ "Hello", 5, false, null, { "key": "value", "number": 6 } ],
8    "arrayOfObjects": [
9      { "name": "Jerry", "age": 28 },
10     { "name": "Sally", "age": 26 }
11   ]
12 }
13
```





# Approfondimenti

# A - Thread e concorrenza in Java - Propedeutica

# I Thread<sup>34</sup>

La **programmazione concorrente** permette di scrivere programmi che sono in grado di eseguire più operazioni in parallelo e in modo indipendente, permettendo all'utente di eseguire comunque molteplici operazioni, senza dover attendere la conclusione di un'operazione onerosa e potenzialmente bloccante, come ad esempio la scrittura o la lettura su disco. In un modello di programmazione tradizionale, non concorrente, dove può essere svolta una sola operazione alla volta, l'esecuzione della lettura o della scrittura di un file potrebbe causare un blocco del programma finché la stessa non sia stata completata, causando all'utente il disagio di avere l'applicazione in stallo e inutilizzabile per altri compiti. In un modello di **programmazione concorrente**, invece, l'operazione di lettura o di scrittura del file è eseguita da un thread separato in modo indipendente e non bloccante per l'applicazione nel suo complesso, permettendo all'utente di svolgere altre operazioni come quella, per esempio, di continuare a editare un altro file.

L'uso dei thread è inoltre particolarmente importante nella **comunicazione client-server** in quanto un server deve essere in grado di offrire il servizio a più client contemporaneamente per rendere accettabili i tempi di risposta. Per fare ciò un server avvia un thread per ogni richiesta da parte dei client; il thread sarà relativo alla porzione di codice che implementa il servizio richiesto. L'uso dei thread permette al server di mantenere sempre libera la porta di ascolto, spostando il servizio offerto ad un client su una porta diversa. Se non venisse utilizzato questa tecnica il server sarebbe in grado di soddisfare un solo client alla volta, bloccando di fatto la porta di ascolto per l'offerta del servizio fino a quando non sarà terminato la richiesta in corso.

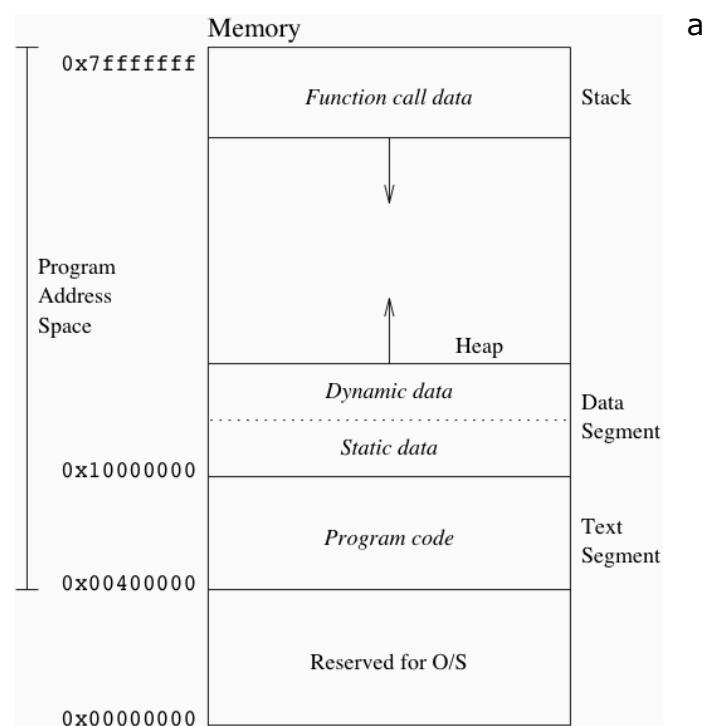
## CLIL Reading - Concurrency

Read the Oracle' guide "[Concurrency](#)", it is simple and complete guide about concurrency programming.

## Processi e Thread

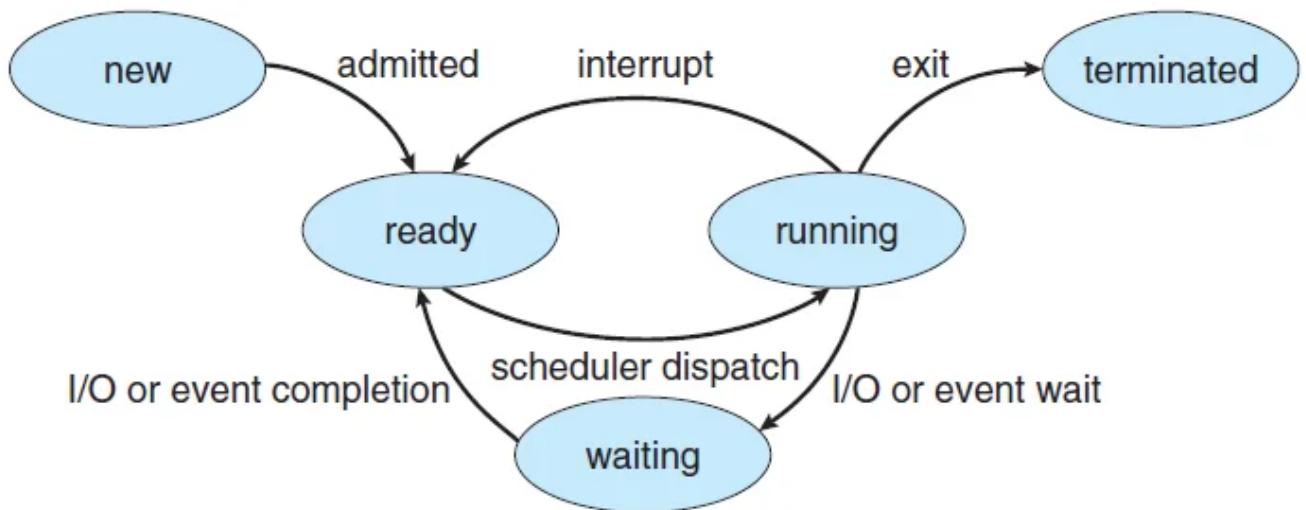
Un **processo** è definibile come un programma in esecuzione, indipendente dagli altri processi, costituito dal proprio spazio di memoria, dal proprio codice eseguibile e da riferimenti a eventuali risorse di sistema allocate per esso.

Il **ciclo di vita di un processo** prevede che il processo stesso passi attraverso diversi stati che dipendono sia dalle politiche di schedulazione del sistema operativo per l'accesso al processore, sia



<sup>34</sup> Tratto dal libro di P. Principe, *Java 8*, ed. Apogeo, marzo 2014

dagli input e dagli output previsti dal processo.



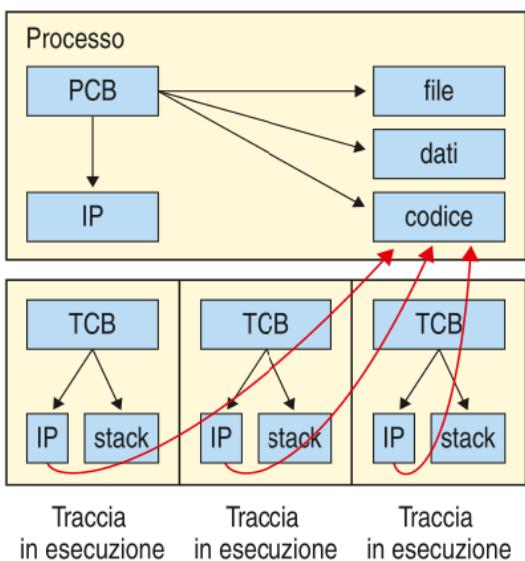
Il significato degli stati è il seguente:

- **new**: il processo viene creato;
- **running**: il processo è in esecuzione;
- **waiting** : il processo è in attesa di un dato evento;
- **ready**: il processo è pronto per essere eseguito;
- **terminated**: il processo ha completato la sua esecuzione.

All'interno del sistema operativo processo viene rappresentato attraverso una struttura dati denominata **Process Control Block (PCB)** che contiene tutte informazioni relative al processo.

Un **thread** è invece definibile come un'unità di elaborazione in cui può essere diviso un processo, in

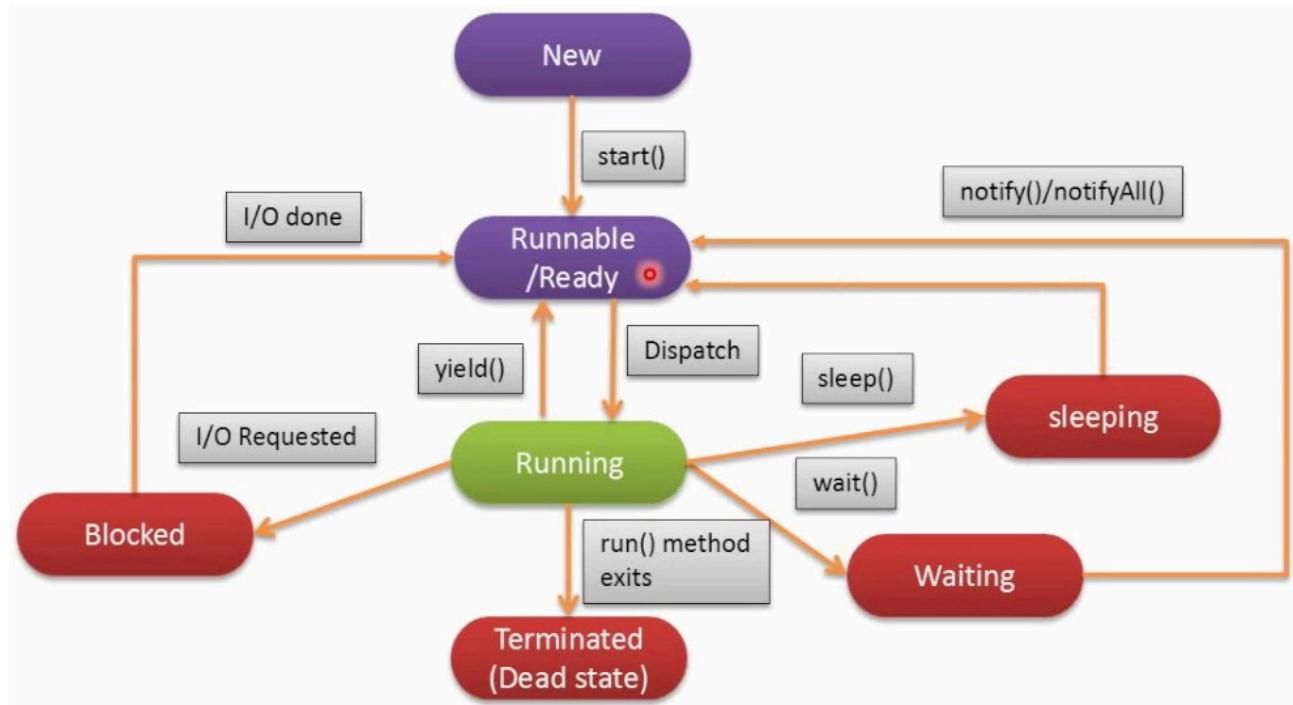
<table border="1"> <tr><td>Nome</td><td>PID</td></tr> <tr><td>Stato</td><td></td></tr> <tr><td>Program counter</td><td></td></tr> <tr><td>Altri registri</td><td></td></tr> <tr><td>Priorità</td><td></td></tr> <tr><td>Limiti di memoria</td><td></td></tr> <tr><td>File aperti</td><td></td></tr> <tr><td>...</td><td></td></tr> </table>	Nome	PID	Stato		Program counter		Altri registri		Priorità		Limiti di memoria		File aperti		...		<p>Contesto del processo</p>
Nome	PID																
Stato																	
Program counter																	
Altri registri																	
Priorità																	
Limiti di memoria																	
File aperti																	
...																	



pratica **una porzione di codice del processo** stesso che può essere invocata parallelamente ad altre porzioni di codice dello stesso processo, anch'essi definiti come thread. Esso vive, pertanto, all'interno di un processo dove condivide, con altri thread eventualmente presenti, le risorse, la memoria e le informazioni di stato del processo medesimo.

Un processo ha sempre un thread in esecuzione rappresentato da se stesso (*main thread*), ma può avere anche altri thread creati al suo interno che vengono eseguiti in parallelo.

Come il processo, anche i **thread** hanno un **Thread Control Block** che ne definisce lo stato, e un **ciclo di vita** che dipende dalle politiche di schedulazione e dalle interazioni richieste dal thread stesso.



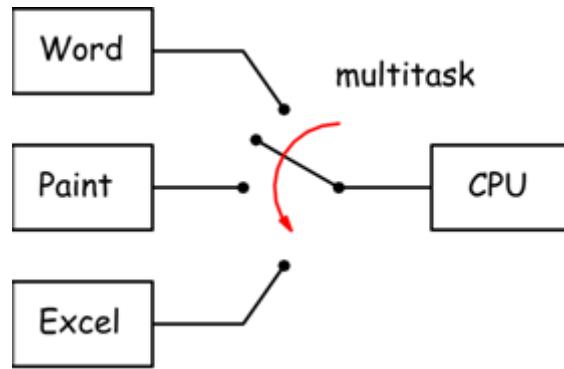
Il significato degli stati è il seguente:

- **new**: creato ma non ancora avviato;
- **ready**: pronto per l'esecuzione ma in attesa della CPU;
- **running**: in esecuzione;
- **waiting**: in attesa di un evento;
- **sleeping**: sospeso per un certo periodo;
- **blocked**: in attesa di completare un'operazione di input/output;
- **terminated**: terminato.

Quando il sistema operativo decide a quale thread **assegnare il tempo di CPU** per la sua esecuzione lo fa in base a un **sistema di priorità**, dove il thread con la più alta priorità è, ovviamente, preferito rispetto a uno con priorità più bassa. Se i thread hanno uguale priorità, il sistema operativo assegnerà circolarmente, per ciascuno, un'uguale quantità di tempo CPU (*time slice*) entro la quale i thread dovranno eseguire i propri processi computazionali (*round-robin scheduling*). Tuttavia, se un thread non avrà ancora terminato il suo processo computazionale quando il suo *time slice* sarà cessato, il sistema operativo gli toglierà la CPU e lo assegnerà al successivo thread di pari priorità, se presente, o ad un thread con priorità minore diversamente. Questo meccanismo si ripeterà finché tutti i thread avranno terminato la propria elaborazione.

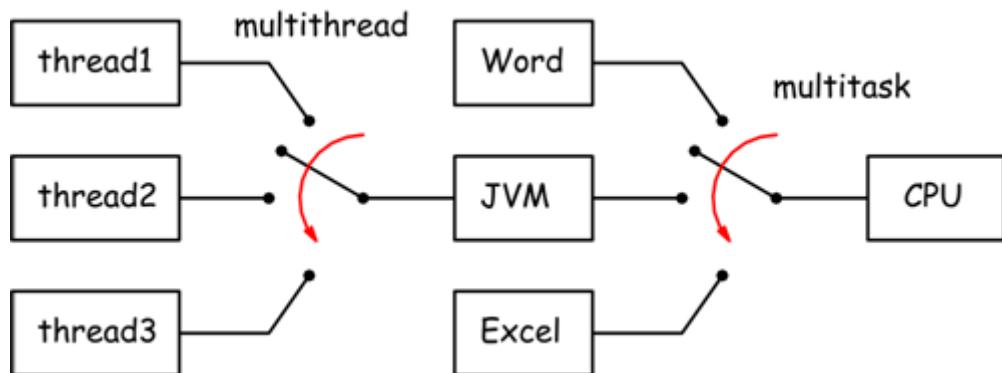
I processi e i thread, prima ancora di essere supportati nei linguaggi di programmazione, devono essere implementati a livello di sistema operativo. Oggi tutti i moderni sistemi operativi (Windows, GNU/Linux, MacOS, ecc.) sono **multitasking** e **multithreading**, ovvero consentono l'esecuzione di più task o processi (*multitasking*) e l'esecuzione di più thread (*multithreading*) contemporaneamente.

Nei sistemi a singolo processore il parallelismo è implementato virtualmente da complessi algoritmi software che garantiscono sia il cambio di contesto esecutivo, sia la ripartizione del tempo CPU tra processi (*scheduling*). In effetti, nei sistemi a singolo processore questo parallelismo è virtuale perché, se si volesse realizzare un vero *multitasking*, si dovrebbe avere una CPU dedicata ad ogni processo. Il parallelismo con macchine monoprocessoare è simulabile grazie alla velocità delle moderne CPU.



Con la tecnica del multitasking i programmi competono per l'uso delle risorse ed in particolare per la CPU. I vari programmi, vengono fatti funzionare in 'time-sharing' (condivisione di tempo) cioè il tempo di CPU suddiviso tra i vari processi per piccole quantità di tempo, e viene fatto così rapidamente che per la percezione umana questo processo assume una caratteristica di quasi simultaneità.

Anche la Java Virtual Machine consente di eseguire più attività contemporaneamente lavorando in multithreading; in questo caso lo scheduling dei thread è a carico della JVM, che a sua volta viene schedulata dal sistema operativo per essere eseguita nella CPU.



# CLIL Reading - Java Concurrency / Multithreading Basics

Read the following tutorial, it explains what concurrency is and why it is so useful in modern programming. It also gives hints about Multiprocessing, Multitasking, and Multithreading. The tutorial clarifies the difference between Process and Thread, and summarizes the common problems associated with concurrency.

[Java Concurrency / Multithreading Basics](#)

## Definizione di thread in Java

In linguaggio Java un **thread** è rappresentato da un **oggetto istanza della classe Thread**, mentre l'**operazione che il thread eseguirà**, costituita da una porzione di codice del processo, è **rappresentata dalla definizione di un apposito metodo denominato run()**.

La definizione del metodo **run()**, che rappresenta l'operazione che sarà eseguita dal thread, può essere effettuata utilizzando due modalità implementative:

- attraverso la **definizione di una classe che estende la classe Thread**;
- attraverso la **definizione di una classe che implementa l'interfaccia Runnable**.

Estendendo la classe **Thread** l'utilizzo risulterà più semplice, ma si avrà lo svantaggio che la classe implementata, ereditando la classe **Thread**, non potrà estendere nessun'altra classe, dato che il linguaggio Java non permette l'ereditarietà multipla.

Implementando invece l'interfaccia **Runnable** si avrà una maggiore flessibilità, sia per la netta separazione tra l'oggetto che compie l'azione e l'azione stessa, sia per la possibilità di poter utilizzare l'ereditarietà sulla classe che implementa l'interfaccia **Runnable**.

Nel seguente esempio sull'uso dei thread questi saranno creati sia estendendo la classe **Thread**, sia implementando l'interfaccia **Runnable**.

Il **main thread** avviato con l'esecuzione del metodo **main()** del programma, si occuperà di istanziare i due thread, di mandarli in esecuzione invocando il metodo **start()** e di segnalare la propria terminazione.

### SimpleThread.java

```
public class SimpleThread {  
  
    public static void main(String[] args) {  
        // Creazione del primo thread dalla classe che implementa l'interfaccia  
        // Runnable: si istanzia l'oggetto e poi lo si usa per creare un thread  
        // istanziando un oggetto Thread  
        DoJobRunnable job = new DoJobRunnable();  
        // Il secondo parametro (facoltativo) assegna il nome al thread  
        Thread thread1 = new Thread(job, "***THREAD 1***");  
        // Avvio del thread
```

```
thread1.start();
// Creazione del secondo thread dalla classe che estende la classe
// Thread: si istanzia l'oggetto direttamente usando la classe che estende
// la classe Thread
DoJobThread thread2 = new DoJobThread();
// viene assegnato un nome al thread
thread2.setName("***THREAD 2***");
// avvio del thread
thread2.start();

// Visualizzazione fatta dal main thread
System.out.println("Thread principale (" +
                    Thread.currentThread().getName() + ") finito");
}
```

}

Nella classe **DoJobRunnable**, che implementa l'interfaccia **Runnable**, il thread verrà posto ciclicamente nello stato **SLEEP** per 6 secondi, per 5 cicli consecutivi.

### **DoJobRunnable.java**

```
/**
 * La classe implementa l'interfaccia Runnable
 */
class DoJobRunnable implements Runnable {

    // Il metodo run() contiene la porzione di codice che sarà eseguita dal
    // thread
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                int ms = 6000; // 6 secondi
                // Visualizzazione del nome del thread corrente e del tempo di
                // attesa
                System.out.println(Thread.currentThread().getName()
                        + " sarà occupato per i prossimi " + ms
                        + " millisecondi");
                // il thread è posto in sleep per 6000 millisecondi
                Thread.sleep(ms);
            } catch (InterruptedException ex) {
                System.out.println("Thread uscito dallo sleep per interruzione");
            }
        }
    }
}
```

Nella classe **DoJobThread**, che estende la classe **Thread**, il thread verrà posto ciclicamente nello stato **SLEEP** per un numero casuale di secondi minore o uguale a 3, per 5 cicli consecutivi.

**DoJobThread.java**

```
import java.util.Random;

/**
 * La classe estende la classe Thread
 */
public class DoJobThread extends Thread {

    // Il metodo run() contiene la porzione di codice che sarà eseguita dal
    // thread
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                // Creazione di un oggetto generatore di numeri casuali
                Random r = new Random();
                // Generazione di un numero casuale tra 0 e 3000 che sarà usato
                // come numero di millisecondi di attesa
                int ms = r.nextInt(3000);
                // Visualizzazione del nome del thread corrente e del tempo di
                // attesa
                System.out.println(Thread.currentThread().getName()
                    + " sarà occupato per i prossimi " + ms
                    + " millisecondi");
                // Il thread è posto in sleep per il valore random di
                // millisecondi precedentemente selezionato
                Thread.sleep(ms);
            } catch (InterruptedException ex) {
                System.out.println("Thread uscito dallo sleep per interruzione");
            }
        }
    }
}
```

Si osservi come l'esecuzione dei tre thread risulti indipendente, e come il **main thread** possa terminare anche prima dei due thread che ha avviato.

```
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
Thread principale (main) finito
***THREAD 2*** sarà occupato per i prossimi 1365 millisecondi
***THREAD 2*** sarà occupato per i prossimi 874 millisecondi
***THREAD 2*** sarà occupato per i prossimi 117 millisecondi
***THREAD 2*** sarà occupato per i prossimi 474 millisecondi
***THREAD 2*** sarà occupato per i prossimi 2753 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
```

## Alcuni metodi della classe Thread

- **public long getId()** ritorna il numero dell'identificativo assegnato al thread. Questo numero è univoco e rimane assegnato per tutto il ciclo di vita del thread. Quando il thread sarà terminato tale identificativo potrà essere riutilizzato.
- **public final String getName()** ritorna il nome associato al thread.
- **public final int getPriority()** ritorna il valore di priorità associato al thread.
- **public Thread.State getState()** ritorna lo stato in cui si trova il thread, che può essere **NEW, RUNNABLE, BLOCKED, WAITING, TIMED\_WAITING** e **TERMINATED**.
- **public void interrupt()** interrompe il thread.
- **public final boolean isAlive()** ritorna **true** se il thread è ancora in esecuzione.
- **public final void join() throws InterruptedException** pone in attesa il thread corrente finché non cessa di esistere il thread su cui è invocato il metodo **join()**.
- **public final void setPriority(int newPriority)** consente di impostare un valore di priorità per il thread mediante il parametro **newPriority**. Questo valore deve essere incluso tra i valori **Thread.MIN\_PRIORITY** che vale 1 e **Thread.MAX\_PRIORITY** che vale 10.
- **public static void yield()** consente di fornire allo scheduler l'indicazione, non vincolante, che il corrente thread vuole cedere spontaneamente il proprio tempo CPU ad altri thread.

Per comprendere almeno uno dei metodi elencati si può modificare la classe **SimpleThread** esposta precedentemente in modo tale che il main thread attenda la conclusione dei due thread avviati prima di concludere la sua esecuzione. Per fare ciò il main thread deve invocare il metodo **join()**, come esposto nell'esempio seguente.

### SimpleThread.java

```
public class SimpleThread {  
  
    public static void main(String[] args) throws InterruptedException {  
        // Creazione del primo thread dalla classe che implementa l'interfaccia  
        // Runnable: si istanzia l'oggetto e poi lo si usa per creare un thread  
        // Istanziando un oggetto Thread  
        DoJobRunnable job = new DoJobRunnable();  
        // Il secondo parametro (facoltativo) assegna il nome al thread  
        Thread thread1 = new Thread(job, "***THREAD 1***");  
        // Avvio del thread  
        thread1.start();  
  
        // Creazione del secondo thread dalla classe che estende la classe  
        // Thread: si istanzia l'oggetto usando la classe Thread  
        DoJobThread thread2 = new DoJobThread();
```

```
// Viene assegnato un nome al thread
thread2.setName("***THREAD 2***");
// Avvio del thread
thread2.start();

// Il main thread si pone in attesa che i due thread concludano la loro
// esecuzione prima di continuare la propria
thread1.join();
thread2.join();

// Visualizzazione fatta dal main thread
System.out.println("Thread principale (" +
                    Thread.currentThread().getName() + ") finito");
}

}
```

L'esecuzione si modificherà nel seguente modo.

```
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 2*** sarà occupato per i prossimi 2817 millisecondi
***THREAD 2*** sarà occupato per i prossimi 301 millisecondi
***THREAD 2*** sarà occupato per i prossimi 104 millisecondi
***THREAD 2*** sarà occupato per i prossimi 496 millisecondi
***THREAD 2*** sarà occupato per i prossimi 1061 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
***THREAD 1*** sarà occupato per i prossimi 6000 millisecondi
Thread principale (main) finito
```

## CLIL Reading - Java Thread and Runnable Tutorial

Read the following tutorial, it explains the two techniques can be used to create a thread, and a lot of other useful things, such as:

- how starting a thread;
- how temporarily stop a thread putting it in sleeping mode;
- how to force a thread to wait for the completion of the thread it called for.

It also gives you an inside about anonymous class and lambda expression, that are two efficient methods of programming. These are only hints of a different programming style.

[Java Thread and Runnable Tutorial](#)

# Esempi d'uso dei thread

## Nomi thread e priorità

Questo primo esempio estende la classe Thread, assegna un nome al thread, modifica la sua priorità e lo manda in esecuzione. Sia il main thread, sia il nuovo thread visualizzeranno il proprio nome, il main thread sarà sospeso per un secondo, mentre il nuovo thread visualizzerà la propria priorità.

### Main.java<sup>35</sup>

```
/* This simple project creates a thread extending the Thread class, executes the new
thread, prints the name of the main thread and that of the new thread, and changes
the new thread priority*/
public class Main1 {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new NewThread();
        // Set a new name for thread t
        t.setName("New Worker Thread");
        // Set a new priority for thread t
        t.setPriority(Thread.MIN_PRIORITY);
        System.out.println("We are in thread: " + Thread.currentThread());
        System.out.println("We are in thread: " +
            Thread.currentThread().getName()+
            " before starting a new thread");
        // Start thread t
        t.start();
        // The main thread is put in sleep mode
        Thread.sleep(1000);
        System.out.println("We are in thread: " +
            Thread.currentThread().getName()+
            " after starting a new thread");
    }
}
```

### NewThread.java

```
public class NewThread extends Thread {
    @Override
    public void run() {
        //Code that will run in a new thread
        System.out.println("We are in thread: " + Thread.currentThread());
        // Print the thread name
        System.out.println("We are now in thread " +
            Thread.currentThread().getName());
        // Print the thread priority
        System.out.println("Current thread priority is " +
            Thread.currentThread().getPriority());
```

---

<sup>35</sup> [Michael Pogrebinsky, Java Multithreading, Concurrency & Performance Optimization, Udemy](#)

```
    }  
}
```

## Settenani

In questo esempio il main thread avvierà sette thread passando ad ognuno di essi come parametro il nome di uno dei sette nani. I thread visualizzeranno il loro nome.

### SetteNani.java<sup>36</sup>

```
/*  
 * Progettare un programma che permetta di avviare 7 thread ai quali verrano  
 * assegnati i nomi dei sette nani tramite il costruttore NomiNani() e poi  
 * verranno avviati.  
 * La schedulazione non prevedibile produrrà risultati diversi ad ogni  
 * riavvio.  
 */  
  
public class SetteNani {  
  
    public static void main(String[] args) {  
  
        String[] nomi = new String[7];  
        nomi[0] = "Dotto";  
        nomi[1] = "Brontolo";  
        nomi[2] = "Pisolo";  
        nomi[3] = "Mammolo";  
        nomi[4] = "Gongolo";  
        nomi[5] = "Eolo";  
        nomi[6] = "Cucciolo";  
  
        System.out.println(Thread.currentThread().getName());  
        //System.out.println(this.getName()); NON VA BENE  
  
        for(int i = 0; i < 7; i++) {  
            Thread thread = new NomiNani(nomi[i]);  
            thread.start();  
        }  
    }  
}
```

---

<sup>36</sup> P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 2, Hoepli, 2020

## NomiNani.java

```
/*
 * Nella classe NomiNani si è creato un costruttore che richiama quello della
 * classe Thread per poi aggiungere l'impostazione del nome del thread passato
 * come parametro al costruttore.
 * Viene inoltre ridefinito il metodo run() della classe Thread che si occuperà
 * di visualizzare il nome del thread.
 */
public class NomiNani extends Thread {
    /* Costruttore */
    public NomiNani(String nome) {
        super();
        this.setName(nome);
    }

    @Override
    public void run() {
        /* Visualizzo il nome del nuovo thread */
        System.out.println(this.getName());
        //System.out.println(Thread.currentThread().getName());
    }
}
```

## Runnable e Thread

In questo esempio la classe **DinDonDan** avvierà tre thread di tipo **Campana** che permetterà di ripetere per un numero prefissato di volte il suono della campana passatogli come argomento.

### DinDonDan.java<sup>37</sup>

```
/**
 * Definire una classe Dindondan che permetta di avviare tre thread che
 * facciano riferimento alla classe Camapana. Quest'ultima implementerà
 * l'interfaccia Runnable ed avrà un costruttore che prenderà come argomenti
 * il suono e il numero di volte che dovrà essere ripetuto.
 * L'output dipenderà dalla schedulazione dei processi e quindi non risulterà
 * deterministico.
 */

public class DinDonDan {

    public static void main(String[] args) {
        // Prima modalità di definizione
        Runnable camp1 = new Campana("din", 5);
        Thread thr1 = new Thread(camp1);
        thr1.start();
```

<sup>37</sup> P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 2, Hoepli, 2020

```
// Seconda modalità di definizione
Thread thr2 = new Thread(new Campana("don", 5));
thr2.start();

// Terza modalità di definizione
new Thread(new Campana("dan", 5)).start();

}

}
```

## Campana.java

```
/** 
 * La classe Campana implementa l'interfaccia Runnable.
 * Prevede un costruttore a cui verranno passati come parametri il suono della
 * campana ed il numero di volte che deve essere ripetuto.
 * In questo caso non abbiamo potuto usare il metodo getName() perché gli
 * oggetti della classe campana non appartengono più alla classe Thread; la
 * classe Campana implementa solo l'interfaccia Runnable che ha un unico
 * metodo run().
 */
public class Campana implements Runnable {
    String suono;
    int volte;

    public Campana(String suono, int volte) {
        this.suono = suono;
        this.volte = volte;
    }

    public Campana(int volte) {
        this.suono = "deng";
        this.volte = volte;
    }

    @Override
    public void run() {
        for(int i = 0; i < volte; i++)
            System.out.println((i + 1) + suono + " ");
    }
}
```

# Terminazione di un'applicazione java - Shutdown Hook

In questo esempio vedremo come terminare regolarmente un'applicazione Java priva di interfaccia grafica. Per farlo sarà aggiunto all'applicazione uno **Shutdown Hook**.

L'applicazione è pensata come un servizio che continuerà ad essere eseguita fino a quando non sarà terminata esplicitamente, oppure usando il comando kill (in ambiente Linux) tramite il sistema operativo. In entrambi i casi sarà attivato lo **Shutdown Hook**.

L'applicazione implementa un contatore che si incrementa ogni secondo fino a quando non raggiungerà il valore 1000000. Raggiunto il valore sarà modificato un flag che imporrà la terminazione dell'applicazione.

Generalmente le applicazioni prive di interfaccia grafica sono dei servizi che vengono eseguiti in background. Quando un'applicazione viene terminata usando un comando kill viene inviato un segnale SIGTERM al processo Java e il segnale viene intercettato dalla classe Runtime (java.lang) che è la classe che lega il processo al sistema operativo e che avvierà la procedura di shutdown.

Uno **Shutdown Hook** è un oggetto Thread e la procedura di shutdown deve essere implementata nel metodo run(). L'oggetto Thread deve essere passato all'oggetto Runtime usando il metodo addShutdownHook(Thread t), e quando il segnale SIGTERM sarà ricevuto dalla classe Runtime, questa avvierà automaticamente il thread; non sarà quindi necessario avviare lo **Shutdown Hook** usando il metodo start().

## CounterShutdownHook.java<sup>38</sup>

```
public class CounterShutdownHook {  
    public static boolean running = true;  
  
    public static void main(String[] args) {  
        System.out.println("== CounterShutdownHook started ==");  
  
        Runtime.getRuntime().addShutdownHook(new MyShutdownHook());  
  
        int i = 0;  
        while (running) {  
            System.out.println("Counter: " + i);  
            i++;  
            try {  
                Thread.sleep(1);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
  
            if (i == 1000000) {  
                running = false;  
            }  
        }  
    }  
}
```

<sup>38</sup> L'esempio è preso da [Shutdown hook - How to regularly stop Java application](#) di Matjaž Cerkvenik

```
        }
    }

    System.out.println("==> CounterShutdownHook ended ==>");
}
}
```

## MyShutdownHook.java

```
public class MyShutdownHook extends Thread {

    @Override
    public void run() {
        System.out.println("==> MyShutdownHook activated ==>");
        // stop running threads
        // store data to DB
        // close connection to DB
        // disconnect...
        // release other resources...

        CounterShutdownHook.running = false;
    }
}
```

Per effettuare un kill dell'applicazione in ambiente Linux è possibile usare o il comando ps o il comando jps che è parte della JDK (non della JRE).

Ad esempio una esecuzione di ps fornisce i seguenti dati:

```
$ ps -ef | grep java
mgm      24907  21609  4 19:19 ?          00:00:00 ... CounterShutdownHook
$ kill 24907
```

Il comando kill provoca l'invio di un segnale SIGTERM catturato dall'oggetto Runtime che avvierà il thread di tipo MyShutdownHook per terminare l'applicazione.

Analogamente è possibile usare jsp per recuperare il PID del thread e procedere con la sua terminazione, come mostrato di seguito.

```
$ jps -ml
24907 CounterShutdownHook
$ kill 24907
```

L'uso di un kill di livello 9 (kill -9 <pid>) è invece invia un segnale SIGKILL che termina immediatamente l'applicazione non permettendo l'avvio dello **Shutdown Hook**.

Se il sistema operativo deve riavviarsi o spegnersi inizialmente invierà un segnale SIGTERM a tutte le applicazioni per permettergli di chiudersi correttamente, ma in caso di fallimento della chiusura, dopo un po' di tempo invierà un segnale SIGKILL che forza la chiusura delle applicazioni pendenti.

# Esercizi svolti sui thread

## Moltiplicazione tra matrici<sup>39</sup>

La moltiplicazione tra matrici è una delle operazioni base quando si lavora con esse, ed è un classico problema di programmazione concorrente e parallela.

La moltiplicazione tra la matrice  $A$  di  $m$  righe e  $n$  colonne e la matrice  $B$  di  $n$  righe e  $p$  colonne, produrrà una terza matrice  $C$  di  $m$  righe e  $p$  colonne.

In generale, al crescere del numero di righe e di colonne delle due matrici operandi della moltiplicazione, aumenterà in modo non trascurabile il tempo di esecuzione. Per questa ragione è opportuno sfruttare la programmazione concorrente e il parallelismo per limitare i tempi di esecuzione.

Di seguito saranno forniti una versione seriale della soluzione e tre versioni concorrenti, la prima che prevede un thread per ogni elemento della matrice risultante, la seconda che prevede un thread per ogni riga della matrice risultante e la terza che avvia tanti thread quanti sono i processori disponibili nella JVM.

Da questi esempi si potrà osservare sia il netto miglioramento nei tempi di esecuzione rispetto alla versione seriale con le ultime due versioni concorrenti della soluzione, sia l'effetto negativo in tempo di esecuzione di una errata scelta del livello di granularità della concorrenza con il primo esempio della versione concorrente, peggiore anche della versione seriale.

## Classi e metodi comuni

La classe **Main** crea due matrici di 2000 righe e 2000 colonne riempite con dei valori pseudo-casuali, istanziandole dalla classe **MatrixGenerator**. Viene anche creata la matrice risultato prendendo il numero di righe della prima matrice e il numero di colonne della seconda matrice. Prima e dopo l'operazione di moltiplicazione vengono calcolati i tempi di avvio e terminazione del prodotto matriciale.

Si osservi che le due matrici operandi sono dichiarate di tipo **MatrixGenerator**, ma dato che questa classe è astratta, l'istanziazione dell'oggetto dovrà essere fatta usando la specifica classe che, nelle diverse versioni, implementa la classe astratta **MatrixGenerator**.

### Main.java<sup>40</sup>

```
public class Main {  
    public static void main(String[] args) {  
        // First matrix  
        MatrixGenerator fstMatrix = new SerialMultiplier(2000, 2000);  
    }  
}
```

<sup>39</sup> Tratto dal libro di J. F. Gonzalez, *Mastering Concurrency Programming with Java 9- Fast, reactive and parallel application development*, 2nd edition, Packt, 2017, con diverse migliorie sul codice.

<sup>40</sup> Per la misurazione del tempo di esecuzione risulta più preciso l'uso di `System.nanoTime()` per il calcolo del tempo di start e end dell'esecuzione di interesse, trasformandolo eventualmente in millisecondi usando `TimeUnit.NANOSECONDS.toMillis(end - start)`;

```
// Second matrix
MatrixGenerator sndMatrix = new SerialMultiplier(2000, 2000);
// Matrix product start time
long startTime = System.currentTimeMillis();
// Matrix product
fstMatrix.multiply(sndMatrix);
// Matrix product end time
long endTime = System.currentTimeMillis();
// Display the time needed for the matrix product
long duration = endTime - startTime;
System.out.printf("Time: %d\n", duration);
}
}
```

La classe **MatrixGenerator** crea le due matrici che devono essere moltiplicate caricando in ognuna di esse dei valori pseudo-casuali. La classe è astratta in quanto il metodo atto alla moltiplicazione varierà nelle diverse versioni e sarà implementato opportunamente a seconda dei casi. Tutti gli altri metodi permettono di gestire adeguatamente la matrice.

## MatrixGenerator.java

```
import java.util.Random;

public abstract class MatrixGenerator {
    private int rows;           // number of rows
    private int columns;         // number of columns
    private double[][] theMatrix; // the matrix

    // Default constructor
    public MatrixGenerator() {}

    // Constructor
    public MatrixGenerator(int rows, int columns) {
        this.setRows(rows);
        this.setColumns(columns);
        this.theMatrix = new double[rows][columns];
        this.generate();
    }

    // Fills the matrix with random numbers
    private void generate() {
        Random random = new Random();
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.columns; j++) {
                theMatrix[i][j] = random.nextDouble() * 10;
            }
        }
    }

    // Gives the rows number
    public int getRows() {
        return rows;
    }

    // Sets the number of rows, if it is suitable
    public void setRows(int rows) {
        if (rows <= 0) {
```

```
        throw new IllegalArgumentException("Rows number incorrect");
    }
    this.rows = rows;
}

// Gives the columns number
public int getColumns() {
    return columns;
}

// Sets the number of columns, if it is suitable
public void setColumns(int columns) {
    if (columns <= 0) {
        throw new IllegalArgumentException("Columns number incorrect");
    }
    this.columns = columns;
}

// Gives the element in (row, column) position
public double getElement(int row, int column) {
    return theMatrix[row][column];
}

/**
 * Make the matrix product between this and the matrix
 * given as parameter.
 * @param sndMatrix - the second matrix used for matrix product.
 */
public abstract void multiply(MatrixGenerator sndMatrix);
}
```

## Versione seriale

Questa versione non prevede l'uso di altri thread all'infuori del main thread. La moltiplicazione matriciale viene effettuata semplicemente tramite dei cicli.

### SerialMultiplier.java

```
public class SerialMultiplier extends MatrixGenerator {

    public SerialMultiplier(int rows, int columns) {
        super(rows, columns);
    }

    @Override
    public void multiply(MatrixGenerator sndMatrix) {
        int rowsFstMatrix = this.getRows();
        int colsFstMatrix = this.getColumns();
        int colsSndMatrix = sndMatrix.getColumns();
        double [][] result = new double[rowsFstMatrix][colsSndMatrix];

        for (int i = 0; i < rowsFstMatrix; i++) {
            for (int j = 0; j < colsSndMatrix; j++) {
                result[i][j] = 0;
                for (int k = 0; k < colsFstMatrix; k++) {
                    result[i][j] += this.getElement(i, k) * sndMatrix.getElement(k, j);
                }
            }
        }
    }
}
```

```
        }
    }
}
}
```

Usando un PC a 64-bit, con un AMD® Ryzen™ 7 3700x 8-core processor × 16, RAM 16 GiB e sistema operativo Ubuntu 21.10, il tempo di esecuzione della versione seriale è di circa 37390 ms.

## Prima versione concorrente - Un thread per ogni elemento

La classe **ParallelIndividualMultiplier** estende la classe **MatrixGenerator** e ne implementa il metodo astratto **multiply()**. La classe prevede l'esecuzione dei thread a gruppi di 10, usando un *ArrayList*, uno per ogni elemento della matrice risultato. Se il numero di thread supera il limite di 10, il main thread si mette in attesa che terminino prima di iniziare con un altro gruppo. È possibile testare le prestazioni della soluzione variando questo valore.

### ParallelIndividualMultiplier.java

```
import java.util.ArrayList;
import java.util.List;

public class ParallelIndividualMultiplier extends MatrixGenerator {

    // Constructor
    public ParallelIndividualMultiplier (int rows, int columns) {
        super(rows, columns);
    }

    // Make the matrix product between this and the matrix given as parameter
    @Override
    public void multiply(MatrixGenerator sndMatrix) {
        List<Thread> threads = new ArrayList<>();    // group of threads that can be
                                                       // executed
        int rowsFstMatrix = this.getRows();
        int colsSndMatrix = sndMatrix.getColumns();
        double[][] result = new double[rowsFstMatrix][colsSndMatrix];

        for (int i = 0; i < rowsFstMatrix; i++) {
            for (int j = 0; j < colsSndMatrix; j++) {
                // This object is used for generating each single element of result matrix
                // creating a separate thread for each of them
                IndividualMultiplierTask task =
                    new IndividualMultiplierTask(result, this, sndMatrix, i, j);
                Thread thread = new Thread(task);
                thread.start();
                // The created thread is grouped
                threads.add(thread);
                // If the maximum number of executable thread is reached, the main thread
                // waits for their termination
                if (threads.size() % 10 == 0) {
                    waitForThreads(threads);
                }
            }
        }
    }
}
```

```
}

private static void waitForThreads (List<Thread> threads) {
    for (Thread thread : threads) {
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    threads.clear();
}
}
```

La classe **IndividualMultiplierTask** contiene il codice eseguito da ogni singolo thread lanciato dalla classe **ParallelIndividualMultiplier**, infatti implementa l'interfaccia **Runnable** e il metodo **run()**. La classe si occupa di calcolare un singolo elemento della matrice risultato.

### **IndividualMultiplierTask.java**

```
public class IndividualMultiplierTask implements Runnable {
    private final double[][] result;
    private MatrixGenerator fstMatrix;
    private MatrixGenerator sndMatrix;

    private final int row;
    private final int column;

    // Constructor
    public IndividualMultiplierTask (double[][] result,
                                    MatrixGenerator fstMatrix,
                                    MatrixGenerator sndMatrix,
                                    int row,
                                    int column) {
        this.result = result;
        this.fstMatrix = fstMatrix;
        this.sndMatrix = sndMatrix;
        this.row = row;
        this.column = column;
    }

    // This method is executed for each element of the matrix result
    @Override
    public void run() {
        result[row][column] = 0;
        for (int k = 0; k < fstMatrix.getColumns(); k++) {
            result[row][column] += fstMatrix.getElement(row, k) * sndMatrix.getElement(k, column);
        }
    }
}
```

Usando un PC a 64-bit, con un AMD® Ryzen™ 7 3700x 8-core processor × 16, RAM 16 GiB e sistema operativo Ubuntu 21.10, il tempo di esecuzione della versione seriale è di circa 119198 ms, che risulta essere circa tre volte più lento della versione seriale. Questo dimostra che una scelta sbagliata nella progettazione del parallelismo può portare a soluzioni meno performanti della versione seriale.

## Seconda versione concorrente - Un thread per ogni riga

La classe **ParallelRowMultiplier** estende la classe **MatrixGenerator** e ne implementa il metodo astratto **multiply()**. La classe prevede l'esecuzione dei thread a gruppi di 10, usando un **ArrayList**, uno per ogni riga della matrice risultato. Se il numero di thread supera il limite di 10, il main thread si mette in attesa che terminino prima di iniziare con un altro gruppo. È possibile testare le prestazioni della soluzione variando questo valore.

### ParallelRowMultiplier.java

```
import java.util.ArrayList;
import java.util.List;

public class ParallelRowMultiplier extends MatrixGenerator {

    // Constructor
    public ParallelRowMultiplier (int rows, int columns) {
        super(rows, columns);
    }

    // Make the matrix product between this and the matrix given as parameter
    @Override
    public void multiply(MatrixGenerator sndMatrix) {
        List<Thread> threads = new ArrayList<>();    // group of threads that can be
                                                       // executed
        int rowsFstMatrix = this.getRows();
        double[][] result = new double[rowsFstMatrix][colsSndMatrix];

        for (int i = 0; i < rowsFstMatrix; i++) {
            // This object is used for generating each single row of result matrix
            // creating a separate thread for each of them
            RowMultiplierTask task = new RowMultiplierTask(result, this, sndMatrix, i);
            Thread thread = new Thread(task);
            thread.start();
            // The created thread is grouped
            threads.add(thread);
            // If the maximum number of executable thread is reached, the main thread
            // waits for their termination
            if (threads.size() % 10 == 0) {
                waitForThreads(threads);
            }
        }
    }

    private static void waitForThreads (List<Thread> threads) {
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        threads.clear();
    }
}
```

La classe **RowMultiplierTask** contiene il codice eseguito da ogni singolo thread lanciato dalla classe **ParallelRowMultiplier**, infatti implementa l'interfaccia **Runnable** e il metodo **run()**. La classe si occupa di calcolare una singola riga della matrice risultato.

### RowMultiplierTask.java

```
public class RowMultiplierTask implements Runnable {
    private final double[][] result;
    private MatrixGenerator fstMatrix;
    private MatrixGenerator sndMatrix;

    private final int row;

    // Constructor
    public RowMultiplierTask(double[][] result, MatrixGenerator fstMatrix,
                           MatrixGenerator sndMatrix, int row) {
        this.result = result;
        this.fstMatrix = fstMatrix;
        this.sndMatrix = sndMatrix;
        this.row = row;
    }

    // This method is executed for each row of the matrix result
    @Override
    public void run() {
        for (int j = 0; j < sndMatrix.getColumns(); j++) {
            result[row][j] = 0;
            for (int k = 0; k < fstMatrix.getColumns(); k++) {
                result[row][j] += fstMatrix.getElement(row, k) * sndMatrix.getElement(k, j);
            }
        }
    }
}
```

Usando un PC a 64-bit, con un AMD® Ryzen™ 7 3700x 8-core processor × 16, RAM 16 GiB e sistema operativo Ubuntu 21.10, il tempo di esecuzione della versione seriale è di circa 9809 ms, che risulta essere circa 4 volte più veloce della versione seriale.

## Terza versione concorrente - Un thread per processore

La classe **ParallelProcessorMultiplier** estende la classe **MatrixGenerator** e ne implementa il metodo astratto **multiply()**. La classe prevede l'esecuzione di un numero di thread pari al numero di processori disponibili per la JVM. Ogni thread effettuerà il calcolo di un gruppo di elementi della matrice risultato suddividendo tutte le righe della prima matrice tra i vari thread avviati. I thread saranno collezionati usando un **ArrayList** e saranno tanti quanti sono i processori disponibili per la JVM.

### ParallelProcessorMultiplier.java

```
import java.util.ArrayList;
import java.util.List;

public class ParallelProcessorMultiplier extends MatrixGenerator {

    // Constructor
```

```
public ParallelProcessorMultiplier (int rows, int columns) {
    super(rows, columns);
}

// Make the matrix product between this and the matrix given as parameter
@Override
public void multiply(MatrixGenerator sndMatrix) {
    List<Thread> threads = new ArrayList<>();
    int rowsFstMatrix = this.getRows();
    double[][] result = new double[rowsFstMatrix][colsSndMatrix];

    // The method availableProcessors() gives the number of the available processors
    int numThreads = Runtime.getRuntime().availableProcessors();
    System.out.println("Number of processors: " + numThreads);
    int startIndex, endIndex, step;

    // Number of result elements that each thread has to calculate
    step = rowsFstMatrix / numThreads;
    startIndex = 0;
    endIndex = step;

    for (int i = 0; i < numThreads; i++) {
        // This object is used for generating a thread that has to calculate a number
        // of result elements given by "step"
        GroupMultiplierTask task =
            new GroupMultiplierTask(result, this, sndMatrix, startIndex, endIndex);
        Thread thread = new Thread(task);
        thread.start();
        // The created thread is grouped
        threads.add(thread);

        // Each thread has to calculate "step" different result elements
        startIndex = endIndex;
        // There is the presumption that the PC has at least two processors
        endIndex = ((i == (numThreads - 2)) ? rowsFstMatrix : (endIndex + step));
    }

    // The main thread waits for the termination of each thread
    waitForThreads(threads);
}

private static void waitForThreads (List<Thread> threads) {
    for (Thread thread : threads) {
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    threads.clear();
}
```

La classe **GroupMultiplierTask** contiene il codice eseguito da ogni singolo thread lanciato dalla classe **ParallelProcessorMultiplier**, infatti implementa l'interfaccia **Runnable** e il metodo **run()**. La classe si occupa di calcolare una singola riga della matrice risultato.

## GroupMultiplierTask.java

```
public class GroupMultiplierTask implements Runnable {
    private final double[][] result;
    private MatrixGenerator fstMatrix;
    private MatrixGenerator sndMatrix;

    private final int startIndex;
    private final int endIndex;

    // Constructor
    public GroupMultiplierTask(double[][] result, MatrixGenerator fstMatrix,
                               MatrixGenerator sndMatrix, int startIndex, int endIndex) {
        this.result = result;
        this.fstMatrix = fstMatrix;
        this.sndMatrix = sndMatrix;
        this.startIndex = startIndex;
        this.endIndex = endIndex;
    }

    // This method is executed for each group of "step" elements of the matrix result
    @Override
    public void run() {
        for (int i = startIndex; i < endIndex; i++) {
            for (int j = 0; j < sndMatrix.getColumns(); j++) {
                result[i][j] = 0;
                for (int k = 0; k < fstMatrix.getColumns(); k++) {
                    result[i][j] += fstMatrix.getElement(i, k) * sndMatrix.getElement(k, j);
                }
            }
        }
    }
}
```

Usando un PC a 64-bit, con un AMD® Ryzen™ 7 3700x 8-core processor × 16, RAM 16 GiB e sistema operativo Ubuntu 21.10, il tempo di esecuzione della versione seriale è di circa 6551 ms, che risulta essere quasi sei volte più veloce della versione seriale, considerando che il numero di processori disponibili per la JVM è pari a 16.

# La sincronizzazione dei thread

## Race condition

Il **race condition** è una proprietà di un algoritmo o di un programma che si manifesta con un output anomalo causato da un errato ordinamento degli eventi<sup>41</sup>. Ad esempio, un **race condition** si verifica quando delle operazioni non atomiche sono state progettate per essere eseguite su una risorsa condivisa da più thread, dei quali almeno uno tenta di modificarne lo stato. La risorsa rischierà di trovarsi in uno stato errato a causa delle tempistiche di scheduling dei thread che non sono stati sincronizzati.

Per **risolvere** il problema di **race condition** bisogna identificare la **sezione critica** e proteggerla con un **blocco synchronized**.

## Sezione critica e sincronizzazione

I seguenti esercizi mostrano l'uso dell'istruzione **synchronized** per sincronizzare i thread nell'accesso ad una risorsa condivisa.

### InventoryCounter

Questa semplice applicazione effettua lo scarico e il carico di merci in un magazzino usando un contatore che sarà condiviso simulando una situazione reale in cui vengano usati, ad esempio, dei lettori di codici a barre usati dai magazzinieri. Tramite questa piccola applicazione viene introdotto il concetto di sezione critica e la sua gestione tramite i monitor con l'istruzione **synchronized**. Il contatore definito nella classe InventoryCounter deve avere i metodi sincronizzati tramite un monitor per garantire la corretta gestione dell'incremento e del decremento della variabile di istanza `items` che viene acceduta in modalità concorrente.

#### Main.java

```
public class Main {  
    public static void main(String[] args) throws InterruptedException {  
        InventoryCounter inventoryCounter = new InventoryCounter();  
        IncrementingThread incrementingThread =  
            new IncrementingThread(inventoryCounter);  
        DecrementingThread decrementingThread =  
            new DecrementingThread(inventoryCounter);  
  
        incrementingThread.start();  
        System.out.println(incrementingThread.getState());  
        decrementingThread.start();  
  
        incrementingThread.join();  
        decrementingThread.join();  
    }  
}
```

---

<sup>41</sup> [V. Kovalenko, Race Condition vs. Data Race in Java, DZone, Java Zone, 7 agosto 2018](#)

```
        System.out.println("We currently have " + inventoryCounter.getItems() + " items");
    }
}
```

La sincronizzazione della sezione critica può essere effettuata sia sulla dichiarazione del metodo, sia utilizzando l'istruzione synchronized sull'oggetto stesso usando this. Questa tecnica di sincronizzazione previene l'accesso contemporaneo a qualsiasi metodo della classe da parte di thread diversi, in quanto la sincronizzazione è fatta sull'oggetto istanziato dalla classe. Quindi un solo thread alla volta potrà eseguire un qualsiasi metodo dell'oggetto.

## InventoryCounter.java

```
public class InventoryCounter {
    private int items = 0;

    public synchronized void increment() {
        //2. synchronized (this) {
            items++;
        //}
    }

    public synchronized void decrement() {
        //2. synchronized (this) {
            items--;
        //}
    }

    public /*synchronized*/ int getItems() {
        //2. synchronized (this) {
            return items;
        //}
    }
}
```

## IncrementingThread.java

```
public class IncrementingThread extends Thread {

    private InventoryCounter inventoryCounter;

    public IncrementingThread(InventoryCounter inventoryCounter) {
        this.inventoryCounter = inventoryCounter;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10000; i++) {
            inventoryCounter.increment();
        }
    }
}
```

}

## DecrementingThread.java

```
public class DecrementingThread extends Thread {  
  
    private InventoryCounter inventoryCounter;  
  
    public DecrementingThread(InventoryCounter inventoryCounter) {  
        this.inventoryCounter = inventoryCounter;  
    }  
  
    @Override  
    public void run() {  
        for (int i = 0; i < 10000; i++) {  
            inventoryCounter.decrement();  
        }  
    }  
}
```

## EvenOdd

Questa applicazione gestisce un contatore che viene incrementato e decrementato da due thread paralleli, a seconda che il valore del contatore stesso sia pari o dispari. Oltre ad essere necessaria la sincronizzazione dei metodi di gestione del contatore stesso, risulta necessaria anche la sincronizzazione delle istruzioni di visualizzazione e di controllo della parità del contatore in quanto, diversamente, queste istruzioni verrebbero eseguite parallelamente senza alcun ordine predicibile. Nell'esempio proposto la sincronizzazione dei metodi della classe **Count** potrebbe essere omessa in quanto non sono richiamati in contesti diversi dalla sezione critica individuata tramite l'oggetto count nella classe **EvenOdd**.

### Main.java

```
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class Main {  
    public static void main(String[] args) {  
        Count count = new Count();  
        Thread mng1 = new Thread(new EvenOdd(count));  
        Thread mng2 = new Thread(new EvenOdd(count));  
        mng1.start();  
        mng2.start();  
        try {  
            mng1.join();  
            mng2.join();  
        } catch (InterruptedException ex) {  
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
        }  
        System.out.println("The value of count is " + count.getCount());  
    }  
}
```

}

## Count.java

```
public class Count {  
    protected int count;  
  
    public Count() { this.count = 0; }  
  
    public synchronized int getCount() { return this.count; }  
  
    public synchronized void increment() { count++; }  
  
    public synchronized void decrement() { count--; }  
}
```

## EvenOdd.java

```
public class EvenOdd implements Runnable {  
    private Count count;  
  
    public EvenOdd(Count count) { this.count = count; }  
  
    @Override  
    public void run() {  
        for(int i = 0; i < 10; i++) {  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            synchronized(count) {  
                if ((count.getCount() % 2) == 0) {  
                    System.out.print(Thread.currentThread().getName() + ": ");  
                    count.increment();  
                    System.out.println(" count = " + count.getCount());  
                } else {  
                    System.out.print(Thread.currentThread().getName() + ": ");  
                    count.decrement();  
                    System.out.println(" count = " + count.getCount());  
                }  
            }  
        }  
    }  
}
```

## BankAccount

La seguente applicazione gestisce un conto corrente cointestato tra due clienti di una banca. Il conto corrente avrà un saldo iniziale a scelta, e i due correntisti potranno effettuare a piacimento sia operazioni di prelievo, sia di deposito, ma appena uno dei due

correntisti manderà il saldo del conto in rosso, non potrà più effettuare alcuna operazione su di esso.

L'applicazione deve mostrare a video il succedersi di depositi e prelievi da parte dei due correntisti, segnalando anche quando un correntista dovrà terminare qualsiasi operazione avendo mandato in rosso il conto. Quando entrambi i due correntisti avranno mandato in rosso il conto l'applicazione terminerà mostrando l'ammontare del saldo negativo raggiunto.

Per randomizzare le operazioni di prelievo e deposito si procede con l'estrazione di un numero casuale che se risulta maggiore di 0.5 effettua un prelievo, diversamente farà effettuare un deposito. Rendendo però casuale la sequenza di depositi e prelievi non sarà prevedibile quando terminerà l'applicazione. Il saldo del conto viene inizializzato ad un valore contenuto, e le operazioni saranno effettuate con ammontari pari a circa la metà del saldo più un valore casuale normalizzato alle cifre che si sta trattando. Ad esempio, se il saldo fosse inizializzato a 1000€, le somme da muovere possono essere calcolate come (500 + (random \* 200.)).

Tutte le operazioni sul conto sono sincronizzate in quanto il conto è una risorsa condivisa tra i due clienti, così come lo sono anche le operazioni di visualizzazione del prelievo e del deposito da parte di ogni cliente.

Di seguito viene mostrato un esempio di esecuzione del programma.

```
Saldo iniziale 1000.0€  
Thread-0: Prelevati 676€ - Saldo 324.0€  
Thread-1: Depositati 578€ - Saldo 902.0€  
Thread-0: Prelevati 605€ - Saldo 297.0€  
Thread-1: Prelevati 692€ - Saldo -395.0€  
Thread-1 ha finito i soldi  
Thread-0: Depositati 589€ - Saldo 194.0€  
Thread-0: Prelevati 617€ - Saldo -423.0€  
Thread-0 ha finito i soldi  
The amount is -423.0
```

## Operazioni atomiche e l'istruzione volatile

Le operazioni atomiche sono operazioni che vengono eseguite completamente in modo non interrompibile.

In Java le operazioni atomiche sono le seguenti.

- **Assegnamenti di riferimenti ad oggetti:**

```
Object a = new Object();  
Object b = new Object();  
a = b; // operazione atomica
```

Questo permette di non dover sincronizzare tutti i getter e setter che impostano array, String e oggetti di qualsiasi tipo.

- Tutti gli **assegnamenti di tipi primitivi, ad eccezione dei tipi long e double**. Per questi ultimi Java non garantisce l'atomicità in quanto, essendo lunghi 64 bit,

l'operazione può avvenire con due assegnamenti separati, a blocchi di 32 bit alla volta, anche se si usa una macchina a 64 bit. **Per questi tipi è possibile usare l'istruzione volatile** per avere la garanzia che l'operazione sia effettuata in modo atomico.

```
volatile double z = 1.0;
volatile double y = 9.0;
x = y; // operazione atomica
```

L'uso dell'istruzione **volatile** permette anche di non sincronizzare interi metodi semplicemente perché gestiscono una variabile long o double, ottimizzando le prestazioni di esecuzioni in quanto si garantisce l'esecuzione parallela del metodo da parte di più thread. Quindi l'istruzione **volatile** permette di risolvere anche alcuni casi di **race condition**.

Un'altra **caratteristica importante dell'istruzione volatile è il fatto che una variabile così dichiarata deve essere sempre memorizzata nella memoria principale**. Questo vuol dire che la lettura di una variabile volatile avverrà dalla memoria centrale e non dalla cache della CPU, e la scrittura di una variabile volatile sarà effettuata direttamente in memoria centrale, e non solo nella cache della CPU.

## Esempio d'uso di volatile

Il seguente esempio mostra una tecnica di misurazione di metriche di esecuzione di un'applicazione. La necessità di tali misurazioni permette di effettuare delle stime per il miglioramento del prodotto. L'applicazione presa in oggetto sarà solo simulata tramite uno `sleep()`, per concentrarci esclusivamente sulla gestione del parallelismo.

### Main.java

```
public class Main {
    public static void main(String[] args) {
        Metrics metrics = new Metrics();
        BusinessLogic businessLogic1 = new BusinessLogic(metrics);
        BusinessLogic businessLogic2 = new BusinessLogic(metrics);
        MetricsPrinter metricsPrinter = new MetricsPrinter(metrics);

        businessLogic1.start();
        businessLogic2.start();
        metricsPrinter.start();
    }
}
```

La classe **Metrics** colleziona i campioni di misurazione dei diversi thread avviati dall'applicazione rappresentata dalla classe **BusinessLogic**.

La variabile `average` di tipo `double` viene impostata come `volatile`, in modo da manipolarla in modo completamente atomico, evitando di sincronizzare l'intero metodo `getAverage()`, e riducendo così l'impatto della sua esecuzione sul calcolo delle metriche della classe **BusinessLogic**.

Il metodo `addSample()` dovrà invece essere sincronizzato in quanto non risulta essere atomico.

## Metrics.java

```
public class Metrics {
    private long count = 0;
    private volatile double average = 0.0;

    public synchronized void addSample(long sample) {
        double currentSum = average * count;
        count++;
        // L'assegnamento è atomico in quanto average è volatile.
        average = (currentSum + sample) / count;
    }

    /*
    Dato che il metodo non è sincronizzato, sarà eseguito in parallelo
    ottimizzando le prestazioni. Non è necessario sincronizzarlo in quanto
    l'attributo average è volatile.
    */
    public double getAverage() {
        // Il return, che corrisponde ad un assegnamento, è atomico in quanto
        // average è volatile.
        return average;
    }
}
```

La classe **BusinessLogic** simula un'applicazione della quale si vuole misurare la prestazioni ogni volta che viene avviato un thread. L'invocazione del metodo `addSample()` avviene in modo sincronizzato, uno solo thread alla volta potrà invocarlo.

## BusinessLogic.java

```
import java.util.Random;

public class BusinessLogic extends Thread {
    private Metrics metrics;
    private Random random = new Random();

    public BusinessLogic(Metrics metrics) {
        this.metrics = metrics;
    }

    @Override
    public void run() {
        while (true) {
            long start = System.currentTimeMillis();
            try {
                Thread.sleep(random.nextInt(10));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
        }
        long end = System.currentTimeMillis();
        metrics.addSample(end - start);
    }
}
}
```

La classe **MetricsPrinter** visualizza le misurazioni effettuate ad intervalli regolari. Il thread che esegue **MetricsPrinter** è eseguito completamente in parallelo con i thread di **BusinessLogic** in quanto il metodo `getAvarage()` della classe **Metrics** non è sincronizzato.

### MetricsPrinter.java

```
public class MetricsPrinter extends Thread {
    private Metrics metrics;

    public MetricsPrinter(Metrics metrics) {
        this.metrics = metrics;
    }

    @Override
    public void run() {
        while(true) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            double currentAvarage = metrics.getAverage();
            System.out.println("Current Average is " + currentAvarage);
        }
    }
}
```

Un esempio di esecuzione è il seguente.

```
Current Average is 4.581395348837209
Current Average is 4.728260869565219
Current Average is 4.6666666666666669
Current Average is 4.653631284916205
Current Average is 4.696832579185524
Current Average is 4.688212927756657
Current Average is 4.66558441558442
Current Average is 4.64305949008499
Current Average is 4.568578553615964
Current Average is 4.5936794582392775
...
...
```

## Data race

Un **data race** è una proprietà di una esecuzione di un programma, non del programma stesso come invece si verifica per il **race condition**. Un'esecuzione di un programma contiene un **data race** se sono presenti almeno due accessi in conflitto ad una medesima variabile di tipo non volatile, e questi accessi non risultano ordinati da una relazione **happens-before**. I due accessi sono solitamente almeno una lettura e almeno una scrittura<sup>42</sup>.

Una relazione **happen-before** garantisce un ordinamento preciso delle azioni svolte da più thread, definendo un ordinamento parziale di esecuzione delle azioni di un programma. In assenza di una relazione **happen-before**, la JVM è libera di riordinare le azioni di una processo per ragioni di efficienza esecutiva, portando ad esecuzioni potenzialmente errate in caso di multi-threading e azioni di lettura e scrittura su variabili non volatile<sup>43</sup>.

Generalmente per ottimizzare le performance di utilizzo del sistema, il compilatore e la CPU possono eseguire le istruzioni con un ordine diverso da quanto indicato nel programma (**out of order**). Questi riordinamenti vengono fatti fin tanto che la logica del codice rimane corretta.

L'esecuzione **out of order** è una caratteristica molto importante per velocizzare l'esecuzione del codice di un processo, e il compilatore riorganizza le istruzioni per effettuare una migliore predizioni su condizioni o cicli (**branch prediction**), usare al meglio gli array processor nelle architetture SIMD o effettuare il prefetching di istruzioni in modo tale da migliorare le performance di caching delle stesse. La CPU invece riordina le istruzioni per una più efficiente gestione delle singole unità hardware, evitando di sprecare cicli macchina nel caso in cui delle unità non siano disponibili mentre lo sono altre.

Il riordino delle istruzioni può avvenire solo se le istruzioni non sono strettamente dipendenti l'una dall'altra, come nel caso seguente dove i due metodi risultano equivalenti, e il riordino delle istruzioni non varia la logica del programma.

```
public void increment1() {  
    x++;  
    y++;  
}  
  
public void increment2() {  
    y++;  
    x++;  
}
```

Se le istruzioni fossero però strettamente dipendenti l'una dall'altra, come nel caso seguente, non potrebbero essere riordinate perché diversamente diverrebbe errata la logica del programma.

```
public void someMethod() {  
    x = 1;
```

<sup>42</sup> [V. Kovalenko, Race Condition vs. Data Race in Java, DZone, Java Zone, 7 agosto 2018](#)

<sup>43</sup> [Java - Understanding Happens-before relationship, LogicBig, 19 maggio 2018](#)

```
y = x + 2;  
z = y + 10;  
}
```

In una applicazione mono thread il riordino delle istruzioni è completamente ininfluente, ma nel caso di applicazioni multi thread potrebbe portare a risultati errati e paradossali in quanto l'esecuzione contemporanea da parte di due thread, di due metodi che necessitano di un preciso ordine di esecuzione, stravolgerebbe la logica del programma portando a possibili esecuzioni errate dovute al **data race**.

Si consideri il seguente programma nel quale la classe **SharedClass** prevede due metodi, `increment()` e `checkForDataRace()` che vengono eseguiti da due thread separati, `thread1` e `thread2`. Per brevità di codice i due thread vengono creati tramite delle lambda expression e la classe **SharedClass** viene creata come classe statica interna alla classe **Main**.

### Main.java

```
public class Main {  
  
    public static void main(String[] args) {  
        SharedClass sharedClass = new SharedClass();  
        // Lambda expression  
        Thread thread1 = new Thread(() -> {  
            for (int i = 0; i < Integer.MAX_VALUE; i++) {  
                sharedClass.increment();  
            }  
        }) ;  
  
        // Lambda expression  
        Thread thread2 = new Thread(() -> {  
            for (int i = 0; i < Integer.MAX_VALUE; i++) {  
                sharedClass.checkForDataRace();  
            }  
        }) ;  
  
        thread1.start();  
        thread2.start();  
    }  
  
    // Inner static class  
    public static class SharedClass {  
        private int x = 0;  
        private int y = 0;  
  
        public void increment() {  
            x++;  
            y++;  
        }  
  
        public void checkForDataRace() {  
            if (y > x) {  
                System.out.println("Data race detected!");  
            }  
        }  
    }  
}
```

```
        System.out.println("y > x - Data Race is detected");
    }
}
}
```

L'istruzione `if (y > x)` del metodo `checkForDataRace()` non è atomica, quindi la valutazione dei valori presenti in `y` e in `x` può avvenire con qualsiasi forma di interferenza da parte del metodo `increment()` a causa dello interleaving di esecuzione:

Thread2: checkForDataRace()	Thread1: increment()
1. read <code>y ← 0</code>	
2. read <code>x ← 0</code>	
	3. <code>x++;</code>
	4. <code>y++;</code>
<code>if (y &gt; x) FALSE (0 == 0)</code>	

Thread2: checkForDataRace()	Thread1: increment()
1. read <code>y ← 0</code>	
	2. <code>x++;</code>
	3. <code>y++;</code>
4. read <code>x ← 1</code>	
<code>if (y &gt; x) FALSE (0 &lt; 1)</code>	

Thread2: checkForDataRace()	Thread1: increment()
1. read <code>y ← 0</code>	
	2. <code>x++;</code>
3. read <code>x ← 1</code>	
	4. <code>y++;</code>
<code>if (y &gt; x) FALSE (0 &lt; 1)</code>	

Qualunque sia l'ordine relativo di esecuzione di `checkForDataRace()` e `increment()`, la variabile `x` risulterà sempre maggiore o uguale alla variabile `y`, e quindi la condizione `if (y > x)` dovrebbe sempre dare il risultato false, non visualizzando mai la stringa "y > x - Data Race is detected". **A causa però del riordino delle istruzioni da parte del**

**compilatore e della CPU ciò non è sempre vero**, risultando in un **data race**, e quindi in una esecuzione errata del codice.

Una generica esecuzione del codice precedente potrebbe portare al seguente risultato.

```
y > x - Data Race is detected  
y > x - Data Race is detected  
y > x - Data Race is detected  
y > x - Data Race is detected
```

Le soluzioni per evitare il **data race** sono due:

- **sincronizzare i metodi** che modificano o leggono le variabili condivise, in modo tale che un solo thread alla volta possa accedere alla variabile condivisa, con la penalizzazione che si perde l'esecuzione parallela dei metodi;
- **dichiarare le variabili condivise come volatile** che permette di garantire l'ordine di esecuzione, eliminando al contempo l'overhead del locking tramite la sincronizzazione e permettendo quindi l'esecuzione parallela dei due metodi.

È comunque possibile usare qualsiasi meccanismo che permetta di imporre una relazione **happen-before** per risolvere un **data race**, come l'uso del metodo **join()** per imporre un'attesa tra i thread, ordinare l'avvio dei thread ordinando le chiamate dei metodi **start()**, o usare un'applicazione single-thread.

## Strategie di locking delle risorse e deadlock [DA FINIRE]

La scelta della strategia di locking delle risorse condivise è una scelta importante quando si sviluppa un'applicazione multi-thread.

Un locking grossolano che preveda la sincronizzazione dei metodi di una classe su un solo oggetto è semplice e non crea alcun problema di deadlock, in quanto un solo thread alla volta potrà eseguire un metodo dell'oggetto, mentre tutti gli altri thread dovranno attendere anche se richiedono l'esecuzione di un metodo su un'altra risorsa condivisa. Questa tecnica riduce le performance dell'applicazione in quanto riduce il parallelismo, oltre a non consentire l'esecuzione parallela di operazioni che non si influenzano tra di loro in quanto potrebbero non coinvolgere le risorse condivise.

Un locking molto preciso potrebbe prevedere una sincronizzazione su ciascuna risorsa condivisa permettendo l'indubbio vantaggio di permettere un maggior parallelismo in fase di esecuzione. Il problema che può verificarsi in questo caso è una situazione di **deadlock**, cioè un blocco critico, a causa di un'**attesa circolare** sulle risorse di tipo seriale (utilizzabili in **mutua esclusione**) e **non prerilasciabili**, e a seguito di una richiesta multipla e bloccante (**hold-and-wait**) da parte di più thread.

Vediamo un semplice esempio di deadlock. Si supponga che le risorse A e B siano risorse seriali e non prerilasciabili, e che i due thread, Thread1 e Thread2 debbano acquisirle entrambe per evolvere, diversamente l'operazione non potrà essere portata a termine. La seguente logica porterà ad un deadlock in base allo interleaving di esecuzione dei due thread, senza possibilità di prevedere quando e se si verificherà il blocco critico. Le

operazioni `lock(risorsa)` e `unlock(risorsa)` corrispondono in Java alla sincronizzazione sull'oggetto acquisendone il monitor, e al suo rilascio quando termina il blocco sincronizzato.

Thread1	Thread2
<code>lock(A)</code> <code>lock(B)</code> <code>delete(B, item)</code> <code>add(A, item)</code> <code>unlock(B)</code> <code>unlock(A)</code>	<code>lock(B)</code> <code>lock(A)</code> <code>delete(A, item)</code> <code>add(B, item)</code> <code>unlock(A)</code> <code>unlock(B)</code>

In base all'ordine di esecuzione dei due thread potrebbe o meno verificarsi un deadlock. Se l'ordine di esecuzione risulterà il seguente sicuramente il sistema raggiungerà lo stato di blocco critico.

Thread1	Thread2
<b>1. lock(A)</b>	
	<b>2. lock(B)</b>
	<b>3. lock(A)</b>
<b>4. lock(B)</b>	

Il primo lock di entrambi i thread non crea problemi, ma nel momento che Thread2 cerca di effettuare il lock su A viene bloccato in quanto questo è già stato acquisito da Thread1, quindi dovrà aspettare che quest'ultimo rilasci il monitor. Analogamente, quando il Thread1 cerca di effettuare il lock su B viene bloccato in quanto questo è già stato acquisito da Thread2. Entrambi i thread aspettano circolarmente il rilascio di una risorsa che non sarà mai rilasciata, portando il sistema in un blocco critico.

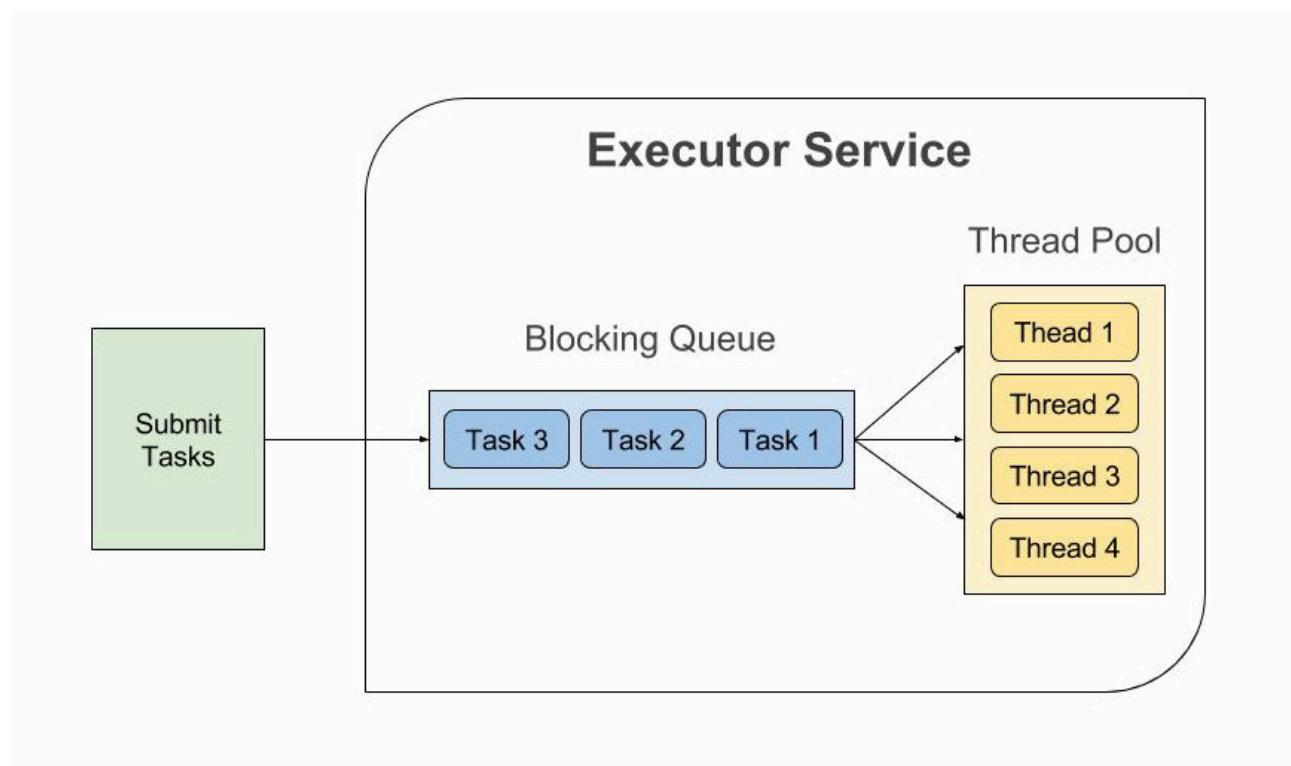
# ExecutorService

Normalmente i server gestiscono grandi quantità di thread simultaneamente, nell'ordine delle centinaia se non anche delle migliaia, e con questi numeri diventa essenziale separare la creazione e la gestione dei thread dal resto dell'applicazione.

Gli Executor sono un framework che svolge proprio l'azione di creazione e gestione dei thread, con una serie di servizi che rendono semplice la gestione di grandi quantità di thread.

Negli esempi seguenti viene mostrato esplicitamente l'uso di un **Thread Pool** a dimensione fissa, mentre nella sezione CLIL si potranno analizzare degli esempi anche per gestioni diverse.

In generale lo **Executor** crea un pool di thread che verranno utilizzati più volte per eseguire gruppi di task. Questo pool di thread esiste separatamente dai task creati implementando l'interfaccia **Runnable**, e sono completamente gestiti dallo **Executor**. Tutti i task da eseguire vengono posti in una coda di attesa, denominata **Blocking Queue** in attesa che si liberi un thread del pool per poter essere eseguiti. Nel caso la **Blocking Queue** risultasse piena, i nuovi task verrebbero scartati.



## CLIL Reading - Executor Service & Thread Pool

Read the following tutorial, it explains the Executors framework that is used for creating and managing threads. Specifically Executors helps with:

- thread creation using a thread pool that the application can use to run tasks concurrently;

- thread management of the threads' life cycle in the thread pool, in order to use the threads to submitting tasks for execution, without any regards about threads state;
- task submission and execution providing the methods for submitting a task and eventually scheduling the execution of the task in the future.

## [Java ExecutorService and Thread Pools Tutorial](#)

# Esempi d'uso degli Executor

## Moltiplicazione tra matrici

In questo paragrafo viene riproposta una versione modificata della moltiplicazione tra matrici usando un thread per ogni riga della matrice risultato. Si lascia al lettore, come esercizio, la modifica della terza versione, dove il numero dei thread dipendeva dal numero di processori dedicati per la JVM.

Di seguito verranno proposti solo i metodi che sono stati modificati rispetto alla versione originale.

La classe **Main.java** non è più deputata al calcolo del tempo in quanto l'operazione viene svolta ora in concomitanza dell'uso dell'interfaccia ExecutorService all'interno della classe **ParallelRowMultiplier.java**.

### **Main.java**

```
/*
 This concurrent version is faster then the parallel individual, it
 needs a factor of 10 less then the parallel individual.

*/
public class Main {
    public static void main(String[] args) {
        // First matrix
        MatrixGenerator fstMatrix = new ParallelRowMultiplier(2000, 2000);
        // Second matrix
        MatrixGenerator sndMatrix = new ParallelRowMultiplier(2000, 2000);
        // Matrix product
        fstMatrix.multiply(sndMatrix);
    }
}
```

La classe **ParallelRowMultiplier.java** prevede la creazione di un oggetto **ExecutorService** che si occuperà di gestire un pool fisso di thread che eseguiranno in concorrenza i task di calcolo delle righe della matrice risultato. Come si può vedere non risulta più necessario usare un oggetto ArrayList e contare il numero dei thread in esecuzione in modo tale che non venga superato il limite imposto; tutti questi controlli vengono svolti in automatico dall'oggetto di tipo **ExecutiorService**.

Il calcolo del tempo trascorso è stato spostato in questa classe in quanto, dopo aver richiesto l'interruzione dell'esecuzione dell'oggetto di tipo **ExecutiorService**, viene invocato il metodo [awaitTermination\(\)](#) che permette di interrompere la terminazione dell'**ExecutiorService** fino a quando non saranno conclusi tutti i task.

## ParallelRowMultiplier.java

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

public class ParallelRowMultiplier extends MatrixGenerator {

    // Constructor
    public ParallelRowMultiplier (int rows, int columns) {
        super(rows, columns);
    }

    // Make the matrix product between this and the matrix given as parameter
    @Override
    public void multiply(MatrixGenerator sndMatrix) {
        int rowsFstMatrix = this.getRows();
        double[][] result = new double[this.getRows()][sndMatrix.getColumns()];

        // Creating executor Service with a thread pool of size 10
        ExecutorService executorService = Executors.newFixedThreadPool(10);
        // Matrix product start time
        Date start = new Date();
        for (int i = 0; i < rowsFstMatrix; i++) {
            // This object is used for generating each single row of result matrix
            // creating a separate thread for each of them
            RowMultiplierTask task = new RowMultiplierTask(result, this, sndMatrix, i);
            executorService.submit(task);
        }
        // The Executor Service stops accepting new tasks, not waits for previously submitted
        // tasks to execute, and then terminates the executor.
        executorService.shutdown();
        try {
            // The method awaitTermination(long timeout, TimeUnit unit) blocks the
            // ExecutorService until all tasks have completed execution after a shutdown
            // request, or the timeout occurs, or the current thread is interrupted,
            // whichever happens first.
            executorService.awaitTermination(3, TimeUnit.MINUTES);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // Matrix product end time
        Date end = new Date();
        // Display the time needed for the matrix product
        System.out.printf("Time: %d\n", end.getTime() - start.getTime());
    }
}
```

## B - Design Patterns - Propedeutica

# Design Patterns

The following article describes what Design Patterns are. Read carefully in order to grasp the main purpose of the topic.

[What's a design pattern?](#) by [refactoring.guru](#)

## Observer Pattern

The Observer Pattern is one of the most used Design Patterns. Read the following articol in order to understand the logic that defines it.

[Observer](#) by [refactoring.guru](#)

At the following link you can find a code example in Java.

[Observer in Java](#) by [refactoring.guru](#)

## Esempi visti a lezione

### Observer

L'Observer Pattern prevede una classe che simula l'entità osservata, che in questo caso è la classe **Uno**, e più classi che la osservano e useranno il dato generato da essa, nel nostro esempio le classi **Due** e **Tre**.

Il progetto in questione prevede che la classe **Uno** generi un numero casuale tra 0 e 10, e gli osservatori **Due** e **Tre** consumino il numero visualizzandolo in modo discriminato: **Due** visualizza il valore se è maggiore di 4, mentre **Tre** lo visualizza se è minore di 5.

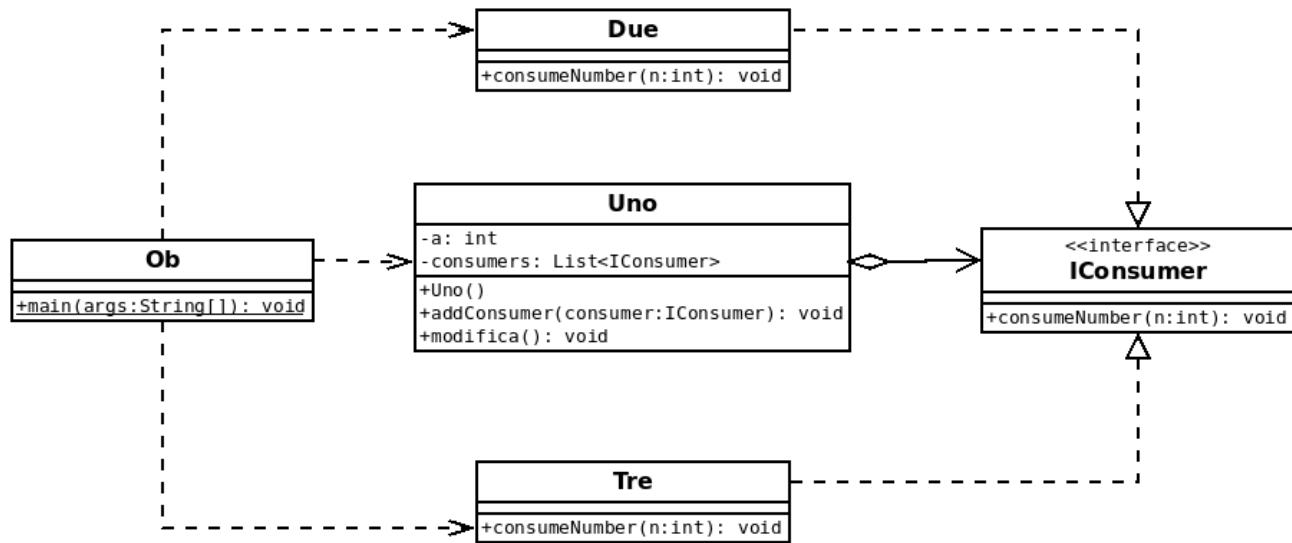
La classe **Uno** prevede tutti i metodi per permettere agli osservatori di essere aggiunti nella lista degli osservatori (`addConsumer()`), e di inviare le notifiche di cambiamento (`modifica()`) agli osservatori quando nell'entità osservata avviene un cambiamento, nel nostro esempio quando viene generato un nuovo numero casuale. Si osservi che nell'esempio non è stato previsto un metodo per rimuovere dalla lista gli osservatori, ma se necessario può essere aggiunto.

L'interfaccia **IConsumer** dichiara il metodo che serve per notificare a tutti gli osservatori (**Due** e **Tre**) il cambiamento avvenuto nell'entità **Uno**, nel nostro esempio il metodo `consumeNumber()`.

Le due classi **Due** e **Tre** implementano concretamente il comportamento rappresentato dall'interfaccia **IConsumer**, e costituiscono le entità che osservano **Uno** consumando i numeri random generati tramite il metodo `consumeNumber()`. Le classi **Due** e **Tre** devono implementare la stessa interfaccia in modo tale che **Uno** non sia accoppiato direttamente a nessuna classe concreta. Questo comportamento rispecchia il quinto principio S.O.L.I.D.<sup>44</sup>.

---

<sup>44</sup> Dependency Inversion Principle of [The S.O.L.I.D. Principles](#) by [Ugonna Thelma](#). If necessary you can also read these: [A Solid Guide to SOLID Principles](#) by [Baeldung](#)



Di seguito viene riportato il codice che implementa il progetto<sup>45</sup>.

### Ob.java

```
public class Ob {
    public static void main(String[] args) {
        Uno u = new Uno();
        Due d = new Due();
        Tre t = new Tre();
        u.addConsumer(d);
        u.addConsumer(t);
        u.modifica();
        u.modifica();
        u.modifica();
        u.modifica();
    }
}
```

### IConsumer.java

```
public interface IConsumer {
    public void consumeNumber(int n);
}
```

### Uno.java

```
import java.util.ArrayList;
import java.util.List;

public class Uno {
    private int a;
    private List<IConsumer> consumers;

    public Uno() {
        consumers = new ArrayList<>();
        a = 0;
    }
}
```

<sup>45</sup> Il codice è stato sviluppato dal Prof. Bellucci.

```
public void addConsumer(IConsumer consumer) {  
    consumers.add(consumer);  
}  
  
public void modifica() {  
    a = (int)(Math.random()*10);  
    for (IConsumer consumer: consumers) {  
        consumer.consumeNumber(a);  
  
    }  
}  
}
```

### **Due.java**

```
public class Due implements IConsumer{  
    @Override  
    public void consumeNumber(int n){  
        if(n > 4) {  
            System.out.println("due: " + n);  
        }  
    }  
}
```

### **Tre.java**

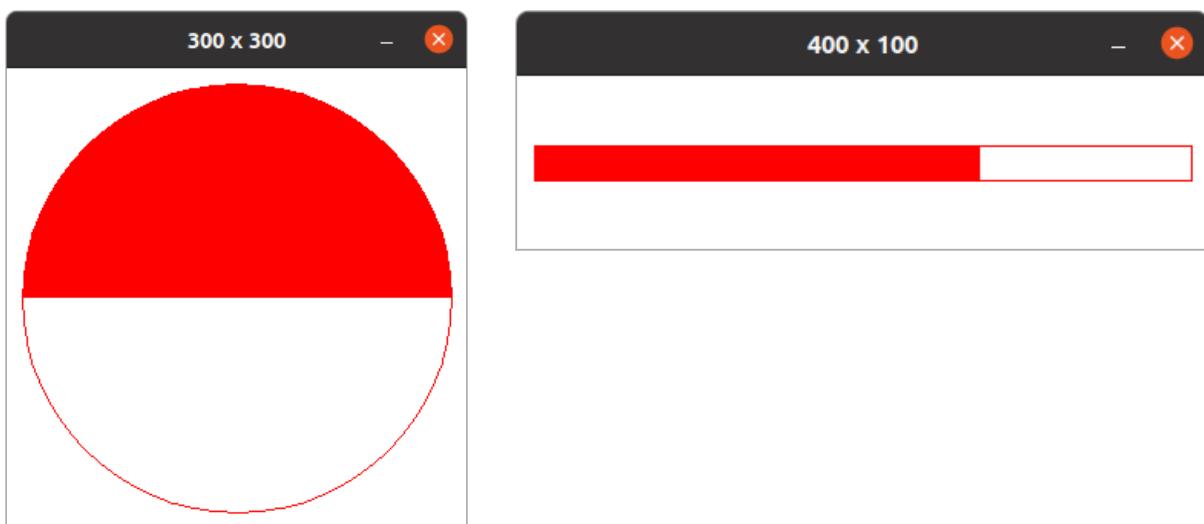
```
public class Tre implements IConsumer{  
    @Override  
    public void consumeNumber(int n){  
        if(n < 5) {  
            System.out.println("tre: " + n);  
        }  
    }  
}
```

## Esercizio grafica observer

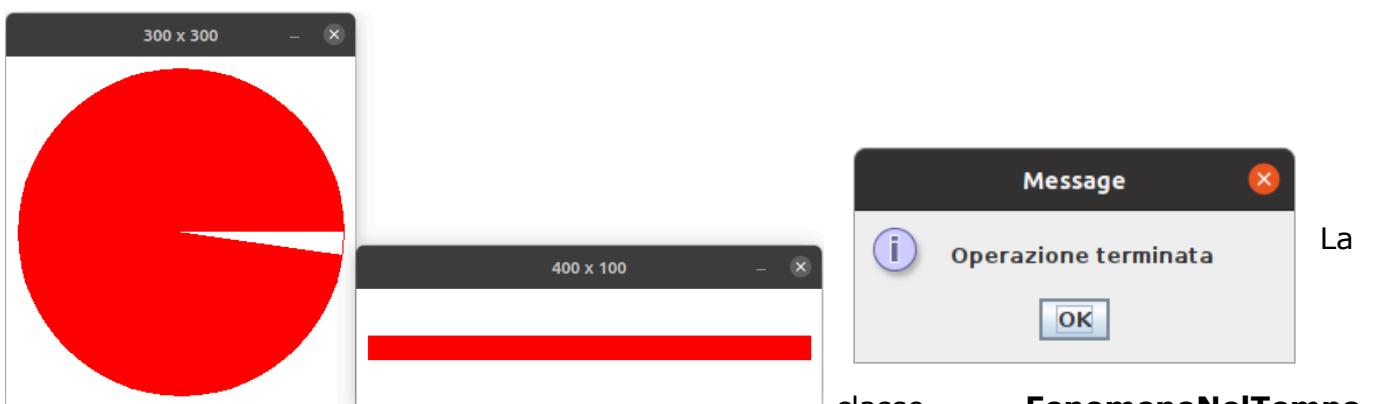
In questo esercizio l'entità osservata è la classe **FenomenoNelTempo**, mentre le classi che osservano e che useranno il dato generato da **FenomenoNelTempo** sono le classi **ViewA** e **ViewB**.

Il progetto in questione prevede che la classe **FenomenoNelTempo** frazioni un tempo definito per permettere ai due osservatori di simulare lo scorrere del tempo, rappresentandoli graficamente con due figure geometriche, un rettangolo (**ViewA**) e un cerchio (**ViewB**). Il thread di **FenomenoNelTempo** viene posto in sleep per un tempo random per simulare un comportamento di analisi di ipotetici fenomeni diversi nelle due rappresentazioni grafiche dello scorrere del tempo.

Durante l'esecuzione le due View mostreranno lo scorrere del tempo riempiendo le rispettive figure geometriche,



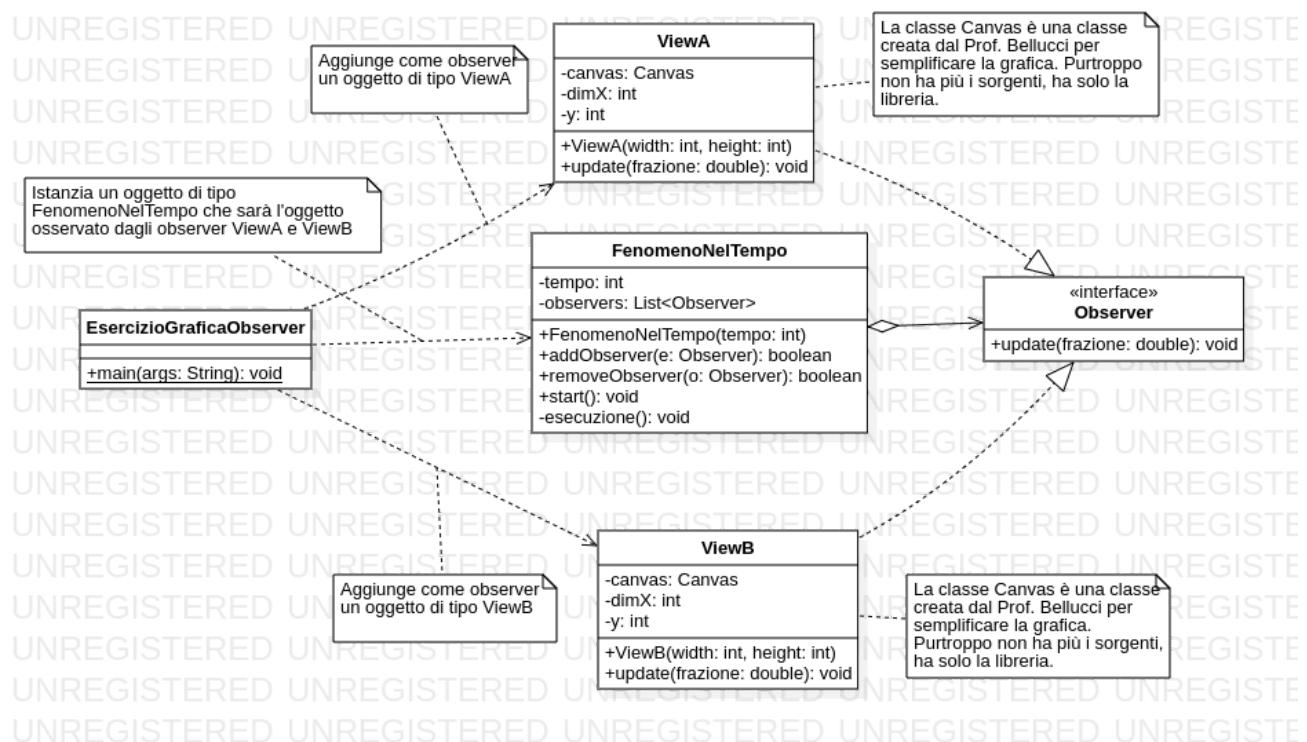
e al termine dell'esecuzione plausibilmente una delle due View concluderà il riempimento dell'area prima dell'altra terminando l'esecuzione dell'applicazione.



aggiunge gli osservatori **ViewA** e **ViewB** tramite il metodo `addObserver()`, e li rimuove usando il metodo `removeObserver()`, mentre notifica i cambiamenti agli osservatori tramite il metodo `esecuzione()`, invocato nel metodo `start()`. La notifica prevede l'invio di una frazione del tempo trascorso, dopo aver messo in `sleep()` per un tempo random il thread di **FenomenoNelTempo**.

L'interfaccia **Observer** dichiara il metodo che serve per notificare a tutti gli osservatori (**ViewA** e **ViewB**) il cambiamento avvenuto nell'entità **FenomenoNelTempo**, nel nostro esempio il metodo `update()`.

Le due classi **ViewA** e **ViewB** implementano concretamente il comportamento rappresentato dall'interfaccia **Observer**, e costituiscono le entità che osservano **FenomenoNelTempo** consumando le frazioni di tempo generate da esso tramite il metodo i numeri random generati tramite il metodo `esecuzione()`. Le classi **ViewA** e **ViewB** devono implementare la stessa interfaccia in modo tale che **FenomenoNelTempo** non sia accoppiato direttamente a nessuna classe concreta. Questo comportamento rispecchia il quinto principio S.O.L.I.D.<sup>46</sup>.



Di seguito viene riportato il codice che implementa il progetto<sup>47</sup>.

### EsercizioGraficaObserver.java

```

public class EsercizioGraficaObserver {
    public static void main(String[] args) {
        FenomenoNelTempo fenomenoNelTempo = new FenomenoNelTempo(50);
        fenomenoNelTempo.addObserver(new ViewA(400, 100));
        fenomenoNelTempo.addObserver(new ViewB(300, 300));
        fenomenoNelTempo.start();
    }
}

```

### Observer.java

```

public interface Observer {
    public void update(double frazione);
}

```

<sup>46</sup> Dependency Inversion Principle of [The S.O.L.I.D. Principles](#) by [Ugonna Thelma](#). If necessary you can also read these: [A Solid Guide to SOLID Principles](#) by [Baeldung](#)

<sup>47</sup> Il codice è stato sviluppato dal Prof. Bellucci.

}

**FenomenoNelTempo.java**

```
import java.util.ArrayList;
import java.util.List;

public class FenomenoNelTempo {
    private int tempo;
    private List<Observer> observers;

    public FenomenoNelTempo(int tempo) {
        this.tempo = tempo;
        observers = new ArrayList<>();
    }

    public boolean addObserver(Observer e) {
        return observers.add(e);
    }

    public boolean removeObserver(Observer o) {
        return observers.remove(o);
    }

    private void esecuzione() {
        for (int i = 1; i <= tempo; i++) {
            try {
                Thread.sleep((int) (Math.random() * 1000));
            } catch (InterruptedException ex) {
                throw new RuntimeException(ex);
            }
            for (Observer observer : observers) {
                observer.update((double)i / tempo);
            }
        }
    }

    public void start() {
        esecuzione();
    }
}
```

**ViewA.java**

```
import graphic.Canvas;
import java.awt.Color;
import javax.swing.JOptionPane;

public class ViewA implements Observer {
    /*
     * La classe Canvas è una classe creata dal Prof. Bellucci per semplificare
     * la grafica.
     * Purtroppo non ha più i sorgenti, ha solo la libreria.
     */
}
```

```
private Canvas canvas;
private int dimX;
private int y;

public ViewA(int width, int height) {
    canvas = new Canvas(width, height);
    canvas.setColor(Color.red);
    dimX = width - 20;
    y = height / 2 - 10;
    canvas.drawRect(10, height / 2 - 10, width - 20, 20);
}

@Override
public void update(double frazione) {
    canvas.fillRect(10, y, (int) (dimX * frazione), 20);
    if (frazione == 1) {
        JOptionPane.showMessageDialog(null, "Operazione terminata");
        canvas.close();
    }
}
}
```

## ViewB.java

```
import graphic.Canvas;
import java.awt.Color;
import javax.swing.JOptionPane;

public class ViewB implements Observer {
    /*
     * La classe Canvas è una classe creata dal Prof. Bellucci per semplificare
     * la grafica.
     * Purtroppo non ha più i sorgenti, ha solo la libreria.
     */
    private Canvas canvas;
    private int dimX;
    private int dimY;

    public ViewB(int width, int height) {
        canvas = new Canvas(width, height);
        canvas.setColor(Color.red);
        dimX = width - 20;
        dimY = height - 20;
        canvas.drawOval(10, 10, dimX, dimY);
    }

    @Override
    public void update(double frazione) {
        canvas.fillArc(10, 10, dimX, dimY, 0, (int) (360 * frazione));
        if (frazione == 1) {
            JOptionPane.showMessageDialog(null, "Operazione terminata");
            canvas.close();
        }
    }
}
```

```

    }
}

```

## Roulette

In questo esercizio l'entità osservata è la classe **Roulette**, mentre la classe che osserva e che userà il dato generato da **Roulette** è la classe **Player**.

Il progetto in questione prevede che la classe **Roulette** permetta di simulare il gioco della roulette. I giocatori scelgono a caso un valore all'interno di un intervallo stabilito al momento della creazione della roulette sperando di vincere nel momento che la roulette simula il giro della ruota.

La classe **Roulette** aggiunge l'osservatore **Player** tramite il metodo `addPlayer()`, e lo rimuove usando il metodo `removePlayer()`, mentre notifica i cambiamenti all'osservatore tramite il metodo `play()`. I cambiamenti da notificare sono i numeri selezionati dal movimento della ruota, simulata con la selezione di un valore pseudo-casuale nell'intervallo definito nel momento dell'instanziazione della roulette. All'interno di questo metodo viene invocato il metodo `verify()`, definito nella classe **Player** e dichiarato nell'interfaccia **IPlayer**, che verifica se lo specifico giocatore ha vinto o meno. L'interfaccia **IPlayer** viene implementata concretamente dalla classe **Player**. Questo comportamento rispecchia il quinto principio S.O.L.I.D.<sup>48</sup>.

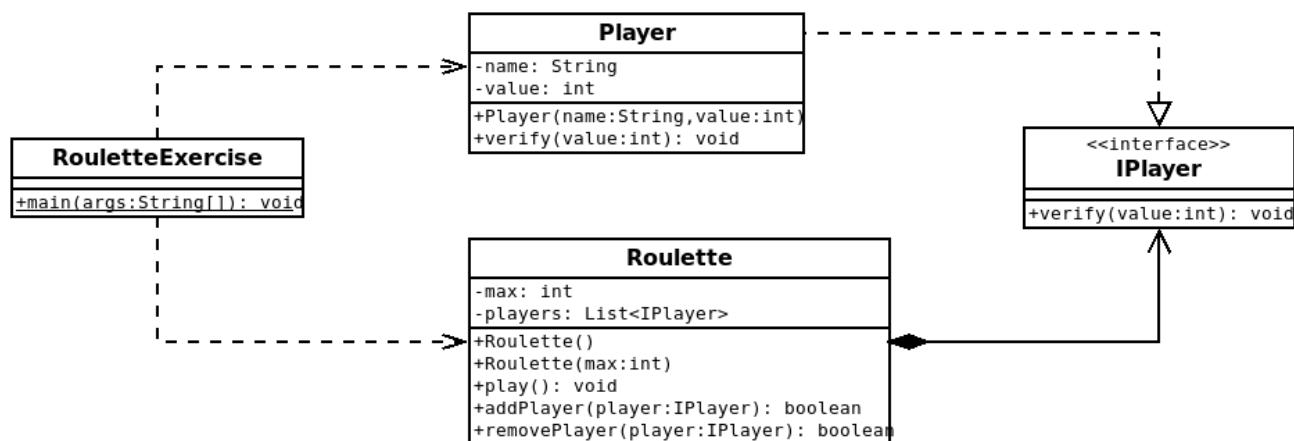
Una possibile esecuzione che prevede inizialmente quattro giocatori, poi ridotti a due, produce il seguente risultato:

```

A non hai vinto
B hai vinto
C non hai vinto
D non hai vinto
A non hai vinto
B hai vinto

```

Com'è possibile vedere dal diagramma delle classi di seguito riportato, la logica è sempre la stessa.



<sup>48</sup> Dependency Inversion Principle of [The S.O.L.I.D. Principles](#) by [Ugonna Thelma](#). If necessary you can also read these: [A Solid Guide to SOLID Principles](#) by [Baeldung](#)

Di seguito viene riportato il codice che implementa il progetto<sup>49</sup>.

### RouletteExercise.java

```
public class RouletteExercise {
    public static void main(String[] args) {
        Roulette roulette = new Roulette(3);
        IPlayer p1 = new Player("A", 0);
        roulette.addPlayer(p1);
        IPlayer p2 = new Player("B", 1);
        roulette.addPlayer(p2);
        IPlayer p3 = new Player("C", 2);
        roulette.addPlayer(p3);
        IPlayer p4 = new Player("D", 3);
        roulette.addPlayer(p4);
        roulette.play();
        roulette.removePlayer(p4);
        roulette.removePlayer(p3);
        roulette.play();
    }
}
```

### Roulette.java

```
import java.util.ArrayList;
import java.util.List;

public class Roulette {
    private int max;
    private List<IPlayer> players;

    public Roulette(){
        this(36);
    }

    public Roulette(int max){
        this.max = max;
        players = new ArrayList();
    }

    public void play(){
        int value = (int)(Math.random()*max);
        for(IPlayer player : players){
            player.verify(value);
        }
    }

    public boolean addPlayer(IPlayer player) {
        return players.add(player);
    }
}
```

---

<sup>49</sup> Il codice è stato sviluppato dal Prof. Bellucci.

```
public boolean removePlayer(IPlayer player) {  
    return players.remove(player);  
}  
}
```

### IPlayer.java

```
public interface IPlayer {  
    public void verify(int value);  
}
```

### Player.java

```
public class Player implements IPlayer {  
    private String name;  
    private int value;  
  
    public Player(String name, int value) {  
        this.name = name;  
        this.value = value;  
    }  
  
    @Override  
    public void verify(int value) {  
        String message = name + " non hai vinto";  
        if(this.value==value){  
            message = name + " hai vinto";  
        }  
        System.out.println(message);  
    }  
}
```

## Bank Account

In questo esercizio l'entità osservata è la classe **BankAccount**, mentre le classi che osservano e che useranno il dato generato da **BankAccount** sono le classi **Mail** e **SMS**.

La classe **BankAccount** simula un conto corrente bancario e i correntisti possono effettuare operazioni di prelievo di denaro (`withdraw()`) e di deposito (`deposit()`). Nel caso in cui un prelievo porti ad un ammontare di denaro inferiore a 100, vengono inviati dei messaggi via mail e tramite SMS per avvisare il correntista del raggiungimento di questo livello di conto considerato critico.

La classe **BankAccount** aggiunge gli osservatori **Mail** e **SMS** tramite il metodo `add()`, e li rimuove usando il metodo `remove()`, e notifica i cambiamenti agli osservatori tramite il metodo `send()`. I cambiamenti da notificare sono relativi alle operazioni effettuate e al raggiungimento di un livello monetario nel conto corrente inferiore a 100 a seguito di un prelievo. Le due classi **Mail** e **SMS** rappresentano due modalità distinte di notifica che implementano concretamente il metodo `send()` dichiarato nell'interfaccia **IMessaging**. Questo comportamento rispecchia il quinto principio S.O.L.I.D.<sup>50</sup>.

---

<sup>50</sup> Dependency Inversion Principle of [The S.O.L.I.D. Principles](#) by [Ugonna Thelma](#). If necessary you can also read these: [A Solid Guide to SOLID Principles](#) by [Baeldung](#)

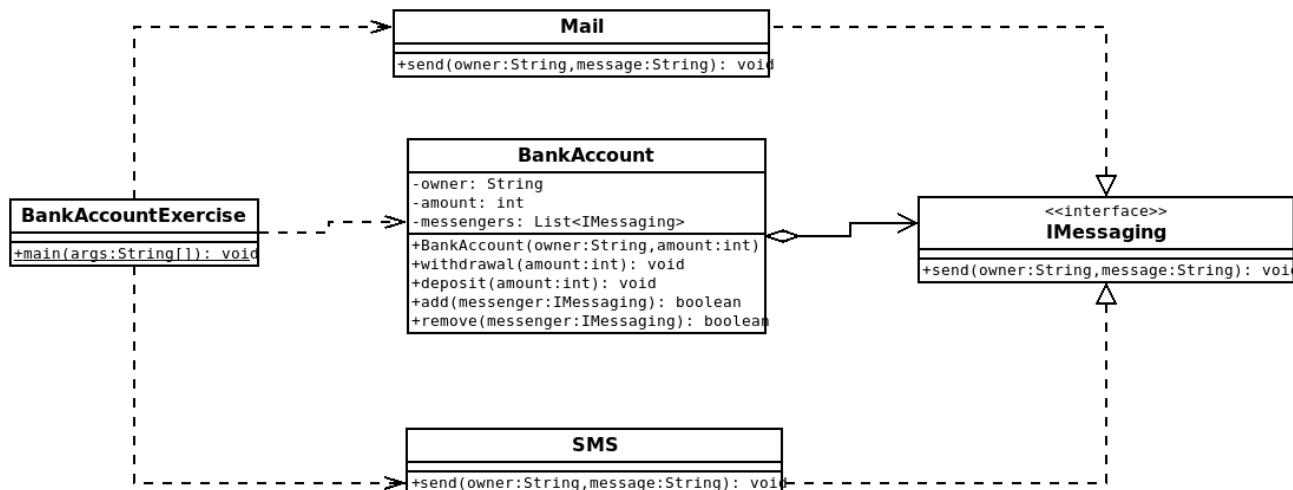
Di seguito viene mostrato il comportamento di un conto corrente bancario intestato all'utente Spendthrift che parte da un saldo pari a 500 e prevede notifiche sia tramite e-mail, sia tramite SMS. Dopo aver prelevato 401 il correntista riceve le notifiche con la segnalazione di aver superato il livello di soglia. Il correntista annulla la notifica tramite e-mail ed effettua un deposito di 150, ricevendo una notifica solo tramite SMS.

MAIL - Spendthrift you have withdrawn 401, and your money account is lower than 100.

SMS - Spendthrift you have withdrawn 401, and your money account is lower than 100.

SMS - Spendthrift You have deposited 150.

Com'è possibile vedere dal diagramma delle classi di seguito riportato, la logica è sempre la stessa.



Di seguito viene riportato il codice che implementa il progetto<sup>51</sup>.

### **BankAccountExercise.java**

```

public class BankAccountExercise {
    public static void main(String[] args) {
        BankAccount bankAccount = new BankAccount("Spendthrift", 500);
        Mail mail = new Mail();
        bankAccount.add(mail);
        bankAccount.add(new SMS());
        bankAccount.withdrawal(401);
        bankAccount.remove(mail);
        bankAccount.deposit(150);
    }
}
  
```

### **BankAccount.java**

```

import java.util.ArrayList;
import java.util.List;

public class BankAccount {
  
```

<sup>51</sup> Il codice è stato sviluppato dal Prof. Bellucci.

```
private String owner;
private int amount;
private List<IMessaging> messengers;

public BankAccount(String owner, int amount) {
    this.owner = owner;
    this.amount = amount;
    messengers = new ArrayList<>();
}

public void withdrawal(int amount) {
    String message;
    this.amount -= amount;
    message = " you have withdrawn " + amount;
    if(this.amount < 100) {
        message += ", and your money account is lower than 100.";
    }
    for(IMessaging messenger: messengers) {
        messenger.send(owner, message);
    }
}

public void deposit(int amount) {
    String message;
    this.amount += amount;
    message = "You have deposited " + amount + ".";
    for(IMessaging messenger: messengers) {
        messenger.send(owner, message);
    }
}

public boolean add(IMessaging messenger) {
    return messengers.add(messenger);
}

public boolean remove(IMessaging messenger) {
    return messengers.remove(messenger);
}
```

## IMessaging.java

```
public interface IMessaging {
    void send(String owner, String message);
}
```

## Mail.java

```
public class Mail implements IMessaging {
    @Override
    public void send(String owner, String message) {
        System.out.println("MAIL - " + owner + message);
    }
}
```

```
    }  
}
```

## SMS.java

```
public class SMS implements IMessaging {  
    @Override  
    public void send(String owner, String message) {  
        System.out.println("SMS - " + owner + message);  
    }  
}
```

## Libreria di comunicazione per protocolli

# C - CLIL - Introduction to Distributed system

# Unit 1 - Transport Protocol

## Vocabulary

Match the words (1-3) with the correct definition (A-C).

1	UDP	A	is a numbered logical construct allocated specifically for each of the communication channels an application needs. For many types of services, they have been standardized so that client computers may address specific services of a server computer without the involvement of service announcements or directory services.
2	TCP	B	uses a connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network and so there is no guarantee of delivery, ordering, or duplicate protection.
3	Port	C	provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating by an IP network.

# Language for thinking: defining

This list contains language for expressing thinking processes which learners are required to engage in lessons. Each category is divided into question which teachers ask learners, and statements. Some of the items are too formal to use with young children: they are in italics.<sup>52</sup>

## Teacher questions

What is a...?

Give me definition of a...

*How would you define a...?*

Who can define/give me a definition of...?

Can anyone give me a definition of...?

What do we call this?

What is the name/(technical) *term* for this?

## Statements

(A)	.....	is a	(generic term) place person thing <i>concept</i> <i>entity</i> <i>device</i> <i>instrument</i> <i>tool</i> etc	where who which that	.....
				for	...-ing ...

... is called ...

The *term/name* for this is...

We call this...

---

<sup>52</sup> Barbero T., Clegg J., *Programmare Percorsi CLIL*, Carocci, Roma 2005

# Anticipation guide

Try to respond to the following questions about the Transport Protocol before watching the video, and try again after you will have watched it.

## **Transport Protocol characteristics**

- What is the main purpose of the Transport Protocol?
- What are the main characteristics of UDP?
- What are the main characteristics of TCP?
- What is a port in the Transport Protocol?
- What kind of applications use TCP?
- What kind of applications use UDP?

# Comparison of Transport Protocols: UDP and TCP

## UDP and TCP

### Comparison of Transport Protocols

<https://youtu.be/Vdc8TCESIg8>

**UDP and TCP: Comparison of Transport Protocols – [PieterExplainsTech](#)**

The transport layer provides multiple applications to use one network connection simultaneously, creating 65,536 **ports** on the computer that are reserved and used by applications. An **application can use multiple ports at the same time**, and they are used to **multiplex and demultiplex** the different messages sent and received by the communication nodes.

The two transport protocols differ heavily:

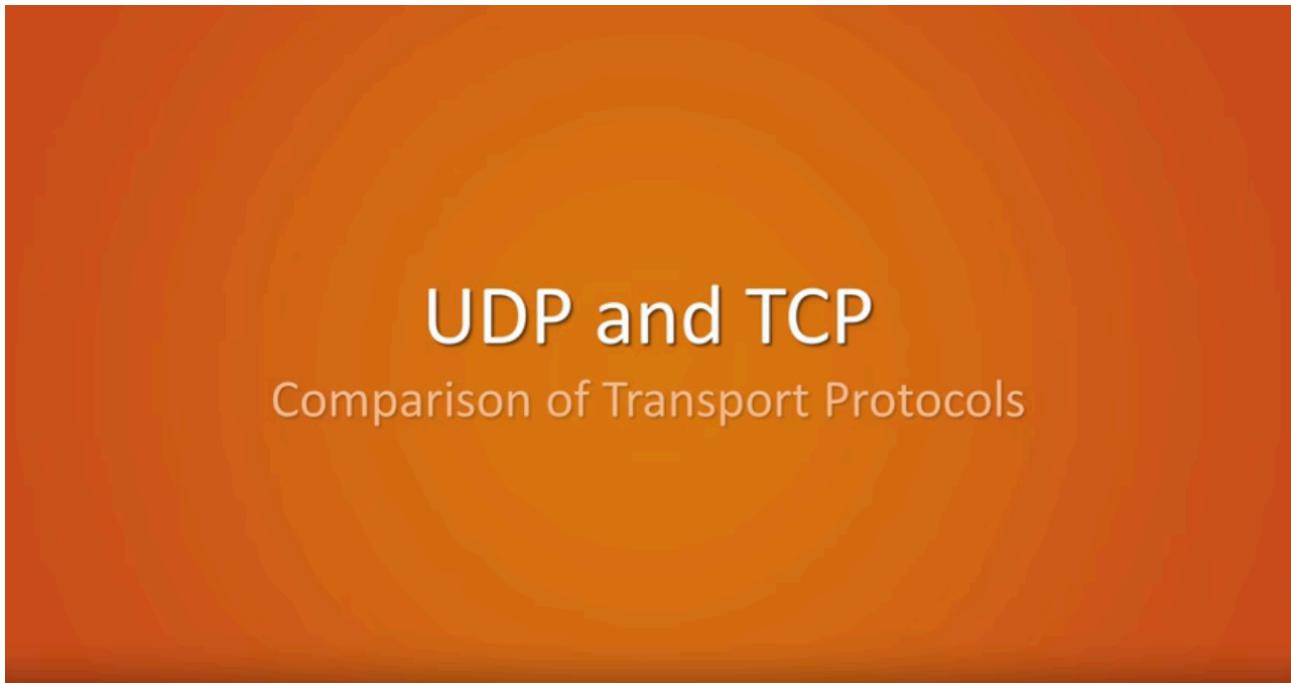
- **UDP is connectionless** and does not create a connection before sending out of data. Instead, **TCP is connection-oriented** and the two nodes have to create a connection in order to communicate.
- UDP **header** size is smaller than the TCP one.
- UDP does not try to recover corrupted datagrams, and does not guarantee in order datagrams delivery and congestion control. Instead, **TCP is more reliable than UDP** because the two nodes have to negotiate a connection using the three-way handshake. TCP offers also segments retransmission, implements in order messages delivery, and include congestion control.
- **UDP is a message oriented protocol** which means that applications send their data using independently datagram. **TCP** on the other hand **is a stream oriented protocol** that uses a continuous flow of data, and applications do not need to know how data they sent are sliced into segment and recomposed on the other hand.

When an application is designed, it is necessary to decide which protocol fits better its requirements. For example, TCP is appropriate for a text communication application or a file download application because it is guaranteed an ordered delivery and retransmission

of missing or corrupted segments. UDP is preferred when it is important to reduce the transmission overhead or to reduce the bandwidth occupation, and it is tolerated some amount of packets loss, like small question and answer transactions such as DNS lookups.

## Activity: Comparison of Transport Protocols

Watch the video below and create a [concept map](#)<sup>53</sup>, using [Cmap](#), to explain the main characteristics of the Transport Protocol. Send to your teacher your final product or the link to it.



<https://youtu.be/Vdc8TCESIq8>

***UDP and TCP: Comparison of Transport Protocols – [PieterExplainsTech](#)***

---

<sup>53</sup> See also [Mappe mentali, concettuali, cognitive](#)

# Unit 2 - Interprocess communication - Sockets

## Vocabulary

Use the following table to write as many different words you know about **Transport protocols, sockets** and **interprocess communication**. You have **two minutes** to write down all the words you know and each word has to start with the corresponding letter of the list. After you finished the list, compare and discuss it with your classmates.

Words that start with...	
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	
K	
L	
M	
N	
O	
P	
Q	
R	
S	
T	
U	

V	
W	
X	
Y	
Z	

# Language for thinking: linking words

You need some simple linking words and natural phrases to communicate your ideas.

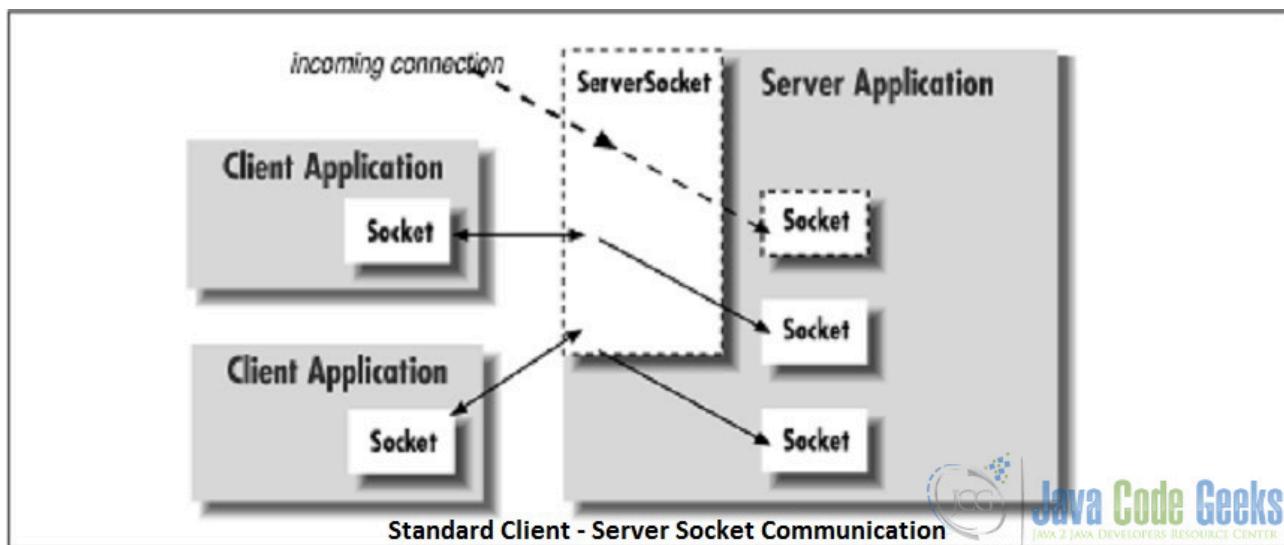
<b>Adding more information</b>	<b>Time phrases now</b>	<b>Causes and solutions</b>
and also as well as another reason is In addition / additionally / an additional Furthermore	at the moment at present right now these days nowadays in the past before then at that time years ago when I was younger	I guess it's because The main reason is It was caused by Because (of) I suppose the best way to deal with this problem is I reckon the only answer is to The best way to solve this is For Since As
<b>Expressing ideas</b>	<b>Examples</b>	<b>Being clear</b>
I think one important thing is I guess one difference is I suppose the main difference between X and Y is	for example for instance such as like That is (ie) Including Namely	What I mean is What I want to say is As I was saying
<b>Contrasting and concessions</b>	<b>Sequence</b>	<b>Result</b>
but on the other hand while or However Nevertheless Nonetheless Still Although / even though Though Yet Despite / in spite of In contrast (to) / in comparison Whereas On the contrary	First / firstly, second / secondly, third / thirdly etc Next, last, finally In addition, moreover Further / furthermore Another Also In conclusion To summarise	So As a result As a consequence (of) Therefore Thus Consequently Hence Due to
<b>Emphasise</b>	<b>Comparison</b>	

Undoubtedly Indeed Obviously Generally Admittedly In fact Particularly / in particular Especially Clearly Importantly	Similarly Likewise Also Like Just as Just like Similar to Same as Compare compare(d) to / with Not only...but also	
--	--	--

# Interprocess communication characteristics

This unit is concerned with some communication aspects of **middleware**. The previous unit topic was the Internet transport level protocols UDP and TCP. In that unit was said anything about how middleware and application programs could use these protocols. This unit introduces some characteristics of interprocess communication and UDP and TCP are discussed from a programmer's point of view.

When a **server accepts a connection**, it generally **creates a new thread** in which to communicate with the new client. The **advantage** of using a separate thread for each client is that the **server can block its thread** when waiting for input **without delaying other clients**.



The **application program interface (API) to UDP** provides a **message** passing abstraction that enables a sending process to transmit a single message to a receiving process. The independent packets containing these messages are called **datagrams**.

The **application program interface (API) to TCP** provides the abstraction of a **two-way stream** between pairs of processes. The information communicated consists of a stream of data items with no message boundaries. **Streams** provide a building block for **producer-consumer communication** (remember what you have studied last year).

Message passing between a pair of processes can be supported by two message communication operations, **send** and **receive**. To communicate, **one process sends a message** (a sequence of bytes) to a destination and **another process receives the message**. This activity involves the communication of data from the sending process to the receiving process and may involve the **synchronization of the two processes**.

## Synchronous and asynchronous communication

A **queue** is associated with each message destination. **Sending processes** cause **messages** to be **added** to **remote queues** and **receiving processes remove messages** from **local queues**. **Communication** between the sending and receiving

processes **may be either synchronous or asynchronous**. In the **synchronous** form of communication, the sending and receiving processes synchronize at every message. In this case, **both send and receive are blocking** operations. Whenever a send is issued the sending process (or thread) is blocked until the corresponding receive is issued. Whenever a receive is issued by a process (or thread), it blocks until a message arrives.

In the **asynchronous** form of communication, the use of the **send** operation is **non-blocking** in that the sending process is allowed to proceed as soon as the message has been copied to a local buffer. The **receive** operation can have **blocking and non-blocking variants**. The **most used** variant is the **blocking** one because in modern multi-threading system environment such as Java, a single thread can manage the receiving operation, and in the meanwhile other threads of the same process can provide for other executions.

## Transport layer and interprocess communication



<https://youtu.be/hi9BVTNvl4c>

### **Computer Networks 6-1: Transport Layer Overview**

The **transport layer builds on the services of the network layer**, and it delivers data across the network to applications with **different kinds of reliability or quality** depending on the transport service model.

The transport layer is **communicating information from host to host** across the network, without any examination by intermediate devices inside the network, like routers or switches. The transport layer is using the network simply to get information between hosts.

The **transport service could provide either reliable or unreliable transport**. By **reliable service** a PDU will be delivered to the other side, and the **transport service takes care of fixing any lost packets**. By **unreliable service PDU can be lost** in the network and it will be exposed to the transport user.

There are also a distinction in the unit of data that is transferred: it can be transferred an **individual message**, or a **reliable bi-directional stream of bytes**. The **first** one is provided by **UDP**, and the **second** one is provided by **TCP**.

Based on the kind of services are used by the application, it is possible to compare the features of these two different transport services:

<b>TCP (Stream)</b>	<b>UDP (Datagram)</b>
Connections	Datagrams
Bytes are delivered once, reliably, and in order	Messages may be lost, reordered, duplicated

Arbitrary length content	Limited message size
Flow control matches sender to receiver	Can send regardless of receiver state
Congestion control matches sender to network	Can send regardless of network state

As you can see from the table above, **TCP is a full-featured protocol**, in fact **it does a lot on top of the network layer** to provide a service for applications. On the other hand **UDP is just a glorified packet transport**, because **it does very little over IP**.

The **interface between the transport layer and the application layer are the socket APIs**, and a socket is a simple abstraction to use the network. **Socket APIs support** both the Internet transport services: the **TCP service** is also known as a **connection-oriented service**, and the **UDP service** is known as a **connectionless service**.

**Sockets let applications attach to the local network using ports. Different ports** let multiple applications use the same transport layer instance on a given machine, in order to **multiplex the network among different applications**.

The Berkeley APIs used for streams and datagrams are illustrated in the following table:

Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address (port) with a socket
LISTEN	Announce willingness to accept connections
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND(TO)	Send some data over the socket
RECEIVE(FROM)	Receive some data over the socket
CLOSE	Release the socket

The **LISTEN, ACCEPT and CONNECT APIs are needed only for stream communication**, and the **TO/FORM forms are used for datagram communication**.

An **application process is identified by the tuple: IP address, protocol** (TCP or UDP), and **port**. **Ports are 16-bits integers leased by applications**. Clients and servers use ports slightly differently. **Servers** are often **bound to “well-known” ports**, which **numbers are less than 1024** and require **administrative privileges to be used**. On the other hand, **clients can use temporarily any other ports, normally chosen by OS, and known as “ephemeral” ports**.

Some “**well-known” ports** are showed by the following table:

Port	Protocol	Use

20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

## Activity: Interprocess communication and Transport layer

After you have watched the video below and read the paragraph *Interprocess communication characteristics*, create a [concept map](#)<sup>54</sup> or improve the last you have already done, to summarize the main concepts you have learnt about Interprocess communication. Use [Cmap](#) to create the concept map and send to your teacher your final product or the link to it.



<https://youtu.be/hi9BVTNvl4c>

### **Computer Networks 6-1: Transport Layer Overview**

Create your concept map answering the following questions:

1. Which is the main purpose of the Transport Protocol?
2. Which are the main characteristics of UDP?
3. Which are the main characteristics of TCP?
4. Which applications use TCP?
5. Which applications use UDP?
6. Are the network devices involved in transport layer communication?
7. Which data unit is transferred by UDP?
8. Which data unit is transferred by TCP?
9. What are socket APIs?
10. What is a port in the Transport Protocol?
11. Which are the Berkeley APIs used for stream communication? Explain their meaning.
12. Which are the Berkeley APIs used for datagram communication? Explain their meaning.
13. Which are the elements that identify a communication tuple? Explain their meaning.
14. What are the “well-known” ports?

---

<sup>54</sup> See also [Mappe mentali, concettuali, cognitive](#)

15. What are the “ephemeral” ports?
16. Which is the server advantage of using separate threads accepting connection requests?
17. Which are the characteristics of a synchronous communication?
18. Which are the characteristics of an asynchronous communication?

# TCP stream communication



<https://youtu.be/WyrxZB7Mbs4>

## **Computer Networks 1-4: Sockets**

Applications are attached to the hosts that are using the network, and they use an interface to talk over the network. This interface defines how applications can use the network.

The purpose of using APIs is to let apps to talk to other apps via the local host. APIs allow applications to communicate each other, hiding all the details of the network.

In a client-server architecture, there is a client that sends a request to the server over the network. A client is an app or a program running on the client host. The server receives the client request and responds with a replay. A server is an app or a program running on the server host. This simple pattern is the basis of many different application which are used in the Internet, for example:

- File transfer: client sends the name of file, server responses sending the file.
- Web browsing: client sends the URL of the page it needs, server responses sending the page.
- Echo: client sends a message, server responses sending the same message back to the client (useful for test functionality).

In order to write applications like the previous ones it is necessary to use Socket APIs. Socket APIs provide a simple abstraction to use the network, and they are used to write all Internet applications.

Sockets were devised around 1983, and were part of the one of the early Berkeley UNIX operating system releases. Since then sockets were present in all the major operating systems, and all major programming languages offer libraries to access them.

Socket APIs provide two different kinds of network service. The first one is a reliable byte stream service, that allows one application to send reliably a stream of bytes to another application, and vice versa.

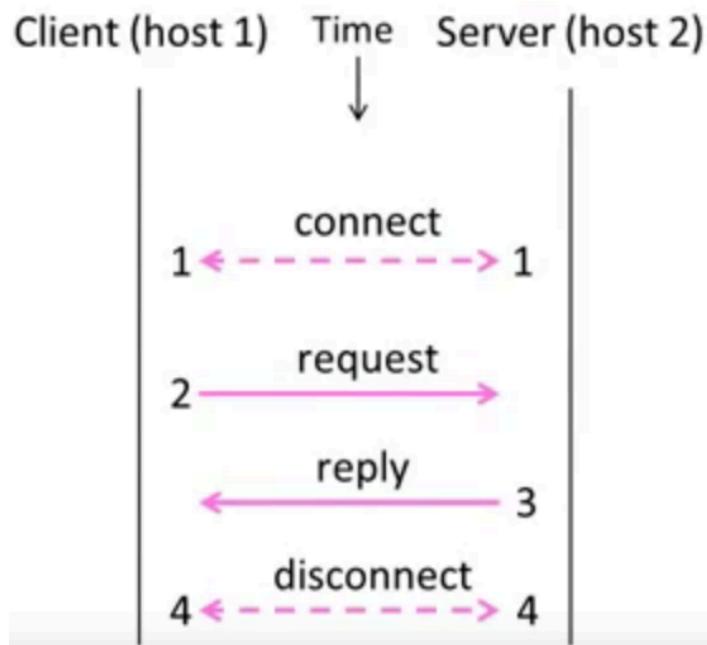
The other kind of service that is provided by Socket APIs is an unreliable datagram service that allows an application to send a message to another application.

Sockets use a data structure called socket to let an application to attach to the network. Sockets also use port numbers that provide applications addressing, and allow multiplexing of different applications on a single host.

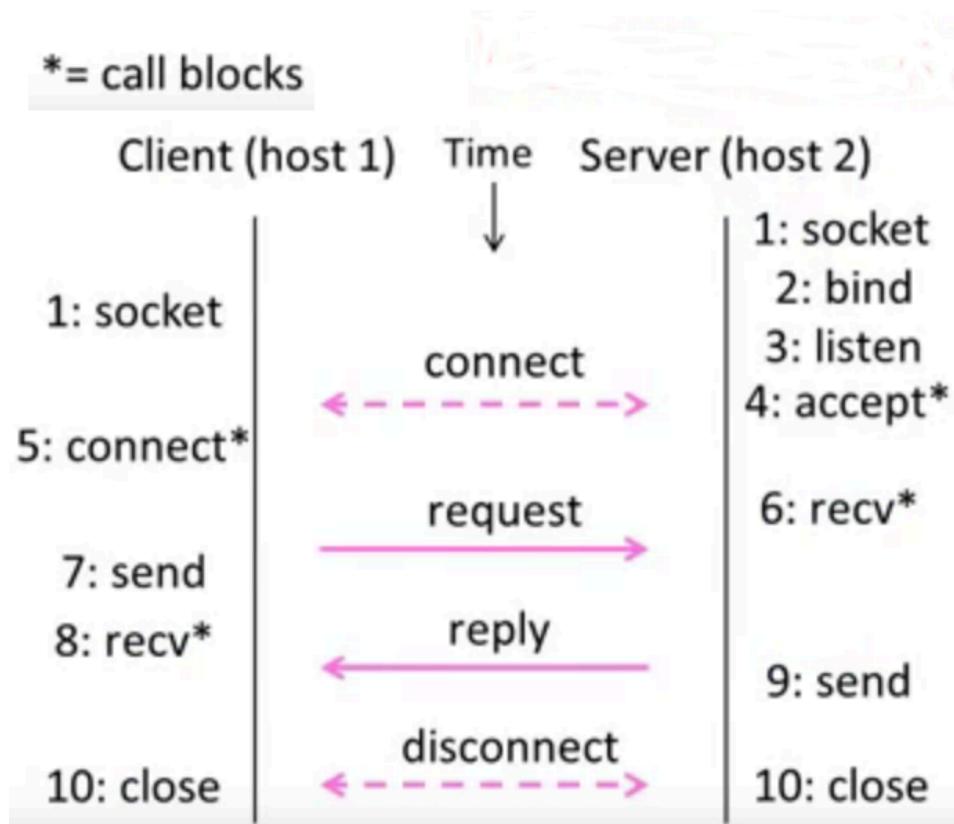
The following table shows all the main APIs used to work with socket:

Primitive	Meaning
SOCKET	Create a new communication endpoint that an application can use to access the network.
BIND	Associate a local address with a socket.
LISTEN	Announce willingness to accept connection.
ACCEPT	Passively establish an incoming connection. BIND, LISTEN and ACCEPT are used by the incoming side to get ready and accept an incoming call.
CONNECT	Actively attempt to establish a connection to another side.
SEND	Send some data over the connection. Applications use it to send bytes reliably across the network to another application.
RECV	Receive some data from the connection. It allows the application to the other side of the connection to receive the bytes that came across the network from another application.
CLOSE	Release the connection when both sides are finished.

The following time diagram shows the sequence interaction between a client and a server when they are using Socket APIs in a reliable byte stream communication:

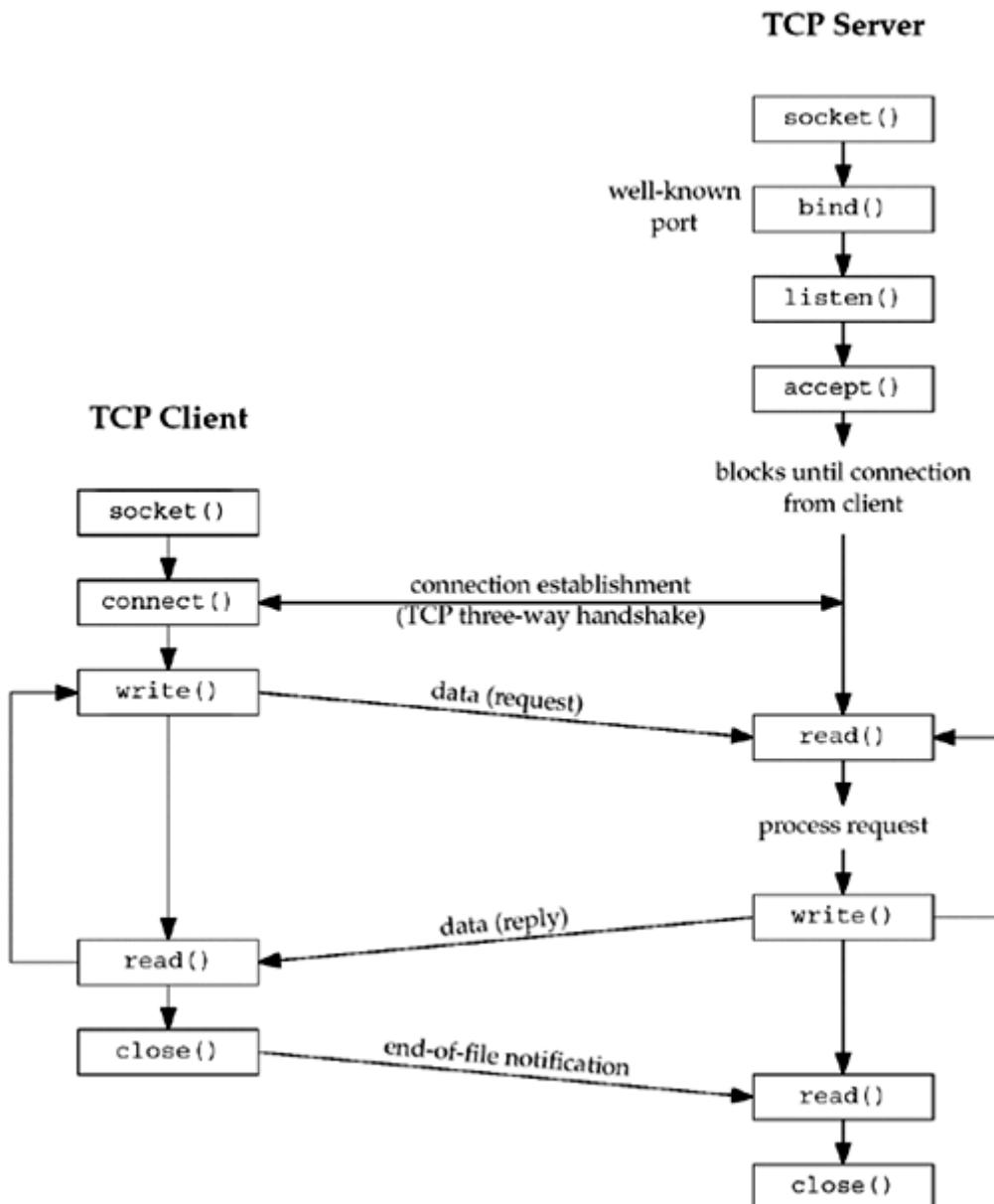


The following time diagram shows the different kinds of Socket APIs calls which are used to cause the previous sequence interaction happens:



All the calls marked with a \* are blocking calls. When the program makes one of these calls it has been halted by the operating system until something happens on the network side. When the network operation is completed and the program gets the result, it can continue.

This is another useful image to understand how reliable byte stream connections works:



## Activity: TCP stream communication

You will hear an explanation about TCP sockets and stream communication. After you have watched the video, complete the following exercises using what you have learnt by this explanation.



<https://youtu.be/WyrxZB7Mbs4>

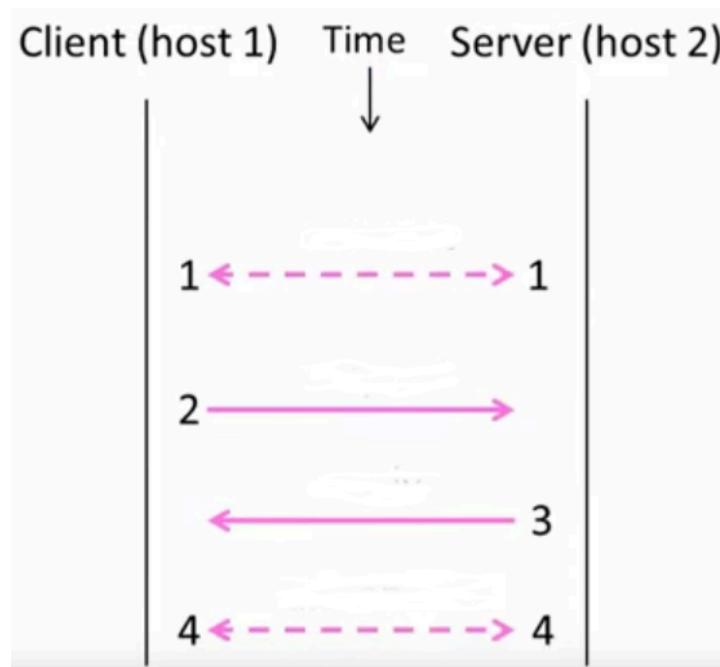
### **Computer Networks 1-4: Sockets**

1. Select all the applications that use TCP sockets:
  - a. DNS
  - b. Web browsing
  - c. DHCP
  - d. Voice-over-IP
  - e. File transfer
  - f. RPC
  - g. Echo
2. Given the following Berkeley APIs used for TCP stream communication, decide what Java class/method can be matched with each one or group of them. Specify also if the class/method is used by the server or the client.

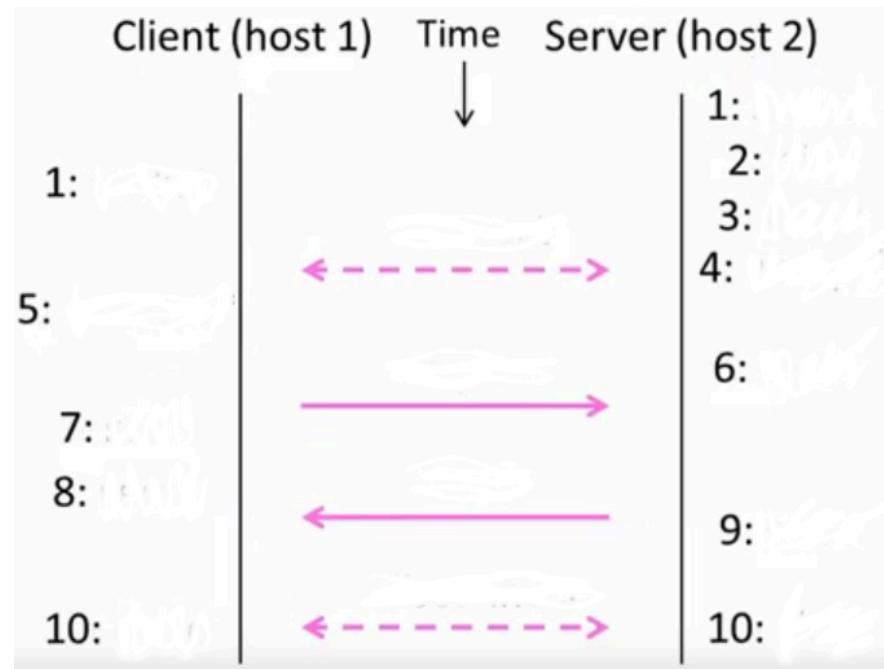
Primitive	Meaning	Java class/method
-----------	---------	-------------------

SOCKET	Create a new communication endpoint	
BIND	Associate a local address with a socket	
LISTEN	Announce willingness to accept connection	
ACCEPT	Passively establish an incoming connection	
CONNECT	Actively attempt to establish a connection	
SEND	Send some data over the connection	
RECV	Receive some data from the connection	
CLOSE	Release the connection	

3. Using the following image, put in the right order all the phases of a TCP communication: REPLY, DISCONNECT, CONNECT, REQUEST



4. After you have done the previous exercise, complete the following image using the given APIs. Put a \* near the blocking calls: RECV, SOCKET, CLOSE, RECVFROM, ACCEPT, SEND, LISTEN, SENDTO, CONNECT, BIND.  
When does three-way handshake happen?



# UDP datagram communication



<https://youtu.be/Z1HqgQJG0Fc>

## **Computer Networks 6-2: User Datagram Protocol (UDP)**

UDP provides a datagrams service and it is a glorified layer on top of IP. UDP simply allows packets (IP layer PDU) with datagrams (UDP layer PDU) to be delivered. It does not do a lot more than IP.

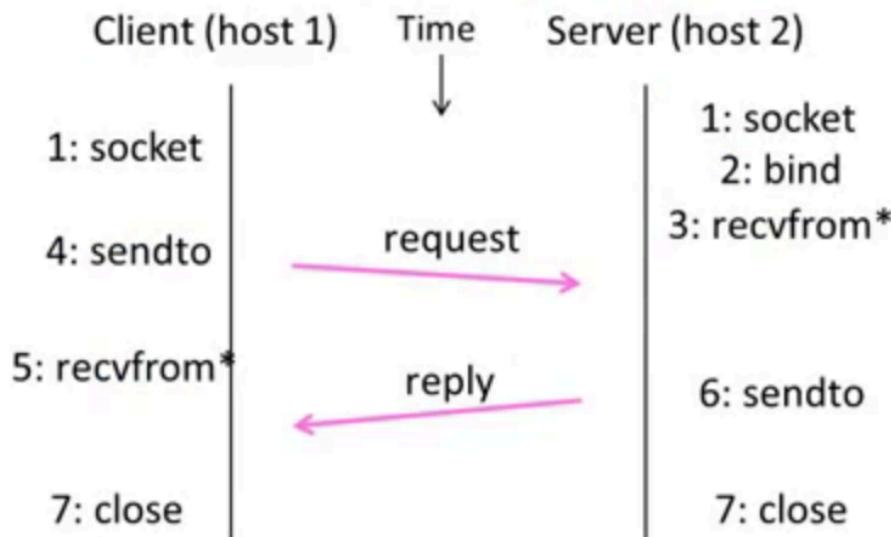
UDP is an unreliable protocol, and it is used by applications that do not want reliability or do not want some other kind of abstraction than messages, such as byte stream. Some examples of these kind of applications are:

- Voice-over-IP: used for real-time conferencing, it does not need reliability.
- DNS: maps hosts name, like www.peano.it, to IP addresses and it is a message oriented protocol.
- RPC: remote procedure call and it is a message oriented protocol.
- DHCP: a bootstrapping protocol used to get the network going. For this reason it must be simple, and UDP responds to this request better than TCP.

If an application want reliability and still want use messages, reliability will be an application responsibility, because reliable messages is something in between. For instance, DNS protocol must provide reliability itself as part of the application level protocol.

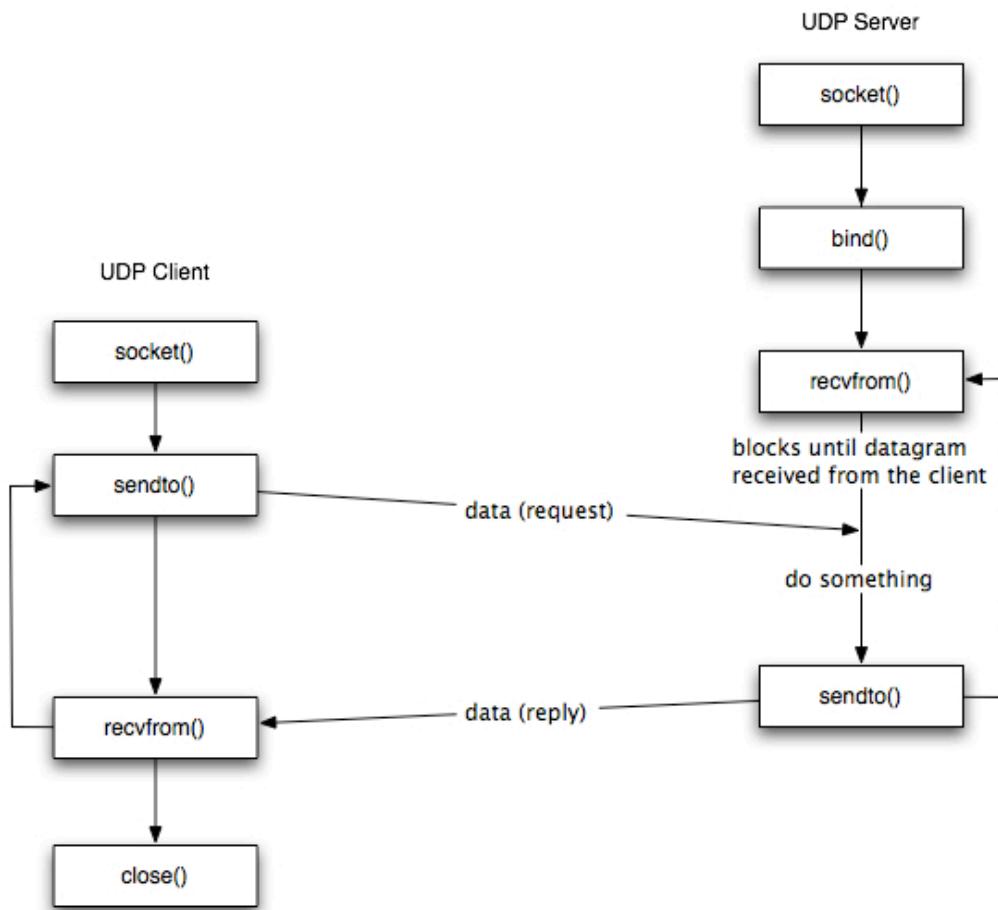
The following time diagram shows the different kinds of Socket APIs calls which are used between a client and a server when they are using Socket APIs in a datagram communication:

\*= call blocks

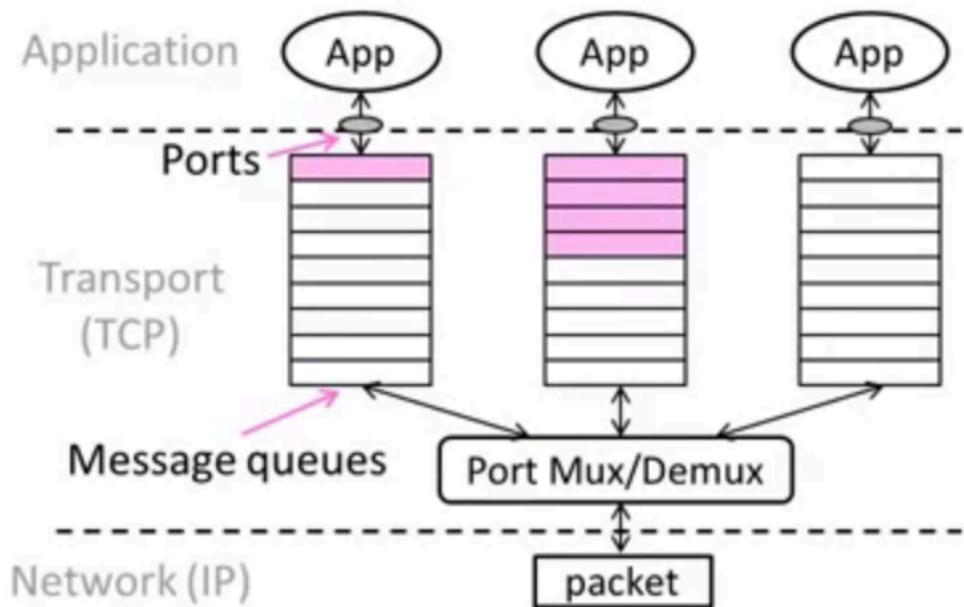


All the calls marked with a \* are blocking calls. When the program makes one of these calls it has been halted by the operating system until something happens on the network side. When the network operation is completed and the program gets the result, it can continue.

This is another useful image to understand how datagram communication works:



The following picture shows the buffering that happens inside the Transport layer:



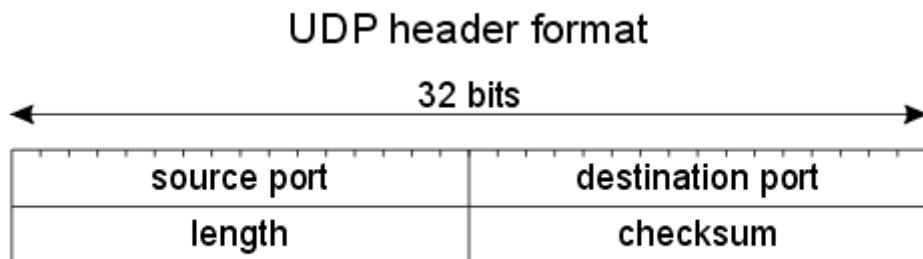
At the *Application layer* there are all the app processes running and they all use the socket support to interface to the Transport layer. At the *Network layer* there are packets sent by the *Physical layer* below.

When segments (TCP) or datagrams (UDP) arrive to a host, they are buffered inside the Transport layer. All the Transport PDUs wait inside the buffer until the app calls a *receive*. At this time the Transport PDU are pulled from the queue and handed up to the application.

When an application has data to send, they will go down to the queue where they temporarily could stay if the application can send data more quickly than the information could be put on the local network wire. In this case there will be a little bit of temporary buffering inside the queue.

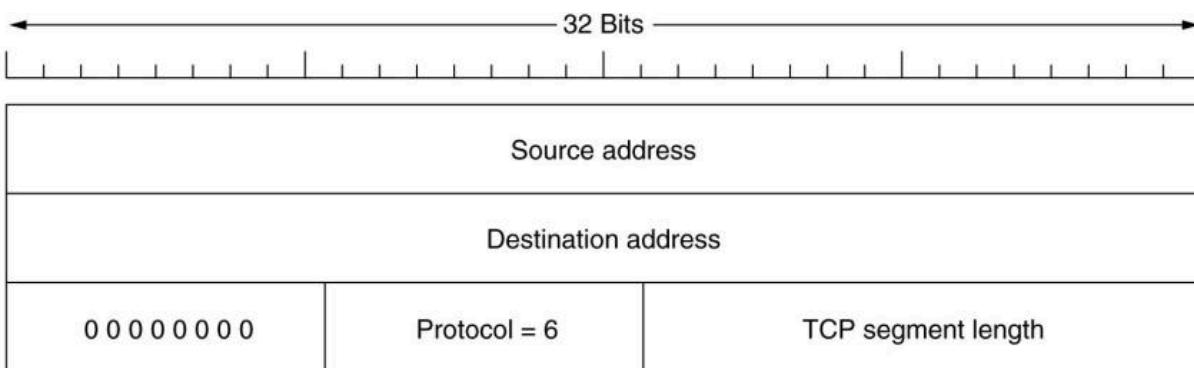
The *Port Mux/Demux* is the multiplexer and demultiplexer which uses the port information inside a Transport PDU to assign it to the right queue and vice versa, taking all Transport PDUs from the different queues and sending them into the network.

The UDP header is very simple, and it is showed by the following image:



It uses ports to identify sending and receiving application processes. There is a length field that is of 16 bits, so it is possible to send datagrams about to 64KB in length. There is also an optional 16 bits checksum used to add a little bit of reliability. The optional UDP

checksum does not cover only UDP datagram, it also covers some information taken by the IP header, called *IP pseudoheader*:



If the value in the checksum is zero, it will mean that there is no checksum. Instead, if the checksum comes out to be really zero, it will be considered in the form of all one, that is an equivalent value for zero.

## Activity: UDP datagram communication

You will hear an explanation about UDP sockets and datagram communication. After you have watched the video, complete the following exercises using what you have learnt by this explanation.



<https://youtu.be/Z1HqgQJG0Fc>

### **Computer Networks 6-2: User Datagram Protocol (UDP)**

1. Select all the applications that use UDP sockets:

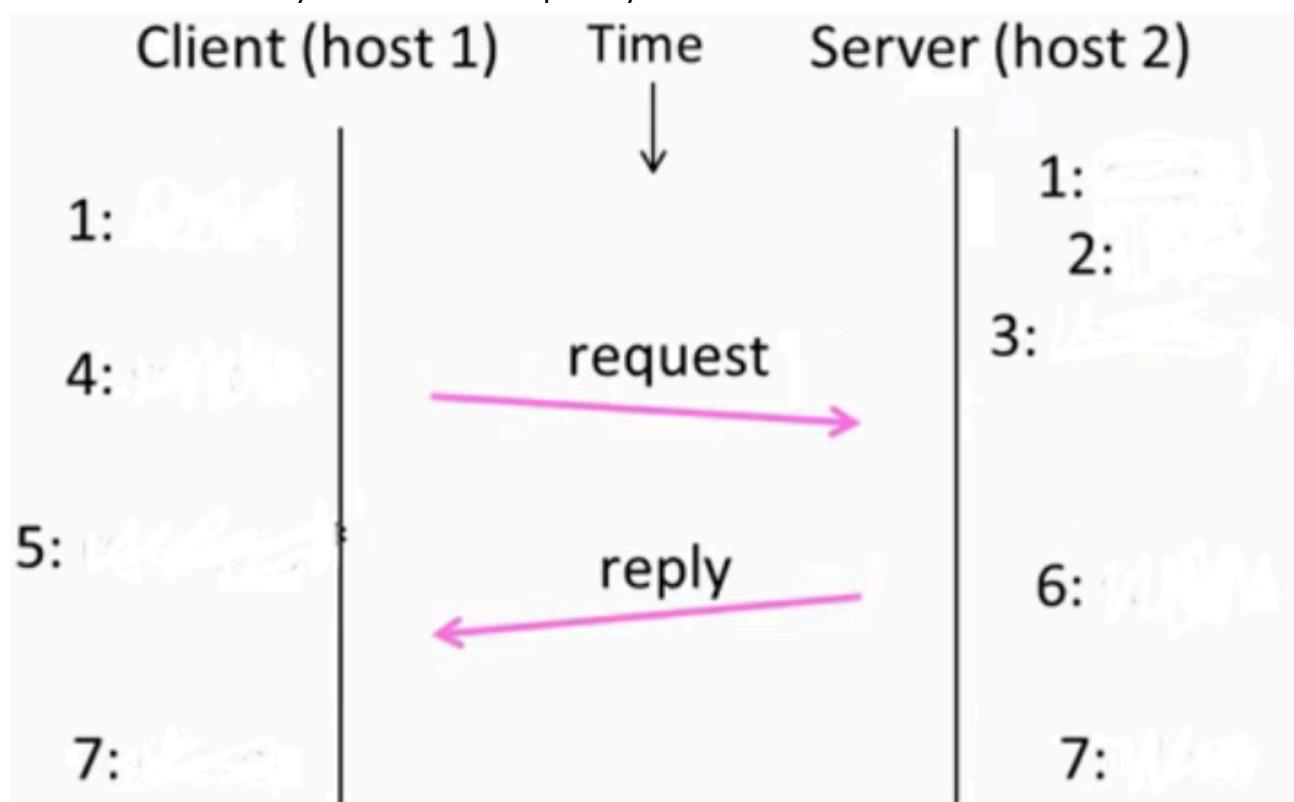
- a. DNS

- b. Web browsing
  - c. DHCP
  - d. Voice-over-IP
  - e. File transfer
  - f. RPC
  - g. Echo
2. Given the following Berkeley APIs used for UDP stream communication, decide what Java class/method can be matched with each one or group of them. Specify also if the class/method is used by the server or the client.

<b>Primitive</b>	<b>Meaning</b>	<b>Java class/method</b>
SOCKET	Create a new communication endpoint	
BIND	Associate a local address with a socket	
SENDTO	Send a message	
RECVFROM	Receive a message	
CLOSE	Release the resources used for communication	

3. Complete the following image using the given APIs. Put a \* near the blocking calls: RECV, SOCKET, CLOSE, RECVFROM, ACCEPT, SEND, LISTEN, SENDTO, CONNECT, BIND.

There is a three-way handshake? Explain your answer.



4. Complete the image of the UDP header and answer to the following question:

- a. Why there is an UDP checksum?

---

---

---

- b. What is the value used to set that there is no UDP checksum?

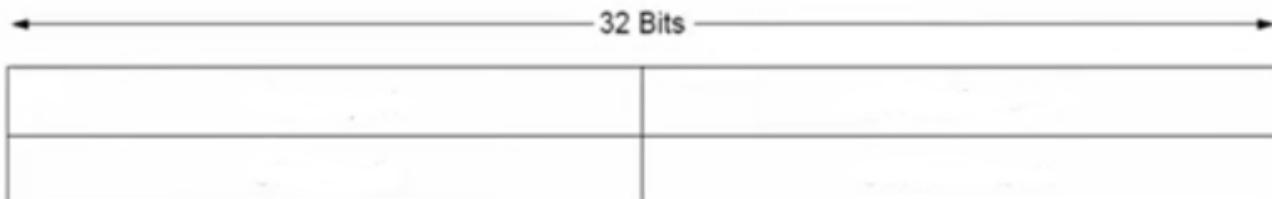
---

---

- c. What is the value used for a UDP checksum equal to zero?

---

---



## Activity: UDP and TCP Sockets

After you have watched the two videos below, create a concept map or improve the last you have already done, to summarize the main concepts you have learnt about UDP and TCP sockets. Use [Cmap](#) to create the [concept map](#)<sup>55</sup> and send to your teacher your final product or the link to it.



<https://youtu.be/WyrxZB7Mbs4>

**Computer Networks 1-4: Sockets**



<https://youtu.be/Z1HqgQJG0Fc>

**Computer Networks 6-2: User Datagram Protocol (UDP)**

---

<sup>55</sup> See also [Mappe mentali, concettuali, cognitive](#)

# CLIL Writing and Speaking - Networking, TCP and UDP protocols

Answer to the following questions.

1. What is an IP address?
2. Which is the purpose of a netmask?
3. What is the main purpose of the Transport Protocol?
4. What are the main characteristics of UDP?
5. What are the main characteristics of TCP?
6. What is a port in the Transport Protocol?
7. What kind of applications use TCP?
8. What kind of applications use UDP?
9. What is a protocol?
10. Which data unit is transferred by UDP?
11. Which data unit is transferred by TCP?
12. Which are the "well-known" ports?
13. Which are the "ephemeral" ports?
14. What is a socket?
15. Which is the server advantage of using separate threads accepting connection requests?
16. Which are the characteristics of a synchronous communication?
17. Which are the characteristics of an asynchronous communication?
18. Describe how packet switched network works.
19. Describe how circuit switched network works.
20. Compare packet switched and circuit switched network and give a description of the main differences.

# CLIL Writing and Speaking - Java sockets

Answer to the following questions.

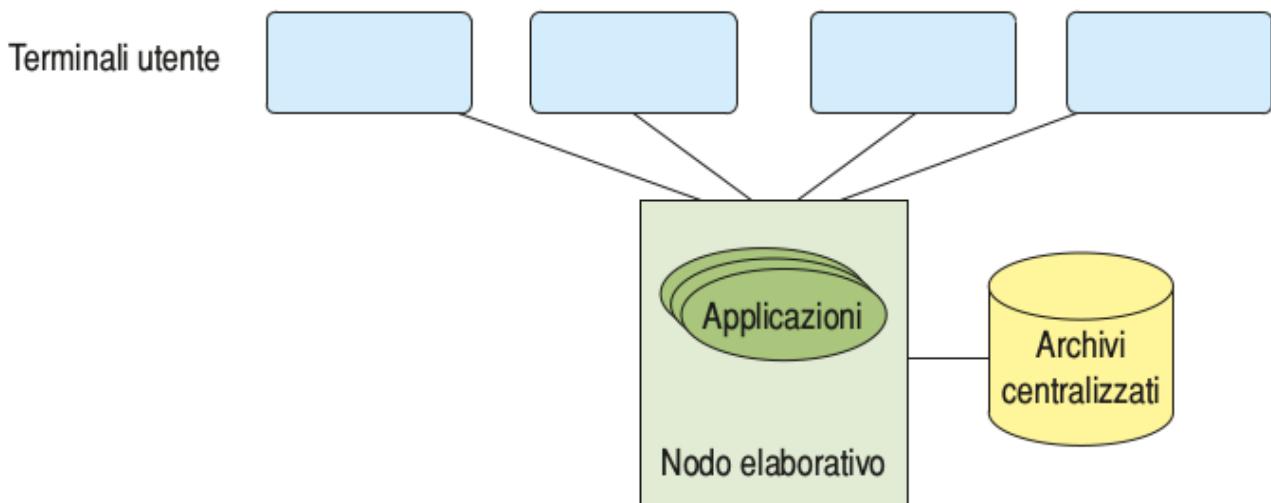
21. Describe the Java classes used to create a TCP socket.
22. Describe the Java classes used to create a UDP socket.
23. Watch [this video](#) and answer to the following questions.
  - a. In this video is not used a method that you used in all your TCP socket projects. Are you able to identify it?
  - b. The class Socket uses different constructors, [here](#) you can find all of them. Try to identify which of them requires the use of the method you found on question one.

## D - I sistemi distribuiti: modelli architetturali hardware e software

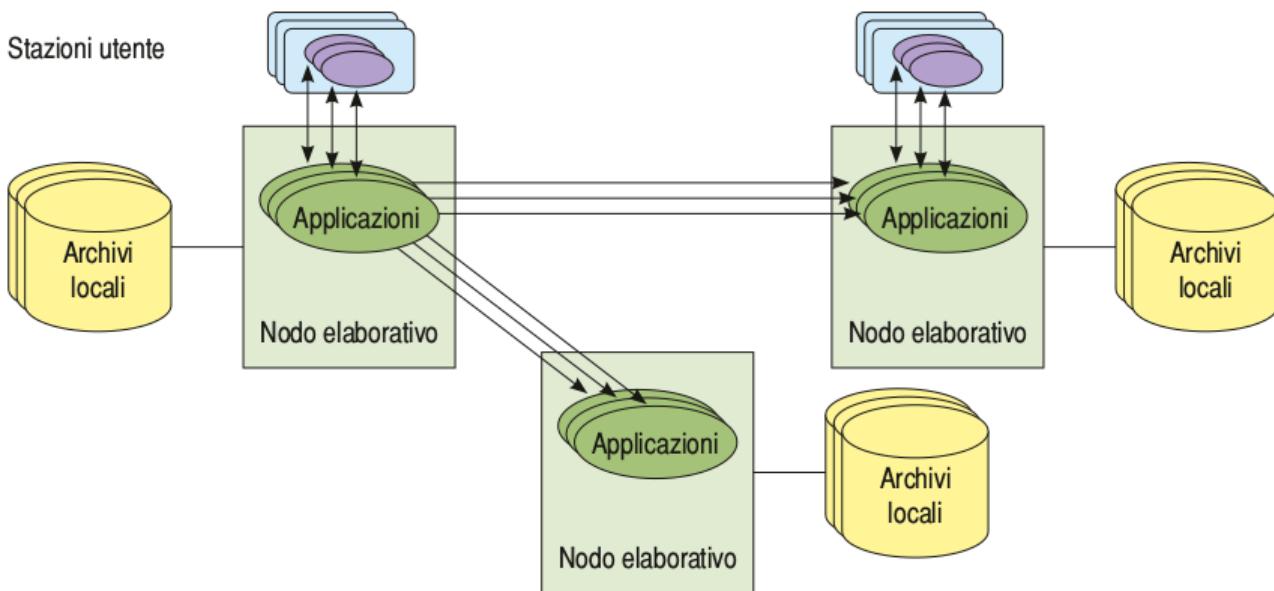
# I sistemi distribuiti

Le architetture dei sistemi informativi si sono sviluppate ed evolute nel corso degli anni passando da schemi centralizzati a modelli distribuiti, più aderenti alla necessità di decentralizzazione e cooperazione delle moderne organizzazioni.

Si parla di **sistema informatico centralizzato** quando i dati e le applicazioni risiedono in un unico nodo elaborativo.



Un **sistema informatico distribuito** invece è costituito da un insieme di applicazioni logicamente indipendenti che collaborano per il perseguitamento di obiettivi comuni attraverso una infrastruttura di comunicazione hardware e software.

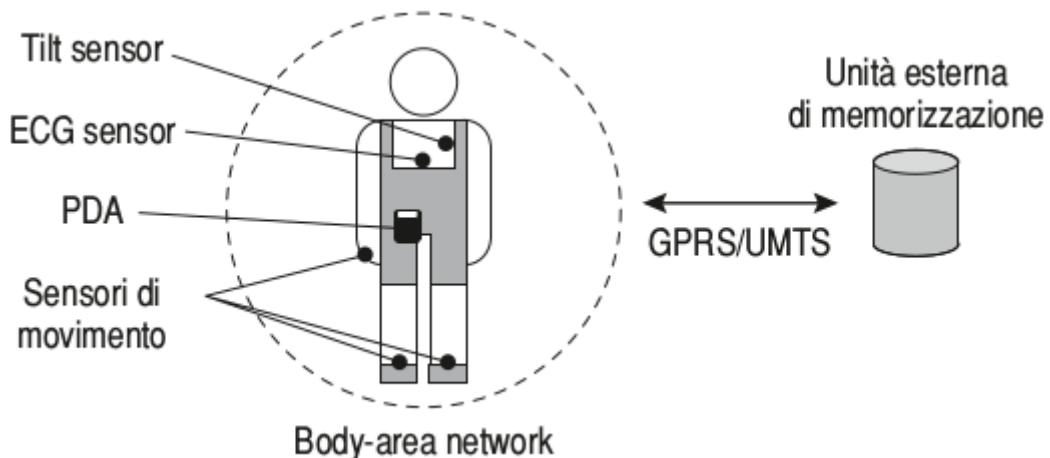


Le applicazioni che costituiscono un sistema distribuito hanno ruoli diversi all'interno del sistema stesso, e sono identificate nel seguente modo:

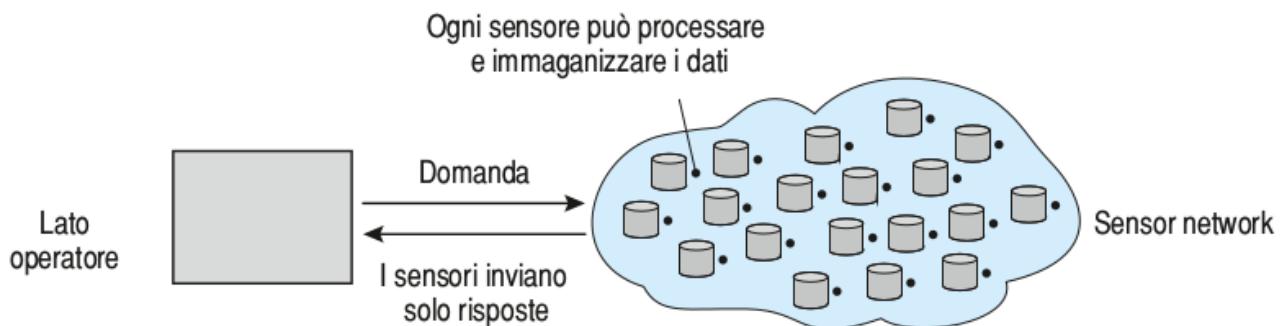
- **Client:** un'applicazione assume il ruolo di client quando è utilizzatore di servizi messi a disposizione da altre applicazioni;

- **Server:** un'applicazione assume il ruolo di server quando è fornitore di servizi usati da altre applicazioni;
- **Actor:** un'applicazione assume il ruolo di actor quando svolge sia il ruolo di client, sia quello di server in diverse situazioni nel contesto del sistema.

Alcuni esempi di sistemi distribuiti sono a livello macroscopico **Internet**, mentre a livello di singolo individuo può essere una **PAN** (*Personal Area network*) che fa riferimento al *wearable computing*,



o una **rete di sensori**.



## Caratteristiche dei sistemi distribuiti

I sistemi distribuiti presentano una serie di caratteristiche che possono essere interpretate sia come vantaggi, sia come svantaggi, ma indubbiamente, vista l'attuale affermazione dell'uso dell'architettura distribuita per tutti i grandi sistemi informatici, è evidente che gli svantaggi sono considerati come un prezzo adeguato da pagare per garantirsi i vantaggi che derivano dall'uso di un sistema distribuito.

### Affidabilità

Uno dei principali vantaggi dei sistemi distribuiti è l'**affidabilità** in quanto la **ridondanza dei sistemi** hardware e software, **quando prevista** ed implementata in modo sistematico, **permette** al sistema **di sopravvivere al guasto** di un suo componente, introducendo al limite un livello di inefficienza nei tempi di risposta.

Questa caratteristica è raggiungibile solo se vengono predisposti gli strumenti hardware e software in grado di intervenire automaticamente al verificarsi di situazioni indesiderate. Ad esempio, l'implementazione di algoritmi operativi che permettano alle entità non guaste di sostituire quella danneggiata per non interrompere il funzionamento del sistema sono un esempio di utilizzo di sistemi di backup che vengono attivati nel momento in cui l'elemento principale non sia più operativo.

Risulta però evidente che la **difficoltà di realizzazione e i costi** connessi di un simile sistema crescono con l'aumentare del livello di affidabilità che si vuole raggiungere, quindi non è sempre implementabile un sistema completamente affidabile.

## Integrazione

Un'altra importante caratteristica dei sistemi distribuiti è la capacità di **integrare componenti eterogenei tra loro**, sia per tipologia hardware (dai PC ai mainframe, dagli smartphone ai tablet), sia per sistema operativo.

Per garantire questa caratteristica è fondamentale che ogni componente si possa **interfacciare allo stesso modo** con il **sottosistema di comunicazione** del sistema distribuito, **indipendentemente dalle differenze hardware e di sistema operativo**. L'**interfaccia di comunicazione** gioca quindi un ruolo fondamentale per permettere di rendere trasparenti all'intera rete i componenti dello strato inferiore del sistema.

Un classico esempio di integrazione è la possibilità di connettere **dispositivi di nuova generazione** con **sistemi legacy**<sup>56</sup>, che di fatto utilizzano tecnologie obsolete e spesso incompatibili con quelle più recenti.

Altri esempi di **interfacce di comunicazione** che soddisfano il criterio di integrazione è la **tecnologia Ethernet** per la connessione di dispositivi di rete, anche prodotti da società diverse, o la **piattaforma Web** che permette a sistemi operativi e architetture hardware di interconnettersi e scambiarsi informazioni.

Un altro esempio di interfaccia volta al perseguitamento dell'integrazione è costituita dalle **API** (*Application Program Interface*) che rappresentano un insieme di procedure, utilizzabili dal programmatore, utili alla stesura di applicazioni di rete. Le **API** forniscono l'**interfaccia** tra l'applicazione e lo strato di trasporto realizzando quella che si chiama astrazione **tra hardware e programmazione**, svincolando in tal modo gli sviluppatori dalle problematiche di comunicazione e trasferimento dati che sono i compiti degli strati inferiori. In particolare i **socket API** mettono a disposizione del programmatore gli strumenti necessari a **sviluppare programmi di connessione** che implementano il protocollo di comunicazione desiderato.

Il **linguaggio XML** (eXtensible Markup Language) è un'altra **tecnologia che permette di integrare sistemi diversi** in quanto è stato creato appositamente per favorire lo scambio di informazioni nel Web e permettere un'agevole ed efficiente pubblicazione di dati complessi.

---

<sup>56</sup> I sistemi legacy (*legacy system*) identificano i sistemi obsoleti presenti in una azienda ereditati dalle prime fasi di informatizzazione della stessa. Spesso rivestono ruoli critici e possono anche avere dimensioni notevoli, ma nonostante la possibile lunga storia di interventi di manutenzione, non sono mai stati rimpiazzati perché sostanzialmente funzionano ancora e, probabilmente, l'attività svolta non è ancora stata sostituita da tecnologie più recenti.

In sostanza la caratteristica di **integrazione** è resa possibile dalla definizione di protocolli standard o sistemi architetturali de facto che favoriscono la **portabilità** di applicazioni fra sistemi operativi diversi, conservando la medesima interfaccia utente, e la **interoperabilità** tra componenti diversi.

## Trasparenza

Con il termine **trasparenza** si intende il concetto di **considerare il sistema distribuito non come un insieme di componenti ma come un sistema di elaborazione unico**.

Lo scopo è quello di rendere trasparente all'utente la presenza di un sistema distribuito composto da molteplici entità, anche fisicamente distanti fra loro e con configurazioni mutevoli nel tempo, in modo che abbia invece la percezione di utilizzare un singolo elaboratore.

L'ANSA nell'ISO 10746, *Reference Model of Open Distributed Processing*, identifica otto forme di trasparenza, delle quali le prime due risultano fra le più importanti da implementare nell'ambito un sistema distribuito:

- **trasparenza di accesso** (*access transparency*): permette l'accesso alle risorse locali e remote usando operazioni identiche e uniformi, nascondendo le differenze nella rappresentazione dei dati e nelle modalità di accesso;
- **trasparenza di locazione** (*location transparency*): permette l'accesso alle risorse utilizzando un identificatore indipendente dalla reale disposizione geografica della risorsa stessa, ad esempio utilizzando un URL;
- **trasparenza di concorrenza** (*concurrency transparency*): permette ai processi di operare in maniera concorrente per l'accesso ad una risorsa condivisa, lasciandola sempre in uno stato consistente implementando, per esempio, meccanismi di locking (semafori, monitor);
- **trasparenza di replicazione** (*replication transparency*): permette di aumentare l'affidabilità e le prestazioni del sistema effettuando duplicazioni delle risorse mantenendone sempre lo stesso nome, e senza che l'utente ne abbia la percezione;
- **trasparenza ai guasti** (*failure transparency*): permette di mascherare un eventuale guasto di una risorsa e l'eventuale ripristino;
- **trasparenza alla migrazione** (*mobility transparency*): nasconde l'eventuale spostamento (logico o fisico) di una risorsa senza interferire sulla sua modalità di accesso, e quindi senza influenzare le operazioni degli utenti che la riguardano;
- **trasparenza alle prestazioni** (*performance transparency*): nasconde le operazioni necessarie per riconfigurare il sistema al variare del carico, allo scopo di migliorarne le prestazioni;
- **trasparenza di scalabilità** (*scaling transparency*): permette di espandere il sistema senza interromperne o modificarne il funzionamento.

Il perseguitamento di ognuna di queste forme di trasparenza necessita di investimenti e competenze tecniche per realizzarle.

## Economicità

Il **rapporto costo-prestazione** di un **sistema distribuito** permette effettivamente di considerare **economico** un investimento, anche ingente, per la sua realizzazione. Di fatti i

**costi**, spesso notevoli, per realizzare un simile sistema vengono normalmente **ripartiti fra una massa considerevole di utilizzatori**, che beneficiano a tal punto dei servizi offerti dal sistema distribuito da essere disposti a sostenerne la spesa.

Rispetto ai costi di realizzazione di un sistema centralizzato di un tempo, basato su mainframe, il costo di un sistema distribuito risulta sicuramente maggiore di diversi ordini di grandezza, ma a differenza della vecchia tecnologia, il costo procapite dell'uso del sistema viene ripartito su un maggior numero di soggetti, fornendo anche un **margine di guadagno** di molto superiore.

Inoltre la possibilità di **condividere** risorse hardware e software comporta vantaggi economici dovuti alla condivisione di apparecchiature speciali di costo elevato, che diversamente potrebbero risultare sotto-utilizzate.

Infine un sistema distribuito ben progettato dovrebbe risultare facilmente **scalabile** permettendo l'aggiunta di componenti al fine di migliorarne le prestazioni realizzando un bilanciamento di carico.

## Complessità

La **complessità** di un sistema distribuito ha **richiesto lo sviluppo di hardware** opportuno per garantire l'interconnessione dei suoi componenti e l'instradamento dei messaggi, in modo da fronteggiare adeguatamente la crescente richiesta dell'utenza.

Inoltre ha portato allo **sviluppo di nuovi paradigmi di programmazione e di nuovi linguaggi**, modificando radicalmente l'idea di programmazione e di sviluppo di sistemi software.

Infine sono sorte molteplici **problematiche legate alla sicurezza** che un tempo non era possibile neanche prevedere; l'accesso remoto alle risorse e lo sviluppo delle transazioni commerciali ha reso necessario lo sviluppo di sofisticate tecniche volte alla protezione dei dati e delle infrastrutture.

# Storia dei sistemi distribuiti e modelli architetturali

L'**obiettivo** primario della **progettazione dei computer** è stato sempre quello di **aumentarne le prestazioni**, soprattutto in termini di velocità, in modo che si potessero eseguire il maggior numero di istruzioni nel minore tempo possibile.

Le velocità raggiungibili hanno però un limite superiore imposto dalle leggi della fisica sulla velocità della luce, che nel vuoto raggiunge i 300.000 Km/s, mentre nel rame è possibile approssimarla a 200.000 Km/s. Da ciò si deduce che per raggiungere frequenze di lavoro dell'ordine dei GigaHertz, e quindi tempi di esecuzione delle istruzioni dell'ordine dei ns, è necessario che le distanze percorse nei conduttori interni non superino i 20 cm ( $v = (200.000 \cdot 10^3)/10^9 = 0,2$  m/s). **Le dimensioni dei componenti dei computer sono quindi importanti per non introdurre ritardi nell'esecuzione delle istruzioni.**

L'**eccessiva riduzione** delle dimensioni degli elaboratori ha però provocato **problematiche** legate sia alla **dissipazione dell'energia**, sia alla **meccanica quantistica** che entra in gioco pesantemente provocando fenomeni non controllabili, ma che allo stesso tempo ha indirizzato lo sviluppo dei nuovi computer quantistici.

L'**evoluzione dei computer non quantistici** ha optato per una **riorganizzazione dell'elaborazione** che consentisse di costruire macchine sempre più performanti gestendo le informazioni in modo diverso, e quindi **definendo architetture di elaborazione diverse**, sia dal punto di vista costruttivo (**hardware**), sia dal punto di vista logico (**software**).

Dal punto di vista **hardware** si è passati alla realizzazione di macchine e **sistemi dotati di più CPU** in modo da avere più potenza di calcolo senza esasperare i limiti di velocità di ogni singola CPU, sviluppando di macchine parallele o **macchine ad architettura parallela**.

Per il **software** lo sviluppo delle **architetture distribuite** hanno permesso di seguire, se non addirittura anticipare, i cambiamenti delle architetture hardware e dei loro sistemi operativi.

## CLIL Listening and Speaking - Quantum Computers

In these videos you will find how normal computer evolution has taken a strange path due to quantum mechanics. Watch the three videos and discuss the topic with your classmate.

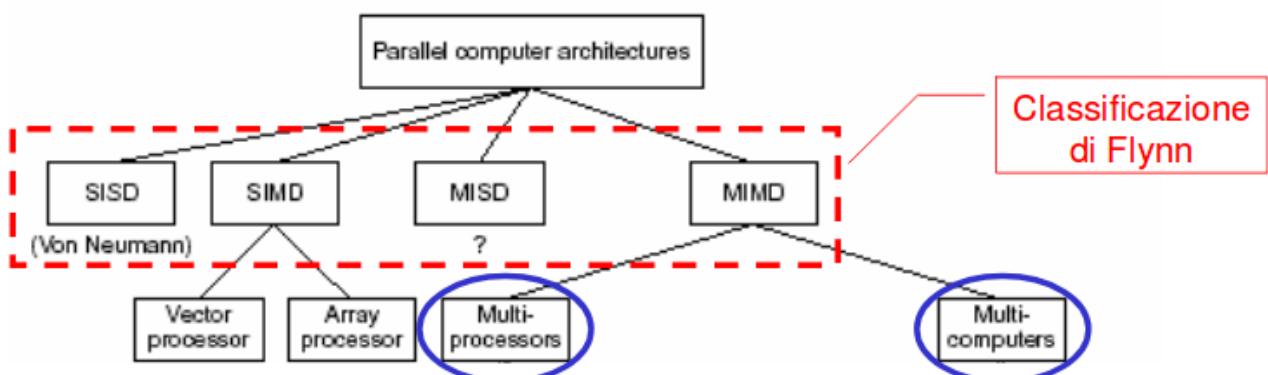
1. [Transistors & The End of Moore's Law](#)
2. [Quantum Computers Explained – Limits of Human Technology](#)
3. [How Does a Quantum Computer Work?](#)

# Architetture distribuite hardware

Esistono diverse possibilità per classificare le architetture hardware a seconda dei fattori che si prendono come riferimento.

La **tassonomia di Flynn**<sup>57</sup> è un sistema di classificazione delle architetture dei calcolatori ideata da Michael J. Flynn fra gli anni '60 e '70 che permette di identificare i sistemi di calcolo a seconda della **molteplicità del flusso di istruzioni e del flusso dei dati che possono gestire**. In seguito questa classificazione è stata estesa con una sottoclassificazione per considerare anche il tipo di architettura della memoria.

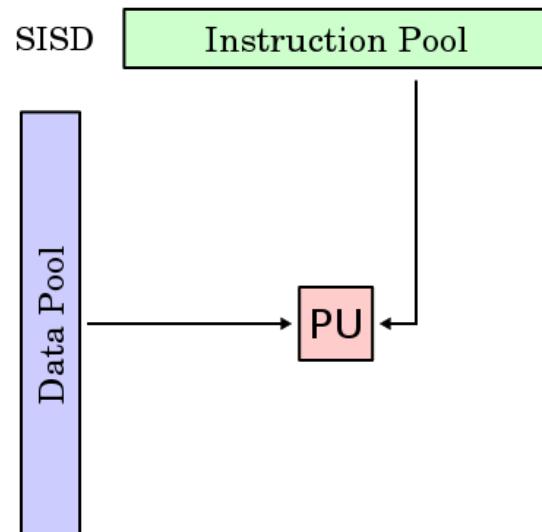
A seconda di come si combinano il flusso di dati e il flusso delle istruzioni abbiamo quattro possibili macro-gruppi, che possono ulteriormente suddividersi considerando anche l'architettura della memoria.



## SISD

L'architettura **SISD** (*Single Instruction Single Data*) è quella presente negli elaboratori che **prevedono l'uso di una singola CPU**, nei quali il flusso di istruzioni è unico e quindi viene eseguito un solo programma alla volta che agisce su un singolo flusso di dati. Dopo l'esecuzione di una istruzione si passa alla successiva seguendo un **processo esecutivo rigorosamente sequenziale**.

Un esempio teorico di questo tipo di architettura è costituito dalla macchina di Von Neumann, mentre nella realtà tutti i PC e i mainframe di vecchia generazione erano macchine ad una sola CPU e quindi la loro architettura era SISD.

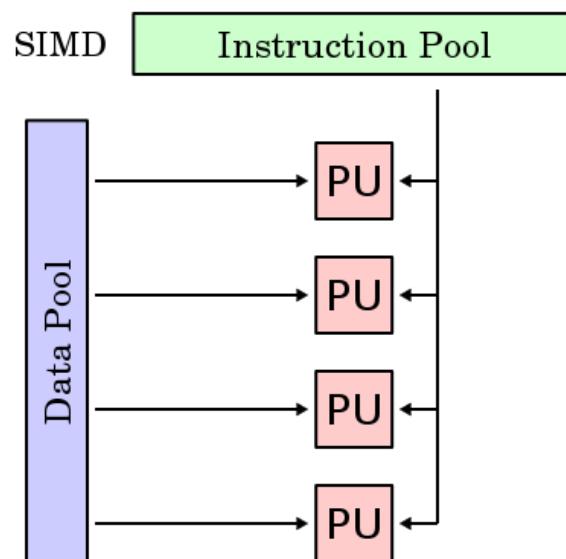


<sup>57</sup> [Tassonomia di Flynn](#)

## SIMD

Le architetture **SIMD** (*Single Instruction Multiple Data*) sono composte da **molte unità di elaborazione che eseguono contemporaneamente la stessa istruzione ma lavorano su insiemi di dati diversi**. Generalmente, il modo di implementare le architetture SIMD è quello avere un processore principale che invia le istruzioni da eseguire contemporaneamente un insieme di elementi di elaborazione che provvedono ad eseguirle. Il processore principale spesso è ospitato all'interno di un calcolatore convenzionale che provvede a supportare anche l'ambiente di sviluppo.

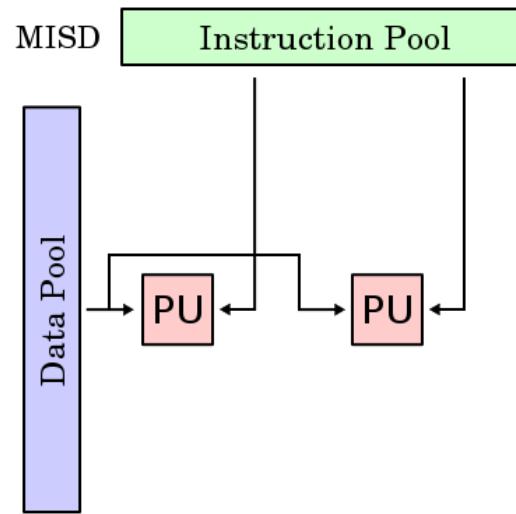
I sistemi SIMD sono utilizzati principalmente per supportare computazioni specializzate in parallelo, come ad esempio i supercomputer vettoriali usati per particolari applicazioni scientifiche dove si lavora su grandi matrici, gli array processor che in origine erano progettati per applicazioni di intelligenza artificiale e di calcolo simbolico, anche se versioni successive ebbero successo nelle scienze applicate che richiedevano elevate potenza di calcolo, e le GPU (*Graphics Processing Unit*) utilizzate per la grafica ad alto livello.



di  
ad

## MISD

Le architetture **MISD** (*Multiple Instruction Single Data*) prevedono molteplici sistemi che eseguono **processi diversi che agiscono su unico flusso di dati**. Questo tipo di architettura, pur non essendo particolarmente diffusa, viene usata generalmente in termini di tolleranza ai guasti (*fault tolerance*), in cui sistemi eterogenei devono operare sullo stesso flusso di dati dovendo concordare sul risultato finale. Un esempio reale è presente nel computer di controllo del volo dello *Space Shuttle*.



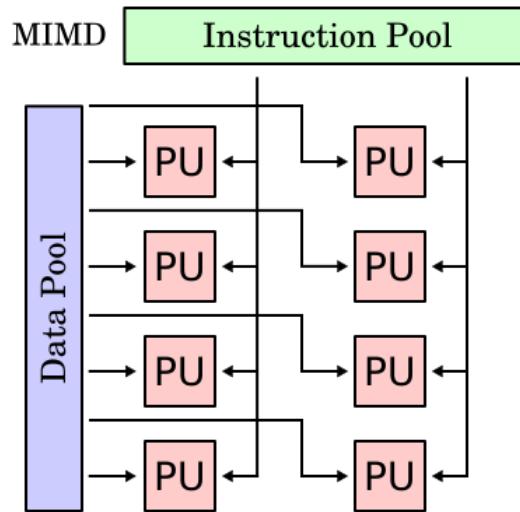
un

## MIMD

L'architettura **MIMD** (*Multiple Instruction Multiple Data*) comprende tutte le tipologie di elaboratori composti da **più unità centrali di elaborazione indipendenti che possono lavorare su stream di dati anch'essi indipendenti**, raggiungendo un parallelismo a livello di thread.

Per questa architettura viene effettuata una ulteriore classificazione delle in base alla suddivisione della memoria fisica:

- macchine **MIMD a memoria fisica condivisa**;
- macchine **MIMD a memoria fisica distribuita** e private.

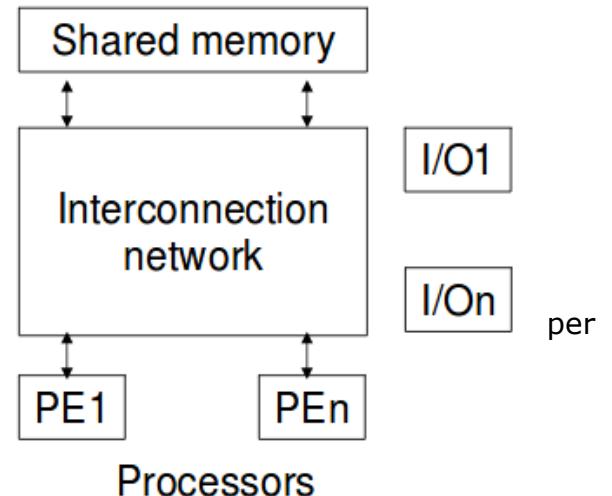


Le prime sono anche conosciute con il nome di *multiprocessor*, mentre le seconde con quello di *multicomputer*.

### MIMD a memoria fisica condivisa

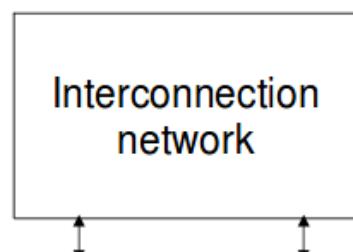
I **sistemi MIMD multiprocessori** sono architetture a **memoria fisica condivisa** (*shared memory*) che costituisce un unico spazio di indirizzamento condiviso tra tutti i processori. Dato che la comunicazione tra processi avviene mediante variabili condivise, risulta necessario implementare opportuni meccanismi di sincronizzazione per regolare gli accessi alla memoria, in modo da coordinare i diversi processi gestire la competizione alle risorse comuni.

In questo tipo di architettura non necessariamente i processori devono avere un'unica memoria in comune centralizzata, possono anche avere ciascuno una propria memoria e condividerne una parte con gli altri processori, in modo da realizzare una memoria condivisa distribuita.



### MIMD a memoria fisica distribuita

I **sistemi MIMD multicomputer** sono architetture che non hanno una memoria condivisa e quindi la



**comunicazione** avviene mediante lo **scambio di messaggi** (*message passing*) esplicito effettuato mediante apposite procedure (*send* e *receive*), come avviene utilizzando i **socket**. Ogni computer possiede una propria area di memoria privata, non indirizzabile da parte dei processori remoti.

Un tipico esempio di questa architettura è fornito dai **cluster di PC**.

## Cluster di PC

Un **computer cluster** è un insieme di computer connessi tra loro tramite una rete, e nasce dall'esigenza di distribuire fra i vari computer del cluster elaborazioni molto complesse, scomponendole in sotto-elaborazioni separate che vengono eseguite in parallelo. Questo ovviamente aumenta potenza di calcolo del sistema, solitamente garantendo un'alta disponibilità di servizio. Per contro la gestione dell'infrastruttura può avere maggiori costi e complessità di gestione, pur risultando economico nella sua implementazione, fortemente scalabile e affidabile.



la

Si può parlare propriamente di cluster di PC quando un insieme di computer completi e interconnessi ha le seguenti proprietà:

- i vari computer appaiono all'utente come una singola risorsa computazionale;
- le varie componenti sono risorse dedicate al funzionamento dell'insieme.

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 normale rete di PC principalmente:

- per la velocità del trasferimento dati (oltre 1 Gbit/s);
- per la centralizzazione fisica delle macchine (generalmente tutti i PC sono montati sullo stesso rack);
- per la presenza di una applicazione di management, residente su un singolo PC, che permette di lanciare processi su altri PC, monitorare il loro comportamento.

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 e di ridurre il gravoso inconveniente del tempo di elaborazione anche per quei problemi che si potrebbero risolvere con un singolo computer ma al prezzo di un tempo molto elevato.

## CLIL Listening and Speaking - Flynn Taxonomy and Cluster Architecture

In these videos you will understand the Flynn's taxonomy and why cluster architecture is widely used in big enterprise. Watch the videos and discuss the topics with your classmate.

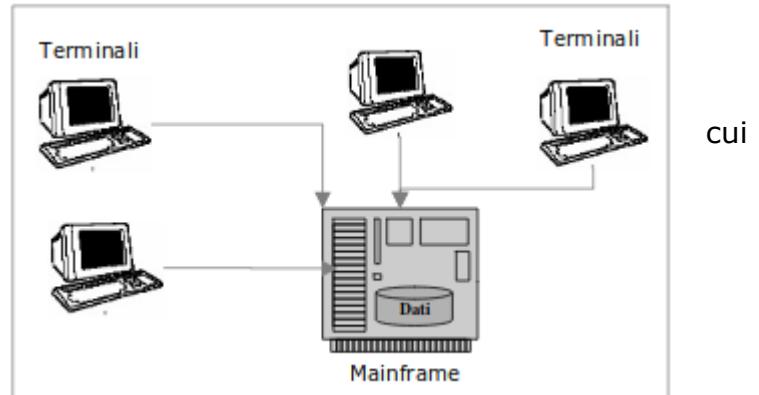
1. [Flynn's Taxonomy of Parallel Machines - Georgia Tech - HPCA: Part 5](#)
2. [Big Ideas: Simplifying Cluster Architectures](#)

## Architetture distribuite software

Come per l'hardware, anche per il software si è avuta avuto una evoluzione nelle architetture distribuite che spesso hanno anticipato i cambiamenti delle architetture hardware e dei loro sistemi operativi, passando attraverso tre fasi principali: l'architettura centralizzata, l'architettura client-server e l'architettura multi-tier.

### Architettura centralizzata

La prima architettura di elaboratori, definita anche **1 tier o ad un livello** (anni '70), prevedeva un'unica unità centrale di elaborazione (*mainframe*) a erano collegati terminali privi di capacità computazionali. Lo scopo dei terminali remoti era quello di inviare i dati all'unità centrale e di ricevere i relativi risultati per visualizzarli, mentre l'unità centrale gestiva i dati, la logica di business e l'interfaccia utente.



I terminali remoti avevano solitamente caratteristiche omogenee e l'unità centrale replicava per ciascuno di essi un'area di memoria riservata, facendo evolvere singolarmente i singoli task avviati dai diversi terminali remoti.

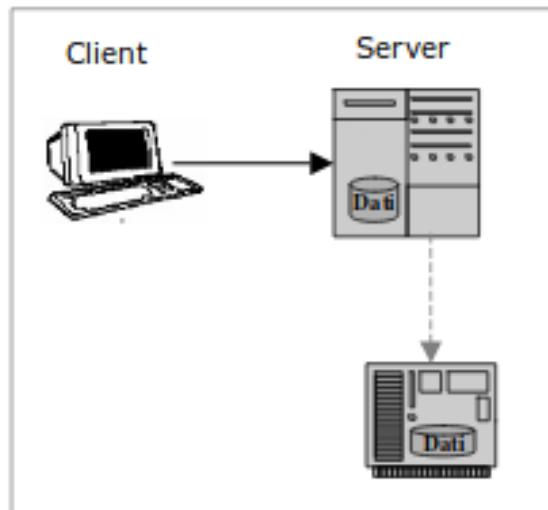
I principali vantaggi di questa architettura erano l'elevata sicurezza delle funzioni, l'efficienza esecutiva, in quanto non era previsto un overhead per la comunicazione remota in fase elaborativa e uno sviluppo architetturale relativamente semplice. Per contro l'hardware era estremamente costoso, poco scalabile e il sistema presentava molti problemi di integrazione in quanto erano solitamente sistemi chiusi.

Alla base dell'evoluzione di questa architettura c'è principalmente la nascita della rete Internet, che pur non essendo inizialmente quella che oggi conosciamo, presenta possibilità che spingono fortemente verso una maggiore distribuzione. Contemporaneamente inizia una spinta di informatizzazione presso le piccole e medie imprese vista la massiva produzione di client più prestanti e a costi contenuti se paragonati ai mainframe usati nelle grandi aziende.

## Architettura client-server

L'architettura client-server, definita anche **architettura a due livelli** o **2 tier** (anni '80), differenza della precedente prevede dei client con una loro capacità di elaborazione in grado iniziare la richiesta di un servizio ad un server, che invece si occupa di elaborare la richiesta e inviare la risposta al client.

Un client può richiedere più servizi a server diversi e un server può ricevere più richieste molteplici client. Inoltre un server può assumere il ruolo anche di client nel momento cui necessita di un servizio offerto da un altro server per soddisfare la richiesta iniziale del client.



a  
di  
da  
in

In questo tipo di architetture due client possono collaborare tra loro unicamente attraverso uno o più server che permettono la coordinazione e la condivisione dei dati. Client e server possono essere tecnologicamente diversi, sia come hardware che come sistema operativo, e in generale questa architettura è quella che meglio si presta a far comunicare e cooperare entità non omogenee.

La comunicazione tra client e server avviene mediante protocolli ben definiti e generalmente sono sistemi aperti che riescono a raggiungere un buon livello di integrazione. Generalmente l'hardware può o meno essere costoso a seconda della potenza di calcolo richiesta, richiede un discreto costo di amministrazione, soprattutto per questioni di sicurezza e la scalabilità dipende dalla progettazione effettuata a monte.

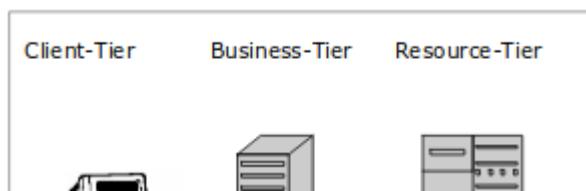
Nel tempo questa architettura ha previsto lo sviluppo di due modelli:

- il modello **thin-client** (primi anni '80) in cui il server è il responsabile della logica applicativa e della gestione dei dati, mentre il client è responsabile dell'esecuzione del software di presentazione;
- il modello **thick-client** o **fat-client** (fine anni '80) in cui il server è responsabile della gestione dei dati, mentre il client è responsabile del software di presentazione e della logica applicativa, grazie all'aumento della potenza di calcolo dei PC.

Le principali ragioni che hanno portato un'evoluzione rispetto a questa architettura sono ascrivibili alla standardizzazione dei protocolli di rete, al crescente ampliamento della banda disponibile, alla necessità di distribuire l'informazione e i servizi su media diversi e di gestire molte transazioni on line, oltre alla necessità di evitare, per questioni di sicurezza, il *single point of failure*.

## Architettura multi-tier

Per alleggerire il carico elaborativo dei server vennero sviluppati **sistemi multilivello** (anni '90), nei quali avvenne la **separazione delle funzionalità logiche del sistema in livelli software diversi**. Allo scopo vennero

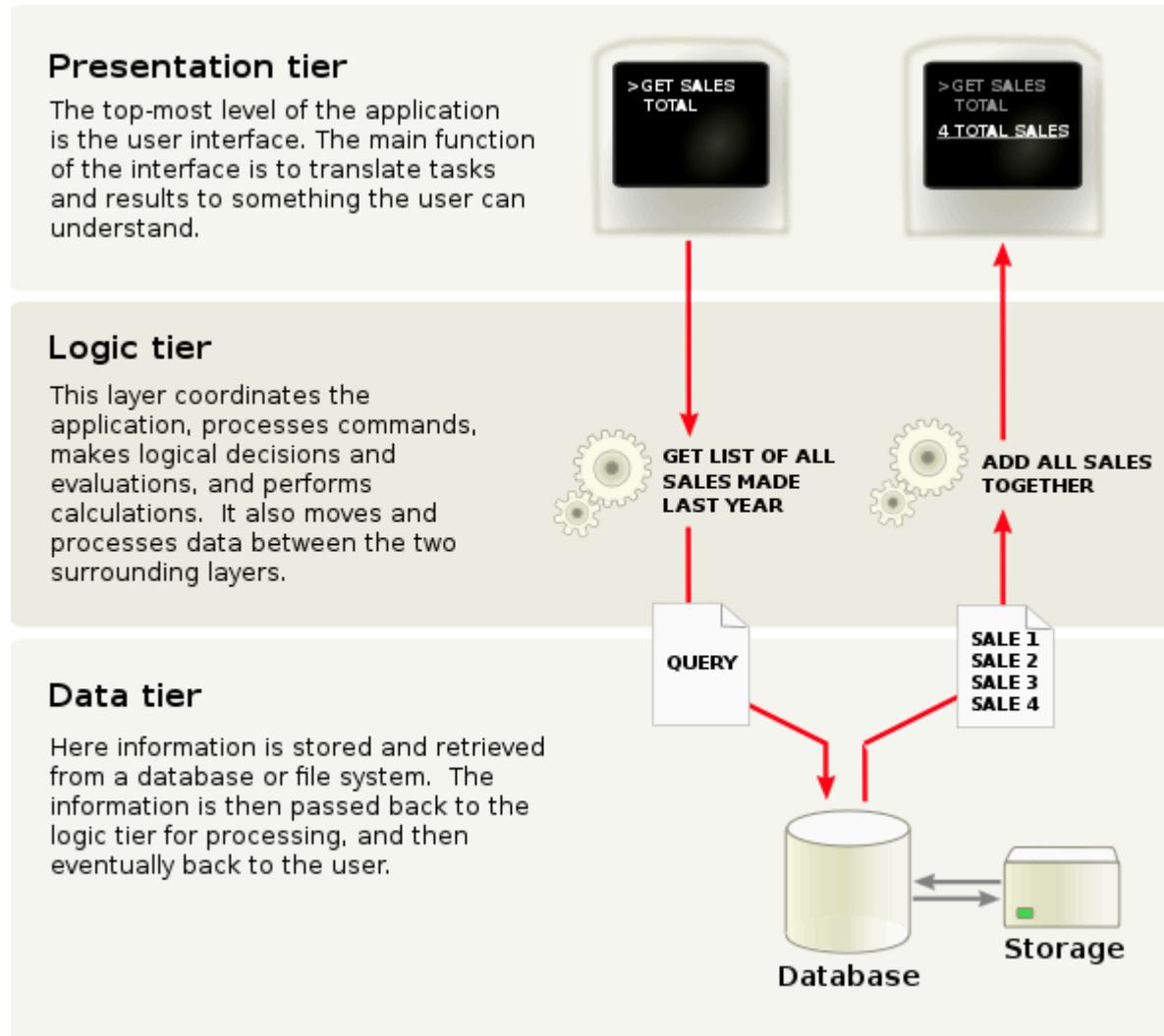


introdotti un insieme di strumenti, complessivamente identificati con il termine **middleware**, che rappresenta uno strato software che si colloca sopra al sistema operativo ma sotto i programmi applicativi, rappresentando l'evoluzione dei sistemi operativi distribuiti.

Il **modello a multi-tier** richiede una chiara definizione delle interfacce e dei protocolli di comunicazione usati fra i vari livelli, in modo che possano richiedere ed offrire servizi fra loro. Inoltre questa separazione logica delle funzionalità permette di apportare modifiche ai vari livelli, limitando al minimo l'impatto sugli altri livelli.

In genere l'architettura multi-tier prevede lo sviluppo di applicazioni su tre livelli distinti:

1. **Presentation Layer:** rappresenta l'interfaccia utente ed è composta dall'insieme delle procedure 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.
2. **Business Logic Layer o Application Logic Layer:** è il corpo centrale dell'applicazione che comprende la logica di elaborazione e la definizione delle relazioni esistenti tra le diverse entità. Per esempio, questo layer potrebbe comprendere 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 on-line.
3. **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 sia implementato tramite un DBMS, questo layer è detto semplicemente Data Access Layer.



I principali **vantaggi** di questa architettura sono la **scalabilità**, la possibilità di applicare sistemi di **sicurezza a livello di singolo servizio** e con **livelli diversi** a seconda del layer considerato.

Sicuramente un'architettura di questo tipo è **complicata** sia a livello progettuale, sia a livello di sviluppo e amministrazione.

In ogni caso questo modello permette lo sviluppo di architetture distribuite che prevedono la specializzazione dei server per servizi diversi, bilanciare il carico di lavoro tra varie entità e l'integrazione di servizi che risiedono su server diversi.

## CLIL Listening and Speaking - n-Tier Architecture

In these videos you will understand how n-tier architecture works. Watch the videos and discuss the topics with your classmate.

1. [3 Tier Client Server Architecture](#)
2. [n-Tier Architecture Explained](#)
3. [N-Tier Architecture for kids](#)
4. [Middleware Architecture](#)

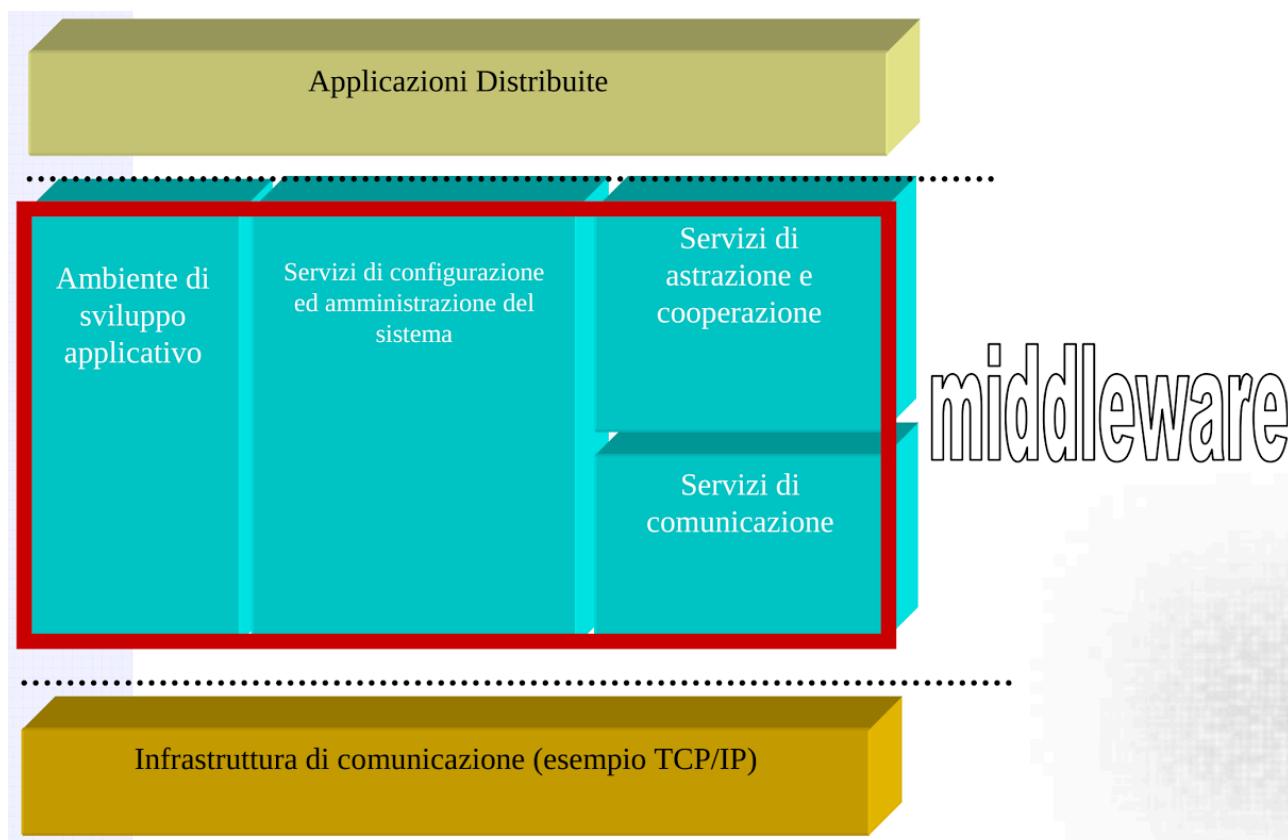
# Middleware

Il **middleware** ha lo scopo di **realizzare la comunicazione e le interazioni tra i diversi componenti software di un sistema distribuito** e rappresenta una classe di tecnologie software sviluppate per aiutare gli sviluppatori nella gestione della complessità e della eterogeneità presenti nei sistemi distribuiti.

Il middleware è quindi un software di connessione che consiste di un insieme di servizi e/o ambienti di sviluppo di applicazioni distribuite che permettono a più entità (processi, oggetti, ecc.), residenti su uno o più elaboratori, di interagire attraverso una rete di interconnessione a dispetto di differenze nei protocolli di comunicazione, architetture dei sistemi locali e sistemi operativi.

Di conseguenza il middleware garantisce:

- **interoperabilità** o **integrazione** tra applicazioni e sistemi operativi diversi;
- **connettività** tra servizi che devono interagire e collaborare su piattaforme distribuite, utilizzando meccanismi di programmazione e API.



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 (Socket).

Tra le funzionalità del middleware ricordiamo:

- **servizi di astrazione e cooperazione**: rappresentano il cuore del middleware e comprendono:

- **directory service:** provvede alla identificazione e alla localizzazione delle entità, rendendo le applicazioni al di sopra del middleware indipendenti dal sottosistema di instradamento dei messaggi (**trasparenza di locazione**);
- **security service:** finalizzato alla protezione degli elementi critici, come i dati e i servizi applicativi, utilizzando tecniche di autenticazione, autorizzazione e crittografia;
- **time service:** assicura che tutti i clock interni tra client e server siano sincronizzati entro un accettabile livello di varianza.
- **servizi per le applicazioni:** permettono alle applicazioni di avere un accesso diretto con la rete e con i servizi offerti dal middleware, ad esempio, un servizio orientato alle transazioni può fornire un supporto per un accesso transazionale a basi di dati eterogenee;
- **servizi di amministrazione del sistema:** servono per effettuare monitoraggio, configurazione e pianificazione di interventi sul sistema;
- **servizio di comunicazione:** questo servizio offre delle API che permettono all'applicazione distribuita di scambiare informazioni fra tutte le sue componenti residenti su altri elaboratori aventi anche caratteristiche hardware o software diverse. Lo scopo di questo servizio è di nascondere le disomogeneità dovute alla rappresentazione dei dati usata dai vari elaboratori, dai sistemi operativi locali e dalle diverse reti che costituiscono l'infrastruttura della piattaforma. I paradigmi di comunicazione possono essere basati su RPC, messaggistica applicativa, o altre;
- **ambiente di sviluppo applicativo:** offre diversi tool, come strumenti di aiuto alla scrittura di programmi, debugging, gestione di applicativi sia ad oggetti che a processi, *Interface Definition Language* che permette l'interconnessione tra moduli scritti in diversi linguaggi di programmazione e residenti su elaboratori distinti.

Il middleware offre molti servizi, anche più di quanti in realtà potrebbero essere necessari per una applicazione, e quindi a livello progettuale si dovrà mediare prestazioni dell'applicazione e efficienza esecutiva, cercando di includere solo i servizi necessari per non appesantire troppo il sistema finale.

## CLIL Listening and Speaking - Middleware

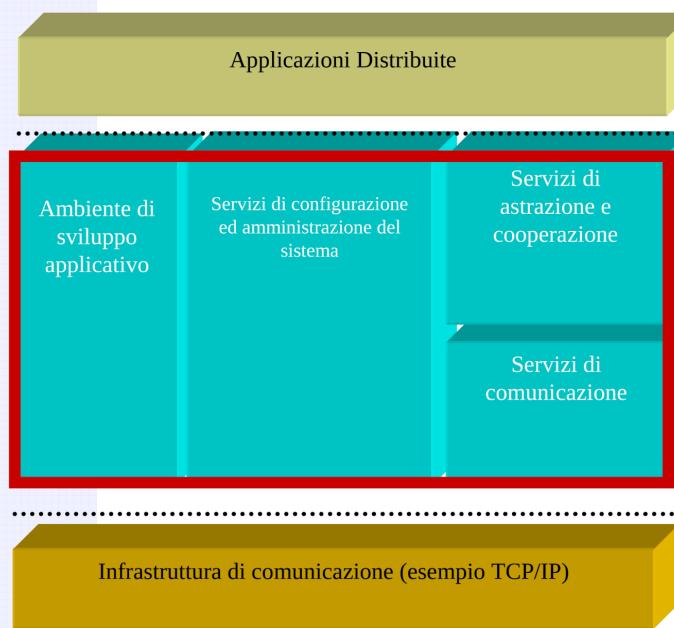
In this video you will understand what middleware is. Watch the videos and discuss the topics with your classmate.

1. [Middleware Concepts](#)

# MIDDLEWARE

*Il middleware è un software di connessione che consiste di un insieme di servizi e/o di ambienti di sviluppo di applicazioni distribuite che permettono a più entità (processi, oggetti ecc.), residenti su uno o più elaboratori, di interagire attraverso una rete di interconnessione a dispetto di differenze nei protocolli di comunicazione, architetture dei sistemi locali, sistemi operativi ecc..*

# MIDDLEWARE



# Middleware: Servizi

**Servizio di Comunicazione.** Questo servizio offre una API che permette all'applicazione distribuita di scambiarsi informazioni tra le sue componenti residenti su elaboratori con caratteristiche hardware e/o software differenti. Lo scopo di questo servizio è di nascondere le disomogeneità dovute alla rappresentazione dei dati usata dai vari elaboratori, ai sistemi operativi locali ed a differenti reti che costituiscono l'infrastruttura della piattaforma. Per comunicazioni si intendono diversi paradigmi di interazione che vanno dalle *RPC*, alla *messaggistica applicativa* al modello *publish/subscribe*, al mantenimento di discipline diverse da quella *FIFO* (esempio *causal order*, *total order*)

# Middleware: Servizi

**Servizi di astrazione e cooperazione.** Questi servizi rappresentano il cuore del middleware e comprendono tra gli altri:

- *Directory Service* (o servizio di “pagine gialle”) che provvede alla identificazione ed alla localizzazione di entità. Questo servizio rende le applicazioni al di sopra del middleware indipendenti dal sottosistema di instradamento dei messaggi.
- *Security Service* che è finalizzato alla protezione di elementi critici come dati e servizi applicativi attraverso tecniche come autenticazione, autorizzazione e criptografia.
- *Time Service* il quale assicura che tutti i clock interni tra client e server siano sincronizzati entro un accettabile livello di varianza

# Middleware: Servizi

**Servizi per le applicazioni.** Questi servizi permettono ad una applicazione di avere un accesso diretto o con la rete o con i servizi offerti dal middleware. Ad esempio un servizio orientato alle transazioni, all'interno di un middleware, può fornire un supporto per accesso transazionale a basi di dati eterogenee.

• **Servizi di amministrazione del sistema.**

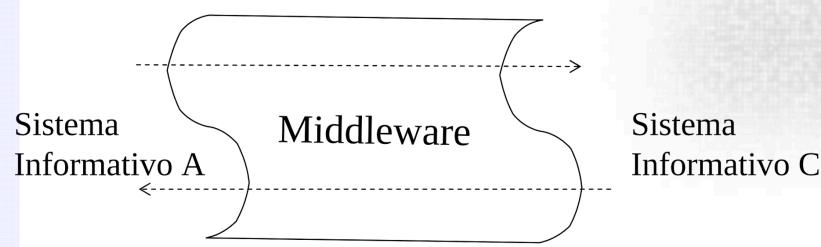
- monitoring
- configurazione
- pianificazione.

**Ambiente di sviluppo applicativo.**

- strumenti di aiuto alla scrittura
- debugging
- caricamento di applicazioni distribuite sia ad oggetti che basate su processi.
- (*Interface Definition Language*) per l'interconnessione tra moduli scritti in diversi linguaggi di programmazione e residenti su elaboratori distinti.

# Middleware: Obiettivi

- interoperabilità e connettività per applicazioni distribuite su piattaforme eterogenee
- collante delle applicazioni utilizzate all'interno di una azienda.



## Middleware: Problemi

Un middleware molto spesso offre più servizi di quelli di cui una applicazione ha realmente bisogno, quindi si pone un problema progettuale (scegliere il migliore middleware in rapporto ai servizi richiesti da una applicazione) e di prestazioni (il middleware scelto non deve appesantire troppo il sistema locale o la rete di interconnessione).

Configurabilità

## E - Vecchi esempi di socket TCP e UDP

# Daytime - Server iterativo

La seguente classe<sup>58</sup> implementa un semplice server TCP in linguaggio Java. Il server restituisce al client, che effettua la connessione, una stringa di caratteri ASCII riportante l'indicazione della data e ora correnti. Il server può accettare e servire contemporaneamente fino a 50 connessioni in modo iterativo e non concorrente.

## DayTimeServer.java

```
import java.io.*;
import java.net.*;
import java.util.Date;

public class DayTimeServer implements Runnable {
    private ServerSocket server;

    /**
     * Constructor that accepts network connection to any NIC.
     * @param port server port number
     * @throws IOException
     */
    public DayTimeServer(int port) throws IOException {
        // 50 is the greatest number of pending connection.
        // A null address specify that the server accepts network connection to
        // any NIC.
        server = new ServerSocket(port, 50, null);
        // Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
        // With this option set to a non-zero timeout, a read() call on the
        // InputStream associated with this Socket will block for only this
        // amount of time. If the timeout expires, a
        // java.net.SocketTimeoutException is raised, though the Socket is still
        // valid. The option must be enabled prior to entering the blocking
        // operation to have effect.
        // The timeout must be > 0. A timeout of zero is interpreted as an
        // infinite timeout.
        server.setSoTimeout(100000);
    }

    /**
     * Constructor that accepts network connection only to the specified NIC.
     * @param port server port number
     * @param addr server IP address
     * @throws IOException
     */
    public DayTimeServer(int port, String addr) throws IOException {
```

---

<sup>58</sup> La classe è quella presente sul libro di testo G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni - per Informatica*, vol. 3, ed. Zanichelli, 2017, 2<sup>a</sup> edizione, p.109 con alcune modifiche sul codice proposto.

```
InetAddress address = InetAddress.getByName(addr);
server = new ServerSocket(port, 50, address);
server.setSoTimeout(100000);
}

/**
 * Default Constructor that accepts connection to the ordinary day-time
 * port, number 13.
 * @throws IOException
 */
public DayTimeServer() throws IOException {
    // la porta TCP standard del servizio daytime è 13
    server = new ServerSocket(13);
    server.setSoTimeout(100000);
}

/**
 * Start a thread to serve a client
 */
@Override
public void run() {
    // client socket
    Socket connection = null;

    // this thread will continue until it is not interrupted
    while (!Thread.interrupted()) {
        try {
            // the server will wait for a client connection for 100 seconds
            connection = server.accept();
            // server is printing client IP address and port number
            System.out.println("Data/ora richiesta da: " +
                connection.getInetAddress().toString() +
                ":" + connection.getPort());
            // creation of an output stream
            PrintWriter out = new PrintWriter(connection.getOutputStream());
            // current date and time
            Date now = new Date();
            // server sends to the client a string containing current date and
            // time
            out.println(now.toString() + "\r\n");
            out.flush();
            // output stream and socket closure
            out.close();
            connection.close();
        } catch (IOException ex) {
            System.err.println("Error run()" + ex.getMessage() + " " +
ex.toString());
            Thread.currentThread().interrupt();
        }
    }
}
```

```
        } finally {
            if (connection != null) {
                try {
                    connection.close();
                } catch (IOException ex) {
                    System.err.println("Finally error in run()");
                }
            }
        }
    }
// close server socket
try {
    server.close();
} catch (IOException ex) {
    System.err.println("Server socket closure error");
}
}

public static void main(String[] args) {
    int c;
    Thread thread;

    try {
        // on *nix OS do not use the 13th port because is probably used by the
        // the day-time service
        DayTimeServer daytimeServer = new DayTimeServer(4444);
        thread = new Thread(daytimeServer);
        thread.start();
        // the server will stop after 100 seconds pressing any button
        c = System.in.read();
        thread.interrupt();
        thread.join();
    } catch(IOException ex) {
        System.err.println("Error " + ex.getMessage());
    } catch (InterruptedException ex) {
        System.err.println("End");
    }
}
}
```

Nel server il metodo `main()` esemplifica l'uso della classe `DayTimeServer`, nel quale viene eseguito il metodo `run()` tramite l'invocazione del metodo `start()` su un oggetto `Thread`.

Si osservi che il metodo `run()` è stato definito nella classe `DayTimeServer` dato che la classe implementa l'interfaccia `Runnable`, la quale richiede appunto la ridefinizione del metodo `run()`. Il thread viene avviato fornendo l'oggetto `DayTimeServer` che implementa l'interfaccia `Runnable` come argomento del costruttore di una istanza della classe `Thread`.

Il thread sarà interrotto tramite l'invocazione del metodo `interrupt()` della classe `Thread`, mentre l'uso del metodo `join()` della classe `Thread` attende l'effettiva terminazione del metodo `run()` in quanto il thread rimane sospeso al massimo per 100 secondi ad ogni ciclo, in attesa di una connessione tramite il metodo `accept()` della classe `ServerSocket`.

**Il funzionamento del server TCP daytime può essere verificato impiegando un qualsiasi client Telnet**, la stringa contenente la data e l'ora viene ricevuta dal client appena si effettua la connessione e, immediatamente dopo, la connessione viene chiusa da parte del server.

```
mgm@umgm:~$ telnet 127.0.0.1 4444
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Tue Oct 02 12:23:14 CEST 2018
```

**Connection closed by foreign host.**

```
mgm@umgm:~$
```

Se invece si volesse creare un client, questo potrebbe ad esempio connettersi al server imponendo un'attesa massima di 100 secondi per la risposta da parte del server. Nel caso il server non rispondesse entro questo tempo verrebbe generato un `SocketTimeoutException`. Una volta connesso, il client riceverà dal server la data e l'ora attuali e chiuderà la connessione, seguito anche dal client.

### DayTimeClient.java<sup>59</sup>

```
import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

public class DayTimeClient {

    private String serverName; // server address
    private int serverPort; // server port

    /**
     * Constructor using the server IP address and port number
     * @param server server IP address
     * @param port server port number
     */
    public DayTimeClient(String server, int port) {
        serverName = server;
        serverPort = port;
    }
}
```

---

<sup>59</sup> La classe è quella presente sul libro di testo G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni - per Informatica*, vol. 3, ed. Zanichelli, 2017, 2a edizione, p.112 con alcune modifiche sul codice proposto.

```
/**  
 * Constructor using the server IP address and the port 13  
 * @param server server IP address  
 */  
public DayTimeClient(String server) {  
    serverName = server;  
    // port 13 is the standard port number of day-time service  
    serverPort = 13;  
}  
  
/**  
 * Client request of server day-time  
 * @return the actual server day and time  
 * @throws java.net.SocketTimeoutException  
 * @throws IOException  
 */  
public String getDayTime() throws SocketTimeoutException, IOException {  
    BufferedReader in; // client input stream  
    String answer = ""; // used for response  
  
    // Socket clientSoket = new Socket("127.0.0.1", 4444);  
    // client socket without any indication of server IP address and port  
    // number  
    Socket clientSocket = new Socket();  
    // creation of an inet socket address using server parameters  
    InetSocketAddress serverAddress  
        = new InetSocketAddress(serverName, serverPort);  
    // Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.  
    // With this option set to a non-zero timeout, a read() call on the  
    // InputStream associated with this Socket will block for only this  
    // amount of time. If the timeout expires, a  
    // java.net.SocketTimeoutException is raised, though the Socket is still  
    // valid. The option must be enabled prior to entering the blocking  
    // operation to have effect.  
    // The timeout must be > 0. A timeout of zero is interpreted as an  
    // infinite timeout.  
    clientSocket.setSoTimeout(100000);  
    // client connection request and it will wait at least 100 seconds  
    // for server connection  
    clientSocket.connect(serverAddress, 100000); // 100000ms = 100s  
    // client input stream  
    in = new BufferedReader(  
        new InputStreamReader(clientSocket.getInputStream()));  
    answer = in.readLine();  
    // client stream closing  
    in.close();  
    // client socket closing
```

```
clientSocket.close();
return answer;
}

public static void main(String args[]) {
    String server; // server IP address
    int port; // server port number
    String daytime; // server day and time
    DayTimeClient client; // object DayTimeClient

    server = JOptionPane.showInputDialog("Indirizzo del server");
    port = Integer.parseInt(JOptionPane.showInputDialog("Porta del server"));

    try {
        client = new DayTimeClient(server, port);
        daytime = client.getDayTime();
        JOptionPane.showMessageDialog(null, daytime, "Server day and time",
JOptionPane.QUESTION_MESSAGE);
    } catch (SocketTimeoutException ex) {
        System.err.println("Nessuna risposta dal server");
    } catch (IOException ex) {
        System.err.println("Errore di comunicazione");
        Logger.getLogger(DayTimeClient.class.getName()).log(Level.SEVERE, null,
ex);
    }
}
}
```

Si osservi che l'esempio appena proposto prevede una gestione iterativa di possibili richieste concorrenti da parte di più client: in pratica **ogni client attenderà il proprio turno fino a quando il server non sarà in grado di servirlo sulla porta 4444**.

Questa soluzione può essere utilizzata esclusivamente per implementare semplici protocolli, come quello presentato, che nella realtà è gestito dal protocollo NTP ([Network Time Protocol](#)) e permette l'acquisizione della data e dell'ora da una rete.

Normalmente invece la comunicazione tra client e server potrebbe essere più lunga e complessa, e ciò implica che il server sia in grado di servire in modo concorrente più richieste da parte di client distinti. Per fare ciò il server dovrà creare uno specifico thread di comunicazione per ogni client, in modo tale da spostarlo su una nuova porta di servizio e lasciare così libera quella principale di richiesta di connessione alla quale altri client potranno richiedere il servizio. Questa tecnica può prevedere una gestione senza limiti dei thread o una gestione che limiti il numero massimo di thread, e quindi di client, servibili.

# Echo - Server concorrente

Il seguente esempio<sup>60</sup> è relativo ad un server concorrente che avvia un thread per ogni nuova richiesta di un client, senza porre limiti al numero di client servibili in contemporanea.

Il server semplicemente effettua l'eco di ciò che riceve dal client e poi chiude la connessione. Per fare ciò crea un nuovo thread che libererà così la porta principale di ascolto del server.

Il metodo `Clientthread()` gestisce la comunicazione con il client all'interno del metodo `run()`

## ServerTCPConcorrente.java

```
import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * The server manages only the client connection request .
 * The communication protocol is entirely managed by the run() method in
 * ClientThread class. The communication socket created by the accept()
 * method invocation is given as parameter to the ClientThread constructor.
 * @author mgm
 */
public class ServerTCPConcorrente extends Thread {

    public final static int SERVER_PORT = 5555;
    private ServerSocket server;

    public ServerTCPConcorrente(int port) throws IOException {
        server = new ServerSocket(port);
        server.setSoTimeout(10000);           // timeout: 10s
    }

    @Override
    public void run() {
        Socket connection;

        while (!Thread.interrupted()) {
            try {
                // the server will wait at least 10s
                connection = server.accept();
                Clientthread client = new Clientthread(connection);
                client.start();
            } catch (IOException ex) {
                Logger.getLogger(ServerTCPConcorrente.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

---

<sup>60</sup> La classe è quella presente sul libro di testo G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni - per Informatica*, vol. 3, ed. Zanichelli, 2017, 2a edizione, p.114 con alcune modifiche sul codice proposto.

```
        System.out.println("Connection requested by: "
            + connection.getInetAddress().toString()
            + ":" + connection.getPort());
        // the server starts a new thread to serve the current client
        Thread clientThread = new ClientThread(connection);
        clientThread.start();
    } catch (SocketTimeoutException ex) {
        System.out.println("Connection closed for timeout ");
    } catch (IOException ex) {
        Logger.getLogger(ServerTCPConcorrente.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
try {
    server.close();
} catch (IOException ex) {
Logger.getLogger(ServerTCPConcorrente.class.getName()).log(Level.SEVERE, null,
ex);
}
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    int c;

    try {
        ServerTCPConcorrente server = new ServerTCPConcorrente(SERVER_PORT);
        server.start();
        c = System.in.read();
        server.interrupt();
        server.join();
    } catch (IOException | InterruptedException ex) {
Logger.getLogger(ServerTCPConcorrente.class.getName()).log(Level.SEVERE, null,
ex);
    }
}
}
```

### ClientThread.java<sup>61</sup>

```
import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;
```

---

<sup>61</sup> La classe è quella presente sul libro di testo G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni - per Informatica*, vol. 3, ed. Zanichelli, 2017, 2a edizione, p.116 con alcune modifiche sul codice proposto.

```
public class ClientThread extends Thread {

    private Socket connection;          // client socket connection
    private BufferedReader input;        // wrapper of the socket input stream
    private InputStream in;              // input stream socket
    private PrintWriter output;         // wrapper of the socket output stream
    private OutputStream out;           // output stream socket

    public ClientThread(Socket connection) throws IOException {
        this.connection = connection;
        in = this.connection.getInputStream();
        input = new BufferedReader(new InputStreamReader(in));
        out = this.connection.getOutputStream();
        output = new PrintWriter(out);
    }

    @Override
    public void run() {
        String echo;
        try {
            echo = input.readLine();
            output.println(echo);
            output.flush();
        } catch (IOException ex) {
            Logger.getLogger(ClientThread.class.getName()).log(Level.SEVERE, null,
ex);
        }
        System.out.println("Connection closed");
        try {
            in.close();
            out.close();
            connection.close();
        } catch (IOException ex) {
            Logger.getLogger(ClientThread.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
}
```

## TFTP

Il server e il client<sup>62</sup> presentati in questo esempio simulano una parte del protocollo TFTP descritto nello [RFC 1350](#). Questo protocollo è un buon esempio per capire cosa implica l'uso del protocollo UDP a livello di trasporto quando si vuole comunque gestire a livello

---

<sup>62</sup> Il progetto è tratto da un lavoro originale dello studente Oulai Li diplomatosi nell'a.s. 2017-2018 presso l'IIS "Giuseppe Peano" di Torino.

applicativo un protocollo connesso, affidabile, che faccia controllo di flusso e che gestisca una serie di errori.

Dopo aver letto con cura lo [RFC 1350](#), si analizzi il progetto considerando che è stata gestito solo il download di un file presente su di un server, lasciando come possibile esercizio l'implementazione del resto protocollo.

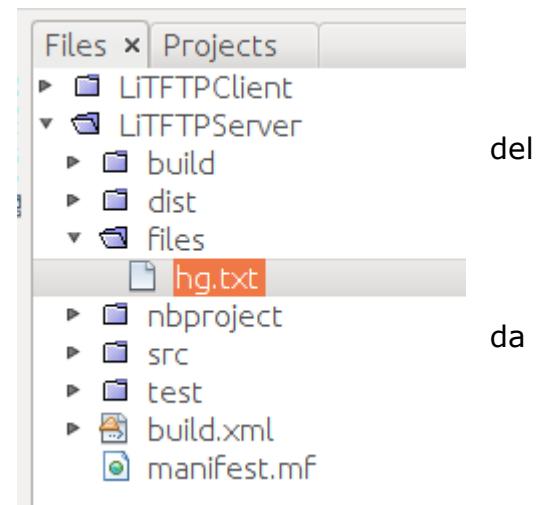
Affinché sia possibile utilizzare il progetto sarà necessario creare una cartella di nome files nella root del progetto LiTFTPServer, con all'interno il file trasferire, come mostrato nell'immagine di lato.

### **LiTFTPServer.java**

```
public class LiTFTPServer {  
    /**  
     * Istanzia un oggetto server e lancia il relativo thread.  
     * @param args argomenti a linea di comando  
     */  
    public static void main(String[] args) {  
        TFTPServer server = new TFTPServer();  
        Thread serverThread = new Thread(server);  
        serverThread.start();  
    }  
}
```

### **TFTPServer.java**

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.net.SocketException;  
import java.net.SocketTimeoutException;  
import java.util.Arrays;  
  
public class TFTPServer implements Runnable {  
    // Socket del server.  
    private DatagramSocket serverSocket;  
    // Porta di default del server.  
    private final int DEFAULT_SERVER_PORT = 7000;  
    // Cartella dove risiedono i file da trasferire.  
    private final String defaultDataFolderPath = "files";  
  
    // I codici delle operazioni (opcode) sono quelli dell'RFC 1350.  
    private final byte OP_RRQ = 1;      // opcode 1: read
```



del  
da

```
private final byte OP_DATA = 3;    // opcode 3: invio dati
private final byte OP_ACK = 4;     // opcode 4: conferma
private final byte OP_ERROR = 5;   // opcode 5: errore

// Questa costante è usata per indicare la terminazione del nome del file
// e della modalità di trasferimenti nel primo pacchetto inviato, che apre
// anche la connessione.
private final byte OP_ZEROBYTE = 0;

/***
 * Costruttore del server TFTP.
 */
public TFTPServer() {
    try {
        serverSocket = new DatagramSocket(DEFAULT_SERVER_PORT);
    } catch (SocketException ex) {
        throw new RuntimeException(ex);
    }
}

/***
 * Nel metodo run() viene selezionata l'operazione da effettuare.
 */
@Override
public void run() {
    // I pacchetti dei dati sono costituito da 2 byte che indicano la
    // operazione, 2 byte che indicano il numero di frammento e 512 byte di
    // dati, in totale 516 byte (RFC 1350).
    byte[] inBuf = new byte[516];
    DatagramPacket inDatagramPacket = new DatagramPacket(inBuf, 516);
    // Viene usato un do-while() in quanto deve essere ricevuto il primo
    // messaggio per creare la connessione e indicare il file sul quale
    // agire.
    do {
        System.out.println("Waiting...");
        try {
            // Il timeout di ricezione viene posto ad un tempo illimitato
            // per consentire ad un qualsiasi client di connettersi al server.
            serverSocket.setSoTimeout(0);
            // Il server è in attesa di ricevere una richiesta di connessione
            // da un client.
            serverSocket.receive(inDatagramPacket);
            // Dopo aver ricevuto il primo datagram per creare la connessione
            // si imposta un limite nel tempo di attesa della ricezione dei
            // datagram; sarà gestito un opportuno timer in combinazione con
            // questo tempo di timeout.
            serverSocket.setSoTimeout(4000);
        } catch (IOException ex) {
```

```
        throw new RuntimeException(ex);
    }
    // Viene controllata la posizione 1 del messaggio ricevuto in quanto
    // le possibili operazioni TFTP non superano la decina, il numero
    // utile dell'opcode è nel secondo byte.
    switch (inBuf[1]) {
        case OP_RRQ:
            // Richiesta di lettura del file.
            rrq(inDatagramPacket);
            break;
        // Qui dovranno essere poste le eventuali altre operazioni da
        // svolgere.
        default:
            // Lo opcode non è stato riconosciuto e quindi viene generato
            // un messaggio di errore con error code 4 e descrizione
            // "Illegal TFTP operation" (RFC 1350).
            sendError(4, "Illegal TFTP operation",
                      inDatagramPacket.getAddress(),
                      inDatagramPacket.getPort());
            break;
    }
} while (true);
}

/**
 * Lettura di un file da parte del client.
 * @param inDatagramPacket datagram iniziale che apre la connessione ed
 * indica il file su cui agire.
 */
private void rrq(DatagramPacket inDatagramPacket) {
    // File da aprire in lettura da cui leggere i dati da spedire.
    FileInputStream file;
    // Usato per la creazione del frammento da inviare, completo di header.
    byte[] outBuf = new byte[516];
    // Usato per la ricezione del primo messaggio di richiesta di connessione
    // e indicazione dell'operazione e del file sul quale agire.
    byte[] inBuf = inDatagramPacket.getData();
    // Creazione del datagram di uscita, recuperando l'indirizzo e il numero
    // di porta del client dal datagram appena ricevuto.
    DatagramPacket outDatagramPacket = new DatagramPacket(
        outBuf,
        outBuf.length,
        inDatagramPacket.getAddress(),
        inDatagramPacket.getPort());
    // Recupero del nome del file sul quale agire.
    String filename = getFilename(inBuf);
    // Recupero della modalità di trasferimento, per il nostro lavoro sarà
    // solo "octet".
```

```
String mode = getMode(inBuf, filename.length());
// Se la modalità di trasferimento è "octet" viene gestita la richiesta,
// qualsiasi altra modalità comporterà la non gestione della richiesta.
if (mode.equals("octet")) {
    try {
        // Costruzione del nome del file con il nome della cartella
        // (files), il simbolo di separazione usato dal sistema
        // (File.separator) e il nome del file.
        file = new FileInputStream(defaultDataFolderPath
            + File.separator
            + filename);
        // Usato per la conferma dei frammenti. Si ricordi che il numero
        // di frammento è composto da due byte e per questa ragione
        // ackFragmentNumber è un array di 2 byte.
        byte[] ackFragmentNumber = {0, 0};
        // Usato per la numerazione dei frammenti. Si ricordi che il
        // numero di frammento è composto da due byte e per questa ragione
        // fragmentNumber è un array di 2 byte.
        byte[] fragmentNumber = {0, 0};
        // Usato per stabilire la lunghezza dei dati inviati nel
        // frammento.
        int dataLength = 0;
        // Dati inviati nel frammento.
        byte[] data;
        // Opcode OP_DATA = 3: invio dati
        outBuf[1] = OP_DATA;
        System.out.print("Sending " + filename + "...");
        // Si cicla ripetendo le operazioni di invio del frammento e
        // ricezione della conferma fino a quando non sarà ricevuto un
        // frammento con un numero di byte inferiore a 512 byte, che
        // chiude la connessione.
        do {
            // Si dimensionano i dati a 512 byte
            data = new byte[512];
            // Se l'array delle conferme e quello dei numeri di frammenti
            // combaciano nella numerazione vuol dire che non si è in
            // attesa di una conferma di un precedente frammento ed è
            // quindi possibile inviarne un altro.
            if (Arrays.equals(ackFragmentNumber, fragmentNumber)) {
                // Lettura dal file di esattamente 512 byte (dimensione
                // dell'array data). Il numero di byte effettivamente
                // letti viene restituito come intero dalla read().
                dataLength = file.read(data);
                // Incremento del numero di frammento.
                fragmentNumber = fragmentNumberIncrease(fragmentNumber);
                // Preparazione del datagram da inviare usando il numero
                // di frammento, i dati, la lunghezza dei dati,
                // l'indirizzo e la porta del client.
            }
        } while (dataLength == 512);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
        outDatagramPacket = makeData(fragmentNumber, data,
                                      dataLength, inDatagramPacket.getAddress(),
                                      inDatagramPacket.getPort());
        // Invio del datagram con il nuovo frammento di dati.
        send(outDatagramPacket);
    }
    // Attesa di ricezione della conferma del frammento appena
    // inviato.
    receive(inDatagramPacket, outDatagramPacket);
    // Se il datagram ricevuto è di conferma (opcode = 4 posto
    // nella posizione 1 del datagram ricevuto) si copia il
    // numero di frammento, che parte dalla posizione 2 nello
    // array ackFragmentNumber. Si copiano esattamente 2 byte.
    if (inBuf[1] == OP_ACK) {
        System.arraycopy(inBuf, 2, ackFragmentNumber, 0, 2);
    }
} while (dataLength == 512);
System.out.println("Done!");
// Terminato il trasferimento si chiude il file.
file.close();
} catch (FileNotFoundException ex) {
    // Nel caso il file richiesto non esista viene generato un
    // messaggio di errore con error code 1 e descrizione "File not
    // found" (RFC 1350).
    sendError(1, filename, inDatagramPacket.getAddress(),
              inDatagramPacket.getPort());
} catch (IOException ex) {
    throw new RuntimeException(ex);
}
}
}

/**
 * Recupera il nome del file dal primo messaggio inviato dal client.
 *
 * @param inBuf i dati del messaggio
 * @return il nome del file
 */
private String getFilename(byte[] inBuf) {
    // Il nome del file inizia dopo lo opcode che occupa due byte in
    // posizione 0 e 1. Inoltre è terminato da uno 0, quindi per recuperare
    // il nome del file si inizia dalla posizione 2 e si cerca il valore 0.
    int filenameZeroPosition = 0;
    for (int i = 2; i < inBuf.length && filenameZeroPosition == 0; i++) {
        if (inBuf[i] == 0) {
            filenameZeroPosition = i;
        }
    }
}
```

```
// Si predisponde un array di byte di dimensione adeguata per contenere
// il nome del file, senza i due byte dell'opcode.
byte[] filenameBytes = new byte[filenameZeroPosition - 2];
// Copia del nome del file dall'array inBuf all'array filenameBytes.
System.arraycopy(inBuf, 2, filenameBytes, 0, filenameBytes.length);
// Converte il nome del file contenuto nell'array filenameBytes in una
// stringa e la restituisce al chiamente.
return new String(filenameBytes);
}

/**
 * Recupera la modalità di trasferimento, nel nostro caso solo "octet".
 * @param inBuf dati del messaggio ricevuto
 * @param filenameLength lunghezza del nome del file per posizionarsi
 * all'inizio del campo contenente la modalità di trasferimento
 * @return la modalità di trasferimento
 */
private String getMode(byte[] inBuf, int filenameLength) {
    // La modalità di trasferimento inizia dopo lo opcode e dopo lo zero che
    // termina il nome del file. In totale il campo mode inizierà dopo 2
    // byte di opcode più la lunghezza del file più lo 0 terminatore del
    // nome del file, quindi partirà dalla posizione (filenameLength + 3
    // byte).
    // La modalità di trasferimento è a sua volta terminata da uno 0.
    int modeZeroPosition = 0;
    for (int i = filenameLength + 3; i < inBuf.length
        && modeZeroPosition == 0; i++) {
        if (inBuf[i] == 0) {
            modeZeroPosition = i;
        }
    }
    // Si predisponde un array di byte di dimensione adeguata per contenere
    // la modalità di trasferimento.
    byte[] modeBytes = new byte[modeZeroPosition - filenameLength - 3];
    // Copia la modalità di trasferimenti dall'array inBuf all'array
    // modeBytes.
    System.arraycopy(inBuf, filenameLength + 3, modeBytes, 0, modeBytes.length);
    // Converte la modalità di trasferimento contenuto nell'array modeBytes
    // in una stringa e la restituisce al chiamente.
    String mode = new String(modeBytes);
    return mode;
}

/**
 * Costruisce e invia un messaggio di errore.
 * @param errorCode codice di errore (RFC 1350)
 * @param errorMsg messaggio di errore associato al codice errore (RFC 1350)
 * @param clientIP indirizzo IP del client

```

```
* @param clientPort numero di porta del client
*/
private void sendError(int errorCode, String errorMsg, InetAddress clientIP,
                      int clientPort) {
    // Array usato per la spedizione del messaggio di errore.
    byte[] buf;
    // La dimensione dei dati da inviare è data da 2 byte per l'opcode (05),
    // 2 byte per il codice di errore, la lunghezza del messaggio di errore
    // e lo 0 terminatore del messaggio di errore.
    buf = new byte[5 + errorMsg.getBytes().length];
    // OP_ERROR = 05: errore (RFC 1350). Copia il valore 5 nella posizione 1
    // dell'array da spedire.
    buf[1] = OP_ERROR;
    // Copia il codice del messaggio di errore nella posizione 3 dell'array
    // da spedire.
    buf[3] = (byte) errorCode;
    // Copia il messaggio di errore nell'array da spedire, dopo i due byte
    // del codice operazione e dopo i due byte del codice di errore.
    System.arraycopy(errorMsg.getBytes(), 0, buf, 4, errorMsg.getBytes().length);
    // Inserimento dello zero finale terminatore del messaggio di errore.
    buf[buf.length - 1] = OP_ZEROBYTE;
    // Creazione del datagram da inviare.
    DatagramPacket outDatagram = new DatagramPacket(buf, buf.length,
                                                      clientIP, clientPort);
    // Invio del messaggio di errore.
    send(outDatagram);
    // Chiusura del socket.
    //serverSocket.close();
}

/**
 * Invia un datagram.
 * @param datagram datagram da inviare
 */
private void send(DatagramPacket datagram) {
    try {
        serverSocket.send(datagram);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

/**
 * Incrementa il numero che conteggia i frammenti.
 * @param fragmentNumber array di 2 byte contenente il numero di frammenti
 * inviati finora.
 * @return array di byte contenete il nuovo numero di frammento
 */

```

```
private byte[] fragmentNumberIncrease(byte[] fragmentNumber) {
    // Se il byte contenete la cifra meno (posizione 1) significativa è
    // andata in overflow si incrementa di 1 il byte della cifra più
    // significativa (posizione 0) e si pone a zero la cifra meno
    // significativa.
    if (fragmentNumber[1] == -1) {
        fragmentNumber[0]++;
        fragmentNumber[1] = 0;
    // Altrimenti si incrementa la cifra meno significativa (posizione 1).
    } else {
        fragmentNumber[1]++;
    }
    // Si restituisce l'array di byte contenete il nuovo numero di frammento.
    return fragmentNumber;
}

/**
 * Ricezione di un datagram con la gestione del timeout.
 * @param inDatagramPacket datagram in ricezione
 * @param lastSentDatagram ultimo datagram inviato
 * @throws SocketTimeoutException
 */
private void receive(DatagramPacket inDatagramPacket,
                     DatagramPacket lastSentDatagram)
                     throws SocketTimeoutException {
    // Inizializzazione del timeout.
    int timeouts = 0;
    // Si prevedono 8 cicli di attesa da 4 secondi per dichiarare fallita la
    // ricezione del datagram e sollevare la relativa eccezione.
    // I 4 secondi sono stati impostati nel metodo run() tramite il
    // metodo setSoTimeout().
    do {
        try {
            // Ricezione del datagram.
            serverSocket.receive(inDatagramPacket);
        } catch (SocketTimeoutException ex) {
            // Allo scadere dei 4 secondi si tenta di spedire l'ultimo
            // datagram già inviato e si incrementa il timeout.
            send(lastSentDatagram);
            timeouts++;
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    } while (timeouts > 0 && timeouts < 8);
    // Se il timeout ha raggiunto gli 8 tentativi si dichiara fallita la
    // ricezione e si solleva la relativa eccezione.
    if (timeouts >= 8) {
        throw new SocketTimeoutException();
    }
}
```

```
        }
    }

/***
 * Creazione di un nuovo frammento da inviare al client.
 * @param fragmentNumber numero del frammento
 * @param data dati del frammento
 * @param dataLength lunghezza dei dati del frammento
 * @param ipAddress indirizzo IP del client
 * @param port numero di porta del client
 * @return il datagram da inviare
 */
private DatagramPacket makeData(byte[] fragmentNumber, byte[] data,
                                int dataLength, InetAddress ipAddress, int port) {
    // I frammenti completi di header devono essere lunghi al massimo 516
    // byte.
    byte[] buf = new byte[516];
    // Opcode = 3: invio di dati
    buf[1] = OP_DATA;
    // Copia del numero di frammento nell'array di spedizione
    System.arraycopy(fragmentNumber, 0, buf, 2, 2);
    // Copia dei dati nell'array di spedizione.
    System.arraycopy(data, 0, buf, 4, 512);
    // Creazione del datagram e restituzione al chiamante.
    return new DatagramPacket(buf, 4 + dataLength, ipAddress, port);
}
}
```

**LiTFTPClient.java**

```
public class LiTFTPClient {  
    private final static String serverIP = "localhost";  
    private final static int serverPort = 7000;  
    private final static String filename = "hg.txt";  
  
    /**  
     * Istanzia un oggetto client che effettua immediatamente il download  
     * del file specificato in filename.  
     * @param args argomenti a linea di comando  
     */  
    public static void main(String[] args) {  
        TFTPClient client = new TFTPClient(serverIP, serverPort);  
        client.rrq(filename);  
    }  
}
```

**TFTPClient.java**

```
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.net.SocketTimeoutException;  
  
public class TFTPClient {  
  
    // Socket del client.  
    private DatagramSocket clientSocket;  
    // Indirizzo IP del server.  
    private InetAddress serverIP;  
    // Porta di ascolto del server.  
    private int serverPort;  
  
    // I codici delle operazioni (opcode) sono quelli dell'RFC 1350.  
    private final byte OP_RRQ = 1;      // opcode 1: read  
    private final byte OP_DATA = 3;     // opcode 3: invio dati  
    private final byte OP_ACK = 4;      // opcode 4: conferma  
    private final byte OP_ERROR = 5;    // opcode 5: errore  
  
    // Usato nel messaggio iniziale come terminatore del nome del file e della  
    // modalità di trasferimento:  
    //          2 bytes      string      1 byte      string      1 byte  
    //          -----  
    //          | Opcode |  Filename  |  0  |      Mode   |  0  |  
    //          -----  
    private final byte ZEROBYTE = 0;
```

```
/**  
 * Costruttore del client TFTP.  
 *  
 * @param serverURL URL del server  
 * @param serverPort porta di ascolto del client  
 */  
public TFTPClient(String serverURL, int serverPort) {  
    try {  
        // Creazione del socket del client  
        this.clientSocket = new DatagramSocket();  
        // Gestione dell'indirizzo del server.  
        this.serverIP = InetAddress.getByName(serverURL);  
        // Salvataggio della porta di ascolto del server.  
        this.serverPort = serverPort;  
        // Impostazione di un tempo limite di attesa per il metodo  
        // receive().  
        this.clientSocket.setSoTimeout(4000);  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
}  
  
/**  
 * Lettura di un file che si trova sul server.  
 *  
 * @param filename nome del file da scaricare  
 */  
public void rrq(String filename) {  
    // Datagram usato per l'invio.  
    DatagramPacket outDatagramPacket;  
    // Datagram usato per la ricezione.  
    DatagramPacket inDatagramPacket;  
    // File da aprire in scrittura su cui salvare i dati ricevuti.  
    FileOutputStream file;  
    // Usato per numerare i frammenti. Si ricordi che il numero di frammen-  
    // to è composto da due byte e per questa ragione fragmentNumber è un  
    // array di 2 byte.  
    byte[] fragmentNumber = new byte[2];  
    // Dati ricevuti nel frammento.  
    byte[] data;  
    // Usato per la creazione del frammento da ricevere, completo di header.  
    byte[] inBuf;  
  
    // Creazione del messaggio di richiesta al server, contenente il nome  
    // del file e la modalità di trasferimento (solo "octet" nel nostro  
    // caso, come da specifica dell'RFC 1350. Questo messaggio effettua  
    // anche l'apertura della connessione.
```

```
// Il codice dell'operazione (opcode) corrisponde a OP_RRQ = 1 che è la
// richiesta di lettura, come da specifica dell'RFC 1350.
outDatagramPacket = makeRequest(OP_RRQ, filename, "octet");
// Invio del messaggio di connessione e richiesta.
send(outDatagramPacket);
try {
    // Apertura in scrittura del file da ricevere. In questo caso il
    // file sarà salvato nella root del progetto; sarebbe opportuno
    // almeno predisporre una cartella apposita per la ricezione, come
    // fatto nel server.
    file = new FileOutputStream(filename);
    // Si cicle ripetendo l'operazione di ricezione del frammento e
    // spedizione della relativa conferma fino a quando non sarà ricevu-
    // to un frammento con un numero di byte inferiore a 512 byte, che
    // chiude la connessione.
    do {
        // Istanziazione dell'oggetto che conterrà i dati ricevuti.
        // Come stabilito nell'RFC 1350 il payload di ogni messaggio non
        // potrà essere più lungo di 512 byte.
        data = new byte[512];
        // Istanziazione dell'oggetto che conterrà l'intero messaggio
        // ricevuto, costituito da un header di 4 byte e un payload di
        // 512 byte.
        inBuf = new byte[516];
        // Istanziazione del datagram packet usato per la ricezione.
        inDatagramPacket = new DatagramPacket(inBuf, 516);
        // Attesa di ricezione del frammento del file richiesto.
        receive(inDatagramPacket, outDatagramPacket);
        // Se il datagram ricevuto contiene dei dati (OP_DATA = 3) allo-
        // ra si dovrà provvedere a salvare i dati nel file.
        if (inBuf[1] == OP_DATA) {
            // Copia del numero di frammento che si trova nella posizio-
            // ne 2 di inBuf, all'interno dell'array fragmentNumber a
            // partire dalla sua posizione 0. Si copiano esattamente due
            // byte.
            System.arraycopy(inBuf, 2, fragmentNumber, 0, 2);
            // Copia dei dati contenuti nel frammento che partono dalla
            // posizione 4 di inBuf, all'interno dell'array data a par-
            // tire dalla sua posizione 0. si copiano esattamente 512
            // byte.
            System.arraycopy(inBuf, 4, data, 0, 512);
            // Se la lunghezza del messaggio ricevuto è superiore a 4
            // byte, vuol dire che ci sono dei dati utili da registrare
            // nel file. I primi 4 byte contengono l'opcode e il numero
            // di frammento.
            if (inDatagramPacket.getLength() > 4) {
                // Nel file vengono scritti i dati presenti nell'array
                // data a partire dalla sua posizione 0, per una lun-
```

```
// ghezza pari a quella del messaggio ricevuto esclusi
// i primi 4 byte che contengono l'opcode e il numero
// del frammento.
    file.write(data, 0, inDatagramPacket.getLength() - 4);
}
// Creazione del messaggio di conferma.
outDatagramPacket = makeACK(
    fragmentNumber,
    inDatagramPacket.getPort());
// Invio del messaggio di conferma.
send(outDatagramPacket);
}
} while (inDatagramPacket.getLength() == 516);
// Chiusura del file in quanto è stato ricevuto tutto.
file.close();
// Chiusura del socket del client.
clientSocket.close();
} catch (IOException ex) {
    throw new RuntimeException(ex);
}
}

/**
 * Creazione del primo messaggio che stabilisce la connessione e indica il
 * nome del file da gestire e la modalità di trasferimento (solo "octet" nel
 * nostro caso).
 *
 * @param request codice dell'operazione (OP_RRQ = 1)
 * @param filename nome del file da gestire
 * @param mode modalità di trasferimento (solo "octet" nel nostro caso)
 * @return il datagram da inviare
 */
private DatagramPacket makeRequest(byte request, String filename,
        String mode) {
    // Dichiarazione del buffer contenente i dati da inviare.
    byte[] buf;
    // Istanziazione del buffer contenente i dati da inviare, con una dimensione adeguata a contenere il codice dell'operazione (2 byte), il nome del file, lo 0 terminatore del nome file, la modalità di trasferimento e lo 0 terminatore della modalità di trasferimento.
    buf = new byte[4 + filename.getBytes().length + mode.getBytes().length];
    // Copia dell'opcode nella posizione 1 del buffer.
    buf[1] = request;
    // Copia del nome del file nel buffer, a partire dalla posizione 2.
    System.arraycopy(filename.getBytes(), 0, buf, 2,
        filename.getBytes().length);
    // Terminazione del nome del file con lo 0.
    buf[2 + filename.getBytes().length] = ZERODYTE;
```

```
// Copia della modalità di trasferimento nel buffer a partire dalla posizione successiva a quella dello 0 che termina il nome del file.
System.arraycopy(mode.getBytes(), 0, buf,
                 3 + filename.getBytes().length,
                 mode.getBytes().length);
// Terminazione della modalità di trasferimento con lo 0.
buf[buf.length - 1] = ZEROBYTE;
// Creazione del datagram di spedizione.
DatagramPacket outDatagramPacket = new DatagramPacket(buf, buf.length,
                                                       serverIP, serverPort);
// Restituzione del datagram da spedire.
return outDatagramPacket;
}

/**
 * Creazione di un messaggio di conferma.
 *
 * @param fragmentNumber numero del frammento
 * @param serverPort porta di ascolto del server
 * @return il datagram da inviare
 */
private DatagramPacket makeACK(byte[] fragmentNumber, int serverPort) {
    // Dichiarazione e istanziazione del buffer per la conferma che è lunga 4 byte.
    byte[] buf = new byte[4];
    // Copia del codice dell'operazione nella posizione 1 del buffer
    // (OP_ACK = 04).
    buf[1] = OP_ACK;
    // Copia del numero di frammento a partire dalla posizione 2 del buffer,
    // per due byte.
    System.arraycopy(fragmentNumber, 0, buf, 2, 2);
    // Creazione del messaggio di conferma da inviare al server.
    DatagramPacket datagramPacket = new DatagramPacket(buf, 4, serverIP,
                                                       serverPort);
    // Restituzione del datagram da spedire.
    return datagramPacket;
}

/**
 * Invia un datagram.
 *
 * @param datagram datagram da inviare
 */
private void send(DatagramPacket datagram) {
    try {
        clientSocket.send(datagram);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

```
        }
    }

/***
 * Ricezione di un datagram con la gestione del timeout.
 *
 * @param inDatagramPacket datagram in ricezione
 * @param lastSentDatagram ultimo datagram spedito
 */
private void receive(DatagramPacket inDatagramPacket,
                     DatagramPacket lastSentDatagram)
    throws SocketTimeoutException {
    // Inizializzazione del timeout.
    int timeouts = 0;
    // Si prevedono 8 cicli di attesa da 4 secondi per dichiarare fallita la
    // ricezione del datagram e sollevare la relativa eccezione.
    // I 4 secondi sono stati impostati nel costruttore tramite il
    // metodo setSoTimeout().
    do {
        try {
            clientSocket.receive(inDatagramPacket);
        } catch (SocketTimeoutException ex) {
            send(lastSentDatagram);
            timeouts++;
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    } while (timeouts > 0 && timeouts < 8);
    // Se il timeout ha raggiunto gli 8 tentativi si dichiara fallita la
    // ricezione e si solleva la relativa eccezione.
    if (timeouts >= 8) {
        throw new SocketTimeoutException();
    }
}
```

## F - Android e app

# Android e dispositivi mobili [LIBRO]

**STUDIARE:** P. Camagni, R. Nikolassy, *Nuovo Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2021.

## Unità 8 - Android e i dispositivi mobili

- **8.1 Dispositivi e reti mobili pp.468-477**
  - Premessa
  - Reti mobili
  - software per dispositivi mobili
  - Sistemi operativi per dispositivi mobili
  - ambienti di sviluppo per dispositivi mobili
  - Xamarin
  - React native
  - Unity
- **8.2 Android: un sistema operativo per applicazioni mobili (area digitale)**
  - Lab 1 Android Studio: installazione e configurazione
  - Lab 2 L'interfaccia grafica di Android Studio
  - Lab 3 Utilizzare i widget nelle app Android
  - Lab 4 Un'app completa: la calcolatrice
  - Lab 5 Utilizzare i sensori nelle app

Vedere il seguente sito:

- [CorsoAndroid.it](http://CorsoAndroid.it)

## CLIL - Developing Android Apps

You can study how to develop Android Apps through the Udacity course [Developing Android Apps](#).

## G - Web Service REST usando PHP

# Creare un client REST

I cinque principi analizzati nelle sezioni precedenti ci consentono di definire un **Web service** aderente ad una architettura in stile **REST**.

Inizialmente creeremo un **client REST** che interagisca con un Web service di tipo REST esistente, come il server di *geocoding* offerto da Google e, per fare ciò si avrà bisogno di:

- un client HTTP
- le URI delle risorse a cui vogliamo accedere
- la capacità di interpretare il formato della rappresentazione delle risorse.

**Creare un client REST** è in un certo senso simile a realizzare un piccolo browser Web specializzato nell'interpretazione delle risorse messe a disposizione dal Web service.

## XML over HTTP

Interrogare un Web service esistente tramite un programma sviluppato con il linguaggio Java prevede l'uso delle classi astratte *URLConnection* e della sua classe derivata *HttpURLConnection*, che aggiunge alcuni metodi per la gestione delle specifiche caratteristiche del protocollo HTTP.

Di seguito vengono descritti i metodi fondamentali della classe *URLConnection*.

Metodo	Descrizione
URLConnection	Costruttore. Richiede di specificare l'URL della risorsa Web a cui connettersi
setRequestProperty	Imposta un <i>header</i> nella richiesta da inviare costituito da coppie <i>nome/valore</i>
setConnectTimeout	Imposta il tempo massimo di attesa in millisecondi per la richiesta di connessione
setReadTimeout	Imposta il tempo massimo di attesa in millisecondi per la ricezione dei dati
setDoInput	Abilita/disabilita la ricezione dati dalla risorsa Web
setDoOutput	Abilita/disabilita la trasmissione dati verso la risorsa Web
connect	Esegue la connessione alla risorsa Web identificata dallo URL fornito come argomento del costruttore
getContentLength	Restituisce il valore dello <i>header content-length</i> della

	risposta
getContentType	Restituisce il valore dello <i>header content-type</i> della risposta
getContentEncoding	Restituisce il valore dello <i>header content-encoding</i> della risposta
getDate	Restituisce il valore dello <i>header date</i> della risposta
getExpiration	Restituisce il valore dello <i>header expires</i> della risposta
getLastModified	Restituisce il valore dello <i>header last-modified</i> della risposta
getInputStream	Restituisce lo <i>stream</i> per la ricezione del contenuto del <i>body</i> della risposta
getOutputStream	Restituisce lo <i>stream</i> per la trasmissione del <i>body</i> della richiesta

Di seguito vengono descritti alcuni dei metodi specifici della classe *HttpURLConnection* che si aggiungono ai precedenti.

Metodo	Descrizione
setRequestMethod	Imposta il metodo HTTP di richiesta (GET, POST, HEAD, OPTIONS, PUT, DELETE o TRACE)
getResponseCode	Restituisce lo <i>status-code</i> della risposta HTTP
getResponseMessage	Restituisce il messaggio relativo allo <i>status-code</i> della risposta HTTP
disconnect	Esegue la disconnessione della risorsa Web

## Client REST usando le API di Google

Come abbiamo visto in precedenza Google rende disponibili dei Web service per effettuare *geocoding*. La richiesta effettuata direttamente al server [maps.googleapis.com](http://maps.googleapis.com) tramite un browser, effettuando una richiesta HTTP tramite l'URL seguente

`http://maps.googleapis.com/maps/api/geocode/xml?address=corso+Venezia.29+Torino+Italia&sensor=false`

restituisce un file XML che comprende i dati dell'indirizzo e le coordinate geografiche poste nella richiesta.

Utilizzando il linguaggio Java e le classi astratte *URLConnection* e *HttpURLConnection* è possibile creare un'applicazione che gestisca automaticamente la risposta; invece di visualizzarla nel browser, sarà salvata in un file per future elaborazioni. Il metodo *main* esemplifica l'uso della classe *Geocoding* richiedendo l'inserimento dell'indirizzo da localizzare e il nome del file XML in cui salvare la risposta.

L'istanza della classe creerà automaticamente il file XML solo se si riceverà una risposta che dichiari la buona riuscita dell'operazione, con *status-code* HTTP di tipo **200 OK**.

Il parser DOM è utilizzato per estrarre la latitudine e la longitudine dell'indirizzo fornito in input, simulando una possibile utilizzazione dei dati ricevuti dal Web service.

```
import java.net.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

class GeocodingException extends Exception {
}

public class Geocoding {
    // prefisso dell'URL
    private String prefix =
        "http://maps.googleapis.com/maps/api/geocode/xml?address=";
    // Suffisso dell'URL
    private String suffix = "&sensor=false";
    // URL passato come parametro al costruttore
    private String url;
    // Nome del file XML in cui salvare la risposta restituite dal Web

    // service
    private String filename;
    // Indica se il salvataggio del file XML è stato effettuato
    private boolean saved = false;
    // Indica se il parsing della risposta è stata effettuata
    private boolean parsed = false;
    // Latitudine dell'indirizzo
    private double latitude;
    // Longitudine della risposta
    private double longitude;

    /**
     * Costruttore.
     */
```

```
* Crea la richiesta HTTP da inviare al Web service, utilizzando il
* metodo GET, si connette al server e se la risposta ricevuta è di tipo
* 200 OK crea il file XML per salvare la risposta inviata.
* @param address URL del Web service
* @param filename nome del file XML in cui salvare la risposta
*/
public Geocoding(String address, String filename) {
    // URL del Web service
    URL server;
    // Oggetto remoto identificato dallo URL
    HttpURLConnection service;

    // Stream di ricezione dalla risorsa Web
    BufferedReader input;

    // Stream di scrittura del file
    BufferedWriter output;

    // status-code HTTP
    int status;

    // Singola linea da scrivere nel file
    String line;

    this.filename = filename;
    try {
        // Costruzione dello URL di interrogazione del Web Service.
        // Il metodo statico encode() della classe URLEncoder effettua
        // lo URL-encoding della stringa fornita come argomento.
        url = prefix + URLEncoder.encode(address, "UTF-8") + suffix;
        System.out.println(url);
        server = new URL(url);
        // openConnection() è un metodo della classe URL. Restituisce
        // una istanza della classe URLConnection che rappresenta una
        // connessione all'oggetto remoto identificato dallo URL
        service = (HttpURLConnection)server.openConnection();
        // Impostazione degli header della richiesta.
        service.setRequestProperty("Host", "maps.googleapis.com");
        service.setRequestProperty("Accept", "application/xml");
        service.setRequestProperty("Accept-Charset", "UTF-8");
        // Impostazione del metodo GET per la richiesta.
        service.setRequestMethod("GET");
        // Attivazione della ricezione.
        service.setDoInput(true);
        // Connessione al Web Service.
        service.connect();
        // Verifica lo stato della risposta.
        status = service.getResponseCode();
        if (status != 200) {
```

```
        return; //non OK
    }
    // Apertura dello stream di ricezione dalla risorsa web.
    input = new BufferedReader(new

        InputStreamReader(service.getInputStream(), "UTF-8"));
    // Apertura dello stream per la scrittura del file.
    output = new BufferedWriter(new FileWriter(filename));
    // Ciclo di lettura dal Web e scrittura sul file.
    while((line = input.readLine()) != null) {
        output.write(line);
        output.newLine();
    }
    input.close();
    output.close();
    saved = true;
} catch (IOException e) {
}

}

/**
 * Parser della risposta restituita dal Web service.
 * Se la risposta è di tipo 200 OK, identifica i tag lat e lng per
 * recuperarne i valori.
 * @throws GeocodingException
 * @throws org.xml.sax.SAXException
 */
private void parseXML()

    throws GeocodingException, org.xml.sax.SAXException {
Document document;
Element root;
NodeList list;

if (!saved) {
    throw new GeocodingException();
}
try {
    DocumentBuilderFactory factory =

        DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    // Costruzione albero DOM del file XML
    document = builder.parse(filename);
    root = document.getDocumentElement();
    // Ricerca elemento "status"
    list = root.getElementsByTagName("status");
    if (list != null && list.getLength() > 0) {
        // verifica esito (valore elemento "status" = "OK")
```

```
        if (list.item(0).getFirstChild().getNodeValue().

                equalsIgnoreCase("OK")) {
            // ricerca elemento "location"
            list = root.getElementsByTagName("location");
            if (list != null && list.getLength() > 0) {
                Element loc = (Element) list.item(0);
                // ricerca elemento "lat"
                NodeList lat = loc.getElementsByTagName("lat");
                // conversione in valore numerico
                latitude = Double.parseDouble(lat.item(0).

                        getFirstChild().getNodeValue());
                // ricerca elemento "lng"
                NodeList lng = loc.getElementsByTagName("lng");
                // conversione in valore numerico
                longitude = Double.parseDouble(lng.item(0).

                        getFirstChild().getNodeValue());
                parsed = true;
            }
        }

    } catch (IOException e) {
        throw new GeocodingException();
} catch (ParserConfigurationException e) {
    throw new GeocodingException();
}

}

/***
 * Metodo che restituisce la longitudine dell'indirizzo.
 * @return longitude
 * @throws GeocodingException
 */
public double getLongitude() throws GeocodingException {
    if (!saved) {
        throw new GeocodingException();
    }
    if (!parsed) {
        try {
            parseXML();
        } catch (SAXException ex) {
            Logger.getLogger(Geocoding.class.getName()).
                log(Level.SEVERE, null, ex);
        }
    }
}
```

```
    return longitude;

}

/***
 * Metodo che restituisce la latitudine dell'indirizzo
 * @return latitude
 * @throws GeocodingException
 */
public double getLatitude() throws GeocodingException {
    if (!saved) {
        throw new GeocodingException();
    }
    if (!parsed) {
        try {
            parseXML();
        } catch (SAXException ex) {
            Logger.getLogger(Geocoding.class.getName()).
                log(Level.SEVERE, null, ex);
        }
    }
    return latitude;
}

public static void main(String[] args) {
    String addr;
    String fn;
    try {
        BufferedReader tastiera = new BufferedReader(new
            InputStreamReader(System.in));
        // Richiesta dell'indirizzo di cui si vogliono visualizzare le
        // coordinate.
        System.out.println("Inserire l'indirizzo: ");
        addr = tastiera.readLine();
        // Richiesta del nome del file XML che si creerà con la risposta
        // fornita dal Web service.
        System.out.println("Inserire il nome del file");
        fn = tastiera.readLine();
        // Istanziazione di un oggetto Geocoding.
        Geocoding g = new Geocoding(addr, fn);
        // Visualizzazione delle coordinate dell'indirizzo.
        System.out.println("(" + g.getLatitude() + ";" +
            g.getLongitude() + ")");
    } catch (IOException e) {
        System.out.println(e.getMessage());
        System.out.println("Errore nell'indirizzo o nel nome del file");
        System.exit(1);
    }
}
```

```
    } catch (GeocodingException e) {
        System.out.println("Errore invocazione Web Service");
    }
}
```

## PHP REST API

### CLIL Comprehension - What a RESTful API is

Watch the following video in order to understand what a RESTful API is.

[What Is A RESTful API? Explanation of REST & HTTP](#)

### CLIL Reading - RESTful API using PHP

Read the article [RESTful API](#) and try to implement or, at least try to understand how the RESTful API shown is written.

In order to use the REST API proposed in the article you have to modify the Apache2 configuration file (in Ubuntu 19.10 is located in /etc/apache2/apache2.conf) allowing the URL rewriting. In this way the rules wrote in the .htaccess file are allowed to rewrite the URL in RESTful style.

In the example proposed the file .htaccess is this

```
# Turn rewrite engine on
Options +FollowSymlinks
RewriteEngine on

# map neat URL to internal URL
RewriteRule ^mobile/list/?$ RestController.php?view=all [nc, qsa]
RewriteRule ^mobile/list/([0-9]+)/?$ RestController.php?view=single&id=$1
[nc, qsa]
```

and the modification that I made in apache2.conf is the following one

```
<Directory /var/www/html>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

The folder html is the root folder for all my Web application.

### CLIL Comprehension - PHP REST API

Watch the following videoS in order to understand how create a PHP REST API.

[Vai all'indice](#)

[PHP REST API From Scratch](#)

## H - Vecchi esempi di Web service

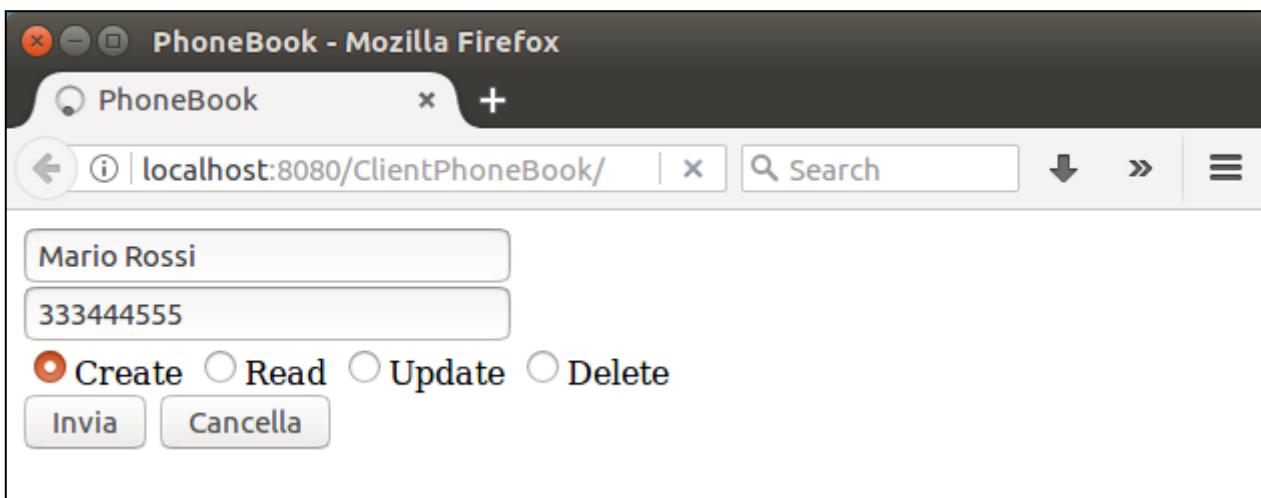
# Rubrica telefonica REST su file

Le due **servlet** seguenti implementano un **client** e un **server REST** per la gestione dei numeri di telefono memorizzati all'interno di una rubrica telefonica.

Il **client** e il **server** sono pensati come due applicazioni indipendenti che comunicano fra di loro per manipolare dei dati. L'interazione con l'utente è ridotta al minimo, con il solo scopo di avviare l'interazione con il Web service e controllare il risultato delle operazioni effettuate.

La rubrica telefonica è stata gestita come file XML, *phonebook.xml*.

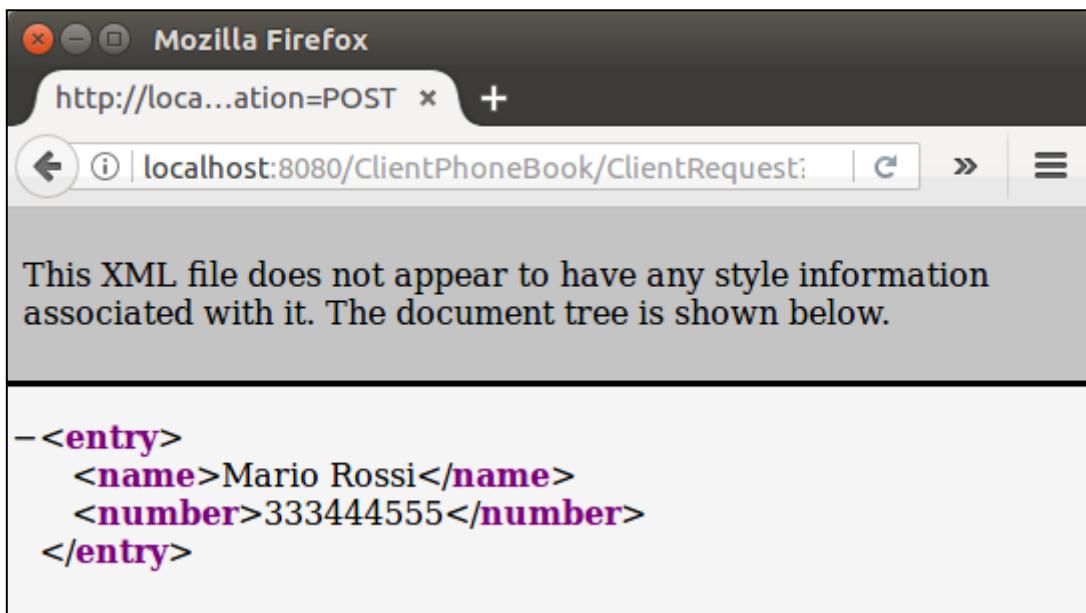
Il **client** differenzia la richiesta inviata al Web service, a seconda della scelta effettuata nel form iniziale.



Nel caso fosse richiesta una operazione di **creazione** il **client** genera una richiesta utilizzando il metodo **POST** con l'invio del nome e del numero di telefono da aggiungere. Il **server** modifica il file *phonebook.xml* aggiungendo un elemento *entry* al fondo del file. In questo Web service una operazione di **creazione** non prevede alcun controllo sull'esistenza di un contatto con gli stessi dati, **la risorsa viene modificata ad ogni invocazione**.

```
--<phonebook>
--<entry>
  <name>Gianni Verdi</name>
  <number>322565656</number>
</entry>
--<entry>
  <name>Marta Bruni</name>
  <number>344878787</number>
</entry>
--<entry>
  <name>Paolo Bianchi</name>
  <number>399525252</number>
</entry>
--<entry>
  <name>Mario Rossi</name>
  <number>333444555</number>
</entry>
</phonebook>
```

Il **server** invia come risposta il nuovo contatto inserito, sempre in formato XML.

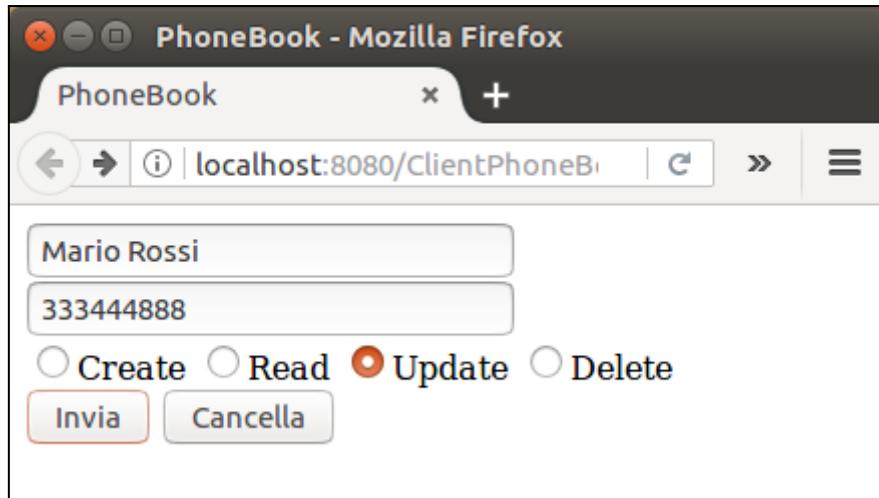


Nel caso fosse richiesta una operazione di **lettura** il **client** genera una richiesta utilizzando il metodo **GET** con l'invio del solo nome. Il **server** legge il file *phonebook.xml* inviando una risposta XML con il nome e il numero di telefono del contatto.

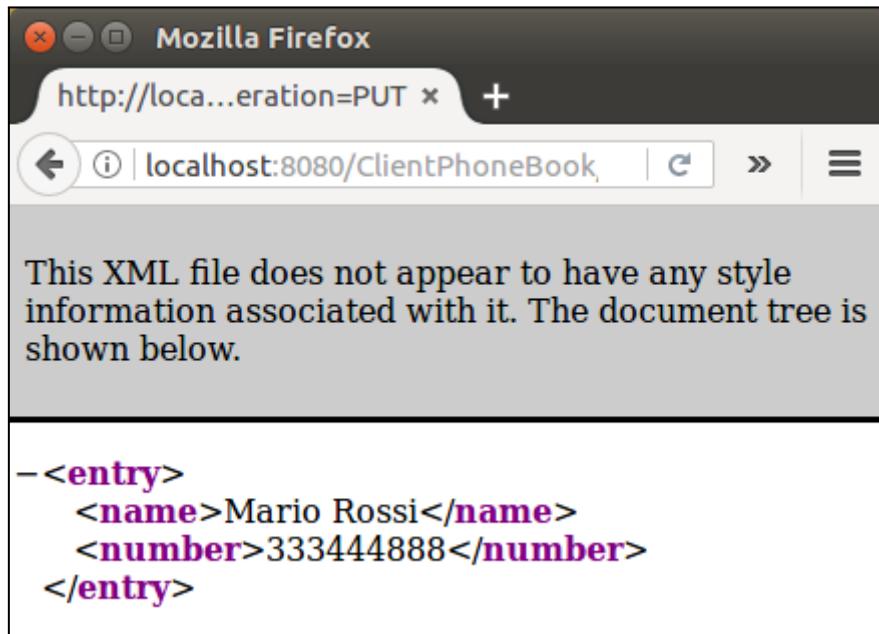
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
--<entry>
<name>Mario Rossi</name>
<number>333444555</number>
</entry>
```

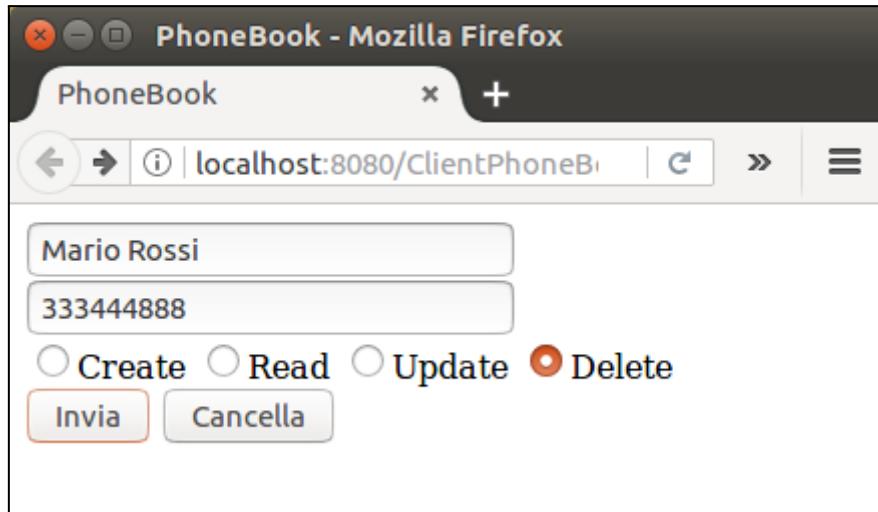
Una richiesta di **aggiornamento** prevede da parte del **client** la generazione di una richiesta utilizzando il metodo **PUT**, con l'invio del nome e del nuovo numero di telefono. Il **server** cerca il nome nel file *phonebook.xml*, modificando il numero di telefono solo se la ricerca risulta andata a buon fine. In questo Web service l'operazione di **aggiornamento** è **idempotente**, cioè **la risorsa *phonebook.xml* non viene modificata se viene ripetuta più volte la medesima richiesta**, a differenza dell'operazione di **creazione** che, come detto in precedenza, aggiunge una entry ad ogni richiesta, anche se era già stata inserita in precedenza.



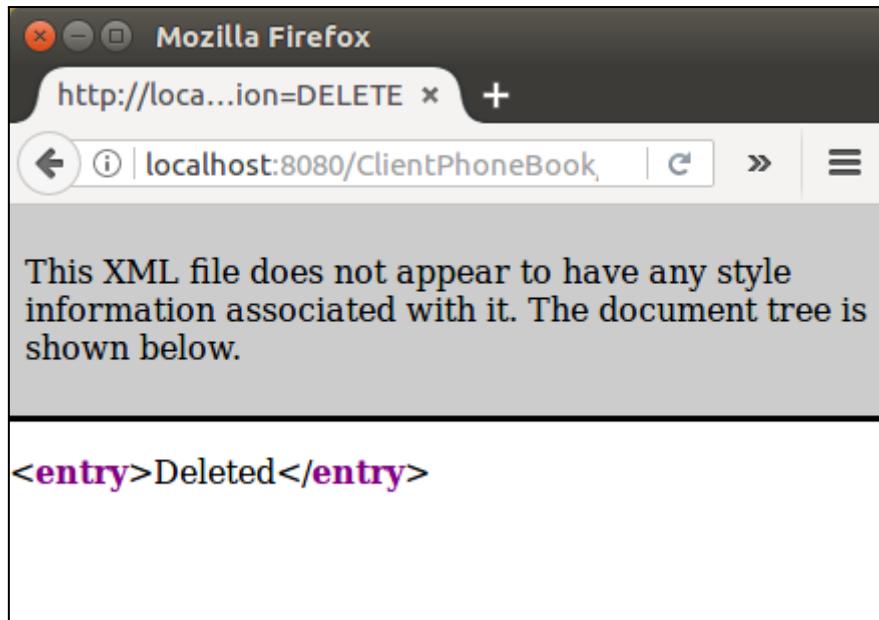
```
-<phonebook>
-<entry>
  <name>Gianni Verdi</name>
  <number>322565656</number>
-</entry>
-<entry>
  <name>Marta Bruni</name>
  <number>344878787</number>
-</entry>
-<entry>
  <name>Paolo Bianchi</name>
  <number>399525252</number>
-</entry>
-<entry>
  <name>Mario Rossi</name>
  <number>333444888</number>
-</entry>
</phonebook>
```



Infine, la richiesta di **cancellazione** prevede da parte del **client** la generazione di una richiesta utilizzando il metodi **DELETE**, con l'invio del nome e del numero di telefono del contatto da cancellare. Il **server** cerca nel file *phonebook.xml* una *entry* con entrambe le corrispondenze e, solo in questo caso elimina il contatto. In questo Web service, **se non venisse identificata una esatta corrispondenza** sia nel nominativo, sia nel numero di telefono, la **risorsa non verrebbe modificata**.



```
-<phonebook>
-<entry>
  <name>Gianni Verdi</name>
  <number>322565656</number>
</entry>
-<entry>
  <name>Marta Bruni</name>
  <number>344878787</number>
</entry>
-<entry>
  <name>Paolo Bianchi</name>
  <number>399525252</number>
</entry>
</phonebook>
```



Per rendere **seriale l'accesso alla risorsa *phonebook.xml*** da parte di più client, nel **server** è stato creato un **membro statico**, istanza che gestisce l'albero DOM e la relativa parsificazione e riscrittura. L'accesso ai **metodi dell'oggetto statico** è sempre inserito in un **blocco synchronized** per rendere mutualmente esclusivo l'accesso alla risorsa e **atomica l'invocazione dei metodi** da parte di thread concorrenti.

Di seguito viene fornito il codice del lato **server**.

### ServletPhoneBook.java

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

// Se non si usa web.xml possono essere impostate le annotazioni indicando il
// nome della servlet nell'attributo name, che corrisponde al tag
// <servlet-name> di web.xml, e il pattern dell'URL nell'attributo urlPattern,
// che corrisponde al tag <url-pattern> di web.xml. Impostando l'attributo
// urlPatern o il tag <url-pattern> a "/" permette di intercettare richieste
// con sezioni terminali dell'URL diverse, la cui struttura viene analizzata
// nel codice dei metodi della servlet.
@WebServlet(name = "ServletPhoneBook", urlPatterns = {"/*"})
public class ServletPhoneBook extends HttpServlet {
    // Istanza statica all'albero DOM del file phonebook.xml.
    // L'accesso ai metodi dell'oggetto sono gestiti tramite un blocco
    // synchronized per rendere atomica l'invocazione da parte di thread
    // concorrenti.
    private final static ParserDOM PARSER = new ParserDOM();
```

```
// richiesta POST - CREATE
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {

    // recupero del body della richiesta
    String name = request.getParameter("name").replaceAll("\\+", " ");
    String number = request.getParameter("number").replaceAll("\\+", " ");

    if (name == null || number == null) {
        response.sendError(400, "Malformed XML in doPost - null
parameters");
        return;
    }
    if (name.isEmpty() || number.isEmpty()) {
        response.sendError(400, "Malformed XML in doPost - empty
parameters");
        return;
    }

    synchronized (PARSER) {
        PARSER.loadDOM();
        PARSER.createEntry(name, number);
        PARSER.writeDOM();
    }

    response.setStatus(201);      // OK
    response.setContentType("text/xml; charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<?xml version=\"1.0\" encoding=\"UTF-8\""
standalone="no"?>" +
                + "<entry>\n\t<name>" + name.replaceAll("\\+", " ") +
                + "</name>\n\t<number>" + number.replaceAll("\\+", " ") +
                + "</number>\n</entry>");
}

// richiesta GET - READ
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {

    String number;
    try {
        // recupero del body della richiesta
        String name = request.getParameter("name").replaceAll("\\+", " ");
```

```
    if (name == null) {
        response.sendError(400, "Malformed XML in doPut - null
parameter");
        return;
    }
    if (name.isEmpty()) {
        response.sendError(400, "Malformed XML in doPut - empty
parameter");
        return;
    }

    synchronized (PARSER) {
        PARSER.loadDOM();
        number = PARSER.readEntry(name);
    }

    response.setStatus(200);      // OK
    response.setContentType("text/xml; charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<?xml version=\"1.0\" encoding=\"UTF-8\""
standalone="no"?>"
        + "<entry>\n\t<name>" + name.replaceAll("\\+", " ")
        + "</name>\n\t<number>" + number.replaceAll("\\+", " ")
        + "</number>\n</entry>");
} catch (NullPointerException e) {

}
}

// richiesta PUT - UPDATE
@Override
protected void doPut(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {

boolean modified;
// recupero del body della richiesta
String name = request.getParameter("name").replaceAll("\\+", " ");
String number = request.getParameter("number").replaceAll("\\+", " ");

if (name == null || number == null) {
    response.sendError(400, "Malformed XML in doPut - null
parameters");
    return;
}
if (name.isEmpty() || number.isEmpty()) {
    response.sendError(400, "Malformed XML in doPut - empty
parameters");
}
```

```
        return;
    }

    synchronized (PARSER) {
        PARSER.loadDOM();
        modified = PARSER.updateEntry(name, number);

        if (modified) {
            PARSER.writeDOM();
            response.setStatus(200);          // OK
            response.setContentType("text/xml; charset=UTF-8");
            PrintWriter out = response.getWriter();
            out.println("<?xml version=\"1.0\" encoding=\"UTF-8\""
standalone=\"no\"?>" +
+ "<entry>\n\t<name>" + name.replaceAll("\\+", " ") +
+ "</name>\n\t<number>" + number.replaceAll("\\+", " ")
)
+ "</number>\n</entry>");
        } else {
            // Non eseguito l'aggiornamento perché il nome non esiste
            response.setStatus(202);
        }
    }
}

// richiesta PUT - UPDATE
@Override
protected void doDelete(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {

    boolean deleted;
    // recupero del body della richiesta
    String name = request.getParameter("name").replaceAll("\\+", " ");
    String number = request.getParameter("number").replaceAll("\\+", " ");

    if (name == null || number == null) {
        response.sendError(400, "Malformed XML in doPut - null
parameters");
        return;
    }
    if (name.isEmpty() || number.isEmpty()) {
        response.sendError(400, "Malformed XML in doPut - empty
parameters");
        return;
    }

    synchronized (PARSER) {
```

```
PARSER.loadDOM();
deleted = PARSER.deleteEntry(name, number);

if (deleted) {
    response.setStatus(200);          // OK
    PARSER.writeDOM();
} else {
    // Non eseguito la cancellazione perché non esiste la
combinazione
    // nome, numero
    response.setStatus(202);
}
}

// richiesta informazioni servlet
@Override
public String getServletInfo() {
    return "PhoneBook";
}
}
```

### ParserDOM.java

```
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class ParserDOM {

    private final String filepath = "/home/mgm/phonebook.xml";
    private Document doc = null;

    // Creazione dell'albero DOM e del parser.
```

```
public void loadDOM() {
    try {
        DocumentBuilderFactory docFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        System.out.println(filepath);
        doc = docBuilder.parse(filepath);
    } catch (ParserConfigurationException | SAXException | IOException ex) {
        Logger.getLogger(ParserDOM.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

// Invocato dal metodo doPost() per aggiungere un elemento <entry>
public void createEntry(String name, String number) {

    // Recupero dell'elemento radice
    Node phonebook = doc.getFirstChild();

    // Processo di aggiunta di un nuovo nodo <entry> al nodo radice
    // <phonebook>.
    // Creazione dei nuovi elementi.
    Element elEntry = doc.createElement("entry");
    Element elName = doc.createElement("name");
    Element elNumber = doc.createElement("number");
    // Aggiunta del testo agli elementi che lo richiedono e aggiunta
    // successiva degli elementi ai rispettivi nodi.
    elName.appendChild(doc.createTextNode(name));
    elEntry.appendChild(elName);
    elNumber.appendChild(doc.createTextNode(number));
    elEntry.appendChild(elNumber);
    phonebook.appendChild(elEntry);

    System.out.println("Done");
}

// Invocato dal metodo doGet() per leggere un elemento <entry>
public String readEntry(String name) {
    String elNumber = null;

    // Recupero dell'elemento radice
    Element phonebook = doc.getDocumentElement();
    // Recupero la lista degli elementi <entry>
    NodeList lEntry = phonebook.getElementsByTagName("entry");
    if (lEntry != null && lEntry.getLength() > 0) {
        for (int i = 0; i < lEntry.getLength(); i++) {
            // Scorriamo ogni singolo elemento "libro".
            Element element = (Element) lEntry.item(i);
```

```
        String elName =
element.getElementsByTagName("name").item(0).getFirstChild().getNodeValue();
        if (elName.equalsIgnoreCase(name)) {
            elNumber =
element.getElementsByTagName("number").item(0).getFirstChild().getNodeValue();
            System.out.println(elName + " " + elNumber);
        }
    }
}

return elNumber;
}

// Invocato dal metodo doPut() per modificare un elemento <entry>
public boolean updateEntry(String name, String number) {

    boolean modified = false;
    // Recupero dell'elemento radice
    Element phonebook = doc.getDocumentElement();
    // Recupero la lista degli elementi <entry>
    NodeList lEntry = phonebook.getElementsByTagName("entry");
    if (lEntry != null && lEntry.getLength() > 0) {
        for (int i = 0; i < lEntry.getLength(); i++) {
            // Scorriamo ogni singolo elemento "entry".
            Element element = (Element) lEntry.item(i);
            String elName =
element.getElementsByTagName("name").item(0).getFirstChild().getNodeValue();
            if (elName.equalsIgnoreCase(name)) {

element.getElementsByTagName("number").item(0).getFirstChild().settextContent(
number);
                modified = true;
            }
            String elNumber =
element.getElementsByTagName("number").item(0).getFirstChild().getNodeValue();
            System.out.println(elName + " " + elNumber);
        }
    }
}

System.out.println("Done");

return modified;
}

// Invocato dal metodo doDelete() per eliminare un elemento <entry>
public boolean deleteEntry(String name, String number) {

    boolean deleted = false;
```

```
// Recupero dell'elemento radice
Element phonebook = doc.getDocumentElement();
// Recupero la lista degli elementi <entry>
NodeList lEntry = phonebook.getElementsByTagName("entry");
if (lEntry != null && lEntry.getLength() > 0) {
    for (int i = 0; i < lEntry.getLength(); i++) {
        // Scorriamo ogni singolo elemento "entry".
        Element element = (Element) lEntry.item(i);
        String elName =
element.getElementsByTagName("name").item(0).getFirstChild().getNodeValue();
        String elNumber =
element.getElementsByTagName("number").item(0).getFirstChild().getNodeValue();
        if (elName.equalsIgnoreCase(name)
            && elNumber.equalsIgnoreCase(number)) {
            // Rimozione dello specifico nodo.
            lEntry.item(i).getParentNode().removeChild(element);
            // Normalizzazione dell'albero DOM, in modo tale che
            // tutti i nodi text siano foglie, dopo la rimozione
            // del nodo
            doc.normalize();
            deleted = true;
        }
    }
}
System.out.println("Done");

return deleted;
}

// Scrittura dell'albero DOM nel file phonebook.xml.
public void writeDOM() {
/*
L'impostazione corretta della modifica del file XML è stata effettuata
seguendo le indicazioni presenti al link:
https://www.mkyong.com/java/how-to-modify-xml-file-in-java-dom-parser/
*/
try {
    // Scrittura dell'albero DOM nel file phonebook.xml.
    TransformerFactory transformerFactory =
TransformerFactory.newInstance();
    Transformer transformer = transformerFactory.newTransformer();
    // Definizione della sorgente DOM il cui contenuto verrà
    // registrato in phonebook.xml.
    DOMSource source = new DOMSource(doc);
    // Definizione dello stream utilizzato per memorizzare il
    // contenuto dell'albero DOM.
```

```
        StreamResult result = new StreamResult(new File(filepath));
        // Memorizzazione dell'albero DOM nel file phonebook.xml.
        transformer.transform(source, result);
    } catch (TransformerConfigurationException ex) {
        Logger.getLogger(ParserDOM.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (TransformerException ex) {
        Logger.getLogger(ParserDOM.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
```

Di seguito viene fornito il codice del lato **client**.

### index.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>PhoneBook</title>
</head>
<body>
<form action="ClientRequest" method="GET">
    <input type="text" name="name" /><br />
    <input type="text" name="number" /><br />
    <input type="radio" name="operation" value="POST" checked/>Create
    <input type="radio" name="operation" value="GET" />Read
    <input type="radio" name="operation" value="PUT" />Update
    <input type="radio" name="operation" value="DELETE" />Delete <br />
    <input type="submit" value="Invia"/>
    <input type="reset" value="Cancella"/>
</form>
</body>
</html>
```

### ClientRequest.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;
import java.net.URLConnection;
```

```
import java.net.URLEncoder;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "ClientRequest", urlPatterns = {"/ClientRequest"})
public class ClientRequest extends HttpServlet {
/*
L'impostazione corretta della connessione è stata effettuata seguendo le
indicazioni presenti al link:

http://stackoverflow.com/questions/2793150/using-java-net-urlconnection-to-fir e-and-handle-http-requests
*/
String baseUrl;

@Override
public void init(ServletConfig config) {
    baseUrl = "http://localhost:8080/Z202ServletPhonebook/ServletPhoneBook";
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
    int status;
    // preparazione dei parametri da passare alla richiesta, sostituendo
    // tutti gli spazi con +
    String name = request.getParameter("name").replaceAll(" ", "+");
    String number = request.getParameter("number").replaceAll(" ", "+");
    String operation = request.getParameter("operation");

    if (operation.equalsIgnoreCase("POST")) {
        HttpURLConnection service = httpPost(name, number, operation);
        if (service != null) {
            // apertura stream di ricezione dati dal Web service
            InputStream res = service.getInputStream();
            Scanner scanner = new Scanner(res);
            // \A      The beginning of the input
            String responseBody = scanner.useDelimiter("\A").next();
            // Visualizzazione di controllo
        }
    }
}
```

```
        System.out.println(responseBody);
        // Invio risposta al client
        response.setContentType("text/xml;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println(responseBody);
        scanner.close();
    }
}

if (operation.equalsIgnoreCase("GET")) {
    HttpURLConnection service = httpGet(name, operation);
    if (service != null) {
        // apertura stream di ricezione dati dal Web service
        InputStream res = service.getInputStream();
        Scanner scanner = new Scanner(res);
        String responseBody = scanner.useDelimiter("\A").next();
        // Visualizzazione di controllo
        System.out.println(responseBody);
        // Invio risposta al client
        response.setContentType("text/xml;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println(responseBody);
        scanner.close();
    }
}

if (operation.equalsIgnoreCase("PUT")) {
    HttpURLConnection service = httpPut(name, number, operation);
    if (service != null) {
        // apertura stream di ricezione dati dal Web service
        InputStream res = service.getInputStream();
        Scanner scanner = new Scanner(res);
        String responseBody = scanner.useDelimiter("\A").next();
        // Visualizzazione di controllo
        System.out.println(responseBody);
        // Invio risposta al client
        response.setContentType("text/xml;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println(responseBody);
        scanner.close();
    }
}

if (operation.equalsIgnoreCase("DELETE")) {
    status = httpDelete(name, number, operation);
    if (status == 200) {
        response.setContentType("text/xml;charset=UTF-8");
        PrintWriter out = response.getWriter();
    }
}
```

```
        out.println("<?xml version=\"1.0\" encoding=\"UTF-8\""
standalone=\"no\"?>"
                + "<entry>Deleted</entry>");
            }
        }

    }

private HttpURLConnection httpPost(String name, String number, String
operation) {
    try {
        // specifico che il set di caratteri usato è UTF-8
        String charset = java.nio.charset.StandardCharsets.UTF_8.name();
        // URL di connessione alla servlet che gestisce phonebook.xml
        URLConnection server = null;
        String doc = String.format("name=%s&number=%s",
                URLEncoder.encode(name, charset),
                URLEncoder.encode(number, charset));
        int n = doc.length();
        // Costruzione dello URL di interrogazione del Web service.
        // Nel caso fosse invocata una creazione (POST) non si userebbe
        // URLEncoder.encode(...) perché non si devono aggiungere
        // parametri all'URL.
        server = new URL(baseUrl).openConnection();
        // Conversione dello URLConnection in HttpURLConnection
        HttpURLConnection service = (HttpURLConnection) server;
        // Impostazione del metodo di richiesta POST.
        service.setRequestMethod(operation);
        // impostazione header richiesta
        service.setRequestProperty("User-Agent", "Mozilla/5.0");
        service.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
        // L'impostazione del Content-type è fondamentale per la buona
        // riuscita della richiesta.
        service.setRequestProperty("Content-type",
"application/x-www-form-urlencoded; charset=" + charset);
        service.setRequestProperty("Content-length", Integer.toString(n));
        // L'impostazione di setDoOutput(true) implica che si vuole
        // inviare in output un body di richiesta tramite i metodi HTTP
        // POST.
        // Questa impostazione rende superfluo l'invocazione
        // successiva di connect()
        service.setDoOutput(true);
        // Invio dei dati al Web service specificando come set di
        // caratteri UTF-8.
        try (OutputStream output = service.getOutputStream()) {
            output.write(doc.getBytes(charset));
            output.flush();
            output.close();
        }
    }
}
```

```
        }
        System.out.println(service);
        // connessione al Web service
        service.connect();
        System.out.println(service.getURL());
        // verifica stato risposta
        int status = service.getResponseCode();
        if (status == 201) {
            // se l'operazione POST è riuscita visualizzo una frase di
            //OK
            System.out.println("OK - " + operation);
            return service;
        } else {
            System.out.println("Operazione non riuscita - " + status + "
" + service.getResponseMessage());
            return null;
        }

    } catch (ProtocolException ex) {
        Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (MalformedURLException ex) {
        Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (UnsupportedEncodingException ex) {
        Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (IOException ex) {
        Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
    }

    return null;
}

private HttpURLConnection httpGet(String name, String operation) {
    try {
        // URL di connessione alla servlet che gestisce phonebook.xml
        URLConnection server = null;
        String doc = "?name=" + name;
        // Costruzione dello URL di interrogazione del Web service.
        // Nel caso di una lettura (GET) si deve aggiungere all'URL il
        // nome della persona di cui si vuole il numero di telefono.
        server = new URL(baseUrl + doc).openConnection();
        // Conversione dello URLConnection in HttpURLConnection
        HttpURLConnection service = (HttpURLConnection) server;
        // Impostazione del metodo di richiesta GET
        service.setRequestMethod(operation);
```

```
// impostazione header richiesta
service.setRequestProperty("User-Agent", "Mozilla/5.0");
service.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
// L'impostazione del Content-type è fondamentale per la buona
// riuscita della richiesta.
service.setRequestProperty("Accept-Charset", "UTF-8");
service.setRequestProperty("Accept", "text/xml");
// L'impostazione di setDoINPUT(true) implica che si vuole
// ricevere in input una risposta il metodo HTTP GET.
// Questa impostazione rende superfluo l'invocazione successiva di
// connect()
service.setDoInput(true);
System.out.println(service);
// connessione al Web service
service.connect();
System.out.println(service.getURL());
// verifica stato risposta
int status = service.getResponseCode();
if (status == 200) {
    // se l'operazione GET è riuscita visualizzo una frase di OK
    System.out.println("OK - " + operation);
    return service;
} else {
    System.out.println("Operazione non riuscita - " + status + "
" + service.getResponseMessage());
    return null;
}

} catch (UnsupportedEncodingException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
} catch (MalformedURLException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
} catch (IOException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
}
return null;
}

private HttpURLConnection httpPut(String name, String number, String
operation) {
try {
    // URL di connessione alla servlet che gestisce phonebook.xml
    URLConnection server = null;
    String doc = "?name=" + name + "&number=" + number;
    // Costruzione dello URL di interrogazione del Web service.
```

```
// Nel caso di una modifica (PUT) si deve aggiungere all'URL il
// nome della persona e il numero da modificare.
// L'operazione è idempotente.
server = new URL(baseUrl + doc).openConnection();
// Conversione dello URLConnection in HttpURLConnection
HttpURLConnection service = (HttpURLConnection) server;
// Impostazione del metodo di richiesta PUT
service.setRequestMethod(operation);
// impostazione header richiesta
service.setRequestProperty("User-Agent", "Mozilla/5.0");
service.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
// L'impostazione del Content-type è fondamentale per la buona
// riuscita della richiesta.
service.setRequestProperty("Accept-Charset", "UTF-8");
service.setRequestProperty("Accept", "text/xml");
// L'impostazione di setDoOutput(true) implica che si vuole
// inviare in output un body di richiesta tramite i metodi HTTP
// PUT.
// Questa impostazione rende superfluo l'invocazione successiva di
// connect()
service.setDoOutput(true);
System.out.println(service);
// connessione al Web service
service.connect();
System.out.println(service.getURL());
// verifica stato risposta
int status = service.getResponseCode();
if (status == 200) {
    // se l'operazione PUT è riuscita visualizzo una frase di OK
    System.out.println("OK - " + operation);
    return service;
} else if (status == 202) {
    // se l'operazione PUT è stata accettata ma non eseguito
    // perché non esiste il nome indicato
    System.out.println("Accepted but not fulfilled - " +
operation);
    return null;
} else {
    System.out.println("Operazione non riuscita - " + status + " "
+ service.getResponseMessage());
    return null;
}

} catch (UnsupportedEncodingException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
} catch (MalformedURLException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
```

```
null, ex);
} catch (IOException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
}

return null;
}

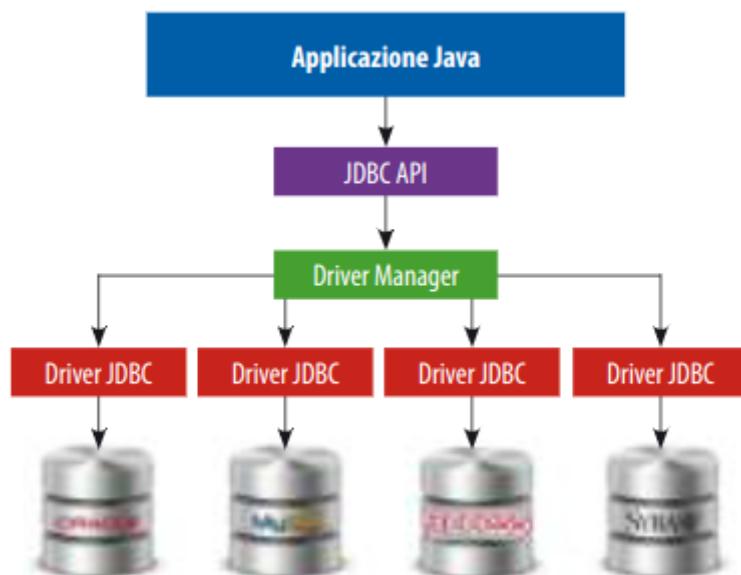
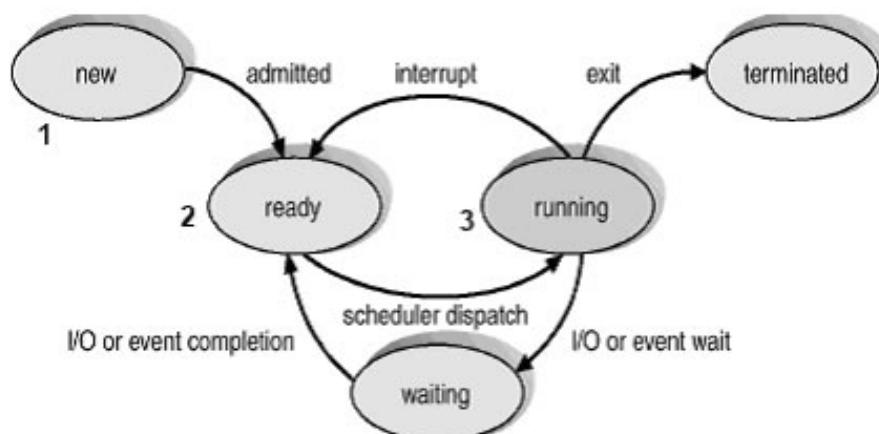
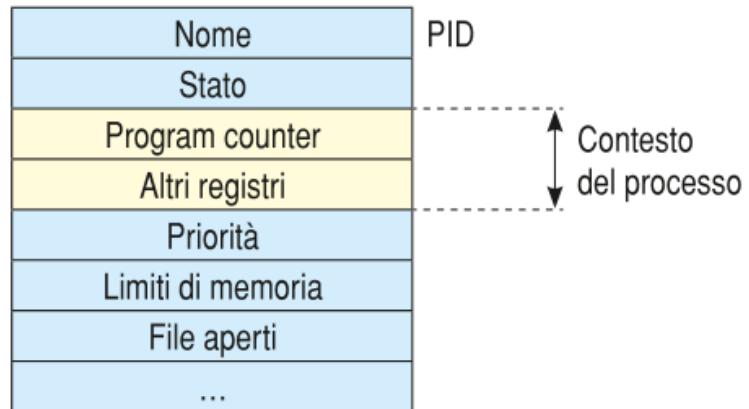
private int httpDelete(String name, String number, String operation) {
    int status = 0;
    try {
        // URL di connessione alla servlet che gestisce phonebook.xml
        URLConnection server = null;
        String doc = "?name=" + name + "&number=" + number;
        // Costruzione dello URL di interrogazione del Web service.
        // Nel caso di una modifica (PUT) si deve aggiungere all'URL il
        // nome della persona e il numero da modificare.
        // L'operazione è idempotente.
        server = new URL(baseUrl + doc).openConnection();
        // Conversione dello URLConnection in HttpURLConnection
        HttpURLConnection service = (HttpURLConnection) server;
        // Impostazione del metodo di richiesta PUT
        service.setRequestMethod(operation);
        // impostazione header richiesta
        service.setRequestProperty("User-Agent", "Mozilla/5.0");
        service.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
        // L'impostazione del Content-type è fondamentale per la buona
        // riuscita della richiesta.
        service.setRequestProperty("Accept-Charset", "UTF-8");
        service.setRequestProperty("Accept", "text/xml");
        // L'impostazione di setDoOutput(true) implica che si vuole
        // inviare in output un body di richiesta tramite i metodi HTTP
        // PUT.
        // Questa impostazione rende superfluo l'invocazione successiva di
        // connect()
        //service.setDoOutput(true);
        System.out.println(service);
        // connessione al Web service
        service.connect();
        System.out.println(service.getURL());
        // verifica stato risposta
        status = service.getResponseCode();
        if (status == 200) {
            // se l'operazione PUT è riuscita visualizzo una frase di OK
            System.out.println("OK - " + operation);
        } else if (status == 202) {
            // se l'operazione PUT è stata accettata ma non eseguito
```

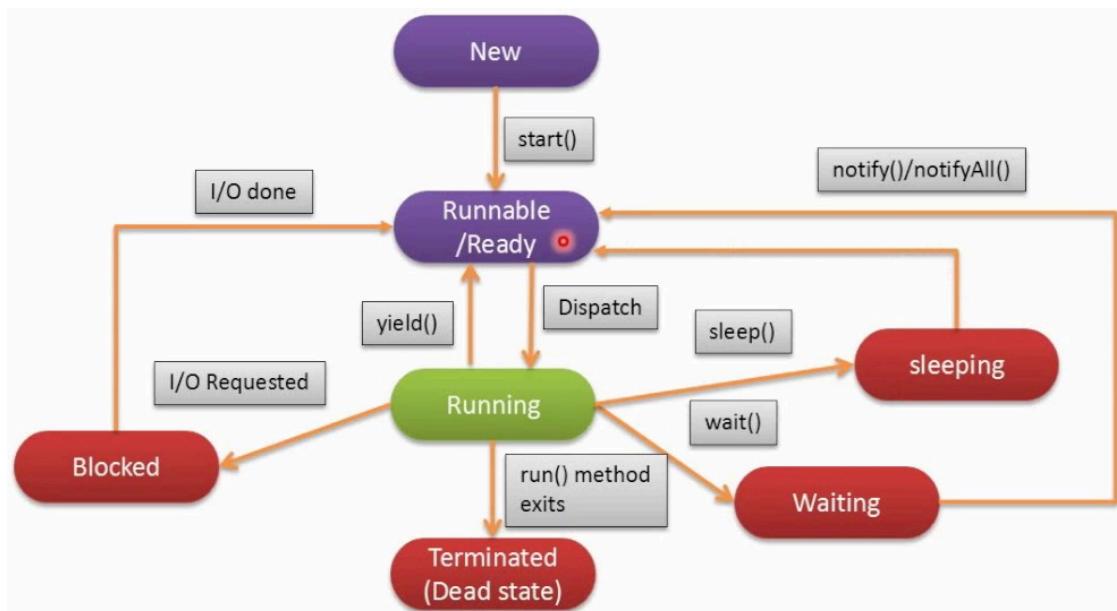
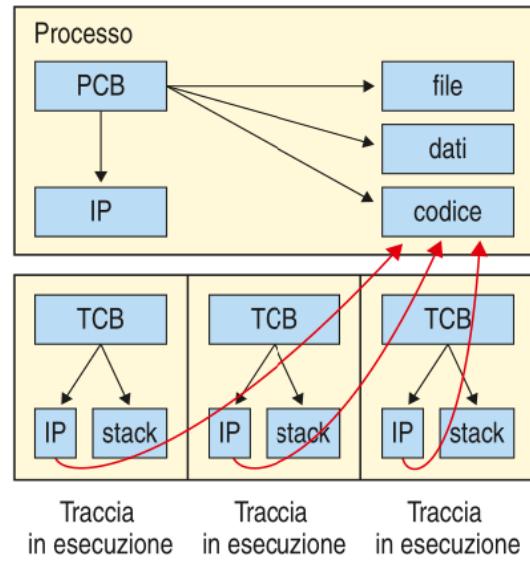
```
// perché non esiste il nome indicato
System.out.println("Accepted but not fulfilled - " +
operation);
} else {
    System.out.println("Operazione non riuscita - " + status + "
" + service.getResponseMessage());
}

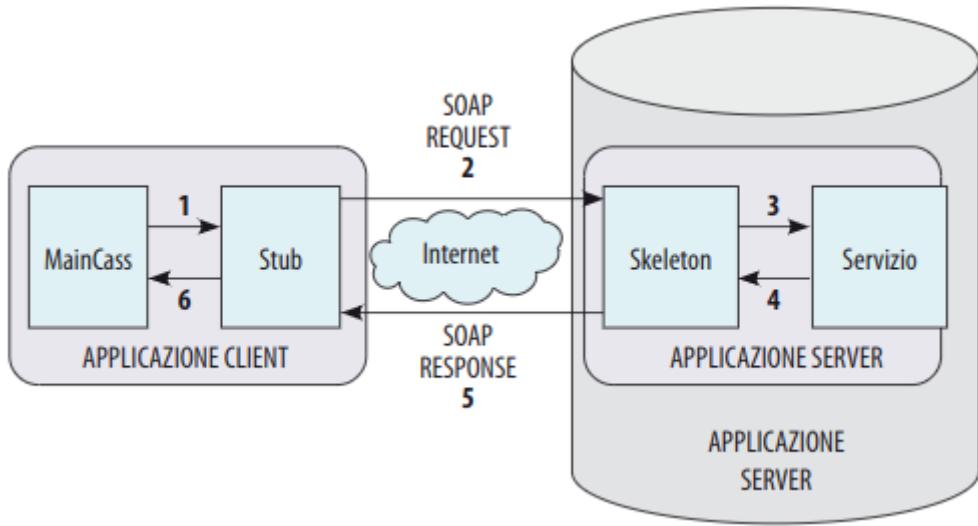
} catch (UnsupportedEncodingException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
} catch (MalformedURLException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
} catch (IOException ex) {
    Logger.getLogger(ClientRequest.class.getName()).log(Level.SEVERE,
null, ex);
}
return status;
}

}
```

## I - Immagini colloquio non utilizzabili







---

@PATH, @CONSUMES, @PRODUCES

---

@GET, @POST, @PUT, @DELETE

---

# Bibliografia e sitografia

# Thread e concorrenza in Java - Propedeutica

[V. Kovalenko](#), [Race Condition vs. Data Race in Java](#), [DZone](#), [Java Zone](#), 7 agosto 2018

[Java - Understanding Happens-before relationship](#), [LogicBig](#), 19 maggio 2018

# Design Patterns - Propedeutica

Ugonna Thelma, *The S.O.L.I.D. Principles*, ultimo accesso ottobre 2021

Baeldung, *A Solid Guide to SOLID Principles*, ultimo accesso ottobre 2021

# I socket e la comunicazione in rete

P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2017.

P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 2, ed. Hoepli, 2016.

V. Auletta, *Corso di Reti di calcolatori*,  
[http://www.di.unisa.it/professori/auletta/DIDATTICA/RETI\\_04/](http://www.di.unisa.it/professori/auletta/DIDATTICA/RETI_04/), autunno 2003

M. Maggini, *Corso di Reti di calcolatori*,  
[http://www.dii.unisi.it/~maggini/Teaching/reti\\_di\\_calcolatori.html](http://www.dii.unisi.it/~maggini/Teaching/reti_di_calcolatori.html), a.a. 2015/2016

Università di Verona, Dipartimento di Informatica, *Scrittura di applicazioni di rete*,  
<http://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid546139.pdf>, ultimo accesso gennaio 2017

D. J. Dubois, *Programmazione di rete in Java*,  
<http://home.deib.polimi.it/dubois/provafinale/rete.pdf>, Politecnico di Milano, ultimo accesso gennaio 2017

G. Anastasi, N. Iardella, *Corso di Reti Informatiche*,  
<http://www2.ing.unipi.it/~a008149/corsi/reti/materiale.html>, Università di Pisa, ultimo accesso gennaio 2017

[Prof. Raj Jain, CIS677: Introduction to Computer Networking, Washington University in St. Louis](#), ultimo accesso settembre 2018

P. Principe, Java 8, ed. Apogeo, marzo 2014

Sic-Oding, <http://sic-oding.blogspot.com/>

J. F. Gonzalez, Mastering Concurrency Programming with Java 9- Fast, reactive and parallel application development, 2nd edition, Packt, 2017

[Prof. Giuseppe Pappalardo, Università di Catania](#), ultimo accesso settembre 2019

[Callicoder, Concurrency basic](#), ultimo accesso ottobre 2019

[Jencov.com, Java Concurrency and Multithreading Tutorial](#), ultimo accesso ottobre 2019

[OpenSourceForUcom, A Guide to Using Raw Sockets](#), ultimo accesso novembre 2019

[Ghini V., Lo Stack TCP/IP e i Socket](#), ultimo accesso novembre 2019

[JournalDev, Thread.sleep\(\) in Java – Java Thread sleep](#), ultimo accesso dicembre 2019

[SequenceDiagram.org](#), ultimo accesso ottobre 2021

# CLIL - Introduction to Distributed system

2veritasium, *Transistors & The End of Moore's Law*, <https://youtu.be/rtI5wRyHpTg>

A337 | S400, *3 Tier Client Server Architecture*,  
[https://youtu.be/jJYv-nfkMXk?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2\\_4CM](https://youtu.be/jJYv-nfkMXk?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2_4CM)

Barbero T., Clegg J., *Programmare Percorsi CLIL*, Carocci, Roma 2005.

Bubble.us, <https://bubbl.us/>

CLIL Media, <http://clilmedia.com/>

Cmap, <http://www.cmaptools.com/>

Christopher Kalodikis, *Client-server model*,  
[https://youtu.be/ntp9XZ4hOrY?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2\\_4CM](https://youtu.be/ntp9XZ4hOrY?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2_4CM)

English tutorial, <http://www.englishpage.com/>

G. Coulouris, J. Dollimore, T. Kindberg, G. Blair, *Distributed system, Concept and Design*, Fifth Edition, Addison-Wesley, May 2011, <http://www.cdk5.net/wp/>

Games to learn English, <http://www.engames.eu/>

Google, <http://www.google.it>

Grewal, *N-Tier Architecture for kids*,  
[https://youtu.be/8gfTBQhh1kM?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2\\_4CM](https://youtu.be/8gfTBQhh1kM?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2_4CM)

C. Kalodikis, *Client-Server Model*, <https://youtu.be/IOnWn5u-sXE>

Kurzgesagt – In a Nutshell, *Quantum Computers Explained – Limits of Human Technology*,  
<https://youtu.be/JhHMJCUmq28>

Linking Words for IELTS Speaking: Word List & Tips,  
<http://ieltsliz.com/linking-words-for-ielts-speaking/>

ovp , *How Cloud Computing Work*, <https://youtu.be/DGDtujmOBKc>

PieterExplainsTech, *UDP and TCP: Comparison of Transport Protocols*,  
<https://youtu.be/Vdc8TCESIg8>

S. Haridi, *Distributed Algorithms*,  
<https://www.youtube.com/playlist?list=PL700757A5D4B3F368>

Regis University CPS SCIS, *Middleware Concepts*,  
[https://youtu.be/S8sqGXUqw30?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2\\_4CM](https://youtu.be/S8sqGXUqw30?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2_4CM)

Regis University CPS SCIS, *Middleware Architecture*,  
[https://youtu.be/E2jFOTDK0tY?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2\\_4CM](https://youtu.be/E2jFOTDK0tY?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2_4CM)

The m-Power Platform, *n-Tier Architecture Explained*, <https://youtu.be/KIHvRKSH4pk>

Udacity, Georgia Tech, Advanced Operating Systems part 2 of 4,  
<https://www.youtube.com/playlist?list=PLAwxTw4SYaPm4vV1XbFV93ZuT2saSq1hO>

Udacity, Georgia Tech, HPCA: Part 5,  
[https://youtu.be/WKXbhkzBUo?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2\\_4CM](https://youtu.be/WKXbhkzBUo?list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2_4CM)

WhatIs.com, <http://whatismytechtarget.com/>

Wikipedia, The Free Encyclopedia, <https://www.wikipedia.org/>

## Java e XML

P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2014

G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, Zanichelli, 2014

Wikipedia, The Free Encyclopedia, <https://www.wikipedia.org/>

R. Golia, *Design pattern per esempi: i GoF creazionali*, Microsoft® Development Network,  
<https://msdn.microsoft.com/it-it/library/cc185067.aspx#ID0ECBAC>, novembre 2006,  
ultima visita gennaio 2017

TutorialsPoint, *Design Pattern - Factory Pattern*,  
[https://www.tutorialspoint.com/design\\_pattern/factory\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/factory_pattern.htm), ultima visita gennaio 2017

B. McLaughlin & J. Edelson, *Java & XML*, 3rd edition, O'Reilly, 2006

Mkyong.com, *How to read XML file in Java – (DOM Parser)*,  
<http://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/>, ultima visita gennaio 2017

## HTTP e Servlet

P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2014

w3schools.com, *HTTP Methods*, [http://www.w3schools.com/tags/ref\\_httpmethods.asp](http://www.w3schools.com/tags/ref_httpmethods.asp), ultimo accesso gennaio 2017

JavaTpoint, *Web Terminology*, <http://www.javatpoint.com/web-terminology>, ultimo accesso gennaio 2017

Wikipedia, L'enciclopedia libera, <https://it.wikipedia.org>

Lifewire, *HTTP Status Codes*, <https://www.lifewire.com/http-status-codes-2625907>, 28 novembre 2016

Mikalai Zaikin, *IBM WebSphere Application Server Network Deployment V8.0 Core Administration Guide*, <http://java.boot.by/ibm-317/index.html>, agosto 2013

Jayson Online, *Configuring Apache in front of JBoss Application Server Using mod\_jk*, [http://www.jaysonjc.com/programming/configuring-apache-in-front-of-jboss-application-server-using-mod\\_jk.html](http://www.jaysonjc.com/programming/configuring-apache-in-front-of-jboss-application-server-using-mod_jk.html), 19 aprile 2011

## I sistemi distribuiti: modelli architetturali hardware e software

P. Camagni, R. Nikolassy, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, ed. Hoepli, 2014 e 2017

[Kurzgesagt – In a Nutshell](#), *Quantum Computers Explained – Limits of Human Technology*, <https://youtu.be/JhHMJCUMq28>, ultimo accesso marzo 2017

[2veritasium](#), *Transistors & The End of Moore's Law*, <https://youtu.be/rtI5wRyHpTg>, ultimo accesso marzo 2017

[Udacity](#), *Flynn's Taxonomy of Parallel Machines - Georgia Tech - HPCA: Part 5*, <https://youtu.be/WKXbvhkzBUo>, ultimo accesso marzo 2017

[Audiopedia](#), *What is COMPUTER CLUSTER? What does COMPUTER CLUSTER mean? COMPUTER CLUSTER explanation*, <https://youtu.be/aNa9SWPEHIQ>, ultimo accesso marzo 2017

[Ross Dickson](#), *What's a cluster?*, <https://youtu.be/7rooWbLe1iI>, ultimo accesso marzo 2017

[Dell EMC](#), *Big Ideas: Simplifying Cluster Architectures*, <https://youtu.be/4M3cROio9vU>, ultimo accesso marzo 2017

[Regis University CPS SCIS](#), *Middleware Concepts*, <https://youtu.be/S8sgGXUqw30>, ultimo accesso marzo 2017

Wikipedia, L'enciclopedia libera, <https://it.wikipedia.org>

R. Baldoni, *Middleware - Sistemi Operativi II - Corso di Laurea in Ingegneria Informatica*, Università di Roma "La Sapienza" Dipartimento di Informatica e Sistemistica, <http://www.dis.uniroma1.it/~baldoni/SOII-middleware.pdf>, ultimo accesso marzo 2017

## Web Service REST

[Chiarelli A.](#), *RESTful Web Services – La Guida*, <http://www.html.it/guide/restful-web-services-la-guida/>, [HTML.it](#), 6 febbraio 2012

G. Meini, F. Formichi, *Tecnologie e progettazione di sistemi informatici e di telecomunicazioni*, vol. 3, Zanichelli, 2014

Java tutorial for beginners, <http://www.java2blog.com/2013/03/web-service-tutorial.html>, ultimo accesso gennaio 2017

ORACLE, Java™ Platform, Standard Edition 7 API Specification,  
<https://docs.oracle.com/javase/7/docs/api/>, ultimo accesso gennaio 2017

Stack Overflow, <http://stackoverflow.com/>

[Javarevisited](#), Java Synchronization Tutorial : What, How and Why?,  
<http://javarevisited.blogspot.it/2011/04/synchronization-in-java-synchronized.html>  
aprile 2011, ultimo accesso febbraio 2017

[Java Brains](#), Developing RESTful APIs with JAX-RS,  
<https://www.youtube.com/playlist?list=PLqq-6Pq4lTTZh5U8RbdXq0WaYvZBz2rbn>, ultimo  
accesso marzo 2017

L. Petrosino, Luca's blog, <http://luca-petrosino.blogspot.it/>, ultimo accesso marzo 2017

Team SGTEAM, Progetto MONK - Gestione del protocollo della corrispondenza della  
biblioteca di Santa Giustina, [http://www.dei.unipd.it/~fantozzi/MONK/descr\\_progetto.html](http://www.dei.unipd.it/~fantozzi/MONK/descr_progetto.html),  
30 settembre 1999, ultimo accesso marzo 2017

Maria Lăiu, Botnaru Elvis, Nicolae Abăcioaiei, Ionuț Stîncescu and Dumitru Lesnic,  
[pentastagiultrashop - Wiki](#), <https://github.com/pentastagiultrashop/wiki>, ultimo  
accesso marzo 2017

[F. Corno](#), D. Bonino, 01KTF - Architetture distribuite per i sistemi infomatici aziendali,  
<https://elite.polito.it/teaching/past-courses/19-01ktf?showall=>, e-lite research group, 9  
marzo 2008, ultimo accesso aprile 2017

Agile Modeling (AM) Home Page, Effective Practices for Modeling and Documentation,  
<http://agilemodeling.com/>, ultimo accesso aprile 2017

The Unified Modeling Language, <http://www.uml-diagrams.org/>, ultimo accesso aprile 2017

IBM developerWorks®, <https://www.ibm.com/developerworks/>, ultimo accesso aprile 2017

WebSequenceDiagrams, <https://www.websequencediagrams.com/>, ultimo accesso aprile  
2017

StarUML, <http://staruml.io/>, ultimo accesso agosto 2017

Violet UML Editor, <http://alexdp.free.fr/violetumleditor/page.php>, ultimo accesso agosto  
2017

CodingJam, <https://codingjam.it/>, ultimo accesso maggio 2019

Java2Blog, <https://java2blog.com/>, ultimo accesso maggio 2019

Unsupervised Learning, <https://danielmiessler.com/>, ultimo accesso maggio 2024