

# Università di Verona, Dipartimento di Informatica

## Programmazione e Sicurezza delle Reti

Marco De Bona, Davide Quaglia

### Architetture orientate ai servizi

L'architettura orientata ai servizi (Service Oriented Architecture o SOA) definisce un nuovo modello logico secondo il quale sviluppare il software. Tale modello è realizzato dai **Web Service**, che si presentano come moduli software distribuiti i quali collaborano fornendo determinati servizi in maniera standard. Il Web Service è una funzionalità messa a disposizione in modalità server ad altri moduli software implementati come client. I componenti software **interagiscono tra loro attraverso protocolli Web Service (ad es. REST, SOAP) trasportati in connessioni HTTP (da cui il nome Web Service)**. E' compito del protocollo Web Service definire una forma serializzata dei parametri attuali da passare al web service e dei valori restituiti da quest'ultimo.

Quando uno sviluppatore crea un Web Service, si deve preoccupare di definire:

1. la **logica di funzionamento del servizio**, ovvero quello che dovrà fare; questo può spaziare da una semplice classe ad un'applicazione molto complessa;
2. il **web container** su cui verrà installato (ad es. Apache, Tomcat, Jboss, Glassfish) e che ne consentirà l'uso da parte dei client.

### 1 Vantaggi tecnologici

La nuova struttura collaborativa distribuita porta ad una serie di **vantaggi tecnologici** che riassumiamo brevemente.

- **Software come servizio**: Al contrario del software tradizionale, una collezione di metodi esposta tramite Web Service può essere utilizzata come un servizio accessibile da qualsiasi client.
- **Interoperabilità**: I Web Service consentono l'incapsulamento; i componenti possono essere isolati in modo tale che solo lo strato relativo al servizio vero e proprio sia esposto all'esterno. Ciò comporta due vantaggi fondamentali: indipendenza dall'implementazione e sicurezza del sistema interno. la logica applicativa incapsulata all'interno dei Web Service è completamente decentralizzata ed accessibile attraverso Internet da piattaforme, dispositivi, sistemi operativi e linguaggi di programmazione differenti.
- **Semplicità di sviluppo e di rilascio**: Un'applicazione è costituita da moduli indipendenti, interagenti tramite la rete; la modularità semplifica lo sviluppo. Rilasciare un WS significa solo esporlo al Web.
- **Semplicità di installazione**: la comunicazione avviene grazie allo scambio di informazioni in forma testuale all'interno del protocollo HTTP usato per il Web e utilizzabile praticamente su tutte le piattaforme hardware/software. I messaggi testuali viaggiano sullo stesso canale utilizzato per il Web e quindi sono già abilitati dai firewall. La comunicazione tra due sistemi non deve essere preceduta da noiose configurazioni ed accordi tra le parti.
- **Standard**: concetti fondamentali che stanno dietro ai Web Service sono regolati da standard approvati dalle più grandi ed importanti società ed enti di Information Technology al mondo.

## 1.1 Motivazioni

Si sta osservando un sempre maggiore utilizzo dell'approccio SOA nella creazione di applicativi software per i seguenti motivi:

- **Protezione della Proprietà Intellettuale.** Il cuore dell'applicazione rimane sul server e non viene distribuito neanche come eseguibile agli utenti.
- **Requisiti di potenza di calcolo.** La potenza di calcolo richiesta per l'applicazione può essere soddisfatta dal server senza gravare sulla CPU e il consumo energetico del client (ad es. smartphone)
- **Requisiti di memoria di massa.** La grande memoria di massa richiesta per l'applicazione può essere soddisfatta dal server senza gravare sulle risorse del client.
- **Comodità di distribuzione agli utenti.** Ogni utente usa un software client minimale (eseguibile per PC, plugin nel browser web, app per smartphone) che si connette al server. Questo comporta le seguenti conseguenze:
  - **Aggiornamento istantaneo.** La logica applicativa interna al server può cambiare in qualsiasi momento (ad es. per eliminare bug, adeguamenti normativi di un SW legale, ecc.) senza dover re-distribuire aggiornamenti agli utenti che si trovano istantaneamente ad utilizzare il software aggiornato.
  - **Sviluppo e mantenimento di una sola versione del prodotto** a fronte di molteplici piattaforme utente (ad es. Windows, Linux, MAC, smartphone).
  - **Maggiori ritorni economici** col nuovo approccio pay-per-use rispetto al tradizionale approccio della distribuzione e installazione dell'applicativo sui PC degli utenti. Eliminazione del fenomeno della pirateria.

# REST

## 2 Metodologia REST

In questa esercitazione vedremo come realizzare dei programmi Java che si interfaccino all'archivio genomico KEGG (<http://www.kegg.jp/>) allo scopo di aumentare l'automatizzazione nella ricerca biotecnologica.

Per fare ciò KEGG mette a disposizione delle API che, tramite metodologia REST, forniscono una serie di metodi per recuperare le informazioni che si trovano sul database attraverso programmi di interrogazione scritti dall'utente (ad es. noi useremo il linguaggio Java) in alternativa alla consultazione manuale tramite form web. Questa interfaccia al server KEGG è fondamentale per costruire programmi che automatizzano il recupero di informazioni evitando noiose e lente operazioni di interrogazione manuale.

### 2.1 Note tecniche

Un Web Service (servizio web), secondo la definizione data dal World Wide Web Consortium (W3C), è un sistema software progettato per **l'interazione tra diversi elaboratori in rete**; tale caratteristica si ottiene associando all'applicazione un'interfaccia software che espone all'esterno il servizio associato (**server**) e utilizzando la quale altri sistemi (**client**) possono interagire con

l'applicazione stessa attivando le operazioni descritte nell'interfaccia (servizi o richieste di procedure remote) tramite appositi messaggi di richiesta; tali messaggi di richiesta possono avere un formato particolare (il più famoso è SOAP) e sono sempre incapsulati e trasportati tramite i protocolli del Web (solitamente HTTP), da cui deriva appunto il nome web service.

REST è un approccio alternativo a SOAP per la realizzazione di web services. SOAP si basa sul concetto di chiamata remota e utilizza una complessa infrastruttura di messaggi XML, stub/skeleton e application container. Al contrario, REST è molto più leggero perché si basa sulla corrispondenza diretta tra:

- oggetti remoti da leggere/scrivere → Uniform Resource Identifiers (URI) cioè i comuni link web
- comandi da applicare agli oggetti → comandi HTTP (GET, PUT, POST, DELETE)

Se ad esempio vogliamo leggere la lista dei prodotti presenti su un catalogo, verrà semplicemente invocato il metodo HTTP GET sulla URI che rappresenta il catalogo prodotti; siccome il metodo GET è quello usato per default dai browser web nell'accesso alle pagine, sarà quindi sufficiente aprire la pagina

`http://www.mionegozio.it/lista/prodotti`

Sarà stata ovviamente cura del progettista del server far corrispondere l'URI lista/prodotti l'opportuna query sul database.

Se il server lo supporta, la stessa risorsa può essere richiesta in formati diversi, come ad esempio TXT, HTML, XML o JSON.

REST assume che le interazioni tra client e server devono avvenire senza memorizzazione di informazioni di stato sul server. Cioè **le interazioni tra client e server devono essere senza stato (stateless)**. È importante sottolineare che sebbene REST assuma la comunicazione stateless, non vuol dire che un'applicazione non deve avere stato. Tuttavia la responsabilità della gestione dello stato dell'applicazione non deve essere conferita al server, ma rientra nei compiti del client.

## 2.3 Esercizi

NOTA: in laboratorio, qualora non ci sia spazio nella propria home, si può utilizzare la cartella /tmp/ in cui vi è tutto lo spazio necessario; attenzione però che tale cartella viene svuotata ad ogni avvio della macchina per cui sui propri PC personali se ne sconsiglia l'uso.

Occorre creare una cartella di lavoro (ad esempio basta copiare la cartella rest/ dell'archivio) all'interno della quale andremo a mettere i seguenti files:

- **Definition.java** (classe corrispondente alla definizione di organismo nel DB di KEGG)
- **RESTCall.java** (classe che incapsula le chiamate REST per il KEGG, utilizza gli oggetti httpclient)
- **httpclient-4.2.5.jar** (jar scaricabile dal sito apache foundation, serve per eseguire una richiesta HTTP)
- **httpcore-4.2.4.jar** (jar scaricabile dal sito apache foundation, serve per eseguire una richiesta HTTP)
- **commons-logging-1.1.1.jar** (libreria che fornisce un sistema evoluto di logging)

I file jar sono anche scaricabili da questo link: <http://hc.apache.org/downloads.cgi>

Infine occorre mettere nella cartella di lavoro tutti i programmi Java che si vogliono compilare ed eseguire. Al momento della compilazione e dell'esecuzione dei nostri programmi Java, dovremmo aver cura di comprendere i file di supporto elencati; per farlo dobbiamo includere nel classpath le librerie come spiegato di seguito.

Portarsi nella cartella dei sorgenti

### ***Compilazione:***

```
javac -classpath
:./httpclient-4.2.5.jar:./httpcore-4.2.4.jar:./commons-logging-1.
1.1.jar *.java
```

### ***Esecuzione:***

```
java -classpath
:./httpclient-4.2.5.jar:./httpcore-4.2.4.jar:./commons-logging-1.
1.1.jar classe [parametri]
```

Vediamo quindi subito un esempio:

```
public class KeggOrganismsList {
    public static void main(String[] args) {
        // creazione di un oggetto RESTCall
        // che effettua le chiamate al KEGG
        RESTCall rest= new RESTCall();

        // richiediamo la lista degli organismi presenti in KEGG
        Definition[] def = rest.list_organisms();

        // tramite un ciclo for stampiamo la lista degli organismi
        for (int i = 0; i < def.length; i++) {
            System.out.println(i+" "+def[i].getDefinition());
        }
    }
}
```

Compilate ed eseguite il codice; come si evince dai commenti, questo semplice programma recupera la lista degli organismi contenuti nel database KEGG e ne stampa la descrizione a video.

Se si analizza l'implementazione del metodo `list_organisms()` in `RESTCall.java` si vede che la chiamata REST è <http://rest.kegg.jp/list/organism>

## **Esercizio 0**

Provare a copiare e incollare in un web browser (ad es. Firefox) la URI corrispondente alla chiamata REST e vedere cosa succede.

Definition è una classe che contiene informazioni su un organismo; di conseguenza un array di Definition contiene un organismo per ogni valore dell'indice. La descrizione della classe Definition è in Appendice C; si chiede di analizzarne i metodi.

**--testare che la banca dati kegg sia ancora attiva, e provarlo in delta**

## Esercizio 1

Si voglia verificare la presenza di un certo organismo all'interno del database e stamparne anche l'Entry\_id oltre che al nome (vedere metodo corrispondente). Modificate quindi il codice in modo da ricercare un particolare organismo passato come input sulla riga di comando.

Suggerimento: ricavare la stringa con `def[i].getDefinition()` e su questa chiamare il metodo `indexOf(args[0])`.

Si noti che passando sulla linea di comando un nome di organismo senza delimitarlo da doppi apici solo la prima stringa viene messa in `args[0]` e quindi possono essere associati più risultati che contengono tale stringa.

Vediamo ora un altro esempio:

```
public class GetGenesByEnzyme {
    public static void main(String[] args) {
        RESTCall serv= new RESTCall();
        String[] result;

        // recuperiamo il numero di geni dato l'organismo
        result = serv.get_genes_by_enzyme("ec:2.7.1.6", "ljo");
        System.out.println(result[0]);
    }
}
```

Il metodo `get_genes_by_enzyme()` accetta come parametro due stringhe, una corrispondente all'enzima e l'altra corrispondente all'id dell'organismo. Se si va ad analizzarne l'implementazione si vede che la chiamata REST corrispondente è <http://rest.kegg.jp/link/ljo/ec:2.7.1.6>

I due sorgenti appena visti non avrebbero molto senso se eseguiti da soli, ma componendo il codice del primo con il codice del secondo si incomincia a vedere l'utilità di applicazioni che automatizzano la ricerca.

## Esercizio 2

Provate quindi a scrivere un programma che dato il nome di un organismo ne ricavi il suo id e dall'id ricavi il gene associato all'enzima `ec:2.7.1.6` (la spiegazione della scrittura `ec:2.7.1.6` si trova più avanti nella dispensa).

Provatelo con input: “Lactobacillus johnsonii NCC 533”, il vostro codice dovrebbe restituire: ljo:LJ0859 (dove ljo è l'organismo e LJ0859 il gene, questa scrittura sarà spiegata più avanti nella dispensa).

Si noti che passando sulla linea di comando un nome di organismo senza delimitarlo da doppi apici solo la prima stringa viene messa in `args[0]` e quindi possono essere associati più risultati che contengono tale stringa. Questo caso deve essere rilevato (quanti parametri vedo sulla linea di comando?) e bisogna avvisare l'utente di delimitare il nome dell'organismo con doppi apici.

Passiamo ora a vedere una struttura diversa chiamata DBGET. Si tratta di un metodo di recupero delle informazioni formattate come file di testo, detti anche flat-file. La definizione di flat-file di KEGG non è limitata ai soli file di testo ma si estende a immagini GIF (per i pathway di KEGG), grafica 3D per la struttura di proteine, ecc.

La maggior parte degli attuali database biologici possono essere utilizzati in questa specifica.

Quando si passa un input ad un metodo di DBGET bisogna seguire un certo pattern:

**dbname:identifier**

ovvero nome del database (abbreviato) seguito dall'identificatore di ciò che ci interessa. Per una lista completa dei database si rimanda a: <http://www.genome.jp/dbget/>.

Il catalogo dei geni in KEGG considera anche come identificatore la combinazione organismo e gene:

**organismo:gene**

Vediamo subito un esempio di codice e cosa ci restituisce:

```
public class KeggEsDBGET {  
    public static void main(String[] args) {  
        RESTCall serv= new RESTCall();  
        String result = serv.bget(args[0]);  
        System.out.println(result);  
    }  
}
```

Provate a compilare ed eseguire questo codice con input “eco:b0004” dove in questo caso “eco” è l'organismo Escherichia coli e “b0004” è l'id del gene. Se si va ad analizzarne l'implementazione si vede che la chiamata REST corrispondente è <http://rest.kegg.jp/get/eco:b0004>

Il metodo `bget()` restituisce una stringa che contiene tutte le informazioni relative all'argomento della ricerca.

```
ENTRY      b0004          CDS      E.coli  
NAME       thrC, ECK0004, JW0003  
DEFINITION threonine synthase (EC:4.2.3.1)  
ORTHOLOGY  K01733 threonine synthase [EC:4.2.3.1]
```

PATHWAY      eco00260    Glycine, serine and threonine metabolism  
               eco00750    Vitamin B6 metabolism  
               eco01100    Metabolic pathways  
 CLASS        Metabolism; Amino Acid Metabolism; Glycine, serine and  
               threonine  
               metabolism [PATH:eco00260]  
               Metabolism; Metabolism of Cofactors and Vitamins; VitaminB6  
               metabolism [PATH:eco00750]  
 POSITION      3734..5020  
 MOTIF        Pfam: PALP  
               PROSITE: DEHYDRATASE\_SER\_THR  
 DBLINKS      NCBI-GI: 16127998  
               NCBI-GeneID: 945198  
               RegulonDB: B0004  
               EcoGene: EG11000  
               UniProt: P00934  
 STRUCTURE    PDB: 1VB3  
 AASEQ        428  
               MKLYNLKDHNEQVSFAQAVTQGLGKNQGLFFPHDLPEFSLTEIDEMLKLDVTRSakilS  
               AFIGDEIPQEILEERVRAAFAPVANVESDVGCLELFHGPTLAFKDFGGRFMAQMLTH  
               IAGDKPVTILTATSGDTGA AVAHAFYGLPNVKVVILYPRGKISPLQEKLFC TLGGNIETV  
               AIDGDFDACQALVKQAFDDEELKVALGLNSANSINISRLLAQICYFEAVAQLPQETRNQ  
               LVVSVPSGNFGDLTAGLLAKSLGLPVKRFIAATNVNDTVPRFLHDGQWSPKATQATLSNA  
               MDVSQPNNWPRVEELFRRKIWQLKELGYAAVDDETQQTMRELKELGYTSEPHAAVAYRA  
               LRDQINPGEYGLFLGTAHPAKFKESVEAILGETLDLPKELAEADLPLLSHNLPADFAAL  
               RKLMMNHQ  
 NTSEQ        1287  
               atgaaactctacaatctgaaagatcacaacgagcaggtcagctttgcgcaagccgtaacc  
               caggggttgggcaaaaatcaggggctgtttttccgcacgacctgccggaattcagcctg  
               actgaaattgatgagatgctgaagctggattttgtcaccgcagtgcggaagatcctctcg  
               gcgtttattggtgatgaaatcccacaggaaatcctggaagagcgcgctgcgcgcggcgcttt  
               gccttcccggtccggtcgccaatgttgaaagcgatgtcggttgctctggaattgttcac  
               gggccaacgctggcatttaagatttcggcggtcgctttatggcaciaaatgctgacctat  
               attgcgggtgataagccagtgaccattctgaccgcgacctccggtgataccggagcggca  
               gtggctcatgctttctacggtttaccgaatgtgaaagtggttatcctctatccacgaggg  
               aaaatcagtcactgcaagaaaaactgttctgtacattgggcggcaatatcgaaactgtt  
               gccatcgacggcgatttcgatgcctgtcaggcgctggtgaagcaggcggttgatgatgaa  
               gaactgaaagtggcgctaggggttaaaactcggctaactcgattaacatcagccgtttgctg  
               gcgcagatttgctactactttgaagctgttgcgcgagctgccgcaggagacgcgcaaccag  
               ctggttgctctcggtgccaagcggaaacttcggcgatttgacggcggtctgctggcgaag  
               tcaactcggctctgcgggtgaaacgttttattgctgcgaccaacgtgaacgataccgtgcca  
               cgtttcctgcacgacggtcagtggtcaccctaaagcgactcaggcgacgttatccaacgcg  
               atggacgtgagtcagccgaacaactggccgctgtggaagagttgttccgcgcaaaaatc  
               tggcaactgaaagagctgggttatgcagccgtggatgatgaaaccacgcaacagacaatg  
               cgtgagttaaaagaactgggctacacttcggagccgcacgctgccgtagcttatcgtgcg  
               ctgcgtgatcagttgaatccaggcgaatatggcttgcttcctcggcaccgcgcacccggcg  
               aaatttaaagagagcgtggaagcgattctcggtgaaacgttggtatctgcaaaagagctg  
               gcgaacgtgctgatttacccttgctttcacataatctgcccgccgattttgctgcgttg  
               cgtaaatgatgatgaatcatcagtaa

///

Chiaramente questa rappresentazione dei dati contiene un po' troppe informazioni e nessuna utilizzabile per qualche elaborazione successiva, è quindi necessario suddividere questa grossa stringa “ritagliando” solo le parti che ci interessano.

### Esercizio 3

Recuperare il valore di POSITION.

SUGGERIMENTO: utilizzate il metodo *indexOf()* per recuperare l'indice della parola POSITION e la posizione del primo punto dopo la parola POSITION con il metodo *indexOf(int ch, int fromIndex)*, poi recuperate la sottostringa con il metodo *substring(start\_index+12, end\_index)*.

### Esercizio 4

Scrivere un programma che:

1. Riceva il nome di un organismo come primo parametro sulla linea di comando e ne recuperi l'ID.

**ATTENZIONE:** abbiamo già visto un possibile codice che lo fa, ma ad una voce inserita dall'utente possono essere associati più risultati, sarà quindi necessario gestire questa evenienza (anche semplicemente prendendo il primo risultato).

2. Dato l'ID dell'organismo e una serie di enzimi, restituisca la posizione dei geni corrispondenti. Provate con i seguenti enzimi:
  - a. ec:2.7.1.6
  - b. ec:5.4.2.2

SUGGERIMENTO: usate il metodo *get\_genes\_by\_enzyme()* per recuperare il gene e passate ciò che viene restituito al metodo *bget()* già visto precedentemente. Notare che la stringa ottenuta va convertita in numero.

3. Recupera la lunghezza del genoma:

SUGGERIMENTO: usate sempre *bget()* con la stringa “gn:”+id\_organismo (vedere in Appendice B un esempio di output) e recuperate il numero che segue la stringa “LENGTH” estraendo la sotto-stringa da LENGTH+10 alla stringa “\n”. Notare che la stringa ottenuta va convertita in numero.

4. Tramite l'oggetto Canvas disegni una retta che identifica il genoma e un pallino che identifica la posizione degli enzimi sul genoma (vedere Appendice A).

Attenzione alla scala e disegnate ogni enzima su una retta diversa, come in Figura 1:



Figura 1

Provate usando come input “Lactobacillus johnsonii NCC 533” tra doppi apici. Se non usate i doppi apici o passate solo “johnsonii”, il vostro programma dovrebbe farvi scegliere attraverso un menù composto da indice e nome dell’organismo, tra altri organismi che contendono quella parola.

NOTA: il programma è solo un esempio prototipale e funziona bene solo con “Lactobacillus johnsonii NCC 533”; altri organismi potrebbero non contenere gli enzimi specificati oppure generare l'informazione della posizione con una sintassi diversa da quella vista e parserizzata negli esercizi.

# Appendice A

## Java e le primitive grafiche

Introduciamo brevemente le primitive che Java mette a disposizione del programmatore per la gestione della grafica elementare. Queste primitive consentono da un lato di implementare GUI (Graphical User Interface) interattive con l'utente, dall'altro di giocare con i costrutti geometrici fondamentali e realizzare ogni sorta di disegno e rappresentazione grafica. Vedremo dunque come implementare a livello applicativo la visualizzazione di un semplice grafico per la rappresentazione di figure geometriche elementari, al fine di visualizzare semplici rappresentazioni grafiche dei risultati.

### Panoramica sul package AWT

Di seguito faremo riferimento all'Abstract Window Toolkit, la libreria contenente le classi e le interfacce fondamentali per la creazione di elementi grafici, inserita nelle API standard di Java.

I package fondamentali in cui si articola sono i seguenti:

- `java.awt` package principale che contiene le classi di tutti i componenti che possiamo utilizzare nella GUI, quali ad esempio `Frame`, `Panel`, `Button`, `Label`, `Checkbox`, `Canvas`, `Menu`, `MenuBar`, `TextArea` e molti altri
- `java.awt.event` fornisce le classi per la gestione degli “eventi”, ovvero il sistema che awt utilizza per passare il controllo al programmatore in seguito ad azioni avvenute sui componenti, come la pressione di un bottone, il passaggio del mouse su un componente, l'apertura di una finestra

La classe che rappresenta una “finestra” di interazione grafica con l'utente è la `java.awt.Frame`. Essa presenta fondamentalmente la classica barra del titolo e un bordo.

Ci concentreremo sull'uso di oggetti `Canvas`. `Canvas` è un componente che rappresenta un rettangolo vuoto dello schermo all'interno del quale è possibile disegnare oggetti geometrici elementari.

Vediamo subito un esempio per spiegare come utilizzare questo componente.

```
import java.awt.*;
import java.awt.event.*;
public class DrawExample extends Canvas {

    public void paint(Graphics g){
        g.drawRect(10, 50, 400, 1);
    }
    public static void main (String args[]){
        Canvas frame = new DrawExample();
        Frame finestra = new Frame();
        finestra.add(frame);
        finestra.setSize(450, 90);
        finestra.setResizable(false);
        finestra.setVisible(true);
    }
}
```

Come vedete la classe DrawExample estende Canvas, infatti per poter disegnare qualcosa sul canvas è necessario ridefinire il metodo paint(). L'oggetto passato come parametro al metodo paint() è un oggetto Graphics, che definisce un contesto grafico con il quale disegnare sul Canvas.

Graphics mette a disposizione molti metodo per disegnare delle figure geometriche elementari quali rettangoli, cerchi, rette ecc. Nell'esempio si utilizza il metodo drawRect(int x, int y, int width, int height) i cui argomenti sono abbastanza autoesplicativi.

Analizziamo ora il main del progetto:

- Come prima cosa si crea un oggetto della classe che estende Canvas e quindi definisce cosa bisogna disegnare;
- L'oggetto Canvas appena creato deve essere messo in un Frame per poterlo visualizzare a video e quindi creiamo un oggetto di tipo Frame;
- Una volta creato il Frame basta impostare le dimensioni, togliere il resize della finestra e settare la visibilità.

Come già detto Graphics mette a disposizione un metodo per quasi tutte le forme geometriche elementari, si rimanda quindi alla Javadoc di Graphics per una lista completa:

(Graphics: <http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Graphics.html>)

## Appendice B

### Esempio di output della chiamata bget() con argomento “gn:”+id\_organismo

```
ENTRY      T00158          Complete Genome
NAME       ljo, L.johnsonii, LACJO, 257314
DEFINITION Lactobacillus johnsonii NCC 533
ANNOTATION manual
TAXONOMY   TAX:257314
    LINEAGE Bacteria; Firmicutes; Lactobacillales; Lactobacillaceae;
            Lactobacillus
DATA_SOURCE RefSeq
ORIGINAL_DB Nestle
CHROMOSOME Circular
    SEQUENCE RS:NC_005362
    LENGTH 1992676
STATISTICS Number of nucleotides:      1992676
            Number of protein genes:    1821
            Number of RNA genes:        97
REFERENCE  PMID:14966310
    AUTHORS Pridmore RD, et al.
    TITLE   The genome sequence of the probiotic intestinal bacterium
            Lactobacillus johnsonii NCC 533.
    JOURNAL Proc Natl Acad Sci U S A : (2004)
///
```

## Appendice C

### Descrizione della classe Definition

Definition è una classe che contiene informazioni su un organismo; di conseguenza un array di Definition contiene un organismo per ogni valore.

```
public class Definition {
    private String definition=null;
    private String Entry_id;
    private String org;

    public String getDefinition() {
        return definition;
    }
    public void setDefinition(String definition) {
        this.definition = definition;
    }
    public String getEntry_id() {
        return Entry_id;
    }
    public void setEntry_id(String entry_id) {
        Entry_id = entry_id;
    }
    public String getOrg() {
        return org;
    }
    public void setOrg(String org) {
        this.org = org;
    }
}
```