Provider di servizi Web Rest in Java

Per eseguire un provider di servizi Web in Java, deve essere disponibile un ambiente di esecuzione adatto, quindi un *Servlet Container* o un *Server di Applicazioni JEE*.
Il servizio Web sarà installato in un **Servlet Container** mediante un archivio Web strutturato **.war** (*Web application ARchive*), di cui si è parlato nel capitolo precedente: esso contiene un'applicazione Web dinamica, configurata per consentire l'accesso al provider di servizi Web.

PROGETTO 3

Realizzare un provider di servizi Web REST in Java che visualizza un elenco di città.

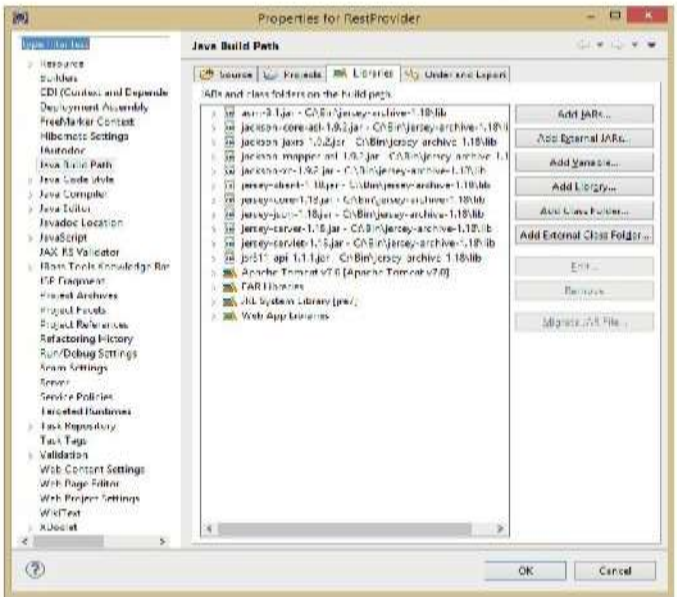
Per sviluppare il provider di servizi Web REST in Java utilizzeremo l'IDE di sviluppo *Eclipse*, creando un nuovo progetto di tipo *Sito Web Dinamico* (**Dynamic Web Project**).
Per il sito Web è necessario, come prerequisito, definire anche l'ambiente di esecuzione (il *Server di Applicazioni JEE* o il *Servlet Container* in cui sarà installato il sito Web dinamico).
Dopo aver scaricato ed estratto (o installato) il *Servlet Container* **Apache Tomcat**, si può procedere con la creazione guidata del progetto: dal menu **File** si seleziona **New** e poi **Project**.

Per aggiungere un nuovo provider di servizi Web REST, abbiamo bisogno di utilizzare una libreria che implementi lo standard JAX-RS definito per la piattaforma Java.
L'implementazione più utilizzata per i servizi REST è attualmente *Jersey*, la cui distribuzione, in formato compresso, può essere scaricata dal sito del progetto all'indirizzo *https://jersey.java.net*.
Nella sezione *Download* è disponibile il link all'archivio compresso della versione 1.x della libreria, che utilizziamo in questo testo; il download è contrassegnato dalla dicitura *Jersey 1.x ZIP bundle* dove *1.x* è la versione del *bundle*.
L'archivio deve essere estratto in una directory sul proprio sistema: per esempio, scarichiamo la versione 1.18 ed estraiamo nel percorso *C:\Bin\ jersey-archive-1.18*.
Per includere nel progetto le librerie *Jersey*, è necessario configurare il percorso di generazione Java (**Java Build Path**): fare clic con il testo destro sul progetto di sito Web dinamico e selezionare la voce **Build Path** e poi **Configure Build Path**.
Nella scheda **Libraries** premere il pulsante **Add External JARs** e individuare i seguenti file nell'archivio di *Jersey* estratto sul proprio disco:

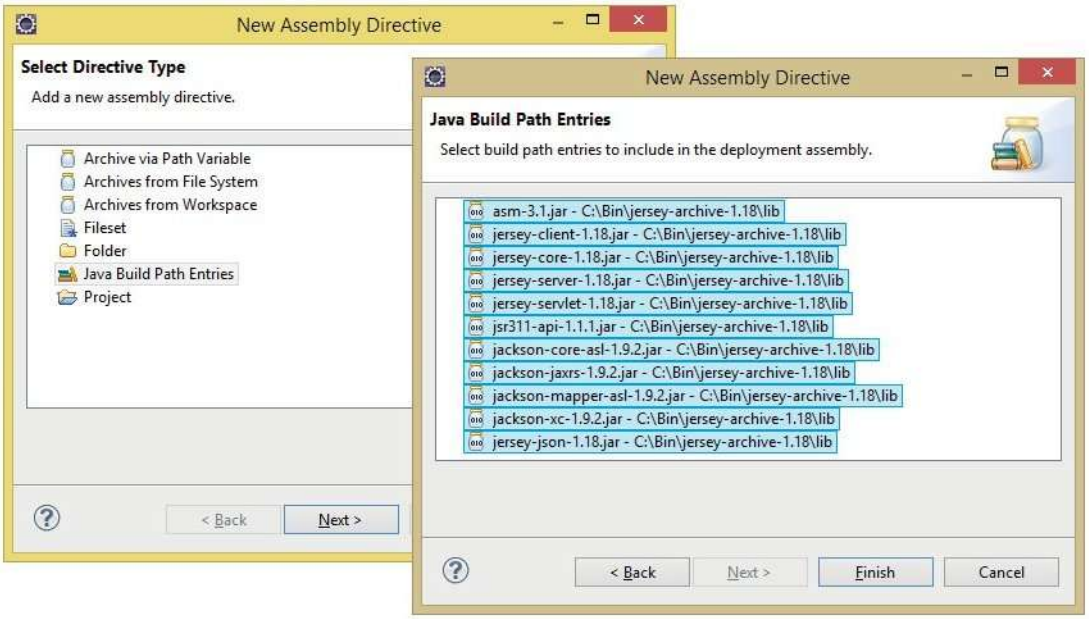
- asm-3.1.jar
- jersey-client-1.18.jar
- jersey-core-1.18.jar
- jersey-server-1.18.jar
- jersey-servlet-1.18.jar
- jsr311-api-1.1.1.jar

Le risposte che i servizi REST restituiscono ai client dovranno essere di tipo JSON. Come visto nel primo paragrafo, JSON è un formato adatto all'interpretazione delle informazioni da parte di programmi automatici. Per supportare questo formato per i risultati dei servizi, dobbiamo aggiungere al percorso di generazione i riferimenti alla libreria *Jackson*, ampiamente utilizzata per le operazioni di trasformazione da e verso il formato JSON.
Aggiungiamo quindi anche le seguenti librerie, presenti nella distribuzione di *Jersey*:

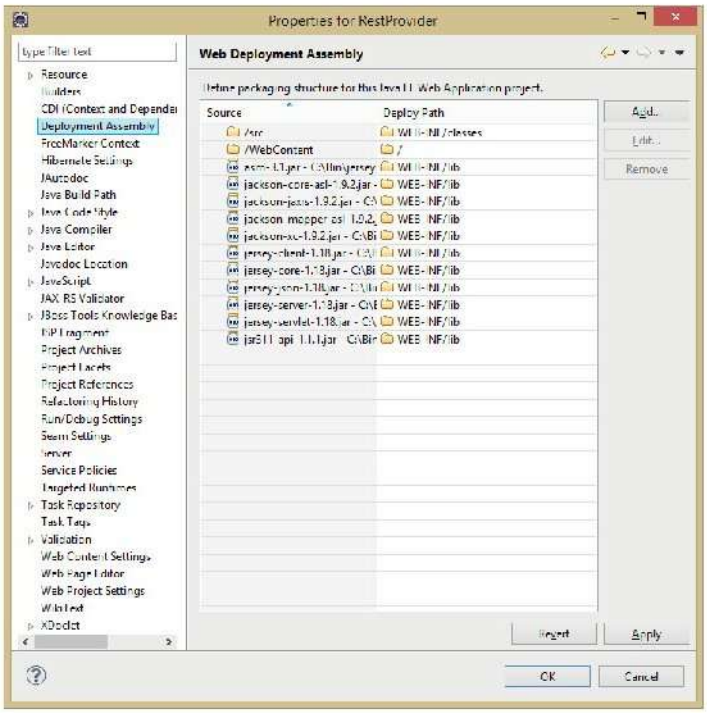
- jackson-core-asl-1.9.2.jar
- jackson-jaxrs-1.9.2.jar
- jackson-mapper-asl-1.9.2.jar
- jackson-xc-1.9.2.jar
- jersey-json-1.18.jar



Oltre ad aggiungere le librerie, queste vanno anche referenziate nel descrittore di distribuzione, in modo che siano copiate nel percorso appropriato al momento dell'installazione. Per configurare questo aspetto del sito Web dinamico, dal pannello delle proprietà del progetto, si deve selezionare la voce **Deployment Assembly**, quindi fare clic sul pulsante **Add**.
È necessario aggiungere tutte le voci presenti nel percorso di generazione, come indicato nelle figure seguenti.



Il risultato finale è illustrato in figura.



Per caricare l'infrastruttura *Jersey*, è necessario indicare la corretta configurazione nel descrittore di distribuzione **web.xml**. Come in ogni applicazione Web dinamica JEE, il file di configurazione *web.xml* si trova nella cartella *WEB-INF* del sito Web. Nei progetti di tipo *Dynamic Web Project* in *Eclipse*, il sito Web è posizionato nella cartella *WebContent*.

File web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   id="WebApp_ID" version="2.5">
   <display-name>RestProvider</display-name>
   <servlet>
      <servlet-name>Jersey REST Service</servlet-name>
      <servlet-class>
         com.sun.jersey.spi.container.servlet.ServletContainer
      </servlet-class>
      <init-param>
         <param-name>
            com.sun.jersey.config.property.packages
         </param-name>
         <param-value>servizi</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>Jersey REST Service</servlet-name>
      <url-pattern>/rest/*</url-pattern>
   </servlet-mapping>
</web-app>
```

Abbiamo configurato il *servlet* di *Jersey* che caricherà i servizi REST definiti nel sorgente Java del programma, per cercare tali servizi nel package Java denominato *servizi*: definiamo il servizio REST in una classe di tale package.

La classe che implementa il servizio REST è riportata di seguito.

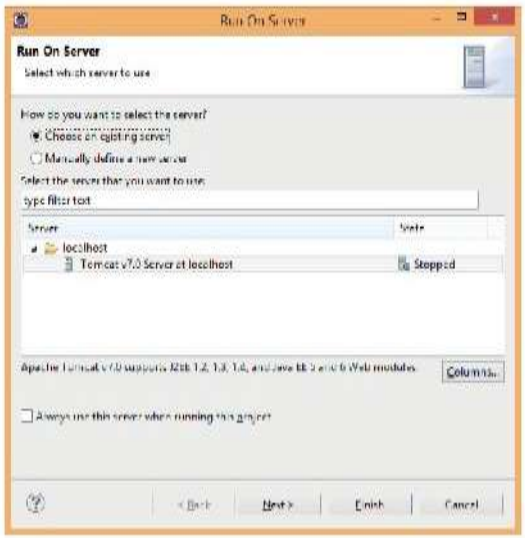
Classe ServizioREST (*ServizioREST.java*)

```
package servizi;

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

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("citta")
public class ServizioREST {
   @GET
   @Path("/elenco")
   @Produces(MediaType.APPLICATION_JSON)
   public List<String> elencoCitta() {
      List<String> elenco = new ArrayList<String>();
      elenco.add("Milano");
      elenco.add("Roma");
      elenco.add("Torino");
      elenco.add("Napoli");
      return elenco;
   }
}
```

Come si può notare dall'implementazione del servizio, *Jersey* ricava la definizione del servizio dalle annotazioni Java riportate nella classe. Nel nostro caso abbiamo implementato le seguenti caratteristiche del servizio:

- URL principale del servizio `@Path("citta")`: il servizio REST sarà accessibile all'URL relativo indicato nell'annotazione.
- URL del metodo `@Path("/elenco")`: nel perimetro della definizione del servizio, il metodo che produce l'elenco delle città, sarà accessibile all'URL relativo */elenco*.
- Verbo del metodo `@GET`: il metodo sarà accessibile attraverso il verbo GET del protocollo HTTP.
- Tipo di informazione prodotta `@Produces(MediaType.APPLICATION_JSON)`: il risultato del metodo sarà una stringa JSON.

Come si vede dalla configurazione dell'applicazione nel file *web.xml*, il servlet di *Jersey* risponde a tutti gli URL che iniziano con la stringa */rest* (*url-pattern*).
Ora avviamo il server *Tomcat*, configurato precedentemente nell'ambiente *Eclipse*: facciamo clic con il tasto destro sul progetto *RestProvider* e selezioniamo il comando **Run As** e poi **Run on Server**.



Facendo clic sul pulsante **Finish**, il server *Tomcat* carica l'applicazione.
Ora possiamo aprire un browser Web e scrivere il seguente URL nella casella dell'indirizzo:

```
http://localhost:8080/RestProvider/rest/citta/elenco
```

Il risultato del nuovo servizio REST, mostrato dal browser, è il seguente:

```
["Milano","Roma","Torino","Napoli"]
```