Lab 7
RESTful Web Service

Multi-thread based concurrent server programming

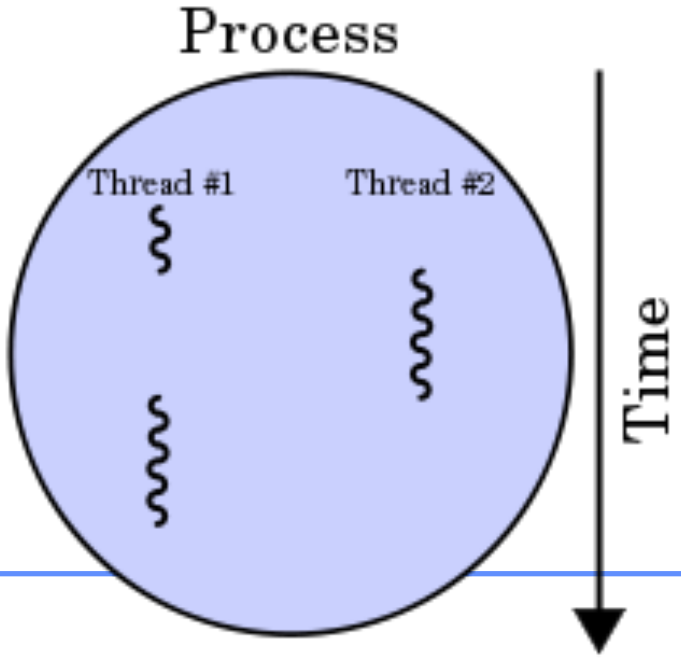
Why non-concurrency happens?

- Process resource (CPU, especially program counter) occupied by blocking functions

```
                         for ( ; ; ) {
                             clilen = sizeof(cliaddr);
No longer available ———→     connfd = Accept(listenfd, (SA *) &cliaddr, &clilen);
        Stucked     ———→     str_echo(connfd);
                             Close(connfd);
                         }
```

Thread

- In computer science, a **thread** of execution is the **smallest sequence of programmed instructions** that can be managed independently by a scheduler
- Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources.



Context Switching

- Single processor system
  - Generally implement multithreading by time slicing: the CPU switches between different software threads.
- Multiprocessor system
  - Every processor executing a separate thread simultaneously on a processor with hardware threads. software threads can also be executed concurrently by separate hardware threads.

RESTful Web Service Programming with IntelliJ

Example of HTTP Request/Response

- **Client request (Header)**

```
GET /index.html HTTP/1.1 Host: www.example.com
```

- **Server response (Header)**

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux) Last-Modified: Wed, 08
Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b“
Accept-Ranges: none Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Example of RESTful Web Service

- Openstack API is composed of RESTful Web Service

Ex) Virtual Machine Instance Creation Request API & Response (With JSON Format)

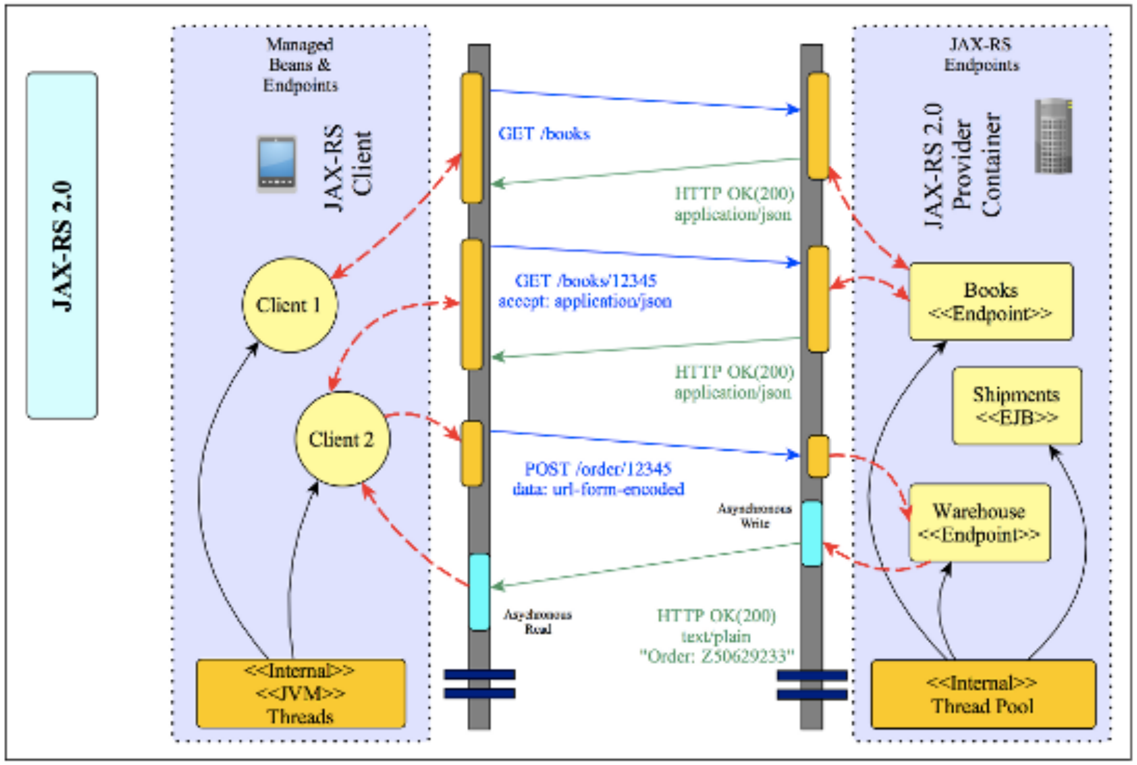
```
POST /v2/214412/servers HTTP/1.1 Host:
servers.api.openstack.org Content-Type: application/json
Accept: application/xml X-Auth-Token: eaaafd18-0fed-
4b3a-81b4-663c99ec1cbb

{
    "server" : {
        "name" : "new-server-test",
        "imageRef" :
        "http://servers.api.openstack.org/1234/images/5241
        5800-8b69-11e0-9b19-734f6f006e54",
        "flavorRef" : "52415800-8b69-11e0-9b19-
        734f1195ff37"
    }
}
```

```
HTTP/1.1 200 OK Date: Mon, 12 Nov 2007 15:55:01 GMT Server:
Apache Content-Length: 1863 Content-Type: application/xml;
charset=UTF-8

{
    "server": {
        "id": "52415800-8b69-11e0-9b19-734f565bc83b",
        "tenant_id": "1234",
        "user_id": "5678",
        "name": "new-server-test",
        "created": "2010-11-11T12:00:00Z",
        "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
        "accessIPv4" : "67.23.10.138",
        "accessIPv6" : "::babe:67.23.10.138",
        "progress": 0,
        "status": "BUILD",
        "adminPass": "GFf1j9aP",
        "image" : {
            "id": "52415800-8b69-11e0-9b19-734f6f006e54",
            "name": "CentOS 5.2",

…
```

For more API specification, refer the document
http://docs.openstack.org/api/api-specs.html

JAX-RS

- JAX-RS: Java API for RESTful Web Services (JAX-RS)
  - A Java programming language API spec to support the Representational State Transfer (REST) architecture.
  - JAX-RS uses annotations, to simplify the development and deployment of web service clients and endpoints.


JAX-RS 2.0
Managed Beans & Endpoints
JAX-RS Client
Client 1
Client 2
<<Internal>> <<JVM>> Threads
GET /books
HTTP OK(200) application/json
GET /books/12345 accept: application/json
HTTP OK(200) application/json
POST /order/12345 data: url-form-encoded
Asynchronous Write
Asynchronous Read
HTTP OK(200) text/plain "Order: Z50629233"
JAX-RS Endpoints
JAX-RS 2.0 Provider Container
Books <<Endpoint>>
Shipments <<EJB>>
Warehouse <<Endpoint>>
<<Internal>> Thread Pool

JAX-RS

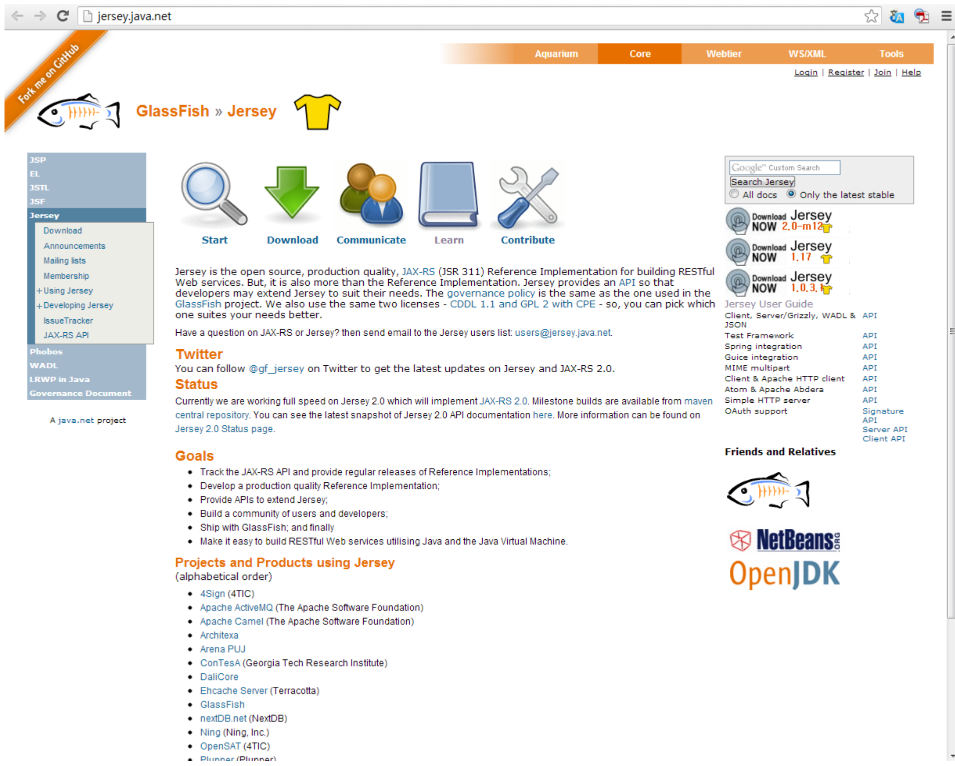
- JAX-RS provides some annotations to aid in mapping a resource class (a POJO) as a web resource. The annotations include:
  - *@Path* specifies the relative path for a resource class or method.
  - *@GET, @PUT, @POST, @DELETE and @HEAD* spe cify the HTTP request type of a resource.
  - *@Produces* specifies the response Internet media types (used for content negotiation).
  - *@Consumes* specifies the accepted request Internet media types.

Implementation of JAX-RS

- Implementations of JAX-RS include:
  - Apache CXF, an open source Web service framework.
  - **Jersey**, the reference implementation from Sun (now Oracle).
  - RESTeasy, JBoss's implementation.
  - Restlet, created by Jerome Louvel, a pioneer in REST frameworks.
  - Apache Wink, Apache Software Foundation Incubator project, the server module implements JAX-RS.
  - WebSphere Application Server from IBM via the "Feature Pack for Communications Enabled Applications"
  - WebLogic Application Server from Oracle, see notes

Jersey

- http://jersey.java.net
- Jersey is the open source, production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.
- Also more than the Reference (JAX-RS) Implementation. Jersey provides an API so that developers may extend Jersey to suit their needs
- Download Link: http://jersey.java.net/nonav/documentation/latest/chapter_deps.html



Jersey is the open source, production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services. But, it is also more than the Reference Implementation. Jersey provides an API so that developers may extend Jersey to suit their needs. The governance policy is the same as the one used in the GlassFish project. We also use the same two licenses - CDDL 1.1 and GPL 2 with CPE - so, you can pick which one suites your needs better.

Have a question on JAX-RS or Jersey? then send email to the Jersey users list: users@jersey.java.net.

Twitter

You can follow @gf_jersey on Twitter to get the latest updates on Jersey and JAX-RS 2.0.

Status

Currently we are working full speed on Jersey 2.0 which will implement JAX-RS 2.0. Milestone builds are available from maven central repository. You can see the latest snapshot of Jersey 2.0 API documentation here. More information can be found on Jersey 2.0 Status page.

Goals

- Track the JAX-RS API and provide regular releases of Reference Implementations;
- Develop a production quality Reference Implementation;
- Provide APIs to extend Jersey;
- Build a community of users and developers;
- Ship with GlassFish; and finally
- Make it easy to build RESTful Web services utilising Java and the Java Virtual Machine.

Projects and Products using Jersey
(alphabetical order)

- 4Sign (4TIC)
- Apache ActiveMQ (The Apache Software Foundation)
- Apache Camel (The Apache Software Foundation)
- Architexa
- Arena PUJ
- ConTesA (Georgia Tech Research Institute)
- DaliCore
- Ehcache Server (Terracotta)
- GlassFish
- nextDB.net (NextDB)
- Ning (Ning, Inc.)
- OpenSAT (4TIC)
- Plunner (Plunner)

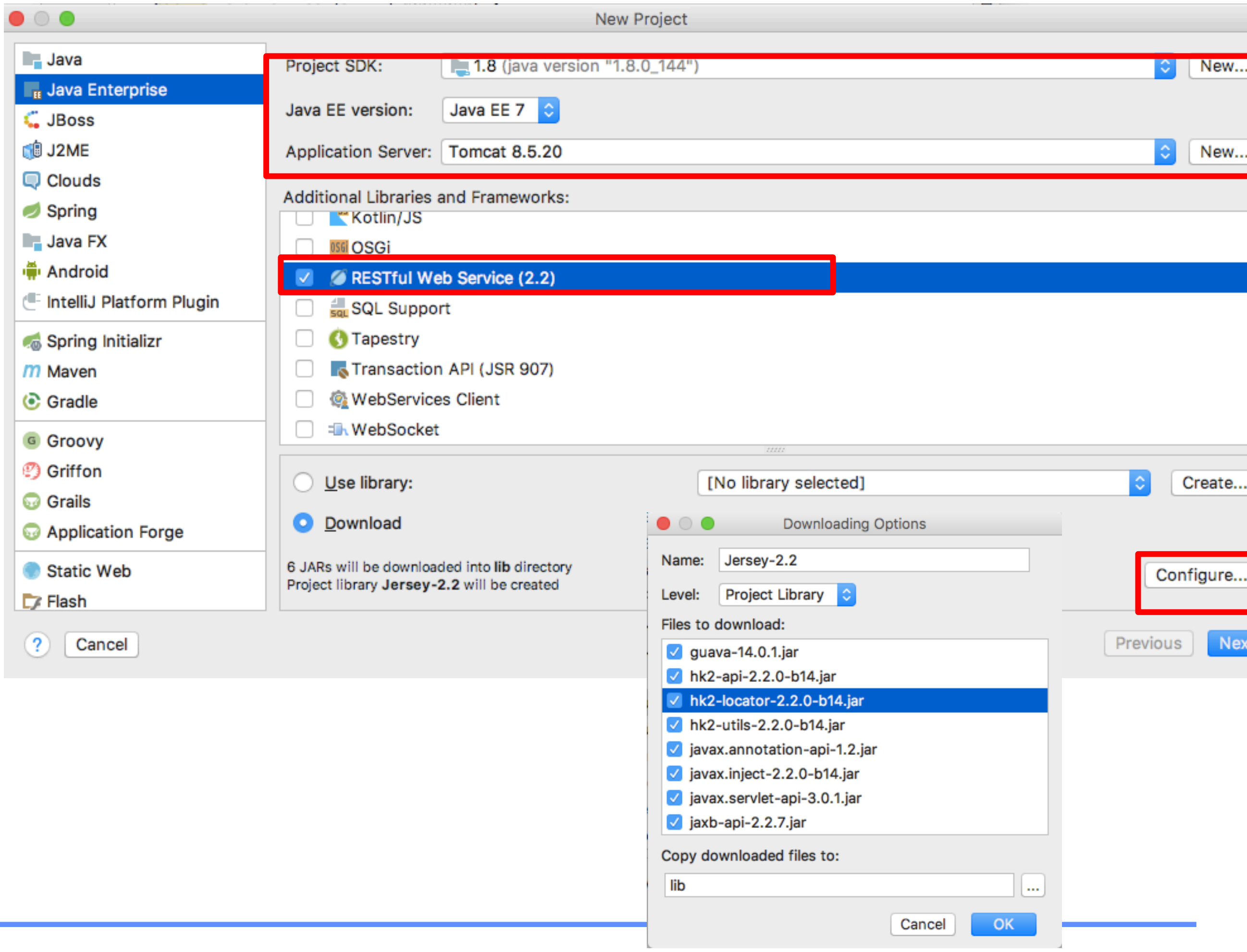
Jersey User Guide

Client, Server/Grizzly, WADL & JSON	API
Test Framework	API
Spring integration	API
Guice integration	API
MIME multipart	API
Client & Apache HTTP client	API
Atom & Apache Abdera	API
Simple HTTP server	API
OAuth support	Signature API Server API Client API

Friends and Relatives

Create Web Service Project

- Java Enterprise → Web Application → RESTful Web Service
- Select Application Server as 'Tomcat'
- In lib configuration, select all possible libraries



Project SDK: 1.8 (java version "1.8.0_144")

Java EE version: Java EE 7

Application Server: Tomcat 8.5.20

Additional Libraries and Frameworks:

[✓] RESTful Web Service (2.2)

Use library: [No library selected]

[✓ Download]

6 JARs will be downloaded into **lib** directory
Project library **Jersey-2.2** will be created

Name: Jersey-2.2

Level: Project Library

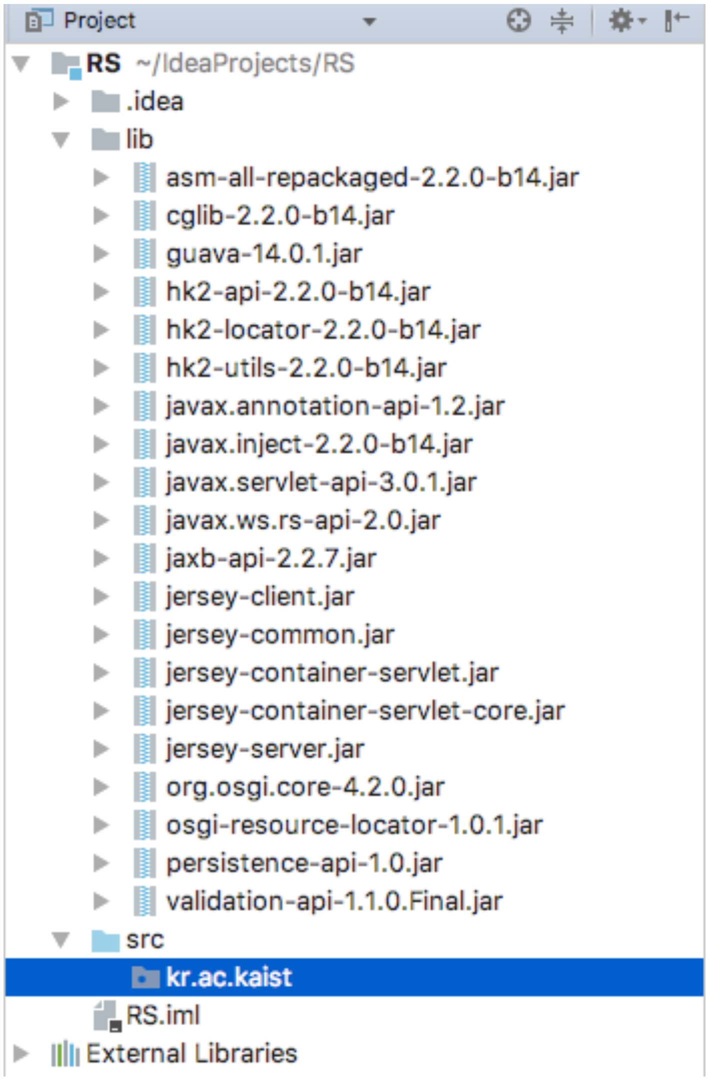
Files to download:

- [✓] guava-14.0.1.jar
- [✓] hk2-api-2.2.0-b14.jar
- [✓] hk2-locator-2.2.0-b14.jar
- [✓] hk2-utils-2.2.0-b14.jar
- [✓] javax.annotation-api-1.2.jar
- [✓] javax.inject-2.2.0-b14.jar
- [✓] javax.servlet-api-3.0.1.jar
- [✓] jaxb-api-2.2.7.jar

Copy downloaded files to: lib

RESTful Service Implementation

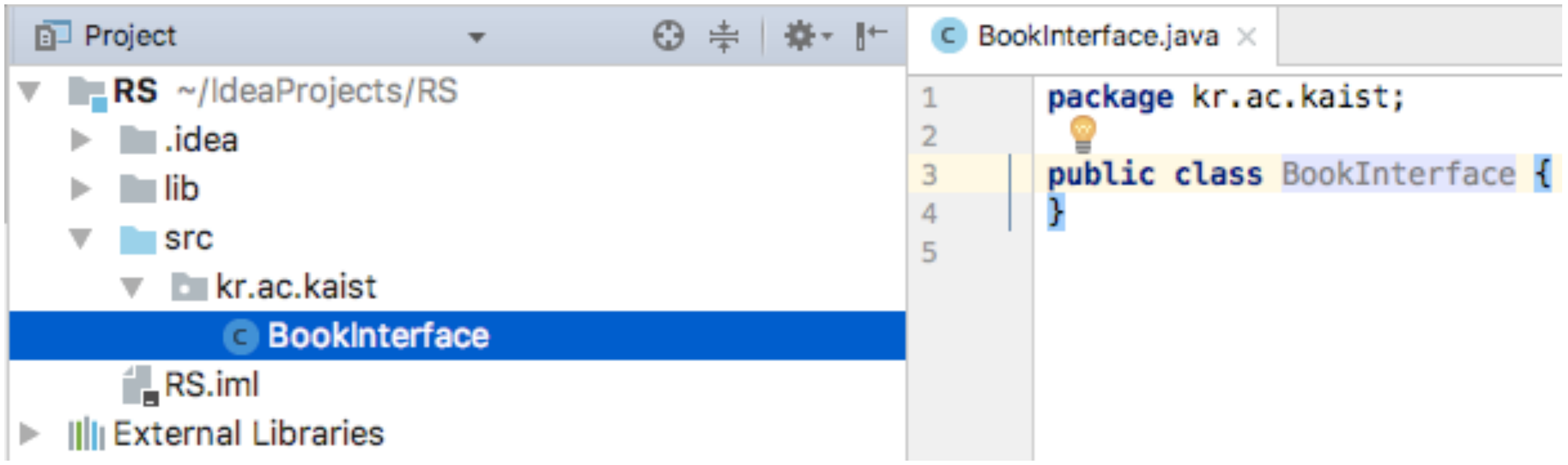
1. Jersey library files will be downloaded to /lib directory
2. Create package 'kr.ac.kaist' and java class to /src directory



```
Project
RS ~/IdeaProjects/RS
  .idea
  lib
    asm-all-repackaged-2.2.0-b14.jar
    cglib-2.2.0-b14.jar
    guava-14.0.1.jar
    hk2-api-2.2.0-b14.jar
    hk2-locator-2.2.0-b14.jar
    hk2-utils-2.2.0-b14.jar
    javax.annotation-api-1.2.jar
    javax.inject-2.2.0-b14.jar
    javax.servlet-api-3.0.1.jar
    javax.ws.rs-api-2.0.jar
    jaxb-api-2.2.7.jar
    jersey-client.jar
    jersey-common.jar
    jersey-container-servlet.jar
    jersey-container-servlet-core.jar
    jersey-server.jar
    org.osgi.core-4.2.0.jar
    osgi-resource-locator-1.0.1.jar
    persistence-api-1.0.jar
    validation-api-1.1.0.Final.jar
  src
    kr.ac.kaist
  RS.iml
External Libraries
```

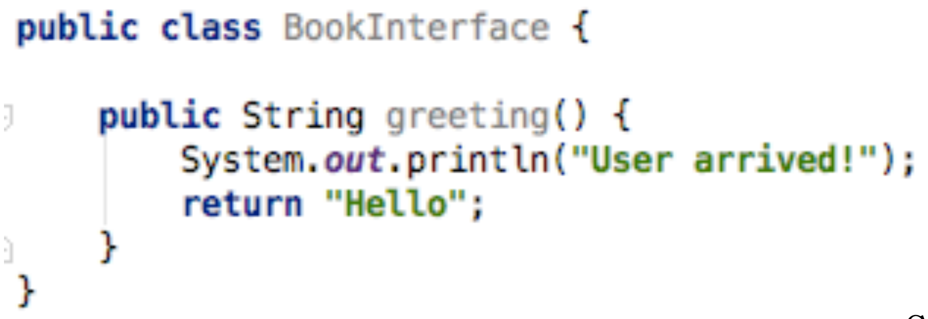
RESTful WS Programming

- Under package 'kr.ac.kaist', create Java Class named 'BookInterface'



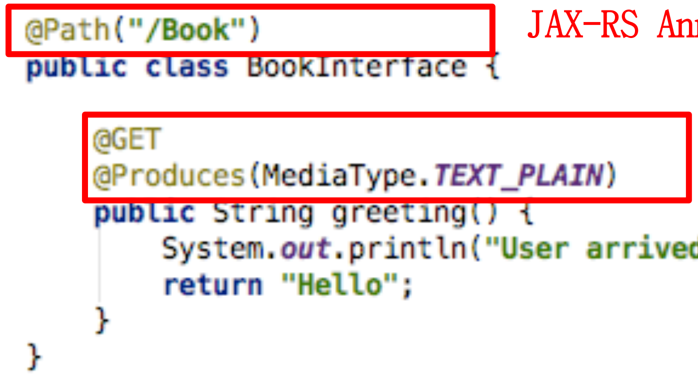
```java
package kr.ac.kaist;

public class BookInterface {
}
```

Interface Description with JAX-RS Annotation



```
public class BookInterface {

    public String greeting() {
        System.out.println("User arrived!");
        return "Hello";
    }
}
```

Code at Lab Materials
 - Lab7/src/BookInterface0.java

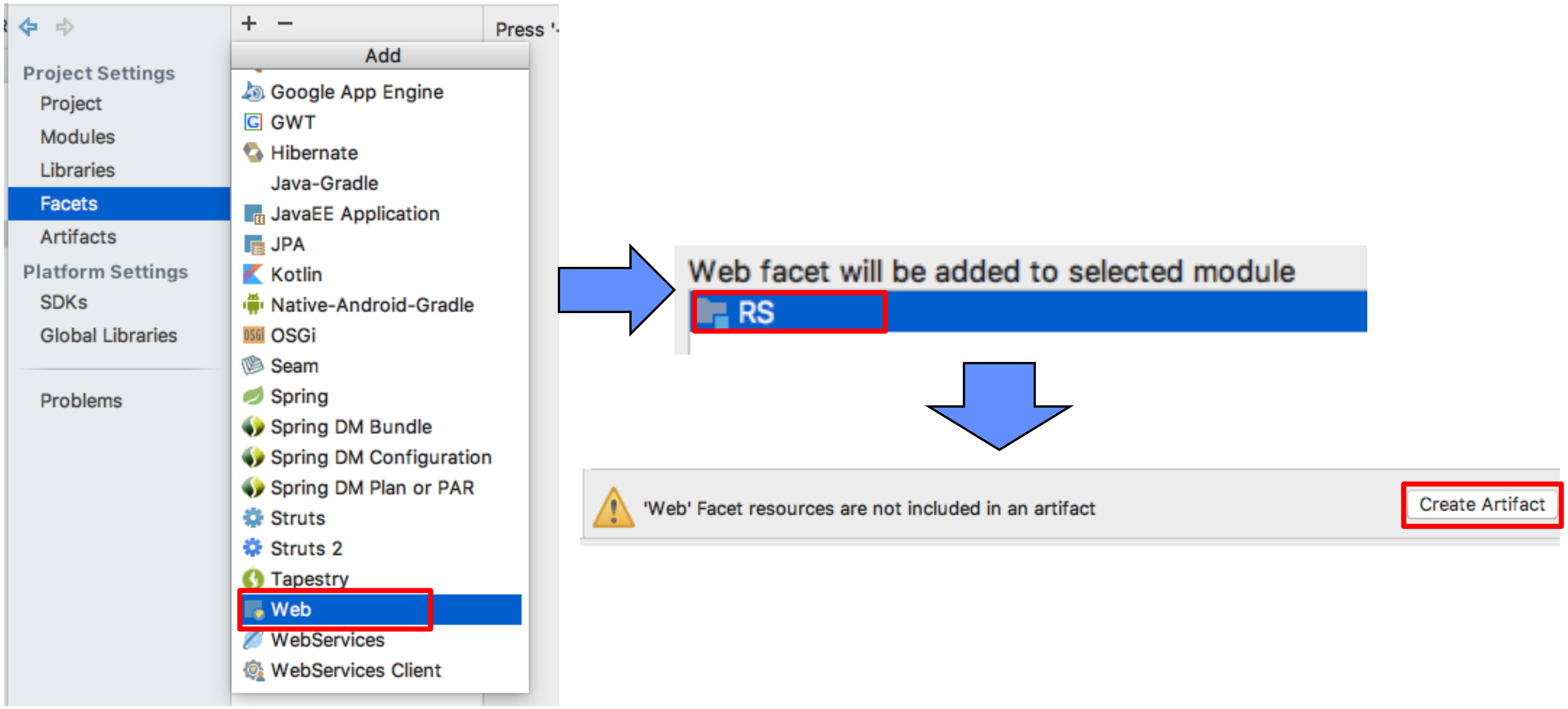


```
@Path("/Book")
public class BookInterface {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String greeting() {
        System.out.println("User arrived!");
        return "Hello";
    }
}
```

JAX-RS Annotation API for declaring router path

JAX-RS Annotation API for declaring method

Add Facets

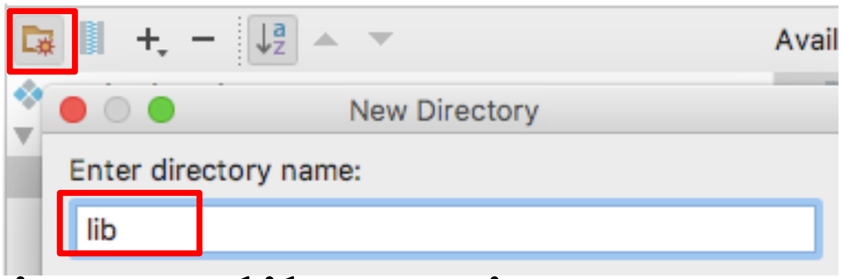


Web facet will be added to selected module

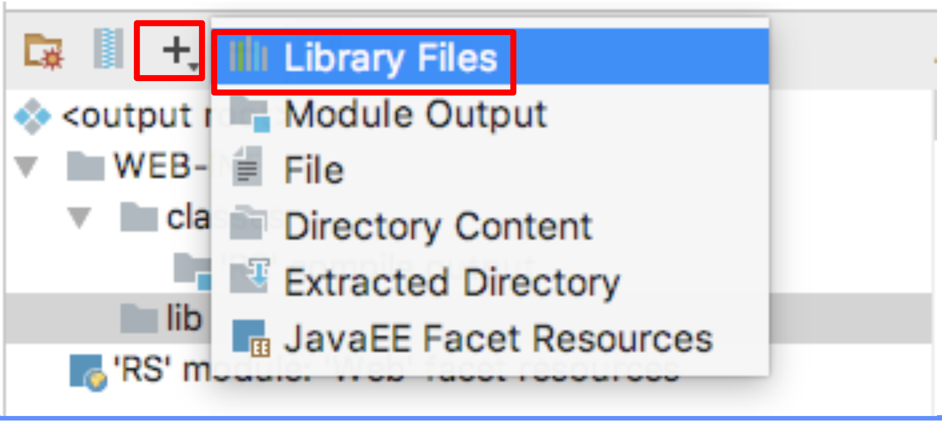
'Web' Facet resources are not included in an artifact

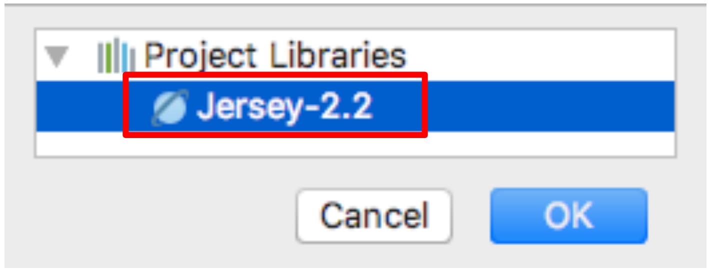
Configure Artifacts

- Create lib dir into WEB-INF



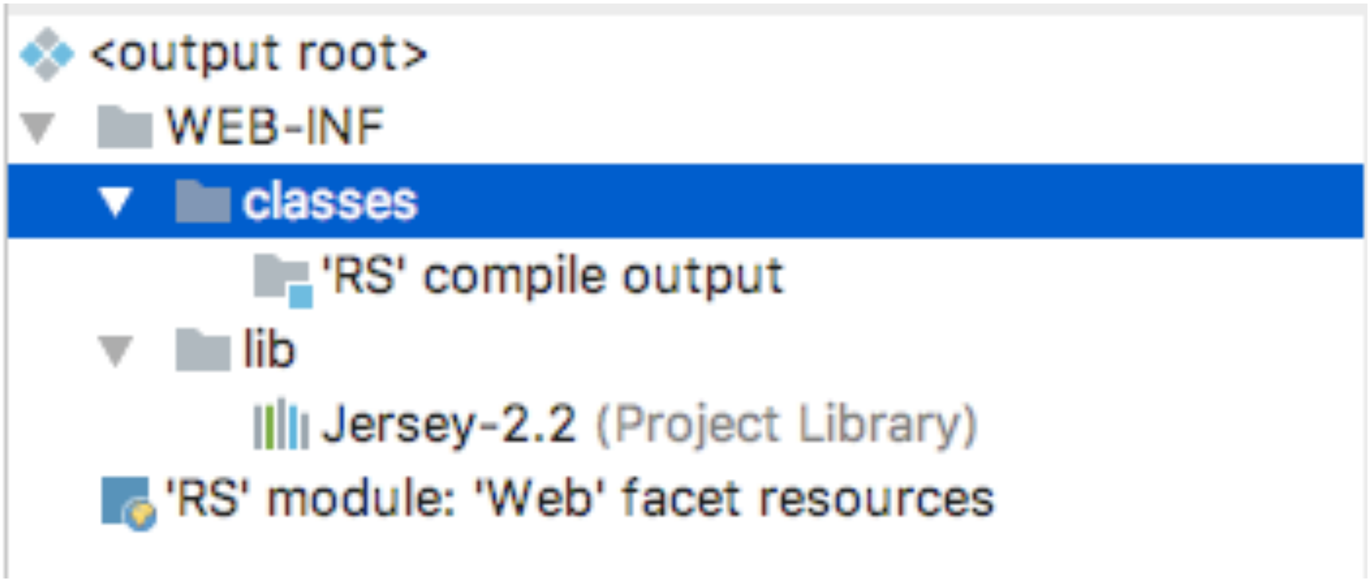
- Add a jersey library into WEB-INF/lib Directory





Configure Artifacts

- The artifacts should be set to the following structure

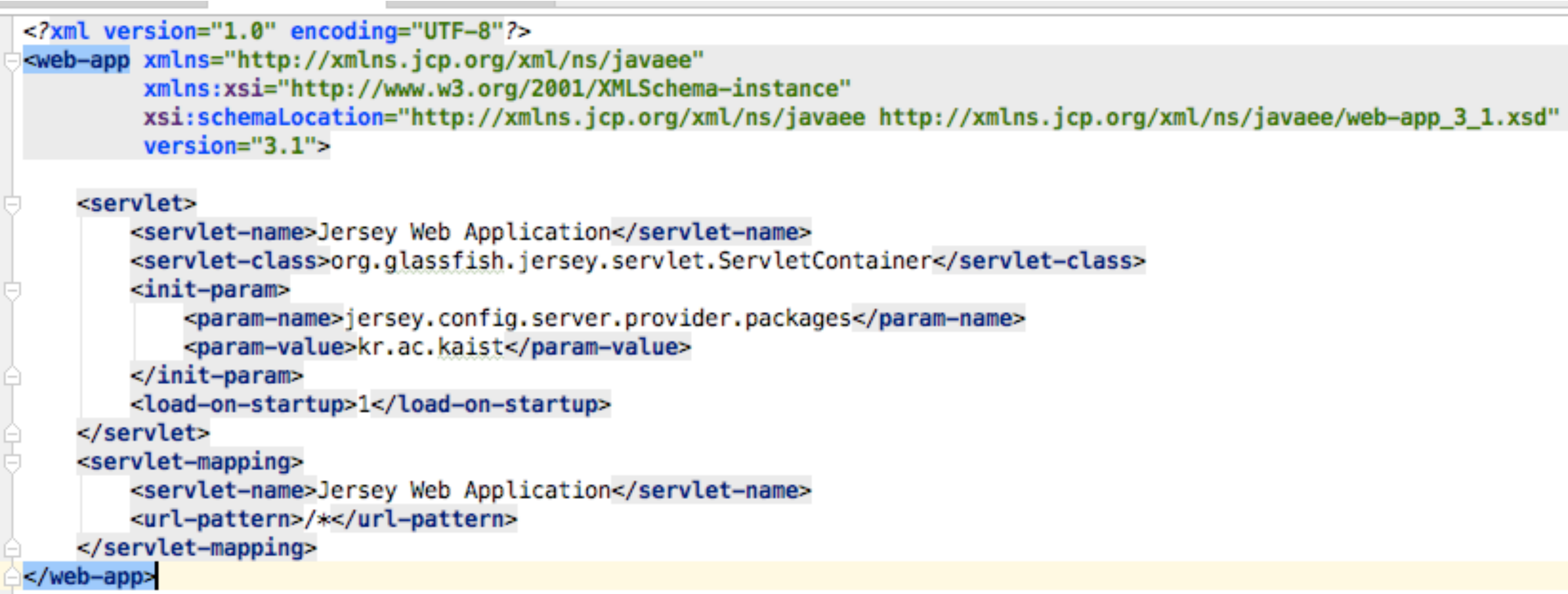


```
<output root>
▼ WEB-INF
   ▼ classes
         'RS' compile output
   ▼ lib
         Jersey-2.2 (Project Library)
   'RS' module: 'Web' facet resources
```

Web Configuration

- Edit web/WEB-INF/web.xml for declaring Web application

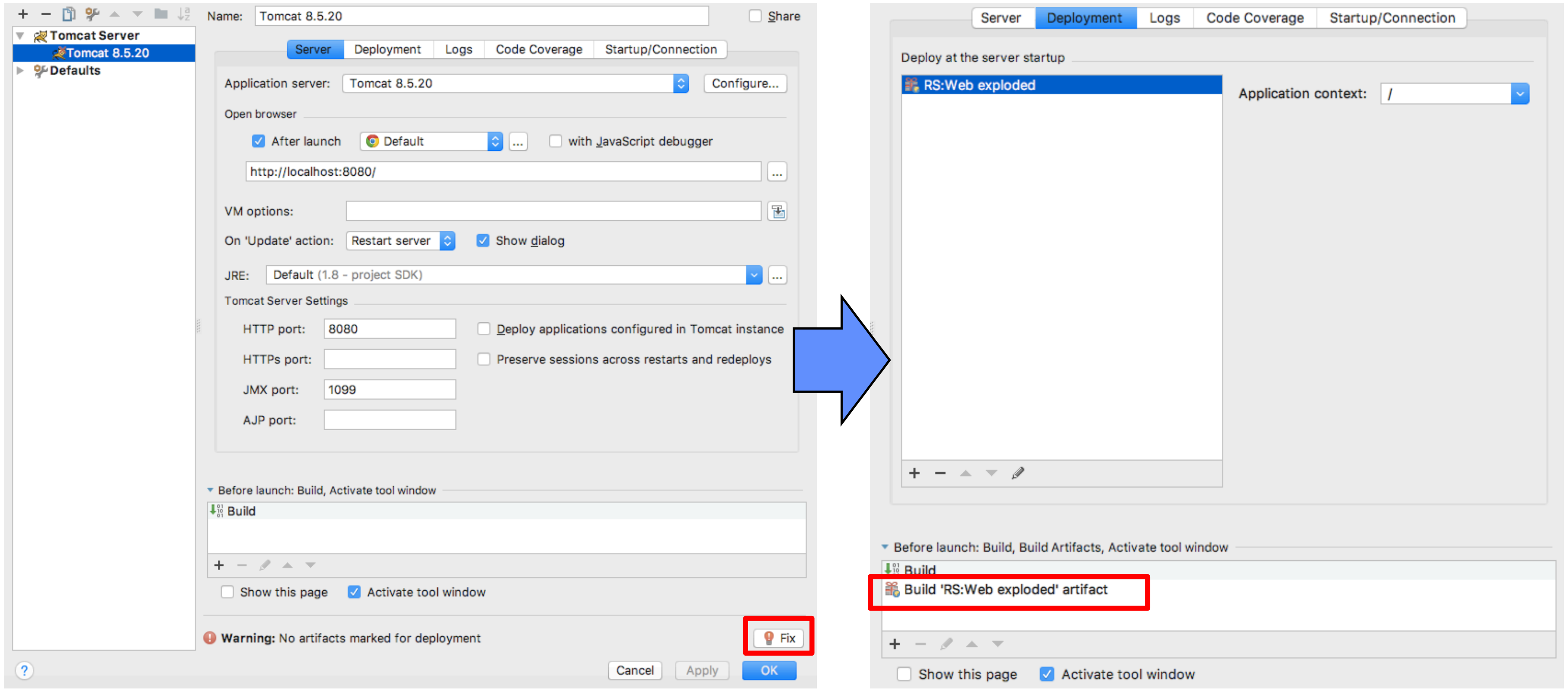
Code at Lab Materials
 - Lab7/src/web.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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"
         version="3.1">

    <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>kr.ac.kaist</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Deploy artifacts

- Add ‘RS:Web exploded’ artifacts to deploy


Name: Tomcat 8.5.20
Tomcat Server
Tomcat 8.5.20
Defaults
Share

Server | Deployment | Logs | Code Coverage | Startup/Connection

Application server: Tomcat 8.5.20 Configure...
Open browser
After launch: ✓ Default ... with JavaScript debugger
http://localhost:8080/
VM options:
On 'Update' action: Restart server ✓ Show dialog
JRE: Default (1.8 - project SDK)
Tomcat Server Settings
HTTP port: 8080 Deploy applications configured in Tomcat instance
HTTPs port: Preserve sessions across restarts and redeploys
JMX port: 1099
AJP port:

Before launch: Build, Activate tool window
Build
Show this page ✓ Activate tool window

Warning: No artifacts marked for deployment
Fix
Cancel Apply OK

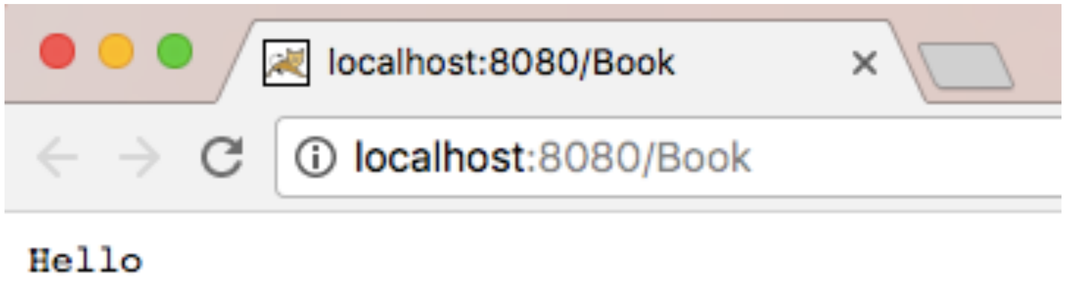
Server | Deployment | Logs | Code Coverage | Startup/Connection

Deploy at the server startup
RS:Web exploded
Application context: /

Before launch: Build, Build Artifacts, Activate tool window
Build
Build 'RS:Web exploded' artifact
Show this page ✓ Activate tool window

Deploy service with IDE

- Run → Run 'Tomcat'
- Access a page 'http://localhost:8080/Book' with your web browser

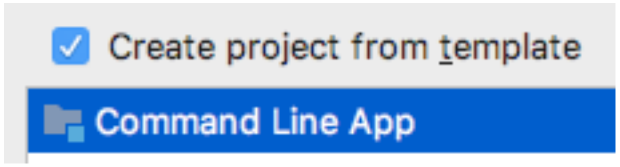


```
Hello
```

RESTful Client

Create default JAVA Project

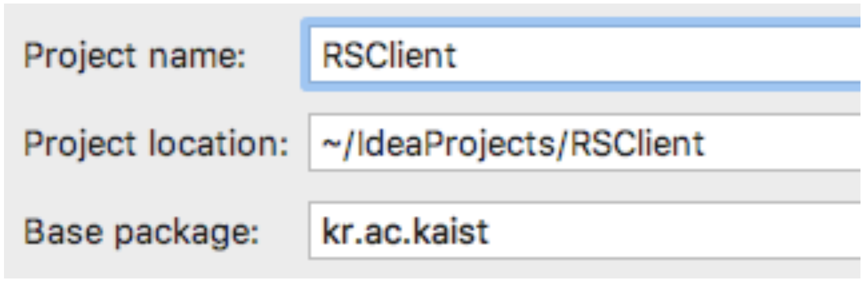
- File → New → Project → Next
- Create project with template



✓ Create project from template

Command Line App

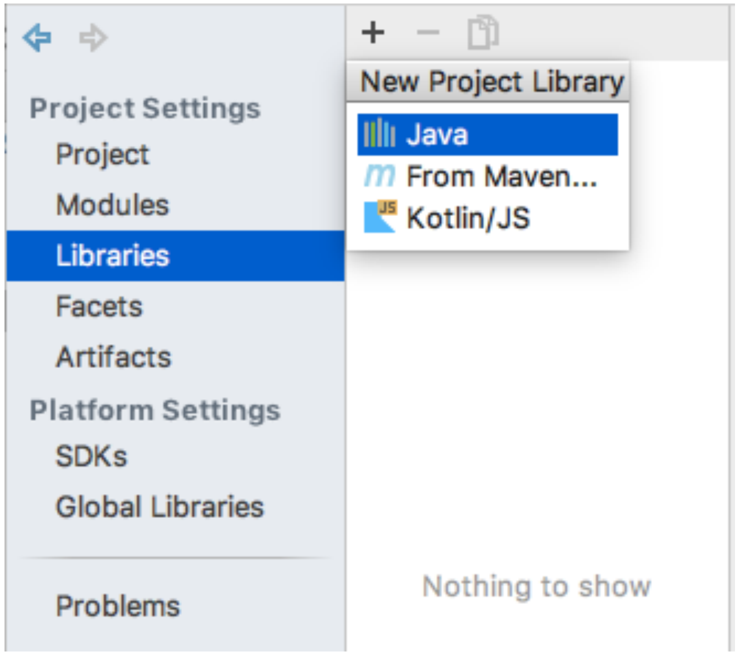
- Specify package name

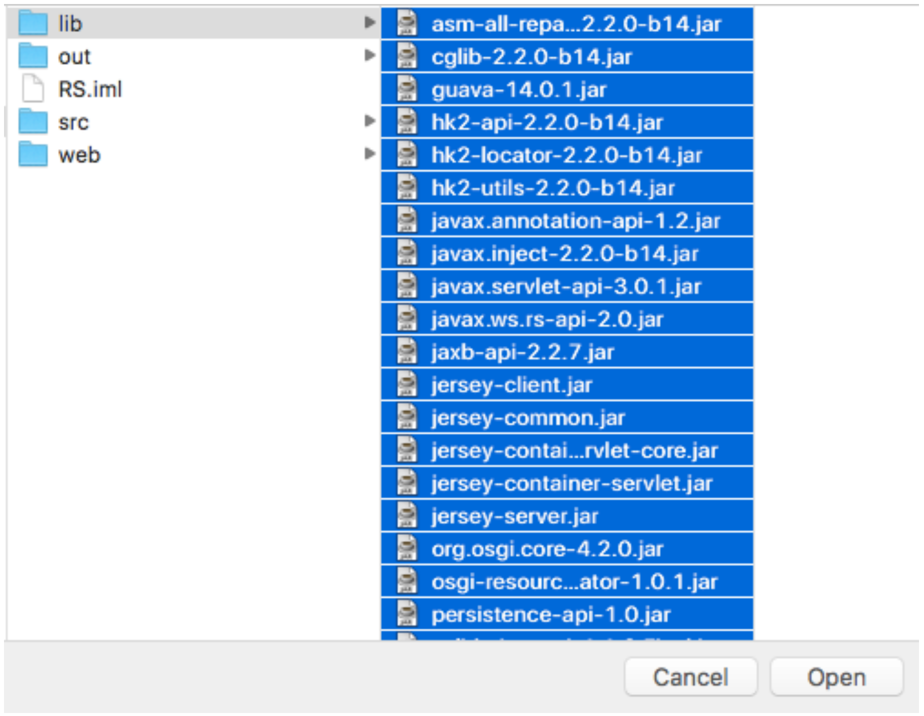


Project name:	RSClient
Project location:	~/IdeaProjects/RSClient
Base package:	kr.ac.kaist

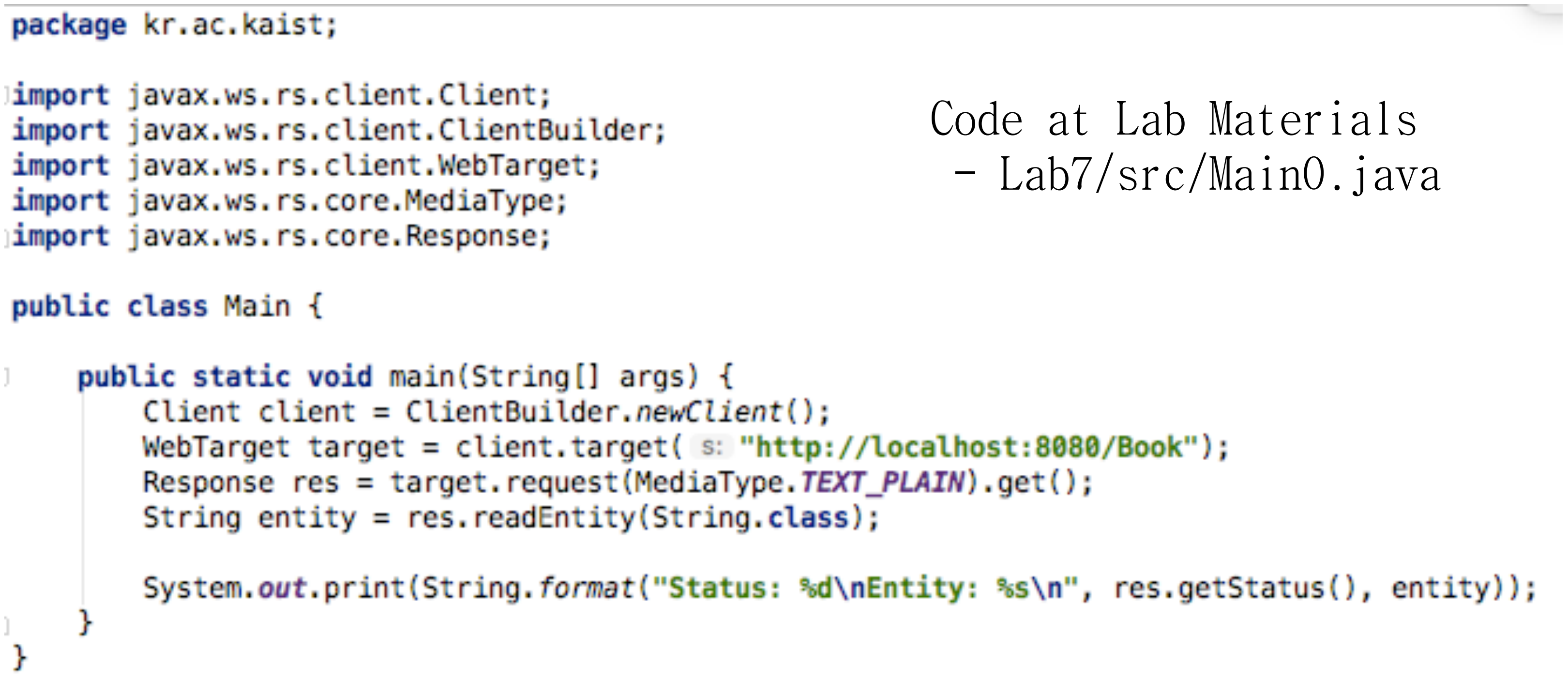
Library Import

- Add new JAVA Libraries from RESTful server



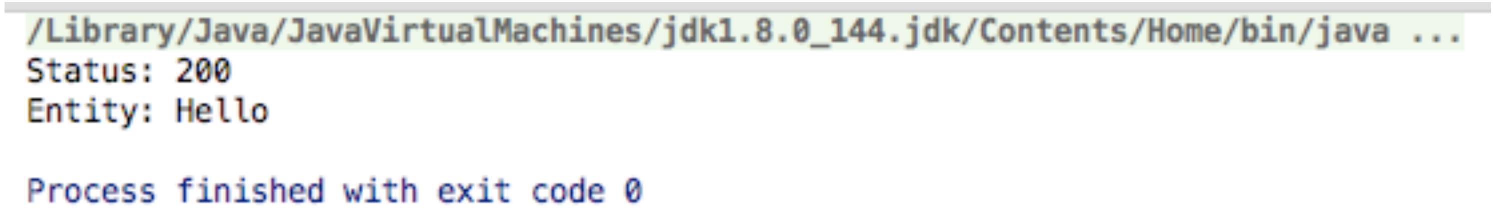


RESTful Client Implementation



```
package kr.ac.kaist;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

public class Main {

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target( s: "http://localhost:8080/Book");
        Response res = target.request(MediaType.TEXT_PLAIN).get();
        String entity = res.readEntity(String.class);

        System.out.print(String.format("Status: %d\nEntity: %s\n", res.getStatus(), entity));
    }
}
```

Code at Lab Materials
- Lab7/src/Main0.java

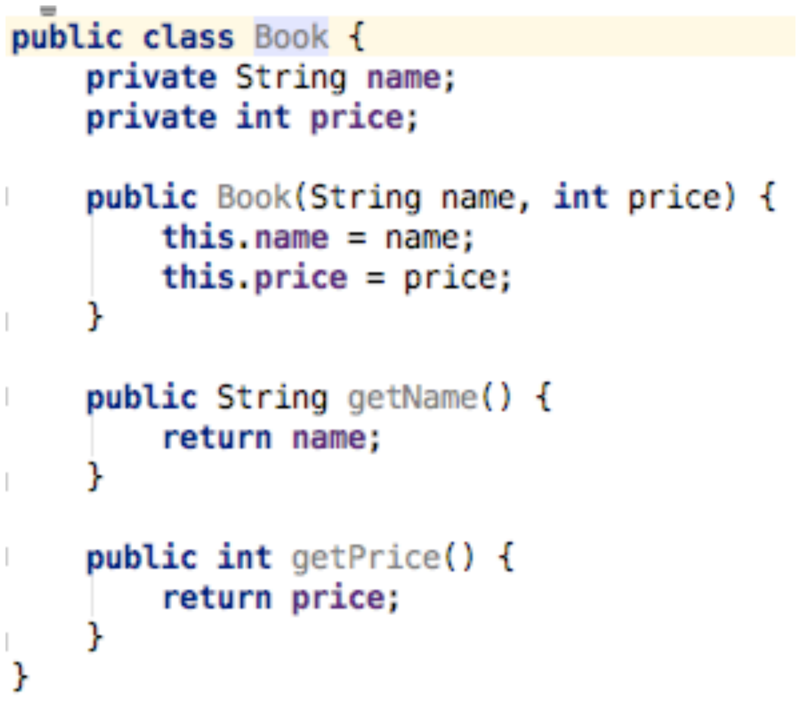


```
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java ...
Status: 200
Entity: Hello

Process finished with exit code 0
```

Object Representation with JSON

Book Class Example

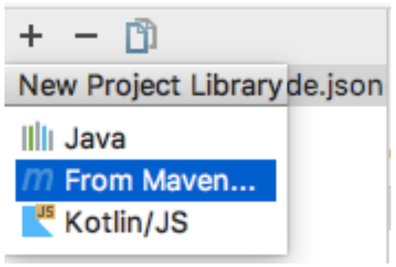
- Simple Book Class

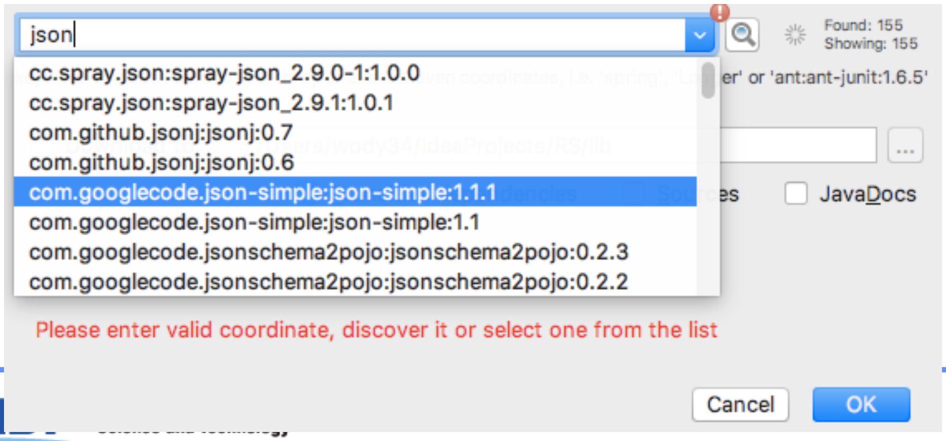


```
public class Book {
    private String name;
    private int price;

    public Book(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }
}
```

Add JSON Library

- Download 'json-simple' library from maven repository and add it to the artifacts

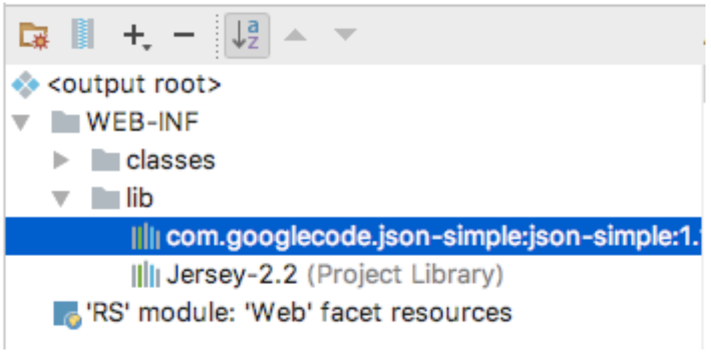




```
json
cc.spray.json:spray-json_2.9.0-1:1.0.0
cc.spray.json:spray-json_2.9.1:1.0.1
com.github.jsonj:jsonj:0.7
com.github.jsonj:jsonj:0.6
com.googlecode.json-simple:json-simple:1.1.1
com.googlecode.json-simple:json-simple:1.1
com.googlecode.jsonschema2pojo:jsonschema2pojo:0.2.3
com.googlecode.jsonschema2pojo:jsonschema2pojo:0.2.2
```

Found: 155
Showing: 155

Please enter valid coordinate, discover it or select one from the list

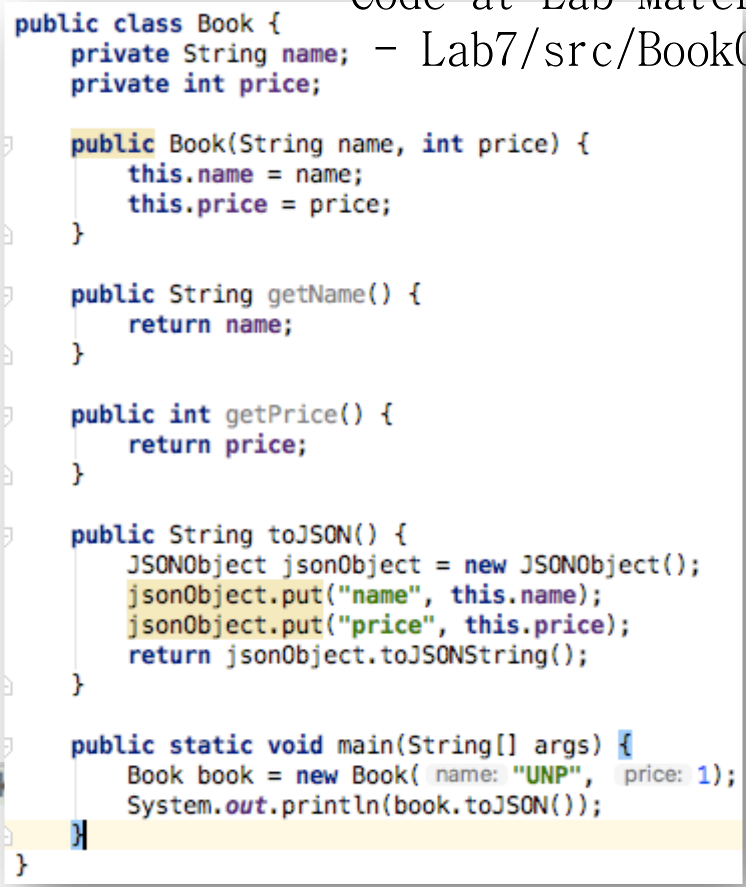


```
<output root>
WEB-INF
  classes
  lib
    com.googlecode.json-simple:json-simple:1.
    Jersey-2.2 (Project Library)
'RS' module: 'Web' facet resources
```

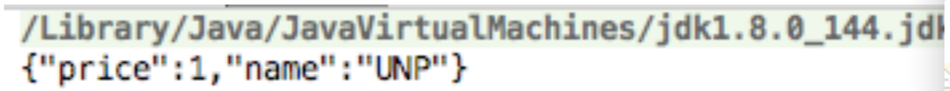
JSON Enabling

- New representation of book object
  - Interoperable
  - Text-based
  - Descriptive

Code at Lab Materials
– Lab7/src/Book0.java



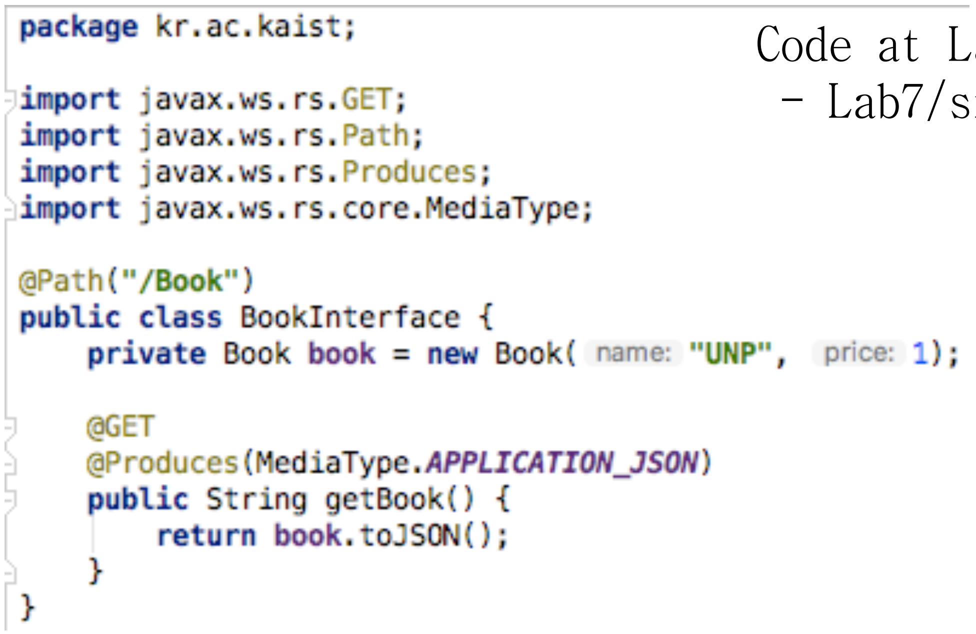
```
public class Book {
    private String name;
    private int price;

    public Book(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }

    public String toJSON() {
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("name", this.name);
        jsonObject.put("price", this.price);
        return jsonObject.toJSONString();
    }

    public static void main(String[] args) {
        Book book = new Book( name: "UNP",  price: 1);
        System.out.println(book.toJSON());
    }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk
{"price":1,"name":"UNP"}
```

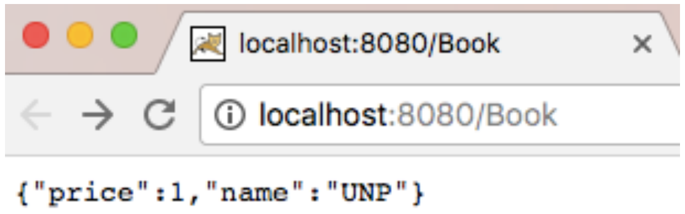
JSON in RESTful Server

- We can get Book Object through RESTful Service



```java
package kr.ac.kaist;

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

@Path("/Book")
public class BookInterface {
    private Book book = new Book( name: "UNP",  price: 1);

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public String getBook() {
        return book.toJSON();
    }
}
```

Code at Lab Materials
 - Lab7/src/BookInterface1.java

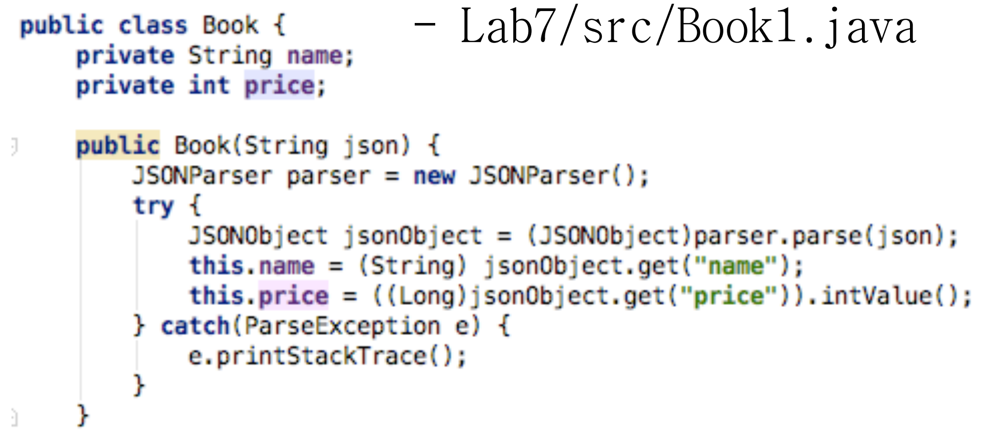


```
{"price":1,"name":"UNP"}
```

JSON in RESTful Client

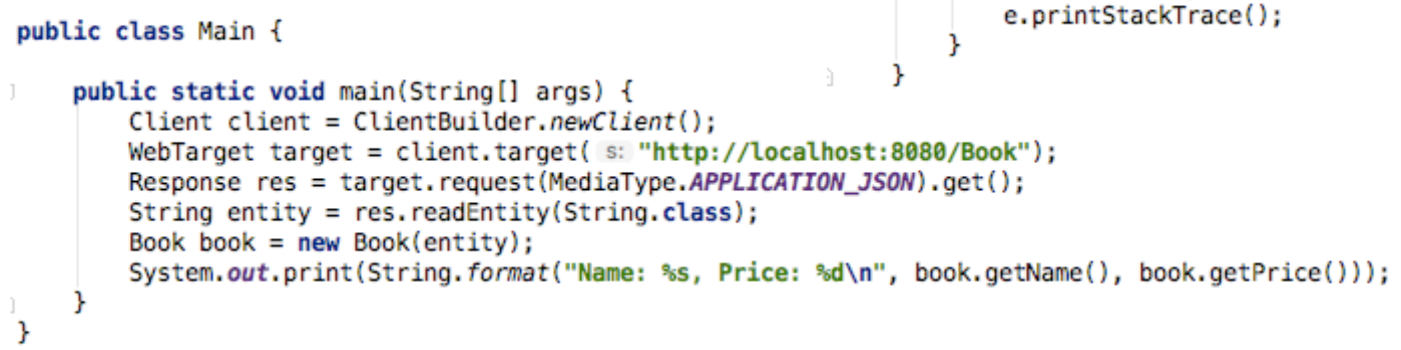
- Book object is de-serialized using JSON Parser

Code at Lab Materials
 – Lab7/src/Book1.java



```java
public class Book {
    private String name;
    private int price;

    public Book(String json) {
        JSONParser parser = new JSONParser();
        try {
            JSONObject jsonObject = (JSONObject)parser.parse(json);
            this.name = (String) jsonObject.get("name");
            this.price = ((Long)jsonObject.get("price")).intValue();
        } catch(ParseException e) {
            e.printStackTrace();
        }
    }
```

Code at Lab Materials
 – Lab7/src/Main1.java



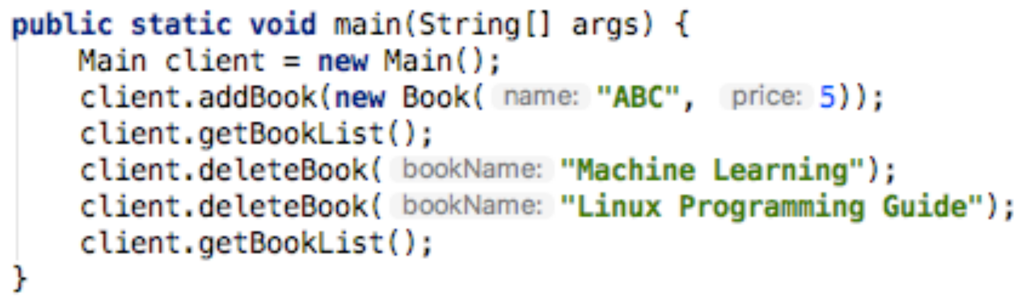
```java
public class Main {

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target( s: "http://localhost:8080/Book");
        Response res = target.request(MediaType.APPLICATION_JSON).get();
        String entity = res.readEntity(String.class);
        Book book = new Book(entity);
        System.out.print(String.format("Name: %s, Price: %d\n", book.getName(), book.getPrice()));
    }
}
```

Add functionalities

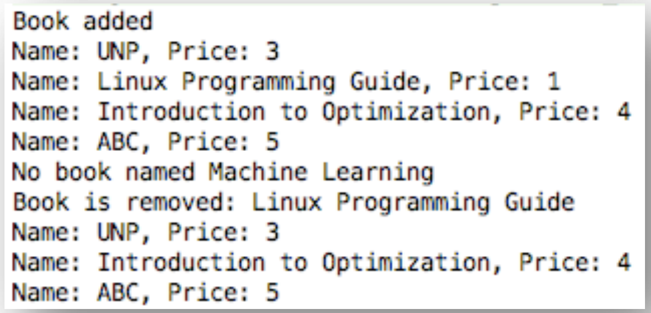
Add functionalities

- GET / POST / DELETE method is added
  - Create / Retrieve / Delete function is available

Code at Lab Materials
- Lab7/src/Book2.java
- Lab7/src/Main2.java
- Lab7/src/BookInterface2.java



```
public static void main(String[] args) {
    Main client = new Main();
    client.addBook(new Book( name: "ABC",  price: 5));
    client.getBookList();
    client.deleteBook( bookName: "Machine Learning");
    client.deleteBook( bookName: "Linux Programming Guide");
    client.getBookList();
}
```



```
Book added
Name: UNP, Price: 3
Name: Linux Programming Guide, Price: 1
Name: Introduction to Optimization, Price: 4
Name: ABC, Price: 5
No book named Machine Learning
Book is removed: Linux Programming Guide
Name: UNP, Price: 3
Name: Introduction to Optimization, Price: 4
Name: ABC, Price: 5
```