Bachelor's Degree in Techniques for Software Development

Network Applications and Internet
Activity: Practical exercise 2: Create your own online app!

In this activity, we propose you to create applications that communicate different network devices, by programming in Java some of the communication paradigms most used nowadays.

In the first two parts of this practical exercise, we will focus on socket programming, both connection-oriented and connectionless. In the first part, you will implement a client program, which communicates with UDP servers. In the second part, you will implement an HTTP client that will connect to a given web server.

In parts 3 and 4 of this practical exercise, we will create a REST API to make some simple operations with words and we will interchange data using different coding alternatives, including JSON.

Each part of the practice has some initial code that you should use as a template. That is, you have to add programming instructions to make it work as this document describes. You cannot delete any template line of code, as they are used in the automatic correction of the tool in DSLab. You are not allowed to copy code from any source (Internet, books, etc.).

You will see that there are some instructions regarding log, where the executed instructions are registered. If you want to use them for your own control, you can find a brief tutorial in the following address: https://sd.uoc.edu/dslab/help/

The provided projects are prepared to work in Eclipse, a programming and simulation environment. Moreover, in parts 3 and 4, you will use an integrated server, Jetty, that you do not have to install, as it is included in the projects provided. You can find a simple tutorial on how to import Eclipse projects at:

- Annex **Eclipse tutorial (parts 1 and 2)**
- Annex **Installing REST service in Eclipse (parts 3 and 4)**

Other IDE's have a similar functionality. If you use a different IDE, you should configure the classpath of the libraries used. You can also compile and test from the command line of your specific operating system.

As main support material for implementing this practice, you can use the following modules, which include both theory and practice of Java programming. You can find them at:

- **Teaching module. Communication Mechanisms**
- **Teaching module. Java Socket Programming**
- **Teaching module. REpresentational State Transfer (REST)**

Qualification

This practice has four parts. Parts 1 and 2 are independent. Parts 3 and 4 are related between them (4 is an evolution of 3, but both should be delivered independently). It is not required to implement them sequentially (for example, you can do only parts 1 and 3). Qualification is calculated according to the parts delivered:

1 part delivery:	*Maximum qualification C-*
2 parts delivery:	*Maximum qualification C+*
3 parts delivery:	*Maximum qualification B*
4 parts delivery:	*Maximum qualification A*

For each part of the practice, you have to deliver the source code of the modified files, client and server, as explained in this document. Deliver your responses in the subject's classroom. Use some compression program to make the delivery in only one file. Name the file:

Practice2_Surname_Name.[zip, tar, ...]

You can use any server to make the test, local or remote. Nevertheless, **a practice will only be considered correct if it has been run with a successful result in DSLab** https://sd.uoc.edu/dslab/

To run your practice in the automatic correction environment DSLab follow these steps:

- Create a project (menu *Projects> New Project*).
- Select the task that corresponds to the part of the assignment you want to correct: *Assignment i task.*
- Upload source code from this part, clicking in the project you have just created. Take into account that in the lower part of the screen, the files you have to upload appear (*Required files)*.
- Create an automatic correction (menu *Submissions> New Submission*) of the project created in the first step. Execution result can be checked at the *Submissions* menu.

First part: UDP socket programming

In this first part, we will work with connectionless sockets by means of Java programming. You have as support material **Java socket programming** and the annex **Eclipse tutorial**.

In particular, we give you the `xai-sockets-udp--2024P-alumnes.zip` archive with the following project files:

- `UDPclientMain.java` (client main code)
- `RemoteMapUDPclient.java` (client UDP to be modified according to this document)
- `UDPservidorMain.java` (server main code)
- `RemoteMapUDPservidor.java` (server UDP to be modified according to this document)
- `Key.java` (keys management)
- Folder `lib` (libraries)

In this part, you have to program both a client and a server, which will communicate through connectionless sockets. The main objective is that the client builds a unique map from map data received from servers. The data type *Map* stores an object collection, typically relating a key (*Key*) with a value.

First, you should execute class `UDPservidorMain`, passing as a parameter the port number. Servers configured in this practice usually work on ports 5836 and 5838. In this class, server map, where data pairs <key, value> are stored, is created and initialized, and, specifically in this practice, they will be <string, string> pairs with the following test data:

- server 1 port 5836 <k1, R_k1>, <k3, R_k3>, <k4, R_k4>
- server 2 port 5838 <k2, R_k2>

After initializing the maps, servers create an instance of class `RemoteMapUDPservidor.` This class is the one you have to modify. It will be in charge of attending remote client requests, responding with the corresponding value to the specific requested key. For example, if a client asks server 1 the value associated to *k1* key, the server will send *R_k1*. You can add more servers or even more data to the map; or leave just one to make initial tests. Final tests for automatic correction of the practice at DSLab will be done with random maps for each student.

Clients are started executing class `UDPclientMain`. In the first place, a map is built in the client with the keys to be asked for and servers that have them. In our example, client's map contains:

- localhost port 5836 <k1>, <k3>, <k4>
- localhost port 5838 <k2>

Afterwards, an instance of class `RemoteMapUDPclient` is created, and the *getMap* method is invoked. This method goes through the initial key map detailed in the example and asks for the key (method *get*) associated with each entry to the selected server. For example, the client will connect to server *localhost:5836*, and it will ask for the *k1* key. Server will respond with *R_k1* and so on. As it receives key values, the client will build a new map, similar to the servers' one, but with the union of data from all of them. So, following our example, client final map should contain:

- <k1, R_k1>, <k3, R_k3>, <k4, R_k4>, <k2, R_k2>

In clients' case, you should modify class `RemoteMapUDPclient`, which is in charge of making the remote requests for the map's missing values. Specifically, you just have to modify the *get* method, which receives as a parameters server's address and port, as well as the key to be requested. You will have to create a UDP socket and obtain the value associated with the key, asking the associated server. Server's response value is a string returned by *get* method.

Finally, remember closing all sockets created to free system resources.

Second part: TCP sockets programming

In this second part, we will work with connection-oriented sockets, through Java programming. As support material, you have **Java Sockets Programming** and the annex **Eclipse tutorial**.

In particular, we give you the `xai-sockets-tcp—2024P-alumnes.zip` archive with the following project files:

- `TCPclientMain.java` (client main code)
- `HTTPclient.java` (HTTP client to be modified according to this document)
- `HTTPrequestResponse.java` (class to deal with request and response data)
- Folder `lib` (libraries)

From code from file `HTTPclient.java` you have to program an **HTTP protocol mini-client**. We will focus on the `GET` request, which returns the response code, the HTTP headers and the resource itself. In order to make the request as complete as possible, include the `Accept` header in the request.

First, you will execute class `TCPclientMain.java`, where you have to write the web server you want to connect to (address and port). Next, an instance of class `HTTPclient` is created, which will be in charge of managing remote communication between both hosts through TCP sockets. You have to modify this class.

Then, the client will connect to the server defined. When connection is accepted, the client will build the `GET /xai/xai.html HTTP/1.0` request, against server `labxarxes.techlab.uoc.edu`, show it on the screen and send it to the server. When the server response arrives, it has to be printed on the screen. Finally, remember closing sockets to free system resources.

You should take into account that, after each request and response HTTP line, including headers, HTTP standard indicates that characters CRLF (`Carriage return` and `Line Feed \r\n`) must go. If you do not follow this recommendation, DSLab correction will fail even if the rest of the data is correct.

For the local execution, connection is directly done against the final web server. However, DSLab uses a proxy to be able to make the correction. You should consider this when reviewing logs at DSLab.

Third part: REST basic programming

The starting point of a distributed application specification is the description of the resources that can be invoked remotely. In this practice, we ask you to invoke these remote resources with a **REST** based web service (REpresentational State Transfer). Remember that you have the **REpresentational State Transfer (REST)** teaching module as support.

REST does not have any limitation in request and response format (they could be text, JSON, XML, etc.). So, to define a REST service it is needed to describe an API indicating how the resource can be requested, including data that may contain the request and giving an example of response.

We provide an Eclipse project with the skeleton of a service that works over a Jetty server. You do not have to install the Jetty server; it is included inside the project. What you have to do is to import the project we provide into Eclipse. In the annex **Installing a REST service in Eclipse**, you have installation detailed instructions. In this annex, the code from file `xai-rest-base.zip` is used, but it is just a simple REST service example, not all functionality is implemented. The service you have to complete is in file `xai-rest-paraules--2024P-alumnes.zip`.

In this part of the practice, it is required to implement a service that performs some operations with words, as explained next.

Files to be modified are (they have operations from parts 3 and 4 of the practice):

```
RestServerAPI.java
RESTclient.java
```

Access URL of the operation is:

Word length: `http://localhost:7070/words/length/{word}`

This request gets a word as parameter and returns its length in numeric format, but it should return a "text/plain" media type.

You also have to implement a client program that invokes the service operation and shows the result. You can add the required classes in the server project or create a new project.

Fourth part: REST advanced programming

In this section, you have to continue the implementation of the REST client and server. Specifically, you have to implement two more operations to service from section 3 regarding words: split and conversion to lowercase. The parameters and responses for each operation are detailed next. Remember that you have the **REpresentational State Transfer (REST)** teaching module as support.

Each operation has an access URL:

Split based on a regular expression (regex): `http://localhost:7070/words/split/{word}/{regex}`

Lowercase conversion: `http://localhost:7070/words/toLowerCase/{word}`

The parameter indicates the word over which the operation has to be done. The regex parameter indicates the expression to be used to split the word.

The response depends on the operation, as described next:
- Split should return an “application/json" media type where an ArrayList Java class is sent.
- Lowercase conversion has to return an “application/json" media type where the Capitalized class is sent, which we give you partially implemented. Apart from the lowercase conversion, it is also returned a Boolean indicating if the word has been modified or not.

Example result for split operation:
```
["Eph","m","ris"]
```

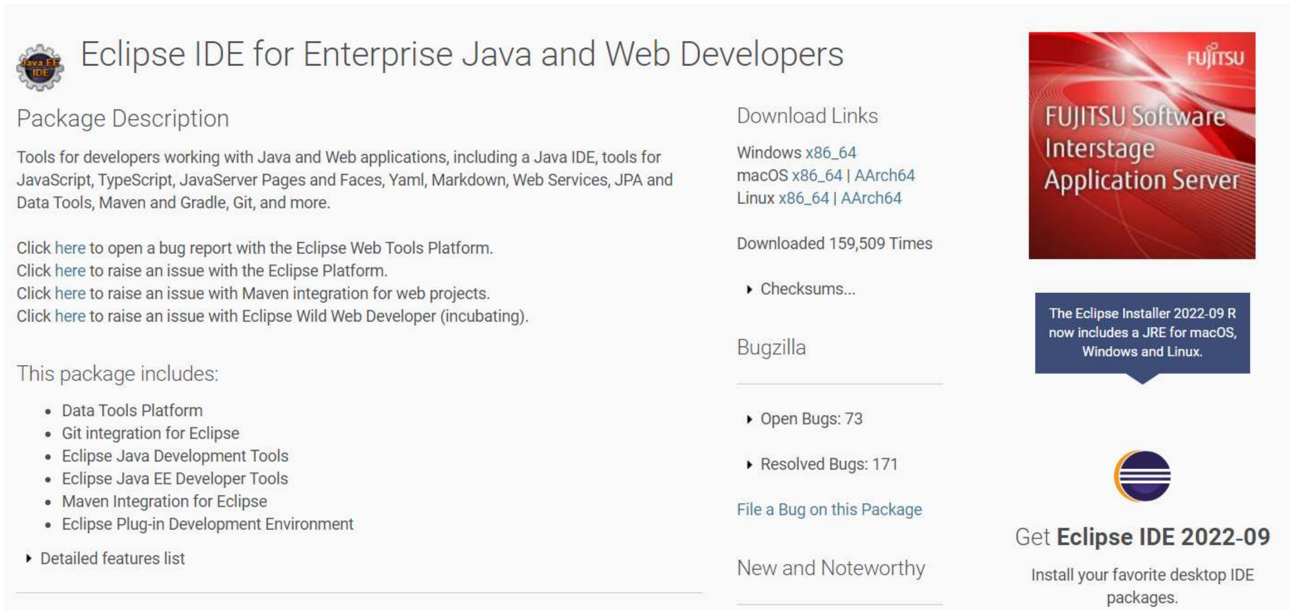
Example result for lowercase conversión in JSON format:
```
{"word":"ephemeris","modified":true}
```

You also have to implement the calls to these new operations from the client program and show the result.

Annex: Eclipse tutorial (Parts 1 and 2)

This annex briefly describes how to import the project we provide you both for parts 1 and 2 of this practice, using Eclipse IDE. You may find some slight variation depending on the version you use.

You need to install Eclipse programming environment. You can download it from `https://www.eclipse.org/downloads/packages/release/2022-12/r/eclipse-ide-enterprise-java-and-web-developers`. If it is not the last version, it should redirect you to the latest version. Installation is very intuitive.



Eclipse IDE for Enterprise Java and Web Developers

Package Description

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

Click here to open a bug report with the Eclipse Web Tools Platform.
Click here to raise an issue with the Eclipse Platform.
Click here to raise an issue with Maven integration for web projects.
Click here to raise an issue with Eclipse Wild Web Developer (incubating).

This package includes:

- Data Tools Platform
- Git integration for Eclipse
- Eclipse Java Development Tools
- Eclipse Java EE Developer Tools
- Maven Integration for Eclipse
- Eclipse Plug-in Development Environment

Detailed features list

Download Links

Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64

Downloaded 159,509 Times

Checksums...

Bugzilla

Open Bugs: 73

Resolved Bugs: 171

File a Bug on this Package

New and Noteworthy

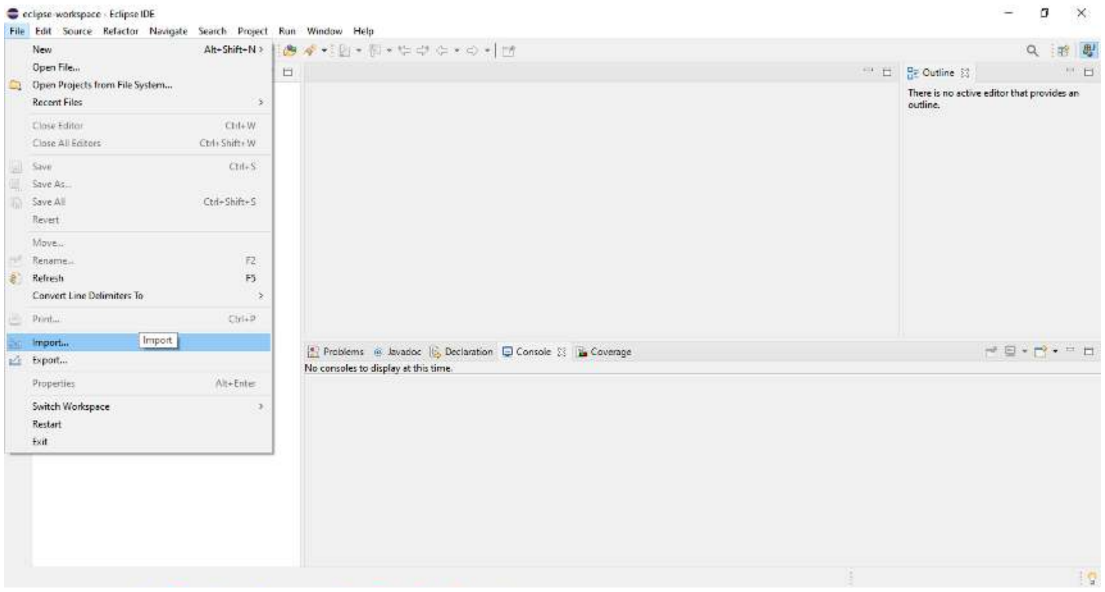
FUJITSU Software Interstage Application Server

The Eclipse Installer 2022-09 R now includes a JRE for macOS, Windows and Linux.

Get Eclipse IDE 2022-09

Install your favorite desktop IDE packages.

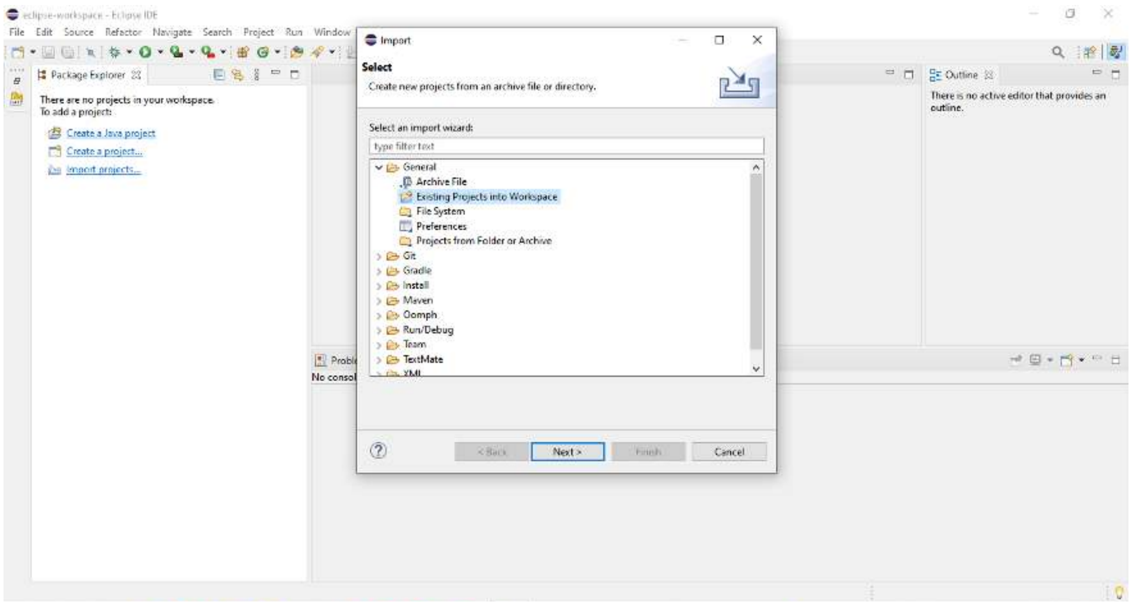
You can directly import the compressed files we provide you, importing the project directly from the zip file, as it will properly import libraries, which have relative routes. Go to the Eclipse menu *File > Import.*

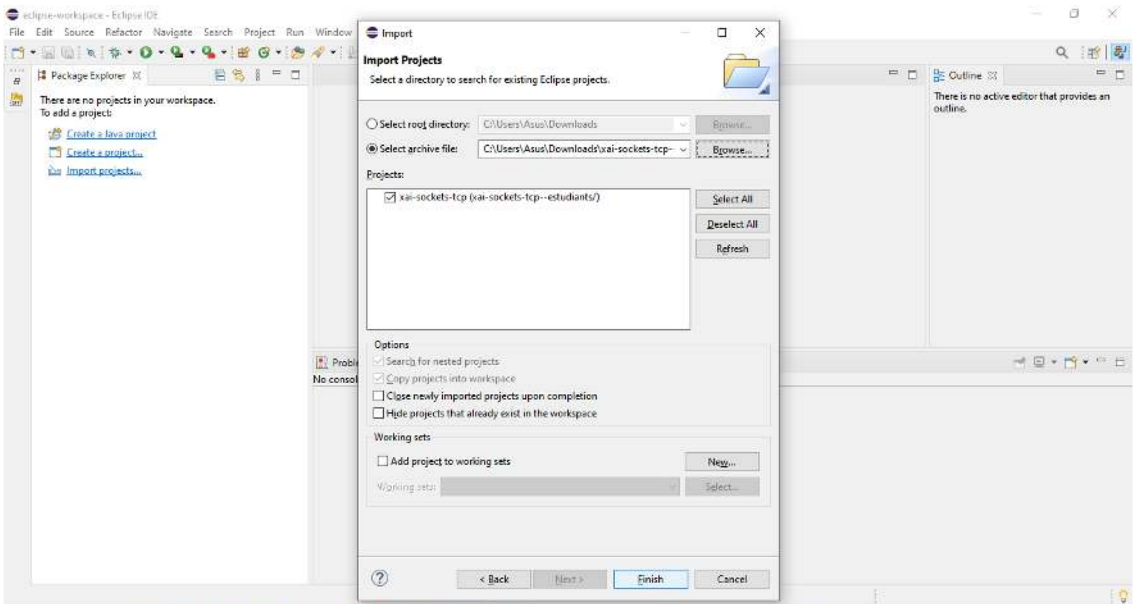

eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Import
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Outline
Problems Javadoc Declaration Console Coverage

There is no active editor that provides an outline.

No consoles to display at this time.

An emerging window will appear where you should select the compressed file we have provided, going to the option *Existing projects into workspace* > *Next* > *Select Archive File,* as it can be seen in the two following screenshots. Once selected, you can click the *Finish* button.


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window
Package Explorer
There are no projects in your workspace.
To add a project:
Create a Java project
Create a project...
Import projects...
Import
Select
Create new projects from an archive file or directory.
Select an import wizard:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
Projects from Folder or Archive
Git
Gradle
Install
Maven
Oomph
Run/Debug
Team
TextMate
XML
< Back
Next >
Finish
Cancel
Outline
There is no active editor that provides an outline.


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window
Package Explorer
There are no projects in your workspace.
To add a project:
Create a Java project
Create a project...
Import projects...
Import
Import Projects
Select a directory to search for existing Eclipse projects.
Browse...
Select All
Deselect All
Refresh
New...
Select...
< Back
Next >
Finish
Cancel
Outline
There is no active editor that provides an outline.

Select root directory: C:\Users\Asus\Downloads
Select archive file: C:\Users\Asus\Downloads\xai-sockets-tcp-

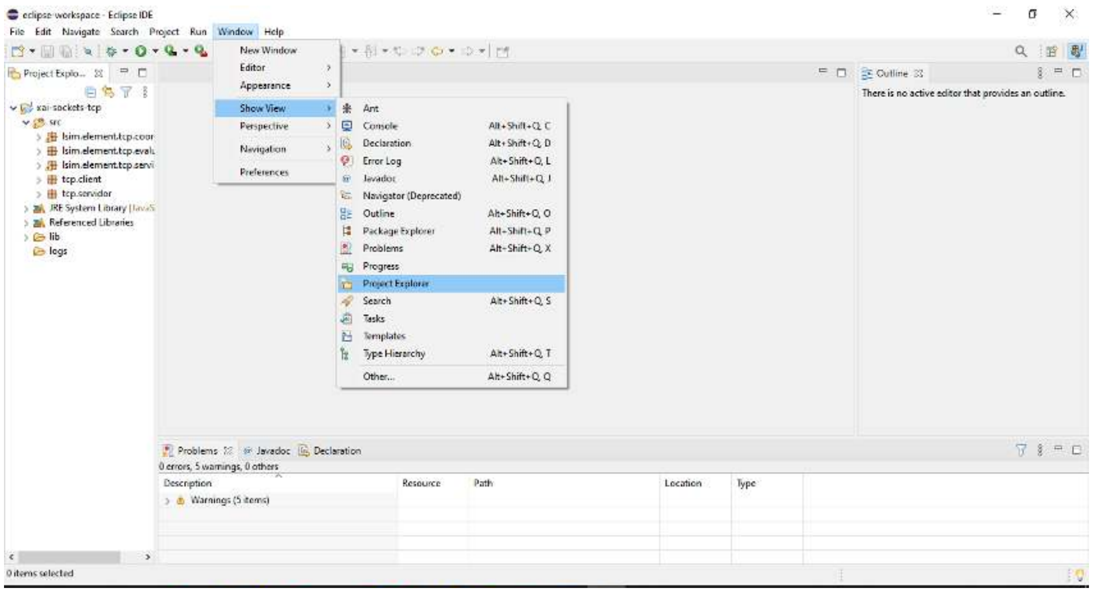
Projects:
- [✓] xai-sockets-tcp (xai-sockets-tcp--estudiants/)

Options
- [✓] Search for nested projects
- [✓] Copy projects into workspace
- [ ] Close newly imported projects upon completion
- [ ] Hide projects that already exist in the workspace

Working sets
- [ ] Add project to working sets

Working sets:

Once accepted by clicking *Finish,* you can see the project using the tree view, at the left side of Eclipse. If this view is not opened, you can find it going to the menu *Window* > *Show view* > *Explore project*. The name of the options may vary depending on the installation language.

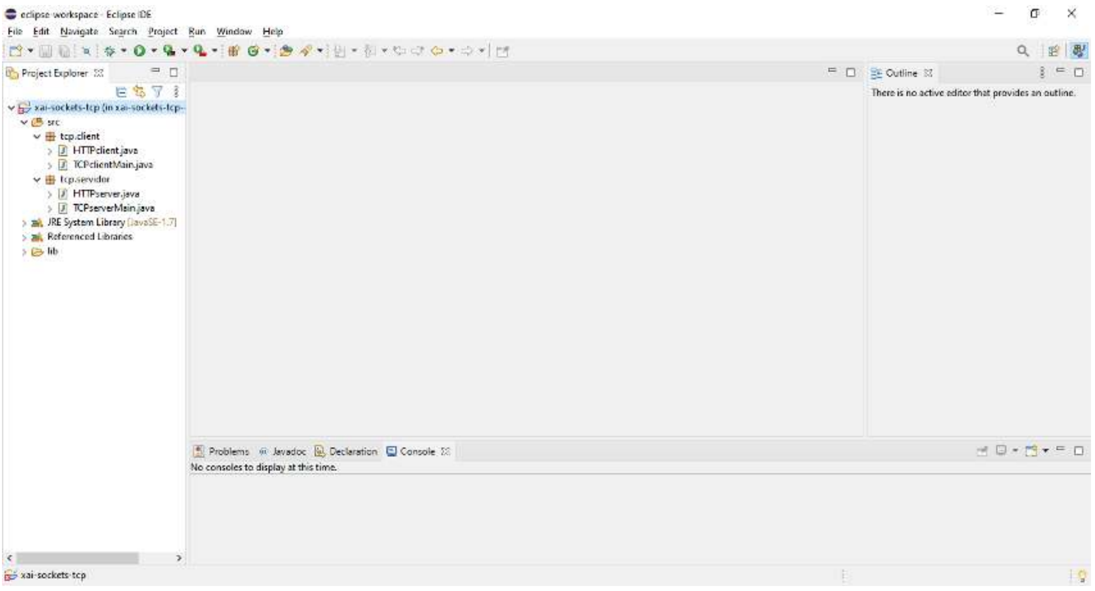


There is no active editor that provides an outline.

0 errors, 5 warnings, 0 others

0 items selected

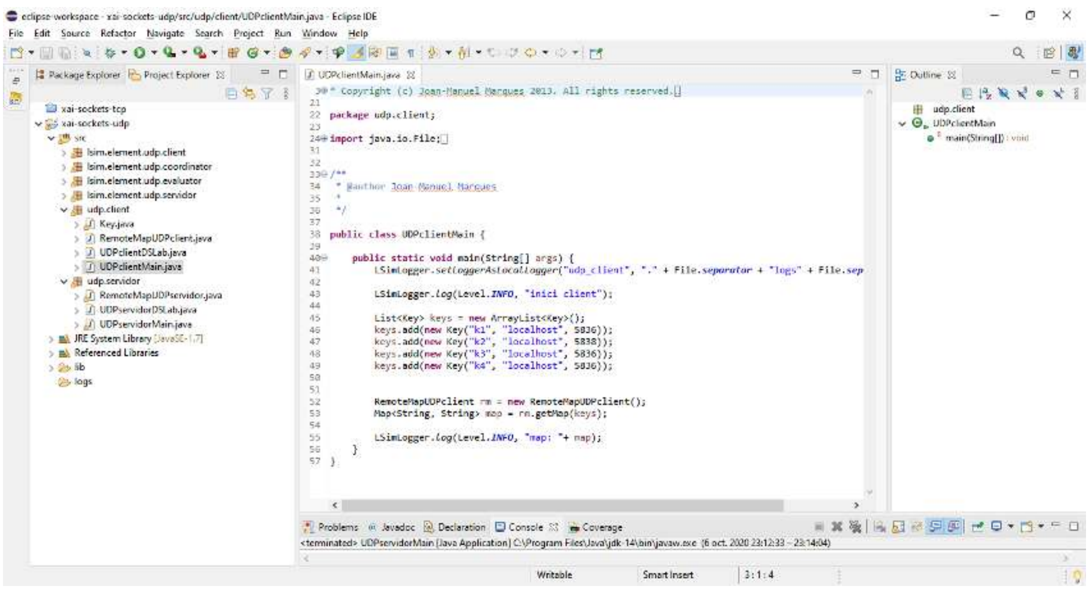
Once you can see the project tree, you can browse between the different classes and libraries that form part of it. In your case, you may find different files from the ones shown; this is just a reference.



There is no active editor that provides an outline.

No consoles to display at this time.

At any moment, to see the source code of clients or servers, you have to click over the class in the project browser at Eclipse left side.

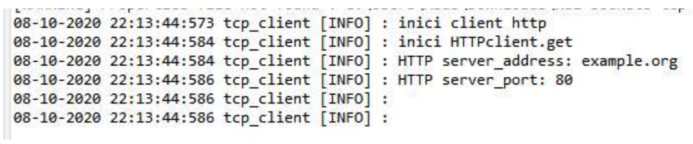


```java
* Copyright (c) Joan-Manuel Marques 2013. All rights reserved.

package udp.client;

import java.io.File;


/**
 * @author Joan-Manuel Marques
 *
 */

public class UDPclientMain {

    public static void main(String[] args) {
        LSimLogger.setLoggerAsLocalLogger("udp_client", "." + File.separator + "logs" + File.sep

        LSimLogger.log(Level.INFO, "inici client");

        List<Key> keys = new ArrayList<Key>();
        keys.add(new Key("k1", "localhost", 5836));
        keys.add(new Key("k2", "localhost", 5838));
        keys.add(new Key("k3", "localhost", 5836));
        keys.add(new Key("k4", "localhost", 5836));


        RemoteMapUDPclient rm = new RemoteMapUDPclient();
        Map<String, String> map = rm.getMap(keys);

        LSimLogger.log(Level.INFO, "map: "+ map);
    }
}
```

```
<terminated> UDPservidorMain [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (6 oct. 2020 23:12:33 – 23:14:04)
```

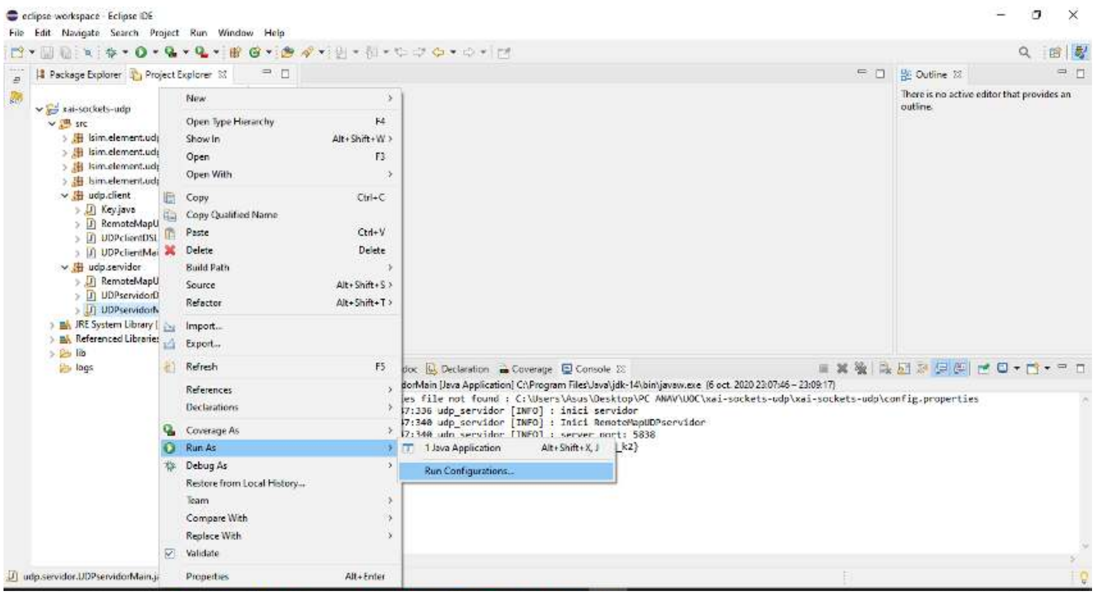
To execute the project, you must select the corresponding class, for example, `TCPclientMain` and click over the green play icon ▶, at the icon menu bar. You can also do it by right-clicking over the class containing the *main* method and selecting the corresponding option on the contextual menu.

The initial result shown by the Eclipse console with code from part 2 should be something similar to the following image:



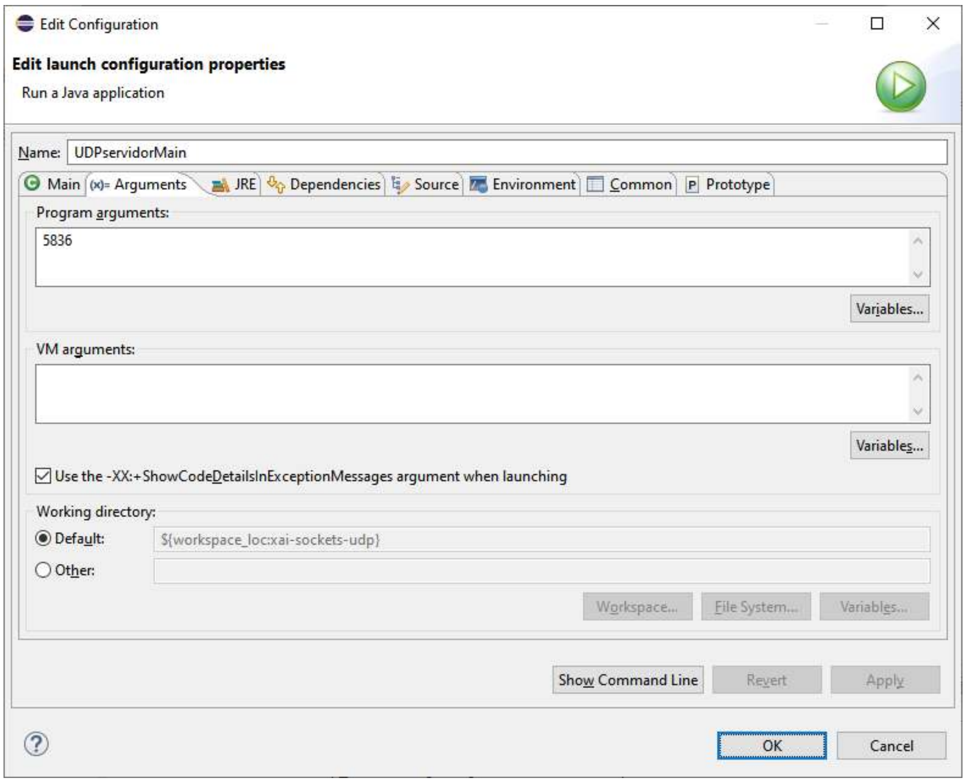
```
08-10-2020 22:13:44:573 tcp_client [INFO] : inici client http
08-10-2020 22:13:44:584 tcp_client [INFO] : inici HTTPclient.get
08-10-2020 22:13:44:584 tcp_client [INFO] : HTTP server_address: example.org
08-10-2020 22:13:44:586 tcp_client [INFO] : HTTP server_port: 80
08-10-2020 22:13:44:586 tcp_client [INFO] :
08-10-2020 22:13:44:586 tcp_client [INFO] :
```

For the first part of the practice, where the server needs to be executed with the port number passed as parameter, you can click with the right button over the class `UDPservidorMain` at the project explorer view (tree at the left side of the screen) and go to the configuration inside the option *Run*.



```
dorMain [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (6 oct. 2020 23:07:46 – 23:09:17)
ies file not found : C:\Users\Asus\Desktop\PC ANAV\UOC\xai-sockets-udp\xai-sockets-udp\config.properties
47:336 udp_servidor [INFO] : inici servidor
47:340 udp_servidor [INFO] : Inici RemoteMapUDPservidor
47:340 udp_servidor [INFO] : server port: 5838
```

As you need to execute two servers, you need to repeat the execution operation twice. A screen will appear where you have to select the server and write port 5836 for the first execution and 5838 at the second. Finally, you have to accept and close the emerging windows.



Edit Configuration

Edit launch configuration properties

Run a Java application

Name: UDPservidorMain

Main | Arguments | JRE | Dependencies | Source | Environment | Common | Prototype

Program arguments:

```
5836
```

VM arguments:

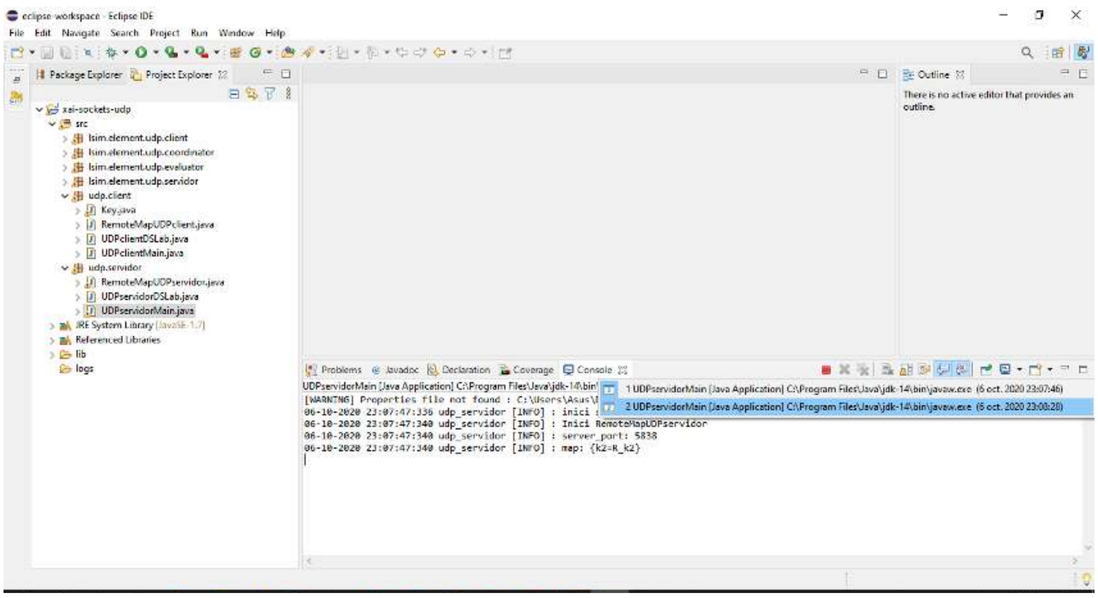
☑ Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching

Working directory:

● Default: ${workspace_loc:xai-sockets-udp}

○ Other:

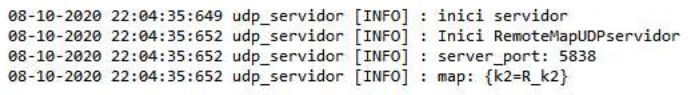
If you want to see different consoles, that is, the screen where the messages shown by any server or the client appear during its execution, you have to use the button for changing console at the right lower part on Eclipse:



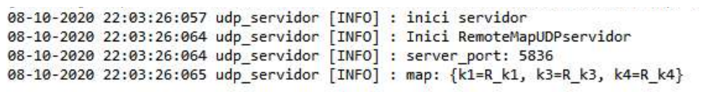
```
UDPservidorMain [Java Application] C:\Program Files\Java\jdk-14\bin\
[WARNING] Properties file not found : C:\Users\Asus\
06-10-2020 23:07:47:336 udp_servidor [INFO] : inici s
06-10-2020 23:07:47:340 udp_servidor [INFO] : Inici RemoteMapUDPservidor
06-10-2020 23:07:47:340 udp_servidor [INFO] : server_port: 5838
06-10-2020 23:07:47:340 udp_servidor [INFO] : map: {k2=R_k2}
```

```
1 UDPservidorMain [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (6 oct. 2020 23:07:46)
2 UDPservidorMain [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (6 oct. 2020 23:00:20)
```

The initial result shown by the console for the source code from part 1 for the servers should be something similar to the following image:



```
08-10-2020 22:04:35:649 udp_servidor [INFO] : inici servidor
08-10-2020 22:04:35:652 udp_servidor [INFO] : Inici RemoteMapUDPservidor
08-10-2020 22:04:35:652 udp_servidor [INFO] : server_port: 5838
08-10-2020 22:04:35:652 udp_servidor [INFO] : map: {k2=R_k2}
```



```
08-10-2020 22:03:26:057 udp_servidor [INFO] : inici servidor
08-10-2020 22:03:26:064 udp_servidor [INFO] : Inici RemoteMapUDPservidor
08-10-2020 22:03:26:064 udp_servidor [INFO] : server_port: 5836
08-10-2020 22:03:26:065 udp_servidor [INFO] : map: {k1=R_k1, k3=R_k3, k4=R_k4}
```

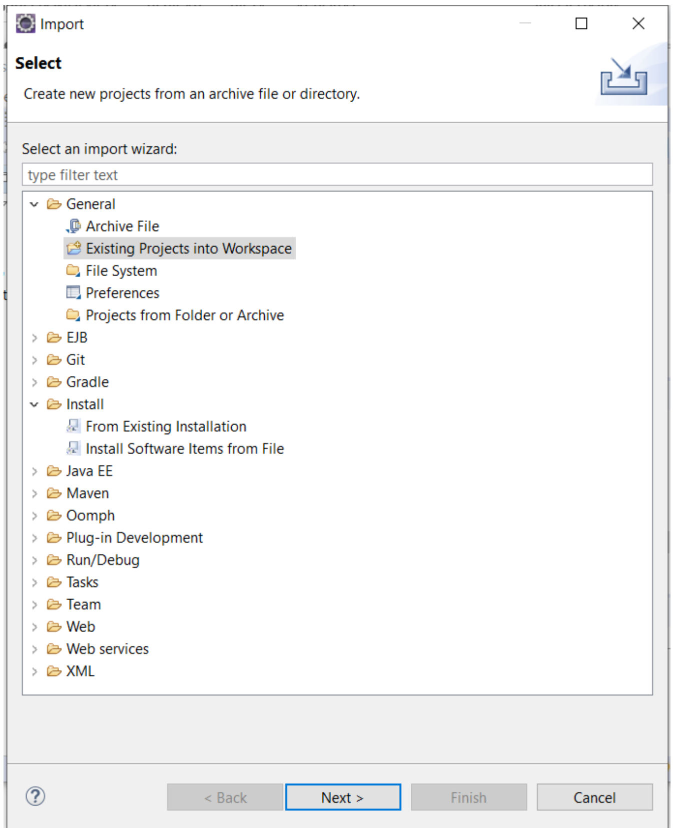
Annex: Installing a REST service in Eclipse (Parts 3 and 4)

In this annex, we explain how to install, execute and test a base REST service we provide you in this practice. It is a simple calculator that allows additions, subtractions and divisions of two integer numbers. With this service as a base, you have to implement what is proposed in this document.

You need to install Eclipse programming environment. You can download it from `https://www.eclipse.org/downloads/packages/release/2022-12/r/eclipse-ide-enterprise-java-and-web-developers`. If this is not the latest version, it will show you how to get it. We will also use the Jetty server (`https://www.eclipse.org/jetty/`), but it is already included in the project, so you do not need to download anything.

Once Eclipse is installed, you need to follow the next steps:

1. Import Eclipse project (`File > Import > General > Existing projects into Workspace`), as presented in the next figure.



Import

Select

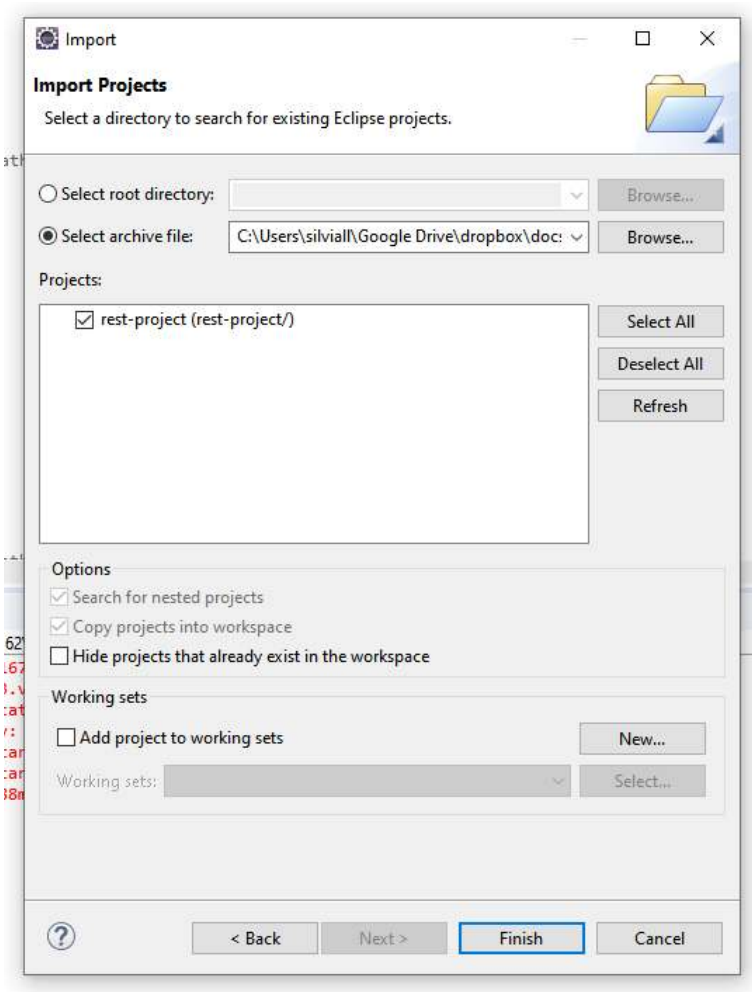
Create new projects from an archive file or directory.

Select an import wizard:

type filter text

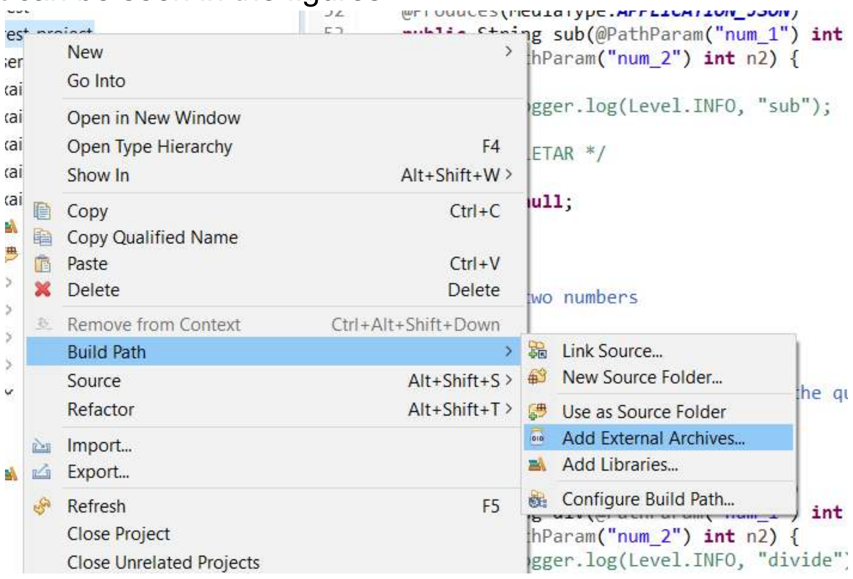
- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
  - Projects from Folder or Archive
- EJB
- Git
- Gradle
- Install
  - From Existing Installation
  - Install Software Items from File
- Java EE
- Maven
- Oomph
- Plug-in Development
- Run/Debug
- Tasks
- Team
- Web
- Web services
- XML

You have to select the file `xai-rest-base.zip` at the option `Select Archive File`. The project will appear (at the figure `rest-project`) and you can click the `Finish` button to end import.

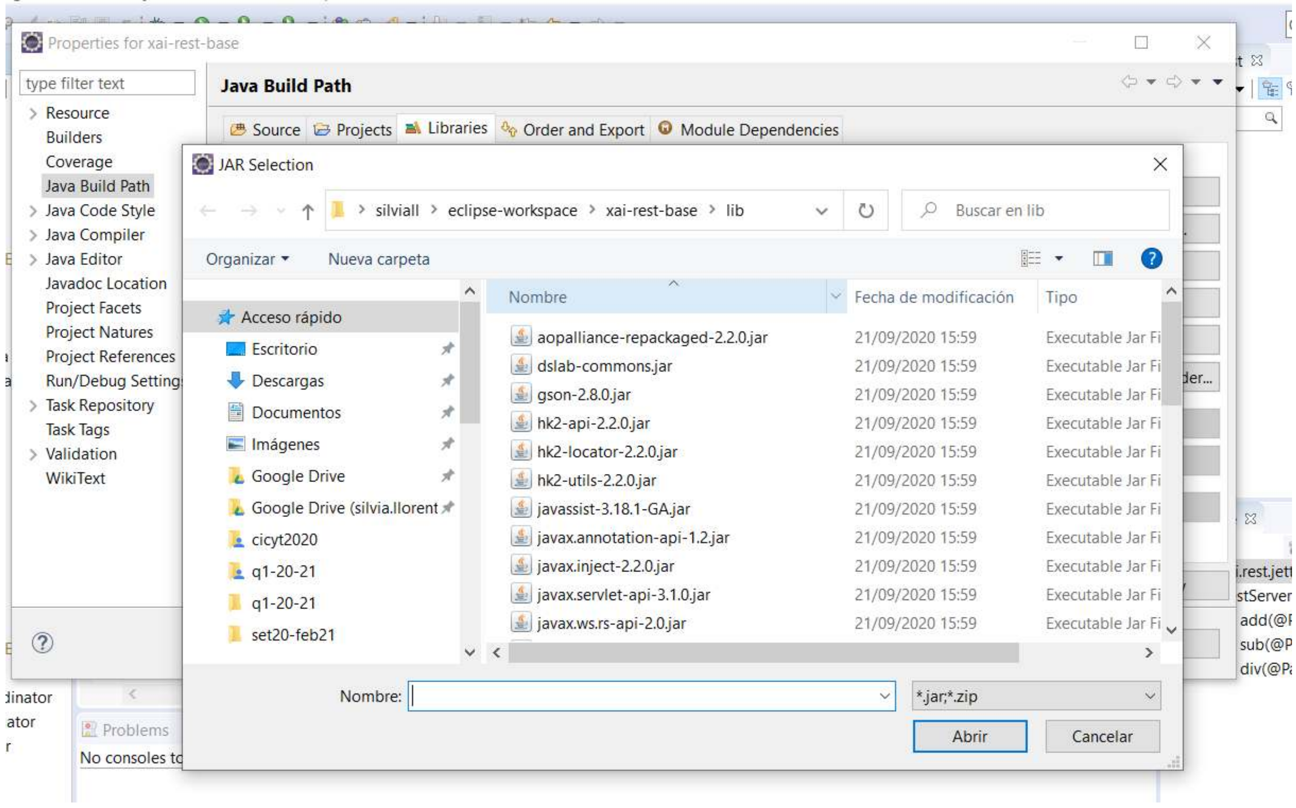

Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
Browse...
Select archive file:
C:\Users\silviall\Google Drive\dropbox\doc:
Browse...
Projects:
rest-project (rest-project/)
Select All
Deselect All
Refresh
Options
Search for nested projects
Copy projects into workspace
Hide projects that already exist in the workspace
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel

The .zip files contain all the needed libraries. If you have dependency errors, you have to solve them as explained in step 2.

2. To add new libraries, you have to modify the project Build Path, adding External Libraries, as it can be seen in the figures.

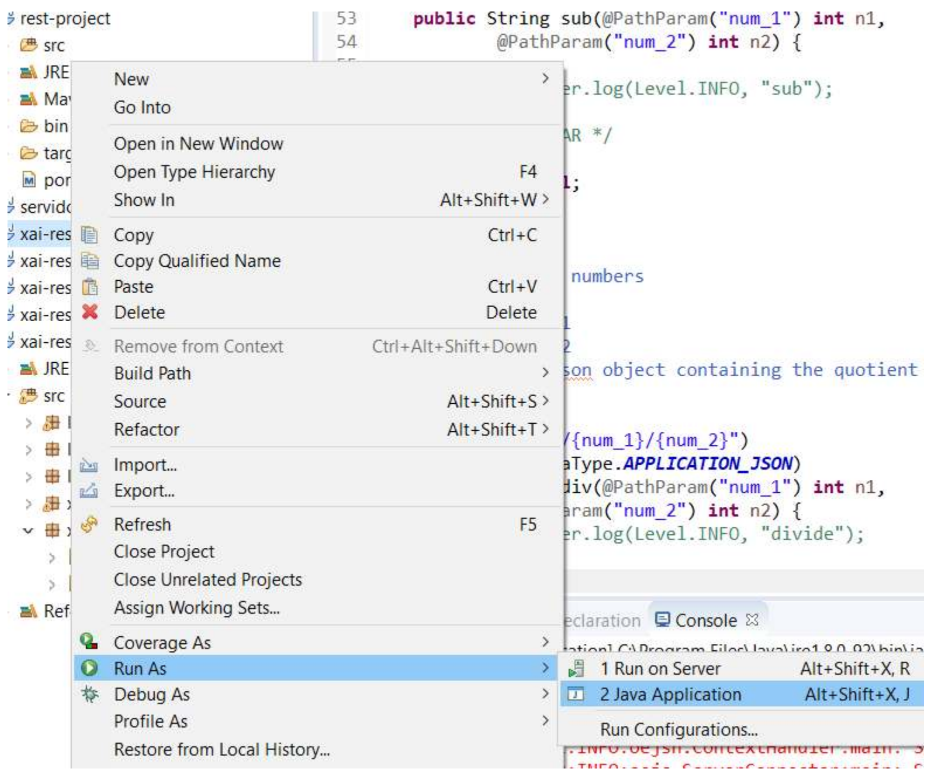


Extract from the zip file the complete lib folder in your machine (can be in the Eclipse project itself). Afterwards, from the JAR window selection, select all libraries inside this folder. The dependency errors should disappear.



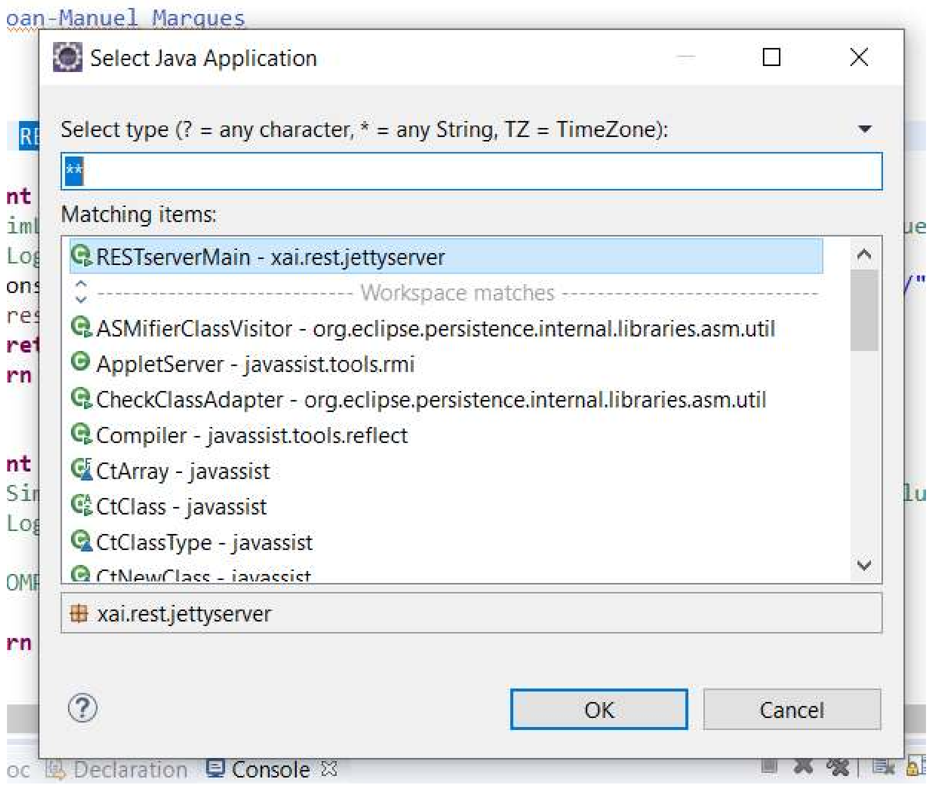
Nombre	Fecha de modificación	Tipo
aopalliance-repackaged-2.2.0.jar	21/09/2020 15:59	Executable Jar Fi
dslab-commons.jar	21/09/2020 15:59	Executable Jar Fi
gson-2.8.0.jar	21/09/2020 15:59	Executable Jar Fi
hk2-api-2.2.0.jar	21/09/2020 15:59	Executable Jar Fi
hk2-locator-2.2.0.jar	21/09/2020 15:59	Executable Jar Fi
hk2-utils-2.2.0.jar	21/09/2020 15:59	Executable Jar Fi
javassist-3.18.1-GA.jar	21/09/2020 15:59	Executable Jar Fi
javax.annotation-api-1.2.jar	21/09/2020 15:59	Executable Jar Fi
javax.inject-2.2.0.jar	21/09/2020 15:59	Executable Jar Fi
javax.servlet-api-3.1.0.jar	21/09/2020 15:59	Executable Jar Fi
javax.ws.rs-api-2.0.jar	21/09/2020 15:59	Executable Jar Fi

3. Once the project is installed and compiled, it can be executed. You have to select the option Java Application. If it asks for which class, look for `RESTserverMain` or `RESTclient`. The server has to be started first.



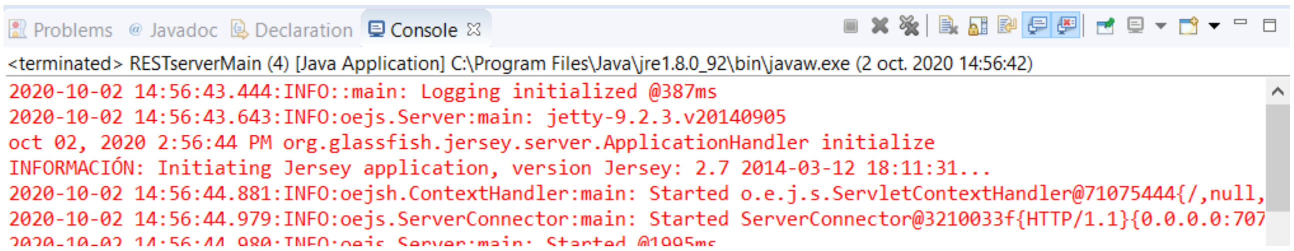
```
    public String sub(@PathParam("num_1") int n1,
            @PathParam("num_2") int n2) {
```

In the following image, you can see the selection of the class to be executed, `RESTserverMain.java`.



Select Java Application
Select type (? = any character, * = any String, TZ = TimeZone):
**
Matching items:
RESTserverMain - xai.rest.jettyserver
Workspace matches
ASMifierClassVisitor - org.eclipse.persistence.internal.libraries.asm.util
AppletServer - javassist.tools.rmi
CheckClassAdapter - org.eclipse.persistence.internal.libraries.asm.util
Compiler - javassist.tools.reflect
CtArray - javassist
CtClass - javassist
CtClassType - javassist
xai.rest.jettyserver
OK
Cancel

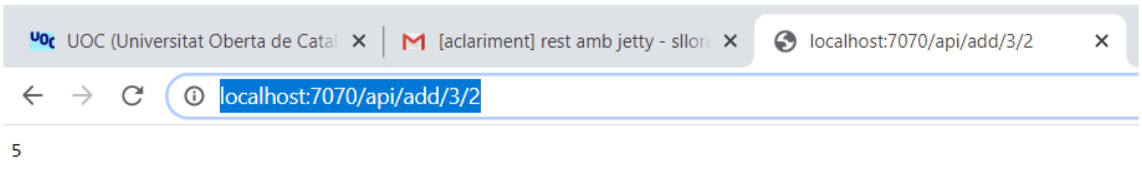
You can find some messages at the console, showing that the server is started.



<terminated> RESTserverMain (4) [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (2 oct. 2020 14:56:42)

```
2020-10-02 14:56:43.444:INFO::main: Logging initialized @387ms
2020-10-02 14:56:43.643:INFO:oejs.Server:main: jetty-9.2.3.v20140905
oct 02, 2020 2:56:44 PM org.glassfish.jersey.server.ApplicationHandler initialize
INFORMACIÓN: Initiating Jersey application, version Jersey: 2.7 2014-03-12 18:11:31...
2020-10-02 14:56:44.881:INFO:oejsh.ContextHandler:main: Started o.e.j.s.ServletContextHandler@71075444{/,null,
2020-10-02 14:56:44.979:INFO:oejs.ServerConnector:main: Started ServerConnector@3210033f{HTTP/1.1}{0.0.0.0:707
2020-10-02 14:56:44.980:INFO:oejs.Server:main: Started @1995ms
```

4. To check the server functionality, you can write a URL in the browser, `http://localhost:7070/api/add/3/2`, as it can be seen in the following figure.


UOC (Universitat Oberta de Catal
[aclariment] rest amb jetty - sllor
localhost:7070/api/add/3/2
localhost:7070/api/add/3/2

```
5
```

You can also use the Linux command `curl --request GET --url 'http://localhost:7070/api/add/3/2'` and verify that the addition of 2 and 3 is equal to 5. The operations subtraction and division are also implemented. You can review the code to see how to invoke them and check its correct operation.