



Bachelor's Degree in Techniques for Software Development

Network Applications and Internet

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 web server.

In parts 3 and 4 of this practical exercise, we will create a REST API to make some operations over strings 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 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 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:	<i>Maximum qualification C-</i>
2 parts delivery:	<i>Maximum qualification C+</i>
3 parts delivery:	<i>Maximum qualification B</i>
4 parts delivery:	<i>Maximum qualification A</i>

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 and 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-2025P-alumnes.zip` archive with the following project files:

- `UDPclientMain.java` (client main code)
- `Key.java` (class for storing the keys)
- `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)
- 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 datatype *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, One>`, `<k3, Eleven>`, `<k4, Twenty-one>`
- server 2 port 5838 `<k2, Thirty>`

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 *One*. 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 `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 *k1* key. Server will

respond *One* 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, One>, <k3, Eleven>, <k4, Twenty-one>, <k2, Thirty>`

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 `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 annex **Eclipse tutorial**.

In particular, we give you the `xai-sockets-tcp-2025P-alumnes.zip` archive with the following project files:

- `TCPclientMain.java` (client main code)
- `HTTPclient.java` (HTTP client to be modified according to this document)
- `HTTPPrequestResponse.java` (class for handling server request and response)
- Folder `lib` (libraries)

From code from file `HTTPclient.java` you have to program an **HTTP protocol mini-client**. We will focus on `HEAD` request, which works like `GET`, but only with the headers, not the body. In order to make the request as complete as possible, include the `Accept` and `Host` headers in the request. Without these headers, the request will be considered incorrect.

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 `HEAD / 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-strings-2025P-alumnes.zip`.

In this part of the practice, it is required to implement a service that returns the first position where a given character appears in a string. The parameters are the string and the character we are searching.

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:

`indexOf`: `http://localhost:7070/strings/indexOf/ephemeris/e`

The first value is the string to be used and the second the the character to be searched. The response should return a "text/plain" media type.

The result for this request should be 0, as it is the first position for letter 'e' in the string.

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 strings: `parts` and `replaceAll`. Remember that you have the **REpresentational State Transfer (REST)** teaching module as support.

Each operation has an access URL:

`parts`: `http://localhost:7070/strings/parts/la casa azul/z`

`replaceAll`: `http://localhost:7070/strings/replaceAll/la casa azul es casa/casa/rosa`

The first parameter is the string to be processed and a character to be searched in `parts`, and two words for `replaceAll`. The response depends on the operation, as described as follows:

- `parts` has to return an "application/json" media type where it is sent a set with the parts of the string separated by the given character. The set can be unordered.
- `replaceAll` has to return an "application/json" media type where the `Replaced` class, which we give you partially implemented, is sent. `Replaced` class contains the original string and the final string where all appearances of the second parameter have been changed by the third parameter.

Example results:

`parts`: `["ul", "la casa a"]`


`replaceAll`: `{"original_string": "la casa azul es casa", "new_string": "la rosa azul es rosa"}`

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 the Eclipse programming environment. You can download it from <https://www.eclipse.org/downloads/packages/release/2024-12/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



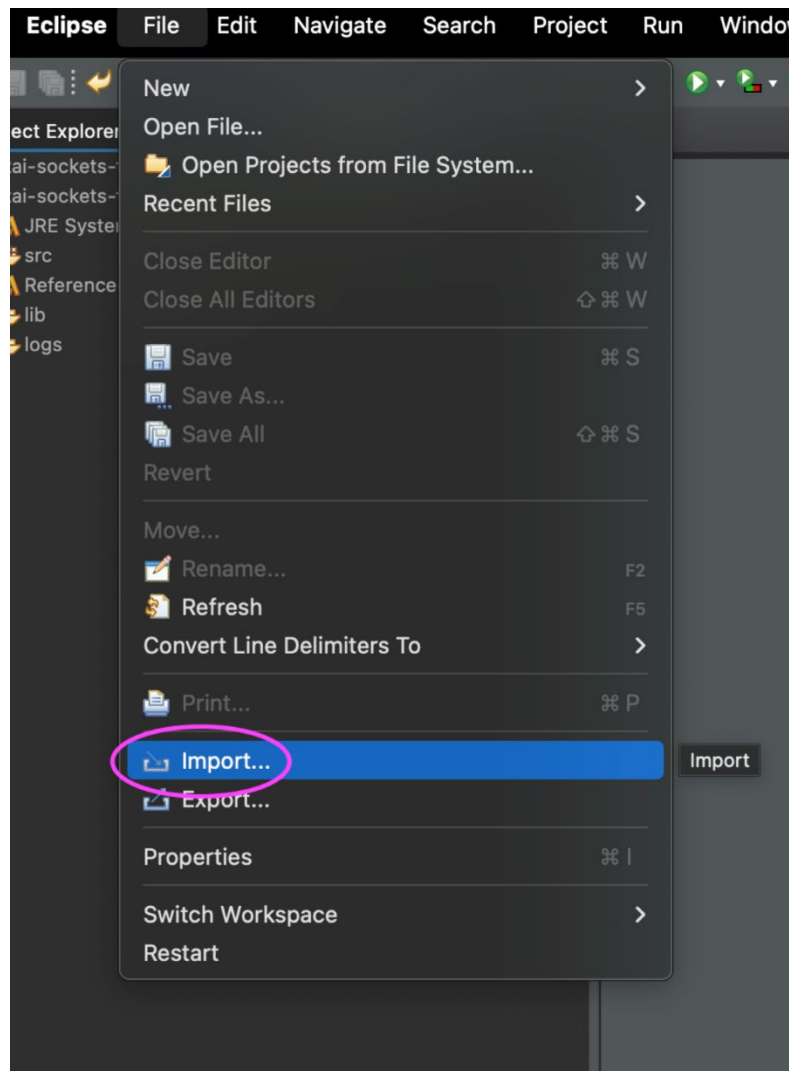
The Eclipse Installer 2022-09 R now includes a JRE for macOS, Windows and Linux.



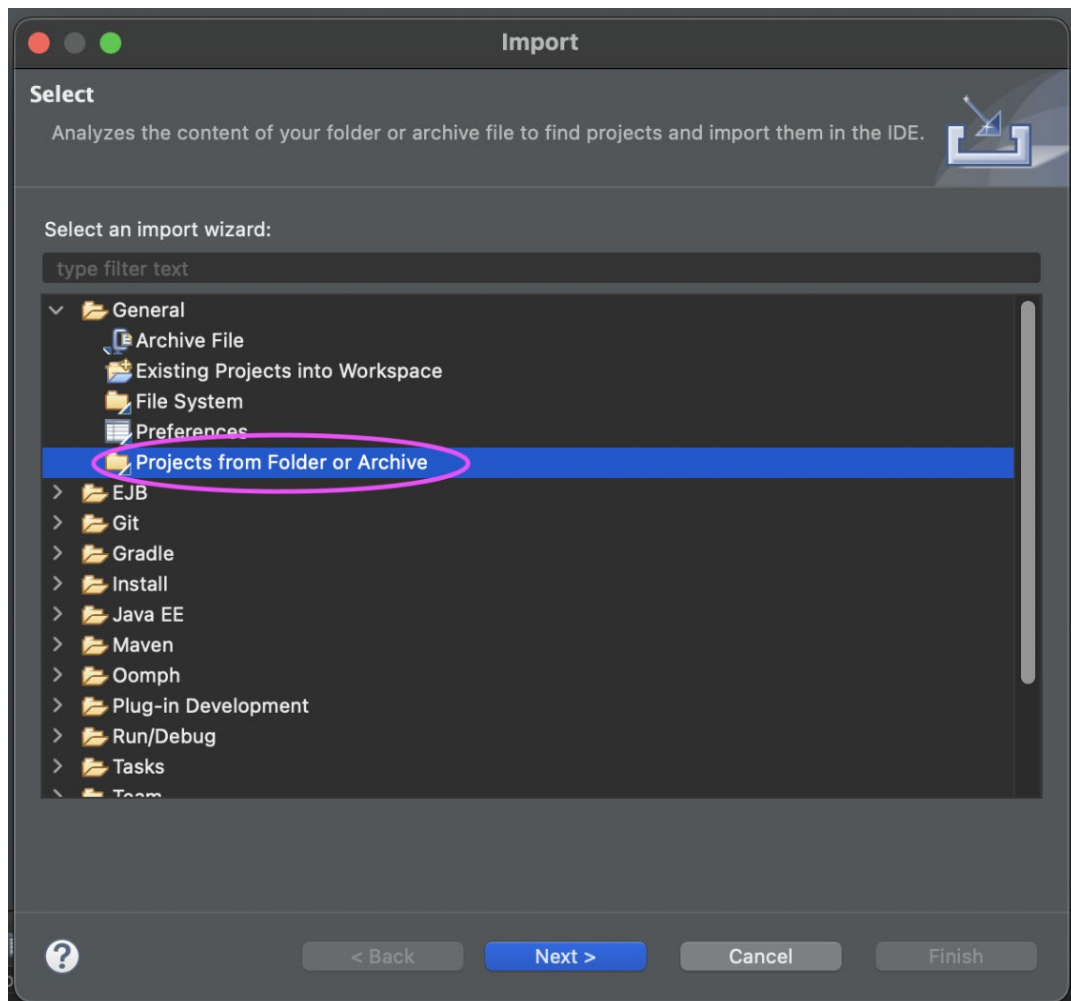
Get **Eclipse IDE 2022-09**

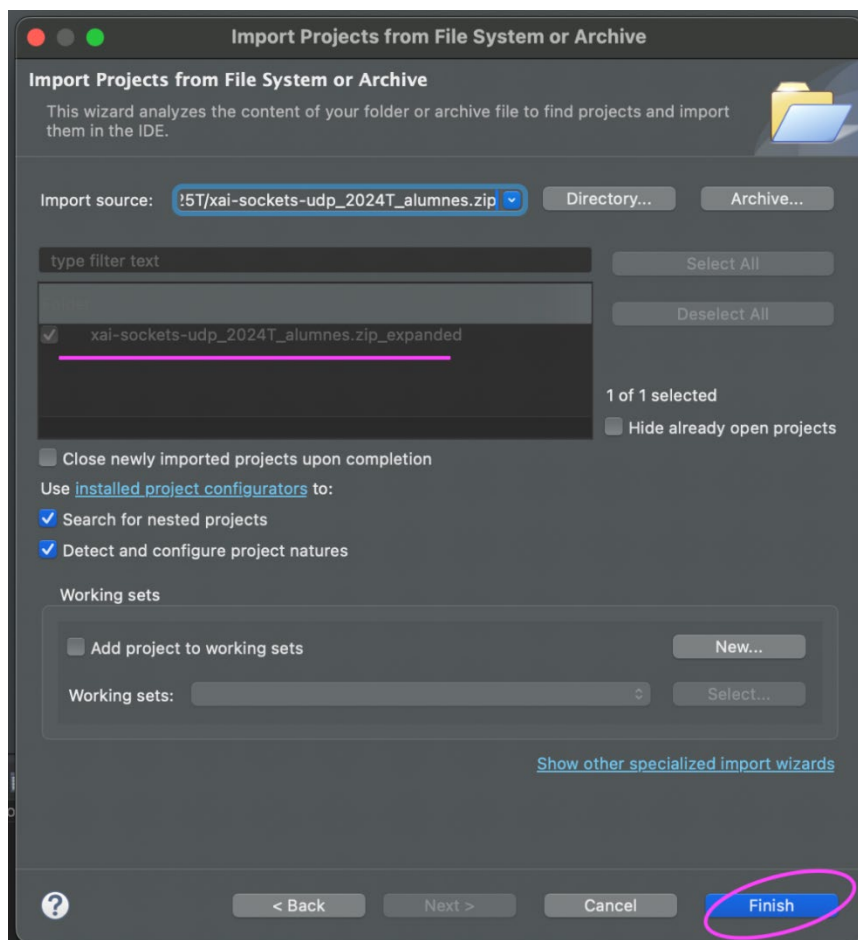
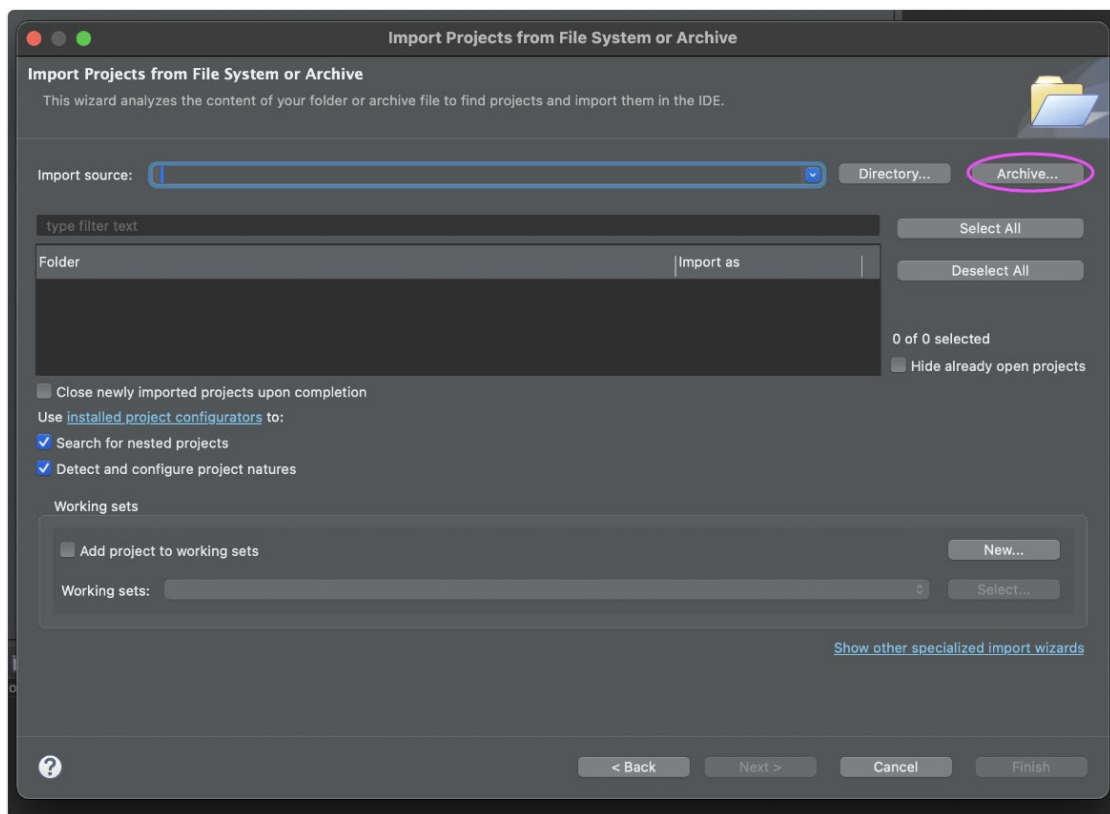
Install your favorite desktop IDE packages.

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*.

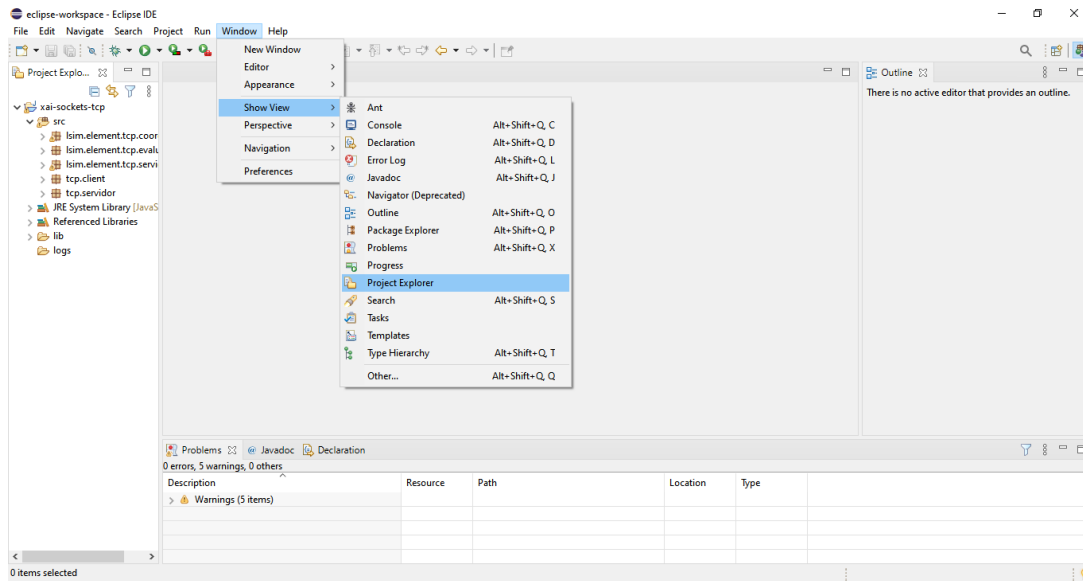


An emerging window will appear where you should select the compressed file we have provided, going to the option *Projects from Folder or Archive*, as it can be seen in the following screenshots. Once selected, you can click the *Finish* button.

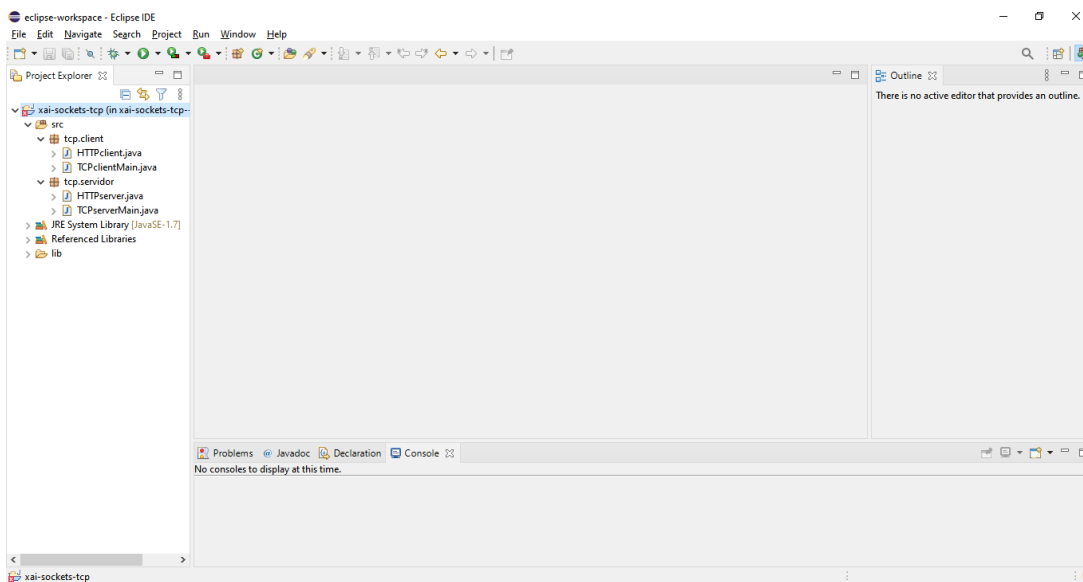




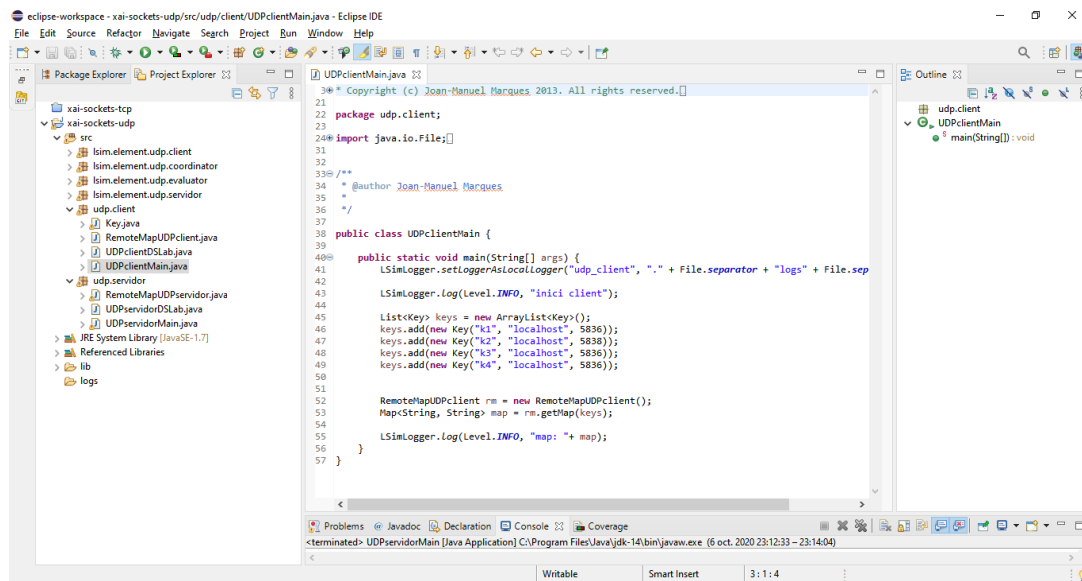
Once accepted by clicking *Finish*, you can see the project using the tree view, at the left side of Eclipse. If this view is not opened, you can find it going to the menu *Window > Show view > Explore project*. The name of the options may vary depending on the installation language.




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.



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.



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.

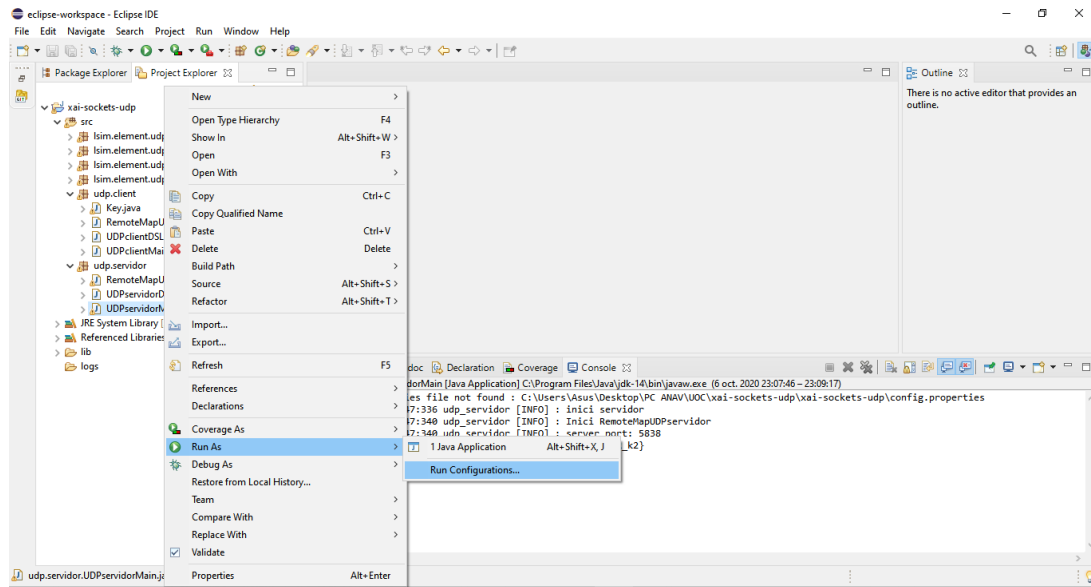
The initial result shown by the Eclipse console with code from part 2 should be something similar to the following image:

```

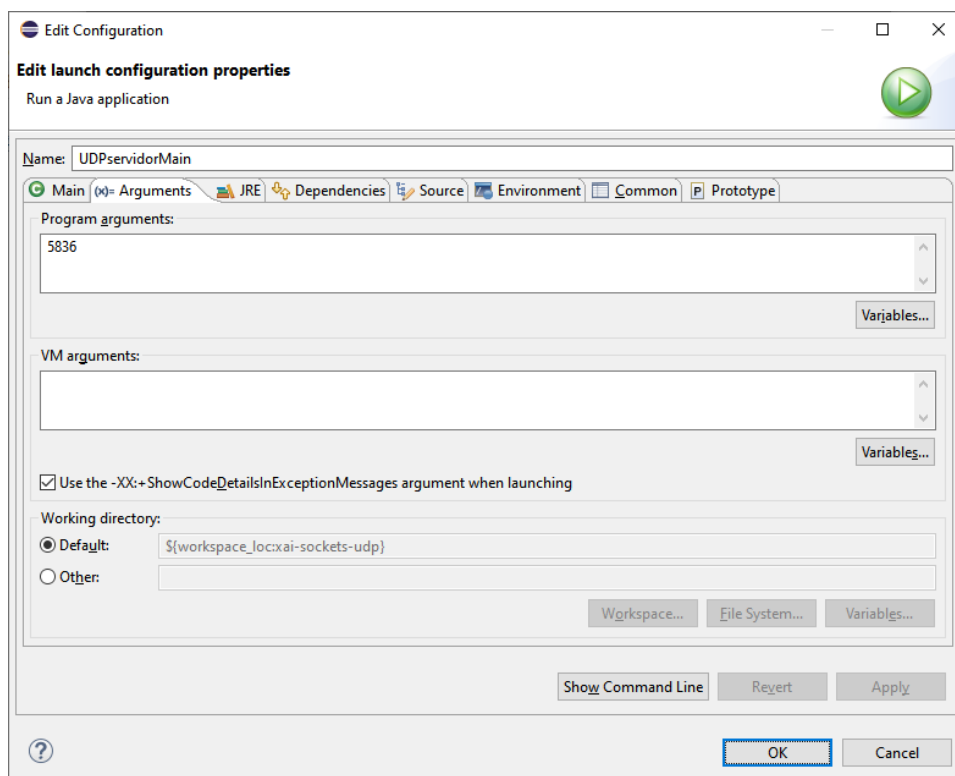
08-10-2020 22:13:44:573 tcp_client [INFO] : inici client http
08-10-2020 22:13:44:584 tcp_client [INFO] : inici HTTPclient.get
08-10-2020 22:13:44:584 tcp_client [INFO] : HTTP server_address: example.org
08-10-2020 22:13:44:586 tcp_client [INFO] : HTTP server_port: 80
08-10-2020 22:13:44:586 tcp_client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

```

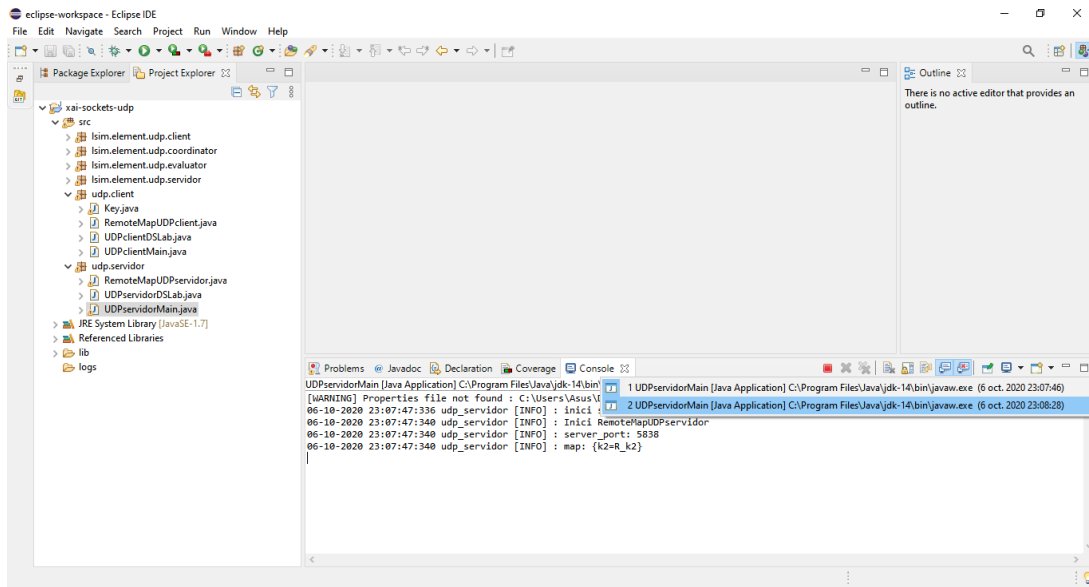
For the first part of the practice, where the server needs to be executed with the port number passed as parameter, you can click with the right button over the class `UDPServidorMain` at the project explorer view (tree at the left side of the screen) and go to the configuration inside the option *Run*.



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.



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:



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}
```

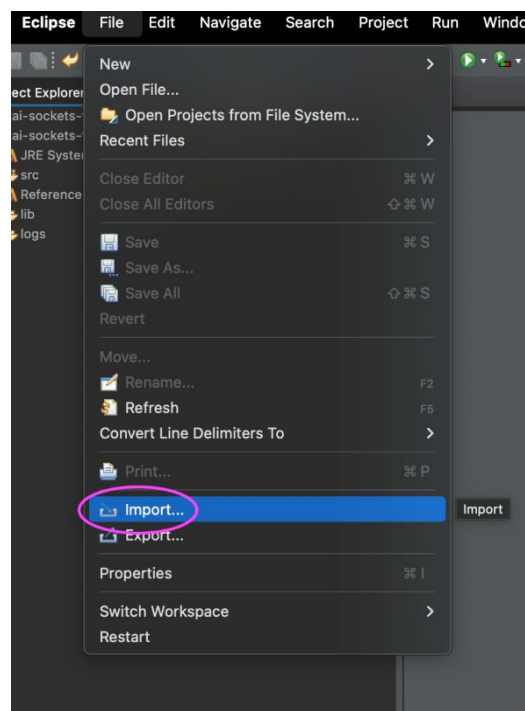
Annex: Installing a REST service in Eclipse (Parts 3 and 4)

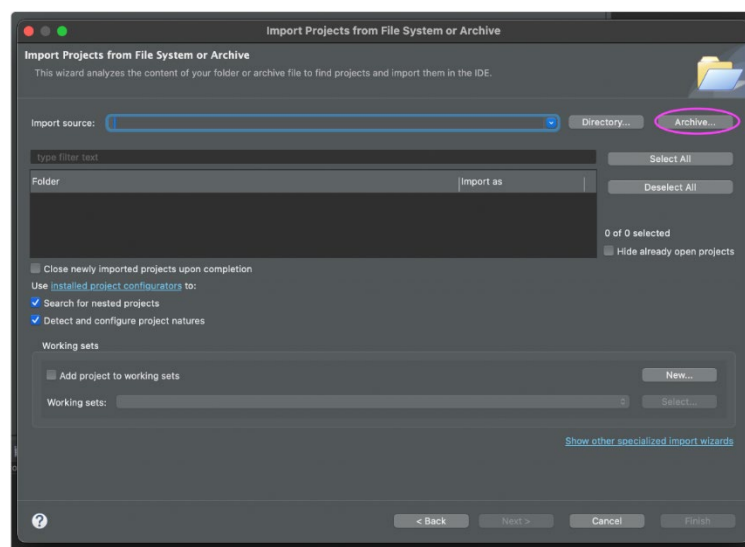
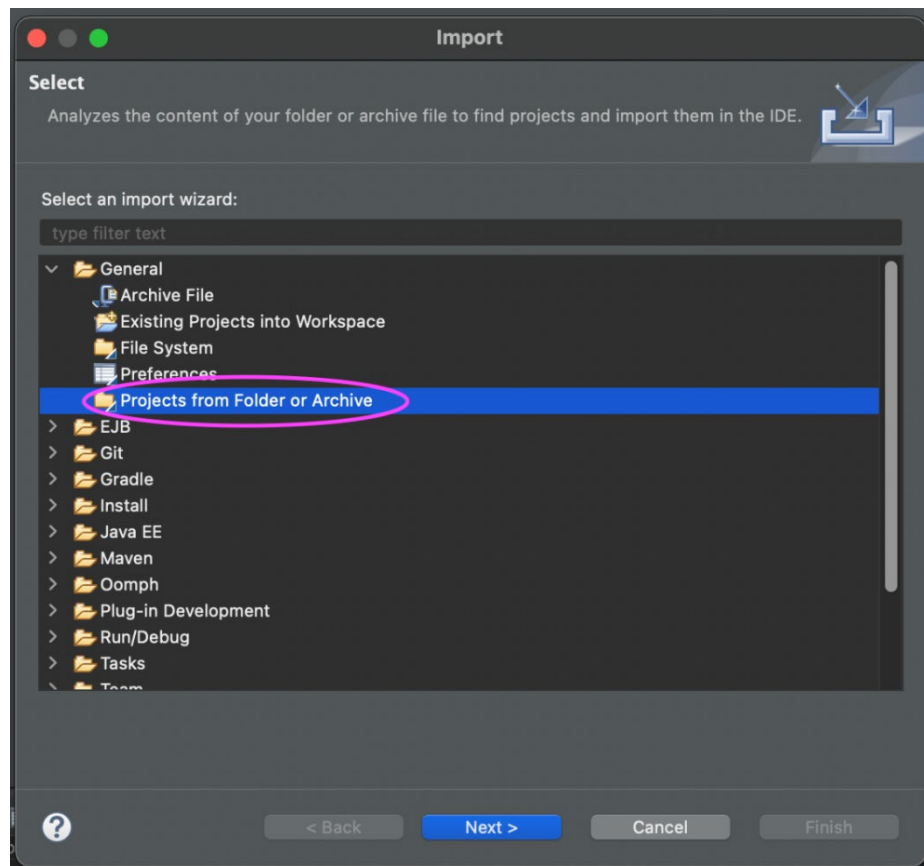
In this annex, we explain how to install, execute and test a base REST service we provide you in this practice. It is a simple calculator that allows additions, subtractions and divisions of two integer numbers. With this service as a base, you have to implement what is proposed in this document.

You need to install the Eclipse programming environment. You can download it from <https://www.eclipse.org/downloads/packages/release/2024-12/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.

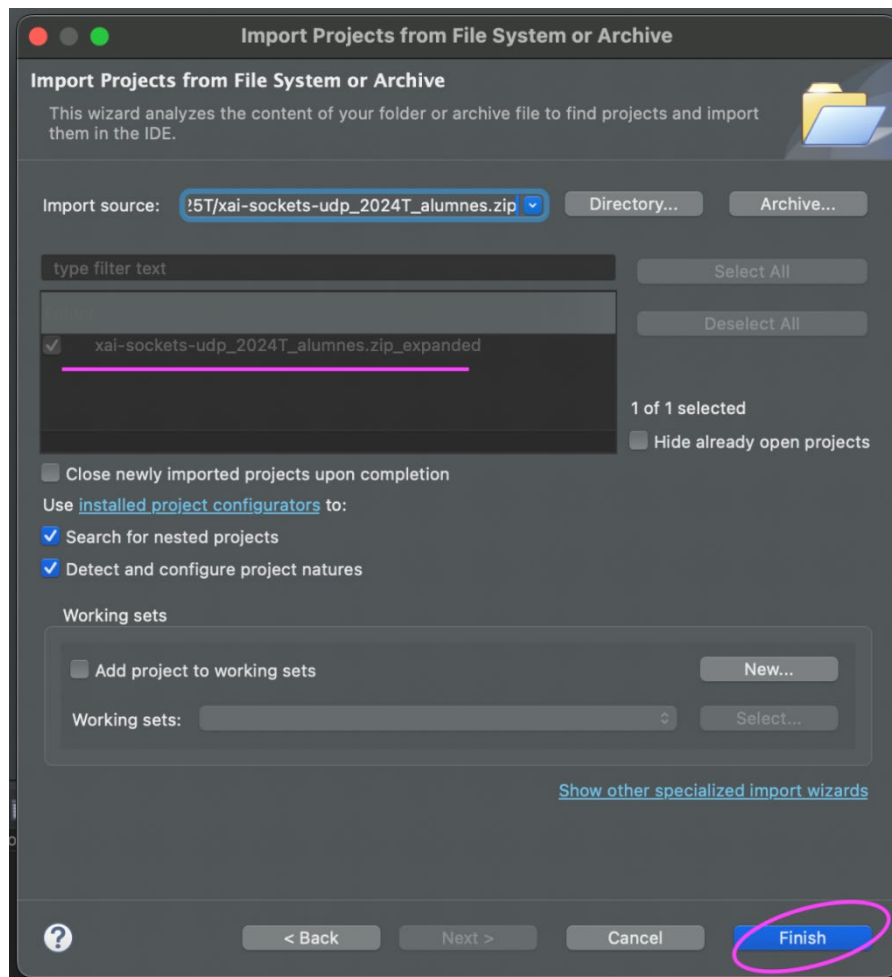
Once Eclipse is installed, you need to follow the next steps:

1. Import Eclipse project (File > Import > General > Projects from Folder or Archive), as presented in the next figure.



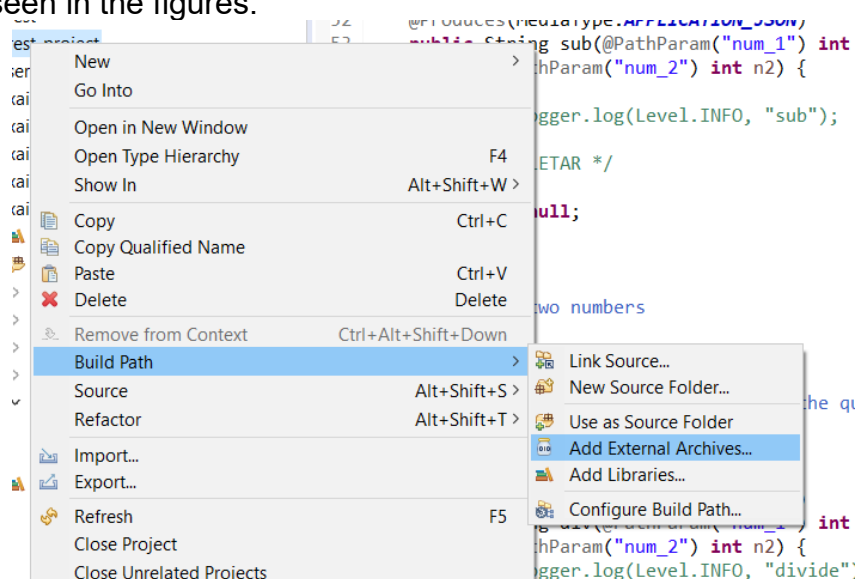


You have to select the file `xai-rest-base.zip` at the option `Select Archive File`. The project will appear (at the figure `xai-sockets-udp_2024T_alumnes.zip`) and you can click the `Finish` button to end importation.

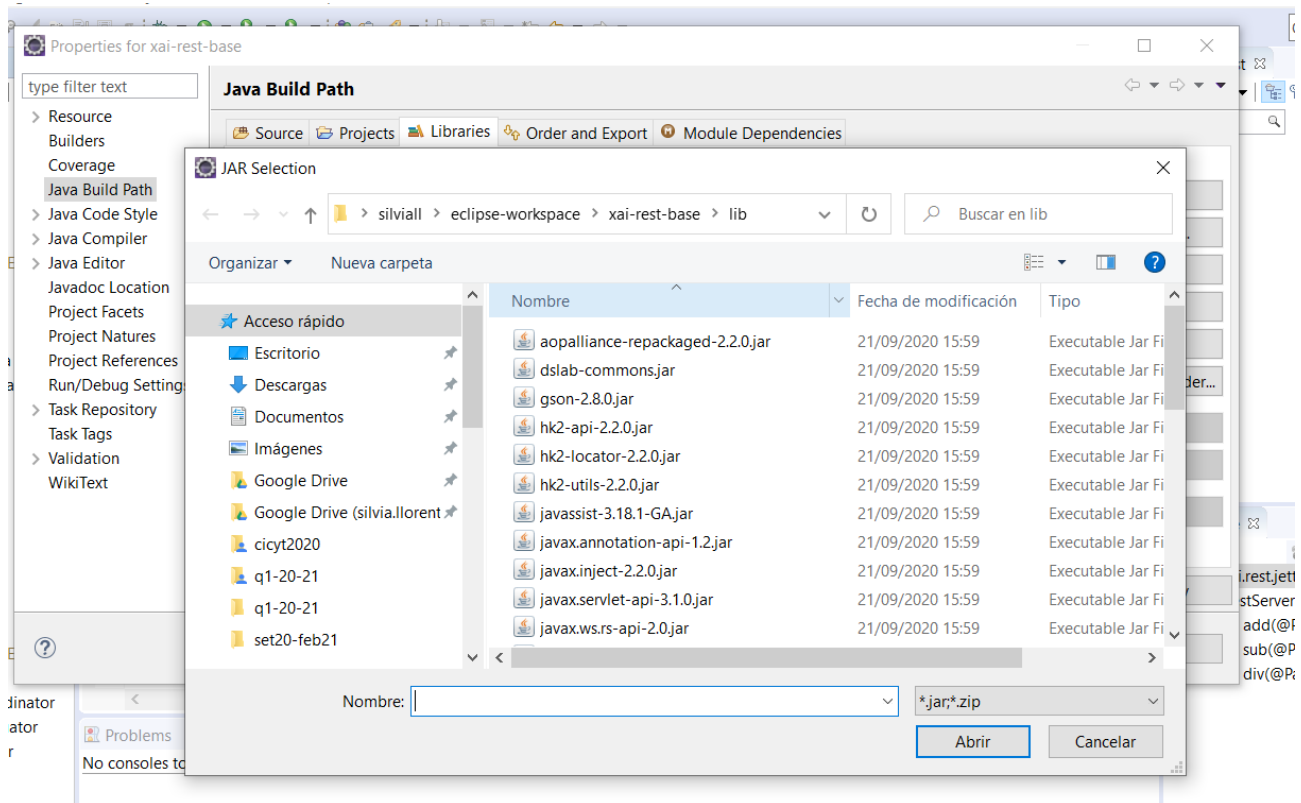


The .zip files contain all the needed libraries. If you have dependency errors, you have to solve them as explained in step 2.

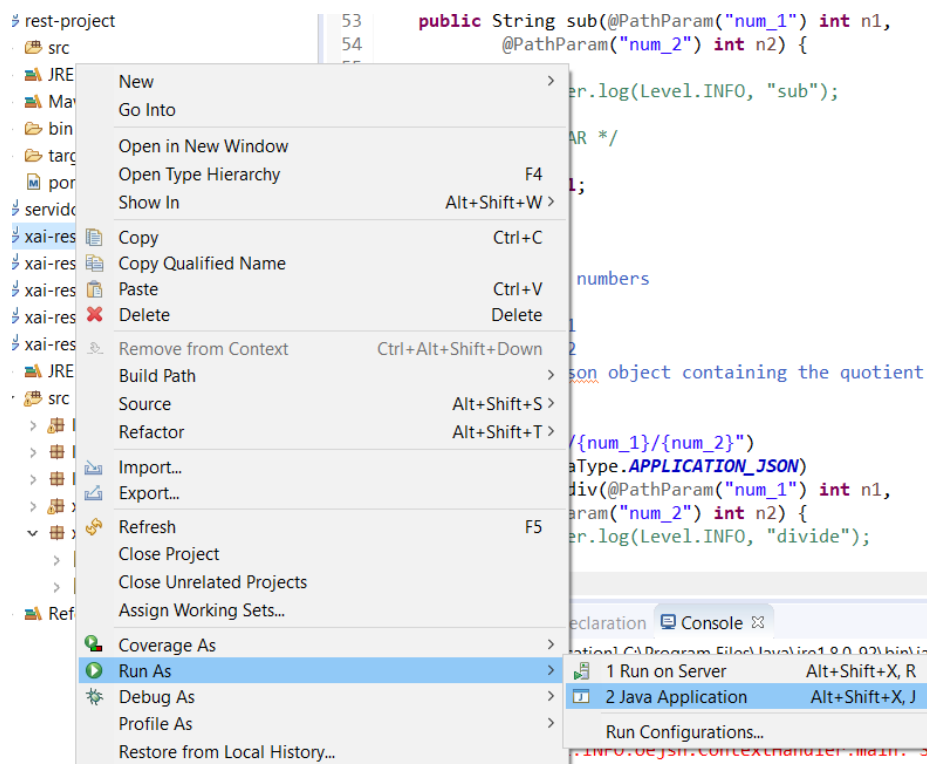
2. To add new libraries, you have to modify the project Build Path, adding External Libraries, as it can be seen in the figures.



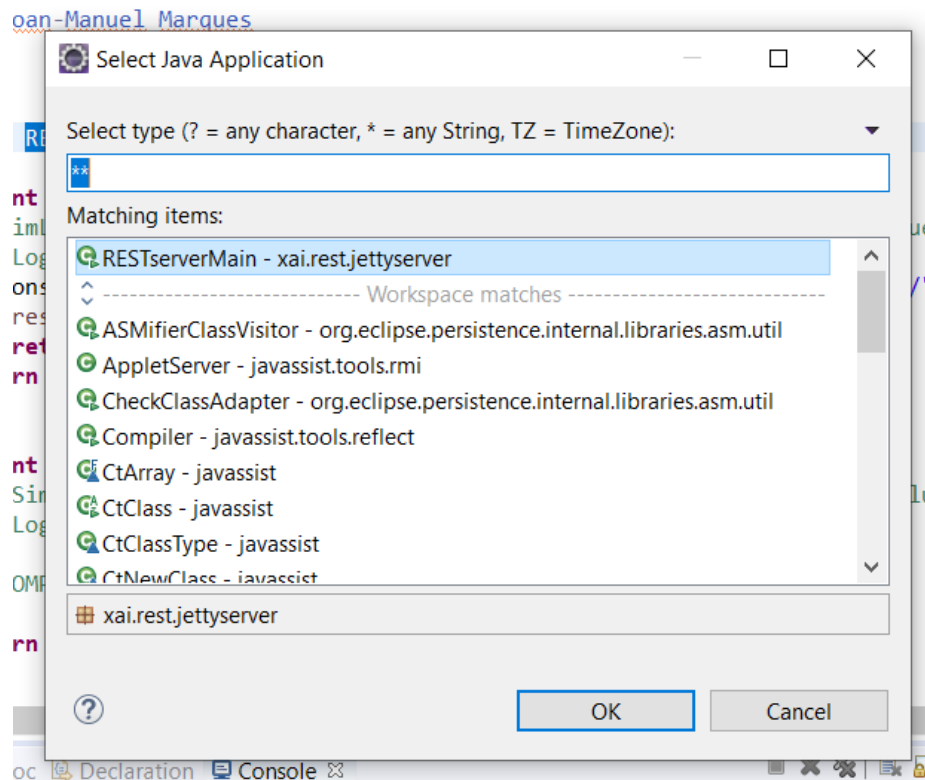
Extract from the zip file the complete lib folder in your machine (can be in the Eclipse project itself). Afterwards, from the JAR window selection, select all libraries inside this folder. The dependency errors should disappear.



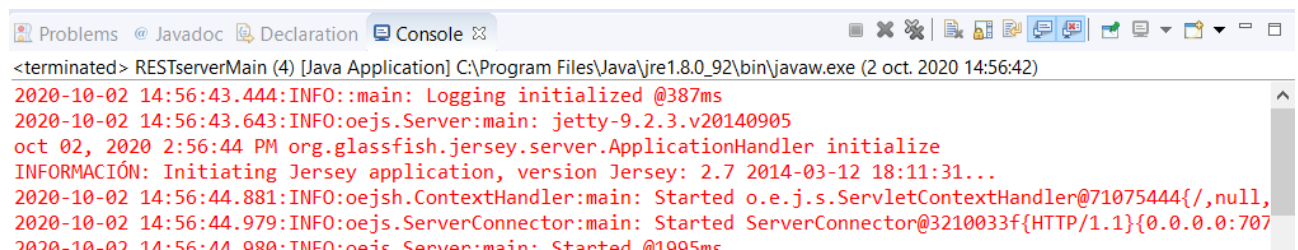
- Once the project is installed and compiled, it can be executed. You have to select the option Java Application. If it asks for which class, look for `RESTserverMain` or `RESTclient`. The server has to be started first.



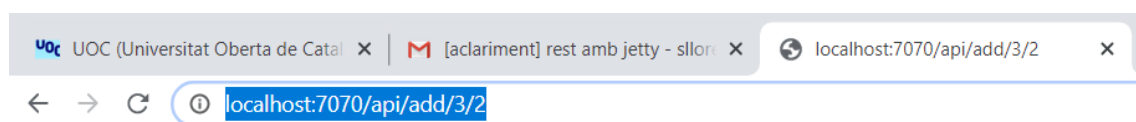
In the following image, you can see the selection of the class to be executed, `RESTserverMain.java`.



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



- 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.



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.