



RAI SI Practica 2 2024P

Redes y aplicaciones Internet (Universitat Oberta de Catalunya)



Escanea para abrir en Studocu



Estudios de Informática, Multimedia y Telecomunicación

Redes y Aplicaciones Internet y Servicios de Internet

Actividad: Práctica 2. ¡Crea tu propia aplicación en red!

En esta actividad, os proponemos crear aplicaciones que comuniquen diferentes dispositivos de la red, a partir de la programación en Java de algunos de los paradigmas de comunicación que más se utilizan hoy en día.

En las dos primeras partes de la práctica nos centraremos en la programación con sockets, tanto orientados a la conexión como no. En la primera parte, implementaremos un programa cliente que se comunicará con servidores UDP. En la segunda parte, programamos un cliente HTTP que hará peticiones a un servidor web dado.

En las partes 3 y 4 de la práctica, vais a crear una API REST para realizar operaciones con palabras e intercambiaremos datos utilizando distintas alternativas de codificación, incluyendo JSON.

En cada una de las partes, os entregamos un código de partida que os tiene que servir como plantilla. Es decir, tenéis que añadir instrucciones para que funcione según se describe en el enunciado. No podéis eliminar ninguna de las líneas de código de la plantilla, ya que están escritas para la corrección automática de la práctica en DSLab. Tampoco podéis copiar código de ninguna otra fuente (Internet, libros, etc.).

Veréis que hay instrucciones relativas al log, donde se registran las instrucciones que se van ejecutando. Si queréis utilizarlas para vuestro propio control, podéis consultar un breve tutorial en la siguiente dirección <https://sd.uoc.edu/dslab/help/>

Los proyectos están preparados para funcionar con Eclipse como entorno de programación y simulación. Además, en las partes 3 y 4 se utiliza un servidor integrado, Jetty, que no tenéis que instalar, ya que está incorporado en el propio proyecto. Podéis encontrar un pequeño manual de cómo importar proyectos en Eclipse en:

- Anexo **Manual de uso de Eclipse (partes 1 y 2)**
- Anexo **Instalación servicio REST en Eclipse (partes 3 y 4)**

Otros IDE's tienen un funcionamiento similar. Si los importáis a otro IDE, configurad los *classpath* de las bibliotecas que se utilizan. También podéis compilar y probar desde la línea de comandos del sistema operativo concreto.

Como material principal de soporte para realizar la práctica, os proponemos los siguientes módulos, que incluyen la parte teórica y la programación en Java. Podéis encontrarlos en el aula:

- **Módulo didáctico. Mecanismos de comunicación**
- **Módulo didáctico. Programación de sockets en Java**
- **Módulo didáctico. REpresentational State Transfer (REST)**

Calificación de la práctica 2

Esta práctica consta de cuatro partes. Las partes 1 y 2 son independientes. Las partes 3 y 4 están relacionadas entre ellas (la 4 es una evolución de la 3, pero es necesario entregar las dos de forma separada). No es necesario que se realicen en orden secuencial (por ejemplo, podéis hacer solo la parte 1 y la parte 3). La nota irá en función de las partes entregadas:

Entrega de 1 parte:	<i>Nota máxima C-</i>
Entrega de 2 partes:	<i>Nota máxima C+</i>
Entrega de 3 partes:	<i>Nota máxima B</i>
Entrega de 4 partes:	<i>Nota máxima A</i>

Para cada parte de la práctica, tenéis que entregar los códigos fuente de los archivos modificados cliente y servidor, tal y como se explica en este documento. Enviad vuestras respuestas al registro de evaluación continua del aula de la asignatura. Usad algún programa de compresión para hacer la entrega en un solo archivo. Nombrad el archivo:

Practica2_Apellido1_Apellido2_Nom.[zip, tar, ...]

Podéis utilizar cualquier servidor para realizar pruebas, local o remoto. No obstante, **solo se considerará como correcta una práctica que se haya ejecutado con resultado satisfactorio en DSLab** <https://sd.uoc.edu/dslab/>

Para ejecutar vuestra práctica en el entorno de corrección automático DSLab tenéis que seguir estos pasos:

- Crear un proyecto (menú *Projects> New Project*).
- Seleccionar la tarea que corresponda a la parte del enunciado que queréis corregir: *Assignment y task*.
- Subir el código fuente de esa parte, clicando en el proyecto que acabáis de crear. Fijaos que en la parte inferior de la pantalla, os aparecen los archivos que tenéis que subir (*Required files*).
- Crear una corrección automática (menú *Submissions> New Submission*) del proyecto creado en el primer punto. El resultado de las ejecuciones se puede consultar en el menú *Submissions*.

Primera parte: Programación con sockets UDP

En esta primera parte trabajaremos con sockets no orientados a la conexión, mediante la programación en Java. Tenéis como material de apoyo el módulo didáctico **Programación de sockets en Java** y el anexo **Manual de uso de Eclipse**.

En particular, os damos el archivo `xai-sockets-udp--2024P-alumnes.zip` donde encontraréis los siguientes archivos del proyecto:

- `UDPclientMain.java` (código principal del cliente)
- `RemoteMapUDPclient.java` (cliente UDP a modificar según el enunciado)
- `UDPservidorMain.java` (código principal del servidor)
- `RemoteMapUDPservidor.java` (servidor UDP a modificar según el enunciado)
- `Key.java` (gestión de las claves)
- Carpeta `lib` (librerías)

En este caso, tendréis que programar tanto el cliente como el servidor, que se comunicarán vía sockets no orientados a la conexión. El objetivo principal es que un cliente construya un solo mapa a partir de los datos de los mapas de los demás servidores. El tipo de datos *Map* se encarga de almacenar una colección de objetos, típicamente relacionando una clave (*Key*) con un valor.

En primer lugar, ejecutaréis la clase `UDPservidorMain`, pasando como parámetro el número de puerto. Los servidores que hay actualmente configurados en la práctica funcionan sobre los puertos 5836 y 5838. En esta clase, se crea e inicializa el mapa del servidor, donde se almacenan parejas de datos <clave, valor> y, concretamente, en vuestra práctica serán parejas <string, string>, con los siguientes datos de prueba:

- el servidor 1 puerto 5836 <k1, R_k1>, <k3, R_k3>, <k4, R_k4>
- el servidor 2 puerto 5838 <k2, R_k2>

Después de inicializar los mapas, los servidores crean una instancia de la clase `RemoteMapUDPservidor`. Esta clase es la que tenéis que modificar. Se encargará de atender peticiones de clientes remotos, respondiendo con el valor correspondiente a la clave concreta que le han pedido. Por ejemplo, si un cliente le pregunta al servidor 1 si contiene el valor *k1*; el servidor le responderá con el valor *R_k1*. Podéis añadir más servidores o incluso más datos al mapa; o dejar uno solo para hacer las pruebas iniciales. Las pruebas finales en la corrección automática de la práctica a DSLab se harán con mapas aleatorios para cada alumno.

Los clientes se inician ejecutando la clase `UDPclientMain`. En primer lugar, se construye el conjunto del cliente con las claves a preguntar y los servidores que las tienen (direcciones y puertos). En vuestro ejemplo, el conjunto del cliente contiene:

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

Después se crea una instancia de la clase `RemoteMapUDPClient`, y se invoca al método `getMap`. Esta función recorre el mapa inicial de claves, que acabamos de detallar en el ejemplo, y pregunta al servidor indicado en cada entrada, el valor asociado a la clave (función `get`). Por ejemplo, el cliente se conectará al servidor `localhost:5836` y le preguntará por el valor `k1`. El servidor le responderá con el valor `R_k1` y así sucesivamente. A medida que va recibiendo los valores de las claves, el cliente va construyendo un nuevo map, similar al de los servidores, pero cuyo contenido es la unión de los datos de todos ellos. Así pues, siguiendo nuestro ejemplo, el conjunto final del cliente tendría que contener:

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

En el caso de los clientes, tenéis que modificar la clase `RemoteMapUDPClient`, ya que es la encargada propiamente de hacer las consultas remotas de los valores que faltan en el mapa. Concretamente, solo tenéis que modificar el método `get`, que recibe como parámetros la dirección y el puerto del servidor, así como la clave a preguntar. Tendréis que crear un socket UDP y averiguar el valor asociado a la clave, preguntando al servidor asociado. El valor respuesta del servidor es la cadena de texto que tiene que devolver el método `get`.

Finalmente, recordad cerrar todos los sockets creados para liberar recursos del sistema.

Segunda parte: Programación con sockets TCP

En esta segunda parte trabajaremos con sockets orientados a la conexión, mediante la programación en Java. Como material de apoyo, tenéis el módulo didáctico **Programación de sockets en Java** y el anexo **Manual de uso de Eclipse**.

En particular, os damos el archivo `xai-sockets-tcp--2024P-alumnes.zip` donde encontraréis los siguientes archivos del proyecto:

- `TCPClientMain.java` (código principal del cliente)
- `HTTPClient.java` (cliente HTTP a modificar según el enunciado)
- `HTTPPrequestResponse.java` (clase para tratar los datos de la petición y la respuesta)
- Carpeta `lib` (librerías)

A partir del código del fichero `HTTPClient.java` tenéis que programar un **mini-cliente del protocolo HTTP**. Nos centraremos en la petición `GET`, que retorna el código de respuesta, las cabeceras HTTP y el recurso solicitado. Para que la petición sea lo más completa posible, incorpora a la petición la cabecera `Accept`.

En primer lugar, ejecutamos la clase `TCPClientMain`, donde manualmente tendréis que escribir el servidor web al que os queréis conectar (dirección y puerto). Acto seguido, se crea una instancia de la clase `HTTPClient` que será la encargada de gestionar la comunicación remota entre los dos hosts mediante sockets TCP. Es esta clase la que tenéis que modificar.

Primero, el cliente se conectará al servidor indicado. Cuando se haya aceptado la conexión, el cliente construirá la petición `GET /xai/xai.html HTTP/1.0`, contra el servidor `labxarxes.techlab.uoc.edu`, la mostrará por pantalla y la enviará al servidor. Cuando llegue la respuesta del servidor, igualmente tendréis que mostrarla por pantalla. Finalmente, recordad cerrar los sockets para liberar recursos del sistema.

Hay que tener en cuenta que al final de cada línea de la petición y la respuesta HTTP, incluyendo las cabeceras, el estándar HTTP indica que tienen que ir los caracteres CRLF (Carriage return y Line Feed `\r\n`). Si no se hace así, os fallará la corrección de DSLab aunque el resto de datos sean correctos.

Para la ejecución local, la conexión se hace directamente contra el servidor web final. En cambio, en DSLab se utiliza un Proxy para poder hacer la corrección. Tenedlo en cuenta a la hora de revisar los logs de DSLab.

Tercera parte: Programación básica REST

El punto de partida de la especificación de una aplicación distribuida es la descripción de los recursos que se pueden invocar remotamente. En esta práctica se pide invocar a dichos recursos remotos con un servicio web basado en **REST** (REpresentational State Transfer). Recordad que tenéis el módulo didáctico **REpresentational State Transfer (REST)** como soporte.

REST no tiene limitación en el formato de peticiones y respuestas (pueden ser texto, JSON, XML, etc.). Así, para definir un servicio REST se puede definir una API indicando como se puede pedir un recurso, incluyendo los datos que tiene que contener la petición y dando un ejemplo de respuesta.

Os proporcionamos un proyecto Eclipse con el esqueleto de un servicio que funciona sobre el servidor Jetty. El servidor Jetty no es necesario instalarlo, está dentro del propio proyecto. Lo único que tenéis que hacer es importar en Eclipse el proyecto que os proporcionamos. En el anexo **Instalación Servicio REST en Eclipse**, tenéis las instrucciones detalladas de instalación. En este anexo se utiliza el código del fichero `xai-rest-base.zip`, aunque no está implementada toda la funcionalidad, es solo un ejemplo más sencillo de Servicio REST. Lo que vosotros tenéis que completar es lo que está en el fichero `xai-rest-paraules--2024P-alumnes.zip`.

En esta parte de la práctica se pide que implementéis un servicio que realice algunas operaciones con palabras tal y como se detalla a continuación.

Los ficheros a modificar son (tienen definidas las operaciones de las partes 3 y 4 de la práctica):

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

La URL de acceso de la operación de calcular la longitud de una palabra tiene que ser:

Longitud de una palabra: `http://localhost:7070/words/length/{paraula}`

A esta petición se le pasa como parámetro una palabra y devuelve su longitud en formato numérico, pero tiene que estar en un media type de tipo "text/plain".

También tenéis que implementar un programa cliente que invoque la operación del servicio y muestre el resultado. Podéis añadir las clases necesarias en el proyecto del servidor o crear un nuevo proyecto.

Cuarta parte: Programación avanzada REST

En este apartado se tiene que continuar con la implementación del cliente y servidor REST. En concreto, se pide que implementéis dos operaciones más sobre palabras en el

servicio del apartado 3: split y paso a minúsculas. Los parámetros y respuestas de cada operación se detallan a continuación. Recordad que tenéis el módulo didáctico **REpresentational State Transfer (REST)** como soporte.

Cada operación tiene una URL de acceso:

Split a partir de una expresión regular (regex):

`http://localhost:7070/words/split/{paraula}/{regex}`

Paso a minúsculas: `http://localhost:7070/words/toLowerCase/{paraula}`

El parámetro palabra indica la palabra sobre la que se quiere hacer la operación. El parámetro regex indica la expresión por la cual se tiene que separar (split) la palabra dada.

La respuesta depende de la operación, tal y como se describe a continuación:

- Split debe devolver un media type de tipo "application/json" donde se envía una clase de Java tipo ArrayList.
- Paso a minúsculas tiene que devolver un media type "application/json" donde se envía la clase Capitalized, que os damos parcialmente implementada. Además de la palabra en minúsculas, también se devuelve un booleano que nos indica si la palabra se ha modificado o no.

Ejemplo de resultado para la operación split:

```
["Eph", "m", "ris"]
```

Ejemplo de resultado en formato JSON para la operación paso a minúsculas:


```
{"word": "ephemeris", "modified": true}
```

También tenéis que implementar las llamadas a estas nuevas operaciones desde el programa cliente y mostrar el resultado.

Anexo: Manual de uso de Eclipse (Partes 1 y 2)

Este anexo describe brevemente cómo importar el proyecto que os proporcionamos tanto en la parte 1 como en la parte 2 de la práctica, utilizando el IDE Eclipse. Puede haber alguna pequeña variación según la versión que utilicéis.

Es necesario tener instalado el entorno de programación Eclipse. Podéis descargarlo de <https://www.eclipse.org/downloads/packages/release/2022-12/r/eclipse-ide-enterpris-e-java-and-web-developers>. Si no es la última versión, se os indicará donde podéis descargar la versión más actual. La instalación es muy intuitiva.



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



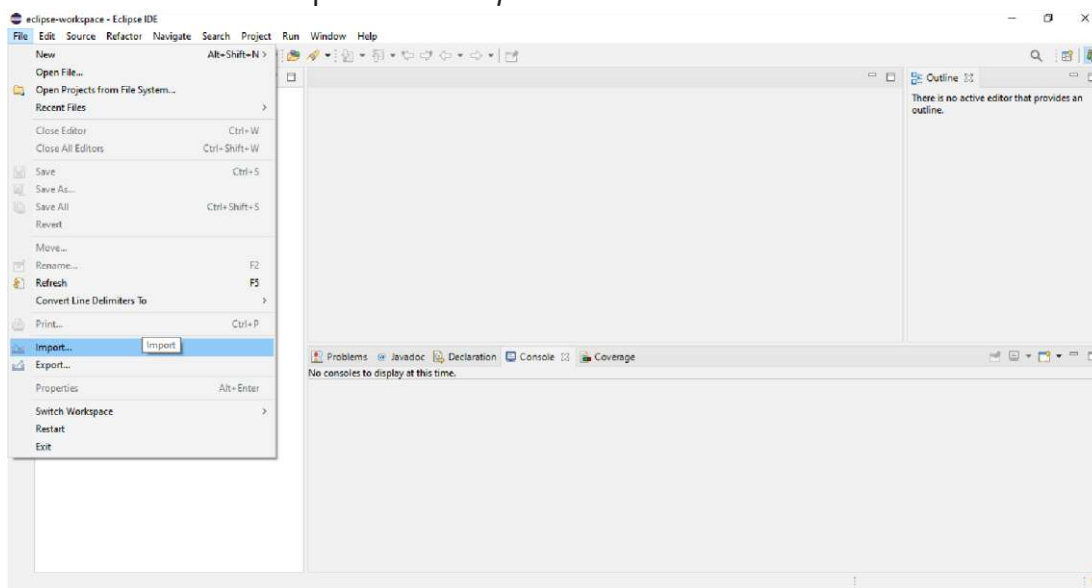
The Eclipse Installer 2022-09 R now includes a JRE for macOS, Windows and Linux.



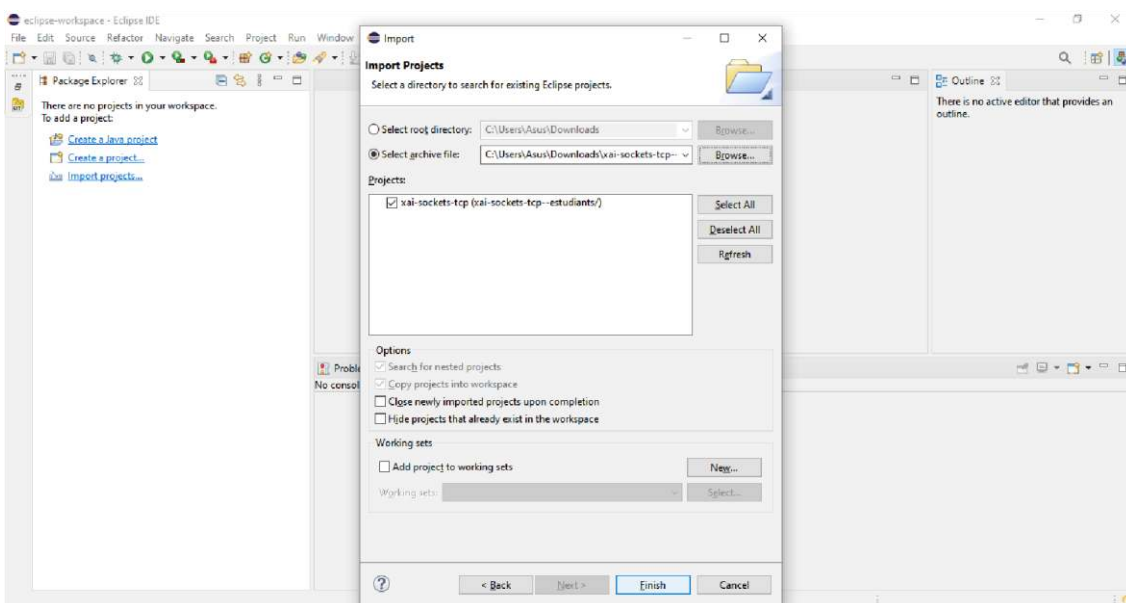
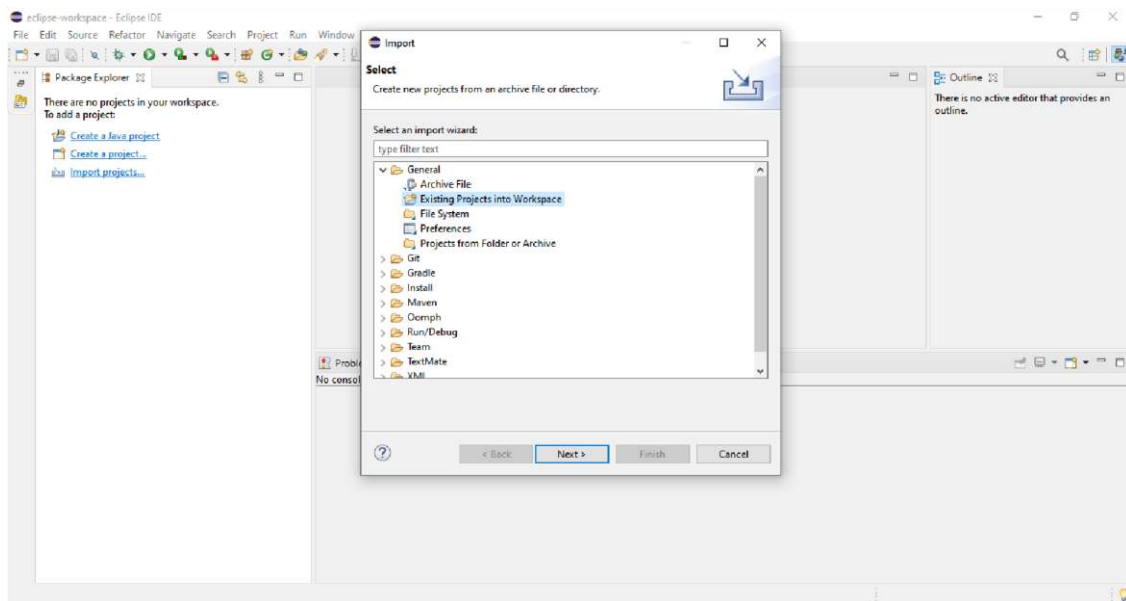
Get Eclipse IDE 2022-09

Install your favorite desktop IDE packages.

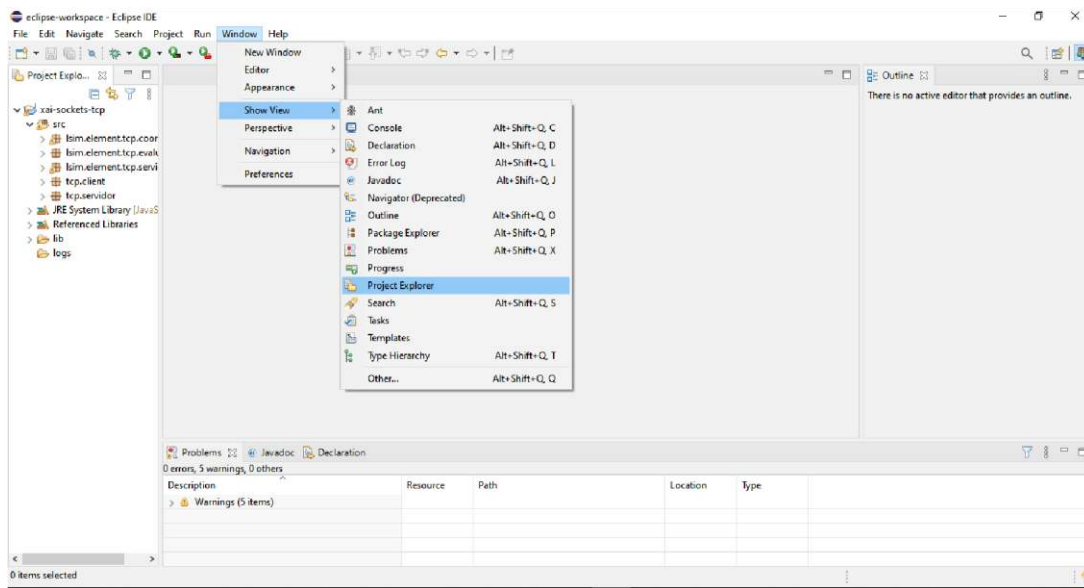
Podéis importar directamente los archivos comprimidos que os damos, directamente desde el archivo zip, para que os coja correctamente las librerías, ya que tienen rutas relativas. Id al menú del Eclipse *File > Import*.



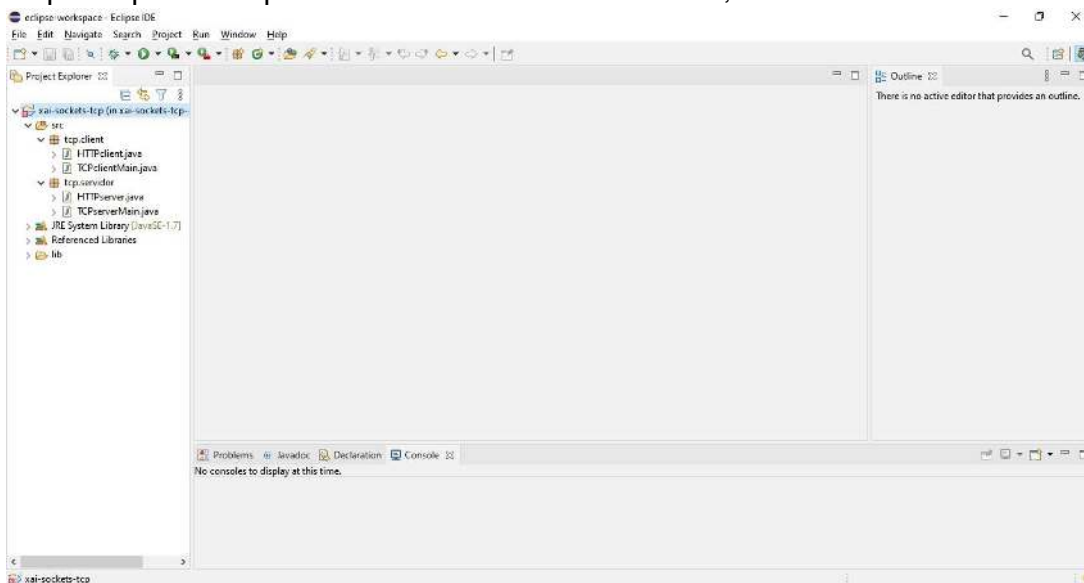
Os aparecerá una ventana emergente donde tenéis que seleccionar el archivo comprimido que os hemos proporcionado, yendo a la opción *Existing projects into workspace > Next > Select Archive File*, tal y como podéis ver en las dos capturas de pantalla que os mostramos a continuación. Tras seleccionar el proyecto a cargar, podéis clicar sobre el botón *Finish*.



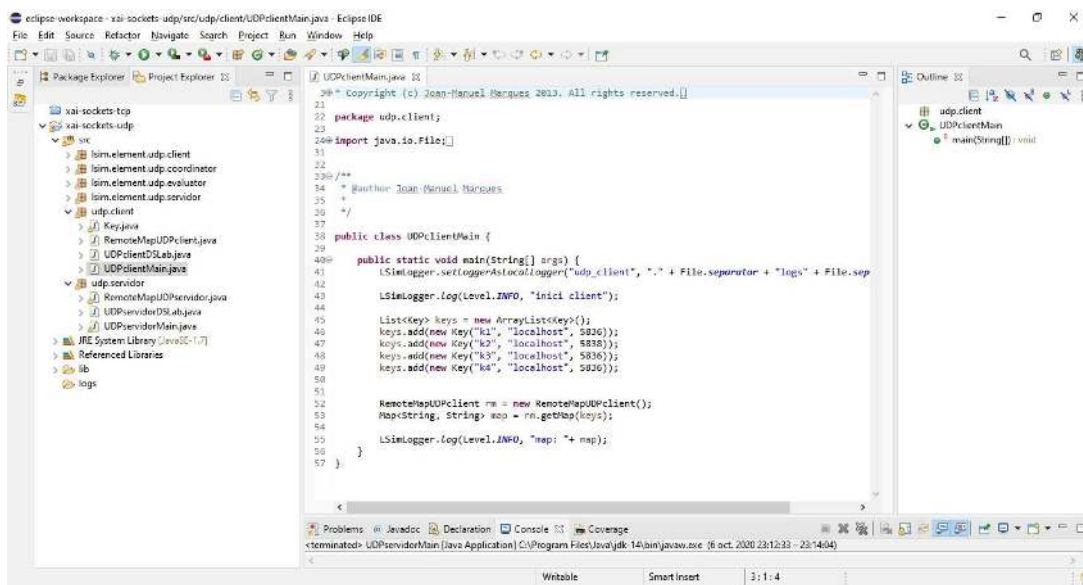
Al aceptar haciendo clic en el botón *Terminar*, os mostrará el explorador con la vista de árbol del proyecto, en la parte izquierda de Eclipse. De lo contrario, podéis forzar esta vista yendo al menú *Ventana > Mostrar vista > Explorar proyecto*. El nombre de las opciones variará en función del idioma de instalación.




Una vez ya veis el árbol del proyecto, podéis navegar por las diferentes clases y bibliotecas que lo componen. En vuestro caso, no tienen porqué aparecer los mismos archivos que la pantalla que se le muestra a continuación; es sólo una referencia.



En cualquier momento, para ver el código de clientes o servidores tenéis que hacer clic sobre la clase en el navegador de la parte izquierda de Eclipse.



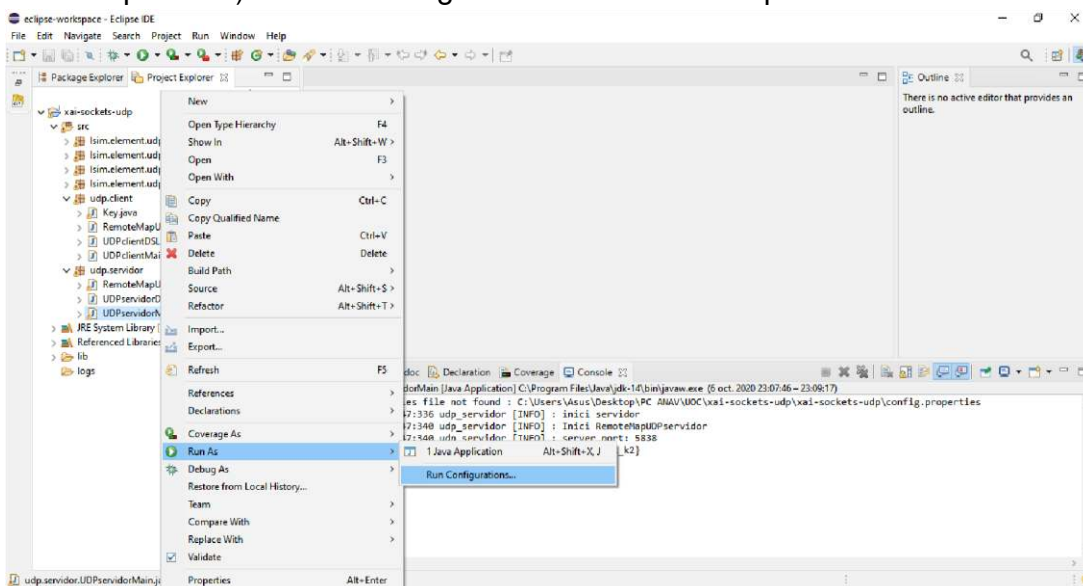
Para ejecutar el proyecto, os tenéis que poner sobre la clase en cuestión, por ejemplo, `TCPclientMain` y pulsar el icono verde de play , situado en la barra de menús superior. También podéis hacerlo pulsando botón derecho sobre la clase que contiene el método `main` dentro del explorador del proyecto, en la parte izquierda de la pantalla. El resultado inicial que se os muestra por consola con el código de partida de la parte 2 tendría que ser similar a la siguiente imagen:

```

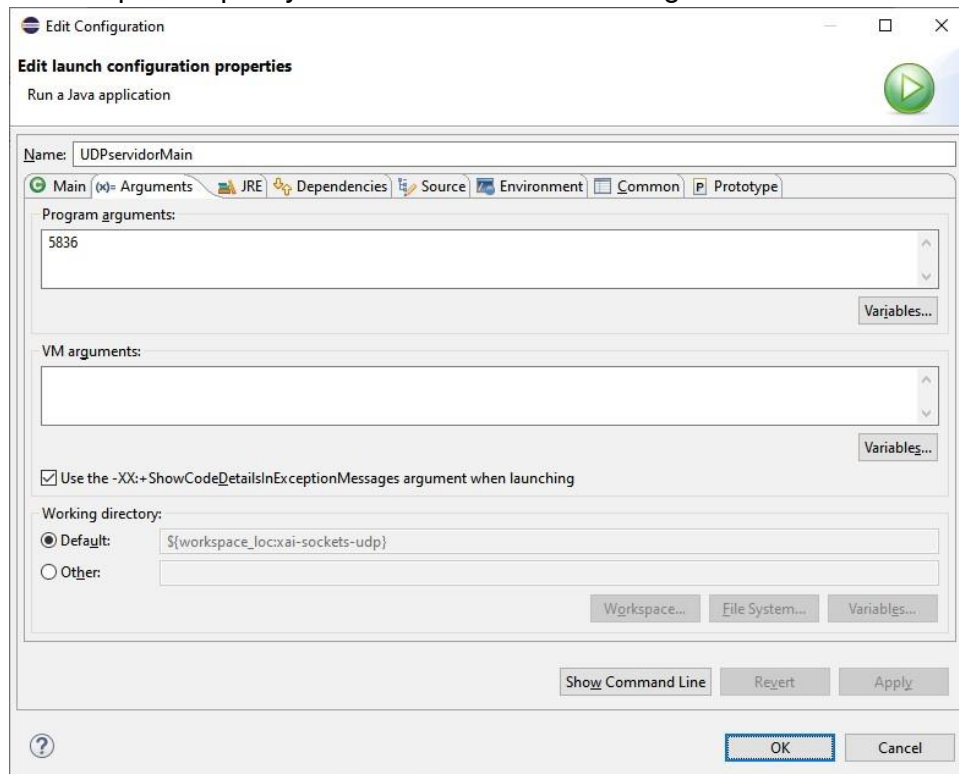
08-10-2020 22:13:44:573 tcp_client [INFO] : inici client http
08-10-2020 22:13:44:584 tcp_client [INFO] : inici HTTPClient.get
08-10-2020 22:13:44:584 tcp_client [INFO] : HTTP server_address: example.org
08-10-2020 22:13:44:586 tcp_client [INFO] : HTTP server_port: 80
08-10-2020 22:13:44:586 tcp_client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

```

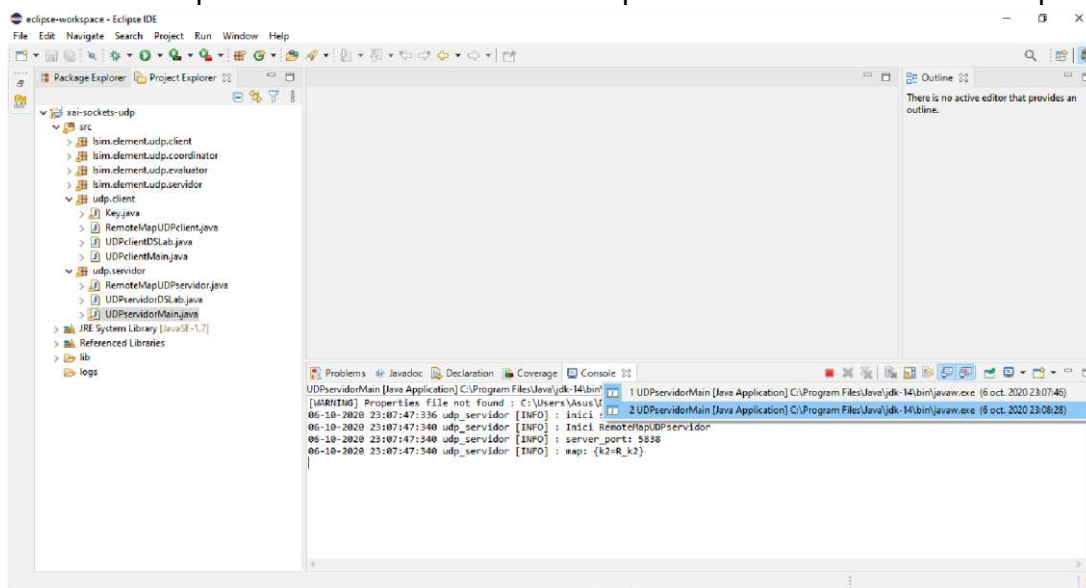
En el caso de la primera parte de la práctica, donde el servidor necesita ejecutarse con el número de puerto pasado como parámetro, tenéis que pulsar botón derecho sobre la clase `UDPServidorMain` de la vista de explorador anterior (árbol situado en la parte izquierda de la pantalla) e ir a la configuración dentro de la opción **Run**.



Como la práctica os pide que haya dos servidores ejecutándose, tendréis que repetir la misma operación dos veces. Aparecerá una pantalla donde tenéis que seleccionar el servidor y ponerle el puerto 5836 en la primera ejecución y 5838 en la segunda. Finalmente tenéis que aceptar y cerrar las ventanas emergentes.



Si deseáis ver diferentes consolas, es decir, los mensajes que cualquiera de los servidores o el cliente os van mostrando por pantalla a lo largo de su ejecución, tenéis que utilizar el botón para cambio de consola en la parte inferior derecha del Eclipse:



El resultado inicial que se os muestra por consola con el código de partida de la parte 1 de los servidores tendría que ser similar a la siguiente imagen:

```
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}
```

Anexo: Instalación Servicio REST en Eclipse (Partes 3 y 4)

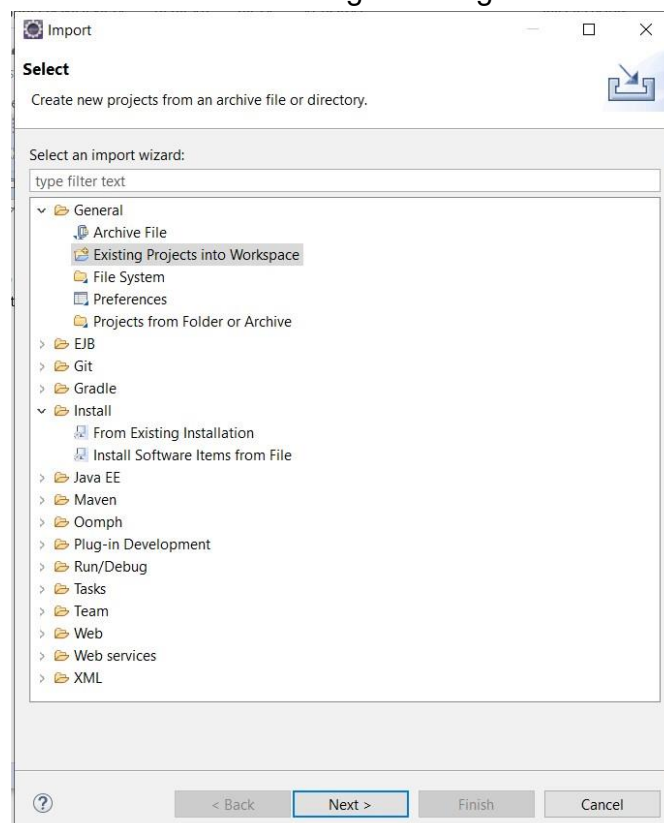
En este anexo se explica la instalación, puesta en marcha y prueba del servicio REST de base que os proporcionamos en esta práctica. Se trata de una calculadora sencilla, que permite realizar sumas, restas y divisiones de dos números enteros. A partir de este servicio, deberéis realizar el que se propone en el enunciado.

Es necesario tener instalado el entorno de programación Eclipse. Podéis descargarlo de <https://www.eclipse.org/downloads/packages/release/2022-12/r/eclipse-ide-enterpris-e-java-and-web-developers>. Si no es la última versión, os dará el enlace al más actual.

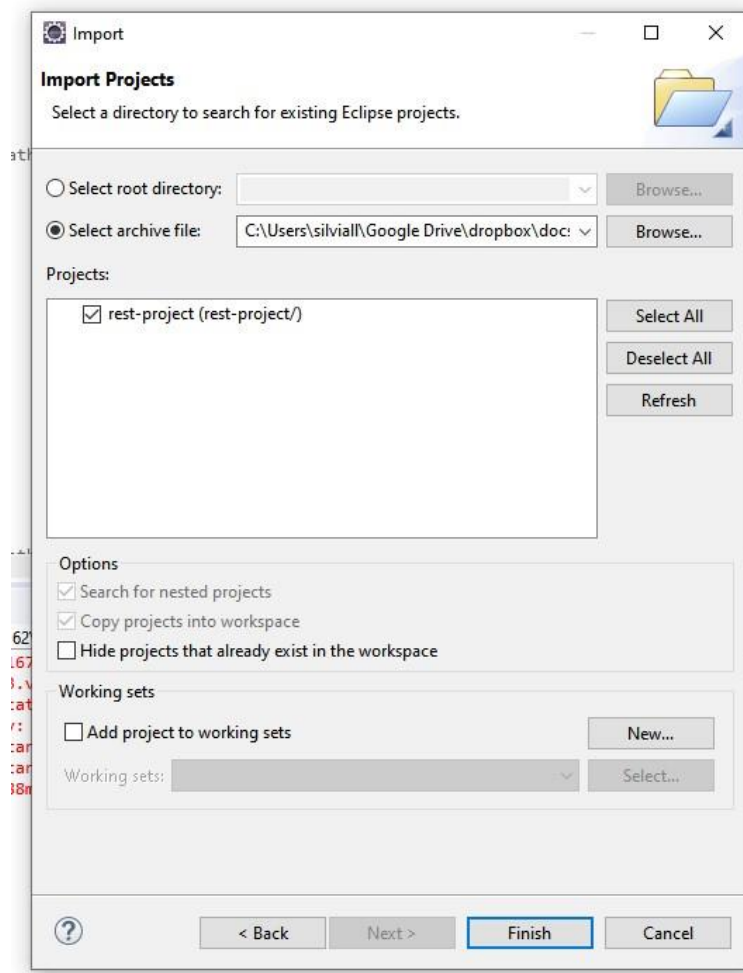
También utilizaremos el servidor Jetty (<https://www.eclipse.org/jetty/>), pero ya está incluido en el proyecto, no es necesario descargar nada.

Una vez instalado Eclipse, se deben realizar los siguientes pasos:

1. Importar proyecto en Eclipse (File > Import > General > Existing projects into Workspace), tal y como se muestra en la siguiente figura.

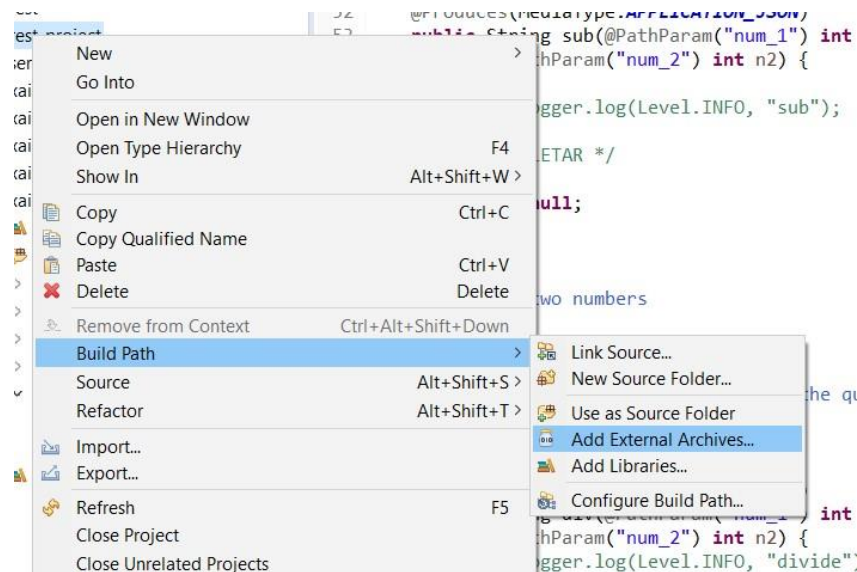


Debéis seleccionar el fichero `xai-rest-base.zip` en la opción **Select Archive File**. Os aparecerá el proyecto (en la figura `rest-project`) y ya podréis pulsar el botón **Finish** para finalizar la importación.

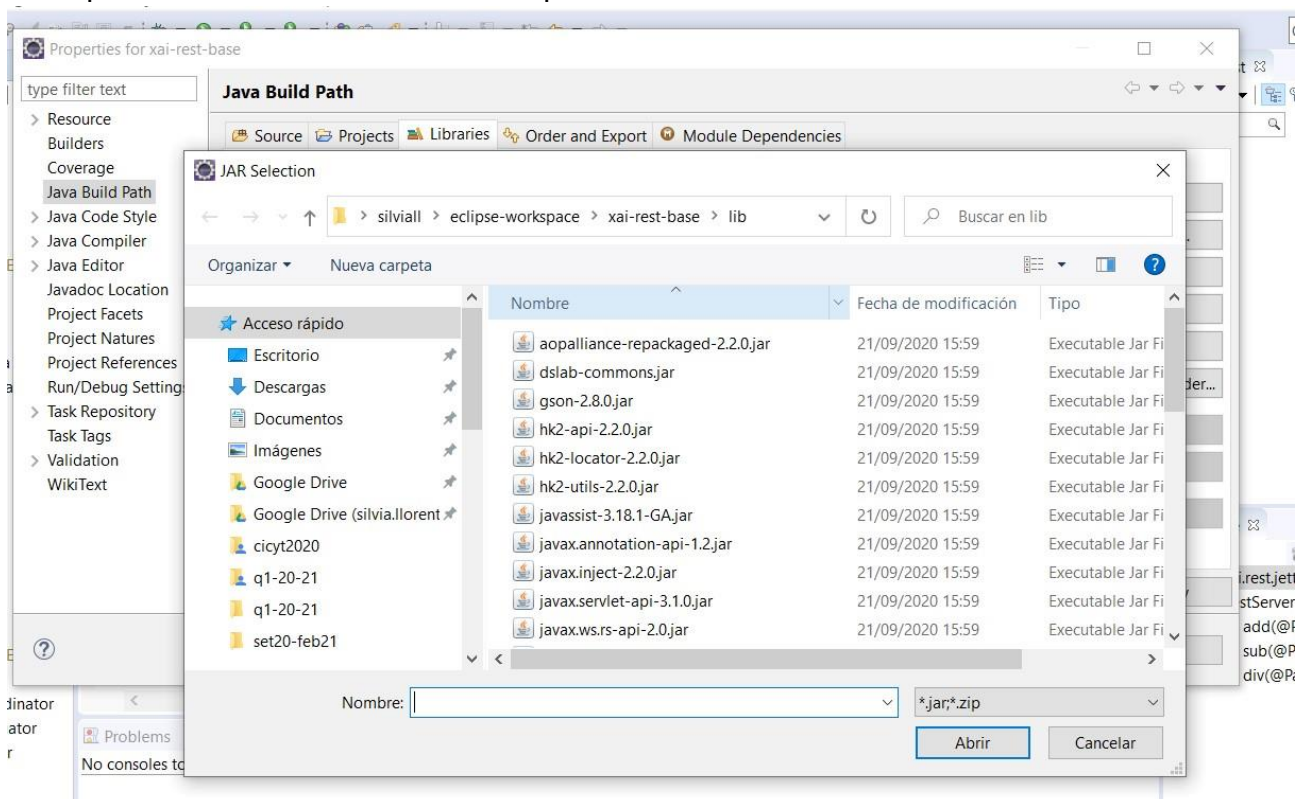


El fichero `.zip` ya tiene todas las librerías necesarias. Si os da errores de dependencias, las tendréis que asociar tal y como explica el paso 2.

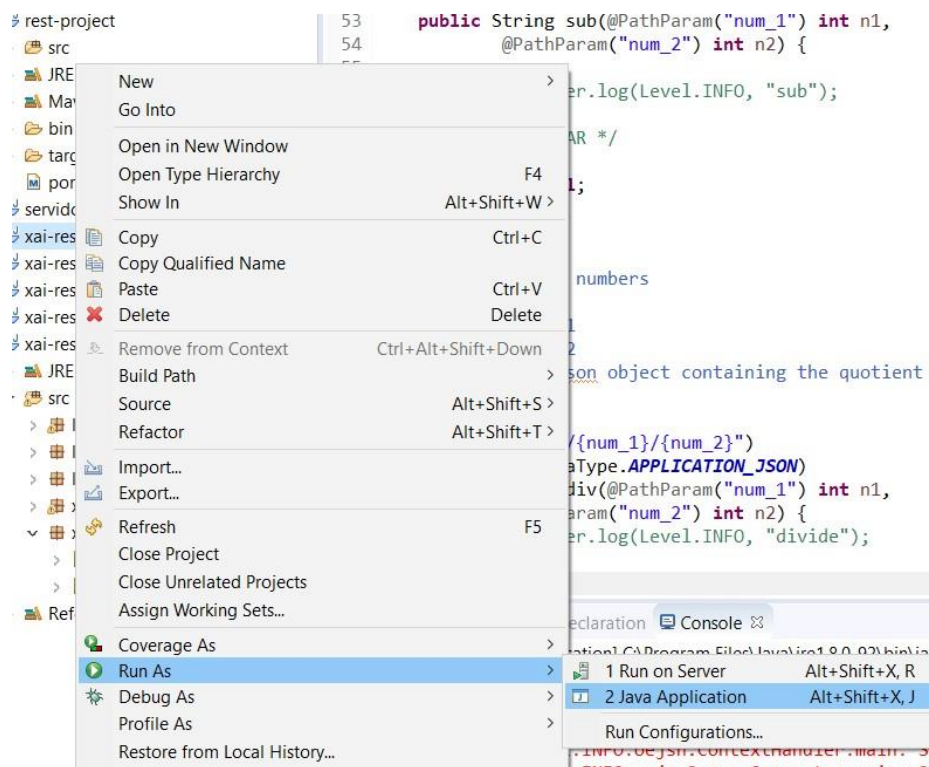
2. Para añadir nuevas librerías hay que modificar el Build Path del proyecto, añadiendo External Libraries, tal y como se ve en las figuras.



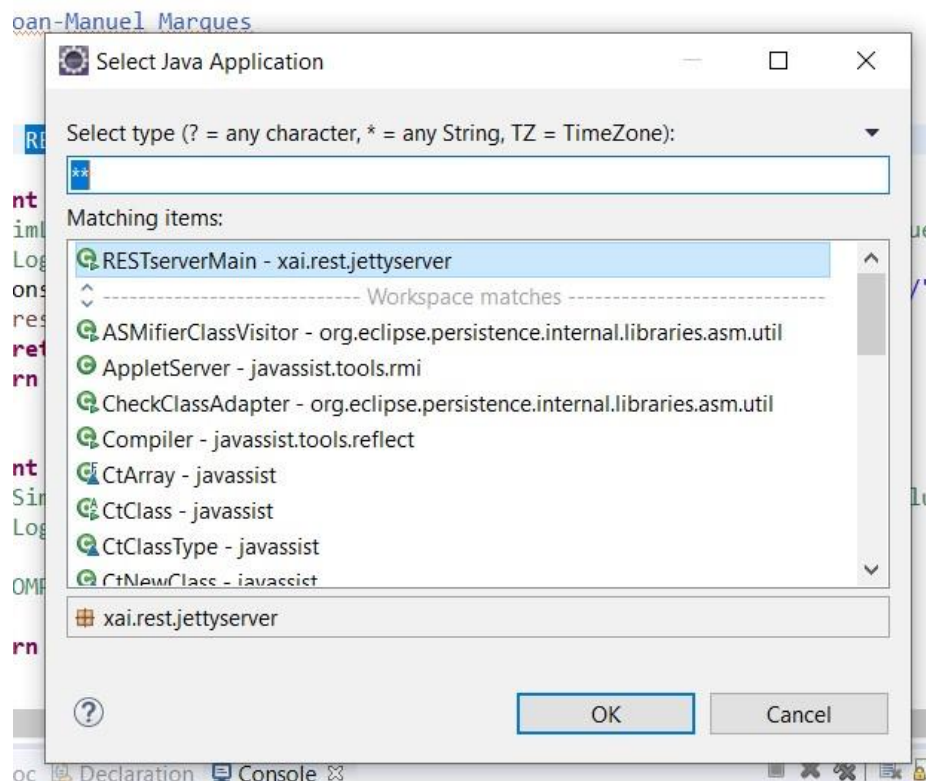
Extrae del zip el directorio lib entero en una carpeta de vuestra máquina (puede ser en el proyecto Eclipse). Después, desde la ventana JAR selection, selecciona todas las librerías que hay en este directorio. Los errores de dependencias deberían desaparecer.



- Una vez instalado y compilado el proyecto, ya se puede ejecutar. Tenéis que seleccionar la opción Java Application. Si os pregunta con qué clase, buscad `RESTserverMain` o `RESTclient`. Primero se tiene que poner en marcha el servidor.



En la siguiente imagen se puede ver la selección de la clase a ejecutar, `RESTserverMain.java`.

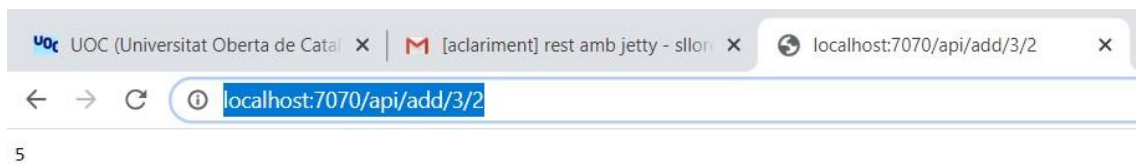


En la consola aparecerán algunos mensajes, indicando que el servidor se ha puesto en marcha.



```
<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}{0.0.0.0:7070}
2020-10-02 14:56:44.980:INFO:oejs.Server:main: Started @1005ms
```

4. Para comprobar su funcionamiento podéis escribir la siguiente URL en un navegador, `http://localhost:7070/api/add/3/2`, tal y como se muestra en la siguiente figura:



También podéis utilizar el comando Linux `curl --request GET --url 'http://localhost:7070/api/add/3/2'` y verificar que la suma de 3 y 2 es igual a 5. También están implementadas las operaciones de resta y división. Podéis revisar el código para ver cómo invocarlas y probar su correcto funcionamiento.