# Exercises: Web Server

Problems for exercises and homework for the [“Java Web Development Basics” course @ SoftUni](https://softuni.bg/courses/java-web-development-basics). Submit your solutions on the **course page** of the **current instance**.

Now that we almost have our final version of the Web Server, or atleast we know what it should look like, we can implement some Framework logic.

## Skeleton

You will be given the **HTTP Server** as a Skeleton, but there will be some things for you to implement. You will be given a RequestHandler interface, which defines logic for **handling requests.**

The RequestHandler interface has **2** methods:

handleRequest():

* **Receives:**
  + an **InputStream** (the **request input data stream**)
  + an **OutputStream** (the **response output data stream**)
* **Returns**
  + **Nothing** (**void**)

hasIntercepted():

* **Receives:**
  + **Nothing**
* **Returns**
  + **Boolean** (**true** / **false**)

## Loading RequestHandlers

Upon initiation, the **Server** should **pre-load** all **RequestHandlers**, which should be **located** in a **folder** named lib, in the **compiled files** of the server(**out / target / etc...**).

You might need some **Reflection** to do that. **Research** on how to load .class files in Java.

## Passing data to the Handler

When a **Socket Connection** is **detected** on the **Server**, the **ConnectionHandler** should **traverse** all the **RequestHandlers**, and **pass** the **Socket**’s **InputDataStream** and **OutputDataStream** to the first **RequestHandler**.

If, by some reason, the RequestHandler **does NOT manage** to **handle** the **Request**, its hasIntercepted() method should **return** false. If the **Request** **is handled**, the method should **return** true.

## Export the “javache” Jar

Export a javache.jar file with the http **package**, the io **package**, and the RequestHandler **interface**, from the **Server** project.

**Research** on how to **export** .jar files in (**IntelliJ** / **Eclipse**).

## Implement a RequestHandler

Create a **separate project** called **“**broccolina**”** which will hold our **dynamic request handlers**. In it, **Implement** a RequestHandler.

* Create a **folder** named lib.
* Put the javache.jar in the lib folder.
* Import the javache.jar from the lib folder in the **Project**.

Create a **class** which **implements** the RequestHandler interface (you **should be able** to **implement** it if the javache.jar was **exported** and **imported** **correctly**). Name the class by your choice.

The **class** should **receive** a **Server root folder path** (as a **String**) in its **constructor**.

## Implementing RequestHandler logic

Implement a **simple test request handling** **logic**, by **sending** a **simple response** (like “Hello World!”).

The **Handler** **receives** an **InputDataStream** and an **OutputDataStream**, so you must use them in order to **handle** a **request** and **send** a **response**.

* To **extract** the **Request content** you should use the Reader from the io package.
* To **send** the **Response content** you should use the Writer from the io package.

You are free to use the HttpRequest and HttpResponse **implementations** from the http package to **ease** your work.

## Polymerization

**Compile** the **Server project** and go into the **compiled files**. Create 2 folders In the root folder – **lib** & **static**.

**Compile** the “broccolina” project and put **its** **compiled files** in the lib folder of the **Server**.

If the **reflection** from Task 2 is implemented **correctly** you should **load** the **compiled Handler** from the lib folder.

## Run the Server

**Run** the **Server** from a **bash console**, or **create** a **batch file** to compile StartUp.class for you. If everything is **implemented correctly**, upon a **request** to the **Server**, you should see the **content** you’ve put in the RequestHandler.