Servlet for Json or CSV (or XML)

A servlet serving either Json or CSV (or XML) based on GET parameter

A Servlet used as an API for data

Let’s say we want to write a Servlet serving as data provider. The servlet should give us data in either Json format or some other format.

How do we tell the servlet what format we want?

How does the servlet discover our preference?

Example - A servlet for either Json or CSV

This is what we'd like:

If we send along a parameter `format` with the value `json` the servlet should give us some data formatted as Json. If we give it instead the value `csv` we should get the same data formatted as CSV (comma-separated values).

```
$ lwp-request -m GET http://localhost:8080/dataservlet?format=json
{"firstName":"Hanky","lastName":"Sandycleavage","age":65,"streetAddress":"Skidrow
88","State":"VGR","postalCode":"66613","phoneNumbers":[{"Mobile":"0700123321"},{"Home":"031-9
0 51 06"}]}
$ lwp-request -m GET http://localhost:8080/dataservlet?format=csv
firstName,lastName,age,streetAddress,State,postalCode,Mobile,Home
Hanky,Sandycleavage,65,Skidrow 88,VGR,66613,0700123321,031-90 51 06
```

What was that parameter thingy?

```
$ lwp-request -m GET http://localhost:8080/dataservlet?format=json
```

The part of a URL which comes after the ? are called GET parameters.

You can have a whole lot of GET parameters, separated by &:

```
$ lwp-request -m GET http://localhost:8080/dataservlet?format=json&lang=sw&id=4&day=Monday
```

Next, we'll look at how the Servlet code reads and uses a GET parameter.

Reading a GET parameter

In the doGET() method:

```
String format = request.getParameter("format");
if (format != null && format.equals("json")) {
  response.setContentType("application/json");
  writeJSON(page);
} else if....
```

Now, the client chooses output format

When the servlet uses the format parameter, it allows the client (requestor) to choose the output format.

We could now add support also for XML and more formats.

Some words of warning

The way we write the servlet now defines the API for our service.

With the format parameter and support for json, xml and csv, we must document the API accordingly so that people know how to use it.

If the format parameter has an unknown value (not one of the above), the servlet should handle this in a documented way.

If the format parameter is mandatory but missing, the servlet should handle this too, in a documented way.

Handling malformed requests

```
$ lwp-request -m GET http://localhost:8080/dataservlet?format=docx
<html><body><h1>Unknown format: docx</h1></body></html>

$ lwp-request -m GET http://localhost:8080/dataservlet?give_me_json
<html><body><h1>Missing parameter: format</h1></body></html>
```

Ideally, we should also give appropriate HTTP headers so that clients “know” whether the request resulted in a normal behavior or an error.

We’ll look into HTTP response codes in a later lecture.

Code snippets

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
  StringBuilder page = new StringBuilder(253);
  String format = request.getParameter("format");
  if (format != null && format.equals("json")) {
    response.setContentType("application/json");
    writeJSON(page); // give a reference to the StringBuilder as argument
  } else if(format != null && format.equals("csv")) {
    response.setContentType("text/csv");
    writeCSV(page); // give a reference to the StringBuilder as argument
  } else if(format == null) {
    response.setContentType("text/html");
    page.append("<html><body><h1>Missing parameter: format</h1></body></html>");
  } else {
    response.setContentType("text/html");
    page.append("<html><body><h1>Unknown format: "+format+"</h1></body></html>");
  }
  PrintWriter out = response.getWriter();
  out.println(page);
  out.flush();
}
```

Hardcoded dummy methods

```
private void writeJSON(StringBuilder page) {
  StringWriter writer = new StringWriter();
  JsonWriter jWriter = Json.createWriter(writer);
  JsonObject jo = Json.createObjectBuilder()
    .add("firstName","Hanky")
    .add("lastName","Sandycleavage")
    .add("age",65)
    .add("streetAddress","Skidrow 88")
    .add("State","VGR")
    .add("postalCode","66613")
    .add("phoneNumbers", Json.createArrayBuilder()
         .add(Json.createObjectBuilder()
              .add("Mobile", "0700123321"))
         .add(Json.createObjectBuilder()
              .add("Home", "031-90 51 06")))
    .build();
  jWriter.writeObject(jo);
  jWriter.close();
  page.append(writer.toString());
}
```

Hardcoded dummy methods

```
  private void writeCSV(StringBuilder page) {
    page.append("firstName,lastName,age,streetAddress,State,postalCode,Mobile,Home\n");
    page.append("Hanky,Sandycleavage,65,Skidrow 88,VGR,66613,0700123321,031-90 51 06");
  }

// In reality, you'd use a database or similar to get the actual data!
// The above is just a dummy example
```

web.xml

```
<?xml version="1.0" encoding="utf-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<servlet>
  <servlet-name>dataservlet</servlet-name>
  <servlet-class>student.servlets.DataServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>dataservlet</servlet-name>
   <url-pattern>/dataservlet</url-pattern>
</servlet-mapping>
</web-app>
```

Directory layout

```
.
├── webroot
│   └── WEB-INF
│       ├── classes
│       │   └── student
│       │       └── servlets
│       │           └── DataServlet.java
│       ├── lib
│       │   └── javax.json.jar
│       └── web.xml
└── jenkins-winstone.jar  # you might have a longer name for this JAR file

# Example build and run relative to “.”:
$ javac -cp jenkins-winstone.jar:webroot/WEB-INF/lib/javax.json.jar webroot/WEB-INF/classes/student/servlets/DataServlet.java

# you need both jenkins-winstone.jar (for the javax.servlet classes) and javax.json.jar on the classpath (for json)

$ java  -jar jenkins-winstone.jar --webroot=webroot

# Note: it is common to use separate directories for source code and class files
# but we skipped that for simplicity
```

We could now add support for XML

```
$ lwp-request -m GET http://localhost:8080/dataservlet?format=xml
<?xml version="1.0" encoding="utf-8"?>
<address>
 <first-name>Hanky</first-name>
 <last-name>Sandycleavage</last-name>
 <age>65</age>
 <street-address>Skidrow 88</street-address>
 <state>VGR</state>
 <phone-numbers>
  <mobile>0700123321</mobile>
  <home>031-90 51 06</home>
 </phone-numbers>
</address>
```

We leave the implementation as a challenge

You can implement xml support using your knowledge on how to create XML from Java (as seen in a previous lecture).

You can start with adding hard-coded XML in a dummy method, to check that the parameter works.

You'd need an additional jar-file if you'd want to programmatically create the XML using e.g. javax.xml etc.

Further reading

- http://docs.oracle.com/javaee/6/tutorial/doc/bnafv.html#bnafw
- https://examples.javacodegeeks.com/enterprise-java/servlet/get-request-parameter-in-servlet/