

10th International IT-seminar



Madrid – Spain

Web services

- Classic (SOAP)
- Data protocol
 - SOAP
- Transport protocol
 - Not specified
 - Usually HTTP
- Restful
- Data protocol
 - Not specified
 - Usually JSON
 - or XML
- Transport protocol
 - HTTP

Representational State Transfer

- Homepages for machines?
- Another way to make remote procedure calls?
- Semireligious paradigm?

Architectural constraints

*If a service violates any of the required constraints,
it cannot be considered RESTful*

- Client–server
- Stateless
- Cacheable
- Layered system
- Code on demand (optional)
- **Uniform interface**

HTTP Request

```
GET /index.html HTTP/1.1
Host: localhost:4711
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml...
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) ...
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,da;q=0.6
```

HTTP Response

HTTP/1.1 200 OK

Content-Type: text/html; charset=utf-8

Content-Length: 156

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <div>Hello Fine World!</div>
  </body>
</html>
```

HTTP POST Request

```
POST /person HTTP/1.1
Host: localhost:4711
Connection: keep-alive
Content-Length: 50
Cache-Control: no-cache
Content-Type: application/json...
```

```
{
  "id": 8,
  "name": "Jens",
  "age": 25
}
```

HTTP Error Response

HTTP/1.1 404 Not Found

Content-Type: text/plain; charset=utf-8

Content-Length: 0

HTTP Methods

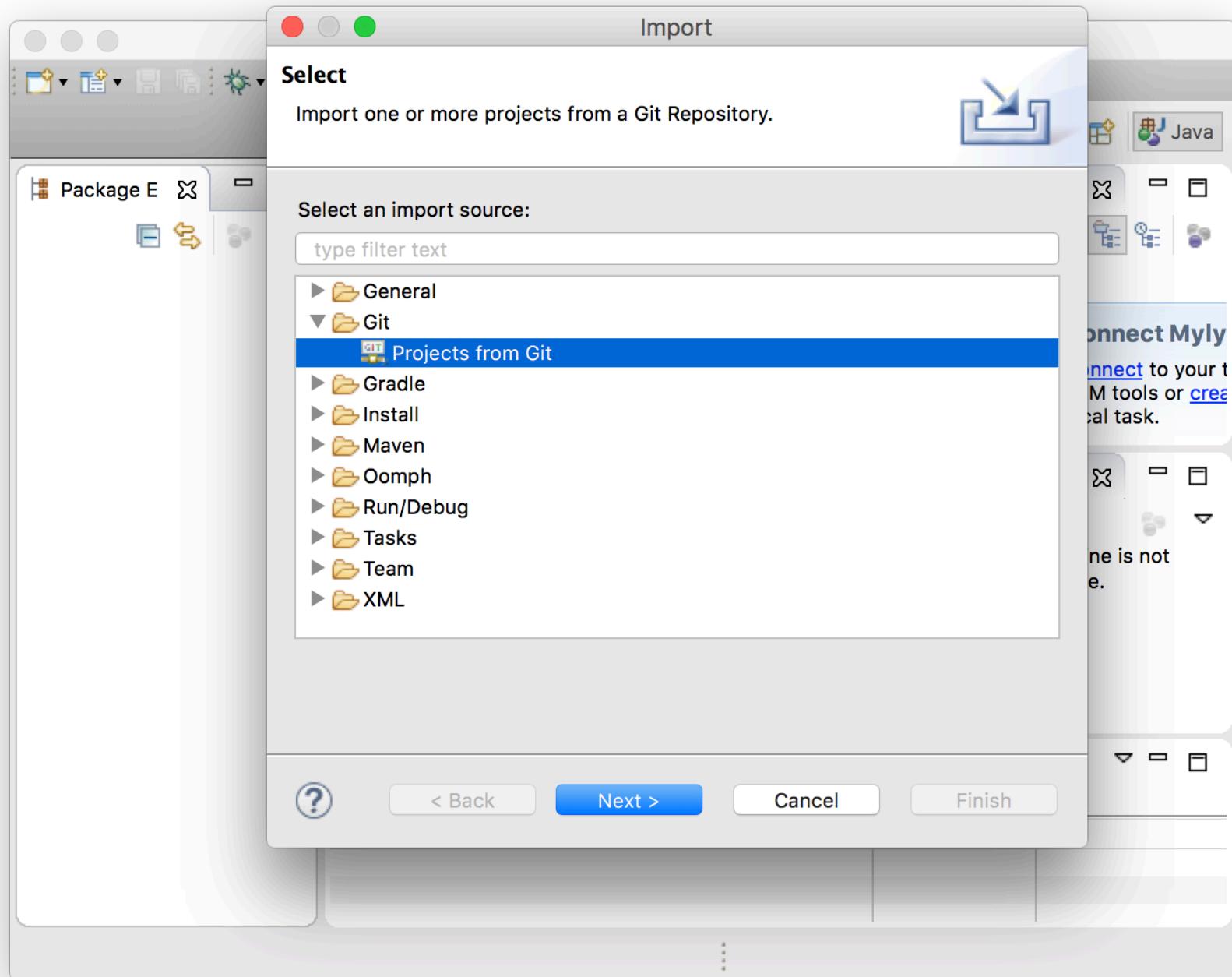
- GET
 - select
- POST
 - update or insert
- DELETE
 - delete
- and many more

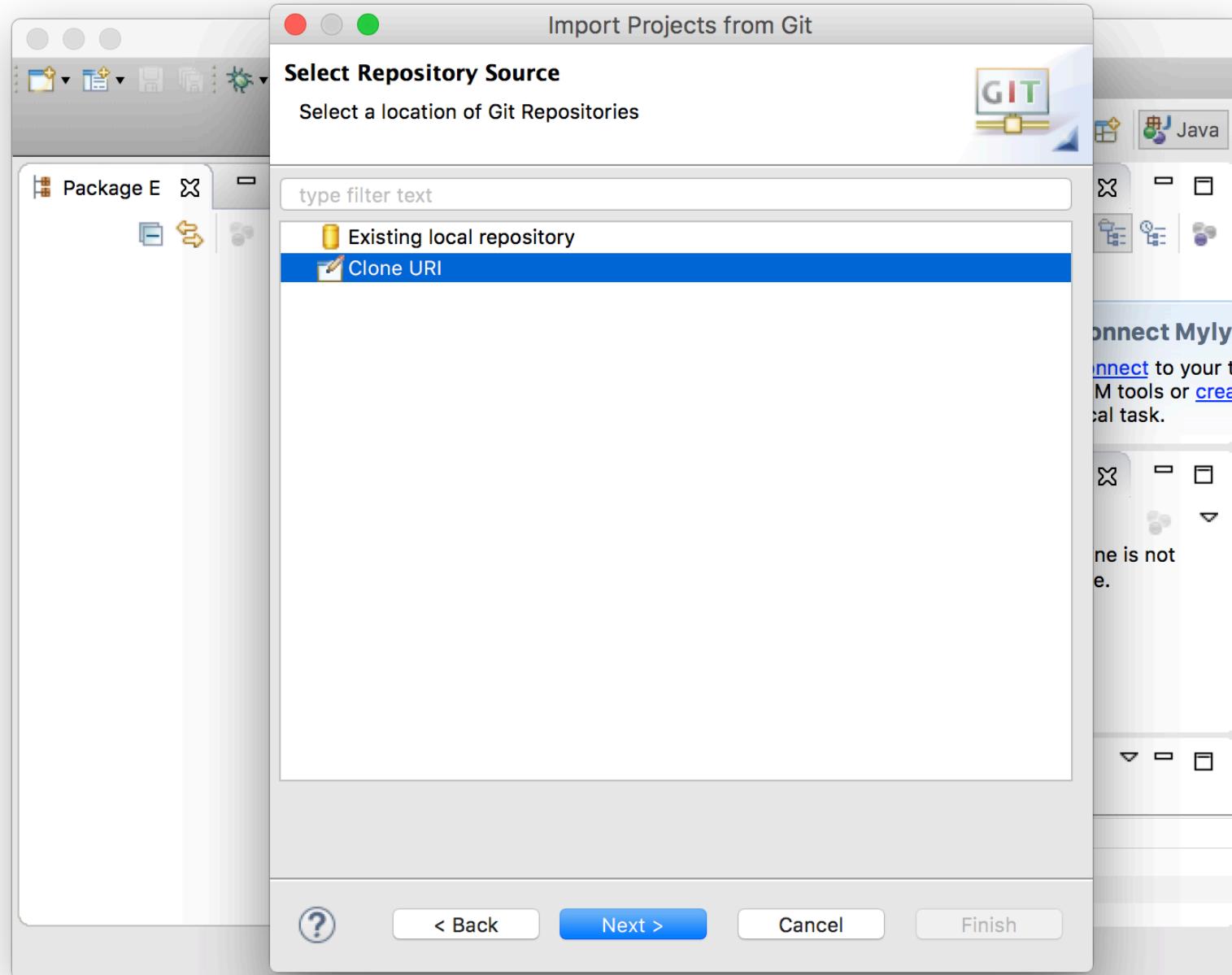
Javascript Simple Object Notation

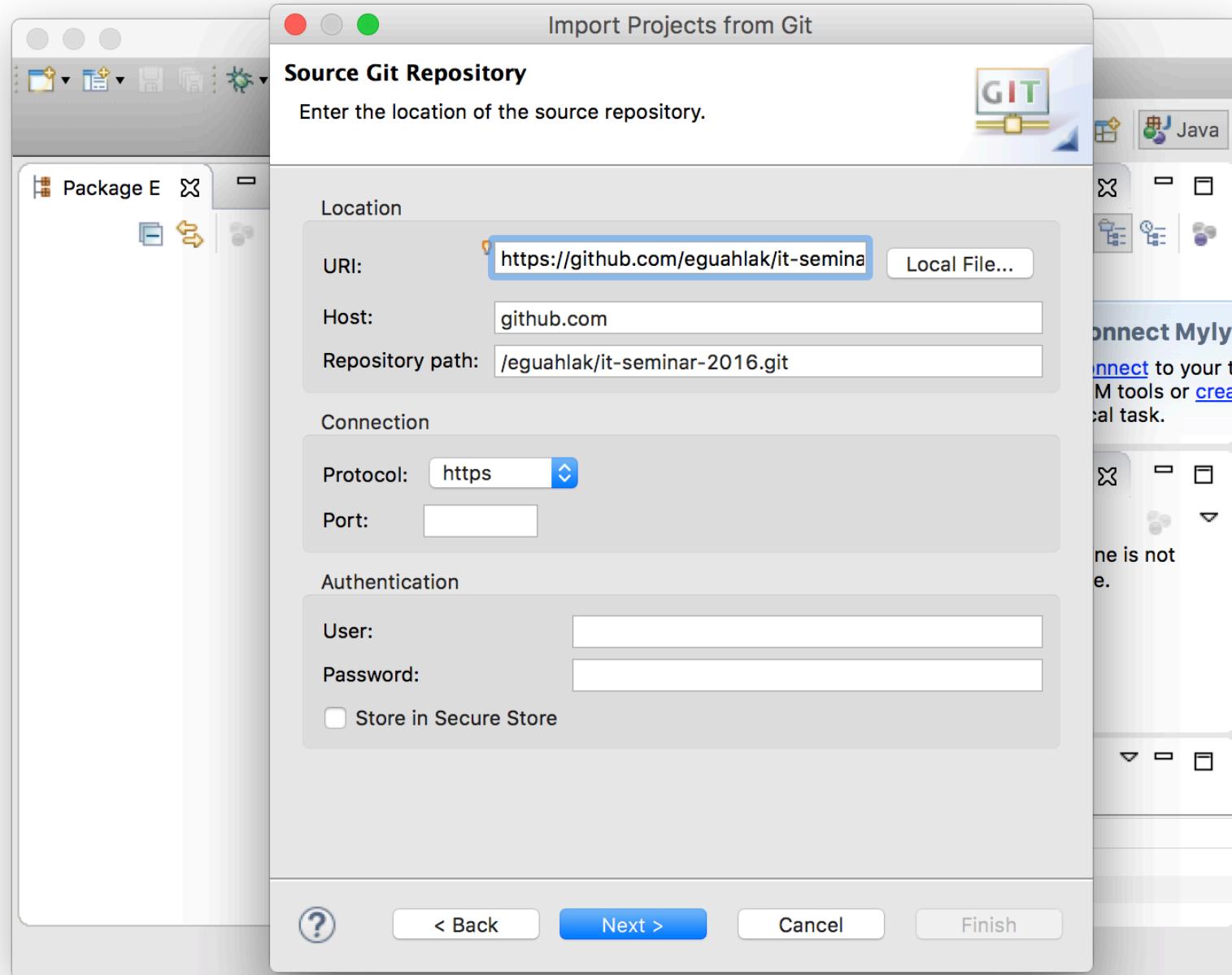
- GSON library
- JSONParser.java
- Strings “In double quotes”
- Numbers 7, 67.5
- Lists [7, 9, 13]
- Objects { “key”: “value”, “anotherKey” : 42 }
- Any combinatione hereof
- [{"id":1,"name":"Kurt","age":34}, {"id":2, "name":"Sonja", "age":25}, {"id":3, "name": "Ib", "age":67}]

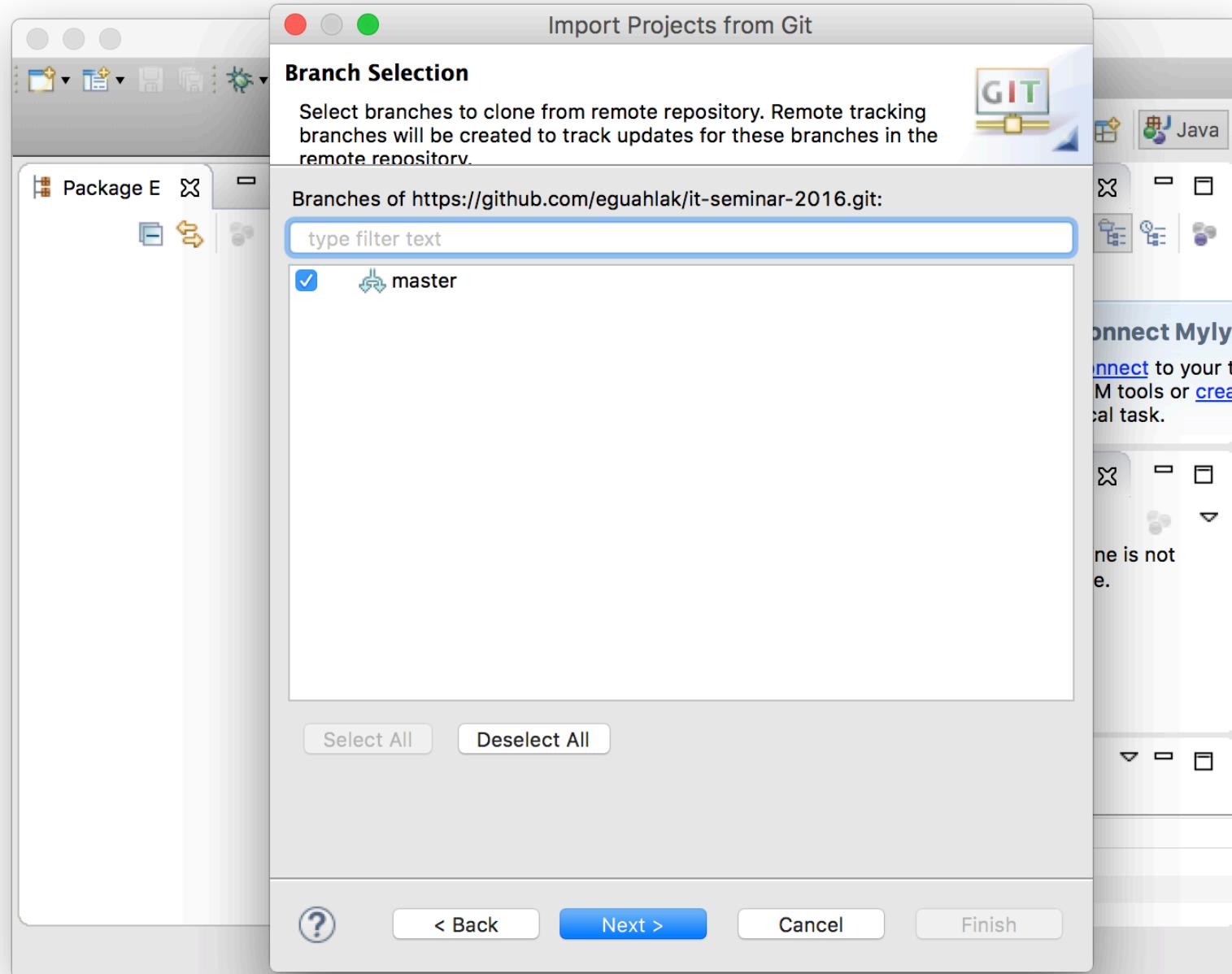
Installing boilerplate code with

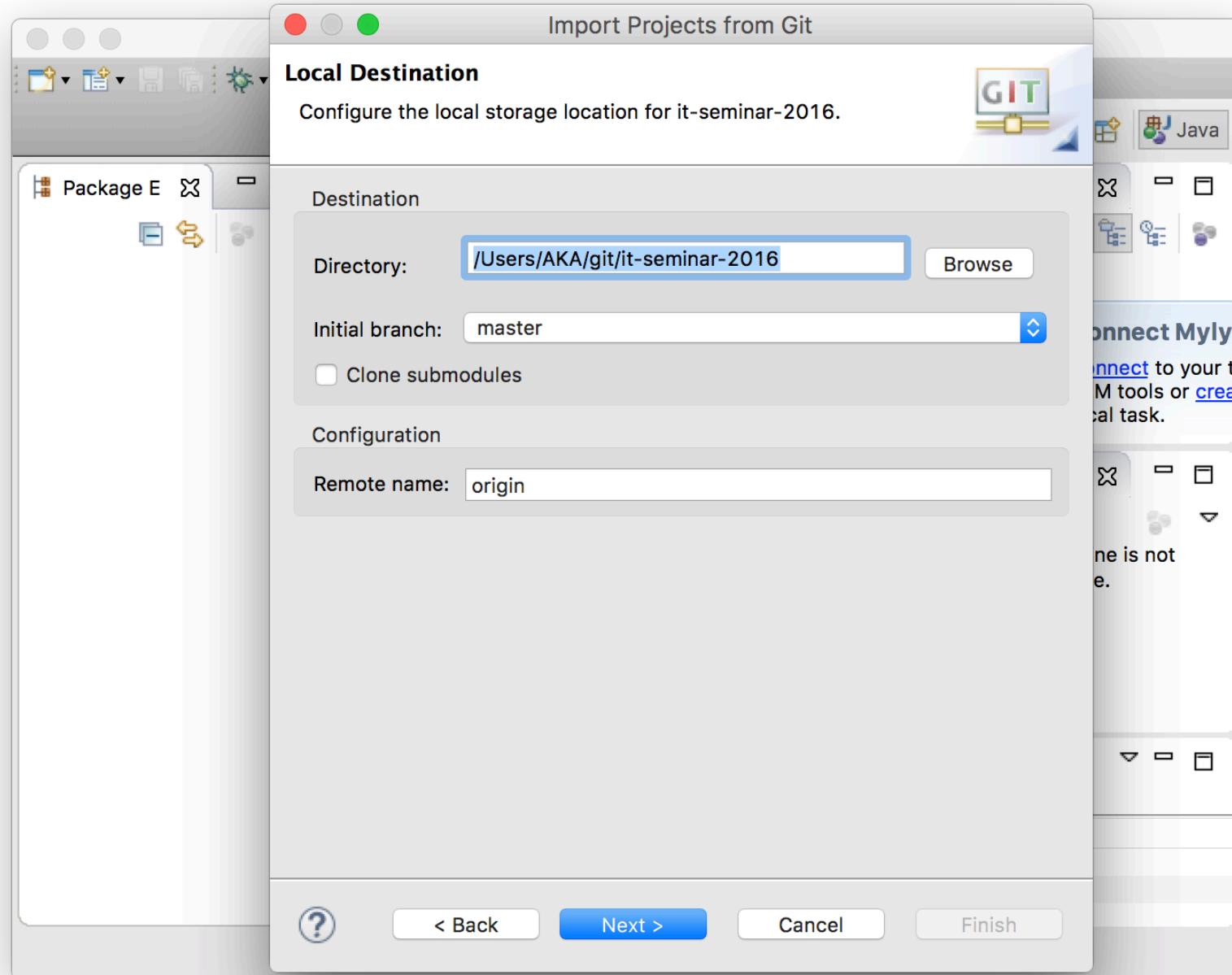


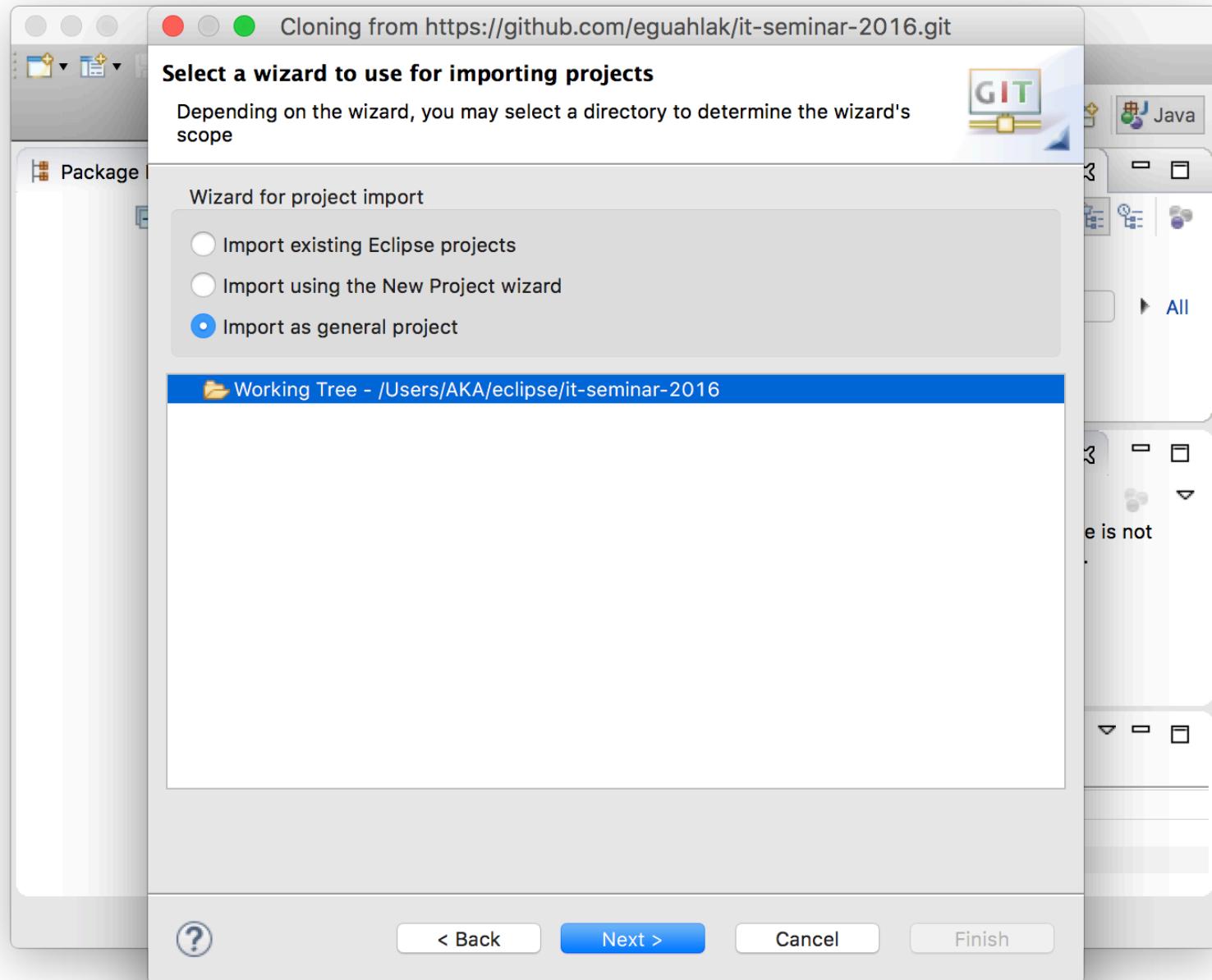


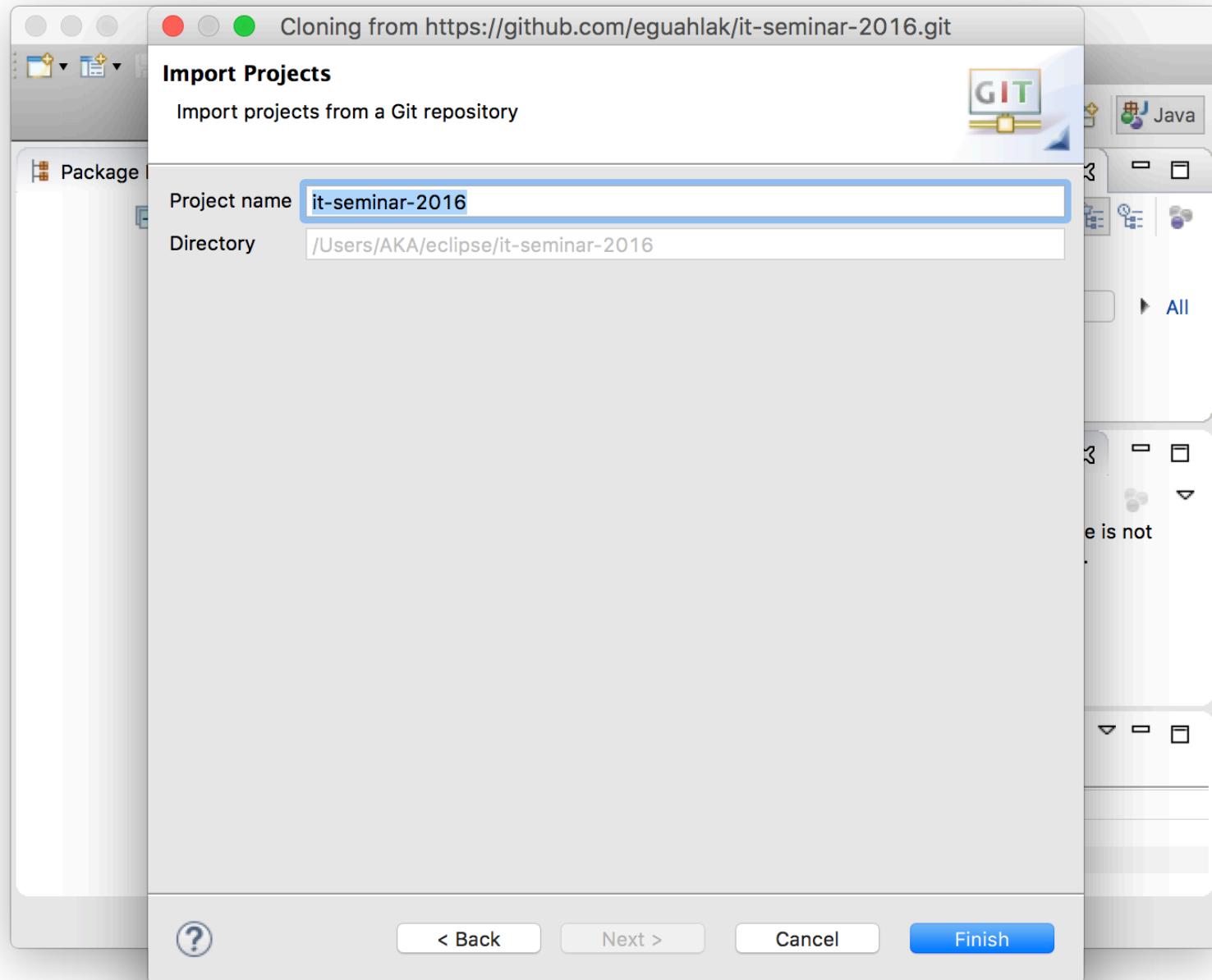


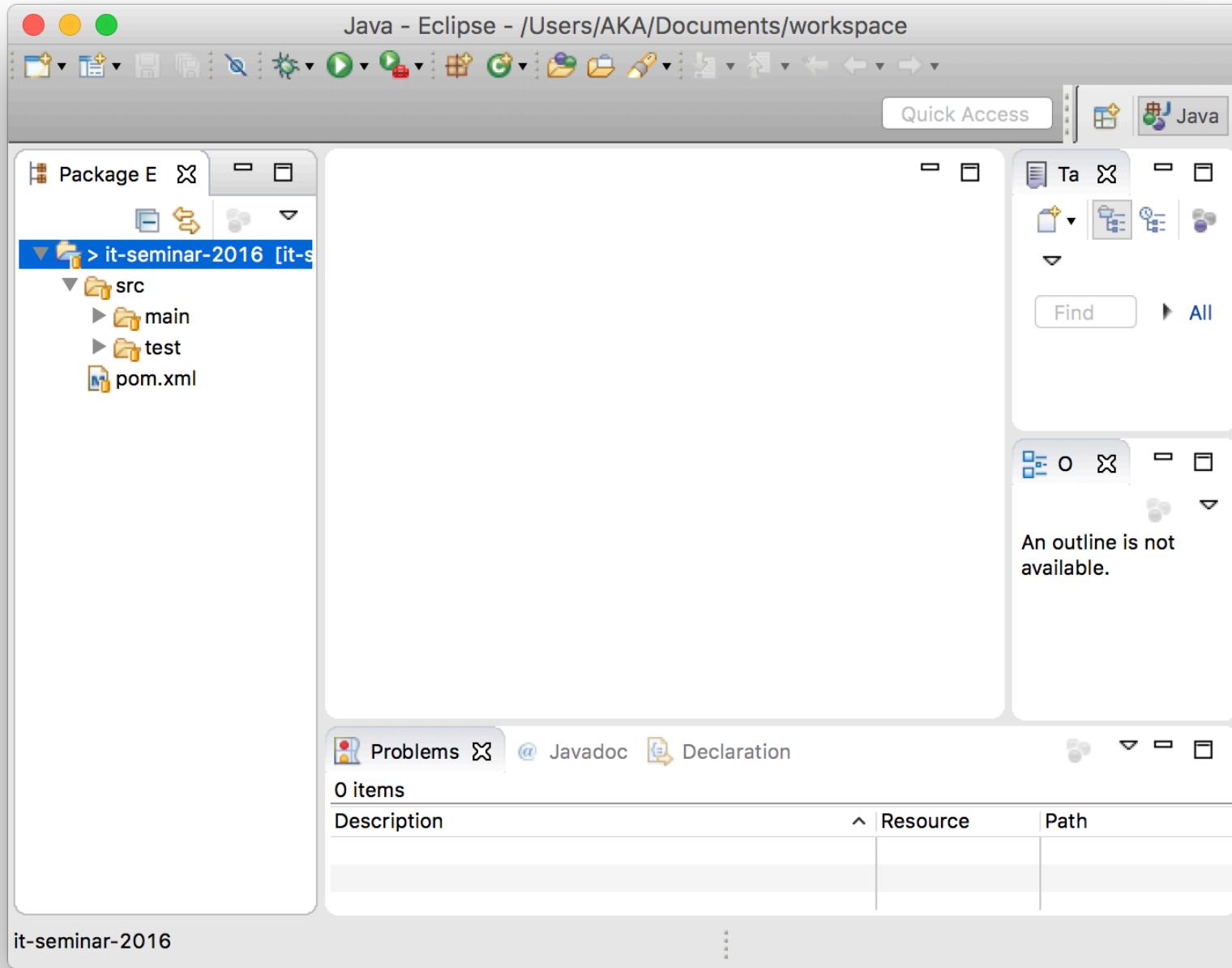






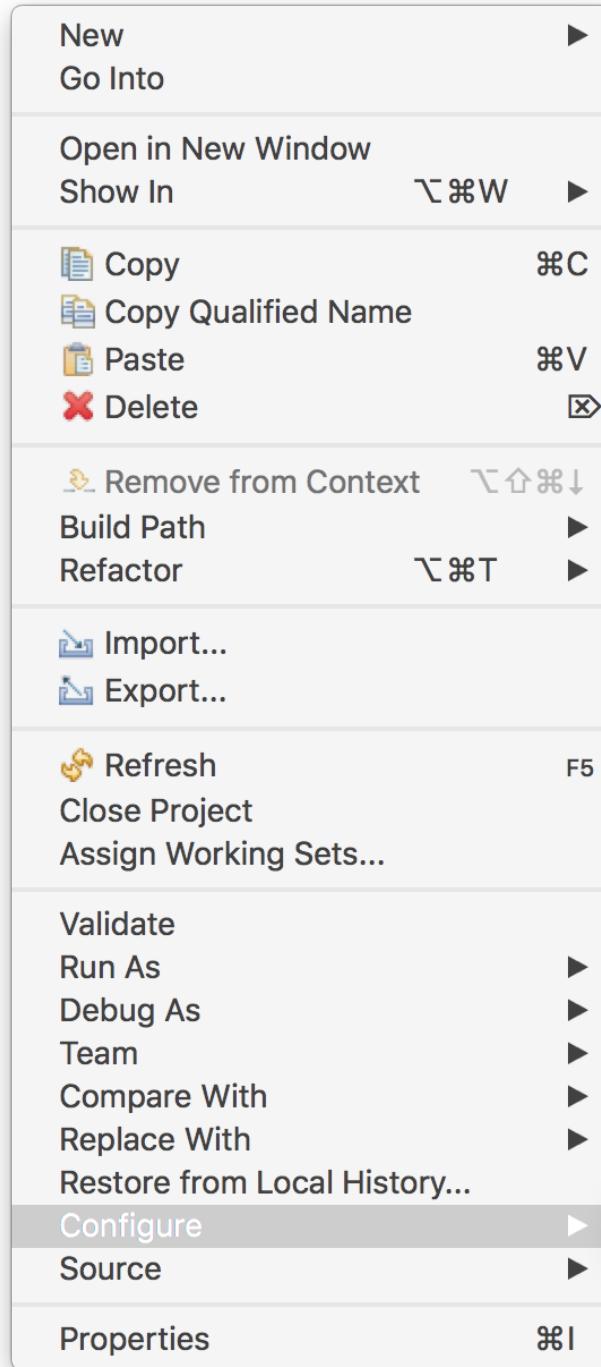




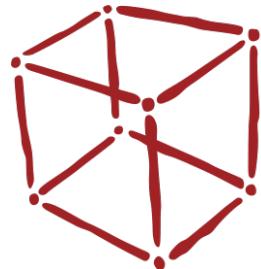


And run it

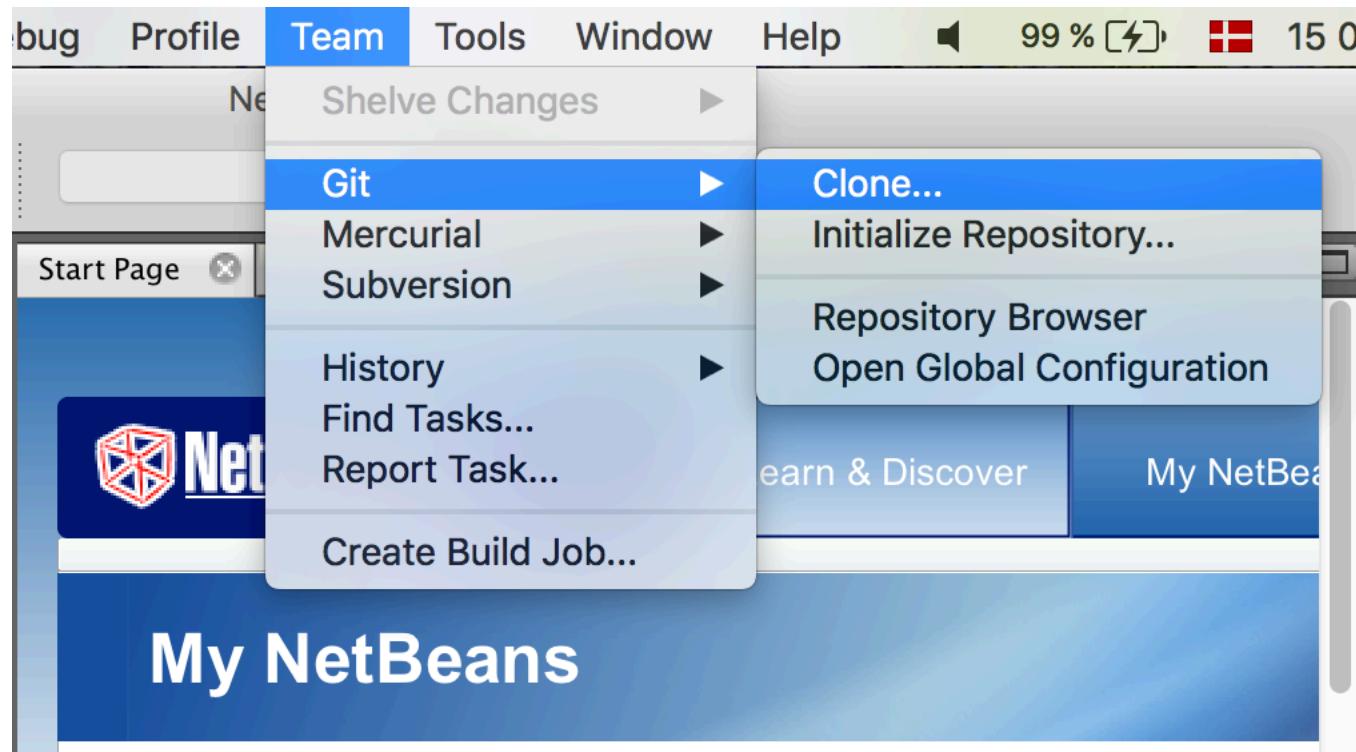
- Right click project
 - Configure >
 - Convert to Maven Project
- Right click SimpleServer.java
 - Run as >
 - 1. Java Application



Installing boilerplate code with



NetBeans



Clone Repository

Steps

1. **Remote Repository**
2. Remote Branches
3. Destination Directory

Remote Repository

Specify Git Repository Location:

Repository URL:

`http[s]://host.xz[:port]/path/to/repo.git/`

User: (leave blank for anonymous access)Password: Save Password**Proxy Configuration...**

Specify Destination Folder:

Clone into: /it-seminar-2016

INS

Clone Repository

Steps

1. Remote Repository
2. **Remote Branches**
3. Destination Directory

Remote Branches

Select Remote Branches

 master*

INS

Clone Repository

Steps

1. Remote Repository
2. Remote Branches
3. **Destination Directory**

Destination Directory

Specify the Parent Directory and Name for this Clone

Parent Directory: /Users/AKA/netbeans

Clone Name: it-seminar-2016

Checkout Branch: master*

Remote Name: origin

Scan for NetBeans Projects after Clone

Help

< Back

Next >

Finish

Cancel

INS

NetBeans IDE 8.1

Project... x Files Services x Start Page x Object.java x

Clone Completed

/Users/AKA/netbeans/it-seminar-2016 project was cloned.
Do you want to open the project?

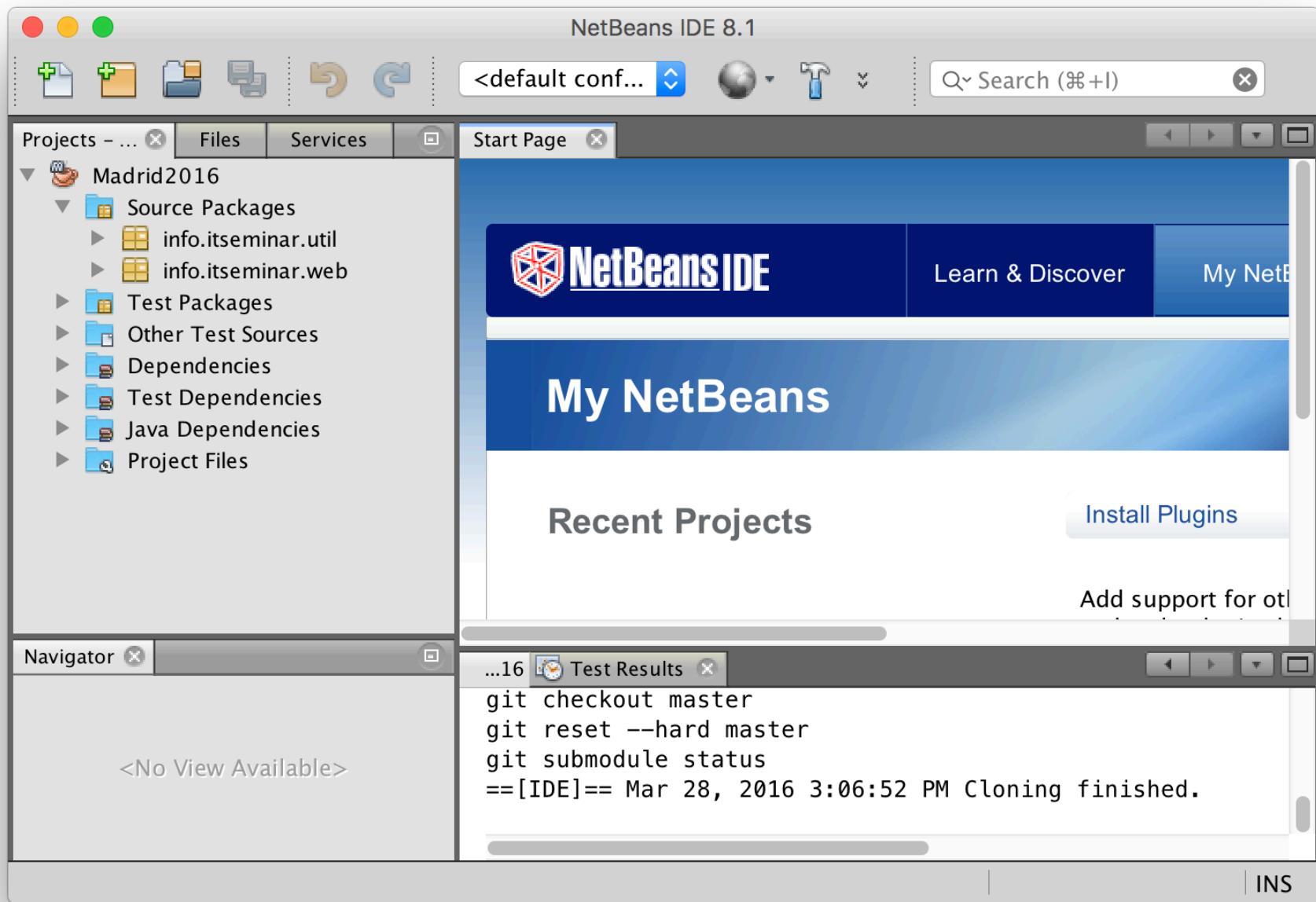
Open Sources in Favorites

Open Project **Close**

...16 Test Results x

```
git checkout master  
git reset --hard master  
git submodule status  
==[IDE]== Mar 28, 2016 3:06:52 PM Cloning finished.
```

INS

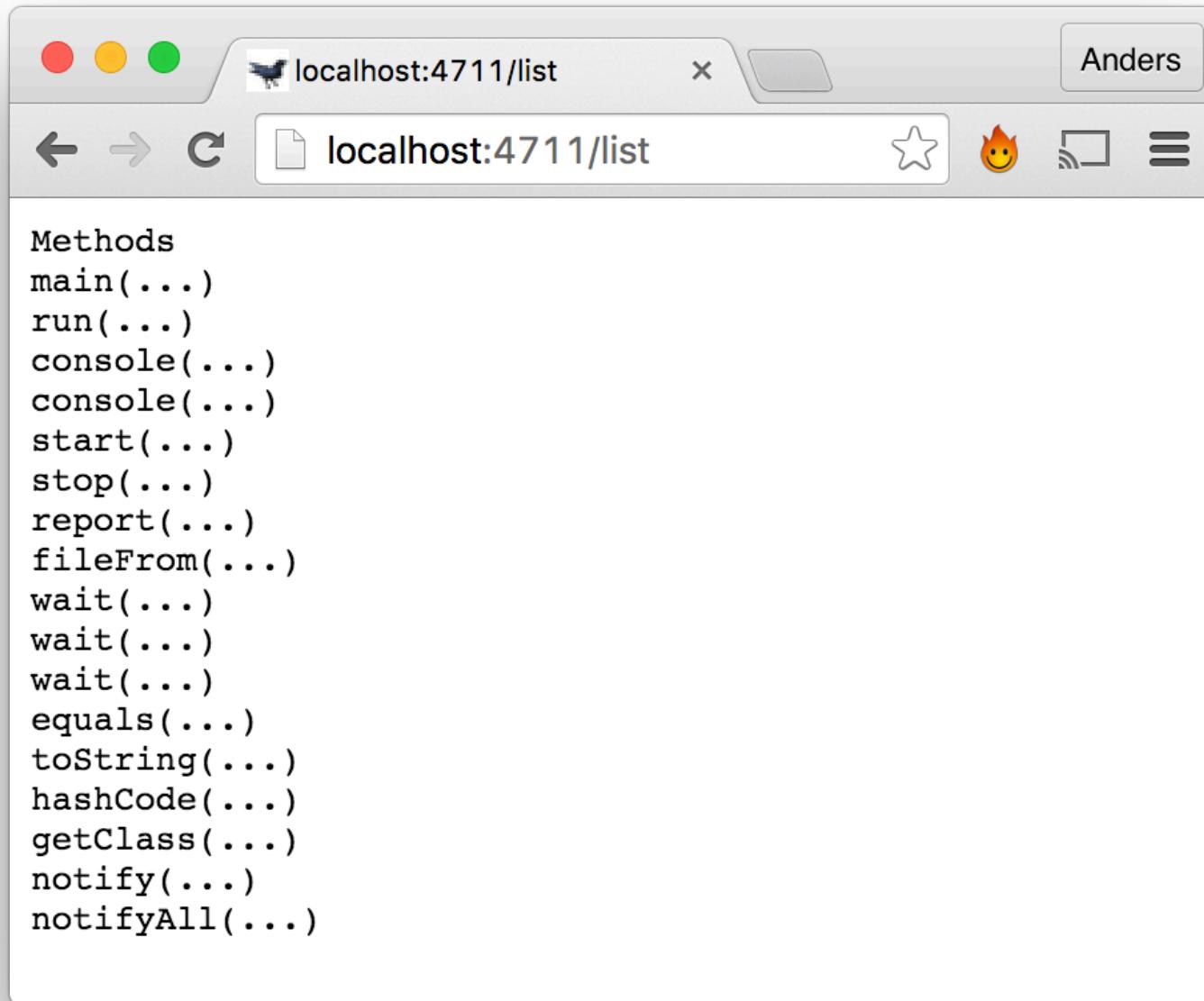


SimpleServer.java with root

```
public class SimpleServer extends Server {  
    public static void main(String... args) {  
        new SimpleServer().root("/Users/AKA/Sites").start();  
    }  
}
```

HttpService.java

```
@Override
public void run() {
    try {
        Request request = new HttpRequest(context, in);
        Response response = new HttpResponse(context, out);
        String resource = request.getResource();
        if ("/list".equals(resource)) {
            String text = "Methods";
            for (Method method : context.getClass().getMethods()) {
                text += "\n"+method.getName()+"(...)";
            }
            response.type("txt");
            response.send(text);
        }
        else if (resource.startsWith("/service/")) {
            // Do service call here
        }
        else {
            File file = context.fileFrom(resource);
            response.send(file);
        }
    }
    catch (IOException ex) {
        ex.printStackTrace();
        context.report(ex.getMessage());
    }
}
```



A screenshot of a web browser window titled "localhost:4711/list". The browser interface includes standard controls like back, forward, and refresh buttons, as well as a search bar with the same URL. A user profile icon "Anders" is visible in the top right. The main content area displays a list of Java method names:

```
Methods
main(...)
run(...)
console(...)
console(...)
start(...)
stop(...)
report(...)
fileFrom(...)
wait(...)
wait(...)
wait(...)
equals(...)
toString(...)
hashCode(...)
getClass(...)
notify(...)
notifyAll(...)
```

Goal #1

- Change `HttpService`, so “/list” will return an html page with a table of methods
- Exclude methods from `Object` and `Server` class
- **Optionally** exclude methods defined in the Context interface

Goal #2

- Create methods in your server in the form `getXyzHtml()` that returns html as a string
- Identify resources ending in “.html”
 - eg: “/welcome.html”
- Ensure that the HTTP method is “GET”
- Check the context (your server) for a corresponding method
 - eg: `getWelcomeHtml()`
- Call the method if it exists, otherwise show file (default)

Goal #3

- Create a local `int count` field in your server
- Update the field for every call to the server
- Create a `getCountHtml()` that reflects that count

Goal #4

- Create a method that handles form data using GET
 - eg: getPersonDataHtml(Request request)

```
<form action="personData.html" method="GET">
    ID: <input value="" name="id"/><br/>
    Name: <input value="" name="name"/><br/>
    Age: <input value="" name="age"/><br/>
    <button name="okButton" value="OK">OK</button>
</form>
```

Goal #5

- Create a method that handles form data using POST
 - eg: **postPersonDataHtml**(Request request)

```
<form action="personData.html" method="POST">
    ID: <input value="" name="id"/><br/>
    Name: <input value="" name="name"/><br/>
    Age: <input value="" name="age"/><br/>
    <button name="okButton" value="OK">OK</button>
</form>
```

Goal #6

- Create a REST service that returns all people in your server
- GET /person HTTP/1.1

Goal #7

- Create a REST service that returns the number of people in your server
- GET /person/count HTTP/1.1

Goal #8

- Create a REST service that returns a specific person in your server (i.e.: id = 2)
- GET /person/2 HTTP/1.1

Goal #9

- Create a REST service that updates or inserts a specific person in your server (i.e.: id = 7)
- POST /person HTTP/1.1
- { “id”: 7, “name”: “Kurt”, “age”: 27 }

Goal #9

- Create a REST service that deletes a specific person in your server (i.e.: id = 7)
- DELETE /person/7 HTTP/1.1