Web-based information systems 1

Getting started with dynamic web projects in IntelliJ Idea

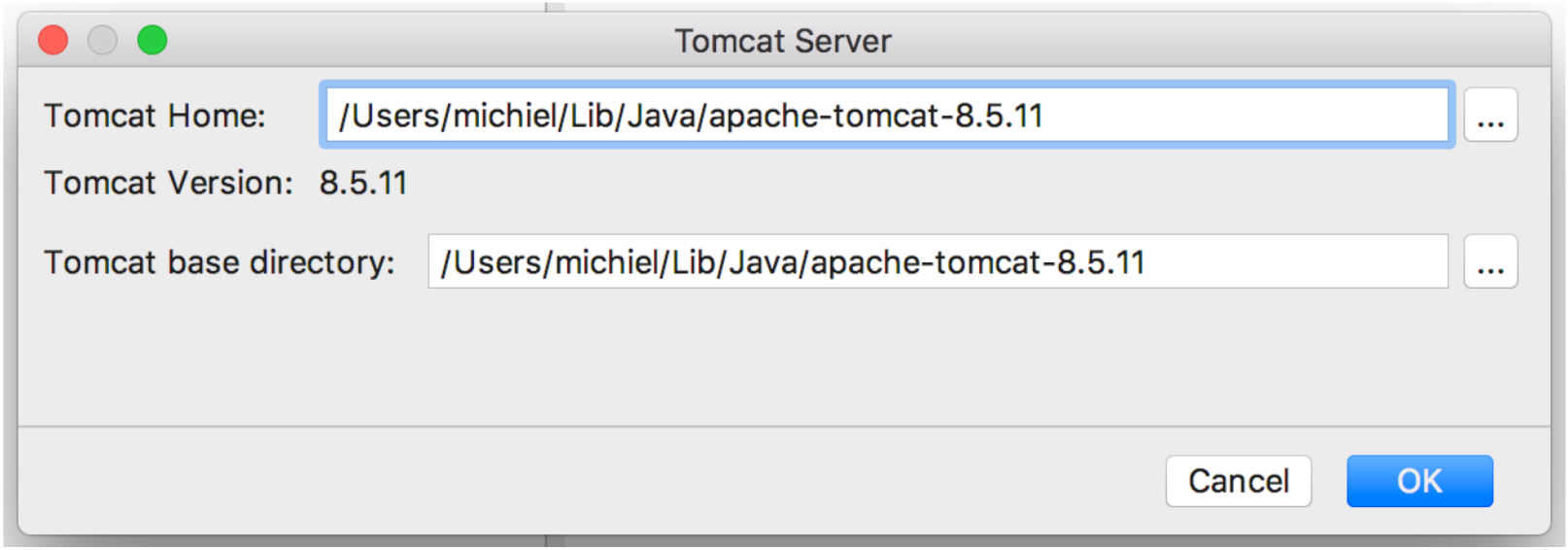
Michiel Noback (NOMI)
Institute for Life Sciences and Technology
Hanze University of Applied Sciences

introduction

- The topic of this presentation is getting started with a Java web application.
- This involves
  - configuring IntelliJ (or NetBeans) with the application server Tomcat
  - creating a web project
  - organizing it into a logical structure
  - creating you first servlet
  - running your first web application

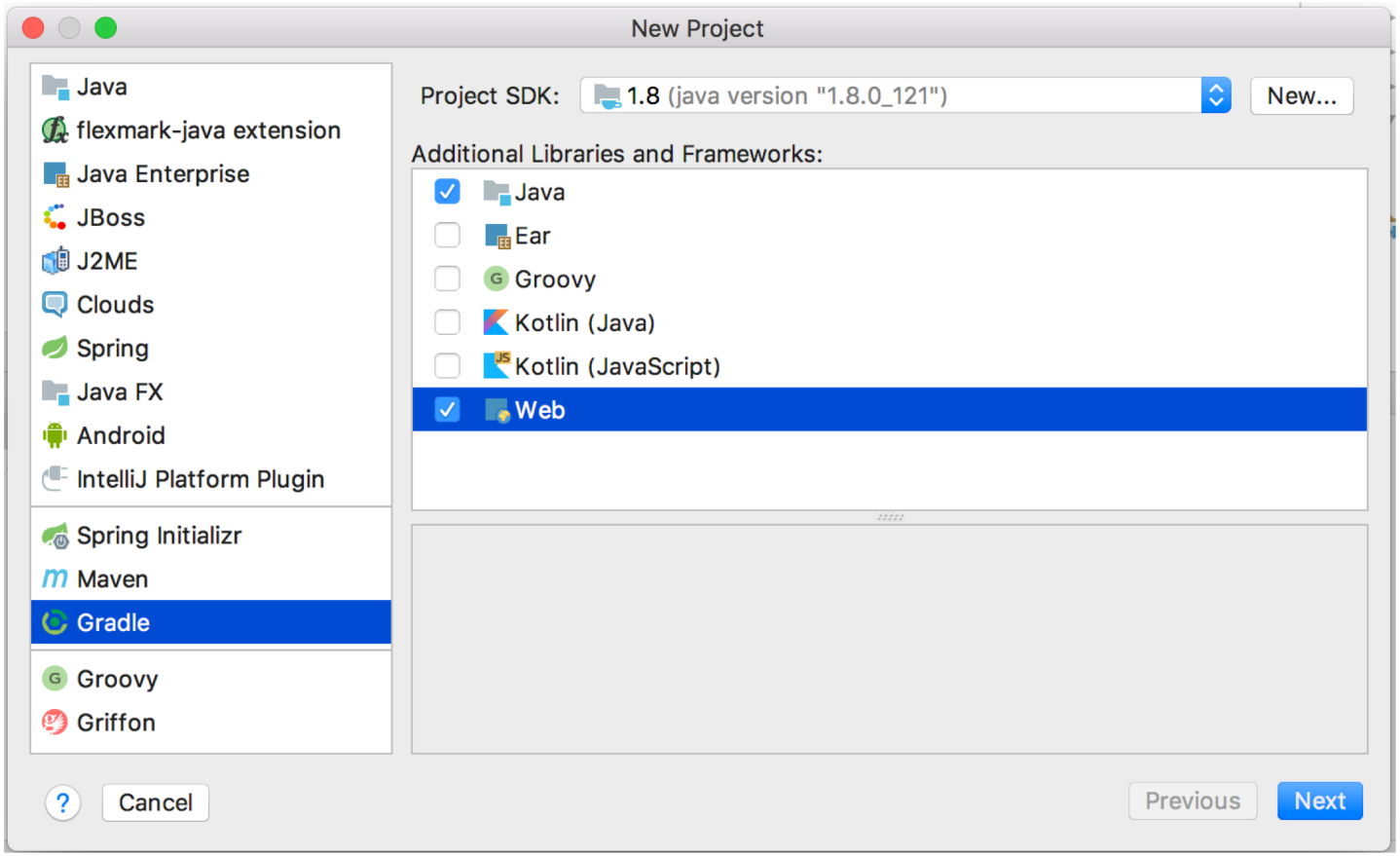
Configuring IntelliJ with Tomcat

- After installation (extraction) of Tomcat, you'll have to let IntelliJ know where it is
- You do this via Preferences -> Build, Execution, Deployment -> Application Servers -> click '+' (Add)



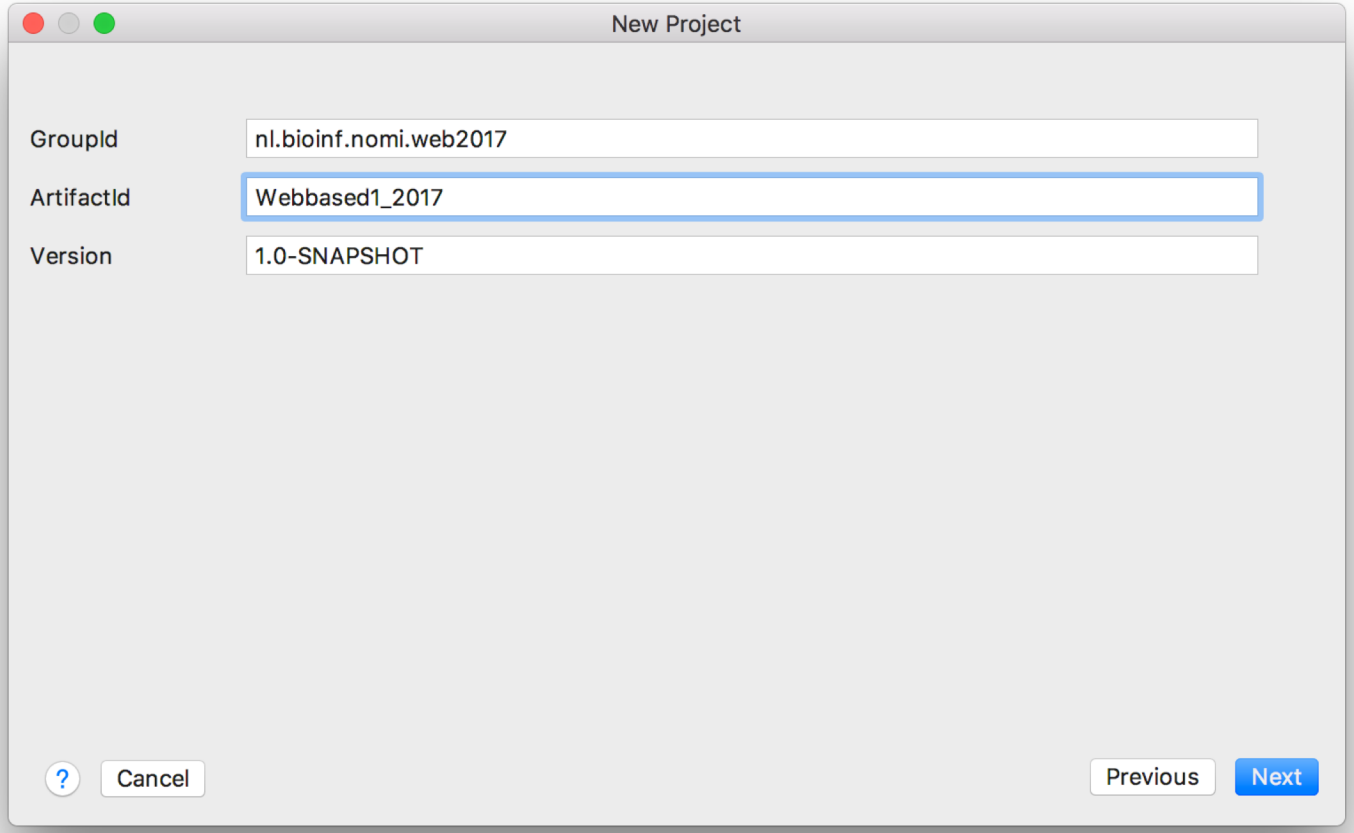
Creating a Gradle-managed web project in IntelliJ (1)

- Choose File -> New -> Project
- Choose Gradle -> + Java + Web -> Next



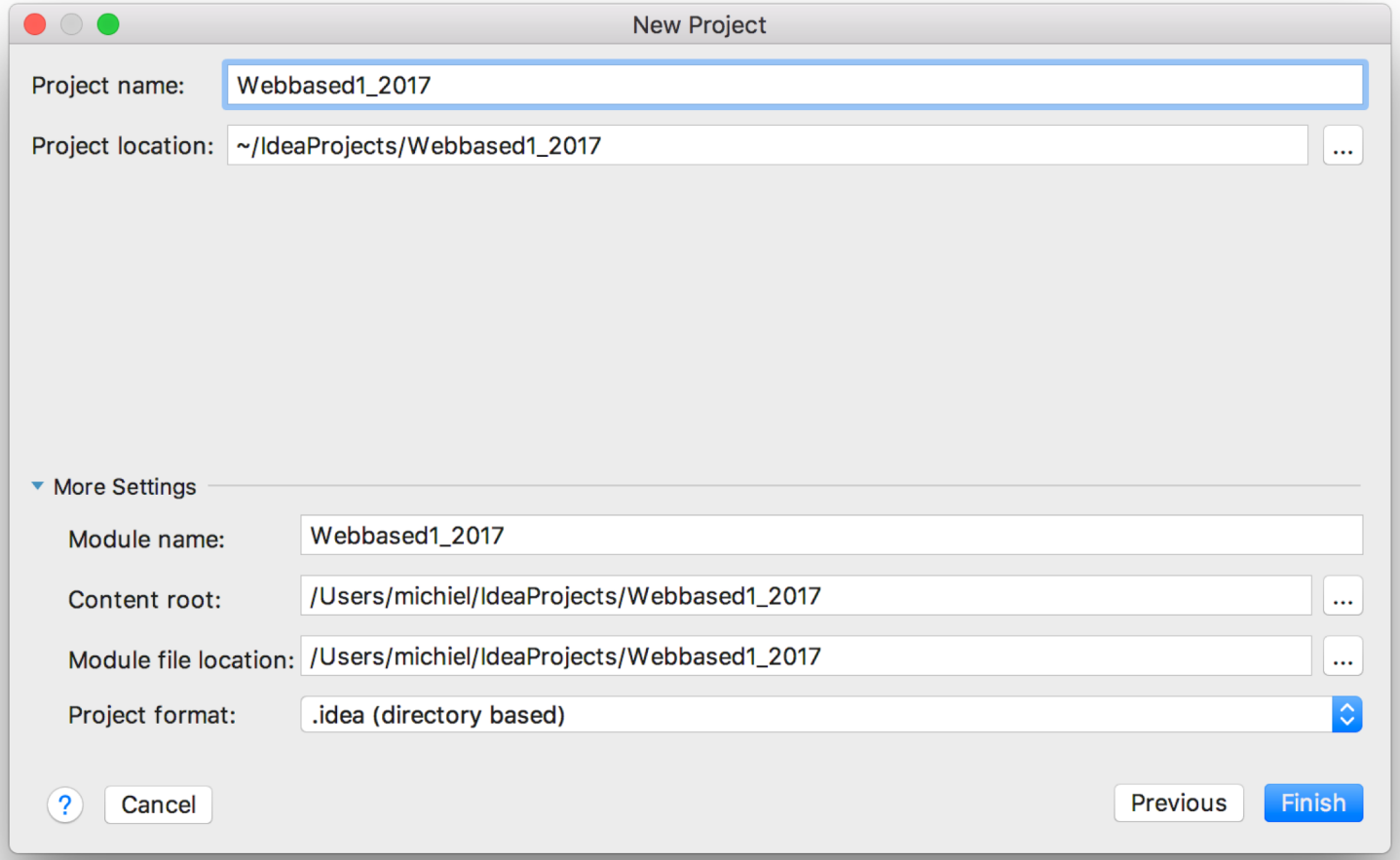
Creating Gradle web project (2)

- Enter GroupId, ArtifactId and Version → Click Next



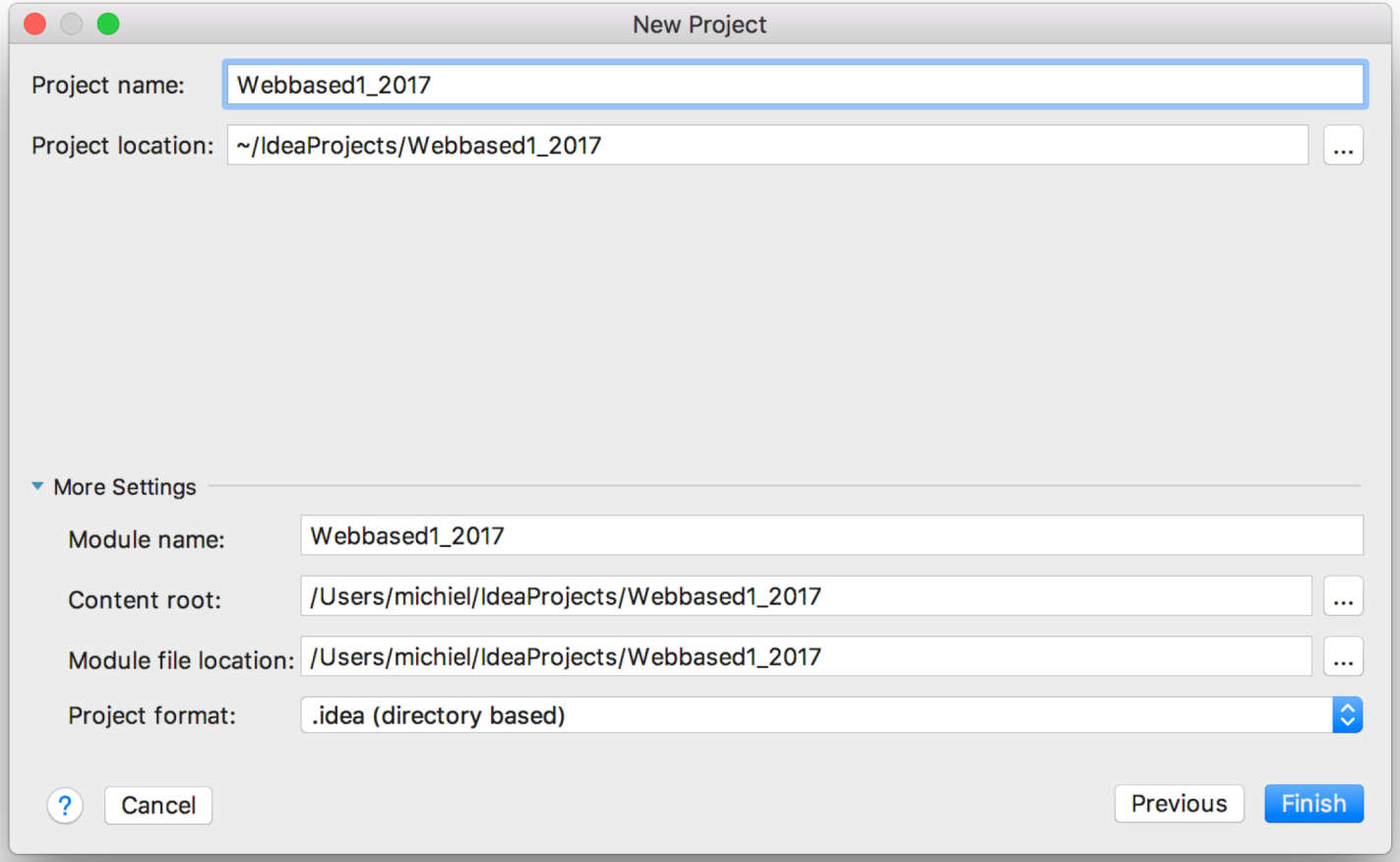
Creating Gradle web project (3)

- Check "Use auto-import" → Click Next → Click Finish



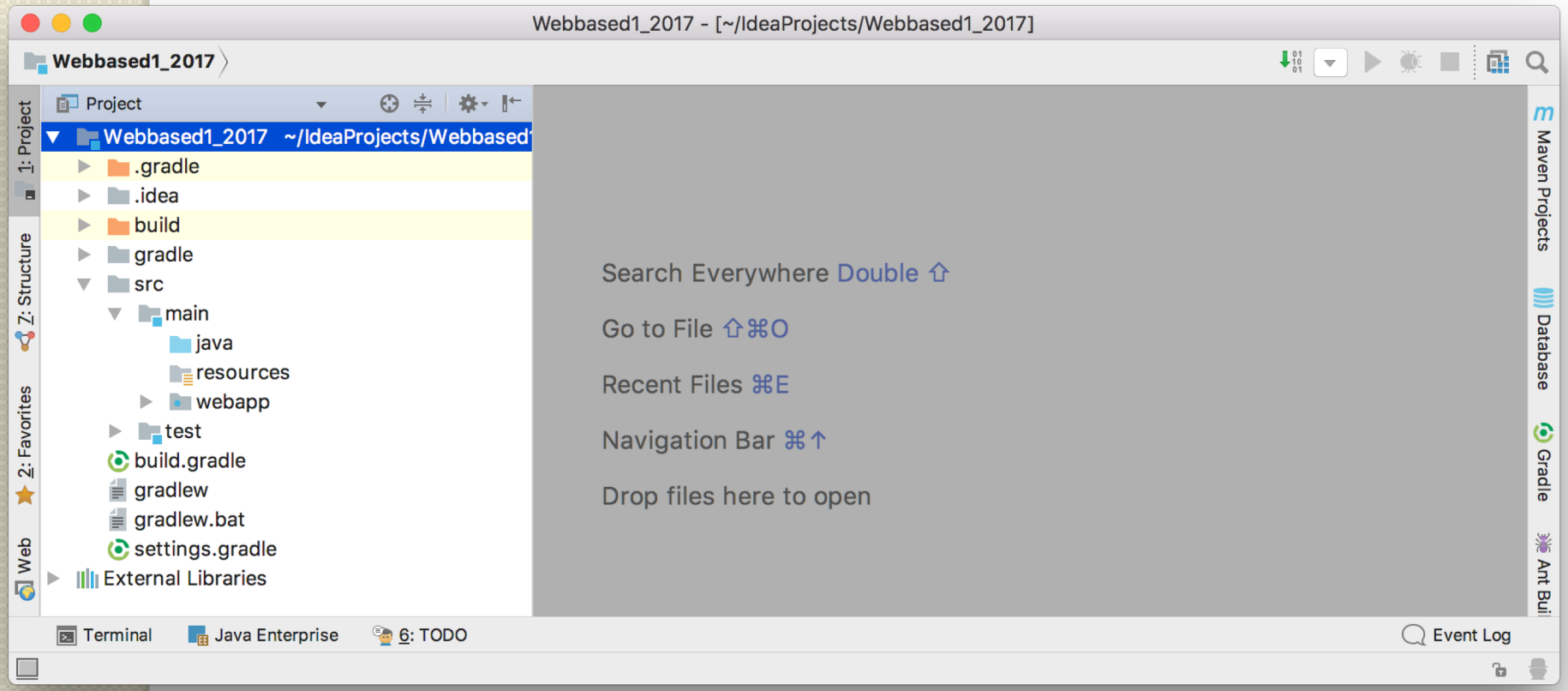
Creating Gradle web project (4)

- Check "Use auto-import" → Click Next → Click Finish

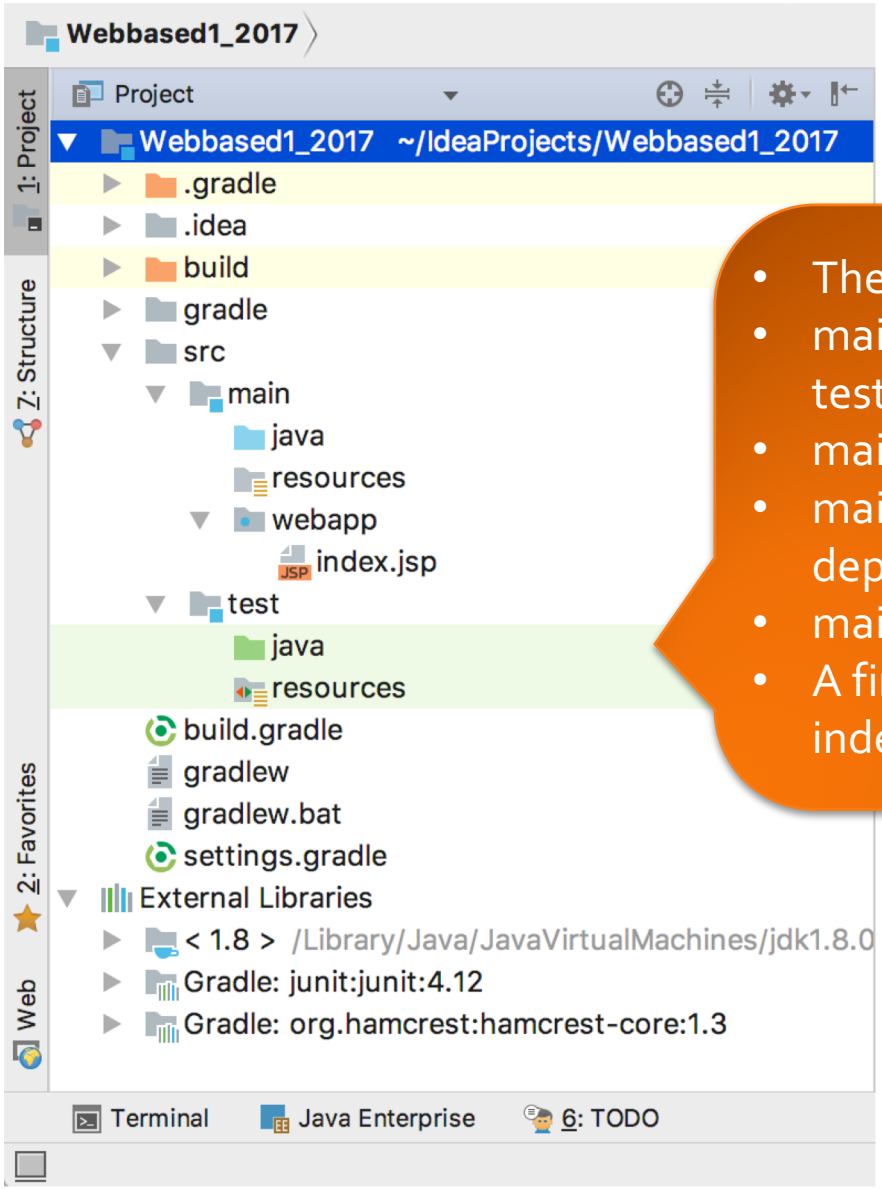


Creating Gradle web project (5)

- You will have something like this



The Gradle web project structure



- The directory src holds ALL your code
- main is for production, test for Junit test code
- main/java will hold java code
- main/resources will hold files not deployed with the app
- main/webapp will hold all web content
- A first web page is created for free: index.jsp

Running the jsp (1)

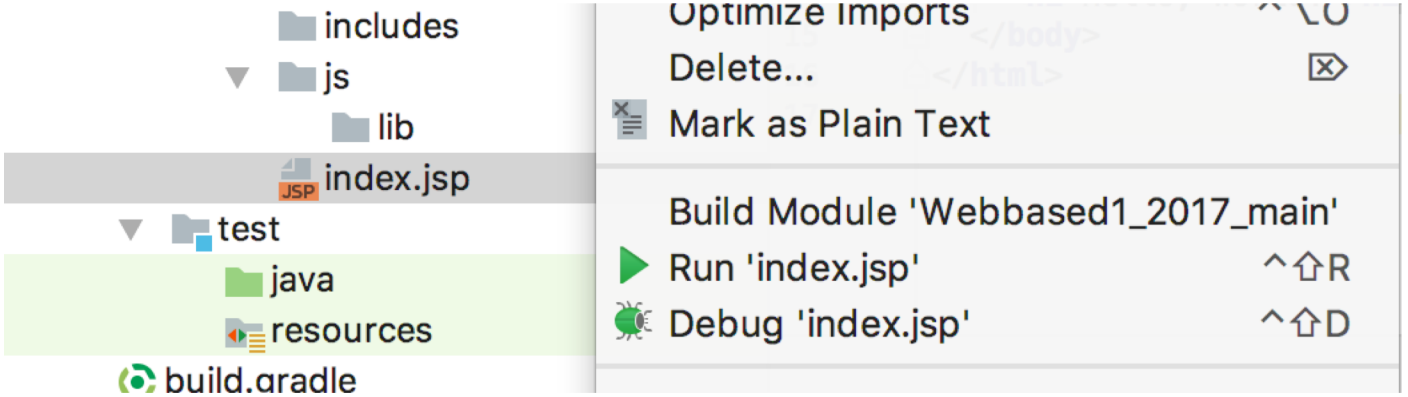
- Open the index.jsp file and make minor edits

index.jsp

```
1   <%--
2     Created by IntelliJ IDEA.
3     User: michiel
4     Date: 30/08/2017
5     Time: 14:54
6     To change this template use File | Settings | File Templates.
7   --%>
8   <%@ page contentType="text/html;charset=UTF-8" language="java" %>
9   <html>
10    <head>
11      <title>My Hello World jsp</title>
12    </head>
13    <body>
14      <h1>Hello, world!</h1>
15    </body>
16  </html>
17
```

Running the jsp (2)

- Right-click it in the explorer panel and select Run 'index.jsp'

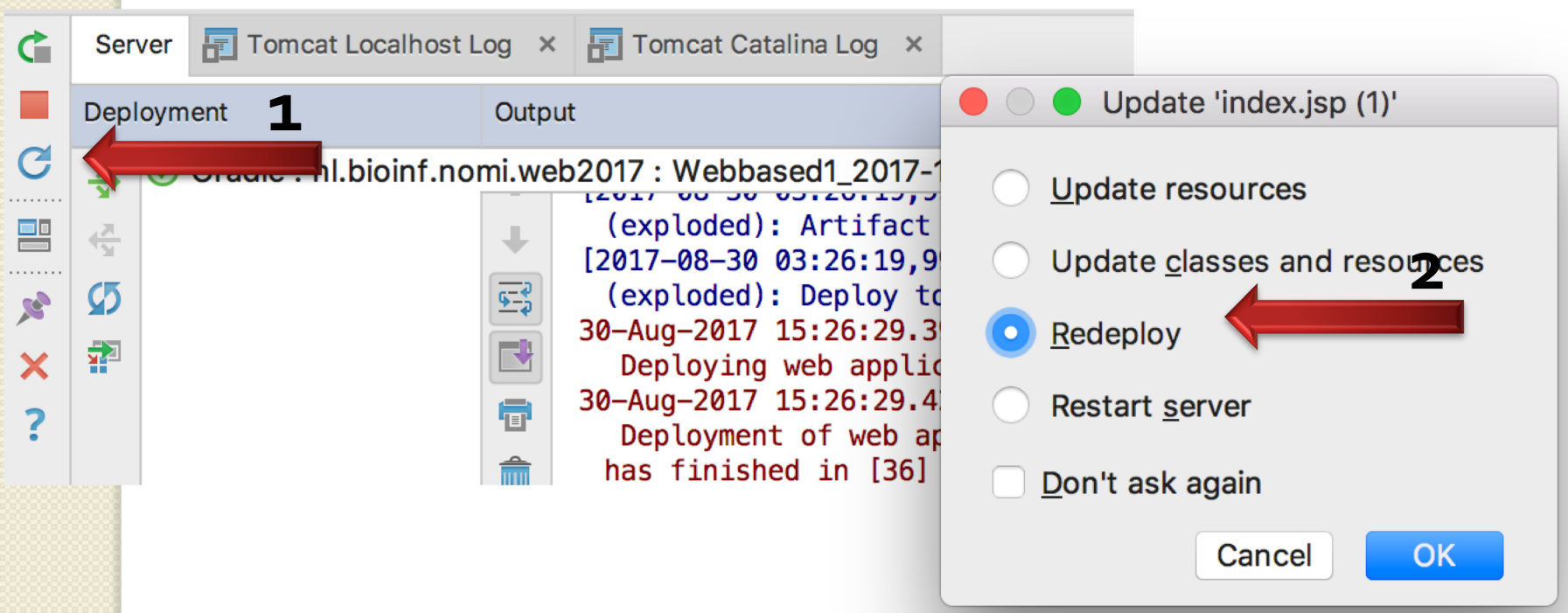


Running the jsp (3)

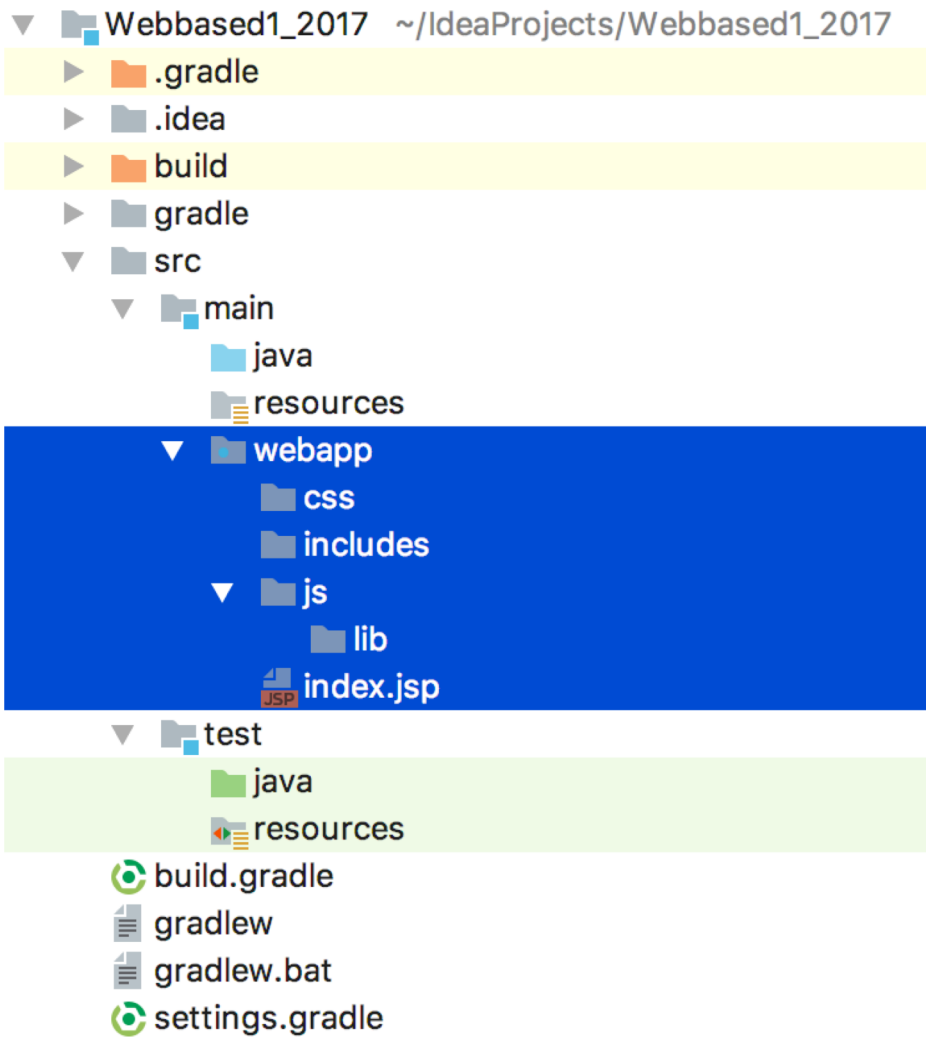
- If you're lucky you will see something like this

Running the jsp (4)

1. Make some change to index.jsp
2. Redeploy OR Update classes and resources (faster!)
3. Refresh browser
4. …Rinse and repeat



Organizing the web project



Create these subdirectories under Webapp: css (styles), includes, js, js/lib. It is important to organise your resources well!

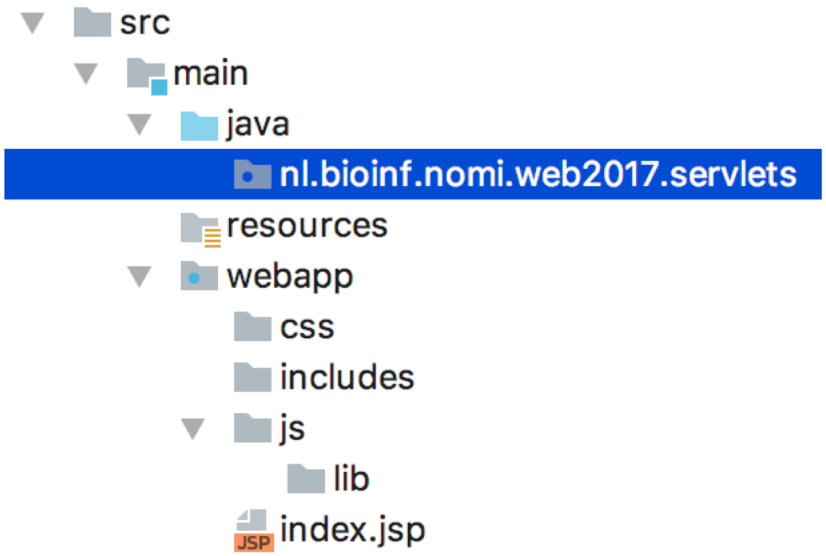
Make Java servlet functionality available

- Open file **`build.gradle`** and make sure the **dependencies** section has this content:

```
dependencies {
    #providedCompile 'javax.servlet:javax.servlet-api:3.1.0'
    compile group: 'jstl', name: 'jstl', version: '1.2'
    compile group: 'taglibs', name: 'standard', version: '1.1.2'
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

- The commented-out line is not required (can be managed by IntelliJ)
- We will come back to this file later!

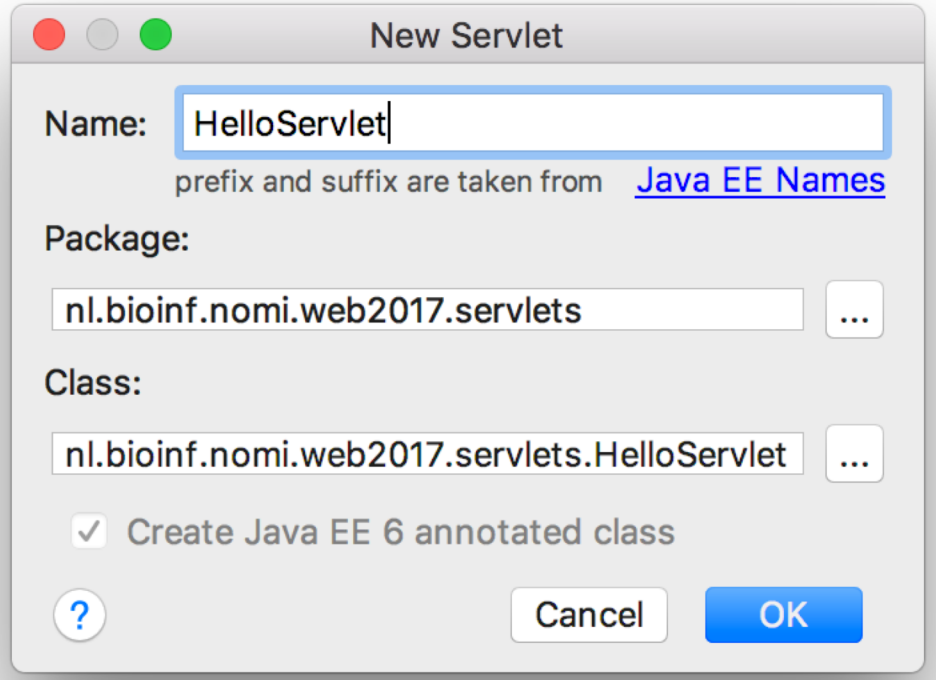
Create and deploy a first servlet (1)



Create a package to hold your Java servlet classes. It is good practice to have this correspond to the GroupId of the project. Later, more packages will be added as you go (e.g. nl.bioinf.nomi.web2017.model). Take care you follow package naming conventions!

Create and deploy a first servlet (2)

- Right-click the package → New → Servlet → Give it a name and click OK



Create and deploy a first servlet (3)

- Define a url to map onto this Servlet

```
@WebServlet(name = "HelloServlet", urlPatterns = "/hello")
public class HelloServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpSe

    }

    protected void doGet(HttpServletRequest request, HttpSer

    }
}
```

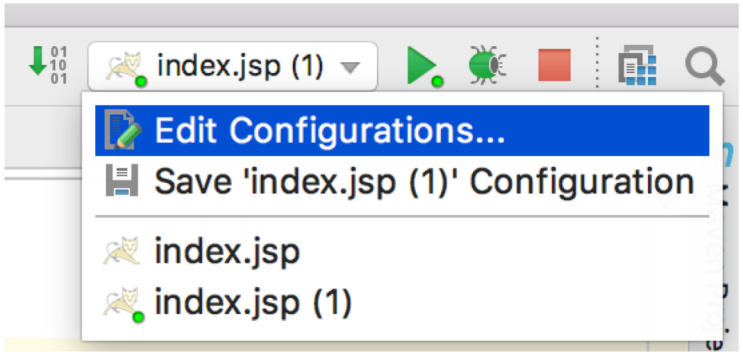
Create and deploy a first servlet (4)

- Add some logic

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws
             ServletException, IOException {
    PrintWriter out = response.getWriter();
    LocalDateTime currentTime = LocalDateTime.now();
    out.println("<html>");
    out.println("<head><title>My Hello World jsp</title>" +
            "</head><body><h1>It is ");
    out.println(currentTime.toString());
    out.println(" O’ Clock</h1></body></html>");
}
```

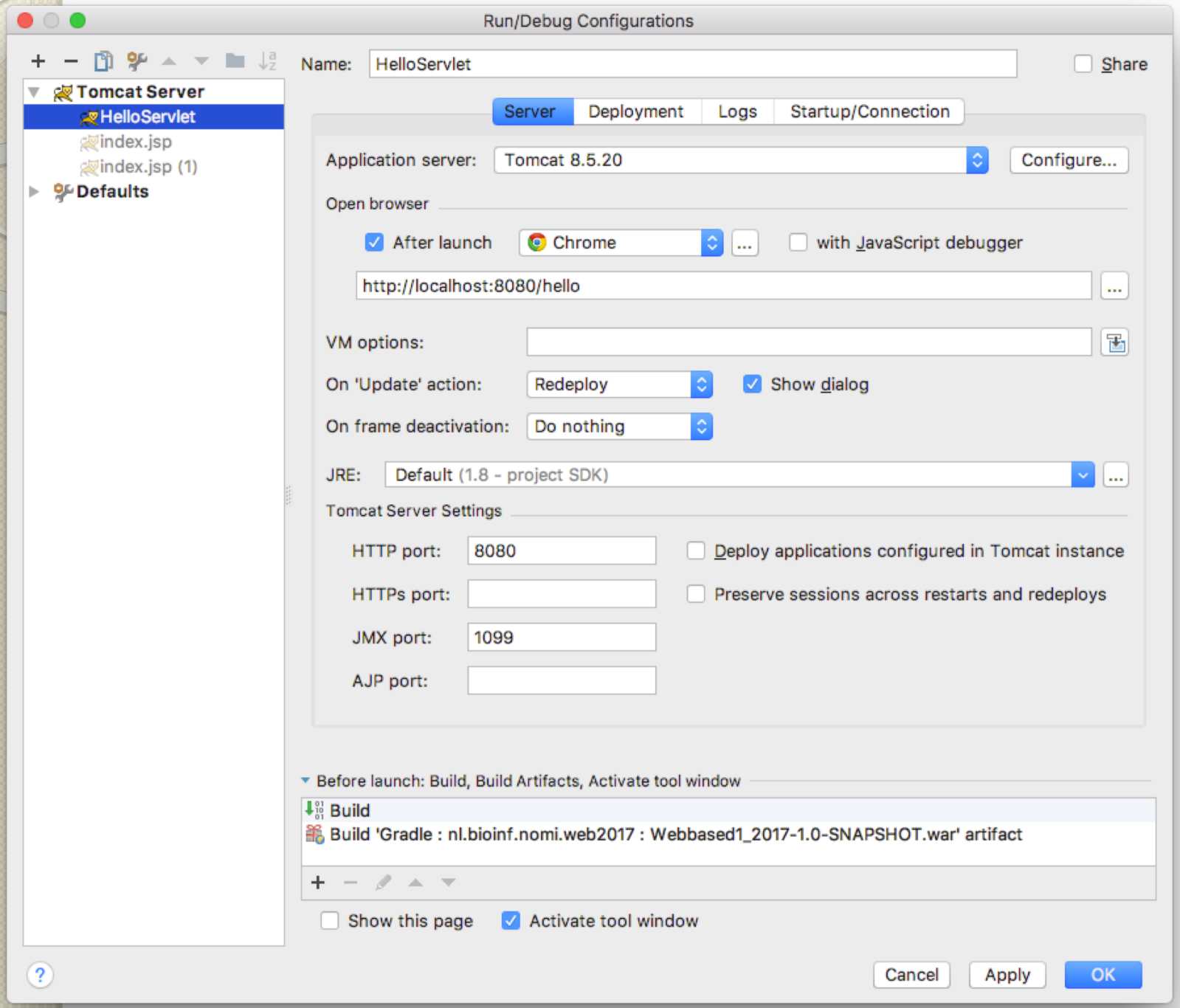
Create and deploy a first servlet (5)

- Create a Run Configuration (Edit Configurations…)



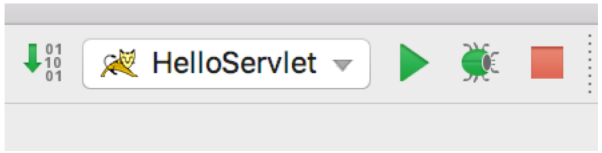
Create and deploy a first servlet (6)

- New Configuration → Tomcat Server → Local
- Give it a name and default behaviour

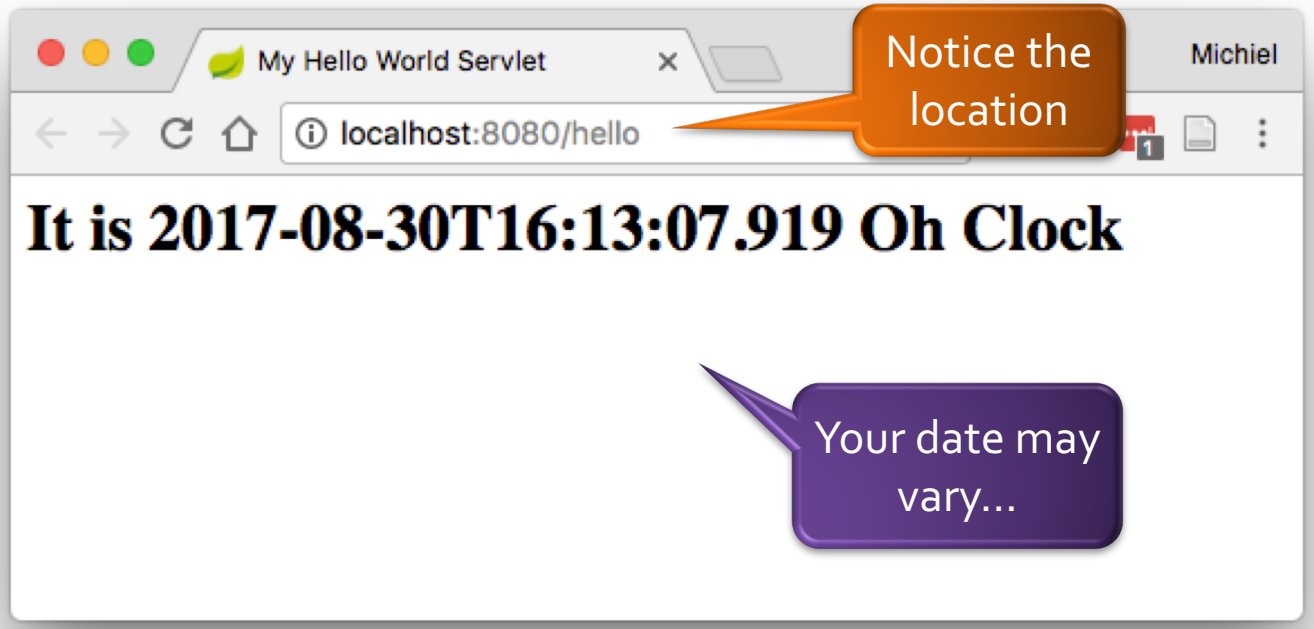

Run/Debug Configurations
Tomcat Server
HelloServlet
index.jsp
index.jsp (1)
Defaults
Name:
HelloServlet
Share
Server
Deployment
Logs
Startup/Connection
Application server:
Tomcat 8.5.20
Configure...
Open browser
After launch
Chrome
with JavaScript debugger
http://localhost:8080/hello
VM options:
On 'Update' action:
Redeploy
Show dialog
On frame deactivation:
Do nothing
JRE:
Default (1.8 - project SDK)
Tomcat Server Settings
HTTP port:
8080
Deploy applications configured in Tomcat instance
HTTPs port:
Preserve sessions across restarts and redeploys
JMX port:
1099
AJP port:
Before launch: Build, Build Artifacts, Activate tool window
Build
Build 'Gradle : nl.bioinf.nomi.web2017 : Webbased1_2017-1.0-SNAPSHOT.war' artifact
Show this page
Activate tool window
Cancel
Apply
OK

Create and deploy a first servlet (7)

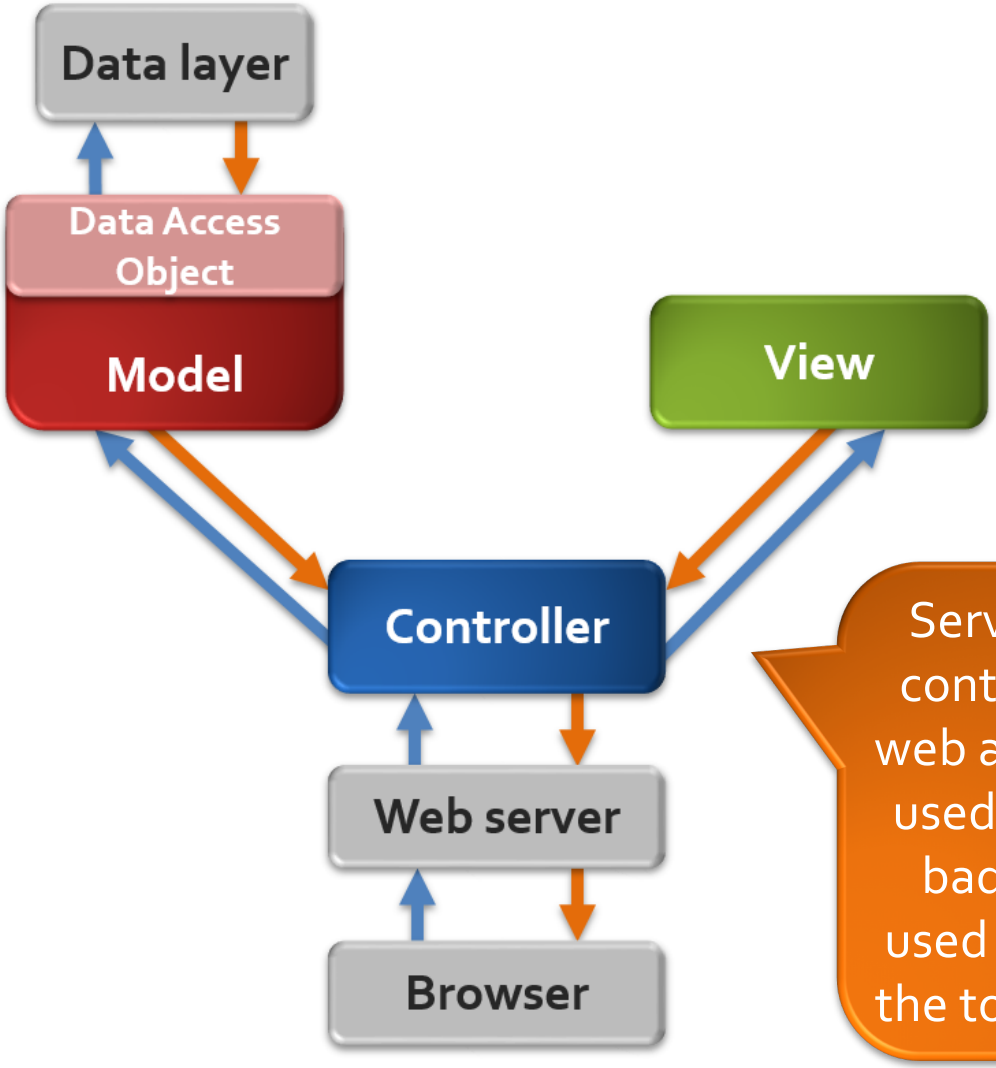
1. Run it



2. View it



MVC design pattern



Servlets are often used as controllers in (simple) Java web apps. You have now also used it as the view. This is a bad idea: JSPs should be used for the view. But that's the topic of the next session.