JSR 371 : MVC 1.0 Workshop

Global Prerequisite:

- Laptop
- Java 7+
- Glassfish Nightly downloads from http://download.oracle.com/glassfish/4.1/nightly/index.html
- e.g. http://download.oracle.com/glassfish/4.1/nightly/glassfish-4.1-b17-09_16_2015.zip
- Eclipse or IntelliJ Idea or even NetBeans :) and configure the just downloaded glassfish.
- Maven
- Optionally : MySQL or any other SQL server (for part 2 of the workshop) if you dont want to use in memory database.

Part 1 Hello World ?

GENERAL RULE : DO NOT COPY CODE. In order to learn you have to TYPE .. also the Apple Pages are messing the quotes .. so you have to fix them by yourself :) “ “ “ “ “” “””” :) ok?

1. Create a new maven project called ozark-sample.

packaging as war.
and add the following dependancy:

```
<dependency>
    <groupId>com.oracle.ozark</groupId>
    <artifactId>ozark</artifactId>
    <version>1.0.0-m01</version>
    <scope>compile</scope>
</dependency>
```

2. If you are using Java 8 add also the following properties:

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

Because we do not want an web.xml file since it is so so old school, but maven searches for it please add also the following property :

```
<failOnMissingWebXml>false</failOnMissingWebXml>
```

So at the end you have the 3.

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <failOnMissingWebXml>false</failOnMissingWebXml>
</properties>
```

This is the ozark reference implementation of the MVC 1.0.

3. We will also use CDI and maybe because of Ozark BUG or something we need beans.xml file. So a beans.xml file located in

%SOURCE%/webapp/WEB-INF/beans.xml is required with the following content :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
beans_1_1.xsd"
       bean-discovery-mode="all">
</beans>
```

This “almost” empty file is required to make CDI work.

4. Then because of JAX-RS we need to create our Application class:

```
package your.group.id;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

/**
 * Created by bg-jug on 30.09.2015
 */
@ApplicationPath("app")
public class MyApplication extends Application {

}
```

This basically says that the MVC application will be on the /app path.

5. Next two bits are a controller and a jsp.
For now create an empty jsp called hello.jsp in webapp folder

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Current Time</title>
</head>
<body>
    I am a jsp
</body>
</html>
```

Create a controller class :

```
@Controller
@Path("hello")
public class HelloController {
 @GET
    public String hello() {
          return “/hello.jsp”;
   }
}
```

What this class do (as easy to see) is to wait for GET requests on the /app/hello path and when some GET request came to show hello.jsp as view. The /app is coming from the Application Path and the /hello is coming from the controller.

6. Because MVC 1.0 steps on top of JAX-RS some of the annotations are in JAX-RS jar and not in the framework jar. Because of that we need to update our pom.xml to fix this.

Add the following dependancy:

```
        <dependency>
          <groupId>javax</groupId>
          <artifactId>javaee-api</artifactId>
          <version>7.0</version>
          <scope>provided</scope>
        </dependency>
```

The full pom.xml should look like this :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.gochev</groupId>
    <artifactId>ozark-sample</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.oracle.ozark</groupId>
            <artifactId>ozark</artifactId>
            <version>1.0.0-m01</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </properties>
</project>
```

7. Run the project and make sure the JSP is shown. Configure the glassfish in your favorite IDE (Eclipse, IDEA or even Netbeans .. or use command prompt if you like)

8. Ok Next : Lets pass some data to the jsp.

Basically there are 2 ways:

-To inject a Models object which implements the Map Interface and acts as a Bag of everything you want to be visible in the view. It is like Injecting ModelMap in Spring MVC or ViewBag in ASp.NET MVC and so on.

Lets do that first:
9. In your controller add as a field:

```
@Inject
Models models;
```

In the method:

```
this.models.put("msg","Hello World”);
```

10. Then update the JSP :

```
This is the message : ${msg}
```

As you can see everything you put in the Models is available in the jsp and you can get it using the JSP EL syntax (when this is not good ? :) .

-The second and more normal way is to create a model class that your controller will use.

```
@Model
public class UserModel {
    private String firstName = "Nayden";

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

11. Then inject the model in the controller.

```
@Inject
UserModel userModel;
```

12. And change the value if you want in the controller method or use the default one.

13. Update the jsp to print the firstName :

```
This is the message : ${msg}  and the user first name is : ${userModel.firstName}
```

14. Run the project