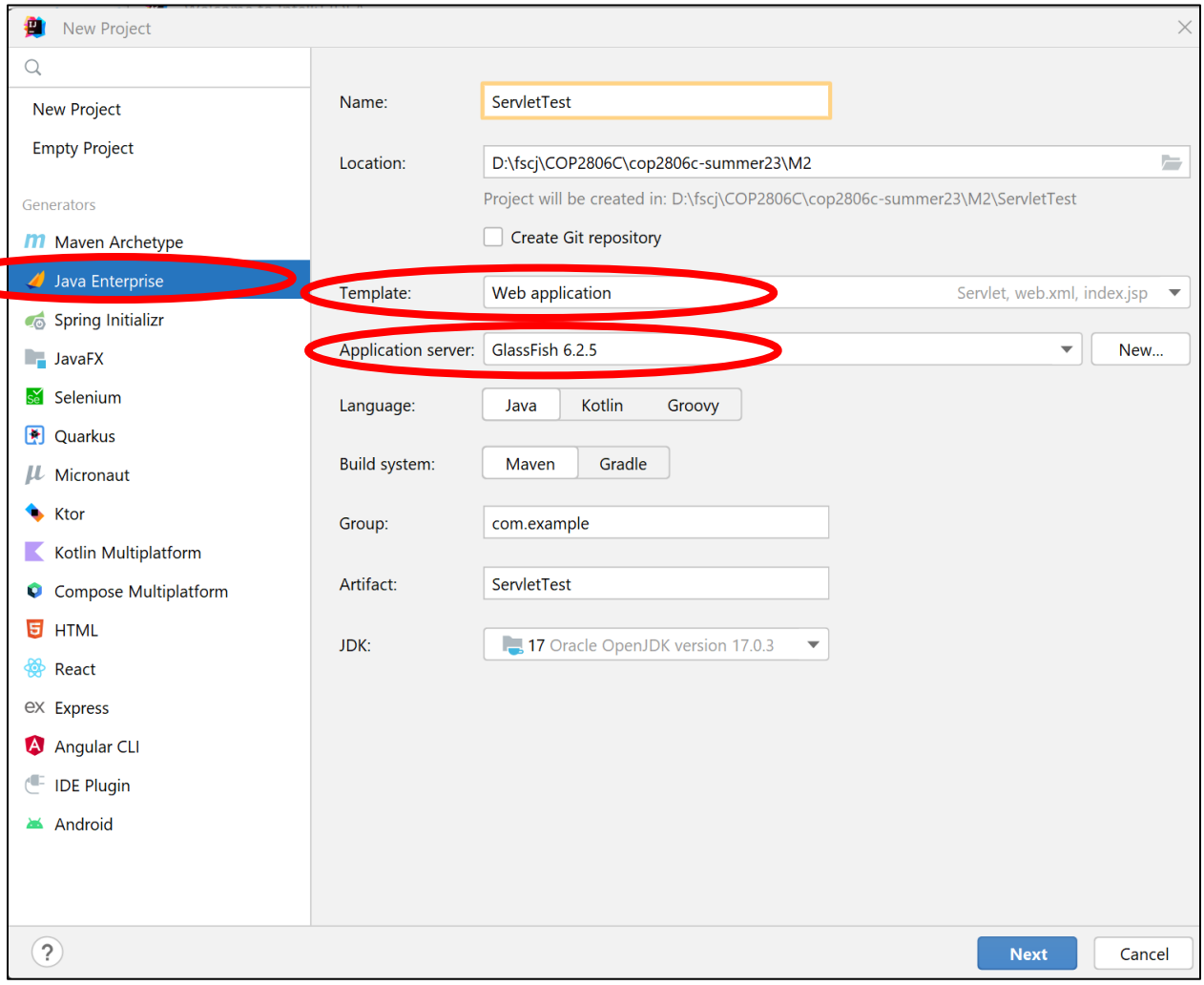
COP2806C Module 2 Programming Assignment Part 2: Simple Injection and Scoping

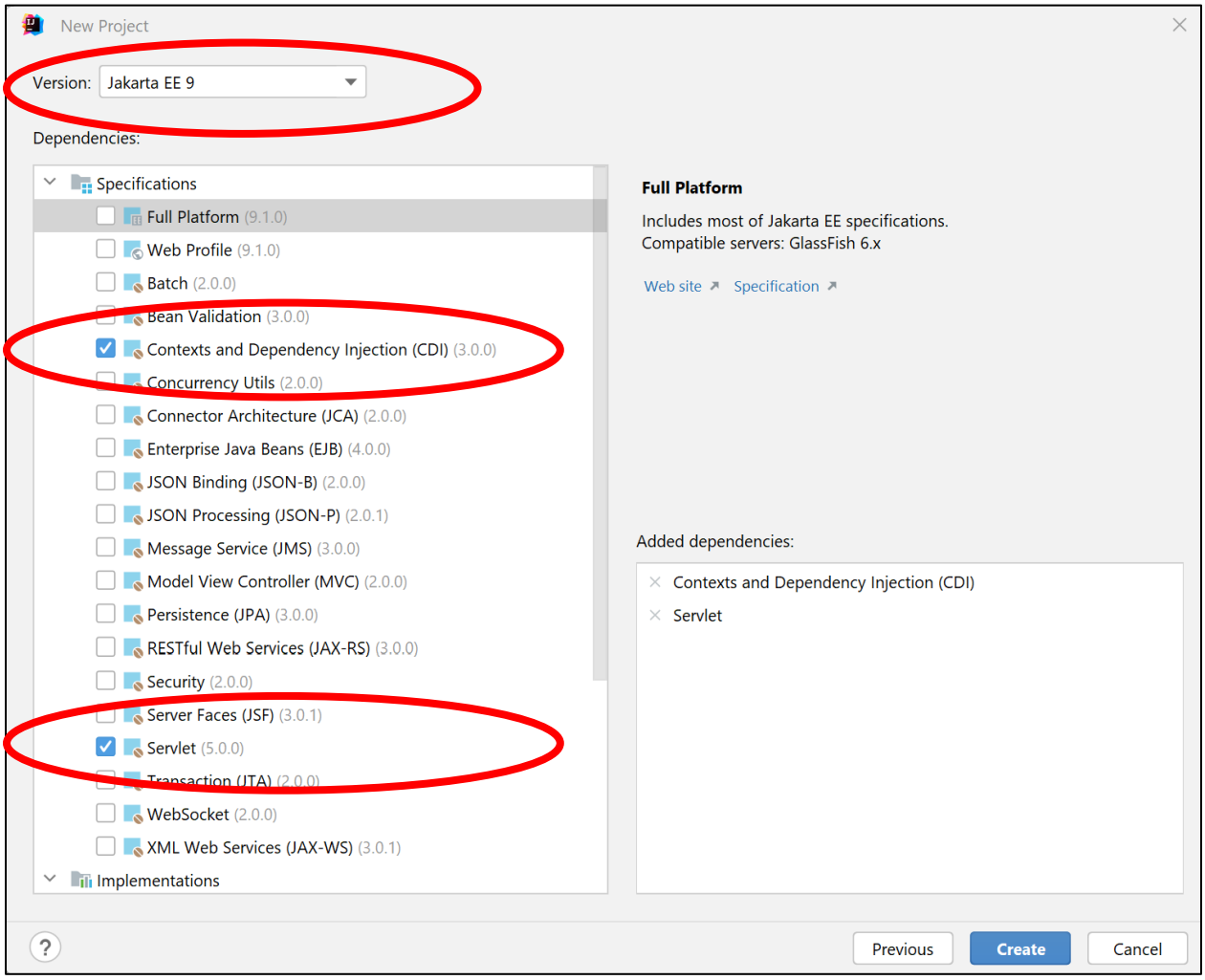
In this assignment we will implement a simple servlet which will inject a bean and demonstrate scoping behavior.

I would recommend you use the Horizon system for this exercise, which has all necessary software installed.

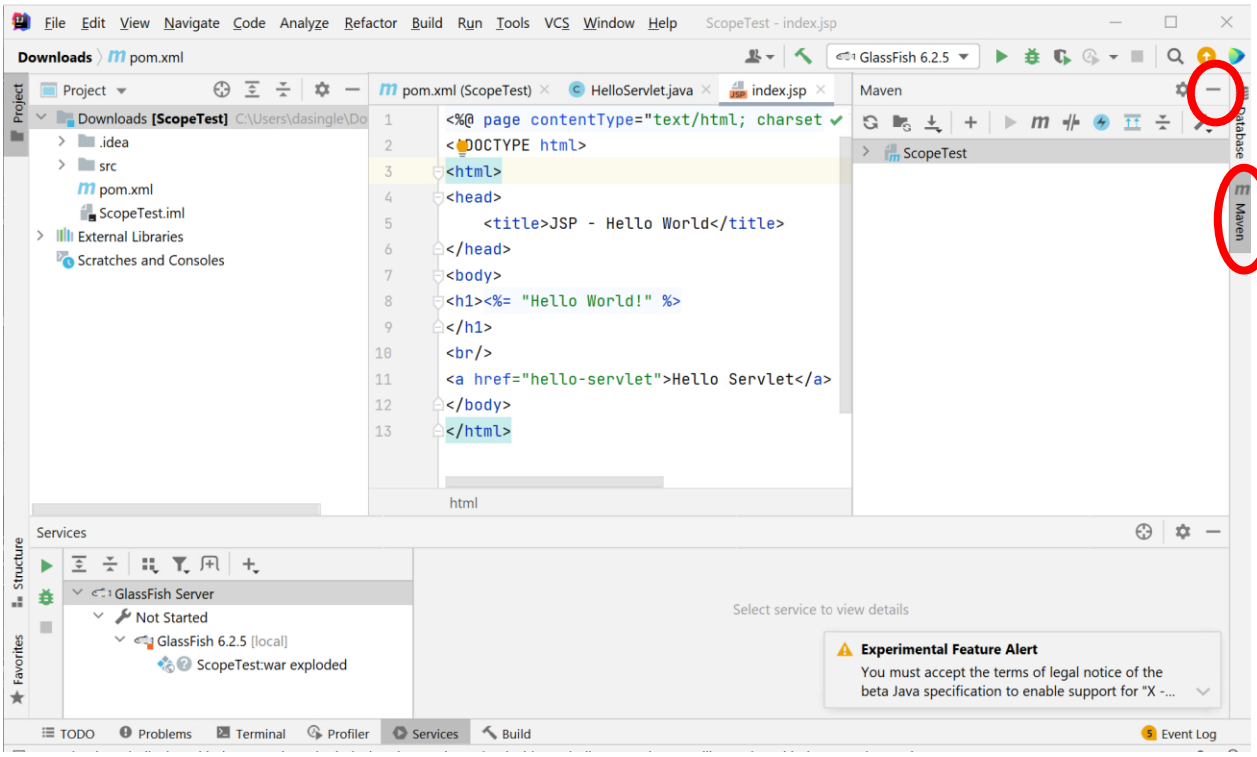
Use IntelliJ to create a new Java Enterprise project. Select "Web application" for the project template and GlassFish (on Horizon, the path is C:\glassfish-6.2.5\glassfish6) as the application server. Accept the defaults for the remaining options as shown in this image:



Select Jakarta EE 9 from the Version dropdown, then check "Contexts and Dependency Injection (CDI)". The Servlet box should already be checked.



The IDE will open with the index.jsp (Java Server Page) file in the edit window. You can collapse the Maven tool by clicking on the Maven button on the right edge of the IDE or the minimize button to create more edit space:



Note the syntax of the index.jsp file. Java Server Pages use HTML with additional features such as JSP expressions (<%= %>). This web page which describes JSP syntax in more detail.

The index.jsp file is similar to the index.html file and acts as the "home page" for a servlet. This page displays a "Hello Servlet" link which passes an HTTP request to the servlet on the back end.

pom.xml (ScopeTest) | HelloServlet.java | index.jsp

```jsp
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>JSP - Hello World</title>
</head>
<body>
<h1><%= "Hello World!" %>
</h1>
<br/>
<a href="hello-servlet">Hello Servlet</a>
</body>
</html>
```

Now examine the HelloServlet class in the HelloServlet.java. This is a standard template IntelliJ provides for a Java Servlet. The "doGet" method receives and processes HTTP requests from the browser; the template prints a simple Hello World message when the user clicks on the Hello Servlet link implemented in the index.jsp file.

```java
@WebServlet(name = "helloServlet", value = "/hello-servlet")
public class HelloServlet extends HttpServlet {
    private String message;

    public void init() { message = "Hello World!"; }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        response.setContentType("text/html");

        // Hello
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>" + message + "</h1>");
        out.println("</body></html>");
    }

    public void destroy() {
    }
}
```

Go ahead and build / run this basic servlet to verify it works before continuing. Click on the Hello Servlet link should execute the servlet and display "Hello World" (if your browser does not automatically start, connect to http://localhost:8080/ServletTest-1.0-SNAPSHOT/)

Now let's create a managed bean and inject it into our servlet object.

Start by creating a UserBean class in the same package as the HelloServlet.java. Add a private String instance variable which contains your name and provide a getter for this variable.

```
public class UserBean {
    private String name = "David";

    public UserBean() {}

    public String getName() {
        return name;
    }
}
```

In the HelloServlet class, instantiate a UserBean object and display the name after the "Hello World" message is displayed (see red text below):

```
@WebServlet(name = "helloServlet", value = "/hello-servlet")
public class HelloServlet extends HttpServlet {
    private String message;

    UserBean userBean = new UserBean();

    public void init() {
        message = "Hello World!";
    }

    public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException {
        response.setContentType("text/html");

        // Hello
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>" + message + "</h1>");
        out.println("Hello " + userBean.getName());
        out.println("</body></html>");
    }

    public void destroy() {
    }
}
```

Verify your name is display by the servlet as expected.

Now let's inject the bean into the application. Start by annotating the UserBean class with **@ManagedBean** (you will need to right-click for context actions and import the import jakarta.annotation.ManagedBean class).

```
@ManagedBean
public class UserBean {
    private String name = "David";

    public UserBean() {}

    public String getName() {
        return name;
    }
}
```

In HelloServlet.java, remove the call to new and annotate the declaration with **@Inject**:

```
@WebServlet(name = "helloServlet", value = "/hello-servlet")
public class HelloServlet extends HttpServlet {
    private String message;
    @Inject
    UserBean userBean;
```

Build and run again (when prompted you can just redeploy the artificact, you do not need to restart the server every time). You should see no change in the behavior of the application.

We are now injecting our bean without needing to instantiate it (the injection takes care of this for us). Let's play around with the scope of the bean.

Add a useCount instance variable to the UserBean class. Include a getter, and whenever the getName accessor is called, increment this variable in getName(). Finally, annotate the bean with **@RequestScoped**:

```
@ManagedBean
@RequestScoped
public class UserBean {
    private int useCount = 0;
    private String name = "David";

    public UserBean() {}

    public String getName() {
        useCount++;
        return name;
    }

    public int getUseCount() {
        return useCount;
    }
}
```

@RequestScoped means every time we hit the servlet from our browser, the bean will be reinstantiated. No matter how many times we run the application, useCount will always be 1. Verify this behavior.

Now change **@RequestScoped** to **@ApplicationScoped**. Build, redeploy, and run again and observe the behavior. Refresh your tab, duplicate the tab, open a different browser, go back and forth between the browsers and check the value of useCount.

Finally, change **@ApplicationScoped** to **@SessionScoped.** For this to build, you will need to mark UserBean as Serializable ("implements Serializable"); this is because @SessionScoped beans are ultimately stored in the user's HTTP session. Build, redeploy, and execute **on different browsers** (e.g. Chrome and Firefox). Refresh the pages and observe the behavior.

The submission for this assignment consists of two parts:

- Write a brief summary of your observations for each scope value in the text submission area for the Canvas assignment (***not*** in the assignment comment section).
- Submit your project to the GitHub Classroom repo.