APPENDIX C

JakartaEE Apps

What we'll cover:

- Getting the IntelliJ Ultimate edition
- Creating a simple JakartaEE web app
- Configuring GlassFish as an application server

There's plenty of support for JakartaEE in IntelliJ — that is if you're using the Ultimate edition. That's not to say you can't build any JakartaEE apps on the Community Edition; you can (by using Maven to create JakartaEE apps). The Ultimate Edition, however, makes developing JakartaEE apps much simpler and easier.

IntelliJ Ultimate

The Ultimate edition is a commercial IDE, but you can use it for free for thirty days. After thirty days, either you get a commercial license, or you'll be limited to thirty minutes of use per session.

You can download the installer for the Ultimate edition from the same place where you downloaded the Community Edition — `https://www.jetbrains.com/idea/download`. Choose the Ultimate edition.

The welcome screen of IntelliJ Ultimate (shown in Figure C-1) is exactly the same as the Community Edition.

© Ted Hagos 2022
T. Hagos, *Beginning IntelliJ IDEA*, https://doi.org/10.1007/978-1-4842-7446-0

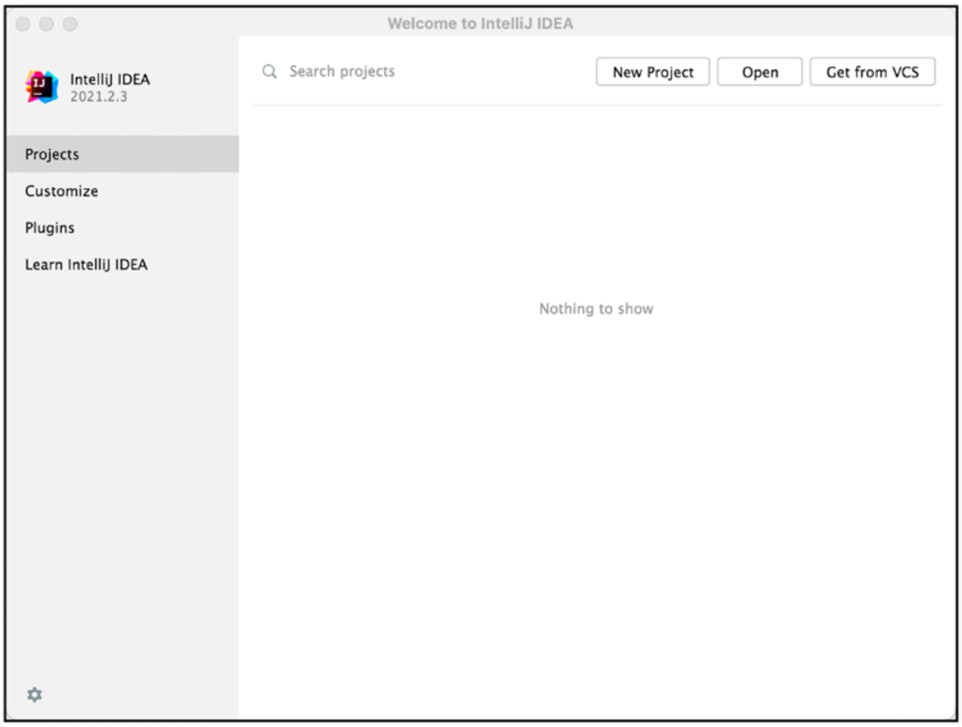
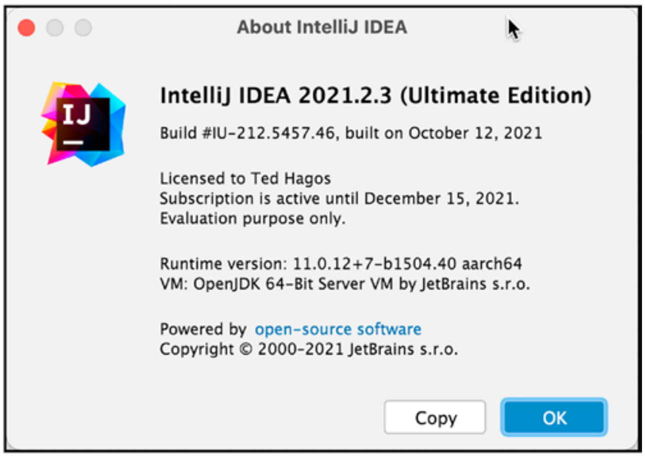


Figure C-1. *Welcome to IntelliJ*

Unless you go to the About dialog (shown in Figure C-2), you can't tell whether what you're using is the Ultimate or the Community Edition.


About IntelliJ IDEA

IntelliJ IDEA 2021.2.3 (Ultimate Edition)
Build #IU-212.5457.46, built on October 12, 2021

Licensed to Ted Hagos
Subscription is active until December 15, 2021.
Evaluation purpose only.

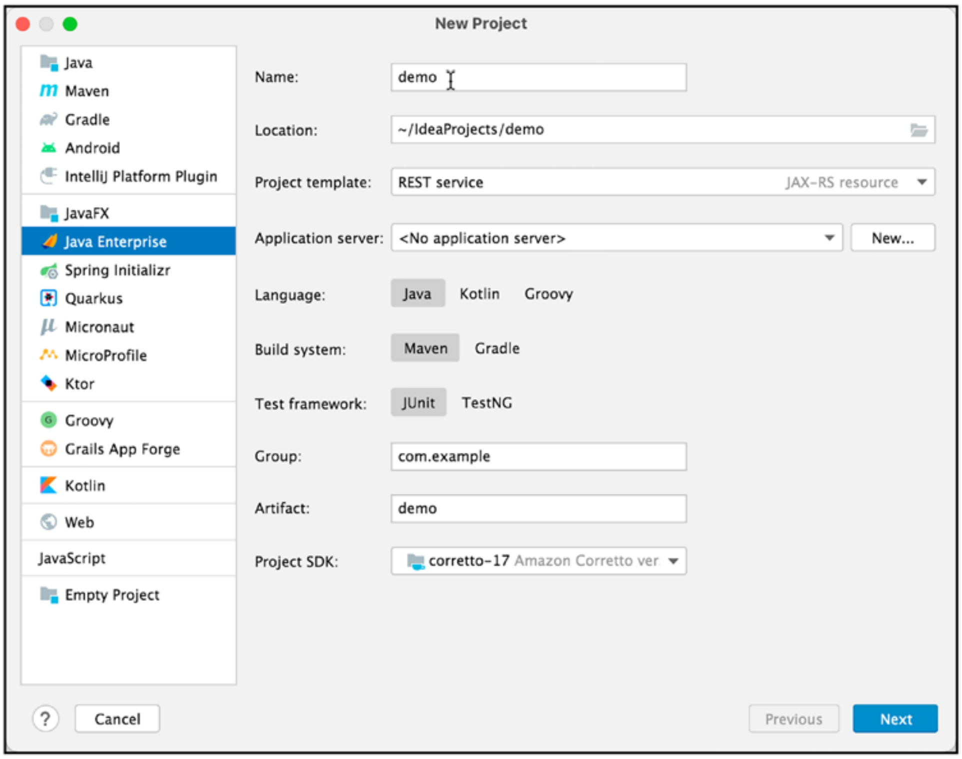
Runtime version: 11.0.12+7-b1504.40 aarch64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

Powered by open-source software
Copyright © 2000-2021 JetBrains s.r.o.

Figure C-2. *About*

Creating a web app

Now, going back to the project, click the *New Project* button (on the Welcome screen), then choose Java Enterprise (as shown in Figure C-3). We'll build a simple web application — Servlet and JSP — that's why we need to choose the "Java Enterprise" template.



Name: demo

Location: ~/IdeaProjects/demo

Project template: REST service JAX-RS resource

Application server: <No application server> New...

Language: Java Kotlin Groovy

Build system: Maven Gradle

Test framework: JUnit TestNG

Group: com.example

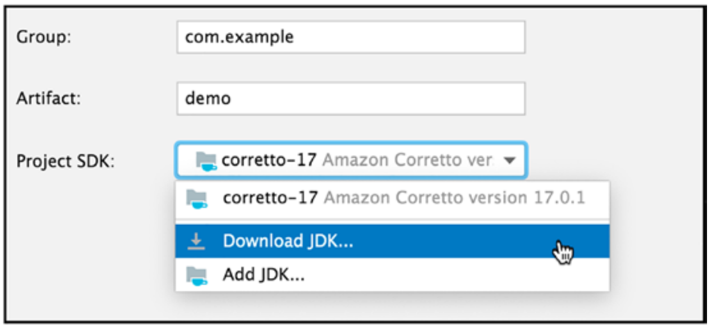
Artifact: demo

Project SDK: corretto-17 Amazon Corretto ver

Figure C-3. *New Project*

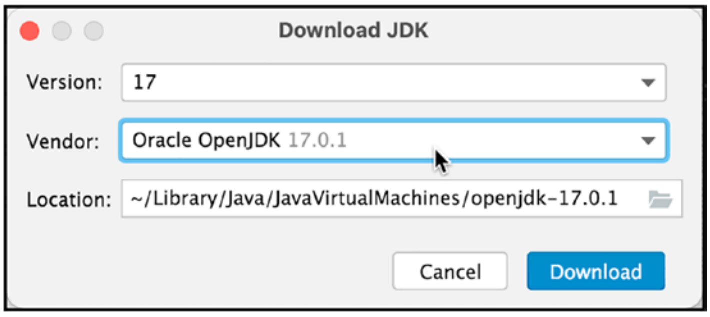
First, let's take care of the project SDK. Just like in our past projects, you can choose to use any of the Java SDKs you've got installed locally; or, you can choose to download a fresh one — which is what I'll do for this example.

Clicking the *Download JDK* option (as shown in Figure C-4) brings you to the *Download JDK* dialog (Figure C-5).



Group: com.example
Artifact: demo
Project SDK: corretto-17 Amazon Corretto ver

Figure C-4. *Project SDK*



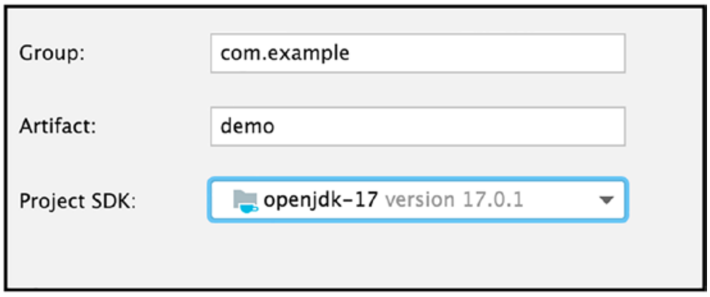
Download JDK

Version: 17
Vendor: Oracle OpenJDK 17.0.1
Location: ~/Library/Java/JavaVirtualMachines/openjdk-17.0.1

Figure C-5. *Download JDK*

Choose the SDK you'd like to use, then click the Download button.

Once the download is done, the Project SDK field will be set to your freshly downloaded SDK (Figure C-6).

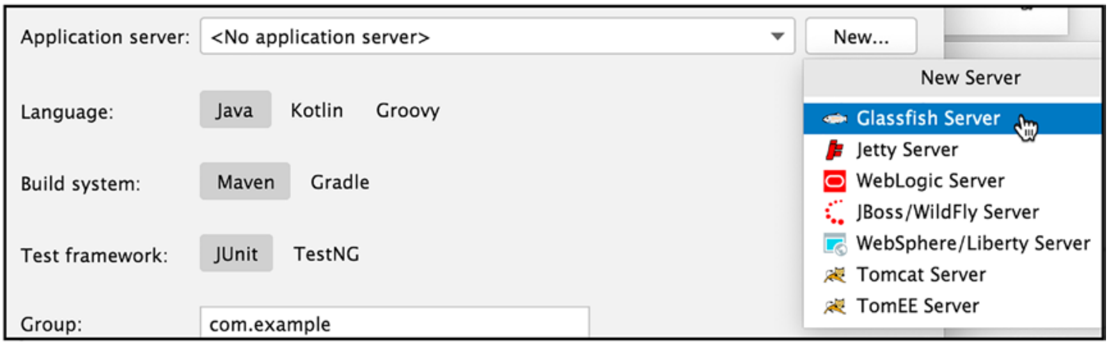


Group: com.example
Artifact: demo
Project SDK: openjdk-17 version 17.0.1

Figure C-6. *Project SDK set to OpenJDK 17*

We will need an application server to run our JakartaEE web app. There are plenty of choices for an application server, but for this example, we'll use GlassFish.

Click the *New* button right next to the *Application Server* field (as shown in Figure C-7), then choose GlassFish Server.

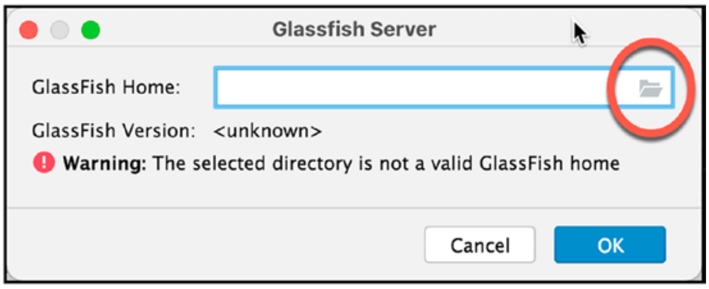


Application server: <No application server>
Language: Java
Build system: Maven
Test framework: JUnit
Group: com.example

Figure C-7. *Application Server*

The GlassFish Server dialog comes up (Figure C-8). We'll leave the installation for a while and download the GlassFish server. You can get download the GlassFish application server from `https://glassfish.org/downlload`. At the time of writing, GlassFish 6.2.2 is the current version.

Follow the download link. The file is a zipped archive. Unzip the file anywhere you prefer — just remember where you unzipped it.



Glassfish Server

GlassFish Home:

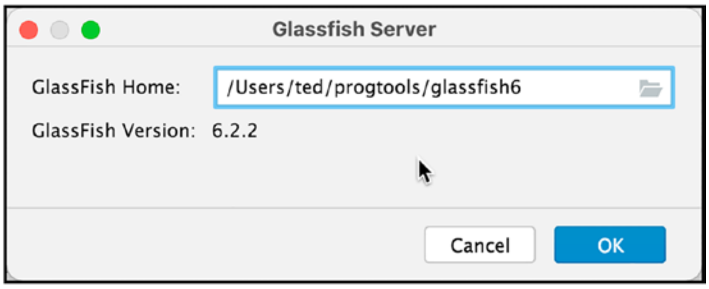
GlassFish Version: <unknown>

Warning: The selected directory is not a valid GlassFish home

Figure C-8. *GlassFish Server*

To set the GlassFish Home, click the folder icon (as shown in Figure C-8), then find the directory where you unzipped the GlassFish server. Set the GlassFish Home to that directory.

When the GlassFish Home is set, the GlassFish version appears on the dialog as well (Figure C-9).

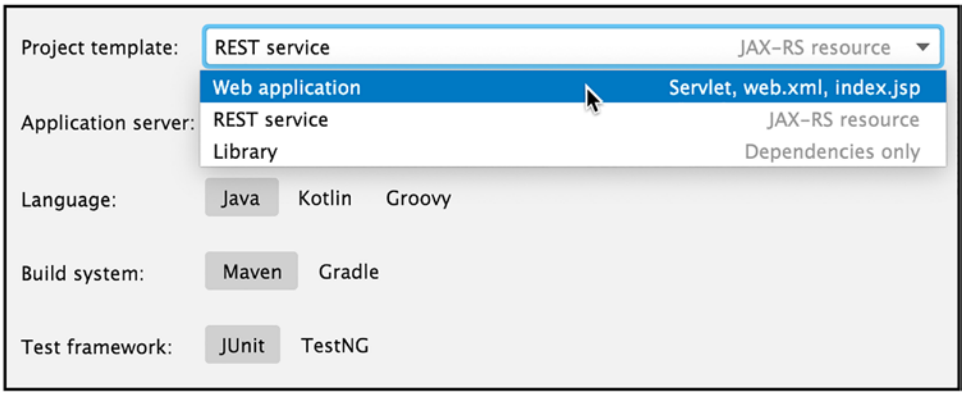


GlassFish Home: /Users/ted/progtools/glassfish6

GlassFish Version: 6.2.2

Figure C-9. *GlassFish Home*

Next, change the Project Template to "Web application", as shown in Figure C-10).


Web application Servlet, web.xml, index.jsp
REST service JAX-RS resource
Library Dependencies only
Kotlin
Groovy
Gradle
TestNG

Project template: REST service JAX-RS resource

Application server:

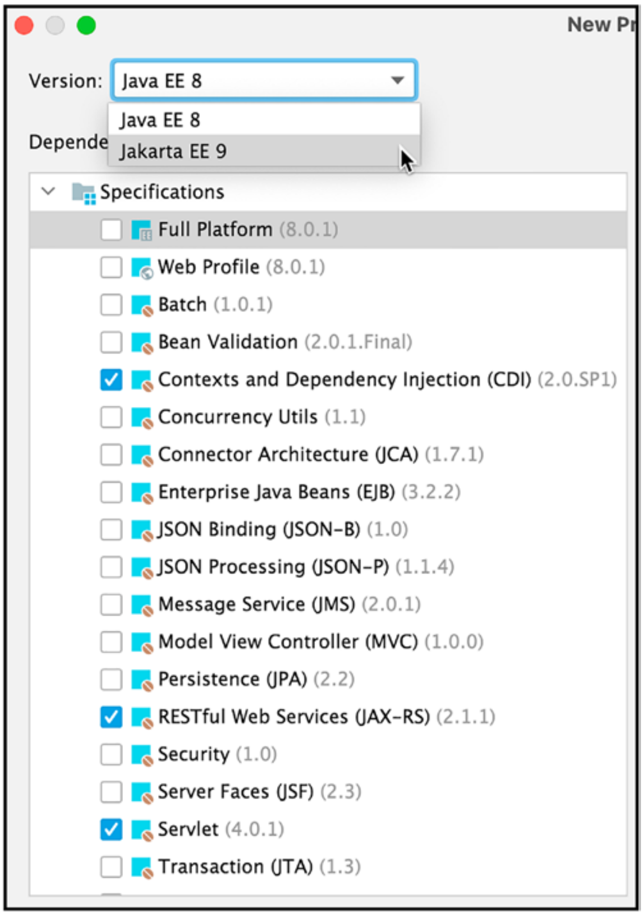
Language: Java

Build system: Maven

Test framework: JUnit

Figure C-10. *Project Template*

Then, change the version to the Jakarta EE 9, as shown in Figure C-11.


New Pr

Version: Java EE 8

Java EE 8
Jakarta EE 9

Depende

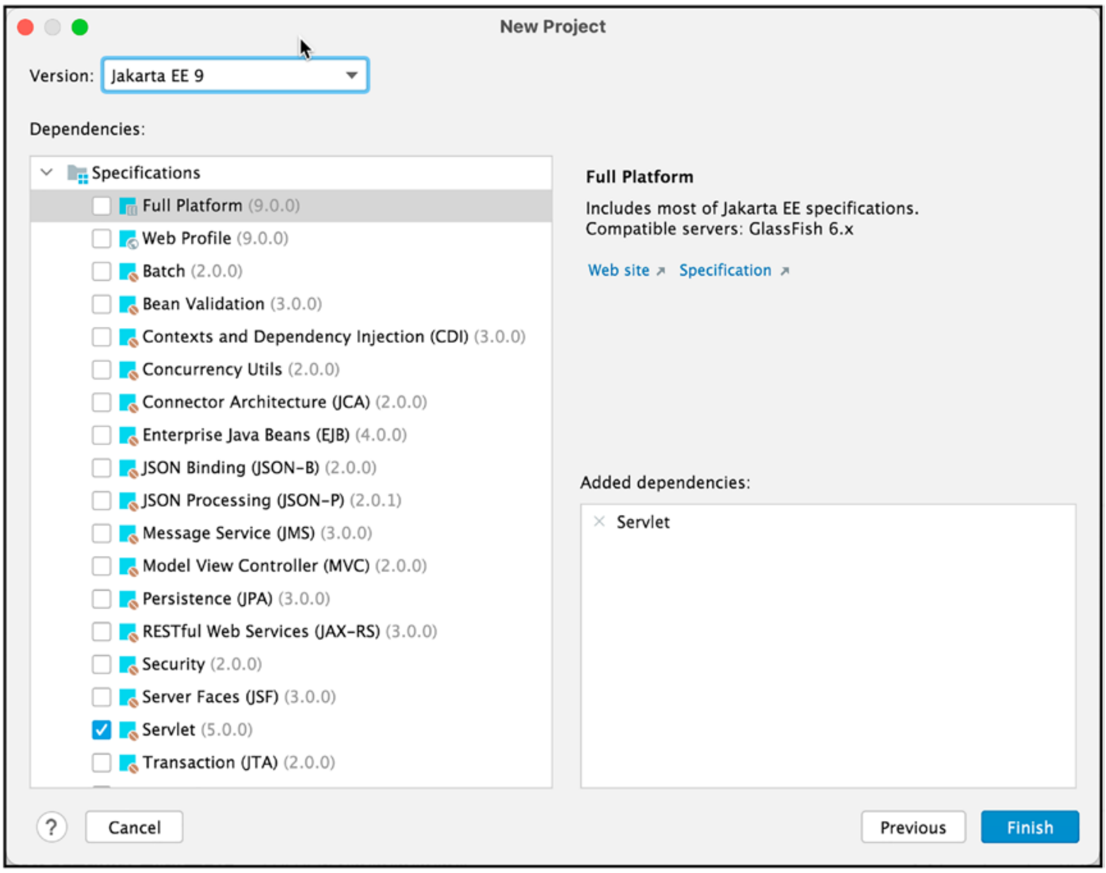
Specifications
- [ ] Full Platform (8.0.1)
- [ ] Web Profile (8.0.1)
- [ ] Batch (1.0.1)
- [ ] Bean Validation (2.0.1.Final)
- [✓] Contexts and Dependency Injection (CDI) (2.0.SP1)
- [ ] Concurrency Utils (1.1)
- [ ] Connector Architecture (JCA) (1.7.1)
- [ ] Enterprise Java Beans (EJB) (3.2.2)
- [ ] JSON Binding (JSON-B) (1.0)
- [ ] JSON Processing (JSON-P) (1.1.4)
- [ ] Message Service (JMS) (2.0.1)
- [ ] Model View Controller (MVC) (1.0.0)
- [ ] Persistence (JPA) (2.2)
- [✓] RESTful Web Services (JAX-RS) (2.1.1)
- [ ] Security (1.0)
- [ ] Server Faces (JSF) (2.3)
- [✓] Servlet (4.0.1)
- [ ] Transaction (JTA) (1.3)

Figure C-11. *JakartaEE version*

In the window that follows, you'll be able to set the libraries you'll need for the app. Since I already chose a web application template, the Servlet library was automatically chosen (Figure C-12).

Click the *Finish* button to proceed (Figure C-12).

this figure will be printed in b/w


New Project

Version: Jakarta EE 9

Dependencies:

- Specifications
  - [ ] Full Platform (9.0.0)
  - [ ] Web Profile (9.0.0)
  - [ ] Batch (2.0.0)
  - [ ] Bean Validation (3.0.0)
  - [ ] Contexts and Dependency Injection (CDI) (3.0.0)
  - [ ] Concurrency Utils (2.0.0)
  - [ ] Connector Architecture (JCA) (2.0.0)
  - [ ] Enterprise Java Beans (EJB) (4.0.0)
  - [ ] JSON Binding (JSON-B) (2.0.0)
  - [ ] JSON Processing (JSON-P) (2.0.1)
  - [ ] Message Service (JMS) (3.0.0)
  - [ ] Model View Controller (MVC) (2.0.0)
  - [ ] Persistence (JPA) (3.0.0)
  - [ ] RESTful Web Services (JAX-RS) (3.0.0)
  - [ ] Security (2.0.0)
  - [ ] Server Faces (JSF) (3.0.0)
  - [✓] Servlet (5.0.0)
  - [ ] Transaction (JTA) (2.0.0)

Full Platform

Includes most of Jakarta EE specifications.
Compatible servers: GlassFish 6.x

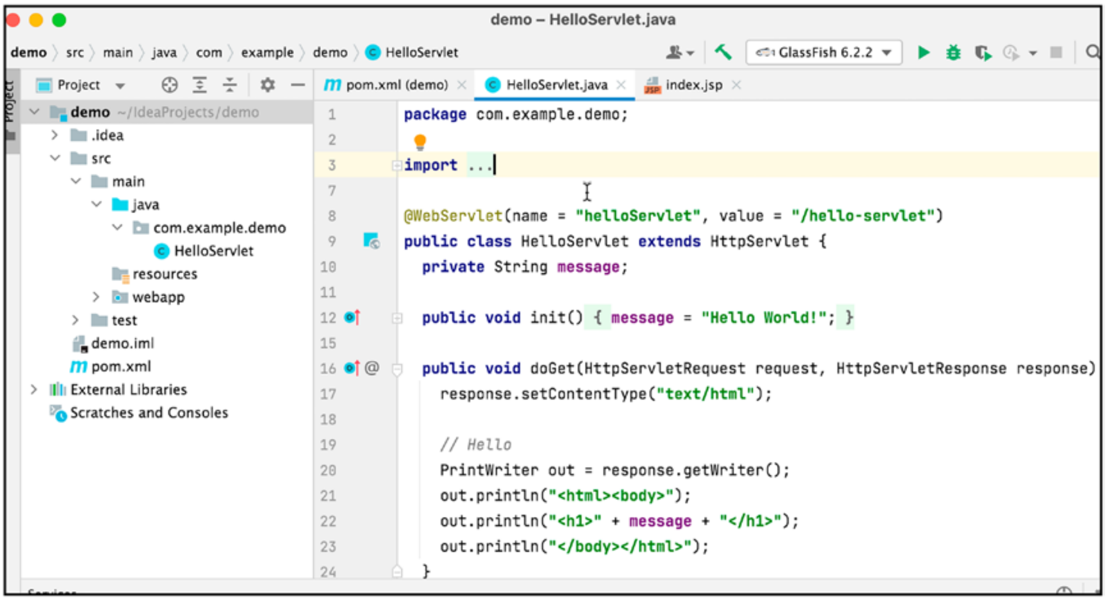
Web site ↗ Specification ↗

Added dependencies:

× Servlet

Figure C-12. *New Project*

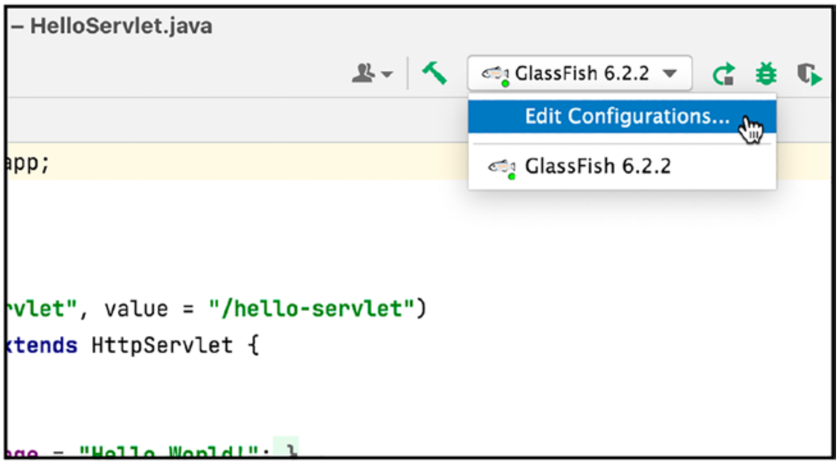
IntelliJ creates a starter web application for us. As you can see in the Project Tool window (Figure C-13), we've got a Servlet, a JSP, and a web configuration file (WEB-INF/web.xml).



```
package com.example.demo;

import ...

@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>");
  }
```

Figure C-13. *Project files*

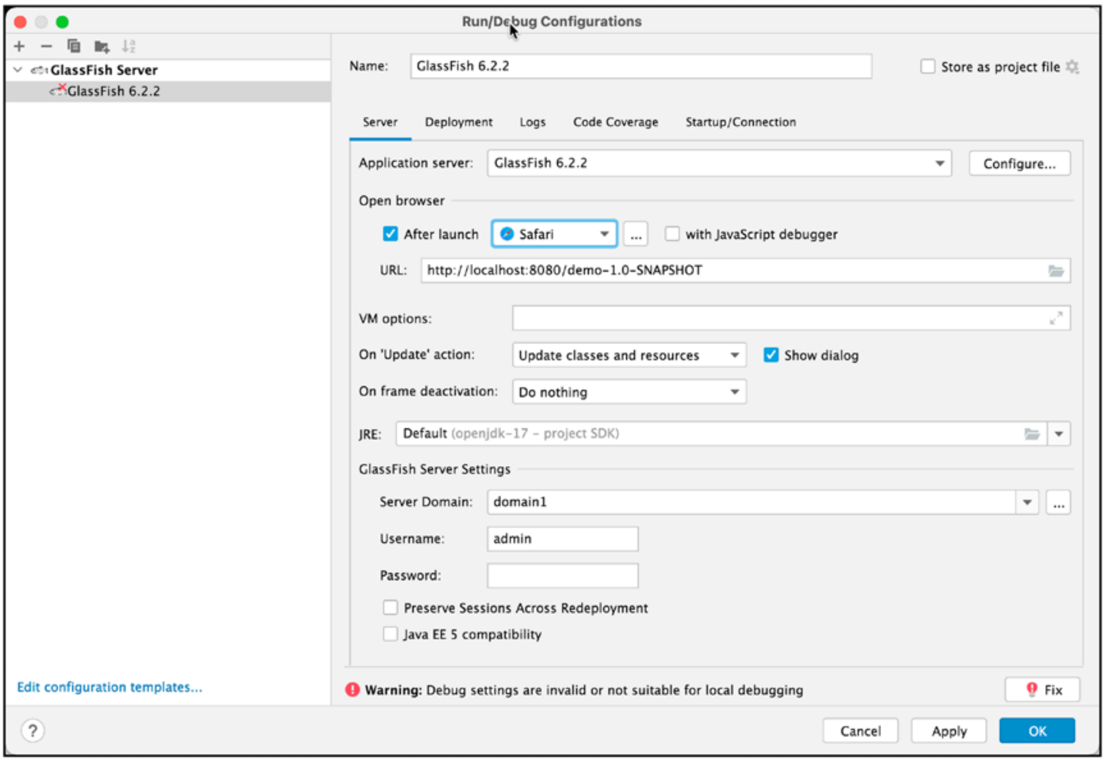
Next thing to do is to configure the GlassFish application server. Application server dropdown (Figure C-14), then choose *Edit Configurations*.



```
pp;

vlet", value = "/hello-servlet")
tends HttpServlet {
```

Figure C-14. *Edit Configurations*

You really don't need to do much in here, but note that I've got a warning — lower portion of the Run/Debug Configuration dialog (Figure C-15). It says, "Debug settings are invalid or not suitable for local debugging" — I need to fix that. Just click the "Fix" button right next to it.


Run/Debug Configurations

GlassFish Server
GlassFish 6.2.2

Name: GlassFish 6.2.2
Store as project file

Server | Deployment | Logs | Code Coverage | Startup/Connection

Application server: GlassFish 6.2.2
Open browser
[✓] After launch: Safari
[ ] with JavaScript debugger
URL: http://localhost:8080/demo-1.0-SNAPSHOT
VM options:
On 'Update' action: Update classes and resources
[✓] Show dialog
On frame deactivation: Do nothing
JRE: Default (openjdk-17 - project SDK)
GlassFish Server Settings
Server Domain: domain1
Username: admin
Password:
[ ] Preserve Sessions Across Redeployment
[ ] Java EE 5 compatibility

Edit configuration templates...

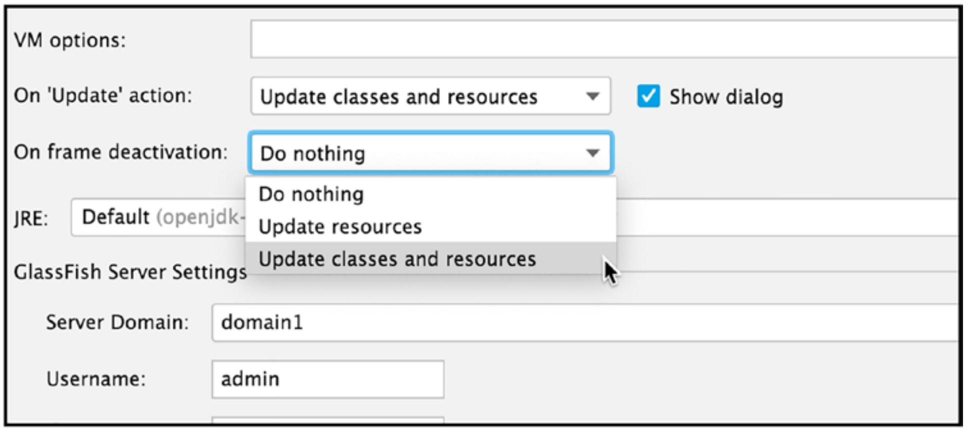
Warning: Debug settings are invalid or not suitable for local debugging

Figure C-15. *Run/Debug Configurations*

The way my server is configured, it will;

1. Launch Safari after completely launching the application server
2. It will open the browser and go to the URL `http://localhost:8080/demo-1.0-SNAPSHOT` - the demo-1.0-SNAPSHOT is from the Maven POM file, which was automatically generated by the project

The last configuration item I want to change is the “on frame deactivation” (Figure C-16). I’m setting it to “Update classes and resources” so that whenever I take the focus away from IntelliJ (say, when I click on the browser), the classes and other resources will automatically refresh.

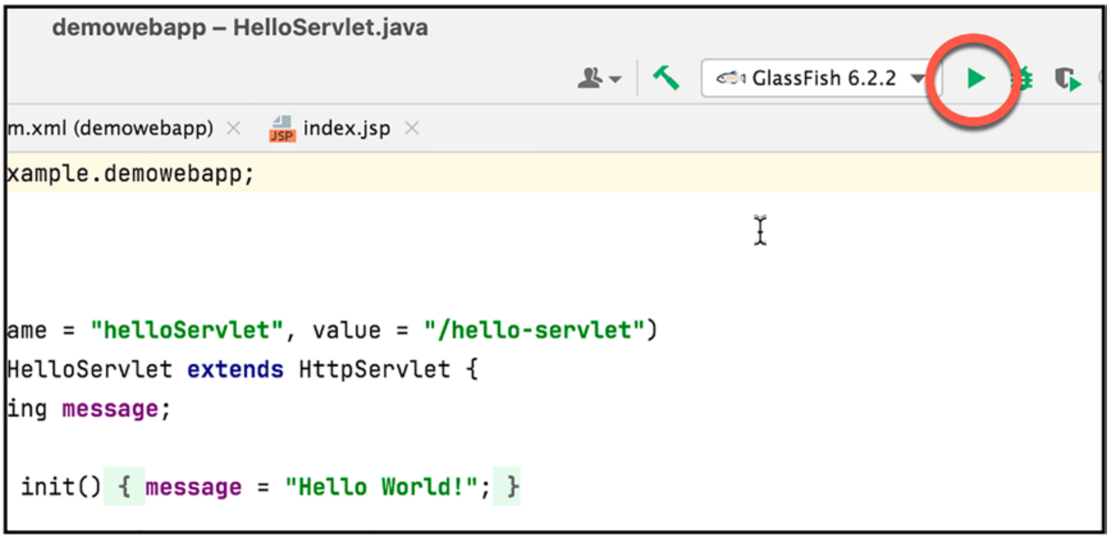


VM options:
On 'Update' action: Update classes and resources ✓ Show dialog
On frame deactivation: Do nothing
JRE: Default (openjdk-
GlassFish Server Settings
Server Domain: domain1
Username: admin

Figure C-16. *On Frame deactivation*

Click the *OK* button to save the changes

Now, Run the app so we can see how it looks like.

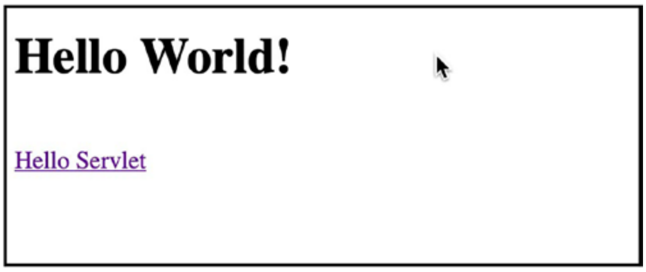


demowebapp – HelloServlet.java

```
xample.demowebapp;



ame = "helloServlet", value = "/hello-servlet")
HelloServlet extends HttpServlet {
ing message;

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

Figure C-17. *Run*

Figure C-18 shows our Hello World app in its glorious runtime.


Hello World!

Hello Servlet

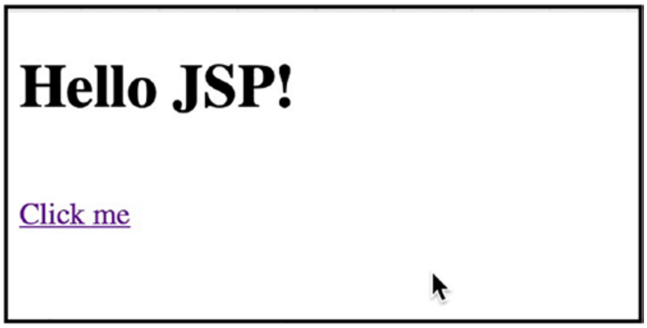
Figure C-18. *hello.jsp*

Now, let's make some changes to the code. Our app has a JSP file that is designed to respond to GET requests. Let's edit it to match the code in Listing C-1.

Listing C-1. src/main/webapp/index.jsp

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

Since we changed the configuration of our GlassFish server to update classes and resources whenever we take the focus away from the IDE, all we have to do is refresh the browser — no need to stop and restart the app server.



Hello JSP!

Click me

Figure C-19. *hello.jsp*

Key Takeaways

- IntelliJ Ultimate Edition makes JakartaEE app development a breeze
- IntelliJ doesn’t come with application servers. You’ve got to install them separately and configure them within your projects