JavaOne Hands-On Lab

LAB-4520

Plug into GlassFish v3 with JavaServer™ Faces and jMaki

LAB-4520: Plug into GlassFish v3 with JavaServer™ Faces and jMaki

Approximate duration: 120 minutes

Contacts: Ken Paulsen, Anissa Lam, Ana Caballero

Last Updated: April 20, 2008

Glassfish Application Server has been widely adopted by many companies as well as individuals. This broad user base means there is a wide variety of needs from GlassFish, both in terms of the services it provides, as well as the management of those services. GlassFish v3 addresses the wide range of requirements through a modular architecture, which allows server components such as EJB, WebServices, JDBC, as well as numerous frameworks to be added and removed from the server as needed.

The Administration Console for GlassFish has not been left out when it comes to plugability. In order to manage a dynamic application server like GlassFish v3, the administration console must itself be very dynamic. This is a big challenge for a JavaServer™ Faces web application. How are resources (pages, images, etc.) discovered when the bits are not available at install time? How do you accommodate multiple "plugins" which want to contribute to the same page?

In this lab, we will explore these issues and learn how creating a pluggable JavaServer™ Faces application is addressed in GlassFish v3 Administration Console. You will learn how you may create a plugin for GlassFish v3 which integrates seamlessly with the administration console. This lab will guide you through the process of adding new tree nodes to the GlassFish navigation tree, adding new JavaServer Faces pages (including a jMaki Chart), and will show you how to bundle it together in a GlassFish v3 plugin module. Since GlassFish is an Open Source, Open Community application server, all the source code is available -- no secrets! Better yet, GlassFish is an Open Community, we welcome everyone to add their own custom GlassFish v3 plugins to the community!

Your completed lab will look similar to the image below:

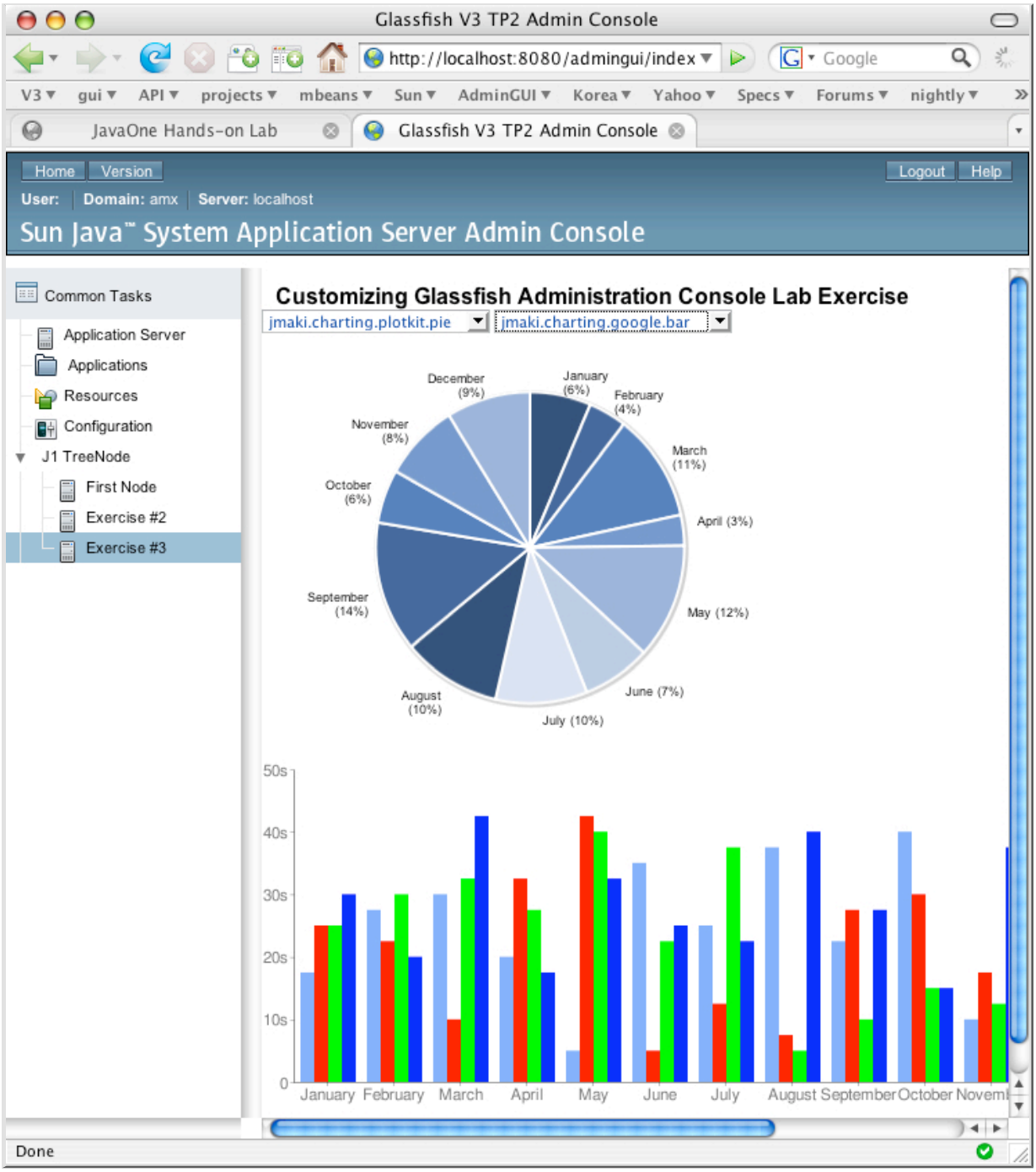


Figure intro-1: FireFox showing the completed lab.

Prerequisites

This hands-on lab assumes you have some basic knowledge or programming experience on the following technologies.

- Java™

Helpful Knowledge

- GlassFish
- JSFTemplating / Facelets
- JSF
- NetBeans
- Subversion
- Maven

System Requirements

- Supported OS: Windows, Solaris™, Linux, MacOS
- Memory requirement: 512M minimum, 2GB recommended
- Disk space requirement: 500 MB

Software needed for the lab

The following set of software will be used in this lab. This software should have been already made available to you, however, if you are setting up the environment yourself you will need to have the following software. If you have any questions on installation, please feel free to send questions to the email alias mentioned below.

- Java SE JDK™ 6
- NetBeans IDE 6.0 or higher
  - On the download page, choose either the Web & Java™ EE download or the All download.
  - The instruction of how to build and deploy the lab exercises in this lab is provided as optional exercise #0.
- Download and unzip lab zip file lab zip file (4520_gfplugin.zip) under a directory of your choice, <lab_root>.
  - (Optional) Read the lab presentation
  - Read <lab_root>/gfplugin/index.html (this document) to proceed

  The latest version of this zip file is available at JavaOne Online.

Notations used in this documentation

- <lab_root> - directory into which lab zip file is unzipped
  - This document uses <lab_root> to denote the directory under which you have unzipped the lab zip file of this lab. The name of the lab zip file is **4520_gfplugin.zip**.
  - Once you unzipped the lab zip file under <lab_root>, it will create a subdirectory called **gfplugin** For example, under Windows, if you have unzipped the lab zip file in the root of drive **E:**, it will create **E:\gfplugin.** Under Linux/Solaris, if you have unzipped the lab zip file in the **$HOME** directory, it will create **$HOME/gfplugin** directory.

Lab exercises

- Exercise 0: Configuring your environment
- Exercise 1: Plugging into GlassFish
- Exercise 2: Charting with jMaki
- Exercise 3: Templating your Plugin

Resources

- Lab Presentation
- GlassFish v3
- GlassFish Admin Console Plugin Documentation
- JSFTemplating
- jMaki
- jMaki Charting
- HK2

Where to send questions or feedbacks on this lab and public discussion forums

- You can send technical questions via email to the authors of this Hands-on lab (and experts on the subject) or you can post the questions to the web.
  Please post questions that are relevant only to this hands-on lab.
  - JavaOne 2008 Hands on Lab Forum
- You can send your other questions to the public mailing lists and forums.
  - GlassFish Forums
  - GlassFish Email Lists (admin, users, or dev)
  - JSFTemplating Email List
  - jMaki Email List

Exercise 0: Configuring your environment

Step-by-Step Instructions:

The following steps will guide you through setting up the environment necessary to complete this lab. This consists of the following steps:

1. **Install GlassFish v3.**
2. **Install Maven.**
3. **Configure NetBeans.**

Steps:

1. **Install GlassFish v3.**
   a. GlassFish v3 is already installed when you unzip the lab .zip file, it can be found at: `gfplugin/glassfish/`. Please use this copy of GlassFish as it has been tested with this lab.

2. **Install Maven.**
   a. Download Maven 2 from http://maven.apache.org/download.html. This page also contains instructions you should follow to install maven on your machine, please follow these instructions.
   A copy of maven 2.0.9 is already installed when you unzip the lab .zip file, it can be found at : `gfplugin/maven2/maven-2.0.9`.
   You can set your MAVEN_HOME to this directory and add `gfplugin/maven2/maven-2.0.9/bin` to your PATH.

3. **Configure NetBeans** (only if you use NetBeans).
   a. Install NetBeans. (Skip if you already have NetBeans 6+ installed.)
      i. Download NetBeans from http://www.netbeans.org.
      ii. Follow the installation instructions on the NetBeans website to install NetBeans.

   b. Launch NetBeans 6.
      i. On Windows: **Start --> Programs --> NetBeans IDE --> NetBeans IDE 6.0 --> NetBeans IDE**
      ii. On Solaris™ or Linux: open a terminal window, and type:

```
netbeans
```

   c. Ensure the Maven 2 plugin for NetBeans 6.0 is installed.
   This plugin is required in order to open the project in NetBeans 6.0. The Maven 2 plugin is available in the `gfplugin/maven2/maven-nb-plugins` directory.
      i. Choose **Plugins** from the **Tools** menu.
      ii. Select the **Downloaded** tab.
      iii. Press the **Add Plugins...** button.

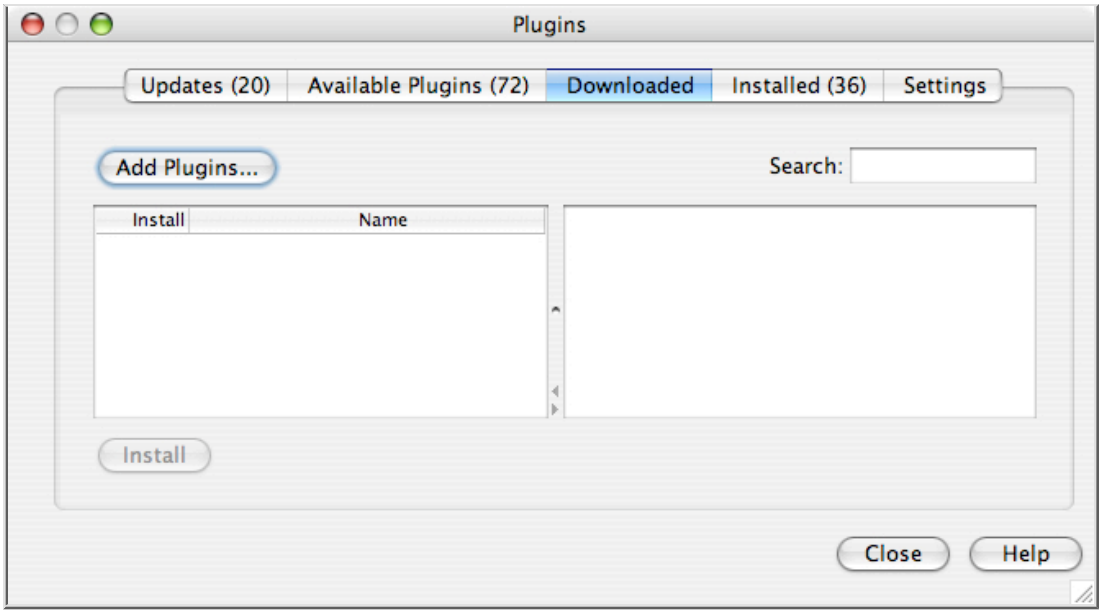


Figure 0-1: "Add Plugins..." Button

iv. Navigate to `gfplugin/maven2/maven-nb-plugins` directory.
v. Select all the .nbm files and Press **Open** button.

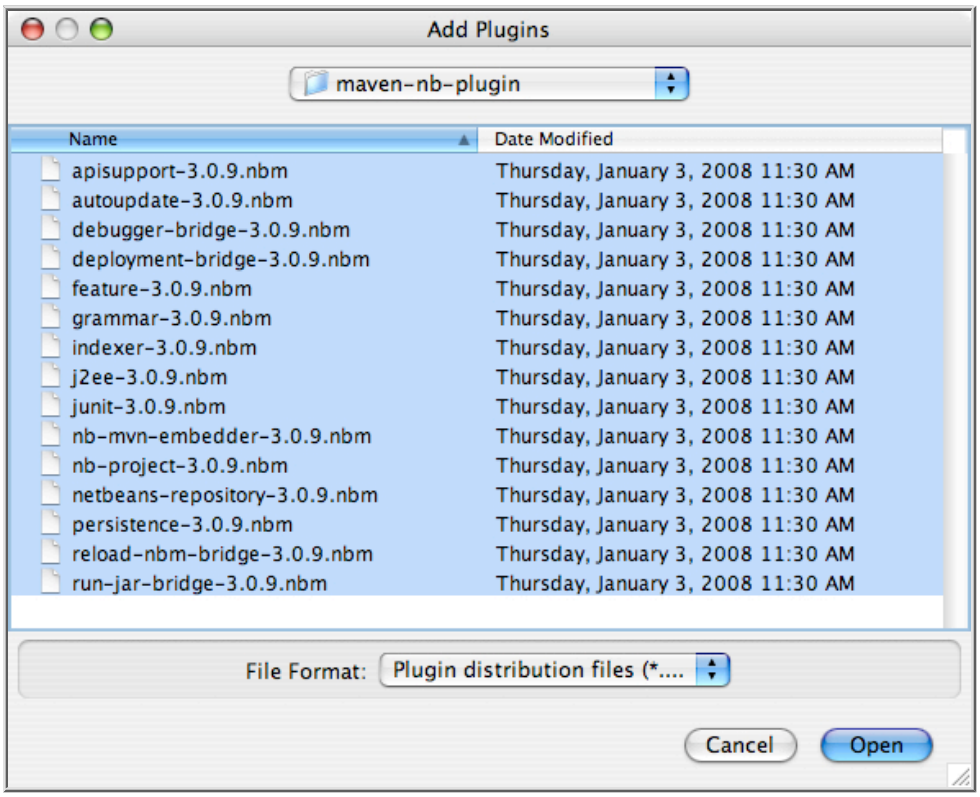


Figure 0-2: Select all .nbm files

vi. Press the **Install** button to start the installation.

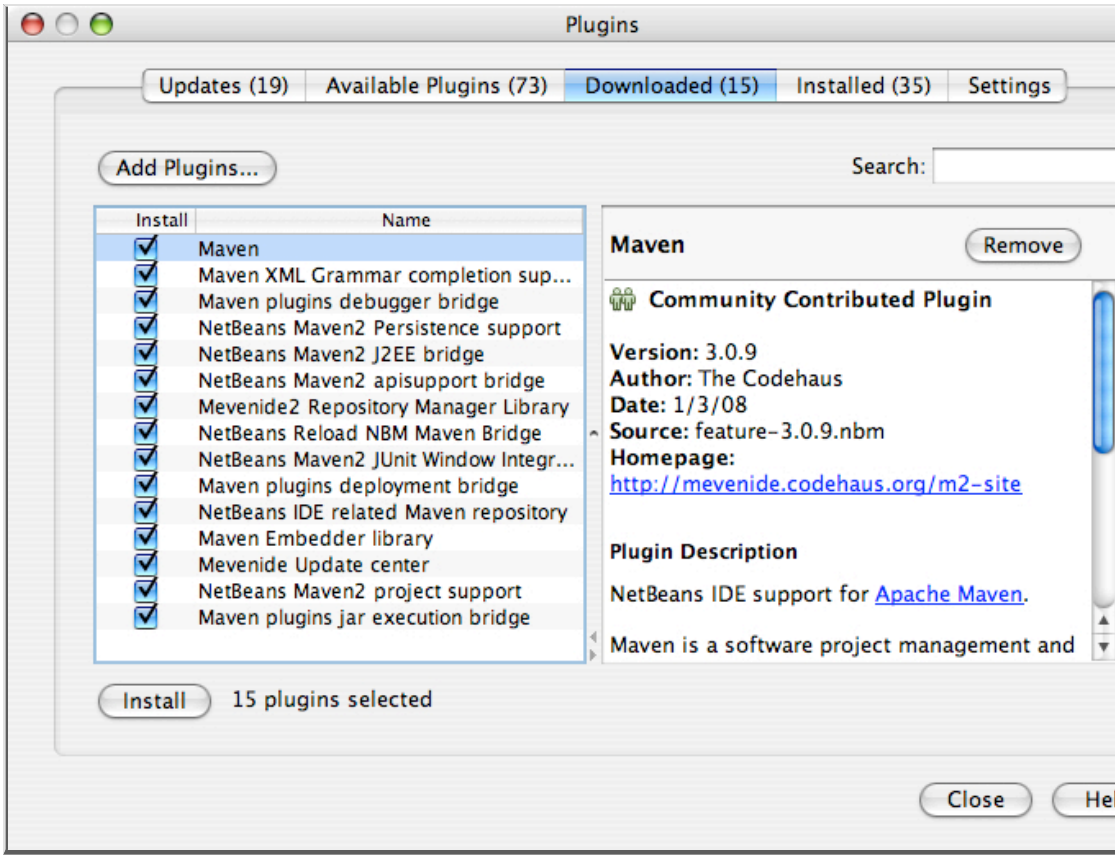


Figure 0-3: Press "Install" button

Additional Information:

- For more information regarding the GlassFish v3 development environment and workspace, refer to the build instructions at: http://wiki.glassfish.java.net/Wiki.jsp?page=V3FullBuildInstruction.

Go to the next exercise

Exercise 1: Plugging into GlassFish

Approximate duration: 45 minutes

Introduction:

The goal of Exercise 1 is to develop a GlassFish plugin module. You will create a GlassFish plugin that will add tree nodes and a page to the GlassFish Administration Console. When you complete this exercise, your plugin will look like this:

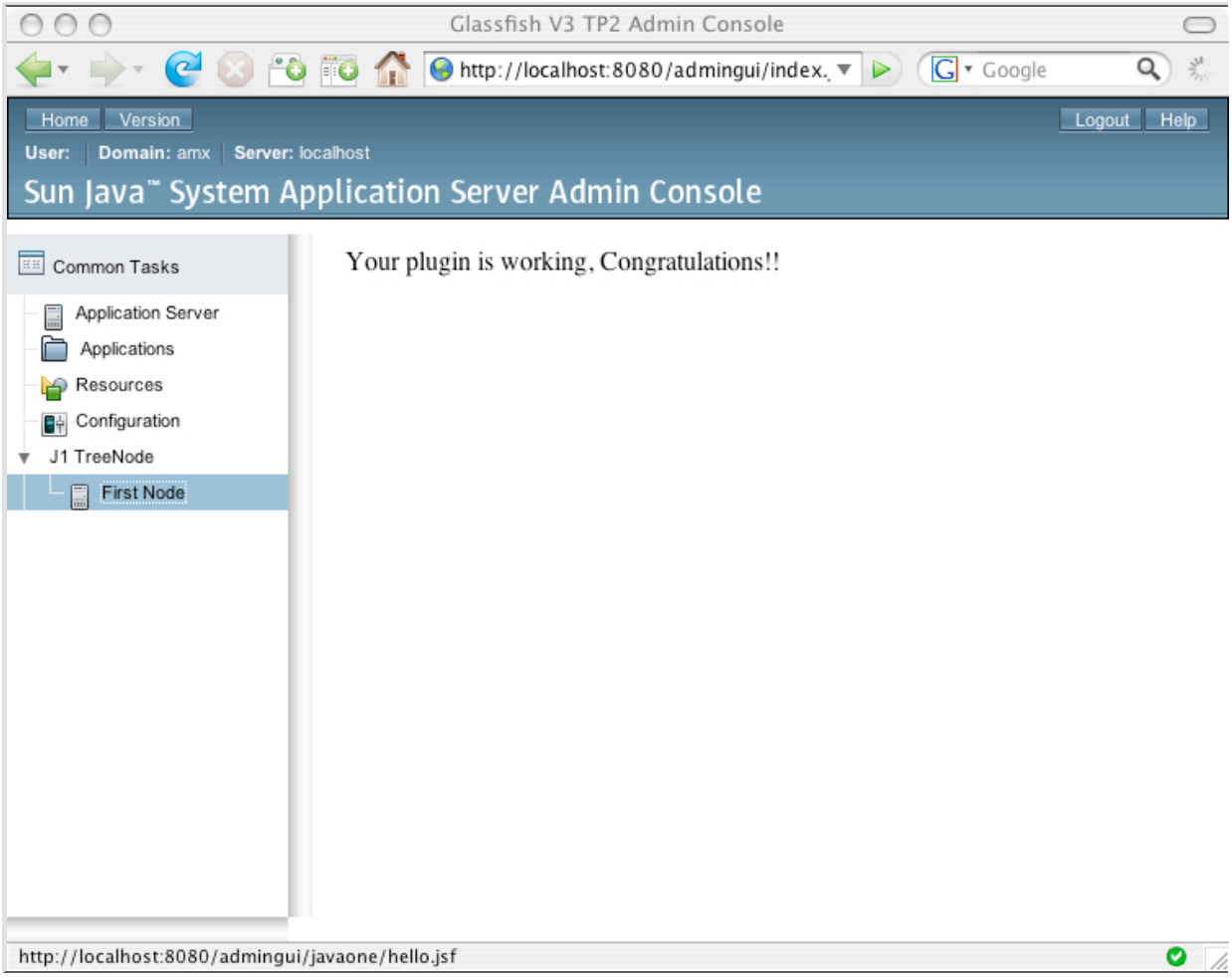


Figure 1-1: Exercise 1 Completed

In this exercise, you will learn how to do the following:

- Build a GlassFish v3 plugin module.
- Integrate a plugin module into GlassFish v3 to extend the Administration Console.
- Access the GlassFish v3 Administration Console and see your plugin.

Background information:

This section provides detail about the GlassFish v3 build environment and architecture. This information is non-essential for completing this lab, however, it will give you better insight into the design of GlassFish v3. To proceed directly to the lab steps, continue to the "Step-by-Step Instructions" section.

GlassFish (HK2) Modules

GlassFish uses the HK2 project (https://hk2.dev.java.net) to provide "modules" which can be plugged together to build an application server with unique capabilities. These GlassFish (or HK2) modules need only be placed in the <install-root>/modules directory to be found by the environment. HK2 modules are OSGi bundle compliant. GlassFish recently announced that it can run on top of an OSGi

container, these plugin modules will also be supported as OSGi bundles. These modules describe themselves to GlassFish so that it knows how to interact with them. This is done using HK2 @Contract and @Service annotations.

The @Contract annotation is typically placed on a Java Interface. Doing this makes the interface known to HK2. Here is an example Java Interface annotated with the @Contract annotation:

```
@Contract
public interface ConsoleProvider {
    /**
     *  <p> Returns a <code>URL</code> to the <code>console-config.xml</code>
     *      file, or <code>null</code>.  If <code>null</code> is returned, the
     *      default (@{link #DEFAULT_CONFIG_FILENAME}) will be used.</p>
     */
    public URL getConfiguration();

    /**
     *  <p> The default location of the <code>console-config.xml</code>.</p>
     */
     public String DEFAULT_CONFIG_FILENAME   =
             "META-INF/admingui/console-config.xml";
}
```

Figure 1-2: ConsoleProvider Interface

The @Service annotation is placed on a Java Class which typically implements a Java Interface which is annotated by the @Contract annotation. When @Service annotations are processed by the Annotation Processor Tool at compile-time, the HK2 processor creates an entry in the META-INF/inhabitants/default file, such as:

```
class=org.example.glassfish.admingui.MyPlugin,index=org.glassfish.api.admingui.ConsoleProvider
```

Figure 1-3: META-INF/inhabitants/default

This is used by HK2 to efficiently locate all implementations of a @Contract in the system.

Lets consider an example of how this is used. The GlassFish Administration Console needs to find GlassFish Modules that wish to integrate content. To do this, each of these GlassFish Modules must provide an implementation of the ConsoleProvider "@Contract" Interface which is annotated with the @Service annotation. The GlassFish Administration Console is then able to easily find these implementations and access the configuration file they specify via their interface (see figure 1-2). Once the configuration information is obtained, the task of integrating the new content is easily performed.

You can find more information about this architecture at the GlassFish or HK2 web sites:

- https://glassfish.dev.java.net
- https://hk2.dev.java.net

The exercises in this lab aim to describe in detail the specific steps required to create a module and to integrate with the GlassFish Administration Console. Complete this lab to gain a better understanding of these steps.

Source code system layout

This section of the document describes the file layout found in the lab exercises, as well as the layout of all GlassFish Modules. To obtain the latest GlassFish source code, you can follow the instructions found on the GlassFish v3 build instructions web site: http://wiki.glassfish.java.net/Wiki.jsp?page=V3FullBuildInstructions.

When viewing the top-level directory of the GlassFish source, you should note all the sub-directories are Maven Modules, which follow a standard maven 2 project layout. The java source files are under `<maven-module>/src/main/java`. Additional resources are under `<maven-module>/src/main/resources`. Unit tests are under `<maven-module>/src/test/java`. There is a pom.xml file for each maven module that specifies information such as: parent Maven Module, groupId, artifactId, version number, build dependencies, and build and packaging instructions. Configuration information from a parent Maven Module is inherited.

The source code for the GlassFish Administration Console is located in the directory named "admingui". This lab does not touch any of the files in the GlassFish Administration Console, it instead provides a plugin which is integrated seamlessly with the GlassFish Administration Console. If you want to know more details about the GlassFish Administration Console, below is an explanation of all the admingui sub-modules. The specifics of the Administration Console itself are, however, outside the scope of this lab. You may become involved the the GlassFish Administration Console team or find out more information by going to https://glassfish.dev.java.net or emailing admin@glassfish.dev.java.net.

Plugin Service Module

This module detects all GlassFish Modules that provide `IntegrationPoints` to the console. It also contains code to discover and serve all additional resources provided by other plugin modules (JSFTemplating JavaServer Faces Pages, Images, Java code, etc.).

```
v3/admingui/plugin-service/pom.xml
v3/admingui/plugin-service/src/main/java/org/glassfish/admingui/plugin/ConsoleConfig.java
v3/admingui/plugin-service/src/main/java/org/glassfish/admingui/plugin/ConsolePluginService.java
v3/admingui/plugin-service/src/main/java/org/glassfish/admingui/plugin/IntegrationPoint.java
```

Figure 1-4: Some plugin-service files

Core Module for Administration Console
This module provides infrastructure for the GlassFish Administration Console. It contains code for initialization, setting up the home page, building the navigation tree, common utilities, etc. This includes the JavaServer Faces pages such as the common task page, header, version window, top-level tree nodes and their corresponding pages.

```
v3/admingui/pom.xml
v3/admingui/core/src/main/java/com/sun/enterprise/tools/admingui/handlers
v3/admingui/core/src/main/resources/home.jsf
```

Figure 1-5: Some admingui/core files

Administration Console Web Application

This is the war packaging module which contains the deployment descriptor and includes the `console-core-1.0-SNAPSHOT.jar` which is built from the core module. After the build cycle, a war file is created ready for deployment.

```
v3/admingui/war/
v3/admingui/war/src/main/webApp
```

Figure 1-6: Some admingui/war files

Step-by-Step Instructions:

Overview:

In this exercise, you will complete the following:

A. **Implement the ConsoleProvider @Contract**

In this step, you will create a Java Class which implements the `ConsoleProvider @Contract`. This is done so your plugin can be discovered by the GlassFish v3 Administration Console. Implementing the `ConsoleProvider` interface is simply a matter of annotating your class with the `@Service` annotation and implementing its single method.

B. **Provide a console-config.xml file**

This file is necessary to declare your "`IntegrationPoints`." In this step, you will learn what an `IntegrationPoint` is and how to define them.

C. **Create a couple JavaServer Faces pages**

JavaServer Faces pages are needed to produce the content you are providing. You will create 2 simple JSFTemplating JavaServer Faces pages.

D. **Build and Install your first GlassFish v3 plugin**

In this step you will install and test your plugin.

Steps:

A. Implement the ConsoleProvider @Contract

In this step, you will create a Java Class which implements the `ConsoleProvider @Contract`. This is done so your plugin can be discovered by the GlassFish v3 Administration Console. Implementing the `ConsoleProvider` interface is simply a matter of annotating your class with the `@Service` annotation and implementing its single method.

For more background information, please see the GlassFish (HK2) Modules section above. Or the HK2 Component Model web page.

If you are not using NetBeans, skip to step A-2 "Create your plugin Java @Service Class".

1. **Open the plugin module in NetBeans**
   a. Launch NetBeans IDE 6.0.
   b. Open plugin module.
      i. Choose **Open Project** from the **File** menu.
      ii. Browse to **gfplugin** and select **myplugin**.
      iii. Press the **Open Project** button.

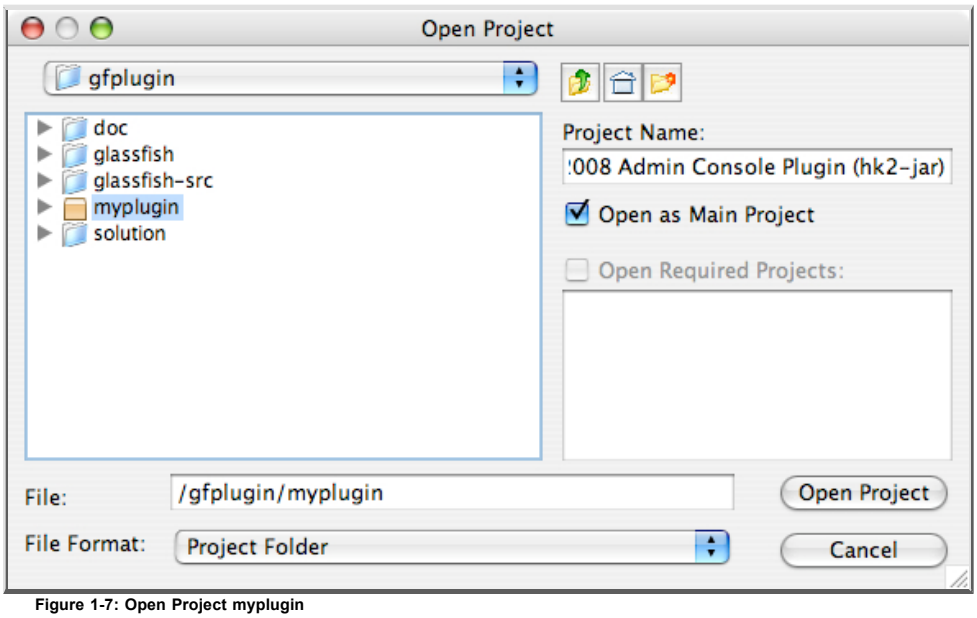


Figure 1-7: Open Project myplugin

c. You should see the following directory layout under the Projects tab:

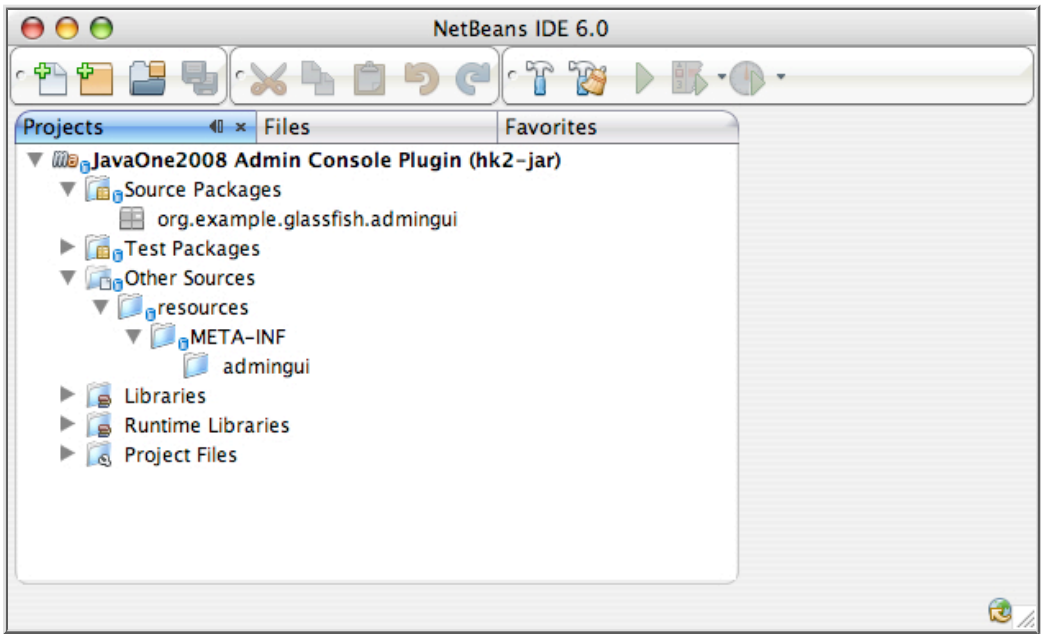


Figure 1-8: Project myplugin layout

2. **Create your plugin Java @Service Class**

   We need a Java Class annotated with the `@Service` annotation which implements `ConsoleProvider`. `ConsoleProvider` is an interface that specifies the method `getConfiguration()`. This method returns a `java.net.URL` pointing to your `config-console.xml` file (you have not yet created this file). If the method returns `null`, the plugin framework will use a default configuration file name: `META-INF/admingui/console-config.xml`.

   For more background information, please see the GlassFish (HK2) Modules section above. Or the HK2 Component Model web page.

   If you are not using NetBeans: create a file similar to that found in Figure 1-10 with any filename or package you like. The lab material, however, will refer to this file as `MyPlugin.java` and the package will by named `org.example.glassfish.admingui`. Skip to step A-3 "Compiling your plugin".

```
package org.example.glassfish.admingui;

import org.glassfish.api.admingui.ConsoleProvider;
import org.jvnet.hk2.annotations.Service;
import java.net.URL;


/**
 *  <p> Returns the URL to <code>console-config.xml</code> where
 *      <code>IntegrationPoint</code>s are defined.</p>
 *
 *  @return URL of configuration file.
 */
@Service
public class MyPlugin implements ConsoleProvider {
    public URL getConfiguration() { return null; }
}
```

Figure 1-10: Edit MyPlugin.java

NetBeans users, follow these steps:

a. Locate the package "**org.example.glassfish.admingui**" under "**Source Packages**".

b. Right click on it and select "**New --> Java Class**". A pop up window will display.

c. Enter the class name "**MyPlugin**" (do **not** enter the ".java" extension).

d. Select **Finish**.

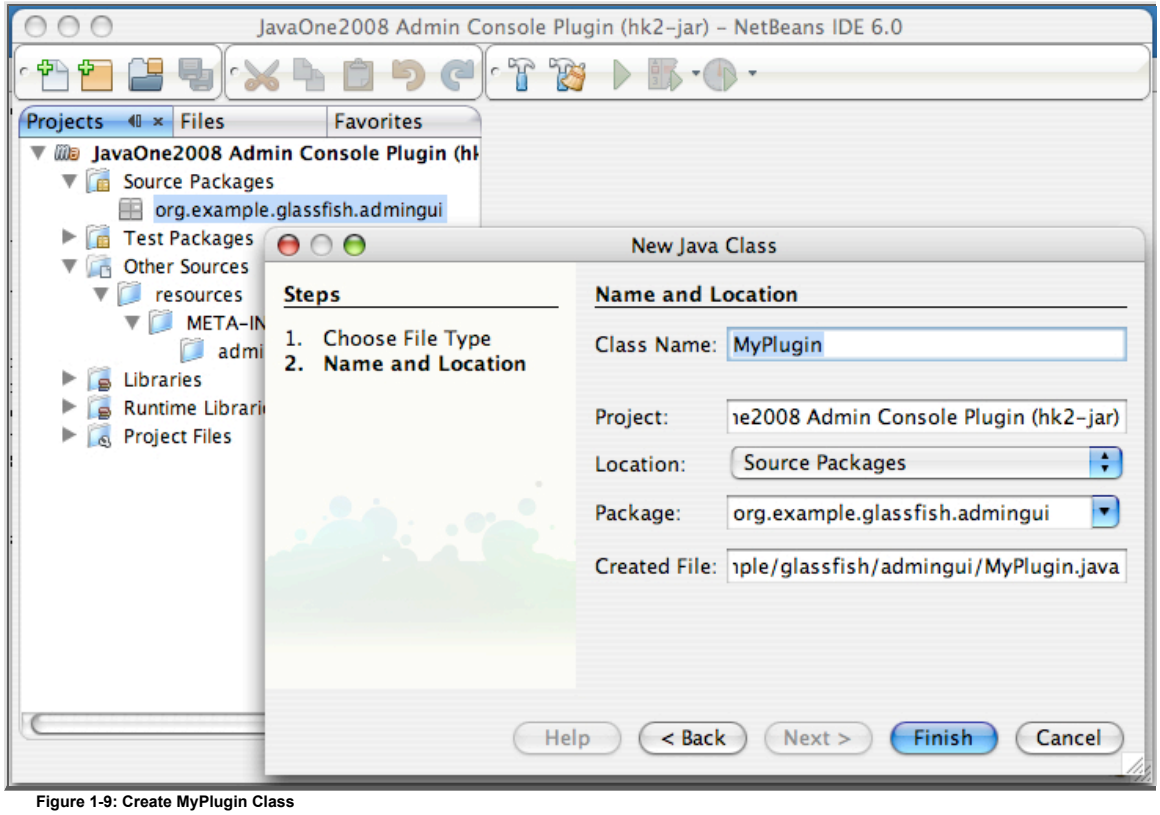


Figure 1-9: Create MyPlugin Class

e. Now open this newly created Java File and add the necessary implementation for the plugin. (Note: This file should be located at: `gfplugin/myplugin/src/main/java/org/example/glassfish/admingui/MyPlugin.java`.)

The important elements for a `ConsoleProvider` implementation are:

i. The @Service Annotation.

ii. Implementation of the ConsoleProvider interface.

iii. A getConfiguration() method that returns a URL to the console-config.xml file. (Note: If null is returned,

"META-INF/admingui/console-config.xml" will the used.

Your implementation will look like this:

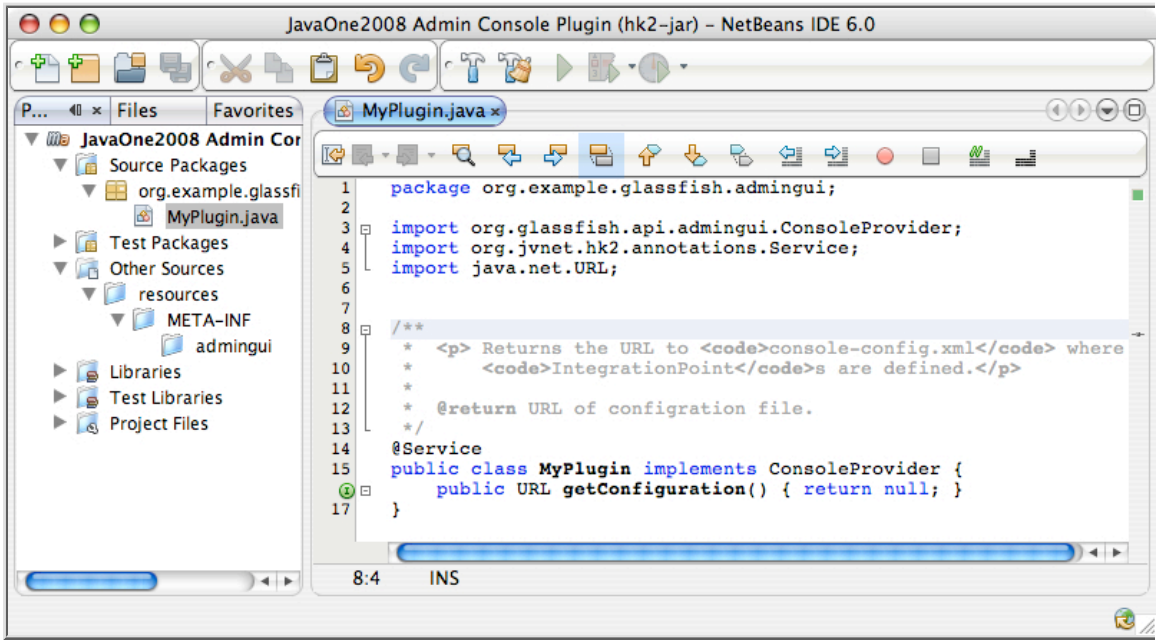


Figure 1-11: Edit MyPlugin.java

3. **Compiling your plugin**

   In this step we will compile the MyPlugin.java you just created. To do this you will need to enter "mvn install" on the command line in your plugin root directory.

   a. Go to your gfplugin/myplugin directory:

```
cd gfplugin/myplugin
```

   b. Compile your code.

```
mvn install
```

   The output will look similar to:

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building JavaOne2008 Admin Console Plugin
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] snapshot org.glassfish.common:glassfish-api:10.0-SNAPSHOT: checking for updates from gfv3

...

[INFO] [hk2:hk2-compile]
[INFO] Compiling 1 source file to .../gfplugin/myplugin/target/classes
[INFO] Picked up annotation processor com.sun.istack.internal.ws.AnnotationProcessorFactoryImpl
[INFO] Picked up annotation processor com.sun.jsftemplating.annotation.FormatDefinitionAPFactory
[INFO] Picked up annotation processor com.sun.jsftemplating.annotation.UIComponentFactoryAPFactory
[INFO] Picked up annotation processor com.sun.jsftemplating.annotation.HandlerAPFactory
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [surefire:test]
[INFO] No tests to run.
[INFO] [hk2:package]
[INFO] Building jar: .../gfplugin/myplugin/target/console-myplugin-1.0.jar
```

```
[INFO] [install:install]
[INFO] Installing .../gfplugin/myplugin/target/console-myplugin-1.0.jar to ...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 55 seconds
[INFO] Finished at: Tue Mar 18 02:45:32 GMT 2008
[INFO] Final Memory: 12M/23M
[INFO] ------------------------------------------------------------------------
```

c. Fix any errors in your code that may be reported.

Congratulations! You've completed your `ConsoleProvider` class!

Back to step overview

B. Provide a console-config.xml file.

Next, you will create the `console-config.xml` file which specifies `IntegrationPoints` for your module. The default location of this file is `META-INF/admingui/console-config.xml` inside your plugin jar file. To place this file in this location, you should create the file under the following directory in your workspace: **`myplugin/src/main/resources/META-INF/admingui`**. In your `console-config.xml` file, you will add the following:

```
<?xml version="1.0" encoding="UTF-8"?>

<console-config id="javaone">
    <integration-point
        id="JavaOneNode"
        type="tree"
        priority="210"
        parentId="tree"
        content="treenode-ex1.jsf"
    />
</console-config>
```

Figure 1-12: console-config.xml

If you are not using NetBeans, simply create the `console-config.xml` file in the location described above. Look at Figure 1-12 above for an example. Proceed to step C to continue.

1. Right click on "**admingui**" under "**Other Sources --> resources --> META-INF --> admingui**".
2. Select "**New**" and "**Other...**".
3. Select "**Other**" in the "**Categories**" list on the left, and "**Empty File**" in "**File Types**" list on the right. Select `Next`.

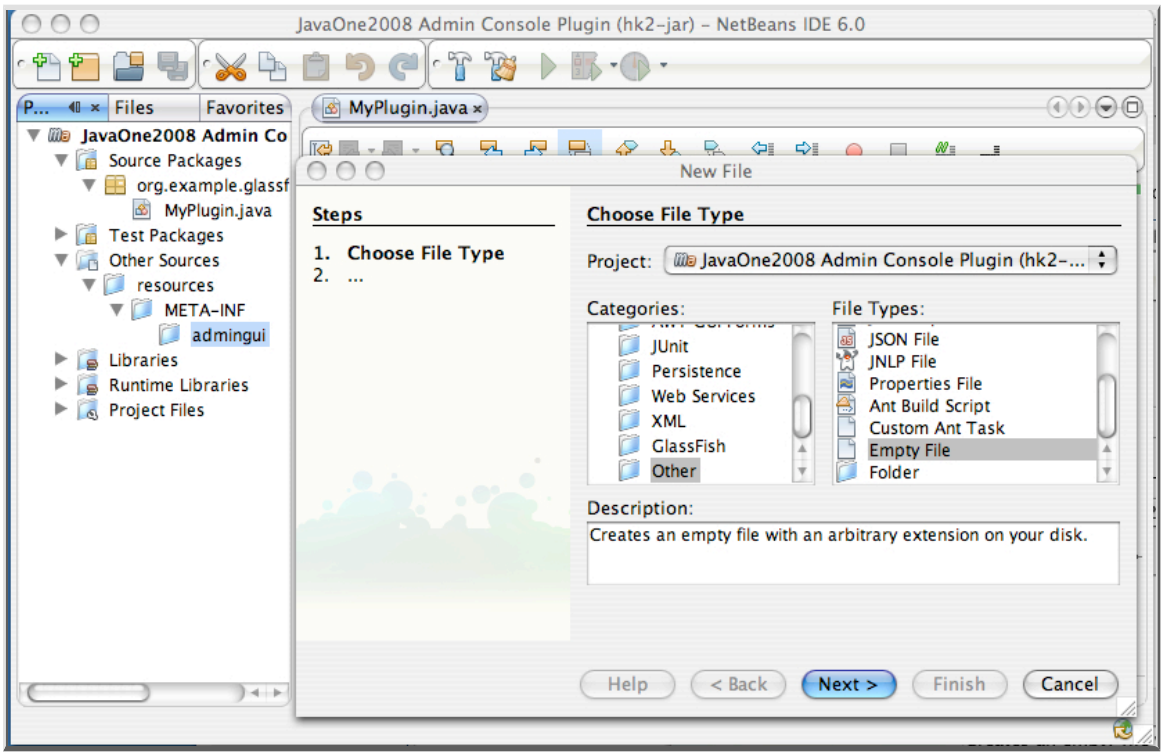


Figure 1-13: Creating console-config.xml

4. Enter "`console-config.xml`" as the file name. Select **Finish**.

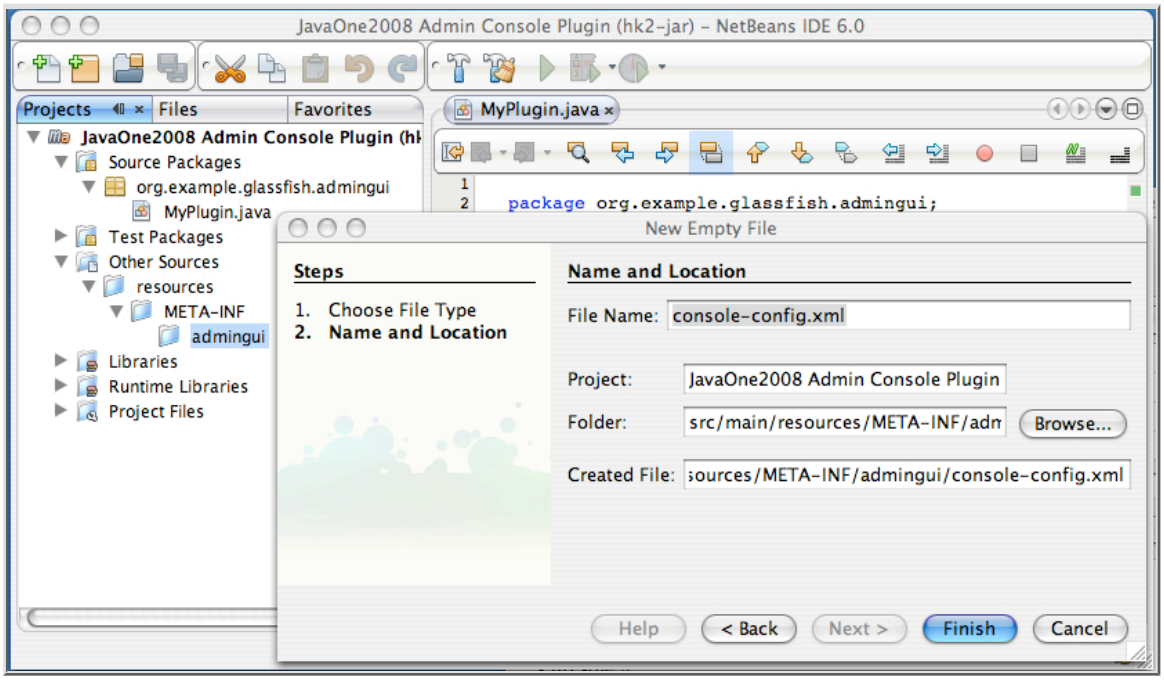


Figure 1-14: Creating console-config.xml (step 2)

5. Edit the "`console-config.xml`" file to specify the `IntegrationPoint` (note: the path to this file is: `myplugin/src/main/resources/META-INF/admingui/console-config.xml`).

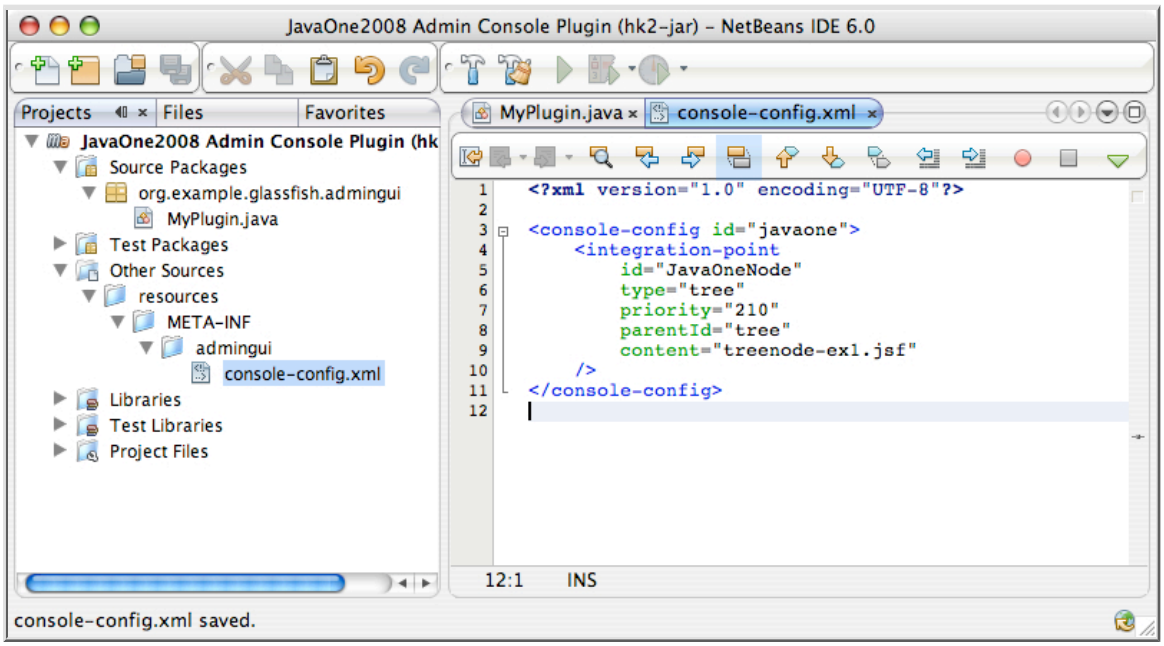


Figure 1-15: Edit console-config.xml

Back to step overview

C. Create a couple JavaServer Faces pages

In this step you will create 2 JSF pages, 1 to display your tree nodes, and 1 to provide content when your tree node is clicked. In the first JSF file you will specify 2 tree nodes: 1 tree node to contain other tree nodes, 1 tree node to link to the content for Exercise 1. The page content will be using the Facelets syntax in JSFTemplating to create a simple "hello world" page that is displayed when it is clicked. To learn more about JSFTemplating or Facelets you can visit the JSFTemplating web site: https://jsftemplating.dev.java.net; or the Facelets web site: https://facelets.dev.java.net.

If you are not using NetBeans, then you will need to create 2 files in your `myplugin/src/main/resources/` directory. The first file should be called **`treenode-ex1.jsf`** and should contain the code in Figure 1-16. The second file should be called **`hello.jsf`** and should contain the code in Figure 1-17. After you have completed these files you may proceed to step D.

```
<sun:treeNode
    id="j1root"
    text="J1 TreeNode"
    imageURL="theme/com/sun/webui/jsf/suntheme/images/tree/tree_folder.gif"
    target="main">

    <sun:treeNode
        id="first"
        text="First Node"
        url="javaone/hello.jsf"
        imageURL="resource/images/instance.gif"
        target="main"
    />
</sun:treeNode>
```

Figure 1-16: treenode-ex1.jsf

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

    <head>
        <title> Hello World </title>
    </head>
```

```
    <body>
        <h:outputText value="Your plugin is working, Congratulations!!" />
    </body>
</html>
```

Figure 1-17: hello.jsf

1. **Create "treenode-ex1.jsf"**
   a. Right click on "**Resources**". Select "**New**" and "**Other...**".
   b. Select "**Other**" in the "**Categories**" column on the right and "**Empty File**" in **File Types** column on the right. Select **Next >**.
   c. Fill in "**treenode-ex1.jsf**" as the "**File Name**". (Note: This is referenced from the "content" specified in the integration-point in the console-config.xml file.)

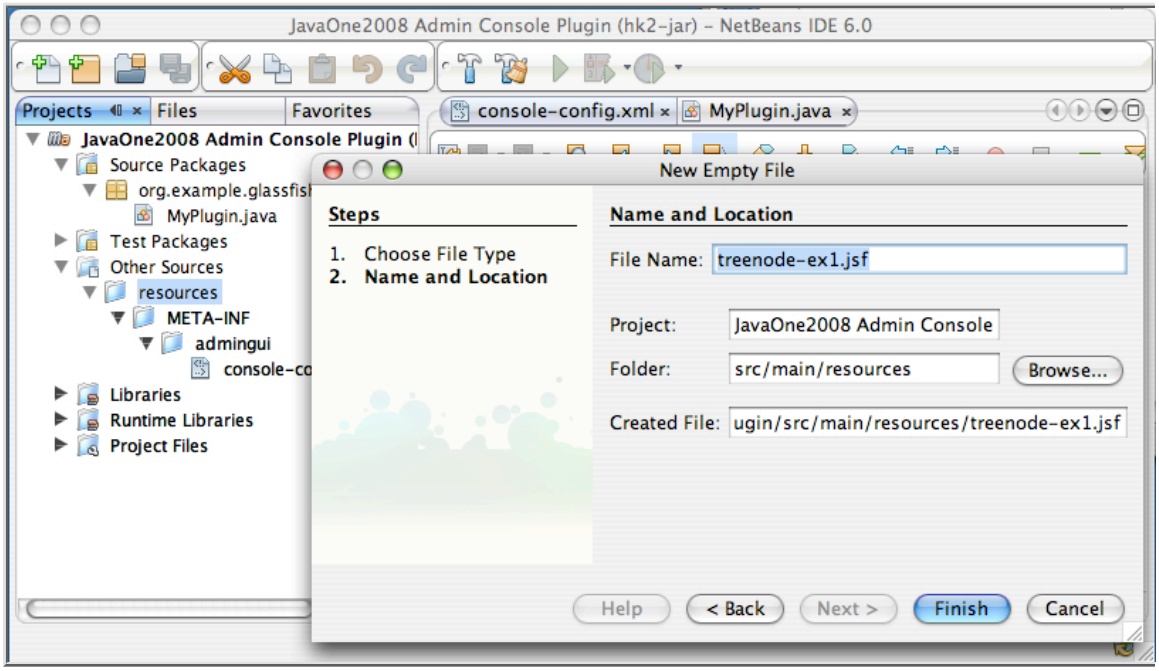


Figure 1-18: Create treenode-ex1.jsf

   d. Add content to the file to specify the new tree nodes (note: the full path of this file is `myplugin/src/main/resources/treenode-ex1.jsf`).

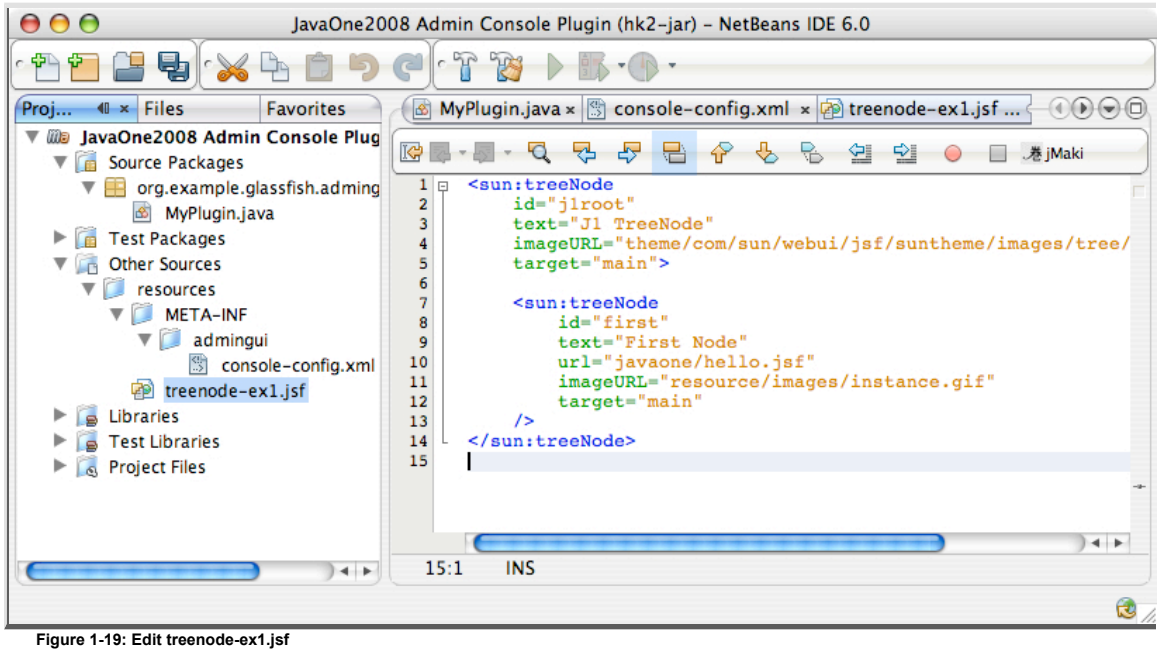


Figure 1-19: Edit treenode-ex1.jsf

2. **Create "hello.jsf"**
   a. Next you will create the "hello.jsf" page which will be displayed on the right frame of the Administration Console when the "Exercise #1" tree node is clicked. The file will be created in the same directory where you created "treenode-1.jsf". The full path to this file is "`myplugin/src/main/resources/hello.jsf`".

      Right click on "**Resources**". Select "**New**" and "**Other...**"

   b. Select "**Other**" in the "**Categories**" column on the right and "**Empty File**" in **File Types** column on the right. Select **Next >**.
   c. Enter "**hello.jsf**" as the file name. This corresponds to the "content" specified in the "treenode-ex1.jsf" file you created.
   d. Open the file and add the following code. (Note: the full path to this file is `myplugin/src/main/resources/hello.jsf`.)

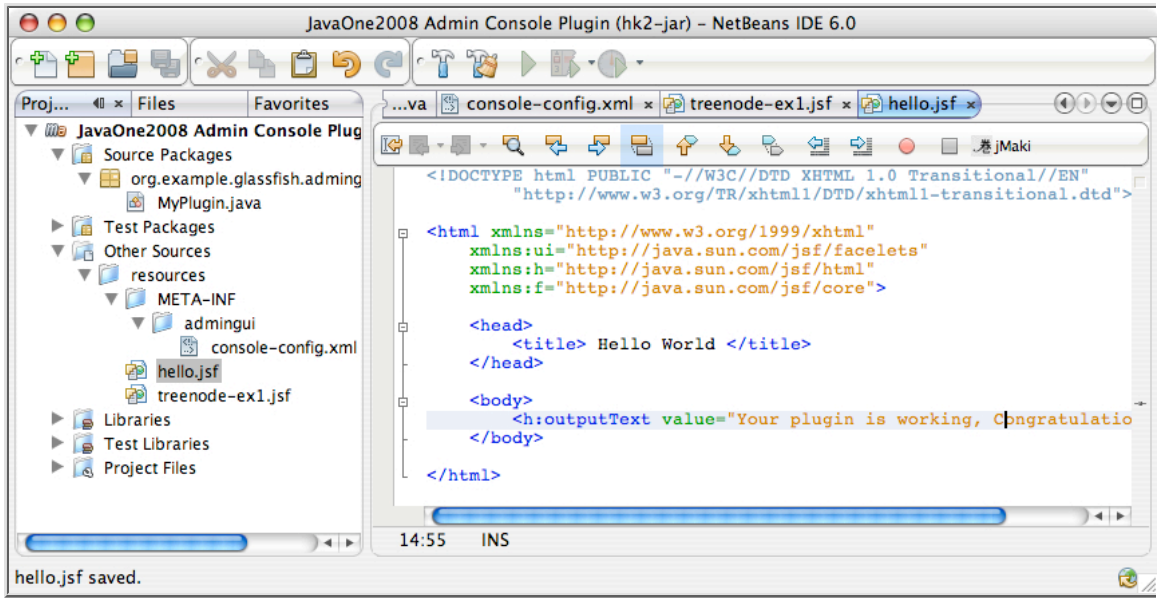


Figure 1-20: Edit hello.jsf

You have completed step C! In this step, you created 2 JSFTemplating files which will be contained in your `plugin.jar` file. One of these files (`treenode-ex1.jsf`) is referenced from your `IntegrationPoint`, which in turn is defined in your `console-config.xml` file. When treenode-ex1.jsf is displayed in the browser as part of the tree page, it links to your `hello.jsf` page.

In the next step, we will see how to package these files into a jar file, along with the files created in step A & B. You can then install the jar file as a GlassFish plugin module and see it work!

Back to step overview

D. Build and Install your first GlassFish v3 plugin

The last step of this exercise is to build the project. You will need to open a terminal window and perform the following steps using the command line.

1. **View the GlassFish Administration Console before your plugin is installed.**

   For your convenience, GlassFish v3 is installed in the `gfplugin/glassfish` directory.

   a. Start the server:

   ```
   glassfish/bin/asadmin start-domain
   ```

   b. Deploy the Administration Console.

   ```
   glassfish/bin/asadmin deploy glassfish/admingui
   ```

   c. In the browser enter the following URL to bring up the Administration Console:

   ```
   http://localhost:8080/admingui/index.jsf
   ```

   Note: The navigation tree on the left does not have the "J1 TreeNode" yet as you have not yet integrated your plugin.

Figure 1-21: Before Integration

2. **Build your plugin.**
   a. Go to your `gfplugin/myplugin` directory:

```
cd gfplugin/myplugin
```

   b. Compile your plugin.

```
mvn install
```

3. **Install your plugin.**

   You now have a jar file named `gfplugin/myplugin/target/console-myplugin-1.0.jar`. This jar file is ready to plugin into the GlassFish v3 Application Server. The following steps show you how to integrate your plugin module.

a. Go to your `gfplugin` directory:

```
cd gfplugin
```

b. Stop the server:

```
glassfish/bin/asadmin stop-domain
```

c. Integrate the jar to GlassFish by copying it to the GlassFish modules directory.

```
cp myplugin/target/console-myplugin-1.0.jar glassfish/modules/console-myplugin-1.0.jar
```

4. **Test your plugin**
   a. Start the server again:

```
glassfish/bin/asadmin start-domain
```

b. In the browser enter the following URL to bring up the Administration Console:

```
http://localhost:8080/admingui/index.jsf
```

i. Note the "**J1 TreeNode**" tree node is now displayed in the navigation tree.

ii. Note that clicking on the "**First Node**" tree node displays your "**hello.jsf**" page.

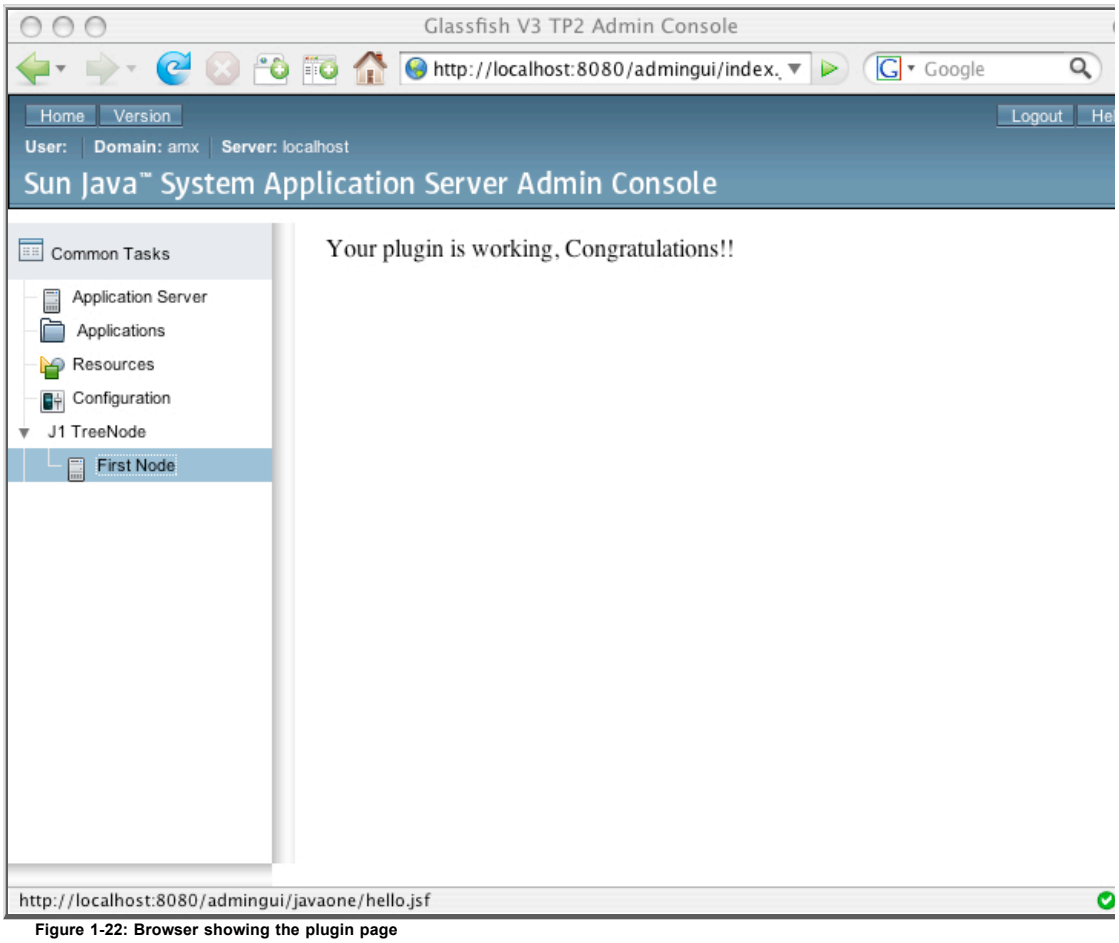


Figure 1-22: Browser showing the plugin page

You have just customized the Administration Console! You plugged in your own tree node, and added a new page to the application. Congratulations!

Summary:

In this exercise, you have learned about GlassFish plugin module and developed a plugin module to the Administration Console where you added a tree node to the navigation tree. Clicking this tree node shows you the plugin page.

Go to solution

Go on to next exercise

Exercise 2: Charting with jMaki

Approximate duration: 25 minutes

Introduction:

The goal of this exercise is to create a JSF page which utilizes the jMaki Charting component and link the page to the GlassFish Administration Console by adding another tree node in the GlassFish Administration Console. When you have completed this exercise, your plugin will resemble the following image:

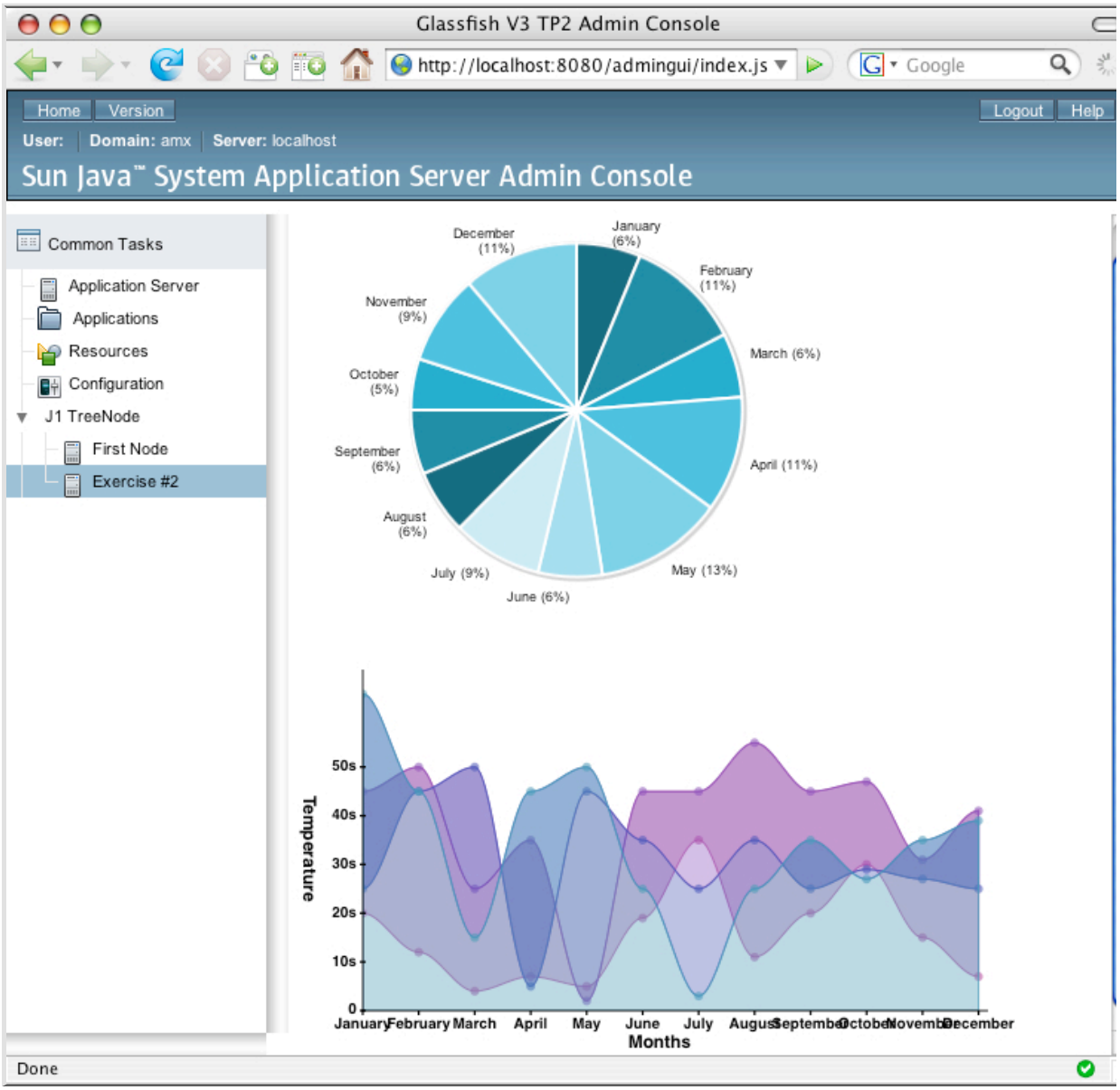


Figure 2-1: Browser showing the plugin page with charts.

In this exercise you will learn the following concepts:

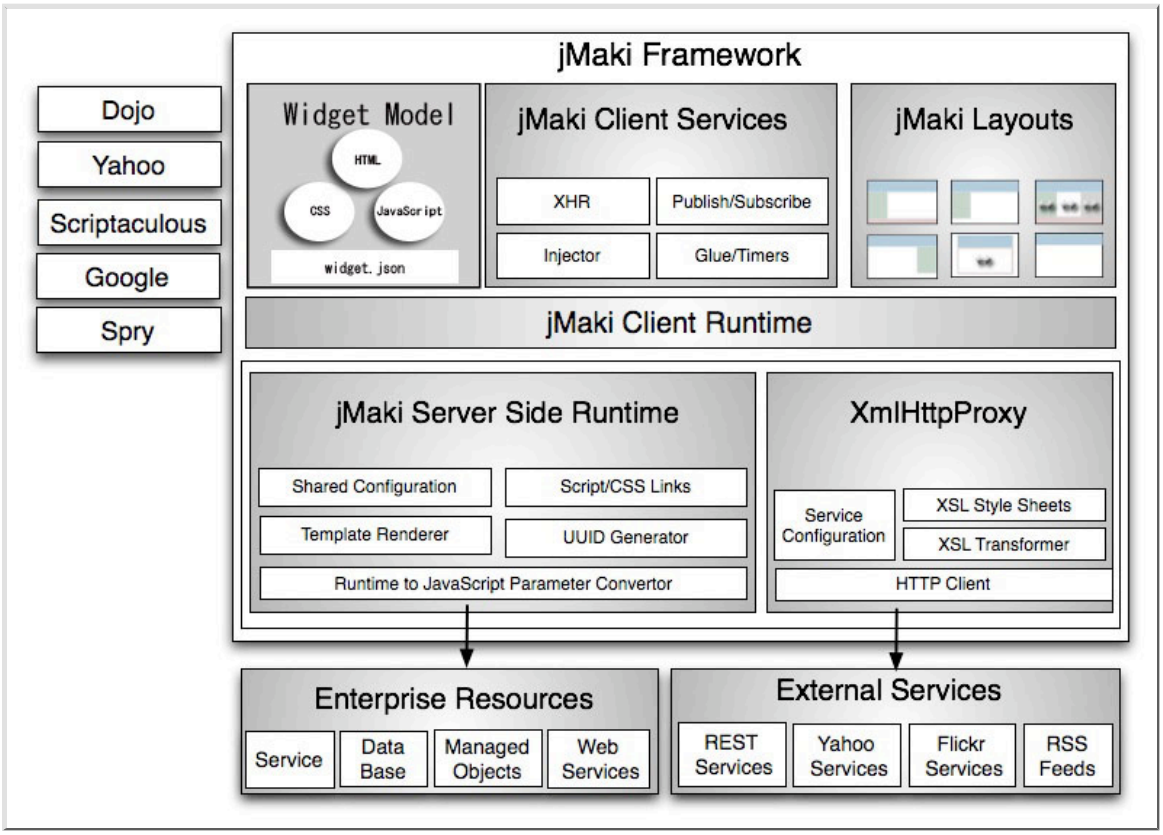
- How to add jMaki widget to a JSF page.
- How to add a plugin to the Administration Console that links to another plugin `IntegrationPoint`.

To learn background information about jMaki, proceed to the next section. To begin creating the jMaki Charting JavaServer Faces page immediately, proceed to the Step-by-Step Instructions.

Background Information:

jMaki

jMaki is a lightweight client/server framework for creating JavaScript centric Web 2.0 applications using CSS layouts, widgets widget model, client services such as publish / subscribe events to tie widgets together, JavaScript action handlers, and a generic proxy to interact with external RESTful web services. While jMaki abstracts much of the JavaScript and CSS by providing defaults for widgets, the JavaScript widgets and CSS are made easily accessible so they may be customized by a designer or page developer. jMaki focuses on the aspects of delivering JavaScript to the client allowing the JavaScript to communicate to various server-technologies including PHP, JavaServer Pages, JavaServer Faces, and Phobos in a server-technology neutral way.



Following are the client and the server modules that make up jMaki.

Client-side Components

- **jMaki Layouts:** - jMaki Layouts provide a standards based starting point for creating your web applications using HTML and CSS. The CSS is in plain view so it can be easily customized for your needs.
- **jMaki Client Runtime:** - The jMaki Client Runtime is the JavaScript responsible for bootstrapping all widgets and passing parameters provided by a server-side runtime to the widgets. Parameters are unique to each widget, and the JavaScript runtime makes sure that each widget instance gets the correct parameters that were passed from the server-side runtime. In cases where parameters are not provided the runtime uses default parameters that may then be customized for each widget.
- **jMaki Client Services:**Services such convenient APIs for performing XMLHttpRequest and publish subscribe on the client are provided to all widgets as a means of communication. jMaki glue is built on top of publish/subscribe events. jMaki Glue is built on top of the publish/subscribe mechanism. It allows you to define application behavior and time widgets together widgets using JavaScript actions when a specific event is published to a topic. jMaki Timers allow JavaScript action handlers to be called or events to be published at a set interval. The jMaki Injector is a client service that allows you to bring in an external page into any given div element. The Injector transfers scripts and CSS to the global page content and allows widgets to be loaded much like an iframe without the usability issues.
- **jMaki Widget Model** - The jMaki widget model provides a component model for reusable JavaScript components. The structure is based on a HTML, JavaScript and CSS. You can use this structure to create your own widgets or wrap widgets from any given toolkit. jMaki provides default wrappers and a server tie-in for many commonly-used Dojo, Yahoo UI, Prototype, and some native widgets. jMaki also defines a widget.json format which is a common way of describing which so that they are accessible by tools.

Server Components

- **jMaki Server Runtime** - The jMaki Server Runtime is responsible for tying the jMaki JavaScript Client Runtime to a server-side runtime such as Java, PHP, or the JavaScript-based Phobos runtime. The server runtime tracks and renders all script and CSS references based on library type being used making sure that the duplicate script and CSS links are not duplicated. The server-runtime also makes sure API keys (such as Google and Yahoo Map keys) are applied when necessary based on a configurable set of keys. The server runtime renders the HTML templates making and serializes data in JavaScript such that each widget instance is provided the proper data.

- **XmlHttpProxy** - The XmlHttpProxy module provides a means for widgets to access JSON or XML access RESTful XML-based services outside of the web application domain. Such services include RSS feeds, Yahoo services such as geocoding, Flickr image searches, and many more to come. The XmlHttpProxy allows widgets to access services in a uniform way by providing XSL-to-JSON transformations that can be easily customized.

For more developer information on jMaki, visit: https://ajax.dev.java.net/developer.html.

jMaki Charting

jMaki charting is an extension of the jMaki project which provides dynamic charting capabilities where charts can be manipulated on the client without round trips to the server. Chart manipulation is handled through the jMaki publish and subscribe mechanism. jMaki provides a common API to the following JavaScript Charting packages, Plotkit, Dojo, Yahoo and Google. Developers can quickly represent their data using bar, line, or area charts.

You will get to experiment with jMaki Charting in this lab. If you would like more details on jMaki Charting, visit: https://jmaki-charting.dev.java.net.

Step-by-Step Instructions:

You will need Exercise #1 completed in order to continue with Exercise #2.

1. To use the solution application instead of your own application (which you should have completed in Exercise #1), execute the following commands from a terminal window:

   Warning! By copying the solution you will no longer see any work you completed in exercise #1. We recommend you complete exercise #1 instead.

```
mv gfplugin/myplugin gfplugin/myplugin.save
cp -r gfplugin/solution/myplugin/ex1 gfplugin/myplugin
```

Overview:

In this exercise, you will complete the following:

A. **Create a jMaki Charting page**

   You will create a JSFTemplating JavaServer Faces pages that contains a jMaki Charting component

B. **Modify your console-config.xml**

   You will modify your configuration file to add a new `IntegrationPoint`

C. **Specify your Tree Node layout**

   In this step you will add the file "treenode-ex2.jsf" for specifying the new tree node that points to the jMaki Charting page

D. **Build and Install your GlassFish v3 plugin**

   In this step you will install and test your plugin.

Steps:

A. Create a jMaki Charting page

```
<sun:page>
<sun:html>
    <sun:head title="chart" />
    <sun:body>
        <sun:form id="form">

<!-- Pie Chart -->
<sun:markup tag="div" style="width:380px; height: 310px;">
<jmaki:widget name="jmaki.charting.plotkit.pie"
    args="{colorScheme:4}"
    value="{
    xAxis : {
        title : 'Months',
        labels : [
        {label : 'January'},
        {label : 'February'},
        {label : 'March'},
        {label : 'April'},
        {label : 'May'},
        {label : 'June'},
        {label : 'July'},
        {label : 'August'},
        {label : 'September'},
        {label : 'October'},
        {label : 'November'},
        {label : 'December'} ] },
    data : [ {
        label : 'Set 1',
```

```
        values : [25, 45, 25, 45, 50, 25, 35, 25, 25, 20, 35, 45]
        } ] }" />
</sun:markup>

<!-- Area Chart -->
<sun:markup tag="div" style="width: 500px; height: 280px;">
<jmaki:widget name="jmaki.charting.dojo.area"
    value="{
        xAxis : {
            title : 'Months',
            labels : [{ label : 'January'},
                { label : 'February'},
                { label : 'March'},
                { label : 'April'},
                { label : 'May'},
                { label : 'June'},
                { label : 'July'},
                { label : 'August'},
                { label : 'September'},
                { label : 'October'},
                { label : 'November'},
                { label : 'December'} ] },
        yAxis : {
            title : 'Temperature',
            labels : [{ label : '0', value : 0},
                { label : '10s', value : 10},
                { label : '20s', value : 20},
                { label : '30s', value : 30},
                { label : '40s', value : 40},
                { label : '50s', value : 50} ] },
        data : [{label : 'Gray Series', values : [65, 45, 15, 45, 50, 25, 3, 25, 35, 27, 35, 39] },
                {label : 'Pink Series', values : [25, 45, 50, 5, 45, 35, 25, 35, 25, 29, 27, 25] },
                {label : 'Blue Series', values : [45, 50, 25, 35, 2, 45, 45, 55, 45, 47, 31, 41] },
                {label : 'Red Series',  values : [20, 12, 4, 7, 5, 19, 35, 11, 20, 30, 15, 7] } ] }"/>
</sun:markup>

        </sun:form>
    </sun:body>
</sun:html>
</sun:page>
```

Figure 2-2: chart.jsf

1. **Open your project in NetBeans**

   You should have `myplugin` project still open in NetBeans. If you have closed the project, follow these steps to open it again.

   a. Launch NetBeans IDE 6.0.

   b. Open `myplugin` module:
      i. Choose **Open Project** from the File menu.
      ii. Browse to **gfplugin** and select **myplugin** module.
      iii. Press the **Open Project button**. Your NetBeans **Projects** tab should show:

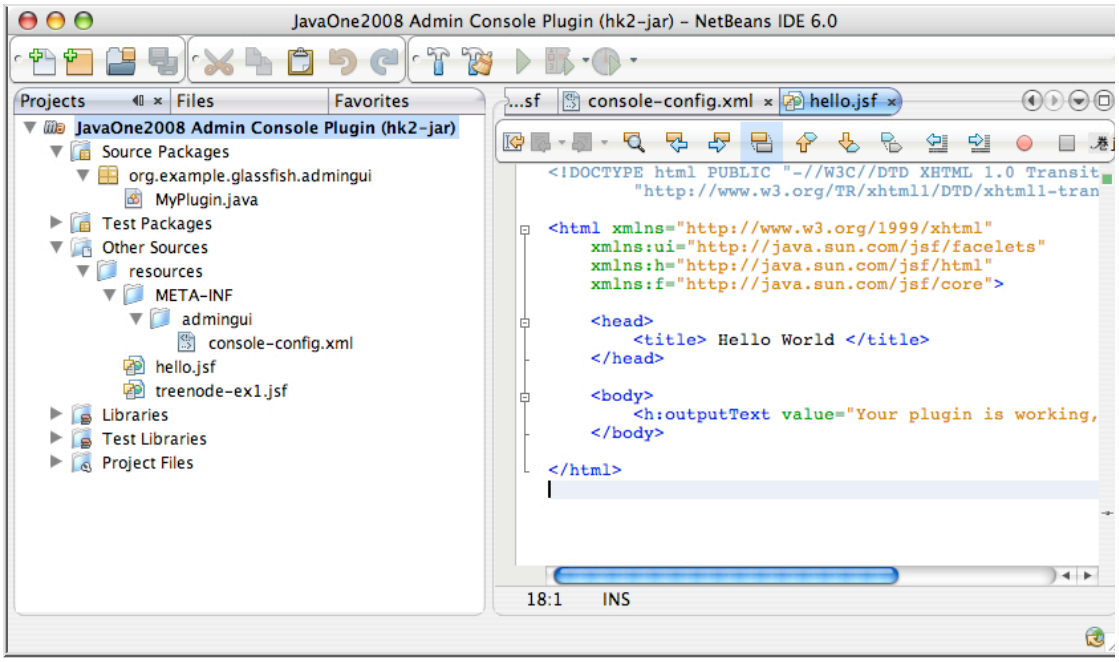


Figure 2-3: Open myplugin module.

2. **Create a jMaki Charting page**
   a. Select "**Other Sources**" and then "**Resources**".
   b. Create an empty file named "**chart.jsf**".
   c. Use Figure 2.2 for the content of chart.jsf.

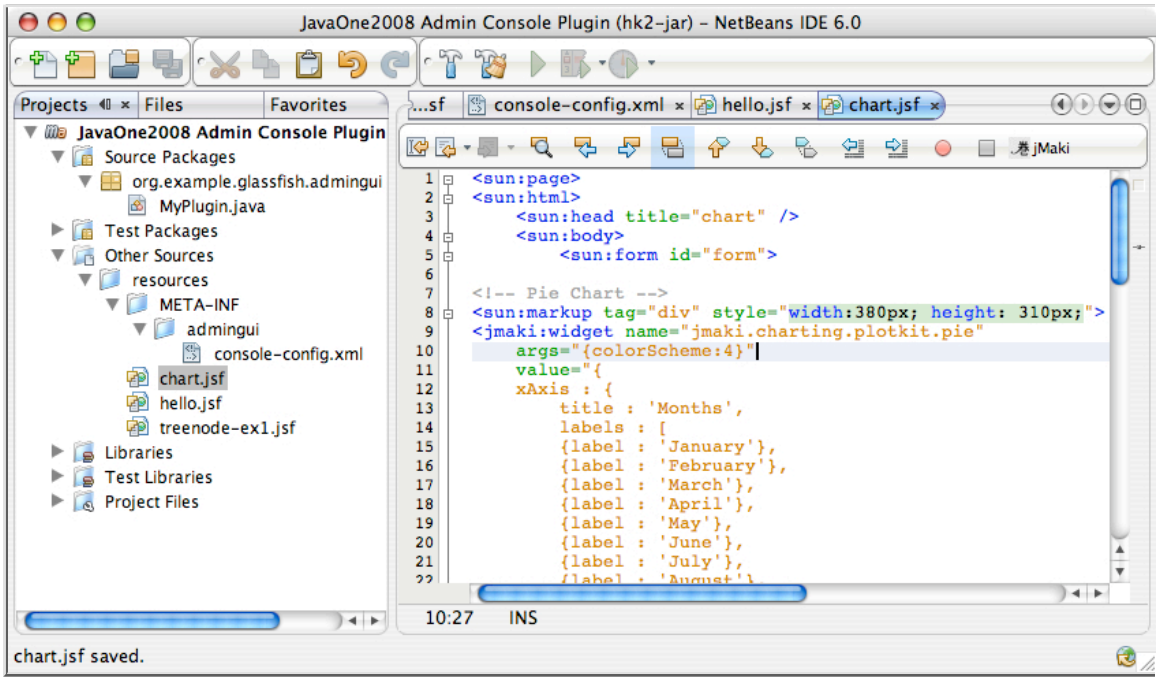


Figure 2-4: Edit chart.jsf.

B. Modify your console-config.xml

You now need to add an `IntegrationPoint` for plugging the Exercise 2 tree node into the Administration Console. Each plugin module should have 1 `console-config.xml`, and you can specify as many `IntegrationPoints` as needed in this configuration file. Since you already created this file in Exercise #1, all you need to do is edit it and add the second `IntegrationPoint`. The Exercise 2 tree node should be under the J1 TreeNode, so you must specify the `parentId` to be "`j1root`". You can refer to Figure 2-5 to see this.

```
<integration-point
    id="Another"
    type="tree"
    priority="230"
    parentId="j1root"
    content="treenode-ex2.jsf"
/>
```

Figure 2-5: New integration-point for console-config.xml

1. Double click on **`console-config.xml`** under **`Other Sources --> resources --> META-INF --> admingui`**
2. Add the `IntegrationPoint` declaration as shown in Figure 2-5 to your `console-config.xml` file.

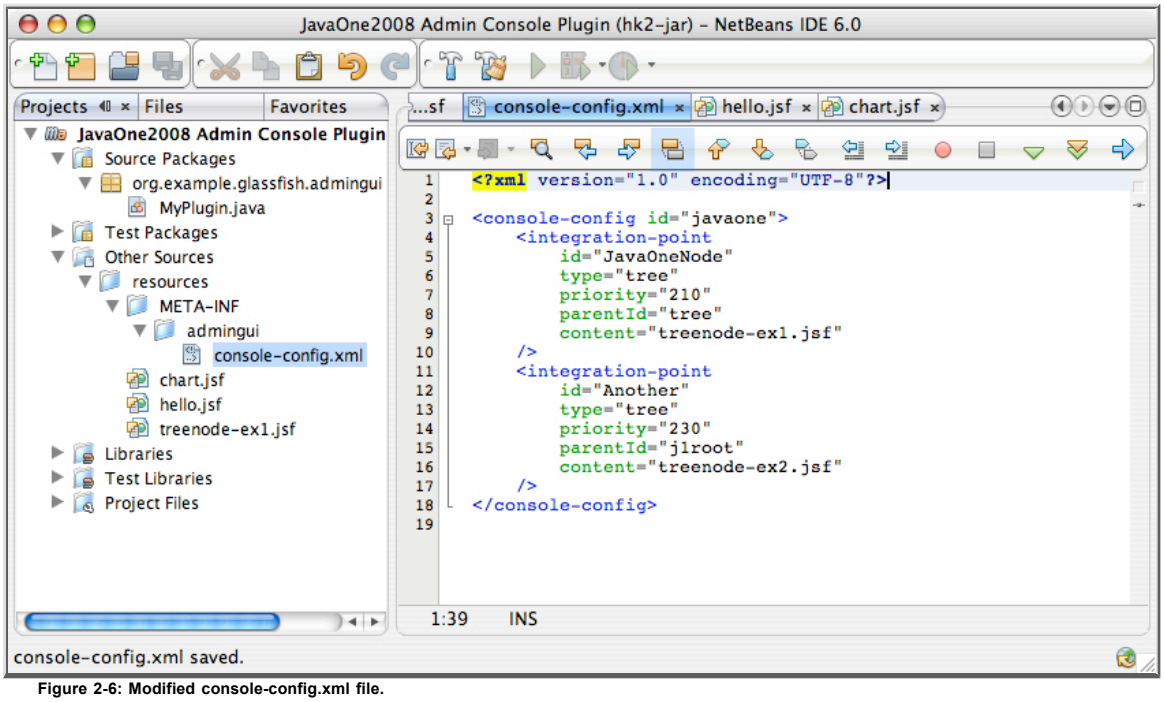


Figure 2-6: Modified console-config.xml file.

C. Specify your Tree Node layout

You need to create `treenode-ex2.jsf`, which is specified as the "`content`" of your new `IntegrationPoint` in your `console-config.xml` file. The `treenode-ex2.jsf` file will provide access to your new JSF jMaki Charting page.

```
<sun:treeNode
    id="second"
    text="Exercise #2"
    url="javaone/chart.jsf"
    imageURL="resource/images/instance.gif"
    target="main"
/>
```

Figure 2-7: Modify treenode-ex2.jsf

1. Select "**Other Sources**" and then "**Resources**".
2. Create an empty file named "**treenode-ex2.jsf**".

3. Edit **`treenode-ex2.jsf`** to specify the information regarding this second tree node.

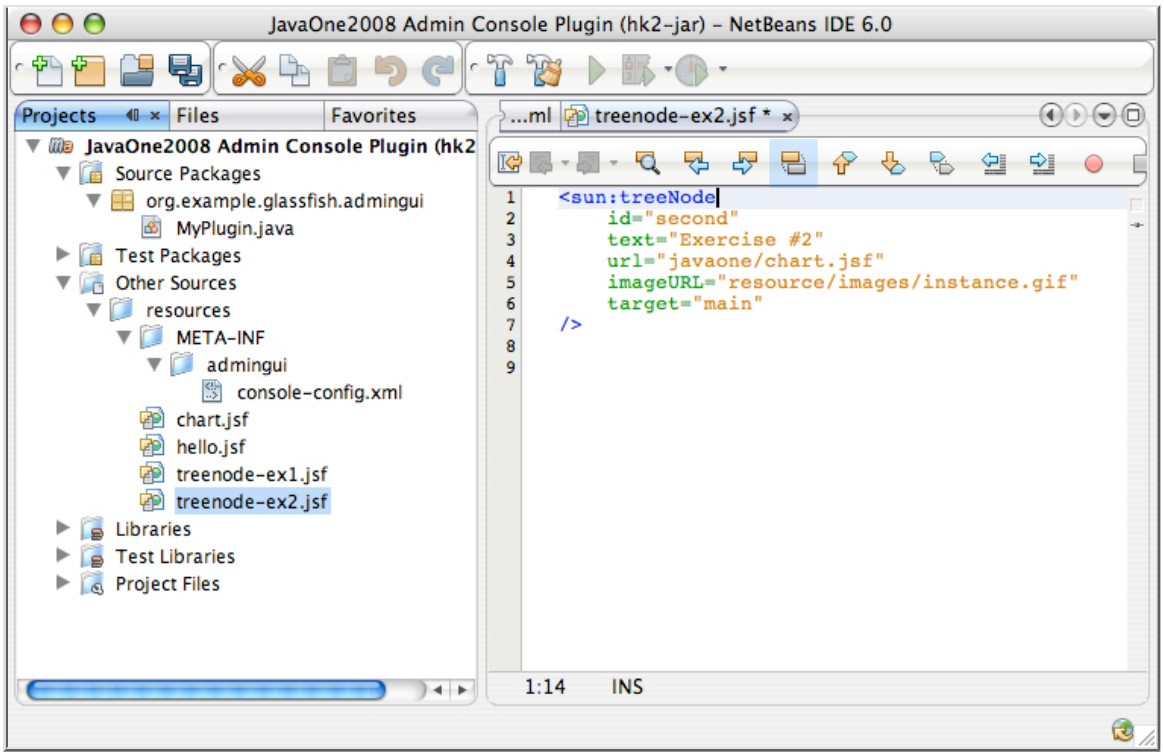


Figure 2-8: treenode-ex2.jsf

D. Build and Install your GlassFish v3 plugin

1. **Build your plugin**

```
cd gfplugin/myplugin
mvn install
```

After the above steps are finished, you will have the jar file named `gfplugin/myplugin/target/console-myplugin-1.0.jar`. This jar file is ready to be integrated into GlassFish.

2. **Install your plugin**

You now need a Terminal to enter some commands. Please bring up a terminal window.

The GlassFish v3 server is installed in the `gfplugin/glassfish` directory. The following steps show you how to integrate and test your plugin module.

a. Go to your `gfplugin` directory.

```
cd <path-to-gfplugin>
```

b. Shut down the server which may have been started in Exercise 1.

```
glassfish/bin/asadmin stop-domain
```

c. Integrate your plugin into GlassFish by copying it to the GlassFish modules directory:

```
cp myplugin/target/console-myplugin-1.0.jar glassfish/modules/console-myplugin-1.0.jar
```

d. Start GlassFish:

```
glassfish/bin/asadmin start-domain
```

e. In the browser enter the following URL to bring up the Administration Console:

```
http://localhost:8080/admingui/index.jsf
```

f. Expand the JavaOne tree node in the navigation tree, note your new tree node "`Exercise #2`" has been added. Click the "`Exercise #2`" tree node to display your chart page.

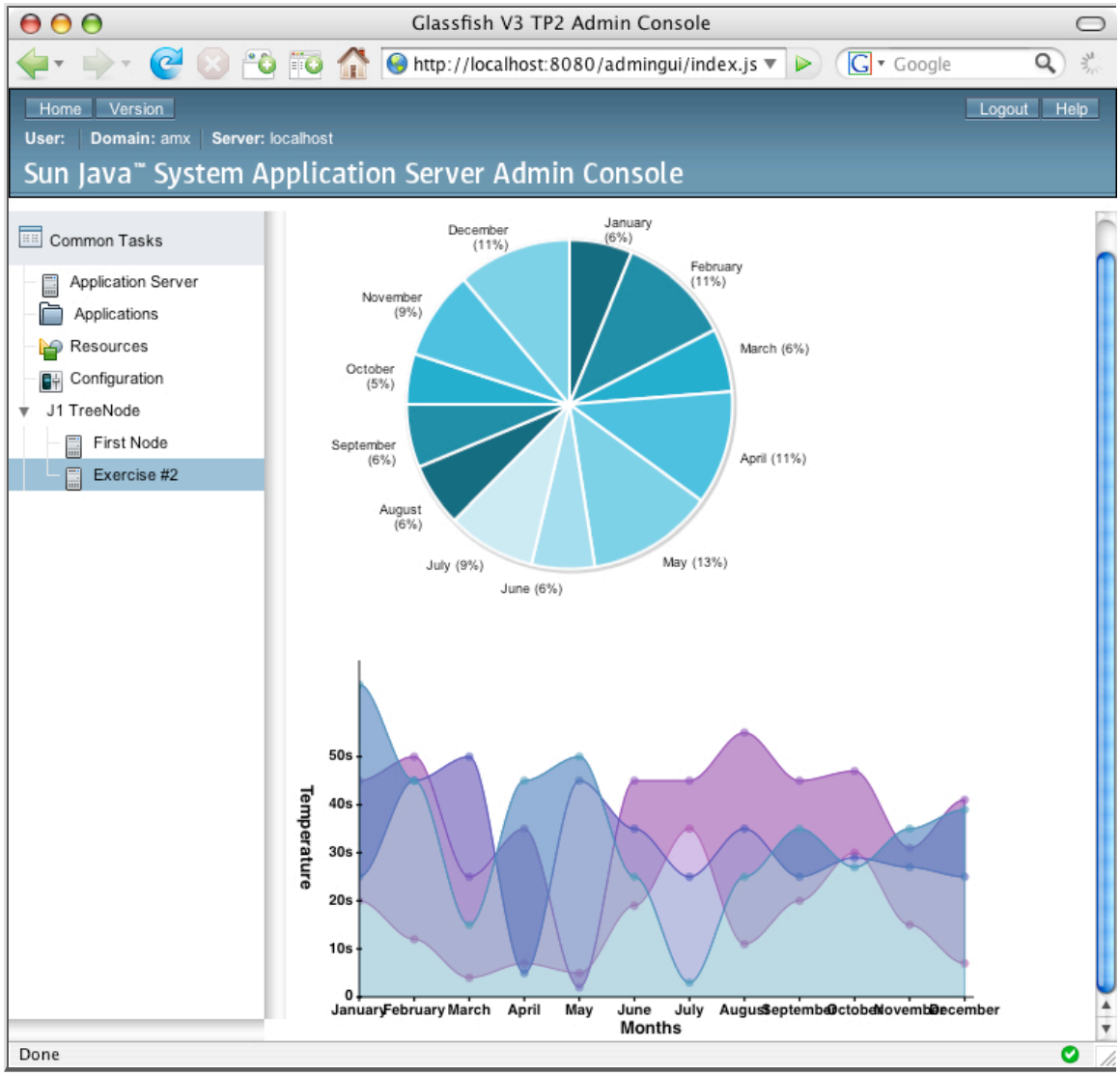


Figure 2-9: Browser showing the plugin page with charts.

Summary:

In this exercise, you created a JSFTemplating page with jMaki Chart. You added a new `IntegrationPoint` to your GlassFish plugin module which added an additional tree node. Clicking on this tree node navigated to your newly added chart page.

Go to solution

Go back to previous exercise

Go on to next exercise

Exercise 3: Templating your Plugin

Approximate duration: 25 minutes

Introduction:

In this exercise you will incorporate the previous exercises. You will also use a template to help you create a new JavaServer Faces page. The template helps you to layout your page and allows you to focus on providing the content. At the completion of this lab your GlassFish v3 Administration Console plugin will look like this:

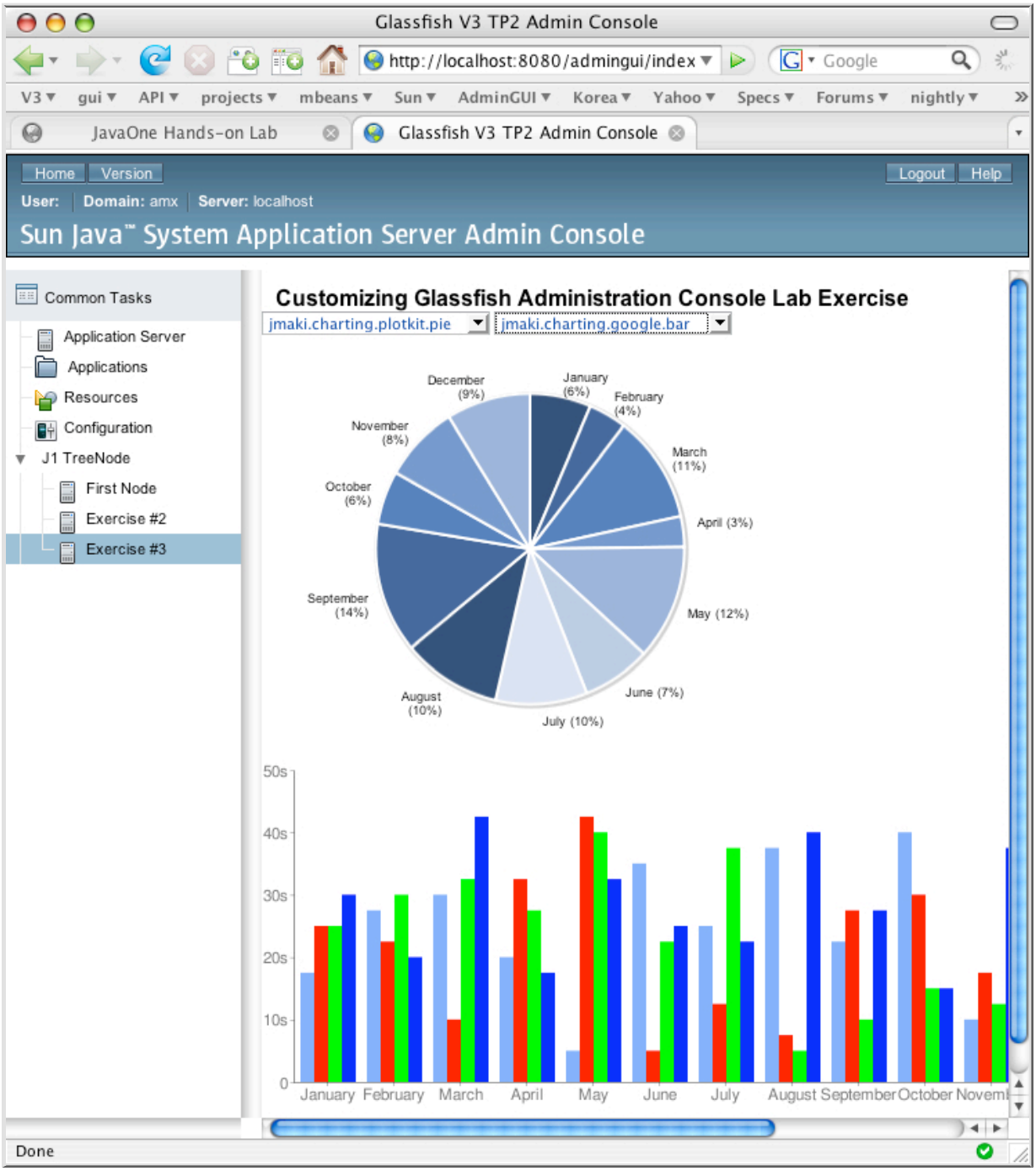


Figure 3-1: Exercise 3 Completed.

In this exercise you will:

- Create a JSF page which utilizes a template and shows a jMaki chart.
- Add another tree node to the Administration Console via an IntegrationPoint.
- Access the GlassFish v3 Administration Console and see the jMaki Charts page with customized header.

To learn background information about using template in JavaServer Faces, proceed to the next section. To begin using a template to create a JavaServer Faces page immediately, proceed to the Step-by-Step Instructions.

Background Information:

JSFTemplating is an open source project hosted on java.net (https://jsftemplating.dev.java.net). This project provides an alternative to using JavaServer Pages for the JSF view definition. JSFTemplating allows for pluggable syntaxes to be used to define JSF pages. One of those syntaxes is the Facelets syntax (https://facelets.dev.java.net), it also includes two other formats, and others can be added. JSFTemplating also provides many other useful features, many of which will appear in JSF 2.0 when JSR 314 is final. Here are some of the useful features in JSFTemplating:

- Facelets-style Templating
- Page Scope
- Events
- Handlers
- Component Factories
- Better Performance than JSP
- Ajax Support

In this lab you will be using the Facelets-style of templating pages. This allows you to define a template which specifies the entire layout of the page. You then build pages which refer to the template and "pass content" into it. The template uses the content which you pass (via ui:define or !define tags) to layout the structure of your page. This means your individual pages do not need to be aware of the general layout of your page, nor do they need to know the ordering in which content appears. In this example, our template is very simple, but more complex examples achieve even greater benefit from this type of templating.

To get involved, or to learn more about the templating features of JSFTemplating, or any of the other features, visit: https://jsftemplating.dev.java.net, or the #jsftemplating IRC channel on irc.freenode.net. Instructions for using IRC can be found at: https://jsftemplating.dev.java.net/irc.html.

Step-by-Step Instructions:

You will need Exercise #2 completed in order to continue with Exercise #3.

1. To use the solution application instead of your own application (which you should have completed in Exercise #2), execute the following commands from a terminal window:

   Warning! By copying the solution you will no longer see any work you completed in exercise #2. We recommend you complete exercise #2 instead.

```
mv gfplugin/myplugin gfplugin/myplugin.save2
cp -r gfplugin/solution/myplugin/ex2 gfplugin/myplugin
```

Overview:

In this excercise, you will complete the following:

A. **Create a jMaki Charting Page within a Template.**

   You will create a JSFTemplating page that contains a jMaki Charting component within a template.

B. **Modify Your console-config.xml.**

   You will modify your `console-config.xml` configuration file to add a new `IntegrationPoint`.

C. **Specify Your Tree Node layout.**

   In this step you will add the file "`treenode-ex3.jsf`" to specify a new tree node to point to the new jMaki Charting page.

D. **Build and Install Your GlassFish v3 Plugin.**

   In this step you will install and test your plugin.

Steps:

A. Create a jMaki Charting Page within a Template

You should have `myplugin` project still open in NetBeans. If you have closed the project, follow these steps to open it again.

1. **Open the plugin module in NetBeans**
   a. Launch NetBeans IDE 6.0.

b. Open plugin module.
   i. Choose **Open Project** from the **File** menu.
   ii. Browse to **gfplugin** and select **myplugin**.
   iii. Press the **Open Project** button.

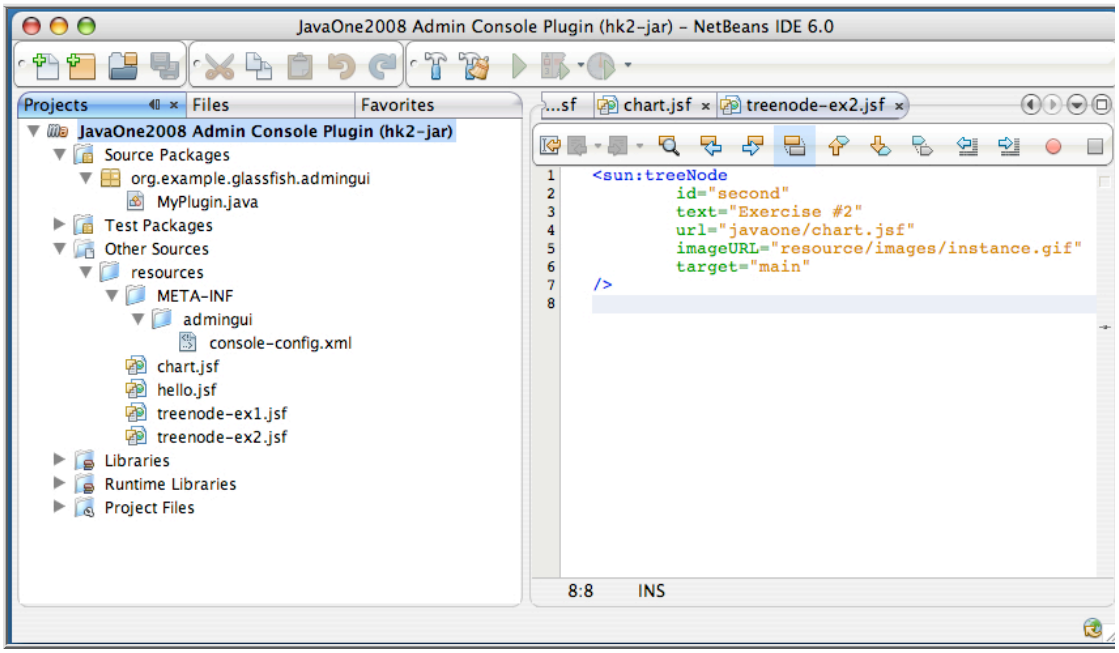


Figure 3-2: Open Project myplugin

2. **Create a page showing a jMaki chart in a template.**

In this step, you will create a JSF page based on a template, and add the jMaki Charting component to it. Notice the only change compared to "chart.jsf" in exercise #2 is the use of template `javaOneTemplate.tpl`. The template defines parts of the page that are shared across multiple pages and allows our page to contain much less redundant code.

```
<!-- Use composition to define template -->
<!composition template="/templates/javaOneTemplate.tpl">
<!define name="content">

<sun:dropDown submitForm="true"
        labels={
            "jmaki.charting.google.pie"
            "jmaki.charting.plotkit.pie"
            "jmaki.charting.yahoo.pie"}
        value="#{requestScope.chart1}">
    <!beforeCreate
        <!-- Set default pie chart, and random data -->
        setAttribute(key="chart1", value="jmaki.charting.plotkit.pie");
        getjMakiValues(values=>$attribute{pievalues});
    />
</sun:dropDown>

<sun:dropDown submitForm="true"
        labels={
            "jmaki.charting.plotkit.area"
            "jmaki.charting.plotkit.line"
            "jmaki.charting.plotkit.bar"
            "jmaki.charting.google.area"
            "jmaki.charting.google.bar"
            "jmaki.charting.google.line"
            "jmaki.charting.yahoo.line"
            "jmaki.charting.yahoo.bar"
            "jmaki.charting.dojo.bar"
            "jmaki.charting.dojo.line"
            "jmaki.charting.dojo.area"}
        value="#{requestScope.chart2}">
    <!beforeCreate
        <!-- Set default chart type and random data -->
        setAttribute(key="chart2", value="jmaki.charting.dojo.area");
        getjMakiValues(values=>$attribute{gryvalues});
        getjMakiValues(values=>$attribute{bluvalues});
        getjMakiValues(values=>$attribute{pnkvalues});
```

```
        getjMakiValues(values=>$attribute{redvalues});
    />
</sun:dropDown>

<sun:markup tag="div" style="width:380px; height: 310px;">
<jmaki:widget name="#{requestScope.chart1}"
    args="{colorScheme:0}"
    value="{
        xAxis : {
            title : 'Months',
            labels : [
                {label : 'January'},
                {label : 'February'},
                {label : 'March'},
                {label : 'April'},
                {label : 'May'},
                {label : 'June'},
                {label : 'July'},
                {label : 'August'},
                {label : 'September'},
                {label : 'October'},
                {label : 'November'},
                {label : 'December'} ] },
        data : [ {
            label : 'Set 1',
            values : $attribute{pievalues}
            } ] }"/>
</sun:markup>

<sun:markup tag="div" style="width: 500px; height: 280px;">
<jmaki:widget name="#{requestScope.chart2}"
value="{  xAxis : {
    title : 'Months',
    labels : [{ label : 'January'},
        { label : 'February'},
        { label : 'March'},
        { label : 'April'},
        { label : 'May'},
        { label : 'June'},
        { label : 'July'},
        { label : 'August'},
        { label : 'September'},
        { label : 'October'},
        { label : 'November'},
        { label : 'December'} ] },
    yAxis : {
    title : 'Temperature',
    labels : [{ label : '0', value : 0},
        { label : '10s', value : 10},
        { label : '20s', value : 20},
        { label : '30s', value : 30},
        { label : '40s', value : 40},
        { label : '50s', value : 50} ] },
    data : [
        {label : 'Gray Series', values : $attribute{gryvalues} },
        {label : 'Pink Series', values : $attribute{pnkvalues} },
        {label : 'Blue Series', values : $attribute{bluvalues} },
        {label : 'Red Series',  values : $attribute{redvalues} } ] }"/>
</sun:markup>

</define>
</composition>
```

Figure 3-3: chart3.jsf

a. Select **Other Sources --> resources**.

b. Create an empty file named "**chart3.jsf**".

c. Add the content defined in Figure 3-3 to "**chart3.jsf**".

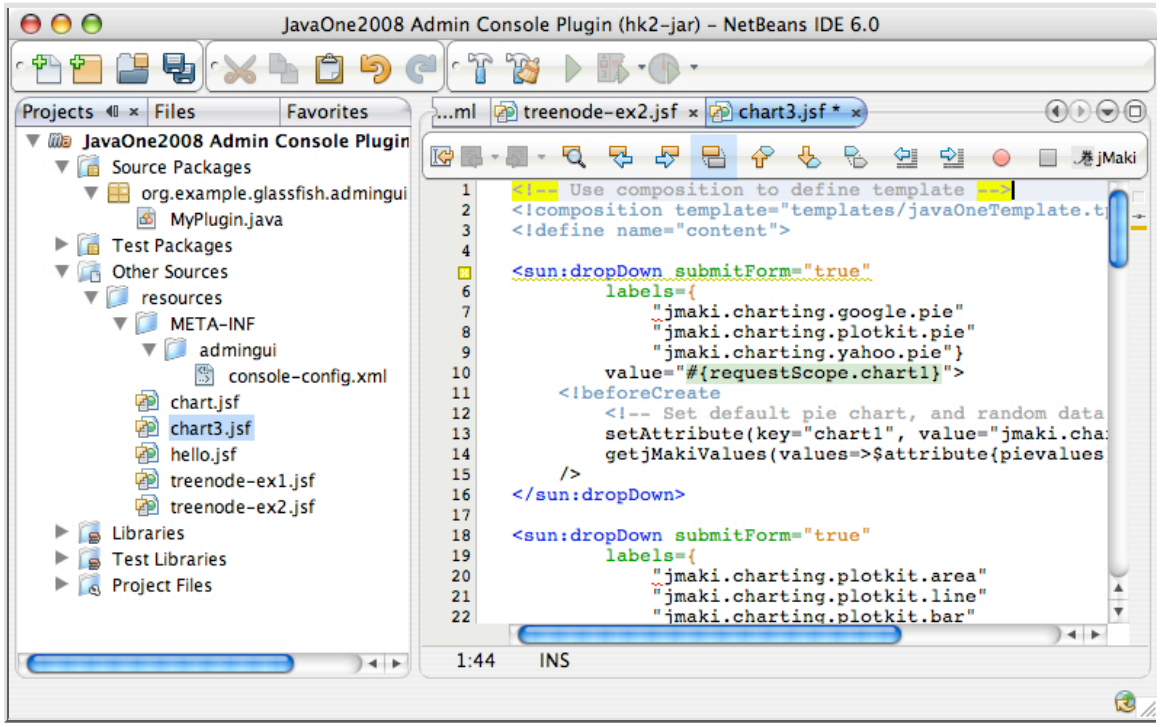


Figure 3-4: Create Chart3.jsf

Congratulations! You've completed creating a JSF page that shows the jMaki chart in a template!

Back to step overview

B. Modify your console-config.xml.

You now need to add an `IntegrationPoint` to plugin your Exercise 3 tree node into the Administration Console. Just as in the previous exercise, you need to edit the `console-config.xml` file to add an `IntegrationPoint`.

```
<integration-point
    id="Third"
    type="tree"
    priority="300"
    parentId="j1root"
    content="treenode-ex3.jsf"
/>
```

Figure 3-5: Add Intergation-point

To complete this step outside NetBeans, add the code in Figure 3-5 specifying the `IntegrationPoint` to your `console-config.xml` file. After completing this, you may proceed to Step C.

1. Double-click on "**console-config.xml**" under **Other Sources --> resources --> META-INF --> admingui**
2. Edit the file and add the `IntegrationPoint` in Figure 3-5 (refer to Solution 3 for the entire file):

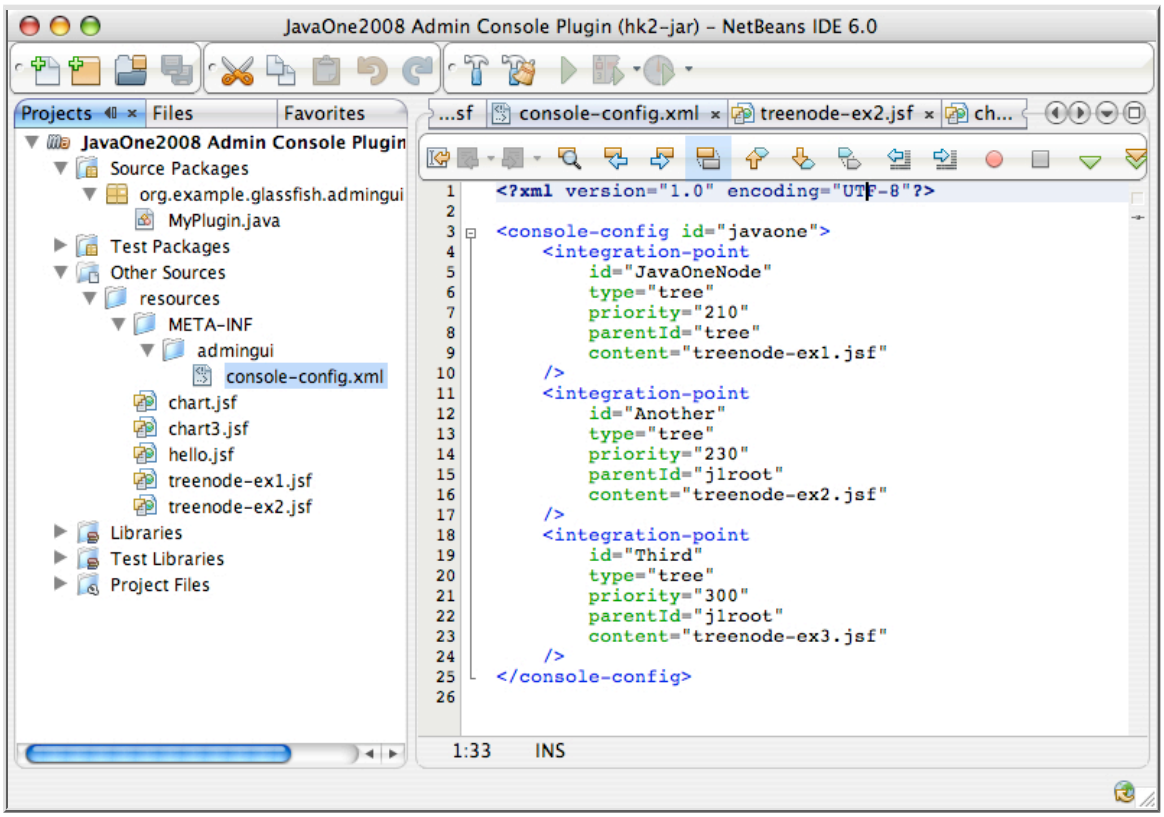


Figure 3-6: Modify console-config.xml

Back to step overview

C. Specify Your Tree Node Layout.

Next create a new file "`exercise-3.jsf`", which is specified as the "content" of the `IntegrationPoint` you just created.

```
<sun:treeNode
    id="third"
    text="Exercise #3"
    url="javaone/chart3.jsf"
    imageURL="resource/images/instance.gif"
    target="main"
/>
```

Figure 3-7: Add treeNode

1. Select **Other Sources --> resources**.
2. Create an empty file named **treenode-ex3.jsf** under **resources**.
3. Edit "**treenode-ex3.jsf**" to specify the new tree node.

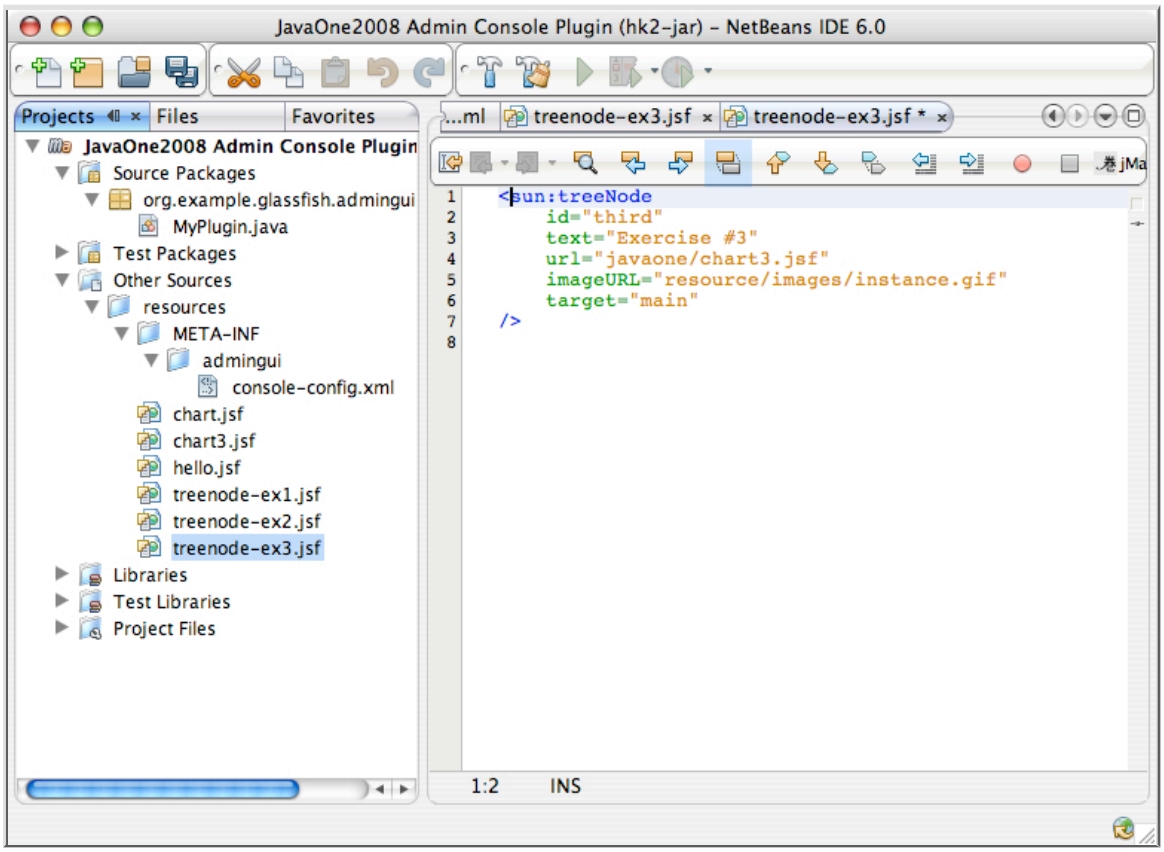


Figure 3-8: Edit of treenode-ex3.jsf

Back to step overview

D. Build and Install Your GlassFish v3 Plugin.

1. **Build your plugin**

```
cd gfplugin/myplugin
mvn install
```

After the above steps are finished, you will have the jar file named `gfplugin/myplugin/target/console-myplugin-1.0.jar`. This jar file is ready to be integrated into GlassFish.

2. **Install your plugin**

You now need a Terminal to enter some commands. Please bring up a terminal window.

The GlassFish v3 server is installed in the `gfplugin/glassfish` directory. The following steps show you how to integrate and test your plugin module.

a. Go to your `gfplugin` directory.

```
cd <path-to-gfplugin>
```

b. Shut down the server which may have been started in Exercise 1.

```
glassfish/bin/asadmin stop-domain
```

c. Integrate your plugin into GlassFish by copying it to the GlassFish modules directory:

```
cp myplugin/target/console-myplugin-1.0.jar glassfish/modules/console-myplugin-1.0.jar
```

d. Start GlassFish:

```
glassfish/bin/asadmin start-domain
```

e. In the browser enter the following URL to bring up the Administration Console:

```
http://localhost:8080/admingui/index.jsf
```

Expand the JavaOne tree node which is displayed in the navigation tree. Note the new tree node "Exercise #3" under it. Click on the "Exercise #3" tree node which displays the charts and notice that there is a header added to the charting page.

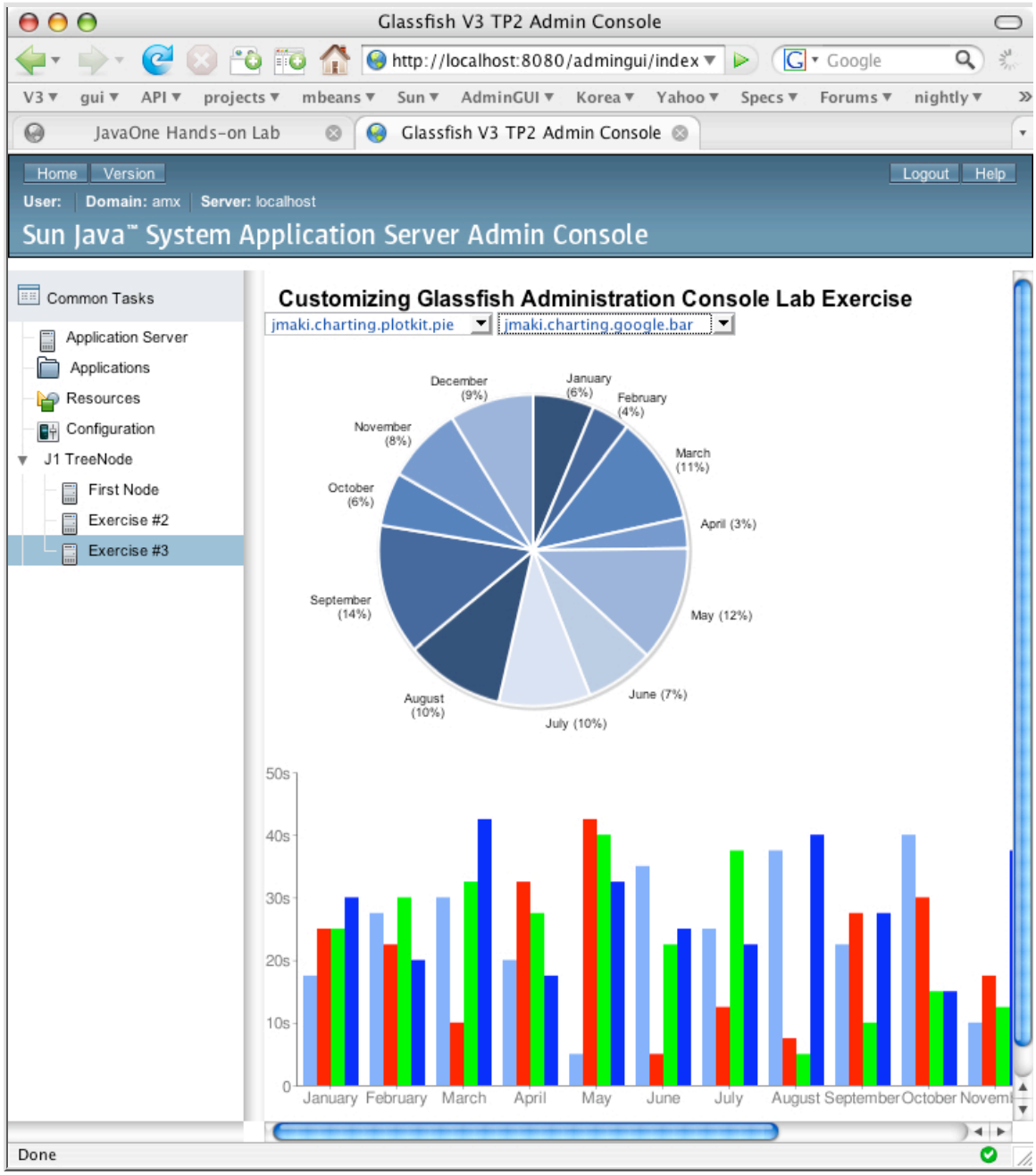


Figure 3-9: Browser showing the plugin page with charts.

Congratulations! You have customized the Administration Console and plugged in your own tree node to show the jMaki Charts with a customized header.
Try to change the dropdown box to select another charting style

Summary:

In this exercise, you have created a JavaServlet Page with jMaki Charting, the chart is customized by using a template. You added the plugin module with the additional tree node to the Administration Console.

Go to Exercise 3 Solution

Go to Exercise 2

Go on to the summary of this Lab

Congratulations!
You have successfully completed the
LAB-4520: Plug into GlassFish v3 with JavaServer Faces and jMaki
Hands-on Lab!!

In this lab you completed 3 exercises. In each of those Exercises you learned a little bit more about building a plugin module for the GlassFish v3 Application Server. Let us review what you have accomplished.

Exercise 1:

In this exercise you completed your first GlassFish v3 Plugin Module. This included implementing an HK2 GlassFish plugin module which was installed in the `modules` directory of the GlassFish v3 Application Server. In that plugin module, you implemented the `ConsoleProvider @Contract interface`, and annotated your implementation with the `@Service` annotation. This allowed the GlassFish v3 / HK2 environment to discover your plugin module, making it available to the GlassFish Administration Console. Your `ConsoleProvider` implementation referenced your `console-config.xml` file, which in turn declared an `IntegrationPoint` enabling you to add a tree node to the GlassFish Administration Console. From this tree node, you created a link to your "Hello World!" page. Well done!

Below is a picture of what Exercise 1 looked like at its completion.

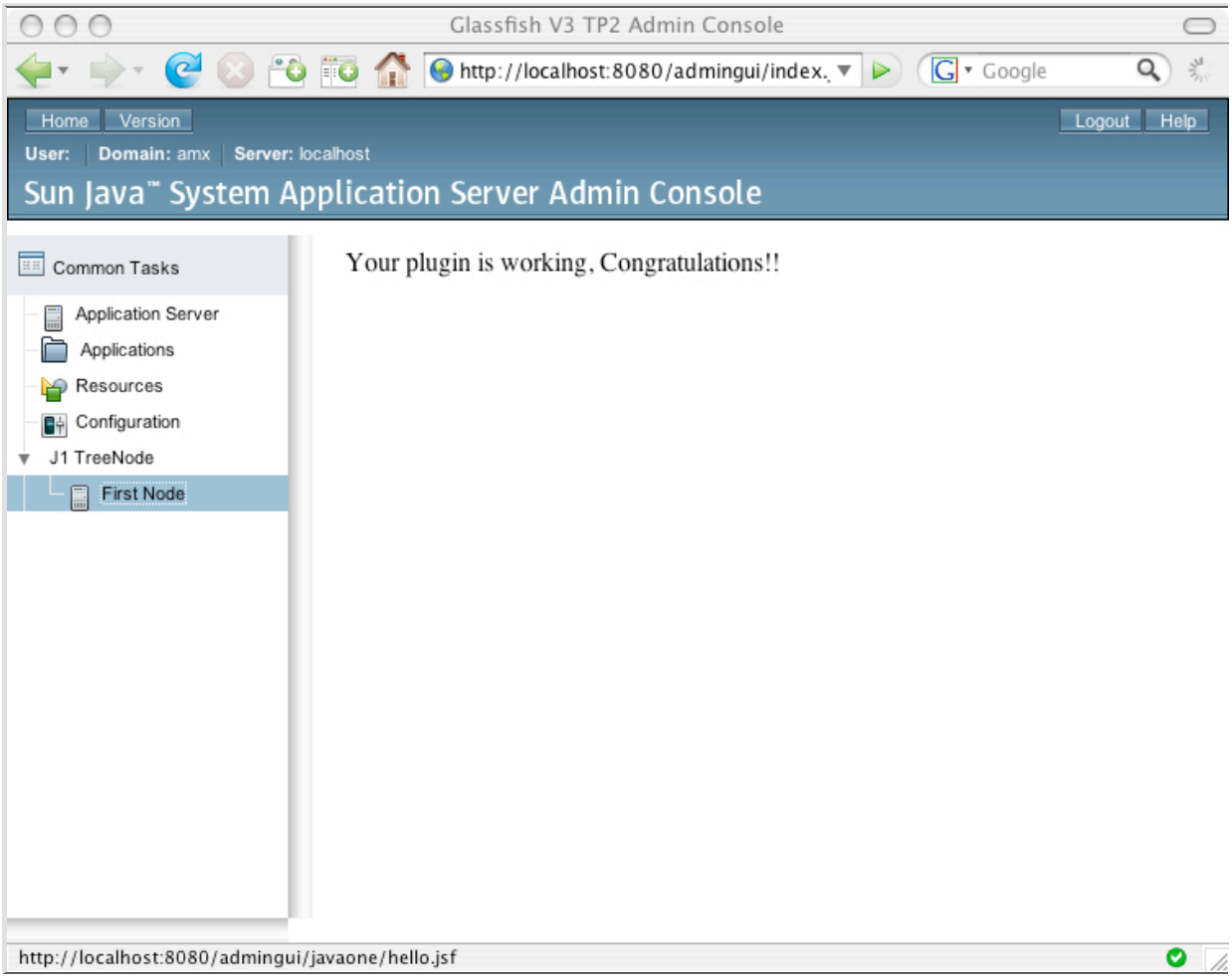


Figure 1: Exercise 1 Completed.

Exercise 2:

In Exercise 2, you learned out to use jMaki Charting. You created another JSFTemplating page, and another tree node. You integrated these into the GlassFish Administration Console by defining another IntegrationPoint, this time using your previous IntegrationPoint as the parent. This exercise demonstrated how easy it is to bring graphing to the GlassFish Administration Console, or to any application by leveraging jMaki. Congratulations!

The result of Exercise 2 is shown in the image below.

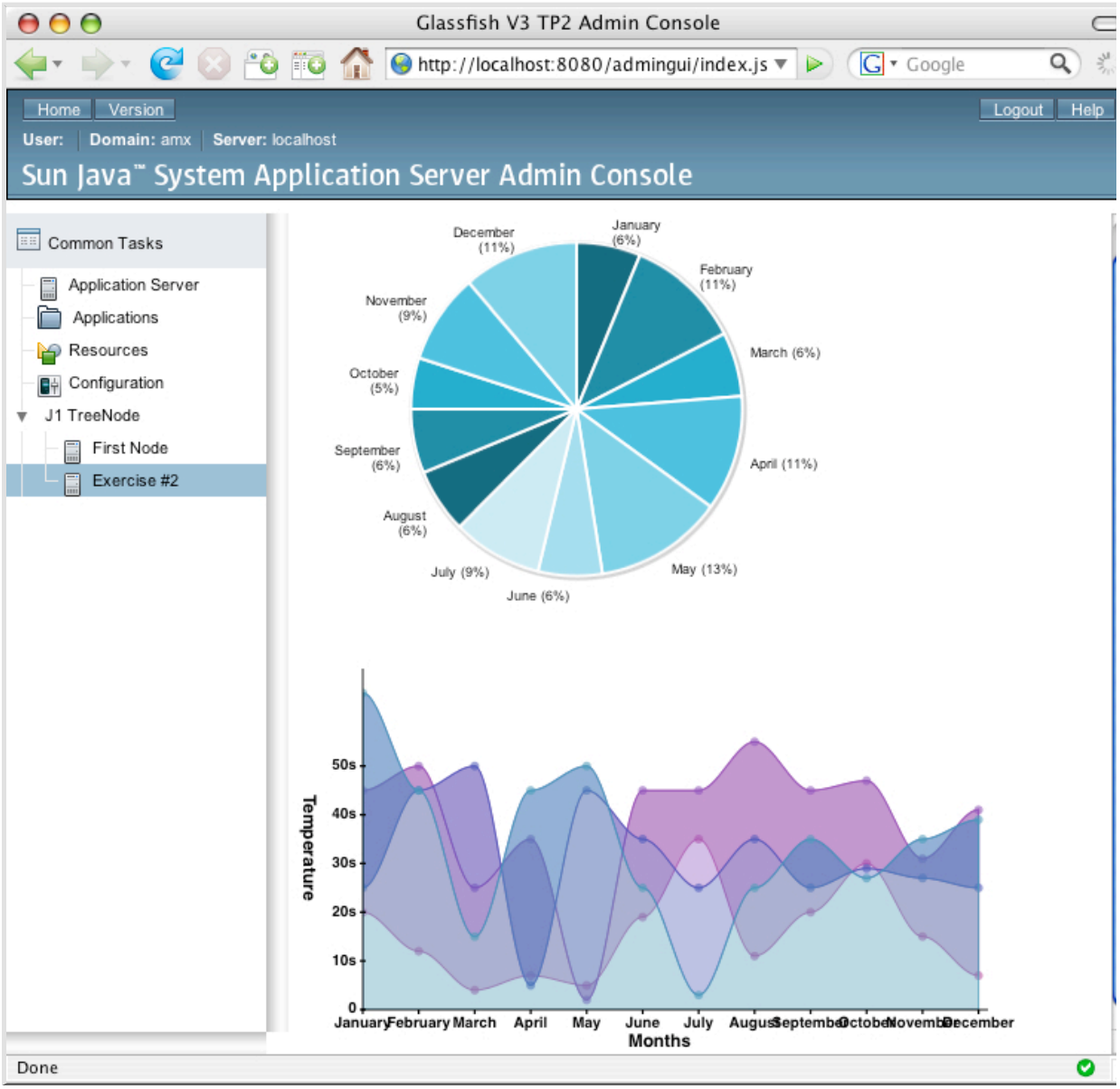


Figure 2: Browser showing the plugin page with charts.

Exercise 3:

In Exercise 3, you learned how to use Facelets-style templating to allow the template to specify parts of your page that don't change, while allowing you provide content. This technique greatly reduces the effort required to maintain your content as it is less mixed with ordinary template text. For the GlassFish Administration Console, it provide a convenient way for the infrastructure of a page, and its look and feel, to be provided automatically.

Additionally in the Exercise, you learned how to make use of JSFTemplating handlers to perform actions during events. You were able to retrieve random data from the server to generate dynamic charts, and you were able to modify the look of the charts via a drop-down menu. This menu demonstrated multiple charting frameworks, however, you **did not** need to learn multiple charting frameworks -- you only need to learn jMaki.

This lab also had you integrate your plugin as in the previous labs. You created your 3rd tree node and gained exerience in configuring IntegrationPoints in the GlassFish Administration Console. Great job getting this far!

The result of Exercise 3 is shown in the image below.

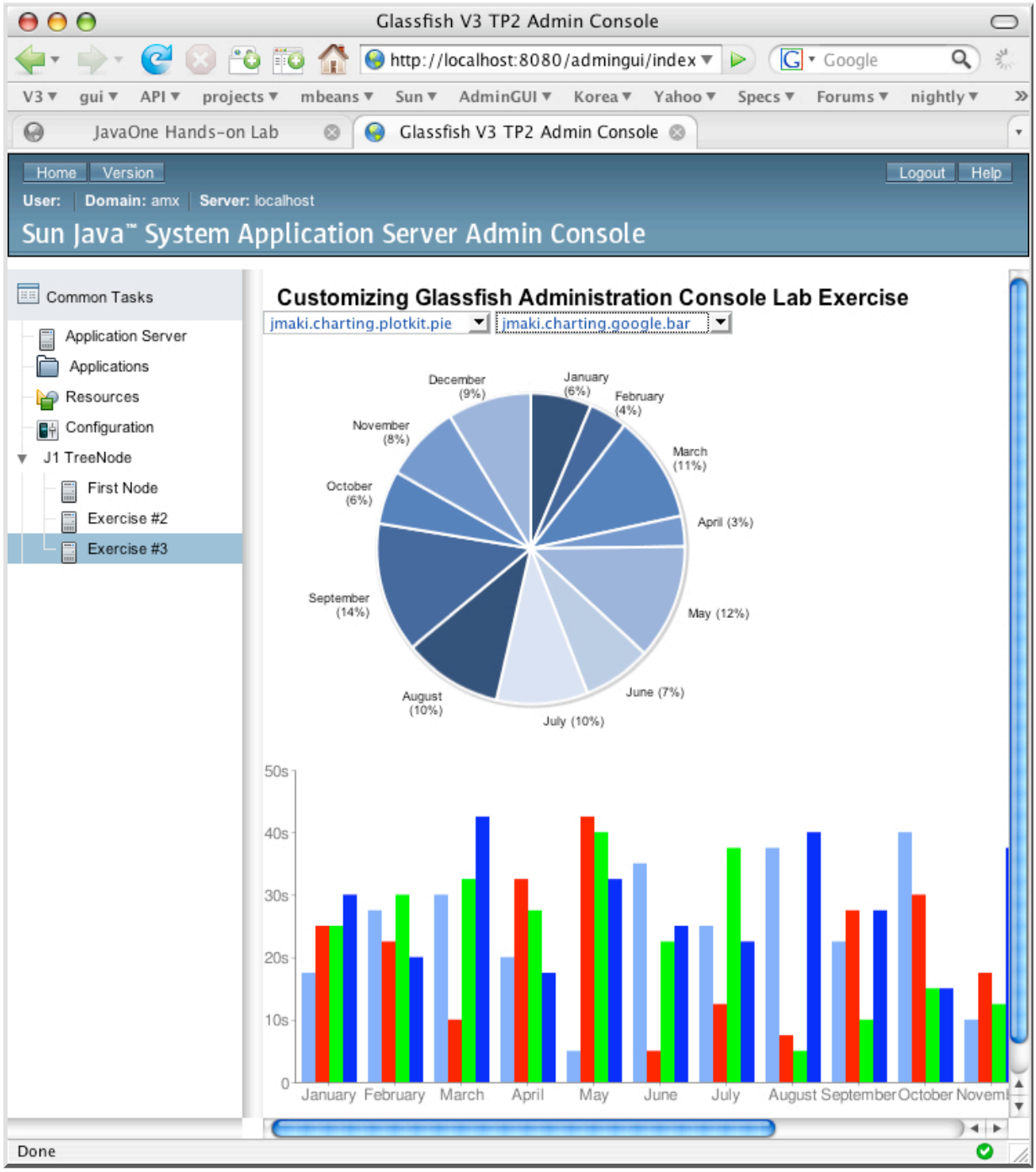


Figure 3: Exercise 3 Completed.

Get Involved!

Now that you have completed this Hands-on Lab, you have gain a significant amount of experience integrating into GlassFish v3. What are you going to do what that experience? The GlassFish community welcomes you to participate! You are encouraged to file bugs,

recommend features you would like to see. You can integrate your person, or company products and services into the GlassFish platform and share it with the rest of the community. You are welcome to take GlassFish technology as a whole, or in pieces to help you in your day-to-day work, or your weekend hobby. You are invited to meet with the GlassFish developers during planning meetings, day-to-day on IRC, or on email or Forums.

To learn more about how you can meet the people behind GlassFish, or to become a "person behind GlassFish", visit some of the sites below. Also come visit at the GlassFish booths at JavaOne or other conferences.

Where to send questions or feedback on this lab or its content:

- You can send technical questions via email to the authors of this Hands-on lab (and experts on the subject), or you can post the questions to the web.
  - JavaOne 2008 Hands-on Lab Forum
- You can send your other questions to the public mailing lists and forums.
  - **GlassFish Forums** (http://forums.java.net/jive/category.jspa?categoryID=58)
  - **GlassFish Email Lists** (https://glassfish.dev.java.net/servlets/ProjectMailingListList)
  - **JSFTemplating Email List** (https://jsftemplating.dev.java.net/servlets/ProjectMailingListList)
  - **JSFTemplating IRC Channel** (https://jsftemplating.dev.java.net/irc.html)
  - **jMaki Email List** (https://ajax.dev.java.net/servlets/ProjectMailingListList)

Thank you for completing this Hands-on Lab!