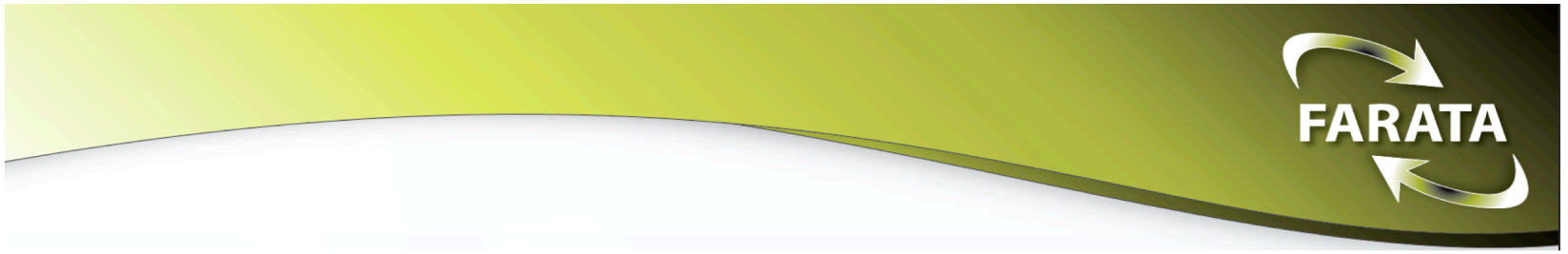


Java Programming
Unit 14

RMI
Java EE
Servlets

Remote Method Invocation (RMI)

RMI allows JVMs communicate with each other.

With sockets, the Java client was directly connecting to Java server running on a different JVM.

With RMI, Java client will make a method call *that looks as if this method is running in the same JVM*, but it's not. Only a *proxy* (*a stub*) of the remote method exists in the client's JVM.

Finding Remote Objects

RMI clients find remote services by using a naming service, which must run on a known host and port number.

The RMI server can start its own *registry* that offers naming services for RMI clients. The behavior of the registry is defined by the interface `java.rmi.registry.Registry`

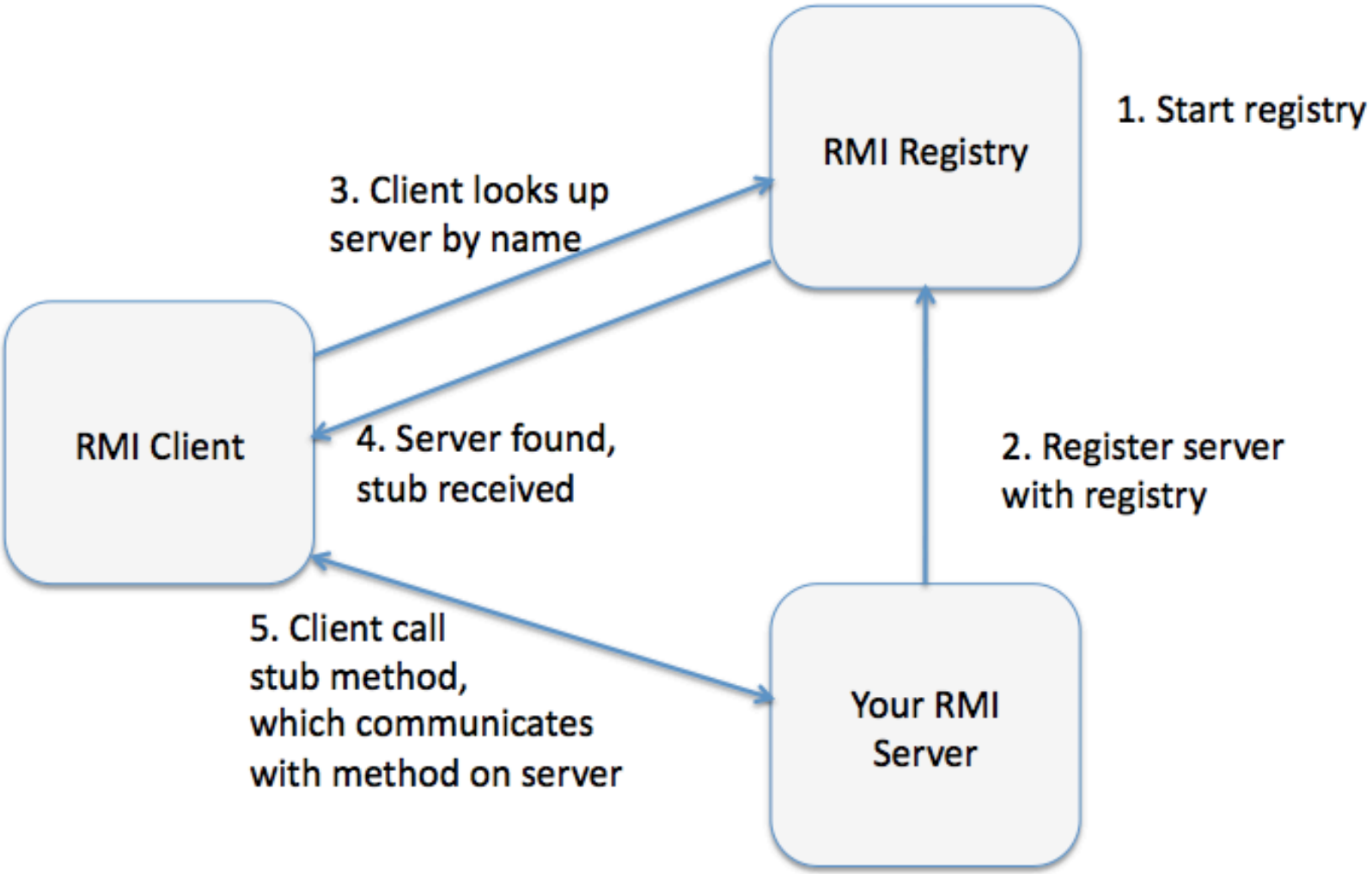
By default, the RMI registry runs on port 1099

The client obtains a reference to a remote object by looking up its name in the registry. This lookup returns to the client a remote reference a.k.a. **stub**.

The method `lookup()` takes the service name URL as an argument in the following format:

rmi://<host_name>[:<name_service_port>]/<service_name>

RMI Players


RMI Registry
1. Start registry
3. Client looks up server by name
RMI Client
4. Server found, stub received
2. Register server with registry
5. Client call stub method, which communicates with method on server
Your RMI Server

Developing and Running an app using RMI

- Declare a remote Java interface
- Implement the remote interface in a Java class
- **Computer A:** Write a Java client that connects to remote server and calls remote methods
- **Computer B:** Start the registry and register the RMI server with it
- **Computer C:** Start the server and the client applications

Walkthrough 1

1. Download and import the project Lesson25 and review the code with the instructor.

2. Add the following statement at the line 12 of StartServer.java:

```
LocateRegistry.createRegistry(1099);
```

3. Add the import statement for `LocateRegistry`

4. Run StartServer and it should give a prompt

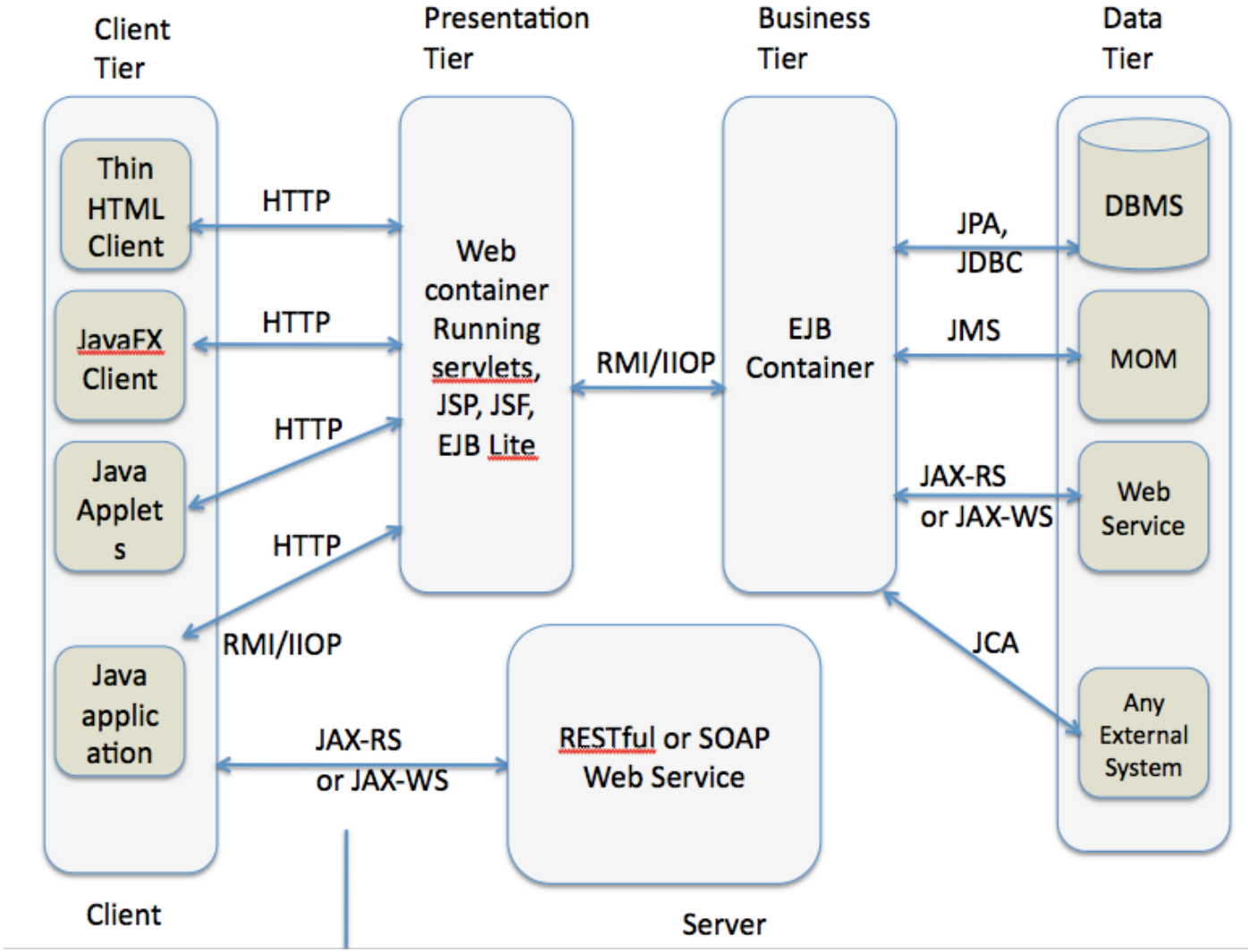
<QuoteService> server is ready

5.Run configuration for Client.java to specify one program argument: AAPL

6. Run the Client and you should get the random price quote like

The price of AAPL is: $1.333365174267477

Java EE 6 Overview



Java EE 7 major additions

- Released in 2013
- Improved JMS and Restful APIs
- Added Java API for JSON
- Added WebSockets support
- Java EE 7 Tutorial is here: http://docs.oracle.com/javaee/7/tutorial/doc/

Walkthrough 2 (GlassFish 4 server)

- Download and unzip GlassFish-4.0.zip from https://glassfish.java.net/download.html

- In the Command (or Terminal) Window switch to the directory glassfish4/bin where you unzipped GlassFish 4 and run *./asadmin start-domain domain1*. Windows users should use asadmin.bat start-domain domain1.

```
Yakov:bin yfain11$ ./asadmin start-domain domain1
Waiting for domain1 to start .....
Successfully started the domain : domain1
domain  Location: /Users/yfain11/glassfish4/glassfish/domains/domain1
Log File: /Users/yfain11/glassfish4/glassfish/domains/domain1/logs/server.log
Admin Port: 4848
Command start-domain executed successfully.
Yakov:bin yfain11$
```

You must have JDK 7 installed

Walkthrough 2 (continue)

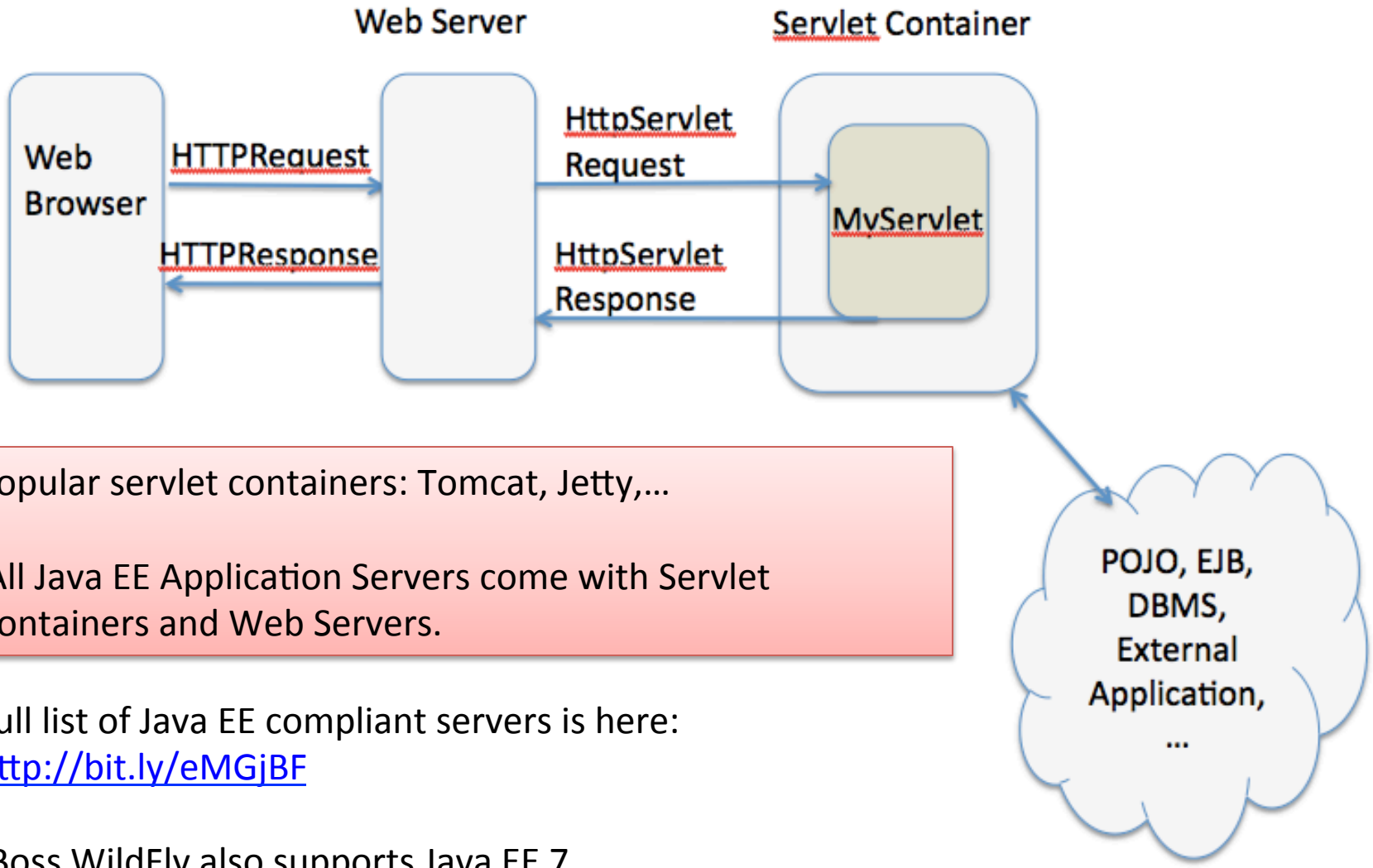
- Test your install by entering http://localhost:8080 - you'll see a Web page stating that from GlassFish server is running.

- Open the admin console by visiting http://localhost:4848

- For future starts and stops of the domain use the instructions from Quick Start Guide, section Starting and Stopping the Default Domain at

https://glassfish.java.net/docs/4.0/quick-start-guide.pdf

Web applications with Servlets



Popular servlet containers: Tomcat, Jetty,…

All Java EE Application Servers come with Servlet Containers and Web Servers.

Full list of Java EE compliant servers is here: http://bit.ly/eMGjBF

JBoss WildFly also supports Java EE 7

Sample site with servlets: MyBooks.com

1. The client's machine just needs a Web browser. The bookstore will consist of a number of HTML Web pages for getting user's input, send it in a form of HTTPRequest object to MyBooks.com.

2. *The computer that MyBooks.com is mapped to* has to run some Web Server software that *listens to* the users' requests. If a **Web server** receives a simple request of a static HTML content, it'll process the request and will send back HTTPResponse with the requested **static content**.

3. The Web site MyBooks.com will also run a **servlet container** with deployed Java servlet(s). If the Web server receives a user request to find books based on some criteria, it'll create and pass HttpServletRequest to the appropriate servlet deployed in the *servlet container*.

4. The servlet creates the HTML page with found books that meet requested search criteria, and sends this **dynamic content** to the Web server in HttpServletResponse, which wraps it inside HttpResponse object and sends it back to the user's Web browser.

5. The user's browser displays the received HTML document.

The Thin HTML Client

```
<HTML>
   <Head>
    <Title>Find a book</Title>
   </Head>

   <Body>
     Enter a word from the book title:

    <Form action=http://www.MyBooks.com/servlet/FindBooks method=Get>
       <input type=Text name=booktitle>
       <input type=Submit value="Search">
    </Form>

   </Body>
</HTML>
```

Walkthrough 3

- Create a plain text file *BookSearch.html* with the content from the previous slide.

- Open this file in a web browser using the menu File | Open, and enter any text in the input field and press the button Search.

- You'll get the error message because there is neither server, nor servlet FindBooks at this address.

How to write a Java servlet

- To create a servlet, write a Java class that extends from `HTTPServlet` and annotate it with `@WebServlet` annotation.

- The class HTTPServlet extends `GenericServlet`, which defines the method `service()`.

- The method `service()` receives the client's response and directs it to one of the methods of HTTPServlet descendent *that you have to override* such as `doGet()`, `doPost()` et al.

Your first Servlet

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;

@WebServlet(urlPatterns="/books", name="FindBooks" )
public class FindBooks extends HttpServlet {

  @Override
  public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException{

      // The code processing request goes here
      // The resulting Web page will be sent back via the
      // I/O stream that response variable contains

        PrintWriter out = response.getWriter();
        out.println("Hello from FindBooks");
  }
}
```

You must have the jar with javax.servlet.* available for the project to compile and run. Java EE SDK includes it, or you can use the one that comes with your application server.

Deploying a servlet

Specify servlet deployment parameters in the annotation `@WebServlet`:

```
@WebServlet(urlPatterns="/books",
name="FindBooks”
```

Every application server or servlet container has a directory known as *document root*.

For example, if you put the HTML file TermAndConditions.html in a subfolder legal of document root in the server MyBooks.com, the users would need to direct their Web browser to http://www.mybooks.com/legal/TermAndConditions.html.

In GlassFish application server, the default document root is directory /glassfish/domains/domain1/docroot.

In Apache Tomcat it's the directory webapps.
The servlets deployment directory will also be located in document root, but it will contain the subdirectories WEB-INF and (maybe) META-INF.

A Sample Directory Structure of a Deployed Servlet

document root dir
- WEB-INF
  - classes
    - com
      - practicaljava
        - lesson27
          - FindBooks.class
  - lib
- META-INF
  - manifest.mf

Walkthrough 4 (Eclipse + Glassfish)

- Shut down the GlassFish server if it's running (in bin dir run ./asadmin stop-domain or).
- In Eclipse Kepler IDE: right-click in the servers view: File, New Server, Download Additional Server adapters.
- Select GlassFish Tools, Press Next, Finish - after completeing install, Eclipse IDE will restart
- Right-click in the servers view select File, New Server, GlassFish 4.0. Select glassfish4/glassfish as your GlassFish Server Directory.
- 4. Press Next, and do not enter the password for the admin user. Press Finish.
- 5. You'll see a new entry for GlassFish 4 in the Eclipse Servers view. Right-click on it and start GlassFish server.

Creating a Servlet Project in Eclipse

- Eclipse for Java EE Developers simplifies creation of Web application. Switch to Java EE perspective and create Dynamic Web Project.

- You'll can also find see this menu under File | New | Other | Web.

Walkthrough 5 (start)

1. Create a dynamic Web project by selecting Eclipse menu File | New | Other | Web | Dynamic Web Project. Name it *lesson27*. Make sure that the target runtime is GlassFish 4.0. Press Next, Next, and Finish.

2. Observe the folder WebContent in your project. This is your server-side deployment part.

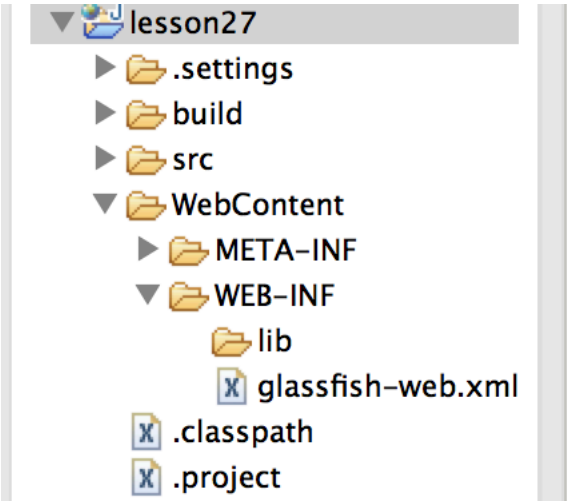
3. Generate new Servlet class: right-click on the project name and select New | Servlet. Specify com.practicaljava.lesson27 as the name of the Java package and the FindBooks as the class name. Press Next.

4. In the URL Mappings field select FindBooks, press Edit, and enter /book in the Patterns field.Press OK and Finish.

Walkthrough 5 (continue)

5. In the next window keep the defaults methods `doGet()` and `doPost()` and press Finish.

6. In the generated code note the annotated class declaration and methods `doGet()` and `doPost()`.




```
3  import java.io.IOException;
9
10 /**
11  * Servlet implementation class FindBooks
12  */
13 @WebServlet("/book")
14 public class FindBooks extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16
17     /**
```

Walkthrough 5 (end)

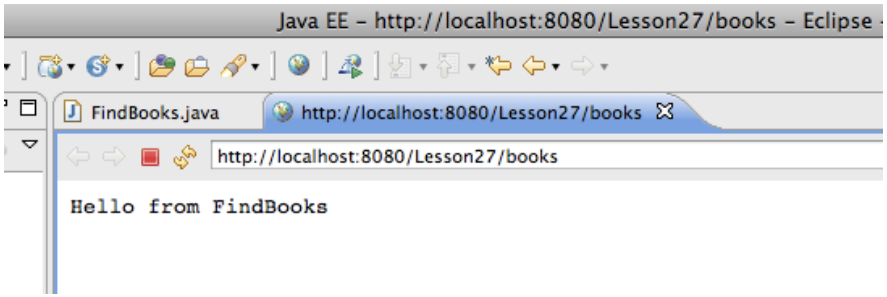
7. Add the following two lines inside the method `doGet()`:

```
PrintWriter out = response.getWriter();
out.println("Hello from FindBooks");
```

8. Correct the errors by importing the `PrintWriter` class.

9. Deploy the servlet in GlassFish: open the Servers view, right-click on the server and select Add and Remove from the menu. Select lesson27 in the left panel and add it to the right one. Check the content of the directory, where this app is deployed: *glassfish4/glassfish/domains/domain1/eclipseApps*

10. Run the servlet: right-click on FindBooks and select Run on Server. Confirm deployment under GlassFish. Eclipse will start its internal browser and display the following:



11. Copy the servlet's URL http://localhost:8080/lesson27/book from Eclipse to your Web Browser - you'll see the same output.

Homework

Study all the materials from Lessons 25-27 from the textbook.

1. Study the following HTTP tutorial:
   a) Part 1: http://bit.ly/17mLK87
   b) Part 2: http://bit.ly/11S639i

2. Do the assignment from the Try It section of the lesson 27.

3. After step 1 is complete, stop GlassFish and start it in the Debug mode. Set a breakpoint in the servlet's `doGet()` method.

   Submit the stock price request from your HTML file and observe the values of the local variables in `doGet()` while stepping through the Java code in the Eclipse debugger.

Additional materials

Watch the video on getting started with GlassFish 4
https://www.youtube.com/watch?v=DQpiuweG7W8

Blog post: "Selecting your Java EE 6 App Server":
http://blog.eisele.net/2013/01/selecting-your-java-ee-6-application.html

A Book on Java EE 7 by Arun Gupta:
http://www.amazon.com/Java-EE-Essentials-Arun-Gupta/dp/1449370179

RMI: http://docs.oracle.com/javase/tutorial/rmi/

Servlets: http://www.servletworld.com/

GlassFish server documentation: http://glassfish.java.net/docs/