Servlet

EPAM JAVA TRAINING

Mogilev 2018

PROBLEM AREA

When instead of this:

```
<html>
<body>
The current time is
always 4:20 PM
on the server
</body>
</html>
```

You want this:

```
<html>
<body>
The current time is
[insertTimeOnServer]
on the server
</body>
</html>
```

TWO THINGS THE WEB SERVER ALONE WON'T DO

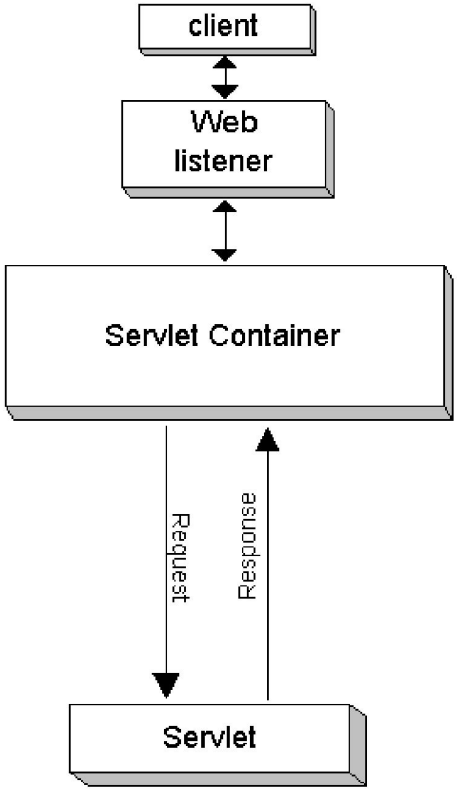
- **Dynamic content.** A dynamic page could be anything from a catalog to a weblog or even just a page that randomly chooses pictures to display.

- **Saving data on the server.** To process that form data, either to save it to a file or database or even just to use it to generate the response page, you need another app.

JAVA SERVLET

A **Servlet** is a Java class in Java EE that conforms to the **Java Servlet API**. A software developer may use a servlet to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML.

To deploy and run a Servlet, a **Servlet container** must be used



SERVLET CONTAINER

A servlet container is essentially the component of a Web server that interacts with the servlets. The servlet container is responsible for managing the **lifecycle** of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

- Apache Tomcat
- Jetty
- Glassfish
- Oracle Weblogic
- IBM WebSphere
- SAP NetWeaver

APACHE TOMCAT

Tomcat is an open source servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Oracle, and provides a "pure Java" HTTP web server environment for Java code to run.

http://tomcat.apache.org/

SERVLET 3.x vs 2.x

	3.x	2.x
Servlet, Filter, Listener Declaration	**web.xml** - optional	**web.xml** - required
Modularization of web.xml	can be done via **web-fragmets.xml** in **META-INF**	**web.xml** - monolith
Asynchronous support	**Yes**	**No**
Programmatic login/logout	**Yes**	**No**

SERVLET 2.5 EXAMPLE

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet2 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
                PrintWriter out = response.getWriter();
                Date today = new java.util.Date();
                out.println("<html> <body>"
                + "<h1 align=center>Hello HelloServlet2!</h1><br> Today is:" + today
                + "</body> </html>");
        }
}
```

DEPLOYMENT DESCRIPTOR 2.5 (web.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

 <display-name>Servlet 2 Example</display-name>

 <servlet>
 <servlet-name>Hello</servlet-name>
  <servlet-class>com.epam.HelloServlet2</servlet-class>
  <init-param>
  <param-name>servletName</param-name>
  <param-value>Servlet2</param-value>
 </init-param>
 </servlet>

 <servlet-mapping>
  <servlet-name>Hello</servlet-name>
  <url-pattern>/hello/*</url-pattern>
 </servlet-mapping>

</web-app>
```

SERVLET 3.0 EXAMPLE

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(asyncSupported = false, name = "HelloServlet", urlPatterns = { "/hello" })
public class HelloServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
                PrintWriter out = response.getWriter();
                Date today = new java.util.Date();
                out.println("<html> <body>"
                + "<h1 align=center>Hello Servlet3!</h1><br> Today is:" + today
                + "</body> </html>");
        }
}
```

DEMPLOYMENT DESCRIPTOR 3.0 (web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">

   <display-name>Servlet 3 Example</display-name>

</web-app>
```

DEPLOYMENT

1. Compile your servlet

>javac -cp %TOMCAT_HOME%/lib/servlet-api.jar Ch1Servlet.java

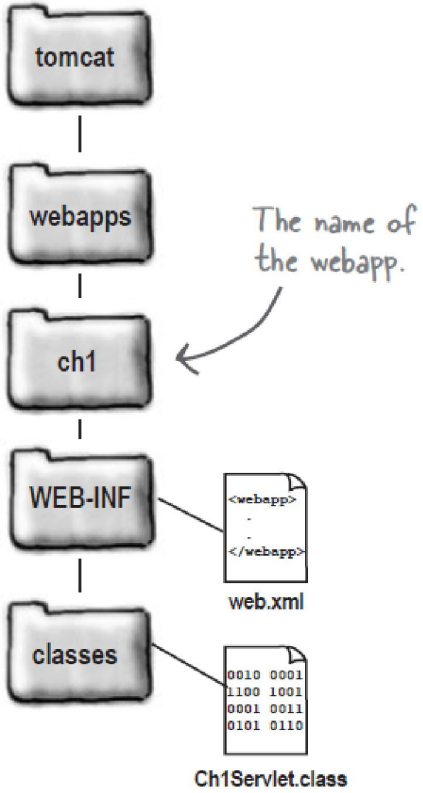
2. Build this directory tree under the existing tomcat directory

3. Start Tomcat

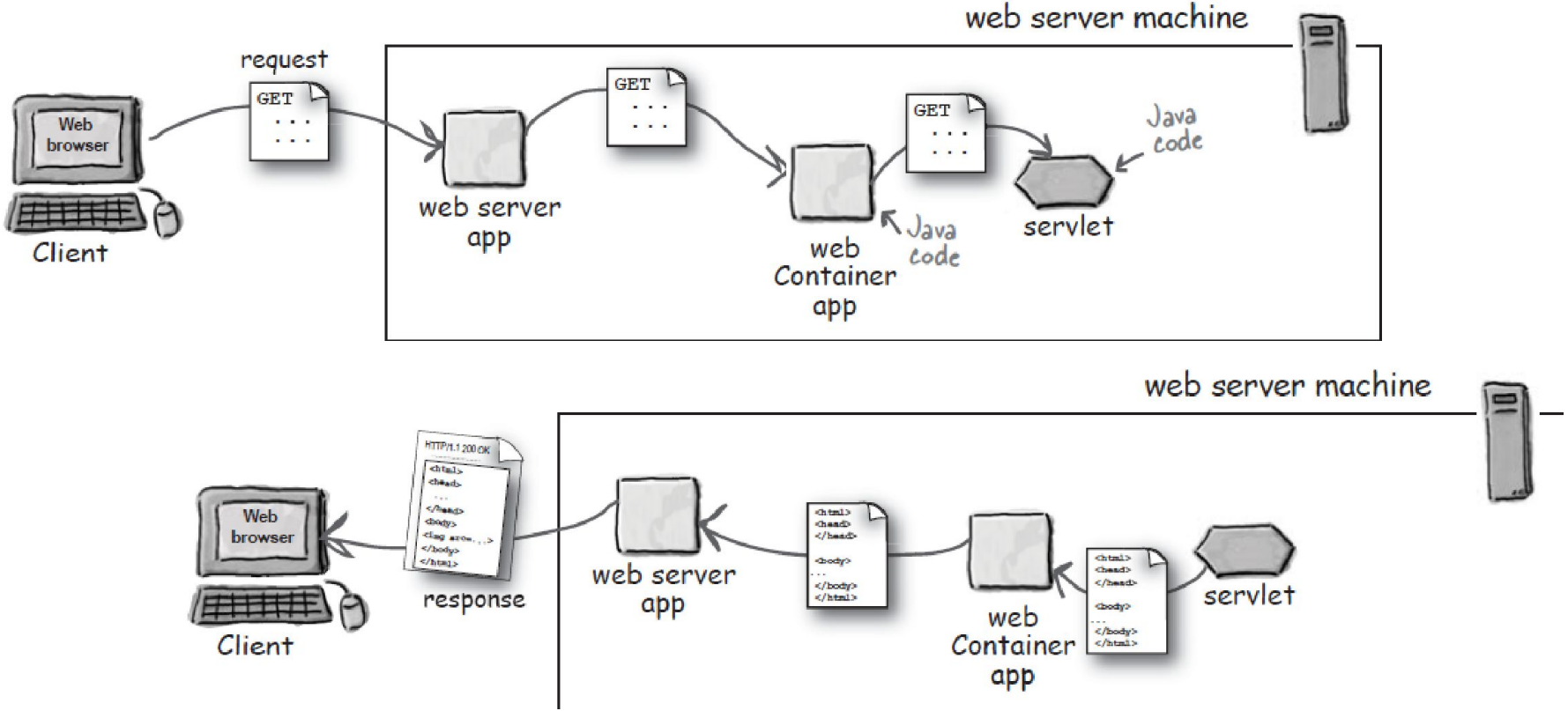
%TOMCAT_HOME%\bin\startup.bat

4. Go

http://localhost:8080/ch1/Serv1



SERVLETS DON'T HAVE A MAIN METHOD

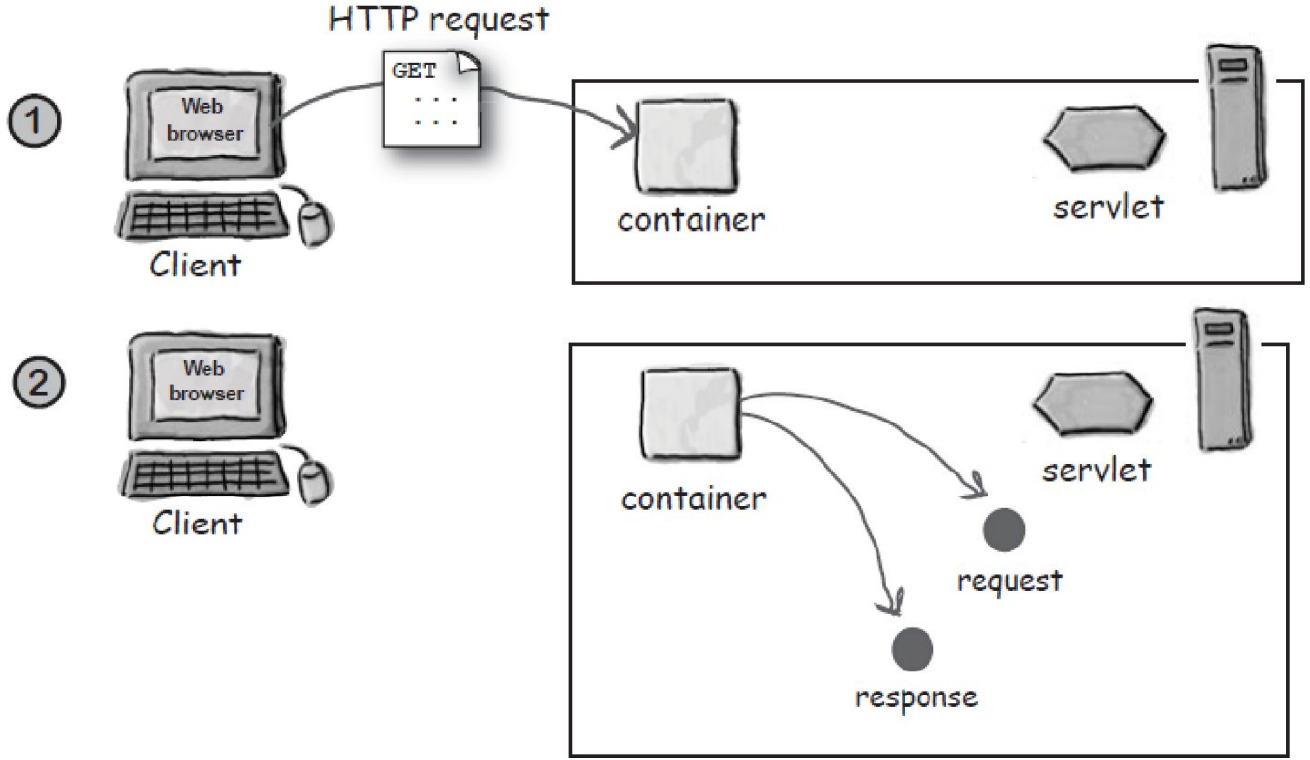


WHAT DOES THE CONTAINER GIVE YOU?

- Communications support
- Lifecycle Management
- Multithreading Support
- Declarative Security
- JSP Support

Thanks to the Container, YOU get to concentrate more on your own business logic instead of worrying about writing code for threading, security, and networking.

HOW THE CONTAINER HANDLES A REQUEST?



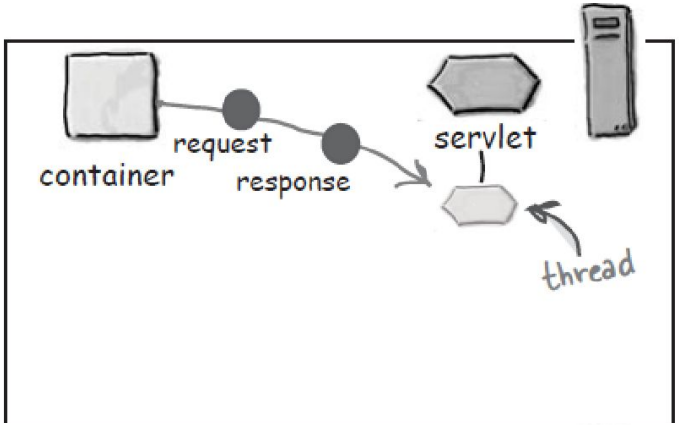
User clicks a link that has a URL to a servlet instead of a static page.

The container "sees" that the request is for a servlet, so the container creates two objects:

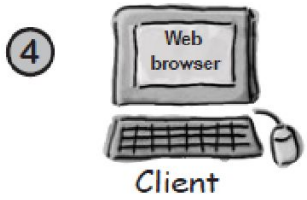
1) HttpServletResponse

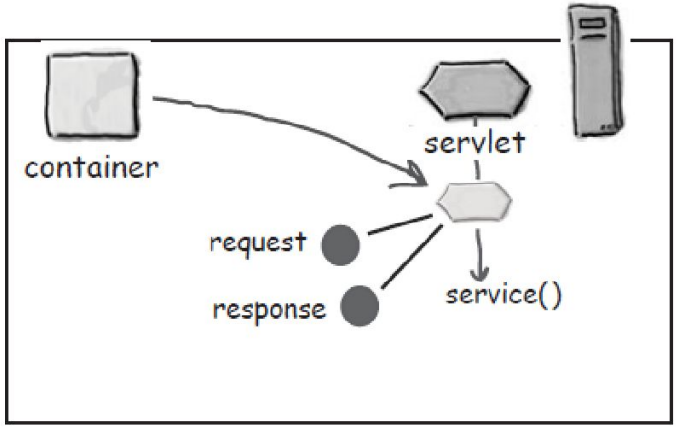
2) HttpServletRequest

HOW THE CONTAINER HANDLES A REQUEST?



The container finds the correct servlet based on the URL in the request, creates or allocates a thread for that request, and passes the request and response objects to the servlet thread.

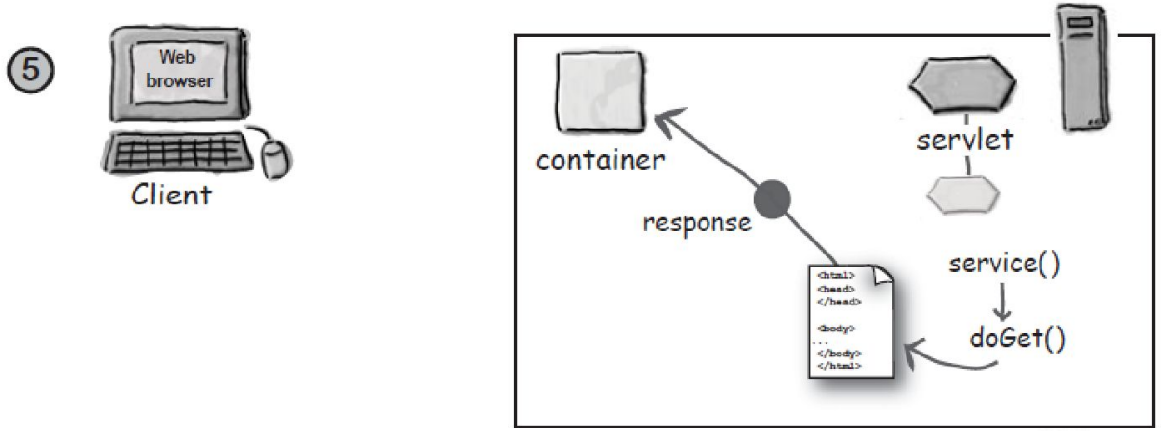




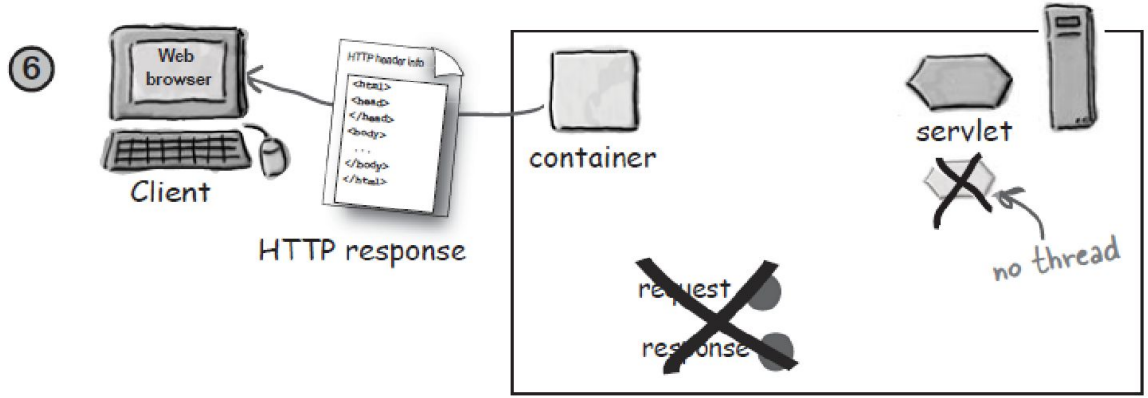
The container calls the servlet's service() method. Depending on the type of request, the service() method calls either the doGet() or doPost() method.

For this example, we'll assume the request was an HTTP GET.

HOW THE CONTAINER HANDLES A REQUEST?

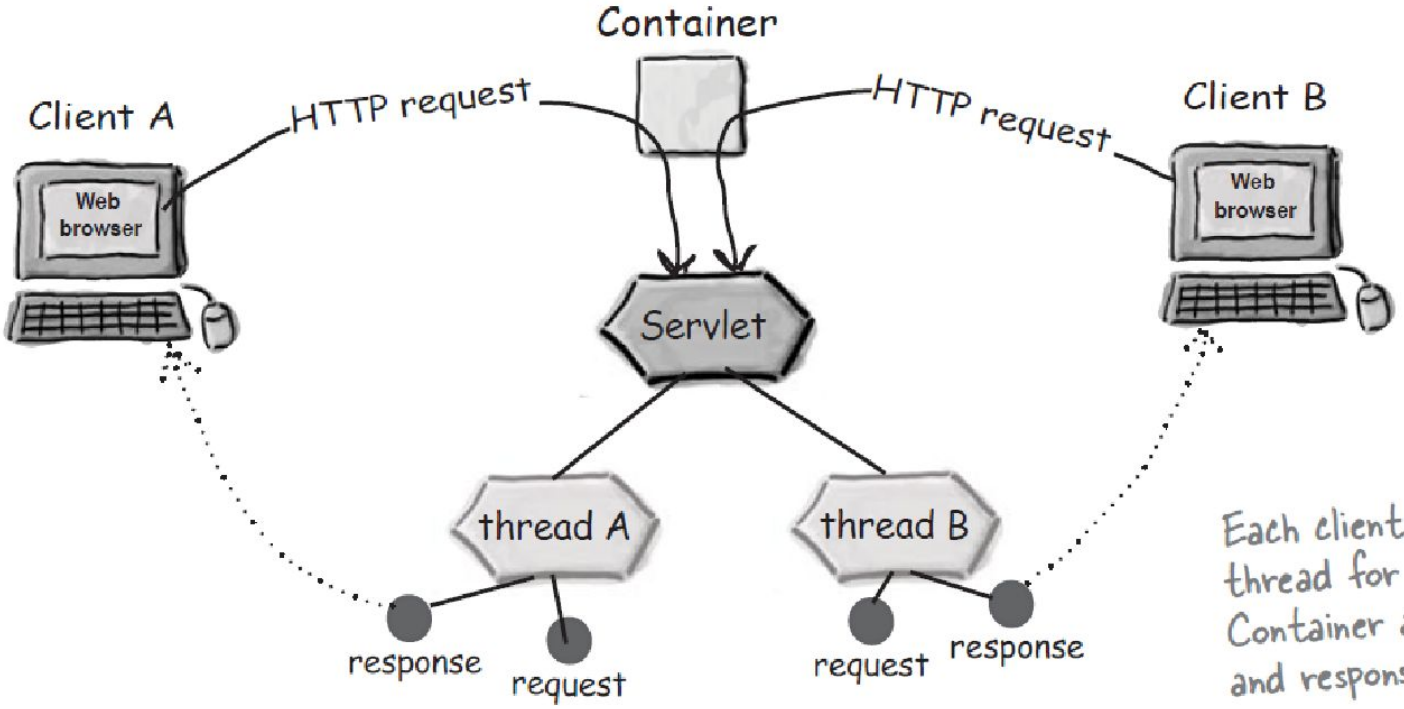


The doGet() method generates the dynamic page and stuffs the page into the response object. Remember, the container still has a reference to the response object!



The thread completes, the container converts the response object into an HTTP response, sends it back to the client, then deletes the request and response objects.

EACH REQUEST RUNS IN A SEPARATE THREAD



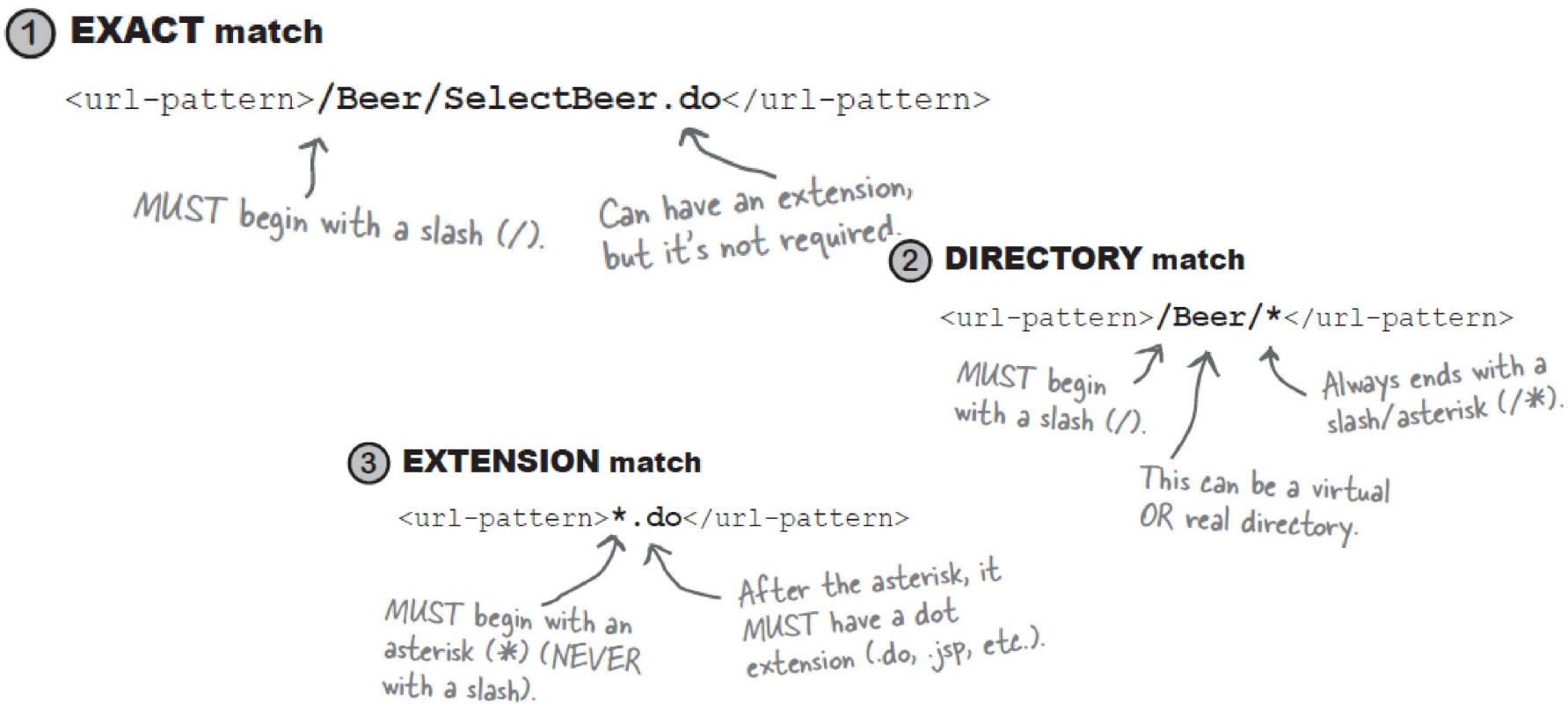
Each client gets a separate thread for each request, and the Container allocates new request and response objects.

URL MAPPING

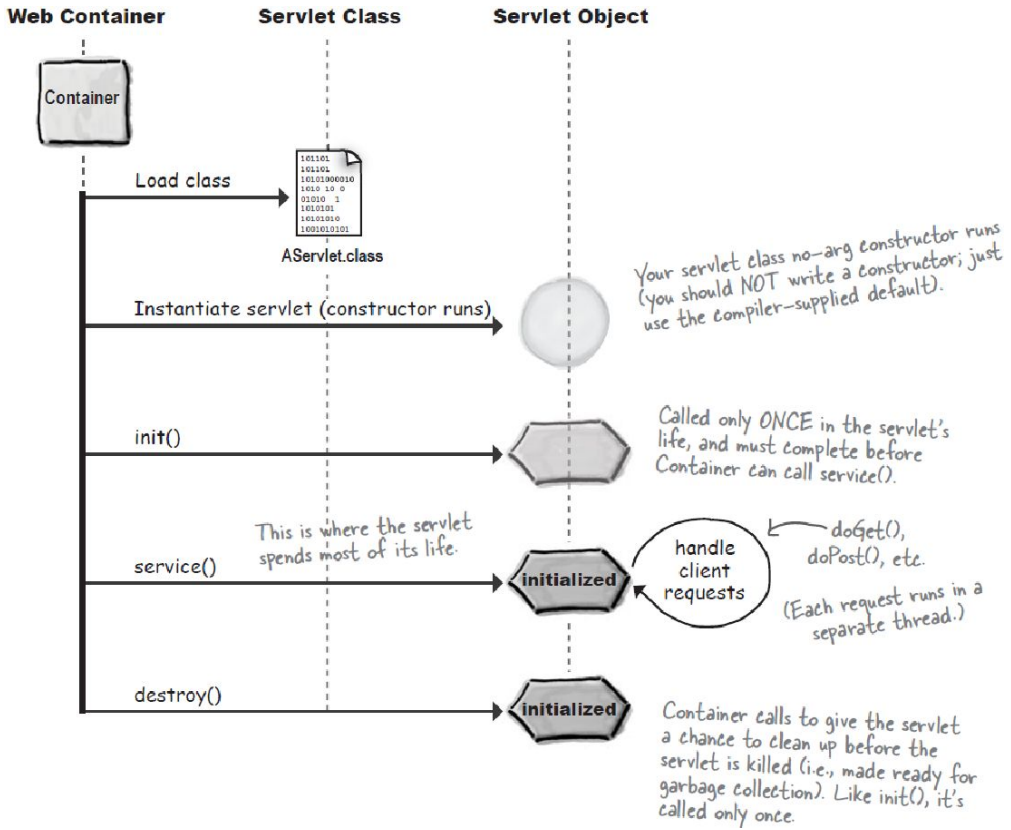
```
<servlet>
  <servlet-name>Internal name 1</servlet-name>
  <servlet-class>foo.Servlet1</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Internal name 1</servlet-name>
  <url-pattern>/Public1</url-pattern>
</servlet-mapping>
```

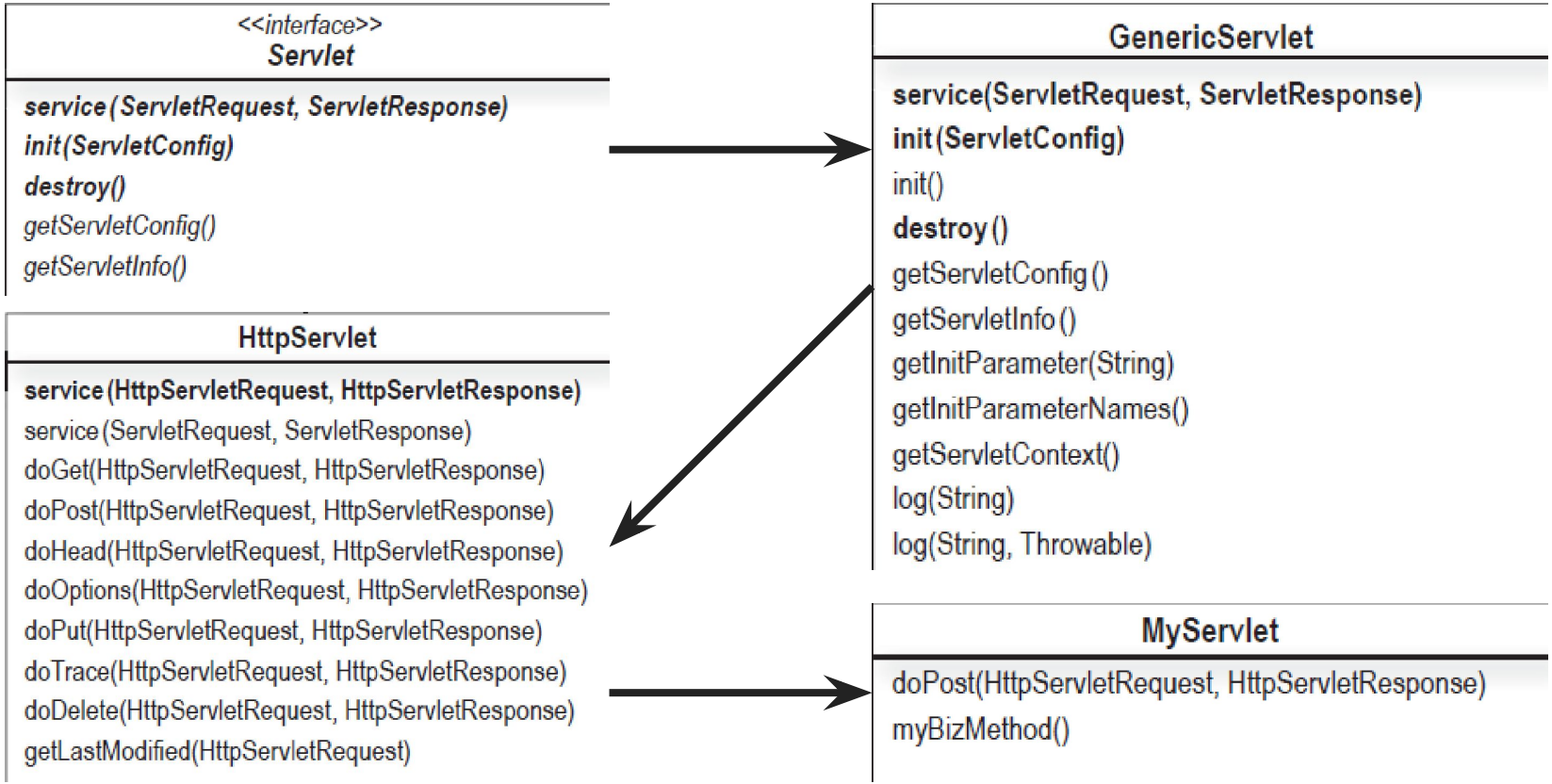
3 TYPES OF <URL-PATTERN> ELEMENTS



SERVLET LIFECYCLE



SERVLET HIERARCHY

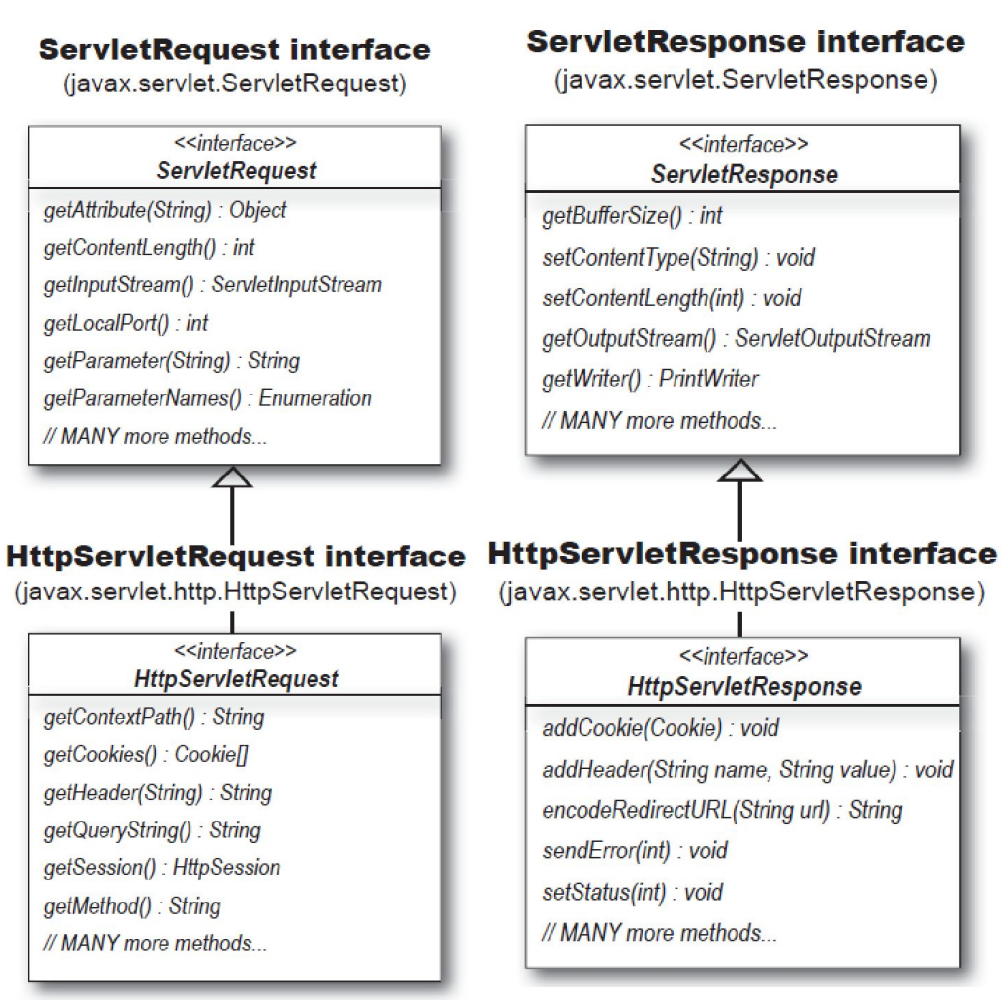


INIT PARAMETERS IN SERVLETS

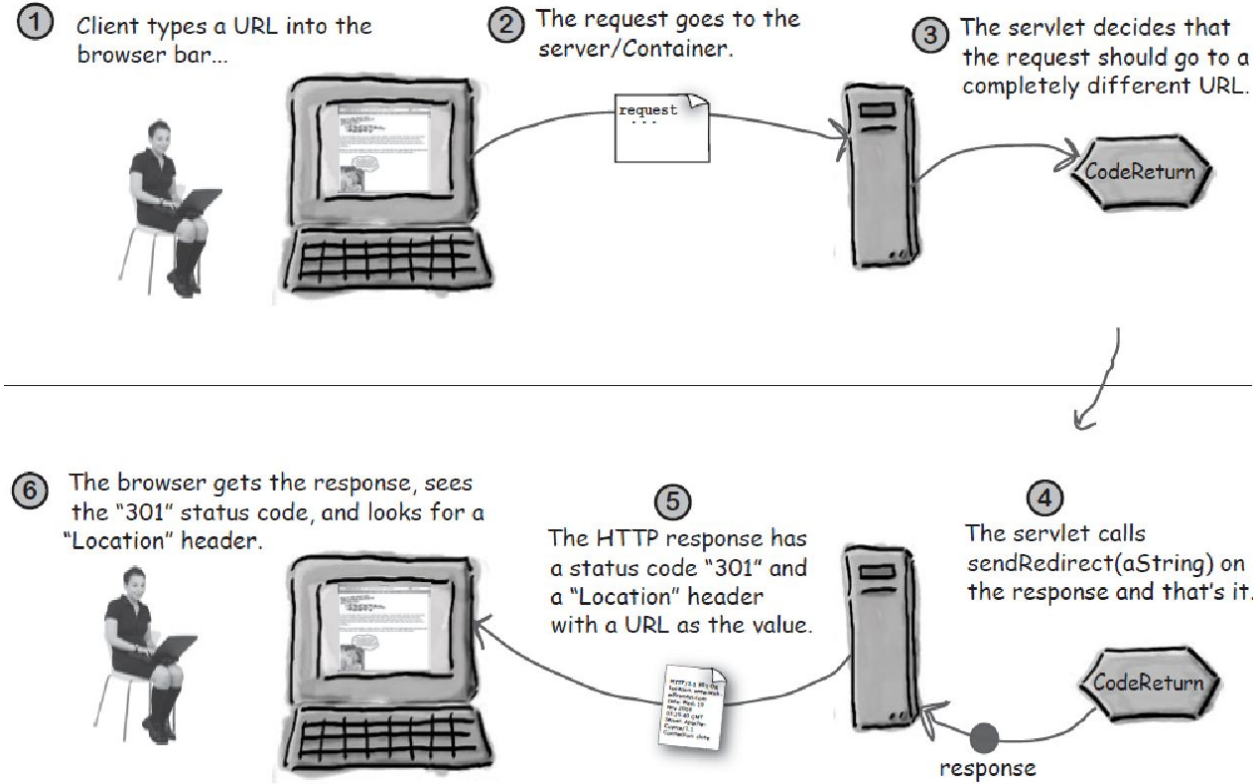
```
<servlet>
  <servlet-name>BeerParamTests</servlet-name>
  <servlet-class>com.example.TestInitParams</servlet-class>
  <init-param>
    <param-name>adminEmail</param-name>
    <param-value>likewecare@wickedlysmart.com</param-value>
  </init-param>
</servlet>
...
private String adminEmail;
private String mainEmail;

@Override
public void init(ServletConfig config) throws ServletException {
  this.adminEmail = config.getInitParameter("adminEmail");
}
```

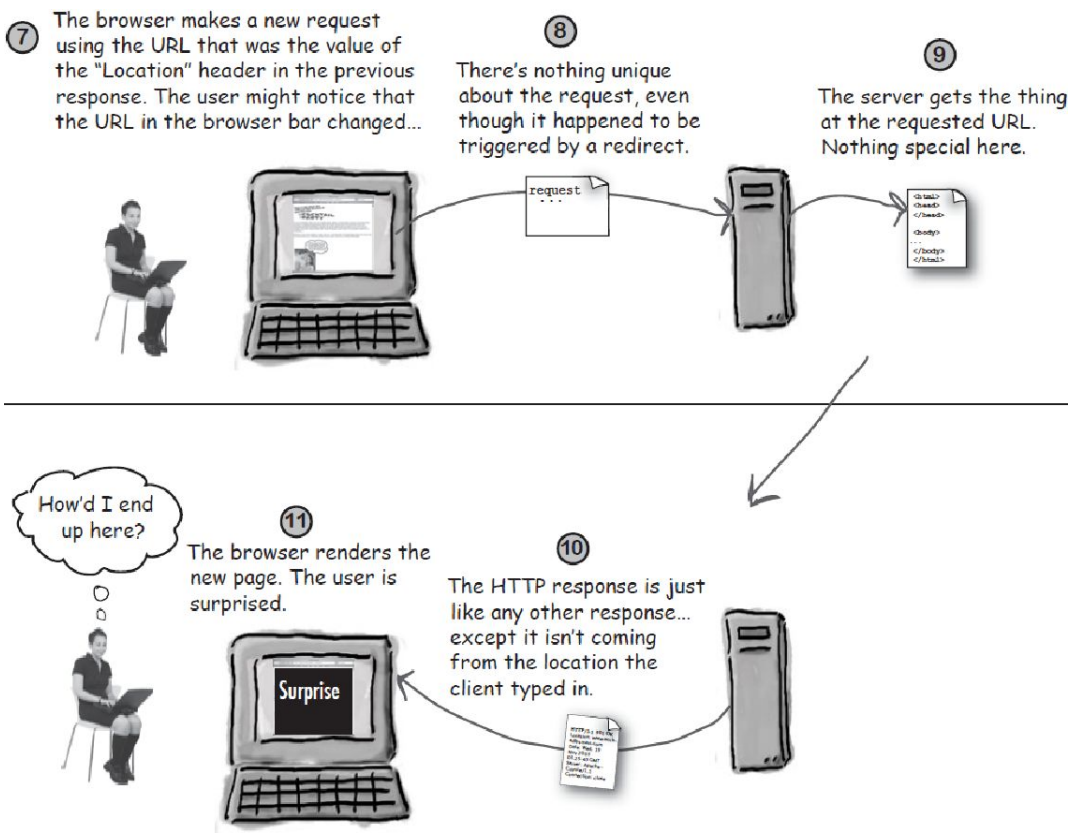
SERVLET REQUEST/RESPONSE



REDIRECT A REQUEST



REDIRECT A REQUEST



LET’S CREATE A WEB APPLICATION

mvn archetype:generate -DgroupId=by.epam.training -DartifactId=servlet-example -DarchetypeArtifactId=maven-archetype-webapp

Project structure

```
servlet-example
|-- pom.xml
`-- src
    `-- main
        `-- webapp
            |-- WEB-INF
            |   `-- web.xml
            `-- index.jsp
```

SUMMARY

- Servlet
- Servlet Container
- Deployment
- URL Mapping
- Servlet Lifecycle

«НОРМАЛЬНО ДЕЛАЙ –
НОРМАЛЬНО БУДЕТ»

THANK YOU.
QUESTIONS?