Web Applications

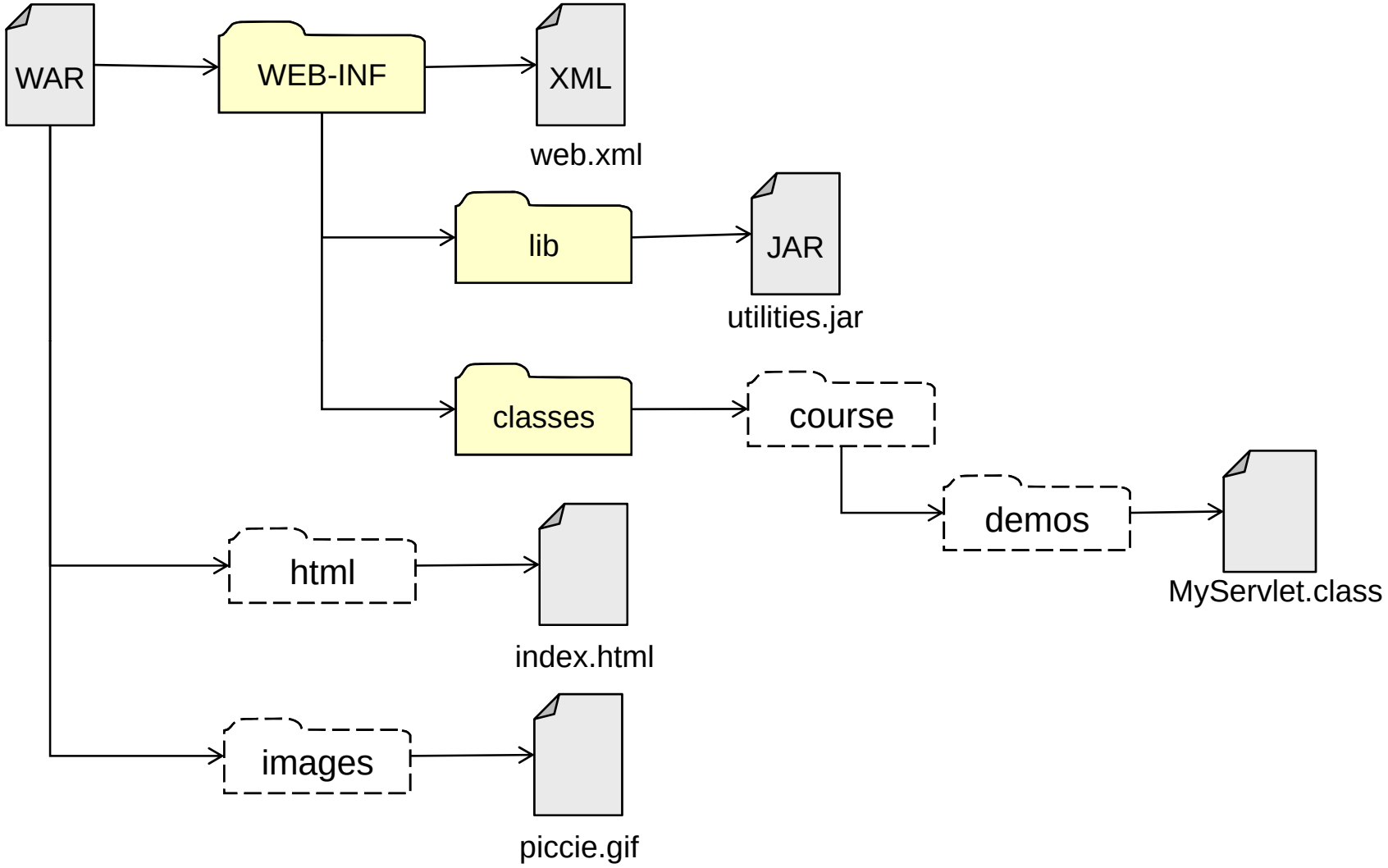
The Structure and Components of a JEE Web Application

© Garth Gilmour 2008
garth@ggilmour.com

The Structure of a Web Application

- The application is deployed in a ‘Web Archive’
  - A structured jar file with the extension ‘.war’
  - Most containers will accept an unzipped folder
- The archive contains a complete Web Application
  - Servlet classes, JSP pages, libraries and other resources
  - Libraries in other archives can be referenced in the manifest
- Dynamic content is placed in a ‘WEB-INF’ folder
  - Static content can be placed anywhere outside of it
  - Java Server Pages count as static content

© Garth Gilmour 2008



© Garth Gilmour 2008

The Contents of WEB-INF

- The ‘WEB-INF’ folder should contain three things
  - A folder called ‘classes’
    - Which contains a package hierarchy of class files
  - A folder called ‘lib’
    - Which contains zero or more JAR files
  - A configuration file named ‘web.xml’
    - Which the container uses to map Servlet classes to URL’s
- A classpath is created for each web app
  - Containing the ‘classes’ folder and all the jars in ‘lib’
  - Container specific libraries will come first
    - This can create dependency problems (e.g. XML Parsers)

© Garth Gilmour 2008

The Deployment Descriptor

- The ‘web.xml’ file contains
  - Configuration information for Servlets
  - Initialization parameters for magic numbers
  - Security roles and authorization schemes
  - Handlers mapped to exception types
  - Pages mapped to HTTP error codes
  - Listeners to respond to application events
  - References to Enterprise JavaBeans
- Note that which DTD you use is important
  - The container uses it to work out which version of the Servlet specification is being used

© Garth Gilmour 2008

A Simple Deployment Descriptor

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
        "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
        <servlet>
                <servlet-name> Fred </servlet-name>
                <servlet-class> com.p1.MyFirstServlet </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>Fred</servlet-name>
                <url-pattern>/servletNo1</url-pattern>
        </servlet-mapping>
</web-app>
```

© Garth Gilmour 2008

Configuration Using Servlet V2.4

- Servlet 2.4 uses a DTD rather than an XML Schema
  - Schemas have replaced DTD's in the XML world as the preferred means of defining an XML language
  - Schemas use the XML Namespaces notation
- The schema allows elements to occur in any order
  - So a 'servlet' tag can be followed by its 'servlet-mapping'

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                                    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
                version="2.4">
</web-app>
```

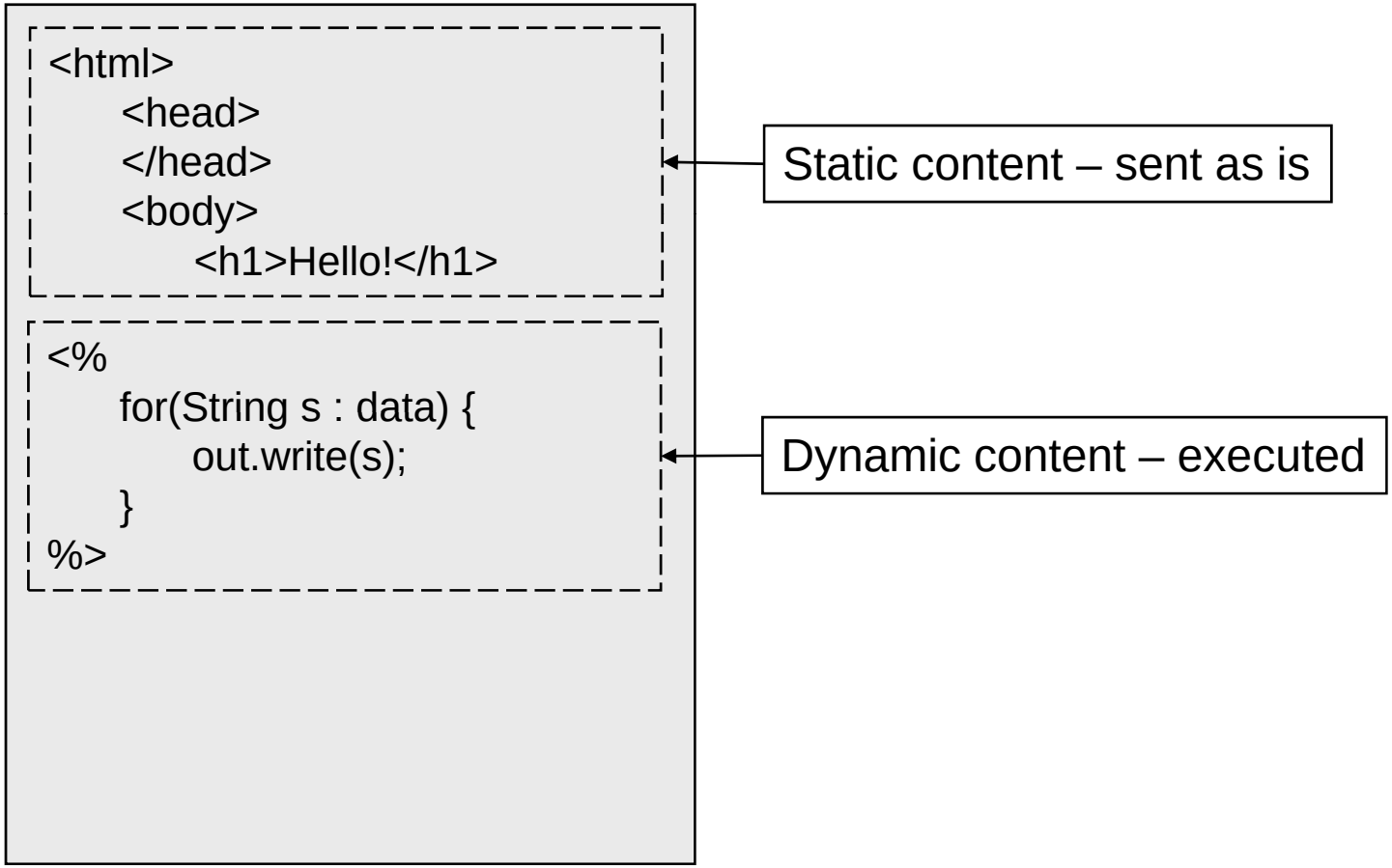
© Garth Gilmour 2008

JavaServer Pages

- JSP's are an alternative to writing Servlets
  - They avoid having to split HTML into dozens of 'println' calls
- JSP's contain a mixture of static markup and Java code
  - The static markup is streamed directly to the client
  - The Java code is executed and its output sent
- A JSP is converted into a Servlet by the container
  - At runtime everything is a Servlet
- JSP's closely resemble Active Server Pages
  - The syntax used in the pages is very similar
  - The implementation is very different
    - But similar to that used by ASP .NET pages

© Garth Gilmour 2008

The Server Page Concept



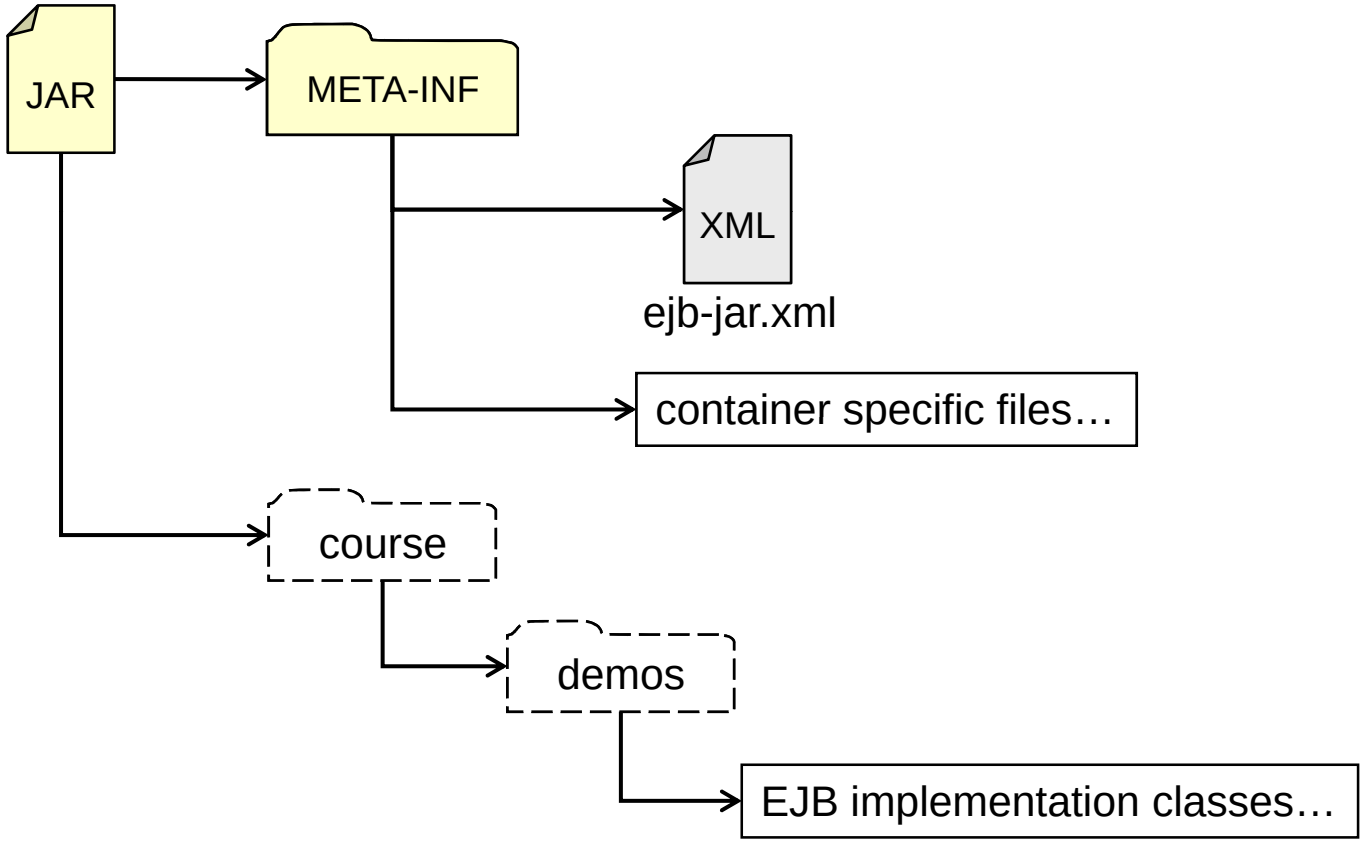
© Garth Gilmour 2008

Other Types of Deployment

- JEE containers deploy modules
  - Modules can be WAR files or JAR's containing EJB's
  - The same module can be deployed several times
    - As long as it is placed in different contexts
- A JEE application is made up of several modules
  - A WAR archive, one or more JAR archives containing Enterprise JavaBeans and any additional libraries
- JEE applications are deployed as an EAR archive
  - The JAR file must have the enterprise archive extension
  - The file contains a deployment descriptor called 'application.xml'

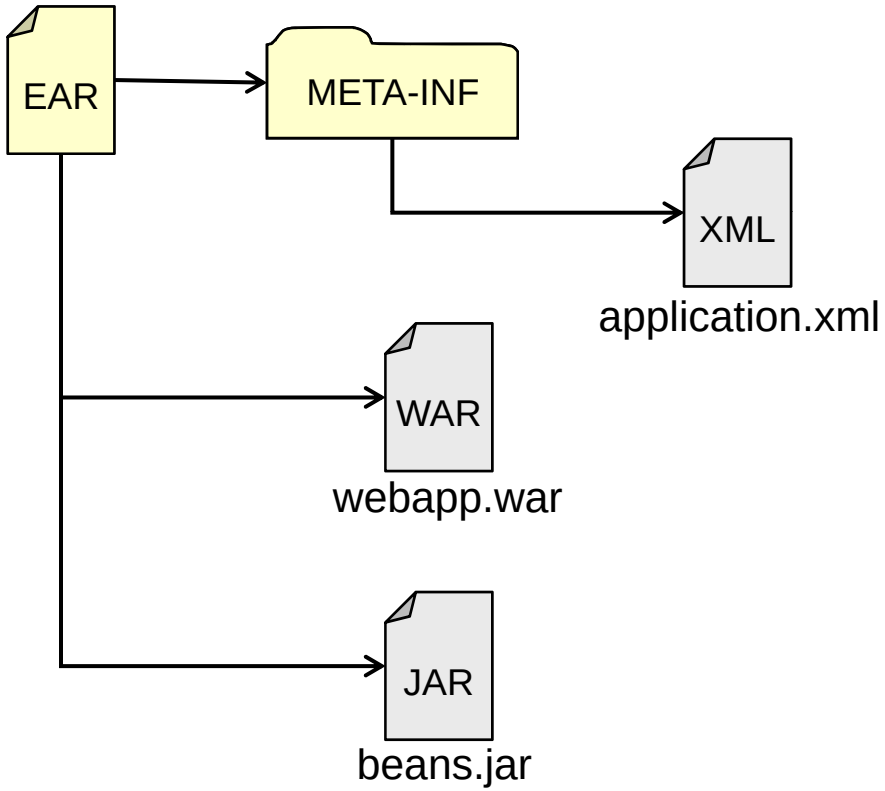
© Garth Gilmour 2008

EJB Archives



© Garth Gilmour 2008

EAR Archives



© Garth Gilmour 2008

Writing Servlets

The Java Servlet API

© Garth Gilmour 2008
garth@ggilmour.com

The Role Of Servlets

- A Servlet is a Java object which:
  1. Receives an HTTP request
  2. Extracts information from the request
     - From parameters, headers and cookies
  3. Applies business logic
     - By talking to a another component
  4. Generates an HTTP response
     - Containing a new HTML or XML document
     - Cookies can be set to identify the browser

© Garth Gilmour 2008

The Servlet Standard And HTTP

- The Servlet standard is a wrapper around HTTP
  - It enables you to process a request and generate a response
- An HTTP request is generated by the browser
  - It contains the request line, headers and optionally a body
  - There are seven different types of request
    - But only GET and POST are commonly used
- An HTTP response is generated by the server
  - It contains the status line, headers and an optional body
  - The status line contains a three digit response code
    - 1=info, 2=success, 3=redirection, 4=client error, 5=server error
  - The body is processed according to the ‘content-type’ header

© Garth Gilmour 2008

```
GET /test HTTP/1.1
Accept: */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
Host: megacorp.com
Connection: Keep-Alive
```

```
POST /test HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
        application/vnd.ms-excel,
        application/vnd.ms-powerpoint,
        application/msword, */*
Accept-Language: en-gb
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705)
Host: megacorp.com
Content-Length: 118
Connection: Keep-Alive
Cache-Control: no-cache

name=Dave&mstatus=S&age=U50&interests=M&interests=B&interests=P
        &comments=This+is+text+from+a+text+area&password=wn1hgb
```

© Garth Gilmour 2008

```
HTTP/1.1 200 OK
Server: SunONE WebServer 6.0
Date: Thu, 29 May 2003 20:40:57 GMT
Set-Cookie: SUN_ID=213.107.102.73:187771054240858;
                        EXPIRES=Wednesday, 31-Dec-2025 23:59:59 GMT;
                        DOMAIN=.sun.com; PATH=/
Content-type: text/html
Etag: "e1da08bc-1c-0-2bf6"
Last-modified: Thu, 29 May 2003 00:13:47 GMT
Content-length: 11254
Accept-ranges: bytes
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head><title> Sun Microsystems </title>

      BODY OF WEB PAGE OMMITTED

</body>
</html>
```

© Garth Gilmour 2008

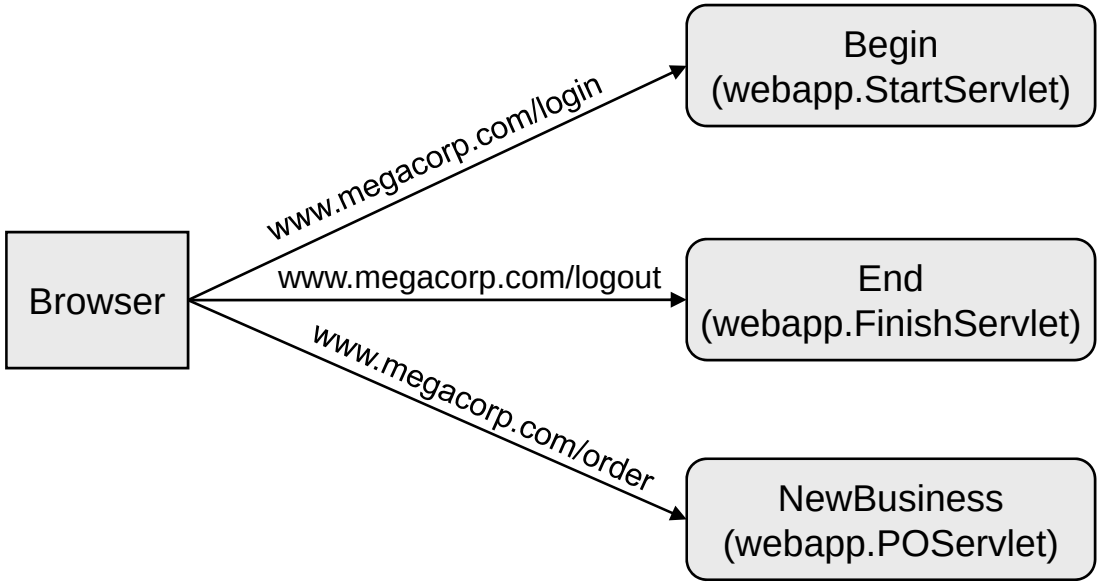
Triggering a Servlet

- A Servlet is mapped to a URL indirectly
  - Each Servlet is given an alias in ‘web.xml’
  - That alias is mapped to one or more URL’s
- Two separate sections are therefore required
  - Many modern IDE’s will write these elements for you
  - If the DTD/Schema is violated deployment will fail
- The container controls a Servlets lifecycle
  - How many instances are created
  - When an instance is created and deleted
  - Which instance receives a particular request
  - How many threads are using an instance

© Garth Gilmour 2008

Configuring a Servlet in ‘web.xml’

URL	Alias	Class Name
/login	Begin	webapp.StartServlet
/logout	End	webapp.FinishServlet
/order	NewBusiness	webapp.POServlet



© Garth Gilmour 2008

Configuring a Servlet in ‘web.xml’

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<!DOCTYPE web-app
        PUBLIC “-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN”
        “http://java.sun.com/j2ee/dtds/web-app_2_2.dtd”>
<web-app>
        <servlet>
            <servlet-name> Begin </servlet-name>
            <servlet-class> webapp.StartingServlet </servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name> Begin </servlet-name>
            <url-pattern> /login </url-pattern>
        </servlet-mapping>
</web-app>
```

© Garth Gilmour 2008

Writing Servlets

- All Servlets inherit from ‘javax.servlet.GenericServlet’
  - Which provides basic lifecycle and request handling methods
- Servlets normally extend ‘javax.servlet.http.HTTPServlet’
  - Which provides HTTP specific request management
- HttpServlet contains stubbed out callback methods
  - One callback for each type of HTTP request
    - doPut, doHead, doDelete, doOptions, doTrace, doPost, doGet
  - To handle one type of request override the matching callback
  - Usually a Servlet only handles GET or POST requests

© Garth Gilmour 2008

Writing Servlets

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyFirstServlet extends HttpServlet {

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

      streamToBrowser.println("<html>");
      streamToBrowser.println("<head><title>Test Page</title></head>");
      streamToBrowser.println("<body><h1>My first servlet</h1></body>");
      streamToBrowser.println("</html>");
    }
}
```

© Garth Gilmour 2008

Loading and Unloading A Servlet

- Servlets contain lifecycle methods
  - Both are inherited from ‘GenericServlet’
  - The server calls ‘init’ before a request is passed
  - The server calls ‘delete’ before garbage collection
- Containers support dynamic reloading
  - When a complete web archive or single class file is overwritten the container reloads the application
    - This requires specialised class loaders that watch the file system
  - The process is non-trivial and historically a source of bugs
    - If in doubt then restart the container itself

© Garth Gilmour 2008

Processing Requests

- Every HTTP callback method takes two parameters
  - Instances of ‘HTTPServletRequest’ and ‘HTTPServletResponse’
- The request object encapsulates the HTTP request
  - You use it to investigate the information you have been sent
- The response object encapsulates an HTTP response
  - You use it to stream content back to the client
- Both of these parameters are interface types
  - Each container supplies its own implementations
  - As always you shouldn’t care about container specific classes

© Garth Gilmour 2008

HTTPServletRequest

- Using the request object you can:
  - Manually process the body of the request
    - Using ‘getInputStream’ or ‘getReader’
  - Extract parameters by name
    - Using ‘getParameter’ or ‘getParameterValues’
    - It does not matter where the parameters were stored
  - Iterate through all the parameters
    - Using ‘getParameterNames’ or ‘getParameterMap’
  - Extract or iterate through HTTP headers
  - Examine cookies, the query string or the URL

© Garth Gilmour 2008

Extracting Parameters

```
String password = request.getParameter("password");
```

```
out.println("<p><h1>Values of all parameters are:</h1></p>");
out.println("<table>");
out.println("<tr><th>Name</th><th>Value(s)</th></tr>");

Enumeration e = request.getParameterNames();
while(e.hasMoreElements()) {
      String name = (String)e.nextElement();
      String [] values = request.getParameterValues(name);
      out.println("<tr><td>" + name + "</td><td>");
      for(int i=0;i<values.length;i++) {
            out.println(values[i] + " ");
      }
}

out.println("</table>");
```

© Garth Gilmour 2008

HttpServletResponse

- Using the response object you can:
  - Set the content type header
    - Which tells the browser how to interpret the request
  - Stream information back to the client
    - Using ‘getOutputStream’ or ‘getWriter’
  - Set the status code in the response
    - Usually to an error code or a redirection
  - Add headers or cookies to the response

© Garth Gilmour 2008

Initialization Parameters

- Context dependant data should never be hard coded
  - This will destroy the portability of your application
  - Examples include usernames, passwords and database URL's
- Initialization parameters can be stored in 'web.xml'
  - Application level parameters are visible to all JSP's and Servlets
    - They are configured using the 'context-param' element
    - They are read via the 'getInitParameter' method of 'ServletContext'
  - Servlet parameters are only read by the corresponding resource
    - They are configured using the 'init-param' element
    - They can only be passed to a JSP if it is configured as a Servlet
    - Read using 'getInitParameter' inherited from 'GenericServlet'

© Garth Gilmour 2008

```
<web-app>
    <context-param>
        <param-name>appParam1</param-name>
        <param-value>This is the text for context level parameter one</param-value>
    </context-param>
    <servlet>
        <servlet-name>InitParamReader</servlet-name>
        <servlet-class>demos.servlets.InitParamReader</servlet-class>
        <init-param>
            <param-name>param1</param-name>
            <param-value>This is the text for Servlet parameter one</param-value>
        </init-param>
        <init-param>
            <param-name>param2</param-name>
            <param-value>This is the text for Servlet parameter two</param-value>
        </init-param>
    </servlet>
    <!– Rest of Deployment Descriptor… -->
```

```
String paramOne = getServletContext().getInitParameter("appParam1");
String paramTwo = getInitParameter("param1");
String paramThree = getInitParameter("param2");
```

© Garth Gilmour 2008

Session Objects

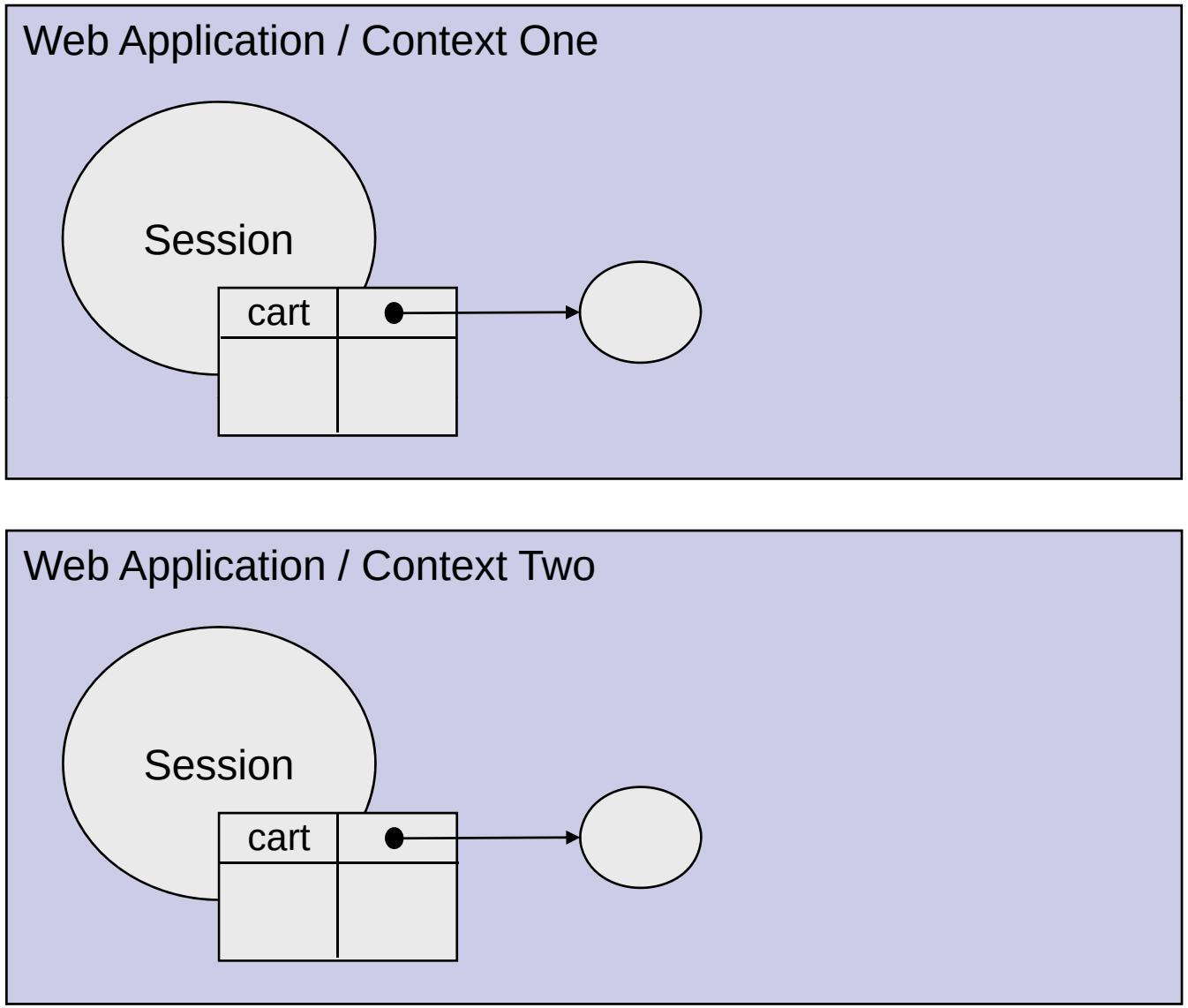
- Servlets are mapped to URL's, not clients
  - Many clients may use the same instance
  - A client may use a different instance each time
- The container is responsible for tracking clients
  - By adding cookies or rewriting URL's
- The container creates an 'HttpSession' for each client
  - You can obtain it via 'HttpServletRequest.getSession'
  - It is pointless to try and cache the session object
- The session timeout period can be set in 'web.xml'
  - Via the 'session-config' and 'session-timeout' elements

© Garth Gilmour 2008

Session Objects

- Each running app has its own session objects
  - Otherwise web apps would interfere with each other
  - Such as by adding items to each others shopping carts
- A container manages a set of contexts
  - A separate context exists for each web app
  - Each web app has it own ‘ServletContext’ object
  - You can deploy the same WAR twice in different contexts
- Session objects don’t have to be persistent
  - They may not survive a server crash or reboot
  - But almost all containers now offer this functionality

© Garth Gilmour 2008



© Garth Gilmour 2008

Session Objects

- By default an ‘HttpSession’ object contains
  - A unique id value (‘getID’)
  - The time of creation (‘getCreationTime’)
  - The last accessed time (‘getLastAccessedTime’)
  - The timeout interval (‘getMaxInactiveInterval’)
- Session objects can store arbitrary information
  - By adding entries to an attributes table
    - Which maps strings to object references
    - A common usage is for ‘ShoppingCart’ objects
- Sessions are only a temporary store
  - They collect data into logical units for processing

© Garth Gilmour 2008

Session Objects

- The methods for adding and removing attributes are:
  - setAttribute(String name, Object value)
  - removeAttribute(String name)
  - getAttribute(String name)
- Be careful when choosing attribute names
  - They must not clash with existing names
  - Including those used by 3rd party frameworks
- Objects can react to being stored in a session
  - By implementing ‘HttpSessionBindingListener’
  - Which contains ‘valueBound’ and ‘valueUnbound’

© Garth Gilmour 2008

Session Objects and Architecture

- Session objects are a security risk
  - The timeout limit should be the shortest possible
    - Even if this could inconvenience users
  - A session can be explicitly destroyed
    - By calling the method ‘Session.invalidate’
    - Use Cases should have an explicit logout step
- Sessions impact the scalability of a design
  - They must be synchronized across servers in a cluster
  - They can consume much needed memory
    - Designs that anticipate extremely heavy usage place data in special database tables instead of using Sessions

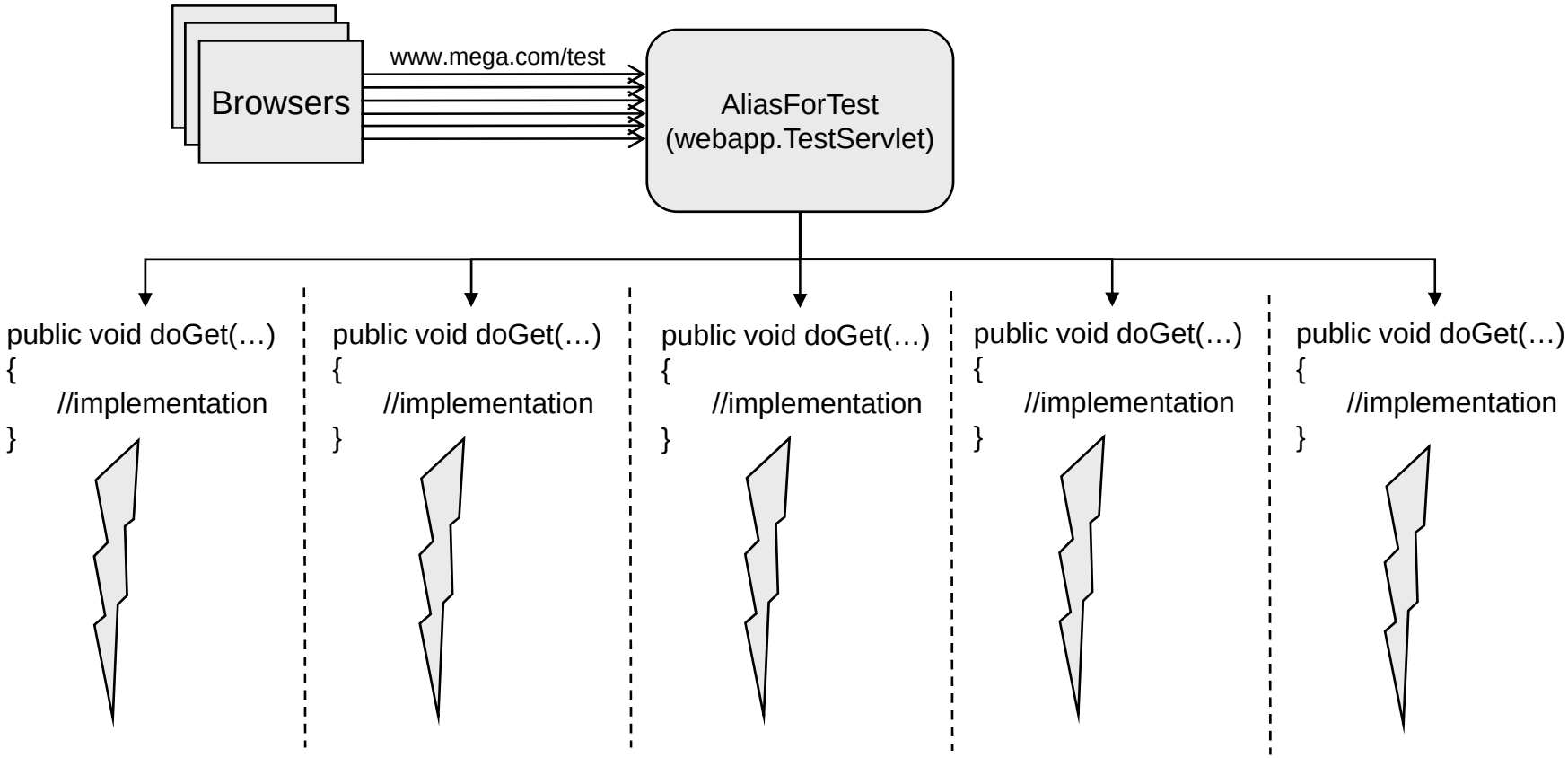
© Garth Gilmour 2008

Threading

- Any number of Servlet objects may be instantiated
  - The container has full control over their lifecycle
- Typically only one will be created
  - The container will thread it as much as necessary
- Each call to ‘do<Type>’ is run in a separate thread
  - If ten clients simultaneously call the same URL the ‘doGet’ method may be running in ten separate threads
  - All the fields of the Servlet are subject to race conditions
    - Which is why you should avoid using them

© Garth Gilmour 2008

Threading



© Garth Gilmour 2008

Threading

- In general Servlets should not have fields
  - Unless the values are final or thread safe
  - Objects used by Servlets need to be thread safe
    - Controller objects need to be stateless or synchronized
    - JDBC and JMS Connection objects are thread safe
    - ‘HTTPSession’ implementations are always thread safe
  - As always local variables are thread safe
    - Because they are allocated on the call stack
- Testing should always be multithreaded
  - Otherwise problems will appear late in development

© Garth Gilmour 2008

SingleThreadModel

- There is a way to enforce single thread behaviour
  - Your Servlet can implement ‘SingleThreadModel’
- ‘SingleThreadModel’ is a marker interface
  - At any time only one thread can run on a Servlet object
  - Useful for wrapping legacy code and testing
- Using this option is a very bad idea
  - It doesn’t eliminate concurrency problems
  - It forces the container to do awkward extra work
  - It is deprecated in the latest version of the Servlet spec

© Garth Gilmour 2008

Request Dispatching

- Servlets should not build a complete page
  - This results in duplicated code and content
- One Servlet can pass a request to another
  - Allowing several to cooperate to build a page
  - This is known as Request Dispatching
- Note dispatching works with JSP's as well
  - Because a JSP is just a special kind of Servlet
- To dispatch a request the Servlet must know
  - The name of the Servlet to be called OR
  - The URL the Servlet is mapped to

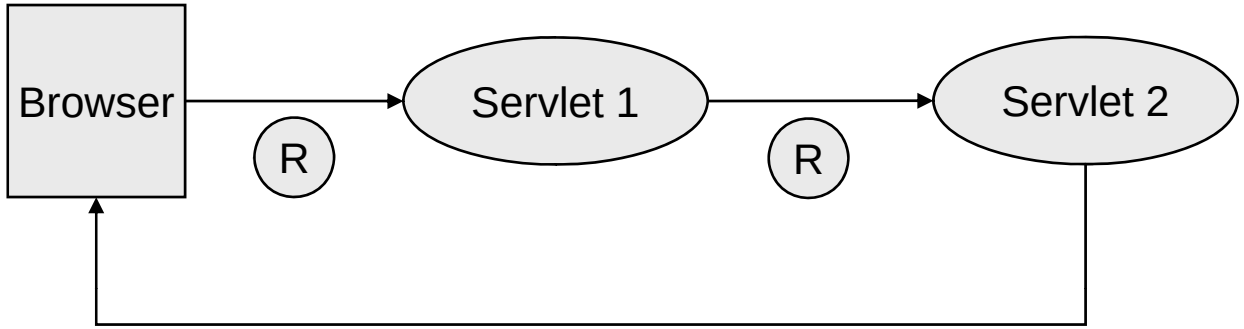
© Garth Gilmour 2008

Kinds of Request Dispatching

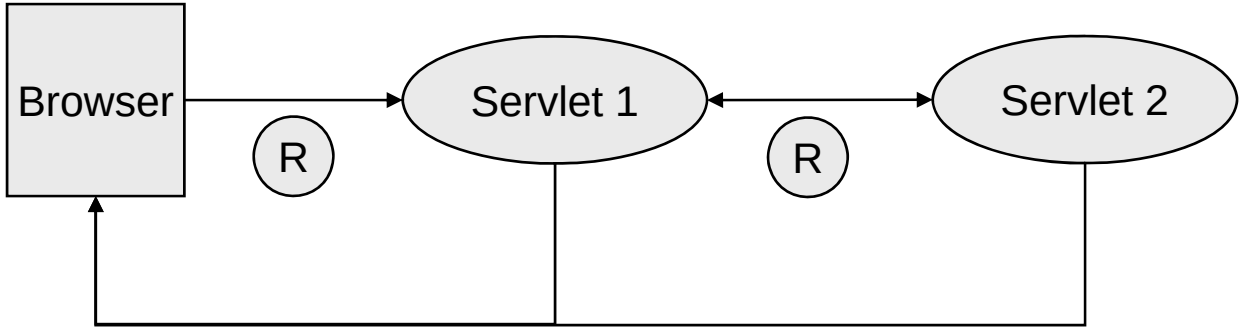
- There are two kinds of Request Dispatching
  - A request can be forwarded to another Servlet
  - The output of another Servlet can be included
- In an include control returns to the calling Servlet
  - The original Servlet can send output to the browser both before and after dispatching the request
  - This is useful for building a complex layout
- In a forward the called Servlet keeps control
  - The original Servlet cannot write to the browser
  - Useful for implementing conditional execution
    - Filters now provide a better way of doing this

© Garth Gilmour 2008

Forwarding a request (only receiving Servlet contributes)



Including a request (all Servlets can contribute)



© Garth Gilmour 2008

Performing Request Dispatching

- Dispatching is always done indirectly
  - One Servlet can never reference another
    - This would interfere with lifecycle management
  - Instead a ‘ServletContext’ object acts as a factory
    - It builds ‘RequestDispatcher’ objects
    - Which contains ‘forward’ and ‘include’ methods
- There are two ways for building dispatchers
  - ‘getRequestDispatcher’ uses a URL
    - Associated with either a Servlet or JSP
  - ‘getNamedDispatcher’ uses a Servlet alias
    - This is preferable as it prevents deep linking

© Garth Gilmour 2008

```
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getNamedDispatcher(“SERVLET_NAME");

response.setContentType("text/html");
PrintWriter out = response.getWriter();

//build the top of the page
out.println("<html>");
out.println("<head>");
out.println("<title>Test Page</title>");
out.println("</head>");
out.println("<body>");
out.println("<h3>Included content begins below</h3>");

//include output of another Servlet
rd.include(request,response);

//build the bottom of the page
out.println("<h3>Included content ends above</h3>");
out.println("</body>");
out.println("</html>");
```

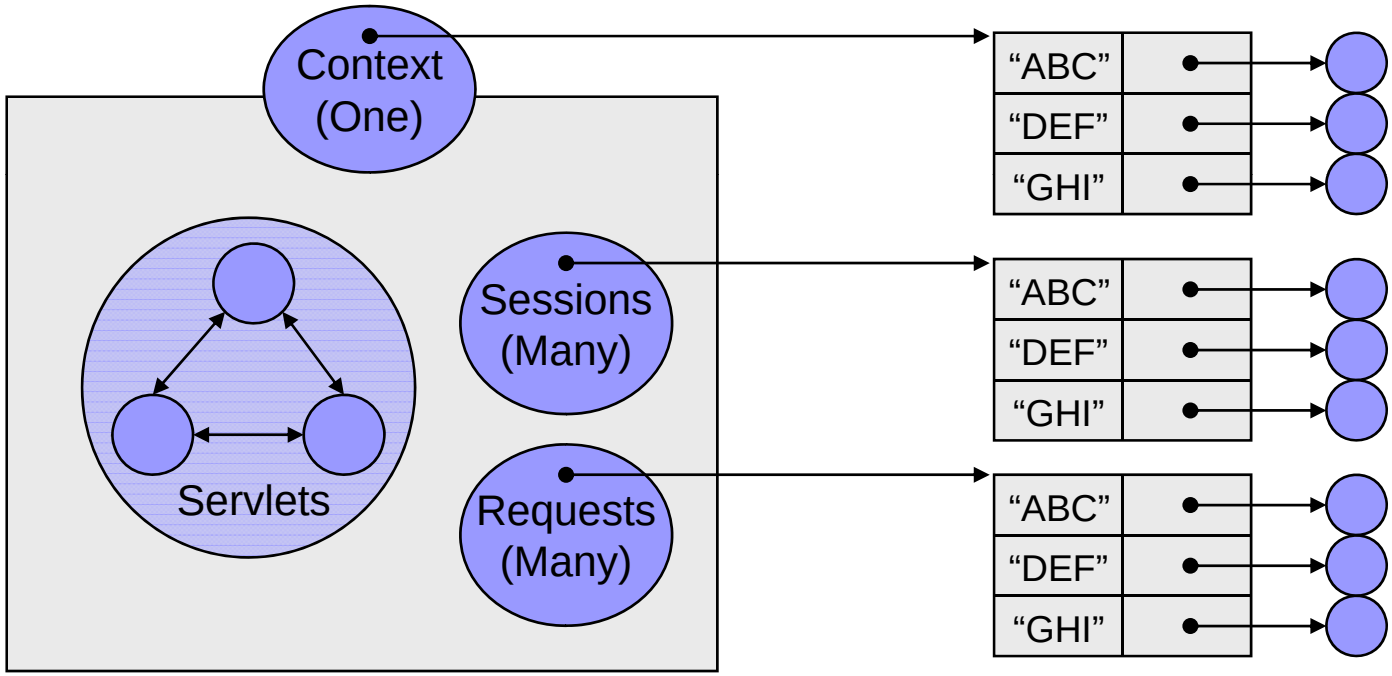
© Garth Gilmour 2008

Using Attributes

- We have covered adding attributes to the HTTPSession
  - To save information between calls from a client
- The request and context objects also support attributes
  - The same method calls are used in each case
  - Attributes are added to the request during dispatching
    - To allow the caller to configure the Servlet it is calling
  - Attributes are added to the context for administration
    - To change how the application behaves while it is running
    - Any Servlet can view attributes set in the context object

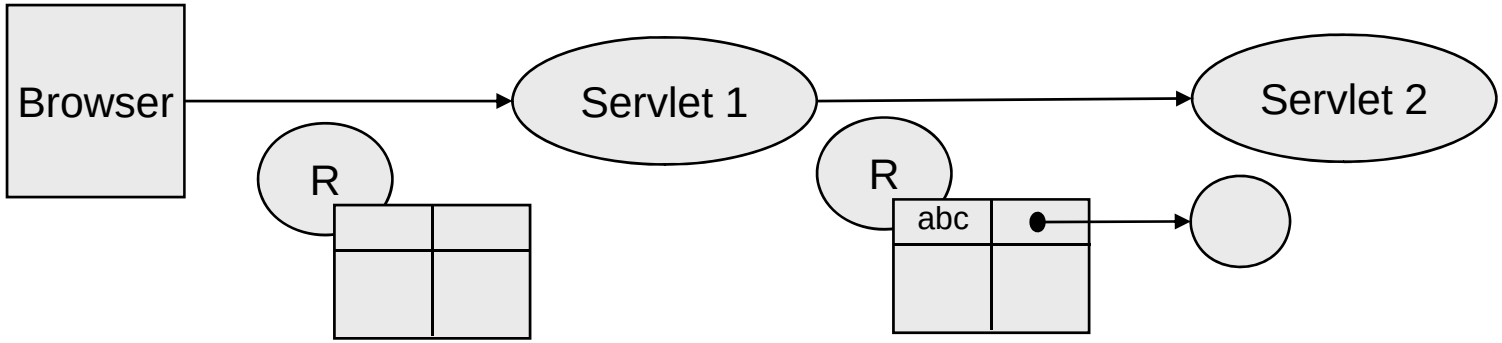
© Garth Gilmour 2008

Attributes Inside Web Applications

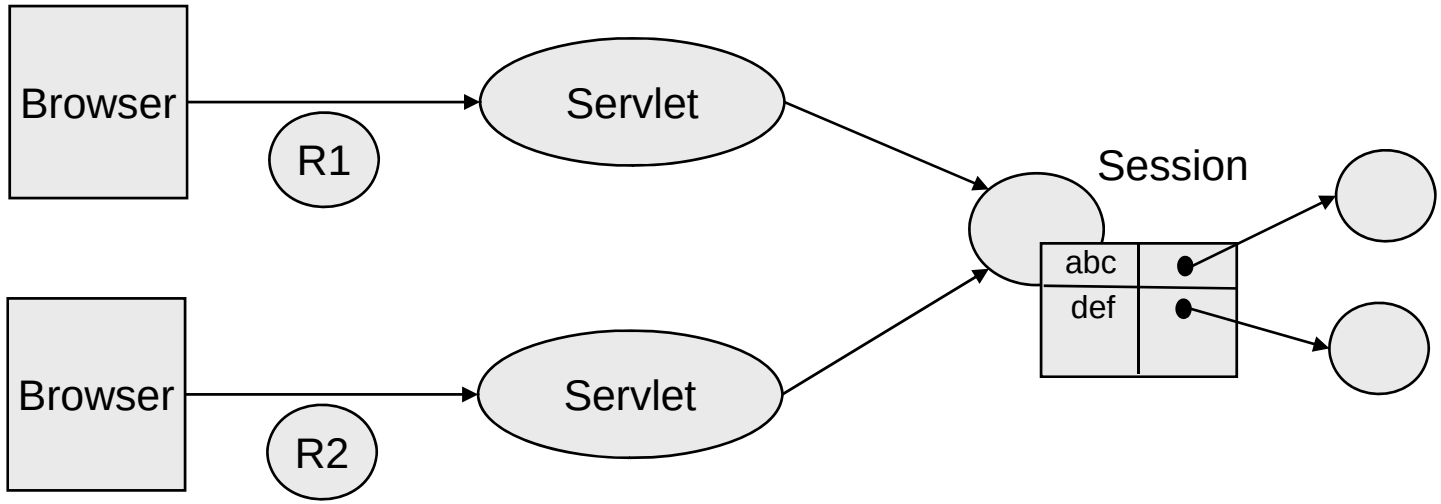


© Garth Gilmour 2008

Adding Information During Request Dispatching



Adding Information Between Multiple Requests



© Garth Gilmour 2008