SUPINFO International University

Sign of Success

Servlets

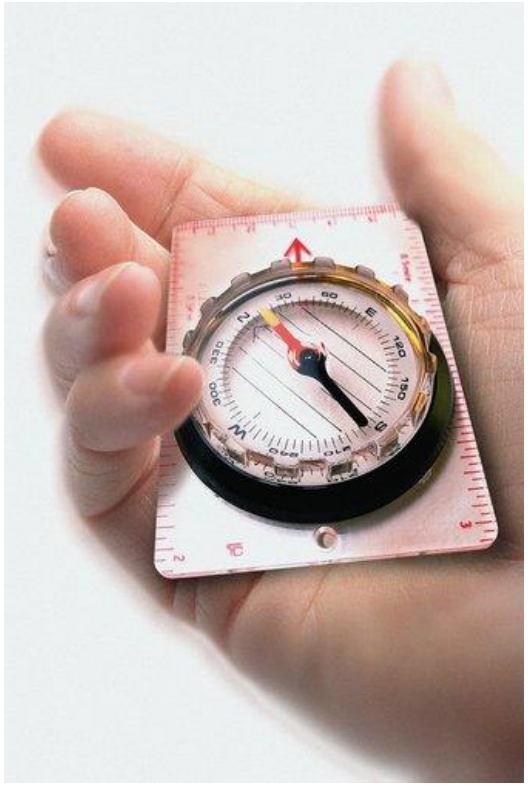
Java Web Applications

Course objectives

By completing this course you will be able to:

- Explain what a servlet is
- Describe the Servlet API
- Create basic and advanced servlets
- Enumerate the main new features in Servlet 3.0

Course plan



- Introduction
- Servlet hierarchy
- Request and response processing
- Deployment descriptor
- The Web Container Model
- What's new in Servlet 3.0?

Servlets

OVERVIEW

Presentation

- Classes executed on the server
- Dynamic request processing
- Dynamic response producing
- Generally used on a Web server

Advantages

- Efficient :
  - JVM’s memory management
  - Only one instance per Servlet
  - One request = One Thread
  - …
- Useful :
  - Cookies
  - Sessions
  - …

Advantages

- Extensible and flexible :
  - Run on numerous platforms and HTTP servers without change
  - Enhanced by many frameworks

- Java Language ! (powerful, reliable with a lot of API)

Drawbacks

- Unsuitable for generating HTML and JavaScript code
  - But JSP does (we'll see it later)
- Needs a Java Runtime Environment on the server
- Needs a special "Servlet Container" on web server
- Low level API
  - But many Frameworks are based on it

Dynamic process

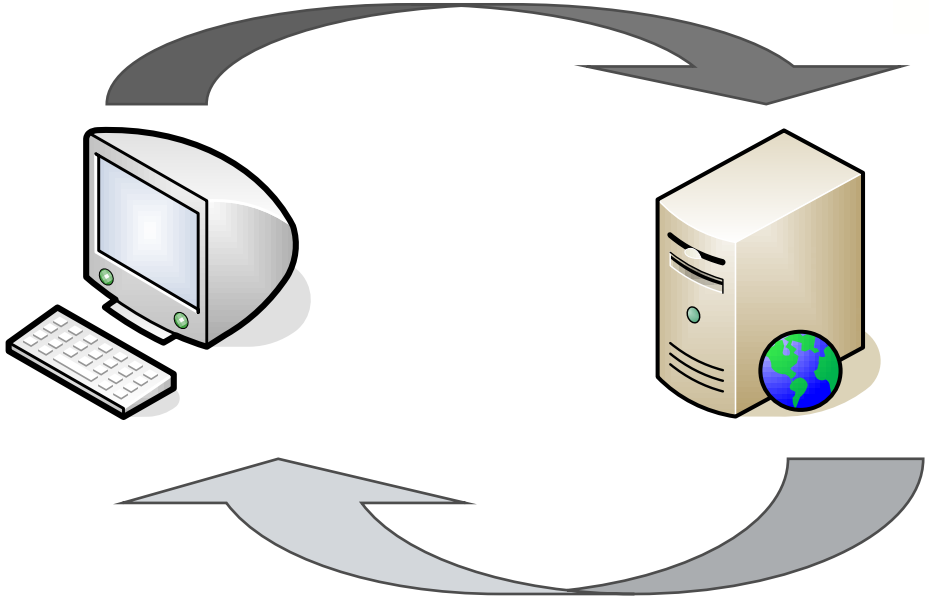
- Basic HTTP request handling
  - Static HTML

≠

- Servlet request handling
  - Dynamic response generated

Basic request

1. Connection and request from client

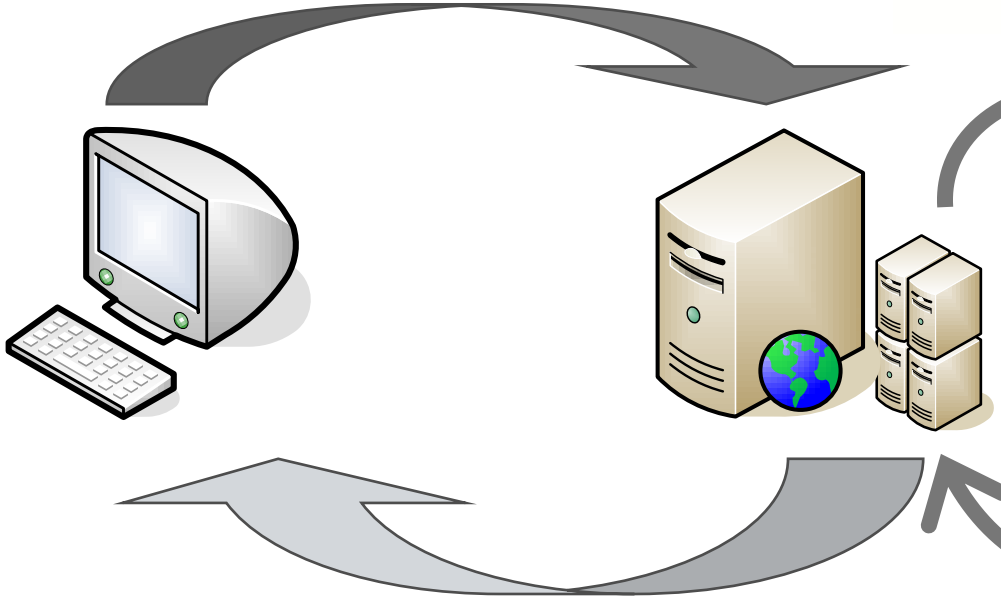


2. Look for the target

3. Page transferred to the client then disconnection

Servlet request – First call

1. Connection and request from client



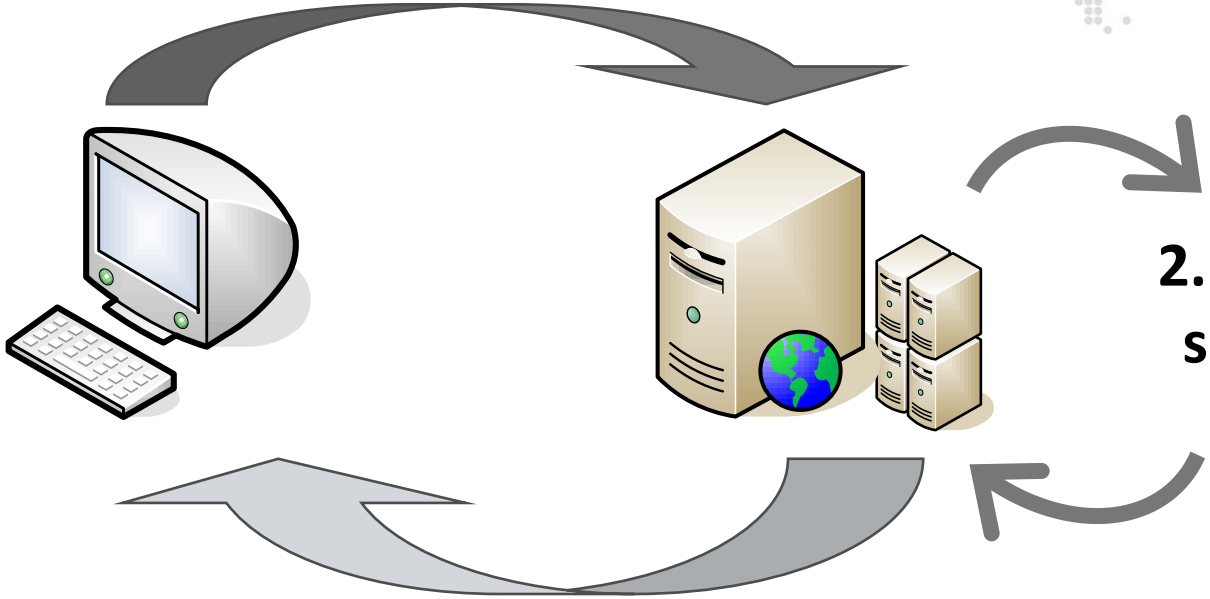
2. Servlet is created and initialized by init() method

3. Servlet executes service() method

4. Response transferred then disconnection

Servlet request – Other calls

1. Connection and request from client



2. Servlet executes service() method

3. Response transferred then disconnection

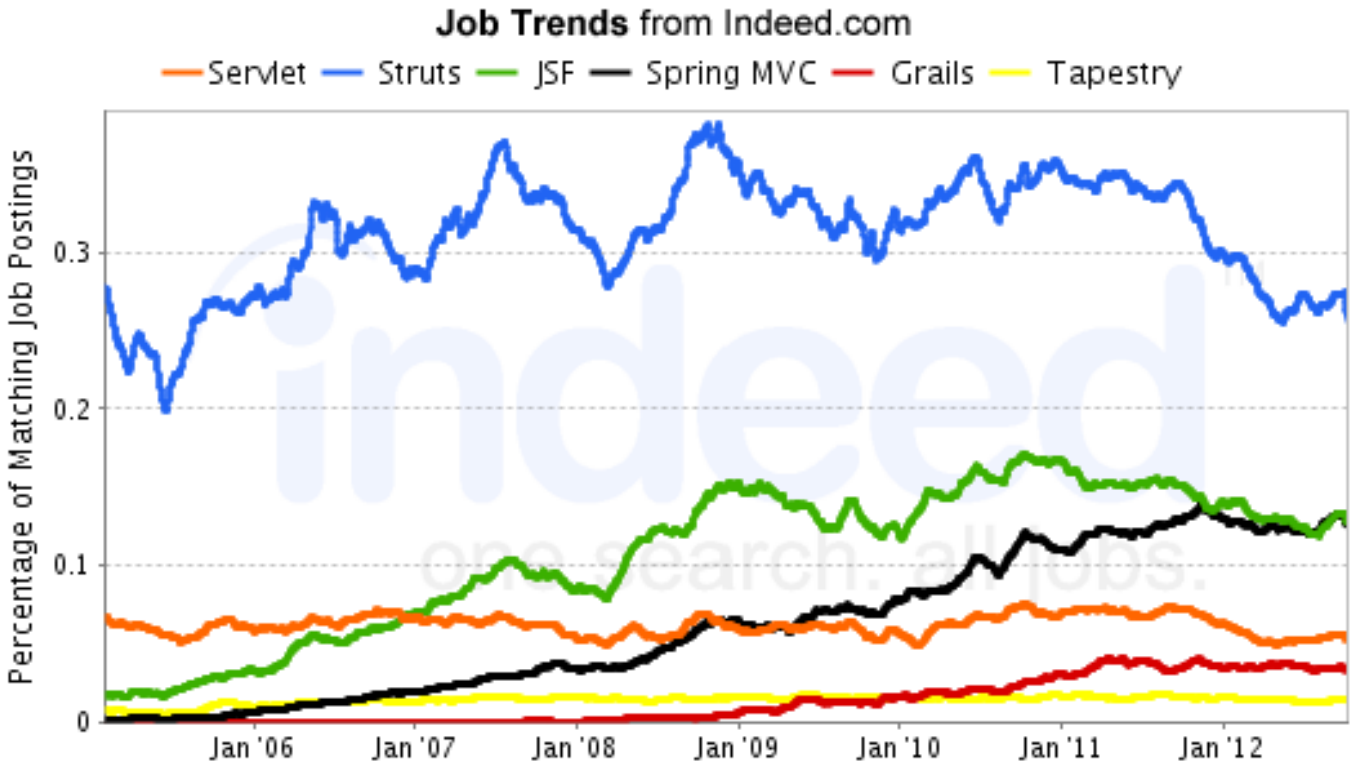
Servlet Container

- Servlet container implements the web component contract of the Java EE architecture
- Also known as a web container
- Examples of Servlet containers are :
  - Tomcat
  - Jetty
  - Oracle iPlanet Web Server

Servlet-Based frameworks

- Few companies used Servlet technology alone…
  - But a lot of companies used Servlet-based Frameworks !
  - A large number exists :
    - Struts
    - JSF
    - Spring MVC
    - Wicket
    - Grails
    - …

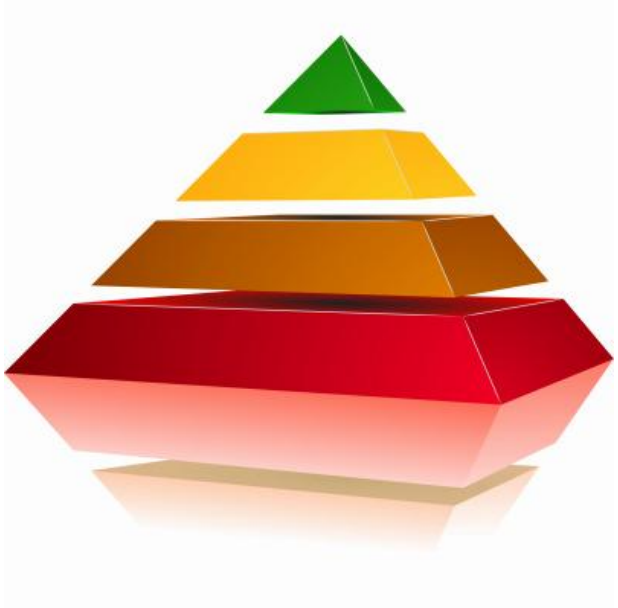
Servlet-Based frameworks



Questions ?

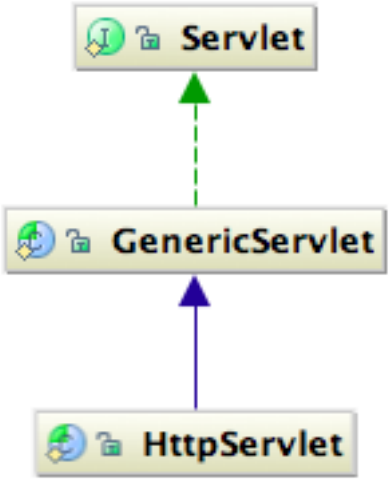
Servlets

SERVLET HIERARCHY



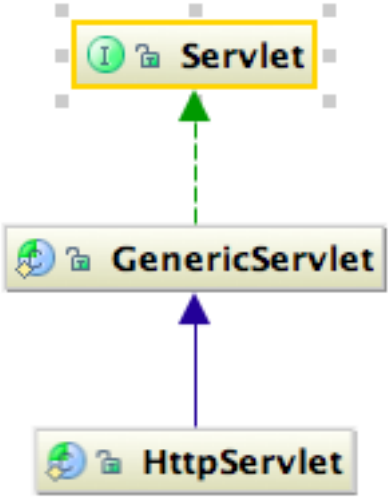
Presentation

- Servlet hierarchy is mainly composed of :
  - *Servlet* interface
  - *GenericServlet* abstract class
  - *HttpServlet* abstract class



Servlet interface

- Defines methods a servlet must implement :
  - How the servlet is initialized?
  - What services does it provide?
  - How is it destroyed?
  - …



Servlet methods overview

Method	Description
void init(ServletConfig sc)	Called by the servlet container to make the servlet active
void service(ServletRequest req, ServletResponse res)	Called by the servlet container to respond to a request
void destroy()	Called by the servlet container to make the servlet out of service
ServletConfig getServletConfig()	Returns a ServletConfig object, which contains initialization and startup parameters for this servlet
String getServletInfo()	Returns information about the servlet, such as author, version, and copyright

Servlet Example 1/2

```
public class MyServlet implements Servlet {

    private ServletConfig config;

    @Override
    public void init(ServletConfig sc) {
        this.config = sc;
    }

    @Override
    public ServletConfig getServletConfig() {
        return this.config;
    }

    ...
```

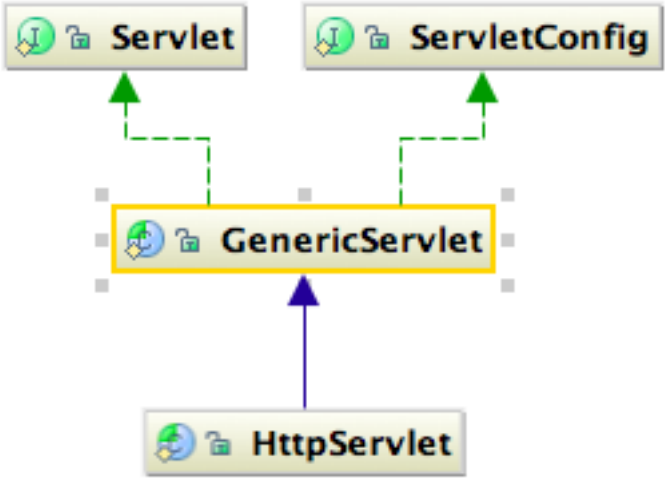
Servlet Example 2/2

```
    ...
    @Override
    public String getServletInfo() {
        return "MyServlet is the best !!";
    }

    @Override
    public void service(ServletRequest req,
                        ServletResponse res) {
        res.getWriter().println("Hello world");
    }

    @Override
    public void destroy() { /* ... */ }
}
```

GenericServlet class

- An independent-protocol Servlet
  - Implements :
    - *Servlet*
    - *ServletConfig*

- Implementations just need to define the **service** method

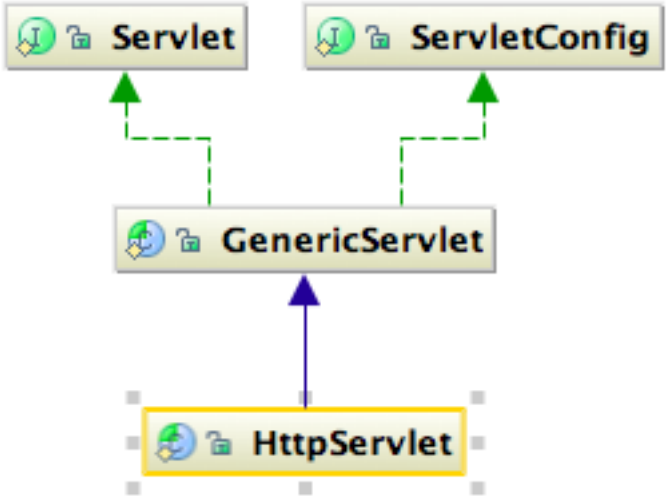


GenericServlet class example

```
public class MyServlet extends GenericServlet {

@Override
public void service(ServletRequest req,
                                ServletResponse res) {
    // Do whatever you want
}
}
```

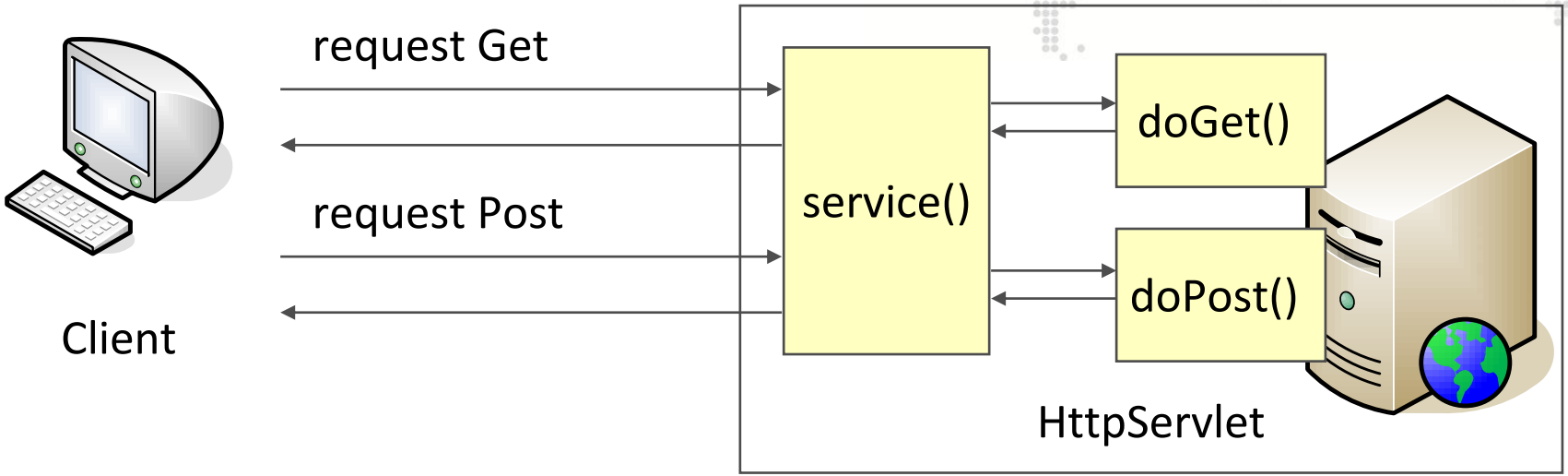
HttpServlet class

- A Servlet suitable for Web sites
- Handles HTTP methods :
  - GET requests with *doGet(…)*
  - POST requests with *doPost(…)*
  - PUT requests with *doPut(…)*
  - Etc…
- You can override one or several of them



HttpServlet class

- How does it work?



Be Careful: *doGet()*, *doPost()* and others, are called by the *HttpServlet* implementation of the service() method.
If it is overridden, the HTTP handler methods will not be called!

Servlets Hierarchy

HttpServlet

- Methods overview:

Method	Description
void service(HttpServletRequest req, HttpServletResponse res)	Receives standard HTTP requests and dispatches them to the doXXX methods defined in this class
void doGet(HttpServletRequest req, HttpServletResponse res)	Handles GET requests
void doPost(HttpServletRequest req, HttpServletResponse res)	Handles POST requests
…	…

HttpServlet example

```
public class MyServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res) {
        // Do whatever you want
    }

    @Override
    public void doPost(HttpServletRequest req,
                       HttpServletResponse res) {
        // Do whatever you want here too ...
    }

}
```

Questions ?

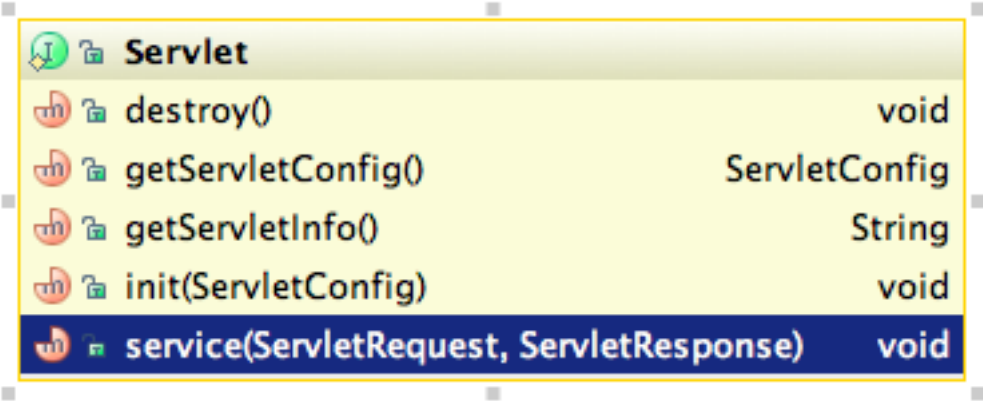
Servlets

REQUEST AND RESPONSE PROCESSING

Introduction

- The Servlet service() method owns:
  - A *ServletRequest* representing a Request
  - A *ServletResponse* representing a Response

- You can use them !



ServletRequest

- Objects defined to provide client request information to a servlet

- Useful to retrieve :
  - Parameters
  - Attributes
  - Server name and port
  - Protocol
  - …

ServletRequest

- Methods overview:

Method	Description
String getParameter(String name) **Map<String, String[]> getParameterMap()**	Return the request parameter(s)
Object getAttribute() **void setAttribute(String name, Object value)**	Retrieves/Stores an attribute from/to the request
boolean isSecure()	Indicates if the request was made using a secure channel such as HTTPS

ServletRequest

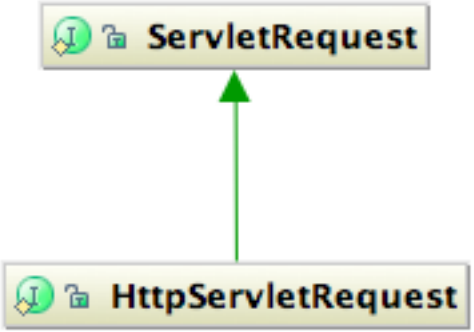
- Methods overview:

Method	Description
String getServerName() **int getServerPort()**	Get the server name and port
String getRemoteAddr() **String getRemoteHost()** **int getRemotePort()**	Get the address, the hostname and the port of the client

HttpServletRequest

- Dedicated to HTTP Requests

- Useful to retrieve :
  - The session associated to the request
  - The headers of the request
  - Cookies
  - …



HttpServletRequest

- Methods overview:

Method	Description
HttpSession getSession() **HttpSession getSession(boolean create)**	Returns the session associated to the request
String getHeader(String name)	Returns the value of the specified header
Cookie[] getCookies()	Get the cookies sent with the request

ServletResponse

- Objects defined to assist a servlet in sending a response to the client

- Useful to retrieve :
  - The output stream of the response
  - A PrintWriter
  - …

ServletResponse

- Methods overview:

Method	Description
ServletOutputStream getOutputStream()	Returns the output stream of the servlet. Useful for binary data.
PrintWriter getWriter()	Returns a writer, useful to send textual response to the client
void setContentType(String content)	Defines the mime type of the response content, like "*text/html*"

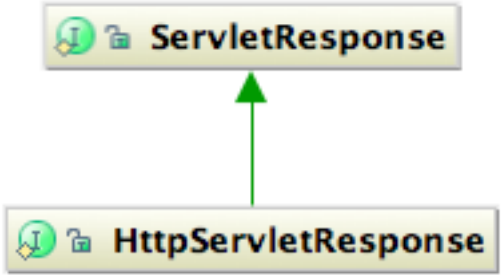
Be Careful: You can't use getOutputStream() and getWriter() together.

Calling both in a servlet gets an **IllegalStateException**.

HttpServletResponse

- Dedicated to HTTP Responses

- Useful to :
  - Add cookies
  - Redirect the client
  - …



HttpServletResponse

- Methods overview:

Method	Description
void addCookie(Cookie c)	Add a cookie to the response
void sendRedirect(String location)	Redirect the client to the specified location

PrintWriter Example

```
public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse resp) {

        String name = req.getParameter("name");

        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>My Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello " + name + " !</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Questions ?

Servlets

DEPLOYMENT DESCRIPTOR

I've got my servlet, and now?

Introduction

- A Servlet application has always (or almost) a deployment descriptor :
  - /WEB-INF/web.xml
- It defines :
  - The servlets classes
  - The servlets mappings
  - The filters classes
  - The filters mappings
  - The welcome files
  - The application resources
  - And some other stuffs…

Deployment descriptor

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns=...>

    <display-name>MyApplication</display-name>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <!-- Servlet declarations -->
    <!-- Servlet mappings -->
    <!- Other stuffs -->

</web-app>
```

Servlet declaration and mapping

- When you create a servlet, in order to use it, you must

1. Declare it in a servlet block
   a. Give it a logical name
   b. Give the “Fully Qualified Name” of the servlet

2. Map it to an url-pattern in a servlet-mapping block

Deployment descriptor

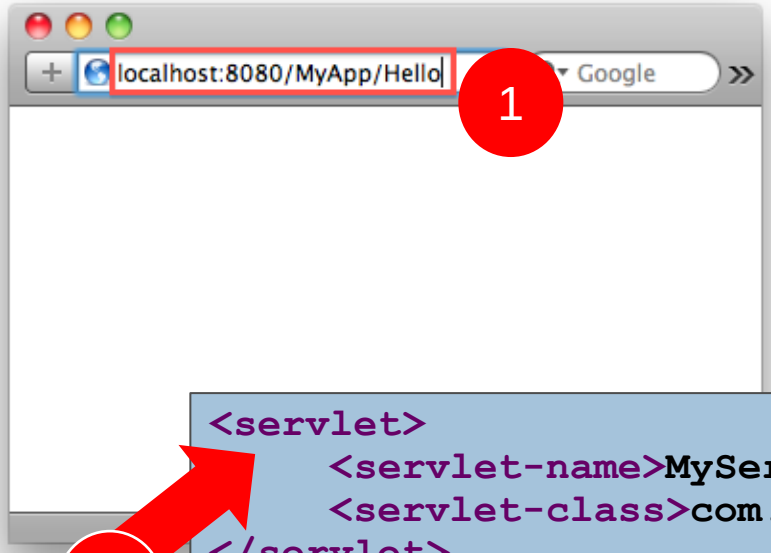
web.xml

```
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.supinfo.app.MyServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/Hello</url-pattern>
</servlet-mapping>
```

⇒ The “servlet-name” in both blocks “servlet” and “servlet-mapping” must be the same!

Deployment descriptor

How it works?




```
<servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>com.supinfo.app.MyServlet</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/Hello</url-pattern>
</servlet-mapping>
```

3

4

2

URL Patterns

Match rule	URL Pattern	URL Pattern form	URLs That Would Match
Exact	/something	Any string with “/” below	/something
Path	/something/*	String beginning with “/” and ending with “/*”	/something/ /something/else /something/index.htm
Extension	*.jsp	String ending with “*.jsp”	/index.jsp /something/index.jsp
Default	/	Only “/”	Any URL without better matching

Questions ?

Exercises (1/3)

- Create a package named:
  - **com.supinfo.supcommerce.servlet**
- Create a HttpServlet inside
  - Name it **InsertSomeProductServlet**
  - Bind it to /**basicInsert** url-pattern
  - Override the service(…) method
    - Create a new **SupProduct** object inside
    - Set name, content and price attributes of the object
    - **Use SupProductDAO.addProduct(SupProduct)** method to stock it in memory

Exercises (2/3)

- Create another HttpServlet
  - Name it **ListProductServlet**
  - Bind it to /**listProduct** url-pattern
- Override the service(…) method
  - Use **SupProductDAO.getAllProducts()** method
    - To retrieve all SupProduct objects added in memory
  - Use a **PrintWriter** obtained from the **ServletResponse** in order to write the response
- Test your two new Servlets ☺

Exercises (3/3)

- Create another HttpServlet
  - Name it **ShowProductServlet**
  - Bind it to /**showProduct** url-pattern
- Override the doGet(…) method
  - Write complete response for the client containing detailed information about a product
    - Who's **id** will be passed as a request parameter
- Test it by enter this kind of URL in your browser:

  http://localhost:8080/SupCommerce/showProduct?id=12

Servlets

THE WEB CONTAINER MODEL

ServletContext, scopes, filters…

Scopes

- Java Web Applications have 3 different scopes
  - Request representing by ServletRequest
  - Session representing by HttpSession
  - Application representing by ServletContext

- We have already seen ServletRequest, we're going to discover the two others…

Scopes

- Each of them can manage attributes
  - Object getAttribute(String name)
  - void setAttribute(String name, Object value)
  - void removeAttribute(String name)

- These functions are useful to forward some values along user navigation, like the user name or the amount of pages visited on a specific day

ServletContext

- Retrieve it with a ServletConfig
  - Or an HttpServlet

    (which implements a ServletConfig, remember ?)
- Allows communication between the servlets and the servlet container
  - Only one instance per Web Application per JVM
- Contains useful data as context path, Application Scope attributes, …

ServletContext

- Add an attribute to the ServletContext
  - This attribute will be accessible in whole application

```
// ...
getServletContext().setAttribute("nbOfConnection”,0);
// ...
```

ServletContext

- Retrieve an attribute from the ServletContext and update it:

```
// ...
ServletContext sc = getServletContext();
Integer nbOfConnection =
       (Integer) sc.getAttribute("nbOfConnection");
sc.setAttribute("nbOfConnection", ++nbOfConnection);
// ...
```

ServletContext

- Remove an attribute from the ServletContext:
  - Trying to retrieve an unknown or unset value returns **null**

```
// ...
getServletContext().removeAttribute("nbOfConnection");
// ...
```

Session

- Interface: HttpSession
- Retrieve it with an HttpServletRequest
  - *request.getSession();*

- Useful methods:
  - *setAttribute(String name, Object value)*
  - *getAttribute(String name)*
  - *removeAttribute(String name)*

Session

- Retrieve a HttpSession and add an attribute to it :
  - This attribute will be accessible in whole user session

```
// ...
HttpSession session = request.getSession();
Car myCar = new Car();
session.setAttribute("car", myCar);
// ...
```

Session

- Retrieve an attribute to it:

```
// ...
Car aCar = (Car) session.getAttribute("car");
// ...
```

Session

- Remove an attribute to it:
  - Trying to retrieve an unknown or unset value returns **null**

```
// ...
session.removeAttribute("car");
// ...
```

Chaining

- From a servlet you can:
  - include the content of another resource inside the response
  - forward a request from the servlet to another resource

Chaining

- You must use a **RequestDispatcher**
  - Obtained with the request

```
// Get the dispatcher
RequestDispatcher rd =
    request.getRequestDispatcher("/somewhereElse");
```

Chaining

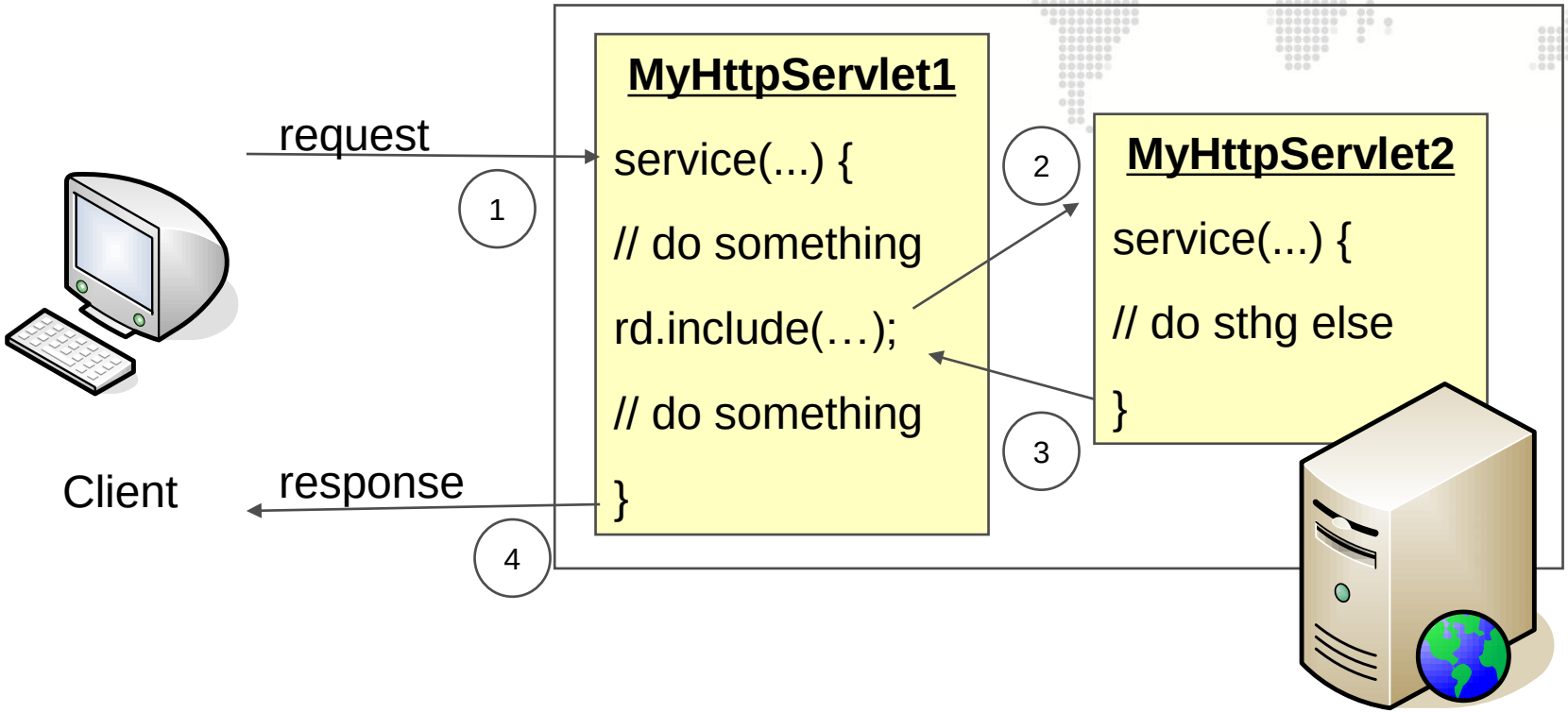
- Then you can include another servlet:

```
rd.include(request, response);
out.println("This print statement will be executed");
```

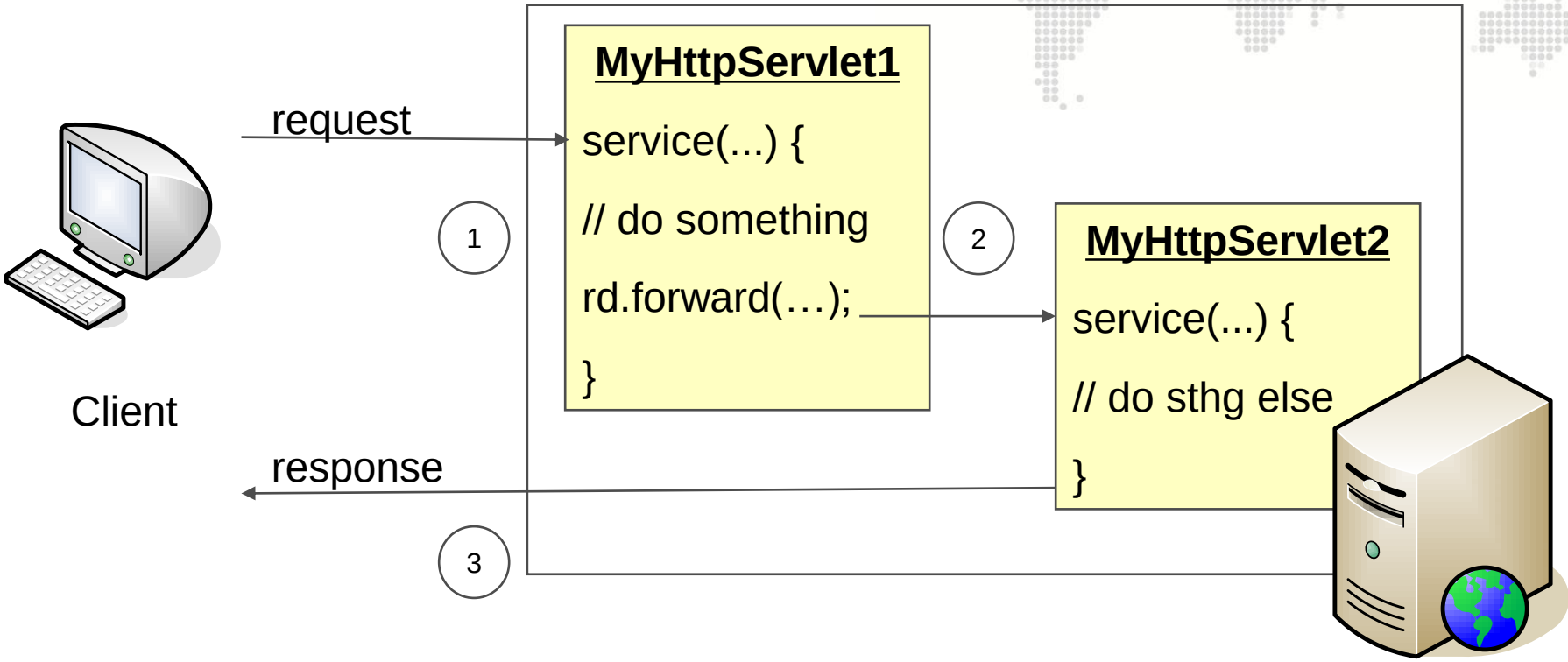
- Or forward the request to another servlet:

```
rd.forward(request, response);
out.println("This print statement will not be executed");
```

Chaining – Include



Chaining – Forward



Chaining

- Use request attributes to pass information between servlets
  - Servlet 1:

```
// Other stuff
Car aCar = new Car("Renault", "Clio");
request.setAttribute("car", aCar);
RequestDispatcher rd =
   request.getRequestDispatcher("/Servlet2");
rd.forward(request, response);
```

Chaining

- Retrieve attribute in another servlet:
  - Servlet 2 mapped to **/Servlet2**

```
// Other stuff
Car aCar = (Car) request.getAttribute("car");
out.println("My car is a " + car.getBrand());
// Outputs "Renault"
```

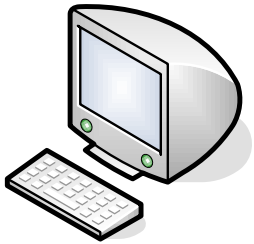
Filter

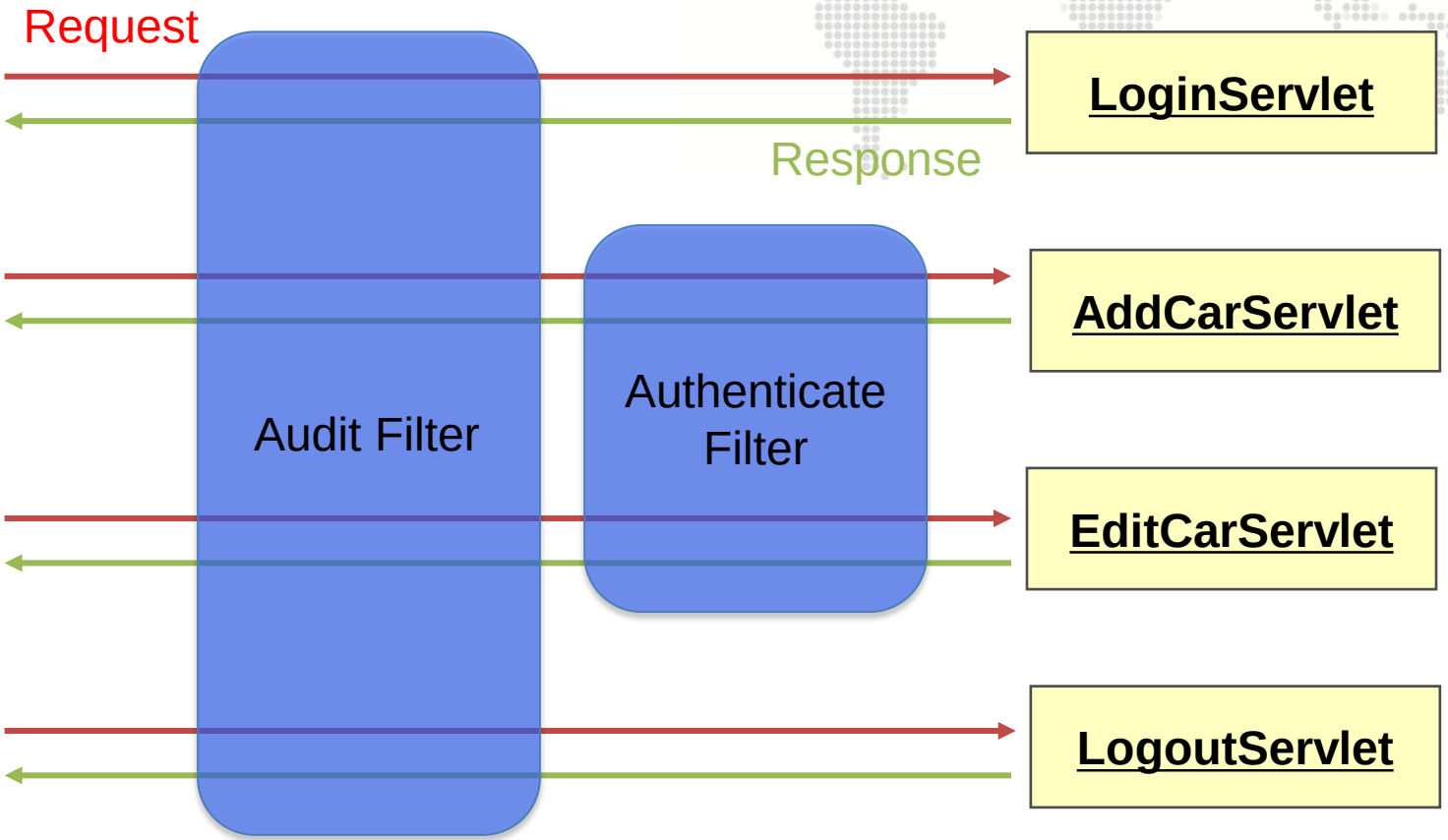
- Component which dynamically intercepts requests & responses to use or transform them
- Encapsulate recurring tasks in reusable units

- Example of functions filters can have :
  - Authentication - Blocking requests based on user identity
  - Logging and auditing - Tracking users of a web application.
  - Data compression - Making downloads smaller.

The web container model

Filter





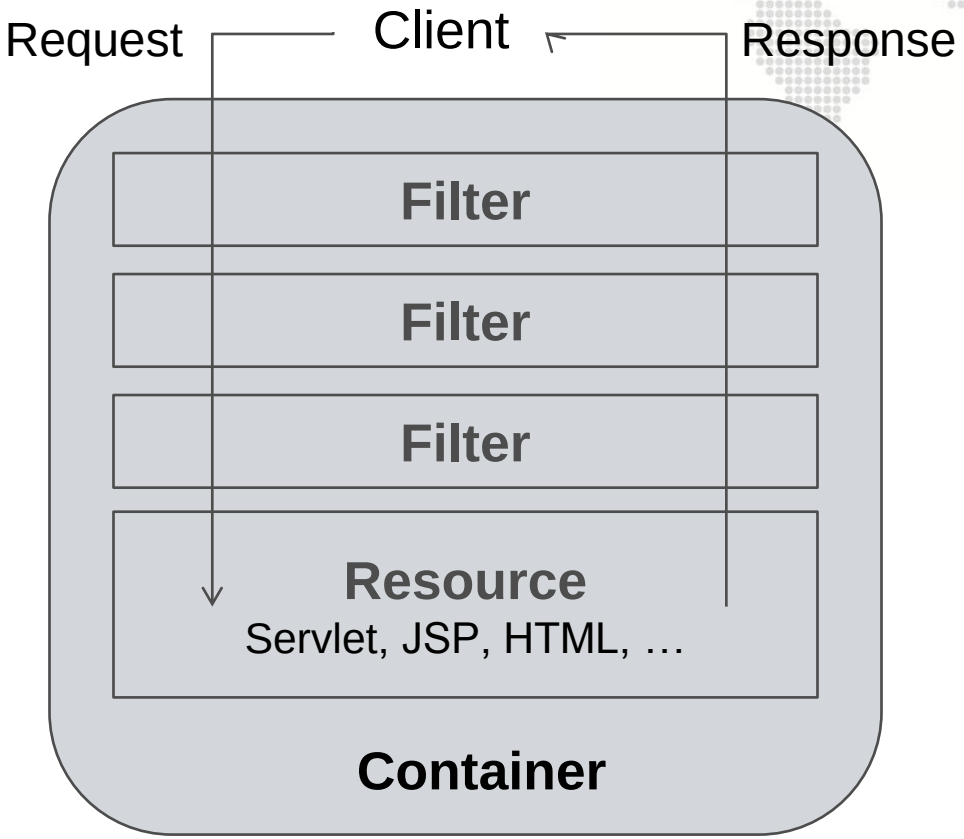
Filter

- Create a class implementing the **javax.servlet.Filter** interface
- Define the methods :
  - init(FilterConfig)
  - doFilter(ServletResquest, ServletResponse, FilterChain)
  - destroy()

Filter

- In the doFilter method
  - Use the **doFilter(…)** of the FilterChain to call the next element in the chain
  - Instructions before this statement will be executed before the next element
  - Instructions after will be executed after the next element
  - Don’t call doFilter(…) method of the FilterChain to break the chain

Filter



Filter example

```
public final class HitCounterFilter implements Filter {
    //...
    public void doFilter(ServletRequest request,
           ServletResponse response, FilterChain chain) {

       ServletContext sc = filterConfig.getServletContext();
       Counter counter =
              (Counter) sc.getAttribute("hitCounter");

        System.out.println("The number of hits is: "
                                    + counter.incCounter());
       chain.doFilter(request, response);
    }
    //...
}
```

Filter declaration

- Declare your filter in the **web.xml** file:

```
<filter>
    <filter-name>MyHitCounterFilter</filter-name>
    <filter-class>
        com.supinfo.sun.filters.HitCounterFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>MyHitCounterFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Cookies

- Can be placed in an HttpServletResponse
- Can be retrieved in an HttpServletRequest
- Constructor :
  - Cookie(String name, String value)
- Methods :
  - String getName()
  - String getValue()

Cookies example

- Add a cookie to a response:

```
// ...
Cookie myCookie = new Cookie("MySuperCookie",
                             "This is my cookie :)");
response.addCookie(myCookie);
// ...
```

Cookies example

- Retrieve cookies from the request:

```
// ...
Cookie[] list = request.getCookies();
for(Cookie c : list) {
System.out.println("Name: " + c.getName()
                    + " Value: " + c.getValue());
}
// ...
```

Questions ?

Exercises (1/3)

- Create a new HttpServlet
  - Name it **LoginServlet**
  - Bind it to **/login** url-pattern
  - Override the doPost(…) method
    - Retrieve the username passed into the request
    - Add it as session attribute
    - Forward the request to the listing servlet

Exercises (2/3)

- Create a new Filter class
  - Named **AuthenticateFilter**
  - Bind to **/auth/*** url-pattern
  - In the doFilter(…) method
    - Check if the username session attribute exists
    - If it does, just call the next element of the chain
    - If it doesn't redirect the user to the login form

Exercises (3/3)

- Create a file named **login.html** in WebContent folder
  - Define a form inside with a text field named username
  - Define the login servlet url-pattern as action

- Add **/auth** at the beginning of the url-pattern of
  - **InsertSomeProductServlet**
  - Test that your AuthenticateFilter and LoginServlet work !

Servlets

WHAT'S NEW IN SERVLET 3.0 ?

Annotations, Web fragments, …

Introduction

- The 10th December 2009, Sun Microsystems releases the version 6 of Java EE
- It includes a lot of new JSR like:
  - EJB 3.1
  - JPA 2.0
  - … and Servlet 3.0 !
- For more information about JSRs include in Java EE 6:

http://en.wikipedia.org/wiki/Java_EE_version_history

Configuration through annotations

- Annotations are used more and more in Java development :
  - To map classes with tables in database (JPA)
  - To define validation rules (Bean Validation)
  - And now to define Servlets and Filters without declare them into deployment descriptor !

Configuration through annotations

- So why not have seen this from the beginning ?
  - Because Servlet 2.5 is still extremely used
  - And you can still used deployment descriptor with Servlet 3.0

Servet 3.0 main annotations

- **@WebServlet** : Mark a class as a Servlet (must still extends HttpServlet)

- Some attributes of this annotation are :
  - **name** (optional) :
    - The name of the Servlet
  - **urlPatterns** :
    - The URL patterns to which the Servlet applies

Servlet 3.0 main annotations

- **@WebFilter** : Mark a class as a Filter

- Some attributes of this annotation are :
  - **filterName** (optional) :
    - The name of the Filter
  - **servletNames** (optional if urlPatterns) :
    - The names of the servlets to which the Filter applies
  - **urlPatterns** (optional if servletNames) :
    - The URL patterns to which the Filter applies

Examples

```
@WebServlet(urlPatterns="/myservlet")
public class SimpleServlet extends HttpServlet {
  ...
}
```

```
@WebFilter(urlPatterns={"/myfilter","/simplefilter"})
public class SimpleFilter implements Filter {
  ...
}
```

Web Fragments

- New system which allow to partition the deployment descriptor
  - Useful for third party libraries to include a default web configuration
- A Web Fragment, as Deployment Descriptor, is an XML file :
  - Named web-fragment.xml
  - Placed inside the META-INF folder of the library
  - With <web-fragment> as root element

Web Fragments Example

```
<!-- Example of web-fragment.xml -->
<web-fragment>
    <name>MyFragment</name>
    <listener>
        <listener-class>
            com.enterprise.project.module.MyListener
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>MyServletFragment</servlet-name>
        <servlet-class>
            com.enterprise.project.module.MyServlet
        </servlet-class>
    </servlet>
</web-fragment>
```

Questions ?

Exercises

- It's time to be modern!
  - Remove the Deployment Descriptor of your project
  - Update your Servlets and your Filter to use Servlet 3.0 annotations

The end

SUPINFO International University

Sign of Success

Thanks for your attention