SOEN 387

Web-based Enterprise Application Design

Credit to Dr. Chalin

- These notes are based on his originals

Reminders

- Read the material!
- I need your teams today
  - Sit with you teammates in class

HTTP Methods

- How many of them were there?
- Name them.

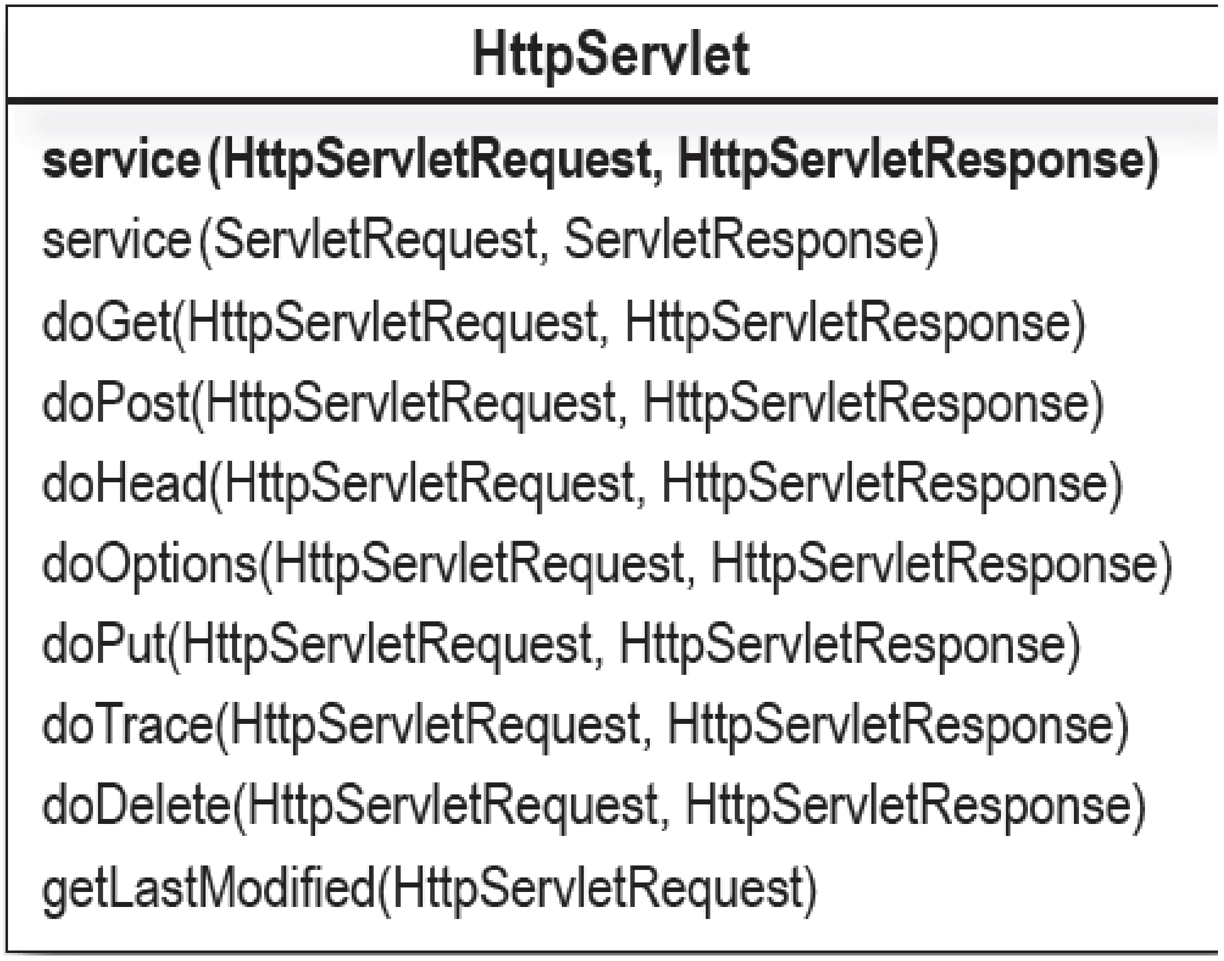
HTTP Methods

- **GET** resource at given URL.
- **POST** – (simplifying) like a GET with extra param.
- HEAD – get header part only.
- *TRACE* – loopback request message.
- **PUT** – put info at given *URL.*
- **DELETE** given *URL.*
- *OPTIONS* – list HTTP methods URL can respond to.
- *CONNECT*.

Servlet Classes

- What do they subclass?
- What interface does this subclassing give?

HttpServlet Interface



HttpServlet

service(HttpServletRequest, HttpServletResponse)
service(ServletRequest, ServletResponse)
doGet(HttpServletRequest, HttpServletResponse)
doPost(HttpServletRequest, HttpServletResponse)
doHead(HttpServletRequest, HttpServletResponse)
doOptions(HttpServletRequest, HttpServletResponse)
doPut(HttpServletRequest, HttpServletResponse)
doTrace(HttpServletRequest, HttpServletResponse)
doDelete(HttpServletRequest, HttpServletResponse)
getLastModified(HttpServletRequest)

Just HTTP?

- Can a servlet subclass something else?

Idempotency

- In mathematics a function is idempotent if returns the same result no matter how many times you apply it. E.g.
  - abs(-3) = abs(abs(-3)) = 3
- An method can be idempotent if repeat calls yield the same (visible) effect and result.

Side Effects

- There can be side effects like?

Which HTTP Methods?

- Which ones should be idempotent?

Which HTTP Methods?

• GET	• idempotent
• POST	• non-idempotent
• HEAD	• idempotent
• PUT	• Idempotent*
• DELETE	• non-idempotent

What about PUT?

What does CRUD stand for?

CRUD as guidance?

- UI Design?
- DB Design?
- Webapp / service design?

CRUD to SQL?

- How do the pieces map to SQL statements?

REST

- Representational State Transfer
  - That’s a mouthful

Simple idea of REST

- Implementing CRUD on resources over HTTP
- Oversimplification to the point of being wrong, but still a useful starting point

Philosophical Tangent

- World currently prizes oversimplification
  - in media
  - In courses

- As Engineers, we must fight this

- In the various golden ages of intellect the other things happened
- Complex thought became prized for its complexity, not its thought
  - Still happens in academia
  - CS – glorification of obfuscation

- As Engineers, we must fight this

CRUD to HTTP Methods?

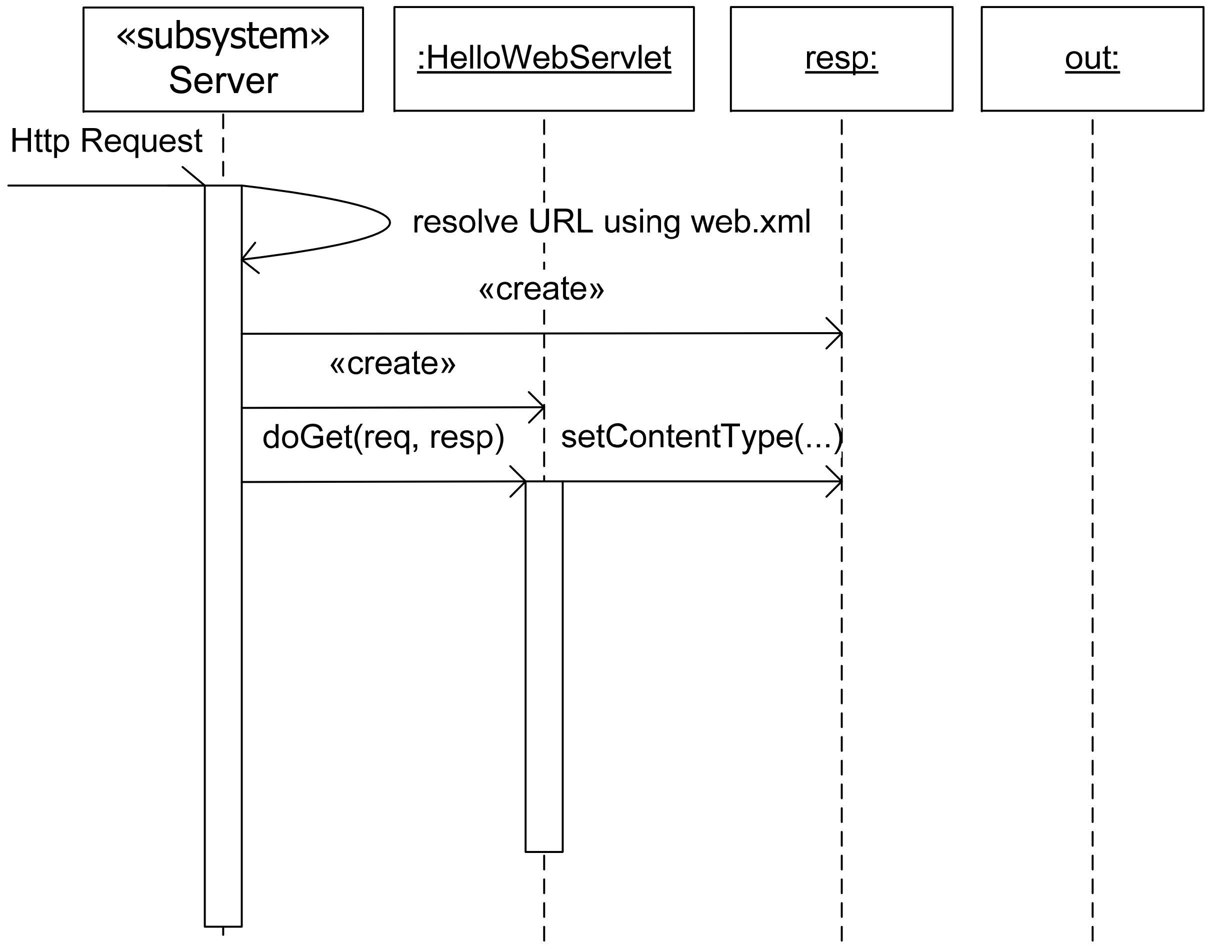
CRUD to HTTP

$$\begin{tikzcd}
\text{• Create} \arrow[ddddr, dash] & \text{• GET} \\
\text{• Read} \arrow[ur, dash] & \text{• POST} \\
\text{• Update} \arrow[ur, dash] \arrow[ddr, dash] & \text{• HEAD} \\
\text{• Delete} \arrow[ddr, dash] & \text{• TRACE} \\
 & \text{• PUT} \\
 & \text{• DELETE} \\
 & \text{• OPTIONS} \\
 & \text{• CONNECT}
\end{tikzcd}$$

More on REST later

- Much more in SOEN487

How Tomcat Processes Requests



What happens at the beginning

- Tomcat gets HTTP Request
- URL determines context
- web.xml from context determines servlet class
  - Simplification…
- If there’s no instance, Tomcat makes it (s.init())
- Tomcat prepares request/response objects

What Happens in the Middle

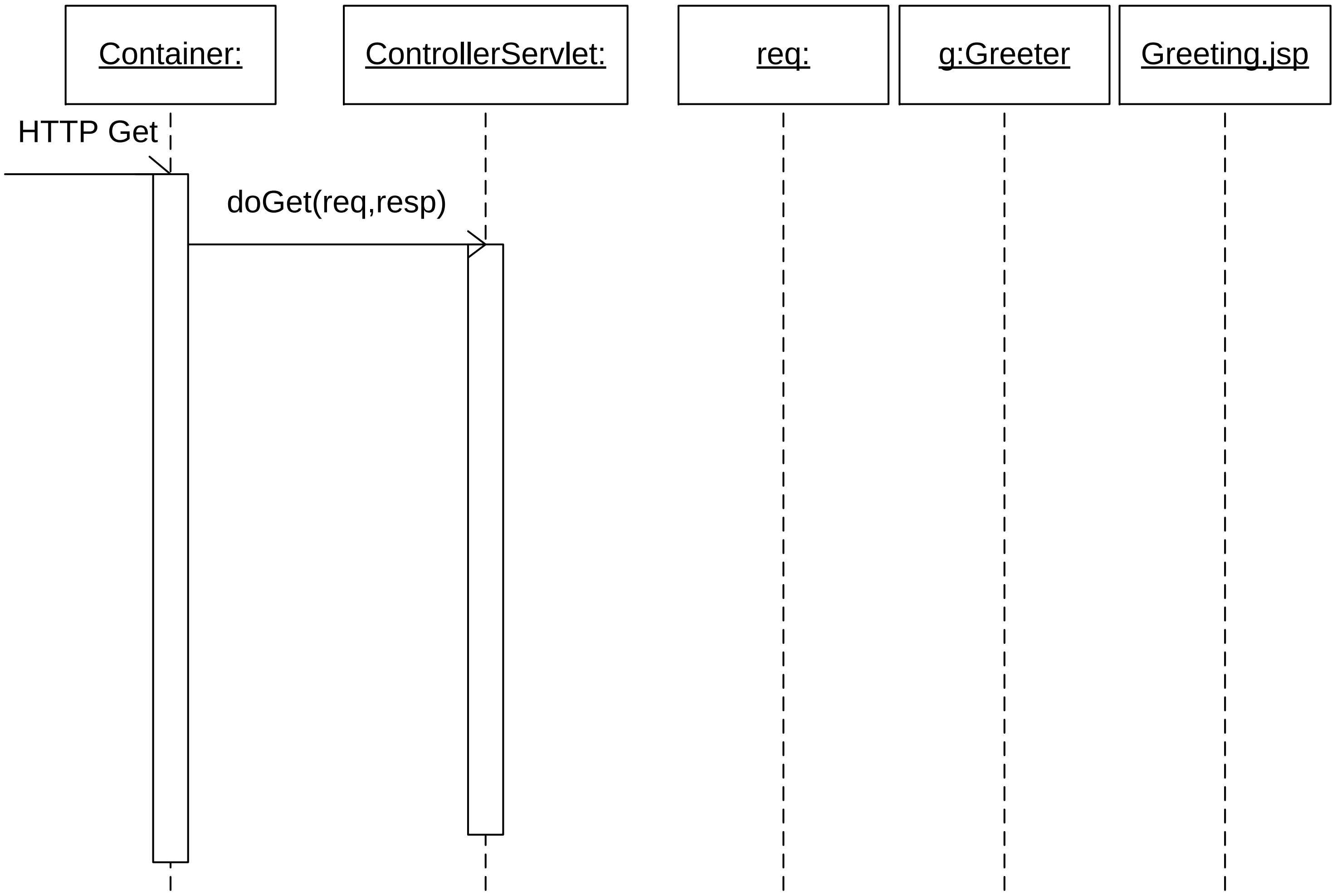
- Tomcat runs a thread on the servlet instance’s service method, passing request and response (s.service(req,resp))
- This method then determines and calls the appropriate servlet method (doGet/doPost(req, resp))

What happens at the End

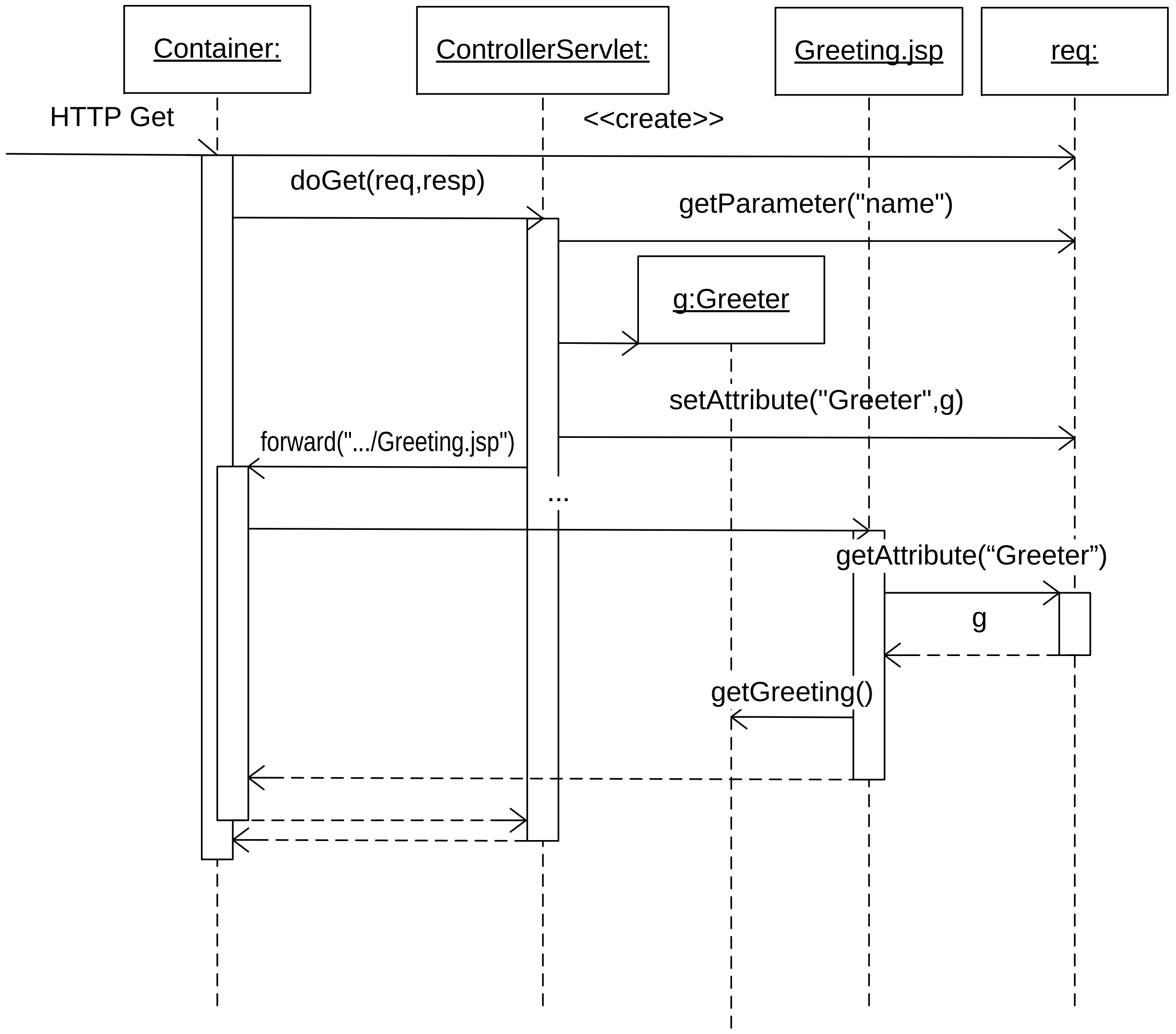
- Some stuff is left for garbage collection
  - Most cleanup is left to the programmer
  - Don’t store the request object in the session/application context!
- When a servlet is shut down call s.destroy()

Hand in diagrams from last week

Chalkboard Solution… what’s happening?



Did we do it right?



Container:
ControllerServlet:
Greeting.jsp
req:
HTTP Get
<<create>>
doGet(req,resp)
getParameter("name")
g:Greeter
setAttribute("Greeter",g)
forward(".../Greeting.jsp")
...
getAttribute("Greeter")
g
getGreeting()