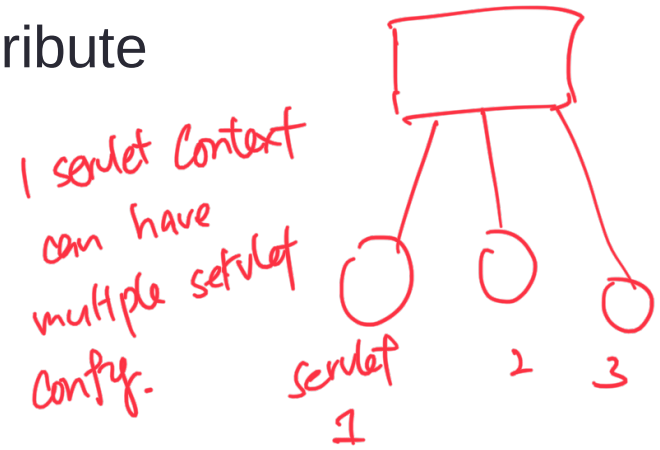
COMP S380F Lecture 2: Servlet

Dr. Keith Lee
School of Science and Technology
Hong Kong Metropolitan University

Overview of this lecture

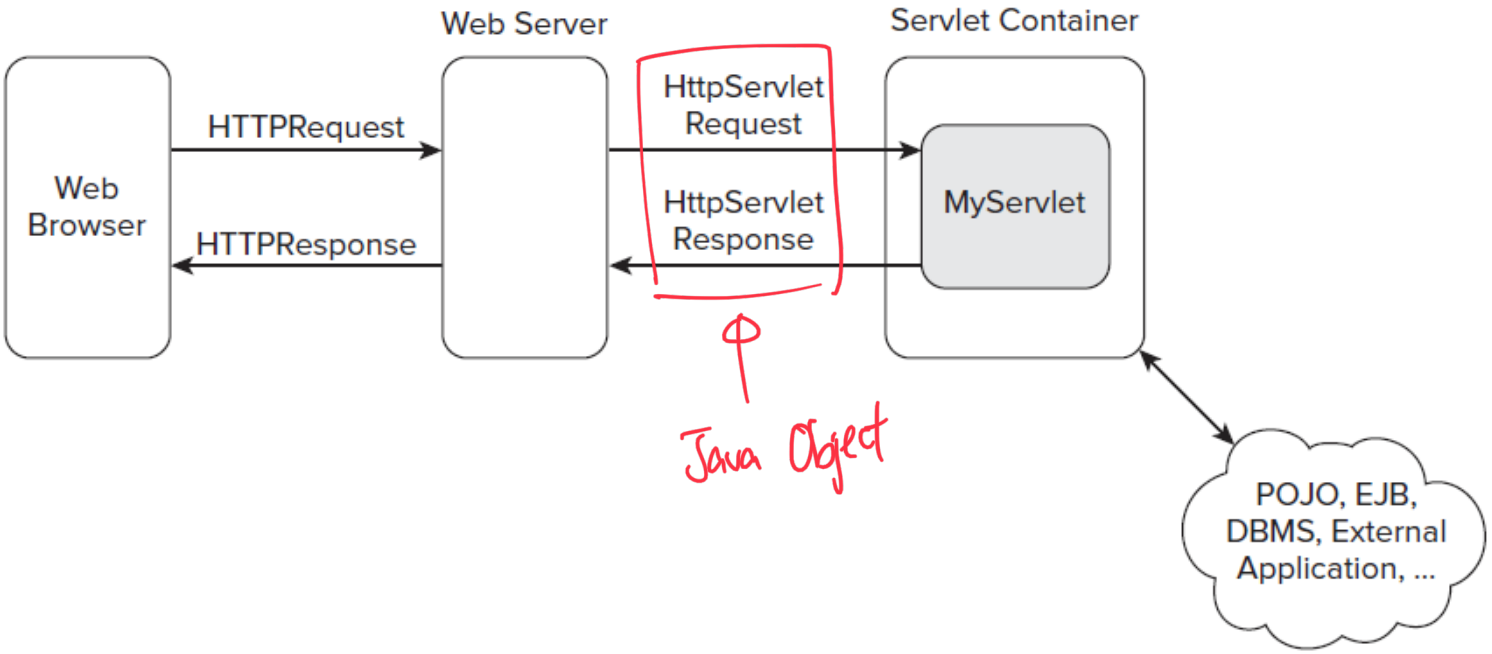
- Java Servlet
- How to deploy Servlets in Web container?
- Servlet Names:
  - In Deployment Descriptor (web.xml)
  - In Annotation @WebServlet
- How Web container handles Servlet's request?
- Servlet's lifecycle
- Request parameter vs. Request attribute
- Request dispatcher
- ServletConfig vs. ServletContext
- Attributes and their Scope
- Attributes vs. Parameters



Java Servlet

static → ① Web server
Servlet { ② App server
③ Web container

- A Servlet is a Java program that runs within a web container (or servlet container).
- Servlets receive and respond to requests from Web clients, usually across HTTP. (can other request)



How to deploy Servlets in a Web Container?

1. Create the standard Web application directory structure
2. Write a Servlet
3. Compile
4. Write a **deployment descriptor (/WEB-INF/web.xml)**
5. Package all up into an archive file and name it **`appname.war`**
6. Deploy the war file to the web container

   E.g., for Apache Tomcat, copy the war file into its **`webapps`** directory
7. The server detects the application and makes it available to the users:

   `http://localhost:8080/appname/` (URL; context root; Base URL)

   `appname` is the **context root** of the web application, i.e., the root directory.

- There are tools developed designed to assist the programmers with the series of tasks involved in writing Web applications.
  - E.g., Apache **NetBeans** IDE, **Eclipse** Web Tools Platform (WTP)

Example of a Servlet

- No main(): The container calls the servlet methods like **doGet()** through service().

HelloServlet.java

```
package hkmu.comps380f;

import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");

    PrintWriter out = response.getWriter();
    Date today = new Date();
    out.println("<!DOCTYPE html><html><body>");
    out.println("<h1>It is now: " + today + "</h1>");
    out.println("</body></html>");
  }
}
```

Output shown in browser:

It is now: Thu Feb 25 15:43:50 GMT+08:00 2021

Servlet Names

- A Servlet can have three different names:

  base URL = server URL + **context root**

  1. Name that the client uses, i.e., URL mapping
     E.g., `http://localhost:8080/Lecture02/greeting`
  2. Name that server uses at deployment time (within the <servlet-name> tag)
     E.g., **`helloServlet`**
  3. Actual class name
     E.g., **`hkmu.comps380f.HelloServlet`** ← Fully Qualified

- All these names are declared and mapped in the **deployment descriptor (web.xml)**
- Mapping servlet names improves the webapp's flexibility and security.

web.xml

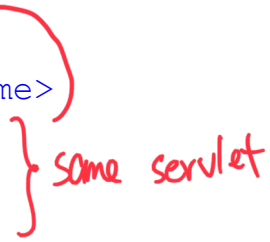
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" ...>
...
  <servlet>
    <servlet-name>helloServlet</servlet-name>
    <servlet-class>hkmu.comps380f.HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>helloServlet</servlet-name>
    <url-pattern>/greeting</url-pattern>
  </servlet-mapping>
...
</web-app>
```

Servlet Names (cont')

- A Servlet can have more than one URL mappings.

web.xml

```
<servlet>
  <servlet-name>helloServlet</servlet-name>
  <servlet-class>hkmu.comps380f.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>helloServlet</servlet-name>
  <url-pattern>/greeting</url-pattern>
  <url-pattern>/salutation</url-pattern>
  <url-pattern>/wazzup</url-pattern>
</servlet-mapping>
```



- The above servlet **`helloServlet`** has the following URL mappings:
  - http://localhost:8080/Lecture02/greeting
  - http://localhost:8080/Lecture02/salutation
  - http://localhost:8080/Lecture02/wazzup

The annotation @WebServlet

- Another way to declare and map the Servlet names is to use the annotation **@WebServlet** on the Servlet class.

HelloServlet2.java

```
…
import javax.servlet.annotation.WebServlet;

@WebServlet(
    name = "anotherHelloServlet",
    urlPatterns = { "/greeting2", "/salutation2", "/wazzup2" }
)
public class HelloServlet2 extends HttpServlet {
    …
}
```

- Now, we only need to declare the servlet name (using **name**) and the URL mappings (using **urlPatterns**), and do not need to modify web.xml.

- We can also declare multiple URL mappings using @WebServlet.

The annotation @WebServlet (cont')

- If we have the deployment parameters for the same Servlet in both annotations and deployment descriptor (web.xml), the values in web.xml will override the corresponding values in the annotations.

- Comparison between the two ways:

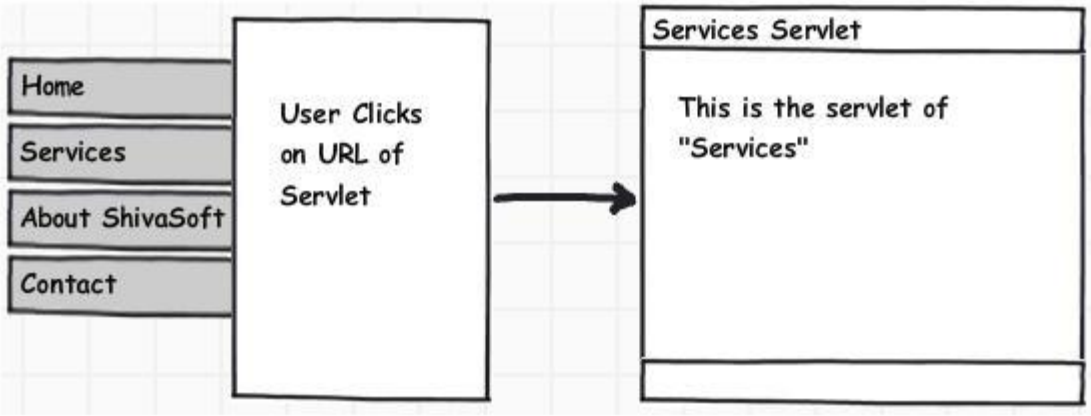
Faster as no need to redeploy

	web.xml	Annotation
Advantage	Changing the deployment parameters does not require recompilation of Servlets.	The syntax is cleaner than XML.
Disadvantage	The XML syntax is more clumsy than Java annotation.	Changing the deployment parameters requires recompilation of Servlets.

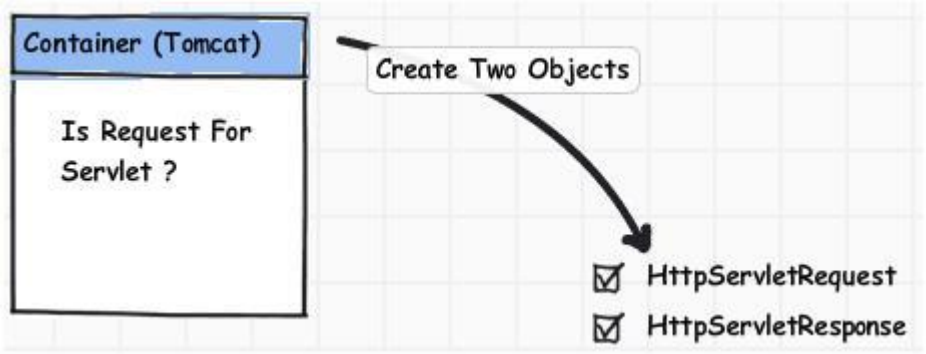
slow as need redeploy

How container handles the Servlet request?

1. ***Client browses the servlet URL:*** User clicks a link that has a URL of servlet.

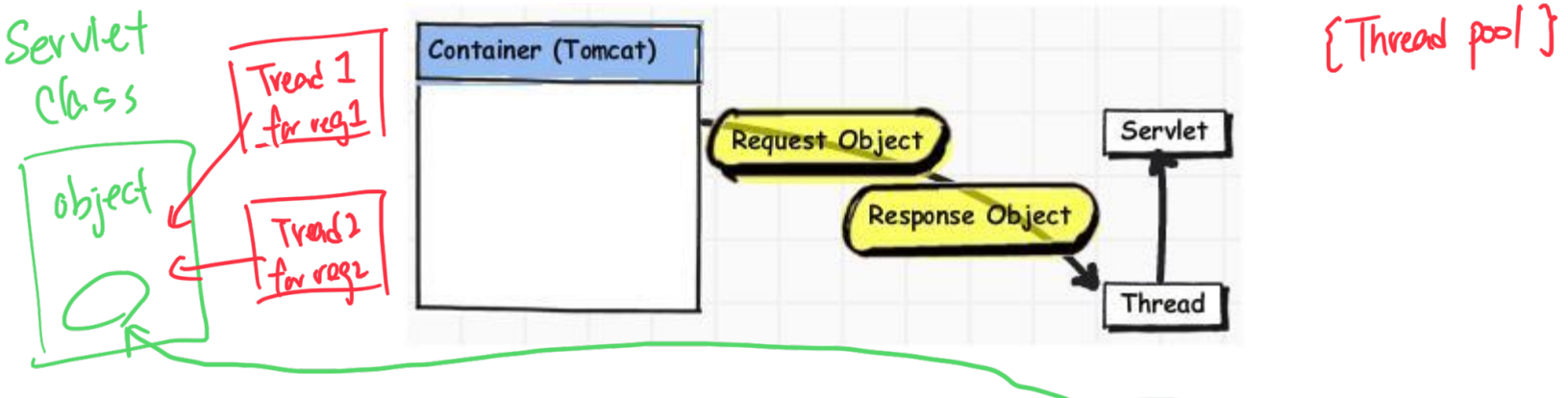


2. ***HttpServletRequest and HttpServletResponse:*** Container (e.g. Apache Tomcat) sees that the request is for servlet , so it creates two objects: Request and Response

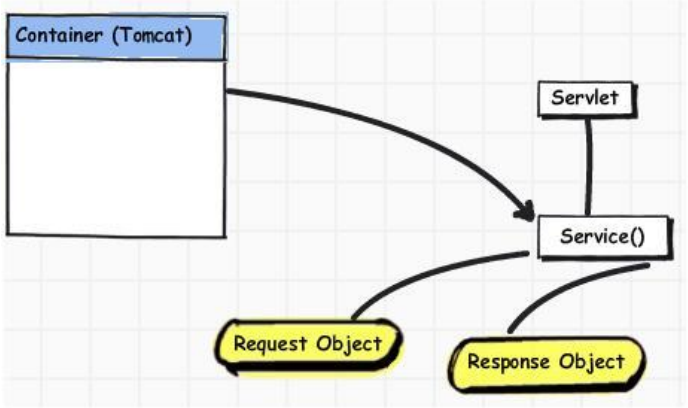


How container handles the Servlet request? (cont')

3. Create thread for Servlet: Container finds the correct servlet using "web.xml" file or @WebServlet annotations; and creates/allocates a thread for that request …

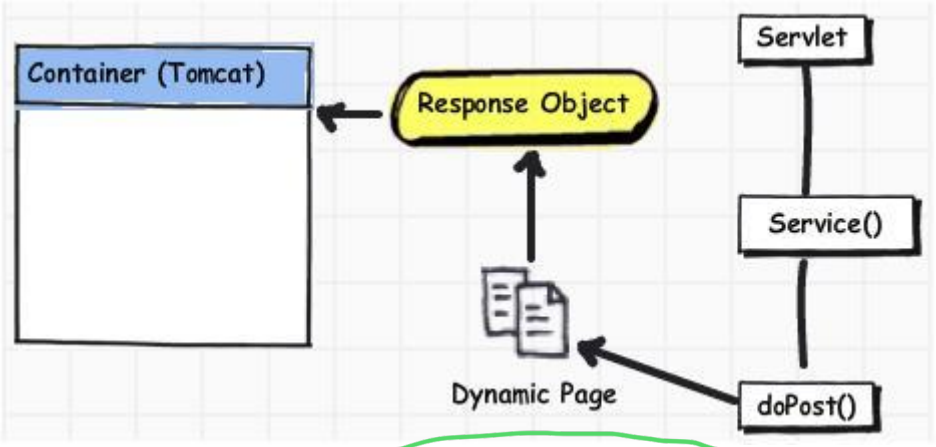


4. Service method of servlet: Container calls the servlet's **service()** method; and based on the type of request, service calls **doGet()** or **doPost()** methods.

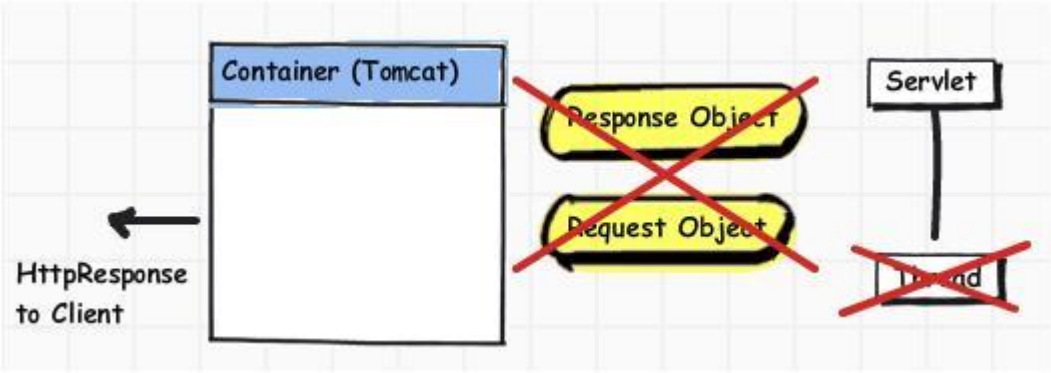


How container handles the Servlet request? (cont')

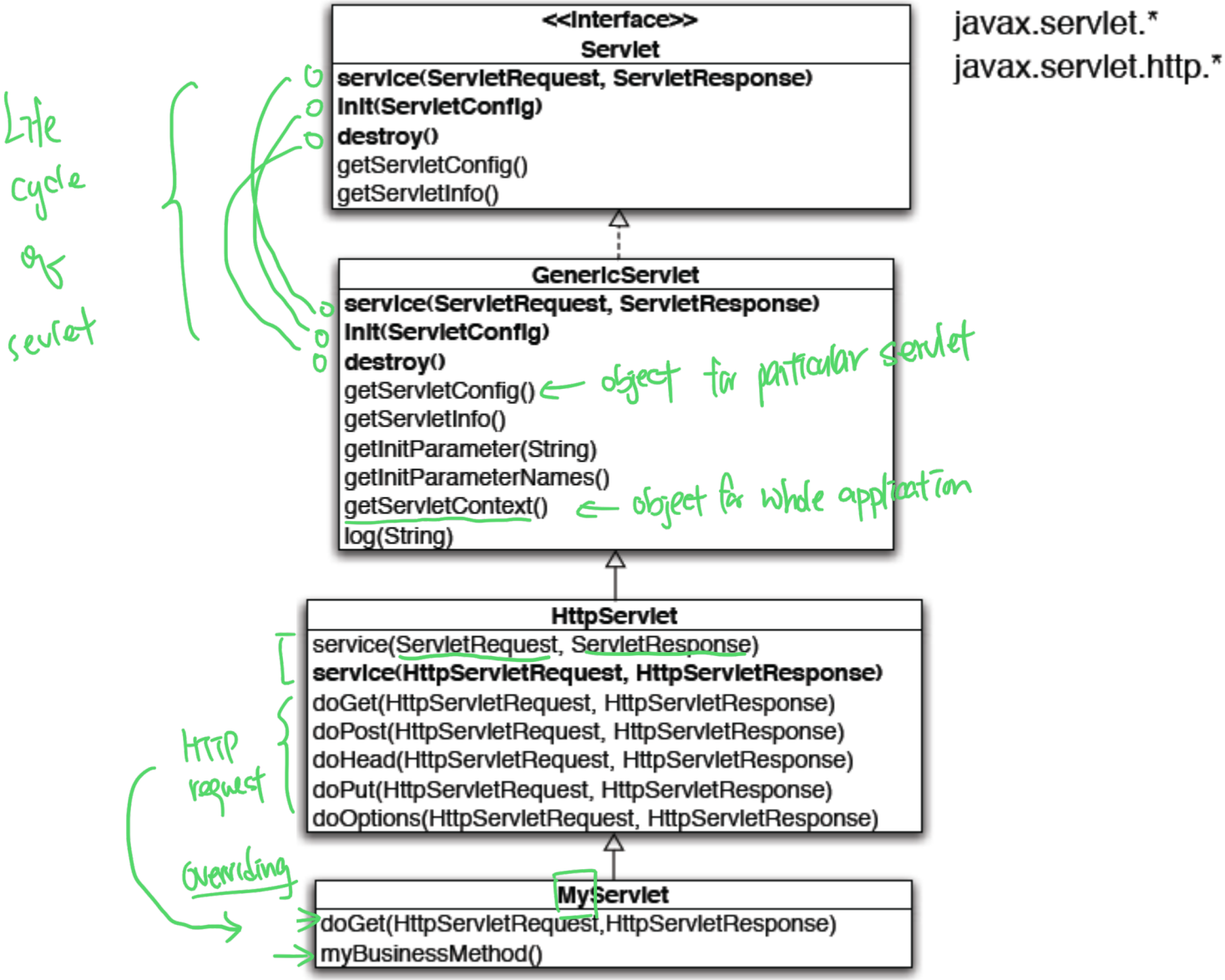
5. Servlet's response object: Let's assume that service() calls the doPost() method. doPost() method **generates dynamic page** and add the page in response object.



6. Destroy response and request object: Thread completes, container converts the response object into HttpResponse object and destroys the Servlet's response and request objects.



Servlet inherits "lifecycle" methods



javax.servlet.*
javax.servlet.http.*

Lifecycle of a Servlet

- The lifecycle of a servlet is controlled by the Web container

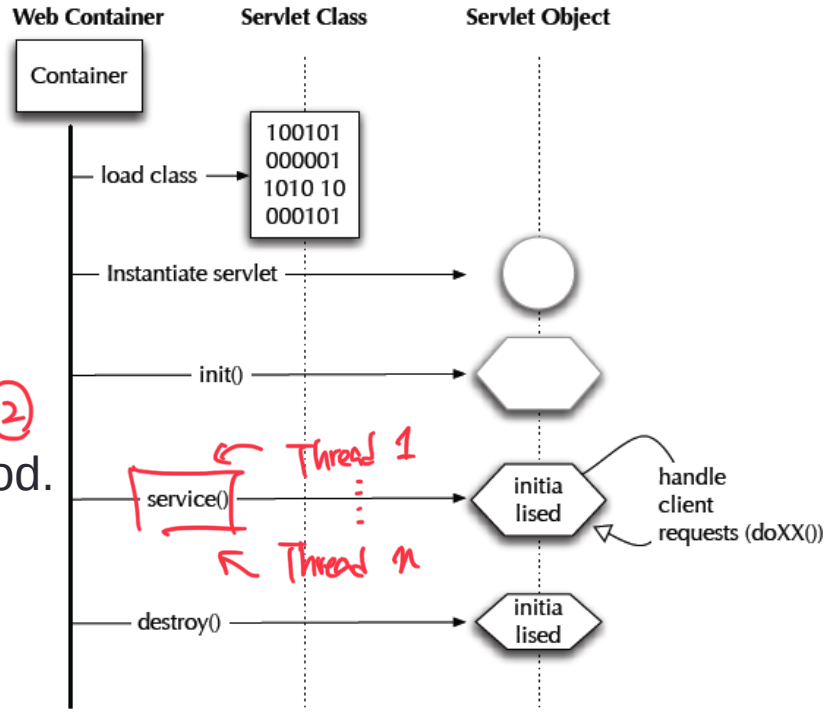
When a request is mapped to a Servlet

1. If an instance of the Servlet does not exist, the web container
   a. Load the Servlet class
   b. Create an instance of the Servlet class
   c. Initialize the Servlet instance by calling the `init` method.

2. Invoke the `service` method, passing request and response objects.

When the web container wants to reclaim memory resource of the servlet /
When the web container is being shut down

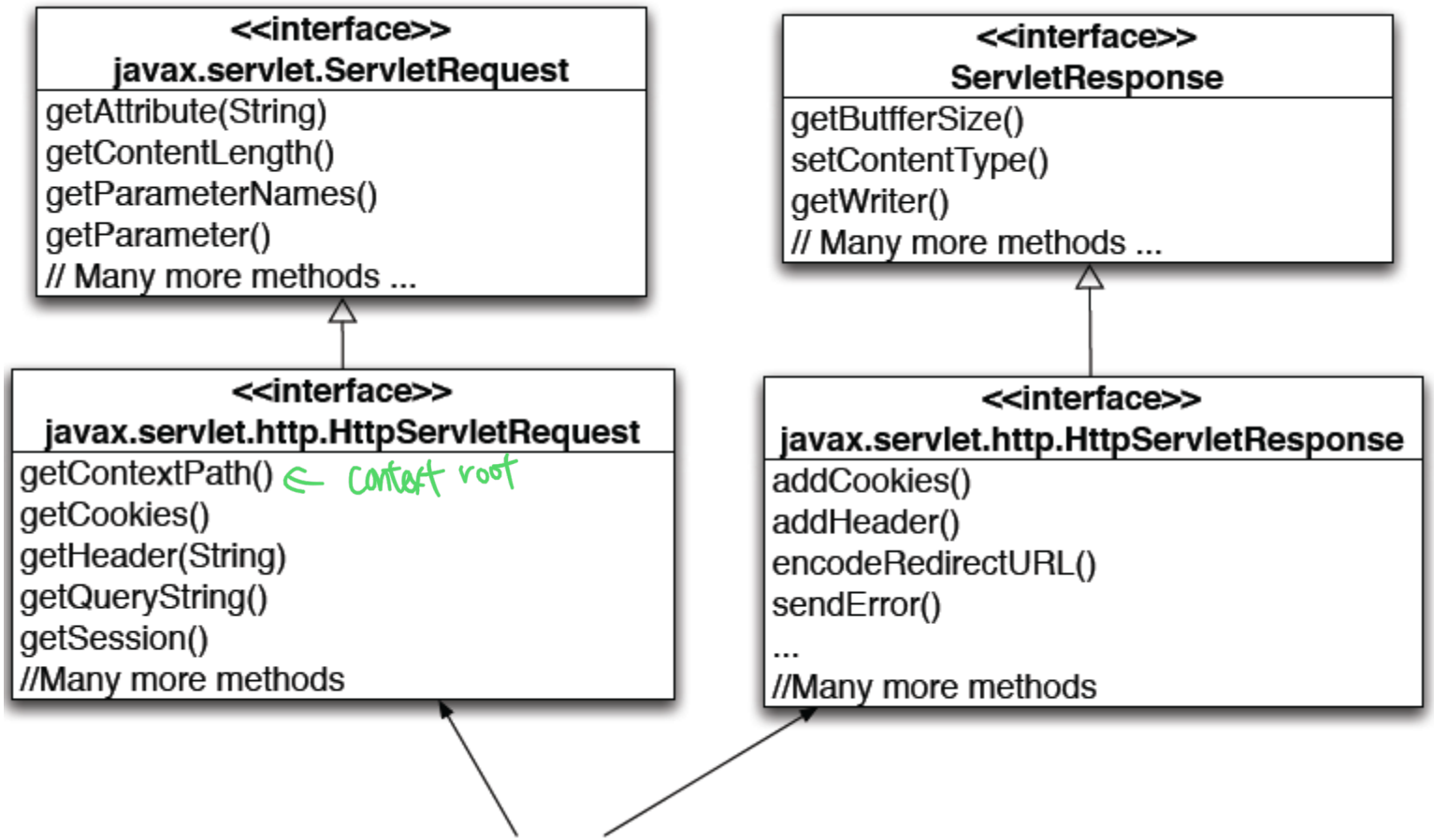
1. The web container calls the `destroy` method.



The Lifecycle of a Servlet (cont')

Lifecycle call	When it is called?	What it is for?	Override it?
init()	Container calls init() after the servlet instance is created but *before the servlet can service client's requests.*	Give you a chance to initialize your servlet before handling any request, e.g., getting a database connection. ★ initial event	Maybe
service()	When the first client request comes in, the container starts a new thread and calls service() method.	This one looks at the request and determines the HTTP method and invokes the matching doXX() on the servlet.	No
doXX() Main implementation	The service() method invokes appropriate doXX(), e.g., doGet() or doPost().	This is where your Servlet code begins. This is the method that is responsible for whatever the servlet is supposed to be doing.	Always
destroy()	Container calls destroy() when the container remove the Servlet from service or when the web container is being shut down.	Give you a chance to release any resource the servlet is using (e.g., memory), save any persistent state, and release the database object created in the init() method. ★ Finalize the res	Maybe

ServletRequest & ServletResponse Interfaces



The **container** implements HttpServletRequest and HttpServletResponse

All you should care about is when servlet is called, the **service()** passes two objects to your servlet.

HttpServletRequest & HttpServletResponse

The service() method invokes appropriate doXXX() method when the servlet receives an HTTP request.

Typically, your (http) servlet code would have the following structure:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        // your code to generate response ...
 }
  public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        // your code to generate response ...
 }
```

HTTP request method determines whether doGet() or doPost() runs.

Request Parameters

- Request parameters come in two different forms:

1. **Query parameters** (also called URL parameters), or
2. **Post variables** or **Form variables**

- Client can send data to the server using an HTML <form> with the method GET / POST:
  - GET: Form data is sent as query parameters
    E.g., http://www.server.com/process**?name=Tim**
    GET
  - POST: Form data is sent as post variables in the HTTP POST request body.
    post → like in form
    (For safty reason)

HTML <form>: GET method

- Append form data into the URL in name/value pairs
- The length of a URL is limited (about 2-3k characters) (1) Can't be too long
- Never use GET to send sensitive data!
  - It will be visible in the URL. like password
- Useful for form submissions where a user want to bookmark the result, to enable quick access in future.
- GET is better for non-secure data, like query strings in Google. (Faster than POST)
- GET is supposed to be used for getting things - information retrieval

With the GET method, the HTTP request looks like this:

```
1 GET /?firstname=Tim&lastname=Berners-Lee HTTP/1.1
2 Host: server.com
```

HTML <form>: POST method

- Append form data inside the body of the HTTP request (data is not shown in URL).
- Has no size limitations.
- Form submissions with POST cannot be bookmarked.
- POST is supposed to be used for sending data to be processed - update or change something on the server

When sent using the POST method, the HTTP request looks like this:

Request Header

Body →

```
1 POST / HTTP/1.1
2 Host: server.com
3 Content-Type: application/x-www-form-urlencoded
4 Content-Length: 34
5
6 firstname=Tim&lastname=Berners-Lee
```

Example: Sending HTTP POST request by Web Form

- Deployment descriptor (/WEB-INF/web.xml)

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

   <servlet>
      <servlet-name>emailServlet</servlet-name>
      <servlet-class>hkmu.comps380f.EmailServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>emailServlet</servlet-name>
      <url-pattern>/echoEmail</url-pattern>
   </servlet-mapping>
</web-app>
```

Example: Sending HTTP POST request by Web Form (cont')

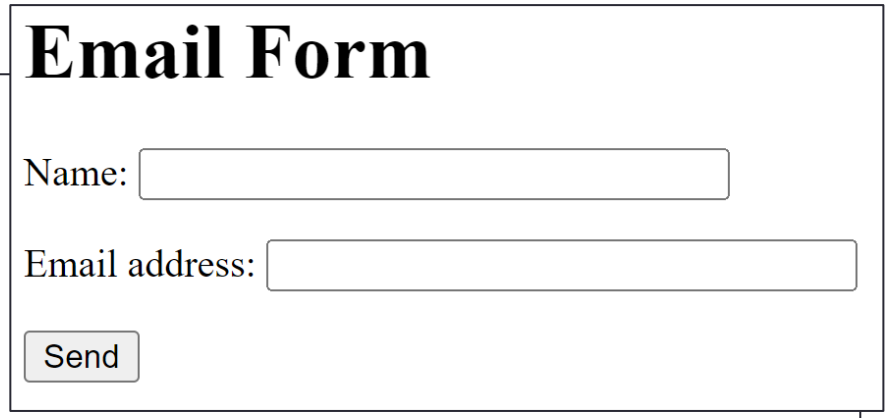
- Servlet

EmailServlet.java

```
package hkmu.comps380f;
…
public class EmailServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html><html>");
        out.println("<head><title>Email Form</title></head>");
        out.println("<body><h1>Email Form</h1>");
        out.println("<form method=\"post\" action=\"echoEmail\">");
        out.println("<p>Name: <input type=\"text\" name=\"name\" size=\"30\" /></p>");
        out.println("<p>Email address: <input type=\"text\" name=\"email\" size=\"30\" /></p>");
        out.println("<p><input type=\"submit\" value=\"Send\" /></p>");
        out.println("</form></body></html>");
    }
     …
```

Example: Sending HTTP POST request by Web Form (cont')

- On web browser, access http://localhost:8080/Lecture02/echoEmail (which is a HTTP **GET** request)
- The following web form will be shown:




```
<!DOCTYPE html>
<html>
    <head>
        <title>Email Form</title>
    </head>
    <body>
        <h1>Email Form</h1>
        <form method = "post" action="echoEmail">
            <p>Name: <input type="text" name="name" size="30"/></p>
            <p>Email address: <input type="text" name="email" size="30"/></p>
            <p><input type="submit" value="Send"/></p>
        </form>
    </body>
</html>
```

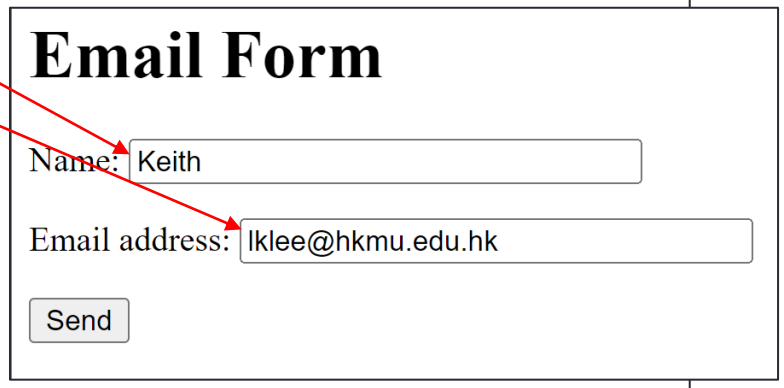
Example: Sending HTTP POST request by Web Form (cont')

- Request parameters can be obtained by using getParameter on the request object.

EmailServlet.java

```
…
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String name = request.getParameter("name");
    String email = request.getParameter("email");

    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE html><html><body>");
    out.println("Hello, " + name + "!");
    out.println("Your email address is " + email);
    out.println("</body></html>");
  }
}
```



Output shown in browser:

Hello, Keith! Your email address is lklee@hkmu.edu.hk

Request attributes and Request dispatcher

- We use **request attribute** when we want some other component of the application take over all or part of your request.

```
// code in doGet()
String postcode = getPostcode(request.parameter("suburb"));
request.setAttribute("pc", postcode);

RequestDispatcher view =
        request.getRequestDispatcher("DisplayPostcode.jsp");
view.forward(request, response);
```

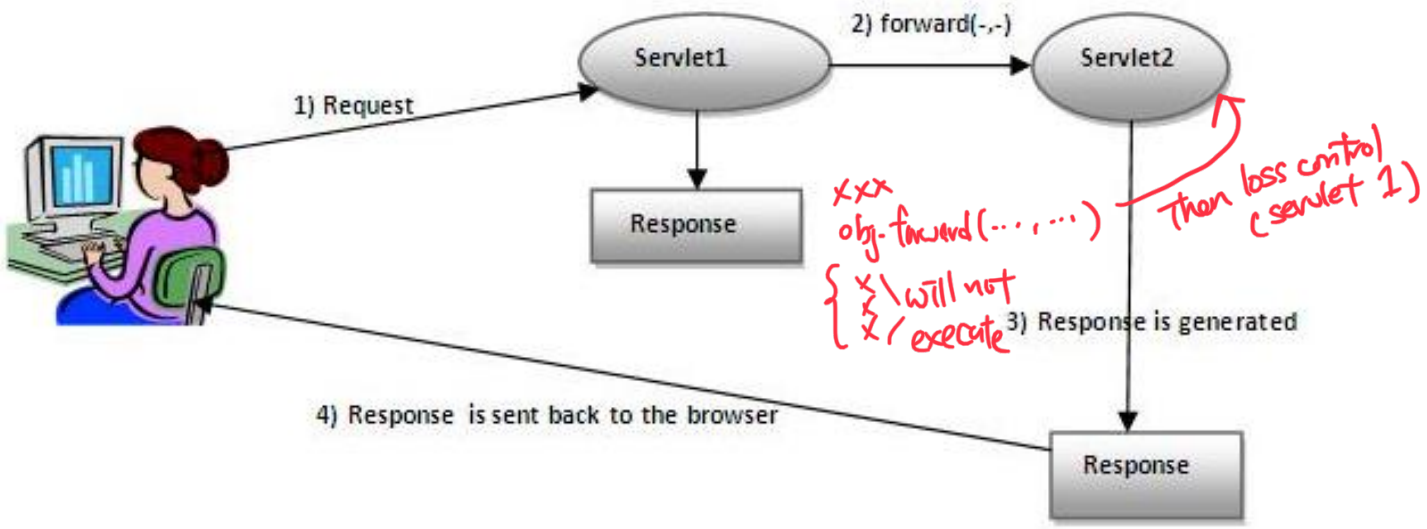
- Here, we use the RequestDispatcher interface to dispatch the request to another JSP page.
- The JSP page can now use the attribute `pc` in the request object to access the postcode.

RequestDispatcher: Forward

- The RequestDispatcher interface provides the facility of dispatching the request to another resource, e.g., servlet, jsp, or html.
- This interface can also be used to include the content of another resource also.
- It is one of the way of **servlet collaboration**.

The RequestDispatcher interface provides two methods: **forward** and **include**

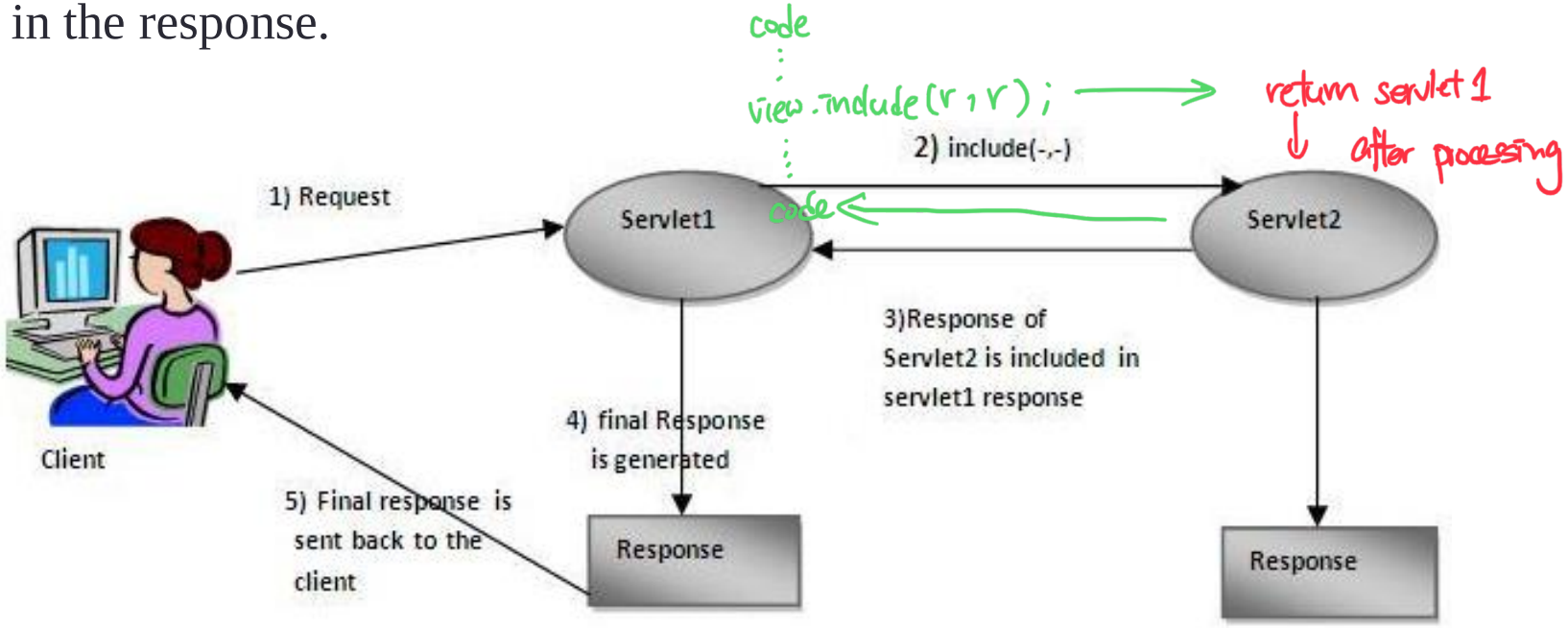
- **Forward:** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.




```
public void forward(ServletRequest request,ServletResponse response)
```

RequestDispatcher: Include

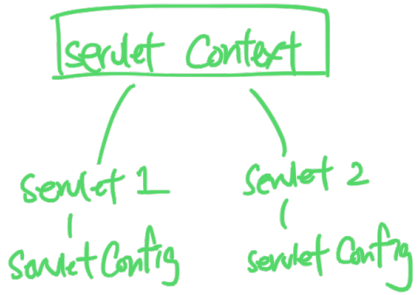
- **Include:** Includes the content of a resource (servlet, JSP page, or HTML file) in the response.




```
public void include(ServletRequest request,ServletResponse response)
```

ServletConfig vs. ServletContext

Once the servlet is initialized, the servlet gets access to two important objects: **ServletConfig** and **ServletContext**



ServletConfig:

- **Each Servlet has its own ServletConfig object.**
- **Servlet init parameters** can be configured for a servlet using the web.xml
  - Any deploy-time info that you do not want to hard-code into the servlet, so updating info does not require recompiling the servlet

↳ @WebServlet

```
<servlet>
    <servlet-name>ServletConfigTest</servlet-name>
    <servlet-class>com.javapapers.ServletConfigTest</servlet-class>
    <init-param>
        <param-name>topic</param-name>
        <param-value>ServletConfig vs. ServletContext</param-value>
    </init-param>
</servlet>
```

ServletConfig vs. ServletContext (cont')

ServletContext:

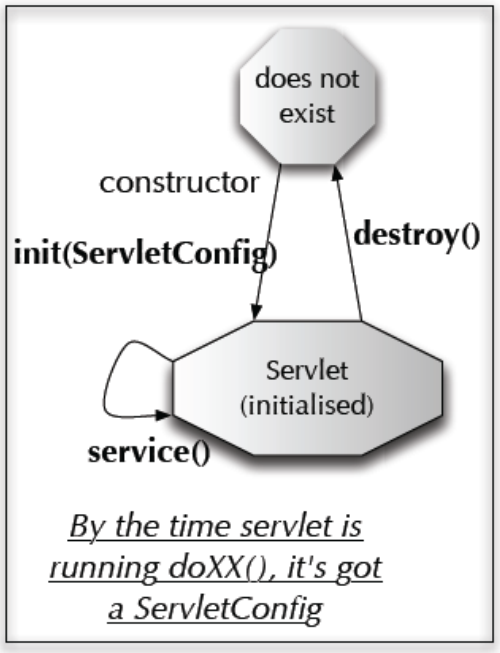
- **A web application has only a single ServletContext**
- The servlet container uses it to communicate with **all servlets**
- **Context init parameters** can be configured for all servlet using the web.xml
  - Details about Servlets' execution environment, e.g., the MIME type of a file, the path of a log file.

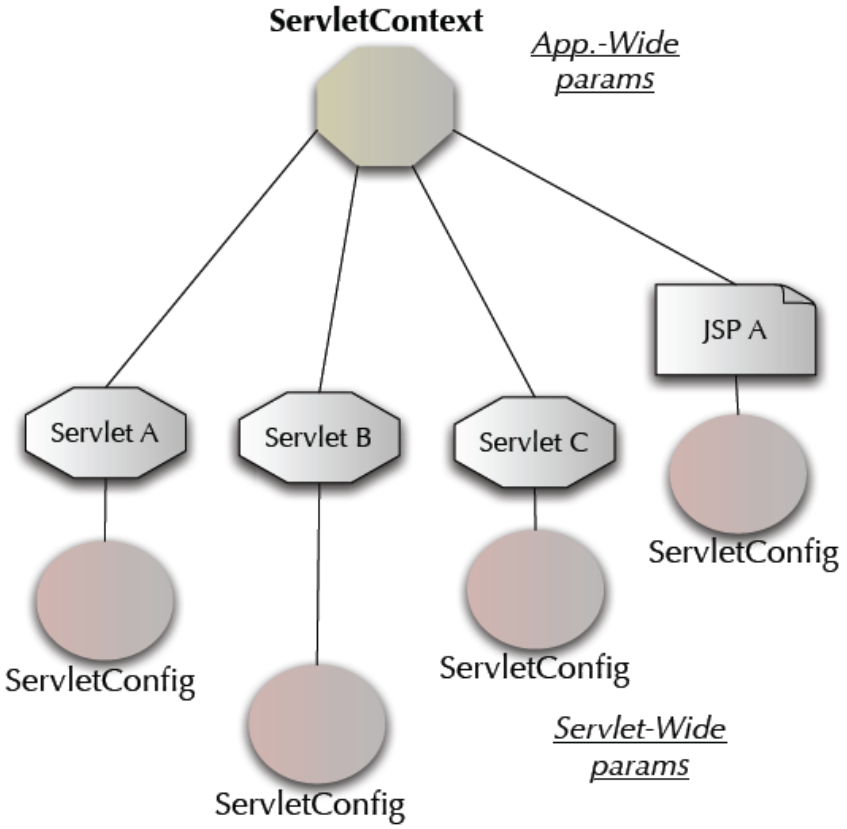
```
<context-param>
    <param-name>globalVariable</param-name>
    <param-value>comps380f</param-value>
</context-param>
```

Globe para

ServletConfig vs. ServletContext (cont')

To sum up, there is only one ServletContext that are shared across the entire app, but each servlet in the web app has its own ServletConfig.

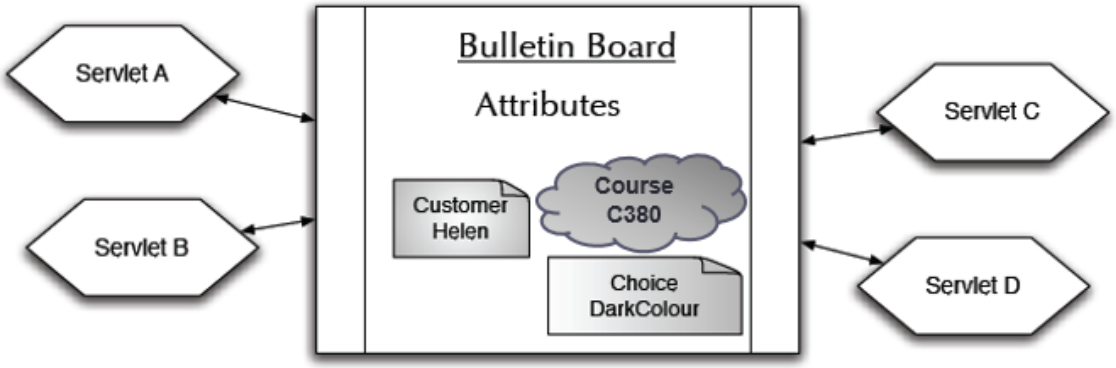




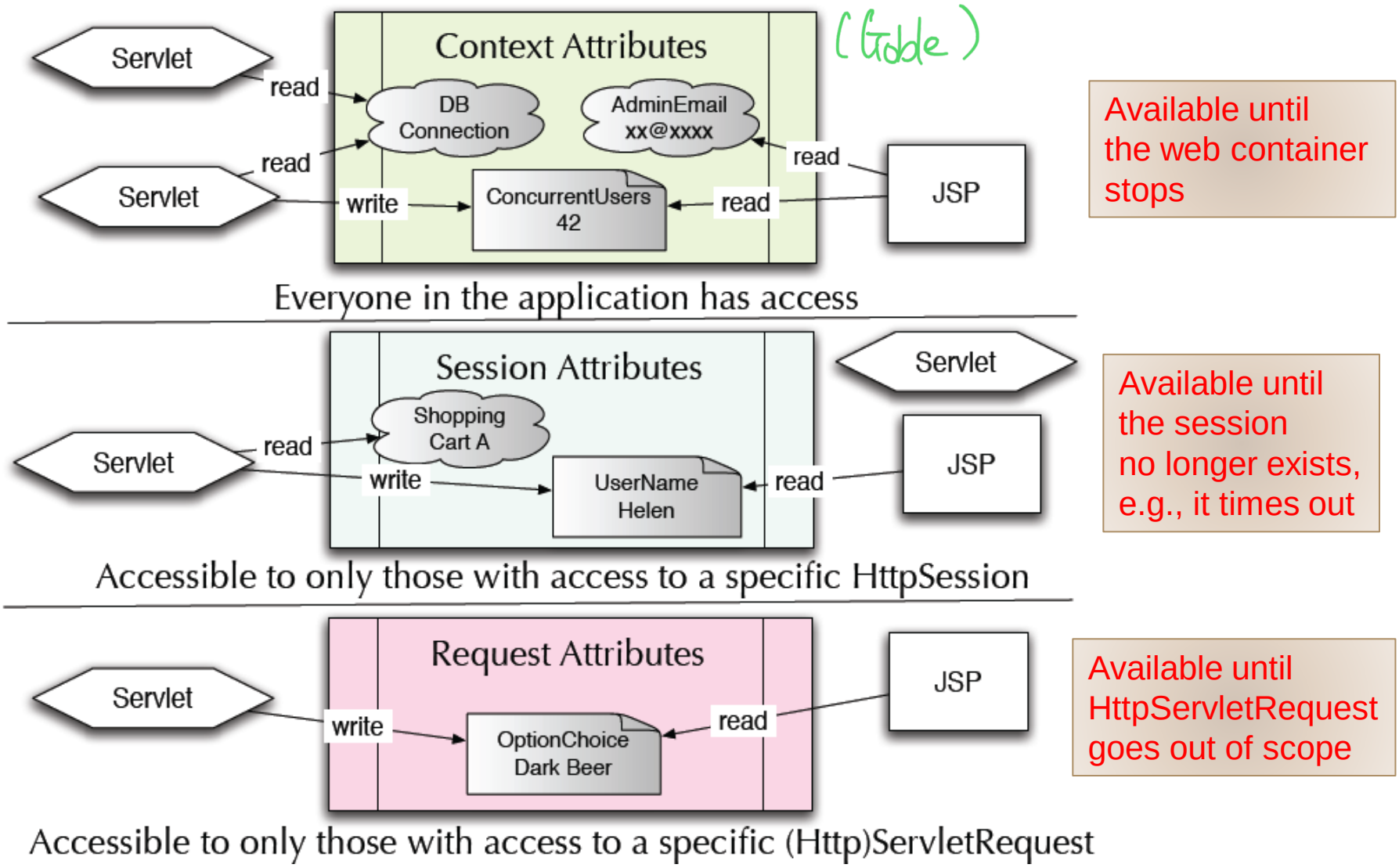
Remember if you change a value you must redeploy (but not recompile) the web app in order to get the new value, because a servlet is only initialized once at the beginning of its life.

Attributes

- An attribute is an object bound to one of the following objects:
  - ServletContext (web-app wide object!): Context attribute
  - HttpServletRequest: Request attribute
  - HttpSession: Session attribute
- An attribute is simply a name/object pair:
  - Name: a String
  - Attribute: an Object
- Think of it as an object pinned onto a bulletin board. Somebody sticks it on the board so that others can get it …



Scope and Lifespan of Different Attributes

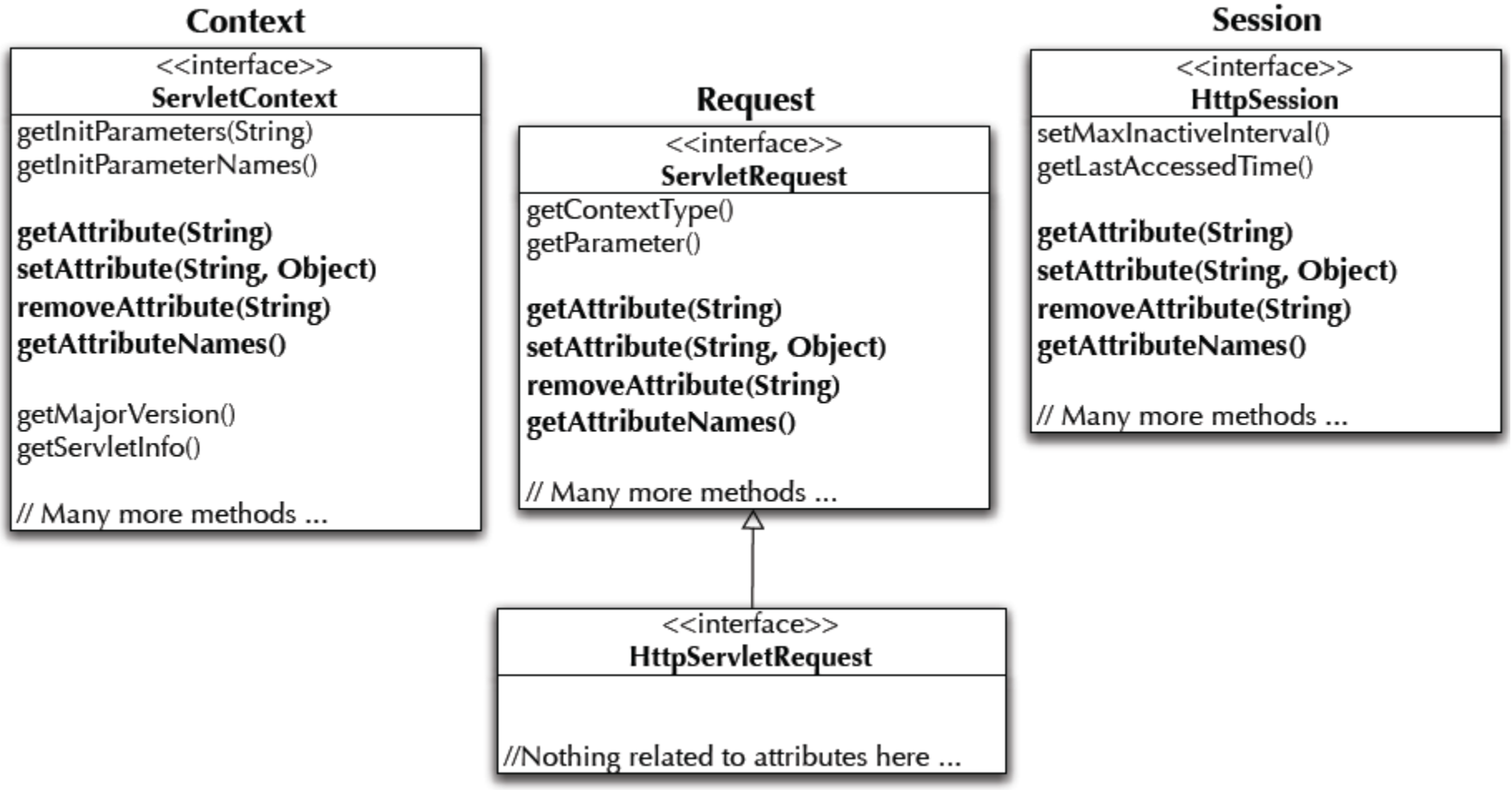


Available until the web container stops

Available until the session no longer exists, e.g., it times out

Available until HttpServletRequest goes out of scope

Attributes API



Attributes vs. Parameters

	Attributes	Parameters
Types	• Context attributes • Request attributes • Session attributes	• Context init parameters • Request parameters • Servlet init parameters
Method to set	`setAttribute(String name, Object value)`	in Deployment Descriptor (DD), or via client input
Return type	**Object**	**String**

Method to get	`getAttribute(String name)`
	`...`
		`getParameter(String name)`
		`getParameterValues(String name)`
		`getInitParameter(String name)`
		`...`

Difference between getAttribute() and getParameter() ?

- Return type for attributes is an **Object**, whereas return type for parameters is a **String**.
- When calling the getAttribute( ) method, bear in mind that the attributes must be **cast to the suitable data type**.
- Note that there is **no servlet specific attributes**, and there are **no session parameters**.