

Tasarım Örüntüleri ile Spring Eğitimi 5

Java Web Uygulamalarına Giriş

Java Servlet Nedir?

- **Web server** tarafında çalışan java nesneleridir
- Client'dan gelen request'i ele alarak **dinamik olarak web sayfaları** oluşturmayı sağlarlar
 - HTML form ile **submit edilen veriyi** ele almakta kullanılırlar
 - Veritabanı **sorgu sonuçlarını** kullanıcılara göstermek için kullanılırlar
 - HTTP requestleri arasında **state bilgisini** korumaya yardımcı olurlar
- Çalışmaları için Servlet uyumlu bir **Web Container** (web server)'a deploy edilmeleri gerekir

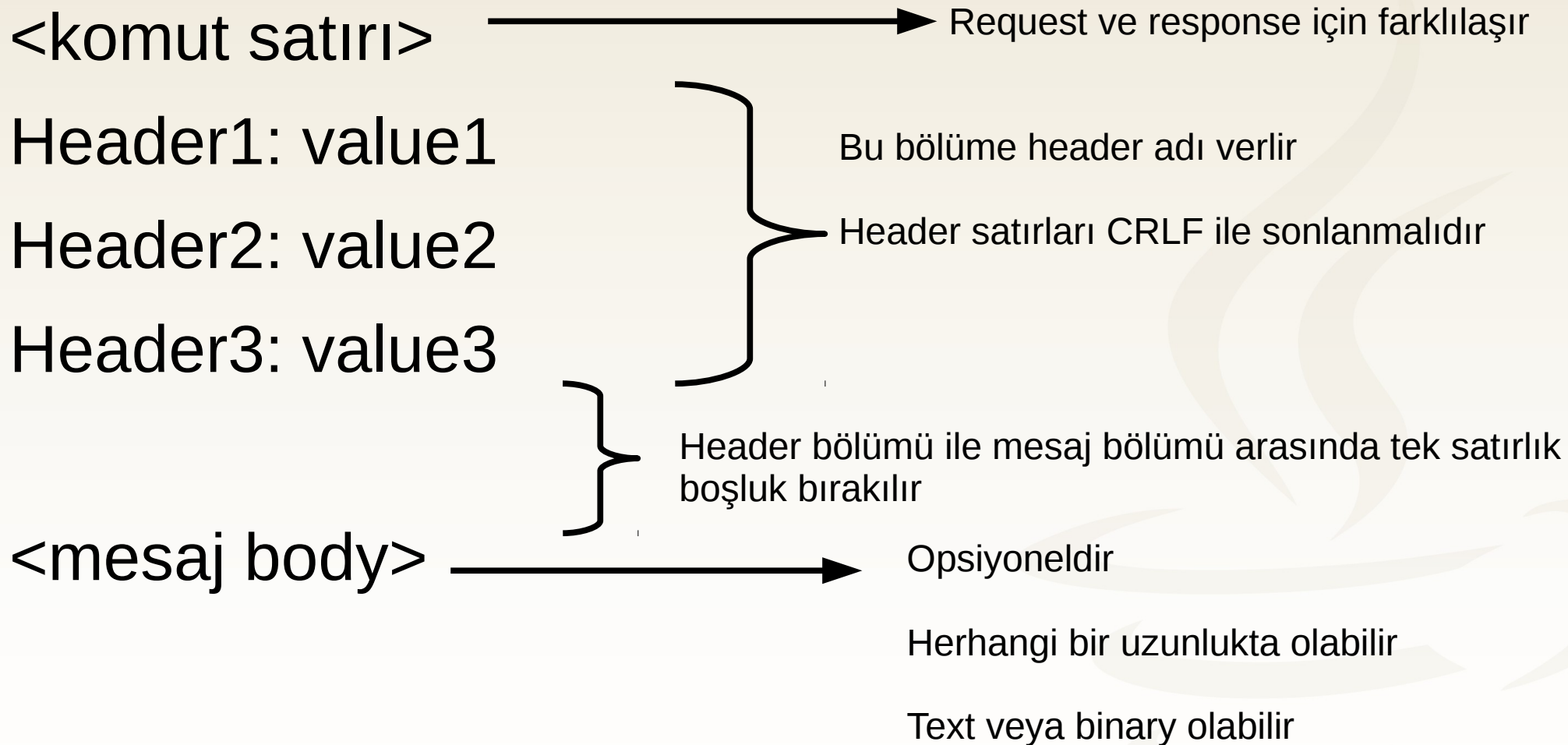
Java Servlets ve HTTP

- Günümüzde etkileşimli, dinamik web uygulamaları yapmak için **temel teknoloji** haline gelmişlerdir
- Servlet teknolojisi herhangi bir **client-server protokolüne bağımlı değildir**
- Ancak HTTP **en yaygın kullanılan protokoldür**
- Java Servlets, HTTP ile özdeşleşmiştir, çoğunlukla **HTTP Servlet** olarak da bilinirler
- Servlet'lar **Java sınıflarıdır** ve nesneleri JVM içerisinde çalıştırılır

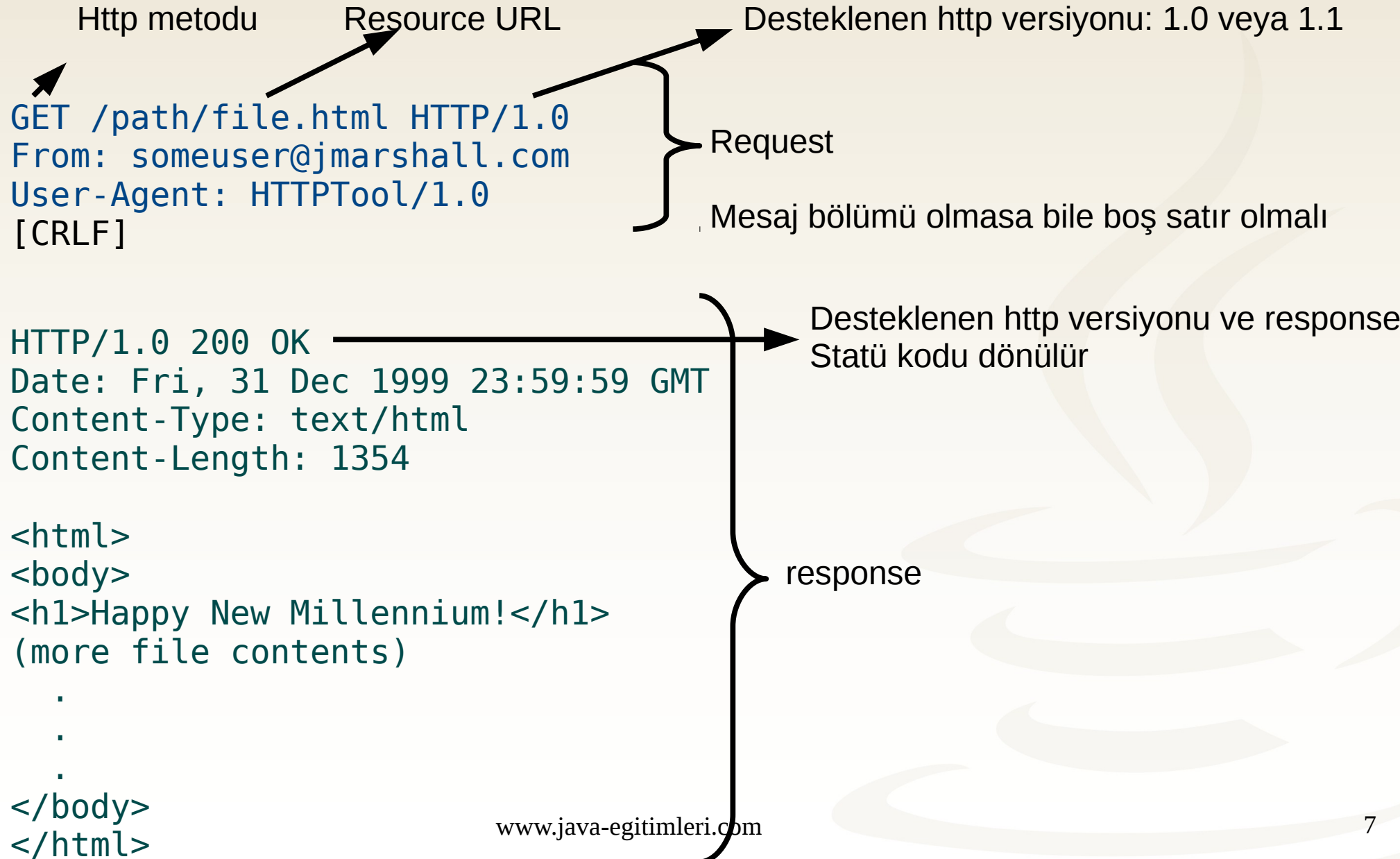
HTTP Nedir?

- Web'in text tabanlı **network protokolüdür**
- Client ve Server arasındaki **request** ve **response**'larla iletişim söz konusu olur
- Bir web sunucusundaki **URL** ile tespit edilebilen herhangi bir “**resource**”a erişim sağlar
- Resource **herhangi bir büyüklükteki veridir**
 - Text dosya, resim, ses, video gibi **statik** dosya olabilir
 - Yada **dinamik** olarak üretilen bir veri olabilir
 - **URL ile tespit edilebilmelidir**

HTTP Request ve Response Yapısı



HTTP Request ve Response Örneği



HTTP GET ve POST Metotları

- **GET**
 - **Resource URI** ile belirtilen veriyi sunucudan alır
 - Mesaj bölümü **Request URL**'inde taşınır
 - Buna **query string** adı verilir
 - Mesaj bölümünün **uzunluğu** bu nedenle **sınırlıdır**
 - Aynı GET request'nin sunucu tarafında tekrar tekrar çalıştırılması herhangi bir **side effect yaratmamalıdır**
 - Başka deyişle sunucu tarafında herhangi bir **değişikliğe neden olmamalıdır**

HTTP GET ve POST Metotları

■ POST

- **Resource URI** ile belirtilen veriyi sunucudan alır
- Request ile gönderilen **mesaj bölümü stream** olarak sunucuya aktarılır
- Bu nedenle gönderilecek **verinin uzunluğu** için herhangi bir **sınır yoktur**
- Sunucu tarafında bir **değişikliğe neden olacak** işlemler bu metot ile gerçekleştirilir
- Örneğin bir hesaptan başka hesaba para transferi yapılması

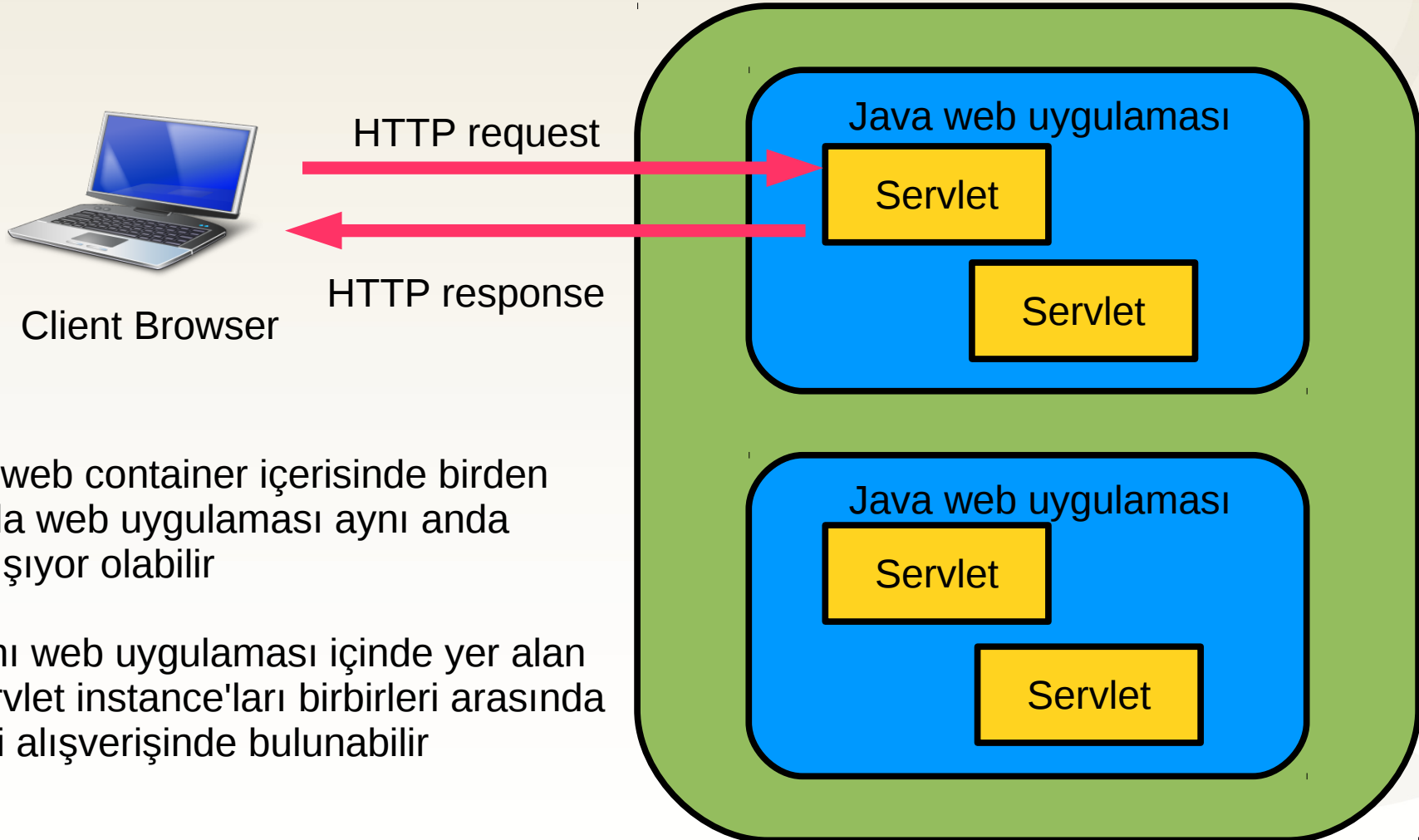
HTTP Response Statü Kodları

- 200: OK
- 201: Resource created
- 204: Response empty
- 30x: Redirect
- 404: Resource not found
- 405: HTTP method not supported
- 409: Resource conflict
- 500: Internal server error

Java Servlets ve Web Uygulamaları

Jetty, Tomcat yaygın kullanılan
iki web container'dır




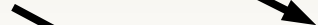
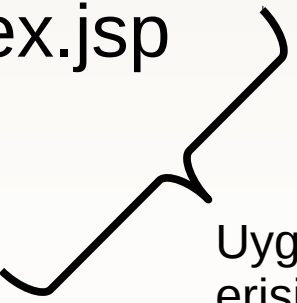
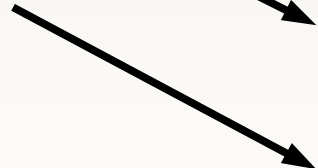
Web/Servlet container



Bir web container içerisinde birden fazla web uygulaması aynı anda çalışıyor olabilir

Aynı web uygulaması içinde yer alan Servlet instance'ları birbirleri arasında veri alışverişinde bulunabilir

Java Web Uygulamalarının Yapısı

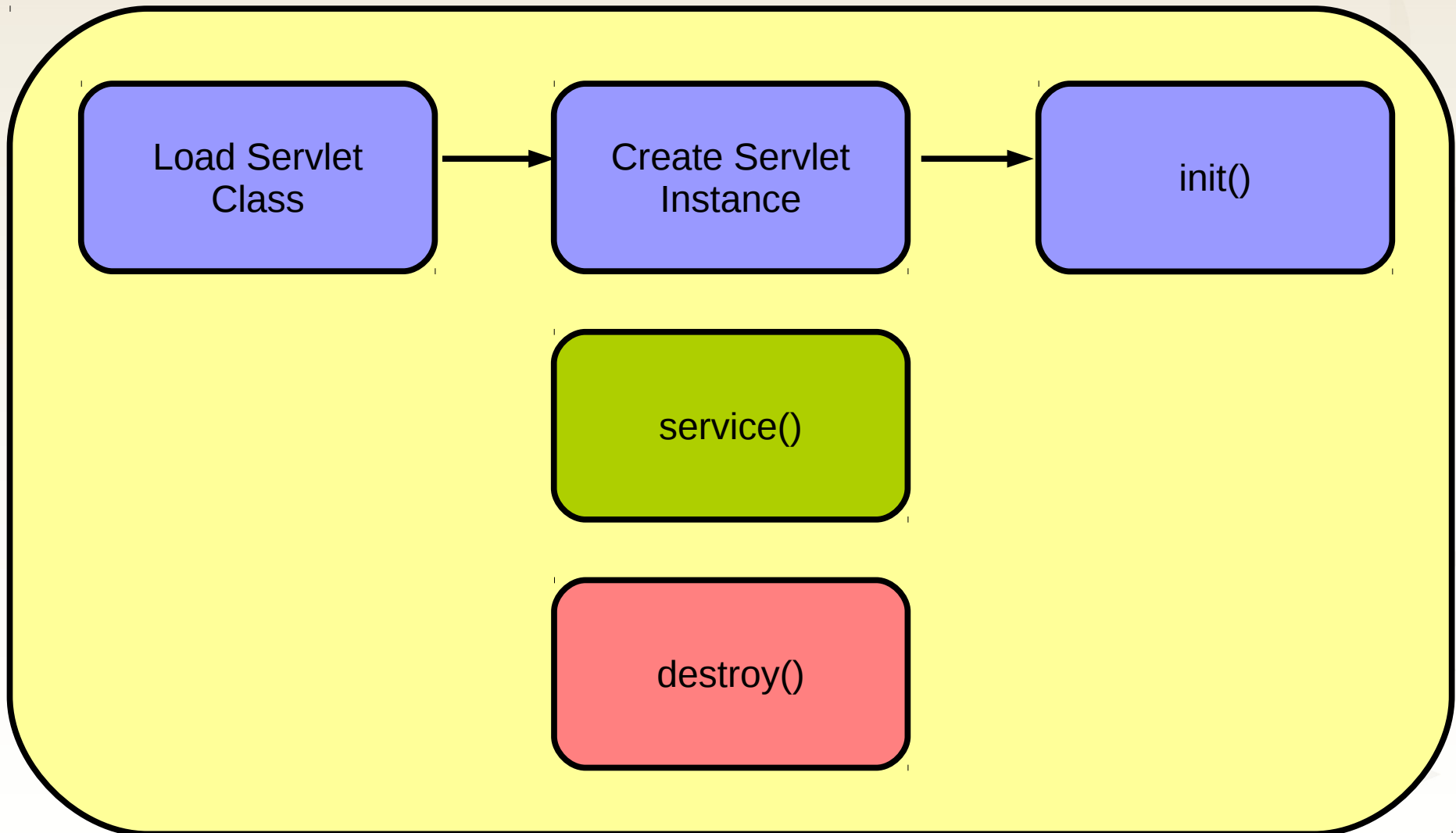
- petclinic (dizin)  Root dizindir, bu dizin altındaki dosyalar uygulama server'a deploy edildiğinde erişilebilir hale gelir
 - WEB-INF (dizin)  Özel bir dizindir, client tarafından doğrudan erişilemez
 - classes (dizin)  Uygulamanın java sınıflarını ve Classpath'de bulunması gereken diğer Resource'larını içerir
 - lib (dizin)
 - web.xml  Web uygulamasının ihtiyaç duyduğu Jar'ları içerir
 - index.jsp
 - ...
-  Uygulamaya ait erişilebilir dosya ve dizinler root dizinin altında oluşturulur
-  Web uygulamasının konfigürasyon dosyasıdır

Web Archive Dosyası (WAR)

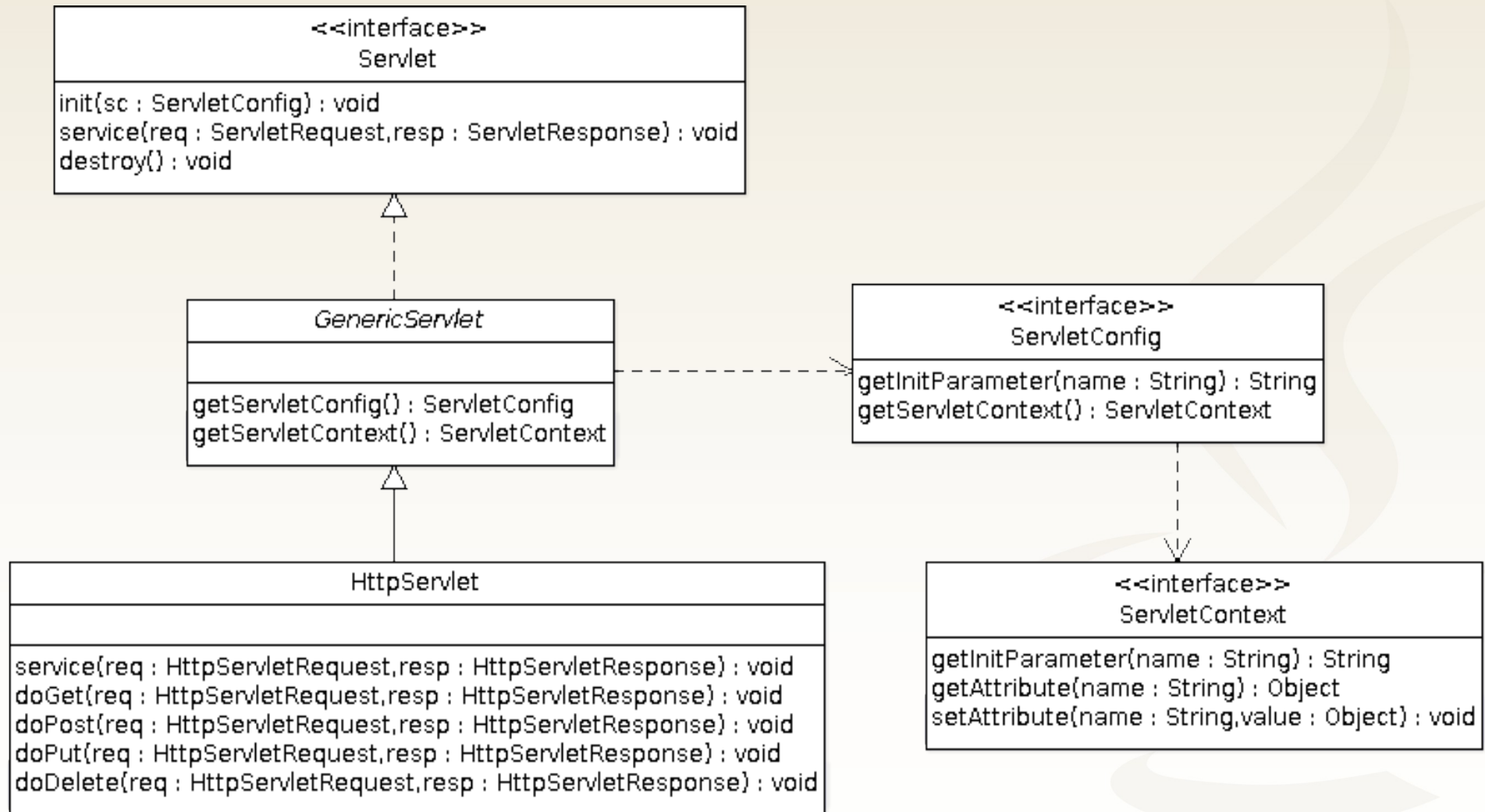
- Java Archive (**JAR**) dosyasıdır
- JAR dosya formatı **ZIP** dosya formatı üzerine bina edilmiştir
- Web uygulama **dizin yapısının tek bir dosya halinde paketlenip deploy edilmesini sağlar**
- Dosyanın uzantısı “**.war**” şeklinde olmalıdır

Servlet Yaşam Döngüsü

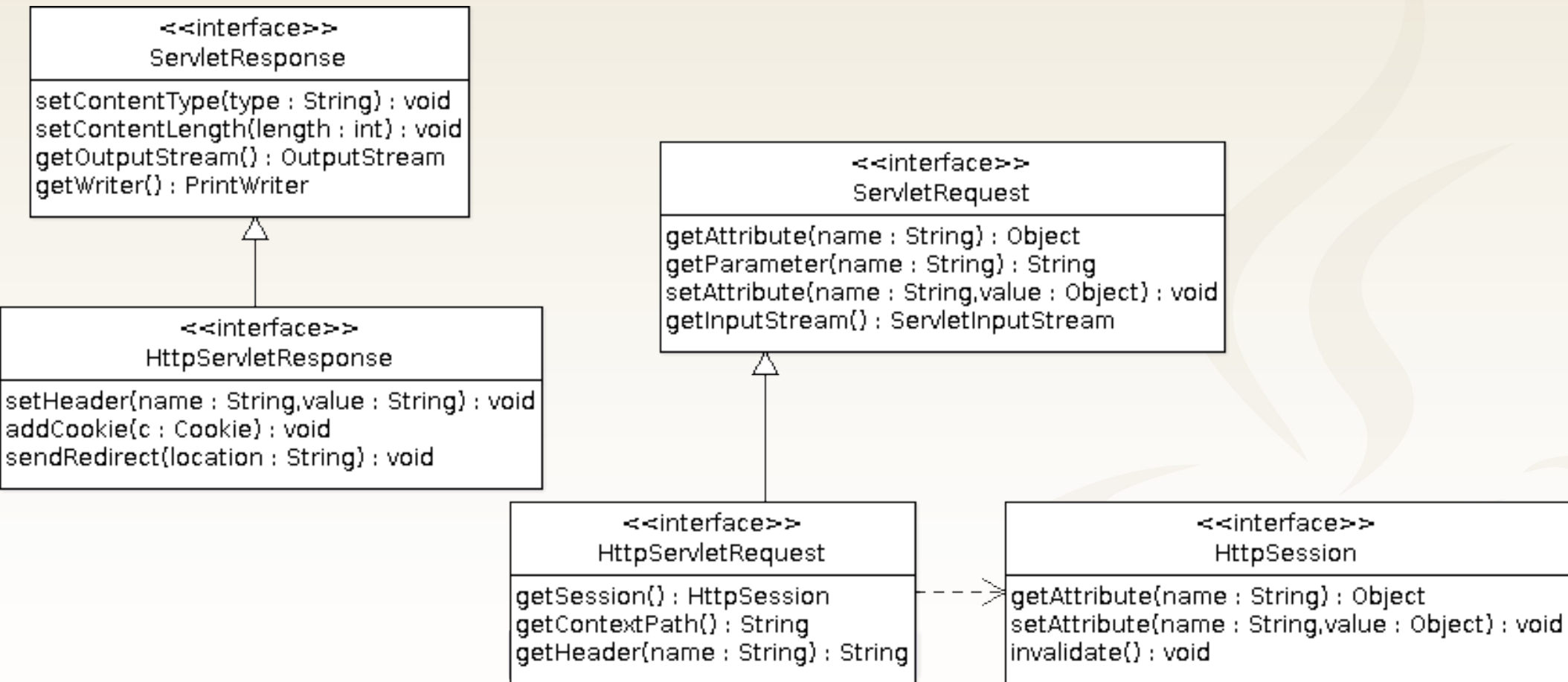
Servlet Container



Servlet Class Diagram



ServletRequest/Response Class Diagram



HttpServlet ve web.xml Konfigürasyon Örneği

```
public class HelloWorldServlet  
    extends HttpServlet {
```

```
    protected void  
doGet(HttpServletRequest request,  
HttpServletResponse response) throws  
ServletException, IOException {
```

```
        doPost(request, response);  
    }
```

```
    protected void  
doPost(HttpServletRequest request,  
HttpServletResponse response) throws  
ServletException, IOException {
```

```
        response.getWriter().write("Hello  
world");  
    }  
}
```

```
<web-app>  
    <display-name>petclinic</display-  
name>  
    <servlet>  
        <servlet-name>  
HelloWorldServlet  
        </servlet-name>  
        <servlet-class>  
com.javaegitimleri.petclinic.web.He  
lloWorldServlet  
        </servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>  
HelloWorldServlet  
        </servlet-name>  
        <url-pattern>  
/hello  
        </url-pattern>  
    </servlet-mapping>  
</web-app>
```

Java Servlets ve Template Method

```
public abstract class HttpServlet extends GenericServlet {
    protected void service(HttpServletRequest req,
        HttpServletResponse resp)
        throws ServletException, IOException
    {
        String method = req.getMethod();

        if (method.equals(METHOD_GET)) {
            long lastModified = getLastModified(req);
            if (lastModified == -1) {
                doGet(req, resp);
            } else {
                long ifModifiedSince =
                    req.getDateHeader(HEADER_IFMODSINCE);
                if (ifModifiedSince < lastModified) {
                    maybeSetLastModified(resp,
                        lastModified);
                    doGet(req, resp);
                } else {
                    resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
                }
            }
        } else if (method.equals(METHOD_HEAD)) {
            long lastModified = getLastModified(req);
            maybeSetLastModified(resp, lastModified);
            doHead(req, resp);
        } else if (method.equals(METHOD_POST)) {
            doPost(req, resp);
        }
    }
}
```

```
        else if (method.equals(METHOD_PUT)) {
            doPut(req, resp);
        } else if (method.equals(METHOD_DELETE)) {
            doDelete(req, resp);
        } else if (method.equals(METHOD_OPTIONS)) {
            doOptions(req, resp);
        } else if (method.equals(METHOD_TRACE)) {
            doTrace(req, resp);
        } else {
            String errMsg =
                lStrings.getString("http.method_not_implemented");
            Object[] errArgs = new Object[1];
            errArgs[0] = method;
            errMsg = MessageFormat.format(errMsg,
                errArgs);

            resp.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED, errMsg);
        }
    }
}
```

Text veya HTML İçerik Oluşturma

http://localhost:8080/petclinic/hello?name=kenan

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

```
response.setContentType("text/html");
```

Response içeriği oluşturulmadan
Evvel Content-Type header
Uygun mime type değerine set
edilmelidir

```
PrintWriter writer = response.getWriter();
```

```
String message = "<html><body><h2>Merhaba " + name +  
"</h2></body></html>";
```

```
response.setContentLength(message.length());
```

```
writer.write(message);
```

```
writer.close();
```

Response içeriğinin uzunluğu da
Content-Length header'ına set edilmelidir
Özellikle binary içerikte önemlidir

Response içeriğini oluşturma işlemi
Bitince Writer nesnesi kapatılmalıdır

Binary İçerik Oluşturma

```
InputStream is = new
FileInputStream("/home/ksevindik/Desktop/harezmi.jpg");
ByteArrayOutputStream bout = new ByteArrayOutputStream();
while(is.available() != 0) {
    bout.write(is.read());
}
is.close();
```

Öncelikle
Binary içerik
Servlet
Instance'a
yüklenir

```
byte[] image = bout.toByteArray();
```

```
response.setContentType("image/jpg");
```

→ Mime type set edilir

```
ServletOutputStream out = response.getOutputStream();
```

```
response.setContentLength(image.length);
```

```
out.write(image);
```

```
out.close();
```

Binary içeriğin uzunluğu
Content-Length header ile
Client browser'a belirtilir

→ İçerik yazıldıktan sonra stream kapatılır

Yaygın Kullanılan MIME Tipleri

- **text/plain** : mail ve news'lerdeki standart tip
- **text/html** : WWW için standart tip
- **image/jpeg** : imaj tipi
- **image/gif** : imaj tipi
- **application/octet-stream** : tipi bilinmeyen byte veri
- **audio/mpeg3** : ses tipi
- **video/mpeg** : video tipi
- **application/pdf** : pdf doküman tipi

Web.xml'de Servlet Konfigürasyonu

```
<web-app>  
  <servlet>  
    <servlet-name>PetClinicServlet</servlet-name>  
    <servlet-class>x.y.z.PetClinicServlet</servlet-class>  
  </servlet>  
  
  <servlet-mapping>  
    <servlet-name>PetClinicServlet</servlet-name>  
    <url-pattern>/petClinicServlet</url-pattern>  
  </servlet-mapping>  
</web-app>
```



url-pattern Servlet'e nasıl erişileceğini tanımlar

* ile wildcard kullanımda mümkündür: *.html, /petClinic* /*

Web.xml'de Servlet Konfigürasyonu

```
<web-app>
```

```
<context-param>
```

```
  <param-name>contextParam</param-name>
```

```
  <param-value>value</param-value>
```

```
</context-param>
```

Context param tanımlarına
ServletContext ile erişilebilir

Bütün Servlet'lar erişebilir

```
<servlet>
```

```
  <servlet-name>PetClinicServlet</servlet-name>
```

```
  <servlet-class>x.y.z.PetClinicServlet</servlet-class>
```

```
  <init-param>
```

```
    <param-name>servletParam</param-name>
```

```
    <param-value>value</param-value>
```

```
  </init-param>
```

```
  <load-on-startup>1</load-on-startup>
```

Init-param servlet tanımına
Özeldir

ServletConfig ile erişilebilir

```
</servlet>
```

```
</web-app>
```

Servlet'in startup sırasında yüklenmesini sağlar
Değer servlet tanımları arasındaki sıralamayı belirler,
Küçük değer önce yüklenir, Negatif değerde veya tanım yoksa
yükleme herhangi bir anda yapılabilir

@WebServlet ile Servlet Konfigürasyonu

Web.xml deki tanımlar yerine **WebServlet** anotasyonu ile de Servlet konfigürasyonu yapılabilir. Servlet'in hangi url'lerde devreye gireceği, init parametreleri vs buradan belirtilebilir

```
@WebServlet(name="HelloServlet",  
            urlPatterns={"/hello"},loadOnStartup=1)  
public class HelloWorldServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request,  
                          HttpServletResponse response)  
        throws ServletException, IOException {  
  
        doPost(request, response);  
    }  
  
    protected void doPost(HttpServletRequest request,  
                          HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.getWriter().write("Hello world");  
    }  
}
```


URL, URN ve URI Nedir?

- Uniform Resource Locator (**URL**) bir Uniform Resource Identifier (URI)'dir
- Bir **resource'un adresini ve ona erişim şeklini** tanımlar
- Uniform Resource Name (**URN**) de bir URI'dir
- Bir resource'un kimliğini lokasyonundan bağımsız olarak tanımlamayı sağlar
- URN **kimlik tanımı**, URL ise **lokasyon bilgisi** olarak düşünülebilir
- Kişinin kimliği ve bu kişinin yaşadığı yerin adresi gibi

URL'in Yapısı

- *scheme://username:password@domain:port/resource_path?query_string#fragment_id*
- Scheme, **URL'in geri kalan syntax'ını** belirler
- <http://www.java-egitimleri.com:80>
- <mailto:ksevindik@harezmi.com.tr>
- <ftp://guest:secret@harezmi.com.tr>
- [http://10.10.0.1/myapp?
firstName=kenan&lastName=sevindik](http://10.10.0.1/myapp?firstName=kenan&lastName=sevindik)
- <http://java.sun.com/docs/jdk6.html#toc>

RequestDispatcher

Ne İşe Yarar?

- Bir **servlet** içerisinde **başka bir servlet'i** **çağırarak** için kullanılır
- RequestDispatcher **HttpServletRequest** üzerinden elde edilir
- **Forward ve include** metotları vardır
- Forward, ilk servlet'in **response'unu sonlandırdıktan sonra** ikinci servlet'i çağırır
- Include, iki servlet'in **response içeriğini merge** etmeye imkan tanır

RequestDispatcher Kullanım Örneği

```
public class HelloWorldServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        response.getWriter().write("hello ");  
        request.getRequestDispatcher("/bye").include(request,  
            response);  
    }  
}
```

Servlet'in web.xml'de map edildiği url-pattern ile
RequestDispatcher nesnesi oluşturulur

Include ilk servlet'in response'u ile
ikincisini merge eder
Forward kullanılsaydı response
sadece ikinci servlet tarafından
oluşturulacaktı

```
public class GoodByeServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        response.getWriter().write("and goodbye!");  
    }  
}
```

HTTP ve Oturum Yönetimi

- HTTP **stateless** bir protokoldür
- Normalde aynı kullanıcının iki request'i arasında herhangi bir **state bilgisi server tarafında tutulmaz**
- HttpSession iki request arasında tutulması gereken **state bilgisinin saklanabileceği yapıdır**
- HttpSession **unique bir ID** ile ifade edilir
- Bu ID iki request arasında ya **cookie** içerisinde ya da **request parametresi** ile taşınır

Session Cookie ile Oturum Takibi

```
public class Servlet1 extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {
```

```
        HttpSession session = request.getSession();  
        String message = request.getParameter("message");  
        session.setAttribute("msg", message);
```

```
    }
```

```
}
```

http://localhost:8080/petclinic/servlet1?message=hello



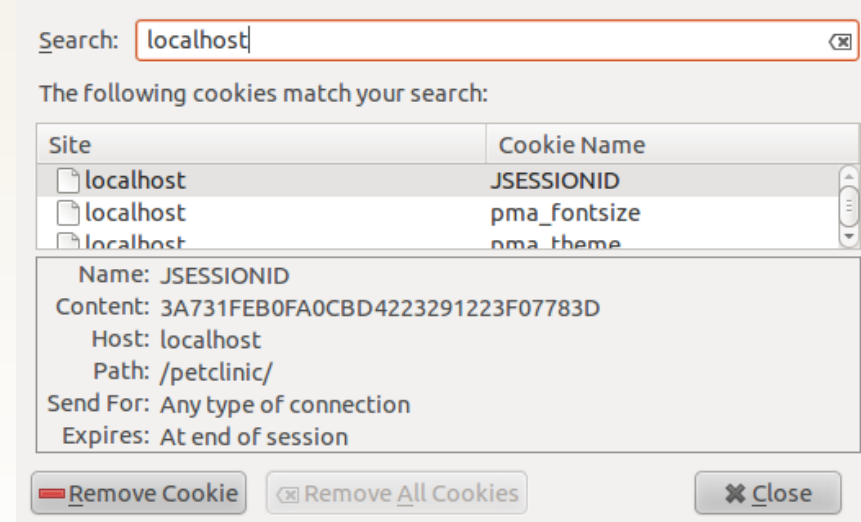
http://localhost:8080/petclinic/servlet2

```
public class Servlet2 extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {
```

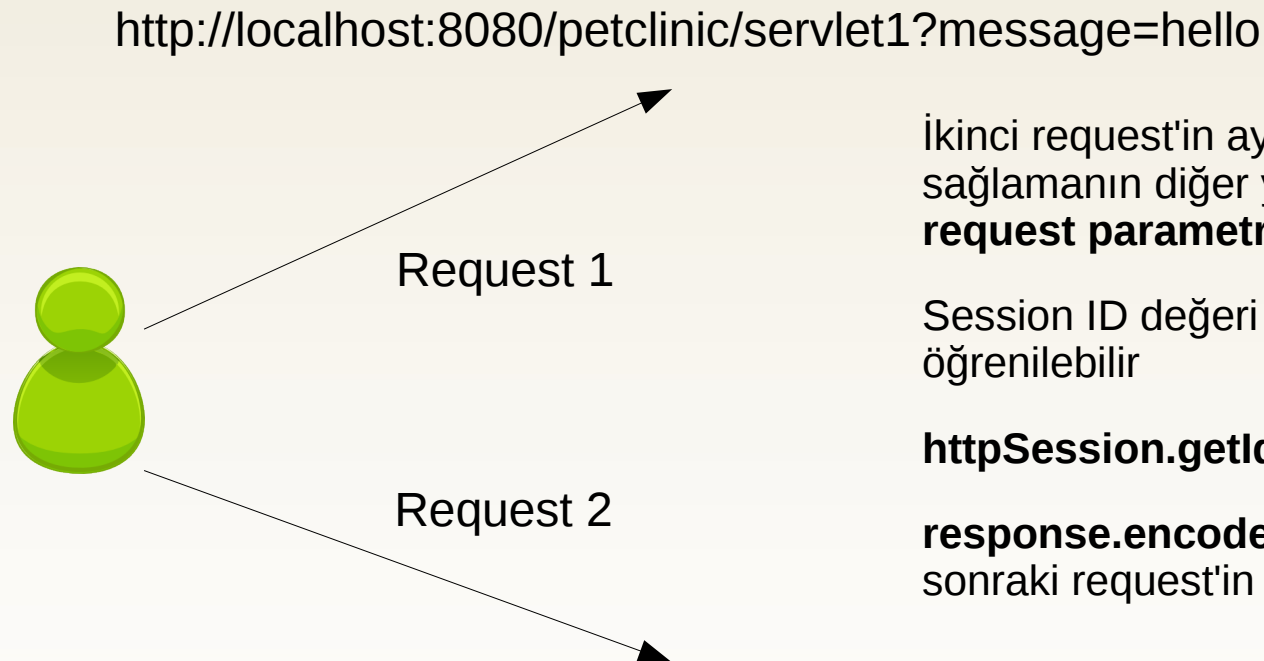
```
        HttpSession session = request.getSession(false);  
        String message = (String) session.getAttribute("msg");  
        response.getWriter().write(message);
```

```
    }
```

```
}
```



JSESSIONID Request Parametresi ile Oturum Takibi



İkinci request'in aynı HttpSession'a erişmesini sağlamanın diğer yolu request'in **JSESSIONID request parametresi** ile çağırılmasıdır

Session ID değeri Cookie içeriğinden öğrenilebilir

httpSession.getId() metodu ile elde edilebilir

response.encodeURL() metodu ile de bir sonraki request'in URL'ine eklenebilir

Session Timeout

- İstemci tarafından **belirli bir süre HttpSession'a erişilmediği vakit** server tarafından Session sonlandırılır
- Bu süre sadece **web.xml içerisinde** kontrol edilebilir
- Session'ı sonlandırmanın diğer bir yolu doğrudan **HttpSession nesnesinin invalidate() metodunu** çağırmaktır
- Varsayılan timeout değeri 30 dk'dır

HttpSessionListener Ne İşe Yarar?

- **HttpSession** nesneleri **yaratıldıkları ve sonlandırıldıkları vakit** web uygulaması içerisinde **bir takım işlemlerin yapılması** gerekebilir
- **HttpSessionListener** arayüzü bu amaçla kullanılır
- Bu arayüzü implement eden sınıflar web uygulamasındaki **HttpSession instance'ları hakkında bilgilendirilirler**
- Devreye girebilmeleri için **web.xml içerisinde tanımlanmaları gerekir**

HttpSessionListener Örneği

```
public class MySessionListener implements HttpSessionListener {

    public void sessionCreated(HttpSessionEvent se) {
    }

    public void sessionDestroyed(HttpSessionEvent se) {
    }

}
```

Web uygulamasında yeni bir HttpSession
Yaratıldığı vakit çağrılır

Mevcut herhangi bir HttpSession sonlandırıldığı
Vakit çağrılır

Sonlandırma session timeout olması veya
Explicit invalidate işlemi ile gerçekleşebilir

Web.xml'de Listener ve Session Timeout Konfigürasyonu

```
<web-app>
```

```
  <listener>  
    <listener-class>x.y.z.MySessionListener</listener-class>  
  </listener>
```

```
  <session-config>  
    <session-timeout>5</session-timeout>  
  </session-config>
```

```
</web-app>
```

Default değer 30 dk'dır

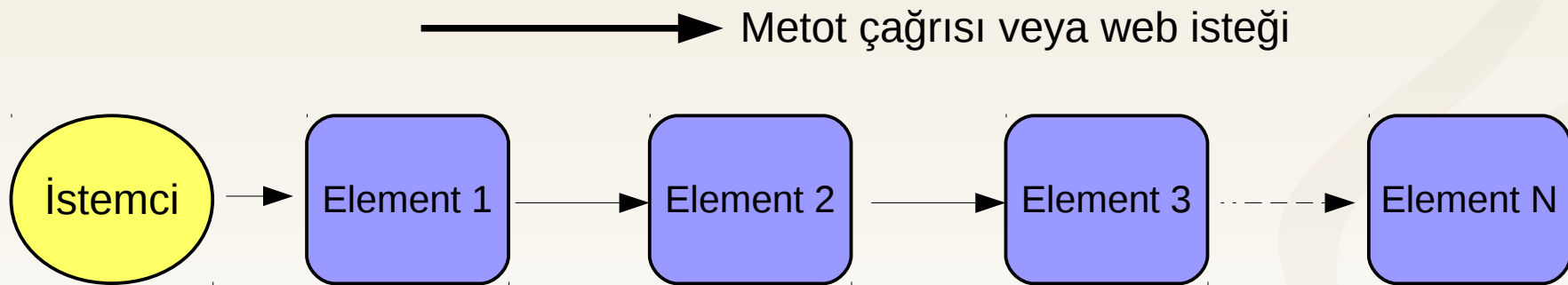
Web.xml'de Listener tanımları, listener elemanları içerisinde ayrı ayrı yapılmalıdır

Tasarım Örüntüleri'ne Devam...

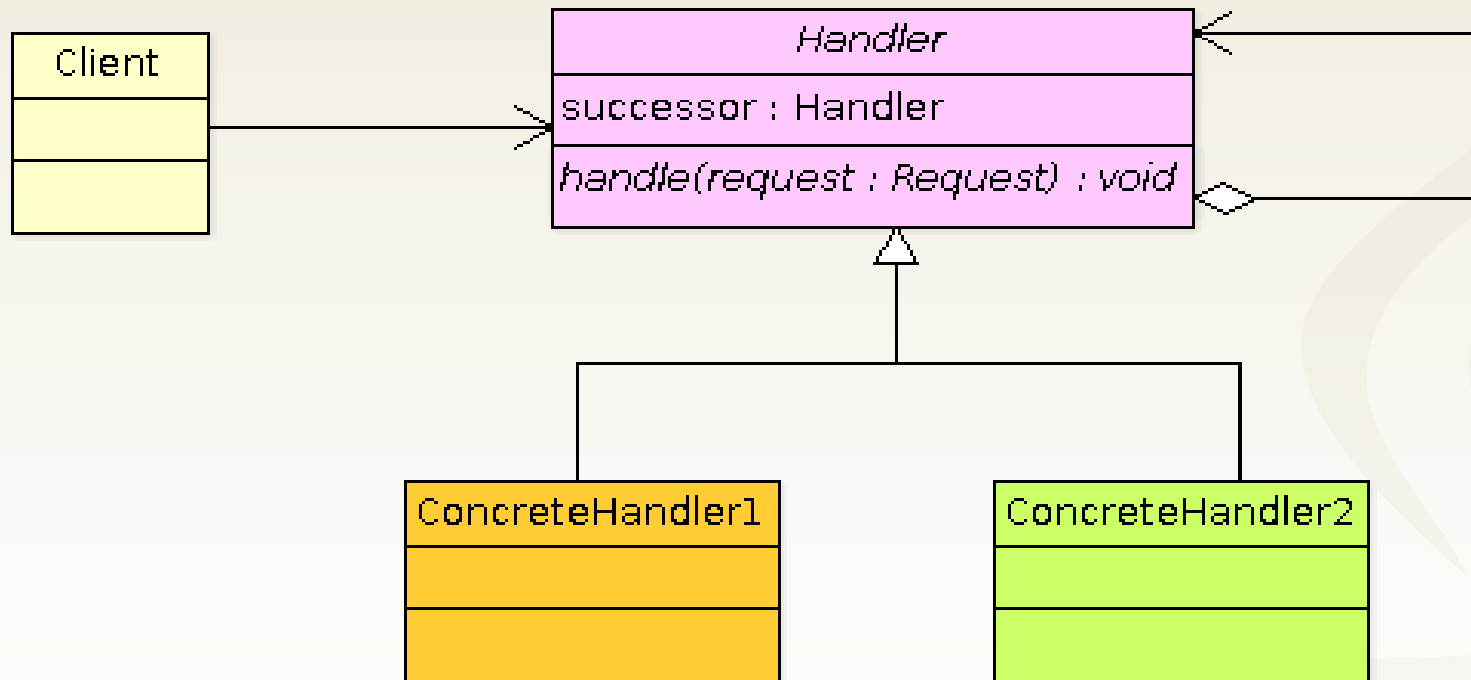
Chain of Responsibility

- **İsteğin birden fazla nesneye gönderilmesi** gerektiği, fakat hangi nesnenin isteği ele alacağının bilinmediği durumlar olabilir
- **İsteği ele alabilecek nesneler** dinamik olarak da değişebilir
- Böyle durumlarda isteği ele alabilecek nesnelerden oluşan **bir nesne zinciri** yaratılır ve istek zincir üzerinde bir nesneden diğerine iletilir
- **Uygun nesne isteğe cevap verir**, isteğin birden fazla nesne tarafından ele alınması da mümkündür

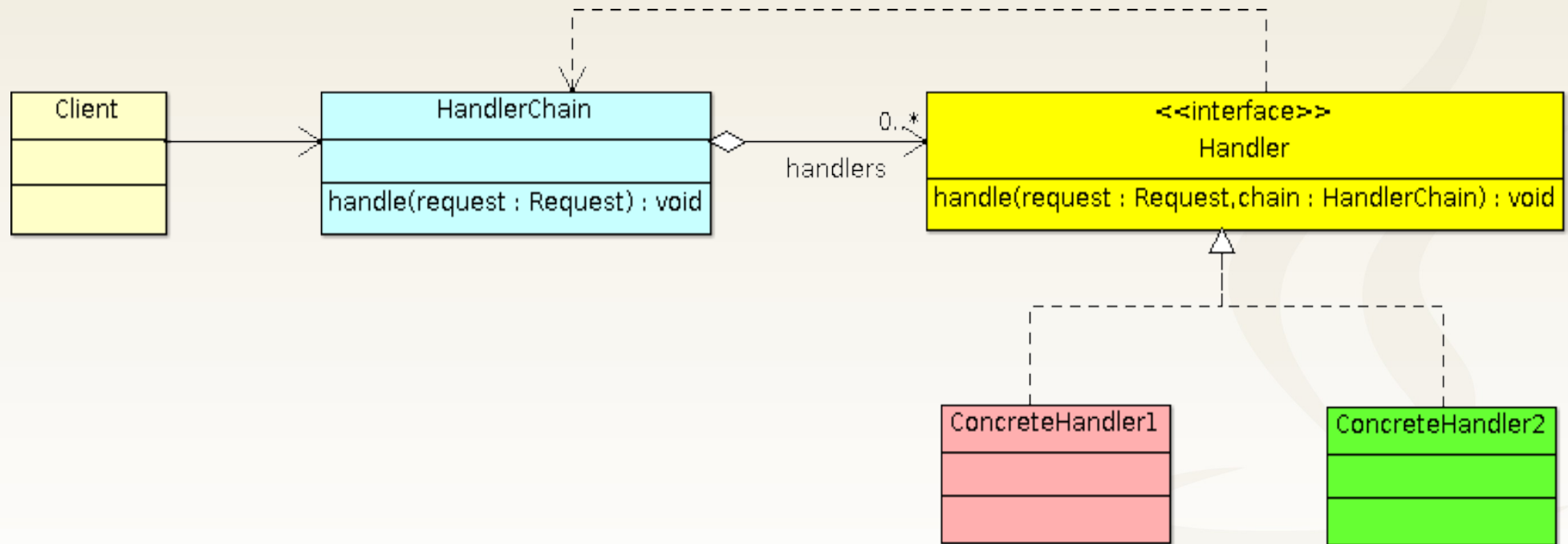
Chain of Responsibility



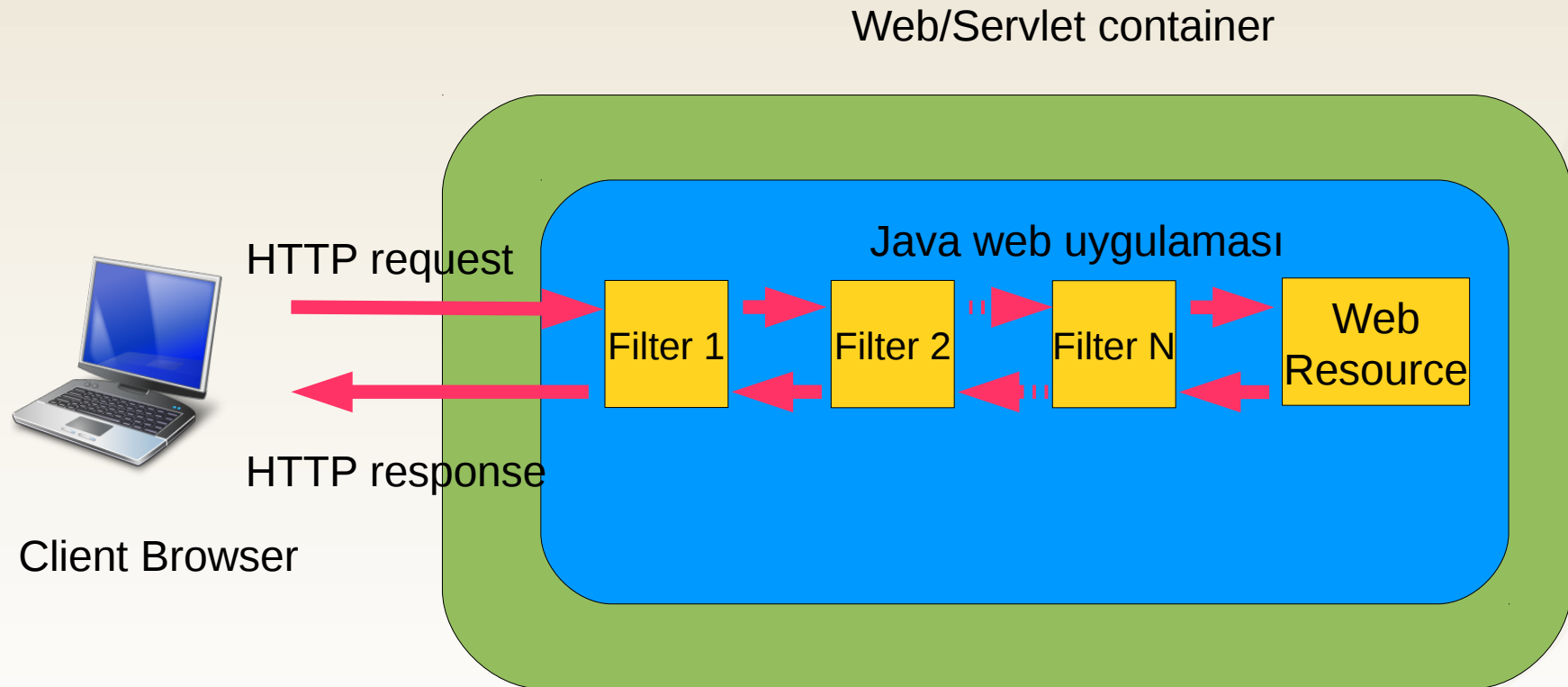
Chain of Responsibility Sınıf Diagramı



Chain of Responsibility Sınıf Diagramı 2



Chain of Responsibility ve Servlet Filter'lar



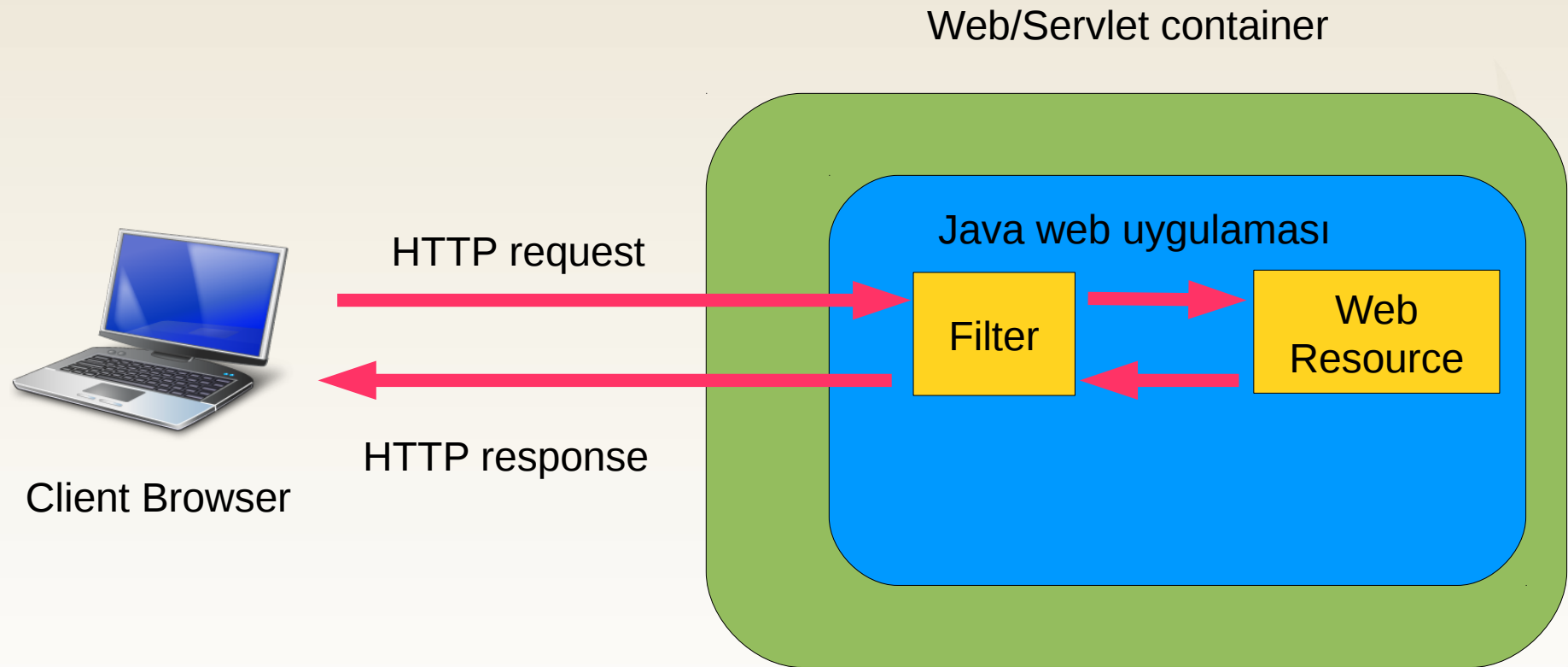
Request'in web resource'a erişmesinden önce birden çok Filter request'i intercept edebilir

Response'da Filter zincirinde tersten sıra ile bu filter'lardan geçerek dönülecektir

Chain of Responsibility Örüntüsünün Sonuçları

- İsteğe cevap verecek **nesnelerin birbirlerini tanımaları** gerekmez
- Nesneler arasındaki ilişki daha basittir, **sadece bir sonraki nesne** bilinir
- **Sorumluluklar** zincirdeki **farklı nesnelere** dağıtılmaktadır

Servlet Filter

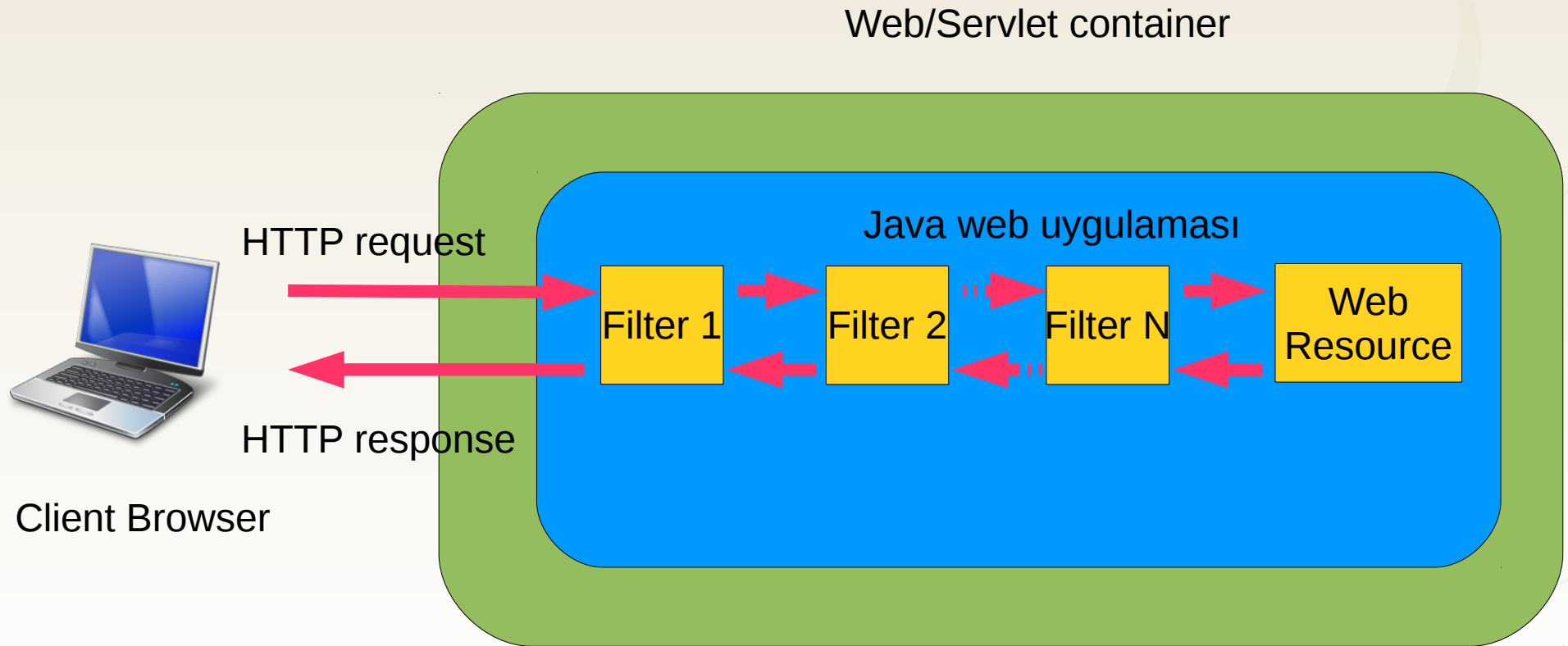


Filter instance'ı web resource'a gelen her türlü request'i ve response'u intercept eder

Web resource herhangi bir servlet, jsp veya statik html sayfası olabilir

Filter oluşturmak için **javax.servlet.Filter** arayüzünü implement etmek gerekir

Filter Zinciri



Request'in web resource'a erişmesinden önce birden çok Filter request'i intercept edebilir

Response'da Filter zincirinde tersten sıra ile bu filter'lardan geçerek dönülecektir

Örnek Filter

```
public class Filter1 implements Filter {

    public void init(FilterConfig filterConfig) throws
ServletException {
        //Filter instance'inin destroy islemi gerçekleştirilir
    }

    public void doFilter(ServletRequest request, ServletResponse
response,
FilterChain chain) throws IOException, ServletException {

        System.out.println("Web resource'a erismeden once...");

        chain.doFilter(request, response);

        System.out.println("Web resource'a existikten sonra");
    }

    public void destroy() {
        //Filter instance'inin destroy islemi gerçekleştirilir
    }
}
```

Web.xml'de Filter Konfigürasyonu

```
<web-app>
  <filter>
    <filter-name>Filter1</filter-name>
    <filter-class>x.y.z.Filter1</filter-class>
  </filter>

  <filter>
    <filter-name>Filter2</filter-name>
    <filter-class>x.y.z.Filter2</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>Filter1</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>Filter2</filter-name>
    <servlet-name>HelloWorldServlet</servlet-name>
  </filter-mapping>
</web-app>
```

Filter'lar mapping sıralamasına göre devreye girerler

Filter'ların belirli bir url-pattern'da Veya spesifik bir servlet öncesinde Devreye girmeleri sağlanabilir

@WebFilter ile Filter Konfigürasyonu

```
@WebFilter(filterName="filter1")  
public class Filter1 implements Filter {  
    ...  
}
```

```
@WebFilter(filterName="filter2")  
public class Filter2 implements Filter {  
    ...  
}
```

```
<web-app>  
  <filter-mapping>  
    <filter-name>filter2</filter-name>  
    <url-pattern>/*</url-pattern>  
  </filter-mapping>  
  
  <filter-mapping>  
    <filter-name>filter1</filter-name>  
    <servlet-name>HelloServlet</servlet-name>  
  </filter-mapping>  
</web-app>
```

WebFilter anotasyonu ile Filter tanımını web.xml'de yapmaya gerek kalmaz

Filter'ların hangi url'lerde devreye gireceği, init parametreleri vb burada tanımlanabilir

Ancak filter'lar arasındaki sıralama için web.xml'de filter-mapping tanımlarına ihtiyaç vardır

ServletContext Nedir?

- HttpSession'da tutulan attribute'lar sadece **tek bir kullanıcı tarafından** erişilebilir
- ServletContext ise **uygulama genelinde bilgi tutmayı** sağlayan global bir **veri yapısıdır**
- ServletContext nesnesine **HttpServletRequest** aracılığı ile erişilebilir
- ServletContext'de tutulan attribute'lar ise **bütün kullanıcılar tarafından** erişilebilir

ServletContextListener Ne İşe Yarar?

- Web uygulamasının **servlet context'inde meydana gelen değişikliklerden** haberdar edilirler
- Değişikliklerden haberdar olabilmesi için **web.xml içerisinde listener sınıfın** tanımlanması gerekir

ServletContextListener Örneği

```
public class MyContextListener implements ServletContextListener {  
  
    public void contextInitialized(ServletContextEvent sce) {  
    }  
  
    public void contextDestroyed(ServletContextEvent sce) {  
    }  
  
}
```

Web uygulamasının initialization
Sürecinin başladığını haber verir

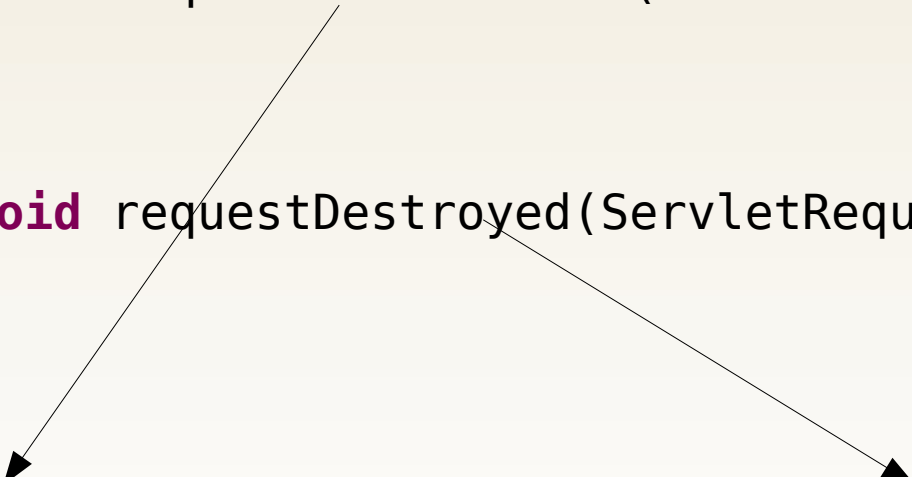
Bu metot çağrıldığında henüz herhangi bir
Servlet veya filter init edilmemiştir

Web uygulamasının ServletContext'inin
Sonlandırıldığını haber verir

Bu metot çağrılmadan evvel bütün
Servlet ve filter instance'ları destroy
edilmiştir

ServletRequestListener Örneği

```
public class MyRequestListener implements ServletRequestListener {  
    public void requestInitialized(ServletRequestEvent sce) {  
    }  
    public void requestDestroyed(ServletRequestEvent sce) {  
    }  
}
```



Her web request'i sırasında devreye girer
HttpServletRequest'in initialize edildiğini
bildirir. Event içerisinden current request'e
erişilebilir

Web request'i sonlandığı vakit çağrılır

Web.xml'de ServletContext ve RequestListener Konfigürasyonları

```
<web-app>
```


```
  <listener>
```

```
    <listener-class>x.y.z.MyContextListener</listener-class>  
  </listener>
```

```
  <listener>
```

```
    <listener-class>x.y.z.MyRequestListener</listener-class>  
  </listener>
```

```
</web-app>
```



ServletContextListener ve ServletRequestListener Tanımlarıda web.xml içerisinde listener elemanları ile ayrı ayrı yapılmalıdır

@WebListener ile Listener Konfigürasyonları

ServletRequestListener, ServletContextListener, HttpSessionListener tanımları web.xml yerine WebListener anotasyonu ile yapılabilir



@WebListener

```
public class MyRequestListener implements ServletRequestListener {  
  
    @Override  
    public void requestInitialized(ServletRequestEvent sre) {  
        System.out.println("request initialized");  
    }  
  
    @Override  
    public void requestDestroyed(ServletRequestEvent sre) {  
        System.out.println("request destroyed");  
    }  
}
```

Java Server Pages (JSP) Nedir?

- JSP öncesinde **HTML sayfa içeriği** Servlet içerisinde oluşturulmaktaydı
- Eğer uygulamanın **HTML layout'unda bir değişiklik** yapmak gerekirse doğrudan Java kodunun değiştirilmesi gerekiyordu
- Uygulamanın iş mantığını oluşturan kod ile web sayfalarının layout'unu oluşturan **kod iç içe** girmekteydi

Problem :Servlet İçinden HTML İçerik Üretilmesi

```
public void service(ServletRequest request,  
                    ServletResponse response){  
  
    PrintWriter writer = ((HttpServletResponse)  
response).getWriter();  
  
    writer.write("<html>");  
    writer.write("<body>");  
    writer.write("<table>");  
  
    for(int i=0; i<10; i++){  
        writer.write("<tr><td>");  
        writer.write("" + i);  
        writer.write("</td></tr>");  
    }  
  
    writer.write("</table>");  
    writer.write("</body>");  
    writer.write("</html>");  
}
```

Çözüm :Java Server Pages (JSP)

- JSP ile HTML içeriğin Java kodu içerisinde yazılması **tersine** döndü
- HTML içerik **normal HTML sayfaları** oluşturulur gibi JSP sayfalarında yazılmaya başlandı
- Java kodu ise **kod parçacıkları (scriptlet) şeklinde JSP sayfasının içerisinde** yazılabilir hale geldi

Sonuç :JSP İçinde Java Kodu

```
<html>
  <body>
    <table>
      <%
        for(int i=0; i<10; i++){
          <tr>
            <td><%=i%></td>
          </tr>
        <%
      %>
    </table>
  </body>
</html>
```

Ancak HTML içerik ile iş mantığı ile ilgili java kodunun iç içe geçmesi problemi ortadan Kalmamış oldu.

Problem sadece bir noktadan alınıp başka bir noktaya taşındı.

JSP Nasıl Çalışır?

JSP sayfaları Servlet sınıflarına dönüştürülerek Çalıştırılırlar

Bu işlem ilk request'de de gerçekleşebilir

Cevap dönme süresini kısaltmak için Uygulamanın startup aşamasında da yapılabilir



Client Browser

HTTP request: **hello.jsp**

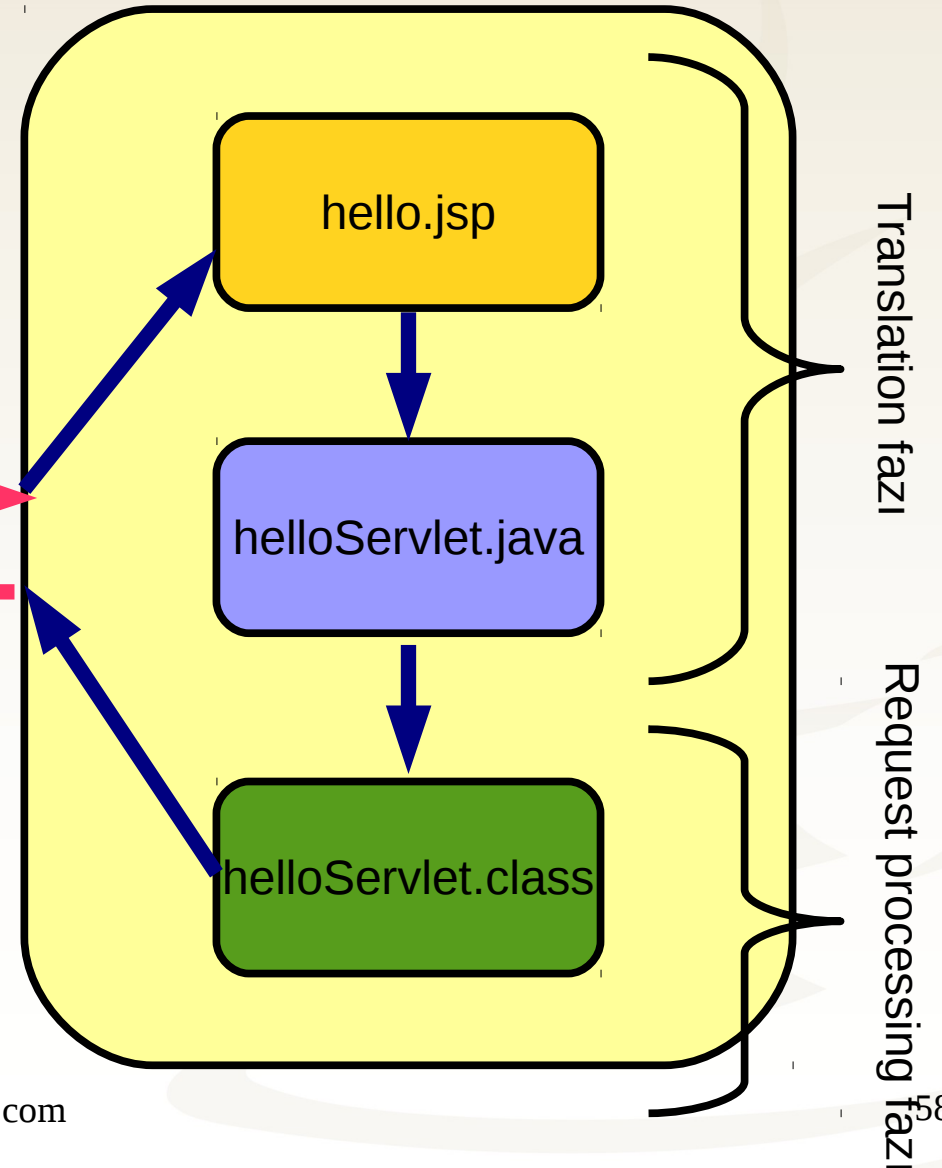
HTTP response:

```
<html>
<body>
Hello World!
</body>
</html>
```

JSP sayfasında yapılacak değişiklikler Dinamik olarak yansıtılmaktadır

www.java-egitimleri.com

Web/JSP Container



JSP Örneği

```
<%@page import="java.util.Date"%>
<%!
    String message = "Hello World";
    Date now = new Date();
%>
<html>
    <!-- bu bir commenttir--%>
    <%@ include file="header.jsp" %>
    <body>
        Session :<%=session.getId()%><br/>
        <%
            out.println(message);
            out.println("Current time :" + now);
        %>
        <table>
            <%
                for(int i=0; i<10; i++){
            %>
            <tr>
                <td><%= (3+9)%> - <%=i%></td>
            </tr>
            <%
                }
            %>
        </table>
    </body>
</html>
```

JSP Örneğinin Anatomisi

`<%@page import="java.util.Date"%>` → JSP page directive

`<%! String message = "Hello World"; Date now = new Date();%>`

↘ JSP declaration

`<html>`

`<%-- bu bir commenttir--%>` → JSP comment

`<%@ include file="header.jsp" %>` → JSP include directive

`<body>`

Session : `<%=session.getId()%>``
`

`<%`

`out.println(message);`
`out.println("Current time :" + now);`

`%>`

`<table>`

`<%`

`for(int i=0; i<10; i++){`

`%>`

`<tr>`

`<td><%= (3+9) %> - <%=i%></td>`

`</tr>`

`<%`

`}`

`%>`

`</table>`

`</body>`

`</html>`

JSP scriptlet
Herhangi bir java ifadesi olabilir
Java ifadelerinin sayısında sınır yoktur

↘ JSP implicit objects

JSP expression
Java ifadesi String'e çevrilir

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com



harezmi
bilişim çözümleri

JAVA
Eğitimleri 