BAB 2

Servlets

2.1 Pendahuluan

2.1.1 Definisi

Servlet adalah sebuah class dalam bahasa pemgrograman Java yang digunakan untuk meningkatkan kapabilitas dari server sebagai host dari aplikasi yang diakses melalui request-response programming model(Diadaptasi dari tutorial J2EE). Servlet adalah sebuah class java yang meng-implement interface Servlet dan menerima request yang berasal dari class Java, web client, atau servlet lain yang membangkitkan response.

"Servlet" juga dipanggil sebagai HTTP Servlet. Hal ini disebakan karena servlets biasanya digunakan dengan HTTP, akan tetap servlet bukanlah merupakan salah satu spesifikasi spesifik dari protokol client-server.

Untuk memulai pembuatan servlet. Anda diharapkan mengerti mengenai pemrograman, konsep client-server, dasar-dasar HTML dan HTTP (HyperTextTransferProtocol). Untuk menciptakan sebuah servlet, Anda perlu untuk meng-import standard extension class dari javax.servlet dan javax.servlet.http ke program java Anda. Javax.servlet berisi framework dasar servlet dimana javax.servlet.http digunakan sebagai ekstensi dari framework servlet bagi servlet yang akan menjawab HTTP request.

2.1.2 Gambaran Arsitektur Servlet

Sebelum Servlet, salah satu dari beberapa cara yang umum digunakan untuk menambahkan sebuah fungsi kedalam web server adalah melalui Common Gateway Interface (CGI). CGI menyediakan sebuah interface ke program eksternal bagi sebuah server, dimana program tersebut akan dipanggil oleh server untuk menangani client request. Bagaimanapun, CGI telah didesain khusus sehingga tiap pemanggilan dari resource CGI akan menciptakan proses yang baru pada server; informasi yang dibutuhkan oleh program akan diberikan kepada proses ini dengan menggunakan standard input dan environment variable. Sekali request terpenuhi, maka proses tersebut akan dimatikan, dan akan kembali kepada resource didalam system. Permasalahan yang muncul pada skenario tersebut adalah proses ini sangat membebani server karena dibutuhkan persyaratan yang cukup banyak pada sistem resource, dan juga membatasi berapa banyak user yang dapat ditangani oleh aplikasi pada waktu yang sama.

Servlet memiliki desain tertentu yang dapat mengatasi segala permasalahan yang melekat pada CGI dan memberikan sebuah solusi java kepada developer untuk membuat sebuah aplikasi berbasis web. Selain menciptakan sebuah proses yang tidak terlalu membebani server pada saat kedatangan setiap request dari client, dengan menggunakan servlet hanya ada satu proses yang akan menangangi SELURUH request: proses tersebut memerlukan servlet container untuk dijalankan. Pada saat kedatangan request yang baru, container akan menciptakan sebuah thread ringan untuk mengeksekusi servlet.

Servlet juga hanya dimasukkan sekali kedalam memori, baik container yang memasukkannya kedalam memori pada saat server mulai bekerja, maupun pada saat pertama servlet dibutuhkan untuk memberikan service kepada client. Sekali servlet dimasukkan kedalam memori, ia akan bertahan didalam memori, dan siap untuk menangani request yang lain dari client. Hal ini tidaklah sama dengan CGI dimana tiap request dari client akan dimasukkan dan dikeluarkan ke dan dari memori.

Perwujudan dari Servlet pertama

Contoh kode dibawah ini menunjukkan baaimana struktur dasar dari sebuah servlet yang menangani request GET, untuk mencetak „Hello World“

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

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

    //"request" digunakan untuk membaca HTTP readers yang datang
    // Form data HTML (contoh data di-enter dan di-submit oleh user )
    // sedangkan data yang lain dapat dimasukkan dari request client

    // "response" adalah sebuah HTTP response line yang spesifik
    // headers(contoh menspesifikasi content type dan men-setting cookies ).
    // Ia juga memiliki sebuah method yang memberikan ijin kepada servlet untuk
//membangkitkan response kepada client

    PrintWriter out = response.getWriter();
    out.println("<HTML> <TITLE>Hello Page</TITLE><BODY><br>");
    out.println("<h1>Hello World!</h1>");
    out.println("</BODY></HTML>");

    //"out" untuk mengirim content kepada browser
  }
}
```

Gambar 2-1 : Servlet dasar yang mencetak Hello World

Bagian pertama dari kode diatas pada gambar 2-1 adalah meng-import class di java.io (bagi PrintWriter, dsb), javax.servlet dan javax.servlet.http. Javax.servlet dan javax.servlet.http adalah package-package yang menyediakan interfaces dan class untuk membuat sebuah servlet(untuk HttpServlet, HttpServletRequest dan HttpServletResponse).

Dengan cara meng-extend HttpServlet, class ini akan meng-inherit method yang akan secara otomatis akan dipanggil oleh server tergantung pada kondisi-kondisi tertentu (akan dibicarakan kemudian). Dengan meng-override method ini, kita dapat membuat servlet kita memiliki fungsi-fungsi yang kita inginkan.

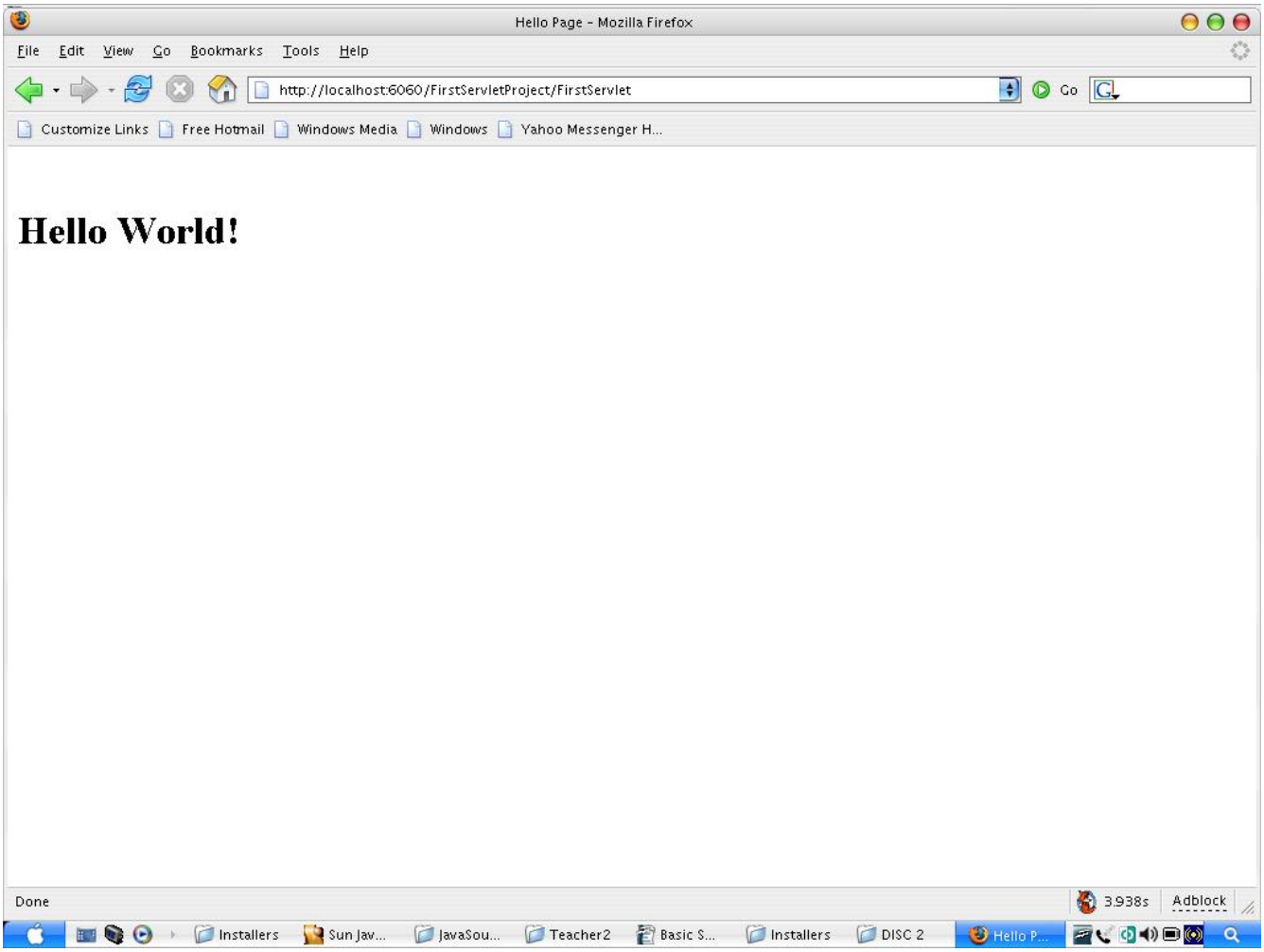
Pada kasus ini, method yang diturunkan dari HTTPServlet dan akan kita override adalah method doGet. Untuk menyederhanakan, ia adalah sebuah method yang akan dipanggil oleh servlet container kapanpun pada saat Get request dipanggil pada servlet tertentu. Ingatlah, pada bab sebelumnya mengenai site navigation, document retrieval, dan page viewing adalah contoh-contoh dari GET request. Oleh karena itu, kapan saja user ingin untuk melihat output dari servlet, sebuah GET request pasti dipanggil.

Apabila kita melihat pada listing code, kita akan dapat melihat bahwa doGet method akan menerima dua parameter yaitu: HttpServletRequest object dan HttpServletResponse object. Darimana object ini berasal, tidak perlu diperhatikan lagi oleh developer. Mereka akan diciptakan dan di-maintain oleh servlet container dan bekerja sederhana bagi kita pada saat container memanggil method doGet. Pada kasus ini, method doGet(dan method-method lain yang akan kita temukan kemudian) hampir sama dengan method public static void main (String[] args) yang kita gunakan pada program java berbasis command line. Kita tidak akan membuat String array untuk diberikan kepada method; hal itu telah disiapkan bagi kita oleh runtime environment.

Object dari HttpServletRequest dan HttpServletResponse telah menyiapkan fungsi-fungsi yang berguna bagi developer:

- Object HttpServletRequest memberikan akses bagi segala informasi terhadap client request, termasuk apa saja bentuk parameter value yang dapat diletakkan pada Http request header, Http request method yang telah mereka gunakan, dan sebagainya.
- Object HttpServletResponse terdiri dari semua method yang dibutuhkan oleh developer untuk memproduksi sebuah response yang akan dikirimkan kembali kepada client. Meliputi method-method yang harus di-set pada HTTP response header, untuk mendeklarasikan tipe MIME dari response, sebaik method yang digunakan untuk mengambil instance dari class Java I/O yang akan kita gunakan secara langsung untuk memproduksi output.

Kembali pada kode diatas, kita akan melihat bahwa selain comment, hanya beberapa baris saja yang kita telah gunakan dalam menciptakan sebuah fungsi yang menampilkan "Hello World!" kepada user. Pertama adalah PrintWriter out = response.getWriter() dan yang lainnya adalah beberapa pemanggilan dari out.println(). Untuk saat ini, berpikirlah secara sederhana untuk menggunakan PrintWriter sebagai object yang dapat kita gunakan untuk memberikan output text kepada user. Dengan pernyataan itu didalam benak kita, terasa sederhana untuk melihat bagaimana pemanggilan out.println() berkali-kali dapat menampilkan content berikut ini:



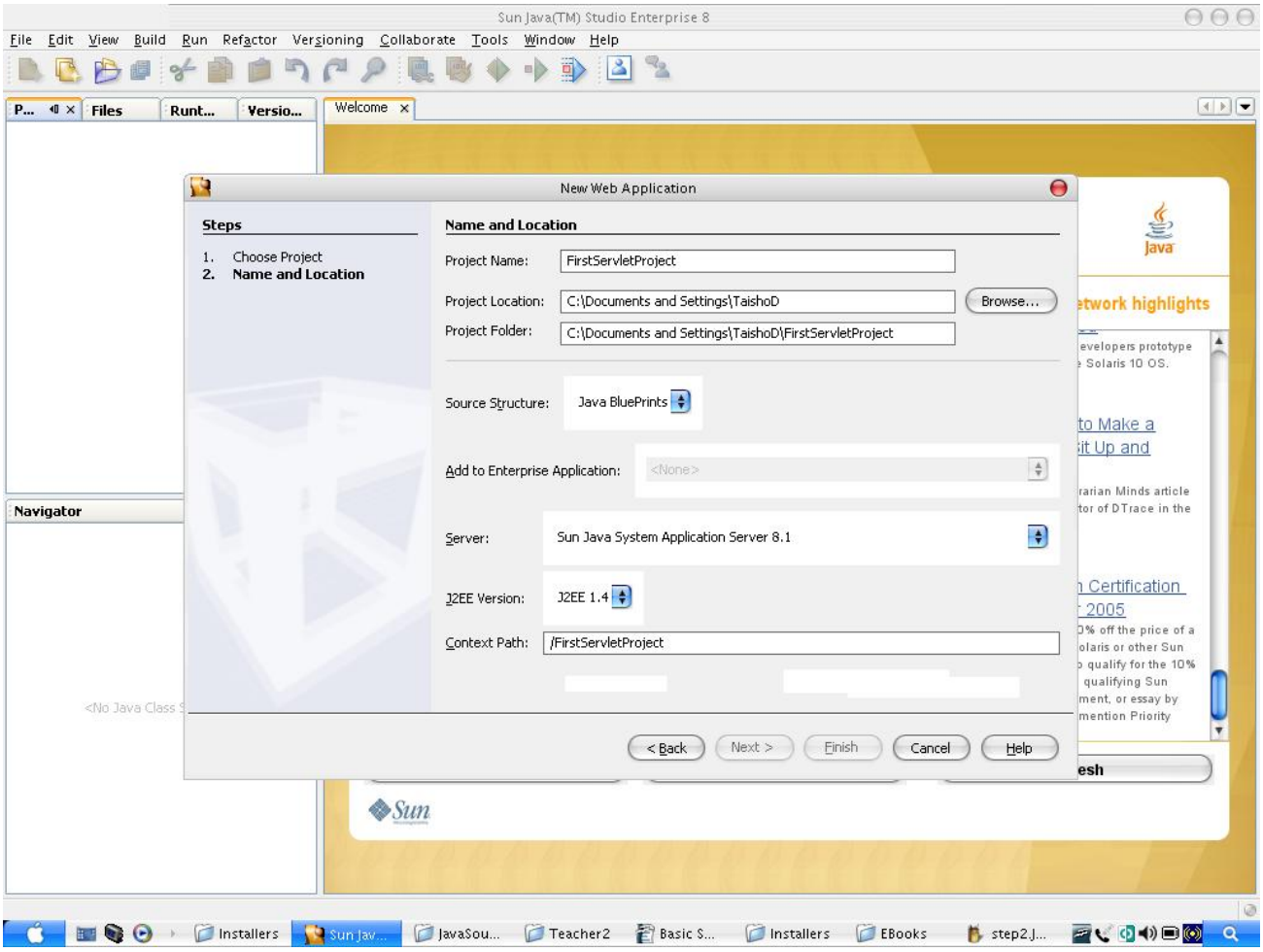
Testing sebuah contoh servlet

Pada poin ini kita diharapkan mampu untuk menampilkan output dari contoh sebuah servlet. Untuk memulai abstract detail dari pembangunan sebuah servlet dan konfigurasinya, kita akan menggunakan tool otomatis yang telah disediakan oleh IDE. IDE yang akan kita gunakan kali ini adalah Sun Studi Enterprise 8 yang akan tersedia bagi Anda secara gratis apabila Anda merupakan anggota dari Sun Developer Network. Ia memiliki dukungan lengkap bagi servlet dan spesifikasi JSP, sebaik sebuah host yang menangani fitur-fitur tambahan.

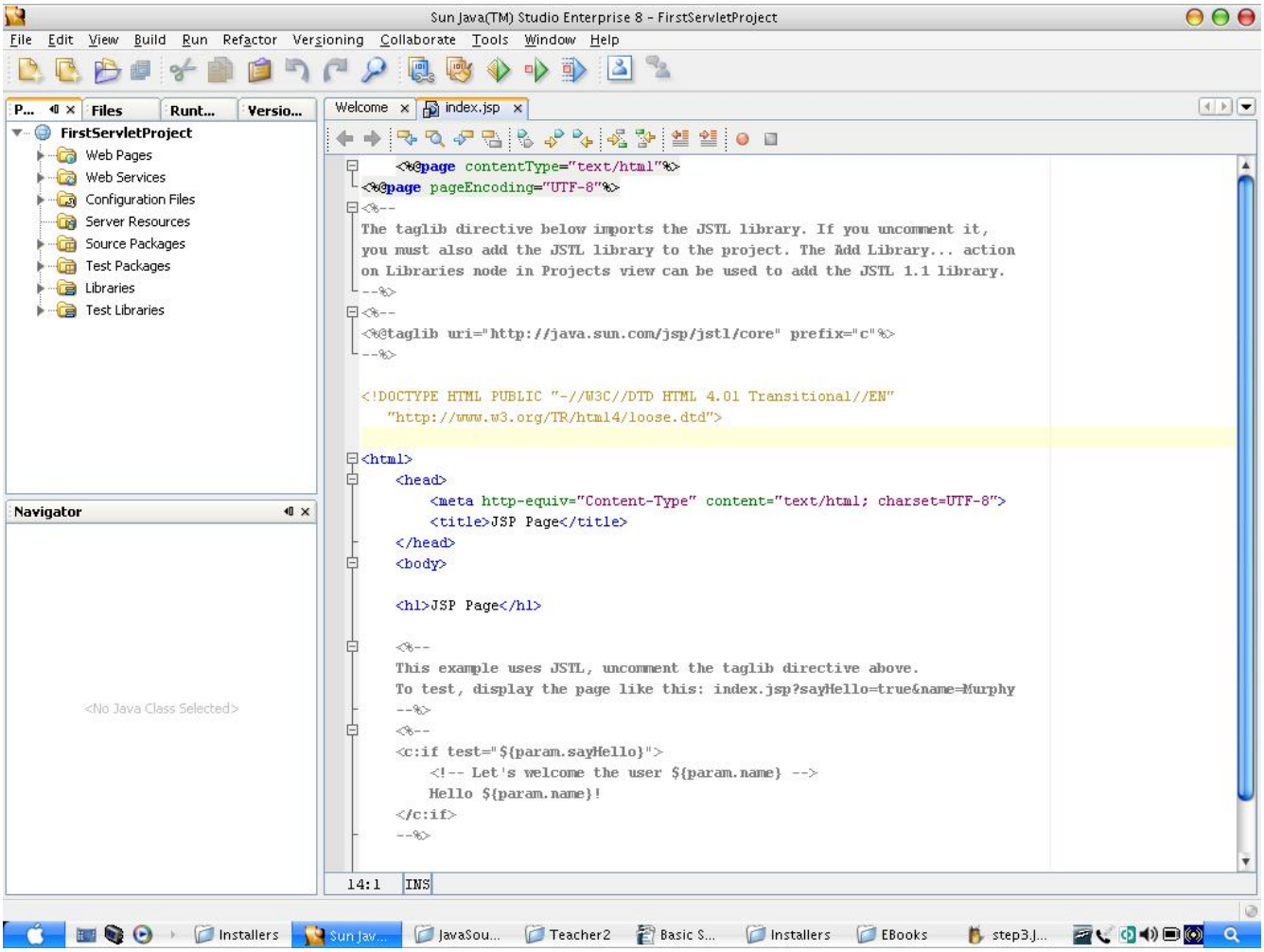
Mulai saat ini, kita akan mengasumsikan bahwa Enterprise 8 telah diinstal dengan sukses pada sistem Anda. Sebagai bantuan pada saat instalasi, Anda dapat merujuk pada bagian appendix yang ada dalam modul ini.

Hal pertama yang harus kita lakukan untuk contoh servlet kita disini adalah kita harus menciptakan project yang baru dari aplikasi web. Untuk melakukan ini, pilih New -> Project. Didalam layar yang kita lihat berikut ini, pilih kategori web. Disebelah kanan, pilih New Web Application

Layar selanjutnya akan mengharuskan Anda untuk memberikan detail dari project. Bagi test servlet pertama kita, marilah membuat "FirstServletProject" sebagai nama dari project kita, dan buatlah default value dari field-field yang lain.

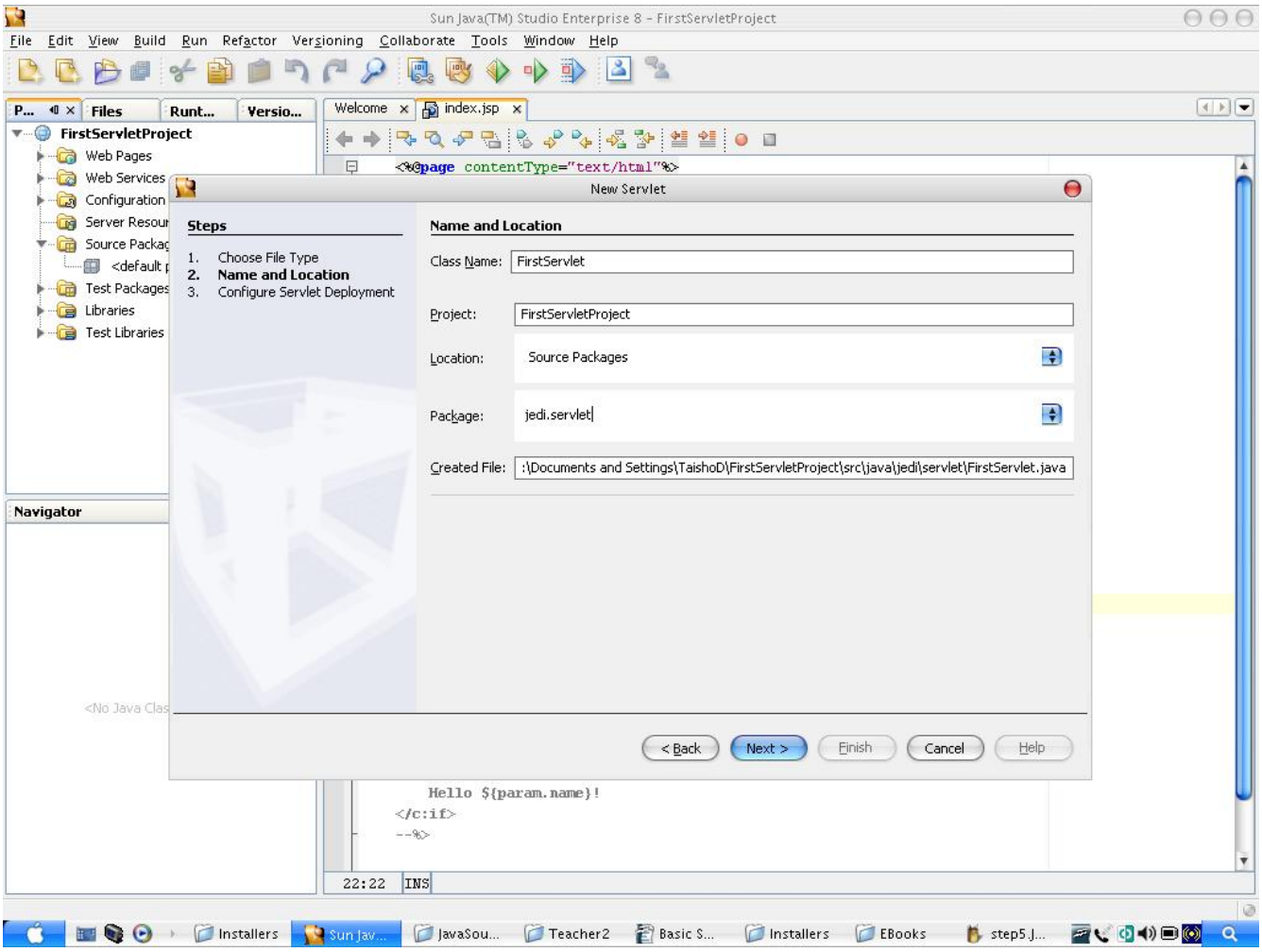


Setelah selesai membuat web project yang baru, layar akan terlihat seperti berikut:

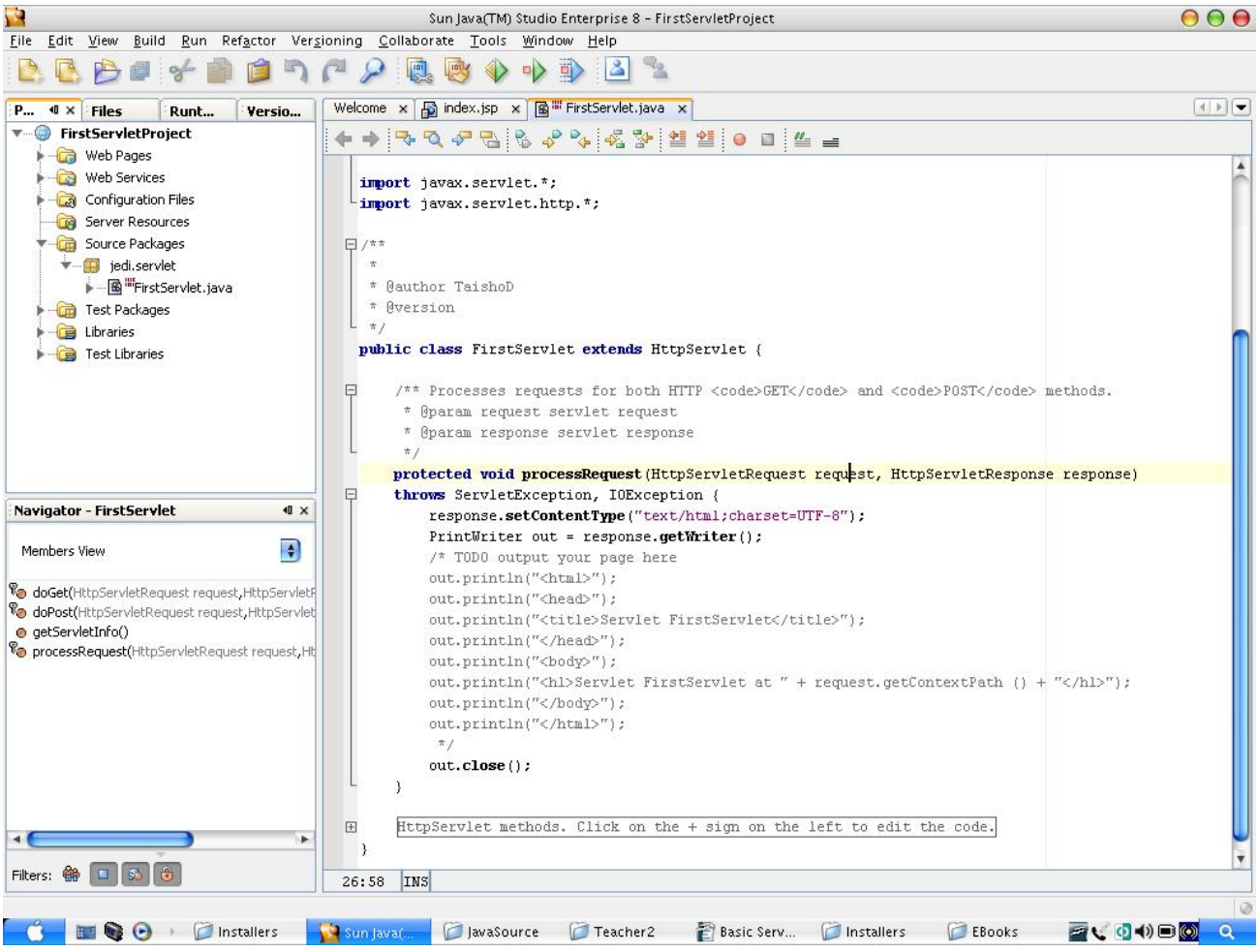


Untuk menambah servlet kita ke aplikasi, klik kanan pada Source Packages, pilih New -> Servlet. Jika Servlet tidak tampil pada menu New context, pilih juga New->File/Folder. Pada layar berikutnya, pilih web category, dan kemudian Servlet.

Kemudian IDE akan menjalankan beberapa screen yang akan menanyakan mengenai detail servlet yang telah diciptakan. Pada layar pertama, nama servlet adalah FirstServlet. Didalam package gunakan nama jedi.servlet.



Kemudian pada layar selanjutnya, biarkanlah default value seperti semua, kemudian klik tombol finish. Hal tersebut akan menghasilkan sesuatu yang akan dibangkitkan seperti pada layar berikut ini:



Dari sini, kita dapat melihat bahwa IDE yang telah kita buat sebagian akan meng-implementasi method processRequest. Jika kita akan menekan pada kotak dengan tanda tambah dibagian kiri bawah, kita akan melihat bahwa processRequest adalah sebuah method sederhana yang dapat kita panggil baik dari doGet maupun doPost. Hal ini berarti content dari method processRequest memberikan bentuk dasar bagi fungsionalitas dari servlet kita.

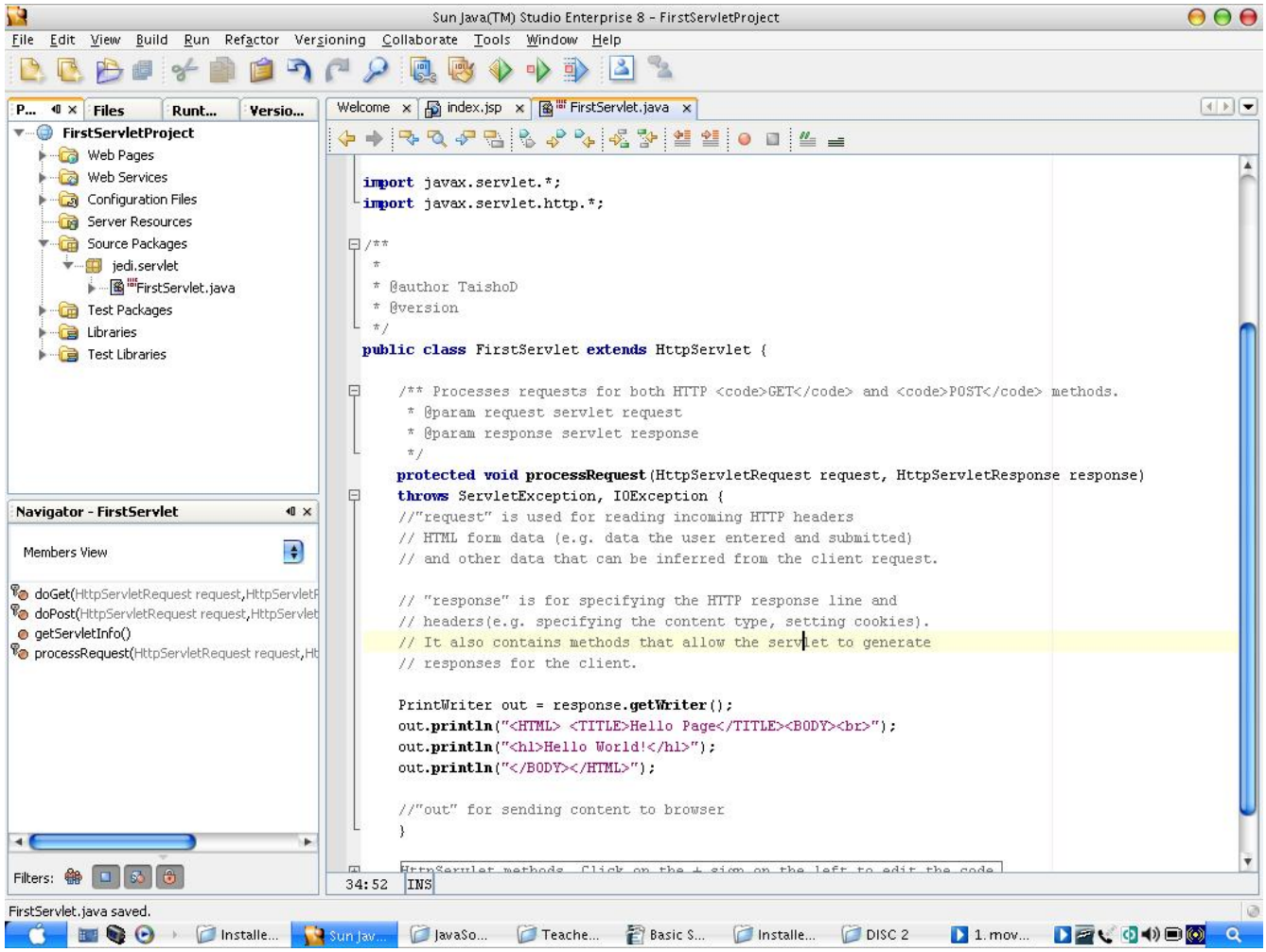
Pertama, singkirkan semua contents dari method processRequest. Kemudian copy paste content method doGet dari listing kode ke test servlet.

Untuk menjalnkan tekan shift + f6. IDE kemudian akan mem-package, deploy, dan invoke servlet secara otomatis bagi kita, untuk menghasilkan output.

2.1.3 Servlet Lifecycle

Sebuah servlet telah diatur melalui sebuah lifecycle yang yang telah dideskripsikan dalam spesifikasi servlet. Servlet lifecycle mendeskripsikan bagaimana servlet di-load, di-instantiate, di-inisialisasi, di-request service-nya, di-destroy, dan yang terakhir bagaimana gerbage collection dijalankan. Lifecycle dari servlet dikontrol oleh sebuah kontainer yang telah dibangun oleh servlet.

Lifecycle dari servlet akan memberikan kesempatan bagi servlet engine untuk mengenali permasalahan dalam hal performance dan CGI resource dan aspek keamanan pada low-level server API program. Sebuah servlet engine mungkin dapat mengeksekusi semua servlet didalam sebuah Java virtual machine(JVM). Karena pada dasarnya mereka berada didalam JVM yang sama, servlet dapat secara efisien melakukan sharing data dengan yang lain, tetapi sekarang oleh Java mengakses private data dari yang lain haruslah dicegah. Servlet juga membuka kemungkinan untuk untuk melakukan persistance diantara request-request sebagai instance dari object, sehingga dapat menghemat memori lebih banyak daripada full-fledged process.



Gambar diatas menunjukkan event utama didalam sebuah kehidupan servlet. Sangatlah penting untuk dicatat bahwa dari setiap event pastilah ada sebuah method pada servlet yang akan di-invoke oleh container. Marilah kita lihat satu demi satu:

2.1.3.1 INSTANTIATION

Pada fase ini, sebuah class servlet akan diload kedalam memori dan instance-nya akan diciptakan oleh servlet container.

Secara default, sebuah servlet container practice disebut lazy loading. Dengan menggunakan method ini, sebuah servlet class akan di-load kedalam memori, di-instantiate, dan diinisialisasi hanya setelah sebuah request telah dibuat untuknya. Hal ini menjadikan waktu startup menjadi lebih cepat bagi aplikasi, tetapi hal ini juga berarti bahwa ada sedikit overhead yang terjadi pada saat pemanggilan pertama sebuah servlet. Apabila kondisi ini tidak diinginkan, maka tiap servlet dapat dikonfigurasikan untuk di-load pada startup server atau aplikasi. Lebih banyak tentang hal ini, akan kita diskusikan lagi pada descriptor bagi application's deployment.

Seperti yang dapat kita lihat pada diagram, sebuah servlet dapat melewati fase instantiation sekali per lifetime. Hal ini berarti bahwa hubungan overhead dengan loading pendefinisian servlet class kedalam memori hanya terjadi sekali. Hal ini menunjukkan keuntungan servlet dibandingkan teknologi yang lain.

Method yang relevan adalah servlet konstruktor tanpa argument. Akan tetapi, tidak direkomendasikan untuk meletakkan kesuluruhan kode didalam sebuah konstruktor. Hampir semua fungsi yang developers inginkan untuk ditambahkan kepada konstruktor termsuk object pada saat set-up, atau inisialisasi value. Servlet membagi kedalam fase terpisah bagi aktifitas tersebut.

2.1.3.2 INITIALIZATION

Pada fase ini servlet digunakan terutama didalam sebuah aplikasi. Seperti pada masa instantiation, servlet hanya berada pada fase ini sekali. Hanya setelah fase ini, instance dari object yang telah kita buat dapat dimulai dipanggil sebagai servlet.

Method yang dipanggil oleh container pada masa ini adalah init() method. Ia merupakan sebuah method signature yang lengkap dan merepresentasikan bahwa servlet sangat mudah untuk dipanggil kembali.

```
public void init(ServletConfig config)
```

Seperti dapat kita lihat, method akan mengambil sebuah parameter sebagai instance dari sebuah object ServletConfig. Object ini adalah object yang bertanggung jawab terhadap pemenuhan status dari servlet. Object ini memiliki informasi mengenai konfigurasi dari servlet, ia juga menyediakan sebuah cara bagi servlet untuk mengakses informasi secara luas dari aplikasi dan memfasilitasi untuk menggunakan object dari ServletContext.

Seperti disebutkan sebelumnya bahwa konfigutasi atau inisialisasi kode manapun haruslah tidak diletakkan dalam konstruktor servlet akan tetapi sebaiknya diletakkan dalam method init. Jika seorang developer ingin menggunakan method ini, sangatlah penting untuk memanggil super.init(config).Untuk memastikan bahwa inisialisasi default dari servlet action tidak tertinggal dan konfigurasi juga telah di-set up

Setelah inisialisasi, servlet akan ditangani oleh request dari client.

Method ini hanya akan dipanggil kembali pada saat server dimasukkan kedalam servlet. Server tidak dapat me-load sebuah servlet sampai server telah menghapus servlet dengan memanggil method destroy.

2.1.3.3 SERVICE

Fase ini adalah fase dimana sebuah servlet berada didalam lifetime-nya. Pada fase ini, servlet dapat berkali-kali dipanggil oleh container untuk menyediakan fungsionalitas-nya.

Method dibawah ini ini adalah method yang akan di-invole oleh servlet container pada fase ini:

public void service(ServletRequest req, ServletResponse res)

Object dari ServletRequest dan ServletResponse diberikan kepada sebuah method yang menyediakan method bagi developer untuk meng-extract informasi dari user request dan method untuk membangkitkan response.

Sebuah hal penting yang harus diingat adalah servlet container akan membuat pemanggilan yang berulang-ulang ini kepada method service dengan menggunakan thread yang berbeda. Oleh karena itu, pastilah hanya satu saja instance dari servlet aktif yang mengambil tempat di dalam memori untuk menangani banyak request. Hal ini adalah salah satu keuntungan yang dimiliki oleh java servlet. Hal ini juga merupakan salah satu alasan mengapa servlet dan service method-nya haruslah selalu didesain untuk menerima thread-safe.

Bagi HTTP servlet yang spesifik misalnya servlets extending HttpServlet,developers seharusnya meng-override service method secara langsung. Jika tidak developer harus meng-override method berikut ini:

public void doGet(HttpServletRequest req, HttpServletResponse)
public void doPost(HttpServletRequest req, HttpServletResponse)
public void doPut(HttpServletRequest req, HttpServletResponse)
public void doTrace(HttpServletRequest req, HttpServletResponse)

Masing-masing method ini cocok bagi HTTP method yang spesifik(GET,POST,...). Method yang tepat kemudian akan dipanggil apabila servlet menerima request dari HTTP. Karena kebanyakan dari pemanggilan HTTP yang ditangani oleh developer adalah GET atau POSt method calls, maka servlet hanya bisa meng-implement doGet,DoPost, dan keduanya.

2.1.3.4 DESTRUCTION

Ada waktu-waktu tertentu dimana sebuah servlet container akan berjalan out of memory, atau untuk mendetekasi jumlah dari memori yang tersisa dengan cara menghapus satu atau lebih instance servlet dari memori. Servlet mana yang dikeluarkan dari memori ditentukan oleh servlet container dan hal ini tidak dapat langsung dikontrol oleh developer.

Sebuah container juga akan membebaskan instance servlet pada saat container tersebut di shut down.

Pada saat sebuah servlet telah dihilangkan dari memori, hal ini akan disebut sebagai fase destruction. Method yang akan dipanggil oleh container sebelum ia selesai adalah method destroy(). Disini, servlet kita seharusnya mampu untuk secara eksplisit membebaskan resource yang harus dihandle seperti koneksi ke database dan sebagainya.

2.1.3.5 GARBAGE COLLECTION

Aturan java standard mengenai garbage collection dari object, tidak akan dijelaskan lebih lanjut disini.

Menangani requests dan responses

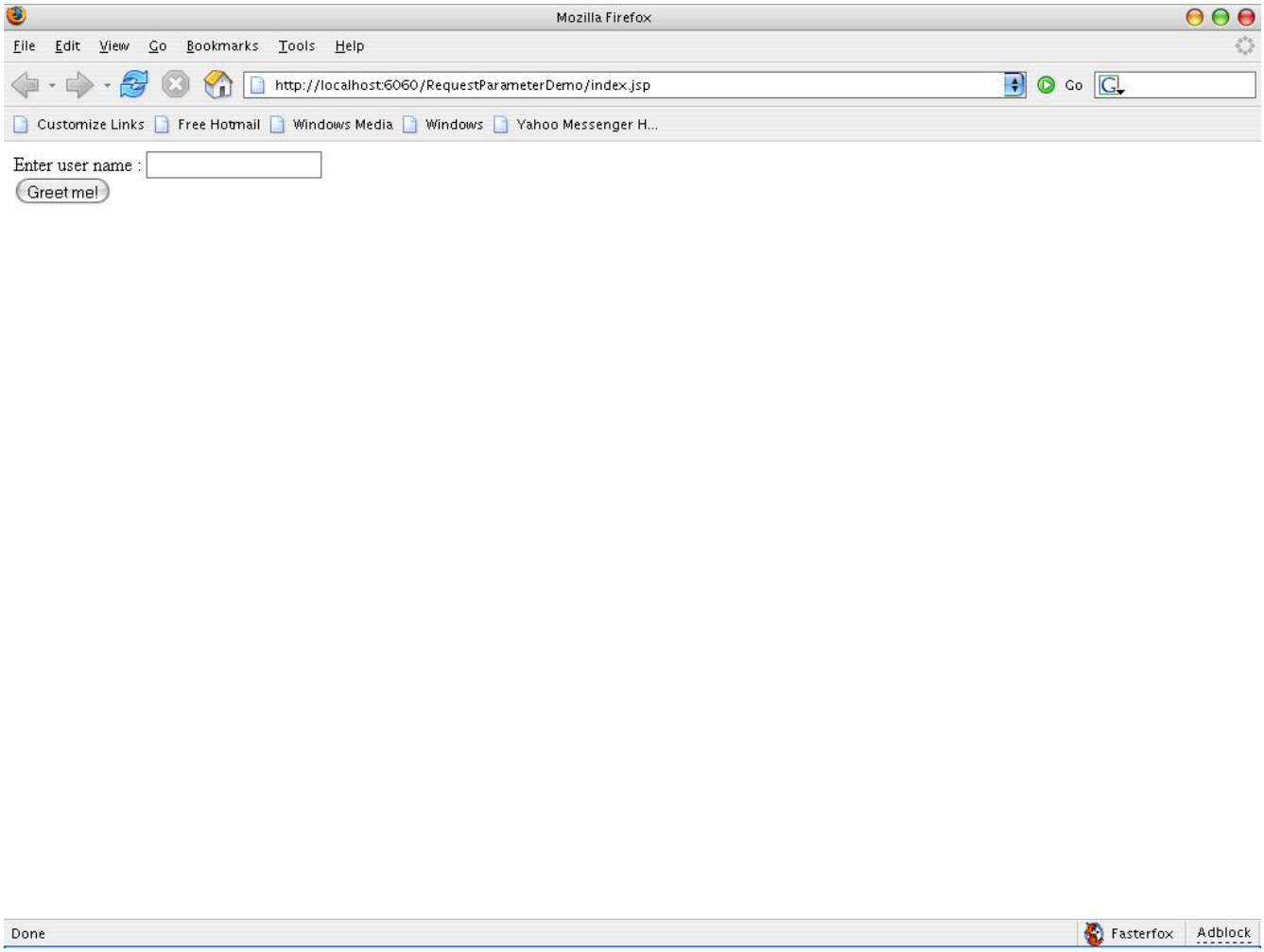
Tujuan utama dari sebuah servlet adalah untuk menyediakan sebuah content yang dinamik bagi user. Berdasarkan definisinya, dynamic content adalah sebuah content yang berubah-ubah response-nya pada kondisi yang bervariasi,sebagai contoh kondisi request dari user, waktu, dan sebagainya.

Untuk memberikan akses kepada servlet bagi user request tertentu,telah disediakan sebuah instance dari ServletRequest object yang menyembunyikan detail dari servlet berbasis HTTP diberikan pada subclass, HTTPServletRequest, yang menyediakan method tambahan untuk kembali lagi mendapatkan informasi HTTP secara spesifik seperti cookies information, detail dari header, dan sebagainya.

Form Data dan Parameter

getParameter

Salah satu skenario yang paling sering ditemukan dan membutuhkan sebuah content yang dinamik adalah apabila kita ingin aplikasi yang me-respon kepada user dipresentasikan dalam bentuk form, seperti contoh berikut ini:



Diberikan form diatas, yang mengambil nama user, dan kita ingin untuk memberikan response tertentu kepada nama user tersebut.

Pada skenario ini dan seperti skenario yang lain, Java telah menyediakan method getParameter didalam object HttpServletRequest. Method ini akan mengambil sebuah parameter String(nama dari element form dimana valuenya dapat diambil kembali) dan mengembalikan response sebuah String(sebuah value dari form elemen spesifik dari form)

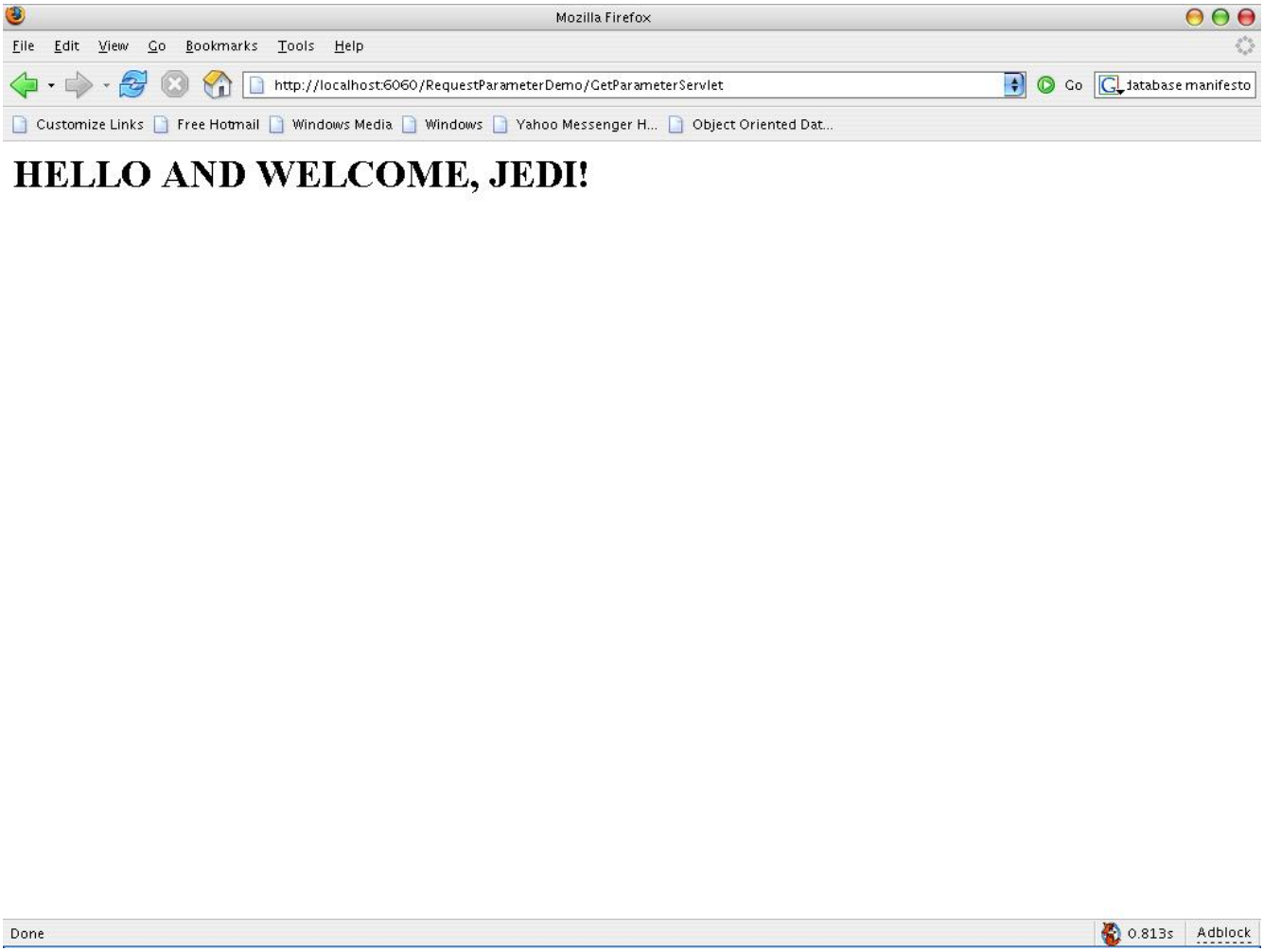
Dibawah adalah contoh untuk mendapatkan nama user dan menampilkan sebuah salam sederhana.

```
public GetParameterServlet extends HttpServlet {

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

      // dapatkan nama yang telah di-inputkan oleh user
      String userName = request.getParameter("userName");

      // dapatkan object dari PrintWriter dan gunakanlah untuk memberikan output berupa
     // salam
      PrintWriter out = response.getWriter();
      out.println("<HTML><BODY><H1>");
      out.println("HELLO AND WELCOME, " + userName + "!");
      out.println("</H1></BODY></HTML>");

      out.close();
   }
}
```

```
<HTML>
   <TITLE>Halloa!</TITLE>
   <BODY>
     <form action="/GetParameterServlet" method="post">
         Enter user name : <input type="text" name="userName"/> </br>
         <input type="submit" value="Greet me!"/>
      </form>
    </BODY>
</HTML>
```

Diatas adalah sebuah HTML yang digunakan untuk menampilkan form yang akan digunakan untuk mendapatkan data dari user. Sebagai catatan value dari parameter yang diberikan kepada method getParameter haruslah cocok dengan nama dari text field.

Berikut ini adalah output dari kode pada saat kita memberikan nama JEDI pada form sebelumnya.



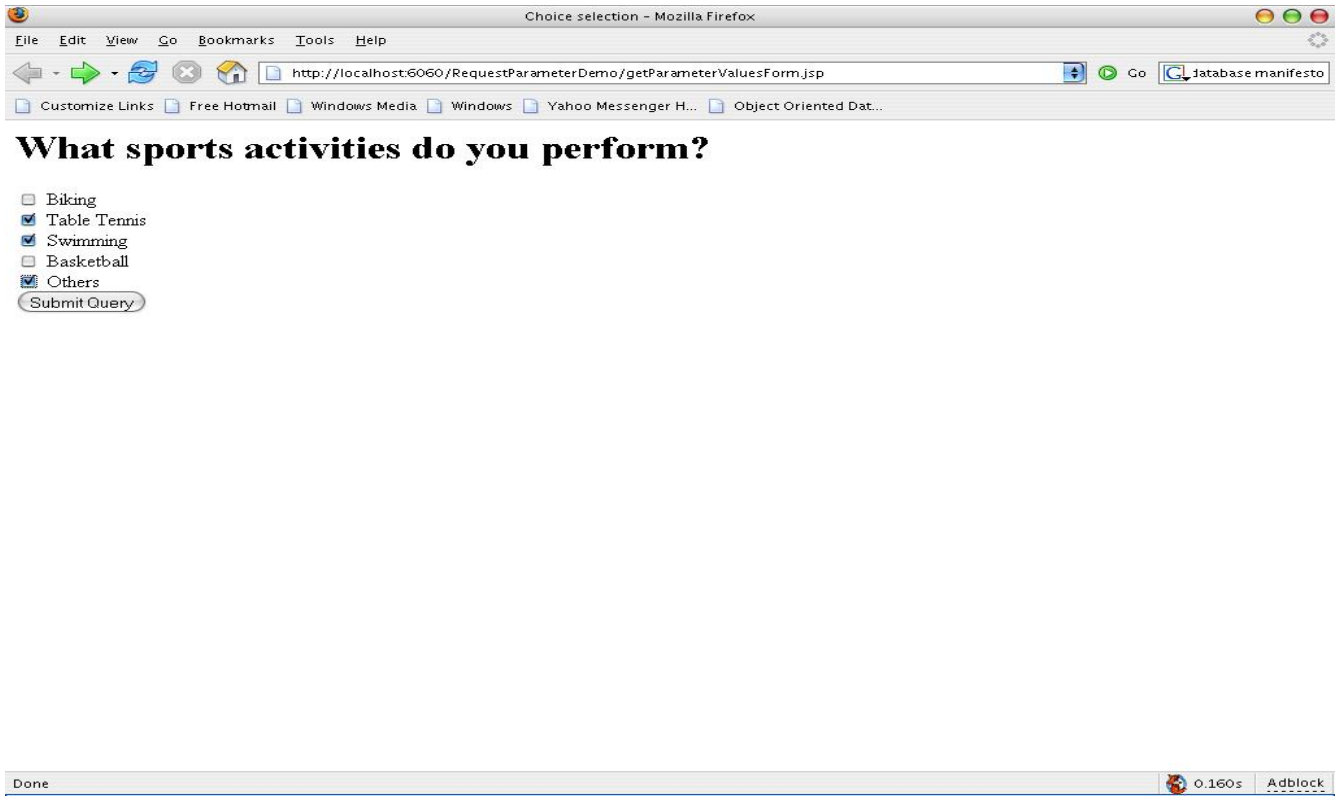
getParameterValues

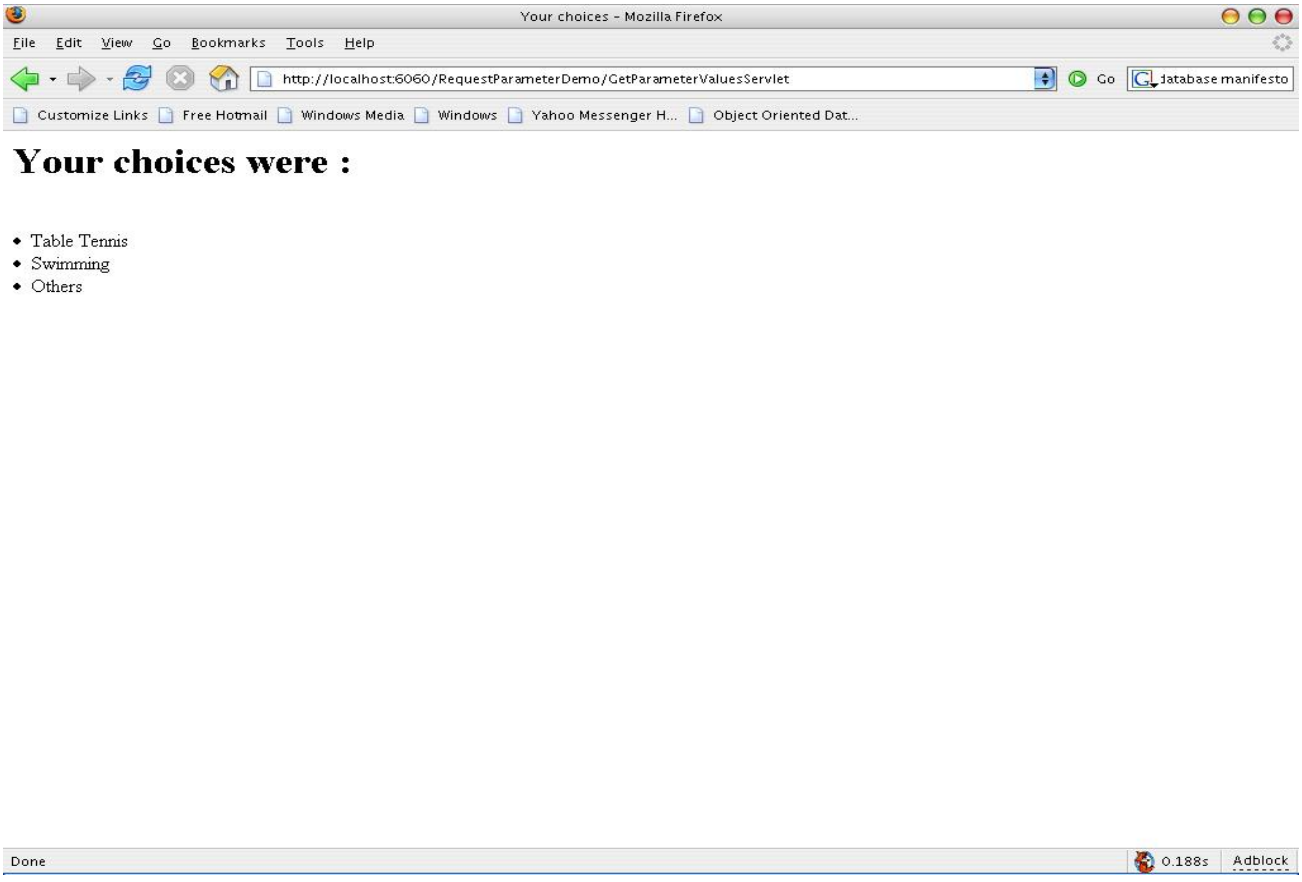
Ada saat-saat dimana kita menginginkan untuk mengambil kembali data form dari sebuah form pada multiple element dengan nama yang sama. Pada kasus ini, gunakanlah getParameter method yang telah disebutkan sebelumnya dan akan mengembalikan sebuah value dari element pertama dari nama yang telah diberikan. Untuk mendapatkan kembali semua value, kita akan menggunakan method getParameterValues. Sekali lagi, method ini akan mengambil dalam bentuk String sebagai nama dari element form, akan tetapi, kali ini, akan mempunyai nilai kembalian String array.

Berikut ini adalah contoh, sampai mengeluarkan output.

```
public GetParameterValuesServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletExceptionm IOException{
      String paramValues[] = request.getParameterValues("sports");
      StringBuffer myResponse = new StringBuffer();
      myResponse.append("<HTML><TITLE>Your choices</TITLE>");
      myResponse.append("<BODY><H1>Your choices were : </H1>");

      for (int i = 0; i < paramValues.length; i++) {
         myResponse.append("<br/><li>");
         myResponse.append(paramValues[i]);
      }

      PrintWriter out = response.getWriter();
      out.println(myResponse.toString());
   }
}
```

```
<html>
  <title>Choice selection</title><body>
   <H1>What sports activities do you perform?</h1>
   <form action="GetParameterValuesServlet" method="post">
 <input type="checkbox" name="sports" value="Biking"> Biking <br/>
<input type="checkbox" name="sports" value="Table Tennis"> Table Tennis <br/>
<input type="checkbox" name="sports" value="Swimming"> Swimming <br/>
      <input type="checkbox" name="sports" value="Basketball"> Basketball <br/>
      <input type="checkbox" name="sports" value="Others"> Others <br/>
      <input type="submit">
   </form></body>
</html>
```


Choice selection - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:6060/RequestParameterDemo/getParameterValuesForm.jsp
Go
database manifesto
Customize Links Free Hotmail Windows Media Windows Yahoo Messenger H... Object Oriented Dat...
What sports activities do you perform?
Biking
Table Tennis
Swimming
Basketball
Others
Submit Query
Done
0.160s
Adblock


Your choices - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:6060/RequestParameterDemo/GetParameterValuesServlet
Go
database manifesto
Customize Links Free Hotmail Windows Media Windows Yahoo Messenger H... Object Oriented Dat...
Your choices were :
Table Tennis
Swimming
Others
Done
0.188s
Adblock

getParameterNames

Ada saat-saat dimana kita menginginkan servlet kita untuk berhati-hati terhadap nama dari satu atau lebih paramater didalam form, tanpa harus menuliskan hardcode mereka kedalam servlet. Pada kasus ini, kita akan menggunakan method getParameterNames. Ia akan mengembalikan sebuah enumeration dari nama yang berasal dari element-element form yang telah digabungkan pada saat user me-request.

Mengambil kembali informasi dari sebuah URL request

Ingatlah, dari HTTP kita, bahwa sebuah URL request tersusun dari bagian-bagian seperti beriku ini:

http://[host]:[port]/[requestPath]?[queryString]

Kita dapat memanggil kembali current value masing-masing bagian dari user request dengan memanggil method pada object HttpServletRequest

- **Host** – request.getServerName()
- **Port** – request.getServerPort()
- **Request Path –** Di Java,path yang di-request dibagi menjadi dua komponen logical yaitu
  - **Context –** context dari aplikasi web. Dapat dipanggil kembali dengan cara meng-invoke request.getContextPath()
  - **Path info –** sisa dari request setelah context name. Dapat dipanggil kembali dengan meng-invoke request.getPathInfo()
  - **Query String –** request.getQueryString()

Oleh karena itu,sebagai contoh, dengan sebuah request URL dari
http://www.myjedi.net:8080/HelloApp/greetUser?name=Jedi

Tabel berikut ini merepresentasikan hasil apabila kita memanggil method tersebut :

request.getServerName()	***www.myjedi.net***
request.getServerPort()	***8080***
request.getContextPath()	***HelloApp***
request.getPathInfo()	***greetUser***
request.getQueryString()	***name=Jedi***

Informasi pada Header

Informasi pada header bisa didapat dari dalam servlet dengan memanggil method-method berikut ini dari HTTPServletRequest:

- **getHeader(String name)** – mengembalikan value dari spesifik header sebagai String. Jika spesifik header tidak ada, maka akan memiliki nilai kembalian null .
- **getHeaders(String name) –** hampir sama dengan getHeader(), hanya jika ia mengembalikan semua value pada spesifik header sebagai object enumeration.
- **GetHeaderNames() -** method ini akan mengembalikan sebuah object Enumeration dari semua header termasuk HTTP request.
- **getIntHeader(String name) –** mengembalikan value dari spesifik header sebagai int. Jika spesifik header tidak tersedia, method ini akan memiliki nilai kembalian -1.
- **getDateHeader(String name) –** mengembalikan value dari spesifik header sebagai sebuah long value yang merepresentasikan sebuah object Date. Jika header tidak dikonversikan menjadi sebuah Date, method ini akan mendapat IllegalArgumentException.

2.1.4 OUTPUT GENERATION

Pada contoh-contoh sebelumnya, kita dapat membangkitkan sebuah output dinamik bagi user. Kita telah melakukannya dengan menggunakan method object HttpServletResponse.

Lebih jauh, kita telah menggunakan sebagian besar method getWriter. Method ini mengembalikan object PrintWriter yang dihubungkan dengan response kita ke user. Untuk membantu meletakkan hal ini dalam perspektif yang tepat, kita harus mengingat bahwa object System.out yang secara teratur kita gunakan untuk isi keluaran ke console juga instance dari object PrintWriter. Sebagian besar mereka berkelakuan pada cara yang sama : jika Anda mempunyai instance dari object PrintWriter, secara sederhana memanggil pada print yang disediakan atau method printIn yang disediakan output.

Kode bagi servlet pertama kita adalah disalin kembali di sini untuk pemanggilan kembali, dengan kode keluaran dalam huruf bercetak tebal.

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

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

    PrintWriter out = response.getWriter();
    out.println("<HTML> <TITLE>Hello Page</TITLE><BODY><br>");
    out.println("<h1>Hello World!</h1>");
    out.println("</BODY></HTML>");

  }
}
```

Method khusus lainnya dalam object HttpServletResponse adalah :

setContentType – ini menginformasikan browser client dari tipe MIME dari output mengenai penerimaan. Semua isi yang telah kita generate sehingga telah menjadi HTML. Kita dapat secara mudah mengirim tipe isi yang lain : JPEG, PDF, DOC, XLS, dsb. Untuk output bukan teks, method print dalam object PrintWriter yang telah kita gunakan tidak cukup. Untuk me-generate outpun bukan teks, kita menggunakan method yang lain.

GetOutputStream – method ini menerima instance dari object OutputStream yang dihubungkan dengan response kita ke user. Menggunakan OutputStream, kita dapat menggunakan object dan method Java I/O standar untuk menghasilkan semua jenis output.

Di bawah ini merupakan contoh kode yang akan mengeluarkan file JPG dalam aplikasi web ke user.

```
public JPEGOutputServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        byte bufferArray[] = new byte[1024];
        ServletContext ctxt = getServletContext();
        response.setContentType("image/gif");
        ServletOutputStream os = response.getOutputStream();

  InputStream is = ctxt.getResource("/WEB-INF/images/logo.gif").openStream();
        int read = is.read(bufferArray);
        while (read != -1) {
            os.write(bufferArray);
            read = is.read(bufferArray);
        }
        is.close();
        os.close();
    }
}
```

Dalam semua contoh yang telah kita lakukah sejauh ini, kita telah menggunakan tool yang dibawa dalam Enterprise IDE untuk meringkas detail konfigurasi, packaging, dan pengembangan aplikasi web. Kita akan melihat rinciannya sekarang.

2.1.4.1 Konfigurasi Aplikasi Web

Spesifikasi servlet mendefinisikan sebuah file XML bernama web.xml yang beraksi sebagai file konfigurasi untuk aplikasi web kita. File ini juga dikenal sebagai *deployment descriptor*.

```
<web-app version="2.4"
   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">
  <servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>jedi.servlet.FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/FirstServlet</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
          30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>
          index.jsp
    </welcome-file>
  </welcome-file-list>
</web-app>
```

Kode di atas adalah web.xml yang kita gunakan pada contoh FirstServlet kita. Kita akan menggunakannya sebagai titik awal dalam eksplorasi web.xml, dan menambah di atasnya.

CATAT : web.xml dari proyek aplikasi web Enterprise dapat ditemukan dengan mengembangkan tab Configuration File dalam view project.

<web-app>
Baris ini dijalankan sebagai kedua elemen root dari file konfigurasi sebaik mendeklarasikan (lewat atributnya) informasi utama untuk isi servlet untuk menjalankan file sebagai file descriptor deployment yang valid.

<servlet>
Setiap instance dari elemen ini mendefinisikan servlet yang digunakan oleh aplikasi. <servlet> memiliki node anak yaitu :

1. <servlet-name> - nama logis yang disediakan oleh developer dimana akan digunakan untuk semua referensi yang akan datang ke servlet ini.
2. <servlet-class> - persyaratan nama class dari servlet yang harus dipenuhi.
3. <load-on-startup> (Pilihan) – mempunyai entri dan nilai dari elemen yang memberitahu container yang servlet harus di-instantiate dan diinisialisasi pada startup container/aplikasi, dengan melewati aturan loading normal lazy. Nilai untuk elemen ini adalah angka dimana memerintah urutan dari loading yang dibandingkan ke servlet <load-on-startup> lainnya.

<servlet-mapping>
Setiap instance dari elemen ini mendefinisikan pemetaan URL ke servlet. Berikut ini node anak yang dimiliki :

1 <servlet-name> - nama logis dari servlet untuk dipetakan. Harus didefinisikan sebelumnya dalam file descriptor.
2 <url-pattern> - pola URL dimana servlet ini akan dipetakan. Mempunyai nilai dari /* akan membuat semua request pada aplikasi Anda secara langsung ke servlet Anda.

Pada contoh yang telah diberikan, pola url dari FirstServlet adalah /FirstServlet. Ini artinya bahwa semua URL me-request ke

http://[host]:[port]/FirstServletProject/FirstServlet
akan ditangani oleh FirstServlet.

Diambil contoh bahwa semua definisi servlet harus pertama disediakan sebelum menambah berbagai pemetaan servlet.

<session-config>
Elemen ini mendefinisikan rincian konfigurasi untuk pengaturan session. Ini akan dibicarakan dalam bab selanjutnya.

<welcome-file-list>
Elemen ini mendefinisikan komponen web yang akan secara otomatis di-load jika user memasukkan request untuk aplikasi tanpa menspesifikasi sumber yang utama. Sebagai contoh, request ke

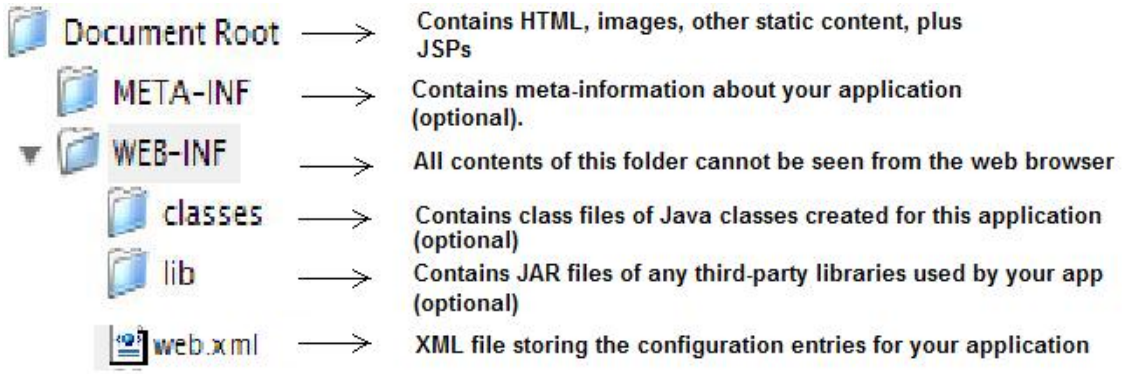
http://[host]:[port]/FirstServletProject

akan menyebabkan file yang didefinisikan di sini untuk di-load

Lebih dari satu file yang dapat ditetapkan dalam list ini. Hanya satu yang dapat muncul pertama ke container web.

2.1.4.2 Packaging aplikasi web

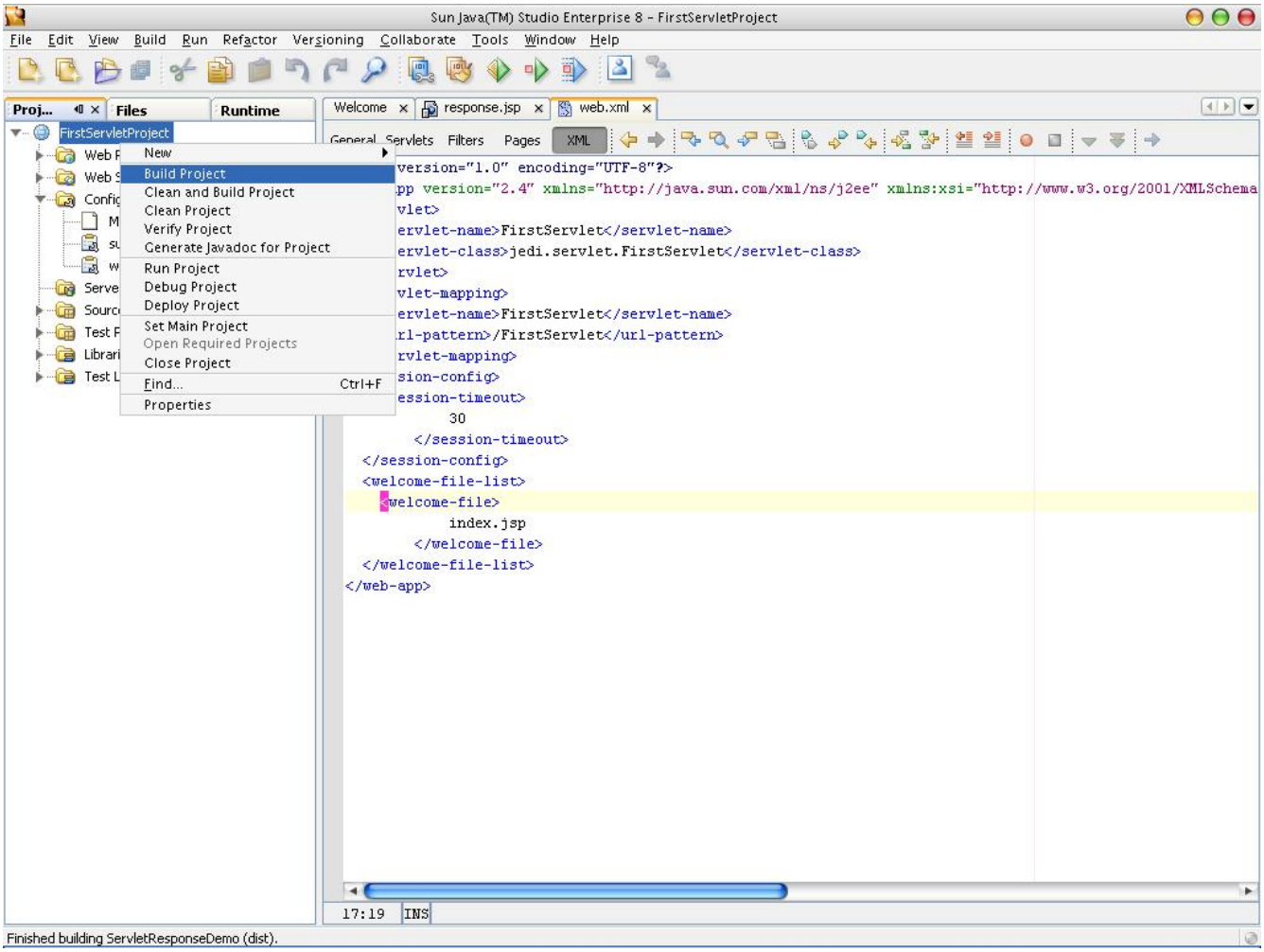
Mari kita lihat lagi pada struktur dari aplikasi web sebagai perintah oleh spesifikasi servlet :



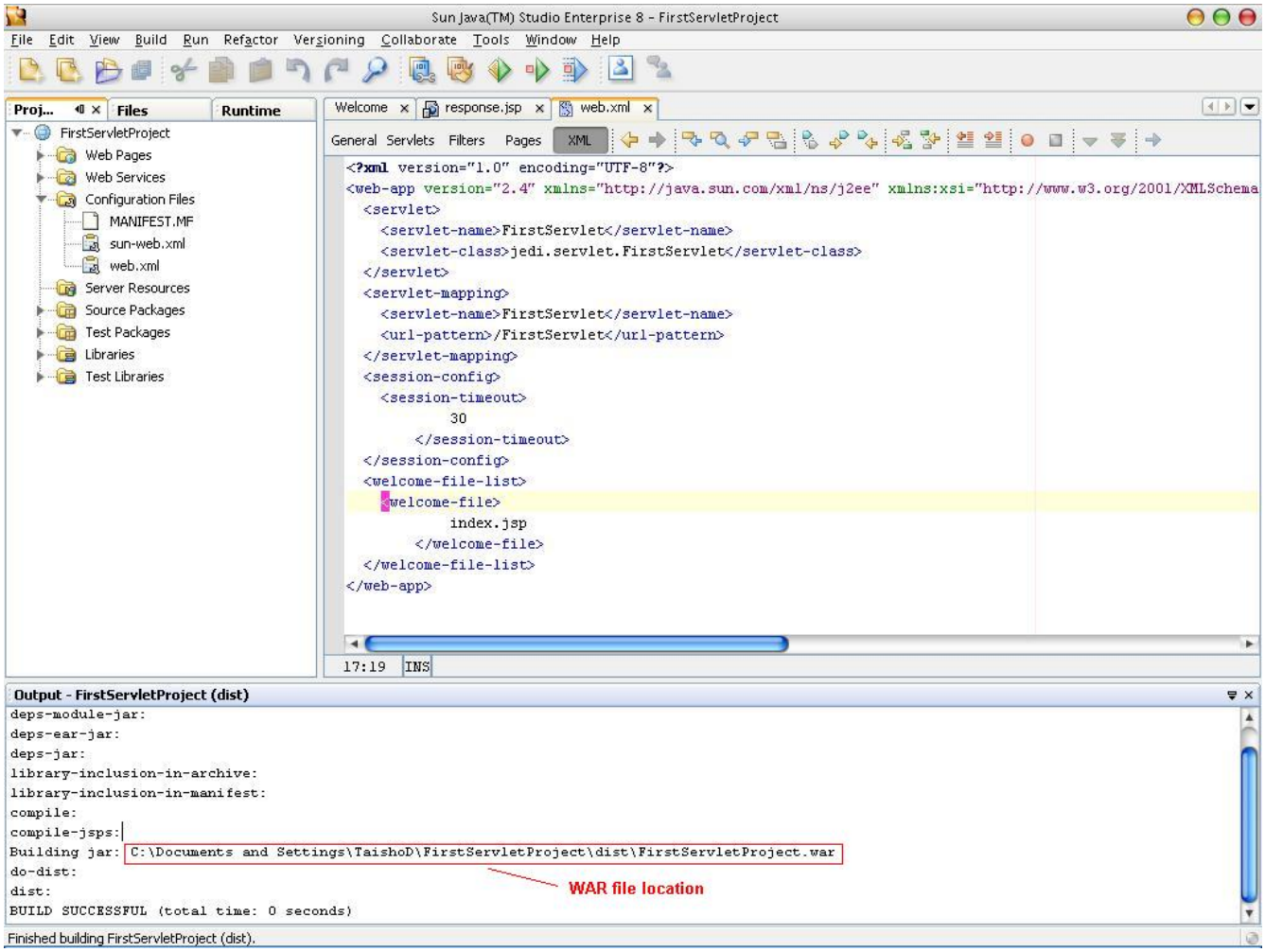
Aplikasi kita dapat dideploy ke server menggunakan struktur file ini, atau sebagai file JAR yang diubah yang dinamakan file WAR. File WAR lebih padat dan portabel; mereka dapat diinstal ke dalam berbagai container yang mengikuti spesifikasi Servlet.

2.1.4.3 Me-generate file WAR dari project Enterprise yang ada

Sangat sederhana untuk menghasilkan file WAR yang berisi aplikasi web kita dari project yang ada dalam Sun Studio Enterprise 8. Ini semudah klik kanan pada nama project dalam view Project, dan memilih Build Project. Kemudian IDE akan memproses ke package aplikasi Anda.



IDE akan menginformasikan Anda jika operasi build sukses, sebaik menginformasikan Anda lokasi dari pembuatan file WAR.



2.1.5 Pengenalan Ant

Disamping dari penggunaan IDE untuk package aplikasi web, kita juga dapat menggunakan build tool yang dapat secara otomatis bagi kita dalam proses compilation dan packaging.

Build tool yang diterima secara luas oleh industri yaitu Ant. Ant adalah project yang open source dari Apache Software Foundation, dan dapat didownload dari

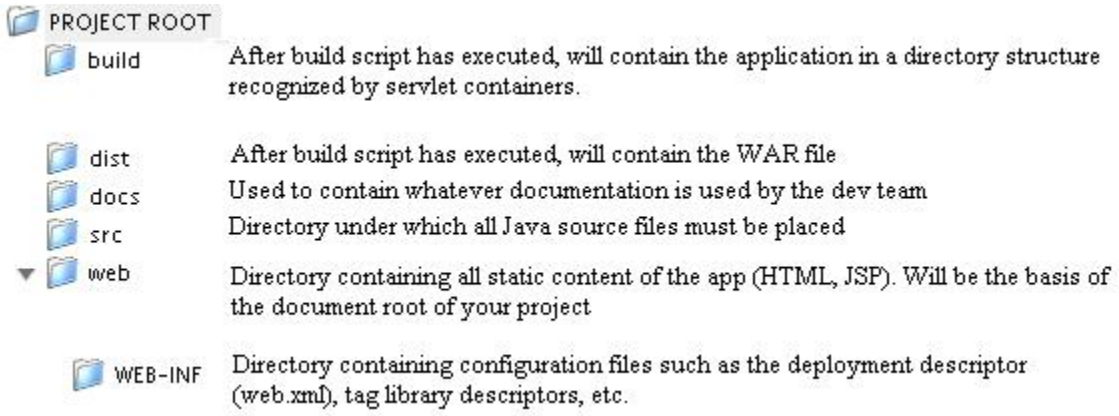
http://ant.apache.org

Pada dasarnya, Ant membaca dalam file build (secara tradisionil bernama build.xml). File build ini terbuat dari *target*, dimana pada dasarnya mendefinisikan aktifitas logis yang dapat dimuat oleh file build. Target ini disusun dari satu atau lebih task yang mendefinisikan rincian bagaimana target menjalankan aktivitasnya.

Build file dapat menjalankan task compilation dan packaging termasuk courseware dan dapat ditemukan dalam direktori *samples/blankWebApp.*
persyaratan pada build file :

1. Build file harus ditempatkan dalam struktur direktori root project yang direkomendasikan oleh Apache Software Foundation untuk pengembangan aplikasi web.
2. Sebagai tambahan, harus ada direktori lib dalam project root yang akan berisi semua dependencies JAR dari aplikasi
3. Harus ada file yang bernama build.properties dalam direktori yang sama sebagai script build dan harus berisi nilai dari properti-properti berikut :
4. app.name – nama dari aplikasi/project
5. appserver.home – direktori instalasi dari instance Sun Application Server 8.1

Berikut ini struktur direktori yang direkomendasikan untuk pengembangan aplikasi web :



Struktur direktori dibuat terpisah dari struktur direktori yang diminta oleh spesifikasi servlet. Berikut keuntungan-keuntungan mempunyai struktur direktori terpisah dari Apache :

- Isi dari direktori source lebih mudah diadministrasi, dipindah, atau di-backup jika versi deployment tidak intermixed.
- Kontrol source code lebih mudah diatur pada direktori yang berisi hanya file source (tidak ada class yang di-compile, dsb).
- File-file yang membuat distribusi mampu diinstal dari aplikasi lebih mudah untuk dipilih ketika hirarki deployment terpisah.

Sekilas hal ini mungkin sedikit susah untuk dimengerti. Untuk dapat membantu mengerti hal ini, semua persyaratan untuk compile dan package contoh FirstServlet disediakan dalam direktori *samples/FirstServlet.*

Untuk melakukan packaging dari sebuah app menggunakan struktur ini, jalankan baris perintah berikut (dalam direktori yang sama yang berisi build file)

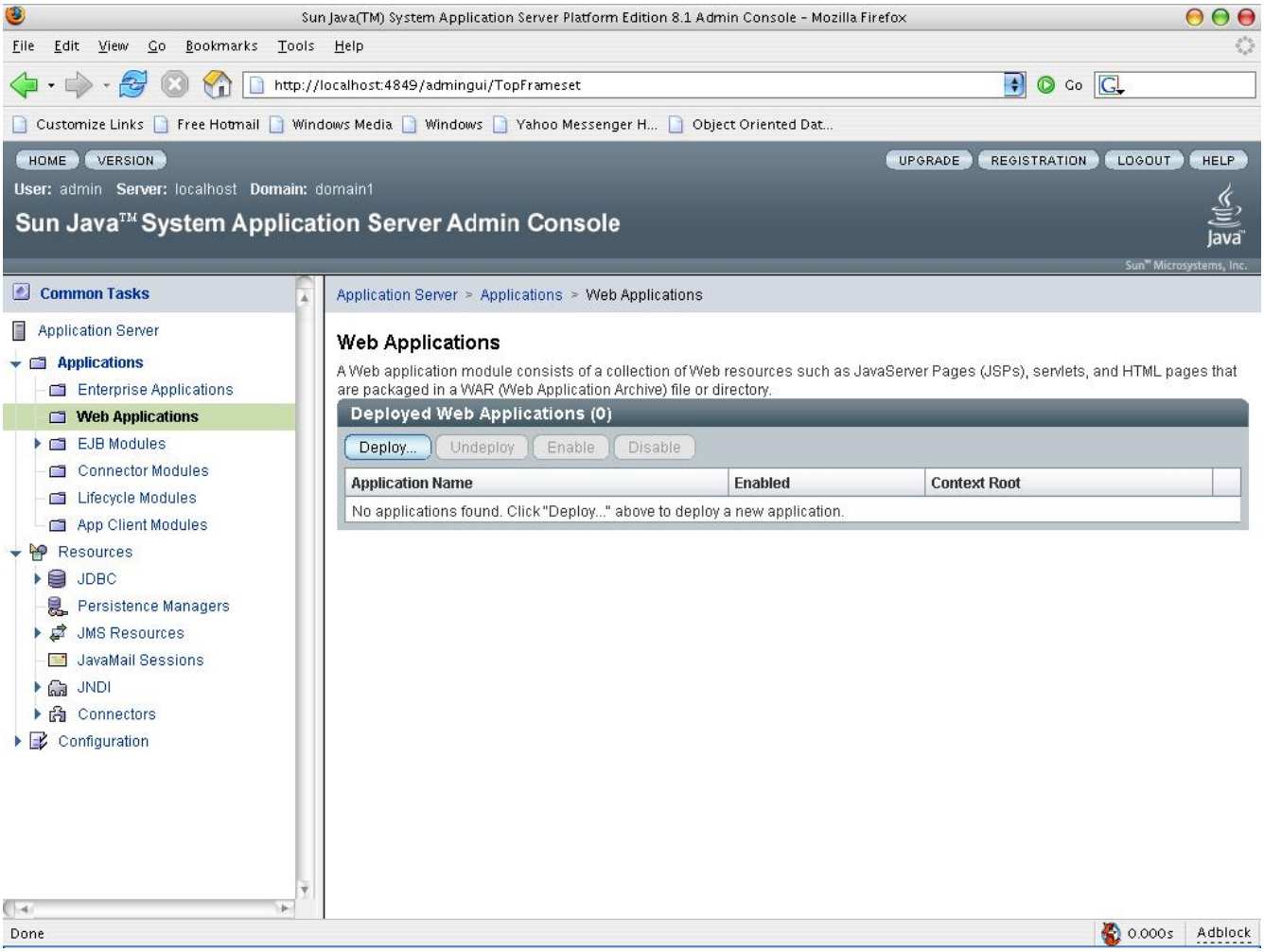
```
ant dist
```

Ini akan memanggil target dist dalam build file dimana akan me-generate file WAR dan menempatkannya ke dalam direktori dist. File WAR ini akan dapat di-generate ke dalam container target servlet menggunakan tool admin yang disediakan oleh container.

2.1.6 Deployment ke dalam server.

Container servlet biasanya berisi tool administratif yang dapat digunakan untuk men-deploy aplikasi web. Berikut kita akan mengikuti langkah yang diminta untuk men-deploy file WAR yang di-generate ke dalam Sun Application Server 8.1.

- Langkah pertama, log-in ke dalam console administratif. Ini dapat diakses dengan memasukkan URL berikut ke dalam browser Anda : http://localhost:[ADMIN_PORT] dimana ADMIN_PORT adalah port yang dikonfigurasi selama instalasi untuk menangani permintaan administratif.
- Kedua, klik kiri pada tab **Web Application** pada panel kiri, kemudian klik tombol **Deploy** yang ditemukan dalam panel kanan.



- Dalam tampilan yang selanjutnya akan muncul, klik pada tombol Browse untuk memilih file WAR yang di-upload. Klik pada tombol Next pada kanan atas.
- Klik pada tombol Finish pada tampilan berikutnya.
- Selamat, aplikasi Anda sekarang telah di-deploy.

Parameter Servlet dan Aplikasi

ServletConfig

Object ServletConfig adalah object yang dilewatkan ke servlet yang ditetapkan selama fase inisialisasi. Dengan menggunakan ini, servlet dapat menerima informasi khusus ke dirinya sendiri, seperti parameter inisialisasi. Juga, menggunakan object ServletConfig, servlet dapat memperoleh akses ke instance dari object ServletContext, dimana kita akan sedikit membicarakannya nanti.

Penggunaan parameter inisialisasi sangat besar, terutama ketika berhubungan dengan informasi yang mungkin bervariasi dengan setiap deployment dari aplikasi. Juga, menyediakan beberapa data ke servlet sebagai parameter, berlawanan dengan hard-coding yang secara langsung ke dalam sevrlet, menyediakan deployer kemampuan untuk mengubah sifat servlet tanpa meng-compile ulang kode.

Kita dapat tambahkan parameter inisialisasi ke servlet dengan menspesifikasikannya ke dalam definisi servlet dalam descriptor deployment. Di bawah ini contohnya :

```
...
<servlet>
   <servlet-name>FirstServlet</servlet-name>
  <servlet-class>jedi.servlet.FirstServlet</servlet-class>
  <init-param>
     <param-name>debugEnabled</param-name>
     <param-value>true</param-value>
   </init-param>
</servlet>
...
```

Tag <init-param> dan </init-param> memberitahu container bahwa kita memulai dan mengakhiri definisi servlet secara berturut-turut. <param-name> mendefinisikan nama dari parameter, dan <param-value> mendefinisikan nilainya.

Untuk memperoleh akses ke parameter servlet, servlet harus mendapat penanganan pertama pada object ServletConfig nya, dimana dapat dilakukan dengan memanggil method getServletConfig(). Setelah itu, nilai parameter dapat diterima sebagai String dengan memanggil method getInitParameter dan menyediakan nilai dari <param-name> sebagai parameter.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
   ServletConfig config = getServletConfig();
   String isDebugEnabled = config.getInitParameter("debugEnabled");
   if (isDebugEnabled.equals("true") {
      ...
}
```

Di atas adalah contoh potongan kode yang mengilustrasikan prosedur.

ServletContext

Object ServletContext adalah object yang memberikan servlet akses ke context aplikasi.

Anggap context aplikasi sebagai area dimana aplikasi berpindah. Area ini disediakan oleh container untuk setiap aplikasi web dengan setiap context aplikasi yang dipisah dari yang lainnya – sebuah aplikasi mungkin tidak mengakses context dari app lain.

Mempunyai akses ke context ini sangat penting karena dengan servlet ini dapat menerima parameter dan data aplikasi yang bermacam-macam. Juga dapat menyimpan data yang dapat diterima oleh berbagai komponen dalam aplikasi.

Parameter Aplikasi

Dalam banyak cara yang sama parameter inisialisasi dapat disediakan untuk servlet individu, mereka juga dapat disediakan untuk digunakan oleh seluruh aplikasi.

```
<context-param>
   <param-name>databaseURL</param-name>
   <param-value>jdbc:postgresql://localhost:5432/jedidb</param-value>
</context-param>
```

Ditunjukkan di atas contoh bagaimana menambah parameter aplikasi yang bermacam-macam. Elemen XML yang digunakan di sini adalah <context-param>. Setiap instance dari elemen yang mendefinisikan parameter untuk digunakan oleh keseluruhan aplikasi. <param-name> dan <param-value> bekerja dalam cara yang sama seperti <init-param>.

CATAT : Sekali lagi, ingat bahwa mengurutkan elemen dalam descriptor deployment haruslah tepat. Untuk menjaga file web.xml tetap valid, semua entri <context-param> harus ditempatkan SEBELUM entri <servlet> apapun.

Prosedur untuk mendapatkan nilai parameter sangat mirip dengan yang digunakan untuk mendapatkan parameter servlet-specific. Merupakan instance dari ServletContext yang servlet harus ditangani juga. Ini dapat didapatkan dengan memanggil method getServletContext() dari instance dari object ServletConfig dari servlet.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    ServletContext ctxt = getServletConfig().getServletContext();
    String jdbcURL = ctxt.getInitParameter("databaseURL");
    setURL(jdbcURL);
....
}
```

2.2 Ringkasan

- Servlet adalah solusi Java untuk menghasilkan content yang dinamis bagi web dan bagian dari CGI.
- Servlet datang dengan berbagai keuntungan daripada CGI, termasuk mengurangi memory footprint dan lebih sedikit pengeluaran dengan setiap permintaan client.
- Servlet sepenuhnya diatur oleh containernya. Kebutuhan kode bagi developer salah satunya mengimplementasikan secara fungsional.
- Untuk membuat servlet, developer akan membuat subclasses dari HttpServlet dan menempatkan implementasi fungsionalnya dalam method doGet atau doPost.
- Rincian request dapat diterima dari HttpServletRequest, dan method response-generating dapat diakses dari HttpServletResponse. Keduanya dilewatkan sebagai parameter ke doGet dan doPost.
- Untuk men-deploy servlet ke dalam container web, dapat di-load sebagai file WAR pre-package. Proses packaging dapat dilakukan secara otomatis dengan IDE atau melalui penggunaan build tool.
- Descriptor deployment adalah bagian yang perlu dari aplikasi. Mereka harus ditempatkan dalam direktori WEB-INF aplikasi dan harus mengikuti aturan yang pasti.
- Parameter dapat disediakan pada aplikasi menggunakan descriptor deployment dan dapat diterima dengan menggunakan method dalam instance ServletConfig atau ServletContext.