الفصل الأول

مقدمة في برمجة تطبيقات الانترنت

تعريف التطبيقات هي البرامج التي تعمل على الحاسب الشخصي وحينما نقول تطبيقات الانترنت تكون هي البرامج المعتمدة على الانترنت في عملها وذلك باستخدام البرتوكول HTTP . وهذه التطبيقات وهي تطبيقات الانترنت هي التي تكون وتعمل بين العميل والخادم وتكون عبارة عن وسيط بين هذين الطرفين . حيث أنها تقوم بأخذ المعلومات من العميل وتوصيلها إلى الخادم من خلال شبكة الانترنت . وفي الأخير تعرض للعميل على أنها صفحة انترنت بسيطة مكتوبة بلغة HTML .

بمعنى آخر يمكن أن تكون تطبيقات الانترنت كأي تطبيق موجود على جهاز الحاسب في مؤسسة تعتمد على أسلوب الخادم والعميل Client-Server . حيث يكون الاتصال في هذا التطبيق معتمد على الشبكة المحلية للمؤسسة . وترتبط المعلومات عندما يقوم العميل Client (**وهو عبارة عن جهاز حاسب يقع في نطاق الشبكة المحلية للمنظومة ويحتوي على تطبيقات العميل**) بإرسال طلب معين إلى الخادم (**وهو في العادة عبارة عن جهاز حاسب ذو مواصفات معينة يقع في نطاق الشبكة المحلية للمنظومة ويحتوي على تطبيقات خاصة بالخادم**) الذي يقوم بدوره بالعمليات المطلوبة من قبل العميل . ولكن باستخدام تطبيقات الانترنت فان العملية سوف تكون اكبر وتبادل المعلومة سوف يكون اكبر كذلك بسبب استخدام الانترنت كوسيلة للربط بين العميل والخادم . بالإضافة إلى أن العميل لا يكون مقيد بنظام تشغيل معين وإنما يكفي وجود برنامج المتصفح محملا على جهازه فقط .

§ الفرق بين الصفحات الثابتة والمتغيرة (Static Pages Vs Dynamic Pages)

في أوائل الانترنت كانت المواقع الموجودة في ذلك الوقت صفحات ثابتة أي أنها مكتوب بلغة HTML فقط . وكان الكثير من هذه المواقع يعتمد على وجود هذه الصفحات لتصل إلى بعض الأحيان إلى صفحة واحدة فقط . وكان من مميزات تلك الصفحات في ذلك الوقت أنها لا تتغير في وقت التنفيذ بمعنى أنها لا تستجيب لطلب المستخدم أو العميل . وكان من الملزم لإجراء أي تغيير عليها استخدام إحدى محررات النصوص وإعادة كتابتها مرة أخرى ومن ثم حفظها على الخادم مرة أخرى . ولهذا فان لغة HTML تعتبر لغة مناسبة للصفحات التي تستخدم لغرض العرض فقط .

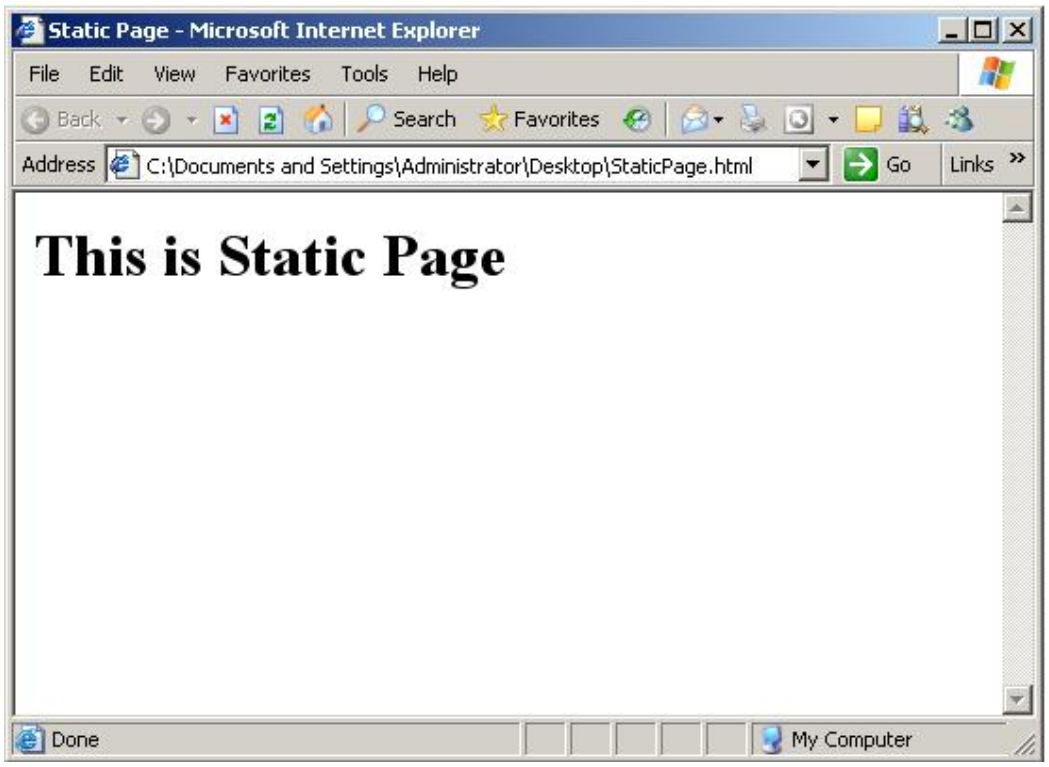
ومع تقدم التقنيات ظهرت صفحات يمكن تغيرها أثناء وقت التنفيذ أو يمكن أن تتغير تلك الصفحات بناء على طلب من المستخدم . ولكن هذه الصفحات هي مكتوبة بلغة HTML أيضا إلا أنها تستخدم تقنيات أخرى بحيث تعطي صفحة التغيير على حسب طلب العميل .

وفيما يلي مثال على الصفحات الثابتة :

مثال رقم ١ : الصفحات الثابتة
StaticPage.html

```
<html>
<head>
meta http-equiv="Content-Type" content="text/html; >
<"charset=windows-1256
<title>Static Page</title>
<head/>
<body>
<h1>This is Static Page</h1>
<body/>
<html/>
```

وفيما يلي عرض للصفحة

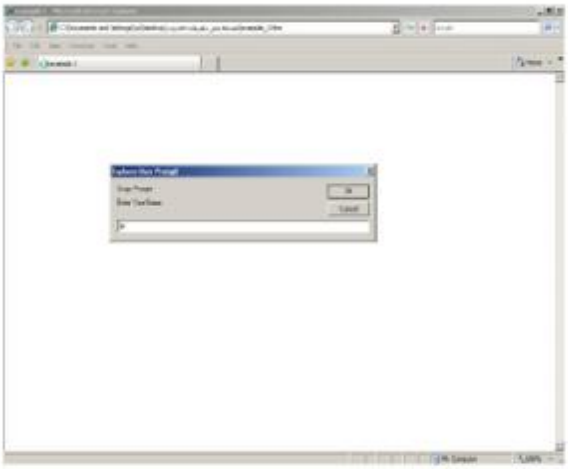


في المثال السابق ترى استخدام لغة HTML فقط في كتابة صفحة ثابتة للعرض فقط وبدون أي تفاعل مع المستخدم . ولكي تميز الفرق بين الصفحات الثابتة والمتغيرة انظر إلى هذا المثال :

مثال رقم ٢ : الصفحات المتغيرة
DynamicPage.html

```
<html>
<head>
<title>example 2</title>
<head/>
<body>
```

```
<"script type="text/JavaScript>
name = prompt("Enter Your Name");
document.writeln("Welcome : " + name");
<script/>
<body/>
<html/>
```



في المثال السابق ترى استخدام HTML مع لغة JavaScript . وأدى ذلك إلى إنشاء صفحة متغيرة أثناء وقت التنفيذ . حيث يتغير محتوى الصفحة على حسب المدخلات من المستخدم . وذلك يظهر التفاعل بين المستخدم والصفحة المتغيرة .

§ البرمجة في جهة الخادم والبرمجة من جهة العميل

البرمجة من جهة الخادم : وهي البرامج التي تكون موجودة على الخادم (Server) . ويقوم بتنفيذها باستمرار وذلك للرد على الطلبات القادمة من جهة العميل (Client) . حيث توجد الكثير من اللغات والتقنيات التي تساعد على عمل مثل هذه البرامج . منها التالي :

- **Php** §
- **Asp** §
- **CGI** §
- **Perl** §
- **JSP** §
- **Servlet** §

البرمجة من جهة العميل : وهي البرامج التي يتم تنفيذها من قبل العميل . حيث يقوم العميل عادةً بتحميل الملفات الموجودة على الخادم ومن ثم يقوم بتنفيذها . ومن أشهر اللغات أو التقنيات التي تنفذ من قبل العميل

- **JavaScript** §
- **Java Applet** §
- **VB Script** §

§ Java - نظرة عامة على التقنيات

لقد أصدرت شركة Sun Microsystems وهي الشركة المنتجة للغة العديد من الحزم التي تدعم التطبيقات والبرامج بلغة Java . ويطلق عليها اسم **JDK** (Java Development Kit) أو **SDK** (Software Development Kit) . وهذه الحزم تنقسم إلى التالي :

١. J2SE (Java 2 Standard Edition) : حيث تحتوي هذه الحزمة على البرامج اللازمة لتطوير تطبيقات Java الشخصية أو المحدودة . وعادة تستخدم J2SE في الأجهزة الشخصية أو التطبيقات على مستوى مؤسسات صغيرة .
٢. J2ME (Java 2 Micro Edition) : حيث تحتوي هذه الحزمة على البرامج اللازمة لتطوير تطبيقات Java التي يمكن استخدامها على الهواتف النقالة أو المحمولة أو الأجهزة الالكترونية الصغيرة
٣. J2EE (Java 2 Enterprise Edition) : حيث تحوي هذه الحزمة على البرامج اللازمة لتطوير تطبيقات Java على مستوى الشركات الكبيرة . حيث تتضمن هذه الحزمة على API اللازمة لتطوير مثل هذه البرامج الداعمة للشبكات عموماً وللانترنت خصوصاً . وعادة تستخدم J2EE في برمجة تطبيقات الويب (Web Application) لتنفيذها على الانترنت . حيث يمكن استخدام J2EE في تطوير العديد من تقنيات لغة Java الداعمة للويب . ومن أهمها :
   - § Servlet
   - § JSP (Java Server Pages)
   - § JB (Java Beans)
   - § EJB (Enterprise Java Beans)
   - § Applet

§ التركيب العام لتطبيقات الانترنت بلغة JAVA

عادة تكون تطبيقات الانترنت المعتمدة على لغة Java مبنية على أساس تركيب (architectures) معين . من أشهرها :

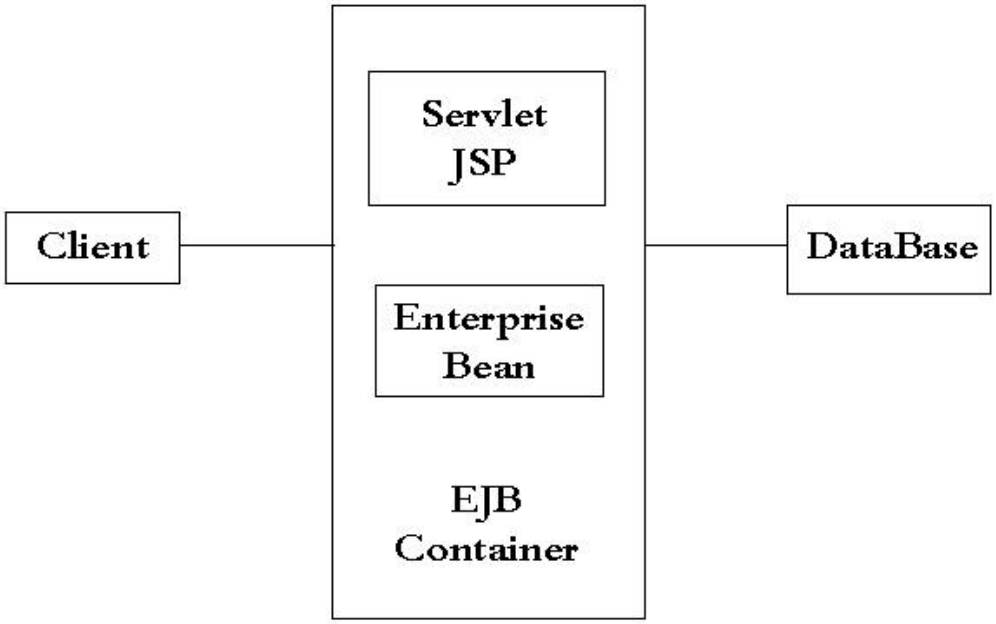
١. تركيب معتمد على وجود تقنيات Java مثل Servlet و JSP كتقنيات بينية بين الخادم والعميل . وفيما يلي رسم توضيحي لهذا البناء :

في هذا البناء يتم توظيف تقنيات Servlet و JSP لتصبح طرف أو طبقة بينية أو ما يسمى (Middle Tier) بين العميل Client و الخادم Server . حيث يطلق على مثل هذا التطبيق اسم تطبيق ثلاثي الطبقات (Three–Tiered Application) . وتنقسم إلى الآتي :

- § طبقة العرض (Presentation Layer) : وتهتم بطريقة العرض للمعلومات عند العميل (Client)
- § طبقة منطقة العمل (Business Logic Layers) : وهي مجموعة من البرامج التي تهتم بمنطق عمل التطبيق توافقاً مع منطق عمل المنظومة .
- § طبقة البيانات (Data Layer) : هنا يكون الاهتمام بطريقة استخراج أو تعديل البيانات في قاعدة البيانات مثلاً أو في الملفات التي تحتوي على البيانات

يعد هذا البناء لتطبيقات الانترنت مناسبا للشركات أو المؤسسات الصغيرة أو متوسطة الحجم

٢ . (J2EE Application Architecture) : هذا النوع من التركيب يعتمد على خصائص إصدار J2EE في خلق تركيب أكثر تعقيداً في تطبيقات الانترنت من حيث الإنشاء . ولكن يتميز بمناسبته لتطبيقات الكبيرة في الشركات والمؤسسات . وقيم يلي رسم توضيحي لهذا البناء



§ النماذج في لغة HTML – HTML Forms

تعتبر النماذج في لغة HTML واجهه سهلة للمستخدم لإدخال البيانات باستخدام مثلاً مربعات النصوص وقوائم الاختيار ومربعات نعم / لا بالإضافة إلى الأزرار. وغير ذلك من الأدوات والعناصر التي توفرها لغة HTML من خلال النماذج . حيث يقوم المستخدم بملأ عناصر نموذج معين ومن ثم الضغط على زر الإرسال حيث تنتقل البيانات من (المتصفح) إلى برنامج موجود على خادم الويب وذلك عن طريق استخدام برتوكول HTTP .

يتم إدراج النموذج بالوسم Form حيث يحتوي الوسم Form على مجموعة من الخواص أهمها :
Action : وفيه يحدد عنوان البرنامج الذي سوف يستفيد من البيانات . فمثلاً يمكن أن يكون هذا البرنامج هو عبارة عن Servlet موجود على الخادم

http://localhost:8080/YourApplicationName/servlet/ClassName

Method : وفيها تحدد طريقة نقل البيانات . ومن أشهر الطرق المستخدمة طريقة post و get .

```
<html>
<head>
<title>example 3</title>
</head>
<body>
<form action="http://localhost:8080/YourApplicationName/servlet/ClassName" method="get">
</form>
</body>
</html>
```

§ استخدام طريقة POST و GET

تعتبر طريقة Post و Get طريقتان مناسبتان لنقل البيانات المدخلة في نموذج . فعندما نريد أن نقوم بإرسال بيانات نموذج من صفحة انترنت إلى صفحة أخرى فلا يمكننا القيام بذلك إلا عن طريق استخدام إحدى هاتين الطريقتين . وينص برتوكول HTTP على أن البيانات ترسل في صيغة أزواج . كل زوج هو عبارة عن اسم الحقل (**اسم الموسم المعطى للنموذج**) و المعلومة نفسها (**القيمة المدخلة في النموذج**) . فإذا أردنا إرسال الاسم مثلا من خلال أي نموذج فإننا نقوم بإرسال العبارة التالية :

```
name=YourName
```

فالجزء الأول وهو (name) يسمى اسم المعلومة (اسم الحقل في النموذج) . والجزء الثاني (YourName) وهي قيمة الحقل أو المعلومة المرسلة . ولهذا فان اسم المعلومة والمعلومة نفسها عند الإرسال تسمى زوج بيانات . ولكن إذا أردنا أن نرسل أكثر من معلومة من خلال نموذج واحد فإننا نقوم بالفصل بين الأزواج بعلامة & . وهذا مثال عندما نريد أن نرسل الاسم والدولة من خلال نموذج ويكون تركيب العبارة كالتالي

```
name=YourName&country=Yourcountry
```

إضافة إلى ما سبق فان هنالك شروطاً على إرسال البيانات في العناوين URL . ومن أهم الشروط هو أننا لا نستطيع وضع مسافات بين الأزواج في العناوين URL . ولهذا فإن جميع المسافات في البيانات تحول إلى إشارة (+) . فإذا كان اسم البلد مثلاً هو United States فان سطر البيانات سيكون كالتالي :

```
name=YourName&country=United+States
```

وهنالك أيضا شروط أخرى لتحويل الرموز الغير انجليزية والكثير من الأسماء التي يتم تطبيقها على البيانات حتى تصبح جاهزة للإرسال . وينص برتوكول HTTP أيضا على أن البيانات ترسل بأكثر من طريقة كما ذكر سابقاً . يتم إرسال البيانات بطريقة get بصورة بسيطة جدا بحيث تكون عبارة عن جزء يضاف على عنوان البرنامج الموجود على الخادم بعد علامة الاستفهام ? . ومثلا إذا كان لديك برنامج يأخذ الاسم والبلد ويخزنها على في قاعدة البيانات . وكان عنوان البرنامج هو :

```
http://localhost:8080/YourApplicationName/servlet/ClassName
```

فان الطريقة get ستقوم بإضافة البيانات إلى العنوان السابق بعد علامة الاستفهام . ليصبح العنوان كالتالي :

```
…/servlet/ClassName?name=YourName&counrty=Yourcountry
```

ولابد أنك رأيت شيئا مثل ذلك عندما تزور المواقع الضخمة كمثل Google أو Yahoo . هذا الأمر يحدث عندما تستخدم طريقة get .

أما الطريقة الثانية فهي طريقة post . وفيها يتم إرسال البيانات للخادم عن طريق HTTP ولكن هذه المرة لن تظهر البيانات في شريط العنوان (URL) بل سوف تكون البيانات المرسلة مخفية عن العميل . ولكن طريقة post لا يمكن لها العمل إلا عن طريق النماذج بحيث أنها لا تعمل إلا عن طريق مفتاح النماذج مثل زر Button أو Submit . ولهذا فان الطريقتين تختلف عن بعضهما

البعض . فطريقة get تعتبر أسهل وابسط بكثير من طريقة post إلى درجة انك تستطيع أن تركب البيانات بنفسك عن طريق وضعها في العنوان نفسه ورؤية النتائج تلقائياً . وطريقة post تعتبر طريقة جيدة ومناسبة في حالة إرسال كمية كبيرة من البيانات مثل نص رسالة أو في الاتصالات السرية أيضاً . (ملاحظة : هذا لا يعني أنها آمنه جدا عبر الويب . وإنما تحتاج إلى تقنيات أخرى لتجعلها أكثر أمان)

الفصل الثاني

Servlet – نظرة عامة

يعتبر الـــServlet برنامج ينفذ على جهة الخادم كما هو معروف عنه (Web Server) . بحيث أنه يكون طبقة بينية (Middle Layer) بين الطلبات القادمة من جهة العميل Client (Web Browser) وقواعد البيانات أو تطبيقات أخرى على الخادم HTTP . وفيما يلي رسم توضيحي للطبقة البينية :

الـــ Servlet يقوم بالمهام التالية :

١ . قراءة البيانات المرسلة من المستخدم
وهذه البيانات هي التي تكون في النماذج أو المدخلة بواسطتها .

٢ . البحث عن البيانات المتعلقة بالطلب (القادمة من العميل)
وتتضمن هذه البيانات معلومات عن المتصفح الخاص بالعميل .

٣ . توليد النتائج (Generate Results)
عادة يقوم الـــServelt بمعالجة البيانات القادمة من العميل وإرسال النتائج إليه العميل مرة أخرى .

٤ . تنسيق النتائج على شكل مستند (HTML)
في الغالب يقوم الـــServlet بتنسيق المخرجات أو النتائج على شكل ملف HTML متضمناً الرد للعميل . ويمكن تنسيق المخرجات على شكل ملف word أو غيرها كما سوف تعرف في هذا الكتاب

٥ . ضبط المعاملات (Parameters) المتعلقة بـــ(HTTP Response)
تتضمن هذه العملية إبلاغ متصفح العميل بنوع المستند المرسل إليه وغير ذلك من معلومات يحتاج إليها المتصفح) .

٦ . إرسال المستند إلى العميل
هذا المستند يمكن إرساله بعده تنسيقات مثل نص (TEXT) كما هو الحال في ملفات HTML أو على هيئة صورة (مثل GIF Image) أو على صورة ملفات مضغوطة أيضاً .

§ إعداد وتهيئة بيئة العمل لتطوير وتنفيذ Servlet و JSP

وعند العمل بهاتين التقنيتين فمن اللازم الحصول على بعض البرامج التي يمكن من خلالها تكوين بيئة عمل بحيث يمكن تطوير التطبيقات بهما ومن ثم اختبارها ومن ثم استخدامها على الانترنت

ويمكن تقسيم هذه البرامج على الشكل التالي :

١ . Web Client : وهذا يتمثل بوجود متصفح الانترنت على الجهاز الشخصي مثل Internet Explorer أو Netscape أو FireFox أو غيرها من المتصفحات . ويمكن الحصول على احد هذه المتصفحات من خلال شبكة الانترنت .

٢ . Web or Application Server : فمن المؤكد أن يقوم الجهاز الشخصي بما يقوم به الخادم للويب . وذلك يتمثل بوجود احد البرامج التي تقوم بهذا العمل مثل IIS أو Tomcat أو غيرها .

٣ . Java Development Kit – JDK : وتحتوي هذه الحزمة على البرامج اللازمة لتطوير تطبيقات الانترنت .

وفيما يلي خطوات أو إعدادات بيئة العمل اللازمة للتطوير

١ . سوف نقوم باستخدام Internet Explorer Browser كعميل للويب

٢ . سوف نقوم باستخدام Apache Tomcat كخادم للويب . وكما يعتبر الـــTomcat داعم لـــServlet2.2 & JSP1.1 . يمكن الحصول على هذا البرنامج من خلال الرابط التالي :
http://jakata.apache.org.tomcat

٣ . تعريف الـــAPI لمترجم الـــJava (javac) من خلال استخدام المتغير البيئي JAVA_HOME . وبالتالي سوف يكون هذا التعريف على هذا الشكل

```
Variable Name  : JAVA_HOME
Variable value : C:\jdk1.6.0_02
```

٤ . تعريف الـــAPI الخاص بالـــServlet من خلال استخدام المتغير البيئي CLASSPATH وذلك من خلال اضافة ملفات jar المحتويه على الـــClasses . وبالتالي سوف يكون التعريف على هذا الشكل :

```
Variable Name  : CLASSPATH
Variable value : install_dir\common\lib\servlet-api.jar
```

حيث أن install_dir هو مكان برنامج Tomcat ومثال ذلك هو :

```
.;C:\apache-tomcat-5.5.17\common\lib\servlet-api.jar
```

ملاحظة : أن الإصدار الجديد يدمج المجلدين lib و common ليصبح كالتالي :

```
C:\apache-tomcat-6.0.14\lib\servlet-api.jar
```

٥ . تعريف خادم الويب من خلال استخدام المتغير البيئي CATALINA_HOME
وبالتالي سوف يكون التعريف على هذا الشكل :

```
Variable Name  : CATALINA_HOME
Variable value : C:\apache-tomcat-5.5.17
```

٥ . ترجمه ملف Servlet من خلال javac servletName حيث ينتج servletName.class

٦ . توضع جميع الملفات من نوع class في مجلد تحت خادم الويب . وفي هذه الحالة سوف يكون المجلد على هذا الشكل :

```
Install_dir/webapps/ROOT/WEB-INF/classes
```

حيث أن المجلد classes سوف توضع فيه جميع الملفات أو البرامج من نوع class ومثال ذلك كالتالي :

```
C:\apache-tomcat-5.5.17\webapps\ROOT\WEB-INF\classes
```

ملاحظة : توضع جميع ملفات HTML وجميع ملفات JSP بالإضافة إلى ملفات الصور وغيرها في مجلد ROOT كالتالي :

```
C:\apache-tomcat-5.5.17\webapps\ROOT
```

٧ . يجب عمل بعض التغيرات على تهيئة Tomcat في ملف Install_dir/conf/web.xml .
وذلك بإزالة التعليق <-- --!> من الجمل التالية ولذلك لتسهيل عملية مناداة الـــServlet في كل مرة :

```
<servlet-mapping>
<servlet-name>invocker</servlet-name>
<url-patten>servlet/*</url-patern>
</servlet-mapping>
```

. وكما أيضا لابد من إضافة السطر التالي إلى install_dir/conf/servlet.xml :

```
<DefaultContext reloadable="true">
```

بعد الجملة

```
<!—Define proreties for each web application …..
```

§ التركيب العام للـServlet :

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

public class HelloWorld extends HttpServlet
}
، public void doGet(HttpServletRequest request
(HttpServletResponse response
} throws ServletException , IOException
{
{
```

حتى يكون لدينا Servlet يجب أن نعرف أن أي Servletclass يرث من HttpServlet من خلال استخدام exteands و إعادة تعريف طريقة doGet . ويمكن استخدام طريقة doPost إذا كان طلب العميل من نوع Post (نوع الـــMethod في الوسم Form) .

كلاً من doGet و doPost يستخدمان HttpServletRequest و HttpServletResponse . حيث تحتوي HttpServletRequest على الطرق اللازمة للحصول على المعلومات من العميل مثل : بيانات النموذج المرسل ، عناوين الطلب (Http Request Headers) ، اسم المضيف (Hostname) وغير ذلك .

أما HttpServletResponse فتحتوي على الطرق التي تساعد على تحديد المعلومات المرسلة للمخدوم . ومن أمثلة هذه المعلومات : Http Response Headers – HTTP Status Code . ولكن أهم ما يميز HttpServletResponse هو تمكن الحصول على PrintWrite والتي من خلالها يمكن إرسال محتويات المستند للعميل . تحتوي PrintWrite على الطريق println والتي من خلالها يمكن بناء المحتوى أو الرد للمخدوم (العميل) .

وبما أن doGet و doPost يمكن أن تطرح exceptions فلابد من استخدام جملة throws كما في المثال السابق .

ومن خلال صنع أي Servlet فلابد من استدعاء عدد من الـــpackages الضرورية التي بدونها لن يعمل أي ServletClass . منها :

- § Java.io.* : من اجل PrintWrite والتي من خلالها يتم إرسال الرد للعميل
- § Javax.servlet.* : من أجل HttpServlet
- § Javax.servlet.http.* : من اجل HttpServletRequest و HttpServletResponse

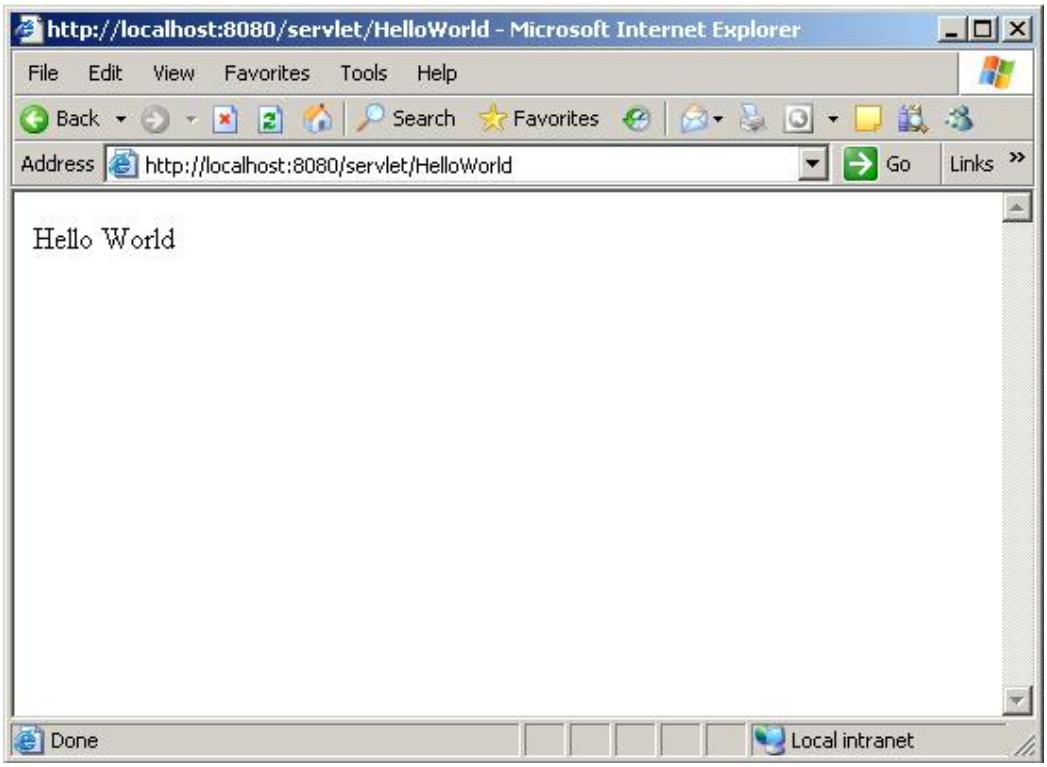
تطبيق ١ : توليد نص من خلال Servlet HelloWorld.java

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

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

             PrintWriter out = response.getWriter();
             out.println("Hello World");
        }
       }
```

في هذا الـــServlet يقوم البرنامج بإرسال جملة Hello World كنص للعميل حيث استخدمنا التركيب العام للـــServlet بالإضافة إلى اختيار الطريقة doGet لمعالجة طلب العميل . وذلك حتى يرسل العميل الطلب مباشرة من خلال العنوان URL . والشكل التالي يوضح نتيجة هذا الطلب



هنا استخدمنا `out.println("")` من اجل إعداد مستند للرد . حيث يتم إضافة الجمل إليها . وفي هذا المثال أضيفت جملة Hello World للمستند المرسل للعميل فقط . وفي النهاية يكون الرد محتوي على الجملة فقط .

<u>لكي نقوم بتنفيذ هذا الـــServlet يجب إتباع التالي :</u>

١ . حفظ الـــServlet باسم HelloWorld.java
٢ . ترجمه البرنامج السابق من خلال Javac .

٣ . بعد الترجمة ينشأ ملف من نوع (.class) . وسوف يوضع الملف في مجلد classes في خادم الويب أسفل WEB-INF . وفي هذه هذا المثال سوف يكون المجلد على هذا النحو :

```
C:\apache-tomcat-5.5.17\webapps\ROOT\WEB-INF\classes
```

٤ . الآن ومن خلال المتصفح يمكن كتابة العنوان التالي :

http://localhost:8080/servlet/HelloWorld

ملاحظة : عند إنشاء Servlet أو القيام بالتعديل عليه فلابد من إطفاء الخادم وتشغيله مرة أخرى لكي يقوم بالتعرف عليه أو على التعديلات التي حدثت على ملف الـــServlet

الفصل الثالث

استخدام الـServlet

من خلال الفصل السابق قد عرفت التركيب الأساسي للـــServlet بالإضافة إلى توليد النصوص والجمل . وهنا سوف ترى كيف يتم توليد النتائج ولكن هذه المرة عن طريق إرسال صفحات الرد بلغة HTML

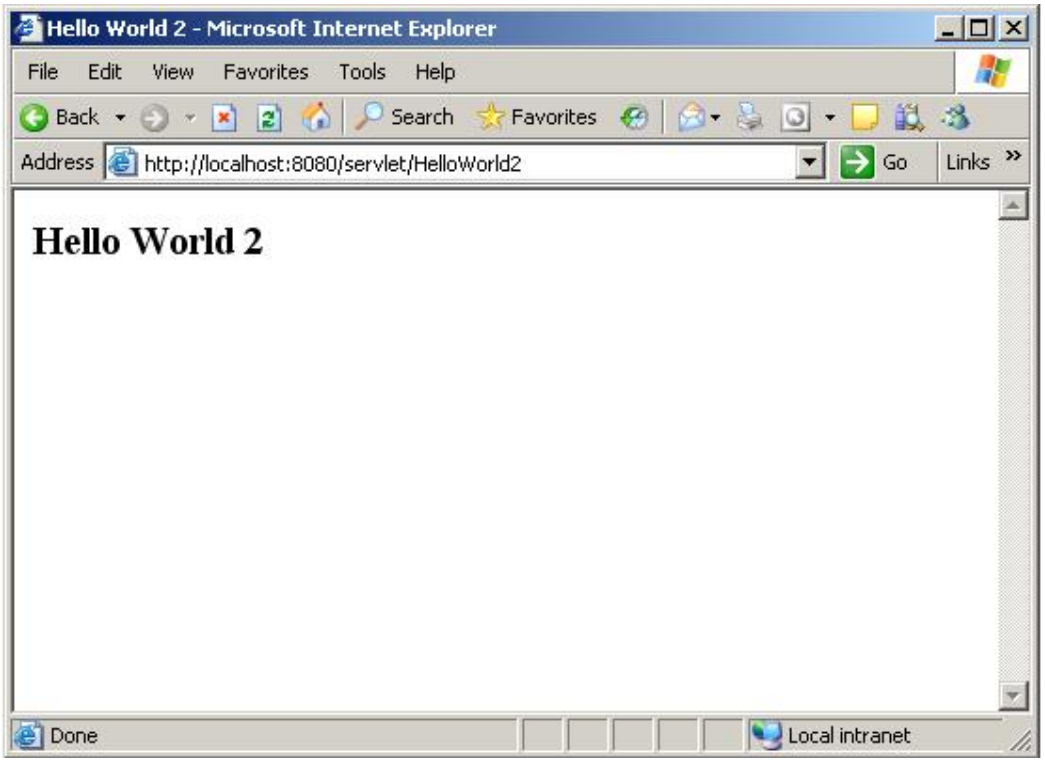
تطبيق ٢ : توليد HTML من خلال Servlet HelloWorld2.java

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

public class HelloWorld2 extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {

response.setContentType("text/html");
PrintWriter out = response.getWriter();
String docType ="<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
" +"Transitional//EN\">\n";
out.println(docType +
"<HTML>\n" +
"<HEAD><TITLE> Hello World 2 </TITLE></HEAD>\n" +
"<BODY>\n" +
"<H1> Hello World 2 </H1>\n" +
"</BODY></HTML>");
}
}
```

يقوم هذا الـــServlet بإرسال جملة Hello World 2 للعميل باستخدام HTML . حيث استخدمنا التركيب العام للـــServlet مع اختيار الطريقة doGet لمعالجة طلب العميل . وذلك حتى نسمح للعميل لطلب الرد مباشرة من خلال العنوان URL . والشكل التالي يوضح نتيجة الطلب



وهنا استخدمنا **reponse.setContentType("text/html")** من اجل إعداد المستند للرد . وهنا استخدمنا **String docType** لحفظ نوع المستند المرسل للمتصفح حتى يتمكن المتصفح من معرفة نوع المستند المرسل إليه من اجل ترجمته وتنفيذه .

أيضا استخدمنا طريقة **out.println();** من اجل بناء المستند المرسل للعميل وكذلك تنسيق المستند بشكل صفحة HTML بوجود الوسوم الأساسية لصفحة HTML .

§ دورة حياة الـServlet

يمر الـــServlet بمراحل منذ إنشاءه إلى نهايته . حيث ينشأ حالة (instance) عند إنشاء الـــServlet . وتتم مناداة طريقة ()init التي بدورها تحوي code يتم تنفيذها لمرة واحد عند إنشاء الـــServlet . بعد ذلك يتم إنشاء thread لكل طلب من المستخدم . حيث تقوم كل thread بمناداة طريقة ()service . بعد ذلك تقوم طريقة ()service بمناداة طريقة doPost أو doGet أو غيرها من الطرق (doXxx) وذلك على حسب نوع طلب http المراد معالجته . أخيرا قبل أن يتم إنهاء الـــServlet من قبل الخادم يتم مناداة طريقة Destroy للقيام بالأعمال النهائية للـــServlet

§ استخدام الطريقة ()init

تتم مناداة ()init لمرة واحدة عند إنشاء الـــServlet . وتعتبر مناسبة لإعدادات الـــServlet عند إنشاءه لأول مرة . ولا تتم مناداتها مرة أخرى عند كل طلب من المستخدم . ولهذا فإنها لا تنادى مرة

أخرى عند النقر على زر التحديث للـــServlet أو فتح صفحة جديدة وكتابة العنوان مرة أخرى .
ويمكن تعريف طريقة ()init على الشكل التالي :

```
Public void init () throws ServletException {

}
```

إن احد أهم أسباب استخدام طريقة ()init هي قراءة المعلومات عن الخادم المضيف للـــServlet

§ استخدام طريقة ()service

لكل مرة يستقبل الـــServlet طلب من المستخدم تنشأ thread . حيث تقوم كل thread بمناداة طريقة ()service . حيث تقوم ()service بفحص نوع الطلب من (http request type) حيث يمكن أن تكون GET أو POST أو DELETE أو غيرها . ومن ثم تقوم طريقة ()service بمناداة الطريقة المناسبة لنوع الطلب المراد معالجته مثل doGet أو doPost أو doDelete أو غيرها من طرق أخرى .

في غالب الأحيان يقوم الـــServlet بمعالجة طلب العميل باستخدام طرق doXxx وقد تكون المعالجة لكلا الطرفين متطابقة . ولهذا يمكن حلة من خلال إعادة كتابة طريقة ()service ولكن هذا غير مستحب أبدا . وبدلا من ذلك يمكن إعادة كتابة طريقة doGet ومن ثم مناداتها من خلال طريقة doPost أو بالعكس. ومثال هذه الطريقة كالتالي :

```
Public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
//Servlet Code
}
Public void doPost (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
doGet (request , response );
}
```

§ استخدام طرق doXxx ، doPost ، doGet

هذه الطرق هي الطرق الرئيسية لمعالجة الطلبات عن طريق الـــServlet . وهي شائعة جدا بحيث أنها تستخدم على نحو ٩٩% . وهذا يعني أن معظم الحالات سوف تقوم بإعادة كتابة طرق doGet أو doPost عند إنشاء أي Servlet . ويمكن للـــServlet أيضا أن يخدم أنواع أخرى من الطلبات باستخدام طرق أخرى مثل doDelete أو doPut ولكنها غير شائعة لهذا الحد .

§ استخدام طريقة destroy

الخادم أو الــServer سوف يقوم بإنهاء Servlet معين من الذاكرة وذلك لأسباب عديدة منها : أن يقوم مدير الخادم (Server administrator) بإصدار أمر لإنهاء الــServlet أو بسبب وجود غير مبرر للــServlet في الذاكرة (لا فائدة منه) . ولهذا فان الــServlet يقوم بمناداة طريقة destroy قبل إنهاءه . وذلك للقيام كما تسمى بعمليات التنظيف اللازمة مثل إغلاق الاتصال بقاعدة البيانات .

§ طلب العميل من خلال نموذج بيانات (Form Data)

معظم البيانات المرسلة من خلال المستخدم تكون عن طريق ملئ نموذج بيانات في صفحة ويب ومن ثم إرسالها إلى الخادم (Server) . حيث يقوم الــServlet فيما بعد باستقبال هذه البيانات وتحليلها ومعالجتها ومن ثم الرد عليها . وفي الحقيقة شكل هذه البيانات قد يختلف باختلاف طريقة الإرسال المستخدمة في نموذج البيانات المرسل من المستخدم . فعند استخدام GET تكون هذه البيانات متصلة بعنوان الــServlet (URL) . حيث تكون البيانات المرسلة في الجزء الواقع بعد علامة الاستفهام ? . وكما أيضا يمكن استخدام طريقة POST ولكن في هذه الحالة يتم إرسال البيانات بشكل مخفي للــServlet . (راجع استخدام طريقة POST و GET)

§ قراءة بيانات نموذج من خلال Servlet

من احد أهم مزايا الــServlet انه يستطيع الحصول على البيانات من أي نموذج مرسل . وهذه العملية سهلة وبسيط وذلك من خلال استخدام طريقة getParameter من HttpServletRequest . سواء كانت البيانات المرسلة بطريقة GET أو POST . ومثال ذلك إذا أردنا أن نرسل الاسم إلى الــServlet فسوف يكون سطر التعليمة في الــServlet كالتالي :

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

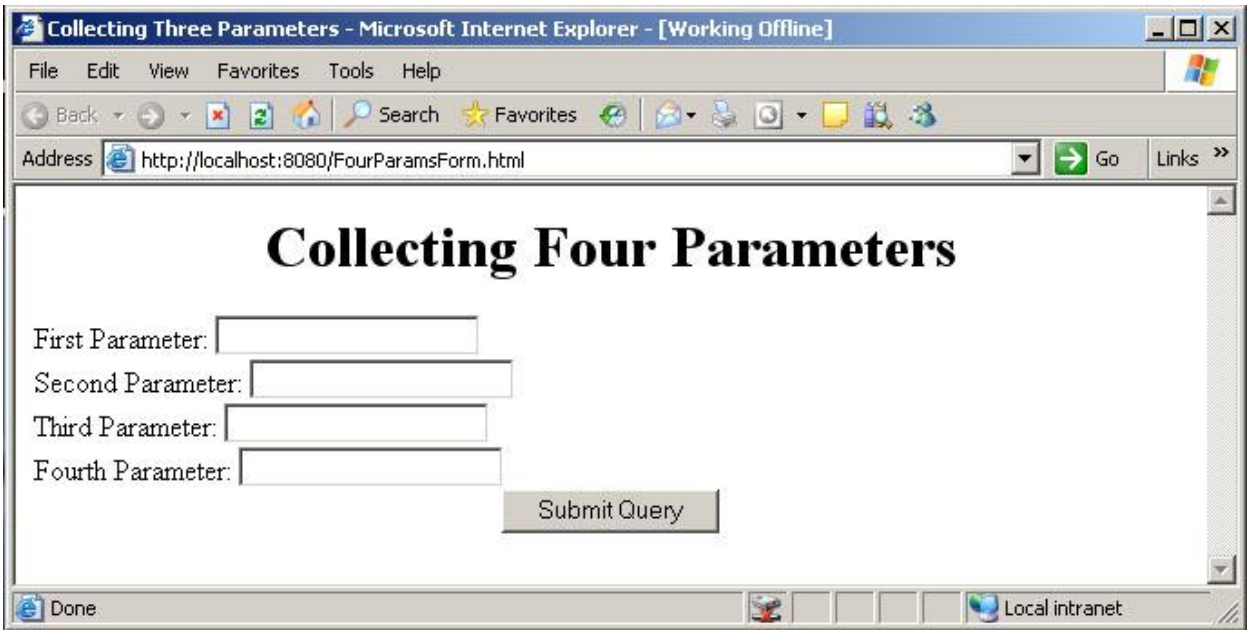
وهذه الطريقة سوف تستقبل قيمة واحدة من نوع String وهو المعامل name من خلال **getParameter** مع الأخذ بالاعتبار تطابق الأسماء أو الأحرف (**case sensitive**) . ولهذا سوف تقوم هذه الطريقة بإرجاع قيمة واحدة إن وجدت . إذا كان المعامل name موجود ولكن بدون قيمة فان القيمة سوف تكون فارغة (null) . بمعنى انه إذا تم إرسال النموذج إلى الــServlet وكان حقل الاسم فارغ فان الــServlet سوف يستقبل هذا النموذج ويبحث عن المعامل name ويستخرج قيمته . وبما أن النموذج أرسل إلى الــServlet ولكن بدون قيمة فسوف تكون قيمة هذه المعامل (name) فارغة (null) .

وفيما يلي تطبيق يقوم بقراءة أربع قيم مرسلة من نموذج بيانات يحتوي على أربع معاملات وهي : param1 و param2 و param3 و param4 .

تطبيق ٣,١ : قراءة المعاملات المرسلة من نموذج
FourParamsForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY>
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>
<FORM ACTION=http://localhost:8080/servlet/FourParams method="GET">
First Parameter: <INPUT TYPE="TEXT" NAME="param1"><BR>
Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
Third Parameter: <INPUT TYPE="TEXT" NAME="param3"><BR>
Fourth Parameter: <INPUT TYPE="TEXT" NAME="param4"><BR>
<CENTER>
<INPUT TYPE="SUBMIT">
</CENTER>
</FORM>
</BODY>
```

وكما تلاحظ في ملف HTML ومن خلال النموذج Form أضفنا عنوان الــServlet الذي سوف يقوم بمعالجة بيانات النموذج من خلال الإشارة إلى مكان الــServlet الذي سوف يقوم بهذه المهمة وذلك عن طريق إدراج الــURL للصفة **action** . بالإضافة إلى إدراج طريقة الإرسال عبر البروتوكول HTTP وهي طريقة **Get** . وفي الوسم Form تم إدراج أربعة حقول من نوع TEXT وأعطيت لها أسماء param1 – param2 – param3 و param4 . وفيما يلي تنفيذ هذا البرنامج

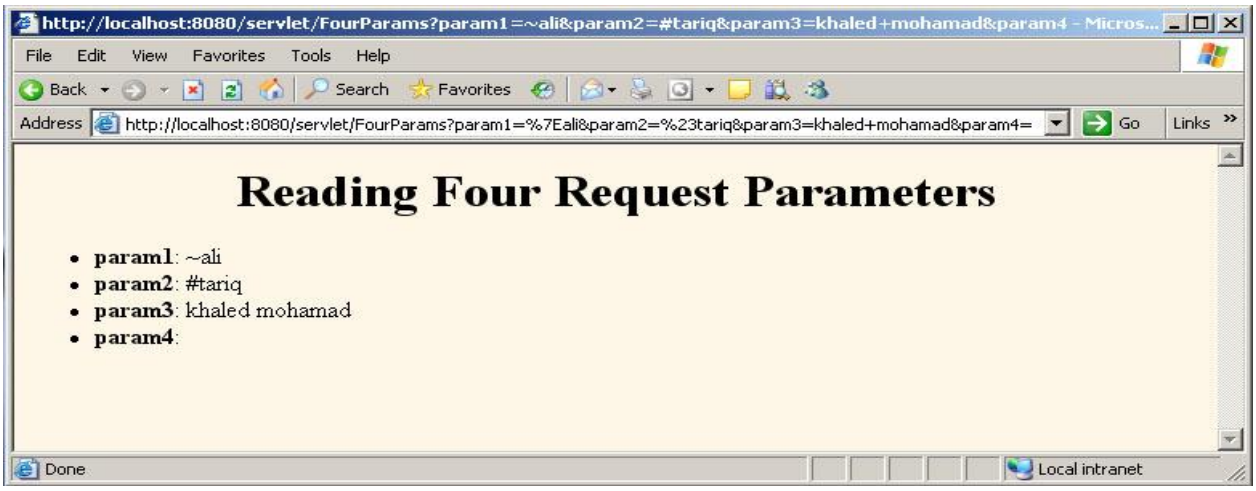


بعد النقر على زر Submit Query فان البيانات الموجودة في الحقول كما هو واضح سوف تنتقل إلى الــServlet وبالتحديد إلى البرنامج المسمى FourParameters كما في التالي :

تطبيق ٣,٢ : قراءة المعاملات المرسلة من نموذج
FourParams.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FourParams extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Reading Three Request Parameters";
out.println(
"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1 ALIGN=CENTER>" + title + "</H1>\n" +
"<UL>\n" +
" <LI><B>param1</B>: "
+ request.getParameter("param1") + "\n" +
" <LI><B>param2</B>: "
+ request.getParameter("param2") + "\n" +
" <LI><B>param3</B>: "
+ request.getParameter("param3") + "\n" +
" <LI><B>param4</B>: "
+ request.getParameter("param4") + "\n" +
"</UL>\n" +
"</BODY></HTML>");
}
}
```

وفيما يلي عرض المستند (الرد) .



فلو لاحظت هنا انتقال البيانات متصلة عبر عنوان الــServlet وذلك بسبب استخدام طريقة GET وذلك بعد علامة الاستفهام ? وتكون بلاشك على شكل أزواج (المعامل وقيمته) . فلو أخذت احد المعاملات وليكن param2 . حيث تمثل القيمة (٢٣%) الواقعة قبل الاسم العلامة (#) المدخلة من قبل العميل في النموذج . ولو قمت بالملاحظة أيضا انه لم يكن هنالك قيمة مدخلة في الحقل الرابع

وهو param4 . ولهذا فعند إرسال الطلب سوف تقوم getParameter بالبحث عن قيمة هذا المعامل وفي الأخير لن تجد له قيمة . ولهذا الأمر كانت قيمته عند الرد (null) .

بعض الدوال التي يمكن استخدامها في الـــServlet :

- getCookies

وتقوم هذه الطريقة بإرجاع محتوى عنوان الطلب على شكل مصفوفة من Cookies Object

- getContentLength

وتقوم هذه الطريقة بإرجاع عنوان الطلب على شكل int

- getContentType

وتقوم هذه الطريقة بإرجاع قيمة عنوان الطلب على شكل String

- getDateHeader and getIntHeader

وتقوم هذه الطريقة بقراءة عنوان الطلب وتحويل قيمته إلى Date أو int

- getHeaderName

وتقوم هذه الطريقة بإرجاع أسماء عناوين الطلب وإرجاع هذه الأسماء على شكل Enumeration

- getHeaders

وتقوم هذه الطريقة بإرجاع أسماء عناوين الطلب والتي يمكن أن يكون لها أكثر من قيمة مثل Accept-Language . وتعتبر هذه الطريقة طريقة بديلة عن getHeader

- getMethod

وتقوم هذه الطريقة بإرجاع طريقة الإرسال عبر البروتوكول HTTP والتي يمكن أن تكون Delete أو Post أو GET

- getRequestURL

وتقوم هذه الطريقة بإرجاع جزء من الـــURL الخاص بعنوان الـــServlet دون إدراج البروتوكول و المخرج و البيانات في العنوان . فمثلا إذا كان لدينا هذا العنوان :

http://localhost:8080/servlet/ServletClass

فان هذه الطريقة ترجع العنوان التالي :

servlet/ServletClass

- getProtocol

وتقوم هذه الطريقة بإرجاع نوع البروتوكول المستخدم ويمكن أن تكون القيمة مثلاً HTTP/1.0 أو HTTP1.1 .

الفصل الرابع

استخدام قواعد البيانات مع Servlet

يعتبر ربط قواعد البيانات مع الـــServlet من الأمور الشائعة جداً في البرمجة . حيث جرت العادة على هذا الاستخدام للاتصال بقواعد البيانات سواء كانت محلية (مؤسسة) أو خارجية (الانترنت) . ولكن يوجد هنالك طريقتين لربط بقواعد البيانات مع الـــServlet . الأولى وتسمى الجسر (Bridge) وذلك عن طريق `ODBC Driver` . والأخرى تسمى تزويد المحرك (Vendor-Supplied) وذلك عن طريق `JDBC Driver` . وكلا الطريقتين متشابهتين لحد كبير ولكن يوجد اختلاف بسيط جدا في بعض التعريفات المتعلقة بعملية الربط .

• ربط الـServlet مع قواعد البيانات باستخدام ODBC

قد وفرت لغة Java حزم خاصة لربط الـــServlet مع قواعد البيانات . ومن قواعد البيانات التي يمكن ربطها بالـــServlet : **Access , MySQL , SQLServer , Oracl** . وجميع الأنواع السابقة يمكن ربطها بالـــServlet باستخدام الجسر `ODBC Driver` .

• الخطوات الأساسية لاستخدام ODBC

خطوات الربط مع قواعد البيانات تمر بسبع مراحل أو بسبع خطوات هي :

١. تحميل البرنامج المشغل (ODBC)
٢. تعريف عنوان الاتصال (URL)
٣. تطبيق الاتصال (Connection)
٤. إنشاء جملة طلب (Statement)
٥. تنفيذ الاستعلام (Query)
٦. إجراء العمليات (ResultSet)
٧. إغلاق الاتصال (Close The Connection)

١ . تحميل البرنامج المشغل

نعني بالبرنامج المشغل هو البرنامج الذي يمكن أن يتخاطب مع قواعد البيانات أو لنقول انه الذي يقع بين قواعد البيانات وبين الـــServlet ويربطهما مع بعضهما البعض . ويتم ذلك عن طريق استخدام صنف (class) وهو Class.forName . حيث يأخذ هذا الصنف سلسة نصية أو تعريف خاص بقاعدة البيانات . والتركيب العام كما في التالي :

```
Class.forName("Connect……Driver");
```

وهذا مثال على التركيب العام :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

حيث أن هذا التعريف هو للربط بأي قاعدة بيانات من نوع Access . ولكن من المهم في ذلك وهو أن مناداة هذا الصنف أو جلبه يمكن أن يرمي بـــClassNotFound لذا من الضروري أن يكون التعريف داخل الكتلة try/catch .

٢ . تعريف عنوان الاتصال

وبعد القيام من تشغيل ODBC يتوجب علينا تحديد مكان قاعدة البيانات أو للـــServlet وذلك عن طريق تحديد مكان قاعدة البيانات والإشارة إليه بواسطة URL . حيث يتضمن URL اسم بروتوكول ODBC و المنفذ (PORT) واسم قاعدة . والتركيب العام كما في التالي :

```
String url = "jdbc:odbc:DataBaseName";
```

وهذا مثال على التركيب العام :

```
String url = " jdbc:odbc:CourseInfo";
```

حيث **CourseInfo** هو اسم قاعدة البيانات .

٣ . تطبيق الاتصال

وبعد القيام بتحديد مكان قاعدة البيانات يتوجب تمرير الـــURL واسم المستخدم وكلمة المرور إلى getConnection من صنف DriverManager . مع العلم أن طريقة تطبيق الاتصال يمكن أن ترمي SQLException . ولهذا فلابد من وضعها داخل الكتلة try/catch . والتركيب العام كما في التالي :

```
Connection connection = null;
connection =DriverManager.getConnection(url,UserName,Password)
```

حيث أن الـــURL هو تعريف مكان قاعدة البيانات و UserName هو اسم المستخدم إن وجد و Password هي كلمة المرور إن وجدت . وهذا مثال على التركيب العام :

```
Connection  connection = null;
connection = DriverManager.getConnection(url ,"","");
```

٤ . إنشاء جملة طلب

عند إنشاء جملة طلب لابد من جلب الكائن Statement لإرسال الاستعلام والأوامر إلى قاعدة البيانات . حيث يتم إنشاء هذا الصنف من الصنف connection باستخدام createStatement . والتركيب العام كما في التالي :

```
Statement statement = connection.CreateStatement();
```

ملاحظة : اغلب برامج التشغيل يمكن لها أن تسمح بفتح عدة كائنات من Statement من اجل الاتصال أو الاتصال الواحد .

٥ . تنفيذ الاستعلام

عندما يكون لدينا الكائن Statement يمكن استخدامه لإرسال الاستعلامات (أوامر SQL) . وذلك باستخدام الطريق executeQuery التي تعيد كائن من نوع ResultSet أو استخدام executeUpdate والتي تعيد رقم من نوع int . والتركيب العام كما في التالي :

```
ResultSet  resultset = statement.executeQuery(Query);
```

وهذا مثال على التركيب العام

```
String Query = "select * from AnyTableName";
ResultSet resultset = statement.executeQuery(Query);
```

بعض الفروقات بين executeQuery و executeUpdate

- executeQuery : تستخدم من اجل Select لتنفيذ استعلام SQL وتعيد البيانات ضمن كائن ResultSet والتي يمكن أن تكون فارغة وليست null .
- executeUpdate : وتستخدم من اجل Delete و Insert و Update وتعيد عدد الأسطر التي تأثرت والتي يمكن أن تكون صفراً وليست null .

٦ . إجراء العمليات

ونقصد بهذه الخطوة هي إجراء العمليات على البيانات التي تم جلبها من قاعدة البيانات . وابسط طريقة لإجراء العمليات على هذه البيانات هو عرضها وذلك باستخدام طريقة next من الكائن ResultSet . وهذه الطريقة هي للتحرك من خلال اسطر الجدول في كل مرة . بمعنى انه عندما نقوم بتنفيذ جملة الاستعلام فان البيانات لا تأتي وتكون على شكل اسطر بل تكون على شكل جدول . كأنها عملية نسخ للجدول الموجود في قاعدة البيانات . وفي هذه الحالة سوف يقف المؤشر قبل السطر الأول في الجدول الذي تم جلبة وفيه البيانات . ولهذا فلابد من التحرك خلال هذه الأسطر عن طريق next() . والتركيب العام لهذا العرض كما في التالي :

```
ResultSet resultset = statement.executeQuery(Query);
resultset.next();
```

ملاحظة أخرى : من الأفضل استخدام أسماء الأعمدة دائما بدل ترتيبها وذلك لتجنب الأخطاء عند تغير ترتيب الأعمدة في الجدول

٧ . إغلاق الاتصال

ولإغلاق الاتصال يمكن القيام بالتالي وإضافة التعليمة :

```
connection.close
```

ملاحظة : إن إغلاق الاتصال السابق يقوم أيضا بإغلاق الكائنات ResultSet و Statement بسبب ارتباطها بها .

• مثال شامل على استخدام قواعد البيانات مع Servlet

في هذا المثال سوف نقوم بربط قاعدة بيانات Access مع ServletClass (ليكن اسمها ServletODBC) وذلك عبر الجسر ODBC . ولكن في هذه المرة لن نستخدم قاعدة بيانات جاهزة كالمعتاد بل سوف ننشئ هذه القاعدة وليكن اسمها ArabTeam . وفيها الحقول التالية : ID – FirstName – LastName – BirthDay – City . وهي كما في الشكل التالي :

اسم الحقل	نوع البيانات
ID	ترقيم تلقائي
FirstName	نص
LastName	نص
BirthDay	نص
City	نص

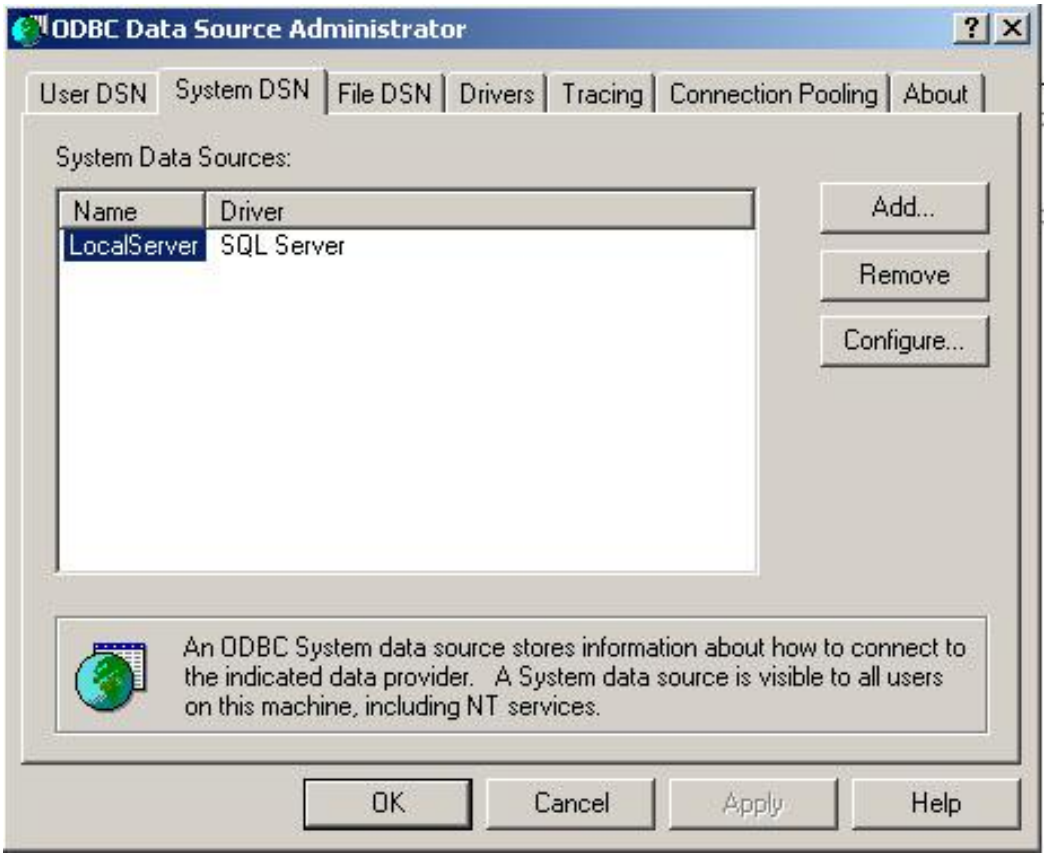
ولقد قمنا بإدخال البيانات التالية لكي نقوم بإجراء العمليات عليها :

ID	FirstName	LastName	BirthDay	City
1	ali	mohammad	12/8/1980	Riyadh
2	khaled	fahad	20/3/1990	Jeddah
3	tariq	nasser	5/5/1995	Dammam
(ترقيم تلقائي)				

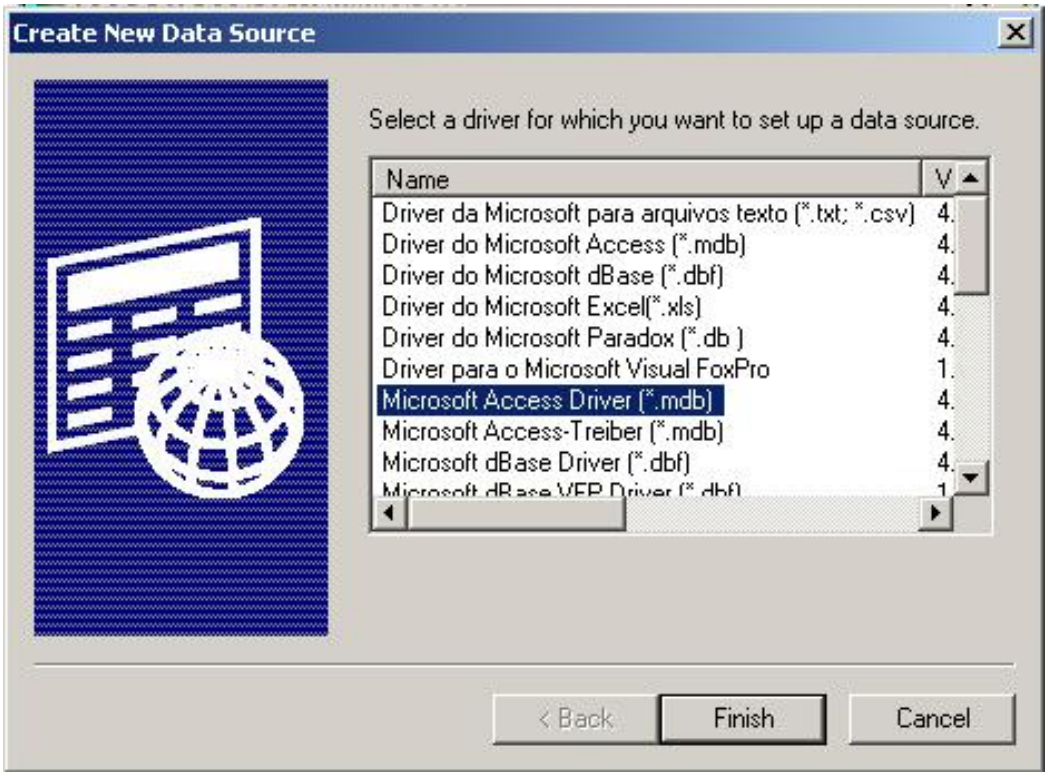
ملاحظة : قبل أن تقوم بتنفيذ البرنامج واستدعاء الـــServlet عبر العنوان URL . لابد من تهيئة مصدر البيانات عبر ODBC . ولعمل هذا الأمر تتبع الخطوات التالية :

- اختر قائمة start من ثم setting ومن ثم control panel ومن ثم administrative tools من ثم ODBC .

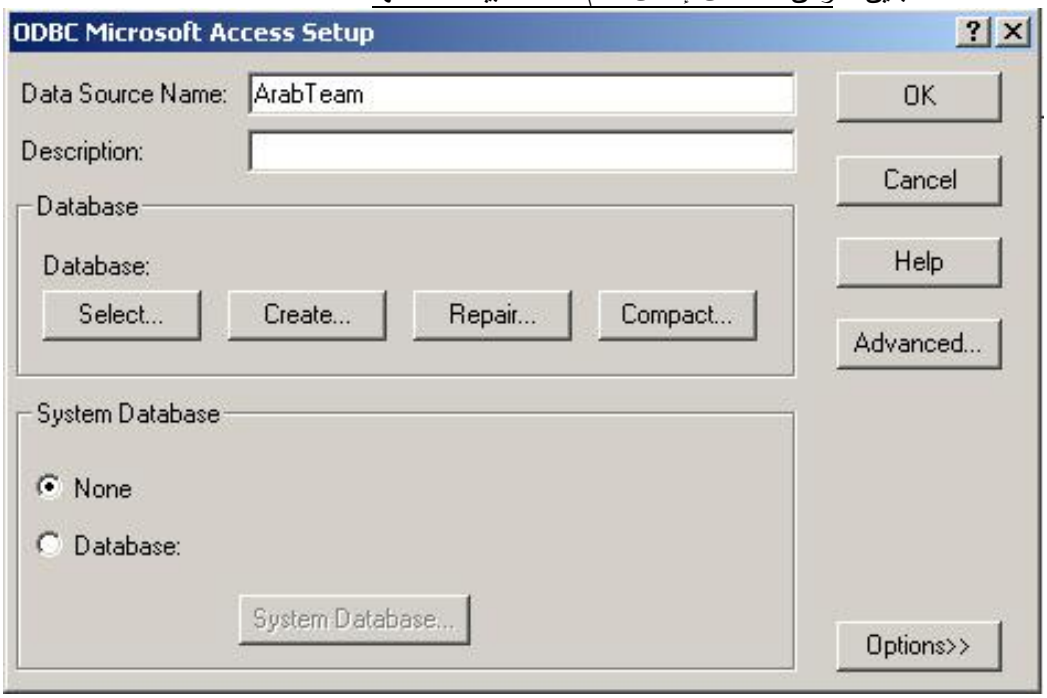
- سوف يعرض مربع حوار قم باختيار علامة SystemDSN ومن ثم قم باختيار Add .



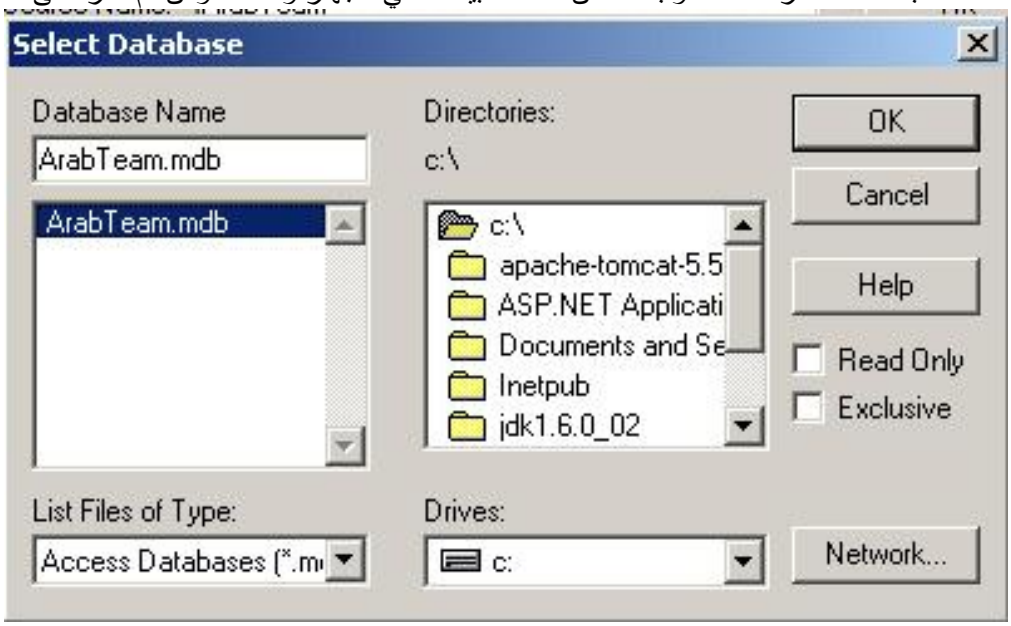
- قم باختيار Microsoft Access Driver



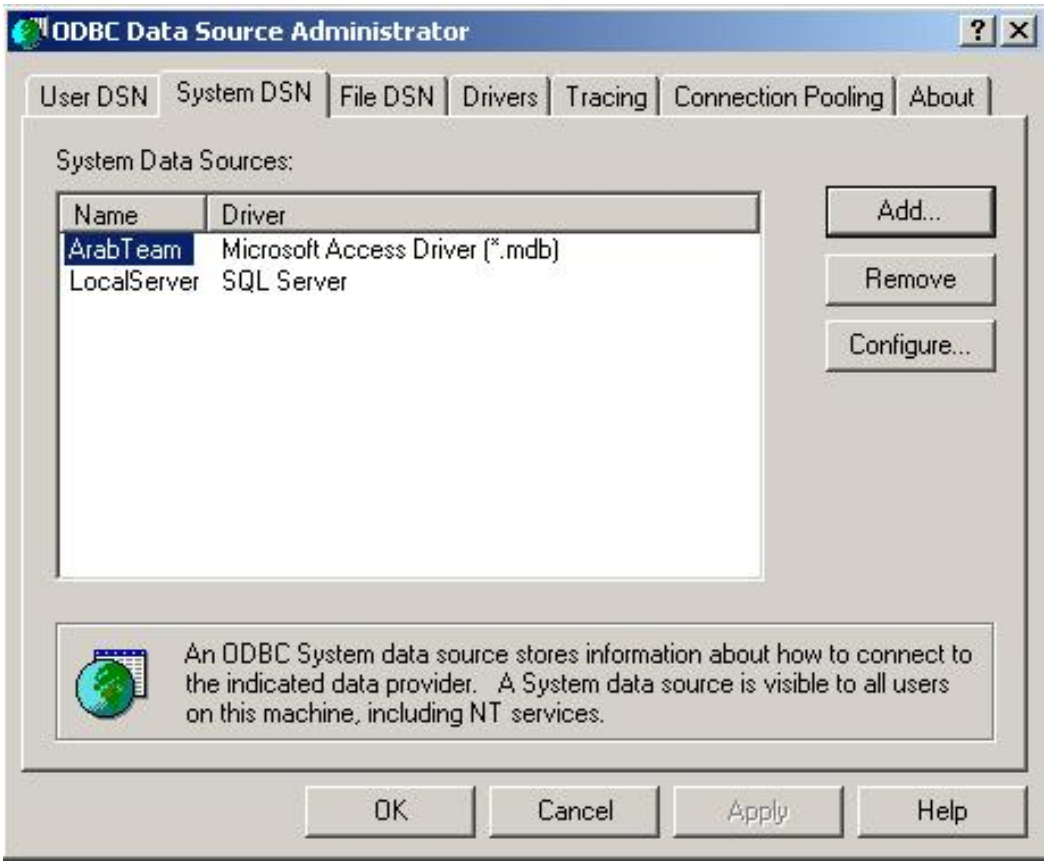
- من ثم قم بكتابة الاسم المناسب . ولكن يجب أن يكون الاسم المدخل والاسم في URL متطابقين . ومن الأفضل إدخال اسم قاعدة البيانات نفسها .



- بعد ذلك اختر select وابحث عن قاعدة البيانات في الجهاز وحددها ومن ثم انقر على OK



- الآن قم باختيار OK



الآن أصبحت قاعدة البيانات المنشئة قيد الاستخدام وجاهزة للتعامل مع الـــServlet وتلقي الأوامر خلال جمل الاستعلام . ولم يبقى سوى إنشاء الـــServlet لجلب البيانات .

١ . سوف نقوم بإنشاء Servlet وحفظة باسم ServletODBC . وبعد هذا الأمر نقوم بكتابة الدوال الرئيسية في هذا البرنامج وهي كالتالي :

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

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

}
}
```

٢ . بعد كتابة الدوال الرئيسية لابد من إعداد نوع مستند من اجل الرد على العميل لتكون محتوياته نص مجرد و HTML Code . وذلك بإضافة سطر التعليمة التالي إلى الـــServlet :

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

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

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

٣ . بعد ذلك لابد من إنشاء الكائن out من الصنف PrintWriter وذلك لتنفيذ أمر الطباعة . فيكون سطر التعليمة كالتالي :

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

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

response.setContentType("text/html");
PrintWriter out = response.getWriter();
}
}
```

٤ . بعد ذلك نقوم بتحميل البرنامج المشغل . فيكون سطر التعليمة التالي :

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

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

response.setContentType("text/html");
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

٥ . بعد ذلك نقوم بتعريف الاتصال أو الإشارة إلى مكان قاعدة البيانات . فيكون سطر التعليمة كالتالي :.

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

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

response.setContentType("text/html");
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc: ArabTeam";
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

٦ . بعد ذلك نقوم بتطبيق الاتصال أو نقوم بفعل اتصال حقيقي مع قاعدة البيانات . فيكون سطر التعليمة كالتالي :

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

public class ServletODBC extends HttpServlet {
public void doGet(HttpServletRequest request,
```

```
HttpServletResponse response)
throws ServletException, IOException {

response.setContentType("text/html");
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc: ArabTeam";
Connection = connection
=DriverManager.getConnection(url,"","");
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

٧ . بعد ذلك سوف نقوم بإنشاء جملة طلب ووضعها قيد الاستعمال . فيكون سطر التعليمة كالتالي :

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

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

response.setContentType("text/html");
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc: ArabTeam";
Connection = connection
=DriverManager.getConnection(url,"","");
Statement statement = connection.CreateStatement();
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

٨ . بعد ذلك نقوم بتنفيذ الاستعلام على قاعدة البيانات . وذلك بإنشاء جملة SQL ومن ثم تنفيذها . ويكون سطر التعليمة كالتالي :

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

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

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

```
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc: ArabTeam";
Connection = connection
=DriverManager.getConnection(url,"","");
Statement statement = connection.CreateStatement();
String Query = " select * from People"
ResultSet  resultset = statement.executeQuery(Query);
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

٩ . بعد ذلك نقوم بإجراء العمليات على النتائج . فيكون سطر التعليمة كالتالي :

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

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

response.setContentType("text/html");
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc: ArabTeam";
Connection = connection
=DriverManager.getConnection(url,"","");
Statement statement = connection.CreateStatement();
String Query = " select * from People"
ResultSet  resultset = statement.executeQuery(Query);
While ( resultset.next()) {
out.println(resultset.getSting(1)+" , "+
out.println(resultset.getSting(2)+…….);
}
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

١٠ . بعد ذلك وفي الأخير نقوم بإغلاق الاتصال مع قاعدة البيانات . فيكون سطر التعليمة كالتالي :

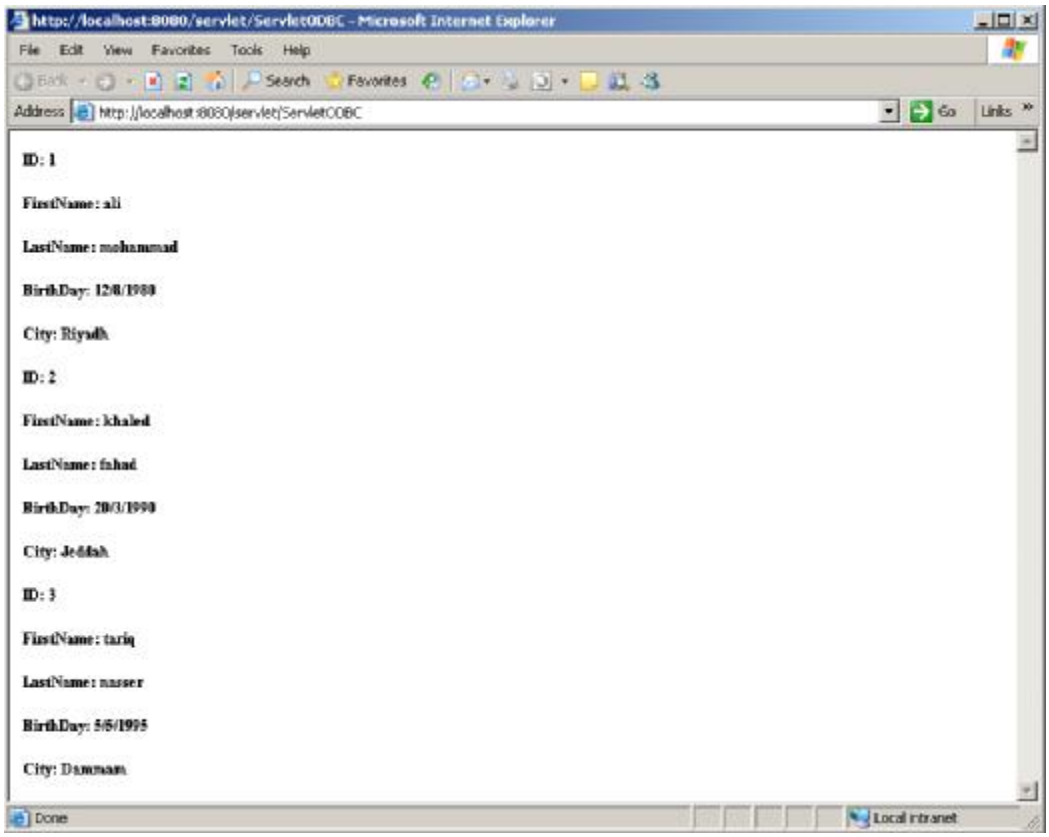
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletODBC extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
```

```
throws ServletException, IOException {

response.setContentType("text/html");
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc: ArabTeam";
Connection = connection
=DriverManager.getConnection(url,"","");
Statement statement = connection.CreateStatement();
String Query = " select * from People"
ResultSet  resultset = statement.executeQuery(Query);
While ( resultset.next()) {
System.out.println(resultset.getSting(1)+" ,
"+System.out.println(2)+…….);
}
connection.close;
} catch ( Exception e ) {
System.out.println( e) ;
}
}
}
```

والآن قم بترجمة ملف الــServlet ومن ثم قم بتشغيل الخادم . وفي الأخير قم بكتابة عنوان URL لترى نتيجة الطلب http://localhost:8080/servlet/ServletODBC . وهي كالتالي :



الفصل الخامس

تتبع فترة الاستخدام

يقصد بتتبع فترة الاستخدام هي الفترة أو المدة الزمنية التي يقضيها العميل وهو متصل بالخادم عبر البرتوكول HTTP . بمعنى آخر أن العميل عندما يقوم بالاتصال بالخادم فان البرتوكول يحدد زمن هذا الاتصال والفترة التي بقي فيها العميل متصل بالخادم بالإضافة إلى الصفحة التي قام بفتحها أو ذهب إليها والصفحة التي قام بإغلاقها أيضا . ولفترة تتبع الاستخدام أو العميل طرق وهي كالتالي :

- طرق تتبع فترة الاستخدام

١. ملفات تعريف الارتباط Cookies
٢. كتابة المعلومات في العنوان URL
٣. حقول النماذج المخفية .

- ملفات تعريف الارتباط

ويقصد بملفات تعريف الارتباط هي الملفات المؤقتة التي تكون تخزن في الذاكرة المؤقتة لجهاز الحاسوب . أي عندما يقوم العميل زيارة موقع مثلاً فليزمة كتابة اسم المستخدم وكلمة المرور لدخول الموقع . وإذا قام العميل بزيارة الموقع مرة أخرى فلا يلزمه كتابة اسم المستخدم وكلمة المرور بسبب أن البيانات المتعلقة بالمستخدم (العميل) قد تكونت في جهاز الحاسوب . ولذا فلن يطلب هذا الموقع اسم المستخدم وكلمة المرور الخاصة بالعميل إذا قام بزيارة الموقع مرة أخرى كمثل منتدى الفريق العربي للبرمجة . والشكل العام لتركيب ملفات تعريف الارتباط كالتالي :

```
String sessionID = makeUniqueString();
HashMap = sessionInfo = new HashMap();
HashMap globalTable = findTableStoringSession();
globalTable.put(sessionID , sessionInfo);
Cookie sessionCookie = new Cookie("JsessionID" , sessionID);
sessionCookie.setPath("/");
response.assCookie(sessionCookie);
```

- كتابة المعلومات في العنوان URL

وهذه الطريقة هي أن يقوم العميل بإضافة بعض البيانات المتعلقة به إلى العنوان URL . ومثال ذلك كالتالي :

http://SomeWhere.file.html?**JseesionID=5555**

وهذه الطريقة جيدة حينما لا يدعم بعض المستعرضات ملفات تعريف الارتباط .

- حقول النماذج المخفية

وهذه الطريقة هي أن يكون هنالك حقل مخفي في النموذج وتوضع في هذا الحقل أي معلومة حول الجلسة المتعلقة بالمستخدم . ومثال ذلك كالتالي :

```
<input type="hidden" name="session" value="JsessionID">
```

لن نتطرق إلى الثلاث أنواع السابقة بالتفصيل ولكن سوف نتطرق إلى شيء آخر معتمد على ملفات تعريف الارتباط وكتابة المعلومات في العنوان دون تخزين معلومات المستخدم . ويرمز لهذا بالرمز session من الكائن HttpSession . والسبب في ذلك هو أن هذا الكائن يستخدم غالبا في المواقع ذات الحساسية الكبرى كالمواقع الحكومية وغيرها . وهذه المواقع لابد لها من تتبع المستخدم من أين دخل ومن أي خرج والوقت الذي قضيه في الموقع بالإضافة إلى عدم إعطاءه الصلاحيات لدخول المواقع إلا من خلال الصفحات المعتادة كصفحات اسم المستخدم وكلمة المرور .

- مبادئ متابعة الجلسات باستخدام الكائن (HtpSession)

وتتلخص هذه المبادئ في أربع خطوات ، وهي كالتالي :

١ . مناداة كائن الجلسة الخاص بمتابعة الجلسة ، كما في التالي :

```
HttpSession session = request.getSession();
```

٢ . حفظ المعلومات المتعلقة بالجلسة في نفس الكائن ، كما في التالي :

```
Session.setAttribute("AnyName","AttributeValue");
```

حيث ترمز AttributeValue إلى المتغير الذي وضع فيه القيمة . و AnyName اسم المتغير الجديد الذي سوف يستخدم من قبل مصمم صفحات الويب .

٣ . البحث عن المعلومة المتعلقة بالجلسة ، كما في التالي :

```
Session.getAttribute("AnyName");
```

٤ . أزاله بيانات الجلسة ، وهنا تنقسم هذه الإزالة إلى نوعين .

الأولى : وهي إزالة قيمة موجودة في الكائن ، كما في التالي :

```
Session.removeAttribut("AnyName");
```

الثانية : وهي أزاله الجلسة كاملة بما تحتويها من قيم ومتغيرات : كما في التالي :

```
Session.invalidate();
```

حيث لا يأخذ النوع الثاني أي قيمة بل سوف يلغي الكائن HttpSession من الجلسة نهائياً .

الفصل السادس

نظرة عامة - Java Server Page

تعتبر تقنية JSP (Java Server Page) تقنية بديلة لتقنية الــServlet . وهذه التقنية تمكن من المزج بين صفحات الانترنت المكتوبة بلغة HTML وبين شيفرة Java كما في تقنية الــServlet . ولقد أتت هذه التقنية وحلت مكان التقنية السابقة لأسباب عديدة . منها :

١. أن الــServlet هو عبارة عن برنامج مكتوب بلغة Java يحتوي على شيفرة HTML . أما الــJSP هو عبارة عن صفحة انترنت ولكن يتم إدراج شيفرة Java بداخلها .
٢. أن الــServlet ضعيفة من ناحية العرض . أما في الــJSP فهي جيدة من ناحية العرض
٣. أن في الــServlet لابد أن يعمل مبرمج صفحات الانترنت بجانب مبرمج Java حين إنشاءه . أما في الــJSP فيمكن تقسيم العمل بحيث يتم تصميم صفحات الانترنت من مصمم صفحات الانترنت ومن ثم يتم إدراج شيفرة Java بواسطة مبرمج الصفحات .
٤. أن في الــServlet سوف يكون من الصعب على مبرمج صفحات الانترنت التعديل عليه من ناحية شيفرة HTML ويلزمه أن يكون خبيراً في الــJava . أما في الــJSP فعكس ذلك فلا يلزمه أن يكون خبيرا بلغة Java.

ومن خلال هذه الفروقات البسيطة نستنتج أن تقنية JSP هي أسهل وابسط بكثير من حيث البرمجة ومن حيث التصميم بسبب أنها عبارة عن صفحة انترنت اعتيادية

والتركيب العام لصفحة الــJSP كما في التالي :

```
<%@ page contentType="text/html; charset=windows-1256"
language="java"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title></title>
</head>
<body>
</body>
</html>
```

- حيث يرمز charset=windows-1256 إلى نوع الترميز للصفحة
- حيث ترمز التعليمة language="java" إلى اللغة المستخدمة في الصفحة
- حيث ترمز التعليمة contentType="text/html إلى نوع مستند الرد على العميل والجدول التالي يوضح بعض الأوامر التي يمكن من خلالها الرد على العميل

نوع الملف	تعليمة الرد
ملفات مايكروسوفت وود – word	Application/msword
ملفات قارئ الكروبات - pdf	Application/pdf
ملفات مايكروسوفت اكسل – excel	Application/vnd.ms-excel
ملفات مايكروسوفت بويربوينت	Application/vnd.ms-powerpoint
ملف صوتي بصيغة wav	Audio/x-wav
صورة بتنسيق GIF	Image/gif
صورة بتنسيق JPEG	Image/jpeg
مستند HTML	Text/html
مستند xml	Text/xml

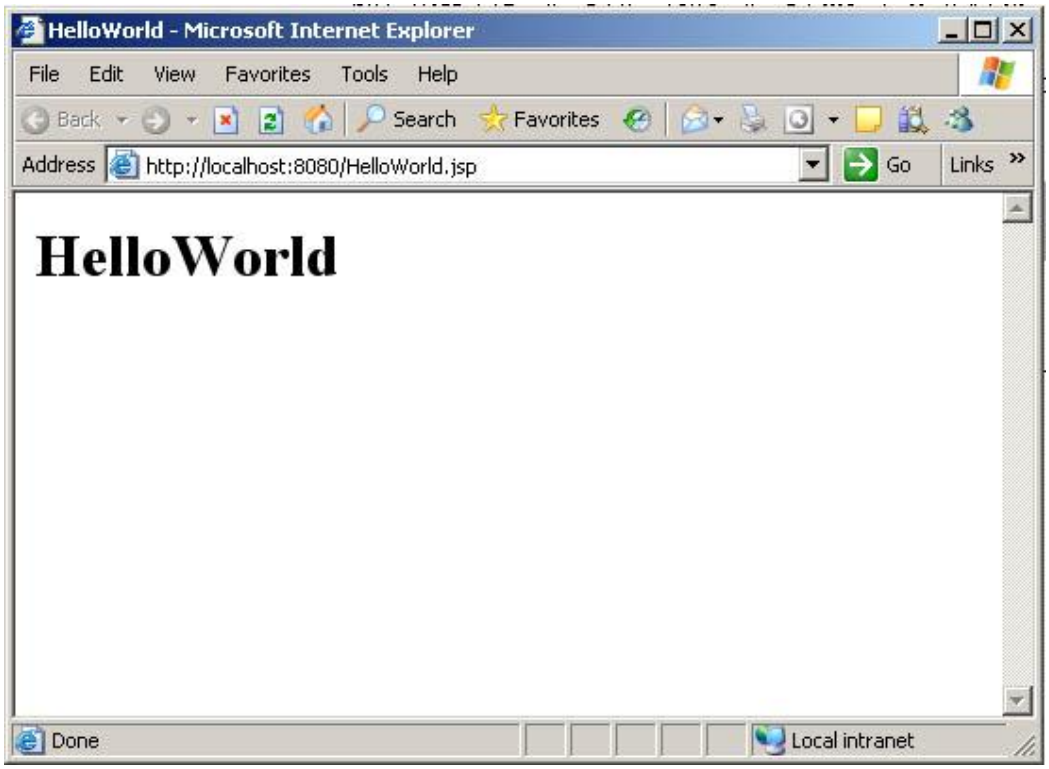
والمثال التالي يوضح كيفية استخدام هذه التقنية جنبا الى جنب مع HTML .

تطبيق ٤ : طباعة جملة Hollow World

HellowWorld.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>HelloWorld</title>
</head>
<body>
<h1><%out.println("HelloWorld");%></h1>
</body>
</html>
```

وفيما يلي عرض صفحة HellowWorld.jsp



وكما تلاحظ . أن صفحة الــJSP قد قامت بما قام به الــServlet وهو الطباعة . وناتج الصفحة هذا هو مشابهه تماماً لناتج الصفحة في التطبيق رقم ١ والذي قمنا بإنشائه سابقاً . ولكن هنا وبالتحديد في صفحة الــJSP قد قمنا بكتابة تعليمة واحدة وهي : **`out.println("HelloWorld")`**
وأخرجت لنا ما أخرجه الــServlet في التطبيق رقم ١ . ومن هنا يخرج لنا سؤال إلا وهو : كيف عرف الخادم بان هذه الصفحة هي صفحة HTML محتويه على شيفرة Java . أي أنها صفحة JSP ..؟

الجواب : هو أن أي صفحة JSP هي بالأصل عبارة عن Servlet . ولهذا فان صفحة الــJSP تحتوي على متغيرات معرفة مسبقا . طبعاً هي غير مشاهدة ولهذا فان الخادم يقوم بكتابتها بدلاً عنك عند ترجمة الصفحة بصورة مخفية . وهذه هي وظيفة الخادم الأساسية وهي : تحويل صفحات الــJSP إلى Servlet . ومن أهم الكائنات المعرفة مسبقا في صفحة الــJSP هي كما في التالي :

```
HttpServletRequest  من request
HttpServletResponse من response
HttpSession من session
PrintWriter من out
```

ملاحظة : لهذه التعريفات المسبقة يمكن استخدام أي كائن من الكائنات السابقة مباشرةً دون الحاجة إلى تعريفها في صفحة الــJSP .

وهنا قد يتساءل المبرمج . ما الفائدة من خروج تقنيتان مع أنهما يستعملان لأمر واحد وهو تطبيقات الانترنت ويؤديان نفس الغرض . السبب يكمن أن الــServlet يعتبر أفضل من الــJSP عندما تكون

الشيفرة البرمجة بلغة Java كبيرة وهو حل امثل لتلقي البيانات ومعالجتها من قبل العميل . أما الـــJSP فانه يستخدم لعرض المخرجات للعميل . أي أن التقنيتين مكملتان لبعضهما . فالأولى تستعمل لاستقبال الطلبات القادمة من العميل والأخرى لعرض المخرجات للعميل . ومن هنا يأتي التكامل بين هاتين التقنيتين .

الفصل السابع

استخدام Java Server Page

في هذا الفصل سوف تتعرف على تقنية JSP بالتفصيل وكيف أنها حلت مكان الـــServlet واستخدامها لغرض عرض الصفحات بالإضافة إلى بعض الميزات التي من الصعب إنشاءها في الـــServlet . والبرمجة بها تنقسم إلى ثلاث أنواع : برمجة التعابير ، برمجة المعرفات ، برمجة البرمجيات النصية.

• التعابير (Expression)

يقصد بالتعابير هي حشر القيم في صفحة الـــJSP ، بمعنى آخر أن التعابير تستخدم لأمر الطباعة . فهي تحل مكان التعليمة `out.print(),out.println()` . والتركيب العام للتعابير كما في الشكل التالي :

```
<%= Java Statement%>
```

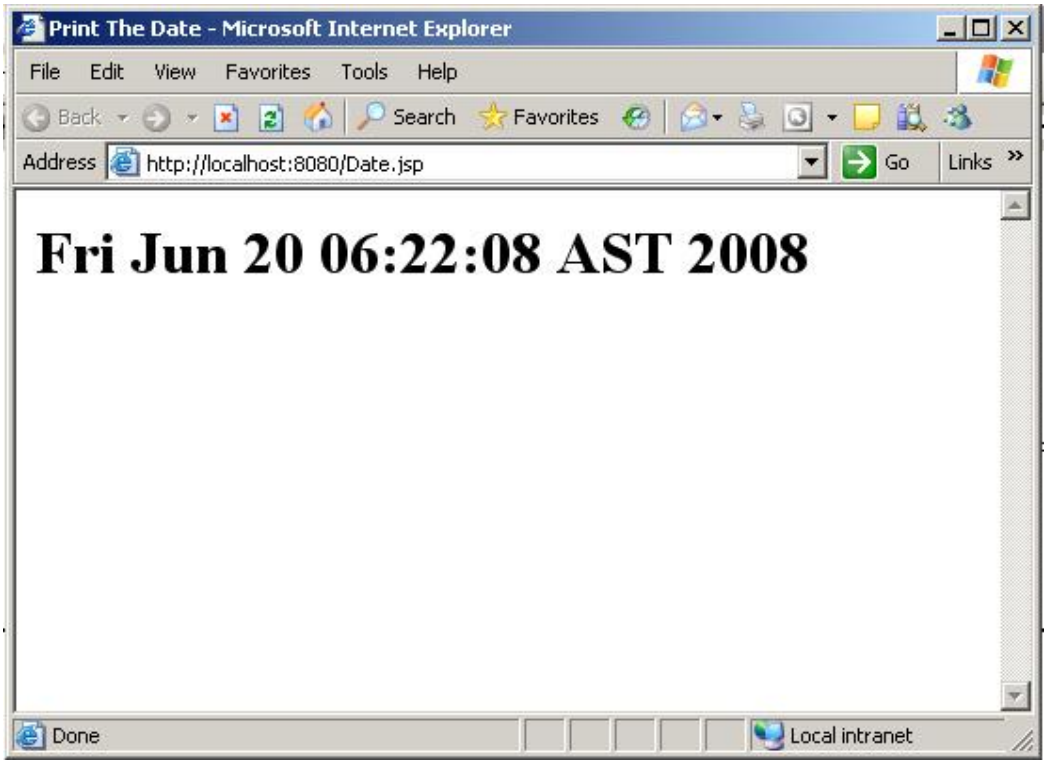
والتطبيق التالي يوضح كيفية استخدام هذا النوع في طباعة التاريخ الحالي

تطبيق ٥ : طباعة التاريخ

Date.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" import="java.util.Date" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Print The Date</title>
</head>
<body>
<h1><%= new java.util.Date()%></h1>
</body>
</html>
```

وفيما يلي عرض صفحة Date.jsp



وكما تلاحظ في هذا التطبيق أن التعابير قد قامت بطباعة التاريخ الحالي ، ولهذا تعتبر طريقة بديلة ومناسبة لطريقة `out.print()` مع أن كلاهما يؤديان نفس الأمر وهو الطباعة . ولكن إذا كانت لديك صفحة JSP تستخدم فيها الكثير من أوامر الطباعة فمن غير المنطقي بل من المستحيل تضمين سطر التعليمة في السابق عند كل أمر طباعة وإلا لأصبحت صفحة الـــJSP في نهاية المطاف Servlet بسبب كثرة هذه الأوامر في الصفحة . وسوف يكون من الملل فعل ذلك بالإضافة إلى الاستفادة الكاملة من لغة HTML من ناحية التنسيق .

• المعرفات (Declatations)

يقصد بالتعريفات هي القيمة المسندة إلى كتلة (class) . بمعنى آخر عندما نريد أن نعرف قيمة مثلاً ولتكن Name وهذه القيمة نصية . فلابد من إسنادها إلى كتلة class من نوع String .
والتركيب العام للتعريفات كما في الشكل التالي :

```
<%! SomeClass Name = "AnyThing"%>
```

ومثالها :

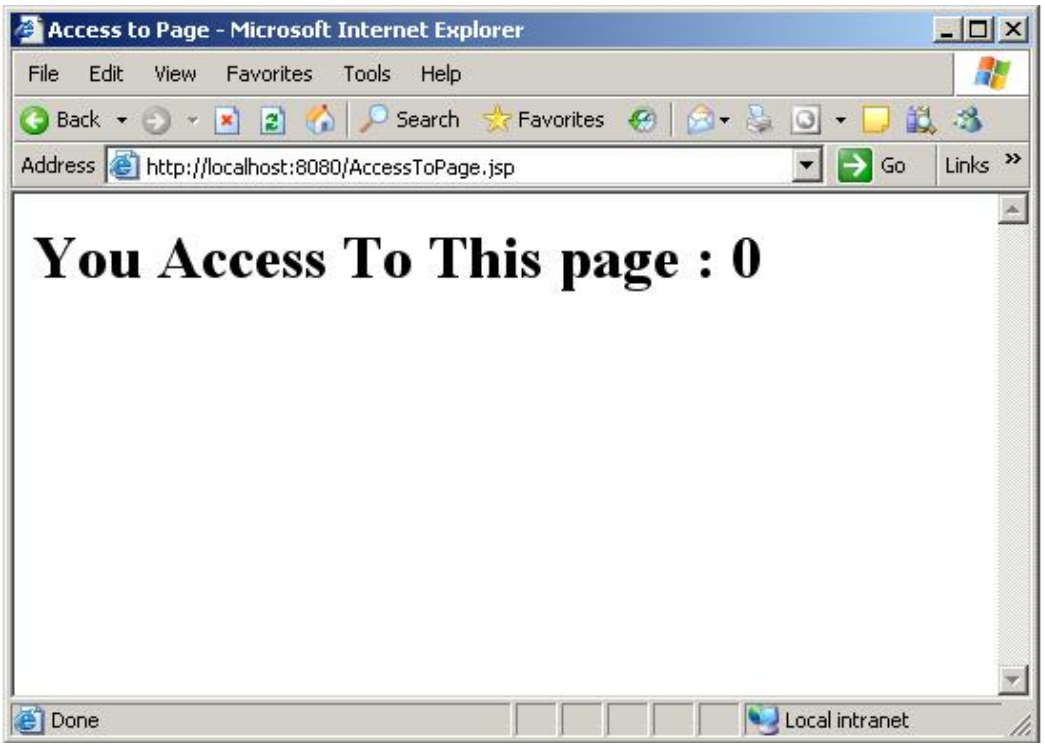
```
<%! String Name = "ArabTeam" %>
```

والتطبيق التالي يوضح كيفية استخدام هذا النوع في طباعة عدد الزيارات للصفحة

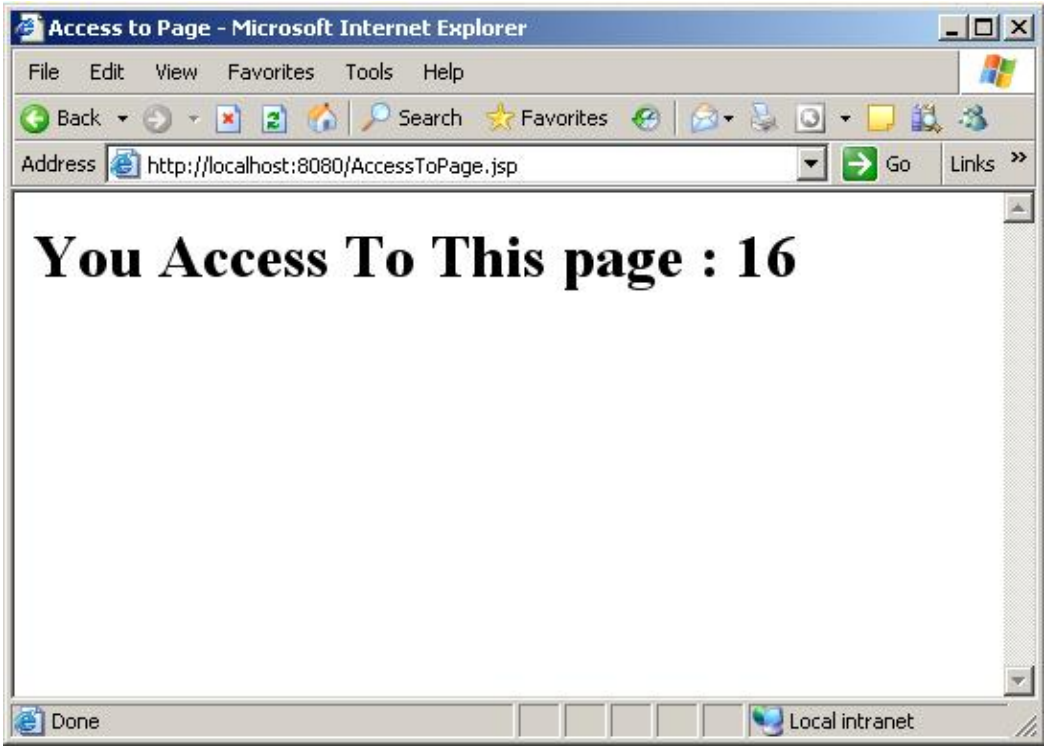
تطبيق ٦ : عدد الزيارات للصفحة
AccessToPage.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" import="java.sql.*" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Print The Date</title>
</head>
<body>
<h1>
<%! int accesstopage = 0; %>
You Access To This page :
<%= accesstopage++ %>
</h1>
</body>
</html>
```

وفيما يلي عرض صفحة AccessToPage.jsp في بداية التشغيل



وبعد النقر على زر تحديث عدة مرات سوف يزيد عدد مرات طلبك الصفحة وبالتالي سوف تزيد عدد زيارتك للصفحة . وفيما يلي عرض الصفحة بعد النقر على زر تحديث عدة مرات :



سؤال : عند النقر على الزر تحديث فانه يقوم بطلب الصفحة مرة أخرى . وبالتالي سوف يقوم بترجمة الشيفرة مرة أخرى . ولكن في نفس الصفحة قد عرفنا المتغير accesstopage من نوع الكتلة int وأسندنا القيمة صفر إليه . لماذا عند النقر على زر تحديث لا يقوم بتعريف المتغير مرة أخرى ويضع قيمة المتغير صفر في كل مرة عند كل تحديث على عكس ما يحدث انه يزيد القيمة في كل مرة واحداً ..؟

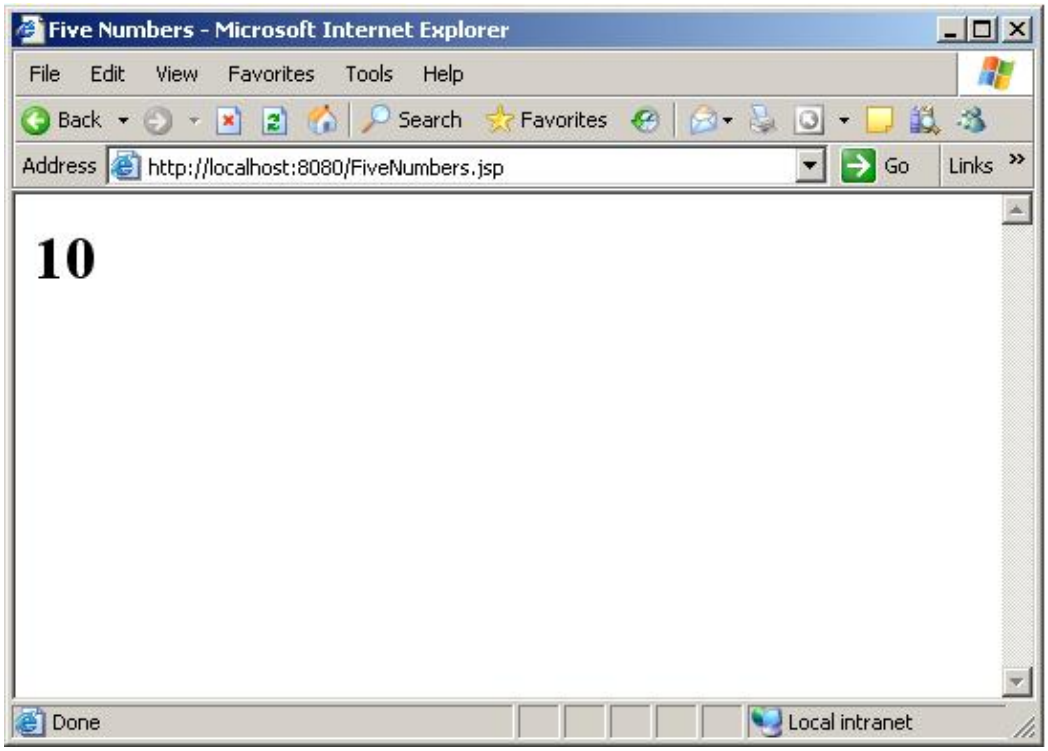
- البرمجيات النصية (Scriplets)

في هذا القسم وهو البرمجيات النصية لن يكون له قاعدة معينه أو يكون له تركيب عام كالنوعين السابقين ... فالنوع الأول وهو التعابير هو خاص بطباعة النتائج في صفحة الـــJSP وله تركيب عام لذلك . أما النوع الثاني فهو خاص بتعريف المتغيرات في صفحة الـــJSP وله تركيب عام لذلك . ولكن هذا النوع ليس له خصوصية معينه أي عندما مثلا تريد أن تكتب شيفرة أكثر تعقيد فلابد من استخدام هذا النوع . فعندما نريد أن نصمم صفحة تجمع خمس أرقام عن طريق For Loop فان النوعين السابقين لا يستطيعان القيام بهذا الأمر . وفي هذا النوع يمكن أيضا استخدام IF Statement و غيرها من الشيفرات المعقدة الخاصة بلغة Java . ويمكن اعتبار هذا النوع كأي تطبيق اعتيادي مكتوب بلغة Java . وفي هذا التطبيق سوف ننشئ صفحة JSP تقوم على جمع خمس أرقام .

تطبيق ٧ : حاصل جمع خمس أرقام
FiveNumbers.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Print The Date</title>
</head>
<body>
<%! int sum = 0; %>
<%
for ( int i = 0 ; i < 5 ; i++){
     sum = sum + i;
}
%>
<h1><%= sum %></h1>
</body>
</html>
```

وفيما يلي عرض صفحة FiveNumbers.jsp



ولكن في النوع هذا وهو البرمجيات النصية لم ينشئ لهذا الغرض فحسب . وهو عملية جمع خمس أرقام أو غير ذلك بل للشيفرة الأكثر تعقيداً . مثلاً إذا كنا نريد أن نقيس درجة طالب في مادة معينه وهل هذا الطالب ناجح أم راسب في المادة . فان البرمجيات النصية تستطيع أن تتعامل مع هذا

التعقيد المشروط (٦٠ وما فوق ناجح – اقل من ٦٠ راسب) بعكس النوع الأول وهو التعابير أو النوع الثاني وهو التعريفات . وفي هذا التطبيق سوف نقوم بحساب درجة طالب في مادة معينه وهل هو ناجح أم راسب باستخدام احد الشروط والتعليمة Math.random() .

تطبيق ٨ : حساب درجة طالب ResultOfDegree.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Result Of Degree</title>
</head>
<body>
<% int Degree = (int) (Math.random() * 100); %>
<% if ( Degree >= 60 ) { %>
<h1>الطالب ناجح : <%= Degree%></h1>
<% } else { %>
<h1>الطالب راسب : <%= Degree%></h1>
<% } %>
</body>
</html>
```

وفيما يلي عرض صفحة ResultOfDegree.jsp عند بداية التشغيل

وفيما يلي عرض صفحة ResultOfDegree.jsp بعد النقر على زر تحديث عدة مرات

وكما تلاحظ في هذا التطبيق أن البرمجيات النصية والكتلة أصبحت أكثر تعقيداً مما قبل . وفي هذا التطبيق لقد أخرجنا درجة الطالب عشوائياً ومن ثم وضعناها في داخل كتله IF Statement وأخذنا نقيس الدرجة . إذا كانت الدرجة اكبر أو تساوي ٦٠ فالطالب ناجح . وإذا اقل فالطالب راسب . بالإضافة إلى أننا قمنا بوضع نوع داخل نوع آخر . بمعنى أننا قمنا بوضع نوع التعابير داخل نوع البرمجيات النصية حينما أردنا إظهار الدرجة للطالب **<%= Degree%>** . وأصبح النوع الأول وهو التعابير معتمد على عمل البرمجيات النصية في الصفحة نفسها . بحيث سوف تظهر لنا الدرجة في كل مرة وهل هذه الدرجة درجة النجاح أم درجة الرسوب على أساس الشرط في كتله البرمجيات النصية ...!!

وان عدنا إلى نفس التطبيق أيضا لرأيت شيء آخر مختلف عما كنا نتبعه . فلقد عرفنا المتغير Degree داخل كتلة البرمجيات النصية وليس داخل كتلة المعرفات . فكما عرفت سابقاً أننا عندما نريد أن نقوم بأي تعريف فلابد من وضعه داخل كتلة المعرفات . وفي هذا التطبيق اختلف الأمر . وعمل هذه الصفحة يقودنا إلى سؤال مهم وهو ما الفرق . ما الفرق بين التعريف في داخل كتلة المعرفات ومابين التعريف داخل كتلة البرمجيات النصية ..؟

أي ما الفرق بين الشكلين التاليين ..؟

```
<%! int Degree = (int) (Math.random() * 100); %>
<%  int Degree = (int) (Math.random() * 100); %>
```

الفصل الثامن

Java Server Pape – تقنيات أخرى

في هذا الفصل سوف تتعرف على تقنيات أخرى خاصة بتقنية JSP . أي بعض الأوامر التي يمكن إضافتها في الصفحة لكي تعمل بأكثر جودة مما قبل . هذه التقنيات متنوعة وكثيرة ويقد لا نستطيع الإلمام بها من كثرتها . ولكن سوف نورد هنا التقنيات الأكثر شيوعا والأكثر استخداما عند مطوري الويب ومستعملي هذه التقنية وهي JSP . وهي عادة تضاف إلى سطر التعليمة الأول في كل صفحة JSP

• خاصية الاستدعاء (import Atributte)

وتسمح هذه الخاصية بالاستدعاء في الصفحة وبالتحديد استعداء الأصناف الموجودة في مكتبة الجافا . فمثلا عندما نريد أن نقوم بطباعة التاريخ في الصفحة ، فلابد من جلب صنف التاريخ من مكتبة الجافا وذلك باستخدام import Attibute . ومثالها كالتالي :

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" import="java.sql.*" %>
```

وكما تلاحظ أن التعليمة السابقة سوف تجلب لنا الصنف المراد استعماله في الصفحة . ولكن إذا أردنا أن نقوم بجلب أكثر من صنف من الكتبة لكي تعمل جميع الأصناف في الصفحة . فهل لابد أن اكتب التعليمة السابقة عند جلب كل صنف . هنا يمكن الجواب بلا . والسبب يمكن أن التعليمة السابقة أيضا تسمح لنا بجلب أكثر من صنف . وذلك بوضع علامة الفاصلة ومن ثم كتابة الصنف المراد جلبه للصفة . مثالها كالتالي :

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" import="java.sql.*,java.io.*"  %>
```

وكما تلاحظ أن التعليمة قد جلبت لنا صنفين . وقد تجلب لنا أكثر من عشرة أصناف أيضا وذلك بنفس الطريقة وهي كتابة الفاصلة بعد كل صنف وكتابة الصنف المراد جلبه للصفحة .

• خاصية إدراج الصفحات (include fle)

وتسمح لنا هذه الخاصية بإدراج الصفحات في صفحة الـــJSP . فعندما يكون لدينا أكثر من صفحة نريد إظهارها للعميل في وقت واحد . فهذه الخاصية تسمح لنا بهذا الأمر . والتركيب العام كما في التالي :

```
<%@include file="pageName"%>
```

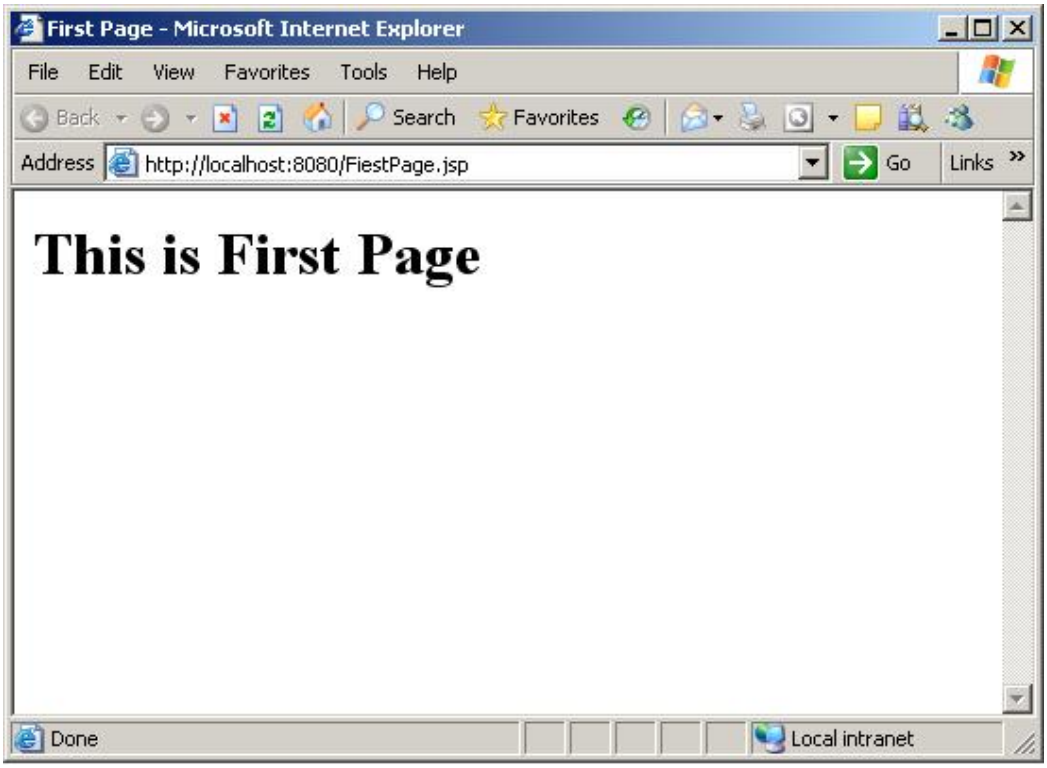
وفيما يلي تطبيق سوف نقوم بإنشاء ثلاث صفحات ومن ثم طلبها أو إدراجها من خلال الخاصية include file في الصفحة الرئيسية .

تطبيق ٩,١ : الصفحة الأولى

FirstPage.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>First Page</title>
</head>
<body>
<h1>This is First Page</h1>
</body>
</html>
```

وفيما يلي عرض للصفحة

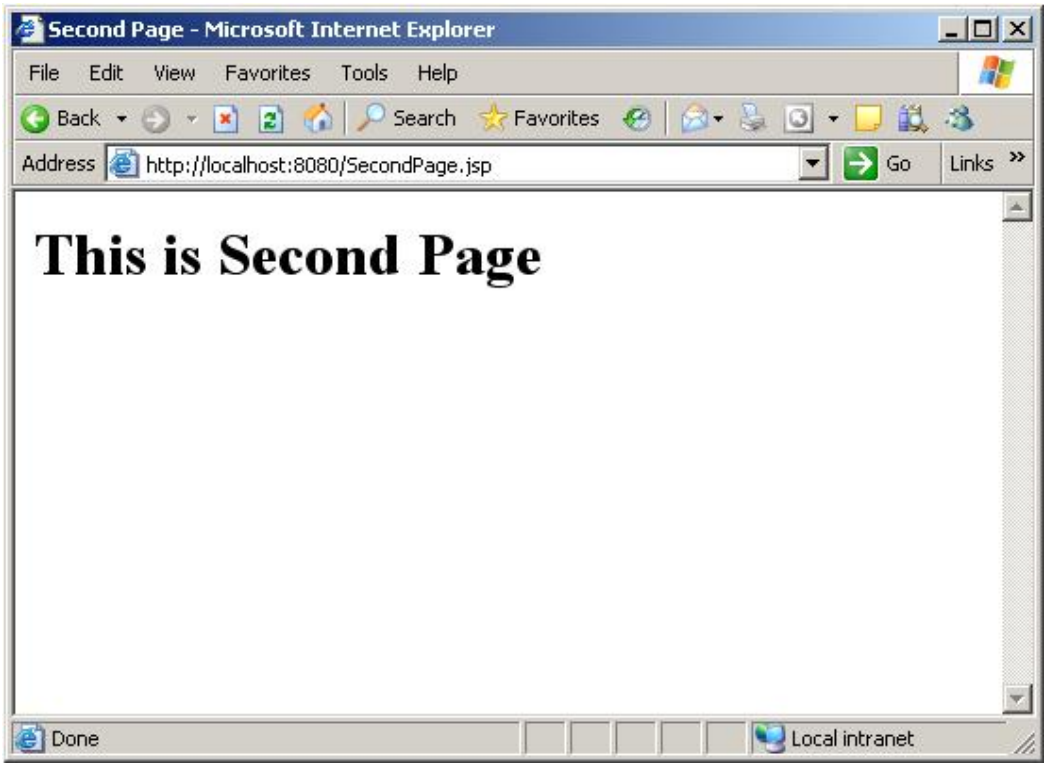


تطبيق ٩,٢ : الصفحة الثانية

SecondPage.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Second Page</title>
</head>
<body>
<h1>This is Second Page</h1>
</body>
</html>
```

وفيما يلي عرض للصفحة



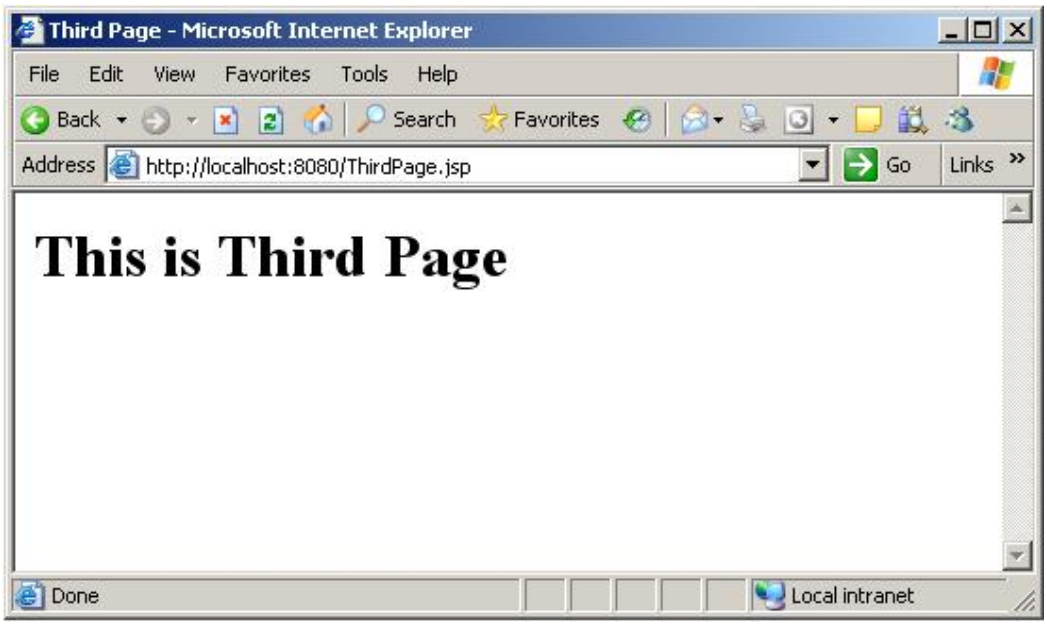
تطبيق ٩,٣ : الصفحة الثالثة

ThirdPage.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Third Page</title>
```

```
</head>
<body>
<h1>This is Third Page</h1>
</body>
</html>
```

وفيما يلي عرض للصفحة

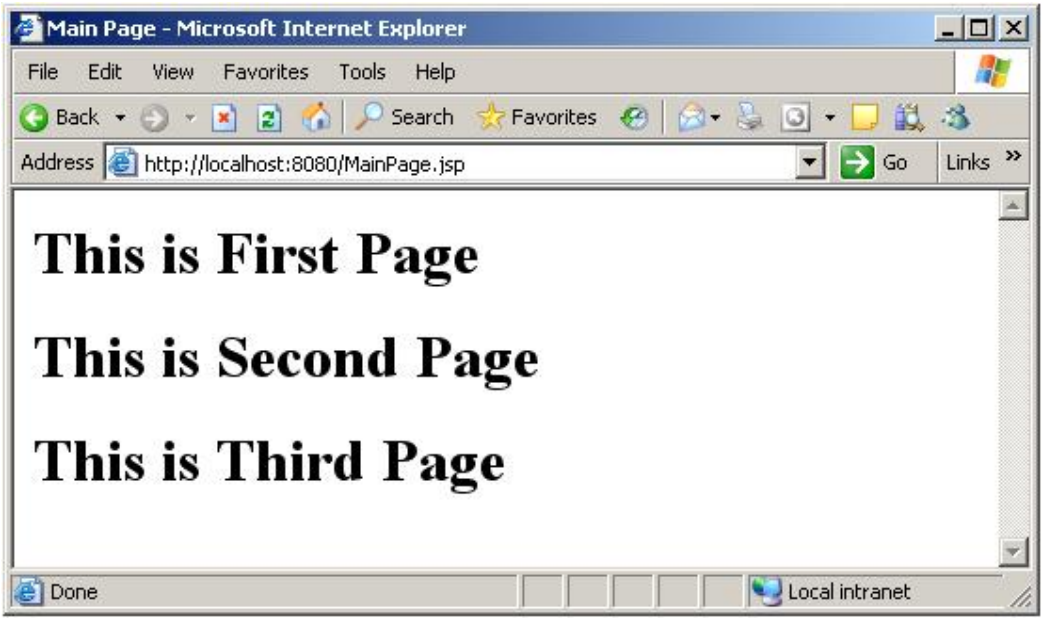


والآن سوف نقوم بإنشاء الصفحة الرئيسية والتي تتضمن سطر التعليمة include file والذي يمكنا من إدراج الصفحات السابقة في صفحة واحدة

تطبيق ٩,٤ : الصفحة الرئيسية MainPage.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Main Page</title>
</head>
<body>
<%@ include file="FiestPage.jsp"%>
<%@ include file="SecondPage.jsp"%>
<%@ include file="ThirdPage.jsp"%>
</body>
</html>
```

وفيما يلي عرض للصفحة الرئيسية MainPage.jsp



- خاصية فترة الاستخدام (session Atribut e)

وتسمح لنا هذه الخاصية بالسماح أو إنهاء فترة الاستخدام أو فترة التتبع في الصفحة . فمن المعروف وكما ذكر سابقا عند أي إنشاء أي صفحة JSP يكون فيها بعض التعريفات دون الحاجة إلى التعريفات من قبل المبرمج . والتركيب العام كما في التالي :

```
<% page session = "true" %>
```

أو

```
<% page session = "false" %>
```

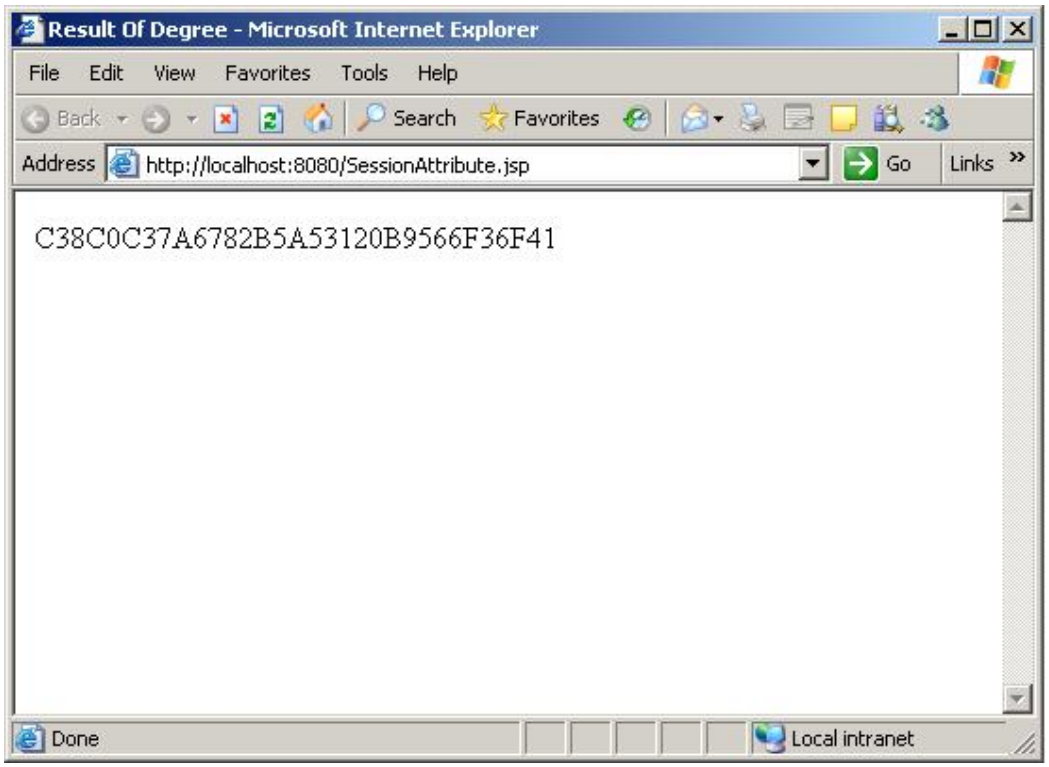
وكما تلاحظ في التركيب العام السابق . ففي التركيب الأول لقد كافئنا قيمة التتبع أو فترة الاستخدام بـــtrue . وهذا يعني أننا قمنا بتفعيل هذه الخاصية في صفحة JSP ويمكن أيضا استعمالها . والتطبيق التالي يوضح لنا كيف يتم تفعيل هذه الخاصية .

تطبيق ١٠,١ : تفعيل فترة التتبع
SessionAttribute.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" session="true"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Result Of Degree</title>
</head>
<body>
```

```
<%= session.getId()%>
</body>
</html>
```

وفي هذا التطبيق وكما تلاحظ لقد قمنا بتفعيل خاصية فترة التتبع ومن ثم قمنا بجلب القيمة الموجودة في هذا المتغير وهو session . وفيما يلي عرض لصفحة SessionAttribute.jsp أو لنتيجة الرد



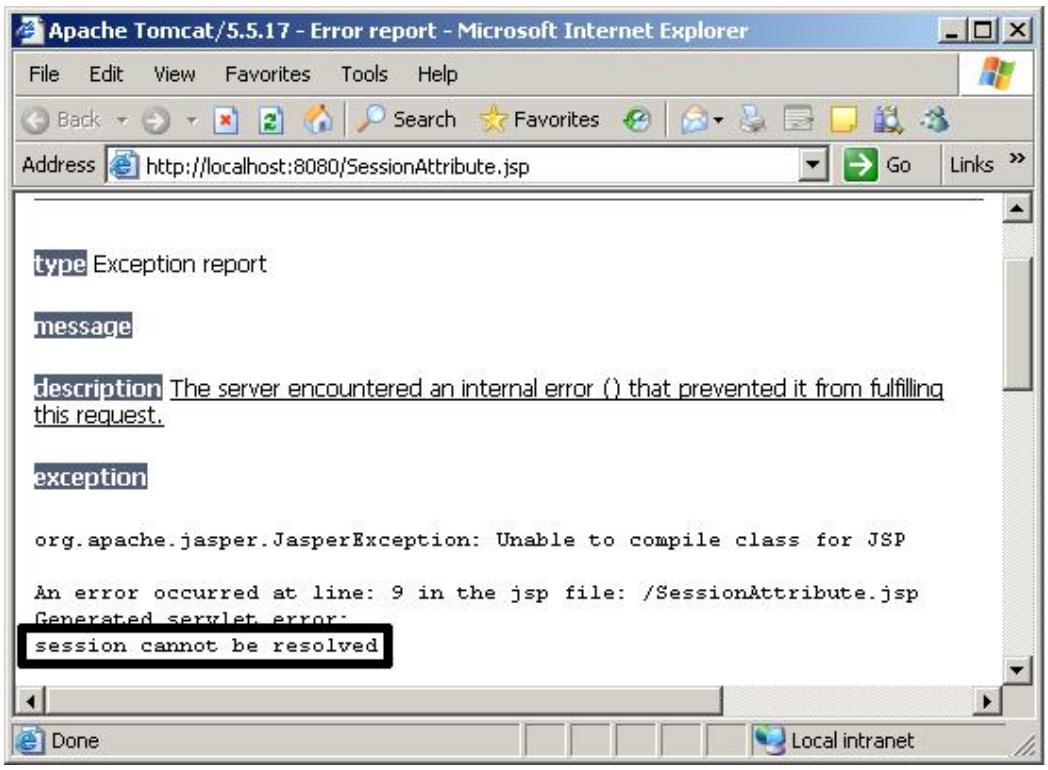
وكما تلاحظ في نتيجة الرد أننا قمنا بجلب القيمة من كائن فترة التتبع وقمنا بإظهاره . وهذا يعني أن خاصية فترة التتبع مفعلة في الصفحة ويمكن استخدامها . وفي التطبيق التالي سوف نستخدم هذه الخاصية ولكن سوف نكافئها بكلمة false أي أننا سوف نقوم بتعطيلها .

تطبيق ١٠,٢ : تعطيل فترة التتبع SessionAttribute.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" errorPage="" session="false"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>Result Of Degree</title>
</head>
<body>
<%= session.getId()%>
```

```
</body>
</html>
```

وفيما يلي عرض نتيجة الطلب للصفحة SessionAttribute.jsp



وكما تلاحظ هنا قد خرج لنا خطا في نتيجة الرد والسبب كما هو واضح في الصفحة أن المترجم لم يستطيع قراءة القيمة الموجودة session من خلال session.getId() . وسبب هذا الخطأ هو أننا قمنا بتعطيل هذه الخاصية في الصفحة وذلك بمكافئتها بكلمة false .

الفصل التاسع

تطبيق شامل على الكتاب

في هذا الفصل وهو الفصل الأخير من هذا الكتاب ، سوف نقوم ببناء تطبيق شامل على جميع الفصول السابقة وجميع ما ذكر فيها من نقاط . وهذا التطبيق سهل جدا وبسيط وهو عبارة عن صفحتان . الأولى : وهي عبارة عن صفحة دخول ، والثانية : وهي عبارة عن صفحة عرض .

بمعنى أن في هذا التطبيق سوف يكون لدنيا صفحة دخول مكونه من اسم المستخدم وكلمة المرور . ولدينا مستخدمين موجودون في قاعدة البيانات . إذا قام احد المستخدمين بالدخول أو المرور عبر صفحة الدخول فان التطبيق سوف يقوم بتخزين وقت دخول المستخدم في قاعدة البيانات ومن ثم إرساله إلى صفحة العرض و إظهار وقت دخوله مباشرةً . وفي صفحة العرض سوف يكون لدنيا زر باسم خروج . عن الضغط عليه سوف يقوم التطبيق بتخزين وقت خروج المستخدم ومن ثم إرجاعه إلى صفحة الدخول . مع مراعاة منعه من العودة على صفحة العرض من خلال طلب الصفحة مباشرةً عن طريق العنوان أو النقر على زر للخلف . وفي هذا التطبيق سوف نستخدم تقنية JSP فقط دون الحاجة إلى الـــServlet مع قاعدة البيانات SQLSERVER باستخدام تزويد المحرك (JDBC Driver) .

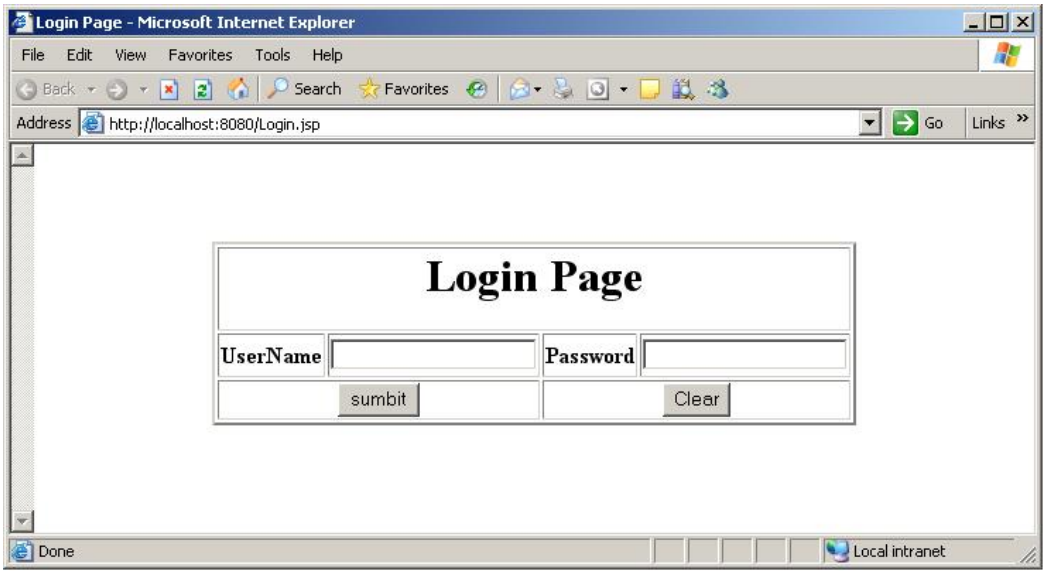
وفيما يلي الصفحات .

تطبيق ١١,١ : صفحة الدخول Login.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" dir="rtl">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256" />
<title>Login Page</title>
</head>
<body>
<br /><br />
<h1></h1>
<form>
<table align="center" width="60%" border="2">
<tr>
<th colspan="4"><h1>Login Page</h1></th>
</tr>
<th><input type="password" /></th>
<th>Password</th>
<th><input type="text" /></th>
<th>UserName  </th>
</tr>
<tr>
```

```
<th colspan="2"><input type="reset"  value="Clear" /></th>
<th colspan="2"><input type="submit" value="sumbit" /></th>
</tr>
</table>
</form>
</body>
</html>
```

وفيما يلي عرض الصفحة Login.jsp



تطبيق ١١,٢ : صفحة العرض History.jsp

```
<%@ page contentType="text/html; charset=windows-1256"
language="java"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1256">
<title>History Page</title>
</head>
<body>
<form>
<table align="center" width="80%" border="2">
<tr>
<th colspan="4"><h1>History Page</h1></th>
<tr>
<th>UserName</th>
<th>SessionID</th>
<th>Login Time</th>
```

```
<th>Logout Time</th>
</tr>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
<tr>
<th colspan="4"><input type="submit" value="Logout" /></th>
</tr>
</table>
</form>
</body>
</html>
```

وفيما يلي عرض للصفحة History.jsp

وفيما يلي عرض للمستخدمين المسجلين في قاعدة البيانات .

	ID	username	password
	1	arabteam	111
	2	servlet	123
	3	jsp	952
▶*	NULL	NULL	NULL

وفيما يلي عرض للجدول الذي سوف نقوم بتخزين بيانات دخول المستخدم وخروجه من الصفحة

	ID	username	sessionID	login_time	logout_time
*	NULL	NULL	NULL	NULL	NULL

ومن خلال الصفحات السابقة وقاعدة البيانات لقد جهزنا بيئة العمل لتطوير الصفحات للعمل بالشكل المطلوب . والآن سوف نقوم بإدراج التعليمات البرمجية إلى الصفحات . ولتكن البداية من صفحة الدخول . وكما تلاحظ في صفحة الدخول لقد وضعنها جميع الأزرار والحقول وغيرها داخل الوسم Form . لأنه وما هو متعارف يعتبر كالحاوية لنقل البيانات من صفحة إلى أخرى . ومن خلال الوسم Form سوف نقوم بتحديد الصفحة التي سوف تستقبل البيانات بالإضافة إلى تحديد طريقة الإرسال هل هي post أو get . ولهذا فيكون سطر التعليمة كالتالي .

```
<form action="Login.jsp" method="post">
```

وكما تلاحظ لقد أضفنا إلى الوسم خاصيتين الأولى وهي action وقمنا بتحديد الصفحة التي سوف تستقبل البيانات عند النقر على زر submit . والخاصية الثانية وحددنا فيها طريقة إرسال البيانات بصورة مخفية من خلال post . ولكن من المهم أن

المصادر

برمجة الانترنت - خالد الحامد

كتاب المبرمج – خالد عبدالستار و أسامه العبدالله

منتديات الفريق العربي للبرمجة

أبو سعود الفرنسي
nickniacky@hotmail.com