ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

Προγραμματισμός στο Διαδίκτυο

Java Servlets

Μανώλης Μαραγκουδάκης

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

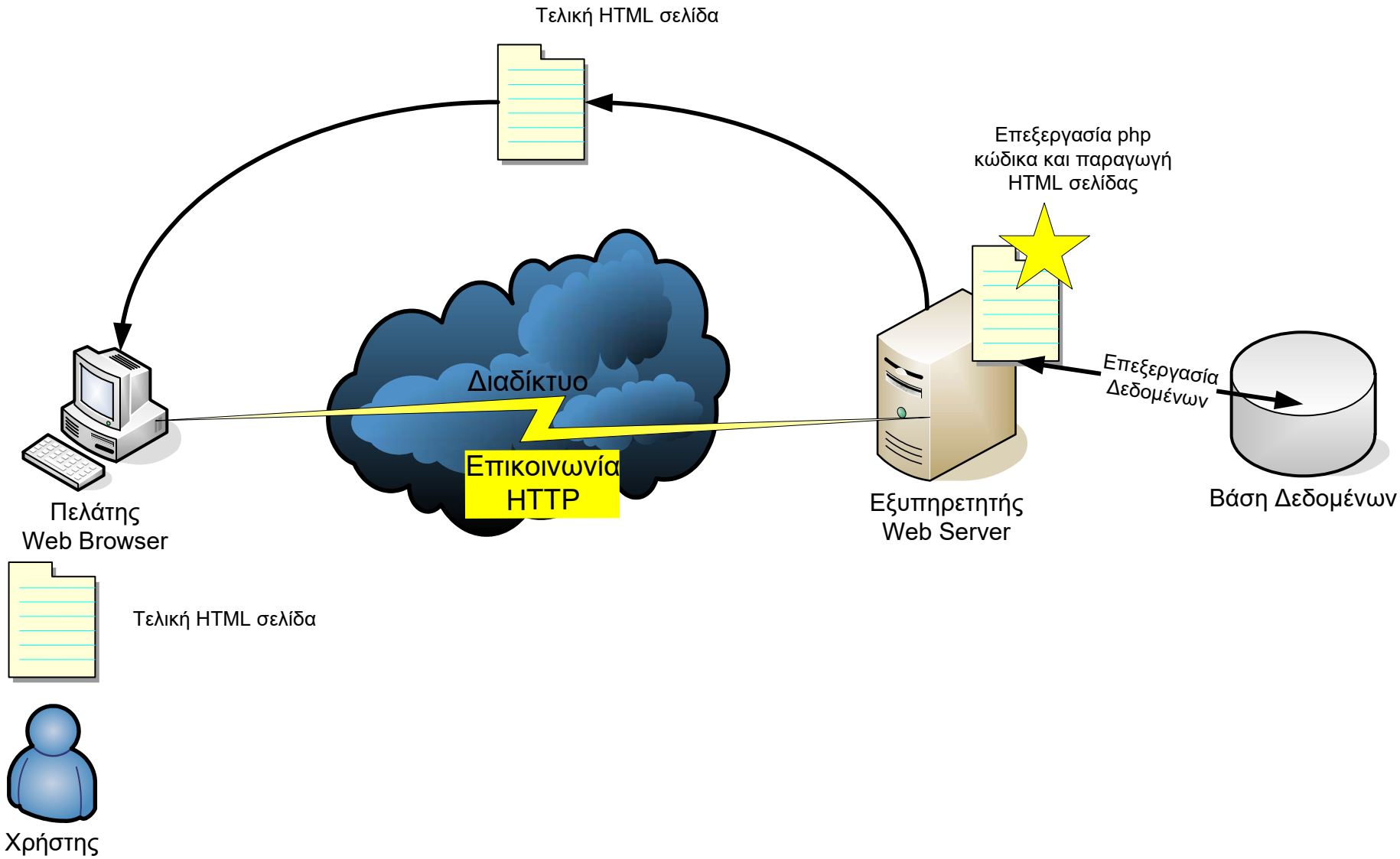
Προγραμματισμός στο Διαδίκτυο

Ενότητα 6 – Java Servlets

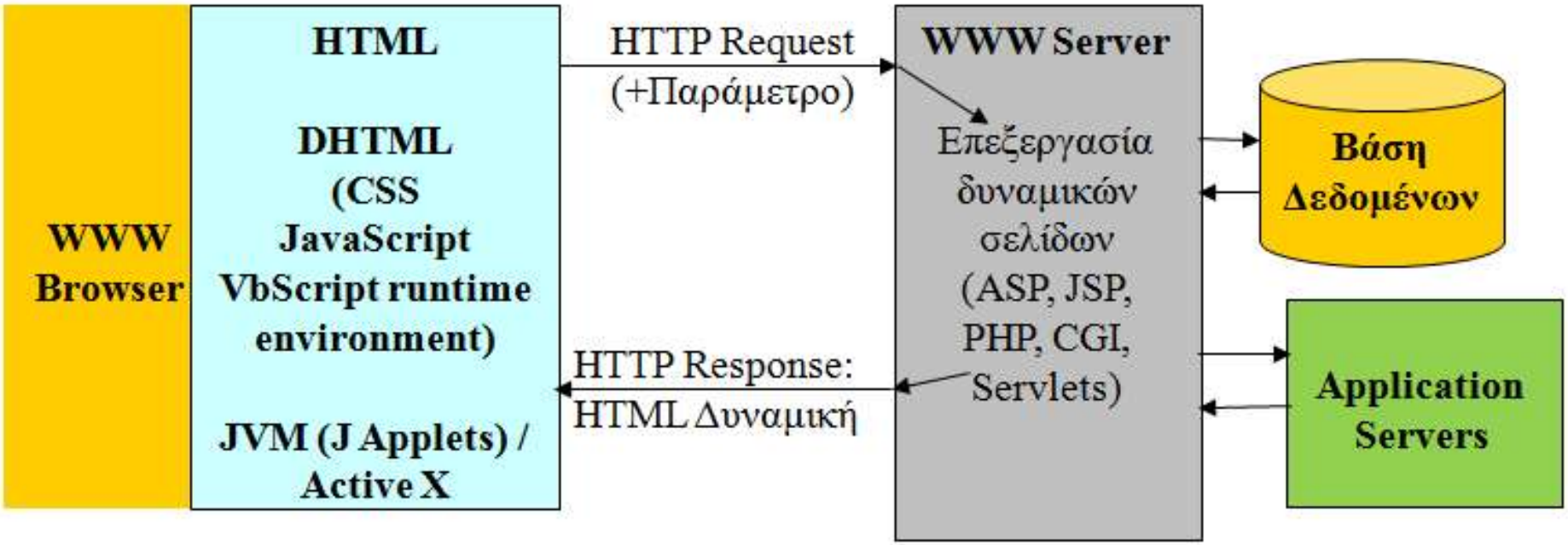
Μανώλης Μαραγκουδάκης

Πανεπιστήμιο Αιγαίου
Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

Προγραμματισμός στην πλευρά του εξυπηρετητή (server side)



Προγραμματισμός στην πλευρά του εξυπηρετητή (server side)



Server Side: Καταλληλότητα, Πλεονεκτήματα, Μειονεκτήματα

- Καταλληλότητα:
  - Δυναμική / Παραμετρική εμφάνιση περιεχομένου
  - Απαραίτητο όταν απαιτείται επικοινωνία (αλληλεπίδραση) με τον Server
  - Δυνατότητα ελέγχου των πελατών, π.χ. μετρητές επισκέψεων (hit counters), ελεγχόμενη πρόσβαση σε κάποιες σελίδες
- Πλεονεκτήματα:
  - Η επεξεργασία μεταφέρεται στο server, χρησιμοποιείται η ισχύς του server
  - Ο κώδικας είναι κρυφός
  - Η εκτέλεση του κώδικα είναι ανεξάρτητη του browser: στέλνεται «καθαρό» HTML που εμφανίζεται πανομοιότυπο σε κάθε browser
  - Η μοναδική λύση για πρόσβαση στο file system του server
- Μειονεκτήματα:
  - Χρησιμοποιεί πολύτιμη επεξεργαστική ισχύ του server.
  - Κλιμάκωση (scalability);

Server Side: Τεχνολογίες

Τεχνολογία	Server	Πλεονεκτήματα
Active Server Pages (.asp)	MS IIS (Internet Information Server)	Σχετικά εύκολο στη χρήση, καλή ενσωμάτωση/ ολοκλήρωση (integration), καλή υποστήριξη (support) από Microsoft
ASP.NET	IIS	Κλιμάκωση (Scalability), web services.
Java Server Pages (.jsp)		Κλιμάκωση, αξιοπιστία
PHP (.php)	Apache (open source)	Δωρεάν, εύκολη στη χρήση
Java Servlets	Any web server	Προγραμματισμός σε Java, Εύκολη πρόσβαση σε ΒΔ

Τι είναι Servlet (SERVer appLET)

- Τα Servlets είναι μικρά προγράμματα γραμμένα στη γλώσσα Java που λειτουργούν στον server και επεκτείνουν τις λειτουργίες ενός Web Server.
- Όπως και οι άλλες αντίστοιχες τεχνολογίες (CGIs, ASP, PHP, ..), χρησιμοποιείται για την δημιουργία Web σελίδων που το περιεχόμενό τους δεν είναι στατικό αλλά μπορεί να εξαρτάται από τα δεδομένα που εισαγάγει ο χρήστης και χρειάζεται να ανακτηθεί από ΒΔ ή από άλλα συστήματα

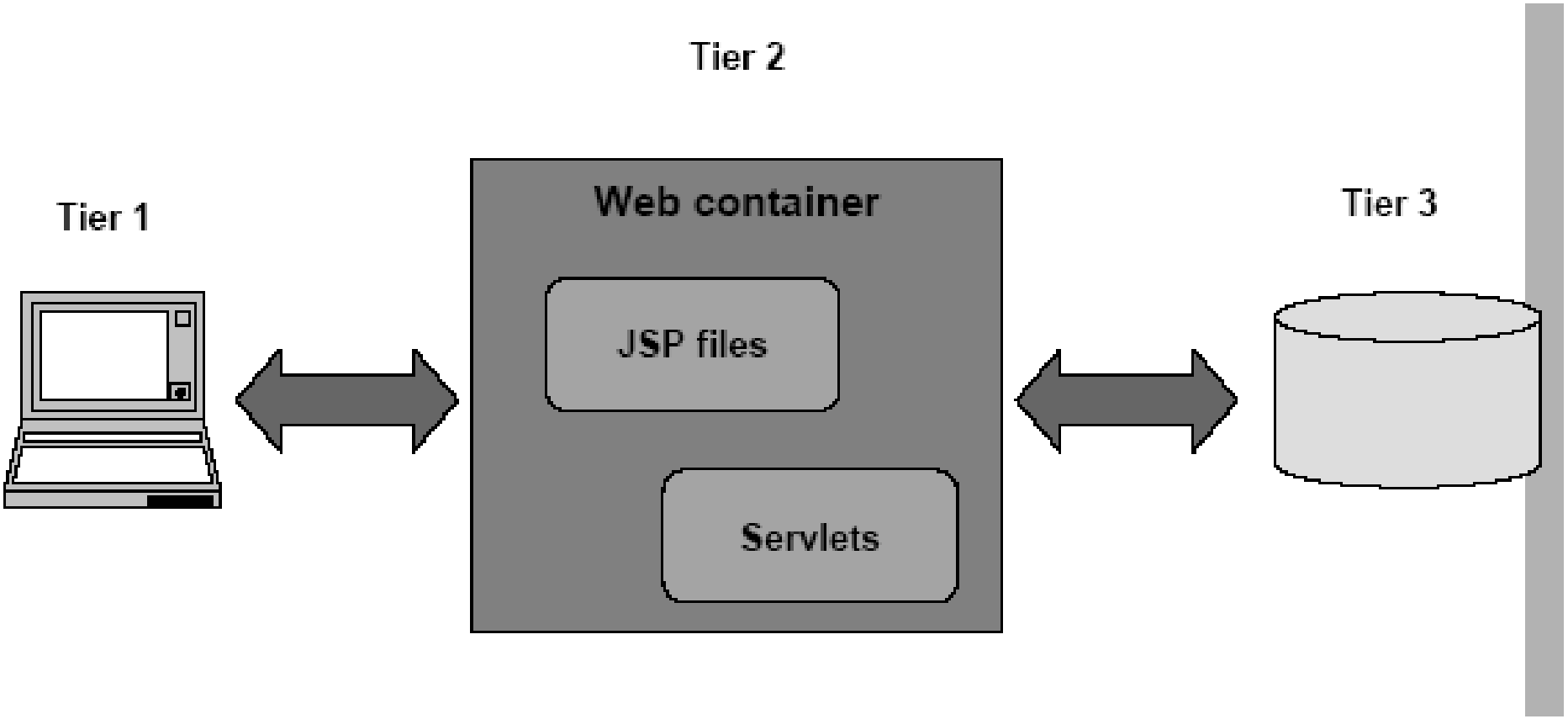
Τι είναι Servlet (SERVer appLET)

- Εκτελούνται σε ένα Web Server
  - Servlets: server-side πρόσωπο της Java
  - Applets: client-side πρόσωπο της Java, δηλ. εκτελούνται σε Web Browsers
- Τα Servlets είναι εγκατεστημένα σε Web Server, δέχονται δεδομένα μέσω του πρωτοκόλλου HTTP και απαντούν στέλνοντας στον Web Browser αρχεία τύπου HTML.
- Για να προγραμματίσουμε Servlets είναι απαραίτητο το JSDK (Java Servlet Development Kit) ή Servlets API (Application Programming Interface) που είναι ενσωματωμένο σε αρκετά εργαλεία προγραμματισμού Java (π.χ. NetBeans)
- Το Servlets API αποτελεί πλέον μέρος του JDK
- Τα Servlets υποστηρίζονται από (μπορούν να τρέξουν σε) σε πολλούς web servers (π.χ. Apache, Microsoft IIS, κλπ.)

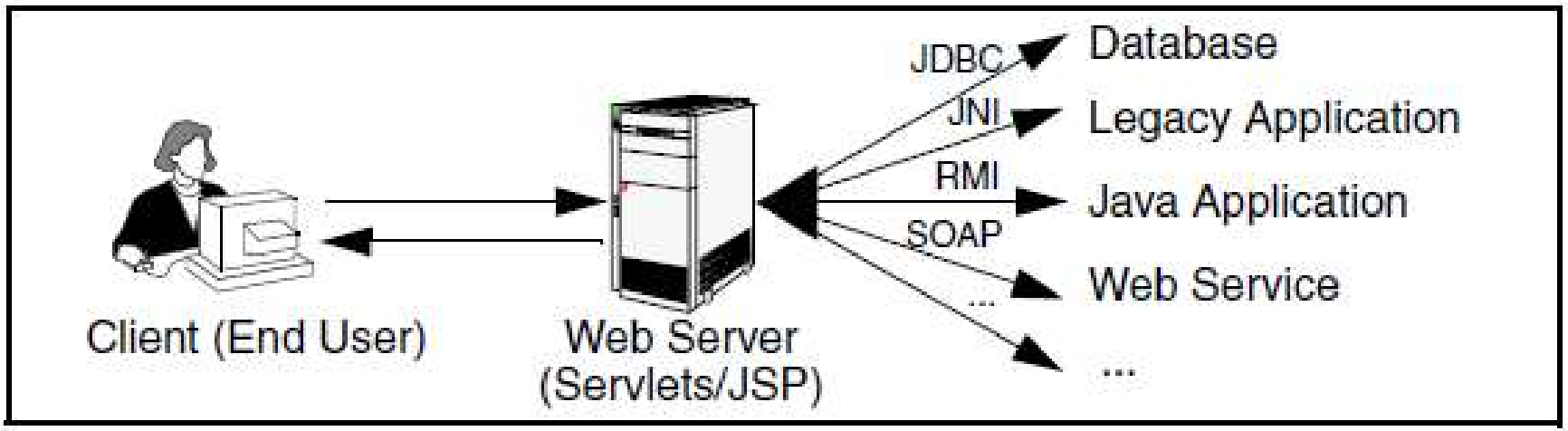
Java Server Pages (JSP)

- Η απάντηση της Java στο ASP, PHP κτλ.
- Μεγάλο μέρος μίας δυναμικής σελίδας είναι στατικό.
- Δεν υπάρχει λόγος το servlet μας να δημιουργεί το στατικό μέρος μίας σελίδας με println.
- Μία JSP σελίδα περιέχει κανονικό HTML κώδικα για τα στατικά μέρη της σελίδας, και ενσωματωμένο κώδικα Java για τα δυναμικά μέρη (όπως και στο PHP).
- Στην πραγματικότητα, την πρώτη φορά που θα ζητηθεί μία JSP σελίδα, αυτή μεταγλωττίζεται σε servlet. Σε κατοπινά αιτήματα λειτουργεί σαν κανονικό servlet.

Java Servlets σε μια 3-tier αρχιτεκτονική



Ο ρόλος του ενδιάμεσου λογισμικού Ιστού



Η δύναμη των Servlet

- Είναι γραμμένα σε γλώσσα Java κατά συνέπεια είναι platform independent: “Write once Serve Everywhere”
- Εκμεταλλεύονται πλήρως το Java API, RMI, CORBA, Database Connectivity.
- Αποδοτικότητα & Αντοχή : Μένουν στην μνήμη μεταξύ διαδοχικών καλεσμάτων
- Κομψότητα (Elegance), Object-Oriented, Clean Code, Modular, Simple
- Λειτουργούν με το πρωτόκολλο HTTP

PHP vs. Servlets

- Τα servlets και τα PHP scripts αποτελούν εναλλακτικές για server-side προγραμματισμό.
- Τα servlets «φορτώνονται» μία φορά και όχι κάθε φορά που καλούνται, αντίθετα με τα PHP scripts
- Τα servlets είναι τεχνολογία που βασίζεται σε μια πλήρη αντικειμενοστρεφή γλώσσα (Java), η PHP είναι γλώσσα σεναρίου (script)
- Υπάρχει διαφορά στη λογική: η PHP μοιάζει περισσότερο με την τεχνολογία JSP, ο PHP κώδικας είναι ενσωματωμένος σε HTML κώδικα, το στατικό HTML διακρίνεται από το HTML που παράγει δυναμικά η PHP. Οι servlets αποτελούν Java κλάσεις που όταν εκτελούνται παράγουν HTML κώδικα (στατικό & δυναμικό)

PHP vs. Servlets

- Οι κλήσεις σε «έτοιμες» (built-in) συναρτήσεις της PHP (που περιλαμβάνονται στις βιβλιοθήκες της PHP) είναι συνήθως γρηγορότερες από κλήσεις σε συναρτήσεις των Servlets. Το αντίστροφο όμως ισχύει για τον επιπλέον κώδικα που γράφει ο προγραμματιστής PHP.
- Η PHP προσφέρεται για γρήγορη ανάπτυξη κώδικα λόγω απλής σύνταξης
- Τα servlets προσφέρονται για μεγαλύτερης κλίμακας έργα λόγω της εκμετάλλευσης των πλούσιων βιβλιοθηκών αλλά και της αντικειμενοστρεφούς (object-oriented) φύσης της Java.
- Με τους servlets είναι εύκολη η μετάβαση από μια ΒΔ σε άλλη (με αλλαγή λίγων γραμμών κώδικα)

Ο κύκλος ζωής του Servlet

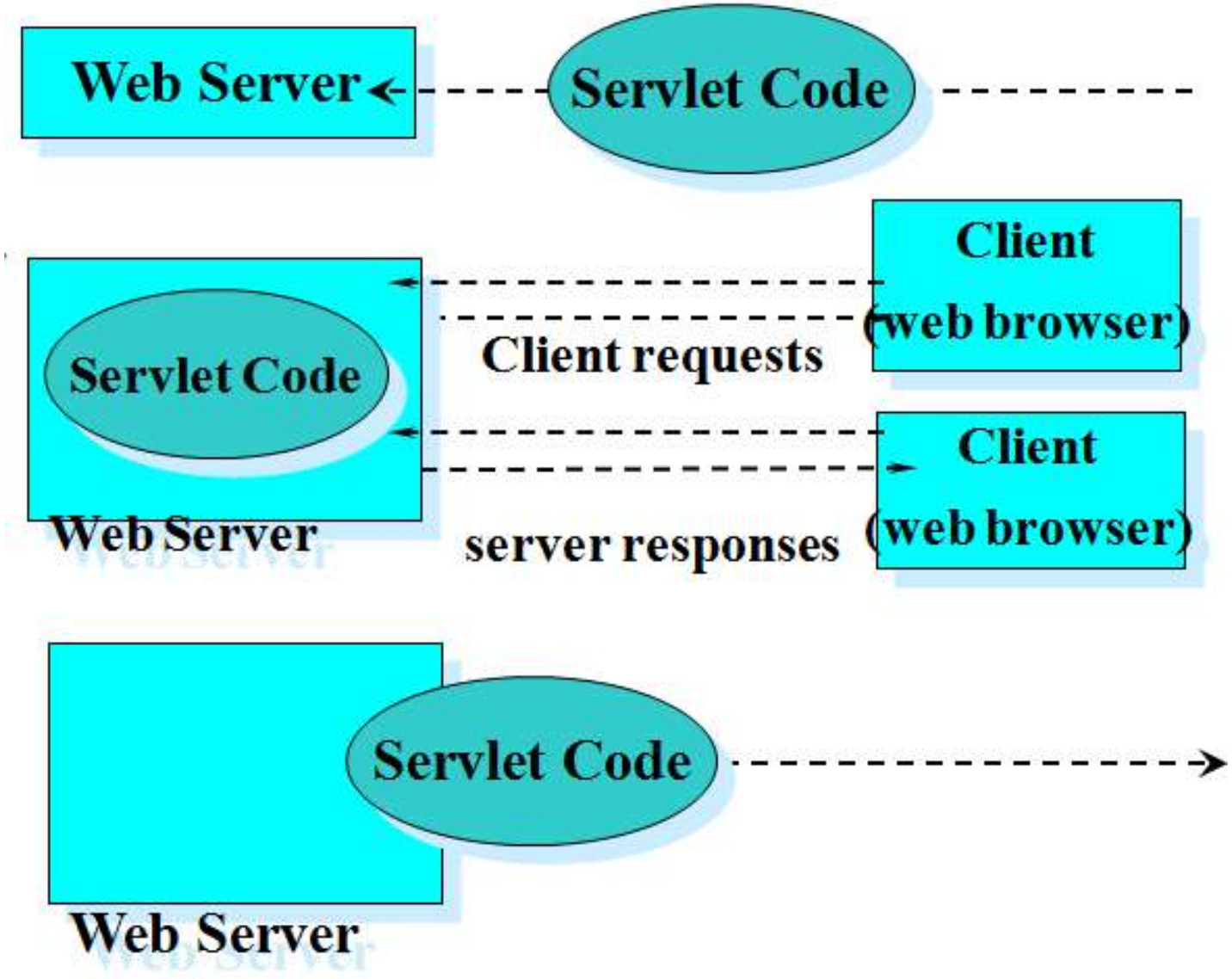
- Κάθε servlet έχει τον ίδιο κύκλο ζωής:
  - Ο server το κάνει load και το αρχικοποιεί: «τρέχει» η μέθοδος init()
  - Το servlet δέχεται μηδέν ή και περισσότερα client requests: «τρέχουν» οι μέθοδοι service() ή doGet()/doPost()
    - Κάθε κλήση (αίτηση χρήστη) δημιουργεί ένα νήμα (thread)
    - Αν για διάφορους λόγους θέλουμε να εμποδίσουμε την πολυνηματική πρόσβαση υλοποιούμε την διασύνδεση SingleThreadModel (*δεν είναι καλή πρακτική όμως*)

    public class YourServlet extends H1ttpServlet implements SingleThreadModel
  - Ο server το κάνει remove (ορισμένοι servers εκτελούν αυτό το βήμα μόνο όταν κάνουν shut down): «τρέχει» η μέθοδος destroy()

Ο κύκλος ζωής του Servlet

- Όταν δημιουργείται για πρώτη φορά μια servlet καλείται η μέθοδός της init η οποία καλείται μια φορά στη ζωή της μικρουπηρεσίας.
- Μετά από αυτό κάθε αίτηση χρήστη δημιουργεί ένα νήμα το οποίο καλεί τη μέθοδο service του στιγμιότυπου που έχει δημιουργηθεί προηγουμένως.
  - Ελέγχει τον τύπο της αίτησης HTTP (GET, POST, PUT, DELETE)
- Κατόπιν η μέθοδος service καλεί τις doGet, doPost, ή κάποια άλλη μέθοδο doXxx ανάλογα με τον τύπο της αίτησης HTTP που έχει ληφθεί.
- Τέλος, αν ο διακομιστής αποφασίσει να απομακρύνει από την μνήμη μια servlet καλεί τη μέθοδο destroy.

Ο κύκλος ζωής του Servlet



Αλληλοεπίδραση του servlet με τον πελάτη

- Όταν ένα servlet δέχεται ένα κάλεσμα από τον πελάτη (client), λαμβάνει δύο αντικείμενα (objects):
  - Ένα ServletRequest, που εξασφαλίζει την επικοινωνία από τον πελάτη προς τον server.
  - Ένα ServletResponse, που εξασφαλίζει την επικοινωνία από το servlet πίσω στον πελάτη.
  - (Τα ServletRequest και ServletResponse είναι interfaces ορισμένα στο javax.servlet package)

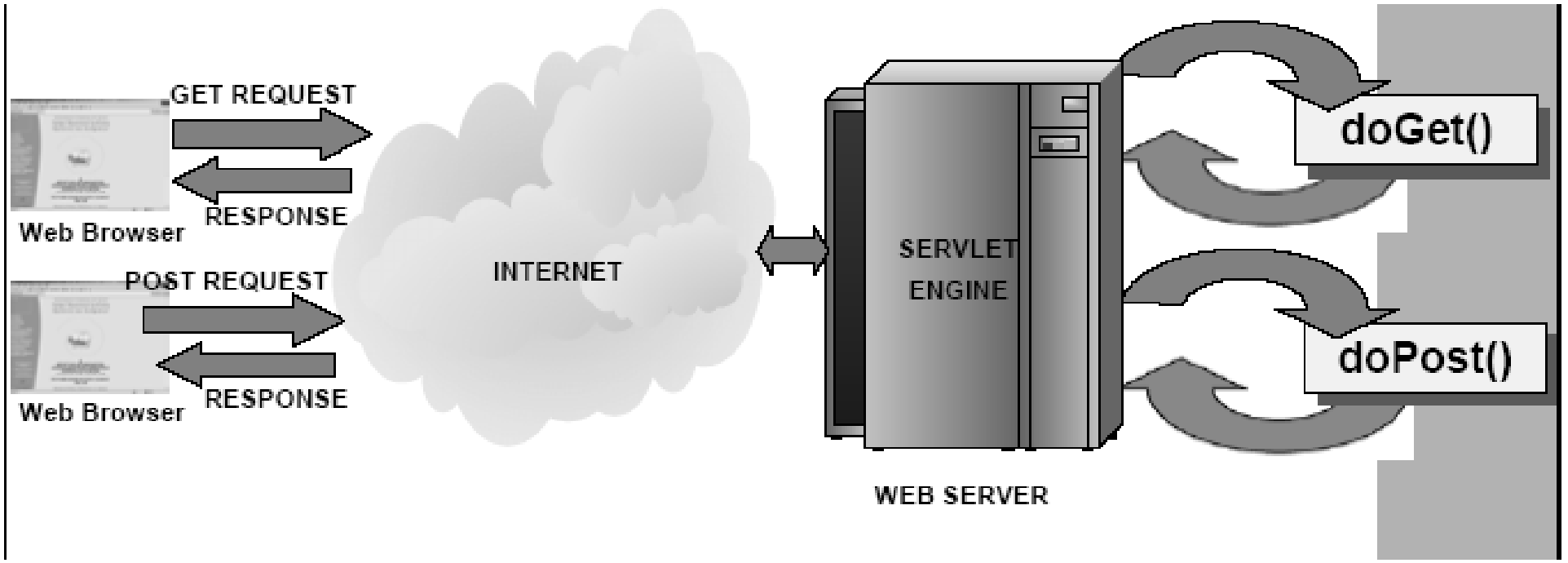
Αλληλοεπίδραση του servlet με τον πελάτη

- Βασικές μέθοδοι:
  - init (καλείται από τον servlet container στην αρχικοποίηση του servlet)
  - doGet (καλείται με HTTP Get request)
  - doPost (καλείται με HTTP Post request)
  - doPut (καλείται με HTTP Put request)
  - service (καλείται από τον servlet container σε κάθε HTTP request)
  - destroy (καλείται από τον servlet container όταν ο servlet πρόκειται να ‘σβηστεί’ από τη μνήμη)

Το Servlet API

- Όλα τα servlets υλοποιούν τη διασύνδεση (interface) Servlet μέσω μιας εκ των δύο βασικών κλάσεων:
  - GenericServlet (javax.servlet package) - γενικά servlets ανεξάρτητα πρωτοκόλλου
  - HttpServlet (javax.servlet.http package) - http servlets

To Servlet API



Hello World Servlet

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet  {

protected void processRequest(HttpServletRequest request, HttpServletResponse
     response) throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      try {
         out.println("Hello World");
      } finally {
         out.close();
      }
   }
}
```

Ένα απλό servlet

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

public class Servlet1 extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/html; charset=windows-1253";
  public void init(ServletConfig config) throws ServletException
  {
     super.init(config);
  }
```

Ένα απλό servlet

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException
{
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<head><title>Servlet1</title></head>");
  out.println("<body>");
  out.println("<p>The servlet has received a GET. This is the reply.</p>");
  out.println("</body></html>");
  out.close();
}
} // class Servlet1
```

Αυτό είναι το output stream αντικείμενο μέσω του οποίου θα σταλεί στον client ο δυναμικός HTML κώδικας

Αν και η HTML είναι το πιο συνηθισμένο είδος εγγράφου,
τα Servlets μπορούν μα δημιουργήσουν και άλλους τύπους εγγράφων
Π.χ. application/vnd.ms-excel, image/jpeg, κτλ.

Προϋποθέσεις για να «τρέξουμε» ένα servlet

- Κάνουμε compilation του java file: SimpleServlet.java -> SimpleServlet.class (πρέπει να είναι εγκατεστημένο το Java Servlet API που περιέχει τις κλάσεις javax.servlet και javax.servlet.http)
- Απαιτείται ένας Application Server (ή Servlet Engine ή Servlet Container), π.χ. οι Oracle OC4J, Tomcat, JavaServer, ...
- Μπορούμε να «τρέξουμε» το servlet απ' ευθείας από κάποιο περιβάλλον προγραμματισμού σε Java (π.χ. το NetBeans έχει ενσωματώσει το Servlet API και τον Application Server Tomcat)
- Να προτιμάτε τη χρήση packages όταν γράφετε servlets σε περιβάλλον παραγωγής.

Πως «τρέχουμε» ένα servlet

- Αν ο servlet ανήκει σε μια εφαρμογή (π.χ. Netbeans project) που ονομάζεται “ServletExamples” και αποθηκεύσαμε το class αρχείο του servlet (αυτό που προκύπτει από το compilation) σε ένα directory, π.χ. το: ServletsExamples\build\web\WEB-INF\classes\
- Κλήση του servlet:

  http://localhost:8080/ServletExamples/TestServlet

Το όνομα (ή IP address) του Η/Υ στον οποίο τρέχει ο servlet

Η πόρτα (port) όπου «ακούει» ο application server (8080: default port του Tomcat)

- Γενικά:

  http://<ServerName>:8080/<ProjectName>/<ServletName>

Εισαγωγή στο NetBeans

- Ένα IDE ανεπτυγμένο από την Sun
- Χρησιμότητα των IDEs (Integrated Development Environment): Ολοκληρωμένα περιβάλλοντα για ανάπτυξη εφαρμογών
  - Εύκολη και γρήγορη συγγραφή κώδικα
  - Ενσωματωμένος compiler
  - Τα IDEs για Java ενσωματώνουν και interpreter, applet viewer
  - Εύκολος εντοπισμός και διόρθωση λαθών
  - Ενσωματωμένος debugger
  - Ενσωματωμένο documentation, help
- Άλλα IDEs για ανάπτυξη εφαρμογών Java (και Servlets): Eclipse, JBuilder (Borland), Java Sun One (Sun)…
- Η έννοια του project

Απλές βοηθητικές κλάσεις δόμησης HMTL

- Συχνά επειδή ο κώδικας HTML παράγεται κάπως άβολα με την println()….
  - Πολλοί μπαίνουν στον πειρασμό να παραλείπουν τμήματα της δομής HTML
    - Π.χ. DOCTYPE
  - Αυτό δεν είναι σωστό καθώς έτσι ακυρώνουμε τα προγράμματα validation
- Προτεινόμενη λύση:
  - Τα τμήματα που συνήθως δεν αλλάζουν (HEAD, DOCTYPE) τα ενσωματώνουμε σε ένα βοηθητικό αρχείο

Παράδειγμα

```
public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";
  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }
```

```
out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1>" + title + "</H1>\n" +
            "</BODY></HTML>");
```

Περνώντας παραμέτρους στο servlet: η HTML φόρμα

```
<html>
<head>
<title> Servlet Example - Passing Parameters</title>
</head>
<body>
<form method="GET" action="Servlet2">
<p>Give your name: <input type="text" name="username" «size="20">
    <input type="submit" value="Try it"></p>
</form>
</body>
</html>
```

Περνώντας παραμέτρους στο servlet: ο κώδικας του servlet

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

public class Servlet2 extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/html; charset=windows-1253";
  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
  }
```

Περνώντας παραμέτρους στο servlet: ο κώδικας του servlet

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException
 {
   response.setContentType(CONTENT_TYPE);
   PrintWriter out = response.getWriter();
   //Store the parameter value passed by the form
   String user = request.getParameter("username");
   // then write the data of the response
   out.println("<html>");
   out.println("<head><title>Servlet talking to an HTML
     form!...</title></head>");
   out.println("<body>");
   out.println("<h2>Hello " + user + "</h2>");
   out.println("<h5>The time is: " + new java.util.Date() + "</h5>");
   out.println("</body></html>");
   out.close();
 }
} // class Servlet2
```

Πως ένα servlet μπορεί να απαντάει σε Get και Post requests με τον ίδιο ακριβώς τρόπο;

- Έχουμε έναν Servlet στον οποίο κάποια HTML φόρμα μπορεί να στείλει είτε GET, είτε POST request.
- Θέλουμε η απόκριση του Servlet να είναι πανομοιότυπη ανεξαρτήτως του τύπου του request, χωρίς όμως να αντιγράψουμε τον κώδικα της doGet() στην doPost()
- Λύση: Απλά, η doPost() καλεί την doGet():

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}
```

Τυπικό HTTP Request

Έστω μια αίτηση του χρήστη στο www.somebooks.com/servlet/Search

```
GET /search-servlet?keywords=servlets+jsp HTTP/1.1
Accept: image/gif, image/jpg, */*
Accept-Encoding: gzip
Connection: Keep-Alive
Cookie: userID=id456578
Host: www.somebookstore.com
Referer: http://www.somebookstore.com/findbooks.html
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

Ανάγνωση των Request Headers (Μέθοδοι του HttpServletRequest)

- **General**
  - getHeader (header name is not case sensitive)
  - getHeaders
  - getHeaderNames
- **Specialized**
  - getCookies
  - getAuthType and getRemoteUser
  - getContentLength
  - getContentType
  - getDateHeader
  - getIntHeader
- **Related info**
  - getMethod, getRequestURI , getQueryString, getProtocol

Ανάγνωση επικεφαλίδων

```
Enumeration headerNames = request.getHeaderNames();
while(headerNames.hasMoreElements())
{
    String headerName = (String)headerNames.nextElement();
    String headerValue = request.getHeader(headerName));
    …
}
```

Δημιουργία/Ανάγνωση Cookies

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
    ...
    Cookie[] cookies = request.getCookies();
    for(int i=0; i<cookies.length; i++) {
        String cookieName = cookies[i].getName();
        String cookieValue = cookies[i].getValue();
        ...
    }
    ...
    Cookie userCookie = new Cookie("user-id", “131977");
    ...
    response.addCookie(userCookie);
    ...
}
```

Συμβουλή: πριν στείλετε το cookie στον πελάτη να καλείται τη συνάρτηση setMaxAge (userCookie.setMaxAge(60*60*24*7) //μια εβδομάδα

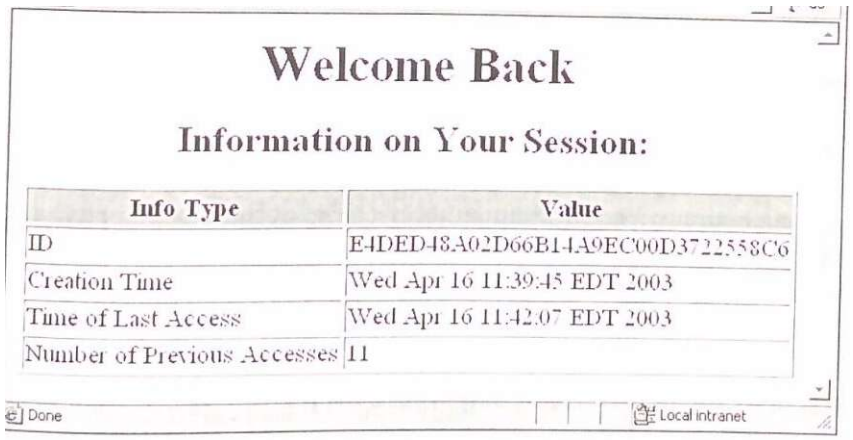
Χειρισμός Sessions

```
// Ανάκτηση του session στο οποίο ανήκει το request (και συνεπώς
// ο πελάτης). Με την παράμετρο true γίνεται αυτόματη δημιουργία νέου
// session σε περίπτωση που δεν υπήρχε.
HttpSession session = request.getSession(true);
…
// Ανάκτηση της αποθηκευμένης κατάστασης ενός αντικειμένου βάσει
// του session του request. Η αναζήτηση γίνεται μέσω του bind name.
SomeClass previousValue = (SomeClass)session.getAttribute(“someIdentifier");
if (previousValue == null)
previousValue = new SomeClass(…);
…
// Προσθήκη αντικειμένου στο παρόν session και ταυτοποίησή του μέσω του
// bind name.
session.setAtrribute("previousValue", previousValue);
…
```

Παράδειγμα Session

```
/** Μικροϋπηρεσία που χρησιμοποιεί παρακολούθηση συνεδριών για τη
 *  διατήρηση του αριθμού επισκέψεων ανά πελάτη. Δείχνει επίσης και
 *  άλλες πληροφορίες σχετικά με τη συνεδρία.
*/

public class ShowSession extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession();
    String heading;
    Integer accessCount =
      (Integer)session.getAttribute("accessCount");
    if (accessCount == null) {
      accessCount = new Integer(0);
      heading = "Welcome, Newcomer";
    } else {
      heading = "Welcome Back";
      accessCount = new Integer(accessCount.intValue() + 1);
    }
```




```
    // Ο τύπος Integer είναι μια αμετάβλητη δομή δεδομένων.
    // Έτσι δεν μπορείτε να αλλάξετε επιτόπου τα παλιά δεδομένα.
    // Αντίθετα, πρέπει να εκχωρήσετε ένα νέο αντικείμενο και να
    // ξαναχρησιμοποιήσετε τη μέθοδο setAttribute.
    session.setAttribute("accessCount", accessCount);
    PrintWriter out = response.getWriter();
    String title = "Session Tracking Example";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<CENTER>\n" +
                "<H1>" + heading + "</H1>\n" +
                "<H2>Information on Your Session:</H2>\n" +
                "<TABLE BORDER=1>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "  <TH>Info Type<TH>Value\n" +
                "<TR>\n" +
                "  <TD>ID\n" +
                "  <TD>" + session.getId() + "\n" +
                "<TR>\n" +
                "  <TD>Creation Time\n" +
                "  <TD>" +
                new Date(session.getCreationTime()) + "\n" +
                "<TR>\n" +
                "  <TD>Time of Last Access\n" +
                "  <TD>" +
                new Date(session.getLastAccessedTime()) + "\n" +
                "<TR>\n" +
                "  <TD>Number of Previous Accesses\n" +
                "  <TD>" + accessCount + "\n" +
                "</TABLE>\n" +
                "</CENTER></BODY></HTML>");
  }
}
```

Αποστολή συμπιεσμένων Ιστοσελίδων

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

// Αλλαγή ορισμού του "out" ανάλογα με το αν
// υποστηρίζεται ή όχι συμπίεση gzip.
PrintWriter out;
if (GzipUtilities.isGzipSupported(request) &&
    !GzipUtilities.isGzipDisabled(request)) {
  out = GzipUtilities.getGzipWriter(response);
  response.setHeader("Content-Encoding", "gzip");
} else {
  out = response.getWriter();
}
```

```
public class GzipUtilities {

  /** Ο πελάτης υποστηρίζει συμπίεση gzip; */

  public static boolean isGzipSupported
      (HttpServletRequest request) {
    String encodings = request.getHeader("Accept-Encoding");
    return((encodings != null) &&
           (encodings.indexOf("gzip") != -1));
  }
```

Επικοινωνία μεταξύ Servlets

- Υπάρχουν πολλοί τρόποι με τους οποίους ένα servlet μπορεί να επικοινωνήσει με ένα άλλο servlet:

1. Servlet Chaining (Κάνοντας διαδοχικά Post, Get)
   - Μπορούμε να αλυσιδώσουμε μια σειρά από servlets
   - Θα μπορούσε π.χ., να είναι μια διαδοχική καταχώρηση δεδομένων.
2. Servlet Interface μέσω μιας ΒΔ (π.χ. για ένα Chat Room)
   - Μπορούμε να καταχωρούμε στοιχεία από ένα servlet σε μια ΒΔ όπου, μπορεί να τα ανακτήσει ένα άλλο servlet ταυτόχρονα και να δίνεται η εικόνα ότι επικοινωνούν ταυτόχρονα.
3. Socket, RMI

ODBC (Open DataBase Connectivity)

- Η διεπαφή ODBC της Microsoft επιτρέπει σε εφαρμογές πρόσβαση σε συστήματα ΒΔ μέσω SQL
- Χρησιμοποιώντας ODBC ένας application developer μπορεί να αναπτύξει, μεταγλωτίσσει και προωθήσει μια εφαρμογή ανεξάρτητη του DBMS
- Διαφορετικά, η εφαρμογή δεν έχει μεταφερσιμότητα (non-portable) $\Rightarrow$ δύσκολη συντήρηση (δεν υποστηρίζονται άλλα DBMS ή άλλες εκδόσεις του ίδιου DBMS)
- Συστατικά μιας ODBC αρχιτεκτονικής:
  - Application: καλεί ODBC συναρτήσεις
  - Driver Manager: «φορτώνει» drivers για την εφαρμογή
  - Driver: επεξεργάζεται και εκτελεί τις κλήσεις ODBC συναρτήσεων, στέλνει SQL requests και επιστρέφει τα αποτελέσματα στην εφαρμογή

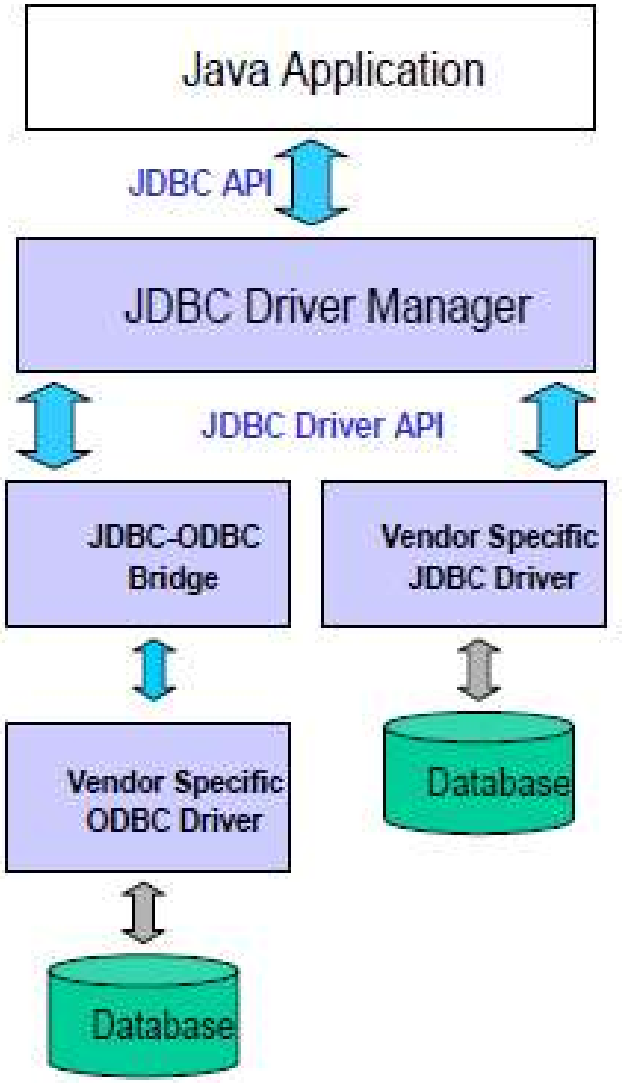
JDBC (Java DataBase Connectivity)

- Το JDBC API της Sun Microsystems παρέχει ένα τυποποιημένο τρόπο πρόσβασης σε DBMS μέσω της γλώσσας Java. Με το JDBC, μια εφαρμογή έχει ομοιόμορφη πρόσβαση (με SQL ερωτήματα) στα δεδομένα ανεξαρτήτως του DBMS και τρέχει πανομοιότυπα σε οποιαδήποτε πλατφόρμα υποστηρίζει
- Το JDBC API ορίζει ένα σύνολο διεπαφών Java που ενσωματώνουν την κύρια λειτουργικότητα ΒΔ (εκτέλεση queries, επεξεργασία αποτελεσμάτων, ...)

JDBC (Java DataBase Connectivity)

- Το JDBC API υλοποιείται μέσω του JDBC driver: ένα σύνολο κλάσεων που αποτελούν διεπαφές για επεξεργασία JDBC κλήσεων και επιστροφή αποτελεσμάτων στην εφαρμογή Java
- Γιατί χρειαζόμαστε το JDBC και δεν αρκούμαστε στο ODBC που είναι ένα API τυποποίησης της πρόσβασης σε ΒΔ;
  - Το ODBC δεν είναι κατάλληλο για άμεση χρήση από εφαρμογές Java γιατί είναι μια διεπαφή γραμμένη σε C
  - Το JDBC προσφέρει μια λύση για μια φυσική διεπαφή Java, δηλαδή μια «αμιγώς Java» στο application development

JDBC Drivers



Τύποι JDBC drivers

- 4 είδη JDBC drivers σε χρήση:
  - Type 1: JDBC-ODBC bridge
  - Type 2: «μερικός» Java driver
  - Type 3: «αμιγής» Java driver προς ενδιάμεσο λογισμικό (middleware) ΒΔ
  - Type 4: «αμιγής» Java driver για άμεση πρόσβαση σε ΒΔ
- Οι «αμιγείς» λύσεις προσφέρουν ανώτερη απόδοση
- Το JDK περιέχει μόνο έναν JDBC driver, το jdbc-odbc bridge
- Για ΒΔ που δεν υποστηρίζονται από το ODBC, χρειαζόμαστε έναν JDBC driver για τη συγκεκριμένη ΒΔ (συνήθως αυτοί drivers πωλούνται)

Διασύνδεση Servlets με ΒΔ

- Τα Servlets όπως όλα τα αλλά προγράμματα Java μπορούν να συνδεθούν με ΒΔ με την χρήση JDBC driver
- Το JDBC είναι database-independent. Π.χ. με αλλαγή 2 γραμμών κώδικα μπορούμε να αλλάξουμε τη βάση μας από Microsoft Access σε MySQL, χωρίς αλλαγή του υπόλοιπου κώδικα
- Κύριο πλεονέκτημα είναι ότι τα Servlets μπορούν να διατηρούν Open Database Connections, με αποτέλεσμα να μπορούν πολλά requests να εξυπηρετηθούν από ένα μόνο κάλεσμα, σε αντίθεση με τα CGI scripts
- Που βρίσκω τον κατάλληλο driver;
  - Η επιλογή εξαρτάται από την πλατφόρμα (λειτουργικό σύστημα) όπου τρέχει ο Servlet και από το RDBMS (ΒΔ) με την οποία θα επικοινωνήσει.

Βασικά βήματα για τη χρήση JDBC

1. Φόρτωση driver
   - Δεν απαιτείται στη Java 6
2. Ορισμός URL σύνδεσης
3. Εγκαθίδρυση της σύνδεσης
4. Δημιουργία ενός Statement αντικειμένου
5. Εκτέλεση query
6. Επεξεργασία αποτελέσματος
7. Κλείσιμο σύνδεσης

Βήμα 1: Φόρτωση driver

- Καθορίζουμε το όνομα κλάσης της ΒΔ στη μέθοδο Class.forName.
- Με αυτόν τον τρόπο δημιουργούμε αυτόματα ένα πρόγραμμα οδήγησης που καταγράφεται στον διαχειριστή προγράμματων οδήγησης της JDBC.

Παράδειγμα:

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    Class.forName("oracle.jdbc.driver.OracleDriver");
} catch (ClassNotFoundException cnfe) {
System.out.println("Error loading driver: " cnfe);
}
```

Βήμα 2: Ορισμός URL σύνδεσης

- Στην JDBC η διεύθυνση URL της σύνδεσης καθορίζει τον υπολογιστή υπηρεσίας του διακομιστή, τη θύρα και το όνομα της ΒΔ με την οποία δημιουργείται η σύνδεση.

Παράδειγμα:

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String mySqlUrl = "jdbc:mysql//" + host + ":" + port +
"/" + dbName;

String embeddedDerbyUrl = "jdbc:derby" + dbName;
```

Βήμα 3: Εγκαθίδρυση σύνδεσης

- Δημιουργείται μια δικτυακή σύνδεση με την ΒΔ
- Μπορούμε στην συνέχεια να εκτελούμε ερωτήματα

Παράδειγμα:

```
String username = "jay_debesee";
String password = "secret";
Connection connection = DriverManager.getConnection(mySqlUrl,
    username, password);
```

Βήμα 4: Δημιουργία Statement αντικειμένου

- Ιδέα
  - Χρησιμοποιείται ένα αντικείμενο Statement για την αποστολή εντολών/ερωτημάτων στη ΒΔ
- Statement types
  - Statement, PreparedStatement, CallableStatement

Παράδειγμα

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

Βήμα 5: Εκτέλεση query

- Ιδέα
  - statement.executeQuery("SELECT … FROM …");
  - Επιστρέφεται ένα ResultSet
  - statement.executeUpdate("UPDATE …");
  - statement.executeUpdate("INSERT …");
  - statement.executeUpdate("DELETE…");
  - statement.execute("CREATE TABLE…");
  - statement.execute("DROP TABLE …");
- Παράδειγμα

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

Βήμα 6: Επεξεργασία αποτελέσματος

Σημαντικές ResultSet μέθοδοι:

- resultSet.next()
  - Πηγαίνει στην επόμενη γραμμή. Επιστρέφει false εάν δεν υπάρχει άλλη γραμμή.
- resultSet.getString("columnName")
  - Επιστρέφει την τιμή της στήλης με το δοθέν όνομα ως String στην τρέχουσα γραμμή. Επίσης διατίθενται: getInt, getDouble, getBlob, κ.α.
- resultSet.getString(columnIndex)
  - Επιστρέφει την τιμή της στήλης με το δοθέν δείκτη. Ο πρώτος δείκτης έχει τιμή 1 (όπως στην SQL) και όχι 0 (όπως στους πίνακες της Java)
- resultSet.beforeFirst()
  - Μετακινεί τον cursor πριν από την πρώτη γραμμή, όπως ήταν αρχικά.
- resultSet.absolute(rowNum)
  - Μετακινεί τον cursor στην δοθείσα γραμμή (ξεκινά από την γραμμή 1)

Βήμα 6: Επεξεργασία αποτελέσματος

Ερώτημα που έχει εκτελεστεί:

"SELECT first, last, address FROM…"

Επεξεργασία με χρήση ονομάτων των στηλών

```
while(resultSet.next()) {
        System.out.printf("First name: %s, last name: %s, address: %s%n",
                          resultSet.getString("first"),
                          resultSet.getString("last"),
                          resultSet.getString("address"));
}
```

Επεξεργασία με χρήση δεικτών των στηλών

```
while(resultSet.next()) {
        System.out.printf("First name: %s, last name: %s, address: %s%n",
                          resultSet.getString(1),
                          resultSet.getString(2),
                          resultSet.getString(3));
}
```

Βήμα 7: Κλείσιμο σύνδεσης

- Θα πρέπει να αναβάλουμε το κλείσιμο της σύνδεσης στην περίπτωση που αναμένουμε ότι μπορεί να εκτελεστούν πρόσθετες λειτουργίες στην ΒΔ, επειδή η επιβάρυνση από το άνοιγμα μιας σύνδεσης είναι συνήθως μεγάλη.

Παράδειγμα:

```
connection.close();
```

Διασύνδεση Servlets με ΒΔ: Μεθοδολογία

```
import java.sql.*;
Class.forName("org.gjt.mm.mysql.Driver");
String url = "jdbc:mysql://host:port/db";
    π.χ. String url = "jdbc:mysql://localhost:3308/books";
con = DriverManager.getConnection(url, "root", "");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT …");
while(rs.next()) { … }
```

Γενική σύνταξη ενός connection URL:<url> ::= jdbc:easysoft:[<server spec>]:[<database>]{:<attribute>=<value>}*<server spec> ::= //[<host name>][:<port>]/<database> ::= <dsn> | DSN=<dsn> | FILEDSN=<filedsn><DSNlessconnection string >

Εξαγωγή δεδομένων από ΒΔ

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException  {
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  out.print("<html><head>");
  out.print("</head><body>");
  out.print("<code><pre>");
  out.print("<font color=green>ID\t Name\t\t Title\n</font>");
  // debugging info
  long time1 = System.currentTimeMillis();
  // connecting to database
  Connection con = null;
  Statement stmt = null;
  ResultSet rs = null;
  try  {
   // Load the JDBC-ODBC Bridge driver
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   // Get a connection to the database
   con = DriverManager.getConnection("jdbc:odbc:FPNWIND", "", "");
   stmt = con.createStatement();
   rs = stmt.executeQuery("SELECT customerId, contactName, contactTitle FROM customers");
   // displaying records
```

Εξαγωγή δεδομένων από ΒΔ

```
    while(rs.next())  {
      out.print(rs.getObject(1).toString() + "\t");
      out.print(rs.getObject(2).toString() + "\t");
      out.print(rs.getObject(3).toString() + "\n");
     }
   }
   catch (SQLException e)  {
    throw new ServletException("Servlet could not display records: " + e.toString(), e);
   }
   catch (ClassNotFoundException e)   {
    throw new ServletException("JDBC Driver not found.", e);
   }
   // debugging info
   long time2 = System.currentTimeMillis();
   out.print("</pre></code>");
   out.print("<p>Search took : ");
   out.print( (time2 - time1) );
   out.print(" ms.</p>");
   out.print("<p\"><a href=\"");
   out.print(request.getRequestURI());
   out.print("\">Back</a></p>");
   out.print("</body></html>");
   out.close();
 }
```

Εισαγωγή δεδομένων σε ΒΔ

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException   {
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  // Διάβασμα παραμέτρων
  String id = request.getParameter("id").trim();
  String name = request.getParameter("name").trim();
  String price = request.getParameter("price").trim();
  Connection con = null;
  Statement stmt = null;
  ResultSet rs = null;
  try {
   // «Φόρτωμα» JDBC driver και σύνδεση στη ΒΔ
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   con = DriverManager.getConnection("jdbc:odbc:FPNWIND", "", "");
   String sql = "INSERT INTO Products(ProductID,ProductName, UnitPrice) VALUES
   (?,?,?)";
   PreparedStatement ps = con.prepareStatement(sql);
   stmt = con.createStatement();
   // Εισαγωγή εγγραφής
```

Εισαγωγή δεδομένων σε ΒΔ

```
    ps.setString(1, id);
    ps.setString(2, name);
    ps.setString(3, price);
    ps.executeUpdate();
   }
   catch (SQLException e)   {
    throw new ServletException("Servlet could not insert records: " + e.toString(), e);
   }
   catch (ClassNotFoundException e)  {
    throw new ServletException("JDBC Driver not found.", e);
   }
   out.print("<html><head>");
   out.print("</head><body>");
   out.print("<p> Data added </p>");
   out.print("</body></html");
   out.close();
 }
```

Ο κύκλος ζωής του Servlet για σύνδεση με ΒΔ

- Η φόρτωση του driver και η σύνδεση γίνονται στην init(). Επειδή το πιο δαπανηρό από άποψη χρόνου είναι η δημιουργία της σύνδεσης, η δημιουργία μιας σύνδεσης στην μέθοδο init και η χρησιμοποίησή της για την εξυπηρέτηση όλων των αιτήσεων για το servlet προτιμάται, από το να δημιουργείται μία νέα σύνδεση κάθε φορά που εκτελείται η doGet.
- Στην doGet ή doPost εμφανίζουμε το μήνυμα της επιτυχούς σύνδεσης με την ΒΔ, εφόσον έχει γίνει η σύνδεση, ή ανεπιτυχούς εφόσον δεν έχει γίνει η σύνδεση (το αντικείμενο Connection con είναι null). Επίσης εκτελούμε όποια ερωτήματα επιβάλει η λογική της εφαρμογής.
- Τέλος, η σύνδεση κλείνει στην μέθοδο destroy με την μέθοδο close() του αντικειμένου con.