

# ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

---

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

## Ανάλυση των Java EE Specifications και Παρουσίαση των Θεωρητικών Εννοιών με Παραδείγματα Κώδικα

**Analysis and Presentation of Java EE Specifications**

**Theoretically and through Code Examples**

Εκπονήθηκε από: Όλεγκ Σοτνίκοβ

Πατρώνυμο: Ιγκόρ

ΑΜ: π09127

Ακαδημαϊκό Έτος: 2016-2017

Καθηγητής: Χρήστος Δουληγέρης



# Περιεχόμενα

ΕΙΣΑΓΩΓΗ .....	7
Γενικοί Στόχοι .....	7
Σύντομη Περιγραφή .....	7
Java EE Runtime Environments (Java Application Servers & Java Servlet Containers) .....	7
Web και Enterprise Java Applications .....	8
Γενικό Πλαίσιο .....	9
Προτιμώμενη Υλοποίηση .....	9
Θεωρητική Εισαγωγή .....	10
Servlet Container .....	10
Application Server .....	11
Web Server .....	12
Multitier Architecture (n-Tier Architecture) .....	12
Client-tier .....	12
Middle-tier .....	13
Web-Tier .....	13
Business-Tier .....	14
EIS-tier .....	15
Συνοψίζοντας .....	16
Containers and Components .....	17
Server Side Containers .....	17
EJB Container .....	17
Web Container (Servlet/JSP Container) .....	17
Client Side Containers .....	18
Application Client Container .....	18
Applet Container .....	18
Container Services .....	18
Web Services .....	18
Security .....	19
Java Naming and Directory Interface (JNDI) .....	20
Java Database Connectivity (JDBC) .....	21
Java Messaging Service (JMS) .....	24
Transactions .....	28
Dependency Injection .....	33
Enterprise Java Beans (EJB) .....	39

Context & Dependency Injection .....	39
Application Packaging.....	40
Εισαγωγή .....	40
Deployment Descriptors.....	40
Java EE deployment descriptors.....	40
Runtime deployment descriptors.....	40
Java EE Application Modules.....	40
Web Module - Directory Structure.....	41
Εγκατάσταση του Weblogic Application Server 12C.....	43
Βασικές έννοιες του Weblogic Application Server 12C.....	45
Domains.....	45
Administration Server .....	45
Managed Servers.....	45
Σημαντικοί κατάλογοι του Weblogic 12C.....	45
Environmental Variables .....	46
Δημιουργία ενός domain .....	47
Configuration Wizard .....	47
Development και Production Mode ενός Domain.....	49
Autodeploy Folder.....	49
Λουπές Διαφορές .....	50
Advanced Configuration.....	50
Administration Server Configuration .....	51
Node Manager Configuration.....	51
Configuration Summary .....	52
Ξεκίνημα του Administration Server.....	54
Σταμάτημα του Administration Server.....	54
Ρύθμιση των αναγνωριστικών για τα startup και shutdown scripts (boot.properties) .....	55
Διαγραφή ενός Domain.....	56
Στήσιμο Web Project στο Eclipse .....	57
Servlets .....	60
Γενική Περιγραφή .....	60
Θέματα Παραλληλίας.....	60
Κύκλος Ζωής .....	61
Servlet Request (HttpServletRequest).....	62
Request Line - Resource URI.....	62
HTTP Request Parameters.....	63
HTTP Request Headers .....	63

Διαφορές μεταξύ Request Headers και Request Parameters.....	64
Header Testing - ModHeader .....	65
Request Attributes .....	65
Servlet Response (HttpServletResponse).....	66
Status Line .....	66
Response Headers .....	70
Response Body (Message Body).....	70
Response Commit.....	71
Πρακτική Εφαρμογή.....	72
Project Development μέσω του Eclipse .....	72
ExampleServlet.java .....	72
Πρώτη εκτέλεση του simpleServletExample Application.....	78
JSP (Java Servlet Pages) .....	83
Γενική Περιγραφή .....	83
JSP – Deployment Hierarchy .....	83
Παράδειγμα JSP (welcome.jsp) .....	84
Content Type & Encoding (headers).....	84
Import packages and classes .....	84
Declare tag libraries.....	84
Doctype (non-jsp specific) .....	85
JSP Declarations (Methods and Fields).....	85
Content Type & Encoding (meta tags, non-jsp specific).....	85
JSP Include directive .....	85
JSP Expressions .....	86
JSP Scriptlets.....	86
Ανακατεύθυνση των Requests - Forward, Redirection και Include .....	87
Εφαρμογή και Επεξήγηση με Κώδικα .....	88
Forwarding.....	90
Redirect to Internal Resource.....	91
Redirect to External Resource .....	93
Include Action μέσα σε ένα Servlet.....	93
Include Action και Include Directive σε μια JSP .....	96
Filters .....	98
Γενική Περιγραφή .....	98
Θεωρητικό Υπόβαθρο .....	98
Δημιουργία/Ορισμός, Lifecycle και Concurrency .....	99
Δημιουργία/Ορισμός.....	99

Lifecycle & Concurrency .....	100
Παραδείγματα με Κώδικα .....	100
Αντιστοιχία των Requests στα Servlets (Servlet Mapping) .....	111
Sessions .....	113
Θεωρητική Περιγραφή .....	113
Κώδικας Παραδείγματος .....	113
Εκτέλεση Παραδείγματος .....	119
Polling των Αρχείων της Εφαρμογής στην Περίπτωση ενός Expanded Directory .....	120
JDBC .....	121
Παράδειγμα με Κώδικα .....	121
Εκτέλεση το Παραδείγματος .....	124
Driver Registration πριν την Java 1.4 .....	126
EJB .....	128
Session Beans .....	129
Singleton Session Beans .....	129
Stateless Session Beans .....	140
Stateful Session Beans .....	145
Entity Beans .....	149
Δημιουργία του DataSource Αντικειμένου .....	149
Ορισμός του persistence.xml .....	154
Ορισμός του DAO Layer .....	155
Χρήση του EntityManager Αντικειμένου για την Διασύνδεση των Αντικειμένων του DAO Layer με την ΒΔ .....	158
Message Driven Beans (MDB) & JMS .....	160
Λειτουργική Περιγραφή του Παραδείγματος .....	160
Δημιουργία JMS Resources .....	161
Ανάλυση Κώδικα Παραδείγματος .....	173
Context and Dependency Injection – CDI .....	180
Γενική Περιγραφή .....	180
Scopes .....	180
Qualifiers & Alternatives .....	181
Γενικά .....	181
Alternatives .....	181
Qualifiers .....	182
Transactions .....	186
Μελέτη των Transactions στα Πλαίσια της Μεθόδου ComputeFibonacciMDB.onMessage ....	186
Συμπεριφορά CMT κατά την Αποστολή JMS Messages .....	191

Security .....	192
Σύντομη Περιγραφή .....	192
Ορισμός των Περιορισμών Ασφάλειας στον Deployment Descriptor .....	192
Container Managed Authentication στα πλαίσια του Weblogic Server 12C με τα Credentials των Χρηστών να Αποθηκεύονται στην ΒΔ της Εφαρμογής.....	193
Ορισμός των Ρόλων.....	193
Αντιστοίχιση Ρόλων και Ομάδων Χρηστών (ή Μεμονωμένων Χρηστών).....	193
Αντιστοίχιση των Ομάδων με τους Χρήστες .....	194
Ρύθμιση του Security Realm και του Authentication Provider στον Weblogic Server 12C ...	195
Java EE Container Managed Authentication API.....	201
Authentication Scheme .....	202
Container Managed Confidentiality .....	205
Διαχείριση των ρυθμίσεων SSL, SSL Certificates και Keystores .....	205
Εφαρμογή του SSL στους Πόρους ενός Web Application .....	210
Web Service.....	211
JAX-WS Service .....	211
Γενική Περιγραφή.....	211
Παράδειγμα με Κώδικα.....	212
Βιβλιογραφία .....	223

# ΕΙΣΑΓΩΓΗ

## Γενικοί Στόχοι

### Σύντομη Περιγραφή

Σκοπός της παρούσας πτυχιακής εργασίας είναι να αναλύσει την διαδικασία ανάπτυξης Επιχειρησιακών Εφαρμογών (Enterprise Applications) που ακολουθούν το Java EE πρότυπο.

Αυτό πραγματοποιείται μέσω της ανάλυσης του **Java Enterprise Edition Specification (Java EE)** και της εξέτασης των specifications και του documentation πάνω στα **Java EE Runtime Environments (Java Application Servers & Java Servlet Containers)** καθώς και την ανάλυση της διαδικασίας ανάπτυξης Enterprise Εφαρμογών μέσω της παρουσίασης και εκτέλεσης απλών παραδειγμάτων κώδικα που παρουσιάζουν με πρακτικό τρόπο τις βασικότερες θεωρητικές έννοιες που θεωρήθηκε ότι είναι απαραίτητες για την ανάπτυξη των επιχειρησιακών εφαρμογών και την κατανόηση των ιδιαιτεροτήτων της λειτουργίας τους στα πλαίσια ενός περιβάλλοντος εκτέλεσης όπως είναι τα Application Servers.

### Java EE Runtime Environments (Java Application Servers & Java Servlet Containers)

Μία βασική διαφορά μεταξύ των απλών Java εφαρμογών και των Enterprise Java εφαρμογών είναι ότι οι πρώτες απαιτούν μόνο ένα Java Runtime Environment (JRE ή και JDK) για να τρέξουν και είναι σχεδιασμένες να εκτελεστούν από την γραμμή εντολών του λειτουργικού συστήματος του τελικού χρήστη της εφαρμογής (ή με διπλό κλικ στο εκτελέσιμο αρχείο τους). Οι Enterprise Java εφαρμογές όμως εκτελούνται μέσα από ένα **Runtime Environment**.

**Runtime Environments** αποτελούν τα **Application Servers** και **Servlet Containers**. Τα Servlet Containers και Application Servers είναι και οι δύο εκφράσεις της έννοιας του **Server (εξυπηρετητή)** με την διαφορά ότι τα Servlet Containers προσφέρουν ένα υποσύνολο της λειτουργικότητας που προσφέρουν οι Application Servers.

Αυτή η διαφορά μεταξύ των απλών και των επιχειρησιακών Java εφαρμογών έχει ως αποτέλεσμα μία πολύ διαφορετική μορφή των επιχειρησιακών εφαρμογών. Ενώ δηλαδή στις απλές Java εφαρμογές μέσα από τον κώδικα μπορούμε να δούμε μία ροή εκτέλεσης η οποία έχει την αρχή της στην μέθοδο “**main**” στις επιχειρησιακές εφαρμογές δεν υπάρχει κάτι τέτοιο. Αντιθέτως ορίζονται **συστατικά (components)** τα οποία προσφέρουν τις επιθυμητές υπηρεσίες στον τελικό απομακρυσμένο πελάτη μέσω των μεθόδων που ορίζουν. Θα μπορούσαμε απλοϊκά να πούμε ότι την κλήση αυτών των μεθόδων καθώς και την διαχείριση του κύκλου ζωής αυτών των συστατικών την αναλαμβάνει το Runtime Environment ή πιο συγκεκριμένα το κατάλληλο **Container** που φιλοξενεί το συστατικό ανάλογα με τον τύπο τους.

Η κλήση των μεθόδων των συστατικών γίνεται προφανώς σε ανταπόκριση στα **HTTP αιτήματα** τα οποία λαμβάνει ο Server (Runtime Environment) από τον Browser του πελάτη.

## Web και Enterprise Java Applications

Επειδή η παρούσα εργασία στα πλαίσια της ανάλυσης του **Java EE Specification**, θα ασχοληθεί και με τα **Web και Enterprise Applications** καλό είναι εξαρχής να τονιστεί η διαφορά μεταξύ των δύο όρων.

### **Web Application**

Ο όρος **Web Application (Διαδικτυακή Εφαρμογή)** αναφέρεται σε εφαρμογές που τρέχουν στο περιβάλλον ενός Server και εξυπηρετούν τον πελάτη μέσω του διαδικτύου ή του τοπικού δικτύου και του HTTP πρωτοκόλλου. Ένα Web Application ουσιαστικά είναι μια δυναμική ιστοσελίδα.

### **Enterprise Application**

To **Enterprise Application (Επιχειρησιακή Εφαρμογή)** είναι ένας πιο γενικός όρος από το Web Application. Είναι μια εφαρμογή η οποία εκτελείται όπως και μία διαδικτυακή εφαρμογή στα πλαίσια ενός Java EE Runtime Environment (Java Server) αλλά ο όρος τονίζει την **υλοποίηση επιχειρησιακών διαδικασιών** και όχι συγκεκριμένα την παραγωγή δυναμικού διαδικτυακού περιεχομένου (ιστοσελίδων) κάτι το οποίο η επιχειρησιακή εφαρμογή δεν είναι απαραίτητο ότι πραγματοποιεί.

Μια επιχειρησιακή εφαρμογή έχει περισσότερες δυνατότητες και μπορεί να εξυπηρετήσει την **αποδοτική εκτέλεση επιχειρησιακών διαδικασιών (business processes)** μιας εταιρείας. Το χαρακτηριστικό τέτοιων εφαρμογών είναι ότι συνήθως επικοινωνούν με τρίτα συστήματα όπως π.χ. κάποιο CRM σύστημα (Customer Relationship Management), συστήματα πληρωμών και χρεώσεων (Billing), συστήματα διαχείρισης προμηθειών (Logistics) κ.α. τα οποία είναι ανεξάρτητα συστήματα μιας επιχείρησης των οποίων η υλοποίηση, η ανάπτυξη, η διαχείριση και η χρήση είναι ανεξάρτητη του Enterprise Application που αναπτύσσουμε.

Μια επιχειρησιακή διαδικασία μίας εταιρείας ενδέχεται να απαιτεί την συλλογή και την επεξεργασία πληροφοριών από τέτοια συστήματα καθώς και την ενημέρωσή τους για τυχών ενέργειες του πελάτη. Κάτι τέτοιο απαιτεί διάφορες εγγυήσεις για την ορθή και ασφαλή εκτέλεση όλων των βημάτων της επιχειρησιακής διαδικασίας καθώς και την σωστή διαχείριση των σφαλμάτων που μπορεί να παρουσιαστούν (αναφέρεται και ως **QoS Guarantees, Quality of Service**) εξαιτίας της φύσης των διαδικασιών οι οποίες εκτελούνται έξω από τα όρια ενός μοναδικού συστήματος και περιλαμβάνουν πολλαπλούς πόρους και συστήματα που είναι απομακρυσμένα.

Για την διευκόλυνση των προγραμματιστών ενός Enterprise Application στην ανάπτυξη Enterprise Εφαρμογών που παρέχουν τέτοιες εγγυήσεις, το Java EE Specification ορίζει κάποιες **υπηρεσίες (services)** τις οποίες θα πρέπει να παρέχουν τα Java EE Runtime Environments στους προγραμματιστές των εφαρμογών που ο server θα φιλοξενήσει. Αυτές θα αναλυθούν αργότερα.

Μία Enterprise Εφαρμογή συχνά χαρακτηρίζεται άτυπα και ως Web Εφαρμογή όμως **στα πλαίσια της παρούσας εργασίας θα προτιμάται ο όρος Enterprise Application επειδή τονίζει την προαναφερθείσα διαφορά και το πεδίο το οποίο καλύπτει το Java EE Specification.**

## Γενικό Πλαίσιο

**Η εργασία εστιάζει μόνο σε στοιχεία που ορίζονται στο Java EE.** Αυτό αποτελείται από βασικές βιβλιοθήκες οι οποίες είναι πάντα διαθέσιμες ασχέτως με τις οποιεσδήποτε επεκτάσεις (frameworks και βιβλιοθήκες π.χ. όπως το Spring) που αποφασίσουμε να χρησιμοποιήσουμε επιπλέον.

Οι γνώσεις σχετικά με το Java EE μπορούν να αξιοποιηθούν για την **δημιουργία δυναμικών ιστοσελίδων** και γενικώς **επιχειρησιακού λογισμικού σε κατανεμημένα περιβάλλοντα** καλύπτοντας όλες τις **QoS** ανάγκες που πρέπει να ικανοποιεί κάθε σύγχρονη επιχειρησιακή εφαρμογή.

Επίσης πολλά frameworks της Java στοχεύουν στο να βελτιώσουν και να αντικαταστήσουν τις λύσεις που προσφέρονται από το Java EE. Στο μεγαλύτερο μέρος τους, η βάση κώδικα αυτών των frameworks στηρίζεται στο Java EE. Η εργασία αυτή προσφέρει πολλές βασικές γνώσεις πάνω στο Java EE τις οποίες κάποιος θα μπορούσε να εκμεταλλευτεί για να **αναπτύξει το δικό του Web ή και Enterprise Framework**.

**Δεν αναλύονται frameworks ανοιχτού ή κλειστού κώδικα** όπως είναι το Struts2, Spring, ATG κτλ. τα οποία προσφέρουν αντίστοιχες δυνατότητες και λύσεις όπως και το Java EE αλλά και επεκτάσεις πάνω σε αυτό.

Μεγάλη έμφαση δίνεται στην απλότητα και αμεσότητα των παραδειγμάτων για την παρουσίαση και την ερμηνεία των θεωρητικών εννοιών **χωρίς να υλοποιείται κάποιο business logic**<sup>1</sup> με πρακτική χρησιμότητα, ώστε η προσοχή του αναγνώστη να μην αναλώνεται σε επιπλέον πολυπλοκότητα πέραν αυτής των καθαρά τεχνικών θεμάτων.

## Προτιμώμενη Υλοποίηση

Το **Java EE** ορίζει το τι πρέπει να υποστηρίζεται από τους δημιουργούς Περιβαλλόντων Εκτέλεσης (Runtime Environments) των Enterprise Applications και όχι το πως αυτά θα υλοποιηθούν. Με λίγα λόγια είναι απλώς ένα σύνολο από **Specifications, Documentations και Διεπαφών (APIs)**.

**Υπάρχουν πολλές υλοποίησεις του Java EE.** Μερικές από αυτές είναι οι Application Servers Weblogic, JBoss, Glassfish, Apache, Jetty κ.α..

Ένας από τους στόχους αυτής της εργασίας είναι η **πρακτική παρουσίαση των εννοιών του Java EE** μέσω παραδειγμάτων εκτέλεσης κώδικα που απαιτεί την χρήση μίας τέτοιας υλοποίησης. Για αυτόν τον σκοπό **θα αναπτυχθούν τα Web Applications, “DemoWebApp” και “DemoWebAppWs” και θα γίνουν deployed στον Weblogic Server 12.1.3.**

---

<sup>1</sup> Το “business logic” ή αλλιώς “taskflow” είναι η επιχειρησιακή λογική της κάθε εφαρμογής. Π.χ. σε μια τραπεζική ιστοσελίδα τα βήματα τα οποία χρειάζονται για την μεταφορά χρημάτων από έναν λογαριασμό σε έναν άλλον μπορεί να περιλαμβάνουν διάφορους ελέγχους για την εγκυρότητα της συναλλαγής π.χ. έλεγχος επάρκειας μετρητών, έλεγχος για το εάν ο λογαριασμός του αποδέκτη είναι ενεργός κ.τ.λ.. Όλα αυτά είναι ενέργειες οι οποίες καθορίζονται από την φύση των υπηρεσιών που προσφέρει η επιχείρηση και όχι των τεχνολογιών που θα χρησιμοποιηθούν για την υλοποίησή τους.

## Θεωρητική Εισαγωγή

Οι κυριότερες έννοιες του Java EE Specification είναι το **Runtime Environment**, τα **Web και Enterprise Applications**, οι **Containers**, τα **Components** και τα **Services**. Επίσης μια πολύ σημαντική έννοια η οποία είναι γενική όσον αφορά τις επιχειρησιακές εφαρμογές που δεν περιορίζεται μόνο στο Java EE είναι αυτή του **Multi-Tier Architecture**.

Οι έννοιες των **Runtime Environments**, **Web και Enterprise Applications** παρουσιάστηκαν προηγουμένως. Είναι σημαντικό να γίνει παρουσίασή των υπόλοιπων εννοιών για την καλύτερη κατανόηση των μετέπειτα κεφαλαίων. Επίσης η έννοια του Runtime Environment αναλύεται περαιτέρω και παρουσιάζεται η διαφορά μεταξύ των **Application Servers** και **Servlet Containers** καθώς επίσης τονίζεται και ο κίνδυνος της σύγχυσής τους με την έννοια του “**Web Server**” η οποία στα πλαίσια αυτής της εργασίας θα έχει συγκεκριμένη σημασία που δεν συμπίπτει με ένα Java EE Runtime Environment.

Η έννοια του **Multi-Tier Architecture** περιλαμβάνει κατασκευαστικές και αρχιτεκτονικές αρχές όπως το **Modularity**, **Scalability** και **Separation of Concerns**. Είναι η έννοια που περιγράφει καλύτερα τις τάσεις και την φιλοσοφία που διέπει το σύγχρονο Web και Enterprise Development.

Αφού παρουσιαστούν τα προηγούμενα και τα **Containers**, **Components** και **Container Services** το υπόλοιπο της εργασίας προχωράει στις **τεχνικές λεπτομέρειες** του Java EE καθώς και την **εξέταση** των εννοιών που αυτό ορίζει (π.χ. **Servlets**, **Enterprise Java Beans**, **Transactions** κ.τ.λ.), την **εγκατάσταση** και **διαχείριση** του **Weblogic Server 12.1.3** καθώς και την **ανάπτυξη** και **εκτέλεση** παραδειγμάτων με κώδικα. Αυτά τα παραδείγματα θα αντιστοιχούν στις έννοιες που θα εξετασθούν θεωρητικά στα αμέσως επόμενα κεφάλαια.

## Servlet Container

**Servlet Container** μπορεί να χαρακτηριστεί οποιοσδήποτε Server που ικανοποιεί τις προδιαγραφές που ορίζονται στο **Java Servlet Specification** και **JavaServer Pages Specification**. Οι δυνατότητες ενός Servlet Container είναι περιορισμένες καθώς η μόνη ουσιαστική απαίτηση είναι να φιλοξενεί components όπως **Java Servlets** και **JavaServer Pages** που είναι συστατικά στοιχεία των Web και Enterprise Applications και που παρέχουν **δυναμικό περιεχόμενο** σε Web Clients (Internet Browsers). To Servlet Container μπορεί επίσης να παρέχει **στατικό περιεχόμενο** όπως εικόνες και απλές HTML σελίδες.

Δεν υποστηρίζει δηλαδή αναγκαστικά τα πολυάριθμα πρότυπα που ορίζει το Java EE όπως είναι π.χ. τα **Enterprise Java Beans (EJBs)**, **Java API for XML Web Services (JAX-WS)**, **Java Transaction API (JTA)**, **Java Message Service (JMS)** κ.τ.λ..

Ένα παράδειγμα ενός Servlet Container είναι ο **Apache Tomcat**.

Ένας Servlet Container αναγκαστικά περιλαμβάνει κάποια υλοποίηση για **εγκατάσταση** και **δημοσίευση** των επιχειρησιακών εφαρμογών που αναπτύσσει ο προγραμματιστής. **Αυτό όμως δεν τυποποιείται από κάποιο πρότυπο της Java EE και διαφέρει από υλοποίηση σε υλοποίηση**. Το ίδιο ισχύει για οποιαδήποτε επιπλέον διαχειριστικά εργαλεία μπορεί να προσφέρει κάποια συγκεκριμένη υλοποίηση.

Συνήθως τα Servlet Containers παρέχουν **επιπλέον υπηρεσίες** εκτός από αυτές που ορίζονται/απαιτούνται στο Servlet Specification όπως π.χ. SSL, και επίσης έχουν κάποιο πρότυπο

το οποίο ορίζει τον τρόπο που μπορεί να γίνει εγκατάσταση **επεκτάσεων** που τους δίνουν κάποιες επιπλέον δυνατότητες και διευκολύνουσεις. Σε καμία περίπτωση πάντως δεν φτάνουν τις δυνατότητες ενός application server οι οποίοι όμως έχουν και πολύ μεγαλύτερες απαιτήσεις χώρου για την εγκατάστασή τους και πολύ μεγαλύτερη πολυπλοκότητα στην εγκατάσταση, ρύθμιση και χρήση τους.

Πολλές φορές αντί για την ίδια την εφαρμογή αναφερόμαστε στο σύνολο της λειτουργικότητας που προσφέρουν οι Servlet Containers οπότε ο όρος χρησιμοποιείται και όταν αναφερόμαστε στο **υποσύστημα ενός Application Server** που την παρέχει.

## Application Server

Application Server στην Java θα χαρακτηρίζονταν ένας server που **υποστηρίζει πλήρως την Java Enterprise Edition πλατφόρμα (Java EE)** και όχι μόνο τα **Java Servlet Specification** και **JavaServer Pages Specification** που είναι υποσύνολό της.

Αυτό σημαίνει ότι εκτός των Java Servlet και JSP Specifications υποστηρίζει **όλες τις τυποποιημένες ιεραρχικές δομές καταλόγων εκτελέσιμων προς δημοσίευση που ορίζονται από το Java EE πρότυπο (packaging)**. Σε αντιδιαστολή με τον Application Server όσον αφορά το packaging ένας Servlet Container υποστηρίζει μόνο την δομή ενός απλού web archive (war).

**Επίσης υποστηρίζει επιπλέον την διαχείριση των Enterprise Java Beans συστατικών (EJB Components)** και όλων των **υπηρεσιών (Container Services)** καθώς και **την επιπλέον υπηρεσία διαχείρισης συστατικών των Enterprise Java Beans (EJB Container)**.

Από την άλλη ένας Servlet Container υποστηρίζει μόνο τον **Servlet Container - JSP Container** (οι οποίοι σύμφωνα με την ορολογία στα πρότυπα είναι το ίδιο υποσύστημα εσωτερικά του Server) και τα αντίστοιχα συστατικά τα οποία είναι τα Servlets και JSPs.

Οι όροι που αναφέρθηκαν σε αυτό το σημείο θα παρουσιάζονται σταδιακά στην συνέχεια της εργασίας. Σε αυτό το σημείο απλώς θα αναφερθεί γενικά ότι τα **“Containers”** φιλοξενούν **“Components”** που είναι στοιχεία τα οποία αναπτύσσει ο προγραμματιστής του Enterprise Application. Η **διαχείριση του κύκλου ζωής** τους αναθέτεται στον Container στον οποίον αντιστοιχούν.

Τα **“Services”** αναφέρονται σε τυποποιημένες υπηρεσίες και λειτουργικότητες που έχει ανάγκη σχεδόν κάθε διαδικτυακή εφαρμογή π.χ. **Συναλλαγές (Transactions), Ασφάλεια (Security), Υπηρεσίες Ονοματολογίας και Ευρετηρίων (Naming and Directory Services), Web Services, Messaging Services κ.τ.λ..**

## Web Server

Σε αυτό το σημείο είναι πολύ σημαντικό να γίνει διαφοροποίηση μεταξύ των **Servlet Containers / Application Servers** και των **Web Servers**.

**Οι Web Servers είναι συνήθως συστήματα τα οποία βρίσκονται «πριν» τους Application Servers.** Ρόλος τους είναι κυρίως το **Load Balancing** (ο αποδοτικός διαμοιρασμός της διαδικτυακής κίνησης μεταξύ των διεργασιών του Application Server), το **Caching** (για αυτό και αναφέρονται συχνά ως **“Reverse Proxies”**) και διάφορα άλλα features όπως π.χ. το **Single Sign On (SSO)**.

**Παραδείγματα Web Servers** αποτελούν συστήματα όπως το **IIS (Internet Information Services)** της Microsoft και το **ACE** της Apache.

**Στην συνέχεια της εργασίας οι όροι “Application Server” και “Web Server” θα χρησιμοποιούνται λαμβάνοντας υπόψη την διαφορά που περιεγράφηκε σε αυτό το σημείο κάθε φορά** και είναι σημαντικό να μην γίνεται σύγχυση μεταξύ τους.

Η ανάλυση της λειτουργικότητας ενός Web Server ξεφεύγει από τα πλαίσια της παρούσας πτυχιακής εργασίας και δεν θα πραγματοποιηθεί. Παρόλα αυτά ένας Web Server θεωρείται απαραίτητο συμπλήρωμα στους Application Servers ενός επιχειρησιακού web συστήματος.

## Multitier Architecture (n-Tier Architecture)

Αρχικά θα γίνει αναφορά σε ένα αρχιτεκτονικό μοτίβο το οποίο υποστηρίζεται από τα Java EE Application Servers. Αυτό είναι το **Multitier Architecture** (συναντάται και ως **n-Tier Architecture**). Το Multitier Architecture ορίζει n tiers, δηλαδή n (συνήθως 3) δομικά επίπεδα στα οποία χωρίζεται ένα λογισμικό σύστημα. Στην πλειονότητα των περιπτώσεων τα επίπεδα αυτά είναι το **Presentation Tier, Application Tier και Data Tier**. Στο πλαίσιο της παρούσας εργασίας και της Java EE αυτά αντιστοιχούν στα **Client-Tier, Middle-Tier και EIS-Tier**.

Μία εφαρμογή που είναι χωρισμένη στα προηγούμενα επίπεδα είναι πολύ ευέλικτη διότι το κομμάτι της εφαρμογής στο κάθε επίπεδο συνεργάζεται και επικοινωνεί με όλα τα υπόλοιπα αλλά είναι ταυτόχρονα εντελώς ανεξάρτητο από αυτά ως προς την **υλοποίηση, την ρύθμιση, την παραμετροποίηση, την ανάπτυξη, την εκτέλεση και την χρήση του**. Παρακάτω παρουσιάζονται τα τρία επίπεδα **Client-Tier, Middle-Tier και EIS-tier**.

## Client-tier

To **Client-Tier** περιλαμβάνει οποιοδήποτε κομμάτι του web application το οποίο τρέχει στο **μηχάνημα του χρήστη**. Αυτό μπορούν να είναι components σε κάποια java εφαρμογή η οποία τρέχει στο μηχάνημα του χρήστη και επικοινωνεί απομακρυσμένα με το αντίστοιχο server side java web application. Συνήθως βέβαια όσον αφορά τις περισσότερες διαδικτυακές εφαρμογές το Client-Tier περιλαμβάνει τον **Browser** του χρήστη και ίσως και κάποιο plugin για να υποστηρίζονται τα **Java Applets**. Στην περίπτωση των smartphones μπορεί να είναι κάποια Java εφαρμογή για το λειτουργικό σύστημα Android και μπορούν να βρεθούν και άλλα παραδείγματα.

Ο ρόλος αυτού του επιπέδου είναι η **παρουσίαση των αποτελεσμάτων** της επεξεργασίας των αιτήσεων του χρήστη. Η ύπαρξη αυτού του επιπέδου βοηθάει στην ανεξαρτησία μεταξύ του επιπέδου επεξεργασίας των πληροφοριών και του επιπέδου παρουσίασής τους.

Π.χ. μία διαδικτυακή εφαρμογή δυναμικών ιστοσελίδων παρέχει σαν το τελικό αποτέλεσμα της επεξεργασίας των HTTP αιτημάτων, αρχεία HTML, Javascript και διάφορους άλλους πόρους όπως εικόνες και CSS αρχεία αλλά **ο χρήστης είναι κατά κανόνα ελεύθερος να επιλέξει τον περιηγητή που θα χρησιμοποιήσει** για την παρουσίασή τους αφού όλοι υποστηρίζουν τα HTTP και HTML πρωτόκολλα.

Επιπρόσθετα με κατάλληλο σχεδιασμό της εφαρμογής η ανεξαρτησία μεταξύ των επιπέδων θα μπορούσε να ενισχυθεί σε τέτοιο βαθμό ώστε να υποστηρίζονται clients εντελώς διαφορετικού είδους. Π.χ. μία διαδικτυακή εφαρμογή μπορεί να βασίζεται σε σελίδες HTML και **Browser** πρόγραμμα για την παρουσίασή τους ή, και συν τοις άλλοις μπορεί η ίδια server-side εφαρμογή να υποστηρίζει ταυτόχρονα μία **Java εφαρμογή** ως client που εγκαθίσταται στο κινητό του χρήστη ως εναλλακτική λύση για την καλύτερη παρουσίαση των αποτελεσμάτων της επεξεργασίας των αιτήσεων σε ένα smartphone. Σαν ένα τέτοιο παράδειγμα ο αναγνώστης μπορεί να φανταστεί μία android εφαρμογή όπως είναι π.χ. το facebook app, χωρίς βέβαια να υπονοείται ότι η ιστοσελίδα του facebook και το android app του όντως επικοινωνούν με τις ίδιες server side web εφαρμογές, αυτό που υπονοείται εδώ είναι ότι με τον κατάλληλο σχεδιασμό θα μπορούσαν να το κάνουν και ίσως και όντως να το υποστηρίζουν.

Αυτό θα μπορούσε να επιτευχθεί π.χ. εάν τα responses που αντιστοιχούν στην βασική λειτουργικότητα επιστρέφονταν ως **JSON** και οποιαδήποτε μετάβαση από την μία HTML σελίδα σε άλλη (στην περίπτωση που ο client είναι ένας browser) να γινόταν με Javascript redirects. Σε αυτήν την περίπτωση μία συμβατική Java εφαρμογή θα μπορούσε να διαχειριστεί τα JSON responses ώστε να εξυπηρετήσουν την δικιά της λογική παρουσίασης αποτελεσμάτων (π.χ. αξιοποιώντας το **Java SWING/AWT** πακέτο ή το **Android SDK**).

Όλα αυτά βέβαια **επιτυγχάνονται με την παράλληλη σχεδίαση του Middle-Tier (και συγκεκριμένα το Web-Tier) που να ακολουθεί τους ίδιους στόχους, δηλαδή το server-side κομμάτι της εφαρμογής που αναφέρεται στην επόμενη παράγραφο. Δεν αρκεί δηλαδή μόνο η κατάλληλη σχεδίαση του client-side επιπέδου.**

## Middle-tier

Το **Middle-Tier** είναι το επίπεδο του web application που στεγάζεται στον server. Χωρίζεται περαιτέρω στο **Web-Tier** και το **Business-Tier**. Περιέχει την κύρια λειτουργικότητα της εφαρμογής δηλαδή το **business logic (Business Tier)** καθώς και την διασύνδεση με το **Client-Tier (Web-Tier)**.

### Web-Tier

To Web-Tier περιλαμβάνει τα **Servlets, HttpServletRequest Classes, HttpServletResponse Classes, Sessions, Filters, JSPs και Request Dispatchers**. Αυτά τα components ορίζουν τις δομές δεδομένων που αντιστοιχούν στις αιτήσεις και τις απαντήσεις του HTTP πρωτοκόλλου, τα δεδομένα **συνόδου (Session)** καθώς επίσης μηχανισμούς που ενεργοποιούν τις **μεταβάσεις και ανακατεύθυνσεις** μεταξύ των web resources (π.χ. HTML σελίδες) μίας web εφαρμογής. Έχουν να κάνουν δηλαδή με την λήψη των HTTP Requests του χρήστη, την μετατροπή τους σε μία δομή της Java και την επιστροφή HTTP Responses ή την ανακατεύθυνση των αιτήσεων του χρήστη σε άλλα components.

Επίσης σε αυτό το επίπεδο ο προγραμματιστής ορίζει τον **κώδικα που περιλαμβάνει την λογική για το ποια resources ή components θα χρησιμοποιηθούν για την προβολή των αποτελεσμάτων της επεξεργασίας των αιτήσεων στον χρήστη** (μπορεί δηλαδή να

χρησιμοποιηθούν διαφορετικά resources για να εμφανίσουμε το ίδιο ή παραπλήσιο αποτέλεσμα επεξεργασίας της αίτησης που θα εξαρτάται από διάφορες παραμέτρους π.χ. σε ποια ομάδα χρηστών ανήκει ο χρήστης, από ποια σελίδα του ιστότοπου πραγματοποίησε την αίτησή του για τον πόρο, ποιοι περιορισμοί ασφάλειας ισχύουν την στιγμή που πραγματοποίησε την αίτηση κ.τ.λ.). Ένα συγκεκριμένο αποτέλεσμα της επεξεργασίας μιας αίτησης δηλαδή δεν αντιστοιχεί αναγκαστικά σε έναν συγκεκριμένο πόρο (είτε η επιλογή του πόρου γίνεται δυναμικά κατά το runtime είτε ενδέχεται να γίνει τροποποίηση του κώδικα στο μέλλον για να αλλάξει ο πόρος). Ως πόρος πολλές φορές εννοείται κάποια JSP.

Η αντιστοίχιση μεταξύ των αποτελεσμάτων της επεξεργασίας των αιτήσεων (που προέρχονται από το Business-Tier που αναλύεται παρακάτω) και των πόρων που αναλαμβάνουν την παρουσίασή τους (που είναι συστατικά του Web-Tier) θα πρέπει να γίνεται στο Web-Tier ώστε αυτή να φαίνεται καθαρά, να μπορεί να δεχτεί αλλαγές χωρίς να επηρεάσει άλλο κώδικα και να μην ανακατεύεται με το business logic.

## Business-Tier

### Περιγραφή

Το **Business-Tier** στεγάζει το **business logic** της εφαρμογής, αυτό έχει να κάνει με την πραγματική λειτουργικότητα της εφαρμογής και τις υπηρεσίες που προσφέρει. Π.χ. σε μία εφαρμογή ενός e-shop το business logic περιλαμβάνει μεθόδους για καταχώρηση προϊόντος στο καλάθι, ολοκλήρωση πληρωμής με πιστωτική κάρτα, αναζήτηση προϊόντος με βάση την μάρκα κατασκευαστή κ.τ.λ.. Αποτελείται από Enterprise Java Beans και όλες τις κλάσεις οι οποίες έχουν να κάνουν με το **business logic** του domain της επιχείρησης (π.χ. telecommunications, banking, e-shop etc.).

### Ο Διαχωρισμός μεταξύ του Web και του Business-Tier

#### Λόγοι

Ο διαχωρισμός του Middle-Tier σε Web και Business-Tier είναι σημαντικός, που σημαίνει ότι ο κώδικας του Business-Tier θα πρέπει να είναι ανεξάρτητος από τις λεπτομέρειες της επικοινωνίας μέσω του HTTP πρωτοκόλλου. Δηλαδή να μην ανακατεύεται με την παραγωγή και την ανάγνωση HTTP μηνυμάτων. Γίνεται για δύο λόγους:

#### Επαναχρησιμοποιησιμότητα (Reusability)

Πρώτον, αυτό θα έχει ως αποτέλεσμα να μπορεί να μεταφερθεί (όλος ή μέρος του) σε οποιοδήποτε Web Framework (Struts2, Spring, JavaServer Faces κ.τ.λ.)<sup>1</sup> εάν στο μέλλον αποφασιστεί να γίνει μια τέτοια αλλαγή αλλά και ακόμα και σε εφαρμογές που δεν είναι διαδικτυακές αλλά εντελώς διαφορετικού τύπου (π.χ. standalone Java εφαρμογές).

#### Εφαρμογή QoS Προτύπων (Concurrency Management, Transaction Management, Dependency Injection, Lifecycle Management)

<sup>1</sup> Τα Web Frameworks που αναφέρθηκαν βασίζονται κατά κανόνα στα Servlets και JSPs για την υλοποίησή τους (που σημαίνει ότι είναι κατασκευασμένα πάνω σε αυτά) αλλά τα αντικαθιστούν και προσφέρουν στον προγραμματιστή μία πλουσιότερη και βολικότερη διεπαφή για την ανάπτυξη του Web-Tier.

**Δεύτερον, υπάρχουν πρότυπα τα οποία είναι σχεδιασμένα για την υποστήριξη του Business-Tier και απαιτούν ο κώδικας του να είναι ορισμένος ξεχωριστά από τα στοιχεία του Web-Tier** όπως είναι τα Servlets και τα JSPs. Ένα τέτοιο πρότυπο το οποίο και θα αναλυθεί στα πλαίσια αυτής της εργασίας είναι και τα **EJB (Enterprise Java Beans)**. Τέτοια πρότυπα προσφέρουν εργαλεία για την αυτόματη και δομημένη διαχείριση της **παραλληλίας** και των **συναλλαγών**, δύο από τις δυσκολότερες προκλήσεις οι οποίες παρουσιάζονται σε επιχειρησιακές εφαρμογές. Η προσπάθεια διαχείρισής τους χωρίς την βοήθεια των προαναφερθέντων προτύπων είναι δύσκολη και επιρρεπής σε προγραμματιστικά λάθη, τα οποία, έχουν ως αποτέλεσμα σφάλματα κατά την εκτέλεση που πολλές φορές δεν είναι δυνατόν να αναπαραχθούν κατά βούληση αυτουνού που θα ασχοληθεί με την διόρθωσή τους κατά την φάση της απασφαλμάτωσης.

Επίσης εκτός από αυτές τις υπηρεσίες υπάρχουν πρότυπα τα οποία ασχολούνται με την **διαχείριση του κύκλου ζωής των αντικειμένων της εφαρμογής (Lifecycle Management)** καθώς και την **διαχείριση των εξαρτήσεων μεταξύ τους (Dependency Injection, DI)**. Όλες οι προαναφερθέσεις υπηρεσίες ενσωματώνονται πιο εύκολα στον κώδικα της εφαρμογής εάν υπάρχει μια σαφής διάκριση μεταξύ του Web και του Business-Tier.

### Συμβουλές Σχεδίασης

Στις περισσότερες περιπτώσεις εφαρμογών είναι δύσκολο να ορίσουμε μία αυστηρή διαχωριστική γραμμή μεταξύ του Web-Tier και του Business-Tier ειδικά εάν δεν έχουμε πολλή εμπειρία στην ανάπτυξη δυναμικών ιστοσελίδων και μεγάλων επιχειρησιακών εφαρμογών. Το Web-Tier καταλήγει συχνά να έχει αρκετή Business λογική εάν δεν σχεδιαστεί προσεκτικά, πράγμα που δεν είναι ιδανικό από σχεδιαστική άποψη αλλά γίνεται πολλές φορές από άγνοια των αρχών του σωστού development των εφαρμογών ή εξαιτίας της απλότητας κάποιων εφαρμογών οι οποίες είναι μικρές και δεν αναμένουμε να υπάρξει ανάγκη επέκτασής τους.

**Για να ξεπεράσουμε αυτήν την δυσκολία και να επιτύχουμε σωστό διαχωρισμό Web και Business-Tier ένας τρόπος θα ήταν να υποθέσουμε ότι κάποια στιγμή θα αναπτύξουμε μία standalone Java εφαρμογή η οποία θα προσφέρει όλη την λειτουργικότητα της δυναμικής ιστοσελίδας που αναπτύσσουμε. Όλος ο κώδικας που μπορεί να μεταφερθεί σε αυτήν (που δεν αναφέρεται σε HTTP Requests/Responses, Sessions, Forwards, Redirects κ.τ.λ.) θα πρέπει να ανήκει στο Business-Tier.**

Αυτό βέβαια είναι ένα εντελώς υποθετικό σενάριο που πρακτικά δεν πρόκειται να προκύψει κατά πάσα πιθανότητα. Παρόλα αυτά σχεδιάζοντας την εφαρμογή με αυτό το σκεπτικό θα καταλήξουμε να έχουμε όσον το δυνατόν περισσότερο κώδικα που θα μπορούσαμε να μεταφέρουμε αυτούσιο, χωρίς καμία αλλαγή π.χ. σε κάποιο Web Framework, εάν αποφασίσουμε να το κάνουμε κάποια στιγμή στο μέλλον.

### EIS-tier

Αυτό το επίπεδο έχει να κάνει με **Enterprise Information Systems** τα οποία μπορεί να είναι απλά όπως ένα **σχεσιακό DBMS (Database Management System)** ή πολύπλοκα και με προχωρημένες δυνατότητες όπως ένα **ERP (Enterprise Resource Planning)**.

Παίρνοντας ως παράδειγμα μία εφαρμογή που επικοινωνεί με ένα συνηθισμένο σχεσιακό DBMS διαπιστώνουμε εύκολα ότι όταν δεν αναθέτουμε στην ίδια την εφαρμογή την αποθήκευση των δεδομένων αλλά σε ένα τρίτο σύστημα έχουμε την δυνατότητα να διαχειριστούμε και να επεκτείνουμε αυτό το σύστημα σε ανεξαρτησία με την υπόλοιπη εφαρμογή όσον αφορά το hardware αλλά και το software. Π.χ. θα μπορούσαμε να ρυθμίσουμε την χρήση των σκληρών

δίσκων, τις δομές δεδομένων που χρησιμοποιούνται, τα ευρετήρια, τα αρχεία καταγραφής για την επαναφορά των δεδομένων, το auditing κ.τ.λ.. Όλα αυτά χωρίς να χρειάζεται οποιαδήποτε αλλαγή στον κώδικα των άλλων δύο tiers αλλά ούτε και το υλικό στο οποίο εκτελούνται τα αντίστοιχα κομμάτια της εφαρμογής.

**Μία επέκταση αυτής της ανεξαρτησίας (σε κατάλληλα σχεδιασμένο Middle Tier) είναι επίσης η ανεξαρτησία του Middle-Tier από την σχεσιακή βάση του EIS-Tier. Π.χ. εάν η πρόσβαση στην βάση γίνεται μέσω κάποιου ORM Framework όπως το Hibernate θα μπορούσαμε να αλλάξουμε την βάση από π.χ. Microsoft SQL Server σε Oracle αλλάζοντας μόνο κάποιες ρυθμίσεις στα κατάλληλα configuration files χωρίς αλλαγές στον ίανα κώδικα.**

Στα πλαίσια αυτής της εργασίας θα παρουσιαστούν παραδείγματα που υλοποιούνται με **JPA (Java Persistence API)** το οποίο είναι το ORM API που προσφέρεται από το Java EE πρότυπο και αποτελεί βάση για το Hibernate.

## Συνοψίζοντας

Από όλα αυτά συνεπάγονται τα εξής για τα εκτελέσιμα συστατικά της επιχειρησιακής εφαρμογής ανά Tier:

1. **Κάθε επίπεδο της εφαρμογής θα μπορούσε να τρέχει σε διαφορετικό φυσικό μηχάνημα από τα άλλα άρα και να υπάρχει ευελιξία στην επιλογή, ρύθμιση και εγκατάσταση του hardware ώστε να γίνεται βελτιστοποίησή του ανά Tier.**
2. Μπορούν να αποδοθούν στο κάθε επίπεδο **περισσότεροι πόροι ανεξάρτητα από τα άλλα επίπεδα (hardware scalability)** όσον αφορά το hardware ή να αυξηθούν τα νήματα και τα processes που εκτελούν τον κώδικα που τρέχει στο συγκεκριμένο επίπεδο (**software scalability**).
3. Μπορεί να γίνεται **παραμετροποίηση της εκτέλεσης και λειτουργίας** του λογισμικού ξεχωριστά ανά tier χωρίς να επηρεάζει το ένα το άλλο.
4. **Ο κώδικάς του κάθε επιπέδου μπορεί να γραφτεί ανεξάρτητα από τα άλλα επίπεδα από διαφορετικές ομάδες προγραμματιστών.** Ως προς την διαδικασία ανάπτυξης δηλαδή της εφαρμογής, μία n-tier αρχιτεκτονική επιτρέπει τον καλύτερο διαχωρισμό του έργου σε πολλούς ανθρώπους όπου ο καθένας από αυτούς δεν χρειάζεται να έχει πλήρη εικόνα όλης της εφαρμογής αλλά ούτε και εξαιρετικά εξειδικευμένες γνώσεις για όλες τις τεχνικές πτυχές της (π.χ. να ξέρει ταυτόχρονα Web Design αλλά και Database Management), αυτό το χαρακτηριστικό των εφαρμογών ονομάζεται **“Separation of Concerns”**.
5. Κάθε Tier μπορεί να έχει ορισμένους **ανεξάρτητους κανόνες ασφάλειας**.
6. Κάθε Tier μπορεί να **αλλάξει** και να **αντικατασταθεί** από μία άλλη υλοποίηση.
7. Σε ένα Tier μπορούν να **συνυπάρχουν δύο υλοποιήσεις** ταυτοχρόνως και να συνεργάζονται χωρίς προβλήματα με τα άλλα Tiers χωρίς επιπλέον αλλαγές σε αυτά (όπως π.χ. το Client Tier ενός web application που είναι υλοποιημένο και για Browsers αλλά και ως Android Apps).

# Containers and Components

Η Java EE ορίζει τις έννοιες των **Containers** και των **Components**. Τα Containers είναι στοιχεία λογισμικού τα οποία παρέχονται από κάποιον κατασκευαστή αυτούσια ή ως μέρος ενός Application Server.

**Ο σκοπός των Containers είναι να φιλοξενούν Components, να διαχειρίζονται δηλαδή τον κύκλο ζωής τους και να τους προσφέρουν υπηρεσίες (Container Services).** Τα Components παρέχονται από τον προγραμματιστή της εφαρμογής και είναι αντικείμενα της Java που δημιουργήθηκαν με βάση κάποιο πρότυπο.

Π.χ. ένας Application Server περιλαμβάνει μεταξύ των άλλων και έναν **Servlet/JSP Container** (ή αλλιώς “**Web Container**”) ο οποίος μπορεί να φιλοξενεί Servlets και JSPs (στοιχεία για την παροχή δυναμικού περιεχομένου ιστοσελίδων). Αυτό σημαίνει ότι ο **Servlet/JSP Container** αναλαμβάνει να φτιάξει τα αντικείμενα (instances) για κάθε Servlet και JSP που έχει δηλώσει ο προγραμματιστής, να διαχειριστεί τον κύκλο ζωής τους και την μεταφορά των HTTP αιτήσεων του χρήστη προς αυτά.

Αφού ο προγραμματιστής γράψει τις κλάσεις των Servlets θα πρέπει να τις δηλώσει έτσι ώστε ο Container να γνωρίζει ποιες είναι. **Η δήλωσή τους γίνεται από αυτόν συνήθως σχεδόν αποκλειστικά με δηλωτικό (declarative) και όχι προγραμματιστικό (programmatic) τρόπο,** δηλαδή μέσω annotations ή/και XML αρχείων και όχι μέσω κώδικα που εκτελείται κατά το runtime (αν και υπάρχει αυτή η δυνατότητα).

Ο δηλωτικός τρόπος προτιμάται καθώς επιτρέπει αλλαγές των URLs (και κάποιων άλλων παραμέτρων) που αντιστοιχούν στους Servlets έξω από τον Java κώδικα καθώς και την ύπαρξη εναλλακτικών συνόλων από configurations (XML αρχείων δηλαδή π.χ. με παραμέτρους εκτέλεσης ορισμένες για εκτέλεση σε τεστ περιβάλλοντα).

**Όσον αφορά τα Container Services, οι Containers αναλαμβάνουν να προσφέρουν στα Components τις διάφορες απαραίτητες υπηρεσίες που έχει ανάγκη μια επιχειρησιακή εφαρμογή όπως είναι π.χ. Transaction Management, Concurrency Management, State Management (Sessions), Messaging Services κ.τ.λ. που θα αναλυθούν στην συνέχεια.**

## Server Side Containers

Παρακάτω παραθέτονται τα είδη των Containers και τα Components που φιλοξενούν.

### EJB Container

Είναι ο Container που φιλοξενεί τα **Enterprise Java Beans**. Αυτά χωρίζονται σε **Session Beans** (υλοποιούν το Business Logic), **Entity Beans** (υλοποιούν την αποθήκευση δεδομένων σε κάποια βάση δεδομένων) και **Message Driven Beans** (υλοποιούν την επικοινωνία μέσω μηνυμάτων των διάφορων συστατικών της εφαρμογής καθώς και με απομακρυσμένα συστατικά έξω από αυτήν).

### Web Container (Servlet/JSP Container)

Είναι ο Container που φιλοξενεί **Servlets** και **JSP**. Διαχειρίζεται το **lifecycle** των Servlets και τα requests και responses που τους αντιστοιχούν. **Αναλαμβάνει την μετατροπή των μηνυμάτων του HTTP πρωτοκόλλου σε κλάσεις της Java λειτουργώντας έτσι ως ένα αποτελεσματικό επίπεδο διαχείρισης της πολυπλοκότητας του.** Επίσης εφαρμόζει τους κανόνες ασφάλειας που έχουν

οριστεί δηλωτικά ή προγραμματιστικά από τον προγραμματιστή. Συχνά αναφέρεται ως **Servlet Container** ή/και **JSP Container**. Ως προς τα επίσημα πρότυπα των Servlets και JSPs αυτά αποτελούν συνώνυμα.

## Client Side Containers

### [Application Client Container](#)

Διαχειρίζεται την εκτέλεση των **Application Client Components**. Έχει να κάνει με το πρότυπο της Java EE για την κατασκευή client side εφαρμογών που επικοινωνούν με κάποια server side υλοποίηση. Το συγκεκριμένο θέμα ξεφεύγει από τους στόχους της παρούσας εργασίας και δεν θα αναλυθεί.

### [Applet Container](#)

Διαχειρίζεται την εκτέλεση των **Java Applets**. Αποτελείται από έναν **Browser** και ένα **Plugin** για την εκτέλεση των **Java Applets**. Τα Java Applets επιτρέπουν την φόρτωση και εκτέλεση προγραμμάτων Java μέσα από τον Web Browser του χρήστη.

Τα Java Applets είχαν κάποια χρησιμότητα στο παρελθόν όταν δεν υπήρχε μεγάλη υποστήριξη και πολλά frameworks για την Javascript. Εξαιτίας πολλών θεμάτων ασφάλειας που προκαλούσε η εκτέλεση κώδικα που κατέβαινε στον υπολογιστή του χρήστη και εκτελούνταν μέσα στο JRE, καθώς και της σχετικά πιο βαριάς εκτέλεσης, τα Applets δεν χρησιμοποιούνται σχεδόν καθόλου σήμερα και για αυτό και δεν θα αναλυθούν ούτε και αυτά στην παρούσα εργασία.

## Container Services

Γενικά τα Containers θα πρέπει να προσφέρουν **διαχειρίσιμες υπηρεσίες (Configurable Services)** και **μη διαχειρίσιμες υπηρεσίες (Non-Configurable Services)** στα Components που φιλοξενούν. Οι διαχειρίσιμες υπηρεσίες είναι οι υπηρεσίες τις οποίες μπορεί να παραμετροποιήσει ο προγραμματιστής ή/και να τις χρησιμοποιήσει ρητά στα προγράμματά του. Οι μη διαχειρίσιμες υπηρεσίες είναι υπηρεσίες όπως π.χ. η διαχείριση του κύκλου ζωής των Servlets από τον Web Container ή διαχείριση του κύκλου των EJBs από τον EJB Container. Ο προγραμματιστής δεν μπορεί να τις χρησιμοποιήσει ρητά στα προγράμματά του αλλά ίσως να έχει δυνατότητα για την παραμετροποίησή τους ως έναν βαθμό.

Παρακάτω παρουσιάζονται όλες οι **διαχειρίσιμες υπηρεσίες** του Java Enterprise Edition Specification που πρέπει να παρέχουν οι Java Application Servers.

### [Web Services](#)

Περιλαμβάνουν τα πρότυπα **JAX-WS (Big Webservices)** και **JAX-RS (REST Webservices, Representational State Transfer Webservices ή και Restful Webservices)**.

Το **JAX-WS** αναφέρεται μερικές φορές και ως **JAX-RPC (έκδοση 2.0)** που όμως είναι όρος που χρησιμοποιούταν μέχρι και το 2005. Εξαιτίας των πολλών και σημαντικών αλλαγών μεταξύ των εκδόσεων 1.x και 2.0 του JAX-RPC προτιμάται ο όρος JAX-WS ώστε να τονίζεται η ασυμβατότητα μεταξύ των δύο εκδόσεων.

Στην περίπτωση των **JAX-WS** επωφελούμαστε από **αυστηρή τυποποίηση** και **QoS (Quality of Service)** ενώ στην περίπτωση των **JAX-RS** webservices επωφελούμαστε από ένα **πιο ελαφρύ**

**πρωτόκολλο και από το καθιερωμένο HTTP πρωτόκολλο με ό,τι αυτό συνεπάγεται** (χρήση ήδη υπάρχων caching υποδομών, διαφοροποίηση των δεδομένων και της αναπαράστασης, ανεξαρτησία από πρωτόκολλα όπως το SOAP κ.τ.λ.).

## JAX-WS

**Παρέχουν απομακρυσμένες κλήσεις μεθόδων μέσω του HTTP πρωτοκόλλου.** Επειδή βασίζεται σε γενικά πρότυπα, όπως τα πρωτόκολλα **HTTP, WSDL (Webservice Description Language) και SOAP (Simple Object Access Protocol)**, όπου τα δύο τελευταία εκφράζονται μέσω XML αρχείων, αποτελεί πρότυπο το οποίο είναι ανεξάρτητο από κάποια συγκεκριμένη γλώσσα προγραμματισμού. Αυτό σημαίνει ότι υποστηρίζει επικοινωνία μεταξύ απομακρυσμένων προγραμμάτων που μπορούν να είναι γραμμένα σε διαφορετικές γλώσσες προγραμματισμού.

Όσον αφορά την Java αυτό επιτυγχάνεται με **μία κλάση υλοποίησης των μεθόδων**, αντικείμενο της οποίας δημιουργείται στον server, και **μία κλάση κλήσης αυτών των μεθόδων** που δημιουργείται στα μηχανήματα των απομακρυσμένων clients. Και οι δύο κλήσεις εφαρμόζουν την ίδια διεπαφή.

**Η κλάση που κάνει τις κλήσεις των μεθόδων στους clients δημιουργείται μέσω ειδικού εργαλείου γραμμής εντολών που προσφέρει η Java (wsimport) και δεν χρειάζεται να γραφτεί από τον προγραμματιστή.**

Η κλάση υλοποίησης θα πρέπει να υλοποιηθεί από τον προγραμματιστή **ορίζοντας μόνο την λογική των μεθόδων που θα υλοποιεί** και όχι κάποιον επιπλέον κώδικα για απομακρυσμένη επικοινωνία μέσω διαδικτύου. Προφανώς αντικείμενό της πρέπει να βρίσκεται σε έναν server για να είναι συνεχώς διαθέσιμο για να λαμβάνει απομακρυσμένες κλήσεις από τους clients. Τον ρόλο του server μπορεί να τον υλοποιήσει ένα Java EE Runtime όπως ένας **Application Server** ή ένας **Servlet Container**.

Οι περισσότερες υλοποιήσεις των Application Servers θα κάνουν αυτόματα χρήση του **“wsgen”** εργαλείου της γραμμής εντολών για να φτιάξουν το **Webservice Endpoint** και να το κάνουν **deploy**, αρκεί η κλάση υλοποίησης του Webservice να έχει δηλωθεί με annotation ή σε κάποιο configuration αρχείο.

Σημειώνεται ότι υπάρχει επίσης και η δυνατότητα δημιουργίας ενός τέτοιου server ο οποίος θα φιλοξενεί την κλάση υλοποίησης και μέσα σε ένα Standalone Java πρόγραμμα (Java SE) χρησιμοποιώντας την κλάση **javax.xml.ws.Endpoint** και την μέθοδο **publish**. Αυτό θα έχει ως αποτέλεσμα την δημιουργία ενός Thread το οποίο θα συνεχίζει να εκτελείται και μετά το πέρας της εκτέλεσης της μεθόδου “main” και θα φιλοξενεί την κλάση της υλοποίησης του Webservice στην οποία θα μεταβιβάζει τις απομακρυσμένες κλήσεις των clients.

Εξαιτίας των ιδιαιτεροτήτων της ενσωμάτωσης των Webservices στα Java EE Runtimes όπου δεν απαιτείται η χρήση της κλάσης **javax.xml.ws.Endpoint** και υπάρχουν κάποιες άλλες μικρές ιδιαιτερότητες ανάλογα με το Runtime που χρησιμοποιούμε, τα Webservices αναφέρονται στο παρών κείμενο ως Διαχειρίσημη Υπηρεσία των Java EE Runtimes.

## Security

Περιλαμβάνει μηχανισμούς και πρότυπα για **Αυθεντικοποίηση/Ταυτοποίηση (Authentication/ Login), Εξουσιοδότηση Πρόσβασης σε Πόρους (Authorization/Access Control), Κρυπτογράφηση (Encryption) και Ακεραιότητα Δεδομένων (Data Integrity, αναφορικά με τα μηνύματα τα οποία ανταλλάσσονται μεταξύ του server και του client).**

Οι μηχανισμοί αυτοί ρυθμίζονται με **δηλωτικό (declarative)** αλλά και **προγραμματιστικό (programmatic)** τρόπο.

Ο δηλωτικός τρόπος ορισμού των παραμέτρων της ασφάλειας:

1. Καθιστά τις ρυθμίσεις της ασφάλειας **περισσότερο σαφείς** και **μειώνει τον χρόνο που χρειάζεται για την εξέτασή** τους καθώς βρίσκονται συγκεντρωμένες σε ένα XML αρχείο περιγραφής, όπου είναι διατυπωμένες σύμφωνα με ένα συγκεκριμένο πρότυπο.
2. **Διευκολύνει τις αλλαγές** και **τις προσθήκες στους περιορισμούς, τους ελέγχους** και **τις ρυθμίσεις** της ασφάλειας.
3. Επιτρέπει σε **μία διαδικτυακή εφαρμογή** να μπορεί να εγκατασταθεί με διαφορετικές ρυθμίσεις ασφάλειας σε διαφορετικά περιβάλλοντα χωρίς να χρειάζεται να γίνει αλλαγή του κώδικα και εκ νέου μετάφρασή του παρά μόνο η αναδημοσίευσή του, **έχοντας διαφορετικά αρχεία περιγραφής** ανά **περίπτωση**. Π.χ. μπορούμε να έχουμε διαφορετικές ρυθμίσεις ασφάλειας στο παραγωγικό περιβάλλον, όπου η εφαρμογή εξυπηρετεί τους τελικούς χρήστες, και άλλες στο τεστ περιβάλλον, στο οποίο πραγματοποιούμε την ανάπτυξη της εφαρμογής και όπου ενδεχομένως θα θέλαμε να τις χαλαρώσουμε ή να τις ακυρώσουμε τελείως για την διευκόλυνση της διερεύνησης προβλημάτων γενικότερης λειτουργίας.

Ο προγραμματιστικός τρόπος για την ρύθμιση των μηχανισμών ασφάλειας από την άλλη μας εξυπηρετεί στην περίπτωση όπου θα πρέπει να αποφασίσουμε για την εφαρμογή ή μη εφαρμογή κάποιας από τις προαναφερθείσες λειτουργίες (Authentication, Authorization, Access Control, Encryption, Integrity Checking) **κατά την εκτέλεση** της εφαρμογής με βάση παραμέτρους οι οποίες είναι διαθέσιμες μόνο την στιγμή που ο χρήστης πραγματοποιεί την αίτηση για τον εκάστοτε πόρο ή εκτελεί κάποια άλλη ενέργεια για την οποία θα πρέπει να εξετασθεί το εάν είναι εξουσιοδοτημένος ή όχι να την πραγματοποιήσει.

Π.χ. σε μια ιστοσελίδα κάποιας αεροπορικής εταιρείας ενδέχεται να επιτρέπεται η προσπέλαση της σελίδας όπου ορίζονται οι προτιμήσεις σε φαγητό που θα σερβιριστεί στον πελάτη κατά την πτήση, γενικά, μόνο για τους golden class επιβάτες. Θα μπορούσε όμως από την πολιτική της εταιρείας να οριστεί ότι εάν την συγκεκριμένη μέρα της πτήσης είναι τα γενέθλια του επιβάτη ο περιορισμός αυτός να αναιρείται. Ο περιορισμός αυτός, τύπου Access Control, μπορεί να υλοποιηθεί μόνο προγραμματιστικά και αναγκαστικά θα πρέπει να ενσωματωθεί στον κώδικα της εφαρμογής μας και όχι σε αρχείο περιγραφής περιορισμών ασφάλειας.

## Java Naming and Directory Interface (JNDI)

**Java Naming and Directory Interface.** Είναι **υπηρεσία ονοματολογίας** και **ευρετηρίων** της Java. Πιο συγκεκριμένα παρέχει ένα ενοποιημένο και τυποποιημένο API για την σύνδεση με υπηρεσίες ονοματολογίας και ευρετηρίων όπως π.χ. **LDAP, DNS, NIS** και των **Directory Services** των διαφόρων λειτουργικών συστημάτων. Η υποστήριξη οποιουδήποτε Directory Service πρωτοκόλλου, είναι δυνατή εξαιτίας του **SPI (Service Provider Interface)** που έχει οριστεί για το JNDI.

Το **Service Provider Interface** είναι μία διεπαφή που έχει οριστεί για τους παρόχους υλοποιήσεων directory service πρωτοκόλλων. Έτσι οποιοσδήποτε παρέχει ένα καινούριο directory service πρωτόκολλο μπορεί να χρησιμοποιήσει αυτό το SPI και να φτιάξει μία βιβλιοθήκη την οποία μπορεί να χρησιμοποιήσει το JNDI μέσω του SPI για να παρέχει στον τελικό προγραμματιστή αυτό το directory service διαμέσου του ήδη υπάρχοντος JNDI πρωτοκόλλου.

Θα μπορούσαμε να παρομοιάσουμε το JNDI πρωτόκολλο σαν μία υπηρεσία όπως αυτή του file explorer στα windows αλλά στην περίπτωση του JNDI αυτή η υπηρεσία χρησιμοποιείται αποκλειστικά από προγράμματα Java και όχι χρήστες λειτουργικού συστήματος και αποθηκεύει Java αντικείμενα και όχι αρχεία.

Το JNDI πρωτόκολλο σε γενικές γραμμές παρέχει στον προγραμματιστή δύο κύριες υπηρεσίες. Η πρώτη (**υπηρεσία ονοματολογίας**) είναι οι μέθοδοι συσχετισμού ονομάτων και Java αντικειμένων (είτε αυτούσιων είτε με την μορφή αναφορών (references)). Η δεύτερη (**υπηρεσία ευρετηρίου**) είναι η υπηρεσία αναζήτησης αντικειμένων με βάση τις ιδιότητες οι οποίες έχουν συσχετιστεί μαζί τους.

Εκτός από αυτές τις υπηρεσίες παρέχει ενοποίηση διαφορετικών directory services και naming conventions (compound namespaces).

Το JNDI χρησιμοποιείται από τους Application Servers κυρίως για την διαχείριση και την παροχή συστατικών του JDBC και του JMS στις εφαρμογές που είναι ανεβασμένες σε αυτούς καθώς και άλλων αντικειμένων τα οποία βρίσκονται μέσα σε αυτά τα Runtime Environments (Application Servers) και πρέπει να είναι προσπελάσιμα ή/και να διαμοιράζονται μεταξύ των εφαρμογών που εκτελούνται σε αυτό.

Όσον αφορά το Java Enterprise Development θα μπορούσαμε να πούμε ότι είναι ένας τρόπος επικοινωνίας των εφαρμογών με το Runtime Environment στο οποίο εκτελούνται καθώς και με τις υπόλοιπες εφαρμογές που είναι ανεβασμένες σε αυτό.

## Java Database Connectivity (JDBC)

### **Γενική Περιγραφή**

Το συγκεκριμένο πρότυπο ονομάζεται **Java Database Connectivity** και είναι ένα API. Δηλαδή μία τυποποιημένη διασύνδεση που δεν ορίζει λεπτομέρειες υλοποίησης. Την υλοποίησή της αναλαμβάνουν οι **JDBC Drivers** οι οποίοι συνήθως αναπτύσσονται από τους κατασκευαστές των συστημάτων βάσεων δεδομένων και βρίσκονται μέσα σε ένα jar αρχείο.

**Αναλαμβάνει την επικοινωνία των Java εφαρμογών με βάσεις δεδομένων** και δεν είναι στην πραγματικότητα αποκλειστικά μέρος του Java EE αλλά **παρέχεται και από το Java Standard Edition** (πιο συγκεκριμένα παρέχονται όλες οι απαραίτητες κλάσεις και διεπαφές που χρειάζονται για την ενσωμάτωση και χρήση των JDBC Drivers). Δεν χρειάζεται δηλαδή κάποιον Application Server για να το υποστηρίξει.

**Αναφέρεται παρόλα αυτά εδώ ως Service** των Java EE Runtime Environments διότι γίνεται ενσωμάτωση αυτού του προτύπου (σε επίπεδο εργαλείων διαχείρισης) σε μεγάλο βαθμό από όλα τα Application Servers για την διευκόλυνση της διαχείρισης και της ρύθμισης των συνδέσεων που ανοίγονται από τις εφαρμογές του Application Server προς τις βάσεις δεδομένων. Οι ίδιες δομές για την παροχή συνδέσεων προς τις βάσεις δεδομένων, μπορούν να διαμοιράζονται μεταξύ πολλών εφαρμογών που είναι ανεβασμένες στον Application Server.

Ουσιαστικά ο Application Server προσφέρει εργαλεία για την δημιουργία και την παραμετροποίηση αντικειμένων τύπου **DataSource** τα οποία καταχωρούνται στο ευρετήριο (JNDI) που συντηρείται από αυτόν, και μπορούν έτσι να χρησιμοποιηθούν από τις εφαρμογές που έχουν γίνει deployed.

Τα αντικείμενα τύπου **DataSource** παράγουν αντικείμενα τύπου **Connection** μέσω των οποίων πραγματοποιείται η διασύνδεση των εφαρμογών του server με το τοπικό ή απομακρυσμένο σύστημα διαχείρισης βάσεων δεδομένων που ορίζει το **DataSource**. Αυτό σημαίνει ότι **το Connection αντικείμενο προσφέρει τρόπους για να μπορούν να στέλνονται Queries, DDL και DML SQL Statements προς την βάση δεδομένων και να λαμβάνονται τα αποτελέσματα στην μορφή των αντικειμένων και τύπων της Java**.

Όλα τα παραπάνω προσφέρονται μέσω ενός **προκαθορισμένου API το οποίο είναι ανεξάρτητο από την βάση δεδομένων στην οποία συνδέεται η εφαρμογή**. Αυτό σημαίνει ότι ο τρόπος υποβολής ερωτημάτων SQL και λίψης αποτελεσμάτων είναι τυποποιημένος, παρόλα αυτά **τα ίδια τα SQL ερωτήματα εξαρτώνται από την ΒΔ που χρησιμοποιείται κάθε φορά**.

Για να επιτευχθεί ανεξαρτησία με την SQL χρειάζεται επιπλέον κάποιο **ORM Framework** όπως το **Hibernate** ή και το **JPA (Java Persistence API)** που είναι μέρος του Java EE.

## **JDBC Drivers**

**Κάθε ΒΔ θα πρέπει να παρέχει έναν JDBC Driver, δηλαδή μία υλοποίηση του JDBC API για την συγκεκριμένη βάση δεδομένων εκτός και αν υποστηρίζει το ODBC.** Ο driver αυτός θα παρέχει την υλοποίηση του JDBC interface.

Στην περίπτωση εφαρμογών που φιλοξενούνται σε έναν Application Server ένας τέτοιος driver καταχωρείται μέσω των διαχειριστικών εργαλείων του Application Server ή με τον οποιονδήποτε άλλον τρόπο που ορίζει αυτός. Αυτό σημαίνει ότι θα μπορούσε να αλλάξει χωρίς να πραγματοποιηθεί αλλαγή στον κώδικα των εφαρμογών που είναι deployed<sup>1</sup>.

**Οι drivers αυτοί χωρίζονται σε 4 τύπους.** Ο διαχωρισμός αυτός σε μεγάλο βαθμό αφορά τους κατασκευαστές αυτών των drivers αλλά η ανάλυσή του δίνει μια γενική εικόνα για τον τρόπο κατασκευής και λειτουργίας τους που μπορεί να φανεί χρήσιμος σε διάφορες περιπτώσεις και στους προγραμματιστές των εφαρμογών που τους χρησιμοποιούν.

### **Type 1: JDBC – ODBC Bridge Driver**

Για να γίνει η περιγραφή αυτού του τύπου των drivers **Θα πρέπει να γίνει πρώτα μία αναφορά στους drivers τύπου ODBC.**

Το **ODBC** μεταφράζεται ως **Open Database Connectivity**, είναι ένα API της γλώσσας “C” και έχει τον ίδιο στόχο με το JDBC αλλά είναι πιο παλιό πρότυπο που αναπτύχθηκε από την Microsoft. Στόχος αυτού του API δηλαδή είναι η διασύνδεση με βάσεις δεδομένων διαμέσου ενός κοινού API για την υποβολή SQL Statements.

**Όπως και το JDBC είναι ανεξάρτητο από την ΒΔ αλλά προφανώς τα SQL Statements που υποβάλλονται διαμέσου αυτού δεν είναι.**

**Το ODBC δεν είναι ανεξάρτητο από το λειτουργικό σύστημα στο οποίο τρέχουν οι εφαρμογές που το χρησιμοποιούνε και θα πρέπει να είναι εγκατεστημένο σε αυτό.** Οι τεχνικές λεπτομέρειες της εγκατάστασής του διαφέρουν από λειτουργικό σε λειτουργικό.

<sup>1</sup> Παρόλα αυτά όταν η αλλαγή αυτή γίνεται εξαιτίας της αλλαγής στην ΒΔ που χρησιμοποιούμε, οι εφαρμογές που είναι ανεβασμένες στον Application Server, στην περίπτωση που δεν χρησιμοποιούν κάποιο ORM, θα πρέπει να ορίζουν τον SQL κώδικα εξωτερικά του πηγαίου Java κώδικα έτσι ώστε να μπορεί να μην χρειαστούν αλλαγές στον κώδικά τους.

**Κάθε κατασκευαστής μιας ΒΔ (π.χ. Microsoft SQL Server) προφανώς θα πρέπει να έχει φροντίσει για την υποστήριξη του ODBC αλλά όχι της Java συγκεκριμένα** οπότε είναι εμφανές ότι στην περίπτωση που χρησιμοποιούμε κάποια ΒΔ που δεν υποστηρίζει Java είναι πολύ πιθανόν να βρούμε την λύση στο ODBC. Από την πλευρά του κατασκευαστή κάτι τέτοιο τον απαλλάσσει από το έργο της ανάπτυξης της υποστήριξης για την Java.

Στην περίπτωση των ODBC drivers υπάρχει και ο περιορισμός του ότι **το λειτουργικό στο οποίο στεγάζεται η ΒΔ θα πρέπει να υποστηρίζει το ODBC**. Ως εκ τούτου δεν μπορούμε να θεωρούμε δεδομένη την πλήρη και χωρίς προβλήματα υποστήριξή του σε όλες τις πλατφόρμες.

**To JDBC-ODBC Bridge Driver αναλαμβάνει την επικοινωνία των Java εφαρμογών με τον ODBC Driver.** Οι εφαρμογές αυτές χρησιμοποιούν τις java μεθόδους του JDBC και διαμέσου αυτού επικοινωνούν με την ΒΔ. Ο driver αυτός παρέχεται από τις βασικές βιβλιοθήκες της java και δεν χρειάζεται να κατέβει ξεχωριστά όπως οι drivers των άλλων τριών τύπων.

Προφανώς **δεν είναι ανεξάρτητος του λειτουργικού συστήματος** όπως εξάλλου και η έκδοση του JDK που τον περιλαμβάνει. Αυτό επειδή χρησιμοποιεί κλήσεις του λειτουργικού συστήματος για το οποίο κατασκευάστηκε για να πραγματοποιήσει τις κλήσεις του εγκατεστημένου ODBC Driver. Η εταιρεία της Java (η Oracle) αναλαμβάνει την ανάπτυξη αυτού του driver για κάθε λειτουργικό σύστημα για το οποίο παρέχει εκδόσεις του JDK που είναι προγενέστερες από την έκδοση JDK 1.8<sup>1</sup>.

**To JDBC-ODBC Bridge Driver χρησιμοποιούνταν περισσότερο στο παρελθόν** όταν ακόμα οι κατασκευαστές των συστημάτων βάσεων δεδομένων δεν είχαν αναπτύξει υποστήριξη και για την Java. Από την στιγμή που μία ΒΔ υποστήριζε το ODBC (που εξυπηρετεί την “C”) δεν χρειαζόταν να αναπτύξει κάτι επιπλέον ώστε να υποστηρίξει και την Java αφού το JDBC-ODBC Bridge το πρόσφερε η ίδια η Sun (η παλιά κατασκευαστρια εταιρεία της Java).

Εξαιτίας της **έμμεσης επικοινωνίας** της εφαρμογής με την ΒΔ ο τύπος αυτών των drivers γενικά αποφεύγεται επειδή υπάρχει ένα **μεγαλύτερο overhead συγκριτικά με τους άλλους τύπους**. Ο JDBC-ODBC Bridge Driver δηλαδή πραγματοποιεί με κάποιον τρόπο κλήσεις προς τον ODBC Driver (ο κώδικας είναι proprietary) και αυτός στη συνέχεια επικοινωνεί με την εκάστοτε ΒΔ μέσω της διασύνδεσης που αυτή υποστηρίζει με το ODBC. Αυτή η διασύνδεση με την οποία επικοινωνεί το ODBC είναι και αυτή που επικοινωνεί τελικά με την ΒΔ.

## Type 2: JDBC – Native API

**Σε αυτόν τον τύπο οι μέθοδοι του JDBC API καλούνται μεθόδους που παρέχονται από την εκάστοτε ΒΔ.** Παράδειγμα τέτοιων μεθόδων που παρέχονται από την ΒΔ της Oracle είναι τα Oracle OCI Drivers που είναι μία διεπαφή με την ΒΔ της Oracle στην γλώσσα “C”. Σε αυτήν την περίπτωση **υπάρχει εξάρτηση από το λειτουργικό σύστημα** αφού οι τρόποι επικοινωνίας μεταξύ διεργασιών διαφέρουν από λειτουργικό σε λειτουργικό και δεν περιλαμβάνουν μόνο τα TCP Sockets. **Ο κατασκευαστής της ΒΔ επιβαρύνεται και με την δημιουργία των JDBC Drivers για κάθε διαφορετικό λειτουργικό.**

<sup>1</sup> Από την έκδοση του JDK 1.8 το JDBC-ODBC Bridge Driver πια δεν παρέχεται αφού πρακτικά όλοι οι κατασκευαστές των σημαντικότερων βάσεων δεδομένων εδώ και καιρό παρέχουν JDBC Drivers των άλλων τύπων.

### Type 3: JDBC - Network-Protocol driver (middleware driver)

Αυτός ο τύπος του driver παρέχεται από τον Application Server, δηλαδή από το Middleware. Αυτό σημαίνει ότι θα παραμένει ο ίδιος για όλες τις ΒΔ τις οποίες υποστηρίζει ο Application Server. Ο driver αυτός δεν επικοινωνεί με τις ΒΔ. Αντιθέτως, αναλαμβάνει να επικοινωνήσει με το middleware, δηλαδή κάποιο υποσύστημα του Application Server. Αυτό το υποσύστημα με την σειρά του επικοινωνεί με τις ΒΔ με τρόπους οι οποίες ιδανικά δεν απασχολούν τους προγραμματιστές και διαχειριστές των web εφαρμογών. Προφανώς το μειονέκτημα αυτού του driver είναι ότι θα πρέπει ο κατασκευαστής του Application Server να φροντίσει για την υποστήριξη πολλών ΒΔ. Από την άλλη, όπως και οι drivers τύπου 4 είναι ανεξάρτητος του λειτουργικού συστήματος αφού είναι γραμμένος εξολοκλήρου σε Java οπότε και διευκολύνει τους διαχειριστές και προγραμματιστές των web applications σε περίπτωση αλλαγής του λειτουργικού συστήματος.

### Type 4: JDBC – Database-Protocol driver (Direct to Database Pure Java Driver)

Οι drivers αυτού του τύπου είναι γραμμένοι εξολοκλήρου σε Java και διαφέρουν στην υλοποίηση για κάθε ΒΔ αφού τα πρωτόκολλα που χρησιμοποιούν αυτές οι βάσεις διαφέρουν κατά πολύ και στην πλειονότητα των περιπτώσεων είναι proprietary. Οι drivers αυτοί μοιάζουν με τους drivers τύπου 2 αλλά επειδή είναι γραμμένοι εξολοκλήρου σε Java έχουν βελτιωμένη απόδοση εξαιτίας της αμεσότητας με την οποία πραγματοποιούν την επικοινωνία. Επίσης προφανώς είναι ανεξάρτητοι του λειτουργικού συστήματος. Συνήθως επικοινωνούν με την ΒΔ μέσω TCP Sockets και κάποιου proprietary database network πρωτοκόλλου του κατασκευαστή της ΒΔ.

## Java Messaging Service (JMS)

### Γενική Περιγραφή

Ένα από τα πρότυπα που ορίζονται στην Java EE είναι το **JMS (Java Message Service)**. Είναι πρότυπο που ικανοποιεί τις ανάγκες για ασύγχρονη απομακρυσμένη επικοινωνία μεταξύ εφαρμογών και μεταξύ των συστατικών τους. Η διαφορά του με τα πρότυπα απομακρυσμένης επικοινωνίας όπως τα JAX-WS και JAX-RS έγκειται στο ότι είναι ασύγχρονο, που σημαίνει ότι το σύστημα που καλείται μπορεί να επεξεργαστεί την κλήση όποτε αυτό είναι έτοιμο να το κάνει, ενώ το σύστημα το οποίο πραγματοποιεί την κλήση δεν χρειάζεται να περιμένει απάντηση αλλά μετά την κλήση μπορεί να συνεχίσει την ροή εκτέλεσης της μεθόδου που την πραγματοποίησε.

Ουσιαστικά αντί για την απομακρυσμένη σύγχρονη κλήση έχουμε την έννοια της **αποστολής μηνύματος** το οποίο καταχωρείται σε ένα ενδιάμεσο σύστημα που ονομάζεται **Destination** και είναι τύπου **Queue** ή **Topic**. Εκεί αποθηκεύεται μέχρις ότου κάποιο από τα συστήματα που έχει αναλάβει τον ρόλο να επεξεργάζεται τα μηνύματα να είναι έτοιμο να το δεχτεί και να το επεξεργαστεί.

Με το JMS υπάρχει η δυνατότητα για την εμφάνιση μηνύματος στον πελάτη ότι κάποιο αίτημά του βρίσκεται υπό επεξεργασία και θα ενημερωθεί κάποια στιγμή για την ολοκλήρωσή του. Κάτι τέτοιο δεν θα ήταν δυνατόν εάν αντί για την αποστολή μηνύματος κάναμε χρήση της κλήσης ενός Webservice, σε αυτήν την περίπτωση θα ήμασταν αναγκασμένοι να περιμένουμε για την ολοκλήρωσή της πριν συνεχιστεί η ροή εκτέλεσης του προγράμματος που στην συνέχεια θα

μπορούσε να πραγματοποιήσει και την επιστροφή του HTTP Response προς τον πελάτη. Κατ' αυτόν τον τρόπο δεν θα ήταν δυνατή η άμεση ενημέρωση του πελάτη όπως πριν που πληροφορήθηκε κατευθείαν ότι το αίτημά του βρίσκεται υπό επεξεργασία.

Άλλη μια σημαντική ιδιότητα της επικοινωνίας μέσω μηνυμάτων που προκύπτει από τα παραπάνω είναι ότι **υπάρχει decoupling μεταξύ του παραλήπτη και του αποστολέα**. Με αυτό εννοείται ότι ο αποστολέας δεν καλεί άμεσα έναν παραλήπτη ο οποίος βρίσκεται σε μια συγκεκριμένη διεύθυνση όπως ισχύει με τα Webservices. Αντιθέτως στέλνει το μήνυμα προς μία ενδιάμεση δομή αποθήκευσης (Queue ή Topic) απ' όπου και καταναλώνεται από οποιονδήποτε παραλήπτη γνωρίζει την διεύθυνση αυτής της δομής και είναι σε θέση να αναλάβει την επεξεργασία του. Αυτό έχει ως αποτέλεσμα τις παρακάτω ιδιότητες:

1. **Ο παραλήπτης μπορεί να αλλάζει διεύθυνση χωρίς να χρειάζονται αλλαγές στον αποστολέα αλλά και το ανάποδο.**
2. **Μπορούν να υπάρχουν πολλαπλοί αποστολείς ή/και παραλήπτες για τον ίδιο τύπο μηνυμάτων** οι οποίοι δύναται να εκτελούνται σε πολλαπλά μηχανήματα διαμοιράζοντας έτσι τον φόρτο επεξεργασίας.
3. **'Ενα μήνυμα μπορεί να επεξεργάζεται από όλους τους διαθέσιμους παραλήπτες ή να καταναλωθεί μόνο από έναν από αυτούς** (ανάλογα με το πώς έχουμε στήσει το JMS σύστημα).
4. **Μπορούν να συνυπάρχουν παραλήπτες οι οποίοι επεξεργάζονται με διαφορετικό τρόπο μηνύματα του ίδιου τύπου.**
5. **Η επικοινωνία δεν επηρεάζεται εάν το σύστημα που λαμβάνει το μήνυμα δεν λειτουργεί την στιγμή της αποστολής.** Μπορεί να είναι προγραμματισμένο να ενεργοποιείται μόνο σε συγκεκριμένες περιόδους, να αντιμετωπίζει κάποιο πρόβλημα ή να βρίσκεται σε διαδικασία επανεκκίνησης. Όπως και να έχει, θα μπορεί να πραγματοποιήσει την επεξεργασία των μηνυμάτων όποτε αυτό θα τεθεί σε πλήρη λειτουργία και θα είναι έτοιμο να το κάνει.

Από τα παραπάνω παίρνουμε μια ιδέα για την τεράστια ευελιξία που μας προσφέρει ένα πρωτόκολλο επικοινωνίας μέσω μηνυμάτων για τον σχεδιασμό επιχειρησιακών συστημάτων.

## ***Βασικές Έννοιες - Ορολογία***

Αυτή η υπηρεσία προσφέρεται από τον Application Server και το πρότυπο ορίζει 4 κύρια συστατικά/έννοιες:

**JMS Provider:** Είναι ο Application Server και συγκεκριμένα οι JMS υπηρεσίες του κατά το JMS πρότυπο. Διαμέσου αυτών είναι δυνατή η **χρήση, παρακολούθηση και διαχείριση** του συστήματος μηνυμάτων και των συστατικών του. Εάν η εφαρμογή μας φιλοξενείται σε έναν Servlet Container τότε ο JMS Provider δεν συμπεριλαμβάνεται στον ίδιο τον server αλλά θα πρέπει να εγκατασταθεί και να ρυθμιστεί ξεχωριστά και να εκκινείτε παράλληλα με τον Servlet Container έτσι ώστε να μπορεί να υποστηρίξει τις εφαρμογές που είναι ανεβασμένες σε αυτόν και που κάνουν χρήση του JMS. Ένας τέτοιος JMS Provider είναι ο **ActiveMQ**.

**JMS Clients:** Ως clients αναφέρονται οι εφαρμογές που συμμετέχουν στην επικοινωνία (**είτε αποστολείς είτε παραλήπτες**).

**Messages:** Είναι τα **μηνύματα** τα οποία ανταλλάσσονται μεταξύ των JMS Clients.

**Administered Objects:** Είναι τα συστατικά τα οποία κυρίως διαχειρίζεται ο JMS Provider, δηλαδή τα **JMS Connection Factories** που χρησιμοποιούνται για την δημιουργία συνδέσεων για την αποστολή μηνυμάτων και τα **Destinations (Queues ή Topics)**.

Εδώ αξίζει να σημειωθεί ότι τα Queues δεν ικανοποιούν πλήρως την περιγραφή της αντίστοιχης δομής δεδομένων αφού τα μηνύματα τα οποία αποθηκεύουν ενδέχεται να επεξεργαστούν με διαφορετική σειρά από αυτήν με την οποία αποθηκεύτηκαν ή και να επεξεργαστούν παράλληλα. Μόνο στην περίπτωση που υπάρχει μόνο ένας αποστολέας και μόνο ένας παραλήπτης η δομή αυτή συμπεριφέρεται σαν ένα τυπικό queue και μπορούμε να εγγυηθούμε την σειρά λήψης και επεξεργασίας των μηνυμάτων. Παρόλα αυτά αυτές οι δομές ορίστηκαν έτσι από την ορολογία του προτύπου.

## Ασύγχρονη Εκτέλεση

Το JMS μας εξυπηρετεί σε δύο πράγματα, την **ασύγχρονη εκτέλεση**, και την **απομακρυσμένη επικοινωνία**. Μας εξυπηρετεί όταν έχουμε ανάγκη και για τις δύο λειτουργικότητες αλλά είναι σημαντικό να το βλέπουμε σαν λύση και στις περιπτώσεις όπου χρειαζόμαστε απλώς να εκτελέσουμε κάτι ασύγχρονα μέσα στην ίδια εφαρμογή.

Στην περίπτωση της ασύγχρονης εκτέλεσης χωρίς απομακρυσμένη επικοινωνία ο παραλήπτης και ο αποστολέας των μηνυμάτων είναι συστατικά (αντικείμενα) που βρίσκονται στην ίδια εφαρμογή και η ρουτίνα αποστολής του μηνύματος επιλέγουμε να εκτελείται ανεξάρτητα από την ρουτίνα λήψης του μηνύματος π.χ. σε διαφορετικό νήμα εκτέλεσης. Σε αυτήν την περίπτωση δεν κάνουμε χρήση της απομακρυσμένης επικοινωνίας αλλά επιτυγχάνουμε:

1. Την **συνέχιση της εκτέλεσης του «νήματος αποστολέα»** χωρίς να μπλοκάρει μέχρι να ολοκληρωθεί η επεξεργασία του μηνύματος. Μπορεί έτσι να ενημερώσει τον πελάτη ότι το αίτημά του βρίσκεται υπό επεξεργασία.
2. Εξαιτίας της ενδιάμεσης καταχώρησης όλων των μηνυμάτων σε ειδικές δομές (Destinations), δηλαδή Queues ή Topics, αυτά μπορούν να καταναλωθούν με τρόπο που να **βελτιστοποιεί τον καταμερισμό του φόρτου επεξεργασίας** π.χ. από ένα thread pool, επιτυγχάνοντας παράλληλα **scalability** επιτρέποντας στο σύστημα να επεξεργαστεί τα μηνύματα στον ρυθμό που μπορεί και όχι όλα ταυτόχρονα.
3. Διευκολύνεται (μετά από ενδεχόμενη μελλοντική αλλαγή του κώδικα) η εκχώρηση της επεξεργασίας των μηνυμάτων σε άλλη εφαρμογή, στον ίδιο ή σε διαφορετικό server δίνοντας έτσι την δυνατότητα της μελλοντικής εκμετάλλευσης της επιλογής για απομακρυσμένη επικοινωνία με στόχο την ανάθεση της επεξεργασίας σε άλλο σύστημα.
4. **Μπορεί να χρησιμοποιηθεί για την επικοινωνία εφαρμογών που βρίσκονται στον ίδιο Application Server.** Δεν μπορούμε ουσιαστικά να θεωρήσουμε κάτι τέτοιο ως απομακρυσμένη επικοινωνία αφού οι εφαρμογές που βρίσκονται στον ίδιο Application Server εκτελούνται στο ίδιο JRE και υπάρχουν και άλλοι τρόποι για επικοινωνία μεταξύ τους π.χ. μέσω JNDI. Παρόλα αυτά έχουμε όλα τα πλεονεκτήματα της ασύγχρονης επικοινωνίας και θα μπορούσαμε π.χ. να φτιάξουμε μια εφαρμογή που να συγκεντρώνει την κοινή λογική (κοινές ρουτίνες) επεξεργασίας μηνυμάτων άλλων εφαρμογών που βρίσκονται ανεβασμένες στον ίδιο Application Server.
5. Διευκολύνεται η **εφαρμογή του Transaction Management** που ενσωματώνεται στα EJB πρότυπα (τα οποία ενσωματώνουν τα JMS πρότυπα μέσω των **Message Driven Beans, MDB**). Εάν είναι ενεργοποιημένη αυτή η λειτουργικότητα, σε περίπτωση που αποτύχει η επεξεργασία ενός μηνύματος αυτό ξαναεπιστρέφει στο Queue ώστε να ξαναγίνει προσπάθεια επεξεργασίας του αργότερα με ταυτόχρονο **rollback** όλων των ενεργειών

που έγιναν κατά την προσπάθεια επεξεργασίας του. Αυτή η συμπεριφορά είναι κάτι που μας εξυπηρετεί συχνά χωρίς να σημαίνει ότι δεν μπορεί να απενεργοποιηθεί εάν δεν την χρειαζόμαστε.

## Απομακρυσμένη Επικοινωνία

Εάν δεν χρειαζόμαστε ασύγχρονη εκτέλεση αλλά μόνο απομακρυσμένη επικοινωνία, τότε δεν υπάρχει λόγος να χρησιμοποιήσουμε JMS, η απομακρυσμένη επικοινωνία μπορεί να υλοποιηθεί με Java RMI, Remote EJBs ή Webservices. Η Java RMI και τα Remote EJBs χρησιμοποιούνται για επικοινωνία μεταξύ Java προγραμμάτων, ενώ τα Webservices εξυπηρετούν και επικοινωνία μεταξύ προγραμμάτων γραμμένων σε διαφορετικές γλώσσες.

Η απομακρυσμένη επικοινωνία μπορεί να υπάρξει σε δύο μορφές:

1. **Σε απομακρυσμένο Java Application Server που εκτελείται σε διαφορετικό process στο ίδιο ή σε διαφορετικό μηχάνημα.**
2. **Σε απομακρυσμένο Application Server που είναι βασισμένος σε άλλη γλώσσα (π.χ. C#).**

Από την στιγμή που το JMS είναι απλά μία διεπαφή αποστολής και λήψης μηνυμάτων και δεν ορίζει κάποιο πρωτόκολλο μεταφοράς δεδομένων μέσω δικτύου, υπάρχουν υλοποιήσεις του (π.χ. ActiveMQ) που με σωστές ρυθμίσεις μπορούν να υποστηρίξουν επικοινωνία με ετερογενή συστήματα. Συνήθως καταλήγουμε όμως σε περιορισμούς της μορφής των μηνυμάτων που μπορούμε να ανταλλάξουμε και η υλοποίηση τέτοιου εγχειρήματος ενδέχεται να παρουσιάσει δυσκολίες που θα οφείλονται στο εκάστοτε σύστημα.

Προφανώς όλα αυτά που επιτυγχάνονται στην περίπτωση της τοπικής ασύγχρονης εκτέλεσης επιτυγχάνονται και στις περιπτώσεις της απομακρυσμένης επικοινωνίας.

Αξίζει να σημειωθεί ότι πρακτικά το JMS χρησιμοποιείται πολύ συχνά με τρόπο ώστε η ρουτίνα λήψης του μηνύματος να εκτελείται τοπικά και να πραγματοποιεί μία απομακρυσμένη κλήση μέσω Webservice. Έτσι επιτυγχάνεται έμμεση και όχι άμεση απομακρυσμένη επικοινωνία μέσω JMS επειδή ενδεχομένως το απομακρυσμένο σύστημα να υποστηρίζει μόνο Webservices, έτσι εκμεταλλευόμαστε τα στοιχεία τις ασύγχρονης εκτέλεσης και με τέτοια συστήματα.

## Transactions

### **Θεωρητική Εισαγωγή**

#### **Γενικά**

Οι συναλλαγές (**Transactions**) ορίζονται πάνω σε κομμάτια του εκτελέσιμου κώδικα, μία συναλλαγή μπορεί να αποτελείται από μερικές συνεχόμενες εντολές μίας μεθόδου (**Bean Managed Transactions, BMT**) ή, όπως συμβαίνει συχνότερα, να εκτείνεται σε όλη την μέθοδο (**Container Managed Transactions, CMT**).

Είναι μία έννοια που υπονοεί την εξασφάλιση τεσσάρων εγγυήσεων, **Atomicity, Consistency, Isolation, Durability** που αναφέρονται και ως **“ACID”**.

**Οι ουσιαστικές εγγυήσεις οι οποίες μας δίνουν μια ιδέα για την χρησιμότητα των συναλλαγών είναι το Atomicity και το Isolation.**

Το **Atomicity** αναφέρεται στην ιδιότητα της **επαναφοράς τροποποιημένων δεδομένων στην αρχική τους κατάσταση (rollback)** στην περίπτωση της εμφάνισης κάποιου σφάλματος. Το **Isolation** είναι ιδιότητα η οποία προστατεύει τα δεδομένα τα οποία τροποποιούνται από μια διεργασία από την ταυτόχρονη τροποποίησή τους από μια άλλη διεργασία η οποία εκτελείται **παράλληλα** με την πρώτη και ενεργεί στα ίδια δεδομένα.

#### **Atomicity**

Το Atomicity αναφέρεται συχνά και ως **“All or Nothing”** property. Διότι ορίζει ότι στα πλαίσια ενός transaction θα πρέπει να ολοκληρωθούν όλες οι ενέργειες επιτυχώς ή σε περίπτωση σφάλματος σε τουλάχιστον μία από αυτές να ακυρωθούν όλες.

**Ένα παράδειγμα του Atomicity** Θα μπορούσε να είναι η μεταφορά χρημάτων από έναν τραπεζικό λογαριασμό σε έναν άλλον. Εάν παρουσιαστεί σφάλμα ανάμεσα στην χρέωση του αποστολέα και την πίστωση του παραλήπτη, και η διαδικασία αυτή δεν έχει οριστεί ως transaction, τότε θα γίνει μόνο αφαίρεση από τον λογαριασμό του αποστολέα χωρίς την πίστωση του παραλήπτη. Εάν από την άλλη η διαδικασία αυτή ανήκει σε ένα transaction τότε σε περίπτωση σφάλματος η χρέωση του αποστολέα θα ακυρωθεί και έτσι θα μπορέσουμε τουλάχιστον να διασφαλίσουμε ότι δεν θα χαθούν τα λεφτά του.

#### **Local & Global Transactions**

Οι συναλλαγές μπορούν να εκτελούνται πάνω σε μία βάση δεδομένων ή ένα JMS Resource (**Resource Local Transactions**) ή να συμπεριλαμβάνουν πολλά resources και των δύο τύπων μέσα στην ίδια συναλλαγή (**JTA Global Transactions**).

Ένα transaction μπορεί είτε να επαναφέρει μια εγγραφή στην αρχική της κατάσταση μετά από αποτυχημένη προσπάθεια τροποποίησης αυτής ή μίας άλλης ακόμα και από άλλη ΒΔ, ή να ξαναεισάγει ένα μήνυμα σε μία ουρά μηνυμάτων (Queue) σε περίπτωση που η πρώτη προσπάθεια επεξεργασίας του απέτυχε (στην περίπτωση συναλλαγής που ξεκινάει από τον παραλήπτη) ώστε να ξαναγίνει προσπάθεια επεξεργασίας του στο μέλλον. Στην περίπτωση της συναλλαγής που ξεκινάει από το αποστολέα μπορεί να ακυρωθεί η αποστολή του μηνύματος ακόμα και εάν δεν υπάρξει πρόβλημα από το JMS αλλά από κάποιο άλλο resource που συμμετέχει στο transaction

όπως ένα connection προς την ΒΔ της εφαρμογής. Έτσι π.χ. εάν αποτύχει η καταγραφή στην ΒΔ της αποστολής ενός μηνύματος μέσω JMS σε ένα απομακρυσμένο σύστημα, δεν θα πραγματοποιηθεί ούτε και η αποστολή του μηνύματος ακόμα και εάν γίνεται νωρίτερα μέσα στον κώδικα.

**Γενικά δεν υπάρχει περιορισμός για τους πόρους και τα συστήματα τα οποία μπορούν να συμμετέχουν σε συναλλαγές, αρκεί αυτά τα επιμέρους συστήματα να τις υποστηρίζουν.**  
**Πρακτικά πάντως ασχολούμαστε μόνο με transactions σε βάσεις δεδομένων και αποστολές μηνυμάτων (JMS).**

Γενικά, για να υποστηρίζει ένα σύστημα τις local ή global transactions θα πρέπει αυτές να υλοποιούνται μέσω του API του. Π.χ. κάποιες από τις μεθόδους του JDBC που υλοποιούν την υποστήριξη των συναλλαγών είναι οι **java.sql.Connection.commit()** και **java.sql.Connection.rollback()**, όπου γενικά τα αντικείμενα τύπου java.sql.Connection χρησιμοποιούνται για να σταλθούν εντολές SQL προς την βάση δεδομένων. Όσον αφορά το JMS αντίστοιχες μέθοδοι είναι οι **javax.jms.Session.commit()** και **javax.jms.Session.rollback()**. Αντικείμενα τύπου javax.jms.Session χρησιμοποιούνται για να σταλθούν μηνύματα προς ουρές μηνυμάτων καθώς και για την λήψη τους.

**Στην περίπτωση των global transactions θα πρέπει οι ιδιότητες του atomicity και του isolation να εξασφαλιστούν για τροποποιήσεις δεδομένων στις οποίες συμμετέχουν πολλαπλές απομακρυσμένες βάσεις δεδομένων και ενδεχομένως και άλλα συστήματα.** Χρειαζόμαστε πρωτόκολλα υψηλότερου επιπέδου από αυτά που ενσωματώνονται στο API των επιμέρους συστημάτων όπως προηγουμένως (όπως π.χ. είναι το **Java Transaction API - JTA**). Ένα τέτοιο πρωτόκολλο θα επικοινωνεί με το κάθε επιμέρους σύστημα ξεχωριστά μέσω των μεθόδων του API αυτού του συστήματος που υποστηρίζουν τα global transactions π.χ. για τα local transactions στο JDBC μπορεί να χρησιμοποιηθεί το DataSource αλλά εάν χρησιμοποιήσουμε το JDBC στα πλαίσια ενός global JTA transaction θα πρέπει να χρησιμοποιήσουμε XADatasource.

Το **JTA (Java Transaction API)** υποστηρίζεται και από τα EJBs. Είναι σημαντικό να τονιστεί ότι **το JTA παρέχει μόνο Atomicity αλλά δεν ορίζει εγγυήσεις για το Isolation**. Οι ανάγκες για isolation καλύπτονται συνήθως από τα ίδια τα resources που συμμετέχουν σε ένα global transaction από την στιγμή που ενεργοποιείται το lock του καθενός από αυτά κατά το ξεκίνημα του global transaction και απελευθερώνεται μόνο με το πέρας του.

## Global Transactions – Two Phase Commit (2PC)

Τα global transactions υλοποιούνται με την τεχνική του **2PC (Two Phase Commit)** η οποία εκτελεί το transaction σε δύο φάσεις.

Στην **πρώτη φάση** εκτελεί όλων τον κώδικα μέσα στο transaction και τις ενέργειες που έχουν οριστεί σε αυτόν για το κάθε resource που συμμετέχει στο transaction χωρίς όμως να κάνει commit σε κανένα resource. Έτσι οι αλλαγές (π.χ. insert statements) είναι ορατές από τον μετέπειτα κώδικα στο transaction (π.χ. select statements) αλλά δεν έχει ακόμα καθορισθεί εάν θα είναι μόνιμες και δεν είναι ακόμα ορατές από κώδικα σε άλλα transactions που εκτελούνται παράλληλα μέσα στην εφαρμογή ή έξω από αυτήν.

Στην **δεύτερη φάση** γίνεται ερώτημα προς το κάθε επιμέρους resource για την δυνατότητα πραγματοποίησης επιτυχούς commit των αλλαγών. Εάν όλα απαντήσουν θετικά τότε γίνεται το commit σε καθένα από αυτά. Εάν έστω και ένα παρουσιάσει σφάλμα θα γίνει rollback (ακύρωση των αλλαγών) σε όλα.

Έτσι, στην περίπτωση σφάλματος μέσα σε ένα global transaction μέσα στον κώδικα της Java ενδέχεται, μέσα στην αναφορά του σφάλματος να παρατηρήσουμε αναφορές προς resources πάνω στα οποία εκτελούνται ενέργειες μετά από το σημείο στο οποίο παρουσιάστηκε το σφάλμα. Αυτό συμβαίνει επειδή το commit των αλλαγών και ως εκ τούτου και του σημείου στο οποίο παρουσιάστηκε το σφάλμα, γίνεται πάντα στο τέλος του transaction.

## Σημεία Προσοχής

Σε αυτό το σημείο σημειώνεται ότι π.χ. ένα απλό DataSource που χρησιμοποιούμε για απλή επικοινωνία με την ΒΔ μέσω JDBC δεν υποστηρίζει global transactions, για κάτι τέτοιο θα χρειαστούμε ένα data source που υλοποιεί το XADataSource interface. Αυτό αναφέρθηκε για να τονιστεί ότι ακόμα και αν ο κώδικάς μας εκτελείται μέσα στα πλαίσια ενός global transaction θα πρέπει να σιγουρευτούμε ότι και τα resources που χρησιμοποιούμε υποστηρίζουν κάτι τέτοιο, διαφορετικά ένα ενδεχόμενο rollback δεν θα έχει καμία επίδραση σε αυτά. Επίσης θα πρέπει να σιγουρευτούμε, αναφορικά με τα EJB, εάν κάποιο resource υποστηρίζει τα Container Managed Transactions ή μόνο τα Bean Managed Transactions.

Καλό είναι σε αυτό το σημείο να σημειωθεί επίσης ότι **τα local transactions μπορούν να εξασφαλισθούν σε επίπεδο βάσης δεδομένων** ορίζοντας procedures στην SQL που εμπεριέχουν εσωτερικά εντολές για transaction management και αποθηκεύοντάς τες στην βάση. Αυτές οι procedures είναι γραμμένες σε μία επέκταση της SQL που επιτρέπει διαδικαστικό τρόπο γραφής κώδικα και όχι καθαρά δηλωτικό (PLSQL). Μέσα τους ορίζονται και οι συναλλαγές. Ο κώδικας της Java μπορεί να καλέσει αυτές τις procedures και να λάβει τα αποτελέσματά τους μέσω του JDBC, όμως δεν θα έχει καμία γνώση των συναλλαγών που ορίζονται μέσα σε αυτές. Μπορεί να έχει κάποια ένδειξη στην επιστρεφόμενη τιμή ότι το transaction απέτυχε και έγινε rollback όμως στο rollback αυτό δεν πραγματοποιήθηκε από το JDBC και δεν θα γίνει αυτόματη ενημέρωση για το συμβάν σε πρωτόκολλά διαχείρισης global transactions (JTA) ώστε να γίνει rollback και στο αντίστοιχο transaction που περιείχε και άλλες βάσεις δεδομένων στο οποίο ενδέχεται να συμμετείχε και η τοπική αυτή κλήση.

Αυτή η παρένθεση δόθηκε για να τονισθεί ότι **τα transactions μπορούν να υλοποιηθούν με πολλούς τρόπους**, όμως είναι σημαντικό να κατανοήσουμε τις ιδιαιτερότητες του καθενός και να αναλογιστούμε το αντίκτυπο το οποίο θα έχει ο καθένας τους στον κώδικά μας όσον αφορά την συμπεριφορά του αλλά και τις ενδεχόμενες μελλοντικές αλλαγές και επεκτάσεις που ενδέχεται να χρειαστεί.

Παρακάτω εξηγούνται οι όροι Atomicity, Consistency, Isolation και Durability σε μεγαλύτερη λεπτομέρεια και δίνονται παραδείγματα με αναφορά σε local transactions που γίνονται σε βάση δεδομένων, επειδή είναι τα πιο απλά αλλά και τα πιο πρακτικά για την κατανόηση των εννοιών.

## Consistency

**Αναφέρεται στην ιδιότητα μιας συναλλαγής να φέρνει τα δεδομένα από μία έγκυρη κατάσταση σε μία άλλη έγκυρη κατάσταση.** Αυτό βέβαια προϋποθέτει την αρχική έγκυρη κατάσταση των δεδομένων. Επίσης προϋποθέτει ότι με κάποιον τρόπο θα είναι καθορισμένοι οι κανόνες εγκυρότητας των δεδομένων και συν τοις άλλοις το transaction service θα επαναφέρει αυτόματα τα δεδομένα στην αρχική τους κατάσταση σε περίπτωση παραβίασης αυτών των κανόνων ή σε περίπτωση που δεν το κάνει αυτόματα, ο προγραμματιστής να έχει διασφαλίσει με τους κατάλληλους ελέγχους να εκτελεί την εντολή επαναφοράς του transaction (rollback).

Στην περίπτωση των βάσεων δεδομένων αυτοί οι κανόνες εκφράζονται ως “CONSTRAINTS”, συχνά κατά την δημιουργία των πινάκων. Άλλοι κανόνες εγκυρότητας μπορεί να εκφράζονται απλώς με συνθήκες ελέγχου μέσα στον κώδικα (δομές if/else). Το σημαντικό είναι κατά την παραβίασή τους να ενημερώνεται ο transaction manager ώστε να γίνει η ακύρωση της συναλλαγής και η επαναφορά των δεδομένων είτε αυτόματα από τα resources που συμμετέχουν στο transaction είτε από τον προγραμματιστή της εφαρμογής μέσω κατάλληλων συνθηκών μέσα στον κώδικα της.

Προφανώς οι κανόνες εγκυρότητας των δεδομένων είναι χαρακτηριστικό της κάθε εφαρμογής αφού αποτελούν κομμάτι του business logic για αυτό και η εγγύηση του data consistency είναι κάτι το οποίο θα πρέπει να εξασφαλιστεί από τον ίδιο τον προγραμματιστή της εφαρμογής και όχι τόσο από το transaction service.

## Isolation

Το Isolation έχει να κάνει με την επίλυση προβλημάτων ταυτόχρονης τροποποίησης δεδομένων στην περίπτωση της παράλληλης εκτέλεσης (Concurrency). Φαινόμενα τα οποία μπορούν να παρουσιαστούν κατά την παράλληλη εκτέλεση διεργασιών πάνω στα ίδια δεδομένα αναφέρονται παρακάτω. Το Isolation είναι η δεύτερη ουσιαστική εγγύηση που μας προσφέρουν τα Transactions (μετά το Atomicity) για αυτό και είναι καλό να ξεκαθαριστεί ακριβώς το τι συνεπάγεται από αυτό.

## DBMS READ PHENOMENA

Παρακάτω παρουσιάζονται τα φαινόμενα τα οποία μπορούν να εμφανιστούν εξαιτίας της παράλληλης εκτέλεσης δύο διεργασιών πάνω στα ίδια δεδομένα. Αναφέρονται ως φαινόμενα και όχι ως προβλήματα επειδή μόνο το φαινόμενο των dirty reads θα μπορούσε να χαρακτηριστεί πάντα ως πρόβλημα ενώ τα non repeatable reads και τα phantom reads είναι φαινόμενα τα οποία ενίστε είναι επιθυμητά.

1. **Dirty Read:** Το φαινόμενο του Dirty Read παρατηρείται όταν συμβαίνει μία ανάγνωση δεδομένων από μία συναλλαγή Β τα οποία η συναλλαγή Α έχει τροποποιήσει αλλά δεν έχει πραγματοποιήσει ακόμα commit. Σε αυτήν την περίπτωση υπάρχουν οι εξής κίνδυνοι:

1. Ενδέχεται η συναλλαγή Α να έχει φέρει τα δεδομένα σε μία ενδιάμεση μη έγκυρη κατάσταση και όχι ακόμα στην τελική και έγκυρη κατάσταση στην οποία θα έρθουν όταν ολοκληρωθεί εντελώς.
2. Ενδέχεται η συναλλαγή Α τελικά να αποτύχει και να γίνει rollback, σε αυτήν την περίπτωση η συναλλαγή Β θα είναι σαν να έχει διαβάσει ανύπαρκτα δεδομένα.

Το φαινόμενο του Dirty Read είναι δυνατό μόνο στην περίπτωση που το isolation level είναι **READ UNCOMMITTED** ή το sql ερώτημα έχει hint **“NOLOCK”**.

2. **Non-Repeatable Read:** Το φαινόμενο αυτό εμφανίζεται όταν η συναλλαγή Α πραγματοποιεί 2 αναγνώσεις δεδομένων που διαβάζουν το ίδιο δεδομένο αλλά ενδιάμεσα στις 2 αυτές αναγνώσεις η συναλλαγή Β αλλάζει το δεδομένο και πραγματοποιεί commit. Το φαινόμενο αυτό εμφανίζεται όταν οι 2 αυτές αναγνώσεις επιστρέφουν διαφορετικά αποτελέσματα.

Το φαινόμενο του Non-Repeatable Read εμφανίζεται στα isolation levels: **READ UNCOMMITTED** και **READ COMMITTED** (default transaction level στα περισσότερα ΣΔΒΔ). Για

να αποφευχθούν Non-Repeatable Reads το isolation level θα πρέπει να τεθεί σε **REPEATABLE READ**.

Πολλές φορές τα Non-Repeatable Reads είναι κάτι που μας εξυπηρετεί και η επιλογή του transaction level REPEATABLE READ εξαρτάται πάντα από τις ανάγκες μας και το πρόβλημα που αντιμετωπίζουμε. Γενικά η επιλογή μας εξαρτάται από το εάν θέλουμε πρόσφατα δεδομένα (current data) ή έγκυρα δεδομένα (consistent data).

3. **Phantom Read:** Το φαινόμενο αυτό παρουσιάζεται όταν μία συναλλαγή Α πραγματοποιεί ένα query 2 φορές, το οποίο επιστρέφει ένα **σύνολο από εγγραφές** την 1<sup>η</sup> φορά, που είναι διαφορετικό από αυτό που επιστρέφει την 2<sup>η</sup> φορά επειδή στο ενδιάμεσο κάποια συναλλαγή Β αφαιρεσε ή πρόσθεσε εγγραφές στους πίνακες του query ή ενημέρωσε τις εγγραφές των πινάκων με άλλες τιμές (στην τελευταία περίπτωση το φαινόμενο παρατηρείται μόνο όταν το isolation level είναι READ COMMITTED ή χαμηλότερο).

Η διαφορά του φαινομένου αυτού με τα Dirty Read και Non-Repeatable Read είναι ότι τα δύο προηγούμενα αναφερόντουσαν σε μεμονωμένες εγγραφές ενώ αυτό αναφέρεται σε σύνολο εγγραφών.

Με λίγα λόγια εάν ένα σύνολο εγγραφών αλλάζει τιμές εξαιτίας update εντολών αλλά το query συνεχίζει να επιστρέφει τις ίδιες εγγραφές τότε δεν πραγματοποιείται το φαινόμενο Phantom Read αλλά ενδέχεται να υπάρχουν φαινόμενα Dirty Read και Non-Repeatable Read ανάλογα με το isolation level που έχουμε θέσει.

Από την άλλη εάν οι update εντολές αλλάζουν τις εγγραφές με τέτοιο τρόπο που να αλλάζει το ποιες εγγραφές ανήκουν στο επιστρεφόμενο σύνολο τότε έχει πραγματοποιηθεί το φαινόμενο Phantom Read.

Εξαιτίας αυτού το isolation level READ REPEATABLE δεν είναι αρκετό για να εξασφαλίσει ότι δεν θα υπάρξει Phantom Read αφού ενώ οι ήδη υπάρχουσες εγγραφές δεν θα αλλάξουν και ούτε θα διαγραφούν κατά τη διάρκεια ενός transaction ενδέχεται να προστεθούν καινούριες εγγραφές οι οποίες να ικανοποιούν τα κριτήρια του query.

Το μόνο isolation level που εξασφαλίζει ότι δεν θα υπάρξουν Phantom Read φαινόμενα είναι το **SERIALIZABLE** (ή αλλιώς **SNAPSHOT** εάν χρησιμοποιούμε ΒΔ της Microsoft). Προφανώς, όπως και με το Non-Repeatable Reads πρέπει να αποφασίσουμε εάν θέλουμε να έχουμε την πιο πρόσφατη εικόνα των δεδομένων ή μια πιο συνεπής. Θα μπορούσαμε να πούμε ότι το isolation level SERIALIZABLE είναι μια επέκταση του REPEATABLE READ σε σύνολα.

## Durability

Έχει την έννοια της **διατήρησης του αποτελέσματος της συναλλαγής** και αναφέρεται περισσότερο στις συναλλαγές που γίνονται σε βάσεις δεδομένων. Τα δεδομένα τα οποία επεξεργάστηκαν από την συναλλαγή δηλαδή θα πρέπει να διατηρηθούν στον σκληρό δίσκο μετά το πέρας της συναλλαγής. Αυτό είναι πιο πολύ μια αρχή την οποία θα πρέπει να εξασφαλίσουν οι κατασκευαστές των συστημάτων βάσεων δεδομένων. Θα πρέπει δηλαδή να παρέχεται η εγγύηση ότι ακόμα και σε περιπτώσεις κατάρρευσης του συστήματος τα δεδομένα θα αποθηκευτούν σωστά, ή τουλάχιστον θα γίνει ό,τι είναι δυνατόν από το σύστημα για να εξασφαλιστεί η αποθήκευσή τους σε όσον το δυνατόν περισσότερα ενδεχόμενα σφαλμάτων. Οι

τεχνικές που το υλοποιούν συνήθως βασίζονται σε αρχεία καταγραφής εντολών που διατηρούνται σε μέσα μόνιμης αποθήκευσης.

## Dependency Injection

### **Θεωρητικός Ορισμός**

To Dependency Injection (DI) είναι η ανάθεση της αρχικοποίησης των εξαρτήσεων μίας κλάσης εξωτερικά από αυτήν.

Αυτό σημαίνει ότι εάν η κλάση μας εσωτερικά χρειάζεται αντικείμενα άλλων κλάσεων, αντί να τα δημιουργεί ή ίδια αναθέτει την δημιουργία τους σε εξωτερικά συστήματα (IoC Container) ή κλάσεις οι οποίες θα την χρησιμοποιήσουν και λαμβάνει αυτά τα αντικείμενα μέσω constructor, setter methods ή απλώς ορίζοντας fields κατάλληλου τύπου τα οποία αρχικοποιούνται με την βοήθεια annotations και ενός **IoC Container**.

Εσωτερικά η κλάση αυτή θα πρέπει να μην αναφέρει τους τύπους των κλάσεων τις οποίες χρησιμοποιεί αλλά μόνο interfaces αυτών. Έτσι είναι απαραίτητο για κάθε κλάση να δημιουργείτε και ένα interface.

### **Πρακτικό Παράδειγμα**

Ακολουθώντας αυτόν τον τρόπο γραφής κώδικα επωφελούμαστε από πολύ σημαντικά πλεονεκτήματα τα οποία θα γίνουν εμφανή από το παρακάτω υποθετικό σενάριο.

Έστω ότι φτιάχνουμε μία enterprise εφαρμογή η οποία εκτός από το να εξυπηρετεί HTTP requests θα φτιάχνει επίσης και μία κονσόλα σε παραθυρικό περιβάλλον η οποία θα εκτελείται στο μηχάνημα στο οποίο τρέχει ο server. Ο ρόλος αυτής της κονσόλας θα είναι η άμεση πρόσβαση του διαχειριστή της εφαρμογής στην κατάσταση της εφαρμογής καθώς επίσης θα του παρέχει και διάφορες άλλες διευκολύνσεις για την διαχείρισή της όπως π.χ. ένα **chat box** το οποίο θα το χρησιμοποιεί για να επικοινωνεί με τους διαχειριστές των υπόλοιπων τμημάτων της εταιρίας που υποστηρίζουν την εφαρμογή (π.χ. CRM, Database και Network Administrators).

Έστω επίσης ότι χρειάζεται στο ίδιο chat box στο οποίο ο χρήστης εισάγει τα μηνύματα που θέλει να στείλει και στο οποίο βλέπει τις απαντήσεις από τους τεχνικούς, να εμφανίζονται και μηνύματα για την ενημέρωση της πορείας της εξέλιξης κάποιων διαδικασιών όπως π.χ. background processes καθώς και προειδοποιητικών μηνυμάτων όπως π.χ. έλλειψη αποθηκευτικού χώρου στον σκληρό δίσκο κ.τ.λ.. Αυτό μπορεί να υλοποιηθεί με πολλούς διαφορετικούς τρόπους, π.χ. εάν αποφασίσουμε να χρησιμοποιήσουμε την **SWING** βιβλιοθήκη θα το φτιάξουμε με ένα αντικείμενο τύπου **javax.swing.JTextArea** και στα σημεία του κώδικα στα οποία θα γίνεται εισαγωγή κειμένου στο chat box θα χρησιμοποιείται η μέθοδος **JTextArea.append(String)** ενώ εάν αποφασίσουμε να χρησιμοποιήσουμε την βιβλιοθήκη **JavaFX** στα αντίστοιχα σημεία θα χρησιμοποιείται αντικείμενο τύπου **javafx.scene.control.TextArea** και η αντίστοιχη μέθοδος **TextArea.appendText(String)**.

Είναι φανερό ότι επιλέγοντας οποιαδήποτε από τις δύο μεθόδους και στην συνέχεια απλώς χρησιμοποιώντας κατευθείαν τις μεθόδους append ή appendText στα σημεία του κώδικα στα οποία γράφεται κείμενο στο chat box, εάν αποφασίσουμε στο μέλλον να αλλάξουμε την υλοποίηση που χρησιμοποιούμε θα πρέπει να κάνουμε τις αλλαγές σε πάρα πολλά σημεία του κώδικα π.χ. εκεί όπου γράφονται εισερχόμενα μηνύματα που έρχονται μέσω δικτύου από τους διαχειριστές των άλλων συστημάτων, στο σημείο στο οποίο γίνεται προειδοποίηση για

ανεπάρκεια χώρου στον σκληρό δίσκο, στα σημεία στα οποία γίνεται η ενημέρωση για την πρόοδο background διαδικασιών κ.τ.λ..

Εάν από την άλλη σχεδιάζαμε τον κώδικα μας σύμφωνα με το DI αυτό που θα κάναμε είναι να φτιάξουμε ένα Interface π.χ. το **ChatBox**. Η μόνη μέθοδος που θα μας ενδιέφερε θα ήταν η **ChatBox.append(String)**, η οποία θα εμφάνιζε ένα νέο μήνυμα με τιμή το όρισμά της κάθε φορά που την καλούμε. Προφανώς εάν η εφαρμογή μας είχε περισσότερες απαιτήσεις θα φτιάχναμε ένα interface με περισσότερες μεθόδους το οποίο θα έκανε extend το ChatBox.

Στην συνέχεια θα φτιάχναμε την κλάση **ChatBoxImpl** η οποία θα έκανε implement το interface ChatBox και η οποία θα λειτουργούσε ως wrapper για την τελική υλοποίηση του chat box που θα επιλέγαμε. Η μέθοδος ChatBoxImpl.append(String) εσωτερικά θα έκανε κλήση την μέθοδο **JTextArea.append(String)** ή **TextArea.appendText(String)** ανάλογα με το τι προτιμούμε.

Θα μπορούσαμε ακόμα και να φτιάξουμε πολλές κλάσεις που κάνουν implement το ChatBox π.χ. ChatBoxSWING, ChatBoxFX και ChatBoxConsole όπου η πρώτη θα χρησιμοποιούσε το SWING, η δεύτερη το JavaFX ενώ η τρίτη το default output stream του συστήματος και θα την χρησιμοποιούσαμε σε περίπτωση που θα θέλαμε να κάνουμε κάποιο testing.

**Τονίζουμε σε αυτό το σημείο ότι ο τελικός μας στόχος δεν είναι να χρησιμοποιήσουμε και τις τρεις αυτές κλάσεις στο πρόγραμμά μας αλλά μάλλον να το φτιάξουμε κατά τέτοιον τρόπο έτσι ώστε με ελάχιστες αλλαγές να αλλάζουμε την υλοποίηση του chat box που χρησιμοποιούμε παντού μέσα στην εφαρμογή.**

Θα φανεί στην συνέχεια ότι αυτό έχει τεράστια πρακτική χρησιμότητα για το testing της εφαρμογής, εξαιτίας της ευκολίας με την οποία θα μπορούμε να κάνουμε εναλλαγές μεταξύ της κύριας υλοποίησης που θέλουμε και αυτής που έχουμε δημιουργήσει για να τεστάρουμε την ορθή λειτουργία της εφαρμογής μας (π.χ. μεταξύ της ChatBoxSWING και της ChatBoxConsole).

Στην συνέχεια για να επιτευχθεί η σωστή εφαρμογή του Dependency Injection απαιτούνται δύο πράγματα. Αρχικά όλες οι κλάσεις της εφαρμογής μας οι οποίες χρησιμοποιούν το chat box (**dependent classes**) πρέπει να ορίζουν ως dependency το interface ChatBox. Δηλαδή να συμπεριλαμβάνουν ένα όρισμα αυτού του τύπου στον constructor τους ή να έχουν public μέθοδο setChatBox(ChatBox) ή εναλλακτικά θα μπορούσαν να έχουν ένα field τύπου ChatBox με το κατάλληλο annotation για την αρχικοποίηση του από έναν IoC Container.

Δεύτερον θα πρέπει οι κλάσεις (**assembler classes**) οι οποίες χρησιμοποιούν τις dependent classes να φτιάχνουν αντικείμενα τύπου ChatBox ώστε να τα περάσουν σαν όρισμα στην setChatBox ή στο αντίστοιχο constructor. Εναλλακτικά, που είναι και το πιο σωστό, αλλά ταυτόχρονα και πρακτικά αναγκαίο, θα πρέπει να κάνουμε χρήση κάποιου **IoC Container** και να ορίζουμε τις εξαρτήσεις των κλάσεων κεντρικά σε κάποιο **XML configuration file** ή πάνω στα fields των ίδιων των dependent κλάσεων με **annotations**.

## **IoC Containers (Inversion of Control Containers)**

### **Χρήση IoC Container για την Διαχείριση των Εξαρτήσεων**

Θεωρητικά, η δημιουργία του κατάλληλου constructor ή ενός setter και η ικανοποίηση των εξαρτήσεων των κλάσεων από τις εξωτερικές τους, θα αρκούσε για να θεωρούμε ότι ο κώδικας μας εφαρμόζει τις αρχές του Dependency Injection. Η αλήθεια είναι όμως ότι πρακτικά δεν αρκεί μόνο αυτό.

Αυτό συμβαίνει διότι θα πρέπει να προσφέρεται μία υλοποίηση των **dependencies** (με αναφορά στο προηγούμενο παράδειγμα - ChatBoxSWING / ChatBoxFX / ChatBoxConsole) σε όλες τις **dependent κλάσεις** (κλάσεις που χρησιμοποιούν το chat box για να ενημερώνουν για την πορεία εκτέλεσης των background processes). Εάν αυτό γίνεται από τις **assembler κλάσεις** (τις κλάσεις που χρησιμοποιούνται τις dependent κλάσεις) τότε το πιο πιθανόν είναι να έχουμε τον ίδιο κώδικα (δημιουργίας και ανάθεσης των dependencies) σε πάρα πολλά σημεία. Επειδή στην συντριπτική πλειοψηφία των περιπτώσεων θα φτιάχνουμε αντικείμενα μόνο ενός τύπου (π.χ. μόνο ChatBoxSWING), θα καταλήξουμε με ίδιο κώδικα σε πολλά σημεία. Ο κώδικας αυτός ενδεχομένως να μην είναι μόνο δύο γραμμές, όπως στην περίπτωση του παραδείγματός μας, όπου δημιουργούμε ένα αντικείμενο και μετά το αναθέτουμε στο αντικείμενο το οποίο το έχει ως εξάρτηση, μπορεί σε άλλες περιπτώσεις και αυτό με την σειρά του να έχει εξαρτήσεις. Επίσης μπορεί και αυτός στο μέλλον να αλλάξει (να αλλάξει π.χ. το signature του constructor της κλάσης υλοποίησης) οπότε οι αλλαγές θα πρέπει να γίνουν σε πολλά σημεία.

**Για να εξαλείψουμε αυτόν τον επαναλαμβανόμενο κώδικα, χρησιμοποιούμε IoC Containers οι οποίοι αναλαμβάνουν να διαχειριστούν αυτές τις εξαρτήσεις είτε μέσω annotations ή μέσω XML configuration files.**

Τα IoC Containers αναλαμβάνουν την ικανοποίηση των εξαρτήσεων για την κάθε dependent κλάση αυτόματα (κατά το ξεκίνημα της εφαρμογής ή όταν χρειαστεί για πρώτη φορά) αφού πρώτα ορίσουμε τις εξαρτήσεις της σε ένα κεντρικό σημείο, είτε αυτό είναι ένα XML configuration file είτε αυτό είναι η ίδια η κλάση, όπου σε αυτήν την περίπτωση οι εξαρτήσεις ορίζονται πάνω στα αντίστοιχα fields με annotations.

Έτσι οι εξωτερικές κλάσεις που προηγουμένως είχαν τον ρόλο του assembler των dependent κλάσεων απαλλάσσονται από την υποχρέωση της ικανοποίησης των εξαρτήσεων των κλάσεων τις οποίες χρησιμοποιούν εσωτερικά.

Υπάρχει πληθώρα από IoC Containers, στην περίπτωση της χρήσης μόνο Java EE χωρίς επεκτάσεις (όπως π.χ. το framework Spring ή ο IoC Container Google GUICE κ.τ.λ.) **τον ρόλο του IoC Container τον αναλαμβάνει ο ίδιος ο Application Server**. Όλες οι λεπτομέρειες της υλοποίησης και του τρόπου λειτουργίας του IoC Container ορίζονται στα specifications των EJB και του CDI (Context and Dependency Injection).

**Η χρήση των IoC Containers έχει ως αποτέλεσμα έναν πολύ καθαρότερο κώδικα καθώς οι εξαρτήσεις μίας κλάσης ορίζονται σε ένα σημείο και όχι οπουδήποτε χρησιμοποιείται αυτή η κλάση.**

Αυτό προφανώς δεν σημαίνει ότι, σε κάποιο συγκεκριμένο σημείο, δεν μπορούμε να ικανοποιήσουμε τις εξαρτήσεις ενός αντικειμένου με διαφορετικό τρόπο από ότι σε όλα τα άλλα σημεία. Εάν προκύψει τέτοια ανάγκη, μπορούμε να δημιουργήσουμε τα αντικείμενα που ικανοποιούν τις εξαρτήσεις όπως και προηγουμένως και να κάνουνε override αυτές που ορίζονται γενικώς (π.χ. στο xml configuration file).

**Σε αυτήν την περίπτωση πάντως η τοπική ικανοποίηση των εξαρτήσεων που θα γίνει όποτε χρειαστεί κατά την διάρκεια εκτέλεσης της εφαρμογής, θα πρέπει να γίνεται και πάλι με τρόπο που ορίζει ο IoC Container και όχι π.χ. κάνοντας instantiation του αντικειμένου που θέλουμε με το new operator, και αυτό διότι ο IoC Container θα πρέπει να έχει γνώση του αντικειμένου για να μπορεί να του προσφέρει άλλες υπηρεσίες (π.χ. Lifecycle Management, Transaction Management κ.τ.λ.).**

## Χρήση των Container Services μέσω IoC Container

Το DI μέσω IoC Container στις enterprise εφαρμογές είναι απαραίτητο εάν θέλουμε να χρησιμοποιήσουμε Container Services.

Εξαιτίας του injection των εξαρτήσεων διαμέσου του IoC Container αυτός έχει την δυνατότητα αντί να κάνει injection την υλοποίηση που έχουμε επιλέξει (implementation class), να κάνει inject ένα proxy object (stub). Το stub αυτό θα κάνει implement το interface που κάνει implement και το implementation class που έχουμε ορίσει. Οι μέθοδοι του interface τις οποίες το stub έχει κάνει implement εν τέλει εσωτερικά θα καλούνε τις αντίστοιχες μεθόδους του implementation class και θα επιστρέψουν τις αντίστοιχες τιμές. Μετά όμως την λήψη των ορισμάτων και πριν επιστρέψουν τα return values το stub μπορεί να ορίσει να γίνεται οποιαδήποτε επεξεργασία είναι επιθυμητή από τον προγραμματιστή. Την επεξεργασία αυτήν θα την έχει ορίσει μέσω annotations ή configuration files. Ο IoC Container, με βάση αυτά τα annotations ή/και configuration files θα αναλάβει να την μεταφέρει παραμετρικά στον κώδικα του stub. Παραδείγματα υπηρεσιών που βασίζονται σε αυτό είναι το Transaction Management, Lifecycle Management, Interceptors, Logging, Security κ.τ.λ..

### Αναγκαίες Προϋποθέσεις για τη Χρήση του Dependency Injection

Παρακάτω καταγράφονται συγκεντρωτικά όλες οι προϋποθέσεις τις οποίες θα πρέπει να πληροί μία εφαρμογή που εφαρμόζει DI. Ως αναγκαία προϋπόθεση θεωρήθηκε και η χρήση ενός IoC Container.

1. Για κάθε κλάση που δεν αντιπροσωπεύει απλώς μια δομή δεδομένων (π.χ. Student, AppointmentRecord κ.τ.λ.) αλλά η οποία ορίζει μεθόδους οι οποίες περιέχουν business logic (π.χ. BillingServiceImpl.approveLoan(Loan)) ή utility functions (π.χ. CryptographyUtilsImpl.encryptText(String text, String encAlgorithm)), θα πρέπει να ορίζεται ένα interface με όλες τις public ή package access μεθόδους τους (π.χ. τα interfaces BillingService και CryptographyUtils για τις προηγούμενες κλάσεις).
2. Όσον αφορά τις κλάσεις αυτές, δεν θα πρέπει να αναφέρονται στον κώδικα της εφαρμογής (ιδανικά καθόλου, εκτός προφανώς από τα αρχεία java στα οποία ορίζονται). Αντιθέτως ο κώδικας θα πρέπει να χρησιμοποιεί μόνο τα αντίστοιχα interfaces. Οι κλάσεις αυτές θα πρέπει να αναφέρονται μόνο στα σημεία στα οποία ορίζονται οι εξαρτήσεις μεταξύ τους (με annotations εσωτερικά σε αυτές ή σε ειδικά xml configuration αρχεία).
3. Οι κλάσεις αυτές δεν θα πρέπει να δημιουργούν εσωτερικά, αντικείμενα άλλων κλάσεων για να ικανοποιήσουν τις λειτουργικές εξαρτήσεις τους από αυτές εκτός και εάν η υλοποίηση για μια συγκεκριμένη διεπαφή (interface) θα πρέπει να αποφασιστεί κατά το runtime και όχι το development. Οι κλάσεις αυτές πρέπει να ορίζουν τις εξαρτήσεις τους από άλλες κλάσεις μέσω fields (μπορούν να είναι και private), constructor methods και setter methods.
4. Η δημιουργία των instances των business logic και utility κλάσεων θα πρέπει να γίνεται από τον IoC Container. Στις πολύ σπάνιες περιπτώσεις στις οποίες η υλοποίηση μίας διεπαφής θα πρέπει να αποφασιστεί κατά το runtime, οι εξαρτήσεις αυτές θα πρέπει να ικανοποιούνται από τις εξωτερικές κλάσεις (assemblers) πάλι όμως μέσω των μηχανισμών που προσφέρει ο IoC Container.
5. **Ιδανικά δεν θα πρέπει να γίνεται καθόλου χρήση static μεθόδων.** Αντί για αυτό θα πρέπει να αντικατασταθούν με non static μεθόδους, να δημιουργηθούν τα αντίστοιχα

interfaces και να οριστούν τα αντίστοιχα dependencies όπως εξηγήθηκε προηγουμένως.

**Το πρόβλημα με τις static μεθόδους είναι ότι δεν μπορούν να γίνουν override και επομένως είναι αδύνατον να εφαρμοστεί DI σε αυτές.** Δηλαδή με βάση τους κανόνες της Java εάν μία κλάση A ορίζει μία static μέθοδο func, και μία κλάση B την κάνει extend και ταυτόχρονα ορίζει και αυτή μία static μέθοδο func τότε εάν αναθέσουμε ένα αντικείμενο της κλάσης B σε μια μεταβλητή τύπου A, έστω a, τότε η κλήση a.func() θα εκτελέσει την μέθοδο func της κλάσης A και όχι της B, όπως συμβαίνει με τις non static μεθόδους.

## Πλεονεκτήματα

### Τα Πλεονεκτήματα του DI στο Σύνολό τους

Σε αυτό το σημείο τα πλεονεκτήματα του DI παρουσιάζονται συγκεντρωτικά. Θεωρούμε προφανώς όλα τα πλεονεκτήματα, είτε αυτά προέρχονται από την εφαρμογή των κανόνων που πρέπει να ακολουθεί ο κώδικας είτε αυτά οφείλονται στην εφαρμογή ενός IoC Container.

- Οποιαδήποτε αλλαγή στην υλοποίηση κάποιου κομματιού του κώδικα γίνεται με τον λιγότερο δυνατό κόπο (**decoupling**).
- Καθιστούμε το **testing** του κώδικα πολύ πιο εύκολο. Το testing μπορεί να γίνει σε οποιοδήποτε κομμάτι του κώδικα επειδή μπορεί να αντικατασταθεί οποιοδήποτε κομμάτι του εύκολα χωρίς να απαιτεί αλλαγές στα υπόλοιπα. Έτσι μέσω τις μεθόδου του αποκλεισμού φτάνουμε στο κομμάτι του κώδικα στο οποίο εμφανίζεται το πρόβλημα.
- Η ικανοποίηση των εξαρτήσεων μέσω των IoC Containers είναι απαραίτητη για την χρήση των **Container Services** του Application Server ή οποιουδήποτε άλλου συστήματος που παίζει τον ρόλο ενός application container.
- Μας αναγκάζει να διαχωρίζουμε τις έννοιες του **“contract”**, δηλαδή των διεπαφών, και του **“implementation”**, δηλαδή της υλοποίησης. Έτσι μας κατευθύνει στο να σκεφτόμαστε αφαιρετικά και συνθετικά, κάνοντας ανάπτυξη της απαιτούμενης λειτουργικότητας από «πάνω προς τα κάτω» χωρίς να υπερ-αναλύουμε και να χανόμαστε στις λεπτομέρειες πριν να έχουμε ξεκαθαρίσει τι είναι πραγματικά απαραίτητο να υλοποιηθεί.
- Το DI δεν είναι απαραίτητο για το **modularity** του κώδικα, παρόλα αυτά μας εξαναγκάζει να το εφαρμόζουμε και ταυτόχρονα όμως διευκολύνει και την εφαρμογή του. Μαθαίνοντας τις αρχές του DI μαθαίνουμε ταυτόχρονα και τις αρχές της δημιουργίας σωστού κώδικα.
- Επιβάλλει την δημιουργία κώδικα με βάση συγκεκριμένους κανόνες και ως εκ τούτου συμβάλει στην **συνοχή και την ομοιομορφία** του. Αυτό από μόνο του είναι ένα μεγάλο πλεονέκτημα σε έργα στα οποία εμπλέκονται πολλοί προγραμματιστές σε διαφορετικές χρονικές περιόδους.
- Το modularity και ως εκ τούτου και το DI συμβάλει στην δημιουργία **εύκολα αναγνώσιμου και κατανοητού κώδικα (code readability)** πολύ απλά διότι δεν βλέπουμε την ίδια λειτουργικότητα να υλοποιείται με πολλούς διαφορετικούς τρόπους και ταυτόχρονα να μπλέκεται και με τις υλοποίησεις πολλών άλλων λειτουργικοτήτων κάτι που τείνει να συμβαίνει σε κώδικα που δεν είναι modular ή αλλιώς, που δεν ακολουθεί το **“Single Responsibility Principle”**.
- **Αποτρέπει από την χρήση static μεθόδων** των οποίων η υλοποίηση δεν μπορεί να γίνει override από κάποια άλλη μέσω του DI όπως συμβαίνει με τις non static μεθόδους,

επομένως δεν μπορεί ούτε να γίνει εύκολο testing πάνω σε αυτές. Επίσης οι static μέθοδοι τείνουν να χρησιμοποιούν και static fields τα οποία έχουν πολλά μειονεκτήματα σε κώδικα που κάνει χρήση νημάτων αφού μπορεί εύκολα να γραφτεί κώδικας με bugs που οφείλονται σε concurrency issues που είναι δύσκολο να εντοπιστούν αλλά και να διορθωθούν.

Γενικά δεν μπορούμε να προβλέψουμε μελλοντικά το αν ο κώδικάς μας θα χρειαστεί μία διαφορετική υλοποίηση σε κάποιο σημείο του αλλά ούτε και ποιο κομμάτι του θα μας βολέψει να αντικαταστήσουμε για λόγους testing. Επίσης δεν μπορούμε να προβλέψουμε και το που θα χρειαστεί να χρησιμοποιήσουμε container services, που ίσως να μην έχουμε ανάγκη την περίοδο που δημιουργείτε ο κώδικας αλλά θα χρειαστούμε μελλοντικά. Για αυτούς τους λόγους **ο γενικός κανόνας είναι η εφαρμογή του DI καθολικά σε όλη την εφαρμογή και όχι επιλεκτικά σε κάποια σημεία της**.

Από όλα αυτά τα πλεονεκτήματα θα μπορούσαμε να πούμε ότι πρακτικά αυτά που καθιστούν αναγκαία την εφαρμογή του DI όσον αφορά τα **enterprise applications** είναι το **ευκολότερο testing** και η **χρήση των container services** διότι είναι δύο πράγματα τα οποία θα χρειαστούν σίγουρα ενώ π.χ. το να καταστήσουμε μία κλάση εύκολη ως προς την αλλαγή της υλοποίησής της πολλές φορές δεν φαίνεται τόσο απαραίτητο διότι όντως οι περισσότερες κλάσεις δεν χρειάζεται ποτέ να αλλάξουν την υλοποίησή τους.

Επίσης ακόμα και εάν φαίνεται ότι δεν θα χρειαστούμε ούτε κάποια διαφορετική υλοποίηση μίας κλάσης στο μέλλον, ούτε να την συμπεριλάβουμε σε testing αλλά ούτε και να χρησιμοποιήσει container services, οποιαδήποτε κλάση θα πρέπει να συμμετέχει στο DI διότι εφόσον ο κώδικάς μας θα ικανοποιεί το modularity θα χρειαστεί τις διευκολύνσεις του DI για ικανοποίηση εξαρτήσεων και για την συγκεκριμένη κλάση άλλα και για κλάσεις οι οποίες εξαρτώνται από αυτήν.

## Μειονεκτήματα

Για την πληρότητα της ανάλυσης του DI αναφέρονται τα ελάχιστα μειονεκτήματά που έχει. Θα διαπιστωθεί πάντως ότι δεν μπορούν να αποτελέσουν εμπόδιο στην απόφασή μας για καθολική εφαρμογή του DI στα projects μας.

- **Απαιτείται η δημιουργία ενός interface για κάθε κλάση.** Παρόλα αυτά σημειώνεται ότι όλα τα σύγχρονα IDE (περιβάλλοντα ανάπτυξης) δίνουν την δυνατότητα εξαγωγής ενός interface από μία κλάση αυτόματα.
- Επειδή δεν αναφέρονται οι κλάσεις υλοποίησης μέσα στον κώδικα στα σημεία στα οποία χρησιμοποιούνται παρά μόνο τα interfaces αυτών **Θα πρέπει, κατά την μελέτη του κώδικα, να ανατρέχουμε είτε στο XML configuration file στο οποίο ορίζονται οι εξαρτήσεις είτε να ψάχνουμε την κάθε κλάση ώστε να διαπιστώσουμε με βάση τα annotations που ορίζονται μέσα σε αυτήν εάν όντως είναι η κλάση υλοποίησης για το interface που μας ενδιαφέρει.** Η αλήθεια είναι όμως ότι όλα τα σύγχρονα IDE αναλύουν αυτόματα τον κώδικα και μας δίνουν την δυνατότητα να βρούμε τις κλάσεις υλοποίησης άμεσα, οπότε αυτό το μειονέκτημα πρακτικά δεν υφίσταται.
- Μπορεί πολλές φορές να διαπιστωθεί ότι κάποιες κλάσεις που τελικά δεν χρειάστηκαν πότε τους κανένα από τα πλεονεκτήματα του DI, θα μπορούσαν να μην είχαν υλοποιηθεί με βάση τις αρχές του DI αλλά π.χ. απλώς όλες οι μέθοδοι τους να είχαν δηλωθεί σαν static αφού εσωτερικά δεν έχουν καθόλου state (δεν ορίζουν fields). Η εφαρμογή των απαιτήσεων του DI βέβαια δεν μπορούμε να υποστηρίξουμε ότι κάνει κάποια ζημιά στον

κώδικα από οποιαδήποτε άποψη αφού εάν λάβουμε υπόψη και τις διευκολύνσεις που παρέχουν τα σύγχρονα IDE, είναι πολύ εύκολη και γίνεται μέσα σε λεπτά, ενώ η μη εφαρμογή δημιουργεί το ρίσκο μελλοντικού προβλήματος που θα απαιτήσει πολύ περισσότερο χρόνο για την επίλυσή του.

- **Απαιτείται προϋπάρχουσα γνώση για τις αρχές, τα πλεονεκτήματα, τα μειονεκτήματα καθώς και τις τεχνικές εφαρμογής του DI**, κάτι το οποίο μπορεί να καταστήσει την υιοθέτησή του ως τρόπου ανάπτυξης προβληματική σε ένα έργο στο οποίο εμπλέκονται πολλά άτομα με διαφορετικά επίπεδα εμπειρίας στην ανάπτυξη εφαρμογών.

Απ' όλα τα προηγούμενα μειονεκτήματα η αλήθεια είναι ότι το μοναδικό σοβαρό είναι το τελευταίο. Σίγουρα όμως ο χρόνος που θα επενδυθεί στην επιμόρφωση των developers ενός project για το DI θα αποδώσει πολλαπλά στο μέλλον από πολλές απόψεις.

## **Επόμενα Κεφάλαια**

Κατά την σταδιακή παρουσίαση των παραδειγμάτων και κυρίως στα κεφάλαια που αφορούν τα EJB θα αναφέρονται διάφοροι τρόποι για εφαρμογή του Dependency Injection που μπορεί να ορίζονται στο EJB πρότυπο αλλά και έξω από αυτό (όπως το **Context and Dependency Injection, CDI**).

Υπάρχει επικάλυψη μεταξύ των προτύπων και των τεχνικών για το Dependency Injection στα πρότυπα των **EJB** και **CDI** και πολλά από αυτά σχετίζονται με το **JNDI** (υποστηρίζουσα υλοποίηση).

## **Enterprise Java Beans (EJB)**

Πρόκειται για το πρότυπο της Java EE που υποστηρίζει κυρίως το **Business Logic** μίας εφαρμογής όπως αυτό ορίζεται σε μία **Multi-Tier Αρχιτεκτονική**. Επίσης καλύπτει **Dependency Injection** ανάγκες.

Τα EJB προσφέρουν στον προγραμματιστή πολλές υπηρεσίες και κυρίως ενσωματώνουν τις υπηρεσίες **Transactions** και **State Management (Session Beans)**, **JMS (Message Driven Beans)**, **Database Access** και **ORM (Object-Relational Mapping)** μέσω των **Entity Beans**. Λέγοντας «**State Management**» εννοείται η διατήρηση της κατάστασης των δεδομένων ενός Session Bean μεταξύ διαδοχικών κλήσεων των μεθόδων του.

Επίσης ο EJB Container που είναι μέρος ενός Application Server διαχειρίζεται τον **κύκλο ζωής (Lifecycle)** των Enterprise Beans και σε συνδυασμό με δυνατότητες **Dependency Injection** η διαχείριση της αρχικοποίησης των Enterprise Beans διευκολύνεται κατά πολύ εξαιτίας της δυνατότητας να ορίζεται **δηλωτικά**.

Ουσιαστικά τα EJBs είναι σαν μια κόλα που συνδέει όλες τις προηγούμενες υπηρεσίες και διευκολύνει την εφαρμογή τους στις επιχειρησιακές εφαρμογές.

## **Context & Dependency Injection**

Το CDI είναι ένα πρότυπο της Java EE που καλύπτει το Dependency Injection. Η διαφορά του με το EJB είναι ότι δεν κάνει κάποια διασύνδεση με τα Container Services αλλά ασχολείται αποκλειστικά με τα θέματα του Dependency Injection, Scope και Lifecycle Management χωρίς να απαιτεί τα συστατικά που διαχειρίζεται να ορίζονται ως EJB, μπορούν δηλαδή να είναι απλά POJO.

Καλύπτει κάποιες ελλείψεις του EJB προτύπου σε αυτά τα τρία θέματα αλλά στην πραγματικότητα σταδιακά γίνεται προσπάθεια συνένωσης των δύο αυτών προτύπων σε επόμενες εκδόσεις της Java EE.

## Application Packaging

### Εισαγωγή

Οι **Enterprise Application Servers** ακολουθούν ένα συγκεκριμένο μοντέλο για την δημοσίευση των **Enterprise Applications**. Πιο αναλυτικά μία εφαρμογή γίνεται packaged σε ένα **module** ή σε συνδυασμό από modules διαφορετικών ειδών τα οποία περιλαμβάνονται μέσα σε ένα **ear αρχείο (Enterprise Archive)**.

Τα modules είναι 4 ειδών, **Web Module**, **EJB Module**, **Client Application Module** και **Resource Adapter Module**. Το καθένα από αυτά γίνεται packaged σε αρχείο τύπου **jar** (EJB Module, Client Application Module), **war** (Web Module) ή **rar** (Resource Adapter Module). Όλα τους ακολουθούν εσωτερικά την δομή ενός **jar αρχείου** απλά αλλάζει η κατάληξη τους, τα components που αποθηκεύουν εσωτερικά και οι deployment descriptors τους.

### Deployment Descriptors

Οι **Deployment Descriptors** είναι xml αρχεία τα οποία είναι προαιρετικά (από την Java 1.7 και μετά), ορίζονται για κάθε τύπο module/container και χωρίζονται σε δύο κατηγορίες, τους **Java EE** και τους **Runtime**.

#### [Java EE deployment descriptors](#)

Οι **Java EE Deployment Descriptors** ορίζονται από τα επίσημα J2EE πρότυπα της Java και περιλαμβάνουν διάφορες παραμέτρους για την ρύθμιση του container και της εφαρμογής που αντιστοιχεί στο module. Π.χ. στον Java EE Deployment Descriptor ενός Web Module (που είναι το αρχείο **web.xml**) μπορούμε να ορίσουμε τον χρόνο στον οποίο κάνει timeout το session, τα URLs των servlets, κανόνες ασφάλειας και περιορισμών πρόσβασης κ.α..

#### [Runtime deployment descriptors](#)

Οι **Runtime deployment descriptors** περιλαμβάνουν και πάλι παραμέτρους για την ρύθμιση του εκάστοτε container αλλά αυτήν την φορά οι descriptors αυτοί **δεν ορίζονται από το πρότυπο της Java EE αλλά από τον εκάστοτε provider του Application Server** που σημαίνει ότι δεν μπορούν να μεταφερθούν εάν αποφασιστεί να γίνει δημοσίευση της εφαρμογής σε κάποια διαφορετική υλοποίηση. Επίσης **προσφέρουν παραμετροποίηση που αφορά τις λειτουργικότητες που υπάρχουν αποκλειστικά στην συγκεκριμένη υλοποίηση του Java EE προτύπου** (π.χ. Weblogic Application Server, JBoss, Glassfish κ.τ.λ.). Στην συγκεκριμένη εργασία θα εξεταστούν οι runtime deployment descriptors του Weblogic Application Server 12.1.3.

## Java EE Application Modules

Παρακάτω παρουσιάζονται οι 4 τύποι των modules που πρέπει να υποστηρίζονται από έναν application server. Στα modules Web, Ejb και Client αντιστοιχούν και οι containers του ίδιου τύπου. Όπως έχει προαναφερθεί τα modules είναι κατάλογοι που μπορεί να έχουν γίνει

packaged σε αρχεία **jar**, **war** ή **rar** ή να είναι **expanded** (δηλαδή απλοί φάκελοι) και περιέχουν αρχεία τύπου class (binary files των java κλάσεων της εφαρμογής), βιβλιοθήκες, configuration files όπως deployment descriptors, static resources και οτιδήποτε άλλο σχετίζεται με τα εκτελέσιμα της εφαρμογής που φτιάχνουμε:

1. **Web Module:** Το module αυτό γίνεται packaged σε αρχείο με κατάληξη war (web archive). Περιλαμβάνει web components όπως servlets, JSPs αλλά και web resources όπως στατικές html σελίδες, εικόνες, javascript files, css κ.τ.λ..
2. **EJB Module:** Το module αυτό γίνεται packaged σε αρχείο με κατάληξη jar (java archive). Περιλαμβάνει java κλάσεις που υλοποιούν τα java enterprise beans.
3. **Client Application Module:** Το module αυτό γίνεται packaged σε αρχείο με κατάληξη jar (java archive). Περιλαμβάνει java κλάσεις που υλοποιούν την εφαρμογή στην πλευρά του πελάτη στο πλαίσιο μίας client – server υλοποίησης. Δεν θα εξετασθεί στα πλαίσια της παρούσας εργασίας.
4. **Resource Adapter Module:** Το module αυτό γίνεται packaged σε αρχείο με κατάληξη rar (resource adapter archive). Περιλαμβάνει java κλάσεις και native libraries που υλοποιούν την διασύνδεση με κάποιο EIS (Enterprise Information System). Δεν θα εξετασθεί στα πλαίσια της παρούσας εργασίας.

Συνδυασμός αυτών των modules μπορεί να δημοσιευτεί μέσα σε ένα **EAR αρχείο (enterprise archive)** και είναι ο τρόπος με τον οποίον ομαδοποιούμε modules διαφορετικού τύπου μίας σύνθετης εφαρμογής.

Ένα Java EE Application θα μπορούσε να αποτελείται από ένα μόνο από αυτά τα modules ή από συνδυασμό τους.

Στην συνέχεια εξετάζονται τα **Java Web Applications** που αποτελούνται μόνο από ένα **web module**.

## Web Module - Directory Structure

Ένα **Java Web Application** μπορεί να γίνει deployed σε έναν application server ως **ένα directory** που έχει αρχεία και μια συγκεκριμένη δομή φακέλων ή **ως ένα web archive**, το οποίο είναι ένα zip αρχείο με κατάληξη war, που ακολουθεί τους ίδιους κανόνες που ισχύουν και για το directory με την διαφορά ότι περιέχει υποχρεωτικά και τον πρόσθετο φάκελο **META-INF** (σε ένα directory χρειάζεται μόνο σε συγκεκριμένες περιπτώσεις) που περιέχει το **MANIFEST.MF** αρχείο το οποίο πρέπει να υπάρχει σε όλα τα archives της Java (όπως π.χ. και τα JAR αρχεία).

To **directory structure** ενός Java Web Application ξεκινάει από τον **root φάκελο** που αντιστοιχεί στο **context root** του web application.

To **context root** είναι το πρόθεμα του URL Path (του μέρους του URL δηλαδή που ορίζεται μετά το host και το port) κάτω από το οποίο βρίσκονται όλα τα resources του web application. **Κάθε web application που είναι deployed σε έναν Servlet Container έχει το δικό του μοναδικό context root που δεν μπορεί να είναι ίδιο με το πρόθεμα ενός άλλου context root.**

Ο root φάκελος περιέχει **JSP σελίδες**, **HTML static pages**, **CSS style sheets**, **images** και οποιοδήποτε άλλο web resource χρειάζεται για την εφαρμογή και μπορεί να προσπελάζεται ελεύθερα από τον χρήστη εισάγοντας το κατάλληλο URL στην γραμμή διεύθυνσης του browser.

Επίσης ο root φάκελος περιέχει τον φάκελο **WEB-INF** ο οποίος περιέχει τους φακέλους **classes** και **lib** και τα αρχεία **web.xml (deployment descriptor)** και το **runtime descriptor** του οποίου το

όνομα διαφέρει από υλοποίηση σε υλοποίηση (στην περίπτωση του Weblogic είναι το **weblogic.xml**).

**Οτιδήποτε βρίσκεται εκτός του φακέλου WEB-INF (και επίσης και του META-INF στην περίπτωση ενός war) είναι προσπελάσιμο από τους χρήστες της εφαρμογής απλώς προσθέτοντας στο URL μετά το context root το μονοπάτι του resource μέσα στον φάκελο. Αργότερα θα αναφερθεί το πως μπορούμε να περιορίσουμε την πρόσβαση σε αυτά τα resources κάνοντας χρήση του Java Security και των security constraints.**

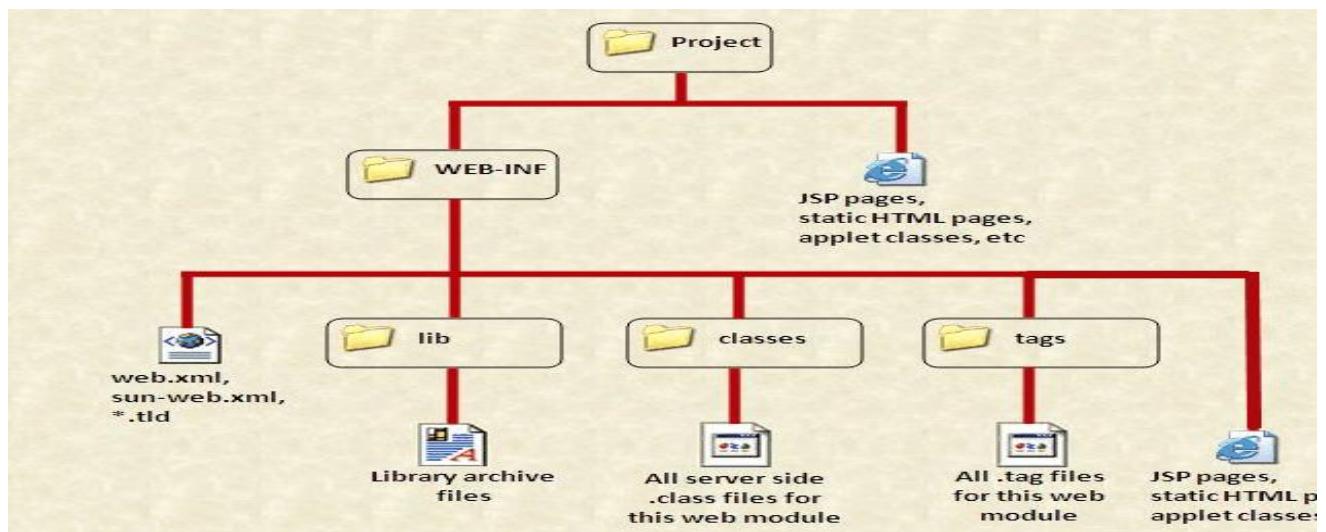
**Οτιδήποτε βρίσκεται μέσα στον φάκελο WEB-INF δεν είναι άμεσα προσπελάσιμο μέσω URLs που αντιστοιχούν στην δομή των φακέλων όπως προηγουμένως αλλά μέσω ειδικών μεθόδων που θα αναλυθούν παρακάτω. Εδώ αποθηκεύονται καταρχάς τα compiled java classes (αρχεία με κατάληξη .class) της εφαρμογής στον φάκελο **classes**. Σε αυτά περιλαμβάνονται προφανώς και τα class files που αντιστοιχούν στους servlets της εφαρμογής μας. Επίσης αποθηκεύομε τις βιβλιοθήκες που χρησιμοποιούνται αποκλειστικά από την εφαρμογή μας στον φάκελο **lib**.**

**Οι κλάσεις των Servlets αντιστοιχίζονται σε URL Patterns (όπου το καθένα αντιστοιχεί σε ένα σύνολο από URLs) μέσω του deployment descriptor και κατ' αυτόν τον τρόπο γίνονται exposed στους χρήστες του web application.**

Εκτός από τα java class files και τις βιβλιοθήκες της εφαρμογής μπορούμε επίσης να βάλουμε JSP αρχεία στον φάκελο WEB-INF. Οι JSPs αυτές δεν είναι προσπελάσιμες από τους χρήστες του web application όπως οι αντίστοιχες JSPs στον root φάκελο όμως **μπορούν να προσπελαστούν προγραμματιστικά** μέσω του java κώδικα των servlets και κατ' αυτόν τον έμμεσο τρόπο να παρουσιαστούν στους χρήστες. Αυτό επιτυγχάνεται μέσω των κλήσεων **RequestDispatcher.forward(HttpServletRequest, HttpServletResponse)** και **RequestDispatcher.include(HttpServletRequest, HttpServletResponse)**. Οι κλήσεις αυτές μπορούν να γίνουν μέσω ενός servlet και θα εξετασθούν αργότερα.

**Επίσης μπορούν να αντιστοιχηθούν σε αυτές URL Patterns μέσω του deployment descriptor.** Έτσι θα είναι πάλι προσβάσιμες μέσω της γραμμή διεύθυνσης στο URL Pattern το οποίο θα οριστεί.

Στην επόμενη εικόνα αναπαρίσταται γραφικά η δομή των φακέλων. Προηγουμένως δεν αναφέρθηκαν τα **JSP tags** τα οποία επίσης αναπαρίστανται στην εικόνα. Αυτά είναι βιβλιοθήκες από **tags (tag libraries)** τα οποία είναι **xml elements** που μπορούν να χρησιμοποιηθούν σε JSPs.



# Εγκατάσταση του Weblogic Application Server 12C

Η εγκατάσταση έγινε σε σύστημα με Windows 7 κατεβάζοντας των installer fmw\_12.2.1.2.0\_wls.jar από αυτήν την σελίδα:

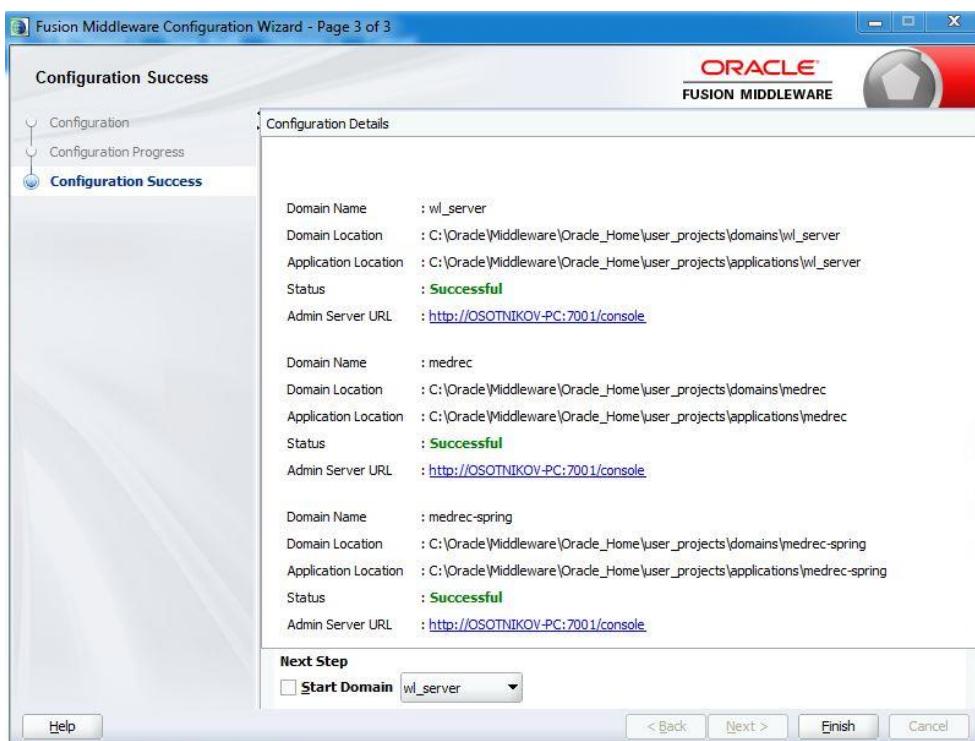
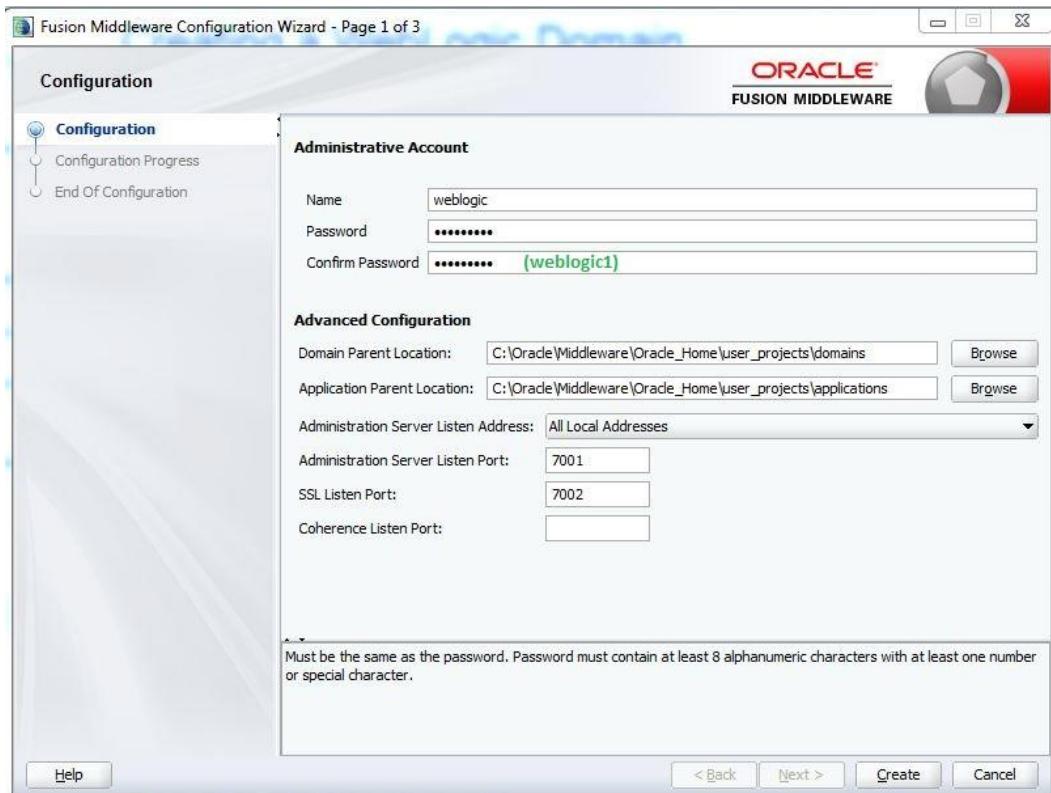
<http://www.oracle.com/technetwork/middleware/fusion-middleware/downloads/index.html>

Για την διευκόλυνση της εγκατάστασης του server έχει δημιουργηθεί το αρχείο installWLS.bat από τον εξεταζόμενο. Στην ουσία θα πρέπει να τρέξει η εντολή java -jar fmw\_12.2.1.2.0\_wls.jar στην γραμμή εντολών των Windows αλλά επειδή θα πρέπει οι μεταβλητές συστήματος PATH και JAVA\_HOME να είναι σεταρισμένες σωστά είναι πιο βολικό να τρέξει το installWLS.bat.

Όλα τα απαραίτητα αρχεία για την εγκατάσταση βρίσκονται στο συνοδευτικό DVD στον φάκελο WLS\_INSTALLATION (fmw\_12.2.1.2.0\_wls.jar, installWLS.bat, jdk-7u79-windows-x64.exe).

Παρακάτω παρουσιάζονται αναλυτικά οι οδηγίες εγκατάστασης:

1. Εάν δεν έχετε εγκατεστημένη την **Java 7 (jdk)** κατεβάστε και τρέξτε το αρχείο jdk-7u79-windows-x64.exe ή κάποιο νεότερο release της Java (υποστηρίζεται και η Java 8).
2. Ανοίξτε γραμμή εντολών ως administrator (πληκτρολογήστε cmd στο start menu και πατήστε δεξί κλικ πάνω στο cmd και επιλέξτε "Run as administrator").
3. Μεταφερθείτε στον κατάλογο που βρίσκεται το **installWLS.bat**.
4. Πληκτρολογήστε installWLS.bat και πατήστε enter. Στην ουσία θα τρέξει το fmw\_12.2.1.2.0\_wls.jar που περιέχει το installation του WLS αλλά αρχικά θέτει τις κατάλληλες τιμές στα environmental variables. Εάν αντιμετωπίσετε πρόβλημα ανοίξτε το αρχείο και αντικαταστήστε το path του JDK με το σωστό (ενδέχεται να διαφέρει στο δικό σας μηχάνημα).
5. Περιμένετε να γίνει το extract των αρχείων και ακολουθήστε τα βήματα του wizard.
6. Στον φάκελο WLS\_INSTALLATION /Screenshots - Installation & Configuration/installation μπορείτε να βρείτε τις επιλογές που έγιναν σε κάθε βήμα του wizard. Στην ουσία η μόνη ουσιαστικά επιλογή ήταν να εγκατασταθούνε όλα τα features (coherence και παραδείγματα εφαρμογών).
7. Μετά την ολοκλήρωση της εγκατάστασης ξεκινάει αυτόματα ο configuration wizard. Οι επιλογές που έγιναν παρουσιάζονται παρακάτω:



# Βασικές έννοιες του Weblogic Application Server 12C

## Domains

Ένα domain είναι ένα σύνολο από servers (με την έννοια τοπικών ή απομακρυσμένων διεργασιών), πόρους, εφαρμογές, services και ρυθμίσεις όλων αυτών. Ο Weblogic server δίνει την δυνατότητα ορισμού πολλαπλών τέτοιων domains όμως κάθε domain είναι ανεξάρτητο από τα υπόλοιπα και δεν μοιράζονται μεταξύ τους κανένα από τα προαναφερθέντα στοιχεία.

## Administration Server

Το κύριο στοιχείο σε ένα domain είναι ο **Administration Server** του. Ο ρόλος αυτού του server είναι η διαχείριση, ο συγχρονισμός και η παρακολούθηση των **Managed Servers** οι οποίοι στεγάζουν τις εφαρμογές. Παρόλα αυτά ένας Administration Server μπορεί και ο ίδιος να στεγάσει τις εφαρμογές του developer αν και αυτό αποφεύγεται για να μην υπάρχει κίνδυνος να καταρρεύσει ο administration server εξαιτίας της δυσλειτουργίας κάποιας εφαρμογής αλλά και να μην επιβαρύνεται με τον φόρτο της εξυπηρέτησης και των αιτημάτων των εφαρμογών αλλά και της διαχειριστικής κίνησης.

## Managed Servers

Ο διαχειριστής του συστήματος μπορεί να φτιάξει έναν ή παραπάνω Managed Server αφού φτιάξει τον Administration Server. **Οι Managed Servers προορίζονται για την εξυπηρέτηση των αιτήσεων προς τις web εφαρμογές** σε αντίθεση με την διαχειριστική κίνηση την οποία την εξυπηρετεί ο Administration Server.

Κάθε Managed Server είναι ένα **ξεχωριστό process** (ένα JVM) το οποίο τρέχει στο ίδιο ή σε διαφορετικό φυσικό μηχάνημα με όλα τα άλλα.

## Σημαντικοί κατάλογοι του Weblogic 12C

Τα μονοπάτια των παρακάτω καταλόγων είναι παραμετροποιήσιμα, παρουσιάζονται όμως οι τιμές οι οποίες χρησιμοποιήθηκαν για τους σκοπούς αυτής της εργασίας που είναι και αυτές που ορίζονται από προεπιλογή.

### Oracle Home (MW\_HOME/BEA\_HOME):

Είναι ο κατάλογος κάτω από τον οποίον εγκαθίσταται ο **Weblogic Application Server 12C (WL\_HOME)** στον φάκελο **MW\_HOME\wlserver**. Επίσης εδώ βρίσκονται και άλλοι συνοδευτικοί κατάλογοι που είναι σημαντικοί για την λειτουργία του. Εδώ βρίσκεται και ο κατάλογος **MW\_HOME\user\_projects** που είναι η by default τοποθεσία για τα **applications** και τα **domains** του χρήστη. Προτιμάται να είναι κάτω από τον κατάλογο **MW\_HOME** αν και δεν είναι απαραίτητο:

C:\Oracle\Middleware\Oracle\_Home

### Weblogic Home (WL\_HOME):

Είναι ο κατάλογος του Oracle Weblogic 12C Application Server. Περιέχει τα αρχεία τα οποία είναι απαραίτητα για το εκτέλεση του:

C:\Oracle\Middleware\Oracle\_Home\wlserver

### **Domain Home (DOMAIN\_HOME):**

Είναι ο κατάλογος που φιλοξενεί το domain που φτιάχτηκε για τους σκοπούς της παρούσας εργασίας.

C:\Oracle\Middleware\Oracle\_Home\user\_projects\domains\demoWebApp

### **Environmental Variables**

**Παρουσιάζονται τα environmental variables έτσι όπως ορίζονται από τα startup scripts του weblogic server.** Στην ουσία οι τιμές αυτές πάρθηκαν από την εντολή **System.getenv()** που εκτελέστηκε μέσα σε ένα web application που είχε γίνει deployed στον server στο domain με όνομα p09127 (στην συνέχεια της εργασίας θα παρουσιαστούν εφαρμογές και από άλλα domains). Οι τιμές αυτές είναι τα short paths των πραγματικών μονοπατιών τα οποία παρουσιάζονται ως “Real path”. Στα windows υπάρχουν κλήσεις συστήματος οι οποίες δίνουν τα real paths για τα αντίστοιχα short paths και δεν υπάρχει κάποιος άλλος τρόπος για να μεταφράσουμε το ένα στο άλλο (δηλαδή δεν υπάρχει κάποιο σύνολο κανόνων για να μπορούμε να εξάγουμε μόνοι μας ένα short path από ένα real path).

**COHERENCE\_HOME=C:\Oracle\MIDDLE~1\ORACLE~1\coherence**

**Real path:** C:\Oracle\Middleware\Oracle\_Home\coherence

**BEA\_HOME=C:\Oracle\MIDDLE~1\ORACLE~1**

**Real path:** C:\Oracle\Middleware\Oracle\_Home

**MW\_HOME=C:\Oracle\MIDDLE~1\ORACLE~1**

**Real path:** C:\Oracle\Middleware\Oracle\_Home

**DOMAIN\_HOME=C:\Oracle\MIDDLE~1\ORACLE~1\USER\_P~1\domains\p09127**

**Real path:** C:\Oracle\Middleware\Oracle\_Home\user\_projects\domains\p09127

**WL\_HOME=C:\Oracle\MIDDLE~1\ORACLE~1\wlserver**

**Real path:** C:\Oracle\Middleware\Oracle\_Home\wlserver

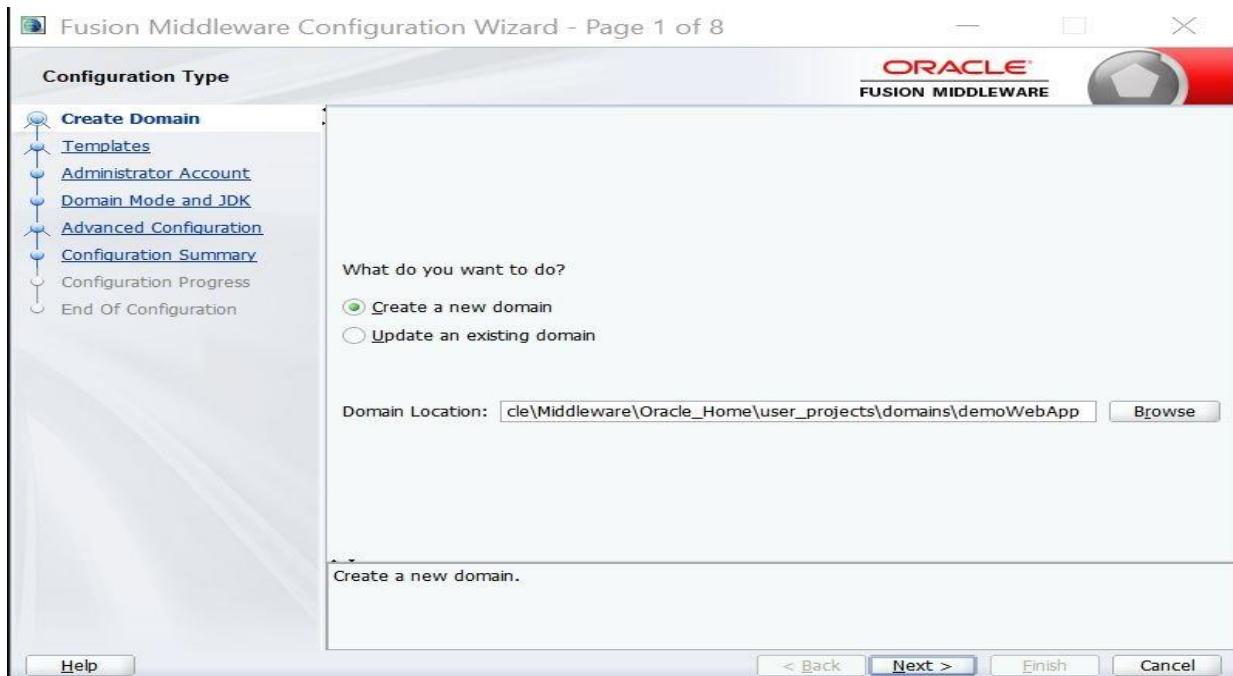
# Δημιουργία ενός domain

Για τα παραδείγματα αυτής της εργασίας θα δημιουργηθεί ένα domain με έναν administration server ο οποίος και θα στεγάσει τις δύο εφαρμογές των παραδειγμάτων.

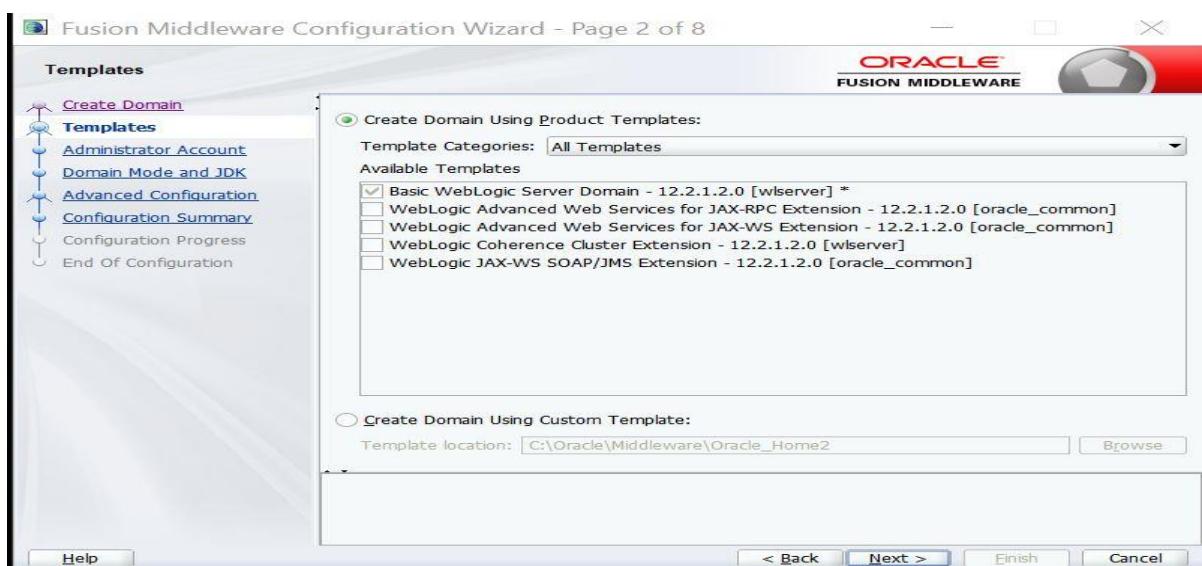
## Configuration Wizard

Ξεκινάμε τον **Configuration Wizard**. Windows Start Menu -> Oracle -> OracleHome -> WebLogic Server 12c (12.2.1) -> Tools -> Configuration Wizard.

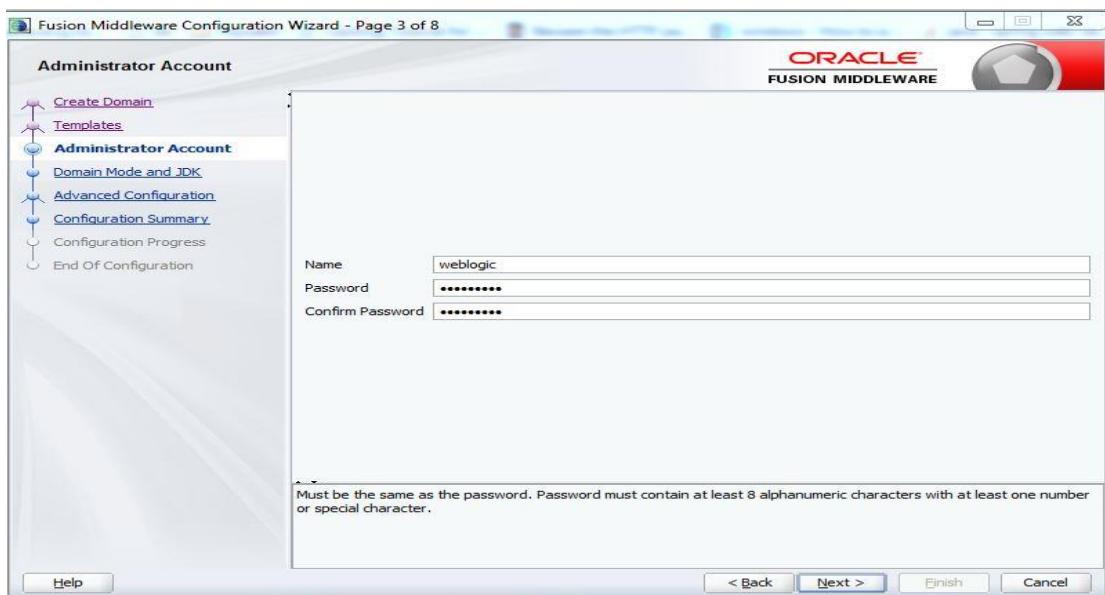
Επιλέγουμε “Create a new domain”.



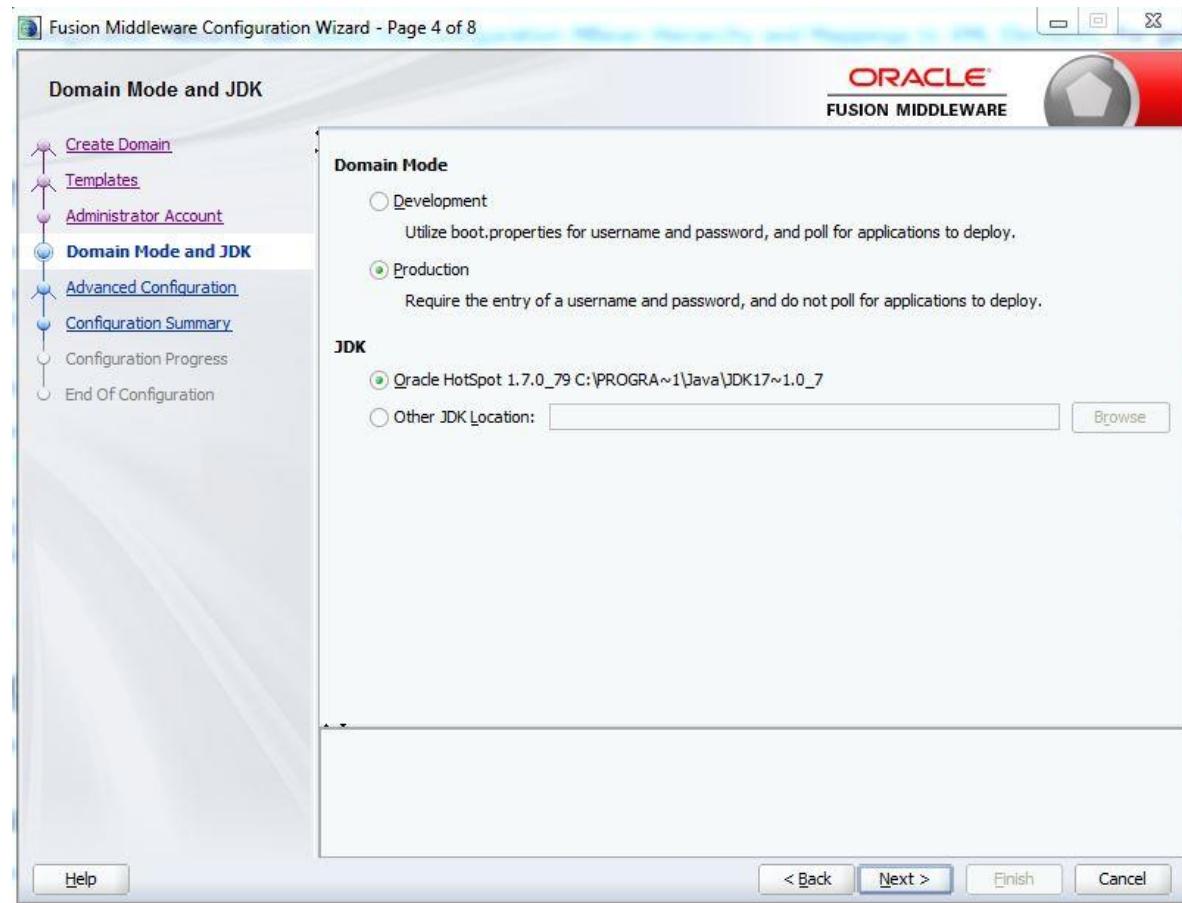
Στην συνέχεια επιλέγουμε «**Basic WebLogic Server Domain**» . Τα templates είναι ένας τρόπος για να αποθηκεύουμε πρότυπα για την δημιουργία domains.



Δίνουμε τα αναγνωριστικά για την κονσόλα διαχείρισης του Administration Server ( επίσης χρησιμοποιούνται και σε scripts π.χ. για το ξεκίνημα και το σταμάτημα των servers). Σε αυτήν την περίπτωση το Name ορίστηκε σε “**weblogic**” και το Password σε “**weblogic1**”.



Στην επιλογή για το JDK που θα χρησιμοποιηθεί επιλέγουμε τον κατάλογο στον οποίον έχουμε εγκαταστήσει το **JDK** της επιλογής μας.



## Development και Production Mode σνός Domain

Επιλέγουμε ως Domain Mode το «**Production**». Η διαφορές μεταξύ του **Development** και του **Production mode** είναι πολλές αλλά όχι πολύ σημαντικές. Οι κυριότερες είναι ότι στο development mode δημιουργείται αυτόματα το αρχείο **boot.properties** το οποίο επιτρέπει να μην εισάγονται τα server credentials κατά το startup και το shutdown τους.

### Autodeploy Folder

Επίσης στο Development Mode επιτρέπεται το **autodeploy** που διευκολύνει το **deployment** των εφαρμογών.

Στην πραγματικότητα όμως το feature αυτό δεν είναι εξαιρετικά χρήσιμο μιας και το μόνο που προσφέρει είναι το αυτόματο deployment για archives και application folders που ο χρήστης πρέπει να βάλει σε συγκεκριμένο κατάλογο (%DOMAIN\_HOME%\autodeploy) αφού προηγουμένως τα έχει προετοιμάσει. Επίσης οι εφαρμογές αυτές μπορούν να τρέξουν μόνο στον admin server και όχι επιπλέον σε κάποιους managed. Συν τοις άλλοις **θα πρέπει να τοποθετούνται και να αφαιρούνται από τον autodeploy folder όσο τρέχει ο server και εάν κάτι τέτοιο γίνει όταν δεν είναι ενεργός θα δημιουργηθούν προβλήματα συγχρονισμού της εικόνας που έχει ο server για τις deployed εφαρμογές**, οπότε όταν ακολουθείται αυτή η πρακτική υπάρχει πάντα το ρίσκο αυτού του προβλήματος εξαιτίας μίας απροσεξίας η άγνοιας από κάποιον τρίτο που να μην έχει γνώση αυτής της λεπτομέρειας.

Το deployment με την χρήση του autodeploy folder γίνεται για να διευκολυνθεί το development ή για να τεσταριστεί γρήγορα κάποια εφαρμογή της οποίας έχουμε το archive ή το expanded φάκελο και δεν θα ήταν πρακτικό να χρησιμοποιηθεί ως τρόπος δημοσίευσης εφαρμογών σε παραγωγικό περιβάλλον εξαιτίας των πόρων που απαιτεί. Ο server δηλαδή θα πρέπει να κάνει **polling** στον autodeploy φάκελο του domain πολύ συχνά, για να μπορεί να αντιδρά σε αλλαγές των περιεχομένων του και να τις κάνει auto deploy.

Εδώ το polling προφανώς αναφέρεται σε έλεγχο για το αν έχει προστεθεί ή έχει αφαιρεθεί κάποια εφαρμογή στον φάκελο. **Δεν αναφέρεται δηλαδή στο polling για αλλαγές στα περιεχόμενα των εφαρμογών** (π.χ. αλλαγή σε κάποια JSP της). Αυτού του είδους το polling ρυθμίζεται μέσα από τον **runtime descriptor** και θα αναφερθεί αργότερα.

### Εναλλακτικές Λύσεις

**Ο developer θα μπορούσε να επιτύχει τον ίδιο στόχο, της διευκόλυνσης δηλαδή του deployment κατά την φάση του development, απλά χρησιμοποιώντας κάποιο IDE που έχει plugin για τον Weblogic.**

Ένα τέτοιο plugin, που υπάρχει σε όλα τα πιο δημοφιλή IDEs, απλοποιεί την διαδικασία του deployment η οποία μπορεί να γίνει μέσα από το IDE με λίγα κλικς, στην συνέχεια το redeployment μπορεί να ρυθμιστεί ώστε να γίνεται αυτόματα μετά από κάθε build (πολύ βολικότερο δηλαδή από το να εξάγουμε κάθε φορά καινούριο war αρχείο και να το μεταφέρουμε στον autodeploy folder).

## Λοιπές Διαφορές

Εκτός από αυτά, μεταξύ των Development και Production Modes υπάρχουν **διαφορές στο logging, την ασφάλεια και την διαχείριση των ρυθμίσεων** που όμως είναι πολύ λεπτές. Επειδή συνήθως ο στόχος κατά την ανάπτυξη ενός project είναι η όσο το δυνατόν καλύτερη προσομοίωση του παραγωγικού περιβάλλοντος για την έγκαιρη αντιμετώπιση θεμάτων που μπορούν να προκύψουν σε αυτό, και επειδή **οι διαφορές μεταξύ των δύο modes δεν είναι ξεκάθαρα καταγεγραμμένες στο documentation της oracle αλλά διάσπαρτες σε πολλά σημεία,** έχει επιλεχθεί το **Production mode**.

Η αλλαγή μεταξύ του development και του production mode μπορεί να γίνει και μετά την δημιουργία του domain, από την κονσόλα του weblogic κάνοντας κλικ πάνω στο όνομα του domain στο αριστερό πλαίσιο της σελίδας, μεταβαίνοντας στο **Configuration > General** και τικάροντας ή ξετικάροντας το **Production Mode** checkbox.

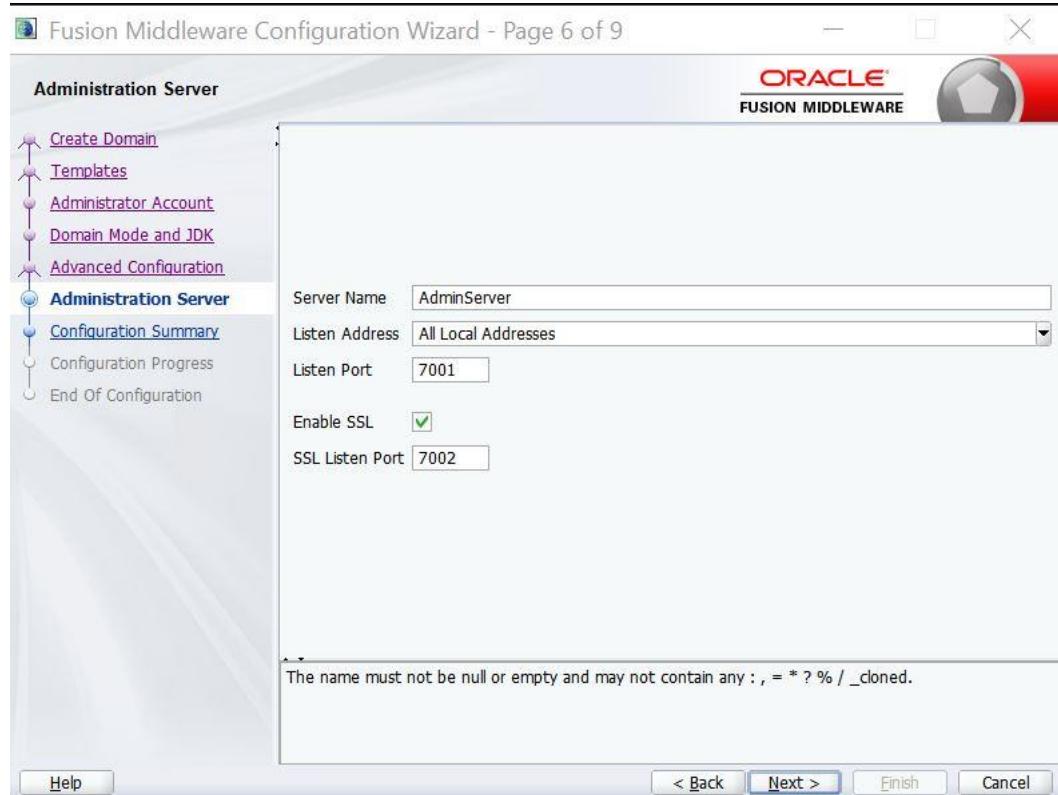
## **Advanced Configuration**

Στην επόμενη σελίδα, όπου επιλέγουμε τις ρυθμίσεις με τις οποίες θα ασχοληθούμε στα επόμενα steps του wizard, τικάρουμε την πρώτη επιλογή. Αφήνουμε την 3<sup>η</sup> επειδή σε πρώτη φάση η ανάλυση θα γίνει μόνο για ανάπτυξη και διαχείριση στον **Administration Server** χωρίς την προσθήκη Managed Servers.



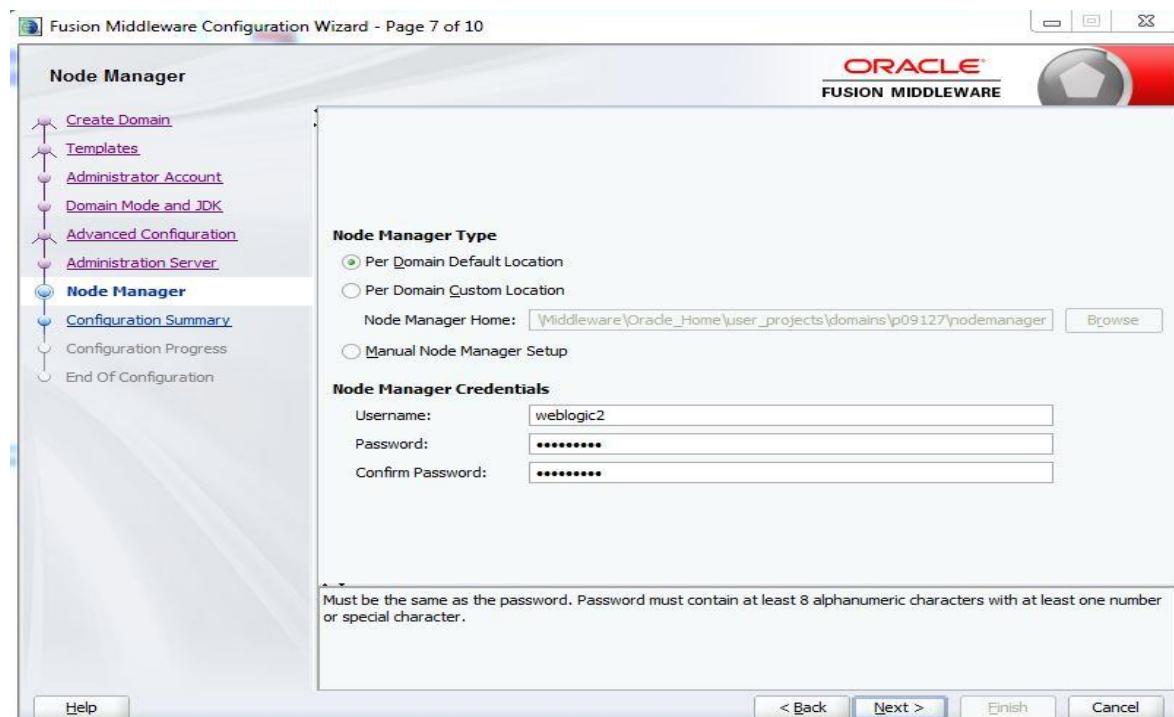
## Administration Server Configuration

Στην επόμενη σελίδα επιλέγουμε τις ρυθμίσεις της εικόνας. Ο administration server θα τρέχει στον localhost με πόρτα 7001 και 7002 για την επικοινωνίας μέσω SSL.



## Node Manager Configuration

Στην επόμενη σελίδα ρυθμίζουμε τον **Node Manager**. Ο Node Manager είναι ένας προαιρετικός μηχανισμός για το ξεκίνημα (χειροκίνητο ή αυτόματο) και το σταμάτημα του Administration και των Manager servers. Ως username και password ορίζουμε το **weblogic2**.

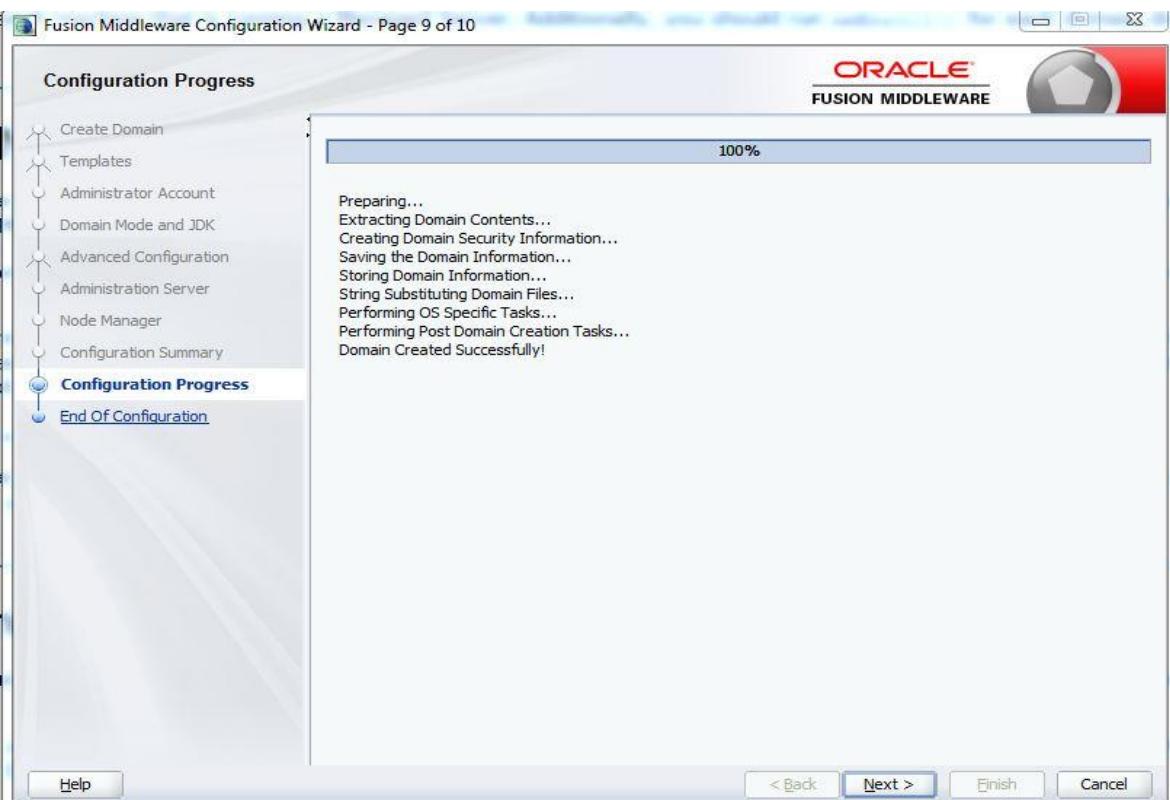


# Configuration Summary

Η επόμενη σελίδα παρουσιάζει μία σύνοψη του domain που πρόκειται να δημιουργηθεί.



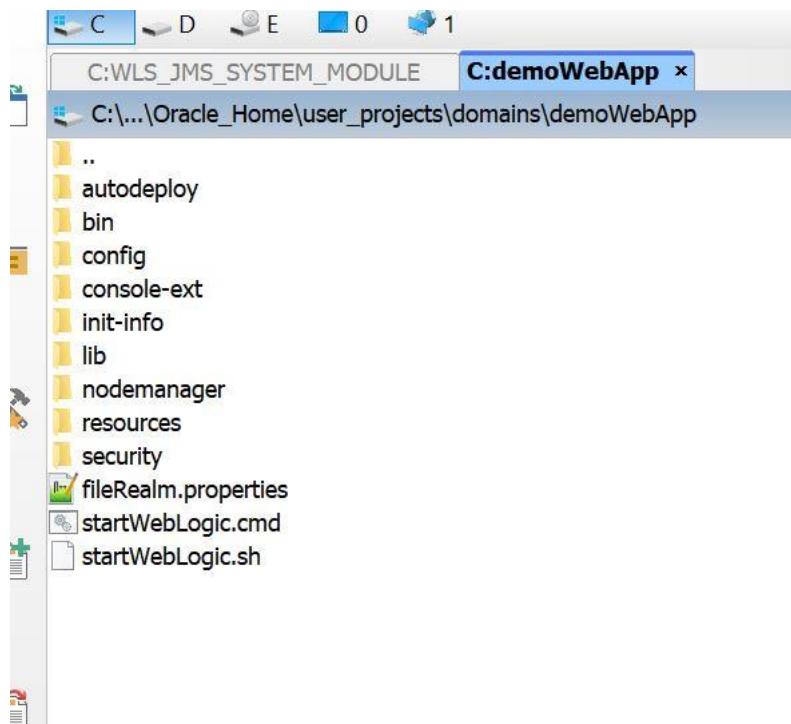
Παρουσιάζεται η επόμενη σελίδα της αναφοράς για την ενημέρωση της εξέλιξης της διαδικασίας δημιουργίας του domain.



Στην τελευταία σελίδα υπάρχει η επιλογή για το ξεκίνημα του administration server.

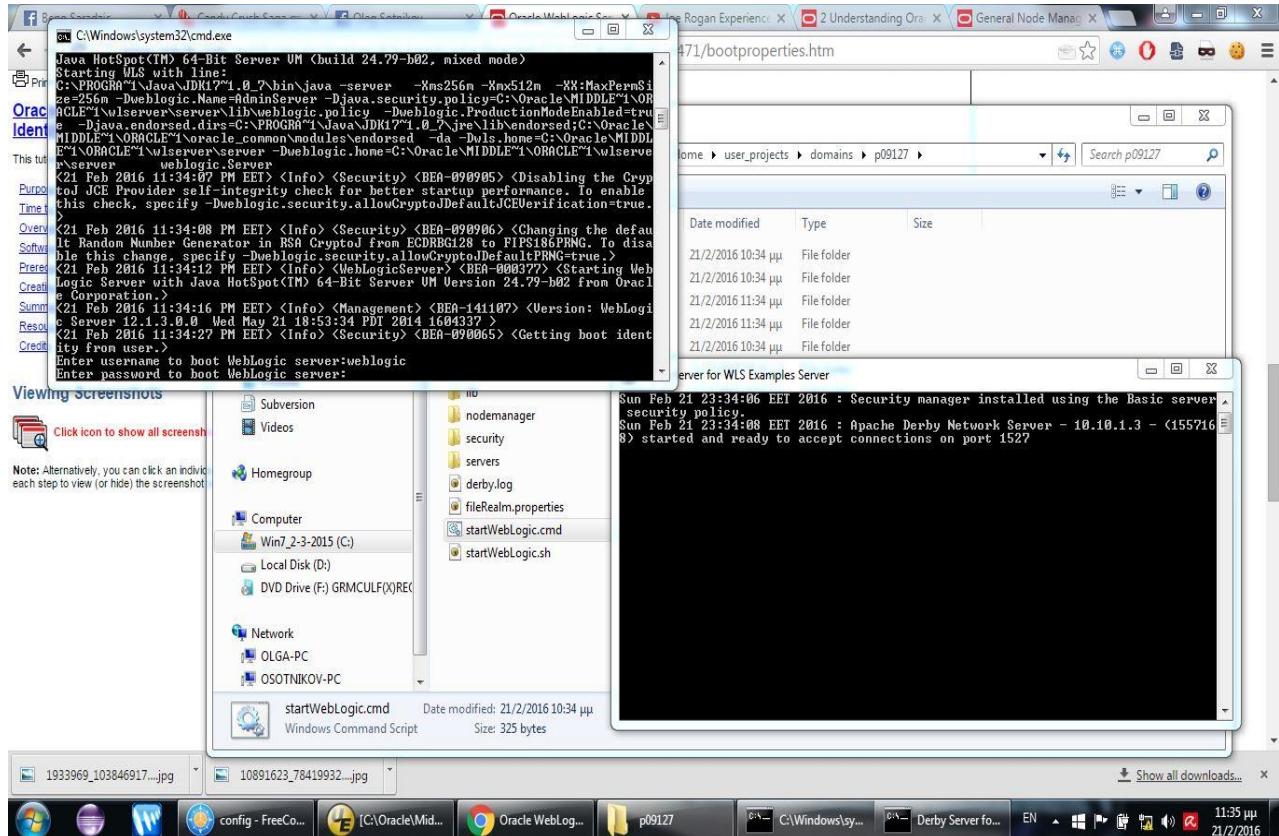


Η διαδικασία παράγει τον παρακάτω φάκελο. Λείπουν οι καταλογοί **common** και **servers**, που θα δημιουργηθούν μετά το πρώτο ξεκίνημα του administration server.



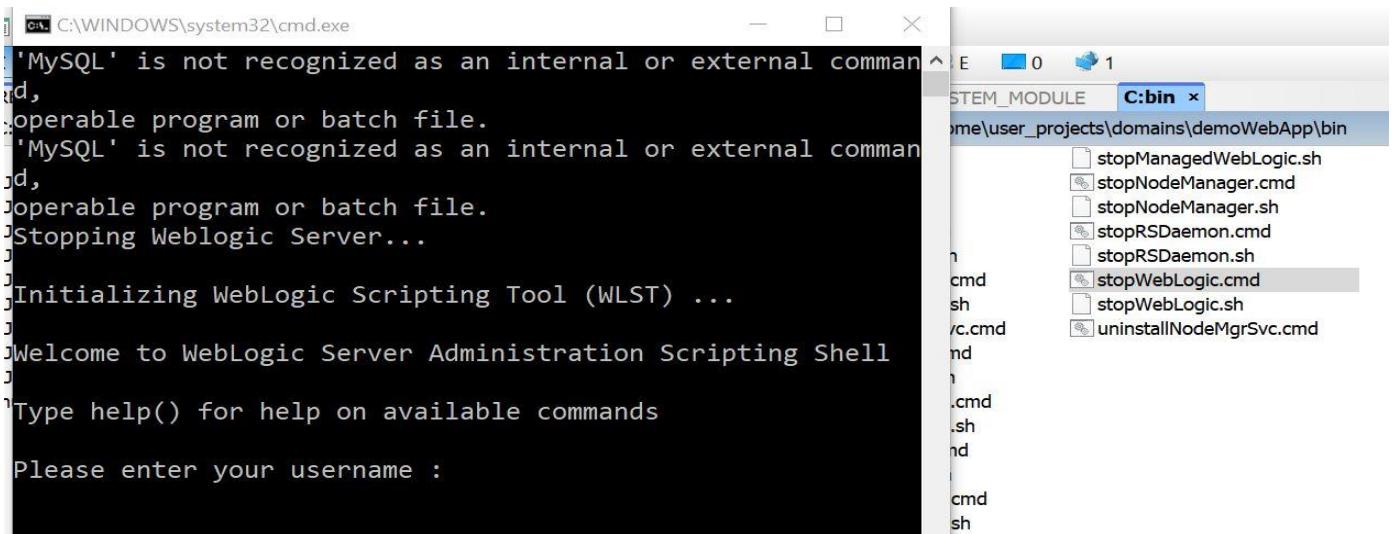
## Ξεκίνημα του Administration Server

Για το ξεκίνημα του Administration Server αρκεί να κάνουμε διπλό κλικ στο αρχείο **%DOMAIN\_HOME%\startWebLogic.cmd**. Μετά από λίγο το startup script μας ζητάει τα αναγνωριστικά του administration account. Εισάγουμε αυτά που είχαμε δώσει κατά την δημιουργία του domain.



## Σταμάτημα του Administration Server

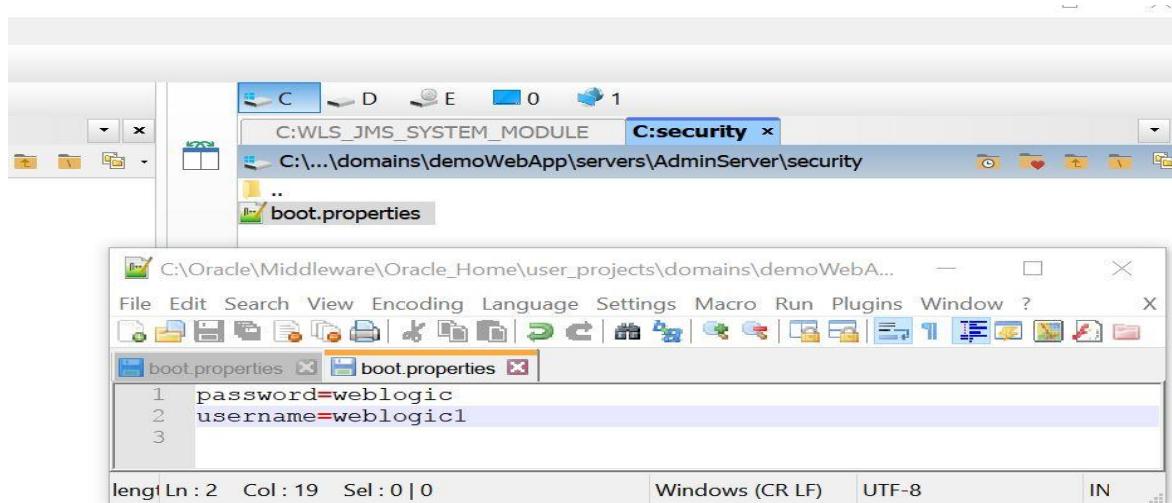
Για το σταμάτημα του administration server αρκεί να εκτελέσουμε το script **%DOMAIN\_HOME%\bin\stopWeblogic.cmd**. Κατά την εκτέλεση του θα μας ζητηθούν και πάλι τα αναγνωριστικά του administration account.



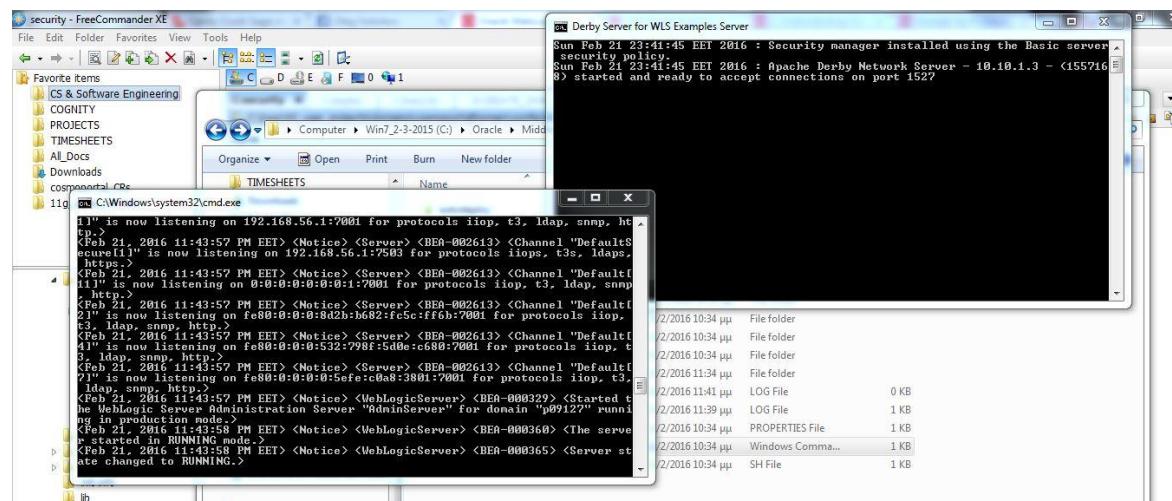
# Πύθμιση των αναγνωριστικών για τα startup και shutdown scripts (boot.properties)

Μετά το πρώτο ξεκίνημα του administration server δημιουργείται ο κατάλογος **servers** και μας δίνεται η δυνατότητα να δημιουργήσουμε το αρχείο

%DOMAIN\_HOME%\servers\AdminServer\security\boot.properties. Αφού το δημιουργήσουμε εισάγουμε σε αυτό τα αναγνωριστικά του administration account όπως στην εικόνα.



Η δημιουργία του αρχείου boot.properties μας επιτρέπει να μην εισάγουμε τα αναγνωριστικά του administration account κάθε φορά που ξεκινάμε τον administration server. Στην επόμενη εικόνα φαίνεται το ξεκίνημα του server χωρίς να έχουν ζητηθεί αναγνωριστικά.



Μετά το δεύτερο ξεκίνημα τα περιεχόμενα του boot.properties κρυπτογραφούνται.



## Διαγραφή ενός Domain

Σε περίπτωση που χρειαστεί για τον οποιοδήποτε λόγο η διαγραφή ενός domain τα βήματα παρουσιάζονται παρακάτω:

1. **Σταματάμε οποιοδήποτε process σχετίζεται με το domain**, δηλαδή τον administration, τους managed servers και τα node manager services εάν τα χρησιμοποιούμε.

2. **Διαγράφουμε το domain entry** που αντιστοιχεί στο domain που θα διαγράψουμε από το αρχείο "%MW\_HOME%\domain-registry.xml" file. Π.χ. στο παρακάτω αρχείο θα διαγράφαμε την 3<sup>η</sup> γραμμή εάν θέλαμε να διαγράψουμε το domain testDomain:

```
<?xml version="1.0" encoding="UTF-8"?>

<domain-registry xmlns="http://xmlns.oracle.com/weblogic/domain-registry">

<domain location="/u01/app/oracle/middleware/user_projects/domains/testDomain"/>

</domain-registry>
```

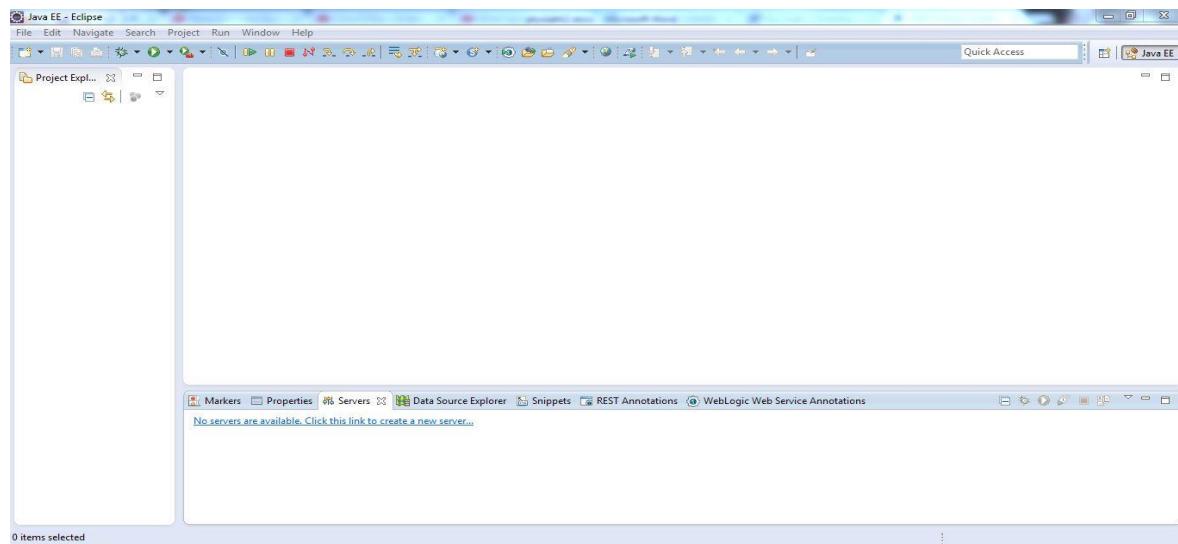
3. Διαγράφουμε τους καταλόγους του domain από τους **application** και **domain** καταλόγους, π.χ. για το domain testDomain θα διαγράφαμε τους:

```
%MW_HOME%\user_projects\applications\testDomain
%MW_HOME%\user_projects\domains\testDomain
```

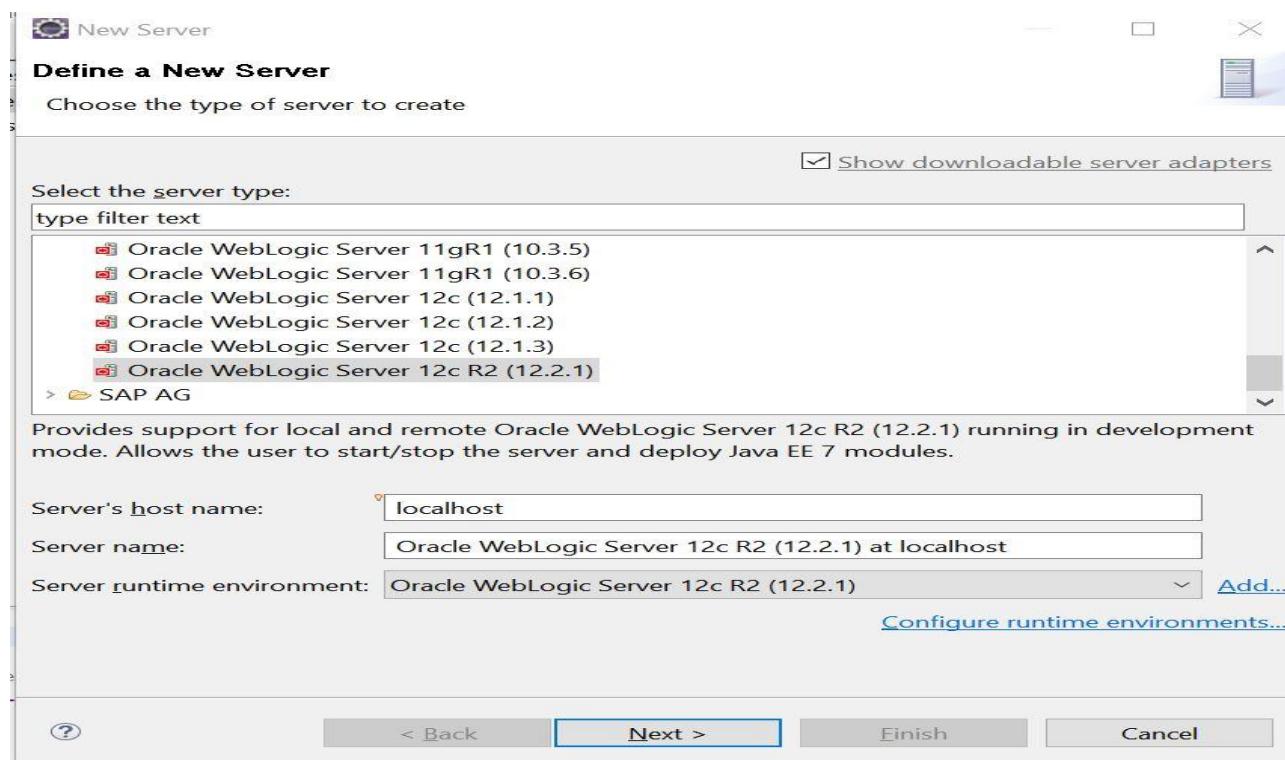
# Στήσιμο Web Project στο Eclipse

Αφού έχουμε φτιάξει το domain είμαστε έτοιμοι να δημιουργήσουμε το project στο οποίο θα δουλέψουμε στο Eclipse. Αρχικά θα πρέπει να συνδέσουμε το workspace του eclipse με το domain του weblogic που φτιάξαμε προηγουμένως.

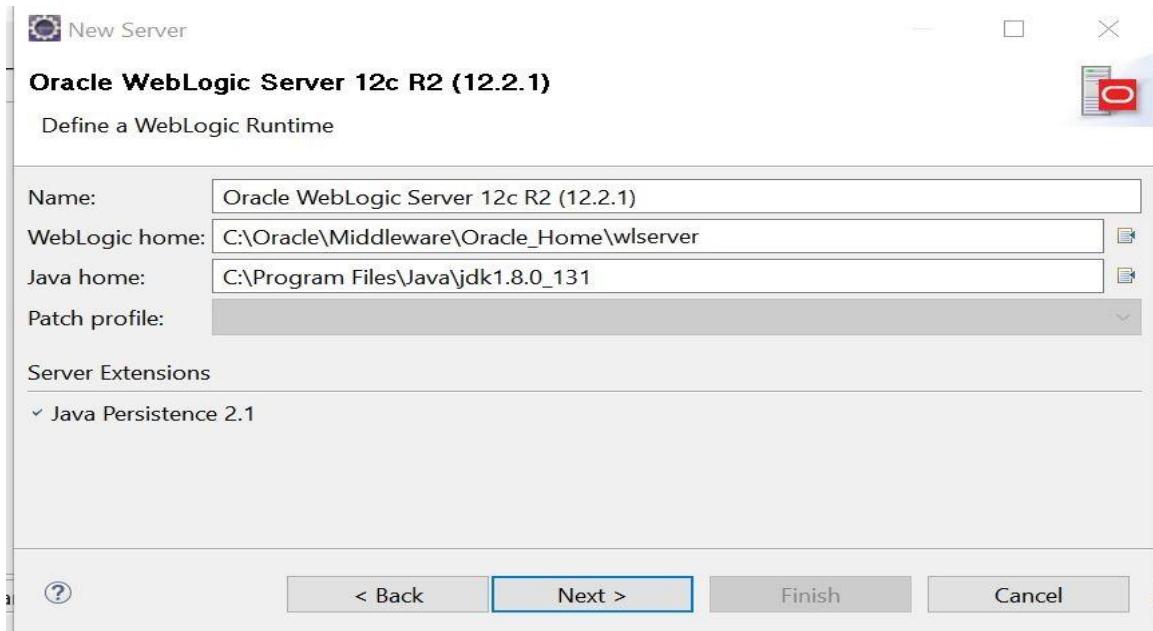
1. Επιλέγουμε το tab “**Servers**” στο κάτω frame του workspace ή μέσω του μενού Window -> Show View.



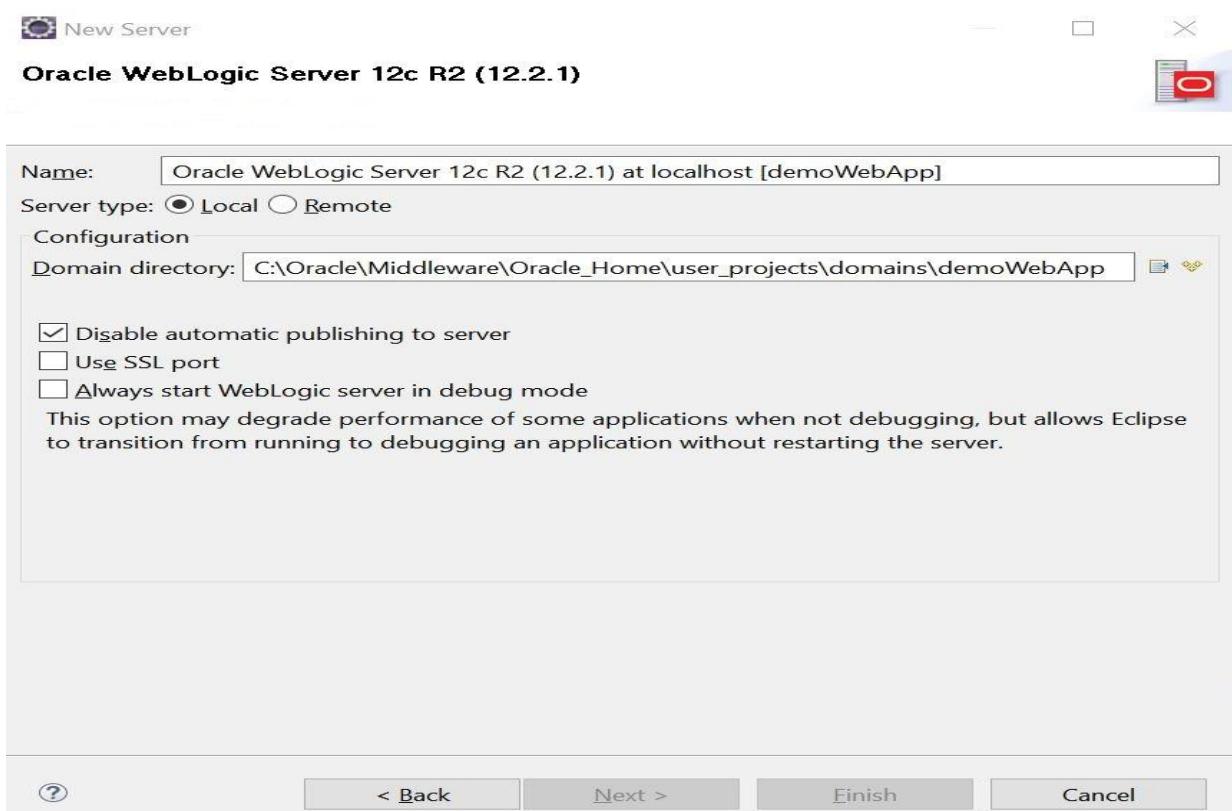
2. Κάνουμε δεξί κλικ στην κενή περιοχή και επιλέγουμε **New -> Server** ή πατάμε στο μπλε κείμενο. Στη συνέχεια επιλέγουμε **Oracle WebLogic 12c (12.2.1)**. Εάν δεν υπάρχει ως επιλογή πατάμε στο «**Download additional server adapters**» και διαλέγουμε «**Oracle Weblogic Server Tools**» ή κάτι αντίστοιχο και κατεβάζουμε τον adapter που να μπορεί να υποστηρίξει **Oracle Weblogic 12.1.3**. Πατάμε Next.



- Το πεδίο του **WebLogic Home** θα πρέπει να δείχνει στον φάκελο **wlserver** του weblogic installation που πραγματοποιήσαμε προηγουμένως (**WL\_HOME**). Το **Java Home** θα συμπληρωθεί αυτόματα, αλλιώς θα πρέπει να δηλώσουμε το Path του JDK που δηλώσαμε και κατά την εγκατάσταση του weblogic. Όπως παρατηρούμε είναι δυνατή η χρήση και ενός JDK 8.



- Στην επόμενη οθόνη επιλέγουμε ως **Server Type** το **Local** επειδή ο server θέλουμε να τρέχει στο τοπικό μας μηχάνημα όπου κάνουμε και development. Σημειώνεται ότι η επιλογή **Remote** θα μας επέτρεπε να ανεβάζουμε αλλαγές κατευθείαν σε κάποιο απομακρυσμένο μηχάνημα που συνήθως είναι βολικό όταν είναι κάποιο testing περιβάλλον που χρησιμοποιείται αποκλειστικά από ένα άτομο.



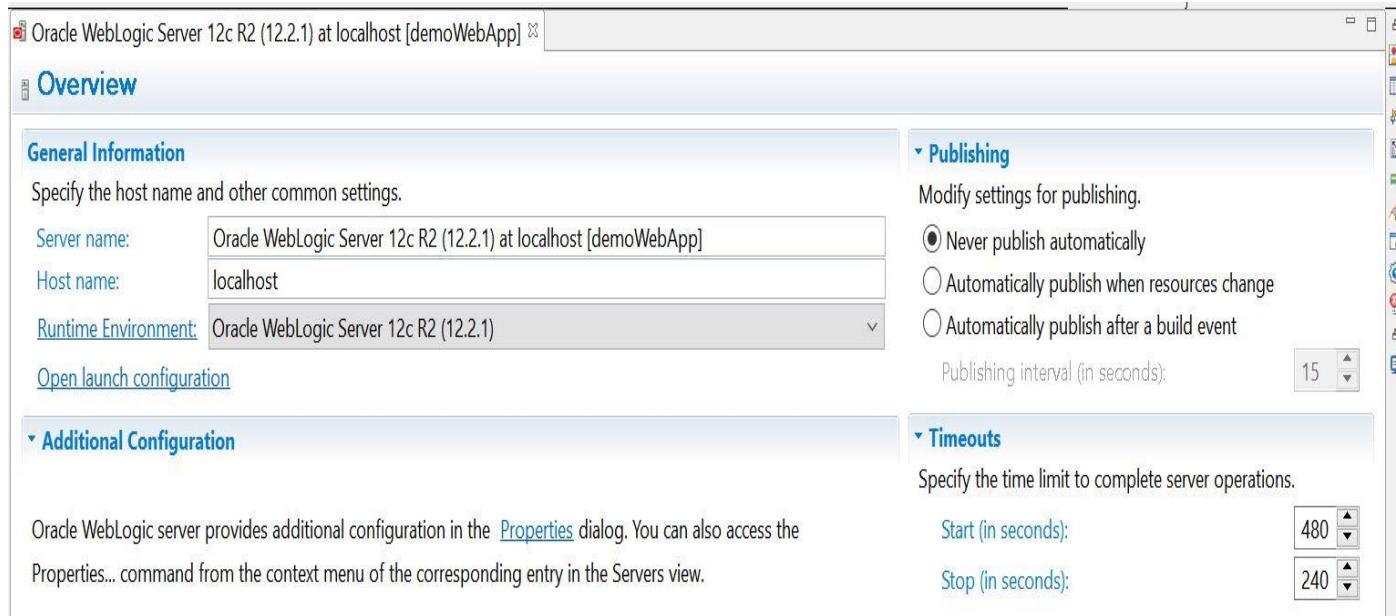
**Τικάρουμε το “Disable automatic publishing to server”.** Το publishing τοπικά είναι συνήθως πολύ σύντομο αλλά υπάρχει μία μικρή πιθανότητα να έχουμε προβλήματα που σχετίζονται με την διαχείριση της μνήμης από τον server με αποτέλεσμα κάποια στιγμή μετά από πολλά publishes να παρουσιάσει πρόβλημα και να χρειαστεί να γίνει restart.

Αφήνουμε ατικάριστο το **“Use SSL port”**. Αυτή η επιλογή ρυθμίζει το feature του ανοίγματος της web κονσόλας του server μέσα από το περιβάλλον του eclipse έτσι ώστε να την ανοίγει στην **https (SSL) Θύρα** by default. **Επίσης ρυθμίζει οποιαδήποτε επικοινωνία του eclipse με το process του server που τρέχει, να γίνεται μέσω SSL.** Παράδειγμα τέτοιας επικοινωνίας είναι η εντολή shutdown που δίνεται μέσω t3 πρωτοκόλλου. Με Server Type Local αυτό έχει νόημα μόνο εάν θέλουμε να τεστάρουμε το SSL configuration και τα certificates που έχουμε στον weblogic μιας και δεν υπάρχουν θέματα ασφάλειας αφού όλη η επικοινωνία γίνεται στο τοπικό μηχάνημα.  
**Η επιλογή αυτή θα είναι λειτουργική μόνο στην περίπτωση που κατά την δημιουργία του domain είχαμε ρυθμίσει τον server να έχει SSL Θύρα.**

Αφήνουμε ατικάριστη και την επιλογή **“Always start WebLogic server in debug mode”**. Το **debug mode** είναι ουσιαστικά ένα mode στο οποίο μπορούν να ορίζονται σημεία μέσα στον κώδικα στα οποία όταν φτάνει το execution flow θα σταματάει και θα περιμένει την εντολή του developer για να συνεχίσει, βήμα βήμα ή κανονικά. Στο εντωμεταξύ ο developer μπορεί και βλέπει τις τιμές όλων των μεταβλητών του προγράμματος.

Πατάμε **Finish**.

5. Με διπλό κλικ πάνω στον server στο tab Servers μπορούμε να δούμε κάποια στοιχεία από το configuration του:



# Servlets

## Γενική Περιγραφή

Τα Servlets είναι components τα οποία φιλοξενούνται σε έναν Servlet Container και παρέχουν δυναμικό περιεχόμενο web ιστοσελίδων. Φτιάχνονται από τον προγραμματιστή του web application φτιάχνοντας υποκλάσεις της κλάσης **HttpServlet**.

Τα Servlets δέχονται αιτήματα του HTTP πρωτοκόλλου και επιστρέφουν HTTP απαντήσεις. Θα μπορούσαν θεωρητικά να φτιαχτούν και Servlets που εξυπηρετούν και άλλα πρωτόκολλα (φτιάχνοντάς τα από την υπερκλάση **GenericServlet** αντί της HttpServlet) αλλά αυτό δεν θα εξετασθεί στην παρούσα εργασία.

Για να οριστεί μία Servlet κλάση από τον προγραμματιστή της web εφαρμογής αρκεί να δημιουργηθεί μία υποκλάση της κλάσης **javax.servlet.http.HttpServlet** και να γίνουν override οι μέθοδοι της οι οποίες καλούνται από τον application server όταν χρειάζεται να εξυπηρετηθεί κάποιο HTTP Request (**doGet**, **doPost**, **doHead**, **doTrace**, **doOptions**, **doDelete**, **doPut**).

Οι μέθοδοι εξυπηρέτησης των HTTP Requests παίρνουν ως ορίσματα τα αντικείμενα **HttpServletRequest** και **HttpServletResponse**. Το πρώτο έχει τις **παραμέτρους** και τα **headers** του **HTTP Request** ενώ το δεύτερο προορίζεται για το **HTTP Response** το οποίο θα σταλεί πίσω στον client. Οι κλάσεις **HttpServletRequest** και **HttpServletResponse** θα εξετασθούν αργότερα σε αυτό το κεφάλαιο θεωρητικά και στην συνέχεια μέσω πρακτικών παραδειγμάτων εκτέλεσης κώδικα.

## Θέματα Παραλληλίας

Ο Application Server θα φροντίσει να δημιουργήσει ένα αντικείμενο ανά JVM (εάν δεν οριστεί κάτι διαφορετικό) για κάθε διαφορετική Servlet κλάση που θα έχει ορίσει ο προγραμματιστής.

Σημειώνεται εδώ ότι σε κάθε instance του Weblogic Application Server (είτε σε έναν Administration Server είτε σε έναν Managed Server) αντιστοιχεί ένα ξεχωριστό JVM. Κάθε τέτοιο instance στεγάζει όλες τις εφαρμογές που έχουν γίνει deployed σε αυτό το instance. Εάν δηλαδή έχουμε 4 Managed Servers και κάποιος Servlet έχει τύχει να ζητηθεί κάποια στιγμή, σε καθέναν από αυτούς θα έχουμε από ένα instance του Servlet και το κάθε instance θα βρίσκεται και σε ένα διαφορετικό JVM.

Πολλά requests προς έναν Servlet θα εξυπηρετούνται **παράλληλα** από ένα αντικείμενο/στιγμιότυπο της αντίστοιχης servlet κλάσης. Αυτό σημαίνει ότι **ο προγραμματιστής θα πρέπει να εξασφαλίσει την ασφαλή ταυτόχρονη εκτέλεση των μεθόδων του Servlet που εξυπηρετούν HTTP αιτήσεις άρα και την ασφαλή ταυτόχρονη πρόσβαση προς τους πόρους, όπως π.χ. συνδέσεις προς βάσεις δεδομένων, τους οποίους διαμοιράζονται οι διαφορετικές αιτήσεις.**

Η δημιουργία ενός μόνο Servlet αντικειμένου ανά instance όμως δεν ισχύει απαραιτήτως εάν η Servlet κλάση έχει οριστεί να κάνει implement το interface **SingleThreadModel** (το interface αυτό δεν δηλώνει μεθόδους, είναι ένα “tagged interface” ή αλλιώς “marker interface”). Σε αυτή την περίπτωση ο Servlet Container θα πρέπει να εξασφαλίσει ότι κάθε Servlet αντικείμενο θα εξυπηρετεί μόνο ένα request την φορά. Για να το κάνει αυτό ενδέχεται να επιλέξει να δημιουργήσει ένα σύνολο από αντικείμενα της ίδιας Servlet κλάσης ώστε να εξυπηρετούν τα

requests προς αυτήν (**pooling**). Προφανώς όταν δεν υπάρχει πιά αντικείμενο μέσα σε αυτό το σύνολο που να μπορεί να εξυπηρετήσει ένα νέο επιπλέον request προς αυτόν τον Servlet, το request θα περιμένει μέχρι να απελευθερωθεί ένα instance από αυτό το σύνολο αλλά επίσης ενδέχεται να δημιουργούνται συνεχώς νέα Servlet instances χωρίς κάποιο περιορισμό στον αριθμό τους. Σε αυτήν την περίπτωση βέβαια ο container θα πρέπει να έχει κάποιον μηχανισμό για να προβλέπει και να διαχειρίζεται κάπως την πιθανότητα για την δέσμευση όλου του διαθέσιμου χώρου μνήμης της JVM.

Αυτοί δεν είναι οι μοναδικοί τρόποι που μπορεί να χρησιμοποιήσει ο Servlet Container για να εξασφαλιστεί ότι ένα request εξυπηρετείται από ένα μόνο instance καθώς θα μπορούσε απλώς να δημιουργήσει ένα instance του servlet αλλά να εξυπηρετεί τα requests προς αυτό σειριακά. **Ο Servlet Container είναι ελεύθερος να χρησιμοποιήσει οποιαδήποτε προσέγγιση αρκεί το τελικό αποτέλεσμα να είναι το ίδιο με το να εξυπηρετείται μόνο ένα request από ένα Servlet instance την φορά.** Οι τεχνικές αυτές μπορεί να ποικίλουν ανάλογα με την υλοποίηση.

Αυτό σημαίνει ότι ειδικά με την τελευταία προσέγγιση η απόδοση του web application πέφτει κατακόρυφα άρα πριν αποφασιστεί να χρησιμοποιηθεί το **SingleThreadModel** σε μία Servlet κλάση θα πρέπει να εξεταστούν ενδελεχώς οι συνέπειες και τις περισσότερες φορές η πιο σωστή απόφαση είναι να αποφεύγουμε την χρήση αυτού του interface εντελώς και να εφαρμόζουμε άλλες λύσεις στο πρόβλημα του συγχρονισμού και της παράλληλης επεξεργασίας των αιτήσεων.

Επίσης στο ίδιο αποτέλεσμα, στην επεξεργασία δηλαδή του κάθε request σειριακά από ένα μόνο instance, φτάνουμε και εάν δεν δηλώσουμε ως interface το **SingleThreadModel** αλλά συν τοις άλλοις δηλώσουμε ως **synchronized** την μέθοδο **service** ή κάποια από τις **doGet, doPost, doHead, doTrace, doOptions, doDelete, doPut**.

## Κύκλος Ζωής

Τα Servlets χαρακτηρίζονται ως components επειδή τα διαχειρίζεται ο Servlet Container ενός Application Server. Αυτό σημαίνει ότι **ο κύκλος ζωής των Servlets δεν είναι στον έλεγχο του προγραμματιστή της web εφαρμογής αλλά του Application Server στον οποίον είναι deployed.**

Μέθοδοι της κλάσης HttpServlet που σχετίζονται με στάδια του κύκλου ζωής ενός servlet είναι οι **init, service και destroy** που αντιστοιχούν στα στάδια **Initialization, Service και End of Service**. Εκτός από αυτά υπάρχουν και τα δύο αρχικά στάδια του **Loading** και του **Instantiation**.

Τα στάδια περιγράφονται αναλυτικά παρακάτω:

- 1. Loading:** Φορτώνεται η κλάση του Servlet στον container (δηλαδή το “Class” αντικείμενο του Servlet). Αυτό μπορεί να γίνει είτε κατά το ξεκίνημα του Application Server είτε την πρώτη φορά που ο Application Server θα χρειαστεί το συγκεκριμένο Servlet για να εξυπηρετήσει κάποιο request. Στην πρώτη περίπτωση θα έχουμε ένα πιο αργό ξεκίνημα του server αλλά δεν θα υπάρχει καθυστέρηση κατά την εξυπηρέτηση των πρώτων requests και στην δεύτερη θα ισχύει το ανάποδο. Η επιλογή για την χρονική στιγμή του loading των Servlets συνήθως παρέχεται μέσω κάποιου configuration από τον Application Server.
- 2. Instantiation:** Κάποια στιγμή μετά το loading και πριν εξυπηρετηθεί το πρώτο request δημιουργείται και το αντικείμενο από την αντίστοιχη κλάση του Servlet.
- 3. Initialization:** Κάποια στιγμή μετά το instantiation και πριν την εξυπηρέτηση του πρώτου request καλείται η μέθοδος **init** του servlet. Αυτή μπορεί να γίνει override από

τον προγραμματιστή και να οριστεί έτσι ώστε να αρχικοποιεί πόρους (όπως π.χ. συνδέσεις προς μία βάση δεδομένων) ή να κάνει άλλες απαραίτητες ενέργειες οι οποίες θα προετοιμάσουν το περιβάλλον του Servlet έτσι ώστε να παρέχει το οτιδήποτε είναι απαραίτητο σε αυτό για να μπορεί να εκτελέσει την λειτουργία του.

4. **Service:** Αυτό είναι το στάδιο όπου το Servlet είναι στην κατάσταση στην οποία εξυπηρετεί πιά τα HTTP requests των χρηστών. Κάθε request εξυπηρετείται από την μέθοδο **service** η οποία καλεί μία από τις μεθόδους **doGet**, **doPost**, **doHead**, **doTrace**, **doOptions**, **doDelete**, **doPut**, ανάλογα με το είδος του HTTP request. Στην συντριπτική τους πλειονότητα τα requests είναι τύπου GET και POST οπότε χρησιμοποιείται η **doGet** και η **doPost** αντίστοιχα.
5. **End of Service:** Είναι στην ευχέρεια του servlet container να αποφασίσει το πότε θα τελειώσει ο κύκλος ζωής ενός Servlet instance. Μπορεί να το κρατήσει στην μνήμη όσο τρέχει ο server ή μπορεί να το κάνει διαθέσιμο για garbage collection μέσα σε milliseconds από την δημιουργία του, ενδεχομένως εξαιτίας περιορισμένης διαθέσιμης μνήμης και στην συνέχεια να το δημιουργήσει πάλι για να εξυπηρετήσει τα requests που είναι pending. Όπως και να 'χει πριν το Servlet καταστραφεί θα κληθεί η μέθοδος **destroy** έτσι ώστε να δοθεί ευκαιρία στον προγραμματιστή ορίζοντάς την να διαχειριστεί την αποδέσμευση πόρων τους οποίους χρησιμοποιούσε το Servlet.

## Servlet Request (HttpServletRequest)

Η κλάση **HttpServletRequest** ενθυλακώνει το **HTTP Request** το οποίο λαμβάνει ο **Servlet Container** από τον **browser** του χρήστη.

Ένα **HTTP Request**, όπως ορίζεται στο HTTP πρωτόκολλο του **W3C**, αποτελείται από:

1. **To Request Line.** Αυτό με την σειρά του αποτελείται από:
  - a. Την δήλωση της **HTTP μεθόδου** (μία από τις GET, POST, TRACE, OPTIONS, HEAD, PUT, DELETE, CONNECT).
  - b. Το **URI του πόρου** για τον οποίον γίνεται η αίτηση.
  - c. Την δήλωση της αποδεκτής **έκδοσης του HTTP πρωτοκόλλου (HTTP/1.0 ή HTTP/1.1)**.
2. **Tα General Headers και τα Request Headers.**
3. **Tα Request Parameters.**

Το **interface HttpServletRequest** σε πρώτη φάση θα εξετασθεί εστιάζοντας στην βασική λειτουργικότητα που προσφέρει, αυτή δηλαδή της μεταφοράς παραμέτρων στον server μέσω **HTTP Request Parameters** και **HTTP Request Headers**.

Κάποιες μέθοδοι αυτού του interface που έχουν να κάνουν με λεπτομέρειες του configuration κάποιων λειτουργικοτήτων (όπως π.χ. του file upload ή των γλωσσικών προτιμήσεων του χρήστη) θα εξετασθούν μέσα σε άλλα πλαίσια αργότερα αν και η αλήθεια είναι ότι και αυτά τελικά στηρίζονται στα HTTP Request Headers.

### Request Line - Resource URI

Όσον αφορά το **URI του πόρου** στην Request Line αυτό είναι είτε **absolute** είτε **relative** (προς το domain). Στην πλειονότητα των περιπτώσεων είναι relative. Εάν το request γίνεται από client που είναι πίσω από κάποιον **forward proxy** και όχι απευθείας προς το origin server το URI αυτό θα πρέπει να είναι absolute. Εάν γίνεται απευθείας στον origin server μπορεί να είναι relative.

Πρακτικά αυτό ακριβώς και συμβαίνει και επιβεβαιώνεται εύκολα εάν ορίσουμε το σύστημά μας να χρησιμοποιεί κάποιον proxy για την επικοινωνία του μέσω internet (μπορούμε να κάνουμε κάποιο request προς μια ιστοσελίδα και να το εξετάσουμε με κάποια browser network tools τα HTTP πακέτα). Παρόλα αυτά **το RFC 2616 (W3C) ενθαρρύνει τους browser στην χρήση του absolute URL σε όλες τις περιπτώσεις**. Αυτή η προτροπή γενικώς δεν ακολουθείται ακόμα.

To URI του request-line παρέχεται μέσω της μεθόδου `HttpServletRequest.getRequestURI` και, εξαιτίας των προηγουμένων, ο κώδικας που θα την χρησιμοποιεί θα πρέπει να προβλέπει και για τις δύο μορφές που ενδέχεται να επιστραφούν. Επίσης **καλό είναι να μην γίνονται υποθέσεις για το αν το request ήρθε κατευθείαν από τον client του χρήστη ή από κάποιον ενδιάμεσο forward server με βάση την μορφή του address στο Request-Line αφού δεν είναι ποτέ γνωστό σε τι βαθμό ο κάθε browser συμμορφώνεται με τα πρότυπα.**

## HTTP Request Parameters

Το αίτημα του HTTP πρωτοκόλλου που μοντελοποιείται ως ένα `HttpServletRequest` αντικείμενο περιλαμβάνει τις παραμέτρους τις οποίες έστειλε ο χρήστης μέσω ενός HTTP αιτήματος (πιθανότατα **GET** ή **POST**). Αυτές είναι προσβάσιμες κυρίως μέσω των μεθόδων `getParameter`, `getParameterMap`. Οι παράμετροι αυτοί στέλνονται μέσω **HTML Form Elements**, μέσω **Query Strings** στα URLs και μέσω **AJAX αιτήσεων**.

## HTTP Request Headers

### *Interface*

Όσον αφορά τους headers οι οποίοι ενδέχεται να μεταφέρονται μέσω του HTTP αιτήματος αυτοί είναι προσβάσιμοι κυρίως μέσω των μεθόδων `getHeader`, `getIntHeader`, `getDateHeader`. Επίσης υπάρχουν αρκετές μέθοδοι αυτού του interface που ίσως να μην είναι προφανές ότι διαβάζουν τιμές από headers όμως πράγματι κάνουν ακριβώς αυτό. Μερικές από αυτές είναι οι `getCookies`, `getAuthType`, `getCharacterEncoding`, `getContentLength`, `getContentType` κ.α..

Υπάρχουν κάποιοι **προκαθορισμένοι headers** στο HTTP πρωτόκολλο που έχουν συγκεκριμένο σκοπό αλλά επίσης **κάθε εφαρμογή που επικοινωνεί μέσω HTTP μπορεί να ορίσει δικούς της headers**.

Οι headers είναι το δεύτερο τμήμα ενός HTTP Request. Το πρώτο (όπως προαναφέρθηκε) είναι το **Request Line**.

## *Request Headers και HTTP μηχανισμοί*

Κάποια στάνταρ General και Request Headers όπως και κάποια Response Headers (τα Response Headers θα εξετασθούν στην συνέχεια) είναι μέρος κάποιων πρότυπων μηχανισμών των browsers όπως του μηχανισμού authentication/authorization του HTTP πρωτοκόλλου (**BASIC**, ή **DIGEST** HTTP Authentication Methods) ή του File Upload μηχανισμού στο οποίο ουσιαστικά κάποιες απαραίτητες παράμετροι για την εκτέλεση του μηχανισμού θα πρέπει να περνάνε μέσω συγκεκριμένων headers (Content-Disposition, Content-Type).

Επίσης και η λειτουργικότητα των cookies, που υλοποιείται από headers, είναι πολλές φορές ο προτιμώμενος μηχανισμός για την διατήρηση του **conversational state (sessions)** και των **persistent παραμέτρων/ρυθμίσεων/πληροφοριών** που δίνει ο χρήστης στην web εφαρμογή

(δηλαδή πληροφορίες που σώζονται στον δίσκο του χρήστη και διατηρούνται πέραν των ορίων ενός session).

Όλα αυτά βέβαια δεν σημαίνουν ότι ο προγραμματιστής χρειάζεται να γνωρίζει και να ασχολείται με όλες τις λεπτομέρειες αυτών των μηχανισμών και πρακτικά δεν χρειάζεται να διαβάζει τις τιμές αυτών των headers μέσω των μεθόδων τύπου getHeader. **Για την χρήση αυτών των μηχανισμών προσφέρεται ένα πιο απλοποιημένο interface που κρύβει την πολυπλοκότητα της διαχείρισης των headers.**

## Διαφορές μεταξύ Request Headers και Request Parameters

Πρακτικά η διαφορά μεταξύ των HTTP Request Parameters και των Request Headers, εκτός του ότι το HTTP πρωτόκολλο δεν ορίζει στάνταρ HTTP Request Parameters όπως κάνει για τα Request Headers, είναι ότι τα Request Parameters δίνονται **από τον χρήστη προς** την web εφαρμογή (μέσω html φορμών ή AJAX requests, δηλαδή μέσω του user interface ή αλλιώς “view” της web εφαρμογής) ενώ **τα Request Headers υποβάλλονται τυπικά (αλλά όχι απαραίτητα) από τον browser του χρήστη** (δηλαδή της ίδιας της εφαρμογής του browser και όχι της συγκεκριμένης HTML σελίδας που βλέπει ο χρήστης μπροστά του, μέσω HTML φορμών ή AJAX).

Επίσης **τα Request Headers, εκτός των browsers, συνηθίζεται να υποβάλλονται και από ενδιάμεσα συστήματα** μεταξύ του browser και του Application Server της web εφαρμογής όπως π.χ. **Proxy Servers (Forward & Reverse)** και **Web Servers (IIS, Apache ACE)** και προτιμώνται για αυτόν τον σκοπό έναντι των request παραμέτρων. Ένας λόγος που συμβαίνει αυτό είναι για να αποφευχθεί ο κίνδυνος του να γίνει override κάποια παράμετρος μίας εφαρμογής του Application Server από το ενδιάμεσο σύστημα και για να γίνεται διαφοροποίηση μεταξύ των δύο αυτών συνόλων παραμέτρων που επιτελούν διαφορετικούς σκοπούς.

Θα μπορούσαμε να πούμε ότι **τα Request Parameters εξυπηρετούν καλύτερα την business λογική του web application** ενώ **τα Request Headers εκφράζουν κυρίως πράγματα όπως οι προτιμήσεις στο encoding, charset, language, caching, συγκεκριμένους μηχανισμούς για authorization, cookies, content description, client info κ.α..** Δηλαδή γενικά πράγματα που δεν έχουν να κάνουν απαραίτητα με ένα συγκεκριμένο task που θέλει να εκτελέσει ο χρήστης αλλά με γενικές πληροφορίες και προτιμήσεις κάποιες από τις οποίες ορίζονται και μέσω των ρυθμίσεων του browser.

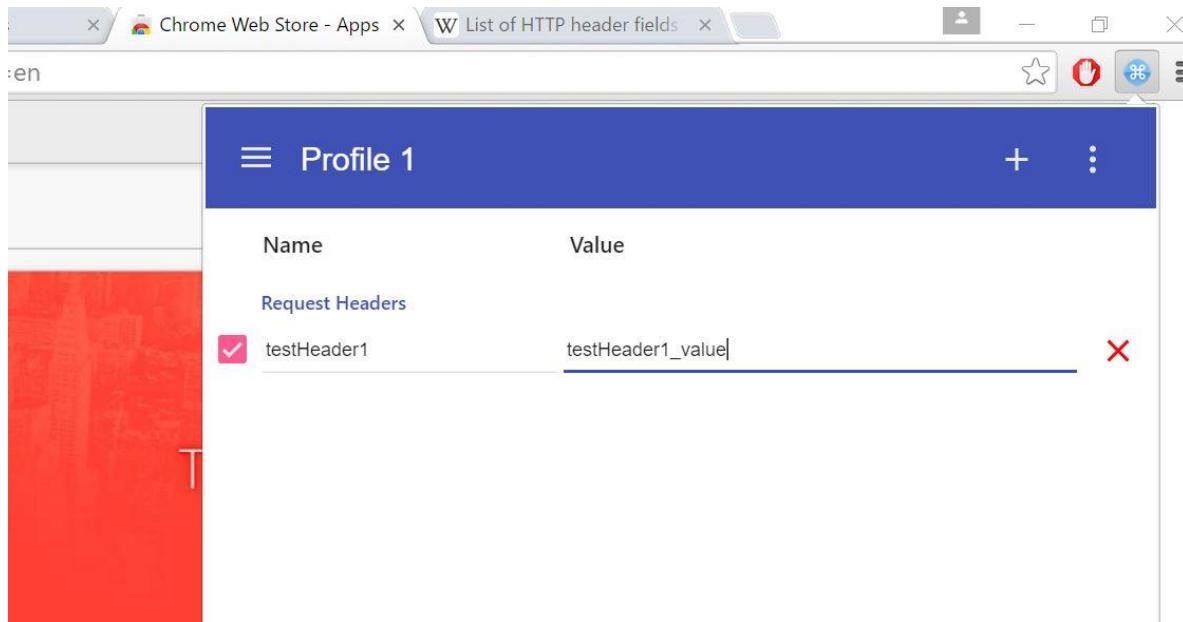
Αυτό δεν σημαίνει ότι τα HTTP Headers παίζουν δευτερεύοντα ρόλο αφού πολλές φορές **μεταφέρουν (λόγω άτυπης σύμβασης) πληροφορίες από ενδιάμεσα συστήματα** όπως Web Servers που υλοποιούν π.χ. SSO μηχανισμούς (Single Sign On, δηλαδή το αυτόματο login σε πολλά domains) όπου το αν ο χρήστης θεωρείται authorized καθορίζεται από headers τους οποίους θέτει το ενδιάμεσο Web Server σύστημα σε κάθε HTTP Request. Δηλαδή **μεταφέρουν πληροφορίες που δεν αφορούν κάποιο συγκεκριμένο task αλλά είναι απαραίτητες για την διεκπεραίωση οποιουδήποτε request.**

Παρόλο που πρακτικά τα HTTP Request Headers θέτονται κυρίως από τον browser του χρήστη ή από ενδιάμεσα συστήματα **υπάρχει και η δυνατότητα να τα θέσουμε και μέσω Javascript σε HTML και js αρχεία** δεν θα εμβαθύνουμε όμως σε αυτό στα πλαίσια της παρούσας εργασίας.

Θεωρητικά τα HTTP Request Parameters θα μπορούσαν να χρησιμοποιηθούν στην θέση των HTTP Request Headers όπως και το ανάποδο για να περαστεί κάποια πληροφορία της web εφαρμογής από τον client στον server αλλά η πρακτική αυτή δεν ακολουθείται ποτέ αφού θα δυσχεραίνει την ανάπτυξη εφαρμογών πάρα πολύ.

## Header Testing - ModHeader

Για τις ανάγκες παρουσίασης των HTTP Request Headers στα πλαίσια αυτής της εργασίας θα χρησιμοποιήσουμε το **ModHeader** plugin στον Chrome μέσω του οποίου μπορούμε να ορίσουμε headers. Το plugin αυτό είναι δωρεάν, η εγκατάσταση γίνεται μέσω του chrome web store και μετά την εγκατάσταση εμφανίζεται στην πάνω δεξιά γωνία. Στην εικόνα επίσης φαίνεται η προσθήκη μίας τεστ τιμής.



## Request Attributes

Ένα HTTP Request ενδέχεται να χρειαστεί να επεξεργαστεί από περισσότερα από ένα Servlets ή JSPs. Για να γίνει αυτό το αντικείμενο τύπου HttpServletRequest θα πρέπει να μεταφερθεί μεταξύ αυτών των Servlets/JSPs, η μεταφορά αυτή γίνεται με τεχνικές forward και include οι οποίες θα αναλυθούν αργότερα στην εργασία. **Για να οριστεί πληροφορία πάνω στο HttpServletRequest αντικείμενο που μεταφέρεται, χρησιμοποιούμε τα Request Attributes.**

Σημειώνεται ότι αναφερόμαστε μόνο στην μεταφορά ενός Instance με τύπο HttpServletRequest εσωτερικά μέσα στο εκτελούμενο web application, από έναν Servlet/JSP σε έναν άλλον. Δεν αναφερόμαστε στην μεταφορά ενός πραγματικού HTTP Request μέσω δικτύου.

Ένας Servlet μπορεί να ορίσει Request Attributes πάνω στο HttpServletRequest αντικείμενο το οποίο δέχτηκε, μέσω της μεθόδου **HttpServletRequest.setAttribute(String attributeName, Object attribute)**. Ο επόμενος Servlet/JSP που θα το δεχτεί μπορεί να διαβάσει τα Request Attributes μέσω των μεθόδων **HttpServletRequest.getAttributeNames()** και **HttpServletRequest.getAttribute(String attributeName)** στο αποτέλεσμα της οποίας θα πρέπει προφανώς να εφαρμοστεί το κατάλληλο downcasting.

# Servlet Response (HttpServletResponse)

Ένα **HTTP Response** αποτελείται από ένα **Status Line**, τα **Response Headers** και το **Message Body**. To interface **HttpServletResponse** προσφέρει μεθόδους για να αποδοθούν τιμές και στα 3 αυτά μέρη.

## Status Line

To **Status Line** δηλώνει το **HTTP Version** που χρησιμοποιείται, το **Status Code** και το **Reason Phrase**. To HTTP Version δηλώνει εάν χρησιμοποιείται το HTTP 1.0 ή το HTTP 1.1. To HTTP 1.0 είναι πρόγονος του HTTP 1.1 και δεν χρησιμοποιείται πια στην συντριπτική πλειονότητα των servers αν και δεν έχει εξαφανιστεί εντελώς αφού είναι λίγο πιο ελαφρύ και υλοποιείται/υποστηρίζεται ευκολότερα από servers και clients.

Tα **Status Codes** και τα **Reason Phrases**, τα οποία είναι μία λεκτική περιγραφή των Status Codes, περιγράφουν αν πέτυχε ή απέτυχε το request ή αν απαιτείται κάποια επιπλέον ενέργεια από τον χρήστη ή τον browser του.

## Status Codes

To Status Code έχει να κάνει με το αν πέτυχε ή απέτυχε το request του χρήστη ή αν χρειάζεται επιπλέον κάποια ενέργεια για να ολοκληρωθεί. Σε γενικές γραμμές τα Status Codes χωρίζονται σε 5 κατηγορίες:

1. **1xx: Informational** - Request received, continuing process
2. **2xx: Success** - The action was successfully received, understood, and accepted
3. **3xx: Redirection** - Further action must be taken in order to complete the request
4. **4xx: Client Error** - The request contains bad syntax or cannot be fulfilled
5. **5xx: Server Error** - The server failed to fulfill an apparently valid request

Πλήρη λίστα των status codes βρίσκεται εδώ <https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html>.

## Tα βασικά και πιο συνήθη status codes

Tα βασικά και πιο συνήθη status codes με τα οποία ασχολείται ένας web developer είναι πολύ λιγότερα σε σύγκριση με αυτά που αναφέρονται στο HTTP πρότυπο του πρωτοκόλλου.

### **2XX**

Statuses σε αυτήν την κατηγορία επιστρέφονται όταν η επεξεργασία του request του user ήταν επιτυχής.

#### **200 (OK)**

To πιο κοινό είναι το status code **200 (OK)** και δηλώνει την επιτυχημένη απάντηση στο request του χρήστη.

### **3XX**

Statuses σε αυτήν την κατηγορία επιστρέφονται όταν ο χρήστης πρέπει να επαναλάβει το request του σε καινούριο URL.

## 301 (Moved Permanently)

Αυτό το status υποδηλώνει ότι το resource που ζήτησε ο χρήστης έχει μετακινηθεί μόνιμα σε άλλη διεύθυνση η οποία και δηλώνεται στον header “**Location**”. Εάν ο browser το υποστηρίζει, θα πρέπει τις επόμενες φορές που ο χρήστης υποβάλει την ίδια διεύθυνση, να τον κατευθύνει αυτόματα στην καινούρια. Εάν το request method δεν ήταν GET πριν γίνει η ανακατεύθυνση θα πρέπει να υποβληθεί ερώτηση στον χρήστη για το αν θέλει να γίνει η ανακατεύθυνση. Το response με αυτόν τον κωδικό θα πρέπει να καταχωρείται στα caches του web εκτός και αν δηλώνεται το αντίθετο από τον origin server.

## 302 (Found), 307 (Moved Temporarily)

Το **302 (Found)**, **307 (Moved Temporarily)** και **303 (See Other)** χρησιμοποιούνται όταν ο server εξαναγκάζει τον φυλλομετρητή του χρήστη να ανακατευθύνει το αρχικό request σε άλλο URL (το οποίο δηλώνεται στο header “**Location**”) ώστε τελικά να μπορεί να εξυπηρετήσει το request του.

Το **status 302** επιστρέφεται στο response που παράγεται από την

**HttpServletResponse.sendRedirect(String)** (από το API των Java Servlets δηλαδή)

χρησιμοποιείται κυρίως όταν υπάρχει η ανάγκη να αλλάξει το URL που βλέπει ο χρήστης στο browser του ώστε να αντιστοιχεί στο resource το οποίο πραγματικά προσπελάζει επειδή πολύ συχνά ενδέχεται το URL που βλέπει ο χρήστης να μην αντιστοιχεί στο resource που του επιστρέφεται εξαιτίας του forwarding των requests (κάτι που θα αναλυθεί αργότερα). Το HTTP πρότυπο δηλώνει ότι ένα response με αυτό το status (302) πρέπει να καταχωρείτε στα web caches μόνο όταν κάτι τέτοιο δηλώνεται ρητά από τα headers Cache-Control ή Expires.

Το **status 302** και το **307** έχουν ακριβώς τον ίδιο ορισμό στο HTTP πρότυπο και το status 302 προϋπήρχε των 307 και 303 όμως επειδή πολλά browsers χειρίζονται (λανθασμένα) το 302 κατά τον ίδιο τρόπο που τώρα είναι ορισμένο το 303 και άλλα όπως είναι τώρα ορισμένο το 307 το HTTP πρότυπο εμπλουτίστηκε με αυτά τα δύο statuses (303, 307).

Ένας browser δεν πρέπει να συγκρατεί το URL στο οποίο μεταφέρθηκε ο χρήστης εξαιτίας ενός **302 ή 307 response** για να το αντικαθιστά αυτόματα με το URL του αρχικού request όπως επιτρέπεται να κάνει στην περίπτωση του status 301.

Όσον αφορά τα web caches αυτά μπορούν να συγκρατούν τα 302 ή 307 responses εάν αυτό δηλώνεται ρητά από τα headers τους.

Τέλος εάν το request method δεν ήταν GET πριν γίνει η ανακατεύθυνση θα πρέπει να υποβληθεί ερώτηση στον χρήστη ώστε να εγκριθεί η ανακατεύθυνση.

## 303 (See Other)

Αυτό το status μοιάζει αρκετά με τα 302 και 307 με την διαφορά ότι το αντίστοιχο response δεν επιτρέπεται να συγκρατείται από τα Web και Browser Caches ασχέτως με το τι δηλώνουν τα headers του. Επίσης ασχέτως με την μέθοδο του αρχικού request του χρήστη το response με αυτό το status υποδηλώνει στον browser του χρήστη να πραγματοποιήσει request με GET στο καινούριο URL.

Η λογική αυτού του status είναι ότι είναι σχεδιασμένο ώστε να χρησιμοποιείται στην περίπτωση που ο χρήστης πραγματοποιήσει κάποιο POST request, οπότε ένα response με status 303 δηλώνει ότι το αίτημά του επεξεργάστηκε και θα πρέπει να πραγματοποιήσει καινούριο request για άλλον πόρο ώστε να δει τα αποτελέσματα του αρχικού του request.

## 304 (Not Modified)

To status αυτό επιστρέφεται εάν κάποιο resource βρίσκεται στην Cache του browser και το request ήταν ένα conditional GET request. Αυτό το response επιστρέφεται δηλαδή εάν ο πόρος δεν έχει αλλάξει και μπορεί να χρησιμοποιηθεί η αναπαράσταση που βρίσκεται στην Cache του browser. Εάν δηλαδή στο δεύτερο request προς τον πόρο χρησιμοποιηθεί το header **If-Modified-Since** και ο server διαπιστώσει ότι ο πόρος που έχει ο client δεν έχει αλλάξει δεν θα του ξαναεπιστρέψει ολόκληρο τον πόρο π.χ. μία εικόνα, αλλά θα του επιστρέψει απλά ένα status 304 το οποίο ο browser το χειρίζεται κατάλληλα παρουσιάζοντας στον χρήστη το ίδιο resource όπως και κατά το πρώτο request.

## 305 (Use Proxy)

Αυτό το response επιστρέφεται εάν το request μπορεί να εξυπηρετηθεί μόνο εφόσον πραγματοποιηθεί μέσω του proxy που ορίζεται στο **Location** header. Εδώ ο όρος proxy αναφέρεται σε έναν **Reverse Proxy**, σε κάποιο μηχάνημα δηλαδή όπως ένας **Web Server** ο οποίος βρίσκεται μπροστά από τον **Application Server** που στεγάζει την εφαρμογή μας και συνήθως προσφέρει υπηρεσίες κρυπτογράφησης, συμπίεσης, caching, load balancing κ.α.. Συχνά αυτό το status επιστρέφεται όταν έχει γίνει λανθασμένο configuration των servers κάποιου οργανισμού και επιτρέπεται να γίνεται άμεση κλήση προς τον Application Server παρόλο που η επικοινωνία θα έπρεπε να γίνεται πάντα μέσω του Web Server.

## 4XX

Statuses αυτής της κατηγορίας επιστρέφονται όταν ο χρήστης πραγματοποίησε **λανθασμένο request** που δεν ικανοποιεί τους περιορισμούς που επιβάλει ο server ή έχει λάθος σύνταξη.

### 400 (Bad Request)

Επιστρέφεται όταν το request έχει λανθασμένη σύνταξη, όταν δηλαδή **δεν είναι συντακτικά έγκυρο σύμφωνα με το HTTP πρωτόκολλο**.

### 401 (Unauthorized)

**Δεν έχει οριστεί η ταυτότητα του χρήστη** και έγινε προσπάθεια προσπέλασης κάποιου προστατευμένου πόρου.

### 403 (Forbidden)

Ο server αρνείται να εξυπηρετήσει το request εξαιτίας κάποιου λόγου. Συνήθως επιστρέφεται όταν ο χρήστης **είναι authorized αλλά παρόλα αυτά τα αναγνωριστικά του δεν έχουν την απαραίτητη εξουσιοδότηση για την προσπέλαση του συγκεκριμένου πόρου**. Πολλές φορές ο server μπορεί να αποφασίσει να μην δίνει την πληροφορία ότι έχει γίνει προσπάθεια προσπέλασης προστατευμένου πόρου για λόγους πρόσθετης ασφάλειας οπότε αντί για το 403 ίσως να επιστραφεί κάποιο άλλο status άσχετο με τον πραγματικό λόγο άρνησης εξυπηρέτησης του request.

### 404 (Not Found)

Είναι το πιο σύνηθες status που επιστρέφεται σε αυτήν την κατηγορία. Δηλώνει ότι **ο server δεν διαθέτει κανέναν πόρο για το URI που δηλώθηκε στο request**. Επίσης συχνά επιστρέφεται στην θέση άλλον statuses σε περίπτωση που ο server δεν επιθυμεί να δημοσιεύσει τον λόγο για τον οποίον αρνήθηκε την εξυπηρέτηση του request.

Statuses σε αυτήν την κατηγορία δηλώνουν αδυναμία εξυπηρέτησης του request του χρήστη **εξαιτίας σφάλματος από την πλευρά του server**.

### 500 (Internal Server Error)

O server συνάντησε ένα **αναπάντεχο σφάλμα** το οποίο τον εμπόδισε να εξυπηρετήσει το request.

### 502 (Bad Gateway)

Κάποιος server ανάμεσα στον client και τον τελικό server, **ενεργώντας ως proxy ή gateway** επέστρεψε αυτό το status επειδή δεν έλαβε σωστή απάντηση από τον τελικό server ή κάποιον ενδιάμεσό τους.

### 503 (Service Unavailable)

Αυτό το status επιστρέφεται εάν **o server αδυνατεί να εξυπηρετήσει το request προσωρινά** αλλά εννοείται ότι μπορεί να ξαναγίνει προσπάθεια από τον χρήστη κάποια στιγμή στο μέλλον. Ιδανικά επιστρέφεται και το header **Retry-After** που υποδηλώνει μετά από πόσο χρόνο θα πρέπει να ξαναπροσπαθήσει ο χρήστης. Αυτό το status θα μπορούσε να επιστραφεί εξαιτίας υπερβολικού φόρτου ή εξαιτίας εργασιών συντήρησης.

### 504 (Gateway Timeout)

Επιστρέφεται από κάποιον ενδιάμεσο server (proxy ή gateway) όταν **δεν έλαβε απάντηση από κάποιον άλλον server που είναι ενδιάμεσος στην επικοινωνία του με τον τελικό, ή από τον τελικό server, μέσα στα επιτρεπτά χρονικά πλαίσια** που έχουν ρυθμιστεί για αυτόν.

## *Status Line Modification Interface*

Για να θέσουμε τιμές στο **Status Line του HTTP Response** το **HttpServletResponse API** μας δίνει αυτές τις μεθόδους:

**sendError(int sc):** Η μέθοδος αυτή χρησιμοποιείται όταν προκύπτει κάποιο αναπάντεχο σφάλμα από την πλευρά του Web Application. Εάν χρησιμοποιείται αποκλειστικά σε κάθε περίπτωση που ο χρήστης θα πρέπει να ενημερωθεί για κάποιο σφάλμα η διαχείριση σφαλμάτων θα μπορεί να ρυθμίζεται από τον deployment descriptor. **Μπορεί να καταλήξει σε αποστολή ενός response με Status Code τύπου 4xx ή 5xx στον χρήστη ή να μετατραπεί σε ένα κανονικό response (status 200) που θα ενημερώνει τον χρήστη για το σφάλμα πιθανότατα στέλνοντάς τον σε κάποιο error page.** Αυτό εξαρτάται από το configuration στον deployment descriptor. Η μέθοδος αυτή στέλνει το response προς τον χρήστη (**response commit**).

**sendRedirect(String location):** Η μέθοδος αυτή **Θέτει το status του response σε 302 (Found)** και τροποποιεί την παράμετρο location σε ένα absolute URL (εάν δεν ήταν ήδη) και **Θέτει κατάλληλα το αντίστοιχο header (Location)**. Με λίγα λόγια **υποχρεώνει τον browser του χρήστη να ανακατευθύνει τον χρήστη στο URL που δόθηκε ως παράμετρος**. Ο χειρισμός του caching (εάν χρειάζεται) θα πρέπει να γίνει εφαρμόζοντας επιπλέον άλλες μεθόδους στο HttpServletResponse. Η μέθοδος αυτή στέλνει το response προς τον χρήστη (**response commit**).

**setStatus(int sc):** Η μέθοδος αυτή απλά **Θέτει το Status Code στο Response Line του HTTP Response** χωρίς να έχει ως αποτέλεσμα την αποστολή του response προς τον χρήστη.

## Response Headers

Υπάρχουν κάποιοι **προκαθορισμένοι response headers** στο HTTP πρωτόκολλο που έχουν συγκεκριμένο σκοπό αλλά επίσης **κάθε εφαρμογή που επικοινωνεί μέσω HTTP μπορεί να ορίσει δικούς της headers**.

## Interface

Όσον αφορά τους headers οι οποίοι ενδέχεται να μεταφέρονται μέσω της HTTP απάντησης αυτοί μπορούν να προστεθούν μέσω των μεθόδων **addHeader**, **addIntHeader**, **addDateHeader** (εάν προστεθεί header που υπάρχει ήδη η προηγούμενη τιμή δεν επαναεγγράφεται αλλά ορίζεται header με πολλαπλές τιμές). Αντίστοιχες μέθοδοι οι οποίες κάνουν επανεγγραφή πάνω σε ήδη υπάρχουσες τιμές ή τις προσθέτουν εάν δεν προϋπήρχαν είναι οι **setHeader**, **setIntHeader**, **setDateHeader**. Επίσης υπάρχουν αρκετές μέθοδοι αυτού του interface που ίσως να μην είναι προφανές ότι θέτουν τιμές σε headers όμως πράγματι κάνουν ακριβώς αυτό. Αυτές είναι οι **addCookie**, **setLocale**, **setCharacterEncoding**, **setContentLength** και **setContentType**.

Πρέπει να τονισθεί ότι οι μέθοδοι **getHeader**, **getIntHeader**, **getDateHeader**, **getHeaders**, **getHeaderNames** επιστρέφουν μόνο τους headers που έχουν προστεθεί με τις αντίστοιχες get και add μεθόδους και δεν θα επιστρέψουν π.χ. τον header που σχηματίζεται με βάση της **setContentType** και **setCharacterEncoding** αλλά ούτε και επιπλέον headers τους οποίους ενδέχεται να προσθέσει ο server. **Γενικά το API των Servlets δεν προσφέρει κάποιον τρόπο για την εξέταση όλων των headers που θα σταλούν τελικά στον χρήστη.** Συνήθως χρησιμοποιούνται τρίτα εργαλεία ανάλυσης της δικτυακής κίνησης για αυτόν τον σκοπό.

## Response Headers και HTTP Μηχανισμοί

**Κάποια στάνταρ response headers είναι μέρος κάποιων πρότυπων μηχανισμών των browsers** όπως του μηχανισμού authentication/authorization του http πρωτοκόλλου (BASIC ή DIGEST HTTP Authentication Methods).

Επίσης και **η λειτουργικότητα των cookies**, που υλοποιείται από headers, είναι πολλές φορές ο προτιμώμενος μηχανισμός για την διατήρηση του **conversational state (sessions)** και των **persistent παραμέτρων/ρυθμίσεων/πληροφοριών** που δίνει ο χρήστης στην web εφαρμογή (δηλαδή πληροφορίες που σώζονται στον δίσκο του χρήστη και διατηρούνται πέραν των ορίων ενός session). Πολλές φορές οι τιμές αυτών των cookies ορίζονται από τον server που λέει στον browser του χρήστη τι να αποθηκεύσει ώστε ο browser να τα ξαναστείλει πίσω στον server, στο μέλλον, όταν ο χρήστης ξαναεπισκεφθεί την σελίδα.

## Response Body (Message Body)

## Writer & Output Stream

Για να ορίσουμε το **Message Body του response** (χωρίς τους headers) χρησιμοποιούμε τον **PrintWriter** που μας δίνεται από την μέθοδο **HttpServletResponse.getWriter()** ή το **ServletOutputStream** από την μέθοδο **HttpServletResponse.getOutputStream**. Δεν επιτρέπεται η ταυτόχρονη γραφή και στα δύο αντικείμενα, ο προγραμματιστής θα πρέπει να διαλέξει ένα από αυτά και να το χρησιμοποιεί σε όλον τον κώδικα που χειρίζεται το συγκεκριμένο instance του response αντικειμένου. Υστερα χρησιμοποιούμε το interface αυτό όπως θα κάναμε π.χ. και στην περίπτωση που γράφαμε σε ένα text αρχείο στο file system με την διαφορά ότι **η μέθοδος flush**

**αυτού του stream/writer στέλνει το response προς τον browser του πελάτη** αντί για τον σκληρό δίσκο.

### ***Ορισμός του Encoding***

Εάν δεν οριστεί το encoding του response (π.χ. μέσω της μεθόδου

**HttpServletResponse.setCharacterEncoding(String charset)** σε συνδυασμό με την **HttpServletResponse.setContentType(String contentType)**) το PrintWriter αντικείμενο θα έχει προκαθορισμένο το encoding **ISO-8859-1** (8 bit λατινικοί χαρακτήρες). Εάν χρησιμοποιηθεί άλλο encoding αυτό θα πρέπει να δηλωθεί **πριν** αρχίσουμε να γράφουμε το οτιδήποτε στον PrintWriter.

Επίσης θα πρέπει να σημειωθεί ότι **το encoding μίας HTML σελίδας ορίζει και το encoding που θα εφαρμόσει ο browser του χρήστη στα URIs** τα οποία ο χρήστης εισάγει στην γραμμή διεύθυνσης ή στα οποία γενικά αυτός μεταβαίνει και συνεπώς με αυτό το encoding κωδικοποιούνται τα URIs στο Request Line του HTTP Request στα requests που θα στέλνονται προς τον server.

### ***Response Buffer***

Έχουμε την επιλογή πριν πραγματοποιήσουμε το πρώτο write να ορίσουμε και το **buffer size** του response καλώντας την **HttpServletResponse.setBufferSize(int size)** (το Servlet API ορίζει ότι θα χρησιμοποιηθεί buffer ίσο η μεγαλύτερο από το δηλωθέν).

**Σημειώνεται ότι το Output Stream, το Writer αντικείμενο και το Response Buffer αναφέρονται όλα στο Message Body του response που δεν περιλαμβάνει τους headers.** Οι headers ορίζονται ξεχωριστά από τις μεθόδους που αναφέρθηκαν στο προηγούμενο υποκεφάλαιο.

### ***Response Commit***

Αφού τελειώσουμε με τον καθορισμό των headers και των περιεχομένων του response (response body), αυτό θα πρέπει να σταλθεί προς τον client. Αυτό γίνεται αυτόματα εάν **τα περιεχόμενα του buffer του response έχουν ξεπεράσει το μέγιστο μέγεθος** ή μετά το τέλος της εκτέλεσης της μεθόδου **HttpServlet.service** (είναι η μέθοδος που καλεί την doGet, doPost κ.τ.λ. ανάλογα με τον τύπο του HTTP Method που δηλώνεται στο HTTP Request). Σημειώνεται ότι **αυτό συμβαίνει πριν επιστρέψουν οι μέθοδοι doFilter** των Filters που ενδέχεται να έχουμε ορίσει πάνω στον πόρο (θα εξετασθούν αργότερα).

Για την αποστολή του response προς τον client πριν τις προηγούμενες περιπτώσεις ο προγραμματιστής μπορεί να καλέσει την μέθοδο **HttpServletResponse.flushBuffer** ή την μέθοδο **flush** στο **Writer** ή **OutputStream** αντικείμενο που έχει λάβει από τις **HttpServletResponse.getWriter** ή **HttpServletResponse.getOutputStream**.

Επίσης comit του response προκαλούν και οι μέθοδοι **HttpServletResponse.sendError** και **HttpServletResponse.sendRedirect** όπου το ένα αντιστοιχεί στην εμφάνιση μίας error page και το άλλο στην ανακατεύθυνση του client σε άλλο uri.

**Μετά το commit του response δεν επιτρέπεται να ξαναγραφτούν δεδομένα σε αυτό** και να γίνει commit εκ νέου για αυτό και οποιαδήποτε μέθοδος που έχει ως αποτέλεσμα, άμεσα ή έμμεσα, την γραφή δεδομένων και headers στο response θα προκαλέσει exception ή θα αγνοηθεί.

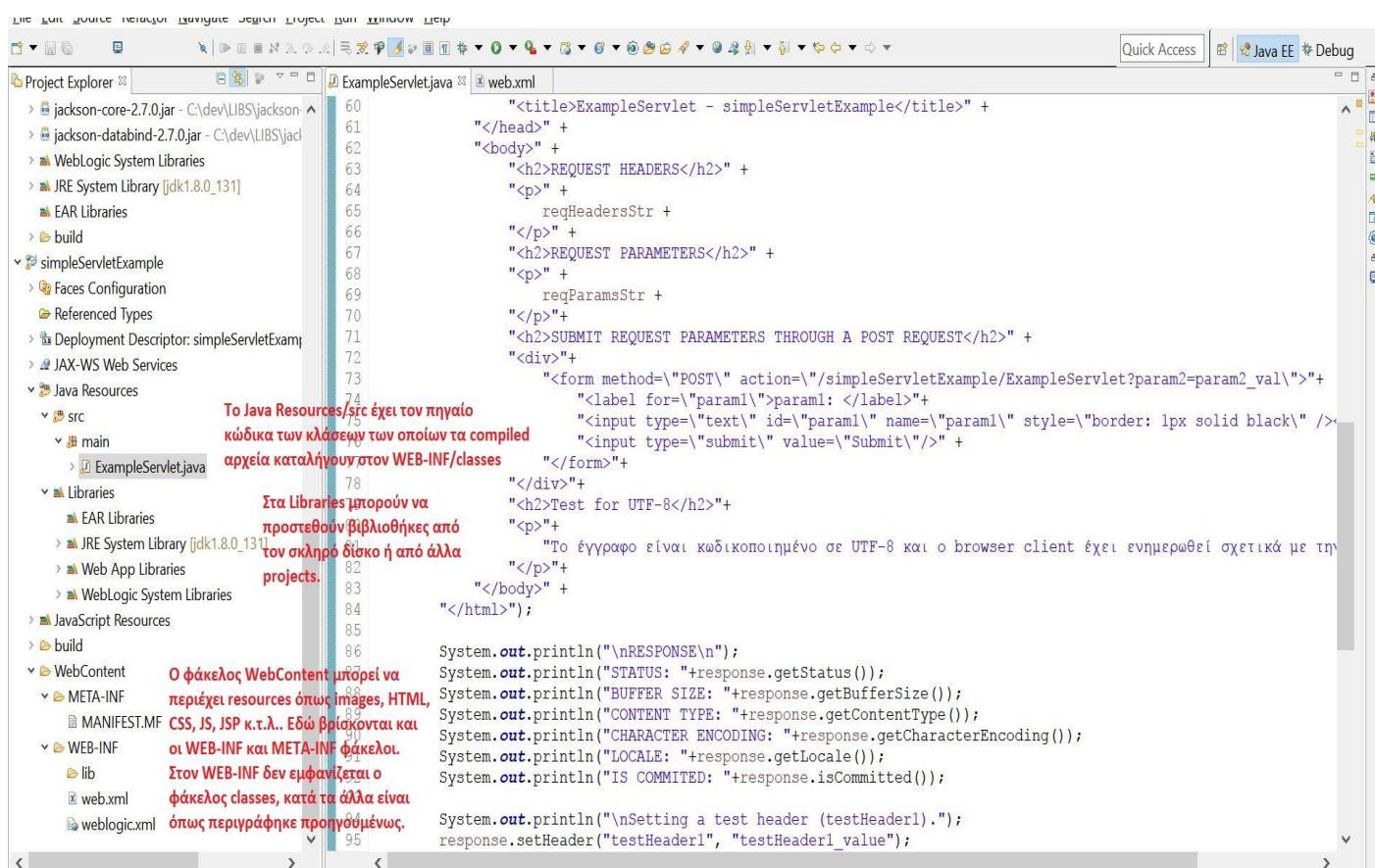
## Πρακτική Εφαρμογή

## Project Development μέσω του Eclipse

Για την πρακτική εφαρμογή των εννοιών που εξετάστηκαν προηγουμένως θα παρουσιαστούν σε ένα project στο **IDE Eclipse Kepler**. Το project αυτό ονομάστηκε **“simpleServletExample”** και δημιουργήθηκε στο domain **“demoWebApp”**.

Αρχικά θα δημιουργήσουμε ένα απλό Servlet που θα λαμβάνει και θα επιστρέφει παραμέτρους και headers που θα του στέλνει ο χρήστης. Στην συνέχεια θα εισάγουμε το Servlet στον deployment descriptor και τελικώς θα γίνει deployment και εκτέλεση της εφαρμογής μέσω του Eclipse στον Weblogic Application Server.

Θα παρατηρήσουμε ότι το Project View που βλέπουμε στον Eclipse δεν συμβαδίζει με το Java Web Application Directory Structure στο οποίο αναφερθήκαμε προηγουμένως. Στην επόμενη εικόνα σημειώνονται κάποια πράγματα για την αντιστοιχία μεταξύ αυτών. Σε κάθε περίπτωση το Eclipse θα παράγει και θα κάνει deploy το project στον Weblogic στην μορφή που περιγράψαμε προηγουμένως (στα κεφάλαια Application Packaging, Web Module – Directory Structure) ασχέτως με το πως εμφανίζεται στην φάση του development.



## ExampleServlet.java

Στην συνέχεια παρουσιάζεται ο κώδικας του ExampleServlet της εφαρμογής **simpleServletExample** που δέχεται **παραμέτρους** και **Request Headers**, τους καταγράφει και επιστρέφει ως απάντηση μία HTML σελίδα που εμφανίζει τις τιμές που είχαν δοθεί. Ο κώδικας

χειρίζεται κατά τον ίδιο τρόπο και GET αλλά και POST requests αφού η doPost μέθοδος απλώς καλεί την doGet με τις παραμέτρους που της έχουν περαστεί.

```
1 package main;
2
3 import java.io.IOException;
4
5
6 public class ExampleServlet extends HttpServlet {
7
8     WebUtils webUtils = new WebUtilsImpl();
9
10
11     public ExampleServlet() {
12         super();
13     }
14
15
16     protected void doGet(HttpServletRequest request, HttpServletResponse response)
17         throws ServletException, IOException {
18
19         System.out.println("ExampleServlet.doGet: started ...");
20
21         // PRINT ALL THE REQUEST HEADERS
22         String reqHeadersStr = webUtils.getStringWithAllTheRequestHeaders(request, "\n");
23         System.out.println("REQUEST HEADERS");
24         System.out.println(reqHeadersStr);
25
26         // PRINT ALL THE REQUEST PARAMETERS
27         String reqParamsStr = webUtils.getStringWithAllTheRequestParameters(request, "\n");
28         System.out.println("REQUEST PARAMETERS");
29         System.out.println(reqParamsStr);
30
31         // CREATE THE RESPONSE
32
33         // SET HEADERS
34
35         response.addHeader("testResponseHeader", "testResonseHeader_val");
36
37         // CREATE RESPONSE
38
39         // SET THE RESPONSE ENCODING
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
```

response.setContentType("text/html"); // From the docs: Note that the character encoding cannot  
// be communicated via HTTP headers if the servlet does not specify a content type.  
response.setCharacterEncoding("UTF-8");  
  
// CREATE THE RESPONSE BODY  
  
PrintWriter resOut = response.getWriter(); // By default in ISO 8859-1 encoding.  
reqHeadersStr = webUtils.getStringWithAllTheRequestHeaders(request, "<br/><br/>");  
reqParamsStr = webUtils.getStringWithAllTheRequestParameters(request, "<br/><br/>");  
resOut.write(  
"<html>" +  
 "<head>" +  
 "<title>ExampleServlet - simpleServletExample</title>" +  
 "</head>" +  
 "<body>" +  
 "<h2>REQUEST HEADERS</h2>" +  
 "<p>" +  
 reqHeadersStr +  
 "</p>" +  
 "<h2>REQUEST PARAMETERS</h2>" +  
 "<p>" +  
 reqParamsStr +  
 "</p>" +  
 "<h2>SUBMIT REQUEST PARAMETERS THROUGH A POST REQUEST</h2>" +  
 "<div>" +  
 "<form method=\"POST\" action=\"/simpleServletExample/ExampleServlet?param2=param2\_val\">" +  
 "<label for=\"param1\">param1: </label>" +  
 "<input type=\"text\" id=\"param1\" name=\"param1\" style=\"border: 1px solid \" +  
 "black\" /><br/><br/>" +  
 "<input type=\"submit\" value=\"Submit\" />" +  
 "</form>" +  
 "</div>" +  
 "<h2>Test for UTF-8</h2>" +  
 "<p>" +  
 "Το έγγραφο είναι κωδικοποιημένο σε UTF-8 και ο browser client έχει ενημερωθεί " +  
 "με την κωδικοποίηση." +

```

84             "</p>" +
85             "</body>" +
86             "</html>");

87
88         System.out.println("\nRESPONSE\n");
89         System.out.println("STATUS: " + response.getStatus());
90         System.out.println("BUFFER SIZE: " + response.getBufferSize());
91         System.out.println("CONTENT TYPE: " + response.getContentType());
92         System.out.println("CHARACTER ENCODING: " + response.getCharacterEncoding());
93         System.out.println("LOCALE: " + response.getLocale());
94         System.out.println("IS COMMITED: " + response.isCommitted());

95
96         System.out.println("\nSetting a test header (testHeader1).");
97         response.setHeader("testHeader1", "testHeader1_value");
98
99         // PRINT ALL THE RESPONSE HEADERS
100        System.out.println("\nRESPONSE HEADERS (before committing the response)\n");
101        String resHeadsStr = webUtils.getStringWithAllTheResponseHeaders(response, "\n");
102        System.out.println(resHeadsStr);
103
104        // Flush/commit the response.
105        response.flushBuffer();
106        System.out.println("IS COMMITED (after flushBuffer): " + response.isCommitted());
107
108        System.out.println("\nRESPONSE HEADERS (after committing the response)\n");
109
110        resHeadsStr = webUtils.getStringWithAllTheResponseHeaders(response, "\n");
111        System.out.println(resHeadsStr);
112
113    }
114
115
116@    protected void doPost(HttpServletRequest request, HttpServletResponse response)
117        throws ServletException, IOException {
118
119        System.out.println("ExampleServlet.doPost: started ... just calls doGet ...");
120
121        doGet(request, response);
122    }
123
124}
125
```

## Request Parameters and Headers

Στις γραμμές **30** έως **37** καταγράφουμε τα Request Headers και Request Parameters στο default output stream του συστήματος. Στην περίπτωση που ο server τρέχει μέσα από το Eclipse αυτό είναι η κονσόλα του Eclipse. Εάν τον ξεκινούσαμε εκτός του Eclipse από μία γραμμή εντολών, το output θα εμφανίζονταν σε αυτήν. Η καταγραφή μπορεί επίσης να γίνεται στο αρχείο **%DOMAIN\_HOME%\servers\AdminServer\logs\AdminServer.log** εάν έχουμε κάνει τις κατάλληλες ρυθμίσεις στην admin console. Για να τις κάνουμε μεταφερόμαστε στο **Environment > Servers > AdminServer > Logging > Advanced** και τικάρουμε τις επιλογές «**Redirect stdout logging enabled**» και «**Redirect stderr logging enabled**».

Για να φτιάξουμε ένα String με όλα τα Request Headers χρησιμοποιούμε την μέθοδο **WebUtilsImpl.getStringWithAllTheRequestHeaders** που έχει οριστεί στην κλάση **osotnikov.web.utils.WebUtilsImpl** του project **Osotnikov\_Web\_Libs**. Ο κώδικας της παρουσιάζεται παρακάτω:

```

3* import java.io.IOException;
13
14 public class WebUtilsImpl implements WebUtils {
15
16*  /* (non-Javadoc)
17*   * @see osotnikov.web.utils.WebUtils#getStringWithAllTheRequestHeaders(javax.servlet.http.HttpServletRequest, String delimiter)
18*   */
19*  @Override
20  public String getStringWithAllTheRequestHeaders(HttpServletRequest request, String delimiter) {
21
22      String result = "";
23
24      Enumeration<String> headerNames = request.getHeaderNames();
25      String headerName, headerValue;
26
27      while(headerNames.hasMoreElements()) {
28          headerName = headerNames.nextElement();
29          headerValue = request.getHeader(headerName);
30          result += headerName + " : " + headerValue + delimiter;
31
32      }
33
34      return result;
35  }
36

```

Όπως βλέπουμε είναι πολύ απλή. Το `HttpServletRequest` αντικείμενο μας παρέχει ένα enumeration με τα ονόματα όλων των Request Headers με την μέθοδο `HttpServletRequest.getHeaderNames()`. Έχοντας το όνομα ενός Request Header μπορούμε να πάρουμε την τιμή του καλώντας την συνάρτηση `HttpServletRequest.getHeader(String)`.

Αντίστοιχος είναι και ο κώδικας της `HttpServletRequest.getStringWithAllTheRequestParameters()`. Παίρνουμε τα ονόματα όλων των request παραμέτρων με την μέθοδο `HttpServletRequest.getParameterNames()`. Παίρνουμε την τιμή της παραμέτρου μέσω της μεθόδου `HttpServletRequest.getParameter(String)`.

```

26  public static String getStringWithAllTheRequestParameters(HttpServletRequest request, String delimiter) {
27
28      String result = "";
29
30      Enumeration<String> parameterNames = request.getParameterNames();
31      String parameterName, parameterValue;
32
33      while(parameterNames.hasMoreElements()) {
34          parameterName = parameterNames.nextElement();
35          parameterValue = request.getParameter(parameterName);
36          result += parameterName + " : " + parameterValue + delimiter;
37
38      }
39
40      return result;
41  }
42

```

Οι μέθοδοι `WebUtilsImpl.getStringWithAllTheRequestHeaders(HttpServletRequest request, String delimiter)` και `WebUtilsImpl.getStringWithAllTheRequestParameters(HttpServletRequest request, String delimiter)` παίρνουν ως δεύτερη παράμετρο το delimiter άρα μπορούμε να τις χρησιμοποιήσουμε για να καταγράψουμε τα parameters και headers σε αρχείο καταγραφής όπου θα θέσουμε το delimiter σε newline ενώ όταν αργότερα θα χρησιμοποιήσουμε αυτά τα Strings μέσα στο HTML που παράγει το Servlet θα αλλάξουμε το delimiter σε "<br/>" που είναι η αλλαγή γραμμής στα HTML Documents.

## *Response Headers*

Θέτουμε ένα Response Header στην γραμμή 97:

```
response.addHeader("testHeader1", "testHeader1_value");
```

Αργότερα θα δούμε ότι φτάνει στον browser του χρήστη και φαίνεται ανοίγοντας την κονσόλα του και εξετάζοντας το response.

## *Response Body*

Για να δημιουργήσουμε το response body αρχικά πρέπει να καθορίσουμε τον τύπο των δεδομένων του. **Στην περίπτωση του MIME τύπου “text/html” θα πρέπει να καθορίσουμε και το encoding που θα χρησιμοποιηθεί εάν θέλουμε να εμφανίσουμε μη λατινικούς χαρακτήρες διότι το προεπιλεγμένο encoding είναι το ISO 8859-1 που έχει μόνο λατινικούς χαρακτήρες.**

Στην γραμμή 48 θέτουμε το content type:

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

Στην γραμμή 50 θέτουμε το character encoding:

```
response.setCharacterEncoding("UTF-8");
```

**Δεν πρέπει να παραλείπουμε να ορίζουμε το Content Type διότι το Character Encoding ορίζεται στον ίδιο header με αυτό στο HTTP πρωτόκολλο. Εάν δεν οριστεί η μέθοδος setCharacterEncoding δεν θα έχει κανένα αποτέλεσμα και οι μη λατινικοί χαρακτήρες δεν θα εμφανίζονται σωστά.**

Για να ορίσουμε τα δεδομένα του response θα πρέπει να το κάνουμε μέσω του PrintWriter αντικειμένου από το **HttpServletResponse** (γραμμή 54) και να γράψουμε σε αυτό τα textual δεδομένα μας όπως θα κάναμε και άμα γράφαμε σε κάποιο αρχείο του file system.

Αφού πάρουμε τα String αντικείμενα με όλα τα request headers και τα request parameters με τις μεθόδους **getStringWithAllTheRequestHeaders** και **getStringWithAllTheRequestParameters** που παρουσιάστηκαν προηγουμένως μπορούμε να φτιάξουμε το body του response (γραμμές 57 έως 86).

Στις γραμμές 63 έως 70 παρουσιάζουμε τα Request Headers και Parameters. Στις γραμμές 71 έως 79 ορίζεται η φόρμα την οποία μπορεί να υποβάλει ο χρήστης στο ίδιο Servlet. Κάτι τέτοιο θα έχει ως αποτέλεσμα ένα POST request πάλι προς τον ExampleServlet ώστε οι παράμετρος που ορίστηκε μέσα στην φόρμα από τον χρήστη να εμφανιστεί στην παράγραφο **REQUEST HEADERS** μαζί με την παράμετρο param2 της οποίας η τιμή δηλώνεται μέσα στον κώδικα και τοποθετείται στο query string του URL που βρίσκεται στο action attribute της φόρμας.

Στην γραμμή 82 έχουμε ένα κείμενο στα ελληνικά για να δοκιμαστεί εάν δουλεύει το UTF-8 encoding που ορίσαμε.

## Deployment descriptor - web.xml

Ορίζουμε έναν **deployment descriptor** για το web application μας. Ο μόνος σκοπός του είναι να καταγράψει τον μέχρι τώρα μοναδικό servlet της εφαρμογής (**ExampleServlet.java**).

Για να μπορεί ένας Servlet να εξυπηρετήσει requests θα πρέπει να έχει οριστεί στον **deployment descriptor**. Ορίζεται δηλώνοντας το element **<servlet>** δίνοντας του ένα όνομα και προσδιορίζοντας την κλάση του μαζί με το package στο οποίο βρίσκεται. Στην συνέχεια θα πρέπει να οριστεί ένα ακόμα element, το **<servlet-mapping>** ώστε να αντιστοιχίσουμε τον servlet σε ένα **URL pattern**. Αντιστοιχίζοντας τον **ExampleServlet.java** στο URL pattern ExampleServlet ορίζουμε ότι όλα τα requests προς τα URLs αυτής της μορφής:

**scheme://domain\_name:port/context\_root/InitializationServlet[optional\_query\_string]** (όπου το optional\_query\_string αναφέρεται σε οποιοδήποτε query string προκύπτει από την υποβολή τυχών request parameters) θα καταλήγουν στην ExampleServlet κλάση.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/x
3
4     <display-name>simpleServletExample</display-name>
5
6     <welcome-file-list>
7         <welcome-file>ExampleServlet</welcome-file>
8     </welcome-file-list>
9
10    <servlet>
11        <servlet-name>ExampleServlet</servlet-name>
12        <servlet-class>
13            main.ExampleServlet
14        </servlet-class>
15    </servlet>
16    <servlet-mapping>
17        <servlet-name>ExampleServlet</servlet-name>
18        <url-pattern>
19            ExampleServlet
20        </url-pattern>
21    </servlet-mapping>
22 </web-app>
```

## Runtime descriptor - weblogic.xml

To **context\_root** κομμάτι του URL

(**scheme://domain\_name:port/context\_root/ExampleServlet/any\_string**) είναι το πρόθεμα του URL path το οποίο πρέπει να είναι μοναδικό για κάθε web application. Το Java EE πρότυπο δεν ορίζει το που θα πρέπει να το ορίσει ο προγραμματιστής οπότε είναι στην ευχέρεια της κάθε υλοποίησης ενός application server να δηλώσει το που ορίζεται. Στην περίπτωση του Weblogic server αυτό ορίζεται στον **runtime descriptor** δηλαδή στο αρχείο **weblogic.xml**.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wls:weblogic-web-app xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app">
3     <wls:weblogic-version>12.2.1</wls:weblogic-version>
4     <wls:context-root>simpleServletExample</wls:context-root>
5 </wls:weblogic-web-app>
```

Στην εικόνα φαίνεται ότι το element που ορίζει το context root είναι το **<wls:context-root>**. Για απλότητα το context root ορίστηκε να είναι ίδιο με το όνομα του project στο Eclipse αν και κάτι τέτοιο δεν είναι απαραίτητο. Ένα βασικό runtime descriptor δεν χρειάζεται να περιέχει τύποτα παραπάνω από αυτά που φαίνονται στην εικόνα. Σε αυτό το σημείο απαιτείται προσοχή, **καλό θα ήταν να ελεγχθεί μετά το build εάν το Eclipse έχει ενημερωθεί για την αλλαγή στο context**

**root της εφαρμογής.** Για να το κάνουμε αυτό πατάμε δεξί κλικ πάνω στο project στο αριστερό πάνελ, κλικάρουμε το “Properties”, μεταφερόμαστε στο “Web Project Settings” και ελέγχουμε το “Context root” και το διορθώνουμε εάν υπάρχει ανάγκη.

### Πρώτη εκτέλεση του simpleServletExample Application

Αφού ξεκινήσουμε τον server μέσα από το περιβάλλον του Eclipse μπορούμε να χτυπήσουμε το URL <http://localhost:7001/simpleServletExample/ExampleServlet> που θα έχει ως αποτέλεσμα ένα GET Request χωρίς παραμέτρους προς το ExampleServlet της εφαρμογής μας. Στην παρακάτω εικόνα φαίνεται το response ενός τέτοιου request.



## REQUEST HEADERS

Host : localhost:7001

Connection : keep-alive

Upgrade-Insecure-Requests : 1

User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36

Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8

Accept-Encoding : gzip, deflate, br

Accept-Language : en,el;q=0.8,ru;q=0.6

Cookie : JSESSIONID=MwZHnhje72Zv2uDLBqV\_ljrzhiu4iIQyCpMbYvSAMAsaPrM57M3g!49954823

testHeader1 : testHeader1\_value

## REQUEST PARAMETERS

### SUBMIT REQUEST PARAMETERS THROUGH A POST REQUEST

param1:

### Test for UTF-8

Το έγγραφο είναι κωδικοποιημένο σε UTF-8 και ο browser client έχει ενημερωθεί σχετικά με την κωδικοποίηση.

Κάτω από την κεφαλίδα **REQUEST HEADERS** φαίνονται τα headers τα οποία στάλθηκαν από τον browser μας κατά το request. Όλοι οι headers που φαίνονται στην εικόνα (εκτός από τον τελευταίο ο οποίος όπως αναφέρθηκε και προηγουμένως έχει οριστεί από το ModHeader Plugin του Chrome) είναι στάνταρ headers που στέλνονται από κάθε browser. Στην πληθώρα των περιπτώσεων ο προγραμματιστής των web applications σε Java δεν χρειάζεται να ασχολείται σε βάθος με αυτά καθώς ως έναν βαθμό τα διαχειρίζεται και ο application server.

Κάτω από την κεφαλίδα **REQUEST PARAMETERS** δεν εμφανίζεται τίποτα διότι δεν έχουμε ορίσει κάποιες παραμέτρους ούτε στο query string του request URL αλλά ούτε και στο message body του.

Κάτω από την κεφαλίδα **Test for UTF-8** φαίνεται το κείμενο στα ελληνικά που υπάρχει για να ελέγχουμε ότι οι μέθοδοι **HttpServletRequest.setContentType** και **HttpServletRequest.setCharacterEncoding** δουλέψανε. Αυτό όντως ισχύει διότι το κείμενο εμφανίζεται σωστά και στα response headers περιλαμβάνουν τον header **Content-Type:text/html; charset=UTF-8**.

Τα **Request και Response Headers** μπορούμε να εξετάσουμε πατώντας **Ctrl + Shift + J** στον Chrome το οποίο ανοίγει την κονσόλα του browser. Εάν μεταβούμε στο **tab Network** και κλικάρουμε πάνω στο request “InitializationServlet” μπορούμε να τα δούμε.

Name		X Headers Preview Cookies Timing
ExampleServlet		<p><b>General</b></p> <p>Request URL: http://localhost:7001/simpleServletExample/ExampleServlet      Request Method: GET      Status Code: 200 OK      Remote Address: [::1]:7001      Referrer Policy: no-referrer-when-downgrade</p> <p><b>Response Headers</b> <a href="#">view source</a></p> <p>Content-Type: text/html; charset=UTF-8      Date: Sat, 15 Jul 2017 21:14:56 GMT      testHeader1: testHeader1_value      testResponseHeader: testResonseHeader_val      Transfer-Encoding: chunked</p> <p><b>Request Headers</b> <a href="#">view source</a></p> <p>Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8      Accept-Encoding: gzip, deflate, br      Accept-Language: en,el;q=0.8,ru;q=0.6      Cache-Control: max-age=0      Connection: keep-alive      Cookie: JSESSIONID=MwZHnhje72Zv2uDLBqv_ljrzhiu4iIQyCpMbYvSAMasaPrM57M3g!49954823      Host: localhost:7001      testHeader1: testHeader1_value      Upgrade-Insecure-Requests: 1      User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36</p>
inject.js		

## Υποβολή Request Parameters

Στην παρακάτω εικόνα φαίνεται ότι συμπληρώνουμε το param1 textbox της φόρμας προετοιμάζοντας κατ' αυτόν τον τρόπο το value της αντίστοιχης παραμέτρου. Επίσης εάν εξετάσουμε την HTML της φόρμας με το **Inspector** του Chrome (**Ctrl + Shift + I** και στην συνέχεια κλικ στο αντικείμενο του textbox) βλέπουμε ότι στο URL του action attribute υπάρχει Query String που ορίζει ένα request parameter και εκεί. Πατώντας submit θα στείλουμε και τις δύο παραμέτρους προς τον servlet. Όλα αυτά παρουσιάζονται παρακάτω:

**Request:**

SUBMIT REQUEST PARAMETERS THROUGH A POST REQUEST

param1:

Submit

### Test for UTF-8

Το έγγραφο είναι κωδικοποιημένο σε UTF-8 και ο browser client έχει ενημερωθεί σχετικά με την κωδικοποίηση.



**Response:**

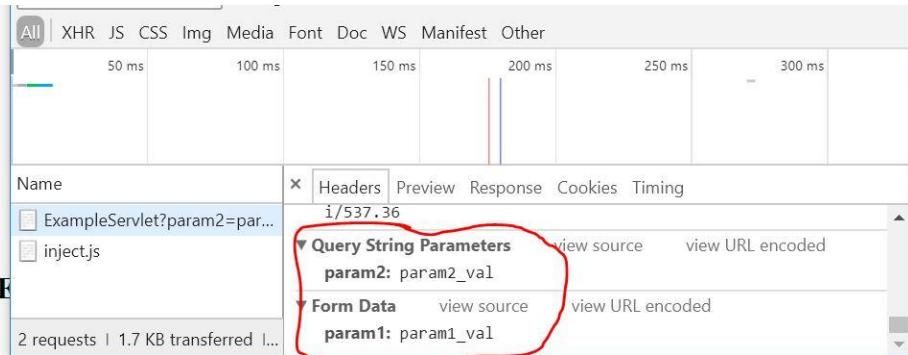
testHeader1 : testHeader1\_value

**REQUEST PARAMETERS**

param1 : param1\_val  
param2 : param2\_val

**SUBMIT REQUEST PARAMETERS**

param1:



Όπως βλέπουμε τα parameters εμφανίζονται και στην HTML του response αλλά και στο request που στείλαμε.

### Response Headers

Στον κώδικα του ExampleServlet ορίσαμε στην γραμμή 43 και στην γραμμή 97 τους παρακάτω response headers:

```
response.addHeader("testResponseHeader", "testResponseHeader_val");
```

```
response.setHeader("testHeader1", "testHeader1_value");
```

Οι headers αυτοί φαίνονται εάν εξετάσουμε το response στην κονσόλα του chrome:

The screenshot shows the Network tab in Chrome DevTools. A specific request to 'ExampleServlet' is selected. In the Headers section, there are two entries highlighted with a red circle: 'testHeader1: testHeader1\_value' and 'testResponseHeader: testResponseHeader\_val'. Other headers listed include 'Content-Type: text/html; charset=UTF-8', 'Date: Sat, 15 Jul 2017 21:35:51 GMT', and 'Transfer-Encoding: chunked'. The 'Request Headers' section also lists 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8'.

## Εξέταση της Προσθήκης και της Ανάγνωσης των Response Headers

Ο κώδικας στην κλάση του ExampleServlet προς το τέλος της μεθόδου έχει σκοπό να εξετάσει την συμπεριφορά των μεθόδων προσθήκης και ανάγνωσης των headers στο HttpServletRespose.

Ο κώδικας αυτός καταγράφει στο default output stream τα διάφορα χαρακτηριστικά του response.

**line 89:** Το **status code** από το status line του http response όπως περιεγράφηκε σε προηγούμενο κεφάλαιο.

**lines 90:** Το **buffer size** αναφέρεται στο μέγεθος του buffer του response αντικειμένου. Όταν το buffer γεμίσει το response γίνεται flushed (δηλαδή committed, που σημαίνει ότι στέλνεται προς τον χρήστη).

**lines 91-93:** Οι γραμμές αυτές αναφέρονται σε τιμές που ενυπάρχουν στα αντίστοιχα headers.

**line 94:** Η μέθοδος **isCommitted** μπορεί να κληθεί για να διαπιστωθεί εάν το response έχει σταλθεί προς τον client.

**line 97:** Προσθέτουμε έναν header μέσω της μεθόδου **setHeader**.

**lines 99 - 111:** Καλούμε την **WebUtilsImpl.getStringWithAllTheResponseHeaders** πριν, αλλά και μετά την αποστολή του response προς τον χρήστη. Η **WebUtilsImpl.getStringWithAllTheResponseHeaders** θα χρησιμοποιήσει της μεθόδους του **HttpServletResponse**, **getHeaderNames** και **getHeader** για να διαβάσει όλους τους headers του response που προστέθηκαν με τις αντίστοιχες **set** και **add** μεθόδους. Όπως αναφέρθηκε προηγουμένως στην παράγραφο “**Servlet Response, Response Headers, Response Headers και HTTP Μηχανισμοί**” μπορούν να διαβαστούν μόνο headers που προστέθηκαν μέσω των **add** και **set** μεθόδων του **HttpServletResponse** οπότε δεν είναι δυνατή η ανάγνωση όλων των headers που θα σταλθούν πραγματικά προς τον πελάτη. Στην συνέχεια στα αποτελέσματα της εκτέλεσης του κώδικα φαίνεται ότι κάποια headers δεν εμφανίζονται πριν από το **commit** του response και κάποια εμφανίζονται μετά το **commit** του αλλά δεν εμφανίζονται οι τιμές τους παρόλο που μέσω μιας κονσόλας του Chrome φαίνεται ότι τελικώς έχουν σταλθεί.

Παρουσιάζονται τα αποτελέσματα της εκτέλεσης του κώδικα. Παρουσιάζεται το output στο default output stream που μπορεί να συγκριθεί με το αντίστοιχο response στην κονσόλα του Chrome.

## Server Console Output

```
Setting a test header (testHeader1).  
  
RESPONSE HEADERS (before committing the response)  
  
Content-Type: , contained in response: false, headerValue: null  
testResponseHeader, contained in response: true, headerValue: testResponseHeader_val  
testHeader1, contained in response: true, headerValue: testHeader1_value  
  
IS COMMITTED (after flushBuffer): true  
  
RESPONSE HEADERS (after committing the response)  
  
Date: , contained in response: false, headerValue: null  
Transfer-Encoding: , contained in response: false, headerValue: null  
Content-Type: , contained in response: false, headerValue: null  
testResponseHeader, contained in response: true, headerValue: testResponseHeader_val  
testHeader1, contained in response: true, headerValue: testHeader1_value
```

Παρατηρούμε ότι η μέθοδος **getHeaderNames** επιστρέφει το **Content-Type** παρόλο που δεν το ορίσαμε μέσω των **get/add** μεθόδων αλλά έμμεσα από την **setContentType** και **setCharacterEncoding**. Παρατηρούμε ότι δεν επιστρέφεται το value του από την **getHeader** ούτε και η **containsHeader** επιστρέφει **true**. Μετά το **commit** του **response** επιστρέφονται επιπλέον μέθοδοι στο ίδιο στυλ.

**Θα παρατηρήσουμε ότι για τα headers Date, Transfer-Encoding, Content-Type εμφανίζονται κανονικά οι τιμές τους στην κονσόλα του Chrome παρόλο που δεν εμφανίζονταν στο default system output stream.**

Όπως βλέπουμε τα API της Java EE δεν προσφέρουν δυνατότητα για την ακριβή εξέταση όλων των headers τα οποία θα επιστραφούν στον χρήστη. Ως εκ τούτου για αυτόν τον σκοπό συχνά καταφεύγουμε σε ειδικά εργαλεία ανάλυσης της διαδικτυακής κίνησης (π.χ. Wireshark).

# JSP (Java Servlet Pages)

## Γενική Περιγραφή

Εκτός από τα Servlets, για την δυναμική δημιουργία περιεχομένου χρησιμοποιούνται επίσης και τα **JSP (Java Servlet Pages)**. Τα JSP ουσιαστικά είναι ένας τρόπος δημιουργίας ενός Servlet που μοιάζει περισσότερο με την δημιουργία HTML σελίδας και λιγότερο με γραφή Java κώδικα. Συνήθως εάν χρειάζεται να γράψουμε περισσότερο HTML κώδικα και λιγότερο Java κώδικα χρησιμοποιούμε JSP και αντίστοιχα εάν συμβαίνει το ανάποδο, Servlet. **Θεωρητικά ο ρόλος των JSP είναι η παρουσίαση των δεδομένων και ο ρόλος των Servlets η επεξεργασία τους** αν και οτιδήποτε μπορεί να γίνει με έναν Servlet μπορεί να γίνει και με μια JSP και το ανάποδο.

## JSP – Deployment Hierarchy

Τα JSP τοποθετούνται συνήθως στον **root application folder** ή σε **υποφακέλους του**, υπάρχει όμως και η δυνατότητα να τοποθετηθούν στον WEB-INF. Δεν χρειάζεται να δηλωθούν στον deployment descriptor όπως τα Servlets εάν βρίσκονται έξω από τον WEB-INF αν και υπάρχει αυτή η δυνατότητα εάν ο προγραμματιστής το επιθυμεί.

Εάν βρίσκονται μέσα στον WEB-INF δηλώνονται υποχρεωτικά προσθέτοντας **< servlet >** elements στον deployment descriptor και ορίζοντας **< jsp-file >** elements αντί για **< servlet-class >**. Σε αυτήν την περίπτωση οι JSP θα είναι άμεσα προσπελάσιμες με custom URL patterns ασχέτως της τοποθεσίας τους στο deployment hierarchy.

Προφανώς εάν βρίσκονται έξω από τον φάκελο WEB-INF όπως και οτιδήποτε άλλο έξω από αυτόν, είναι προσπελάσιμες από τον χρήστη εισάγοντας τα κατάλληλα URLs στον browser του που αντιστοιχούν στις δομές των φακέλων που τις εμπεριέχουν.

# Παράδειγμα JSP (welcome.jsp)

Στην συνέχεια θα γίνει παρουσίαση των βασικών στοιχείων μίας JSP μέσω παραδειγμάτων. Η σελίδα που θα εξετασθεί είναι η **welcome.jsp** και ανήκει στην εφαρμογή **DemoWebApp** που δημιουργήθηκε για τους σκοπούς της εργασίας και θα παρουσιάζεται σταδιακά.

```
1 <%@page import="osotnikov.web.enums.TransferMethodEnum"%>
2 <%@ page contentType="text/html" pageEncoding="UTF-8"%>
3 <%-- <%@ page import="osotnikov.utils.DateUtils"%> <%@ page import="osotnikov.utils.DateUtilsImpl"%> --%>
4 <%@ page import="osotnikov.web.enums.TransferMethodEnum"%>
5 <%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>
6 <%@ page import="javax.inject.Inject"%>
7 <%@ page import="osotnikov.demowebapp.constants.SessionAttributeEnum"%>
8
9<%!
10 //@Inject // Will not work in JSP as it only works in some implementations (but not in WLS).
11 //DateUtils dateUtils;%>
12
13 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
14
15<%!
16     private int getCurrentHour(){
17         java.util.Calendar rightNow = java.util.Calendar.getInstance();
18         int hour = rightNow.get(java.util.Calendar.HOUR_OF_DAY);
19         return hour;
20     } %>
21
22<html>
23    <head>
24
25        <title>Welcome Page</title>
26
27        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
28        <!--
29        <link rel="stylesheet" type="text/css" href="<%=request.getContextPath()%>/css/auth.css" /> -->
30
31        <%@ include file = "/WEB-INF/includes/head/common-style.html" %>
32
33        <%@ include file="/WEB-INF/includes/head/jquery.txt" %>
34
35        <script src="<%=request.getContextPath()%>/js/json2.js" type="text/javascript"></script>
36
```

## Content Type & Encoding (headers)

**line 2:** Θέτει τους **response headers** για το **content type** και το **encoding** της σελίδας όπως οι εντολές `response.setContentType("text/html")` και `response.setCharacterEncoding("UTF-8")` στον κώδικα του `ExampleServlet` που παρουσιάσθηκε στο προηγούμενο.

## Import packages and classes

**line 1:** Όπως και σε μία κλάση της Java για να χρησιμοποιήσουμε μία κλάση χωρίς αναφορά στο πλήρες όνομα (με πρόθεμα τα packages στα οποία βρίσκεται) πρέπει να την κάνουμε import. Αντί για το `import` statement χρησιμοποιούμε το **JSP directive “page”** με attribute `import`.

## Declare tag libraries

**line 5:** Εκτός από την εισαγωγή κώδικα java οι JSP επιτρέπουν τον προγραμματισμό και την δημιουργία δυναμικού περιεχομένου μέσω **JSP tags**. Τα JSP tags βρίσκονται μέσα σε αντίστοιχες βιβλιοθήκες (**taglibs**). Στην γραμμή 3 εισάγουμε την βιβλιοθήκη των **Core JSTL tags** με πρόθεμα “c”.

## **Doctype (non-jsp specific)**

**line 13:** Η γραμμή doctype θα πρέπει να υπάρχει σε κάθε HTML αρχείο (δεν έχει σχέση με το JSP πρότυπο). Ορίζει την έκδοση του HTML προτύπου που χρησιμοποιείται.

## **JSP Declarations (Methods and Fields)**

**lines 15-20:** Ορίζεται η μέθοδος `getCurrentHour`. Η μέθοδος αυτή θα ανήκει στην Servlet κλάση που θα παραχθεί από αυτό το JSP. Επιστρέφει την τρέχουσα ώρα της ημέρας (24-wro format). Για να γίνει ορισμός μεθόδων ή πεδίων χρησιμοποιούνται τα **JSP declarations**. Δηλαδή ο ορισμός γίνεται ανάμεσα στα σύμβολα “`<%!`” και “`%>`”.

## **Content Type & Encoding (meta tags, non-jsp specific)**

**line 27:** Παρόλο που το encoding και το content type ορίστηκαν στην γραμμή 1 μέσω των response headers ο W3C οργανισμός ορίζει ότι είναι καλή πρακτική να ορίζονται και μέσα στο HTML αρχείο. Καλό είναι να ορίζεται σχετικά νωρίς στο αρχείο διότι συνήθως διάφορα εργαλεία επεξεργασίας HTML καθώς και οι browsers εξετάζουν συγκεκριμένο αριθμό από τα πρώτα bytes του αρχείου για αυτήν την πληροφορία (εάν δεν δίνεται στους headers). Κατ’ αυτόν τον τρόπο εάν το αρχείο αυτό ανοιχτεί όχι μέσω του internet αλλά προσπελάζοντας τον σκληρό δίσκο θα μπορεί να γίνει χρήση αυτής της πληροφορίας από τα εργαλεία που θα εξετάσουν το αρχείο (εάν ακολουθούν και αυτά τις W3C οδηγίες)<sup>1</sup>.

## **JSP Include directive**

**line 33:** Στην γραμμή αυτή τα περιεχόμενα του αρχείου jquery.txt ενσωματώνονται μέσα στο JSP αρχείο μέσω του **JSP include directive**. Δηλαδή το αποτέλεσμα αυτής της γραμμής είναι το ίδιο με αυτό που θα παίρναμε εάν την αντικαθιστούσαμε με τα περιεχόμενα του αρχείου jquery.txt.

---

<sup>1</sup> [<https://www.w3.org/International/questions/qa-html-encoding-declarations.en>]

## JSP Expressions

**line 35:** Σε αυτήν την γραμμή ενσωματώνουμε το αρχείο json2.js στην σελίδα. Αυτό βρίσκεται στο path /js/json2.js. Τα paths όλων των resources της εφαρμογή αρχίζουν με πρόθεμα το context path της (DemoWebApp). Δεν θα ήταν σωστό να το εισάγουμε ως στατικό κείμενο διότι ενδέχεται να αλλάξει στο μέλλον και θα έπρεπε τότε να αλλάξουν όλα τα σημεία στα οποία χρησιμοποιείται. Αντί για αυτό παίρνουμε την τιμή του από την συνάρτηση **getContextPath** της κλάσης HttpServletRequest. Το HTTP request που έλαβε η JSP εμπεριέχεται στο **implicit object "request"**. Υπάρχει ένας αριθμός από implicit objects στο πρότυπο της JSP, δεν χρειάζεται να κατασκευαστούν από τον προγραμματιστή διότι δίνονται έτοιμα σε κάθε JSP σελίδα από το πρότυπο. Τέλος η τιμή του context path εισάγεται στην HTML από ένα **JSP expression**. Τα JSP expressions εμπεριέχονται ανάμεσα στα σύμβολα "`<%=`" και "`%>`" και θα πρέπει να είναι τύπου String, οποιαδήποτε τιμή παράγεται ανάμεσα σε αυτά εισάγεται ως κείμενο στον τελικό HTML κώδικα.

```
164      Current Date: <%=new java.text.SimpleDateFormat("dd/M/yyyy").format(new java.util.Date()) %>
165      </p>
166
167      <% ifgetCurrentHour() >= 18 {%
168          <div>
169              Good Evening!
170          </div>
171      <% }else{ %
172          <div>
173              Hello!
174          </div>
175      <% } %
176
177      <%@ include file = "/WEB-INF/includes/body/bg-color-selector.html" %>
178
179
```

## JSP Scriptlets

**lines 168-176:** Εδώ παρουσιάζονται τα **JSP scriptlets**. Εάν θέλουμε να εισάγουμε Java κώδικα στην σελίδα τον γράφουμε ανάμεσα από τα σύμβολα "`<%`" και "`%>`". Στις συγκεκριμένες γραμμές εισάγουμε δομή ελέγχου if-else. Εάν ο χρήστης προσπελάζει την σελίδα από τις 18.00 και μετά βλέπει το κείμενο "Good Evening!" ενώ άμα το κάνει πριν βλέπει το κείμενο "Hello!". Ουσιαστικά το απλό HTML κείμενο μπορούμε να φανταστούμε ότι μετατρέπεται σε java κώδικα που τον κάνει print στο buffer του HTTP response (όπως αντίστοιχα είδαμε και στην περίπτωση του ExampleServlet στο προηγούμενο κεφάλαιο). Οποιοσδήποτε άλλος java κώδικας θα πρέπει να μπαίνει μέσα σε scriptlets.

# Ανακατεύθυνση των Requests - Forward, Redirection και Include

Για την χρήση άλλων πόρων για την δημιουργία του περιεχομένου του response, πέραν του Servlet/JSP για τον οποίον προορίζονται το request, καθώς και για την μετάβαση του χρήστη από την μία σελίδα ενός web application σε μία άλλη, είναι απαραίτητη η μεταβίβαση/ανακατεύθυνση του αρχικού του request σε διαφορετικούς πόρους του web application (δηλαδή Servlets, JSPs και Static Resources) πέραν του αρχικού.

Για να επιτύχουμε τα προηγούμενα χρησιμοποιούμε τέσσερεις τεχνικές:

1. **Forward:** Πρόκειται για την μεταβίβαση του HttpServletRequest αντικειμένου από έναν Servlet ή JSP σε έναν άλλον Servlet ή JSP ή οποιοδήποτε άλλο web resource μέσα στην ίδια enterprise εφαρμογή. Είναι αυτονόητο ότι η μεταβίβαση αυτή γίνεται εσωτερικά του web application χωρίς δηλαδή να συμμετέχει ο browser του πελάτη. Συνήθως γίνεται όταν ο αρχικός Servlet εκτελεί business logic που σχετίζεται με το request (δηλαδή back end processes) και στην συνέχεια πραγματοποιεί forward σε κάποια JSP που αναλαμβάνει να κατασκευάσει το HTML περιεχόμενο που θα αποτελεί το response που θα λάβει ο χρήστης, δηλαδή το JSP αυτό θα έχει το έργο της κατασκευής της αναφοράς για τα αποτελέσματα του business process. Υπάρχουν πολλοί τρόποι για την μεταβίβαση των αποτελεσμάτων της επεξεργασίας από τον αρχικό Servlet/JSP στον τελικό, ένας πολύ συνήθης είναι μέσω των request attributes. Ο πόρος που είναι ο τελικός παραλήπτης τους request αναλαμβάνει πλήρως την κατασκευή του περιεχομένου του response.
2. **Redirect:** Πρόκειται για ανακατεύθυνση του αρχικού HTTP Request από τον ίδιο τον browser του χρήστη αφού λάβει την αντίστοιχη εντολή από τον server μέσω response με status 302. Δηλαδή ενώ κάποιος Servlet/JSP λαμβάνει το request του πελάτη, του επιστρέφει response τύπου redirect, δηλαδή με status code 302 (FOUND) χωρίς κάποιο περιεχόμενο αλλά με το "location" header να περιέχει την διεύθυνση του άλλου πόρου από τον οποίον ο χρήστης θα λάβει το περιεχόμενο το οποίο επιθυμεί. Ο browser του χρήστη αφού λάβει ένα τέτοιο response θα πραγματοποιήσει αυτόματα ένα request στον πόρο που καταδεικνύεται από το header "location".
3. **Include Action:** Πρόκειται για την συμπερίληψη περιεχομένου από έναν πόρο στο περιεχόμενο του response του αρχικού Servlet/JSP που δέχτηκε το request. Το include action είναι παρόμοιο με το forward με την μόνη διαφορά ότι το περιεχόμενο του response δημιουργείται και από το αρχικό Servlet/JSP αλλά και τον πόρο που ορίζεται κατά το include, δηλαδή διαφέρει από το forward όπου το περιεχόμενο του response καθορίζονται αποκλειστικά από τον τελικό πόρο.
4. **Include Directive:** Μπορεί να πραγματοποιηθεί μόνο μέσα σε μία JSP. Λειτουργικά είναι όμοιο με το Include Action αλλά αντί να εκτελείται κατά το runtime η συμπερίληψη του περιεχομένου του πόρου που ορίζεται στο include γίνεται κατά το translation time της JSP (δηλαδή όταν από το περιεχόμενο της JSP ο Application Server παράγει το αντίστοιχο Servlet).

# Εφαρμογή και Επεξήγηση με Κώδικα

Θα γίνει εφαρμογή και επεξήγηση των προηγούμενων τεχνικών μέσω απλών παραδειγμάτων. Ο κώδικας της εφαρμογής DemoWebApp που αντιστοιχεί στα παραδείγματα βρίσκεται στις κλάσεις **ForwardingAndRedirectionServlet**, **RequestResponseAndSessionExaminationServlet** καθώς και στην JSP **welcome.jsp**.

Παρακάτω παρουσιάζεται ο κώδικας που αντιστοιχεί στα παραδείγματα για το forward και το redirection, αργότερα θα παρουσιαστεί και ο κώδικας για το include.

## welcome.jsp, forward example:

```
165  <%-- FORWARDING AND REDIRECTION EXAMPLE REQUEST FORM --%>
166
167<div class="forwardingAndRedirectionExample">
168  <form id="forwardingAndRedirectionExampleForm" name="forwardingAndRedirectionExampleForm"
169    action="<%request.getContextPath()%>/actions/ForwardingAndRedirectionServlet" method="POST">
170
171  <fieldset>
172    <legend>Forwarding and Redirection Example</legend>
173
174  <div>
175    <label for="unprocessed_msg">Message for intermediate processing:</label>
176    <input type="text" id="unprocessed_msg" name="unprocessed_msg"
177      value="This is a sample message."/>
178  </div>
179
180  <div>
181    <label for="transferMethodRadioForward">Transfer to the welcome.jsp by forwarding:</label>
182    <input type="radio" id="transferMethodRadioForward" name="transferMethod"
183      value="<%TransferMethodEnum.FORWARD.getName()%>" checked="checked"/>
184  </div>
185
186  <div>
187    <label for="transferMethodRadioRedirectInternal">
188      Transfer to the welcome.jsp by internal redirection:</label>
189    <input type="radio" id="transferMethodRadioRedirectInternal" name="transferMethod"
190      value="<%TransferMethodEnum.INTERNAL_REDIRECT.getName()%>" />
191  </div>
192  <div>
193    <label for="transferMethodRadioRedirectExternal">
194      Transfer to the google by external redirection:</label>
195    <input type="radio" id="transferMethodRadioRedirectExternal" name="transferMethod"
196      value="<%TransferMethodEnum.EXTERNAL_REDIRECT.getName()%>" />
197  </div>
198
199  <div class="buttonRow">
200    <input type="submit" value="Submit" />
201  </div>
202
203  </fieldset>
204</form>
205</div>
```

## Παραγόμενη HTML φόρμα:

Forwarding and Redirection Example

Message for intermediate processing: This is a sample message.

Transfer to the request, response & session examination page by forwarding:

Transfer to the request, response & session examination page by internal redirection:

Transfer to the youtube by external redirection:

Submit

Στις εικόνες βλέπουμε των κώδικα και την φόρμα της εκτέλεσης των παραδειγμάτων forwarding και redirection, στην οποία ο χρήστης μπορεί να επιλέξει το παράδειγμα που θέλει να εκτελέσει.

Κατά την υποβολή της φόρμας θα σταλθεί ένα **POST request** στο Servlet

**ForwardingAndRedirectionServlet** όπως διαπιστώνουμε και από τα attributes, action και method του form element.

Υπάρχει ένα input field για μήνυμα κειμένου και τρία radio buttons. Το μήνυμα κειμένου αντιστοιχεί στο request parameter “**unprocessed\_msg**” ενώ τα τρία radio buttons αντιστοιχούν στο “**transferMethod**”.

Το πρώτο radio button αντιστοιχεί στο forward στην σελίδα που παράγει το Servlet **RequestResponseAndSessionExaminationServlet**, το δεύτερο στην μετάβαση με redirect στην ίδια σελίδα όπως και στο forward, και το τρίτο στην μετάβαση με redirect στην σελίδα <http://www.youtube.com> που είναι σελίδα του διαδικτύου που βρίσκεται εξωτερικά από το web application.

### **ForwardingAndRedirectionServlet:**

Παρακάτω παρουσιάζεται ο κώδικας του **ForwardingAndRedirectionServlet** το οποίο δέχεται το request της προηγούμενης φόρμας.

```
16 public class ForwardingAndRedirectionServlet extends HttpServlet {
17
18     private static final long serialVersionUID = 1L;
19
20     @Inject
21     StringUtils stringUtils;
22
23     public ForwardingAndRedirectionServlet() {
24         super();
25     }
26
27     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
28
29         // PROCESS
30
31         System.out.println("ForwardingAndRedirectionServlet.doGet: started for request: "
32             + request.getRequestURL());
33
34         String unprocessedInput = stringUtils.getTrimmedOrEmpty(
35             request.getParameter("unprocessed_msg"));
36         String processedMsg = unprocessedInput.toUpperCase();
37         // Processing result:
38         request.setAttribute("processed_msg", processedMsg);
39
40         // transferringMode parameter:
41         String transferMethod = request.getParameter("transferMethod");
42
43         if(TransferMethodEnum.FORWARD.getName().equals(transferMethod)) {
44
45             getServletContext().getRequestDispatcher(
46                 "/actions/RequestResponseAndSessionExaminationServlet").forward(request, response);
47         }else if(TransferMethodEnum.INTERNAL_REDIRECT.getName().equals(transferMethod)) {
48
49             response.sendRedirect(request.getContextPath() +
50                 "/actions/RequestResponseAndSessionExaminationServlet?processed_msg=" + processedMsg+"&param2=param2_val");
51
52         }else if(TransferMethodEnum.EXTERNAL_REDIRECT.getName().equals(transferMethod)) {
53
54             response.sendRedirect("http://www.youtube.com?processed_msg=" + processedMsg);
55
56         }else{
57
58             throw new RuntimeException("ERROR: NO TRANSFER MODE WAS PROVIDED !!!");
59
60         }
61
62         System.out.println("ForwardingAndRedirectionServlet.doGet: ended for request: "
63             + request.getRequestURL());
64
65     }
66
67     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
68
69         doGet(request, response);
70     }
71
72 }
73 }
```

Ο κώδικας του **ForwardingAndRedirectionServlet** έχει ως σκοπό να προσομοιώσει αρχικά την εκτέλεση μίας εργασίας (business logic κομμάτι) και στην συνέχεια την παρουσίαση του αποτελέσματός της με το κατάλληλο “view” (όπως αυτό ορίζεται στο MVC αρχιτεκτονικό πρότυπο) δηλαδή έναν πόρο ο οποίος είναι σχεδιασμένος αποκλειστικά για να αναλαμβάνει την παρουσίαση των αποτελεσμάτων μίας επεξεργασίας.

Στο απλουστευμένο παράδειγμά μας το business process γίνεται στην γραμμή 30 όπου το μήνυμα που εισάγαμε στην φόρμα αλλάζει για να είναι όλα σε κεφαλαία, στην συνέχεια το αποτέλεσμα της επεξεργασίας αποθηκεύεται στο request attribute “processed\_msg” για να μεταβιβαστεί κατά το forward στο view.

Παρατηρούμε ότι με βάση την τιμή της μεταβλητής transferMethod εκτελείται και το αντίστοιχο case στην if-else δομή.

Τον ρόλο του view στα δύο πρώτα cases τον εκτελεί το Servlet

**RequestResponseAndSessionServlet** που παράγει το τελικό περιεχόμενο του response υπό την μορφή μιας HTML σελίδας. Το τρίτο case αποτελεί ένα παράδειγμα μετάβασης σε έναν εξωτερικό σύνδεσμο.

Ο κώδικας του **RequestResponseAndSessionServlet** έχει ως σκοπό την δημιουργία μιας HTML σελίδας η οποία, μεταξύ των άλλων, αναλύει πλήρως το HttpServletRequest αντικείμενο.

Συγκεκριμένα παρουσιάζει όλα τα request parameters που έλαβε και τις τιμές τους.

## Forwarding

Στην πρώτη περίπτωση, για να πραγματοποιηθεί το forward χρησιμοποιούμε την μέθοδο **RequestDispatcher.forward(HttpServletRequest, HttpServletResponse)**. Παίρνουμε το RequestDispatcher αντικείμενο είτε από το **ServletContext** (προσβάσιμο από την **getServletContext()** ενός HttpServlet) ή από το **HttpServletRequest** (η μόνη διαφορά είναι ότι στην δεύτερη περίπτωση μπορούμε να ορίσουμε relative URLs και προς το location του HttpServletRequest ενώ στην πρώτη μόνο προς το context root). Η διεύθυνση στην οποία γίνεται το forward ορίζεται κατά την δημιουργία του RequestDispatcher αντικειμένου στην **ServletContext.getRequestDispatcher(String)**. Επειδή αναγκαστικά θα πρέπει να αναφέρεται σε πόρο ο οποίος βρίσκεται στην ίδια εφαρμογή στην οποία βρίσκεται και ο αρχικός Servlet/JSP παρατηρούμε ότι το URL δεν ξεκινάει με το context root “DemoWebApp”.

Παρακάτω παρουσιάζεται το αποτέλεσμα της εκτέλεσης της μετάβασης μέσω forward:

The screenshot shows a browser window with the URL <https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet>. Below the browser is the Chrome DevTools Network tab, which displays a single XHR request to the same URL. The Response section of the DevTools shows the HTML content of the forwarded page. The browser's developer tools also show the REQUEST PARAMETERS and REQUEST ATTRIBUTES for the original request, which include 'transferMethod : FORWARD' and 'unprocessed\_msg : This is a sample message.' The forwarded page's content is visible in the browser's body, with the 'processed\_msg' attribute highlighted in red.

Name	Headers	Preview	Response	Cookies	Timing
ForwardingAndRedirectionSe...					
inject.js					

**REQUEST PARAMETERS**

```
transferMethod : FORWARD
unprocessed_msg : This is a sample message.
```

**REQUEST ATTRIBUTES**

```
requestInitEventNotified : true
javax.servlet.forward.context_path : /DemoWebApp
javax.servlet.forward.servlet_path : /actions/ForwardingAndRedirectionServlet
weblogic.servlet.request.sslsession : [Session-8, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256]
weblogic.servlet.forward.target_servlet_path : /actions/RequestResponseAndSessionExaminationServlet
javax.servlet.forward.request_uri : /DemoWebApp/actions/ForwardingAndRedirectionServlet
org.jboss.weld.servlet.ConversationContextActivator.contextActivatedInRequest : true
javax.servlet.request.key_size : 128
org.jboss.weld.context.ignore.guard.marker : java.lang.Object@30f62526
processed_msg : THIS IS A SAMPLE MESSAGE.
```

Παρατηρούμε ότι παρόλο που λάβαμε response από τον **RequestResponseAndSessionExaminationServlet** στην γραμμή διεύθυνσης εμφανίζεται η διεύθυνση του **ForwardingAndRedirectionServlet** και αυτό διότι ο browser πραγματοποίησε μόνο ένα request όπως φαίνεται και από την κονσόλα του δεξιά.

Στο πλαίσιο “Form Data” παρουσιάζονται τα request parameters που στάλθηκαν με το request ενώ στο πλαίσιο “General” μεταξύ των άλλων, παρουσιάζεται το status code του response.

Αριστερά στο σώμα του response φαίνονται τα request parameters τα οποία μεταβιβάστηκαν αυτούσια μαζί με το HttpServletRequest αντικείμενο από το ForwardingAndRedirectionServlet στο RequestResponseAndSessionExaminationServlet. Επίσης φαίνεται το request attribute το οποίο ορίστηκε από το ForwardingAndRedirectionServlet στο οποίο αποθήκευσε το αποτέλεσμα της επεξεργασίας του request parameter “unprocessed\_msg”.

Σημειώνεται ότι **εάν πριν από το forward έχουν γραφτεί δεδομένα στο Output Stream του HttpServletResponse αυτά διαγράφονται μετά την μετάβαση στον τελικό Servlet οπότε αυτός ξεκινάει να γράφει το response body εκ νέου.**

Επίσης **εάν έχει γίνει commit του response το forward θα αποτύχει παράγοντας Exception.**

### [Redirect to Internal Resource](#)

Στην δεύτερη περίπτωση εκτελούμε **redirect** όπως και στο forward στο **RequestResponseAndSessionExaminationServlet**. Επειδή το redirect θα πραγματοποιηθεί ουσιαστικά από τον browser του χρήστη είναι λογικό ότι ορίζεται πάνω στο **HttpResponse** αντικείμενο που θα επιστραφεί σε αυτόν σε πρώτη φάση.

Η μέθοδος που χρησιμοποιείται είναι η **HttpResponse.sendRedirect(String url)**. Αντίθετα με την περίπτωση του forward **το URL πρέπει να ξεκινάει με το context root εάν πρόκειται για ένα URL που είναι relative προς το domain**, προφανώς υποστηρίζονται και absolute URLs όπως θα δούμε και στην συνέχεια.

Δεν μπορούμε να αποθηκεύσουμε δεδομένα που θα σταλθούν στον τελικό πόρο στο HttpServletRequest αντικείμενο όπως στην περίπτωση του forward αφού θα πραγματοποιηθεί καινούριο request από τον browser. **Ένας συνήθης τρόπος για μεταβίβαση πληροφορίας στον τελικό πόρο είναι μέσω του URL στο οποίο γίνεται το redirection, ορίζοντας request parameters μέσα στο query string.**

Η μέθοδος **HttpResponse.sendRedirect(String URL)** έχει ουσιαστικά ως αποτέλεσμα να οριστεί το **response status** του HTTP response που θα επιστρέψει το ForwardingAndRedirectionServlet σε **302 και το header “Location”** στην τιμή του URL που ορίστηκε ως παράμετρος στην μέθοδο **sendRedirect**.

Mη ασφαλής | https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet?processed\_msg=THIS%20IS%20A%...

testHeader1 : testHeader1\_value

## REQUEST PARAMETERS

param2 : param2\_val

processed\_msg : THIS IS A SAMPLE MESSAGE.

## REQUEST ATTRIBUTES

requestInitEventNotified : true

weblogic.servlet.request.sslsession : [Session-10, TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256]

org.jboss.weld.servlet.ConversationContextActivator.contextActivatedInRequest : true

javax.servlet.request.key\_size : 128

org.jboss.weld.context.ignore.guard.marker : java.lang.Object@30f62526

javax.servlet.request.cipher\_suite : TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

Στο αποτέλεσμα του redirection σε αντίθεση με το αποτέλεσμα του forward παρατηρούμε ότι στην γραμμή διεύθυνσης εμφανίζεται η διεύθυνση του τελικού πόρου, δηλαδή του RequestResponseAndSessionExaminationServlet και όχι του ForwardingAndRedirectionServlet όπως προηγουμένως.

Όπως παρατηρούμε από το αποτέλεσμα του redirection το status code του response για το αρχικό request στον ForwardingAndRedirectionServlet είναι 302.

Name	Headers	Preview	Response	Cookies	Timing
ForwardingAndRedirectionSe...	<b>General</b> <b>Request URL:</b> https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet?processed_msg=THIS%20IS%20A%20SAMPLE%20MESSAGE.&param2=param2_val <b>Request Method:</b> GET <b>Status Code:</b> 200 OK <b>Remote Address:</b> [::1]:7002 <b>Referrer Policy:</b> no-referrer-when-downgrade				
RequestResponseAndSession...	<b>Response Headers (6)</b> <b>Request Headers (11)</b> <b>Query String Parameters</b> <a href="#">view source</a> <a href="#">view URL encoded</a> processed_msg: THIS IS A SAMPLE MESSAGE. param2: param2_val				
inject.js					

Από την κονσόλα φαίνεται και το δεύτερο request το οποίο πραγματοποίησε ο browser αφού έλαβε το αρχικό response. Τα query string parameters είναι αυτά που ορίστηκαν στο query string του URL του redirection στην HttpServletResponse.sendRedirect(String) μέθοδο.

## Redirect to External Resource

Στην τρίτη περίπτωση εκτελούμε redirect προς έναν εξωτερικό σύνδεσμο. Συγκεκριμένα προς το absolute URL <http://www.youtube.com>. Δεν υπάρχει κάποια ουσιαστική διαφορά με την προηγούμενη περίπτωση.

The screenshot shows a browser window with the YouTube homepage. The search bar contains the Greek text 'Αναζήτηση'. Below the search bar, there are tabs for 'Αρχική σελ.', 'Τάσεις', and 'Συνδρομές'. The main content area displays a grid of video thumbnails under the heading 'Προτεινόμενα'. One thumbnail for 'Joe Rogan's Blue Cheese with Wings Moment' is visible. To the right of the browser window, the Network tab of the developer tools is open. It shows a list of requests and a detailed view of a POST request to 'ForwardingAndRedirectionServlet'. The request details show the URL as 'https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet', the method as 'POST', and the status code as '302 Moved Temporarily'. The response body is highlighted with a red box and contains the text 'unprocessed\_msg: This is a sample message.' and 'transferMethod: EXTERNAL\_REDIRECT'.

## Include Action μέσα σε ένα Servlet

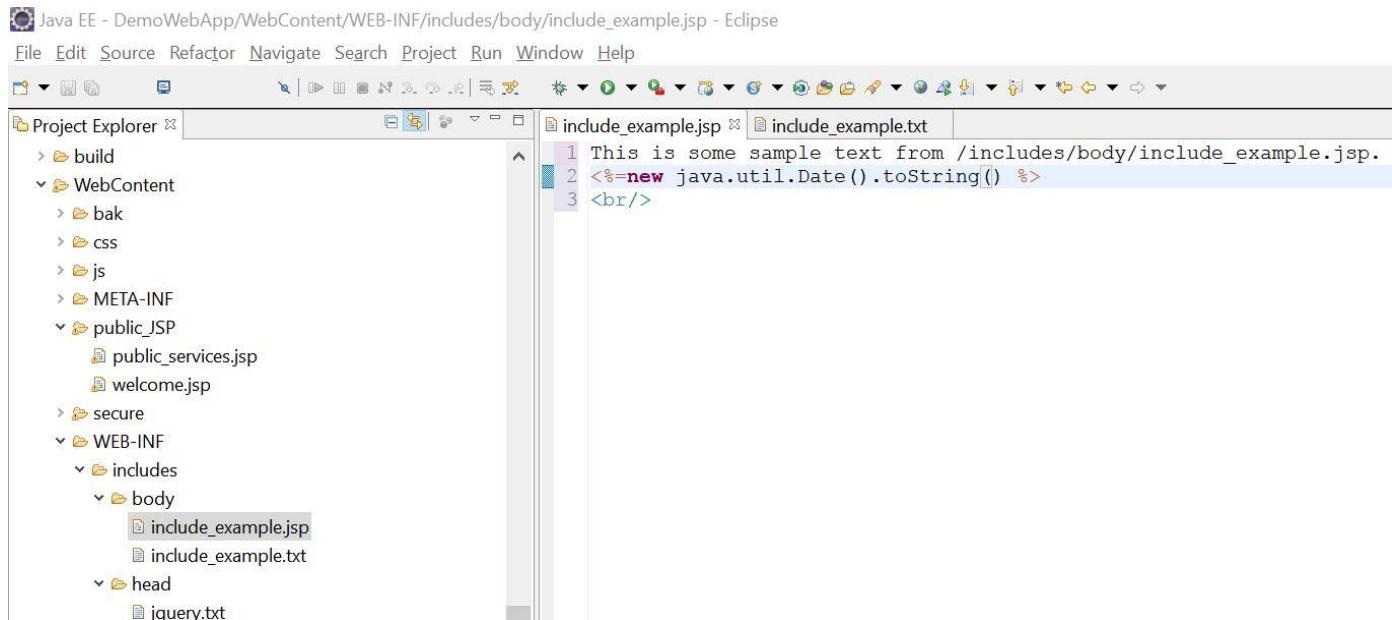
Αρχικά θα παρουσιασθεί το πώς πραγματοποιούμε ένα include action μέσα σε έναν Servlet. Ο κώδικας του παραδείγματος αυτού βρίσκεται και πάλι μέσα στον RequestResponseAndSessionExaminationServlet.

```

153 // Include example with context root as point of reference with a request dispatcher from the ServletContext.
154 resOut.write("<h2>INCLUDE EXAMPLE</h2>");
155 RequestDispatcher rd = getServletContext().getRequestDispatcher("/WEB-INF/includes/body/include_example.jsp");
156 rd.include(request, response);
157 // Include example with context root as point of reference. Getting the RequestDispatcher from the request
158 // is not enough to make the request relative to the servlet path. It also has to
159 // start with no forward slash at the beginning otherwise it's relative to the context root just like the
160 // request dispatcher from ServletContext.
161 rd = request.getServletContext().getRequestDispatcher("/WEB-INF/includes/body/include_example.txt");
162 rd.include(request, response);
163 // Include example with current servlet's previous path element ("inits") as point of reference.
164 rd = request.getRequestDispatcher("../WEB-INF/includes/body/include_example.jsp");
165 rd.include(request, response);
166
167 resOut.write(
    "<h2>THE REQUEST PARAMETER'S VALUE IS A POST REQUEST /?</h2>");
```

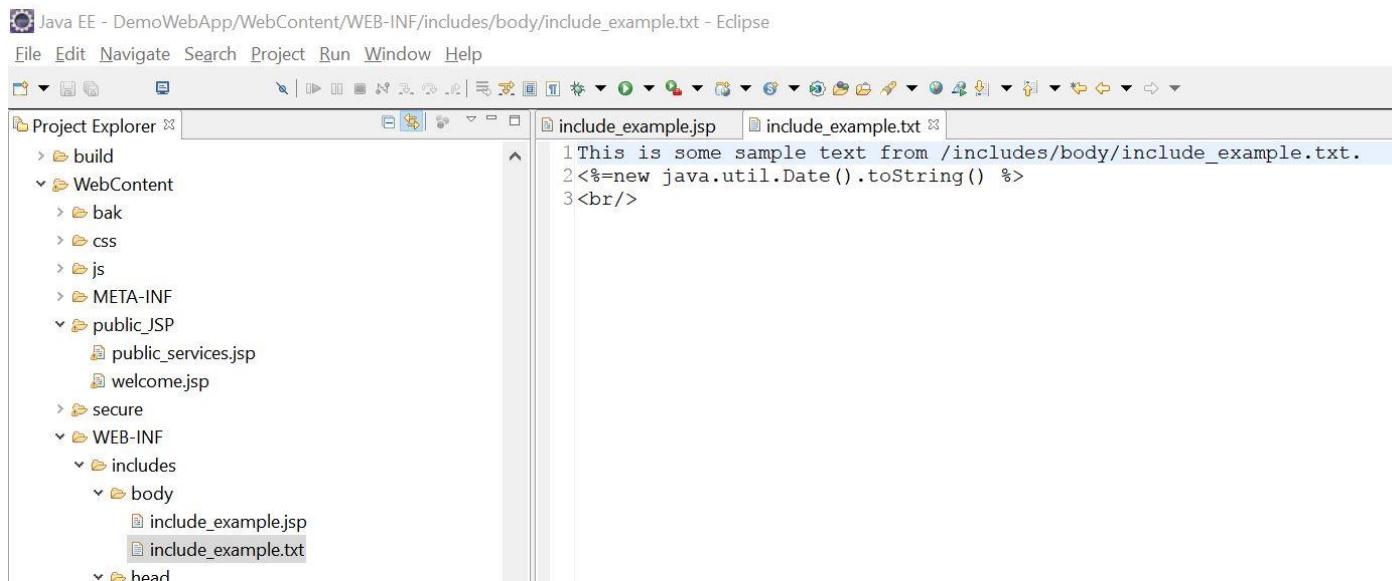
Συγκεκριμένα πραγματοποιούνται 3 include actions όπου όλα ενσωματώνουν περιεχόμενο στο response body που παράγει ο **RequestResponseAndSessionExaminationServlet**. Το περιεχόμενο αυτό ορίζεται από τα αρχεία **include\_example.txt** και **include\_example.jsp**.

### include\_example.jsp:



```
This is some sample text from /includes/body/include_example.jsp.  
<%=new java.util.Date().toString()%>  
<br/>
```

### include\_example.txt:



```
This is some sample text from /includes/body/include_example.txt.  
<%=new java.util.Date().toString()%>  
<br/>
```

Όπως και στην περίπτωση ενός forward για να πραγματοποιηθεί ένα include χρειαζόμαστε ένα **RequestDispatcher** αντικείμενο το οποίο μπορούμε να το πάρουμε είτε από το ServletContext ή από το HttpServletRequest με την μέθοδο **getRequestDispatcher(String)** στην οποία δίνουμε ως όρισμα το path για τον τελικό πόρο.

Για να αναφερθούμε στον τελικό πόρο με ένα path το οποίο είναι relative προς το path για το οποίο προοριζόταν το αρχικό request θα πρέπει να πάρουμε το RequestDispatcher αντικείμενο από το HttpServletRequest αντικείμενο και το path που θα δώσουμε ως όρισμα στην **getRequestDispatcher(String)** να μην ξεκινάει με "/" (τρίτο include). Σε αντίθετη περίπτωση, ασχέτως με το πώς έχουμε πάρει το RequestDispatcher αντικείμενο, εάν το path ξεκινάει με "/" είναι αυτομάτως relative προς το context root.

Το αποτέλεσμα των includes παρουσιάζεται παρακάτω.



## SESSION ATTRIBUTES

WELD\_S\_HASH : 2129431603

## RESPONSE INFO (before the commit)

STATUS: 200  
BUFFER SIZE: 12216  
CONTENT TYPE: text/html; charset=UTF-8  
CHARACTER ENCODING: UTF-8  
LOCALE: en\_US  
IS COMMITED: false

## INCLUDE EXAMPLE

This is some sample text from /includes/body/include\_example.jsp. Thu May 18 18:27:25 EEST 2017  
This is some sample text from /includes/body/include\_example.txt. <%=new java.util.Date().toString()%>  
This is some sample text from /includes/body/include\_example.jsp. Thu May 18 18:27:25 EEST 2017

Παρατηρούμε ότι ενώ το περιεχόμενο των αρχείων include\_example.jsp και include\_example.txt είναι σχεδόν ίδιο, δεν παίρνουμε το ίδιο αποτέλεσμα όταν τα κάνουμε include. Συγκεκριμένα βλέπουμε ότι όσον αφορά το expression <%=new java.util.Date().toString()%> στην περίπτωση του include\_example.jsp παίρνουμε το αποτέλεσμά του ενώ στην περίπτωση του include\_example.txt παίρνουμε απλώς το κείμενο από το οποίο αποτελείται. Αυτό γίνεται διότι στην περίπτωση του include\_example.jsp δημιουργείται ένα JSP εξαιτίας της κατάληξης που δόθηκε στο αρχείο ενώ το include\_example.txt είναι απλώς ένα static resource. Από εκεί και πέρα το include action λειτουργεί παρόμοια με το forward, στέλνει δηλαδή το request και response αντικείμενο στον πόρο που ορίστηκε στον RequestDispatcher, εάν αυτός ο πόρος είναι δυναμικός, όπως μια JSP, θα δημιουργηθεί δυναμικά περιεχόμενο από αυτόν, το οποίο θα ενσωματωθεί στο response αντικείμενο. Από την άλλη, εάν ο πόρος είναι στατικός, θα ενσωματωθεί αυτούσιο το περιεχόμενό του.

Τεχνικά η διαφορά του include με το forward είναι ότι εάν πριν από το include έχουν γραφτεί δεδομένα στο Output Stream του HttpServletResponse αυτά δεν διαγράφονται μετά την μετάβαση στον τελικό πόρο αλλά αντίθετα αυτός συνεχίζει από το σημείο που σταμάτησε ο αρχικός Servlet.

Επίσης δεν πραγματοποιείται commit του response από τον πόρο που γίνεται include για να μπορεί ο αρχικός Servlet/JSP να συνεχίσει να γράφει στο Output Stream.

Όπως και στην περίπτωση του forward εάν έχει γίνει ήδη commit του response το include θα αποτύχει παράγοντας Exception.

## Include Action και Include Directive σε μια JSP

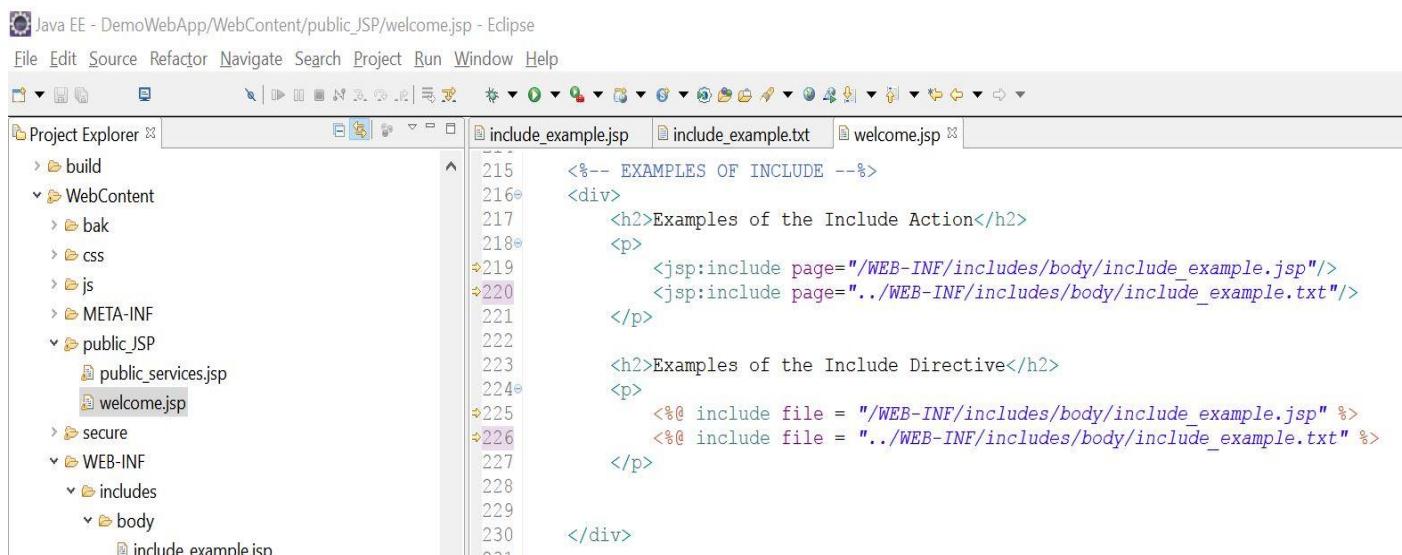
Για να πραγματοποιηθεί include action σε μια JSP θα πρέπει να χρησιμοποιήσουμε το **JSP action tag <jsp:include>**. Στο attribute “page” ορίζουμε το path προς τον πόρο όπως ακριβώς ορίζαμε και στον RequestDispatcher στα παραδείγματα του include στον **RequestResponseAndSessionExaminationServlet**. Υποστηρίζονται και paths που είναι relative προς την JSP από την οποία γίνεται το include.

Εκτός από το include action σε μια JSP μπορούμε να χρησιμοποιήσουμε και το **include directive <% include file="..."%>**. Χρησιμοποιούμε το attribute “file” όπως και το page στο <jsp:include> tag προηγουμένως.

### **Διαφορά μεταξύ του Include Action και του Include Directive**

Η μόνη διαφορά μεταξύ του JSP include action και του JSP include directive είναι ότι το πρώτο μεταφέρει το request και response αντικείμενο στον τελικό πόρο κατά το runtime ενώ το δεύτερο απλώς αντιγράφει τα περιεχόμενα του αρχείου του τελικού πόρου στην θέση του include directive tag κάποια στιγμή πριν από το translation της JSP σε Servlet. Δηλαδή στην δεύτερη περίπτωση θα είχαμε το ίδιο αποτέλεσμα εάν απλώς κάναμε copy – paste τα περιεχόμενα του include\_example.txt στις θέσεις των δύο include directive tags κατά την συγγραφή του κώδικα.

**Ο κώδικας των παραδειγμάτων του include action και του include directive μέσα στην welcome.jsp:**



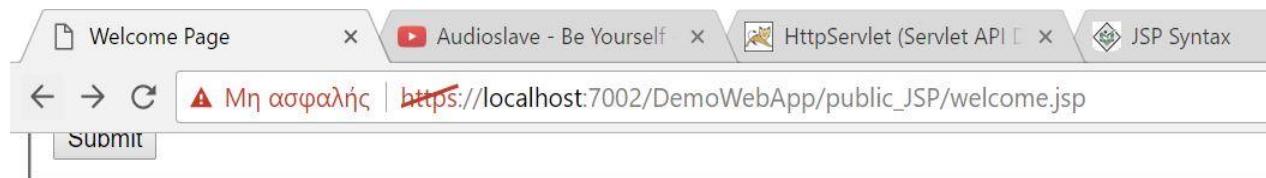
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with folders like build, WebContent (containing bak, css, js, META-INF, public\_JSP, secure, WEB-INF), and includes (containing body). A file named include\_example.iso is also listed.
- Code Editor:** Displays the content of the welcome.jsp file. The code includes examples for both JSP include action and include directive, demonstrating their usage with relative paths.

```
<%-- EXAMPLES OF INCLUDE --%>
<div>
    <h2>Examples of the Include Action</h2>
    <p>
        <jsp:include page="/WEB-INF/includes/body/include_example.jsp"/>
        <jsp:include page="../WEB-INF/includes/body/include_example.txt"/>
    </p>

    <h2>Examples of the Include Directive</h2>
    <p>
        <%@ include file = "/WEB-INF/includes/body/include_example.jsp" %>
        <%@ include file = "../WEB-INF/includes/body/include_example.txt" %>
    </p>
</div>
```

Το αποτέλεσμα του κώδικα:



## Examples of the Include Action

This is some sample text from /includes/body/include\_example.jsp. Thu May 18 19:15:12 EEST 2017  
This is some sample text from /includes/body/include\_example.txt. <%=new java.util.Date().toString()%>

## Examples of the Include Directive

This is some sample text from /includes/body/include\_example.jsp. Thu May 18 19:15:12 EEST 2017  
This is some sample text from /includes/body/include\_example.txt. Thu May 18 19:15:12 EEST 2017

Παρατηρούμε ότι όπως και στην περίπτωση του παραδείγματος στον **RequestResponseAndSessionExaminationServlet** το include action συμπεριφέρεται κατά τον ίδιο τρόπο όπως και προηγουμένως, όσον αφορά το expression <%= new java.util.Date().toString() %> κάτι που είναι αναμενόμενο.

Στην περίπτωση του **include directive** βλέπουμε ότι δεν έχει σημασία εάν το αρχείο είναι με κατάληξη jsp ή txt. Παρόλο που ένα txt αρχείο είναι απλά ένα static resource το αποτέλεσμα είναι να λάβουμε δυναμικό περιεχόμενο. Αυτό έγινε διότι το expression <%= new java.util.Date().toString() %> αντιγράφτηκε στον κώδικα της welcome.jsp πριν αυτή γίνει translated σε Servlet από τον Application Server.

Άλλη μια διαφορά μεταξύ του include action και του include directive αφορά την περίπτωση που γίνει κάποια αλλαγή στον πόρο που γίνεται include ενώ η εφαρμογή είναι ανεβασμένη στον server και εκτελείται. Εάν ο πόρος αυτός είναι κάποιο static resource, **στην περίπτωση του include action δεν χρειάζεται redeployment της εφαρμογής** ενώ εάν είναι κάποιος Servlet/JSP χρειάζεται μόνο στην περίπτωση που δεν έχει ρυθμιστεί να γίνεται polling για αλλαγές σε αυτά από τον server. **Στην περίπτωση του include directive θα χρειαστεί redeployment της εφαρμογής σε κάθε περίπτωση.**

# Filters

## Γενική Περιγραφή

### Θεωρητικό Υπόβαθρο

Τα Filters στο πρότυπο των Servlets είναι μια έννοια/συστατικό που σε άλλα προγραμματιστικά μοντέλα εμφανίζεται ως **Cross Cutting Concerns (Horizontal Concerns)**, **Aspects**, **Aspect Oriented Programming (AOP)** και **Interceptors**. Όλες οι προαναφερθείσες έννοιες αναφέρονται ουσιαστικά στην ίδια θεμελιώδη **ιδέα του καθορισμού και της εκτέλεσης μίας γενικής λειτουργικότητας/προεπεξεργασίας** ορισμάτων που είναι κοινή μεταξύ πολλών διαφορετικών μεθόδων/συναρτήσεων.

Αυτή η ιδέα επεκτείνεται στον ορισμό **pointcuts**, δηλαδή σημείων μέσα σε συναρτήσεις στα οποία μπορεί να εισαχθεί έκ των υστέρων γενικός κώδικας ο οποίος θα δύναται μεταγενέστερα να αλλάξει από ένα κεντρικό σημείο για όλα τα σημεία (pointcuts). Αυτή η επέκταση (η έννοια των pointcuts) δεν θα διερευνηθεί στην παρούσα εργασία.

Η μορφή του AOP που θα εξετάσουμε είναι αυτή των **Filters** και αφορά τον ορισμό αλυσίδων συστατικών για τα εξής:

- Για την **προεπεξεργασία** των **Request Parameters** και **Headers** πριν την επεξεργασία του Request από το τελικό Servlet ή την JSP ή ένα οποιοδήποτε άλλο Static Resource για το οποίο προορίζεται.

**Σημειώνεται ότι δεν μπορεί αντίστοιχα να γίνει επεξεργασία των Response Headers και του Response Body μετά από την επεξεργασία του request από τον τελικό Servlet αφού το πρότυπο ορίζει ότι θα πρέπει να γίνει commit του response αμέσως αφού τελειώσει η μέθοδος `HttpServlet.service(HttpServletRequest, HttpServletResponse)`.**

- Για την **ενθυλάκωση** των HttpServletRequest και HttpServletResponse αντικειμένων σε **wrapper** κλάσεις οι οποίες θα κάνουν override τις μεθόδους αυτών των interfaces.

Ο λόγος είναι ο ορισμός επιπλέον επεξεργασίας των headers και του σώματος του response ή των παραμέτρων του request πριν αυτά σταλθούν από την τελική μέθοδο επεξεργασίας του request (**HttpServlet.service**) στον πελάτη. Αντίθετα με την προεπεξεργασία του request η οποία μπορεί να γίνει χωρίς τον ορισμό wrapper κλάσεων όσον αφορά την επεξεργασία του response body οι wrapper κλάσεις είναι η μόνη λύση.

Η προεπεξεργασία αυτή μέσω των wrapper κλάσεων γίνεται ορίζοντας μία κλάση η οποία κάνει extend την HttpServletRequest ή την HttpServletResponse. Αυτή η κλάση ορίζεται να παίρνει ως όρισμα στον Constructor της το αρχικό HttpServletRequest/HttpServletResponse αντικείμενο. Η wrapper κλάση πρέπει επίσης να ορίζει μεθόδους που θα κάνουν override όλες τις μεθόδους των HttpServletRequest/HttpServletResponse κλάσεων. Αυτές οι μέθοδοι καλούνται αντίστοιχες μεθόδους του αρχικού HttpServletRequest/HttpServletResponse αντικειμένου που πέρασε ως όρισμα στον Constructor, εκτός από αυτό όμως πραγματοποιούνε προεπεξεργασία στις παραμέτρους που δέχονται ή/και κάνουν

επεξεργασία του αποτελέσματος που επιστρέφεται από τις μεθόδους του αρχικού αντικειμένου.

Ο πιο απλός τρόπος π.χ. για να εφαρμοστεί επεξεργασία του response body είναι να φτιάξουμε wrapper κλάση για το output stream του response. Αυτή κλάση θα μπορούσε αντί να γράφει κατευθείαν στο αρχικό output stream, να γράφει σε ένα buffer. Ο κώδικας της μεθόδου flush της wrapper κλάσης όμως θα μπορούσε να εφαρμόζει οποιαδήποτε επεξεργασία θέλει στα δεδομένα του buffer (π.χ. ορθογραφικό έλεγχο, συμπίεση, κρυπτογράφηση κ.τ.λ.) και στην συνέχεια να χρησιμοποιεί τις μεθόδους write του αρχικού output stream για να γράψει τα επεξεργασμένα πια δεδομένα του buffer στο αρχικό output stream. Στην συνέχεια, δηλαδή αφού γίνει η μεταφορά των επεξεργασμένων δεδομένων από το προσωρινό buffer προς το αρχικό output stream αντικείμενο, η μέθοδος flush της wrapper κλάσης αρκεί να καλεί την μέθοδο flush του αρχικού output stream του ServletResponse.

Είναι προφανές ότι θα μπορούσαμε ενθυλακώσουμε wrapper αντικείμενα σε άλλα wrapper αντικείμενα ώστε να δημιουργήσουμε αλυσιδωτή επεξεργασία. Π.χ. να δώσουμε το αρχικό response αντικείμενο σαν όρισμα για μια wrapper κλάση συμπίεσης και αυτήν σαν όρισμα σε μια wrapper κλάση κρυπτογράφησης. Τελικά όταν κληθεί η μέθοδος flush τα δεδομένα του response body πρώτα θα συμπιεστούν, μετά θα κρυπτογραφηθούν και τελικά θα σταλθούν στον πελάτη (κατά την κλήση της flush μεθόδου του output stream του αρχικού response αντικειμένου).

## Δημιουργία/Ορισμός, Lifecycle και Concurrency

### Δημιουργία/Ορισμός

Σκοπός κατά την δημιουργία των Filters είναι να ορίσουμε έναν αριθμό από Filter κλάσεις όπου η κάθε μια θα προορίζεται για την επεξεργασία, με κάποιον τρόπο, των request και response αντικειμένων κυρίως πριν αυτά επεξεργαστούν από την doGet (ή doPost κ.τ.λ.) μέθοδο του Servlet για τον οποίον προορίζονται.

Είναι επίσης δυνατή η εκτέλεση ενεργειών και μετά την ολοκλήρωση της HttpServlet.service (δηλαδή της doGet, doPost κ.τ.λ.) αλλά το Servlet Specification ορίζει ότι το response θα πρέπει να γίνει commit με το πέρας αυτών των μεθόδων άρα και οποιεσδήποτε μεταβολές στο σώμα ή στους headers του response δεν θα έχουν νόημα από εκεί και πέρα.

Επίσης πολλοί Application Servers δεν ακολουθούν κατά γράμμα αυτήν την οδηγία και επιστρέφουν το response αργότερα που σημαίνει ότι η μετέπειτα επεξεργασία του response ενδέχεται και να πετύχει αλλά ακόμα και αν πετύχει μια φορά δεν υπάρχει καμία εγγύηση ότι ένας τέτοιος Application Server θα το κάνει πάντα. Εξαιτίας αυτής της απρόβλεπτης συμπεριφοράς η μεταβολή του response μετά το πέρας της HttpServlet.service θα πρέπει πάντα να αποφεύγεται.

Για να ορίσουμε ένα Filter αρκεί να φτιάξουμε μία κλάση η οποία θα υλοποιεί το interface **javax.servlet.Filter** το οποίο περιλαμβάνει τις μεθόδους **init**, **doFilter** και **destroy**, οι μέθοδοι αυτές είναι παρεμφερείς με τις αντίστοιχες των Servlets. Επίσης θα πρέπει να δηλώσουμε το filter και τα URL patterns στα οποία αντιστοιχεί στον deployment descriptor.

Η μέθοδος `javax.servlet.Filter.doFilter(HttpServletRequest req, HttpServletResponse resp, FilterChain fChain)` είναι αυτή που θα πρέπει να γίνει overridden από τον προγραμματιστή που υλοποιεί ένα νέο φίλτρο.

Για να αντιστοιχίσουμε ένα Servlet σε ένα σύνολο από πόρους θα πρέπει να το ορίσουμε στον deployment descriptor, το ίδιο ισχύει και για ένα Filter. Τα elements που χρησιμοποιούμε είναι τα `<filter>`, `<filter-name>`, `<filter-class>`, `<filter-mapping>` και `<url-pattern>`.

Η σειρά δήλωσης των Filters στον deployment descriptor έχει σημασία καθώς εάν αντιστοιχούν πάνω από ένας Filter σε ένα request θα εκτελεσθούν ο ένας μετά τον άλλον και η σειρά που θα περάσει το request και response αντικείμενο μέσα από αυτά είναι ίδια με την σειρά με την οποία είναι δηλωμένοι στον deployment descriptor.

Η προεπεξεργασία του request και του response που προορίζονται για πόρο στον οποίον αντιστοιχούν πολλαπλά Filters γίνεται αλυσιδωτά. Πρώτα εκτελείται η μέθοδος `Filter.doFilter` του πρώτου Filter (κατά σειρά ορισμού στο deployment descriptor), στην συνέχεια, αφού γίνουν οι ενέργειες για την προεπεξεργασία, καλείται μέσα στην μέθοδο αυτή η `FilterChain.doFilter` η οποία μεταβιβάζει το request και response αντικείμενο στο επόμενο κατά σειρά Filter που είναι ορισμένο στο deployment descriptor κ.ο.κ..

## Lifecycle & Concurrency

Φτιάχνεται ένα Filter ανά JVM το οποίο εξυπηρετεί παράλληλα όλα τα requests στα οποία αντιστοιχεί. Αντίθετα με την περίπτωση των Servlets **το Servlet Specification δεν αναφέρει κάτι συγκεκριμένο για την διάρκεια ζωής του Filter** όμως καλό είναι να υποθέτουμε πάντα ότι ισχύει και για τα Servlets. Ότι δηλαδή μπορεί να καταστραφεί και να δημιουργηθεί εκ νέου οποιαδήποτε στιγμή. Οι μέθοδοι `init` και `destroy` καλούνται στις αντίστοιχες περιπτώσεις.

Όσον αφορά το Thread το οποίο εκτελεί την μέθοδο `doFilter`, εξασφαλίζεται από τον container ότι θα είναι το ίδιο με αυτό που θα εκτελέσει και την `doService` μέθοδο του Servlet για το οποίο προορίζεται το request.

## Παραδείγματα με Κώδικα

Παρακάτω παρουσιάζεται ο κώδικας των παραδειγμάτων.

### AuditingFilter1

Το `AuditingFilter1` εκτελεί την πλήρη καταγραφή του request, καθώς και κάποια άλλα πράγματα, στο default output stream του server. Ο κώδικάς του μοιάζει με αυτόν του `RequestResponseAndSessionExaminationServlet` με την διαφορά ότι ο `RequestResponseAndSessionExaminationServlet` παρουσίαζε τα αποτελέσματά του σαν μία HTML σελίδα.

```

1 package osotnikov.demowebapp.web.filter;
2
3 import java.io.IOException;
4
5 // @WebFilter(urlPatterns={"/"})
6 public class AuditingFilter1 implements Filter{
7
8     @Inject
9     private WebUtils webUtils;
10
11     boolean enabled = false;
12
13     @Override
14     public void init(FilterConfig filterConfig) throws ServletException {
15
16         System.out.println("AuditingFilter1.init was called.");
17
18         enabled = Boolean.valueOf(filterConfig.getInitParameter("enabled"));
19         System.out.println("AuditingFilter1.init enabled: " + enabled);
20     }
21
22     @Override
23     public void doFilter(ServletRequest req, ServletResponse res,
24             FilterChain filterChain) throws IOException, ServletException {
25
26         if(!enabled){
27             filterChain.doFilter(req, res);
28             return;
29         }
30
31         HttpServletRequest request = (HttpServletRequest) req;
32
33         System.out.println("\nAuditingFilter1.doFilter started, for request url: " + request.getRequestURL() + "\n");
34
35         HttpServletResponse response = (HttpServletResponse) res;
36
37
38         System.out.println("\nRECEIVED REQUEST\n");
39         // The address and port of the remote machine that made this request or the last proxy (may
40         // also be some gateway).
41         System.out.println("REMOTE ADDRESS: " + request.getRemoteAddr());
42         System.out.println("REMOTE PORT: " + request.getRemotePort());
43         // May return the address again and not translate it...
44         System.out.println("REMOTE HOST: " + request.getRemoteHost());
45         // Same as the host I guess. It's supposed to favor the Host header value of the request and
46         // not the DNS resolved value...
47         System.out.println("SERVER NAME: " + request.getServerName());
48         // The context path from the runtime descriptor...
49         System.out.println("CONTEXT PATH: " + request.getContextPath());
50         if(request.getContextPath() != null){
51             System.out.println("DECODED CONTEXT PATH: " + URLDecoder.decode(request.getContextPath(), "UTF-8"));
52         }
53         // The part of the request URL that matches the longest possible url-pattern of the
54         // Servlet that will service this request.
55         System.out.println("SERVLET PATH: " + request.getServletPath());
56         System.out.println("PATH INFO: " + request.getPathInfo());
57         if(request.getPathInfo() != null){
58             System.out.println("DECODED PATH INFO: " + URLDecoder.decode(request.getPathInfo(), "UTF-8"));
59         }
60         System.out.println("QUERY STRING: " + request.getQueryString());
61         if(request.getQueryString() != null){
62             System.out.println("DECODED QUERY STRING: " + URLDecoder.decode(request.getQueryString(), "UTF-8"));
63         }
64         // Will return the path of the resource on the server's filesystem. Basically it appends the path
65         // info to the expanded directory of the application on the server's filesystem (found by experimenting,
66         // java documentation and the servlet specification do not state that clearly). Therefore if the
67         // application is archived it will return null.
68         System.out.println("TRANSLATED PATH: " + request.getPathTranslated());
69         System.out.println("SCHEME: " + request.getScheme());
70         // The request URI as inside the first line of the HTTP request. In the majority of times it will
71         // be a relative uri but there is a chance that if there is an inbound proxy present that it will
72         // be an absolute uri. The javadocs say that it will contain everything, from the protocol (scheme)
73         // up to but not including the query string, but if the uri was relative obviously there will be no
74
75
76
77
78
79
80
81
82
83
84
85
86

```

```

87 // scheme or host & port information (which is the majority of the time).
88 System.out.println("REQUEST URI: "+request.getRequestURI());
89 System.out.println("DECODED REQUEST URI: "+URLDecoder.decode(request.getRequestURI(),"UTF-8"));
90 // The full request uri minus the query string.
91 System.out.println("REQUEST URL: "+request.getRequestURL());
92 System.out.println("DECODED REQUEST URL: "+URLDecoder.decode(request.getRequestURL().toString(),"UTF-8"));
93 // Authentication type that is used to protect the target Servlet (maybe generally a resource).
94 // Types are {BASIC, DIGEST, FORM, CLIENT CERTIFICATE}. We'll always use FORM. If none is used
95 // the this method returns null.
96 System.out.println("AUTHENTICATION TYPE: "+request.getAuthType());
97 // With some authentication schemes the remote user may be the user's username only after
98 // he is authenticated for the first time and after that it may just be returning null. I
99 // guess this applies to the BASIC authentication scheme (must investigate). As far as the
100 // FORM authentication scheme is concerned it will return the user's username as long as he's
101 // logged in (just like the getName() of the user's principal). Just always get the user's
102 // username by the getName of the user's principal.
103 System.out.println("REMOTE USER: "+request.getRemoteUser());
104 Principal princ = request.getUserPrincipal();
105 System.out.println("USER PRINCIPAL: "+princ);
106 if(princ != null){
107     // Always use this:
108     System.out.println("USER PRINCIPAL NAME: "+request.getUserPrincipal().getName());
109 }
110 // The address and port of the system that received this request.
111 System.out.println("LOCAL ADDRESS: "+request.getLocalAddr());
112 System.out.println("LOCAL PORT: "+request.getLocalPort());
113 System.out.println("LOCAL NAME: "+request.getLocalName()); // The host name of the address if it had been resolved
114 System.out.println("PROTOCOL USED: "+request.getProtocol());
115 System.out.println("IS SECURE: "+request.isSecure()); // Is a secure protocol (like https) used?
116 System.out.println("METHOD: "+request.getMethod()); // e.g. GET, POST etc.
117 System.out.println("LOCALE: "+request.getLocale()); // "Accept" header stuff...
118 // If the requested session id does not exist in the server then maybe the next if will not
119 // enter inside the case... All other times these two will be the same...
120 System.out.println("REQUESTED SESSION ID: "+request.getRequestedSessionId());
121 if(request.getSession() != null){
122     System.out.println("SESSION ID: "+request.getSession().getId());
123 }
124 System.out.println("CONTENT TYPE: "+request.getContentType());
125 System.out.println("CONTENT LENGTH: "+request.getContentLength());
126 System.out.println("CHARACTER ENCODING: "+request.getCharacterEncoding());
127
128 System.out.println("\nREQUEST HEADERS\n");
129 String reqHeadersStr = webUtils.getStringWithAllTheRequestHeaders(request, "\n");
130 System.out.println(reqHeadersStr);
131 System.out.println("\nREQUEST PARAMETERS\n");
132 String reqParamsStr = webUtils.getStringWithAllTheRequestParameters(request, "\n");
133 System.out.println(reqParamsStr);
134 System.out.println("\nREQUEST ATTRIBUTES\n");
135 String reqAttsStr = webUtils.getStringWithAllTheRequestAttributes(request, "\n");
136 System.out.println(reqAttsStr);
137
138 System.out.println("\nREQUEST COOKIES\n");
139 String reqCookies = webUtils.getStringWithAllTheRequestCookies(request, "\n");
140 System.out.println(reqCookies);
141
142 System.out.println("SESSION INFO\n");
143 HttpSession session = request.getSession();
144 System.out.println("isNew: " + session isNew());
145 System.out.println("sessionId: " + session.getId());
146 System.out.println("creationTime: " + new java.util.Date(session.getCreationTime()));
147 System.out.println("lastAccessedTime: " + new java.util.Date(session.getLastAccessedTime()));
148 System.out.println("maxInactiveInterval: " + session.getMaxInactiveInterval());
149
150 System.out.println("\nSESSION ATTRIBUTES\n");
151 String sessAttsStr = webUtils.getStringWithAllTheSessionAttributes(request.getSession(), "\n");
152 System.out.println(sessAttsStr);
153
154 // AuditingFilter2 handles the logging of the response.
155 filterChain.doFilter(request, response);
156
157 System.out.println("\nAuditingFilter1.doFilter ended, for request url: " + request.getRequestURL() + "\n");
158 }
159
160 @Override
161 public void destroy() {
162     System.out.println("AuditingFilter1.destroy was called.");
163 }
164
165

```

Παρατηρούμε ότι προς το τέλος της μεθόδους doFilter (γραμμή 155) εκτελείται η μεταβίβαση των request, response αντικειμένων στον επόμενο σε σειρά Filter ή εάν δεν υπάρχει άλλος, στον τελικό πόρο για τον οποίον προορίζονταν το request. Η μεταβίβαση αυτή επιτυγχάνεται με την μέθοδο doFilter ενός FilterChain αντικειμένου.

To instance του FilterChain αντικειμένου που χρειαζόμαστε έρχεται ως όρισμα στην `HttpFilter.doFilter(ServletRequest, ServletResponse, FilterChain)`. Εάν ξεχάσουμε δηλαδή να καλέσουμε την `FilterChain.doFilter` μέθοδο το request δεν θα φτάσει στον Servlet για τον οποίον προοριζόταν.

Ο ορισμός του **AuditingFilter1** στον deployment descriptor:

```

160<filter>
161    <filter-name>AuditingFilter1</filter-name>
162    <filter-class>
163        osotnikov.demowebapp.web.filter.AuditingFilter1
164    </filter-class>
165
166    <init-param>
167        <param-name>enabled</param-name>
168        <param-value>false</param-value>
169    </init-param>
170
171</filter>
172<filter-mapping>
173    <filter-name>AuditingFilter1</filter-name>
174    <url-pattern>
175        /*
176    </url-pattern>
177
178    <!-- REQUEST Type Dispatcher means that the request is going to pass through the filter
179    only when it's coming directly from the client's browser. It is the default and can be omitted
180    if no other dispatcher type is declared... -->
181    <dispatcher>REQUEST</dispatcher>
182    <!-- If you want the request to pass through the filter only when it is being forwarded by
183    the error handling mechanism towards the error page you must specify dispatcher type ERROR only. -->
184    <dispatcher>ERROR</dispatcher>
185    <!-- You can combine dispatcher declarations to handle the corresponding cases. -->
186</filter-mapping>
187

```

Παρατηρούμε ότι ορίζεται με παρόμοιο τρόπο όπως και ένας Servlet. Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα elements **<dispatcher>**. Αυτά ορίζουν τους τύπους των requests που θα επεξεργάζεται το Filter. Με την βοήθειά τους μπορούμε να ορίσουμε έναν Filter να εκτελείται και στην περίπτωση ενός forward ή ενός include.

Εάν δεν δηλώσουμε κανένα **<dispatcher>** element είναι σαν να έχουμε δηλώσει μόνο το **REQUEST** dispatcher element. Στην περίπτωση του AuditingFilter1 έχουμε δηλώσει το REQUEST αλλά και το ERROR element. Το **ERROR** element έχει να κάνει με τον μηχανισμό χειρισμού σφαλμάτων που ορίζει το πρότυπο των Servlets όπου μπορεί να οριστούνε κάποιοι Servlets που θα επεξεργάζονται το request του χρήστη στην περίπτωση που το αρχικό request αποτύχει. Στην περίπτωση αυτή το request κατά την αυτόματη εσωτερική ανακατεύθυνσή του θα περάσει και από τους Filters που ορίζουν έναν dispatcher με type ERROR.

Παρατηρούμε επίσης και το element **<init-param>** το οποίο χρησιμοποιείται για να ορίζονται παράμετροι έξω από τον πηγαίο κώδικα του web application. Αυτή η παράμετρος διαβάζεται μέσα στην μέθοδο init του AuditingFilter1 (με την μέθοδο `FilterConfig.getInitParameter`) και ανάλογα με την τιμή της αποφασίζεται εάν αυτός ο Filter θα εφαρμόσει ή όχι την επεξεργασία που ορίζει. Παρόλο που η τιμή στην εικόνα είναι false προφανώς είχε τεθεί σε true πριν την εκτέλεση του κώδικα της οποίας τα αποτελέσματα παρουσιάζονται στην συνέχεια του κεφαλαίου.

## AuditingFilter2

Το AuditingFilter2 καταγράφει το response αντικείμενο αφού ολοκληρωθεί πιά η επεξεργασία από τον Servlet για τον οποίον προορίζεται το request και η service μέθοδος επιστρέψει. Καλεί δηλαδή πρώτα την μέθοδο FilterChain.doFilter και αφού αυτή επιστρέψει (δηλαδή αφού ολοκληρωθεί η doService του τελικού Servlet και επιστραφεί HTTP response στον πελάτη, ή του επιστραφεί κάποιο static resource εάν δεν πρόκειται για Servlet) συνεχίζει με την εκτέλεση του δικού της κώδικα.

Ο AuditingFilter2 παρουσιάζει ένα παράδειγμα εκτέλεσης κάποιας λογικής αφού πια επιστραφεί το response στον πελάτη, στην δικιά μας περίπτωση απλώς καταγράφει το response αντικείμενο στο default output stream.

```
1 package osotnikov.demowebapp.web.filter;
2
3 import java.io.IOException;
4
5 // @WebFilter(urlPatterns={"/"})
6 public class AuditingFilter2 implements Filter{
7
8     @Inject
9     private WebUtils webUtils;
10
11     boolean enabled = false;
12
13     @Override
14     public void init(FilterConfig filterConfig) throws ServletException {
15         System.out.println("AuditingFilter2.init was called.");
16
17         enabled = Boolean.valueOf(filterConfig.getInitParameter("enabled"));
18         System.out.println("AuditingFilter2.init enabled: " + enabled);
19     }
20
21     @Override
22     public void doFilter(ServletRequest req, ServletResponse res,
23             FilterChain filterChain) throws IOException, ServletException {
24
25         if(!enabled){
26             filterChain.doFilter(req, res);
27             return;
28         }
29
30         HttpServletRequest request = (HttpServletRequest) req;
31
32         System.out.println("\nAuditingFilter2.doFilter started, for request url: " + request.getRequestURL() + "\n");
33
34         HttpServletResponse response = (HttpServletResponse) res;
35
36         // AuditingFilter1 handles the logging of the request.
37
38         filterChain.doFilter(request, response);
39
40         System.out.println("\nRESPONSE\n");
41         System.out.println("STATUS: " + response.getStatus());
42         System.out.println("BUFFER SIZE: " + response.getBufferSize());
43         System.out.println("CONTENT TYPE: " + response.getContentType());
44         System.out.println("CHARACTER ENCODING: " + response.getCharacterEncoding());
45         System.out.println("LOCALE: " + response.getLocale());
46         System.out.println("IS COMMITTED: " + response.isCommitted());
47
48         System.out.println("\nRESPONSE HEADERS\n");
49
50         String respHeaders = webUtils.getStringWithAllTheResponseHeaders(response, "\n");
51         System.out.println(respHeaders);
52
53         System.out.println("\nAuditingFilter2.doFilter ended, for request url: " + request.getRequestURL() + "\n");
54     }
55
56     @Override
57     public void destroy() {
58         System.out.println("AuditingFilter2.destroy was called.");
59     }
60 }
61
62 }
```

## Ο ορισμός του AuditingFilter2 στον deployment descriptor:

```
188<filter>
189  <filter-name>AuditingFilter2</filter-name>
190  <filter-class>
191      osotnikov.demowebapp.web.filter.AuditingFilter2
192  </filter-class>
193
194<init-param>
195     <param-name>enabled</param-name>
196     <param-value>false</param-value>
197 </init-param>
198
199</filter>
200<filter-mapping>
201     <filter-name>AuditingFilter2</filter-name>
202     <url-pattern>
203         /*
204     </url-pattern>
205     <dispatcher>REQUEST</dispatcher>
206 </filter-mapping>
```

## FilterForForwards

Ο Filter αυτός απλώς κάνει μία καταγραφή στο default output stream την στιγμή που ξεκινάει και την στιγμή που τελειώνει. Στην συνέχεια στον deployment descriptor κατά τον ορισμό του θα ορίσουμε να έχει μοναδικό dispatcher με type FORWARD.

```
1 package osotnikov.demowebapp.web.filter;
2
3 import java.io.IOException;
4
5 public class FilterForForwards implements Filter{
6
7     boolean enabled = false;
8
9     @Override
10    public void init(FilterConfig filterConfig) throws ServletException {
11        System.out.println("FilterForForwards.init was called.");
12
13        enabled = Boolean.valueOf(filterConfig.getInitParameter("enabled"));
14        System.out.println("FilterForForwards.init enabled: " + enabled);
15    }
16
17    @Override
18    public void doFilter(ServletRequest req, ServletResponse res,
19                          FilterChain filterChain) throws IOException, ServletException {
20
21        if(!enabled){
22            filterChain.doFilter(req, res);
23            return;
24        }
25
26        HttpServletRequest request = (HttpServletRequest)req;
27        System.out.println("\nFilterForForwards.doFilter started, for request url: " + request.getRequestURL() + "\n");
28        filterChain.doFilter(req, res);
29        System.out.println("\nFilterForForwards.doFilter ended, for request url: " + request.getRequestURL() + "\n");
30    }
31
32    @Override
33    public void destroy() {
34        System.out.println("FilterForForwards.destroy was called.");
35    }
36}
```

## Ο ορισμός του FilterForForwards στον deployment descriptor:

```

208<filter>
209    <filter-name>FilterForForwards</filter-name>
210    <filter-class>
211        osotnikov.demowebapp.web.filter.FilterForForwards
212    </filter-class>
213
214    <init-param>
215        <param-name>enabled</param-name>
216        <param-value>false</param-value>
217    </init-param>
218
219</filter>
220<filter-mapping>
221    <filter-name>FilterForForwards</filter-name>
222    <url-pattern>
223        /*
224        </url-pattern>
225        <!-- If you want the request to only pass through the filter when it is being forwarded you must
226        specify dispatcher type FORWARD only. --&gt;
227        &lt;dispatcher&gt;FORWARD&lt;/dispatcher&gt;
228    &lt;/filter-mapping&gt;
229
</pre>

```

## FilterForIncludes

O Filter αυτός απλώς κάνει μία καταγραφή στο default output stream την στιγμή που ξεκινάει και την στιγμή που τελειώνει. Στην συνέχεια στον deployment descriptor κατά τον ορισμό του θα ορίσουμε να έχει μοναδικό dispatcher με type **INCLUDE**.

```

1 package osotnikov.demowebapp.web.filter;
2
3 import java.io.IOException;
4
5 public class FilterForIncludes implements Filter{
6
7     boolean enabled = false;
8
9     @Override
10    public void init(FilterConfig filterConfig) throws ServletException {
11        System.out.println("FilterForIncludes.init was called.");
12
13        enabled = Boolean.valueOf(filterConfig.getInitParameter("enabled"));
14        System.out.println("FilterForIncludes.init enabled: " + enabled);
15    }
16
17    @Override
18    public void doFilter(ServletRequest req, ServletResponse res,
19        FilterChain filterChain) throws IOException, ServletException {
20
21        if(!enabled){
22            filterChain.doFilter(req, res);
23            return;
24        }
25
26        HttpServletRequest request = (HttpServletRequest)req;
27
28        System.out.println("\nFilterForIncludes.doFilter started, for request url: " + request.getRequestURL() + "\n");
29        filterChain.doFilter(req, res);
30        System.out.println("\nFilterForIncludes.doFilter ended, for request url: " + request.getRequestURL() + "\n");
31    }
32
33    @Override
34    public void destroy() {
35        System.out.println("FilterForIncludes.destroy was called.");
36    }
37
38
39
40
41
42
43
44
45

```

Ο ορισμός του **FilterForIncludes** στον deployment descriptor:

```

230<filter>
231  <filter-name>FilterForIncludes</filter-name>
232<filter-class>
233  osotnikov.demowebapp.web.filter.FilterForIncludes
234</filter-class>
235
236<init-param>
237  <param-name>enabled</param-name>
238  <param-value>false</param-value>
239</init-param>
240
241</filter>
242<filter-mapping>
243  <filter-name>FilterForIncludes</filter-name>
244<url-pattern>
245  /*
246  </url-pattern>
247<!-- If you want the request to pass through the filter only when it is being included through
248 an include action (obviously not an include directive) you must specify dispatcher type INCLUDE only. --&gt;
249 &lt;dispatcher&gt;INCLUDE&lt;/dispatcher&gt;
250&lt;/filter-mapping&gt;
</pre>

```

## Παράδειγμα Εκτέλεσης όλων των Filters

### Περιγραφή

Σε αυτό το σημείο θα εκτελεσθεί ένα request προς τον **ForwardingAndRedirectionServlet** το οποίο θα εκτελέσει forward προς το **RequestResponseAndSessionExaminationServlet** για να δούμε και από το output στην κονσόλα του server την σειρά εκτέλεσης των filters.

Προφανώς, για τις ανάγκες αυτού του παραδείγματος όλα τα `<init-param>` elements δηλώνουν την τιμή “true” για την παράμετρο “enabled” στα αντίστοιχα `<filter>` elements των filters.

Επειδή όλες οι `doFilter` μέθοδοι που ορίσαμε καθώς και η `RequestResponseAndSessionExaminationServlet.doGet` που θα επεξεργασθεί το request καταγράφουν την αρχή και το τέλος της εκτέλεσής τους θα είναι εύκολο να παρακολουθήσουμε την πορεία του request μέσα σε αυτά.

Επίσης το filter **FilterForForwards** είχε οριστεί στον deployment descriptor με dispatcher type **FORWARD**, για αυτό θα εκτελεσθεί κατά το forward της `ForwardingAndRedirectionServlet` προς την `RequestResponseAndSessionExaminationServlet`.

Όπως είχαμε δει και από τα προηγούμενα παραδείγματα η `RequestResponseAndSessionExaminationServlet` περιλαμβάνει τρία παραδείγματα με `include` action. Επειδή το filter **FilterForIncludes** είχε οριστεί στον deployment descriptor με dispatcher type **INCLUDE** το filter αυτό θα εκτελεσθεί για το κάθε ένα από τα τρία `includes`.

### Εκτέλεση Παραδείγματος

Αρχικά, αφού έχουμε ξεκινήσει τον server μέσα από το Eclipse, μεταβαίνουμε με έναν browser στην σελίδα του DemoWebApp. Σε αυτό το σημείο είναι καλό να κάνουμε ένα “Clear” στο tab “Console” του Eclipse.

Στην συνέχεια, στην `welcome.jsp`, στην φόρμα **“Forwarding and Redirection Example”** επιλέγουμε το **“Transfer to the request, response & session examination page by forwarding:”**

για να πραγματοποιήσουμε request προς τον ForwardingAndRedirectionServlet που θα κάνει forward στην RequestResponseAndSessionExaminationServlet.

Current Date: 19/5/2017

Good Evening!

[Go to the Request Response & Session Examination Page](#)

Forwarding and Redirection Example

Message for intermediate processing: This is a sample message.

Transfer to the request, response & session examination page by forwarding:

Transfer to the request, response & session examination page by internal redirection:

Transfer to the youtube by external redirection:

Submit

Αφού παρουσιαστεί μπροστά μας η σελίδα του RequestResponseAndSessionExaminationServlet εξετάζουμε το output που δημιουργήθηκε στην κονσόλα του Eclipse.

```
Markers Properties Servers Data Source Explorer Snippets Console Search REST Annotations WebLogic Web Service Annotations Error Log
Oracle WebLogic Server 12c R2 (12.2.1) at localhost [demoWebApp] [Oracle WebLogic Server] Weblogic Server

AuditingFilter1.doFilter started, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet

RECEIVED REQUEST

REMOTE ADDRESS: 0:0:0:0:0:0:0:1
REMOTE PORT: 62936
REMOTE HOST: 0:0:0:0:0:0:1
SERVER NAME: localhost
CONTEXT PATH: /DemoWebApp
DECODED CONTEXT PATH: /DemoWebApp
SERVLET PATH: /actions/RequestResponseAndSessionExaminationServlet
PATH INFO: null
QUERY STRING: null
TRANSLATED PATH: null
SCHEME: https
REQUEST URI: /DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
DECODED REQUEST URI: /DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
REQUEST URL: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
DECODED REQUEST URL: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
AUTHENTICATION TYPE: null
REMOTE USER: null
USER PRINCIPAL: null
LOCAL ADDRESS: 0:0:0:0:0:0:0:1
LOCAL PORT: 7002
LOCAL NAME: 0:0:0:0:0:0:1
PROTOCOL USED: HTTP/1.1
IS SECURE: true
METHOD: GET
LOCALE: en
REQUESTED SESSION ID: K1AhGqGHbdw7X9V782bWe7GU7cwhsSQY9M7Z9MFo2Rkj1QlJqH-TE!1413302094!1495204012425
SESSION ID: K1AhGqGHbdw7X9V782bWe7GU7cwhsSQY9M7Z9MFo2Rkj1QlJqH-TE!1413302094!1495204012425
CONTENT TYPE: null
CONTENT LENGTH: -1
```

CHARACTER ENCODING: null

REQUEST HEADERS

```
Host : localhost:7002
Connection : keep-alive
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36
Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer : https://localhost:7002/DemoWebApp/
Accept-Encoding : gzip, deflate, sdch, br
Accept-Language : en,el;q=0.8,ru;q=0.6
Cookie : JSESSIONID=K1AhGqGHbdw7X9V782bWe7GU7cwhSQY9M7Z9MFo2RkjQlJqH-TE!1413302094
testHeader1 : testHeader1_value
```

REQUEST PARAMETERS

REQUEST ATTRIBUTES

```
requestInitEventNotified : true
weblogic.servlet.request.sslsession : [Session-3, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256]
org.jboss.weld.servlet.ConversationContextActivator.contextActivatedInRequest : true
javax.servlet.request.key_size : 128
org.jboss.weld.context.ignore.guard.marker : java.lang.Object@55f968df
javax.servlet.request.cipher_suite : TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
weblogic.servlet.network_channel.sslport : 7002
javax.servlet.request.ssl_session_id : [B@55bde0c9
```

REQUEST COOKIES

```
domain: null, name: JSESSIONID, value: iYghb4Cui0eXROS1anIyK3tDWTAIZHH0uAsbzW5IRN3HTntZG0Gi!1413302094, is HttpOnly: false, maxAge: -1, path: null, is secure: false
SESSION INFO
isNew: false
sessionId: iYghb4Cui0eXROS1anIyK3tDWTAIZHH0uAsbzW5IRN3HTntZG0Gi!1413302094!1495209574574
creationTime: Fri May 19 18:59:34 EEST 2017
lastAccessedTime: Fri May 19 19:00:12 EEST 2017
maxInactiveInterval: 300
SESSION ATTRIBUTES
WELD_S_HASH : 2129431603

AuditingFilter2.doFilter started, for request url: https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet
ForwardingAndRedirectionServlet.doGet: started for request: https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet
FilterForForwards.doFilter started, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
RequestResponseAndSessionExaminationServlet.doPost: (started ... just calls doGet) ...
RequestResponseAndSessionExaminationServlet doGet: started for request: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
FilterForIncludes.doFilter started, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
FilterForIncludes.doFilter ended, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
FilterForIncludes.doFilter started, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
FilterForIncludes.doFilter ended, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
```

```

FilterForIncludes.doFilter started, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet

FilterForIncludes.doFilter ended, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
RequestResponseAndSessionExaminationServlet doGet: ended for request: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet

FilterForForwards.doFilter ended, for request url: https://localhost:7002/DemoWebApp/actions/RequestResponseAndSessionExaminationServlet
ForwardingAndRedirectionServlet doGet: ended for request: https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet

RESPONSE

STATUS: 200
BUFFER SIZE: 12216
CONTENT TYPE: text/html; charset=UTF-8
CHARACTER ENCODING: UTF-8
LOCALE: en_US
IS COMMITTED: true

RESPONSE HEADERS

Date: , contained in response: false, headerValue: null
Transfer-Encoding: , contained in response: false, headerValue: null
Content-Type: , contained in response: false, headerValue: null
testResponseHeader, contained in response: true, headerValue: testResonseHeader_val

AuditingFilter2.doFilter ended, for request url: https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet

AuditingFilter1.doFilter ended, for request url: https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet

```

Όπως παρατηρούμε πρώτα βλέπουμε ότι ξεκίνησε ο **AuditingFilter1** και στην συνέχεια εκτέλεσε την καταγραφή του request καθώς και άλλων πραγμάτων στο default output stream. Στην συνέχεια αυτός πραγματοποιεί κλήση στην FilterChain.doChain ώστε να επιτρέψει την επεξεργασία του request και του response από το επόμενο στοιχείο στην αλυσίδα. Αυτό το στοιχείο είναι το AuditingFilter2 που εκτελείται στην συνέχεια.

Στο κώδικα του **AuditingFilter2** που παρουσιάστηκε προηγουμένως παρατηρούμε ότι δεν γράφεται κάτι στο default output stream πριν την κλήση της FilterChain.doFilter. Αυτό έχει ως αποτέλεσμα μετά το ξεκίνημά του να βλέπουμε κατευθείαν το ξεκίνημα του ForwardingAndRedirectionServlet.

Ο **ForwardingAndRedirectionServlet** θα εκτελέσει το forward προς την RequestResponseAndSessionExaminationServlet οπότε αμέσως βλέπουμε και το ξεκίνημα του FilterForForwards.

Το **FilterForForwards** πέραν του να μεταβιβάζει το request και το response στο επόμενο στοιχείο στην αλυσίδα, το μόνο που κάνει είναι να καταγράφει την αρχή και το τέλος του οπότε αμέσως βλέπουμε και το ξεκίνημα του **RequestResponseAndSessionExaminationServlet**.

Κατά την εκτέλεση του RequestResponseAndSessionExaminationServlet γίνονται τρία include actions. Εκτελείται επομένως ο **FilterForIncludes** και καταγράφεται το ξεκίνημα και το σταμάτημά του 3 φορές.

Στην συνέχεια παρατηρούμε ότι **τελειώνει αλυσιδωτά η εκτέλεση των components RequestResponseAndSessionExaminationServlet μετά, FilterForForwards και μετά, ForwardingAndRedirectionServlet**.

Τώρα βρισκόμαστε **ακριβώς μετά την κλήση της FilterChain.doFilter στον AuditingFilter2**. Σε αυτό το σημείο ο AuditingFilter2 κάνει την καταγραφή του response.

Αφού τελειώσει βλέπουμε τον **AuditingFilter2** να καταγράφει το τελείωμά του και τον **AuditingFilter1** το δικό του.

## Αντιστοιχία των Requests στα Servlets (Servlet Mapping)

Σε αυτό το κεφάλαιο αναλύεται ο τρόπος με τον οποίον αντιστοιχίζουμε τα requests με βάση το uri στο οποίο προορίζονται με τα Servlets και JSP και κατ' επέκταση και με τα Filters.

Για να γίνει σωστά αρκεί να έχουμε υπόψη κάποιους απλούς κανόνες για τα url patterns που ορίζουμε στον deployment descriptor για τους Servlets, JSPs και Filters.

1. **Exact Mapping:** Το “Exact mapping” αναφέρεται σε οποιοδήποτε url pattern που δεν είναι της μορφής “”, “/”, “\*.extension” ή “.../\*”. Π.χ.  
“/actions/RequestResponseAndSessionExaminationServlet”. Σημειώνεται ότι όταν ο Application Server ελέγχει τα url patterns για να βρει τον πόρο (Servlet, Filter, JSP) ο οποίος ταιριάζει καλύτερα με το url του request δεν λαμβάνει υπόψιν του το query string που το url ενδέχεται να έχει. Κατ' αυτόν τον τρόπο requests με URLs που έχουν path “/actions/RequestResponseAndSessionExaminationServlet” ή  
“/actions/RequestResponseAndSessionExaminationServlet?param1=param1val&param2=param2val” θα αντιστοιχηθούν και τα δύο χωρίς πρόβλημα στον πόρο με url pattern “/actions/RequestResponseAndSessionExaminationServlet”.
2. **Path Mapping:** Τα URL patterns που είναι της μορφής “.../\*” κάνουν path mapping. Αυτό σημαίνει ότι όλα τα requests που προορίζονται για URLs που ξεκινάνε με το κομμάτι του url pattern πριν το “\*” και συνεχίζουνε με οποιαδήποτε συμβολοσειρά θα αντιστοιχηθούν στον πόρο στον οποίον ορίστηκε αυτό το url pattern.

- 3. Context Root Mapping:** Αυτό το mapping αντιστοιχεί στο context root. To url pattern του είναι η κενή συμβολοσειρά “”. Δηλαδή ένας πόρος με url pattern “” στην DemoWebApp εφαρμογή π.χ. θα επιστρέφονταν όταν θα επιστεφόμασταν το url “<http://localhost:7001>”. **Αυτό το url pattern δεν υποστηρίζεται από τις περισσότερες υλοποιήσεις και δεν θα πρέπει να χρησιμοποιείται.** Για αυτόν τον σκοπό καλύτερα να χρησιμοποιείται το <welcome-file-list> element του deployment descriptor.
- 4. Default Servlet Mapping:** Υπάρχει ένας “Default Servlet”. Ο Servlet αυτός αναλαμβάνει να στέλνει στον πελάτη static resources. Εκτελείται κάθε φορά που γίνεται request προς κάποιον πόρο για τον οποίον δεν υπάρχει κανένα url pattern στον deployment descriptor που να του αντιστοιχεί. Εάν οριστεί Servlet με url pattern “/” τότε αυτός θα αντικαταστήσει τον Default Servlet. Αυτό όμως δεν είναι σίγουρο ότι υποστηρίζεται από κάθε Application Servlet αν και με βάση το Servlet Specification θα έπρεπε. Ο καινούριος Servlet θα πρέπει να καλύπτει όλες τις λειτουργικότητες που κάλυπτε και ο αρχικός οπότε πρακτικά το url pattern αυτό αποφεύγεται να χρησιμοποιείται. Το “/” είναι ένα special mapping και αυτό επειδή εάν δεν ορίζονταν από το Servlet Specification ως τέτοιο θα ήταν ένα exact mapping. Εδώ απλώς τονίζεται ότι το “/” δεν είναι ένα exact mapping και δεν θα έπρεπε ποτέ να χρησιμοποιείται ως τέτοιο π.χ. για να ορίσουμε ένα welcome page. Για αυτόν τον σκοπό καλύτερα να χρησιμοποιείται το <welcome-file-list> element του deployment descriptor
- 5. Extension Mapping:** Το extension mapping υπάρχει για να αντιστοιχηθούν requests προς συγκεκριμένους τύπους αρχείων σε πόρους. Τα url patterns που αποτελούν extension mapping θα πρέπει να έχουν την μορφή “\*.extension” δεν θα είχε όμως νόημα να έχουν την μορφή “.../\*.extension” διότι δεν υποστηρίζεται η αντιστοίχιση με requests των οποίων το url ξεκινάει με ένα συγκεκριμένο path και τελειώνει με συγκεκριμένο extension αλλά το ενδιάμεσο τμήμα είναι μεταβλητό. Ένα παράδειγμα του extension mapping θα μπορούσε να είναι ένας Servlet με το url pattern “\*.img” στον deployment descriptor. Σε αυτήν την περίπτωση (εάν το παράδειγμα θεωρείται στα πλαίσια της εφαρμογής DemoWebApp) ακόμα και ένα request προς το url “<http://localhost:7001/DemoWebApp/path1/path2/someimg.jpg>” θα κατέληγε σε αυτόν τον Servlet.
- Προφανώς τα JSP και τα static resources που βρίσκονται έξω από τον WEB-INF είναι προσβάσιμα από το path που έχουν μέσα στο module που βρίσκονται (π.χ. war archive).
- Welcome File List:** Το <welcome-file-list> είναι ένα element που ορίζεται στον deployment descriptor. Σε περίπτωση που δεν βρεθεί κάποιος πόρος για να εξυπηρετηθεί ένα request προσπελάζονται με την σειρά τα <welcome-file> elements του. Κάθε <welcome-file> element ορίζει μια κατάληξη η οποία προστίθεται στο request url. Εάν κατά την προσπέλασή τους βρεθεί μια κατάληξη η οποία όταν προστεθεί στο url θα προκύψει ένα url που αντιστοιχεί σε πόρο του web application τότε σταματάει η προσπέλαση και επιστρέφεται αυτός ο πόρος.
- Εάν υπάρχουν πολλαπλοί πόροι που ταιριάζουν με ένα request τότε θα εξετασθούν τα url patterns τους για να επιλεχθεί αυτός του οποίου το url pattern ταιριάζει καλύτερα με το request url. Αρχικά εξετάζεται εάν υπάρχει κάποιο exact mapping. Εάν δεν υπάρχει, τότε από όλα τα path mappings που ταιριάζουν με το url επιλέγεται αυτό με το μεγαλύτερο μήκος. Εάν δεν υπάρχουν ούτε path mappings που να ταιριάζουν με το url τότε γίνεται αναζήτηση στο welcome file list για ένα suffix που όταν προστεθεί στο request url να έχει ως αποτέλεσμα ένα path για το οποίο να υπάρχει πόρος.

## Sessions

### Θεωρητική Περιγραφή

Το HTTP είναι ένα stateless πρωτόκολλο. Αυτό σημαίνει ότι δεν παρέχει από μόνο του κάποιον τρόπο για την συσχέτιση ενός request ενός χρήστη με ένα άλλο του ίδιου χρήστη. **Στο HTTP κάθε request είναι ανεξάρτητο από όλα τα άλλα.** Για αυτόν τον λόγο έχουν αναπτυχθεί διάφορες τεχνικές για την επίλυση αυτού του προβλήματος.

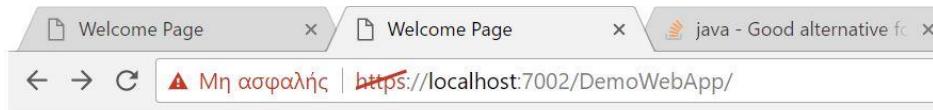
Η τεχνική που εφαρμόζεται κατά κόρον είναι η **αποθήκευση ενός Cookie στον πελάτη με ένα μοναδικό ID κατά την πρώτη σύνδεσή του με τον server**. Ο browser του πελάτη στην συνέχεια στέλνει αυτόματα αυτό το Cookie μαζί με κάθε επόμενο request που κάνει προς τον server και κατ' αυτόν τον τρόπο ο server ανά πάσα στιγμή μπορεί να γνωρίζει σε ποιόν πελάτη αντιστοιχεί το request που έλαβε. Επίσης ο server διατηρεί μία δομή για την αντιστοίχιση των session IDs με τα "Sessions" έτσι ώστε να είναι διαθέσιμο το Session του κάθε πελάτη μέσω της κλήσης `HttpServletRequest.getSession()`.

Εκτός από την τεχνική των Cookies υπάρχουν και άλλες τεχνικές όπως το **URL Rewriting** και τα **SSL IDs**.

### Κώδικας Παραδείγματος

Θα παρουσιασθεί ένα απλό παράδειγμα αποθήκευσης και ανάγνωσης δεδομένων από το Session.

Στο DemoWebApp υπάρχει η δυνατότητα για επιλογή του χρώματος του φόντου της κάθε σελίδας. Σε κάθε σελίδα προστέθηκε η παρακάτω φόρμα:



# Welcome to our secured Web Application

Current Date: 20/5/2017

Good Evening!

## Background Color Selector

Select the background color that you want:  Submit Color & Reload

Πατώντας στο τετράγωνο με το χρώμα επιλέγουμε το χρώμα ενώ πατώντας στο κουμπί "Submit Color & Reload" στέλνεται request προς τον servlet **ChooseCommonBackgroundColorServlet**. Ο servlet ChooseCommonBackgroundColorServlet απλώς αποθηκεύει το χρώμα που επιλέξαμε στο session.

### HTML κώδικας της φόρμας επιλογής χρώματος φόντου (bg-color-selector.html):

```
<h2>Background Color Selector</h2>
<form id="bgColorSelForm" name="bgColorSelForm">
  Select the background color that you want:
  <input type="color" name="chosen_bg_color" value="#000000">
  <button id="bgColorSelBut">Submit Color & Reload</button>
</form>
```

### Η javascript που αντιστοιχεί στην φόρμα επιλογής χρώματος φόντου (common-scripts.js):

```

1 $(function(){
2     "use strict";
3
4     $('#bgColorSelForm #bgColorSelBut').click(function(event){
5
6         var chosenBgColor = $('#bgColorSelForm input[name="chosen_bg_color"]').attr('value');
7         console.log('bgColorSelForm: chosen_bg_color:' + chosenBgColor);
8
9         $.ajax({
10             url: '<%=request.getContextPath()%>/actions/ChooseCommonBackgroundColorServlet',
11             type: "POST",
12             data: {"chosen_bg_color": chosenBgColor},
13             cache: false,
14             dataType: "json",
15
16             success: function (data, textStatus, jqXHR) {
17
18                 if (data.status == "SUCCESS") {
19
20                     $('#bgColorSelForm .action-report').text(
21                         'Successfully changed background color on all pages! The page is being reloaded...');
22                     window.location.href = window.location.href;
23                 } else {
24                     $('#bgColorSelForm .action-report').text('Failed to change the background color!');
25                 }
26
27             },
28
29             error: function (jqXHR, textStatus, errorThrown) {
30                 alert("error - HTTP STATUS: " + jqXHR.status);
31             }
32         });
33     });
34
35     return false;
36 });

```

Όπως παρατηρούμε η φόρμα **bgColorSelForm** αντί για ένα element τύπου submit έχει ένα απλό button element. Αυτό γίνεται διότι αντί να στείλουμε ένα απλό POST HTTP Request έχουμε ως στόχο, όταν ο πελάτης πατήσει το κουμπί της φόρμας, να εκτελεστεί javascript κώδικας που θα εκτελέσει ένα AJAX HTTP Request.

Ένα AJAX Request δεν θα έχει ως αποτέλεσμα το HTTP Response που θα έρθει να αντικαταστήσει ολόκληρη την σελίδα. Αντιθέτως όταν ληφθεί η απάντηση από τον server θα εκτελεσθεί javascript κώδικας που έχουμε ορίσει ώστε να διαχειριστεί το HTTP Response. Στην περίπτωσή μας ο κώδικας αυτός είναι το function που ορίζεται στο field success (γραμμή 16 του common-scripts.js).

Ο κώδικας αυτός θα συμπεριληφθεί σε όλες τις σελίδες με την βοήθεια του include directive που παρουσιάσαμε σε προηγούμενο κεφάλαιο.

### ChooseCommonBackgroundColorServlet.java:

```

19 public class ChooseCommonBackgroundColorServlet extends HttpServlet{
20
21     @Inject
22     StringUtils stringUtils;
23
24     protected void doGet(HttpServletRequest req, HttpServletResponse response)
25             throws ServletException, IOException {
26
27         JsonResponse jsonRes = new JsonResponse(ResponseStatus.ERROR);
28
29         HttpServletRequest request = (HttpServletRequest) req;
30
31         String chosenColor = stringUtils.getTrimmedOrEmpty(request.getParameter("chosen_bg_color"));
32
33         System.out.println("ChooseCommonBackgroundColorServlet.doGet: started for request: "
34             + request.getRequestURL() + "\nchosenColor: " + chosenColor);
35
36         request.getSession().setAttribute(SessionAttributeEnum.COMMON_BG_COLOR_SESSION_ATT.getName(), chosenColor);
37
38         // RESPONSE
39
40         response.setContentType("application/json");
41         response.setCharacterEncoding("UTF-8");
42         PrintWriter resOut = response.getWriter();
43
44         ObjectMapper mapper = new ObjectMapper();
45
46         jsonRes.setStatus(ResponseStatus.SUCCESS);
47         String jsonResString = mapper.writeValueAsString(jsonRes);
48
49         System.out.println("ChooseCommonBackgroundColorServlet.doPost: jsonResString: " + jsonResString);
50
51         resOut.write(jsonResString);
52
53         System.out.println("ChooseCommonBackgroundColorServlet.doGet: ended for request: "
54
55             + request.getRequestURL());
56
57     }
58
58     protected void doPost(HttpServletRequest request, HttpServletResponse response)
59             throws ServletException, IOException {
60
61         System.out.println("ChooseCommonBackgroundColorServlet.doPost: started ... just calls doGet ...");
62         doGet(request, response);
63
64     }
65 }

```

Ουσιαστικά η μόνη γραμμή που μας ενδιαφέρει από αυτό το αρχείο είναι η:

**request.getSession().setAttribute(SessionAttributeEnum.COMMON\_BG\_COLOR\_SESSION\_ATT.getName(), chosenColor);**

Στην γραμμή αυτή παίρνουμε το HttpSession αντικείμενο από το request και αποθηκεύουμε σε αυτό το χρώμα που επέλεξε ο χρήστης με όνομα που ορίζεται από το SessionAttributeEnum.COMMON\_BG\_COLOR\_SESSION\_ATT.

Στην συνέχεια παράγεται JSON (JavaScript Object Notation) Response για το AJAX Request που δέχτηκε ο Servlet ώστε η JQuery να το μετατρέψει κατευθείαν σε Javascript αντικείμενο (το αντικείμενο “data” που ήταν όρισμα στο function που ορίστηκε για το “success” field προηγουμένως στο common-scripts.js). Αυτή η μετατροπή γίνεται με την βοήθεια του **ObjectMapper** αντικειμένου της βιβλιοθήκης **FasterXML/jackson-databind**.

**common-style.html:**

```

1 <style>
2
3 <%
4 String commonBgColor = (String)session.getAttribute(SessionAttributeEnum.COMMON_BG_COLOR_SESSION_ATT.getName());
5 %>
6
7 body{
8     background-color: <=(commonBgColor==null)? "white":commonBgColor
9 }
10
11 </style>

```

Αυτό το αρχείο περιλαμβάνει την CSS που ορίζει το background color της σελίδας με βάση την τιμή του session attribute SessionAttributeEnum.COMMON\_BG\_COLOR\_SESSION\_ATT. Ο κώδικας αυτός θα συμπεριληφθεί σε όλες τις σελίδες με την βοήθεια του include directive που παρουσιάσαμε σε προηγούμενο κεφάλαιο.

Προφανώς ο κώδικας αυτός μπορεί να συμπεριληφθεί μόνο σε JSPs αφού κάνει χρήση scriptlet και evaluation tags. Στην σελίδα που παράγει ο RequestResponseAndSessionExaminationServlet το ίδιο επιτυγχάνεται με τον παρακάτω κώδικα:

```

61 // CREATE THE RESPONSE BODY
62
63
64 PrintWriter resOut = response.getWriter(); // By default in ISO 8859-1 encoding.
65 reqHeadersStr = webUtils.getStringWithAllTheRequestHeaders(request, "<br/><br/>");
66 reqParamsStr = webUtils.getStringWithAllTheRequestParameters(request, "<br/><br/>");
67 reqAttsStr = webUtils.getStringWithAllTheRequestAttributes(request, "<br/><br/>");
68 reqCookies = webUtils.getStringWithAllTheRequestCookies(request, "<br/><br/>");
69 sessAttsStr = webUtils.getStringWithAllTheSessionAttributes(request.getSession(), "<br/><br/>");
70 String commonBgColor = (String)session.getAttribute(SessionAttributeEnum.COMMON_BG_COLOR_SESSION_ATT.getName());
71 resOut.write(
72     "<html>" +
73         "<head>" +
74             "<style>" +
75                 "body{" +
76                     "background-color: " + ((commonBgColor==null)? "white":commonBgColor) +
77                 "}" +
78             "</style>" +
79             "<title>RequestResponseAndSessionExaminationServlet - javaEE_presentation</title>" +
80         "</head>" +
81         "<body>" +
82             "<h2>RECEIVED REQUEST</h2>" +
83
84             "<br/>REMOTE ADDRESS: " + request.getRemoteAddr() + "<br/>" +
85             "<br/>REMOTE PORT: " + request.getRemotePort() + "<br/>" +

```

**To HttpSession.getAttribute(String) επιστρέφει αντικείμενο τύπου Object στο οποίο θα πρέπει να γίνεται πάντα το κατάλληλο downcasting.**

**Include directives στην welcome.jsp για τα αρχεία common-style.html, common-scripts.js και bg-color-selector.html:**

```
1 RequestResponseAndSessionExaminationServlet.java | 2 welcome.jsp |
31<html>
32  <head>
33
34      <title>Welcome Page</title>
35
36      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
37      <!--
38          <link rel="stylesheet" type="text/css" href="<%=request.getContextPath()%>/css/auth.css" /> -->
39
40      <%@ include file = "/WEB-INF/includes/head/common-style.html" %>
41
42      <%@ include file="/WEB-INF/includes/head/jquery.txt" %>
43
44      <script src="<%=request.getContextPath()%>/js/json2.js" type="text/javascript"></script>
45
46      <script type="text/javascript">
47
48          <%@ include file = "/WEB-INF/includes/head/common-scripts.js" %>
49
50          $(function() {
51              "use strict";
52
53              $(document.forms['registerForm']).submit(function(event) {
```

[...]

```
164      <% if(getCurrentHour() >= 18){%>
165          <div>
166              Good Evening!
167          </div>
168      <%}else{ %>
169          <div>
170              Hello!
171          </div>
172      <% } %>
173
174      <%@ include file = "/WEB-INF/includes/body/bg-color-selector.html" %>
175
176      <%-- REQUEST/RESPONSE/SESSION EXAMINATION SERVLET LINK --%>
177      <br/><br/>
178      <a href="<%=request.getContextPath()%>/actions/RequestResponseAndSessionExaminationServlet" >
179          Go to the Request Response & Session Examination Page</a>
180      <br/><br/><br/>
181
182      <%-- FORWARDING AND REDIRECTION EXAMPLE REQUEST FORM --%>
183
```

# Εκτέλεση Παραδείγματος

Αρχικά επιλέγουμε το επιθυμητό χρώμα και πατάμε OK.

The screenshot shows a web browser window with multiple tabs. The active tab is titled "Welcome Page" and has a URL of <https://localhost:7002/DemoWebApp/>. A color selector dialog is overlaid on the page. The dialog title is "Color". It contains two sections: "Basic colors:" and "Custom colors:", each with a grid of color swatches. On the right side of the dialog is a large color picker with a vertical slider for hue. Below the color picker are numerical values for Hue (80), Sat (186), Lum (127), Red (41), Green (228), and Blue (41). At the bottom of the dialog are "OK" and "Cancel" buttons, and a "Add to Custom Colors" button.

Αφού πατήσουμε το "Submit Color & Reload" γίνεται reload της σελίδας με το καινούριο χρώμα:

The screenshot shows the same web browser window after the color selection. The background of the main content area is now a solid green color. The rest of the page content remains the same, including the "Welcome to our secured Web Application" header, the date and time, the greeting message, the "Background Color Selector" section, and the "Forwarding and Redirection Example" section with its message and transfer options.

Όποια άλλη σελίδα και να επισκεφθούμε από εδώ και πέρα θα έχει το καινούριο χρώμα.

The screenshot shows a web browser window with a single tab titled "Welcome Page". The URL is [https://localhost:7002/DemoWebApp/public\\_JSP/public\\_services.jsp](https://localhost:7002/DemoWebApp/public_JSP/public_services.jsp). The page content includes a "Publicly Available Services" header, a "Background Color Selector" section, and a "Compute n-th fibonacci number (odd numbers generate an intentional error)" section. There is also a "Get button press count" section and a "Count Button Press" button.

The screenshot shows a web browser window with a single tab titled "RequestResponseAndSessionExaminationServlet". The URL is <https://localhost:7002/DemoWebApp/actions/ForwardingAndRedirectionServlet>. The page displays the "RECEIVED REQUEST" section, which contains detailed information about the request, such as Remote Address, Remote Port, Remote Host, Server Name, Context Path, Decoded Context Path, Servlet Path, Path Info, Query String, Translated Path, Scheme, Request URI, Decoded Request URI, Request URL, Decoded Request URL, and Authentication Type.

# Polling των Αρχείων της Εφαρμογής στην Περίπτωση ενός

## Expanded Directory

Για να ρυθμίσουμε το πόσο συχνά ο Weblogic Application Server θα ελέγχει τα αρχεία κάποιας ανεβασμένης εφαρμογής για αλλαγές (JSP ή Servlets) θα πρέπει να το κάνουμε μέσω του **runtime descriptor** και των παρακάτω elements:

```
<wls:container-descriptor>
  <wls:servlet-reload-check-secs>1</wls:servlet-reload-check-secs>
</wls:container-descriptor>

<wls:jsp-descriptor>
  <wls:page-check-seconds>1</wls:page-check-seconds>
</wls:jsp-descriptor>
```

Εάν δηλώσουμε 0 το polling για αλλαγές θα απενεργοποιηθεί και θα πρέπει να ενημερώνουμε τον server μέσω ειδικών εντολών της κονσόλας του. Οποιαδήποτε άλλη τιμή δηλώνει τον αριθμό των δευτερολέπτων που μεσολαβούν μεταξύ των ελέγχων για αλλαγές.

Το <wls:servlet-reload-check-secs> αντιστοιχεί στα διαστήματα μεταξύ των ελέγχων για αλλαγές στους Servlets ενώ το <wls:page-check-seconds> στις JSP.

Γενικά ενεργοποιούμε αυτό το polling μόνο σε τεστ και όχι παραγωγικά περιβάλλοντα.

# JDBC

Αφού το JDBC επεξηγήθηκε σε γενικές γραμμές στο εισαγωγικό κεφάλαιο που αναφέρεται στα container services στο παρακάτω απλό παράδειγμα θα γίνει παρουσίαση της επικοινωνίας με μία ΒΔ για την ανάγνωση και εγγραφή δεδομένων σε έναν σχεσιακό πίνακα.

## Παράδειγμα με Κώδικα

Η ΒΔ που επιλέξαμε είναι η **Oracle Database 11g**.

Ο κώδικας του παραδείγματος βρίσκεται στην κλάση **osotnikov.demowebapp.web**.

**JdbcDemoServlet**. Το μόνο που κάνει είναι να δημιουργεί μία εγγραφή στον πίνακα

**JDBC\_DEMO\_ACCESS\_INFO** κάθε φορά που προσπελάζεται ο Servlet και στην συνέχεια να διαβάζει όλες τις εγγραφές αυτού του πίνακα και να επιστρέψει τα δεδομένα αυτά σε HTML μορφή. Η εγγραφή αυτή σημειώνει την ώρα που έγινε η προσπέλαση, τον απομακρυσμένο πελάτη (την IP του), την θύρα του και το αναγνωριστικό συνοδού που του αντιστοιχεί.

Ο πίνακας της ΒΔ του παραδείγματος δημιουργείται με την παρακάτω εντολή:

```
CREATE TABLE WEB_DEMO.JDBC_DEMO_ACCESS_INFO(
    ACCESS_TIMESTAMP TIMESTAMP(6) WITH TIME ZONE,
    REMOTE_HOST      VARCHAR2(1024 CHAR),
    REMOTE_PORT      NUMBER(5),
    SESSION_ID       VARCHAR2(1024 CHAR)
)
```

Ο κώδικας του **JdbcDemoServlet** είναι ο παρακάτω:

```
1 package osotnikov.demowebapp.web;
2
3 import java.io.IOException;
4
5 public class JdbcDemoServlet extends HttpServlet{
6
7
8     static final String DB_URL = "jdbc:oracle:thin:@localhost:1521:XE";
9
10    // Database credentials
11    static final String USER = "WEB_DEMO";
12    static final String PASS = "WEB_DEMO";
13
14    public JdbcDemoServlet() {
15        super();
16    }
17
18    protected void doGet(HttpServletRequest request, HttpServletResponse response)
19        throws ServletException, IOException {
20
21        System.out.println("JdbcDemoServlet.doGet ... started.");
22
23        // CREATE THE HTTP RESPONSE BODY
24
25        Connection conn = null;
26        PreparedStatement preparedStatement = null;
27        Statement statement = null;
28        ResultSet rs = null;
29        try{
30
31            // ==> LOG THE ACCESS ATTEMPT TO THIS SERVLET
32
33            // STEP 1: Register JDBC driver.
34            // Class.forName("oracle.jdbc.OracleDriver").newInstance();
35            // Since Java 4 it is not anymore necessary to register JDBC drivers manually.
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
```

```
56 // Any JDBC 4.0 drivers that are found in your class path at runtime are
57 // automatically loaded by the JVM.
58 // Before Java 4 the registration of the driver with the DriverManager used
59 // to happen inside the static initializer block of the Driver class.
60
61 // STEP 2: Open a connection.
62 conn = DriverManager.getConnection(DB_URL, USER, PASS);
63
64 // STEP 3: Create the sql statement.
65 String sql = "INSERT INTO JDBC_DEMO_ACCESS_INFO(ACCESS_TIMESTAMP, REMOTE_HOST, "
66 + "REMOTE_PORT, SESSION_ID) VALUES (?, ?, ?, ?)";
67
68 preparedStatement = conn.prepareStatement(sql);
69 preparedStatement.setTimestamp(1, new Timestamp(new java.util.Date().getTime()));
70 preparedStatement.setString(2, request.getRemoteHost());
71 preparedStatement.setInt(3, request.getRemotePort());
72 preparedStatement.setString(4, request.getSession().getId());
73
74 // STEP 4: Execute the statement.
75 int recordsInserted = preparedStatement.executeUpdate();
76
77 System.out.println("Inserted records into the table: " + recordsInserted);
78
79 // conn.commit(); // This is not necessary as by default the connection object is in
80 // auto commit mode, which means that each statement gets committed i.e. is executed
81 // as an individual transaction.
82
83 // ==> CREATE AND RETURN THE HTML REPORT OF THE LAST ACCESS ATTEMPTS TO THIS PAGE
84
85 // Set response content type
86 response.setContentType("text/html");
87 PrintWriter out = response.getWriter();
88 String title = "JDBC Demo Servlet";
89 String docType =
90 " <!DOCTYPE html public "-//w3c//dtd html 4.0 " +
91 "transitional//en\">\n";
92
92 out.println(docType +
93 "<html>\n" +
94 "<head><title>" + title + "</title></head>\n" +
95 "<body bgcolor=#f0f0f0>\n" +
96 "<h1 align=center>" + title + "</h1>\n" +
97 "<h2 align=left>Link to Homepage</h2>\n" +
98 "<a href="" + request.getContextPath() + "\">Home</a>" +
99 "<h2 align=left>Access Logs</h1>\n";
100
101 // Execute SQL query
102 statement = conn.createStatement();
103 sql = "SELECT ACCESS_TIMESTAMP, REMOTE_HOST, REMOTE_PORT, SESSION_ID FROM JDBC_DEMO_ACCESS_INFO " +
104 "ORDER BY ACCESS_TIMESTAMP DESC";
105 rs = statement.executeQuery(sql);
106
107 String accessTimestamp, remoteHost, remotePort, sessionId;
108
109 // Extract data from result set
110 while(rs.next()){
111     //Retrieve by column name
112     accessTimestamp = rs.getString("ACCESS_TIMESTAMP");
113     remoteHost = rs.getString("REMOTE_HOST");
114     remotePort = rs.getString("REMOTE_PORT");
115     sessionId = rs.getString("SESSION_ID");
116
117     //Display values
118     out.println("<br/>");
119     out.println("Access Timestamp: " + accessTimestamp + "<br/>");
120     out.println("Remote Host: " + remoteHost + "<br/>");
121     out.println("Remote Port: " + remotePort + "<br/>");
122     out.println("Session ID: " + sessionId + "<br/>");
123 }
124 out.println("</body></html>");
125
126
127 }catch(SQLException se){
```

```

128     //Handle errors for JDBC
129     se.printStackTrace();
130 }catch(Exception e){
131     //Handle errors for Class.forName
132     e.printStackTrace();
133 }finally{
134     //finally block used to close resources
135     try{
136         if(preparedStatement != null){
137             preparedStatement.close();
138         }
139         if(statement != null){
140             statement.close();
141         }
142         if(rs != null){
143             rs.close();
144         }
145         if(conn != null){
146             conn.close();
147         }
148     }catch(SQLException se){
149         System.out.println("ERROR: Could not close a JDBC resource! Possible memory leak!");
150     }
151 }
152
153 System.out.println("JdbcDemoServlet doGet: ended for request: "
154 + request.getRequestURL());
155
156 }
157

```

Για οποιαδήποτε ενέργεια προς την ΒΔ είναι απαραίτητο να δημιουργήσουμε ένα **Connection** αντικείμενο. Αυτό γίνεται μέσω του **DriverManager** αντικειμένου καλώντας την μέθοδο **getConnection** και περνώντας σαν ορίσματα την διεύθυνση της ΒΔ (η μορφή της εξαρτάται από την ΒΔ στην οποία συνδεόμαστε αλλά σίγουρα απαιτεί να αναφέρεται ο host και το port στο οποίο τρέχει η ΒΔ) και τα credentials (username και password) για την σύνδεση με αυτήν.

Παλαιότερα (πριν την Java 1.4) απαιτούνταν αρχικά το registration του JDBC Driver της εκάστοτε ΒΔ (ο σχετικός κώδικας στο παράδειγμά μας βρίσκεται σε σχόλιο). Οι λεπτομέρειες αυτής της διαδικασίας εξηγούνται στο κεφάλαιο «**Driver Registration πριν την Java 1.4**».

Όλος ο κώδικας για την επικοινωνία με την ΒΔ θα πρέπει να βρίσκεται σε ένα **try-catch block**. Exceptions μπορούν να δημιουργηθούν εξαιτίας προβλημάτων επικοινωνίας με την ΒΔ ή σφαλμάτων κατά την εκτέλεση των ερωτημάτων.

Όλα τα αντικείμενα για την εκτέλεση των SQL statements (**Connection**, **Statement**, **PreparedStatement**, **ResultSet**) θα πρέπει να ορίζονται με null τιμές έξω από το try-catch block, να αρχικοποιούνται μέσα σε αυτό και τέλος μετά την εκτέλεση όλων των επιθυμητών ενεργειών να αποδεσμεύονται καλώντας την μέθοδο **close** για το καθένα από αυτά (εάν έχουν όντως αρχικοποιηθεί και δεν είναι null). Οι μέθοδοι close θα πρέπει να καλούνται σε ένα finally block έτσι ώστε να είμαστε σίγουροι ότι θα εκτελεσθούν είτε παρουσιαστεί κάποιο exception είτε όχι. Εάν δεν γίνει κλήση της close μεθόδου σε κάποιο resource τότε παραμένει στην μνήμη και ενδέχεται να δεσμεύει και άλλους πόρους στην ίδια την ΒΔ. Πολλά προβλήματα τύπου memory leak οφείλονται στην αποτυχία αποδέσμευσης αυτών των πόρων.

Για να υποβάλλουμε ένα SQL Statement προς την βάση θα πρέπει να το δημιουργήσουμε, αρχικά δημιουργώντας ένα String με το SQL Statement αλλά αντί να περάσουμε τις παραμέτρους που θέλουμε, στην θέση τους βάζουμε **placeholders ("?")**. Στην συνέχεια δημιουργούμε ένα αντικείμενο τύπο **PreparedStatement** μέσω της μεθόδου **Connection.prepareStatement(String)**. Για να περάσουμε τις παραμέτρους που θέλουμε στην θέση των placeholders χρησιμοποιούμε τις μεθόδους **setString**, **setInt**, **setTimestamp** κ.τ.λ. του PreparedStatement αντικειμένου, όπου

πρώτα δηλώνουμε τον placeholder (η αρίθμηση ξεκινάει από το 1) και στην συνέχεια την τιμή της παραμέτρου.

Αφού το δημιουργήσουμε το υποβάλουμε προς την ΒΔ με την μέθοδο **PreparedStatement.executeUpdate()**.

Στην συνέχεια γίνεται ανάγνωση όλων των τιμών από το table **JDBC\_DEMO\_ACCESS\_INFO** με ένα απλό Select Query. Αυτήν την φορά δεν υπάρχει ανάγκη για ένα PreparedStatement αφού δεν χρειάζεται να ορίσουμε παραμέτρους προς αυτό κατά το runtime. Όταν δηλαδή το SQL statement που χρειάζεται να υποβάλουμε προς την ΒΔ είναι προκαθορισμένο κατά το compile time χρησιμοποιούμε αντικείμενο τύπου **Statement** το οποίο το δημιουργούμε μέσω της μεθόδου **Connection.createStatement()**.

Αυτήν την φορά το Statement εκτελείται μέσω της μεθόδου **Statement.executeQuery(String)** κάτι που γενικά όπως καταλαβαίνουμε επιτρέπει την επαναχρησιμοποίηση του Statement αντικειμένου για την εκτέλεση διαφορετικών SQL Queries αφού δεν περνάμε κάποια παράμετρο στο **Connection.createStatement()**.

Αντί να επιστρέφεται ο αριθμός των εγγραφών που εισάχθηκε όπως προηγουμένως κατά την εκτέλεση του **PreparedStatement.executeUpdate()**, κατά την εκτέλεση του **Statement.executeQuery(String)** επιστρέφεται ένα αντικείμενο τύπου **ResultSet** το οποίο είναι μια δομή δεδομένων με τις εγγραφές που επιστράφηκαν από το select query η οποία είναι προσπελάσιμη μέσω της μεθόδου **ResultSet.next()** η οποία ξεκινάει από μία θέση πριν από την πρώτη εγγραφή και επιστρέφει false εάν δεν έχουν μείνει άλλες εγγραφές. Η ίδια η **ResultSet.next()** δεν επιστρέφει την εγγραφή αλλά απλά μετακινεί έναν εσωτερικό δείκτη στην επόμενη εγγραφή της δομής. Ύστερα οι τιμές της εγγραφής προσπελάζονται μέσω των μεθόδων του **ResultSet**, **getString**, **getInt**, **getTimestamp** κ.τ.λ. οι οποίες παίρνουν ως όρισμα το όνομα του column του οποίου την τιμή επιστρέφουν.

## Εκτέλεση το Παραδείγματος

Η μετάβαση στο JdbcDemoServlet μπορεί να γίνει από την σελίδα των publicly available services:



Publicly Available Services

Background Color Selector

Select the background color that you want:

JDBC Demo

Transfer to the JDBC Demo Page. This page is produced by a Servlet that logs all the access attempts to itself in the DB table "JDBC\_DEMO\_ACCESS\_INFO". Its code is an example of insert and select operations implemented with JDBC.

**Παρουσιάζεται η HTML που επιστρέφεται από το JdbcDemoServlet μετά από αρκετές επισκέψεις σε αυτήν:**

The screenshot shows a web browser window with the title "JDBC Demo Servlet". The address bar displays "Mη ασφαλής | https://localhost:7002/DemoWebApp/actions/JdbcDemoServlet". The main content area contains the following text:

**JDBC Demo Servlet**

**Link to Homepage**

[Home](#)

**Access Logs**

Access Timestamp: 2017-06-27 14:29:50.534 Europe/Athens  
Remote Host: 0:0:0:0:0:0:1  
Remote Port: 13686  
Session ID: i6npUJBmhWL6m8hikqZ1KeXQjbtEvXnU0xa4PnESg7oC3HQCTIa1!-444745442!1498562990182

Access Timestamp: 2017-06-27 01:34:00.656 Europe/Athens  
Remote Host: 0:0:0:0:0:0:1  
Remote Port: 1977  
Session ID: -4rmikY\_tAbraCQ60IyoU4aF\_EeffYwfhRct6ptDVmF0BHvASzR3!-1999172740!1498516440639

Access Timestamp: 2017-06-27 01:29:54.669 Europe/Athens  
Remote Host: 0:0:0:0:0:0:1  
Remote Port: 1873  
Session ID: jLnmhoVcboViB55tG8q\_yKni2jitRv3OD8p1ctvkWQvWUkDaErk!-1999172740!1498516194652

Access Timestamp: 2017-06-27 01:29:07.767 Europe/Athens  
Remote Host: 0:0:0:0:0:0:1  
Remote Port: 1873  
Session ID: Rr\_mhc4h1PIVeVamffKNS24g9aZGv\_r3t\_ixvRaNkmzY7Pvk4gR6!-1999172740!1498516147745

**Τα ίδια δεδομένα όπως παρουσιάζονται στην ΒΔ:**

The screenshot shows the Toad for Oracle interface with the database connection "WEB\_DEMO@LOCALHOST:1521/XE" selected. The "Schema Browser" tab is active, displaying the "WEB\_DEMO" schema. The "Tables" section shows the "JDBC\_DEMO\_ACCESS\_INFO" table. The table has the following columns: ACCESS\_TIMESTAMP, REMOTE\_HOST, REMOTE\_PORT, and SESSION\_ID. The data grid below shows the following rows:

ACCESS_TIMESTAMP	REMOTE_HOST	REMOTE_PORT	SESSION_ID
27/6/2017 1:13:05,782000 ΠΜ +03:00	0:0:0:0:0:0:1	1459	qKrmdp3Gl4bY2mUXxb0wGf4PYAjC4eEWGRstoiAPtMy34DQZjOpI-1999172740!1498562990182
27/6/2017 1:21:03,848000 ΠΜ +03:00	0:0:0:0:0:0:1	1624	nwTmfkRQeeJodcvU2Rjg0lQxMyooK1RzPgKdkyTUMbgY77cWFoGh!-1999172740!1498516440639
27/6/2017 1:21:12,602000 ΠΜ +03:00	0:0:0:0:0:0:1	1624	nwTmfkRQeeJodcvU2Rjg0lQxMyooK1RzPgKdkyTUMbgY77cWFoGh!-1999172740!1498516194652
27/6/2017 1:21:16,666000 ΠΜ +03:00	0:0:0:0:0:0:1	1624	nwTmfkRQeeJodcvU2Rjg0lQxMyooK1RzPgKdkyTUMbgY77cWFoGh!-1999172740!1498516147745
27/6/2017 1:27:35,789000 ΠΜ +03:00	0:0:0:0:0:0:1	1810	GtHmhExaX7mijnkDSd37JJVE5HKU_hgsBj54_m6yZuC8xkPLddOpI-1999172740!1498562990182
27/6/2017 1:29:07,767000 ΠΜ +03:00	0:0:0:0:0:0:1	1873	R_r_mhc4h1PIVeVamffKNS24g9aZGv_r3t_jxvRaNkmzY7Pvk4gR6!-1999172740!1498516147745
27/6/2017 1:29:54,669000 ΠΜ +03:00	0:0:0:0:0:0:1	1873	jLnmhoVcboViB55tG8q_yKni2jitRv3OD8p1ctvkWQvWUkDaErk!-1999172740!1498516194652
27/6/2017 1:34:00,656000 ΠΜ +03:00	0:0:0:0:0:0:1	1977	-4rmikY_tAbraCQ60IyoU4aF_EeffYwfhRct6ptDVmF0BHvASzR3!-1999172740!1498516147745
27/6/2017 2:20:50,524000 ΜΜ +02:00	0:0:0:0:0:0:1	12686	i6npUJBmhWL6m8hikqZ1KeXQjbtEvXnU0xa4PnESg7oC3HQCTIa1!-444745442!1498562990182

## Driver Registration πριν την Java 1.4

Όπως έχουμε αναφέρει και σε προηγούμενο κεφάλαιο είναι αναγκαίος ένας **JDBC Driver** για την επικοινωνία με την ΒΔ αφού το ίδιο το JDBC είναι απλώς μία διεπαφή, δηλαδή ορίζει τις μεθόδους για την επικοινωνία με την βάση αλλά δεν ορίζει την υλοποίησή τους.

Για να επιλεχθεί ο JDBC Driver που θα χρησιμοποιηθεί για την συγκεκριμένη ΒΔ με την οποία θα επικοινωνήσουμε γίνεται κλήση της μεθόδου **DriverManager.registerDriver(Driver)**. Αυτή η κλήση (ενδεχομένως με επιπλέον κώδικα που παραμετροποιεί κάποιες ακόμα λεπτομέρειες για την συγκεκριμένη υλοποίηση του Driver που χρησιμοποιείται) γίνεται εσωτερικά της Driver κλάσης μέσα σε ένα static initialization block. Όπως γνωρίζουμε από τους κανόνες της Java τα static initialization blocks εκτελούνται μετά την πρώτη αναφορά που γίνεται στην κλάση στο πρόγραμμά μας.

Έτσι παλαιότερα ήταν αναγκαίος ο παρακάτω κώδικας (παρουσιάζεται παράδειγμα για το driver registration για την ΒΔ Oracle Database 11G):

```
Class.forName("oracle.jdbc.OracleDriver");
```

Το αναμενόμενο ίσως να ήταν κάτι τέτοιο:

```
oracle.jdbc.OracleDriver od;
```

Καταλαβαίνουμε ότι την στιγμή που θα εκτελούνταν κάποια από τις δύο γραμμές κώδικα κατά το runtime, θα εκτελούνταν και το static initialization block που κάνει registration του JDBC Driver με τον DriverManager του JDBC.

Η διαφορά των δύο γραμμών είναι ότι **ο δεύτερος τρόπος θα απαιτούσε η κλάση oracle.jdbc.OracleDriver να είναι διαθέσιμη κατά το compile time**, που σημαίνει ότι θα έπρεπε να την συμπεριλάβουμε στις βιβλιοθήκες που βρίσκονται στο classpath που ισχύει στο compile time. Από την στιγμή όμως που δεν χρειάζεται να αναφερθούμε σε καμία μέθοδο ή κλάση του JDBC Driver που χρησιμοποιούμε, αλλά μόνο στις διεπαφές του JDBC για να εκτελέσουμε τις ενέργειες που θέλουμε προς την ΒΔ, πολλές φορές για λόγους βολικότητας, θέλουμε να αποφύγουμε να συμπεριλάβουμε τον Driver στο classpath που χρησιμοποιούμε κατά το development των εφαρμογών μας.

Το classpath που ισχύει κατά το compile time είναι διαφορετικό από αυτό που θα ορίσει ο Application Server μας στα startup scripts τα οποία εκτελεί. Το δικό του classpath θα περιλαμβάνει και άλλα directories. Π.χ. ο Oracle Weblogic Server 12C που χρησιμοποιούμε στα παραδείγματα αυτής της εργασίας διαθέτει από μόνος του προεγκατεστημένους JDBC Drivers για τις πιο γνωστές ΒΔ. **Έτσι η κλάση oracle.jdbc.OracleDriver θα είναι διαθέσιμη κατά το runtime αλλά όχι το compile time, αλλά αυτό δεν αποτελεί πρόβλημα διότι θα χρειαστεί μόνο κατά το runtime εάν επιλέξουμε να κάνουμε registration του JDBC Driver με τον πρώτο τρόπο.**

Κάποιοι παλιοί JDBC Drivers ήταν προβληματικοί και δεν ακολουθούσαν την συμπεριφορά που είχε οριστεί από το JDBC πρότυπο. Κάποιοι κάνανε το registration του εαυτού τους μέσα στον constructor τους αντί για το static initialization block της κλάσης τους. Για αυτό μπορεί να συναντήσουμε και τον παρακάτω κώδικα που αντιμετωπίζει και αυτήν αλλά και την προηγούμενη περίπτωση:

```
Class.forName("oracle.jdbc.OracleDriver").newInstance();
```

Άλλοι drivers πάλι δεν κάνανε καθόλου registration του εαυτού τους οπότε ήταν απαραίτητος ο παρακάτω κώδικας με απευθείας κλήση της μεθόδου DriverManager.registerDriver(Driver):

```
try {  
    Driver myDriver = new oracle.jdbc.driver.OracleDriver();  
    DriverManager.registerDriver( myDriver );  
}  
  
catch(ClassNotFoundException ex) {  
    System.out.println("Error: unable to load driver class!");  
    System.exit(1);  
}
```

Από την Java 1.4 και μετά δεν είναι απαραίτητη καμία από τις προηγούμενες μεθόδους διότι έχει οριστεί από το JDBC πρότυπο ότι το registration των drivers θα πρέπει να γίνεται αυτόματα. Συγκεκριμένα στο documentation του DriverManager αναφέρεται του εξής:

Applications no longer need to explicitly load JDBC drivers using Class.forName(). Existing programs which currently load JDBC drivers using Class.forName() will continue to work without modification.

Ο τρόπος με τον οποίον αυτό επιτυγχάνεται είναι με το να παρέχεται το αρχείο **META-INF/services/java.sql.Driver** στο jar του JDBC Driver. Το αρχείο αυτό αναφέρει το όνομα της κλάσης του JDBC Driver και κατ' αυτόν τον τρόπο δίνει την απαραίτητη πληροφορία στο JVM για το registration του JDBC Driver αυτόματα.

## EJB

Το EJB είναι ένα πρότυπο το οποίο σχεδιάστηκε για την μοντελοποίηση του **Business-Tier**.

Παρέχει **Dependency Injection** καθώς και τα **Container Services** που αναφέρθηκαν σε προηγούμενο κεφάλαιο. Επίσης παρέχει και ένα **υψηλότερο επίπεδο διασύνδεσης με την ΒΔ** από το JDBC διαμέσου των **Entity Beans**.

Ορίζει τους παρακάτω τύπους συστατικών:

- **Stateless Session Beans**
- **Stateful Session Beans**
- **Singleton Beans**
- **Entity Beans**
- **Message Driven Beans (MDB)**

Τα συστατικά αυτά υφίστανται μόνο κατά το runtime και μόνο στα πλαίσια ενός **EJB Container** (υποσύστημα του Application Server). Αρχικά δηλαδή δημιουργούμε μια κλάση η οποία δεν χρειάζεται να εφαρμόζει κάποιο interface ή να κάνει extend κάποια άλλη κλάση. Το μόνο που χρειάζεται είναι να περιγραφεί σαν ένα από τα προηγούμενα components στο XML configuration file (δηλαδή ejb-jar.xml) ή να έχει τα αντίστοιχα **annotations**. Κατά αυτόν τον τρόπο **το EJB πρότυπο εισβάλει όσο το δυνατόν λιγότερο μέσα στον κώδικα της εφαρμογής μας** κάτι που σημαίνει ότι πρώτον θα μπορεί να εφαρμοστεί εύκολα σε προϋπάρχον κώδικα και δεύτερον ο κώδικας στον οποίον εφαρμόστηκε θα μπορεί να χρησιμοποιηθεί εύκολα σε άλλα πλαίσια όπου δεν θα υποστηρίζεται το **EJB πρότυπο**. Κάτι τέτοιο δεν ίσχυε σε παλαιότερες εκδόσεις του προτύπου όπου ήταν απαραίτητη η εφαρμογή διαφόρων interfaces και η χρήση κλάσεων που ορίζε το EJB πρότυπο.

Ανάλογα με το ποιο από τα προηγούμενα components θα ορίσουμε να είναι η κλάση μας, τα αντικείμενά της θα έχουν **διαφορετικό κύκλο ζωής και αντιμετώπιση των ταυτόχρονων προσπελάσεων** (Stateless, Stateful, Singleton Beans), θα αποτελούν ένα **ORM** (Object Relational Mapping) επίπεδο διασύνδεσης με την ΒΔ (Entity Beans) ή θα είναι συστατικά για την υλοποίηση **ασύγχρονων κλήσεων** (Message Driven Beans).

## Session Beans

Τα Session Beans έχουν τον ρόλο της υλοποίησης του **business logic** της εφαρμογής. Πέραν από το **Dependency Injection** ορίζουν το **lifecycle, scope και concurrency management**. Το πώς υλοποιούνται όλα αυτά ξεκαθαρίζεται στα επόμενα υποκεφάλαια μέσω απλών παραδειγμάτων κώδικα.

### Singleton Session Beans

#### Γενική Περιγραφή

Εάν οριστεί ένα Singleton Session Bean πάνω σε μία κλάση **θα δημιουργηθεί μόνο ένα αντικείμενο αυτής της κλάσης**. **Μπορεί να οριστεί να δημιουργηθεί κατά το ξεκίνημα της εφαρμογής** ή όταν γίνει για πρώτη φορά αναφορά προς αυτό.

Η **διαχείριση των παράλληλων κλήσεων** προς το αντικείμενο είναι παραμετροποιήσιμη. Εάν επιλέξουμε την παραμετροποίηση σε επίπεδο μεθόδων (**Container Managed Concurrency**) μπορεί να επιλέξουμε να ορίζουμε το αν θα επιτρέπεται η ταυτόχρονη κλήση ορισμένων μεθόδων ή εάν όλες οι κλήσεις θα μπλοκάρονται και θα περιμένουν όση ώρα εκτελείται μία συγκεκριμένη μέθοδος.

Εάν επιλέξουμε την παραμετροποίηση σε επίπεδο κώδικα (**Bean Managed Concurrency**) τότε, όσον αφορά την διαχείριση της παραλληλίας, καθιστούμε το Singleton EJB σαν ένα απλό αντικείμενο της Java. **Αυτό σημαίνει ότι επιτρέπονται παράλληλες κλήσεις και η ευθύνη της διαχείριση των θεμάτων της παραλληλίας πια μεταφέρεται στον προγραμματιστή** ο οποίος θα πρέπει να το κάνει με όποιον τρόπο αυτός προτιμάει (π.χ. με βασικά keywords της Java όπως “synchronized”, “volatile” κ.τ.λ.).

#### Παραδείγματα με Κώδικα

##### ButtonDemoSingleton

Το πρώτο παράδειγμα που θα παρουσιάσουμε αφορά τον ορισμό ενός Singleton Bean το οποίο θα χρησιμοποιηθεί για να δημιουργηθεί **ένα απλό παράθυρο με τρία κουμπιά** κατά το ξεκίνημα της εφαρμογής DemoWebApp.

**Ουσιαστικά το παράδειγμα αυτό έχει σκοπό να παρουσιάσει το πώς ένα Singleton Bean θα μπορούσε να λειτουργήσει ως μία διασύνδεση με ένα παραθυρικό περιβάλλον το οποίο δημιουργείτε κατά το ξεκίνημα του enterprise application.** Καταλαβαίνουμε ότι πάνω στον ίδιο σχεδιασμό θα μπορούσε να βασιστεί και μία κονσόλα διαχείρισης του enterprise application από τους διαχειριστές του (όπως είχαμε περιγράψει και στο θεωρητικό παράδειγμα στο κεφάλαιο του Dependency Injection).

Αρχικά παρουσιάζεται ο κώδικας του Singleton. Δεν θα γίνει ανάλυση του κώδικα SWING που δημιουργεί το περιεχόμενο του παραθύρου και την λειτουργικότητα των κουμπιών. Το μόνο που θα αναλυθεί είναι το κομμάτι του κώδικα που δημιουργεί το ίδιο το παράθυρο.

## ButtonDemoSingletonImpl.java



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a project structure with several packages and files under 'osotnikov.demowebapp.services' and 'osotnikov.demowebapp.web'. On the right, the code editor window is open with the file 'ButtonDemoSingletonImpl.java'. The code implements a Singleton pattern using a static field and a private constructor. It also includes a PostConstruct annotation and a Runnable implementation for the invokeLater method.

```
1 package osotnikov.demowebapp.swingexample;
2
3 import javax.annotation.PostConstruct;
4
5 public class ButtonDemoSingletonImpl implements ButtonDemoSingleton {
6     ...
7     @PostConstruct
8     private void init() {
9         System.out.println("ButtonDemoSingleton STARTED");
10    ...
11    javax.swing.SwingUtilities.invokeLater(new Runnable() {
12        ...
13        public void run() {
14            ButtonJPanel.createAndShowGUI();
15        }
16    });
17 }
18 }
```

Στην γραμμή 16 της προηγούμενης εικόνας φαίνεται η κλήση της μεθόδου

**SwingUtilities.invokeLater(Runnable)**. Αυτή η μέθοδος θα εκτελέσει τον κώδικα μέσα στο Runnable αντικείμενο που τις περνιέται σαν όρισμα στο EDT (Event Dispatching Thread). Το EDT είναι ένα Thread που δημιουργεί το SWING για να εκτελεί όλες τις ενέργειες που έχουν να κάνουν με τον σχεδιασμό των γραφικών αντικειμένων (παράθυρα, κουμπιά, text boxes κ.τ.λ.). Όπως βλέπουμε και από το όνομα της μεθόδου στην γραμμή 18 η μέθοδος στο Runnable αντικείμενο που στέλνουμε για να εκτελεσθεί στο EDT θα δημιουργήσει και θα εμφανίσει το παράθυρο με τα τρία κουμπιά που θέλουμε.

Όπως παρατηρούμε ο κώδικας αυτός βρίσκεται σε μέθοδο η οποία έχει το **annotation "PostConstruct"** που εφαρμόζεται στην μέθοδο η οποία θέλουμε να τρέξει αμέσως μόλις δημιουργηθεί το Singleton Session Bean, αρχικοποιηθούν τα fields, του και τρέξει ο constructor του.

## ButtonJPanel.java:

```
1 package osotnikov.demowebapp.swingexample;
2
3 * import javax.swing.AbstractButton;*
4
5 /**
6  * ButtonDemo.java requires the following files:
7  *   images/right.gif
8  *   images/middle.gif
9  *   images/left.gif
10 */
11 public class ButtonJPanel extends JPanel
12     implements ActionListener {
13     protected JButton b1, b2, b3;
14
15     public ButtonJPanel() {
16         //ImageIcon leftButtonIcon = createImageIcon("images/right.gif");
17         //ImageIcon middleButtonIcon = createImageIcon("images/middle.gif");
18         //ImageIcon rightButtonIcon = createImageIcon("images/left.gif");
19
20         b1 = new JButton("Disable middle button/*, leftButtonIcon*/);
21         b1.setVerticalTextPosition(AbstractButton.CENTER);
22         b1.setHorizontalTextPosition(AbstractButton.LEADING); //aka LEFT, for left-to-right locales
23         b1.setMnemonic(KeyEvent.VK_D);
24         b1.setActionCommand("disable");
25
26         b2 = new JButton("Middle button/*, middleButtonIcon*/);
27         b2.setVerticalTextPosition(AbstractButton.BOTTOM);
28         b2.setHorizontalTextPosition(AbstractButton.CENTER);
29         b2.setMnemonic(KeyEvent.VK_M);
30
31         b3 = new JButton("Enable middle button/*, rightButtonIcon*/);
32         //Use the default text position of CENTER, TRAILING (RIGHT).
33         b3.setMnemonic(KeyEvent.VK_E);
34         b3.setActionCommand("enable");
35         b3.setEnabled(false);
36
37
38
39
40
41
42
43
44
45     //Listen for actions on buttons 1 and 3.
46     b1.addActionListener(this);
47     b3.addActionListener(this);
48
49     b1.setToolTipText("Click this button to disable the middle button.");
50     b2.setToolTipText("This middle button does nothing when you click it.");
51     b3.setToolTipText("Click this button to enable the middle button.");
52
53     //Add Components to this container, using the default FlowLayout.
54     add(b1);
55     add(b2);
56     add(b3);
57 }
58
59     public void actionPerformed(ActionEvent e) {
60         if ("disable".equals(e.getActionCommand())) {
61             b2.setEnabled(false);
62             b1.setEnabled(false);
63             b3.setEnabled(true);
64         } else {
65             b2.setEnabled(true);
66             b1.setEnabled(true);
67             b3.setEnabled(false);
68         }
69     }
70
71     /** Returns an ImageIcon, or null if the path was invalid. */
72     protected static ImageIcon createImageIcon(String path) {
73         java.net.URL imgURL = ButtonJPanel.class.getResource(path);
74         if (imgURL != null) {
75             return new ImageIcon(imgURL);
76         } else {
77             System.err.println("Couldn't find file: " + path);
78             return null;
79         }
80     }

```

```

81
82  /**
83   * Create the GUI and show it. For thread safety,
84   * this method should be invoked from the
85   * event-dispatching thread.
86   */
87  public static void createAndShowGUI() {
88
89      //Create and set up the window.
90      JFrame frame = new JFrame("ButtonDemo");
91      frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
92
93      //Create and set up the content pane.
94      ButtonJPanel newContentPane = new ButtonJPanel();
95      newContentPane.setOpaque(true); //content panes must be opaque
96      frame.setContentPane(newContentPane);
97
98      //Display the window.
99      frame.pack();
100     frame.setVisible(true);
101 }
102
103 /*public static void main(String[] args) {
104     //Schedule a job for the event-dispatching thread:
105     //creating and showing this application's GUI.
106     javax.swing.SwingUtilities.invokeLater(new Runnable() {
107         public void run() {
108             createAndShowGUI();
109         }
110     });
111 }*/
112 }
113

```

Όπως βλέπουμε στις γραμμές 99 και 100 εκτελείται ο κώδικας που παρουσιάζει το παράθυρο (JFrame) αφού προηγουμένως αυτό έχει πάρει ως περιεχόμενο το Panel ButtonJPanel (γραμμή 96) που έχει χτιστεί στον αντίστοιχο constructor.

#### ejb-jar.xml

Για να ορίσουμε το ButtonDemoSingleton να δημιουργείται κατά το ξεκίνημα της εφαρμογής Θεωρητικά αρκεί το annotation “**@Startup**”, πρακτικά όμως δεν λειτουργεί σε αρκετούς Application Servers όπως συμβαίνει και στον Weblogic 12C. Για αυτόν τον λόγο η κλάση **ButtonDemoSingletonImpl** ορίστηκε ως EJB Singleton Session Bean στον EJB deployment descriptor (**ejb-jar.xml**) όπου το αντίστοιχο κομμάτι του παρουσιάζεται παρακάτω.

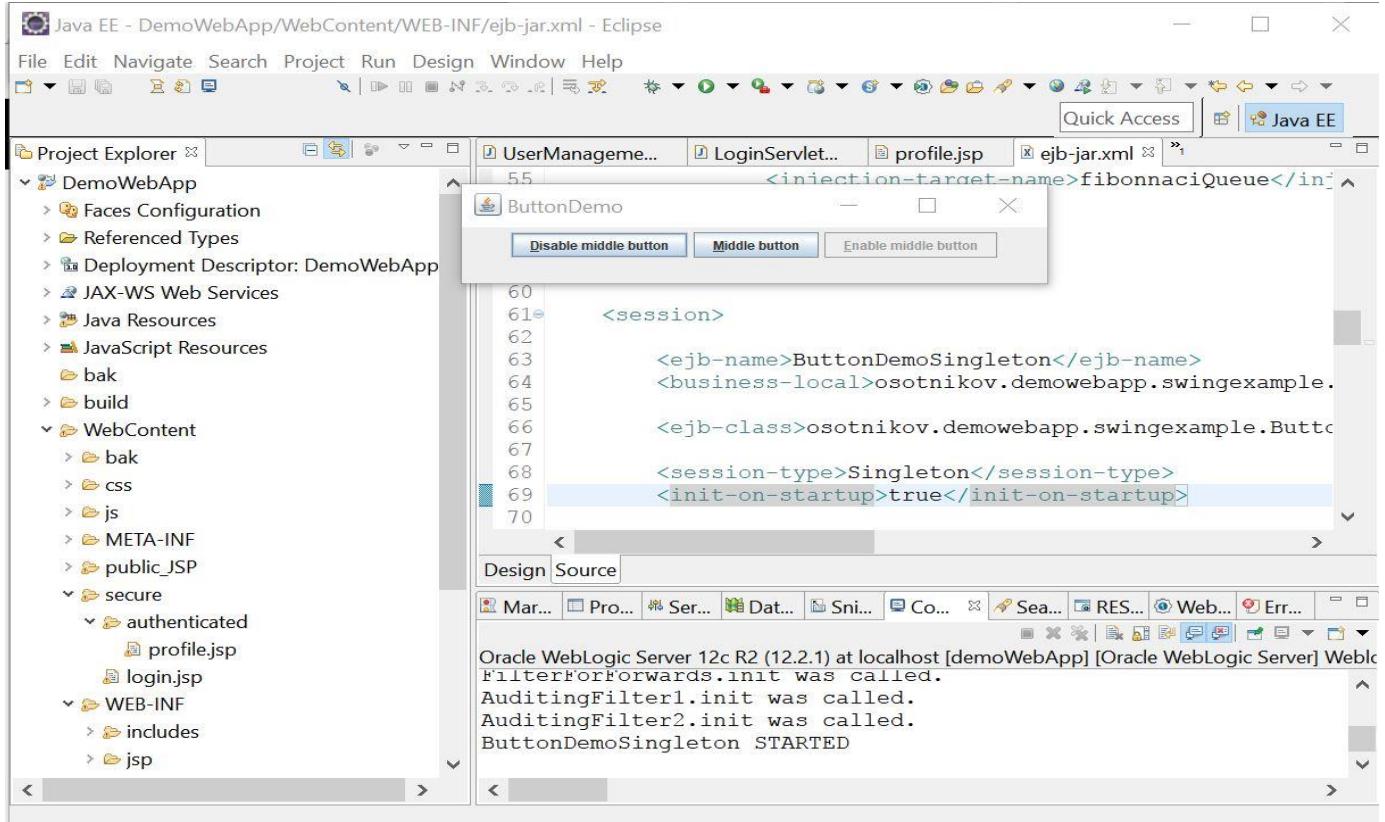
```

61<session>
62
63    <ejb-name>ButtonDemoSingleton</ejb-name>
64    <business-local>osotnikov.demowebapp.swingexample.ButtonDemoSingleton</business-local>
65
66    <ejb-class>osotnikov.demowebapp.swingexample.ButtonDemoSingletonImpl</ejb-class>
67
68    <session-type>Singleton</session-type>
69    <init-on-startup>true</init-on-startup>
70
71    <transaction-type>Container</transaction-type>
72
73</session>
74

```

Όπως φαίνεται είναι απαραίτητος ο ορισμός του ονόματος του EJB (**ejb-name**), του interface (**business-local**), της κλάσης υλοποίησης (**ejb-class**), του τύπου (**session-type**) καθώς και του element που είναι το αντίστοιχο του “**@Startup**” annotation που δημιουργεί το EJB κατά το ξεκίνημα της εφαρμογής (**init-on-startup**). Το transaction-type είναι προαιρετικό καθώς εάν δεν οριστεί εννοείται η τιμή “Container”, έχει να κάνει με την παράλληλη κλήση των μεθόδων του EJB και θα αναλυθεί στο επόμενο παράδειγμα.

## Αποτέλεσμα της εκτέλεσης του ButtonDemoSingleton



## LifecycleScopeAndConcurrencyExperimentsSingleton

Το επόμενο Singleton εξυπηρετεί τον σκοπό μίας κεντρικής, για όλη την εφαρμογή, δομής δεδομένων, η οποία αποθηκεύει στοιχειώδη παραδείγματα από Stateful και Stateless EJBs καθώς και CDI Managed Beans.

Η δομή αυτή θα χρησιμοποιείται στα επόμενα κεφάλαια των Stateful και Stateless Beans καθώς και του CDI.

Σε αυτήν την φάση καλό είναι να την δούμε απλώς σαν έναν τρόπος για να παρακάμψουμε το πρόβλημα της μη προσδιορισμένης διάρκειας ζωής ενός Servlet (αφού όπως έχουμε αναφέρει και στο κεφάλαιο των Servlets το πρότυπο ορίζει ότι ο Servlet Container είναι ελεύθερος να τα καταστρέψει οποιαδήποτε στιγμή και να ξαναδημιουργήσει νέα).

Συγκεκριμένα το Servlet **LifecycleScopeAndConcurrencyExperimentsServlet** έχει ως σκοπό να δέχεται αιτήσεις για κλήσεις προς **Stateless**, **Stateful** καθώς και **CDI Managed Beans** τα οποία παράγουν reports μετά από μία καθυστέρηση τριών δευτερολέπτων.

Τα reports αυτά επιστρέφονται στην σελίδα την οποία βλέπει ο χρήστης και αναφέρουν την κλάση του αντικειμένου που εξυπηρετεί την κλήση, το πότε λήφθηκε η κλήση, το πότε τελείωσε, έναν μοναδικό για το κάθε bean ακέραιο που συμβολίζει το internal data state του καθενός καθώς και τα αναγνωριστικά του bean (proxy) αλλά και του πραγματικού αντικειμένου (backing instance) που εξυπηρετεί την κλήση.

Εδώ υπενθυμίζεται ότι όπως αναφέρθηκε και στο κεφάλαιο του Dependency Injection οι κλήσεις προς τα EJBs καθώς και τα CDI Managed Beans δεν γίνονται πάνω στα πραγματικά αντικείμενα των κλάσεων υλοποίησης αλλά προς proxies (**Call Interception by the EJB Container**) που μεταφέρουν τα ορίσματα προς τα backing instances και επιστρέφουν το return value με την

οποιαδήποτε ενδιάμεση επεξεργασία προκύπτει ότι πρέπει να γίνει με βάση τα annotations που εφαρμόστηκαν στις αντίστοιχες κλάσεις και τις ρυθμίσεις από τον deployment descriptor.

Παρακάτω παρουσιάζεται ο κώδικας του **LifecycleScopeAndConcurrencyExperimentsSingleton**.

```
1 package osotnikov.demowebapp.services.concurrency;
2
3 import javax.ejb.EJB;□
10
11 // Singleton
12
13 /*Annotating a singleton class with @Lock specifies that all the business methods and any
14 timeout methods of the singleton will use the specified lock type unless they explicitly
15 set the lock type with a method-level @Lock annotation.*/
16 @Lock(LockType.READ)
17 public class LifecycleScopeAndConcurrencyExperimentsSingleton implements LifecycleScopeAndConcurrencyExperimentsLocal {
18
19     @EJB(beanName="statelessProcessingBean")
20     private ProcessingMockup statelessDependentScopedProcessingBean;
21
22     @EJB(beanName="statefulProcessingBean")
23     private ProcessingMockup statefulDependentScopedProcessingBean1;
24
25     @EJB(beanName="statefulProcessingBean")
26     private ProcessingMockup statefulDependentScopedProcessingBean2;
27
28     @Inject
29     @RequestScopedEJBQualifier
30     private ProcessingMockup statefulRequestScopedProcessingBean;
31
32     @Inject
33     private ProcessingMockup pojoProcessingBean1;
34     @Inject
35     private ProcessingMockup pojoProcessingBean2;
36     @Inject // @EJB Won't take into account the request scope!!!
37     @RequestScopedQualifier
38     private ProcessingMockup pojoRequestScopedProcessingBean;
39
40     @Override
41     @Lock(LockType.WRITE)
42     public ProcessingMockup getStatelessDependentScopedProcessingBeanWithWriteLock10Seconds() {
43         try{Thread.currentThread().sleep(10000);}catch(Exception e){}
44         return statelessDependentScopedProcessingBean;
45     }
46     @Override
47     public ProcessingMockup getStatelessDependentScopedProcessingBeanWithReadLock10Seconds() {
48         try{Thread.currentThread().sleep(10000);}catch(Exception e){}
49         return statelessDependentScopedProcessingBean;
50     }
51     @Override
52     @Lock(LockType.WRITE)
53     public ProcessingMockup getStatelessDependentScopedProcessingBeanWithWriteLockInstant() {
54         return statelessDependentScopedProcessingBean;
55     }
56
57     @Override
58     public ProcessingMockup getStatelessDependentScopedProcessingBeanWithReadLockInstant() {
59         return statelessDependentScopedProcessingBean;
60     }
61
62     @Override
63     public ProcessingMockup getStatefulDependentScopedProcessingBean1() {
64         return statefulDependentScopedProcessingBean1;
65     }
66
67     @Override
68     public ProcessingMockup getStatefulDependentScopedProcessingBean2() {
69         return statefulDependentScopedProcessingBean2;
70     }
71
72
```

```

73@Override
74 public ProcessingMockup getStatefulRequestScopedProcessingBean() {
75     return statefulRequestScopedProcessingBean;
76 }
77
78@Override
79 public ProcessingMockup getPojoProcessingBean1() {
80     return pojoProcessingBean1;
81 }
82
83@Override
84 public ProcessingMockup getPojoProcessingBean2() {
85     return pojoProcessingBean2;
86 }
87
88@Override
89 public ProcessingMockup getPojoRequestScopedProcessingBean() {
90     return pojoRequestScopedProcessingBean;
91 }
92
93}
94

```

Ουσιαστικά ο κώδικας της LifecycleScopeAndConcurrencyExperimentsSingleton κλάσης αποτελείται από 7 fields και τις αντίστοιχες getter μεθόδους (με εξαίρεση το fields statelessDependentScopedProcessingBean στην οποία αντιστοιχούν 4 getter μέθοδοι σε όλα τα άλλα fields αντιστοιχεί από μία).

Παρατηρούμε ότι πάνω στην κλάση αυτή ορίζεται το annotation “**@Lock**” με τιμή **“LockType.READ”**. Αυτό έχει το ίδιο αποτέλεσμα με αυτό που θα είχαμε εάν εφαρμόζαμε το annotation “**@Lock**” με την τιμή “**LockType.READ**” πάνω σε κάθε μέθοδο της κλάσης εκτός από τις μεθόδους στις οποίες είναι ήδη ορισμένο, στην περίπτωση αυτή το annotation που ορίζεται πάνω σε μία μέθοδο κάνει override αυτό που ορίζεται πάνω σε μια κλάση.

**Το annotation “@Lock” ρυθμίζει την συμπεριφορά του Singleton Bean κατά τις ταυτόχρονες κλήσεις που ενδέχεται να λάβει.**

**Εάν δεν οριστεί το annotation “@Lock” εννοείται η τιμή “LockType.WRITE”.** Εάν ισχύει αυτό τότε το Singleton Bean επιτρέπεται να προσπελάζεται μόνο από ένα Thread.

Γενικά εάν γίνει κλήση μίας μεθόδου κατά την διάρκεια της εκτέλεσης μίας άλλης (η της ίδιας) μεθόδου από άλλο Thread (και για κάποια από τις δύο ή και τις δύο ισχύει το “**LockType.WRITE**”), η μέθοδος αυτή θα πρέπει να μπλοκαριστεί και να περιμένει να τελειώσει η άλλη μέθοδος για να ξεκινήσει.

Εάν από την άλλη και οι δύο κλήσεις μεθόδων γίνονται πάνω σε μεθόδους με “**LockType.READ**” τότε δεν υπάρχει μπλοκάρισμα.

Αυτό μπορεί να επιβεβαιωθεί εάν μεταβούμε στην σελίδα των public services ([https://localhost:7002/DemoWebApp/public\\_JSP/public\\_services.jsp](https://localhost:7002/DemoWebApp/public_JSP/public_services.jsp)) και πειραματιστούμε με τα cases στην παράγραφο **“Lifecycle, Scope and Concurrency Experiments”**. Οι 4 πρώτες περιπτώσεις προορίζονται για να ελέγχουμε την συμπεριφορά ενός Singleton σε αιτήσεις για παράλληλη επεξεργασία.

Publicly Accessible Service

← → C ▲ Μη ασφαλής | https://localhost:7002/DemoWebApp/public\_JSP/public\_services.jsp

## Lifecycle, Scope and Concurrency Experiments

These examples have the goal of showing the behaviour of stateless and stateful beans, as well as POJOs that have been injected through CDI. Try pushing the buttons repeatedly to test concurrent execution. The request will result in a long running process in the back end (around 3 seconds).

### Stateless Bean with Dependent Scope with 10 second WRITE Lock

### Stateless Bean with Dependent Scope with 10 Second READ Lock

### Stateless Bean with Dependent Scope with Instant WRITE Lock

### Stateless Bean with Dependent Scope Instant READ Lock

### First Stateful Bean with Dependent Scope

Ο κώδικας που εξυπηρετήσει τις AJAX αιτήσεις από τα κουμπιά "Execute" βρίσκεται στην κλάση **LifecycleScopeAndConcurrencyExperimentsServlet**.

```
1 package osotnikov.demowebapp.web;
2
3 import java.io.IOException;
4
5 public class LifecycleScopeAndConcurrencyExperimentsServlet extends HttpServlet{
6
7     @EJB
8     private LifecycleScopeAndConcurrencyExperimentsLocal lcycleSingleton;
9
10    protected void doGet(HttpServletRequest req, HttpServletResponse response)
11        throws ServletException, IOException {
12
13        HttpServletRequest request = (HttpServletRequest) req;
14
15        System.out.println("LifecycleScopeAndConcurrencyExperimentsLocal.doGet: started for request: "
16            + request.getRequestURL());
17
18        String experimentType = request.getParameter("experimentType");
19
20        JsonResponse jsonRes = new JsonResponse(ResponseStatus.ERROR);
21        if(experimentType != null){
22
23            String processingReport = null;
24
25            lcycleSingleton.getStatelessDependentScopedProcessingBeanWithReadLockInstant().setMockupState(1);
26            lcycleSingleton.getStatefulDependentScopedProcessingBean1().setMockupState(2);
27            lcycleSingleton.getStatefulDependentScopedProcessingBean2().setMockupState(3);
28            lcycleSingleton.getStatefulRequestScopedProcessingBean().setMockupState(4);
29            lcycleSingleton.getPojoProcessingBean1().setMockupState(5);
30            lcycleSingleton.getPojoProcessingBean2().setMockupState(6);
31            lcycleSingleton.getPojoRequestScopedProcessingBean().setMockupState(7);
32
33            if(LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE_WRITE_LOCK_10_SEC.
34                getName().equals(experimentType)){
35
36                processingReport = lcycleSingleton.getStatelessDependentScopedProcessingBeanWithWriteLock10Seconds().process(
```

```

56     LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE_WRITE_LOCK_10_SEC.getName(),
57     3000);
58 }else if(LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE_READ_LOCK_10_SEC.
59   getName().equals(experimentType)){
60
61   processingReport = lcycleSingleton.getStatelessDependentScopedProcessingBeanWithReadLock10Seconds().process(
62     LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE_READ_LOCK_10_SEC.getName(),
63     3000);
64 }else if(LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE_WRITE_LOCK_INSTANT.
65   getName().equals(experimentType)){
66
67   processingReport = lcycleSingleton.getStatelessDependentScopedProcessingBeanWithWriteLockInstant().process(
68     LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE_WRITE_LOCK_INSTANT.getName(),
69     3000);
70 }else if(LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE.
71   getName().equals(experimentType)){
72
73   processingReport = lcycleSingleton.getStatelessDependentScopedProcessingBeanWithReadLockInstant().process(
74     LifecycleScopeAndConcurrencyExperimentType.STATELESS_DEPENDENT_SCOPE.getName(),
75     3000);
76 }
77 }else if(LifecycleScopeAndConcurrencyExperimentType.STATEFUL_DEPENDENT_SCOPE_1.
78   getName().equals(experimentType)){
79
80   processingReport = lcycleSingleton.getStatefulDependentScopedProcessingBean1().process(
81     LifecycleScopeAndConcurrencyExperimentType.STATEFUL_DEPENDENT_SCOPE_1.getName(),
82     3000);
83 }else if(LifecycleScopeAndConcurrencyExperimentType.STATEFUL_DEPENDENT_SCOPE_2.
84   getName().equals(experimentType)){
85
86   processingReport = lcycleSingleton.getStatefulDependentScopedProcessingBean2().process(
87     LifecycleScopeAndConcurrencyExperimentType.STATEFUL_DEPENDENT_SCOPE_2.getName(),
88     3000);
89 }else if(LifecycleScopeAndConcurrencyExperimentType.STATEFUL_REQ_SCOPE.
90   getName().equals(experimentType)){
91
92   processingReport = lcycleSingleton.getStatefulRequestScopedProcessingBean().process(
93     LifecycleScopeAndConcurrencyExperimentType.STATEFUL_REQ_SCOPE.getName(),
94     3000);
95 }else if(LifecycleScopeAndConcurrencyExperimentType.POJO_DEPENDENT_SCOPE_1.
96   getName().equals(experimentType)){
97
98   processingReport = lcycleSingleton.getPojoProcessingBean1().process(
99     LifecycleScopeAndConcurrencyExperimentType.POJO_DEPENDENT_SCOPE_1.getName(),
100    3000);
101 }else if(LifecycleScopeAndConcurrencyExperimentType.POJO_DEPENDENT_SCOPE_2.
102   getName().equals(experimentType)){
103
104   processingReport = lcycleSingleton.getPojoProcessingBean2().process(
105     LifecycleScopeAndConcurrencyExperimentType.POJO_DEPENDENT_SCOPE_2.getName(),
106     3000);
107 }else if(LifecycleScopeAndConcurrencyExperimentType.POJO_REQ_SCOPE.
108   getName().equals(experimentType)){
109
110   processingReport = lcycleSingleton.getPojoRequestScopedProcessingBean().process(
111     LifecycleScopeAndConcurrencyExperimentType.POJO_REQ_SCOPE.getName(),
112     3000);
113 }
114
115   jsonRes.setStatus(ResponseStatus.SUCCESS);
116   jsonRes.setData(processingReport);
117 }else{
118   jsonRes.setStatus(ResponseStatus.ERROR);
119 }
120
121 // RESPONSE
122 response.setContentType("application/json");
123 response.setCharacterEncoding("UTF-8");
124 PrintWriter resOut = response.getWriter();
125
126 ObjectMapper mapper = new ObjectMapper();
127
128 String jsonResString = mapper.writeValueAsString(jsonRes);
129
130 resOut.write(jsonResString);
131
132 System.out.println("LifecycleScopeAndConcurrencyExperimentsLocal.doGet: ended for request: "
133   + request.getRequestURL());
134
135
136 protected void doPost(HttpServletRequest request, HttpServletResponse response)
137   throws ServletException, IOException {
138
139   doGet(request, response);
140
141 }
142
143 }
```

Τα cases στην παράγραφο “Lifecycle, Scope and Concurrency Experiments” της σελίδας των public services αντιστοιχούνε με τα cases της if-else if δομής. Σε κάθε case γίνεται κλήση προς το Singleton για να πάρουμε ένα bean τύπου **ProcessingMockup** που υλοποιεί την μέθοδο **process(String mockupName, long duration)** η οποία επιστρέφει ένα processing report το οποίο είναι αυτό που εμφανίζεται μετά το πάτημα οποιουδήποτε “Execute” κουμπιού.

Για τους σκοπούς της ανάλυσης της συμπεριφοράς ενός Singleton Bean στην περίπτωση αιτήσεων για παράλληλη επεξεργασία προς αυτό μας ενδιαφέρουν μόνο τα 4 πρώτα cases της if-else if δομής.

Σε αυτά τα cases υπάρχουν δύο όπου η προσπέλαση του αντικειμένου τύπου ProcessingMockup που χρειάζεται για την εκτέλεση της μεθόδου process, καθυστερεί επίτηδες για 10 δευτερόλεπτα. Αυτά τα cases είναι τα **STATELESS\_DEPENDENT\_SCOPE\_WRITE\_LOCK\_10\_SEC** και **STATELESS\_DEPENDENT\_SCOPE\_READ\_LOCK\_10\_SEC**. Αυτό γίνεται επειδή στις αντίστοιχες μεθόδους getStatelessDependentScopedProcessingBeanWithWriteLock10Seconds και getStatelessDependentScopedProcessingBeanWithReadLock10Seconds κάνουμε κλήση της **Thread.currentThread().sleep(10000)** μεθόδου που κάνει παύση του Thread όπου εκτελείται η μέθοδος. Επίσης αυτές οι δύο μέθοδοι έχουν οριστεί με ένα **LockType.WRITE** και **LockType.READ** αντίστοιχα.

Από τα προηγούμενα είναι φανερό ότι εάν πατήσουμε το κουμπί “Execute” στο case “**Stateless Bean with Dependent Scope with 10 second WRITE Lock**” και μετά (όσο πιο γρήγορα μπορούμε) πατήσουμε π.χ. το “Execute” στο case “**Stateless Bean with Dependent Scope Instant READ Lock**” που δεν έχει καθυστέρηση, το πιθανότερο είναι να δούμε το processing report αυτών των δύο να εμφανίζεται σχεδόν ταυτόχρονα. Πάντως σε κάθε περίπτωση το processing report του “**Stateless Bean with Dependent Scope Instant READ Lock**” θα καθυστερήσει να εμφανιστεί αφού θα πρέπει να περιμένει μέχρι να τελειώσει η εκτέλεση της **getStatelessDependentScopedProcessingBeanWithWriteLock10Seconds** για να ξεκινήσει.

Στην επόμενη εικόνα έχουμε πατήσει τα κουμπιά “Execute” των δύο cases που προαναφέραμε με λιγότερο από ένα δευτερόλεπτο διαφορά (πατώντας πρώτα αυτό που αντιστοιχεί στο “**Stateless Bean with Dependent Scope with 10 second WRITE Lock**”). Παρόλα αυτά τα processing reports εμφανίστηκαν 10 συν 3 δευτερόλεπτα μετά και αναφέρουν ότι ξεκινήσανε και τελειώσανε τις ίδιες ώρες. Αυτό συμβαίνει επειδή η καθυστέρηση από την **getStatelessDependentScopedProcessingBeanWithWriteLock10Seconds** ήταν 10 δευτερόλεπτα και η καθυστέρηση από τις process μεθόδους που μετά από αυτό ξεκινάνε ταυτόχρονα και εκτελούνται παράλληλα 3.

**Τα Stateless Session Beans** (τέτοιο είναι το αντικείμενο πάνω στο οποίο γίνονται τα δύο invocations της process μεθόδου) σε αντίθεση με τα Singleton Beans επιτρέπουν την ταυτόχρονη εκτέλεση των μεθόδων τους σε οποιαδήποτε περίπτωση και δεν γίνεται να ρυθμιστούν διαφορετικά. Αυτός είναι και ο λόγος που μετά από 10 δευτερόλεπτα, στην επόμενη εικόνα, θα δούμε ότι τα δύο reports εμφανίζουνε ότι ξεκινήσανε και τελειώσανε την ίδια ώρα (παράγονται μέσω του ίδιου Stateless Bean).

**Lifecycle, Scope and Concurrency Experiments**

These examples have the goal of showing the behaviour of stateless and stateful beans, as well as POJOs that have been injected through CDI. Try pushing the buttons repeatedly to test concurrent execution. The request will result in a long running process in the back end (around 3 seconds).

**Stateless Bean with Dependent Scope with 10 second WRITE Lock**

**Execute**

statelessProcessingBean\_3gqdz4\_Impl: [STATELESS\_DEPENDENT\_SCOPE\_WRITE\_LOCK\_10\_SEC] with EJB proxy hash code: [osotnikov демовебапп.services.concurrency.statelessProcessingBean\_3gqdz4\_ProcessingMockupImpl@4856f2c8] with processing object hash code [1902430632] whose internal state is [1] started at [Fri Jun 30 17:17:22 EEST 2017] ended at [Fri Jun 30 17:17:25 EEST 2017]

**Stateless Bean with Dependent Scope with 10 Second READ Lock**

**Execute**

**Stateless Bean with Dependent Scope with Instant WRITE Lock**

**Execute**

**Stateless Bean with Dependent Scope Instant READ Lock**

**Execute**

statelessProcessingBean\_3gqdz4\_Impl: [STATELESS\_DEPENDENT\_SCOPE] with EJB proxy hash code: [osotnikov демовебапп.services.concurrency.statelessProcessingBean\_3gqdz4\_ProcessingMockupImpl@4856f2c8] with processing object hash code [195037493] whose internal state is [0] started at [Fri Jun 30 17:17:22 EEST 2017] ended at [Fri Jun 30 17:17:25 EEST 2017]

#### First Stateful Bean with Dependent Scope

Πρέπει να σημειωθεί πάντως ότι οι καθυστερήσεις που προαναφέρθηκαν δεν είναι ακριβείς και ενδέχεται να είναι μεγαλύτερες εξαιτίας της φύσης της επικοινωνίας του Browser με τον Application Server στην οποία εμπλέκονται πολλές παράμετροι.

Την ίδια συμπεριφορά θα έχουμε και εάν πατήσουμε τα κουμπιά “Execute” με μικρή χρονική διαφορά μεταξύ τους, στα cases **“Stateless Bean with Dependent Scope with 10 Second READ Lock”** και **“Stateless Bean with Dependent Scope with Instant WRITE Lock”** (πατώντας πρώτο αυτό που αντιστοιχεί στο case **“Stateless Bean with Dependent Scope with 10 Second READ Lock”**).

**Lifecycle, Scope and Concurrency Experiments**

These examples have the goal of showing the behaviour of stateless and stateful beans, as well as POJOs that have been injected through CDI. Try pushing the buttons repeatedly to test concurrent execution. The request will result in a long running process in the back end (around 3 seconds).

**Stateless Bean with Dependent Scope with 10 second WRITE Lock**

**Execute**

**Stateless Bean with Dependent Scope with 10 Second READ Lock**

**Execute**

statelessProcessingBean\_3gqdz4\_Impl: [STATELESS\_DEPENDENT\_SCOPE\_READ\_LOCK\_10\_SEC] with EJB proxy hash code: [osotnikov демовебапп.services.concurrency.statelessProcessingBean\_3gqdz4\_ProcessingMockupImpl@19ed517b] with processing object hash code [1045234492] whose internal state is [1] started at [Fri Jun 30 19:17:10 EEST 2017] ended at [Fri Jun 30 19:17:13 EEST 2017]

**Stateless Bean with Dependent Scope with Instant WRITE Lock**

**Execute**

statelessProcessingBean\_3gqdz4\_Impl: [STATELESS\_DEPENDENT\_SCOPE\_WRITE\_LOCK\_INSTANT] with EJB proxy hash code: [osotnikov демовебапп.services.concurrency.statelessProcessingBean\_3gqdz4\_ProcessingMockupImpl@19ed517b] with processing object hash code [1989834769] whose internal state is [1] started at [Fri Jun 30 19:17:10 EEST 2017] ended at [Fri Jun 30 19:17:13 EEST 2017]

**Stateless Bean with Dependent Scope Instant READ Lock**

**Execute**

Αυτή η συμπεριφορά είναι λογική αφού **οποιαδήποτε μέθοδος με WRITE Lock θα πρέπει σε κάθε περίπτωση να είναι η μόνη που εκτελείται πάνω σε ένα Singleton μία δεδομένη χρονική στιγμή.**

## Stateless Session Beans

### **Γενική Περιγραφή**

Τα Stateless Session Beans χρησιμοποιούνται σε περίπτωση που θέλουμε να μοντελοποιήσουμε λειτουργικότητα η οποία δεν βασίζεται στην επεξεργασία κάποιας δομής δεδομένων μεταξύ διαδοχικών κλήσεων πάνω στο Stateless Bean αλλά πρέπει να διαχειριστούμε παράλληλες κλήσεις προς αυτό.

**Σε αντίθεση με τα Singleton Session Beans αλλά και τα Stateful Session Beans, σε ένα Stateless Session Bean (proxy) αντιστοιχούν περισσότερα από ένα αντικείμενα (backing instances).**

Συγκεκριμένα για το Stateless Session Bean ισχύει ότι υπάρχει ένα pool από instances και κάθε φορά που καλείται μία μέθοδος του Stateless Bean επιλέγεται ένα από αυτά τα instances από το proxy και εξυπηρετείται η κλήση της μεθόδου. Εάν παράλληλα υπάρχει και άλλη κλήση προς το proxy αυτό θα επιλέξει κάποιο άλλο instance από το pool. Εάν τελειώσουν τα instances μέσα στο pool ή θα γίνει μπλοκάρισμα της επόμενης κλήσης μέχρι να απελευθερωθεί ένα instance ή θα δημιουργηθεί καινούριο και θα μεγαλώσει το instance pool. Σε καμία περίπτωση πάντως δεν πρόκειται να εκτελούνται μέθοδοι από διαφορετικά Threads παράλληλα πάνω στο ίδιο instance.

**Επειδή κάθε φορά που γίνεται κλήση μιας μεθόδου του bean επιλέγεται τυχαία ένα backing instance από το pool δεν μπορούμε να είμαστε σίγουροι για τις τιμές των μεταβλητών στα fields αυτού του instance, σε αυτά τα fields δεν εννοούμε αυτά που υλοποιούν dependencies από άλλα beans ή resources αλλά αυτά που μοντελοποιούν κάποιο εσωτερικό state.**

Εάν υπάρχουν κάποια fields τα οποία υλοποιούν dependencies από άλλα beans ή resources τότε αυτά θα αρχικοποιηθούν κανονικά κατά την δημιουργία του instance και θα παραμείνουν καθ' όλη την διάρκεια ζωής του bean ώστε να μπορούν να εξυπηρετούν τις μεθόδους του.

Για οποιαδήποτε άλλα fields όπως π.χ. username, selectedBgColor κ.τ.λ. δεν μπορούμε να είμαστε σίγουροι για τις τιμές τους μεταξύ κλήσεων προς το bean. Επειδή μεταξύ μιας κλήσης μίας μεθόδου και μίας δεύτερης μπορεί να παρεμβληθούν και άλλες (από άλλα Threads) οι οποίες αλλάζουν την τιμή ενός τέτοιου field καθώς και επειδή το πιθανότερο είναι η δεύτερη κλήση να εξυπηρετηθεί από άλλο backing instance στο οποίο το field ίσως να μην έχει καν αρχικοποιηθεί χρησιμοποιούμε τέτοια fields μόνο στα πλαίσια της κλήσης μίας μόνο μεθόδου, προφανώς αυτό συμπεριλαμβάνει την περίπτωση όπου μία μέθοδος καλεί εσωτερικά άλλες μεθόδους, σε αυτήν την περίπτωση τα εσωτερικά δεδομένα αλλάζουν μόνο από αυτές.

### **Παράδειγμα με Κώδικα**

#### **UserManagementService**

Ως παράδειγμα ενός Stateless Session Bean θα χρησιμοποιήσουμε το **UserManagementService**. Είναι το bean που χρησιμοποιείται από τα Servlets **RegistrationServlet** και **LoginServlet** και υλοποιεί τις ενέργειες που είναι απαραίτητες για το registration και το log in των χρηστών του DemoWebApp.

Θα εξετάσουμε την λειτουργία του στα πλαίσια του **RegistrationServlet**.

```

1 package osotnikov.demowebapp.web;
2
3 import java.io.IOException;
4
5
6 public class RegistrationServlet extends HttpServlet {
7
8     @EJB(beanName="UserManagementService")
9     UserManagementService userManagementBean;
10
11     public RegistrationServlet() {
12         super();
13     }
14
15     protected void doPost(HttpServletRequest request, HttpServletResponse response)
16         throws ServletException, IOException {
17
18         System.out.println("RegistrationServlet.doPost started for request: "
19             + request.getRequestURL());
20
21         String jsonData = request.getParameter("jsonData");
22
23         System.out.println("RegistrationServlet.doPost: received jsonData: " + jsonData);
24
25         ObjectMapper mapper = new ObjectMapper();
26
27         User user = mapper.readValue(jsonData,User.class);
28
29         System.out.println("RegistrationServlet.doPost: parsed jsonData to User: " + user);
30
31         user = userManagementBean.validateTweakAndRegisterUser(user);
32
33         if(user != null){
34             System.out.println("RegistrationServlet.doPost: Successfully registered user: " + user);
35         }else{
36             System.out.println("RegistrationServlet.doPost: Failed to register the user.");
37         }
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68 }

```

Όπως παρατηρούμε αρχικά το UserManagementService δηλώνεται ως dependency στο Servlet στις γραμμές 18 και 19 μέσω του annotation **@EJB** το οποίο με το attribute **“beanName”** δηλώνει το όνομα που δόθηκε σε αυτό το bean για να εντοπιστεί από τον IoC Container. Στην συνέχεια το bean αυτό χρησιμοποιείται στην γραμμή 41 όπου καλείται η μέθοδός του **validateTweakAndRegisterUser** η οποία θα ελέγξει όλα τα στοιχεία που δόθηκαν στην φόρμα του registration και εάν υπάρχουν ελλιπή ή μη έγκυρα στοιχεία δεν θα προχωρήσει στο registration. Εάν από την άλλη είναι έγκυρα θα αφαιρέσει τυχόν whitespaces πριν και μετά από τις τιμές και θα ολοκληρώσει το registration.

```

1 package osotnikov.demowebapp.services.user_management.session;
2
3 *import java.util.ArrayList;*
4
5 @Stateless(name="UserManagementService")
6 public class UserManagementService {
7
8     @PersistenceContext(unitName="authPU")
9     private EntityManager em;
10
11     @Inject
12     StringUtils stringUtils;
13
14     private User registeringUser;
15
16     public User fetchUser(String email){
17         User user = em.find(User.class, email); // DAO layer exceptions should be runtime and handled
18         // somewhere all together because you can't really recover from them.
19         return user;
20     }
21
22     /** @return null if the user does not exist or log in failed, the user otherwise... */
23     public User loginAndFetchUser(String email, String password, HttpServletRequest req) {
24
25         email = stringUtils.getTrimmedOrEmpty(email);
26
27         User user = fetchUser(email);
28         System.out.println("UserManagementService.loginAndFetchUser: "
29             + "successfully retrieved User Profile from DB: " + user);
30
31         // Only login if not already logged in...
32         if(user != null && req.getUserPrincipal() == null){
33             try {
34                 req.login(email, password); // HttpServletRequest.login means that we'll
35                 // have getUserPrincipal returning something after that ...
36                 System.out.println("UserManagementService.loginAndFetchUser: successfully "
37
38                     + "logged in " + email);
39             } catch (ServletException e) {
40                 user = null;
41                 e.printStackTrace();
42             }
43             }else if (user == null){
44                 System.out.println("UserManagementService.loginAndFetchUser: No user found for "
45                     + "email: "+email);
46             }else /*if (user != null && req.getUserPrincipal() != null)*{
47                 System.out.println("UserManagementService.loginAndFetchUser: Skip logged because "
48                     + "already logged in: "+email);
49             }
50
51             return user;
52         }
53
54         /** @return true if the logout succeeded, false if it failed. */
55         public boolean logout(HttpServletRequest req) {
56
57             try {
58                 req.logout();
59                 req.getSession().invalidate();
60                 return true;
61             } catch (ServletException e) {
62                 e.printStackTrace();
63                 System.out.println("UserManagementService.loginAndFetchUser: Logout failed on backend");
64             }
65
66             return false;
67         }
68
69         /** @return The User object that was passed with possibly the whitespace of its fields trimmed
70         * or null if the User object that was passed had invalid data. */
71         public void validateAndTweakUser() {
72             // VALIDATE AND TWEAK REGISTRATION DATA
73
74
75
76
77
78
79
80
81
82

```

```

83     // Check for empty fields.
84     if(stringUtils.isEmpty(registeringUser.getEmail()) ||
85         stringUtils.isEmpty(registeringUser.getFirstName()) ||
86         stringUtils.isEmpty(registeringUser.getLastName()) ||
87         stringUtils.isEmpty(registeringUser.getPassword()) ||
88         stringUtils.isEmpty(registeringUser.getConfirmPasswordValue())) {
89
90         registeringUser = null;
91     }else if ( ! registeringUser.getConfirmPasswordValue().equals(registeringUser.getPassword()) ) {
92         registeringUser = null;
93     }else{
94         // Remove whitespace.
95         registeringUser.setEmail(registeringUser.getEmail().trim());
96         registeringUser.setFirstName(registeringUser.getFirstName().trim());
97         registeringUser.setLastName(registeringUser.getLastName().trim());
98
99         System.out.println("UserManagementService.validateAndTweakUser, user is valid: "
100             + registeringUser.toString());
101     }
102 }
103
104 /**
105 * @return The User object that was passed with possibly the whitespace of its fields
106 * trimmed or null if the User object that was passed had invalid data. */
107 public User validateTweakAndRegisterUser(User registeringUser) {
108
109     // VALIDATE AND TWEAK REGISTRATION DATA
110     this.registeringUser = registeringUser;
111
112     validateAndTweakUser();
113
114     if(this.registeringUser == null){
115         return null; // The user data is invalid. Registration failed.
116     }
117
118     // Register the user with all the groups.

119     List<GroupEnum> groups = new ArrayList<GroupEnum>();
120     groups.add(GroupEnum.ADMINISTRATOR);
121     groups.add(GroupEnum.USER);
122     groups.add(GroupEnum.DEFAULT);
123     registeringUser.setGroups(groups);
124
125     registeringUser.setRegisteredOn(new java.util.Date());
126
127     try{
128         String encPass = "{SHA-1}" + new sun.misc.BASE64Encoder()
129             .encode(java.security.MessageDigest.getInstance("SHA1")
130                 .digest(registeringUser.getPassword().getBytes()));
131         registeringUser.setPassword(encPass);
132
133         System.out.println("Successfully encrypted the password of the user with SHA1 to: "
134             + registeringUser.getPassword());
135     }catch(Exception e){
136         e.printStackTrace();
137         throw new RuntimeException();
138     }
139
140     // This could cause a runtime exception, i.e. in case the user already exists.
141     em.persist(registeringUser);
142     System.out.println("UserManagementService.validateTweakAndRegisterUser: Successfully "
143         + "registered new user: '" + registeringUser);
144
145     return registeringUser;
146 }
147 }
```

Όπως παρατηρούμε το bean αυτό ορίζει στα **dependencies** του έναν **EntityManager (em)** και ένα αντικείμενο της κλάσης **StringUtils (stringUtils)**.

Το αντικείμενο τύπου **EntityManager** είναι απαραίτητο για να γίνεται επικοινωνία με την ΒΔ για την αποθήκευση του registered χρήστη και την εύρεση και ανάγνωση του κατά το login. Το dependency για αντικείμενο τέτοιου τύπου πρέπει να ικανοποιείται μέσω του annotation **@PersistenceContext**. Αφορά το **JPA (Java Persistence API)** και τα **Entity Beans** για αυτό και δεν θα αναλυθεί εις βάθος σε αυτό το κεφάλαιο.

Το αντικείμενο τύπου **StringUtils** είναι απλώς ένα instance με μεθόδους για την επεξεργασία συμβολοσειρών και θα χρησιμοποιηθεί για να ελέγχουμε εάν έχουμε κενές συμβολοσειρές και επίσης για την αφαίρεση περιττού whitespace από τις τιμές που εισήγαγε ο χρήστης. Δεν αποτελεί EJB αλλά απλώς ένα managed bean που εισάγετε μέσω του **CDI** προτύπου της Java (**Context and Dependency Injection**). Η ανάλυση αυτού του προτύπου θα γίνει σε επόμενο κεφάλαιο.

Αυτά τα δύο dependencies θα υπάρχουν σε κάθε backing instance του Stateless Bean. Δεν χρειάζεται να αποθηκεύουν δεδομένα (state) που να είναι διαφορετικά για κάθε client για αυτόν τον λόγο δεν υπάρχει πρόβλημα στην χρήση τους (**στην ορολογία του EJB προτύπου ο client ενός EJB είναι το component που κάνει χρήση του proxy του bean και όχι ο πελάτης του enterprise application που την προσπελάζει μέσω browser**).

Υπάρχει όμως και ένα τρίτο field, το **registeringUser**. Αυτό ορίστηκε για να τονιστεί ότι στα πλαίσια της κλήσης μίας μόνο μεθόδου από τον client μπορούμε να αποθηκεύσουμε προσωρινά δεδομένα σε fields ενός Stateless Bean.

To field αυτό αποθηκεύει την παράμετρο που αρχικά περνάει στην **validateTweakAndRegisterUser** από τον RegistrationServlet (η παράμετρος αυτή είναι τύπου User και έχει τα δεδομένα όπως ακριβώς τα πέρασε ο χρήστης συμπληρώνοντας την φόρμα του registration).

Το πρώτο πράγμα που κάνει η validateTweakAndRegisterUser αφού αρχικοποιήσει την τιμή του registeringUser field είναι να καλέσει την **validateAndTweakUser**. Αντί να τις περάσει ως παράμετρο το registeringUser την αφήνει να το διαβάσει από το registeringUser field αφού είναι σίγουρο ότι ο κώδικας εκτελείται στο ίδιο backing instance όπου δεν πρόκειται να παρεμβληθεί κλήση από άλλο Thread. Η validateAndTweakUser θα ελέγχει το registeringUser για το εάν όλα τα πεδία του είναι έγκυρα. Εάν δεν είναι θα θέσει το registeringUser σε null κάτι το οποίο δηλώνει αποτυχία του registration. Άλλιως απλώς θα αφαιρέσει τυχόν extra whitespaces από τα πεδία.

Τώρα θα δοθεί ένα παράδειγμα λανθασμένης χρήσης ενός Stateless Session Bean. Έστω ότι η validateTweakAndRegisterUser δεν πραγματοποιούσε κλήση της validateAndTweakUser εσωτερικά αλλά πραγματοποιούσαμε κλήσεις και των δύο μεθόδων μέσα στην doPost του RegistrationServlet καλώντας πρώτα την validateAndTweakUser και μετά την validateTweakAndRegisterUser. Έστω επίσης ότι η παράμετρος registeringUser περνούσε στην validateAndTweakUser και αυτή αρχικοποιούσε το field registeringUser εσωτερικά του userManagementBean το οποίο μετά και θα χρησιμοποιούσε η validateTweakAndRegisterUser. Σε αυτήν την περίπτωση ο κώδικάς μας θα ήταν λάθος επειδή το backing instance που θα χρησιμοποιούνταν κατά την κλήση της validateAndTweakUser ενδέχεται να ήταν διαφορετικό από αυτό που θα χρησιμοποιούνταν κατά την validateTweakAndRegisterUser άρα και το field registeringUser δεν θα είχε την τιμή την οποία θα του είχε αφήσει η validateAndTweakUser.

Προφανώς ο τρόπος αυτός (του να αποφεύγουμε να περνάμε παραμέτρους από μία μέθοδο σε μία άλλη με το να τις αποθηκεύουμε στα fields του αντικειμένου) φαίνεται περίεργος στο συγκεκριμένο παράδειγμα και δεν έχει κάποια αξία πρακτικά, όμως μπορούμε να δούμε την αξία του εάν φανταστούμε μία περίπτωση όπου γίνεται κλήση πολλών μεθόδων του bean από μια μέθοδο και οι παράμετροι που πρέπει να του περνιούνται είναι πολλές. Σε αυτήν την περίπτωση αυτός ο τρόπος μας προσφέρει μία διευκόλυνση.

## Lifecycle, Scope and Concurrency Experiments

Στην σελίδα των public services στην παράγραφο **Lifecycle, Scope and Concurrency Experiments** που αναφέρθηκε και στο κεφάλαιο για τα Singleton Beans μπορούμε να πειραματιστούμε με το case **“Stateless Bean with Dependent Scope Instant READ Lock”** για να κατανοήσουμε καλύτερα τον τρόπο συμπεριφοράς των Stateless Beans στις παράλληλες κλήσεις. Εάν π.χ. πατήσουμε αρκετές φορές το κουμπί “Execute” με το κάθε πάτημα να γίνεται ανά ένα δευτερόλεπτο τότε το πιθανότερο είναι να παρατηρήσουμε ότι ο χρόνος που ξεκινάει η διαδικασία αλλάζει και αυτός ανά ένα δευτερόλεπτο, αυτό δείχνει ότι υπάρχει παράλληλη επεξεργασία των αιτήσεων και όχι μπλοκάρισμά τους και επεξεργασία τους μία μία. Επίσης θα παρατηρήσουμε ότι το “processing object hash code” θα αλλάζει και αυτό εάν πατάμε το κουμπί περισσότερες από μία φορές ανά 3 δευτερόλεπτα επειδή για να γίνει η παράλληλη επεξεργασία απαιτείται να χρησιμοποιηθεί ένα διαφορετικό backing instance από το instance pool. Τέλος θα παρατηρήσουμε ότι παρόλο που το internal state είχε τεθεί σε “1” μέσα στον LifecycleScopeAndConcurrencyExperimentsServlet ειδικά στις περιπτώσεις παράλληλης επεξεργασίας θα παίρνει μερικές φορές την τιμή 0. Αυτό συμβαίνει επειδή τέθηκε σε “1” μόνο σε ένα backing instance και όχι σε όλα.

## Stateful Session Beans

### **Γενική Περιγραφή**

Τα Stateful Session Beans χρησιμοποιούνται σε περίπτωση που χρειάζεται να αποθηκεύσουμε internal state μεταξύ των κλήσεων του client (όπως ορίζεται στο EJB πρότυπο) προς το bean. Σε αντίθεση με τα Stateless Session Beans σε κάθε Stateful Bean αντιστοιχεί μόνο ένα backing instance.

Όσον αφορά την διαχείριση της παραλληλίας τα Stateful Session Beans θα μπλοκάρουν τις παράλληλες κλήσεις και θα τις εκτελούν μία μία έτσι ώστε να εκτελείται μόνο μία μέθοδος κάθε φορά πάνω στο bean (ένα Thread). Δεν υπάρχει δυνατότητα για την παραμετροποίηση αυτής της συμπεριφοράς ώστε να καταστήσουμε κάποιες μεθόδους να είναι δυνατόν να εκτελούνται παράλληλα πάνω στο ίδιο bean όπως π.χ. κάναμε στο Singleton Bean. Πάντως σημειώνεται ότι σε παλαιότερες εκδόσεις του προτύπου αντί να γίνεται μπλοκάρισμα των παράλληλων κλήσεων δημιουργούνταν ένα Exception στην περίπτωση της κλήσης μεθόδου πάνω στο bean ενώ ταυτόχρονα εκτελούνταν κάποια άλλη. Ο Weblogic 12C δίνει την δυνατότητα του ορισμού αυτής της παλαιάς λειτουργικότητας μέσω του element **<allow-concurrent-calls>** σε συνδυασμό με κάποια άλλα elements στον weblogic-ejb-jar.xml runtime descriptor αλλά γενικά προτιμάται η χρήση του annotation **javax.ejb.AccessTimeout** που ορίζει ότι θα προκύψει exception μόνο εάν μία μέθοδος έχει μπλοκάρει την πρόσβαση στο bean πάνω από το ορισμένο χρονικό όριο. Το javax.ejb.AccessTimeout προφανώς ορίζεται στο EJB πρότυπο και είναι διαθέσιμο σε κάθε υλοποίησή του.

Η διάρκεια ζωής ενός Stateful Session Bean είναι ίδια με αυτή του client (όπως ορίζεται στο EJB πρότυπο), δηλαδή του component, στον οποίον έγινε injected εκτός και αν το injection έχει γίνει κατά το runtime μέσω JNDI lookup ή lookup μέσω του EJBContext, σε αυτήν την περίπτωση η διάρκεια ζωής του είναι ίδια με την διάρκεια ζωής του proxy αντικειμένου.

## CounterServiceBean

Παράδειγμα ενός Stateful Bean στην εφαρμογή DemoWebApp είναι το **CounterServiceBean** που χρησιμοποιείται από το Servlet **CountButtonPressServlet**.

Ο σκοπός αυτού του Servlet είναι να μετρήσει το πόσες φορές κάποιος πάτησε το κουμπί “**Count Button Press**” στην παράγραφο “**Get button press count:**” στην σελίδα των **public services** κατά την διάρκεια ζωής του Servlet **CountButtonPressServlet** (όπως έχουμε αναφέρει προηγουμένως η διάρκεια ζωής ενός Servlet είναι ακαθόριστη).

**Παρουσιάζεται το local interface του bean που θα υλοποιήσει τον counter:**

```
1 package osotnikov.demowebapp.services.counter;
2
3 import javax.ejb.Local;
4
5 @Local
6 public interface CounterService {
7     public int getCount();
8     public void incrementCount();
9 }
```

**To CounterServiceBean που θα υλοποιήσει τον μετρητή τον κλικ:**

```
1 package osotnikov.demowebapp.services.counter;
2
3 import javax.ejb.Stateful;
4
5 @Stateful(name="CounterServiceBean")
6 public class CounterServiceBean implements CounterService{
7
8     int count = 0;
9
10    @Override
11    public int getCount() {
12        // TODO Auto-generated method stub
13        return count;
14    }
15
16    @Override
17    public void incrementCount() {
18        ++count;
19    }
20
21 }
```

Όπως βλέπουμε η υλοποίηση δεν παρουσιάζει καμία πολυπλοκότητα. Παρόλα αυτά δεν θα μπορούσε να υλοποιηθεί εάν δεν γινόταν με ένα bean που δημιουργεί μόνο ένα instance και το οποίο μπλοκάρει την πρόσβαση στις μεθόδους του όταν ήδη χρησιμοποιούνται εκείνη την στιγμή από κάποιο Thread.

## Παρουσιάζεται και ο αντίστοιχος Servlet:

```
1 package osotnikov демовебапп.web;
2
3 import java.io.IOException;
4
5 public class CountButtonPressServlet extends HttpServlet {
6
7     @EJB (beanName="CounterServiceBean")
8     CounterService counterService;
9
10    public CountButtonPressServlet() {
11        super();
12    }
13
14    protected void doGet(HttpServletRequest request, HttpServletResponse response)
15        throws ServletException, IOException {
16
17        System.out.println("CountButtonPressServlet.doGet: started for request: "
18            + request.getRequestURL());
19
20        counterService.incrementCount();
21        int curCount = counterService.getCount();
22
23        System.out.println("CountButtonPressServlet, curCount: " + curCount);
24
25        JsonResponse jsonRes = new JsonResponse(ResponseStatus.SUCCESS); // The json response object.
26
27        // CREATE THE HTTP RESPONSE BODY
28
29        ObjectMapper mapper = new ObjectMapper();
30        jsonRes.setData(new Integer(curCount));
31
32        String jsonResponseString = mapper.writeValueAsString(jsonRes);
33
34        System.out.println("CountButtonPressServlet.doGet: jsonResponseString: " +
35            jsonResponseString);
36
37
38        response.setContentType("application/json");
39        response.setCharacterEncoding("UTF-8");
40        PrintWriter resOut = response.getWriter();
41        resOut.write(jsonResponseString);
42
43        System.out.println("CountButtonPressServlet.doGet: ended for request: "
44            + request.getRequestURL());
45
46    }
47
48
49    protected void doPost(HttpServletRequest request, HttpServletResponse response)
50        throws ServletException, IOException {
51
52        System.out.println("CountButtonPressServlet.doPost: started ... just calls doGet ...");
53        doGet(request, response);
54
55    }
56
57 }
58
```

Και ο Servlet δεν παρουσιάζει καμία δυσκολία στην κατανόηση. Όταν δέχεται ένα αίτημα ανεβάζει τον μετρητή κατά μία μονάδα και στο response που παράγει στέλνει την καινούρια του τιμή.

Στην αντίστοιχη σελίδα ο χρήστης θα παρατηρεί απλώς ότι θα αυξάνεται ένας μετρητής κάθε φορά που πατάει το κουμπί και όταν θα ξαναεπισκέπτεται την σελίδα θα βλέπει την τιμή που είχε όταν έφυγε από αυτήν. Στην περίπτωση που η εφαρμογή προσπελάζεται από πολλούς χρήστες η τιμή θα προκύπτει από όλους τους οπότε όποτε κάποιος την ξαναεπισκέπτεται ενδέχεται να δει μία πολύ μεγαλύτερη τιμή από αυτήν που υπήρχε όταν έφυγε, ανάλογα με το πόσες φορές πατήθηκε το κουμπί από τους άλλους χρήστες.

Θεωρητικά επίσης υπάρχει η πιθανότητα ο μετρητής να μηδενιστεί οποιαδήποτε στιγμή επειδή εάν ο Application Server αποφασίσει να καταστρέψει το αντικείμενο του Servlet και να το ξαναφτιάξει θα καταστραφεί και το bean (CounterServiceBean) και θα αρχικοποιηθεί από την αρχή στο καινούριο Servlet instance. Πρακτικά όμως κάτι τέτοιο το πιο πιθανό είναι να μην γίνει ποτέ.

## *Lifecycle, Scope and Concurrency Experiments*

Στην σελίδα των public services στην παράγραφο **Lifecycle, Scope and Concurrency Experiments** που αναφέρθηκε και στο κεφάλαιο για τα Singleton Beans μπορούμε να πειραματιστούμε με το case **“First Stateful Bean with Dependent Scope”** για να κατανοήσουμε καλύτερα τον τρόπο συμπεριφοράς των Stateful Beans στις παράλληλες κλήσεις. Εάν π.χ. πατήσουμε αρκετές φορές το κουμπί “Execute” με το κάθε πάτημα να γίνεται ανά ένα δευτερόλεπτο τότε το θα παρατηρήσουμε ότι ο χρόνος που ξεκινάει η διαδικασία αλλάζει ανά τρία ή παραπάνω δευτερόλεπτα, αυτό δείχνει ότι δεν υπάρχει παράλληλη επεξεργασία των αιτήσεων αλλά μπλοκάρισμά τους και η επεξεργασία της κάθε αίτησης ξεκινάει όταν τελειώνει η προηγούμενη. Επίσης θα παρατηρήσουμε ότι το “processing object hash code” δεν αλλάζει σε κάθε επεξεργασία ακόμα και όταν πατάμε το κουμπί περισσότερες από μία φορές ανά 3 δευτερόλεπτα. Τέλος θα παρατηρήσουμε ότι το internal state παραμένει πάντα στην τιμή “2”.

Εάν τώρα πατήσουμε το “Execute” στο case **“Second Stateful Bean with Dependent Scope”** θα παρατηρήσουμε ένα διαφορετικό **EJB proxy hash code** και **processing object hash code** καθώς και internal state με τιμή “3”. Αυτό δείχνει ότι ένας client μπορεί να έχει πάνω από ένα Stateful Session Bean που είναι εντελώς ανεξάρτητα το ένα με το άλλο.

## Entity Beans

Τα Entity Beans είναι τα components που ορίζει το EJB πρότυπο για την υλοποίηση του **ORM (Object-Relational Mapping)**.

Το ORM είναι ένας τρόπος διασύνδεσης με την ΒΔ που προσπαθεί να αποφύγει εντελώς την χρήση SQL. Αντιθέτως απαιτεί από τον προγραμματιστή να ορίσει μία αντιστοιχία με τις κλάσεις της εφαρμογής του και τους πίνακες της ΒΔ. Δηλαδή να αντιστοιχίσει τα tables της ΒΔ στις κλάσεις της εφαρμογής και τα columns των tables στα fields των κλάσεων.

Η διευκόλυνση που κάτι τέτοιο προσφέρει στον προγραμματιστή είναι προφανής.

Για να επιτύχουμε την υλοποίηση του ORM μέσω των Entity Beans και του JPA προτύπου (Java Persistence API) είναι απαραίτητα 4 πράγματα:

1. Να ορίσουμε ένα **javax.sql.DataSource** αντικείμενο το οποίο θα το διαχειρίζεται ο Application Server.
2. Να ορίσουμε ένα αρχείο **persistence.xml** μέσα στο META-INF κατάλογο της εφαρμογής μας με το κατάλληλο περιεχόμενο το οποίο θα αναφέρει το DataSource, παραμέτρους για το logging των SQL statements, την αυτόματη δημιουργία πινάκων κ.α..
3. Να ορίσουμε το **DAO Layer (Data Access Objects Layer)**. Δηλαδή να εφαρμόσουμε τα κατάλληλα **annotations** του **javax.persistence package** στις κλάσεις της εφαρμογής μας για να ορίσουμε σε ποια tables θα αντιστοιχούν, καθώς και σε ποια columns αυτών των tables θα αντιστοιχούν τα fields τους.
4. Να ορίσουμε ως dependency ένα **javax.persistence.EntityManager** αντικείμενο στις κλάσεις οι οποίες θα πραγματοποιήσουν πρόσβαση στην ΒΔ μέσω ORM και στην συνέχεια να το χρησιμοποιήσουμε για την διασύνδεση των αντικειμένων του DAO Layer με την ΒΔ.

### Δημιουργία του DataSource Αντικειμένου

Το **DataSource** interface ορίζει μια διασύνδεση για να λαμβάνουμε αντικείμενα τύπου Connection σαν και αυτά που χρησιμοποιήσαμε στο παράδειγμά μας στο JDBC κεφάλαιο. Το JDBC πρότυπο δεν ορίζει κάποια υλοποίηση για αυτήν την διασύνδεση, για αυτόν τον λόγο την υλοποίηση θα πρέπει να την προσφέρουν οι Application Servers.

Παρουσιάζεται ο τρόπος δημιουργίας ενός **DataSource** στον **Weblogic Server 12C** που θα χρησιμοποιηθεί για την διασύνδεση της εφαρμογής **DemoWebApp** με την ΒΔ **WEB\_DEMO**.

Αρχικά μπαίνουμε στην κονσόλα του Weblogic και μεταβαίνουμε στο **Services -> Data Sources**.

A JDBC data source is an object bound to the JNDI tree that provides database connectivity through a pool of JDBC connections. Applications can look up a data source on the JNDI tree and then borrow a database connection from a data source.

This page summarizes the JDBC data source objects that have been created in this domain.

**Customize this table**

**Data Sources (Filtered - More Columns Exist)**

Click the **Lock & Edit** button in the Change Center to activate all the buttons on this page.

New	Delete	Name	Type	JNDI Name	Targets
Showing 0 to 0 of 0 Previous   Next					
There are no items to display					
New	Delete				
Showing 0 to 0 of 0 Previous   Next					

Στην συνέχεια πατάμε το κουμπί Lock & Edit για να ξεκινήσουμε τις αλλαγές που θέλουμε.

The following properties will be used to identify your new JDBC data source.

\* Indicates required fields

What would you like to name your new JDBC data source?

**\* Name:** WEB\_DEMO

What JNDI name would you like to assign to your new JDBC Data Source?

**JNDI Name:** WEB\_DEMO

What database type would you like to select?

**Database Type:** Oracle

Ορίζουμε το όνομα του Data Source (**Name**) να είναι ίδιο με το όνομα της ΒΔ στον Oracle Database Server 11G που χρησιμοποιούμε. Επίσης ορίζουμε το ίδιο όνομα και στο **JNDI όνομα** του DataSource με βάση το οποίο θα είναι προσβάσιμο στην εφαρμογή μας. Κανένα από αυτά τα ονόματα βέβαια δεν είναι υποχρεωτικό να είναι ίδια.

Ορίζουμε και το **Database Type** σε Oracle το οποίο στην συνέχεια θα έχει ως αποτέλεσμα να γίνουν δυνατές οι επιλογές μεταξύ διαφόρων JDBC Drivers για τις ΒΔ της Oracle.

**O Weblogic Application Server 12C έχει ήδη προεγκατεστημένους JDBC Drivers για τις πιο γνωστές ΒΔ για αυτό και μας δίνεται τέτοια επιλογή στην κονσόλα διαχείρισής του.**

**Change Center**

**View changes and restarts**

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

[Lock & Edit](#)

[Release Configuration](#)

**Domain Structure**

- demoWebApp
  - Environment
  - Deployments
  - Services
    - + Messaging
    - Data Sources
    - Persistent Stores
    - Foreign JNDI Providers
    - Work Contexts
    - XML Registries
    - XML Entity Caches
    - jCOM
    - Mail Sessions
    - File T3

[How do I...](#)

**Create a New JDBC Data Source**

[Back](#) [Next](#) [Finish](#) [Cancel](#)

**JDBC Data Source Properties**

The following properties will be used to identify your new JDBC data source.

**Database Type:** Oracle

What database driver would you like to use to create database connections? Note: \* indicates that the driver is explicitly supported.

**Database Driver:** \*Oracle's Driver (Thin XA) for Instance connections; Versions:Any

[Back](#) [Next](#) [Finish](#) [Cancel](#)

Εδώ διαλέγουμε τον συγκεκριμένο JDBC Driver για την ΒΔ μας. Επιλέγουμε τον **“Oracle’s Driver (Thin XA) for Instance connections; Versions:Any”**. Επιλέξαμε τον συγκεκριμένο Driver (τύπου XA) επειδή υποστηρίζει **global (distributed) transactions** τα οποία θα τα εξετάσουμε σε επόμενο κεφάλαιο.

**Change Center**

**View changes and restarts**

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

[Lock & Edit](#)

[Release Configuration](#)

**Domain Structure**

- demoWebApp
  - Environment
  - Deployments
  - Services
    - + Messaging

**Create a New JDBC Data Source**

[Back](#) [Next](#) [Finish](#) [Cancel](#)

**Transaction Options**

You have selected an XA JDBC driver to use to create database connection in your new data source. The data source will support global transactions and use the 'Two-Phase Commit' global transaction protocol. No other transaction configuration options are available.

[Back](#) [Next](#) [Finish](#) [Cancel](#)

Configuration button to allow others to edit the domain.

Lock & Edit

Release Configuration

**Domain Structure**

- demoWebApp
  - Environment
  - Deployments
  - Services
    - Messaging
    - Data Sources
    - Persistent Stores
    - Foreign JNDI Providers
    - Work Contexts
    - XML Registries
    - XML Entity Caches
    - jCOM
    - Mail Sessions
    - File T3

**How do I...**

- Create JDBC generic data sources
- Create LLR-enabled JDBC data sources

**System Status**

Health of Running Servers

Failed (0)
Critical (0)
Overloaded (0)
Warning (0)
OK (1)

Back | Next | Finish | Cancel

**Connection Properties**

Define Connection Properties.

What is the name of the database you would like to connect to?

**Database Name:** XE

What is the name or IP address of the database server?

**Host Name:** localhost

What is the port on the database server used to connect to the database?

**Port:** 1521

What database account user name do you want to use to create database connections?

**Database User Name:** WEB\_DEMO

What is the database account password to use to create database connections?

**Password:** \*\*\*\*\*

**Confirm Password:** \*\*\*\*\*

Additional Connection Properties:

**oracle.jdbc.DRCPConnectionClass:**

Back | Next | Finish | Cancel

Σε αυτήν την οθόνη δηλώνονται οι **παράμετροι για την σύνδεση με την ΒΔ**. Στην περίπτωση της Oracle ένα σύνολο από πίνακες, constraints, indexes, procedures κ.τ.λ. ανήκουν σε έναν **Database User** για αυτό και η έννοια του Database είναι διαφορετική απ' ότι στα άλλα Συστήματα Διαχείρισης Βάσεων Δεδομένων. Όταν στο κείμενο αναφερόμαστε σε ΒΔ εννοούμε στην πραγματικότητα τον **Database User**. Στην περίπτωσή μας, με βάση το configuration που έγινε στην Oracle 11G Database, το **“Database Name”** είναι το **“XE”** και το **“Database User Name”** είναι το **“WEB\_DEMO”**. Παρατηρούμε ότι δηλώνουμε τον **host** και το **port** στο οποίο τρέχει το service της ΒΔ καθώς και το **Password** που αντιστοιχεί στον Database User.

View changes and restarts

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

Lock & Edit

Release Configuration

**Domain Structure**

- demoWebApp
  - Environment
  - Deployments
  - Services
    - Messaging
      - Data Sources
      - Persistent Stores
      - Foreign JNDI Providers
      - Work Contexts
      - XML Registries
      - XML Entity Caches
      - jCOM
      - Mail Sessions
      - File T3

**How do I...**

- Create JDBC generic data sources
- Create LLR-enabled JDBC data sources

**System Status**

Health of Running Servers

Failed (0)
Critical (0)

Home > Summary of JDBC Data Sources

Create a New JDBC Data Source

Test Configuration | Back | Next | Finish | Cancel

**Test Database Connection**

Test the database availability and the connection properties you provided.

What is the full package name of JDBC driver class used to create database connections in the connection pool?

(Note that this driver class must be in the classpath of any server to which it is deployed.)

**Driver Class Name:** oracle.jdbc.xa.client.Oracle;

What is the URL of the database to connect to? The format of the URL varies by JDBC driver.

**URL:** jdbc:oracle:thin:@localhost:1521/XE

What database account user name do you want to use to create database connections?

**Database User Name:** WEB\_DEMO

What is the database account password to use to create database connections?

(Note: for secure password management, enter the password in the Password field instead of the Properties field below)

**Password:** \*\*\*\*\*

**Confirm Password:** \*\*\*\*\*

What are the properties to pass to the JDBC driver when creating database connections?

Back | Next | Finish | Cancel

The set of driver properties whose values are derived at runtime from the named system property.

**System Properties:**

What table name or SQL statement would you like to use to test database connections?

**Test Table Name:**

SQL ISVALID

Test Configuration | Back | Next | Finish | Cancel

Σε αυτήν την οθόνη εμφανίζονται συγκεντρωτικά όλες οι παράμετροι που έχουμε εισάγει καθώς και το JDBC URL προς την βάση που έχει την μορφή που ορίζεται από την Oracle 11G.

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

Lock & Edit

Release Configuration

**Domain Structure**

- demoWebApp
  - + Environment
  - Deployments
  - + Services
    - + Messaging
    - Data Sources
    - Persistent Stores
    - Foreign JNDI Providers

Home >Summary of JDBC Data Sources

Create a New JDBC Data Source

Back | Next | Finish | Cancel

**Select Targets**

You can select one or more targets to deploy your new JDBC data source. If you don't select a target, the deployment will be available later time.

Servers

AdminServer

Back | Next | Finish | Cancel

Σε αυτήν την οθόνη επιλέγουμε τους **Servers** στους οποίους θέλουμε να είναι διαθέσιμο το Data Source.

ORACLE® WebLogic Server Administration Console 12c

Change Center

**View changes and restarts**

Pending changes exist. They must be activated to take effect.

Activate Changes

Undo All Changes

**Domain Structure**

- demoWebApp
  - + Environment
  - Deployments
  - + Services
    - + Messaging
    - Data Sources
    - Persistent Stores
    - Foreign JNDI Providers

Home >Summary of JDBC Data Sources

Summary of JDBC Data Sources

Configuration Monitoring

A JDBC data source is an object bound to the JNDI tree that provides database connectivity through a pool of JDBC connections. Applications can look up a data source on the JNDI tree and then borrow a database connection from a data source.

This page summarizes the JDBC data source objects that have been created in this domain.

Customize this table

**Data Sources (Filtered - More Columns Exist)**

New	Delete	Name	Type	JNDI Name	Targets
New	Delete	WEB_DEMO	Generic	WEB_DEMO	AdminServer

Showing 1 to 1 of 1 Previous | Next

To Data Source δημιουργήθηκε. Μένει να πατήσουμε το κουμπί “**Activate Changes**” για να ενεργοποιήσουμε τις αλλαγές.

## Ορισμός του persistence.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence.xsd">
5
6<persistence-unit name="web_demo_pu" transaction-type="JTA">
7   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
8   <jta-data-source>WEB_DEMO</jta-data-source>
9
10  <!-- Classes that belong to this persistence unit. -->
11  <class>osotnikov.demowebapp.services.user_management.vo.User</class>
12  <class>osotnikov.demowebapp.services.fibonacci.entity.FibCompBill</class>
13  <!--<class>osotnikov.demowebapp.services.fibonacci.entity.FibCompHistory</class>-->
14
15  <exclude-unlisted-classes>true</exclude-unlisted-classes>
16
17<properties>
18  <!-- DO NOT DECLARE THESE PROPERTIES AS THEY ARE ALREADY DECLARED IN THE DATASOURCE CONFIGURATION
19  <property name="eclipselink.target-database" value="Oracle11i"/>
20  <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.xa.client.OracleXADataSource"/>
21  <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:XE"/>
22  <property name="javax.persistence.jdbc.user" value="WEB_DEMO"/>
23  <property name="javax.persistence.jdbc.password" value="WEB_DEMO"/>WILL NOT WORK IF YOU DECLARE THEM HERE TOO. -->
24
25  <property name="javax.persistence.jtaDataSource" value="WEB_DEMO" />
26
27  <!-- Automatic Table Creation from Entities at Startup -->
28  <property name="eclipselink.ddl-generation" value="create-tables"/>
29  <property name="eclipselink.ddl-generation.output-mode" value="both"/>
30  <property name="eclipselink.deploy-on-startup" value="true"/>
31
32  <!-- logging -->
33  <!-- log JPA Statements -->
34  <property name="eclipselink.logging.level" value="FINE"/>
35  <!-- also log of the values of the parameters used for the query -->
36  <property name="eclipselink.logging.parameters" value="true"/>
37
38</properties>
39
40</persistence-unit>
41
42<persistence-unit name="web_demo_crm_pu" transaction-type="JTA">
43   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
44   <!-- see step 2 below -->
45   <jta-data-source>WEB_DEMO_CRM</jta-data-source>
46
47  <!-- Classes that belong to this persistence unit. -->
48  <class>osotnikov.demowebapp.services.fibonacci.entity.FibCompCrmLog</class>
49
50  <exclude-unlisted-classes>true</exclude-unlisted-classes>
51
52<properties>
53
54  <property name="javax.persistence.jtaDataSource" value="WEB_DEMO_CRM" />
55
56  <!-- Automatic Table Creation from Entities at Startup -->
57  <property name="eclipselink.ddl-generation" value="create-tables"/>
58  <property name="eclipselink.ddl-generation.output-mode" value="both"/>
59  <property name="eclipselink.deploy-on-startup" value="true"/>
60
61  <!-- logging -->
62  <!-- log JPA Statements -->
63  <property name="eclipselink.logging.level" value="FINE"/>
64  <!-- also log of the values of the parameters used for the query -->
65  <property name="eclipselink.logging.parameters" value="true"/>
66
67</properties>
68</persistence-unit>
69
70</persistence>
```

Κατά τον ορισμό του persistence.xml το σημαντικότερο είναι, μέσα στα elements τύπου **<persistence-unit>**, να ορίσουμε το element **<jta-data-source>** όπως και το αντίστοιχο property (**javax.persistence.jtaDataSource**) στο element **<properties>**, στο όνομα ενός DataSource που έχουμε φτιάξει και έχουμε κάνει deploy στον Application Server.

Στο attribute “name” ενός “**persistence-unit**” element θα πρέπει να δώσουμε ένα μοναδικό όνομα στο οποίο θα χρειαστεί να αναφερθούμε αργότερα στις κλάσεις του κώδικα που θα

υλοποιούν το DAO Layer για να δηλώσουμε ουσιαστικά ότι θα χρησιμοποιούν τις ρυθμίσεις που ορίζονται στο συγκεκριμένο persistence unit.

Επίσης πρέπει να ορίσουμε τις κλάσεις οι οποίες ανήκουν σε κάθε persistence unit. Όπως βλέπουμε ορίσαμε τις κλάσεις User και FibCompBill στο persistence unit “**web\_demo\_pu**”, ενώ την FibCompCrmLog στο “**web\_demo\_crm\_pu**”. Τα δύο αυτά persistence units αντιστοιχούν σε διαφορετικές βάσεις δεδομένων.

Τα properties **eclipselink.ddl-generation** και **eclipselink.ddl-generation.output-mode** έχουν να κάνουν, με την αυτόματη παραγωγή του script με τα DDL Statements που θα δημιουργήσουν τους πίνακες της βάσης για τις κλάσεις που θα ορίσουμε στο πρόγραμμά μας, καθώς και την αυτόματη εκτέλεση αυτών των scripts. Σημειώνεται ότι αυτά τα properties δεν λειτουργούν σωστά σε πολλά ΣΔΒΔ, πάντως λειτούργησαν χωρίς πρόβλημα στις ΒΔ Oracle 11G και MySQL που χρησιμοποιήθηκαν στα πλαίσια αυτής της εργασίας.

Τα properties **eclipselink.logging.level** και **eclipselink.logging.parameters** έχουν να κάνουν με το logging των SQL Statements που εκτελούνται εξαιτίας των ενεργειών της εφαρμογής πάνω στις κλάσεις που αντιστοιχούν στα tables της ΒΔ.

## Ορισμός του DAO Layer

Το **DAO Layer** ορίζεται εφαρμόζοντας τα κατάλληλα annotations στις κλάσεις που μοντελοποιούν την δομή της ΒΔ στην οποία συνδεόμαστε. Το παράδειγμα που θα αναφέρουμε προκύπτει από τις διαδικασίες του registration και του log in. Έχουμε ήδη παρουσιάσει το **UserManagementService** στο οποίο κάναμε χρήση των κλάσεων **User** και του enum **GroupEnum** όμως δεν εξηγήσαμε τον τρόπο με τον οποίον αντικείμενα αυτών των τύπων δημιουργούσαν τις αντίστοιχες εγγραφές στην ΒΔ.

**Αρχικά παρουσιάζουμε την κλάση User:**

```
1 package osotnikov.demowebapp.services.user_management.vo;
2
3 import java.io.Serializable;
4
5 @Entity
6 @Table(name="USERS")
7 @Cacheable(false)
8 public class User implements Serializable {
9
10     @Id
11     @Column(unique=true, nullable=false)
12     private String email;
13
14     @Column(nullable=false)
15     private String firstName;
16
17     @Column(nullable=false)
18     private String lastName;
19
20     /**
21      * A sha512 is 512 bits long -- as its name indicates. If you are using an hexadecimal representation,
22      * each digit codes for 4 bits ; so you need 512 : 4 = 128 characters to represent 512 bits -- plus you
23      * need another 7 for the "{SHA-1}" prefix, thus you need a varchar(135), or a char(135), as the
24      * length is always the same, not varying at all.
25      */
26     @Column(nullable=false, length=135) //sha-512 + hex
27     private String password;
28
29     // @Transient means non persistent.
30     @Transient
31     private String confirmPasswordValue;
32
33     @Temporal(javax.persistence.TemporalType.TIMESTAMP)
34     @Column(nullable=false)
35     private Date registeredOn;
```

```

52@ElementCollection(targetClass = GroupEnum.class)
53@CollectionTable(name = "USERS_GROUPS",
54joinColumns = @JoinColumn(name = "email", nullable=false),
55uniqueConstraints = { @UniqueConstraint(columnNames={"email","groupname"}) } )
56@Enumerated(EnumType.STRING)
57@Column(name="groupname", length=64, nullable=false)
58private List<GroupEnum> groups;
59
60public User() {
61}
62
63
64public String getFirstName() {
65    return firstName;
66}
67
68public void setFirstName(String firstName) {
69    this.firstName = firstName;
70}
71
72public String getLastName() {
73    return lastName;
74}
75
76public void setLastName(String lastName) {
77    this.lastName = lastName;
78}
79
80public String getEmail() {
81    return email;
82}
83
84public void setEmail(String email) {
85    this.email = email;
86}
87
88/**
89 * @return the password in SHA512 HEX representation
90 */
91public String getPassword() {
92    return password;
93}
94
95public void setPassword(String password) {
96    this.password = password;
97}
98
99public Date getRegisteredOn() {
100    return registeredOn;
101}
102
103public void setRegisteredOn(Date registeredOn) {
104    this.registeredOn = registeredOn;
105}
106
107public List<GroupEnum> getGroups() {
108    return groups;
109}
110
111public void setGroups(List<GroupEnum> groups) {
112    this.groups = groups;
113}
114
115public String getConfirmPasswordValue() {
116    return confirmPasswordValue;
117}
118
119public void setConfirmPasswordValue(String confirmPasswordValue) {
120    this.confirmPasswordValue = confirmPasswordValue;
121}
122
123@Override
124
125public String toString() {
126    return "User [email=" + email + ", firstName=" + firstName
127        + ", lastName=" + lastName + ", password=" + password
128        + ", confirmPasswordValue=" + confirmPasswordValue
129        + ", registeredOn=" + registeredOn + ", groups=" + groups + "]";
130}
131
132}

```

## Παρουσιάζουμε τον αντίστοιχο πίνακα στην ΒΔ:

Column Name	ID	PK	Index Pos	Null?	Data Type	Default	Histogram	Num Distinct	Num Nulls	Density	Encryption Alg	Salt	Seq/Trigger	Virtual
EMAIL	1			N	VARCHAR2 (1024 Char)	None		3	0	0,33333	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
FIRSTNAME	2			N	VARCHAR2 (1024 Char)	None		3	0	0,33333	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
LASTNAME	3			N	VARCHAR2 (1024 Char)	None		1	0	1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
PASSWORD	4			N	VARCHAR2 (1024 Char)	None		2	0	0,5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
REGISTEREDON	5			N	TIMESTAMP(6)	None		3	0	0,33333	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
DESCRIPTION	6			Y	VARCHAR2 (1000 Char)	None		0	3	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Για να αντιστοιχίσουμε μία κλάση σε έναν πίνακα είναι απαραίτητο να ορίσουμε πάνω της το annotation “**@Entity**”. Εάν συν τοις άλλοις θέλουμε η κλάση μας να έχει διαφορετικό όνομα από τον πίνακα (η σύγκριση δεν είναι case sensitive) πρέπει να ορίσουμε επιπλέον το annotation “**@Table**” με το attribute **name** να έχει την τιμή του πίνακα στον οποίον αντιστοιχεί η κλάση.

Τα fields δεν χρειάζεται υποχρεωτικά να έχουν το annotation “**@Column**” αλλά μόνο στην περίπτωση που θέλουμε να αντιστοιχίσουμε κάποιο field σε column με διαφορετικό όνομα ή να ορίσουμε attributes όπως nullable ή length.

To field που αντιστοιχεί στο column του primary key θα πρέπει να έχει το annotation “**@Id**”.

Τα fields τα οποία δεν θέλουμε να αντιστοιχίζονται σε κάποιο column του πίνακα της ΒΔ αλλά να αποθηκεύουν προσωρινές τιμές όπως τα fields ενός κανονικού αντικειμένου της Java θα πρέπει να τα ορίσουμε με το annotation “**@Transient**”.

Τα fields τα οποία θέλουμε να αποθηκεύουν τιμές που αντιπροσωπεύουν ημ/νίες και ώρα θα πρέπει να ορίζονται με το annotation “**@Temporal**” με το **TemporalType** να ορίζει τον τύπο της ΒΔ που ορίζεται για το αντίστοιχο column.

Εάν υπάρχει μία ένα-προς-πολλά ή μία πολλά-προς-πολλά σχέση μεταξύ δύο πινάκων και σε ένα field αντιστοιχούν πολλές τιμές του άλλου πίνακα τότε το field αυτό θα πρέπει να έχει τύπο κάποιο Collection. Στην περίπτωσή μας το field αυτό είναι το “**groups**” το οποίο έχει τύπο **List<GroupEnum>**. To field αυτό αποθηκεύει τα groups στα οποία ανήκει ο χρήστης.

Πάνω σε ένα τέτοιο field θα πρέπει να ορίζεται το annotation “**@ElementCollection**” με το **targetClass** attribute να δηλώνει τον τύπο των στοιχείων του Collection (στην περίπτωσή μας το GroupEnum). Επίσης θα πρέπει να ορίζεται το annotation “**@CollectionTable**”. Αυτό θα ορίσει τα “**@JoinColumns**” που είναι τα columns που σχηματίζουν το foreign key του άλλου πίνακα το οποίο δείχνει προς το primary key του πίνακα της κλάσης μας. Επίσης στο ίδιο annotation μπορούν να οριστούν τα constraints τα οποία θέλουμε να εφαρμόζονται πάνω στην σχέση μεταξύ των δύο πινάκων.

Θα παρατηρήσουμε επίσης ότι στο field “**groups**” εφαρμόζεται και το annotation “**@Enumerated**” αυτό το annotation θα πρέπει να εφαρμόζεται σε κάθε field του οποίου ο τύπος είναι ένα **enum** ή κάποιο Collection που εσωτερικά αποθηκεύει αντικείμενα με τύπο κάποιο enum. Το “**@Enumerated**” annotation ορίζει επίσης και σε τι μορφή θα αποθηκεύονται οι τιμές μέσα στο αντίστοιχο column του πίνακα. Το **EnumType.STRING** ορίζει ότι θα αποθηκεύονται σαν συμβολοσειρά με τιμή το όνομα του enum value ενώ το **EnumType.ORDINAL** αντί για το όνομα του value θα αποθηκεύει τον ακέραιο που του αντιστοιχεί (τον ίδιο ακέραιο που επιστρέφει και η μέθοδος “**ordinal**” των enums που ορίζει την θέση του enum value στο enum declaration).

**Παρουσιάζουμε τώρα το enum GroupEnum:**

```
1 package osotnikov.demowebapp.services.user_management.vo;
2
3 public enum GroupEnum {
4
5     ADMINISTRATOR("ADMINISTRATOR"),
6     USER("USER"),
7     DEFAULT("DEFAULT");
8
9     private String name;
10
11     private GroupEnum(String name) {
12         this.name = name;
13     }
14
15     public String getName() {
16         return name;
17     }
18 }
19 }
```

Όπως παρατηρούμε δεν ορίζεται κανένα annotation πάνω σε αυτό. Αυτό γίνεται επειδή η μοντελοποίηση έχει γίνει ήδη στο field “groups” της κλάσης “User”.

**Παρουσιάζουμε τώρα τον πίνακα της ΒΔ που αντιστοιχεί στο enum GroupEnum:**

The screenshot shows the Oracle SQL Developer interface. On the left, there's a tree view with nodes like 'Img', 'Table', 'JDBC\_DEMO\_ACCESS\_INFO', 'USERS', and 'USERS\_GROUPS'. On the right, there's a 'Columns' tab showing the structure of the 'USERS\_GROUPS' table. The columns are: Column Name (EMAIL, GROUPNAME, DESCRIPTION), ID (1, 2, 3), PK (Yes for ID), Index Pos (N, N, Y), Data Type (VARCHAR2 (1024 Byte), VARCHAR2 (1024 Char), VARCHAR2 (1000 Char)), Default (Frequency, Frequency, None), Histogram (Frequency, Frequency, None), Num Distinct (3, 3, 0), Num Nulls (0, 0, 9), Density (0,05556, 0,05556, 0), Encryption Alg (checkboxes), Salt (checkboxes), Seq/Trigger (checkboxes), and Virtual (checkboxes). The 'GROUPNAME' column is highlighted.

Column Name	ID	PK	Index Pos	Null?	Data Type	Default	Histogram	Num Distinct	Num Nulls	Density	Encryption Alg	Salt	Seq/Trigger	Virtual
EMAIL	1			N	VARCHAR2 (1024 Byte)	Frequency	Frequency	3	0	0,05556	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
GROUPNAME	2			N	VARCHAR2 (1024 Char)	Frequency	Frequency	3	0	0,05556	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DESCRIPTION	3			Y	VARCHAR2 (1000 Char)	None	None	0	9	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

[Χρήση του EntityManager Αντικειμένου για την Διασύνδεση των Αντικειμένων](#)

[του DAO Layer με την ΒΔ](#)

Ο κώδικας του UserManagementService έχει ήδη παρουσιαστεί σε προηγούμενο κεφάλαιο. Εδώ θα γίνει επεξήγηση του ρόλου που παίζει το **EntityManager** αντικείμενο στην διασύνδεση των αντικειμένων των τύπων User και GroupEnum με τις εγγραφές στους πίνακες της ΒΔ.

Η μέθοδος **loginAndFetchUser** χρησιμοποιείται κατά το login των χρηστών έτσι ώστε να τους κάνει login στο authentication scheme που έχουμε ορίσει να προστατεύει ορισμένες σελίδες από μη εξουσιοδοτημένη πρόσβαση. Η μέθοδος αυτή, σε περίπτωση ενός επιτυχημένου log in επιστρέφει το αντικείμενο User με δεδομένα από την αντίστοιχη εγγραφή του χρήστη ο οποίος μόλις πραγματοποίησε το login.

Η ανάγνωση του User αντικειμένου γίνεται στην μέθοδο **fetchUser** κάνοντας χρήση της μεθόδου **find** του **EntityManager** η οποία παίρνει ως παράμετρο τον τύπο του αντικειμένου και την τιμή του primary key.

Το αντικείμενο που επιστρέφεται κατά αυτόν τον τρόπο λέμε ότι είναι **“managed”**. Στην ορολογία του JPA αυτό σημαίνει ότι οποιεσδήποτε αλλαγές γίνονται σε αυτό θα αντικατοπτρίζονται στην αντίστοιχη εγγραφή στην βάση.

Σε γενικές γραμμές εάν απλώς φτιάξουμε ένα αντικείμενο τύπου User π.χ. με το new operator της Java αυτό θα είναι **“detached”** και όχι **“managed”** που σημαίνει ότι θα είναι απλώς ένα POJO και οι αλλαγές που γίνονται σε αυτό δεν θα αντικατοπτρίζονται σε κάποια εγγραφή της ΒΔ.

**Ένα αντικείμενο μπορεί να γίνει “managed” μόνο μέσω των μεθόδων του EntityManager.**

Για να αποθηκεύσουμε ένα αντικείμενο τύπου User στην ΒΔ ως εγγραφή θα πρέπει να κάνουμε χρήση της μεθόδου **persist**. Αυτό συμβαίνει στην μέθοδο validateTweakAndRegisterUser η οποία κατά το επιτυχημένο registration αποθηκεύει το User αντικείμενο στον πίνακα USERS καθώς επίσης και τις αντίστοιχες εγγραφές για τα groups στον πίνακα USERS\_GROUPS. Αυτά αποθηκεύονται με βάση τα στοιχεία του Collection στο field groups του User αντικειμένου.

Σε αντίθεση με την JDBC δεν χρειάζεται να ασχολούμαστε με την αποδέσμευση διαφόρων resources όπως π.χ. τα Connection και ResultSet. **Η αποδέσμευση των resources που συμμετείχαν στην επικοινωνία με την βάση καθώς και το commit των αλλαγών γίνεται αυτόματα με το πέρας του transaction μεσα στο οποίο χρησιμοποιήθηκε το EntityManager.**

Με τα transactions θα σχοληθούμε σε επόμενο κεφάλαιο. Εδώ απλώς θα αναφέρουμε επιπρόσθετα ότι **μέσα στα πλαίσια του ίδιου transaction θα προσφέρεται πάντα το ίδιο instance του EntityManager από το proxy του στο field στο οποίο το κάναμε inject με το @PersistenceContext annotation.**

# Message Driven Beans (MDB) & JMS

## Λειτουργική Περιγραφή του Παραδείγματος

Έχουμε ήδη κάνει μια εισαγωγή στα MDB στο κεφάλαιο των Container Services. Εδώ οι θεωρητικές έννοιες θα εξετασθούν μέσω ενός παραδείγματος υπολογισμού αριθμών της fibonacci ακολουθίας.

Η λειτουργικότητα αυτή θα είναι διαθέσιμη στους logged in χρήστες του DemoWebApp από την σελίδα “**private\_services.jsp**” στην παράγραφο “**Compute n-th fibonacci number (term index)**”.

Στην επόμενη εικόνα βλέπουμε το μήνυμα που εμφανίζεται όταν υποβάλουμε αίτηση για ασύγχρονο υπολογισμό του 5<sup>ου</sup> σε σειρά αριθμού στην ακολουθία fibonacci. Επειδή αυτός ο αριθμός δεν είχε υπολογιστεί στο παρελθόν από κανέναν χρήστη του DemoWebApp υποβλήθηκε για επεξεργασία σε ένα **Message Driven Bean (MDB)**. Κάθε φορά που τελειώνει ένας υπολογισμός καταχωρείτε σε ειδική δομή του DemoWebApp οπότε τα επόμενα αιτήματα για τον ίδιο υπολογισμό θα επιστρέφουν κατευθείαν το αποτέλεσμα. Στην περίπτωσή μας όμως θα πρέπει να κάνουμε refresh της σελίδας για να εμφανιστεί το αποτέλεσμα του υπολογισμού.



## Welcome to your privately accessible services!

### Background Color Selector

Select the background color that you want:  Submit Color & Reload

[Go to the Profile Page \(secured, requires authentication\)](#)

**Compute n-th fibonacci number (term index):**

This is an example of calling a message driven bean as well as an example of global transaction management.

### Output:

Your request is being processed, please visit this page at a later instant to view the result of your computation.

Η προηγούμενη σελίδα μετά από ένα refresh:

# Welcome to your privately accessible services!

## Background Color Selector

Select the background color that you want:  Submit Color & Reload

[Go to the Profile Page \(secured, requires authentication\)](#)

**Compute n-th fibonacci number (term index):**

This is an example of calling a message driven bean as well as an example of global transaction management.

Compute Fibonacci

**Output:**

The result of your requested fibonacci computation with argument 5 is 5.

Επίσης μέσα στην λειτουργικότητα του παραδείγματος είναι και η δημιουργία εγγραφών στην ΒΔ **DEMO\_WEB** καθώς και στην **DEMO\_WEB\_CRM** στους πίνακες **FIB\_COMP\_BILL** και **FIB\_COMP\_CRM\_LOG** αντίστοιχα. Και οι δύο πίνακες αποθηκεύουν το αίτημα του χρήστη καταγράφοντας το username του καθώς και τον χρόνο που απαιτήθηκε για τον ζητούμενο υπολογισμό.

Ο πίνακας **FIB\_COMP\_BILL** έχει σχεδιαστεί για την προσομοίωση του υποθετικού σεναρίου της χρέωσης των χρηστών για τους υπολογισμούς που αιτήθηκαν με βάση των χρόνο που αυτοί κατανάλωσαν (η χρέωση αποθηκεύεται στο column “**COST**”). Θα μπορούσαμε να φανταστούμε ότι χρησιμοποιείται από ένα **Τμήμα Λογιστηρίου** στο τέλος κάθε μήνα για να αθροίσει όλες τις οφειλές ενός πελάτη και να χρεώσει αντίστοιχα την πιστωτική του κάρτα.

Ο πίνακας **FIB\_COMP\_CRM\_LOG** σχεδιάστηκε με την υπόθεση ότι η εταιρεία της εφαρμογής DemoWebApp διαθέτει και **Τμήμα Υποστήριξης Πελατών** το οποίο έχει δικιά του MySQL ΒΔ. Στον πίνακα αυτόν αποθηκεύονται πάλι οι ίδιες πληροφορίες όπως και στον **FIB\_COMP\_BILL** αλλά αυτήν την φορά μπορούμε να υποθέσουμε ότι εξυπηρετεί τους υπαλλήλους που αναλαμβάνουν την τηλεφωνική υποστήριξη.

Οι καταγραφές στους δύο αυτούς πίνακες γίνονται περισσότερο για να αναλύσουμε τα Transactions και τα rollbacks σε επόμενο κεφάλαιο.

## [Δημιουργία JMS Resources](#)

Πριν παρουσιάσουμε τον κώδικα του παραδείγματος θα αναλύσουμε την δημιουργία των απαραίτητων **JMS Resources** που θα πρέπει να είναι deployed στους servers που θα τον τρέξουν. Αυτά είναι ένας **JMS Server**, ένα **JMS Connection Factory** και ένα **JMS Queue**. Επίσης θα χρειαστούμε ένα **Persistent Store** πριν φτιάξουμε τον JMS Server. Πριν φτιάξουμε το JMS Queue και JMS Connection Factory θα χρειαστούμε ένα **JMS Module** και επιπρόσθετα ένα **JMS Subdeployment** πριν φτιάξουμε το JMS Queue.

## Διαδικασία Δημιουργίας ενός Persistent Store

Ένα “**Persistent Store**” δεν δημιουργείτε μόνο για την χρήση του από JMS Resources αλλά εμείς θα το δημιουργήσουμε για την αποθήκευση των JMS Messages που δεν έχουν σταλθεί ακόμα έτσι ώστε στην περίπτωση που σταματήσει να λειτουργεί ο Server να μπορούν να σταλθούν όταν τεθεί εκ νέου σε λειτουργία.

Μεταβαίνουμε στο **Services -> Persistent Stores** και πατάμε “**New**”.

The screenshot shows the Oracle WebLogic Server Administration Console interface. The title bar indicates "Oracle WebLogic Server 1" and "Summary of Persistent St...". The URL in the address bar is "localhost:7001/console/console.portal?\_nfpb=true&\_pageLabel=JmsStoresJMSStoreTablePage". The main content area is titled "Summary of Persistent Stores". It contains a brief description: "A persistent store is a physical repository for storing subsystem data, such as persistent JMS messages. It can be a JDBC-accessible database, disk-based file, or replicated memory storage. This page summarizes the persistent stores that have been created for this domain." Below this is a section titled "Customize this table" with a "Persistent Stores" table. The table has columns "Name", "Type", and "Target". It lists three entries:

Name	Type	Target
WseeFileStore	FileStore	AdminServer
WseeJaxwsFileStore	FileStore	AdminServer
WseeSoapjmsFileStore	FileStore	AdminServer

Επιλέγουμε το “Create FileStore”.

The screenshot shows the Oracle WebLogic Server Administration Console interface. The left sidebar shows the domain structure with "Persistent Stores" selected. The main panel is titled "Persistent Stores" and shows a table of existing persistent stores. The table has a header row with "Name" and a sorting arrow. Below are three rows: "WseeFileStore", "WseeJaxwsFileStore", and "WseeSoapjmsFileStore". A context menu is open over the "New" button, listing three options: "Create FileStore", "Create JDBCStore", and "Create ReplicatedStore (Exalogic)".

Εισάγουμε το όνομα του JMS Server στο πεδίο “**Name**” στο πεδίο “**Target**” επιλέγουμε τον Server στον οποίον θα ανήκει.

The screenshot shows the Oracle WebLogic Server Administration Console interface. On the left, there's a 'Domain Structure' tree with nodes like demoWebApp, Environment, Deployments, Services (with Messaging, JMS Servers, etc.), Data Sources, Persistent Stores, Foreign JNDI Providers, and Work Contexts. A 'How do I...' panel at the bottom has a 'Create File Stores' link. The main area is titled 'Create a New File Store' with tabs for 'OK' and 'Cancel'. It contains a 'File Store Properties' section with a note about identifying the new file store. Below that is a field for 'Name' (set to 'FileStore-1'), a 'Target' dropdown (set to 'AdminServer'), and a 'Directory' field which is empty. There are also 'OK' and 'Cancel' buttons at the bottom.

## Διαδικασία Δημιουργίας JMS Server

Μεταβαίνουμε στο **Services -> Messaging -> JMS Servers** και πατάμε “**New**”.

The screenshot shows the 'Summary of JMS Servers' page in the Oracle WebLogic Server Administration Console. The 'Domain Structure' tree on the left shows the same nodes as before. The main area has a 'Messages' section with a note about creating JMS servers. Below it is a 'Summary of JMS Servers' section with a note about management containers. A 'Customize this table' section allows filtering. A table titled 'JMS Servers (Filtered - More Columns Exist)' shows three entries:

Name	Persistent Store	Target	Current Target
WseeJaxwsJmsServer	WseeJaxwsFileStore	AdminServer	AdminServer
WseeJmsServer	WseeFileStore	AdminServer	AdminServer
WseeSoapjmsJmsServer	WseeSoapjmsFileStore	AdminServer	AdminServer

At the bottom of the table are 'New' and 'Delete' buttons. A 'How do I...' panel at the bottom has a 'Configure JMS servers' link.

Εισάγουμε το όνομα του JMS Server στο πεδίο “**Name**” και στο dropdown menu “**Persistent Store**” επιλέγουμε το persistent store που δημιουργήσαμε προηγουμένως.

The screenshot shows the Oracle WebLogic Server Administration Console. On the left, there's a 'Domain Structure' tree with nodes like demoWebApp, Environment, Deployments, Services (which is expanded to show Messaging, JMS Servers, etc.). On the right, the 'Create a New JMS Server' wizard is open. Step 2, 'JMS Server Properties', has the 'Name' field set to 'JMSServer-1' and the 'Persistent Store' dropdown set to 'FileStore-1'. Step 3, 'Select targets', has the 'Target' dropdown set to 'AdminServer'. Navigation buttons 'Back', 'Next', 'Finish', and 'Cancel' are visible at the bottom of each step.

Επιλέγουμε τον Server στον οποίον ο JMS Server θα ανήκει.

This screenshot continues from the previous one, showing the 'Create a New JMS Server' wizard. It's now on Step 3, 'Select targets', where the 'Target' dropdown is set to 'AdminServer'. Navigation buttons 'Back', 'Next', 'Finish', and 'Cancel' are present.

## Διαδικασία Δημιουργίας JMS Module

Μεταβαίνουμε στο Services -> Messaging -> JMS Modules και πατάμε “New”.

The screenshot shows the Oracle WebLogic Server Administration Console. In the top navigation bar, there are tabs for Oracle WebLogic Server 1, Summary of JMS Modules, and YouTube. The URL is localhost:7001/console/console.portal?\_nfpb=true&\_pageLabel=JmsModulesTablePage. The main content area is titled "Summary of JMS Modules". It contains a brief description of what JMS system resources are and how they are managed. Below this is a table titled "JMS Modules" with three entries:

Name	Type
WseeJaxwsJmsModule	System
WseeJmsModule	System
WseeSoapjmsJmsModule	System

At the bottom of the table, there are "New" and "Delete" buttons. The "Domain Structure" sidebar on the left shows the demoWebApp domain with various services like Environment, Deployments, Services (Messaging, JMS Servers, Store-and-Forward Agents, JMS Modules), Path Services, Bridges, Data Sources, Persistent Stores, Foreign JNDI Providers, and Work Contexts. The "How do I..." sidebar provides links for configuring JMS system modules and resources.

Εισάγουμε το όνομα του JMS Module στο πεδίο “Name” και επιλέγουμε την τιμή “Global” στο πεδίο “Scope”. Αφήνουμε τα άλλα δύο πεδία κενά.

The screenshot shows the "Create JMS System Module" wizard. The current step is Step 1. The "Name" field is filled with "SystemModule-0". The "Scope" dropdown is set to "Global". The "Descriptor File Name" and "Location In Domain" fields are empty. The "How do I..." sidebar on the left provides links for configuring JMS system modules and servers. The "System Status" sidebar at the bottom shows the health of running servers as of 7:08 PM, with 0 Failed and 0 Critical servers.

Επιλέγουμε τον Server στον οποίον θα ανήκει το JMS Module.

The screenshot shows the Oracle WebLogic Server Administration Console. The title bar says "Create JMS System Modul". The main content area is titled "Create JMS System Module". Below it, a sub-section titled "Targets:" shows a list of servers with "AdminServer" checked. Navigation buttons "Back", "Next", "Finish", and "Cancel" are at the bottom of the form. On the left, there's a "Domain Structure" tree view under "demoWebApp" containing Environment, Deployments, Services (with Messaging, JMS Servers, etc.), Bridges, Data Sources, Persistent Stores, Foreign JNDI Providers, and Work Contexts. A "Change Center" sidebar on the left indicates no pending changes.

Πατάμε "Finish".

The screenshot shows the Oracle WebLogic Server Administration Console. The title bar says "Create JMS System Modul". The main content area is titled "Create JMS System Module". Below it, a sub-section titled "Add resources to this JMS system module" contains a checkbox labeled "Would you like to add resources to this JMS system module?". Navigation buttons "Back", "Next", "Finish", and "Cancel" are at the bottom of the form. The "Domain Structure" tree view on the left is identical to the previous screenshot. A "Welcome, weblogic" message is visible in the top right corner.

## Διαδικασία Δημιουργίας ενός JMS Connection Factory

Μεταβαίνουμε Services -> Messaging -> JMS Modules -> SystemModule-0 και πατάμε "New".

ORACLE WebLogic Server Administration Console 12c

Change Center

View changes and restarts

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

Lock & Edit

Release Configuration

Domain Structure

- JMS Servers
- Store-and-Forward Agents
- JMS Modules
- Path Services
- Bridges
- Data Sources
- Persistent Stores
- Foreign JNDI Providers
- Work Contexts
- XML Contexts
- XML Registries
- XML Entity Caches
- jCOM
- Mail Sessions
- File T3

How do I...

- Configure quotas for destinations
- Configure JMS templates
- Configure destination keys
- Configure topics

Settings for SystemModule-0

Configuration Subdeployments Targets Security Notes

This page displays general information about a JMS system module and its resources. It also allows you to configure new resources and access existing resources.

Name: SystemModule-0

Scope: Global

Descriptor File Name: jms/systemmodule-0-jms.xml

This page summarizes the JMS resources that have been created for this JMS system module, including queue and topic destinations, connection factories, JMS templates, destination sort keys, destination quota, distributed destinations, foreign servers, and store-and-forward parameters.

Customize this table

Summary of Resources

New	Delete	Name	Type	JNDI Name	Subdeployment	Targets
New	Delete					

Showing 1 to 2 of 2 Previous | Next

Αφού επιλέξουμε "Connection Factory" πατάμε "Next".

Create a New JMS System Module Resource

Back Next Finish Cancel

Choose the type of resource you want to create.

Use these pages to create resources in a JMS system module, such as queues, topics, templates, and connection factories.

Depending on the type of resource you select, you are prompted to enter basic information for creating the resource. For targetable resources, like stand-alone queues and topics, connection factories, distributed queues and topics, foreign servers, and JMS SAF destinations, you can also proceed to targeting pages for selecting appropriate server targets. You can also associate targetable resources with subdeployments, which is an advanced mechanism for grouping JMS module resources and the members to server resources.

Connection Factory

Defines a set of connection configuration parameters that are used to create connections for JMS clients. More Info...

Queue

Defines a point-to-point destination type, which are used for asynchronous peer communications. A message delivered to a queue is distributed to only one consumer. More Info...

Topic

Defines a publish/subscribe destination type, which are used for asynchronous peer communications. A message delivered to a topic is distributed to all topic consumers. More Info...

Distributed Queue

Defines a set of queues that are distributed on multiple JMS servers, but which are accessible as a single, logical queue to JMS clients. More Info...

Distributed Topic

Defines a set of topics that are distributed on multiple JMS servers, but which are accessible as a single, logical topic to JMS clients. More Info...

Foreign Server

Defines foreign messaging providers or remote WebLogic Server instances that are not part of the current domain. More Info...

Στο πεδίο "Name" εισάγουμε το επιθυμητό όνομα για το Connection Factory, στο πεδίο "JNDI Name" εισάγουμε το όνομα με το οποίο θα αναφερθούμε στο Connection Factory στον κώδικά

μας. Τα πεδία “Subscription Sharing Policy” και “Client ID Policy” δεν μας ενδιαφέρουν καθώς θα υλοποιήσουμε μόνο Queue και όχι Topic. Το “Maximum Messages per Session” είναι ένας τρόπος για να αναγκάζουμε τους αποστολείς να περιορίζουν τον αριθμό των μηνυμάτων που στέλνουν σε ένα JMS Session (θα αναλυθεί αργότερα) επειδή η αποστολή πολλών μηνυμάτων στο ίδιο Session δεν ευνοεί το load balancing.

Επιλέγουμε ποιοι Servers θα διαθέτουν το Connection Factory που φτιάξαμε και πατάμε “Finish”.

## Διαδικασία Δημιουργίας ενός Subdeployment

Το “**Subdeployment**” είναι ένα διαχειριστικό εργαλείο του Weblogic Server 12C για την διευκόλυνση του deployment των Queues καθώς επιτρέπει τον ορισμό ενός συνόλου από Targets. Η δημιουργία του είναι απαραίτητη για την δημιουργία ενός Queue.

Services -> Messaging -> JMS Modules -> SystemModule-0 και πατάμε το tab “**Subdeployments**”.  
Πατάμε το “**New**”.

The screenshot shows the Oracle WebLogic Server Administration Console interface. On the left, there's a 'Domain Structure' tree with nodes like demoWebApp, Domain Partitions, Environment, Deployments, Services (which is expanded to show Messaging, JMS Servers, etc.), Data Sources, Persistent Stores, and Foreign JNDI Providers. On the right, under 'Settings for SystemModule-0', the 'Subdeployments' tab is selected. A table titled 'Subdeployments' lists one entry: 'Name' (with a dropdown arrow) and 'Resources'. There are 'New' and 'Delete' buttons at the bottom of the table. Above the table, a note says: 'This page displays subdeployments created for a JMS system module. A subdeployment is a mechanism by which JMS module grouped and targeted to a server resource (such as JMS servers, server instances, or cluster).'

Ορίζουμε το όνομα στο πεδίο “**Subdeployment Name**”.

The screenshot shows the 'Create a New Subdeployment' wizard. Step 1: Subdeployment Properties. It has 'Back', 'Next', 'Finish', and 'Cancel' buttons. A note says: 'The following properties will be used to identify your new subdeployment.' Below it, a field labeled '\* Subdeployment Name:' contains 'JMServer-1\_SD'. The rest of the screen is identical to the previous one, showing the 'Subdeployments' table and the note about subdeployments.

Στα Targets επιλέγουμε μόνο τον JMS Server που φτιάξαμε προηγουμένως και πατάμε “Finish”.

The screenshot shows the Oracle WebLogic Server Administration Console interface. The title bar indicates the session is for Oracle WebLogic Server 12c. The main content area is titled "Create a New Subdeployment". The "Targets" section is active, displaying three categories: "Servers", "JMS Servers", and "SAF Agents". Under "Servers", "AdminServer" is listed. Under "JMS Servers", "JMSServer-1" is checked, while "WseeJaxwsJmsServer", "WseeJmsServer", and "WseeSoapjmsJmsServer" are unselected. Under "SAF Agents", "ReliableWseeJaxwsSAFAgent" and "ReliableWseeSAFAgent" are listed. On the left side, there is a "Domain Structure" tree view showing the "demoWebApp" domain with its various components like Environment, Deployments, Services, and Messaging. Below the tree are sections for "How do I..." (with a bullet point about configuring subdeployments) and "System Status" (showing health metrics for running servers). Navigation buttons at the bottom include Back, Next, Finish, and Cancel.

## Διαδικασία Δημιουργίας ενός Queue

Μεταβαίνουμε Services -> Messaging -> JMS Modules -> SystemModule-0, όπως και στην περίπτωση του JMS Connection Factory πατάμε “New” αλλά αυτήν την φορά επιλέγουμε το “Queue” στην επόμενη οθόνη.

The screenshot shows the Oracle WebLogic Server Administration Console. In the top navigation bar, it says "Create a New JMS System". The main content area is titled "Create a New JMS System Module Resource". Below this, there are buttons for "Back", "Next", "Finish", and "Cancel". A section titled "Choose the type of resource you want to create." lists several options: "Connection Factory", "Queue" (which is selected), "Topic", "Distributed Queue", "Distributed Topic", "Foreign Server", and "Quotas". Each option has a brief description. On the left side, there is a "Domain Structure" tree view showing the demoWebApp domain with various services like Environment, Deployments, Services (including Messaging with JMS Servers, Store-and-Forward Agents, JMS Modules, Path Services, Bridges, Data Sources, Persistent Stores, Foreign JNDI Providers, and Work Contexts). There is also a "How do I..." sidebar with a list of configuration tasks.

Δηλώνουμε το όνομα του Queue στο πεδίο “Name” και στο “JNDI Name” δηλώνουμε το όνομα με το οποίο θα αναφερθούμε σε αυτό στον κώδικά μας. Αφήνουμε το “Template” ως “None”.

This screenshot shows the same "Create a New JMS System Module Resource" wizard as the previous one, but the "Name" field now contains "WebAppDemo/jms/queues/". The "JNDI Name" field also contains "WebAppDemo/jms/queues/computationalRequestsQueue". The "Template" dropdown menu is open and shows "None". The "Finish" button at the bottom of the wizard is highlighted with a yellow background. The rest of the interface is identical to the previous screenshot, including the Domain Structure tree and the "How do I..." sidebar.

Στο πεδίο “**Subdeployments**” ορίζουμε το Subdeployment που δημιουργήσαμε προηγουμένως και στα “**Targets**” τον JMS Server. Πατάμε “**Finish**”.

The screenshot shows the Oracle WebLogic Server Administration Console interface. The left sidebar contains a 'Domain Structure' tree with nodes like demoWebApp, Environment, Deployments, Services (Messaging, Data Sources, Bridges, etc.), and Work Contexts. A 'How do I...' panel on the left provides links for configuring quotas, templates, destination keys, topics, queues, connection factories, and uniform distributed topics. The main content area is titled 'Create a New JMS System Module Resource'. It includes a breadcrumb trail: Home > JMServer-1 > Summary of JMS Modules > Summary of JMS Servers > Summary of JMS Modules > SystemModule-1 > Summary of Deployments > Summary of JMS Modules > SystemModule-1 > Summary of JMS Modules > SystemModule-1. A message at the top says 'No pending changes exist. Click the Release Configuration button to allow others to edit the domain.' Below this are 'Lock & Edit' and 'Release Configuration' buttons. The 'Subdeployments' section shows 'JMServer-1\_SD' selected from a dropdown. The 'Targets' section lists four JMS Servers: JMServer-1, WseeJaxwsJmsServer, WseeJmsServer, and WseeSoapjmsJmsServer. The 'JMServer-1' option is highlighted with a radio button.

## Ανάλυση Κώδικα Παραδείγματος

### Γενική Περιγραφή

Ο κώδικας του παραδείγματος αποτελείται από τον Servlet **ComputeFibonacciServlet** ο οποίος δέχεται τα HTTP Requests από το `private_services.jsp`, το Singleton **FibonacciComputationsStoreSingleton** στον οποίο ο `ComputeFibonacciServlet` υποβάλει τα requests για ασύγχρονο υπολογισμό, και το MDB **ComputeFibonacciMDB** στο οποίο ο `FibonacciComputationsStoreSingleton` μεταβιβάζει τα requests που δέχεται εάν δεν έχει αποθηκευμένο εσωτερικά τον υπολογισμό που του ζητήθηκε.

Το **FibonacciComputationsStoreSingleton** σχηματίζεται από τα classes **AsyncCompStoreAbstract**, **AsyncCompStoreSingEJB** και **AsncCmpStrSingToMDB** που κάνουν implement το interface **AsyncCompStore**.

### *AsyncCompStore Interface*

Αρχικά θα παρουσιάσουμε το interface **AsyncCompStore** και θα εξηγήσουμε για ποια λειτουργικότητα προορίζεται η κάθε μέθοδος και τις σχέσεις μεταξύ τους.

```

1 package osotnikov.async.services;
2
3 import osotnikov.async.services.vo.ComputationResponse;□
4
5 public interface AsyncCompStore<Key, CReq extends ObjectWithKey<Key>, CResp extends ComputationResponse<Key>> {
6
7     /** Every request for computation should be made from here. */
8     CResp requestComputation(CReq compReq);
9
10    /** The purpose of this method is to model the atomic action of submitting an asynchronous
11     * request and updating the state of the computation to ComputationState.UNDER_PROCESSING
12     * iff the computation was successfully requested or to ComputationState.FAILED if the
13     * request has failed. There could me a number of ways to obtain the lock e.g. on the
14     * whole container structure or only on the element representing the computation. This
15     * method serves mainly to model the obtaining of the lock. */
16    CResp requestAsyncComputationAndUpdateStatus(CReq compReq);
17
18    /** @return Returns true if the asynchronous computation was successfully requested or false if the
19     * request has failed. */
20    boolean requestAsyncComputation(CReq compReq);
21
22    /** This model has to handle concurrency issues. */
23    void storeComputation(CResp cResp);
24
25    /** Can only be provided by the last class in the hierarchy. */
26    CResp createPlainCRespInstance();
27
28 }
29 }
```

Αρχικά η **requestComputation** προορίζεται για να ψάξει για το computation που ζητήθηκε στην δομή τύπου Map που η υλοποίηση του interface θα πρέπει να διαθέτει εσωτερικά. Εάν βρεθεί και το `ComputationState` του είναι `ComputationState.COMPUTED` ή `ComputationState.UNDER_PROCESSING` τότε επιστρέφει το αντικείμενο από την εσωτερική δομή. Σε όλες τις άλλες περιπτώσεις θα πρέπει να γίνει υποβολή του αιτήματος προς το component που πραγματοποιεί το ασύγχρονο computation (στην περίπτωσή μας το MDB **ComputeFibonacciMDB**) για να μπορέσει αυτό κάποια στιγμή να αποθηκεύσει στην εσωτερική δομή της υλοποίησης του **AsyncCompStore** το `ComputationResponse` κάνοντας χρήση της μεθόδου **storeComputation**.

Για να επιτευχθεί αυτό η **requestComputation** καλεί την μέθοδο **requestAsyncComputationAndUpdateStatus** της οποίας ο ρόλος είναι να κάνει ασφαλή (από την άποψη διαχείρισης παραλληλίας) κλήση της μεθόδου **requestAsyncComputation** που υποβάλει

ασύγχρονα το αίτημα προς κάποιου είδους component που μπορεί να το εξυπηρετήσει (στην περίπτωσή μας το **ComputeFibonacciMDB**). Επιπρόσθετα, μέσα στην αδιαίρετη ενέργεια που υποτίθεται ότι ορίζει η μέθοδος **requestAsyncComputationAndUpdateStatus**, θα πρέπει να ορίσει το status του υπολογισμού για το συγκεκριμένο όρισμα που της δόθηκε, σε **ComputationState.UNDER\_PROCESSING** και να το αποθηκεύσει στην εσωτερική δομή τύπου Map.

Το component στο οποίο η `requestAsyncComputation` υπέβαλλε το αίτημα (`ComputeFibonacciMDB`) θα πρέπει τελικώς να το επεξεργαστεί και να αποθηκεύσει το αποτέλεσμα της επεξεργασίας στην εσωτερική δομή του αντικειμένου υλοποίησης του `AsyncCompStore` (`FibonacciComputationsStoreSingleton`) ανανεώνοντας επιπρόσθετα το `status` του υπολογισμού στο `ComputationState.COMPUTED`.

Μία σημαντική τελευταία παρατήρηση είναι ότι η `requestComputation` είναι η μέθοδος που καλείται από τον `ComputeFibonacciServlet` και γενικά, μαζί με την `storeComputation` θα πρέπει να είναι οι μόνες μέθοδοι που θα πρέπει να καλούνται εξωτερικά της υλοποίησης του `interface AsyncCompStore`.

### *JMS Sender (AsyncCompStore.requestAsyncComputation)*

Από όλες τις μεθόδους του AsyncCompStore interface θα παρουσιάσουμε μόνο τον κώδικα της **requestAsyncComputation** (ορίζεται στην **AsncCmpStrSingToMDB**) καθώς είναι η μόνη που μας ενδιαφέρει στα πλαίσια της μελέτης του JMS και των MDB αφού **αναλαμβάνει την αποστολή του JMS Message**.

```
1 package osotnikov.async.ejb;
2
3 import javax.jms.Connection;□
4
5 public abstract class AsncCmpStrSingToMDB<Key, CReq extends ObjectWithKey<Key>, CResp extends
6 AsyncCompStoreSingEJB<Key, CReq, CResp>{
7
8     private ConnectionFactory connFactory;
9     private Queue queue;
10
11    @Override
12    public boolean requestAsyncComputation(CReq compReq) {
13
14        Connection jmsCon = null;
15        Session jmsSess = null;
16        MessageProducer prod = null;
17        ObjectMessage objMsg = null;
18
19        try {
20            // SEND THE MESSAGE TO THE QUEUE
21            jmsCon = getConnFactory().createConnection();
22            jmsSess = jmsCon.createSession();
23            prod = jmsSess.createProducer(getQueue());
24            objMsg = jmsSess.createObjectMessage();
25            objMsg.setObject(compReq);
26            prod.send(objMsg);
27
28            return true;
29
30        } catch (Exception e) {
31
32            e.printStackTrace();
33            return false;
34        } finally{
35
36            try {
37
38
39
40
41
42
43
44
```

```

45         if (prod != null) {
46             prod.close();
47         }
48         if (jmsSess != null) {
49             jmsSess.close();
50         }
51         if (jmsCon != null) {
52             jmsCon.close();
53         }
54     } catch (JMSEException e) {
55         System.out.println("Failed to release a JMS Resource");
56     }
57 }
58 }
```

Ο κώδικας αυτός κάνει αρχικά χρήση ενός αντικειμένου τύπο **javax.jms.ConnectionFactory**. Το αντικείμενο αυτό παρέχεται από την μέθοδος **getConnFactory** η οποία γίνεται override, στην περίπτωσή μας από την τελική concrete κλάση που υλοποιεί το interface **AsyncCompStore**, την **FibonacciComputationsStoreSingleton**.

Η μέθοδος **getConnFactory** θα επιστρέψει το **ConnectionFactory** που φτιάχαμε προηγουμένως μέσω της κονσόλας του Weblogic Server.

Με το αντικείμενο **ConnectionFactory** δημιουργούμε ένα **javax.jms.Connection** και μέσω αυτού ένα **javax.jms.Session**.

Το **Session αντικείμενο** είναι αυτό που θα χρησιμοποιηθεί για την δημιουργία ενός **MessageProducer** μέσω του οποίου θα γίνεται η υποβολή μηνυμάτων προς την ενδιάμεση δομή αποθήκευσής τους (**Queue**) από όπου θα τα καταναλώσει κάποιος consumer.

Εδώ σημειώνουμε ότι ενώ **ένα αντικείμενο Connection είναι thread safe**, δηλαδή μπορεί να προσπελάζεται ταυτόχρονα από πολλά threads που δημιουργούνε Session instances, **ένα αντικείμενο τύπο Session δεν είναι thread safe (και επακολούθως και ένα αντικείμενο MessageProducer)**, δηλαδή μπορεί να χρησιμοποιηθεί μόνο από ένα thread. Έτσι εάν θέλουμε να στέλνουμε ταυτόχρονα μηνύματα προς ένα Queue από πολλά threads μπορούμε να το κάνουμε αλλά το κάθε thread θα πρέπει να χρησιμοποιεί το δικό του Session instance.

Τέλος δημιουργούμε ένα **ObjectMessage**. Τα JMS Messages τέτοιου τύπου μπορούν να μεταφέρουν ένα οποιοδήποτε **Serializable** αντικείμενο.

Η αποστολή γίνεται τελικά με την μέθοδο **MessageProducer.send**.

Στο finally block βλέπουμε ότι θα πρέπει να κάνουμε **close** τα αντικείμενα **MessageProducer**, **Session** και **Connection**.

## FibonacciComputationsStoreSingleton

Εδώ παρουσιάζουμε το concrete class **FibonacciComputationsStoreSingleton** που ολοκληρώνει την υλοποίηση του AsyncCompStore interface. Ο ρόλος του είναι να προσφέρει τα JMS Resources ConnectionFactory και Queue καθώς και μια μέθοδο για την δημιουργία ενός ComputeFibonacciResponse instance που χρειάζονται οι άλλες μέθοδοι του AsyncCompStore interface.

```
14
15 @Singleton(name="FibonacciComputationsStoreSingleton")
16 @Local(AsyncCompStore.class)
17 public class FibonacciComputationsStoreSingleton
18     extends AsncCmpStrSingToMDB<Integer, ComputeFibonacciRequest, ComputeFibonacciResponse>{
19
20     @Override
21     // ConnectionFactory, Connection and Queue (Destination in general) objects are
22     // thread safe (not that it matters here). Session, MessageProducer, and
23     // MessageConsumer are not.
24     @Resource(
25         name="jms/CF" ,// Will create the reference to the lookup resource in the ENC with the specified name.
26         type=javax.jms.ConnectionFactory.class,
27         lookup="WebAppDemo/jms/conFactories/CF") // Will lookup the resource in the JNDI context.
28     public void setConnFactory(ConnectionFactory connectionFactory) {
29         super.setConnFactory(connectionFactory);
30     }
31
32     @Override
33     @Resource(
34         name="jms/Q", //You can also specify the lookup value here, this attribute exhibits overloaded behavior...
35         type=javax.jms.Queue.class,
36         lookup = "WebAppDemo/jms/queues/computationalRequestsQueue")
37     public void setQueue(Queue fibonacciQueue) {
38         super.setQueue(fibonacciQueue);
39     }
40
41     @Override
42     @Lock(LockType.READ)
43     public ComputeFibonacciResponse createPlainCRespInstance() {
44
45         return new ComputeFibonacciResponse();
46     }
47
48 }
```

Όπως παρατηρούμε τα resources ConnectionFactory και Queue παρέχονται μέσω των annotations “**@Resource**” που εφαρμόζονται πάνω στους κατάλληλους setters. Τα attributes “**lookup**” αυτών των annotations δηλώνουν τα JNDI Names που ορίσαμε για αυτά τα resources όταν τα φτιάχναμε μέσω της κονσόλας του Weblogic Server. Τα attributes “**name**” δεν είναι υποχρεωτικά, εδώ χρησιμοποιούνται για να φτιάξουν μία ακόμη αναφορά σε αυτά τα resources, μέσα στο ENC (Enterprise Naming Context, ένα JNDI Context που ορίζεται από τα EJB πρότυπα) η οποία δεν έχει άμεση χρησιμότητα. Το attribute “**type**”, γενικώς όσον αφορά το annotation “**@Resource**”, θα πρέπει να δηλώνει τον τύπο του resource που γίνεται injected έτσι ώστε να μπορεί να γίνει injected στο field του ίδιου τύπου, πάνω στο οποίο εφαρμόζεται αυτό το annotation.

## **Message Driven Bean – ComputeFibonacciMDB**

Ενώ υλοποιήσαμε την αποστολή του JMS Message μέσω του JMS API δεν κάνουμε το ίδιο και με την παραλαβή του μηνύματος η οποία γίνεται με ένα **EJB τύπου Message Driven Bean (MDB)**, το **ComputeFibonacciMDB**. Ένα MDB βασίζεται στο JMS αλλά μας προσφέρει μία πολύ απλούστερη διεπαφή με αυτό και κάνει αυτόματη διαχείριση θεμάτων όπως π.χ. το **load balancing** όπου σε κάθε MDB αντιστοιχεί ένα instance pool (παρόμοιο με αυτό των Stateless Beans) όπου τα instances μπορούν να λαμβάνουν και να επεξεργάζονται τα μηνύματα από το destination (queue ή topic) παράλληλα. Επίσης το MDB είναι ένα Container Managed Component που σημαίνει ότι όλες οι παράμετροι που έχουν σχέση με την ασύγχρονη επικοινωνία μέσω JMS μπορούν να παραμετροποιηθούν από την κονσόλα του Weblogic.

Γενικά ένα MDB θα πρέπει να δηλώνεται ως τέτοιο με annotations ή στον ejb deployment descriptor (ejb-jar.xml). Στην περίπτωση του ComputeFibonacciMDB χρησιμοποιούμε το annotation **@MessageDriven** που είναι αντίστοιχο των **@Stateless** / **@Stateful** / **@Singleton** που είδαμε σε προηγούμενα κεφάλαια που χρησιμοποιούνται για την δήλωση των αντίστοιχων EJB.

Μέσα στο **@MessageDriven** annotation δηλώνουμε τα properties για τον τύπο του destination (**Queue**) από το οποίο το MDB θα λαμβάνει τα μηνύματα καθώς και το JNDI όνομά του όπως επίσης και το JNDI όνομα του **ConnectionFactory** που θα χρησιμοποιηθεί. Στο ComputeFibonacciMDB χρησιμοποιούμε αυτά που φτιάχαμε στο προηγούμενο κεφάλαιο μέσω της κονσόλας του Weblogic.

Επίσης ένα MDB θα πρέπει να κάνει implement to interface **javax.jms.MessageListener** το οποίο δηλώνει μια μοναδική μέθοδο, την **“void onMessage(Message message)”**.

```
1 package osotnikov.demowebapp.services.mdb;
2
3 import java.util.Date;
4
5 @MessageDriven(
6     name = "ComputeFibonacciBean",
7     activationConfig = {
8         @ActivationConfigProperty(propertyName = "destinationType",
9             propertyValue = "javax.jms.Queue"),
10        @ActivationConfigProperty(propertyName = "connectionFactoryJndiName",
11            propertyValue = "WebAppDemo/jms/conFactories/CF"), // External JNDI Name
12        @ActivationConfigProperty(propertyName = "destinationJndiName",
13            propertyValue = "WebAppDemo/jms/queues/computationalRequestsQueue"), // Ext. JNDI Name
14    }
15)
16 public class ComputeFibonacciMDB implements MessageListener {
17
18     @Inject
19     MathUtils mathUtils;
20
21     @EJB(beanName="FibonacciComputationsStoreSingleton")
22     AsyncCompStore<Integer, ComputeFibonacciRequest, ComputeFibonacciResponse> fibCompStore;
23
24     @PersistenceContext(unitName="web_demo_pu")
25     private EntityManager webDemoEm;
26
27     @PersistenceContext(unitName="web_demo_crm_pu")
28     private EntityManager webDemoCrmEm;
29
30     @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
31     // Don't forget to set the JTA transaction time limit in the server's console.
32     public void onMessage(Message msg) {
33
34 }
```

```

54 ComputeFibonacciResponse fibResp = new ComputeFibonacciResponse();
55 fibResp.setCompState(ComputationState.UNDER_PROCESSING);
56
57 try{
58
59     System.out.println("ComputeFibonacciMDB onMessage started at: " + new Date());
60
61     // EXTRACT THE JMSMESSAGEID
62     String jmsMsgId;
63     jmsMsgId = msg.getJMSMessageID(); // may throw a JMSEException
64     System.out.println("JMSMessageId: " + jmsMsgId);
65
66     // EXTRACT REQUEST
67     ComputeFibonacciRequest fibReq = null;
68     try {
69         fibReq = (ComputeFibonacciRequest) (((ObjectMessage)msg).get\Object());
70     } catch (JMSEException e) {
71
72         fibResp.setCompState(ComputationState.FAILED);
73         fibCompStore.storeComputation(fibResp);
74
75         e.printStackTrace();
76
77         return;
78     }
79
80     // SET THE TERM INDEX (fibonacci computation argument) OF THE RESPONSE
81     fibResp.setTermInd(fibReq.getTermInd());
82
83     // LOG THE COMPUTATION STEP
84     System.out.println("ComputeFibonacciMDB: JMSMessageId: " + jmsMsgId +
85                         ", FibCompStep: " + FibCompStep.STARTED.getStepName());
86
87     // DO THE COMPUTATION
88     Date compStartTime = new Date();
89     fibResp.setResult(mathUtils.computeFibonacci(fibReq.getTermInd()));


90     Date compEndTime = new Date();
91
92     // STORE THE RESULT IN THE COMPUTATION STORE
93     fibResp.setCompState(ComputationState.COMPUTED);
94     fibCompStore.storeComputation(fibResp);
95
96     // LOG THE COMPUTATION STEP
97     System.out.println("ComputeFibonacciMDB: JMSMessageId: " + jmsMsgId +
98                         ", FibCompStep: " + FibCompStep.COMPUTED.getStepName());
99
100    // CREATE A BILLING RECORD
101    FibCompBill fibCompBill = new FibCompBill();
102    fibCompBill.setTermInd(fibReq.getTermInd());
103    fibCompBill.setResult(fibResp.getResult());
104    fibCompBill.setStartTime(compStartTime);
105    fibCompBill.setEndTime(compEndTime);
106    long duration = compEndTime.getTime() - compStartTime.getTime();
107    fibCompBill.setDuration(duration);
108    long cost = duration * FibCompBill.BILLING_FACTOR;
109    fibCompBill.setCost(cost);
110    fibCompBill.setUser(fibReq.getUser());

111    // STORE fibCompBill RECORD IN DB
112    webDemoEm.persist(fibCompBill);
113    webDemoEm.flush();

114
115    // LOG THE COMPUTATION STEP
116    System.out.println("ComputeFibonacciMDB: JMSMessageId: " + jmsMsgId +
117                         ", FibCompStep: " + FibCompStep.CREATED_BILL.getStepName());
118
119    // CREATE A CRM RECORD
120    FibCompCrmLog fibCompCrmLog = new FibCompCrmLog();
121    fibCompCrmLog.setTermInd(fibReq.getTermInd());
122    fibCompCrmLog.setResult(fibResp.getResult());
123    fibCompCrmLog.setStartTime(compStartTime);
124    fibCompCrmLog.setEndTime(compEndTime);


```

```

126     fibCompCrmLog.setDuration(duration);
127     fibCompCrmLog.setCost(cost);
128     webDemoEm.refresh(fibCompBill); // Make sure we get the billing id...
129     fibCompCrmLog.setBillingId(fibCompBill.getId());
130     fibCompCrmLog.setUsername(fibReq.getUser().getEmail());
131     String firstAndLastname = fibReq.getUser().getFirstName() + " " + fibReq.getUser().getLastName();
132     fibCompCrmLog.setFirstAndLastname(firstAndLastname);
133
134     // STORE fibComBill RECORD IN DB
135     webDemoCrmEm.persist(fibCompCrmLog);
136     webDemoCrmEm.flush();
137
138     // LOG THE COMPUTATION STEP
139     System.out.println("ComputeFibonacciMDB: JMSMessageId: " + jmsMsgId +
140                         ", FibCompStep: " + FibCompStep.CREATED_CRM_LOG.getStepName());
141
142 } catch (Exception e) {
143     // Mostly in case of runtime exceptions.
144     e.printStackTrace();
145
146 } finally{
147
148     if(ComputationState.UNDER_PROCESSING.equals(fibResp.getCompState())){
149         fibResp.setCompState(ComputationState.FAILED);
150         fibCompStore.storeComputation(fibResp);
151     }
152 }
153
154 }
155
156 }

```

Συγκεκριμένα το ComputeFibonacciMDB κάθε φορά που δέχεται ένα μήνυμα κάνει καταγραφή του μοναδικού αναγνωριστικού του (JMSMessageId) και μετά το κάνει downcast σε ObjectMessage έτσι ώστε μέσω της μεθόδου getObject να πάρει το ComputeFibonacciRequest αντικείμενο που αποθηκεύσαμε προηγουμένως στο μήνυμα κατά την αποστολή από την **AsncCmpStrSingToMDB.requestAsyncComputation**.

Επίσης φτιάχνεται και ένα ComputeFibonacciResponse αντικείμενο που θα αποθηκευτεί στο FibonacciComputationsStoreSingleton με το κατάλληλο status ανάλογα με την επιτυχία ή αποτυχία του υπολογισμού.

Ο υπολογισμός του αριθμού Fibonacci γίνεται στην γραμμή 89. Αμέσως πριν και αμέσως μετά καταγράφουμε τους χρόνους για να μπορέσουμε να υπολογίσουμε την διάρκειά που θα χρησιμοποιηθεί για τον υπολογισμό της χρέωσης του πελάτη.

Η μέθοδος **MathUtils.computeFibonacci** κάνει αναδρομικό υπολογισμό του αριθμού. Για σχετικά μικρές τιμές εισόδου (στον υπολογιστή του γράφοντος πάνω από 25) αρχίζει και απαιτεί αρκετή ώρα για να τελειώσει. Αυτό προφανώς είναι επιθυμητό καθώς στόχος μας ήταν η προσομοίωση της εκτέλεσης μίας διαδικασίας που χρειάζεται κάποιο χρόνο για να ολοκληρωθεί.

Μετά από αυτό γίνεται η δημιουργία των entities **FibCompBill** και **FibCompCrmLog**. Με το FibCompBill θα φτιάξουμε μία εγγραφή στον **FIB\_COMP\_BILL** στην **WEB\_DEMO** ΒΔ της **Oracle 11G** και με το **FibCompCrmLog** στον πίνακα **fib\_comp.crm\_log** στην **web\_demo.crm** ΒΔ της **MySQL**.

Μεταξύ όλων αυτών των ενεργειών γίνεται καταγραφή των σημείων από τα οποία περνάει η διαδικασία στην κονσόλα της εφαρμογής έτσι ώστε σε περίπτωση αποτυχίας σε ένα από τα βήματα να μπορούμε να διαπιστώσουμε μέχρι που έφτασε η διαδικασία. Αυτό θα μας χρειαστεί στο κεφάλαιο στο οποίο θα κάνουμε μελέτη των **Transactions**.

# Context and Dependency Injection – CDI

## Γενική Περιγραφή

Το CDI είναι ένα πρότυπο της Java EE που καλύπτει το Dependency Injection.

Ενώ το Dependency Injection καλύπτεται και από το “@EJB” annotation, αυτό απαιτεί το resource που γίνεται injected να είναι ένα EJB. Δεν μπορεί δηλαδή να είναι απλώς ένα POJO, κάτι που υποστηρίζεται από το CDI.

Το CDI ορίζει το annotation “**@Inject**” μέσω του οποίου μπορεί να γίνει injection μίας οποιασδήποτε κλάσης καλύπτει τις απαιτήσεις ενός **“Managed Bean”**.

Ένα **Managed Bean** μπορεί να γίνει οποιαδήποτε κλάση της Java που δεν είναι abstract και δεν είναι nonstatic inner class. Επίσης ένα Managed Bean θα πρέπει να έχει ένα no-argument constructor (μπορεί να υποστηριχθεί και κλάση με άλλον constructor αλλά χρειάζεται να παραμετροποιηθεί μέσω του annotation @Inject).

**Δεν χρειάζεται να δηλώσουμε τα Managed Beans με annotations ή μέσα σε configuration files.**

Το annotation “**@Inject**” δουλεύει παρόμοια με το “@EJB” αλλά με την διαφορά ότι κάνει injection ενός proxy της κλάσης που θέλουμε (παρόμοιο με ένα proxy ενός EJB) που όμως είναι ένα Managed Bean και όχι ένα EJB.

Δηλαδή ακόμα και αν κάνουμε Inject μέσω του annotation “**@Inject**” μίας κλάσης που την έχουμε ορίσει ως EJB αυτή δεν θα είναι ένα EJB ούτε και θα συμπεριφέρεται σαν τέτοιο. Π.χ. **δεν θα υποστηρίζεται Transaction Management από τον Container (CMT)** και επίσης π.χ. για ένα Stateless Bean δεν θα φτιάχνεται ένα instance pool αλλά ένα μοναδικό instance ή ό,τι είναι ορισμένο από οποιαδήποτε άλλα CDI annotations εφαρμόσουμε πάνω στο Injection point. Αυτό είναι μια σημαντική διαφορά καθώς τα Managed Beans **δεν υλοποιούν ούτε κάποιο concurrency handling** όπως π.χ. τα Stateful Beans που επιτρέπουν την προσπέλαση μόνο από ένα thread κάθε φορά με blocking όλων των υπόλοιπων. Θα μπορούσαμε να πούμε ότι διαχειρίζονται το concurrency με έμμεσο τρόπο μέσω των Scopes.

## Scopes

Το CDI είναι ένα απαραίτητο συμπλήρωμα στα EJB καθώς ορίζουν την έννοια του **“Scope”**. Ένα CDI σε αντίθεση με ένα EJB μπορεί να υποστηρίξει τα παρακάτω Scopes:

Scope	Annotation	Duration
Request	@RequestScoped	To managed bean που γίνεται Injected με αυτό το annotation θα διαθέτει ένα καινούριο αντικείμενο για κάθε HTTP Request.
Session	@SessionScoped	Θα διαθέτει ένα καινούριο αντικείμενο σε κάθε νέο HTTP Session.
Application	@ApplicationScoped	Θα υπάρχει ένα instance του managed bean σε όλο το application παρόμοια με ένα Singleton EJB.
Dependent	@Dependent	Είναι το Scope που ισχύει by default εάν δεν οριστεί κάτι άλλο. Το managed bean που θα γίνει injected με αυτό το scope έχει την ίδια διάρκεια ζωής με το αντικείμενο στο οποίο έγινε injected.
Conversation	@ConversationScoped	Εφαρμόζεται μόνο μέσα στα πλαίσια του JSF (Java Server Faces) και αφορά την έννοια των “Conversations” που αυτό ορίζει.

Το μειονέκτημα με την υλοποίηση του Scope από το CDI πρότυπο είναι ότι **δεν μπορεί να οριστεί στο σημείο που γίνεται το injection** αλλά θα πρέπει να οριστεί αναγκαστικά πάνω στην κλάση υλοποίησης του managed bean. Έτσι στην περίπτωση που θέλουμε να εισάγουμε την ίδια υλοποίηση σαν δύο διαφορετικά managed beans με διαφορετικό scope θα πρέπει να φτιάξουμε μία κλάση που να κάνει extend την αρχική (χωρίς να κάνει override το οτιδήποτε) και να ορίσουμε πάνω σε αυτήν το επιπλέον scope με το οποίο θέλαμε να χρησιμοποιήσουμε την αρχική. Πάντως δεν θα χρειαστεί να κάνουμε injection σε fields με διαφορετικούς τύπους. Όλα τα fields θα μπορούνε να έχουνε για τύπο την αρχική κλάση ή ένα interface της και η επιλογή στα implementations θα γίνεται με βάση “**Qualifiers**” (αναλύονται στο επόμενο υποκεφάλαιο).

## Qualifiers & Alternatives

### Γενικά

Τα CDI δεν ορίζουν ονόματα για τα **managed beans** όπως γίνεται στα EJB οπότε δεν μπορούμε να κάνουμε injection κάποια υλοποίηση με βάση ένα όνομα. Η διαδικασία που ακολουθείται για την ικανοποίηση ενός dependency είναι με βάση τον τύπο του field στο οποίο γίνεται το injection. Με λίγα λόγια γίνεται αναζήτηση σε όλο το classpath για κάποια κλάση που κάνει implement ή extend τον τύπο του field και **αν βρεθεί παραπάνω από μια υλοποίηση προκαλείται σφάλμα** και το dependency δεν ικανοποιείται.

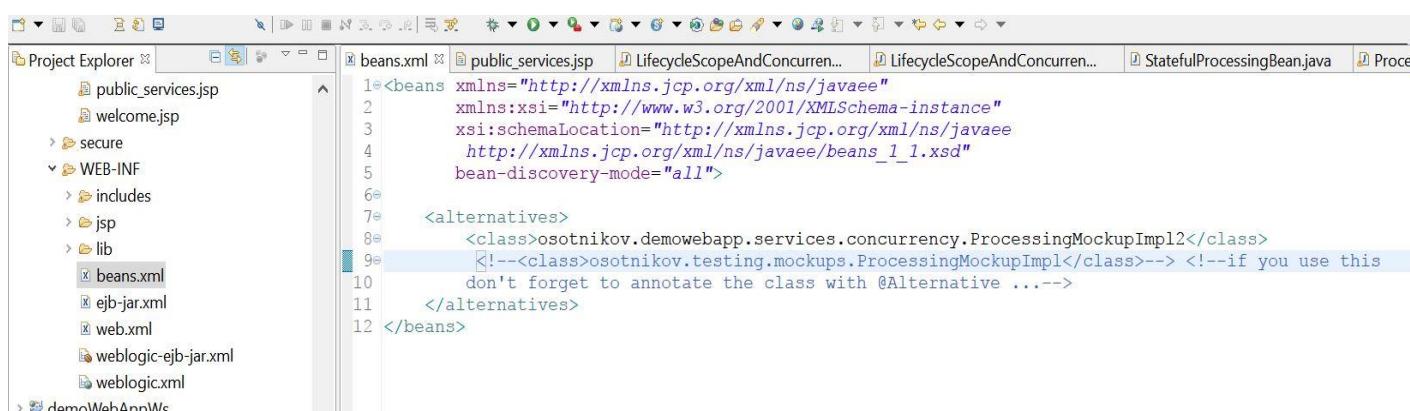
Όταν έχουμε μόνο μία υλοποίηση για τον τύπο του field στο οποίο πραγματοποιούμε το injection αυτό δεν αποτελεί πρόβλημα. Όταν έχουμε όμως π.χ. πολλές υλοποιήσεις του ίδιου interface και θέλουμε να ορίσουμε το ποια από αυτές θα χρησιμοποιηθεί έχουμε δύο λύσεις, τα **Alternatives** και τα **Qualifiers**.

### Alternatives

Εάν θέλουμε να χρησιμοποιηθεί μία υλοποίηση παντού μέσα στον κώδικα αλλά διαθέτουμε πολλές μπορούμε να ορίσουμε το ποια από αυτές θα χρησιμοποιηθεί μέσω του αρχείου “beans.xml”.

Κάθε εφαρμογή που χρησιμοποιεί CDI θα πρέπει να διαθέτει ένα αρχείο “beans.xml”. Στα web applications θα πρέπει να βρίσκεται στον φάκελο WEB-INF.

Παρακάτω παρουσιάζεται το beans.xml του DemoWebApp.



```

<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       bean-discovery-mode="all">
    <alternatives>
        <class>osotnikov.demowebapp.services.concurrency.ProcessingMockupImpl</class>
        <!--<class>osotnikov.testing.mockups.ProcessingMockupImpl</class>--> <!--if you use this
        don't forget to annotate the class with @Alternative ...-->
    </alternatives>
</beans>

```

To element “**<alternatives>**” ορίζει τις υλοποιήσεις που θα χρησιμοποιηθούν. Π.χ. στο DemoWebApp έχουμε δύο υλοποιήσεις του interface **osotnikov.testing.mockups.ProcessingMockup**, την κλάση

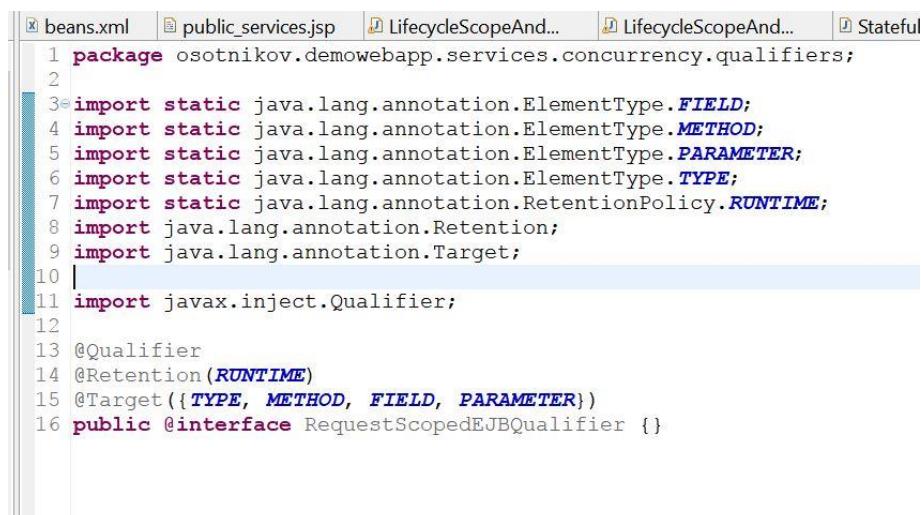
**osotnikov демовебапп.services.concurrency.ProcessingMockupImpl2** και την κλάση **osotnikov.testing.mockups.ProcessingMockupImpl**. Η υλοποίηση που θα χρησιμοποιηθεί όμως στα Injection points με τύπο ProcessingMockup (εάν δεν υπάρχουν annotations που να δηλώνουν κάτι διαφορετικό) θα είναι η **ProcessingMockupImpl2** διότι είναι αυτή που ορίζεται στο element “`<alternatives>`” (η άλλη είναι σε σχόλιο).

To interface ProcessingMockup, καθώς και το implementation του, έχουν χρησιμοποιηθεί και στα προηγούμενα παραδείγματα του EJB κεφαλαίου. Εδώ υπενθυμίζεται ότι η **“ProcessingMockup.process(String mockupName, long duration)”** θα κληθεί με το duration να ορίζει 3 δευτερόλεπτα (το duration ορίζει τον χρόνο ο οποίος θα απαιτηθεί για την ολοκλήρωση της μεθόδου process) και θα επιστρέψει ένα report τύπου String το οποίο θα εμφανιστεί στην public\_services.jsp μετά το πάτημα ενός κουμπιού “Execute”. Το report αναφέρει τον χρόνο που ξεκίνησε και τον χρόνο που ολοκληρώθηκε η μέθοδος process καθώς και το hashCode του αντικειμένου από το οποίο κλήθηκε. Επίσης θα εμφανίζει την παράμετρο mockupName στην οποία θα δοθεί μία περιγραφική ονομασία του case το οποίο έχει σκοπό να εξετάσει η αντίστοιχη παράγραφος με το “Execute” κουμπί στην public\_services.jsp.

## ***Qualifiers***

Η δεύτερη λύση στην περίπτωση που έχουμε πολλές υλοποιήσεις για τον ίδιο τύπο που χρησιμοποιείται σε κάποιο Injection point, είναι τα **Qualifiers**. Τα qualifiers είναι annotations τα οποία πρέπει να τα ορίσει ο προγραμματιστής της εφαρμογής και με τα οποία **μπορούμε να ορίσουμε την προτιμώμενή μας υλοποίηση για κάποιο injection point, κάνοντας ακόμα και override το αρχείο beans.xml και τα alternatives**.

Στην DemoWebApp ορίζονται δύο qualifiers, το **RequestScopedEJBQualifier** και το **RequestScopedQualifier**. Τα qualifiers θα πρέπει να ορίζονται με τα annotations που φαίνονται στις παρακάτω εικόνες.



```
beans.xml public_services.jsp LifecycleScopeAnd... LifecycleScopeAnd... Stateful
1 package osotnikov демовебапп.services.concurrency.qualifiers;
2
3 import static java.lang.annotation.ElementType.FIELD;
4 import static java.lang.annotation.ElementType.METHOD;
5 import static java.lang.annotation.ElementType.PARAMETER;
6 import static java.lang.annotation.ElementType.TYPE;
7 import static java.lang.annotation.RetentionPolicy.RUNTIME;
8 import java.lang.annotation.Retention;
9 import java.lang.annotation.Target;
10
11 import javax.inject.Qualifier;
12
13 @Qualifier
14 @Retention(RUNTIME)
15 @Target({TYPE, METHOD, FIELD, PARAMETER})
16 public @interface RequestScopedEJBQualifier {}
```

```

1 package osotnikov.demowebapp.services.concurrency.qualifiers;
2
3 import static java.lang.annotation.ElementType.FIELD;
4 import static java.lang.annotation.ElementType.METHOD;
5 import static java.lang.annotation.ElementType.PARAMETER;
6 import static java.lang.annotation.ElementType.TYPE;
7 import static java.lang.annotation.RetentionPolicy.RUNTIME;
8 import java.lang.annotation.Retention;
9 import java.lang.annotation.Target;
10
11 import javax.inject.Qualifier;
12
13 @Qualifier
14 @Retention(RUNTIME)
15 @Target({TYPE, METHOD, FIELD, PARAMETER})
16 public @interface RequestScopedQualifier {}

```

Για να επιλέξουμε την υλοποίηση που θα γίνει injected σε ένα injection point με την βοήθεια των qualifiers θα πρέπει να εφαρμόσουμε το qualifier που δημιουργήσαμε στην κλάση υλοποίησης της προτίμησής μας και στο injection point (μαζί δηλαδή με το @Inject annotation) στο οποίο θέλουμε να γίνει injected ένα instance αυτής της κλάσης.

Στην DemoWebApp εφαρμογή χρησιμοποιούμε τα qualifiers σε συνδυασμό με τα scopes για να μπορέσουμε να κάνουμε injection managed beans (που όλα υλοποιούν το ProcessingMockup interface) με διαφορετικά scopes. Παρακάτω παρουσιάζουμε τις κλάσεις στις οποίες εφαρμόζονται τα προηγούμενα qualifiers.

```

1 package osotnikov.demowebapp.services.concurrency;
2
3 import javax.ejb.Local;
4
5 @Stateful(name="statefulRequestScopedProcessingBean")
6 @Local(ProcessingMockup.class)
7 @RequestScoped
8 @RequestScopedEJBQualifier
9 public class StatefulRequestScopedProcessingBean extends ProcessingEjbMockupImpl implements ProcessingMockup{
10
11 }
12
13
14
15
16
17
18

```

```

1 package osotnikov.demowebapp.services.concurrency;
2
3 import javax.enterprise.context.RequestScoped;
4
5 @RequestScoped
6 @RequestScopedQualifier
7 public class PojoRequestScopedProcessingBean extends ProcessingMockupImpl implements ProcessingMockup{
8
9 }
10
11
12
13

```

Τα παραδείγματα των injections των managed beans βρίσκονται στο **LifecycleScopeAndConcurrencyExperimentsSingleton** που έχουμε παρουσιάσει στο προηγούμενο κεφάλαιο των EJB.

Όπως παρουσιάστηκε και στο προηγούμενο κεφάλαιο για τα EJB έτσι και εδώ, όσον αφορά τα Managed Beans, μπορούμε μέσα από τα κουμπιά “Execute” που ορίζονται στην κάθε παράγραφο να υποβάλουμε αιτήματα σε κλάσεις που ναι μεν βασίζονται στην **ProcessingMockupImpl** αλλά αυτήν την φορά δεν είναι EJB αλλά **CDI Managed Beans**. Τα reports που θα παραχθούν θα μας βοηθήσουν να κατανοήσουμε την συμπεριφορά των **Managed Beans**.

στις περιπτώσεις **παράλληλης εκτέλεσης** των μεθόδων τους και να βγάλουμε συμπεράσματα για το **lifecycle** τους.

Παρακάτω παρουσιάζεται το κομμάτι κώδικα με τα **fields** της **LifecycleScopeAndConcurrencyExperimentsSingleton** κλάσης στα οποία θα γίνουν injected τα **Managed Beans** και για τα οποία υπάρχουν οι αντίστοιχες παραγραφοί στην σελίδα `public_services.jsp`.

```
28@  @Inject
29  @RequestScopedEJBQualifier
30  private ProcessingMockup statefulRequestScopedProcessingBean;
31
32@  @Inject
33  private ProcessingMockup pojoProcessingBean1;
34@  @Inject
35  private ProcessingMockup pojoProcessingBean2;
36@  @Inject // @EJB Won't take into account the request scope!!!
37  @RequestScopedQualifier
38  private ProcessingMockup pojoRequestScopedProcessingBean;
39
```

Με βάση τα qualifiers που παρουσιάσαμε πριν, το αρχείο beans.xml και το element “`<alternatives>`” μέσα σε αυτό, μπορούμε να συμπεράνουμε το ποιο implementation θα γίνει injected στο κάθε field.

- `statefulRequestScopedProcessingBean`: `StatefulRequestScopedProcessingBean`
- `pojoProcessingBean1`: `ProcessingMockupImpl2`
- `pojoProcessingBean2`: `ProcessingMockupImpl2`
- `pojoRequestScopedProcessingBean`: `PojoRequestScopedProcessingBean`

Όλα αυτά επιβεβαιώνονται αφού πατήσουμε τα κουμπιά execute στις αντίστοιχες παραγράφους της `public_services.jsp`.

The screenshot shows a web browser window with the URL `https://localhost:7002/DemoWebApp/public_JSP/public_services.jsp`. The page displays four sections, each with an "Execute" button:

- Second Stateful Bean with Dependent Scope**: Shows the output of the `statefulRequestScopedProcessingBean` injection.
- Stateful Bean Class Injected with CDI with Request Scope (Managed Bean with Request Scope)**: Shows the output of the `pojoProcessingBean1` injection.
- First POJO with Dependent Scope**: Shows the output of the `pojoProcessingBean2` injection.
- Second POJO with Dependent Scope**: Shows the output of the `pojoRequestScopedProcessingBean` injection.

In all sections, the "Execute" button is highlighted in blue, indicating it was used to capture the screenshot. The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with various icons.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η πρώτη περίπτωση. Το field **statefulRequestScopedProcessingBean** γίνεται injected με ένα instance της κλάσης StatefulRequestScopedProcessingBean που είναι annotated με το annotation @Stateful. Εάν το injection γινόταν με το annotation @EJB τότε θα είχαμε ένα EJB, όμως επειδή γίνεται με το annotation @Inject είναι απλώς ένα Managed Bean.

Έχει όντως request scope καθώς εάν πατήσουμε 3 φορές διαδοχικά του κουμπί Execute θα παρατηρήσουμε την τιμή του object hash code να αλλάζει 3 φορές σε λιγότερο από 9 δευτερόλεπτα (που θα έπαιρνε εάν είχαμε ένα instance που θα ήταν blocking προς παράλληλη επεξεργασία).

Το αξιοπερίεργο σε αυτό το case είναι ότι μπορούμε να πάρουμε και την τιμή του proxy αλλά με την μεθοδολογία που ακολουθήσαμε και στα EJB στο προηγούμενο κεφάλαιο (μέσω της **SessionContext.getBusinessObject** μεθόδου). Παρόλο που το **statefulRequestScopedProcessingBean** δεν αναφέρεται σε EJB το **SessionContext** γίνεται injected σε αυτό μέσω του “**@Resource**” annotation, που είναι όμως κάτι στο οποίο δεν μπορούμε να βασιζόμαστε πάντα καθώς μπορεί να τυχαίνει μόνο στην συγκεκριμένη υλοποίηση του Java EE Application Server (Weblogic 12C).

Τα δύο επόμενα cases “**First POJO with Dependent Scope**” και “**Second POJO with Dependent Scope**” αντιστοιχούν στα fields **pojoProcessingBean1** και **pojoProcessingBean2**. Πάνω σε αυτά δεν ορίζεται κανένα qualifier άρα η κλάση η οποία θα ικανοποιήσει τα dependencies θα προκύψει από το beans.xml αρχείο, στην δικιά μας περίπτωση η κλάση αυτή είναι η **ProcessingMockupImpl2** όπως διαπιστώνεται και στην αρχή των δύο reports της προηγούμενης εικόνας.

Επίσης βλέπουμε ότι τα processing object hash code είναι διαφορετικά για το **pojoProcessingBean1** και **pojoProcessingBean2** που δείχνει ότι σε κάθε διαφορετικό field με **@Dependent** scope (το default scope δηλαδή) αντιστοιχεί διαφορετικό Managed Bean ακόμα και αν είναι ίδιου τύπου με ένα άλλο Managed Bean σε άλλο field.

Εάν πατήσουμε επαναλαμβανόμενα και γρήγορα το κουμπί “Execute” σε ένα από αυτά τα cases θα διαπιστώσουμε ότι τα αιτήματα θα επεξεργασθούν παράλληλα αλλά αντιθέτως με την προηγούμενη περίπτωση του statefulRequestScopedProcessingBean όπου επεξεργαζόντουσαν από διαφορετικό instance σε κάθε request αυτήν την φορά θα επεξεργασθούν από το ίδιο κάτι που διαπιστώνεται από την σταθερή τιμή του processing object hash code.

Το τελευταίο case “**POJO with Request Scope**” θα έχει την ίδια συμπεριφορά με το πρώτο αφού είναι και αυτό ένα Request Scoped Managed Bean.

# Transactions

## Μελέτη των Transactions στα Πλαίσια της Μεθόδου

### ComputeFibonacciMDB.onMessage

Η θεωρητική περιγραφή των **Transactions** έγινε σε αντίστοιχο εισαγωγικό υποκεφάλαιο στο κεφάλαιο των Container Services. Εδώ θα γίνει επεξήγηση αυτής της έννοιας μέσω του παραδείγματος του ασύγχρονου υπολογισμού αριθμών Fibonacci που παρουσιάσαμε στο προηγούμενο κεφάλαιο.

Θα γίνει αναφορά μόνο σε **Container Managed Transactions (CMT)** καθώς καλύπτουν την πλειονότητα των εφαρμογών. Τέτοιου είδους transactions ορίζονται με annotations επί των μεθόδων ή των κλάσεων στις οποίες ανήκουν.

By default κάθε κλήση μίας μεθόδου ενός EJB ξεκινάει ένα CMT εκτός και εάν η κλήση μίας τέτοιας μεθόδου γίνεται στο πλαίσιο ενός transaction το οποίο εκτελείται ήδη, σε αυτήν την περίπτωση συμμετέχει και αυτή στο ήδη υπάρχων transaction. Μπορούν να οριστούν και άλλες συμπεριφορές όπως π.χ. κάθε μέθοδος να ξεκινάει ένα δικό της transaction ή να μην συμμετέχει σε transaction που ήδη εκτελείται, ακόμα και να πετάει exception στην περίπτωση που εκτελεσθεί μέσα σε transaction. Η default συμπεριφορά πάντως καλύπτει τις περισσότερες ανάγκες μας και είναι η μόνη που θα εξετασθεί.

Για την διαχείριση των global transactions τα EJB βασίζονται στην **JTA (Java Transaction API)** διεπαφή καθώς και στο **JTS (Java Transaction Service)** πρότυπο που ορίζει το τι θα πρέπει να υποστηρίζουν τα resources που συμμετέχουν σε ένα global transaction.

Το παράδειγμά μας θα αποτελέσει η μέθοδος **ComputeFibonacciMDB.onMessage**. Ο κώδικας της παρουσιάσθηκε στο κεφάλαιο που ανέλυε τα Message Driven Beans. Σε αυτό το κεφάλαιο απλώς θα γίνει ανάλυση του ίδιου κώδικα αλλά αυτήν την φορά ως προς τα transactions.

Η μέθοδος αυτή είναι annotated με το annotation “**@TransactionAttribute(value = TransactionAttributeType.REQUIRED)**”. Το annotation αυτό με το parameter value να παίρνει την συγκεκριμένη τιμή είναι περιττό καθώς είναι η default συμπεριφορά. Παρόλα αυτά τοποθετήθηκε για να δοθεί πληρότητα στο παράδειγμα καθώς μέσω αυτού του annotation θα ορίζαμε τις άλλες συμπεριφορές που αναφέραμε εάν το επιθυμούσαμε.

Όπως έχουμε αναλύσει και σε προηγούμενο κεφάλαιο η μέθοδος **ComputeFibonacciMDB.onMessage** κάνει τον Fibonacci υπολογισμό και μετά από αυτό κάνει καταγραφή του σε δύο πίνακες (FIB\_COMP\_BILL και FIB\_COMP\_CRM\_LOG) που βρίσκονται σε διαφορετικές ΒΔ. Αυτό προφανώς έγινε για να παρουσιάσουμε ένα global transaction στα πλαίσια ενός EJB.

Η δημιουργία των εγγραφών σε αυτούς τους πίνακες γίνεται μέσω του JPA και αντικειμένων EntityManager. Τα EntityManager που γίνονται injected σε ένα EJB είναι resources που υποστηρίζουν τα global transactions (σε αντίθεση π.χ. με ένα απλό DataSource αντικείμενο του JDBC).

Θα δούμε κατά την εκτέλεση του παραδείγματος ότι εάν αποτύχει η δημιουργία εγγραφής στον πίνακα FIB\_COMP\_CRM\_LOG θα γίνει rollback και η εγγραφή στον FIB\_COMP\_BILL. Επίσης

επειδή είναι ένα Container Managed Transaction δεν χρειάστηκε να χρησιμοποιήσουμε πουθενά μεθόδους rollback και commit κάτι που έκανε ακόμα πιο απλό τον κώδικα μας.

Εδώ πρέπει να σημειώσουμε ότι rollback θα γίνει σε οποιαδήποτε περίπτωση ένα Exception (ακόμα και RuntimeException) δεν γίνει handled μέσα στην ComputeFibonacciMDB.onMessage.

Επίσης όσον αφορά τα MDB, ισχύει ότι εάν αποτύχει το Transaction της μεθόδου onMessage τότε το JMS Message που αυτή επεξεργάσθηκε θα ξαναεπιστρέψει στο Queue και θα γίνει ξανά προσπάθεια για επεξεργασία του μέχρι να επεξεργασθεί επιτυχώς ή να ξεπεραστεί το όριο επαναληπτικών προσπαθειών που έχει ορίσει ο διαχειριστής που έχει φτιάξει το Queue. Το όριο που έχουμε ορίσει για computationalRequestsQueue του ComputeFibonacciMDB καθώς και ο χρόνος μεταξύ διαδοχικών προσπαθειών εμφανίζεται στην επόμενη εικόνα.

The screenshot shows the Oracle WebLogic Server Administration Console. In the left sidebar, under 'Domain Structure', the 'Services' section is expanded, showing 'Messaging' with sub-options like 'JMS Servers', 'Store-and-Forward Agents', 'JMS Modules', 'Path Services', and 'Bridges'. A 'How do I...' panel provides links to 'Configure queue message delivery failure options' and 'Configure JMS templates'.

The main content area displays the 'Settings for WebAppDemo/jms/queues/computationalRequestsQueue' page. The 'Delivery Failure' tab is active. It contains the following configuration:

- Redelivery Delay Override:** 1000 (ms)
- Redelivery Limit:** 2
- Expiration Policy:** Discard

Below these fields, there are descriptive notes: 'The delay, in millis, redelivered, regardless of connection to their originating destination will no consumer and/or' for Redelivery Delay, and 'The number of re-delivery attempts before error destination, message sender. If is configured, the message is dropped (deleted)' for Redelivery Limit.

Αρχικά μεταβήκαμε στο Services -> Modules -> SystemModule-0 -> computationalRequestQueue -> Configuration -> Delivery Failure και ύστερα θέσαμε το πεδίο "Redelivery Delay Override" σε 1000 milliseconds και το "Redelivery Limit" σε 2 που σημαίνει ότι εάν ένα JMS Message παρουσιάζει exception κάθε φορά κατά την επεξεργασία του, αυτή θα γίνει το πολύ 3 φορές.

Παρουσιάζουμε τώρα τις εγγραφές που δημιουργούνται μετά από μία κανονική εκτέλεση της μεθόδου για το όρισμα 27.

The screenshot shows the Oracle Database SQL Developer interface. The left sidebar lists tables: FIB\_COMP\_BILL, JDBC\_DEMO\_ACCESS\_INFO, SEQUENCE, USERS, and USERS\_GROUPS. The main area displays the FIB\_COMP\_BILL table with the following data:

ID	COST	DURATION	END_TIME	RESULT	START_TIME	TERM_IND	USERNAME
401	25	5	13/7/2017 7:56:45,580000 MM	196418	13/7/2017 7:56:45,575000 MM	27	oleg@mailinator.com
352	5	1	13/7/2017 12:22:39,620000 ΠΜ	28657	13/7/2017 12:22:39,619000 ΠΜ	23	oleg@mailinator.com
351	0	0	13/7/2017 12:10:26,385000 ΠΜ	17711	13/7/2017 12:10:26,385000 ΠΜ	22	oleg@mailinator.com
301	0	0	12/7/2017 11:33:53,728000 ΜΜ	0	12/7/2017 11:33:53,728000 ΜΜ	0	oleg@mailinator.com
254	0	0	13/7/2017 11:33:53,477000 ΜΜ	0	13/7/2017 11:33:53,477000 ΜΜ	0	oleg@mailinator.com

The screenshot shows the MySQL Workbench interface with a connection named 'root\_con'. In the Navigator pane, under the 'web\_demo.crm' schema, the 'Tables' section is expanded, showing the 'fib\_comp.crm\_log' table. A query window titled 'fib\_comp.crm\_log' displays the result of the SQL command 'SELECT \* FROM web\_demo.crm.fib\_comp.crm\_log;'. The result grid shows five rows of data:

ID	BILL_ID	COST	DURATION	END_TIME	FIRST_AND_LAST_NAME	RESULT	START_TIME	TERM_IND	USERNAME
401	401	25	5	2017-07-13 19:56:45	Oleg Sotnikov	196418	2017-07-13 19:56:45	27	oleg@mailinator.com
352	352	5	1	2017-07-13 00:22:39	Oleg Sotnikov	28657	2017-07-13 00:22:39	23	oleg@mailinator.com
351	351	0	0	2017-07-13 00:10:26	Oleg Sotnikov	17711	2017-07-13 00:10:26	22	oleg@mailinator.com
301	301	0	0	2017-07-12 23:33:53	Oleg Sotnikov	0	2017-07-12 23:33:53	0	oleg@mailinator.com

Για να παρουσιάσουμε ένα rollback θα πρέπει να προκαλέσουμε ένα σφάλμα κατά την εισαγωγή μίας εγγραφής στον πίνακα FIB\_COMP\_CRM\_LOG διότι γίνεται μετά από την εισαγωγή στον FIB\_COMP\_BILL και έτσι θα δούμε εάν πράγματι θα ακυρωθεί και η εισαγωγή που έγινε στον FIB\_COMP\_BILL.

Αυτό το επιτυγχάνουμε δημιουργώντας ένα trigger το οποίο θα εκτελείται μετά από κάθε insert στον FIB\_COMP\_CRM\_LOG και θα προκαλεί κάποιο σφάλμα. Το συγκεκριμένο εισάγει μία εγγραφή σε έναν πίνακα ο οποίος δεν υπάρχει οπότε αναγκαστικά προκαλεί σφάλμα:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `web_demo.crm`.`fib_comp.crm_log_BEFORE_INSERT` BEFORE INSERT ON `fib_comp.crm_log` FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO nonexistant_table_name (`column1`)
```

```
VALUES ('value1');
```

```
END
```

The screenshot shows the MySQL Workbench interface with a connection named 'root\_con'. In the Navigator pane, under the 'web\_demo.crm' schema, the 'Tables' section is expanded, showing the 'fib\_comp.crm\_log' table. A query window titled 'fib\_comp.crm\_log' displays the result of the SQL command 'SELECT \* FROM web\_demo.crm.fib\_comp.crm\_log;'. The result grid shows six rows of data:

ID	BILL_ID	COST	DURATION	END_TIME	FIRST_AND_LAST_NAME
401	401	25	5	2017-07-13 19:56:45	Oleg Sotnikov
352	352	5	1	2017-07-13 00:22:39	Oleg Sotnikov
351	351	0	0	2017-07-13 00:10:26	Oleg Sotnikov
301	301	0	0	2017-07-12 23:33:53	Oleg Sotnikov
251	251	0	0	2017-07-12 23:21:24	Oleg Sotnikov

Μετά από αυτό υποβάλουμε ένα αίτημα για το όρισμα 25

[Go to the Profile Page \(secured, requires authentication\)](#)

**Compute n-th fibonacci number (term index):**

This is an example of calling a message driven bean as well as an example of global transaction management.

## 25 Compute Fibonacci

## Output:

Your request is being processed, please visit this page at a later instant to view the result of your computation.

#### —Send Message to Remote System—

Το όρισμα υπολογίζεται κανονικά και με ένα refresh παίρνουμε το αποτέλεσμα που είναι 75025.

This is an example of calling a message driven bean as well as an example of global transaction.

Compute Fibonacci

### **Output:**

The result of your requested fibonacci computation with argument 25 is 75025.

Παρόλα αυτά οι εισαγωγές των αντίστοιχων εγγραφών στους πίνακες αποτυγχάνουν (δεν υπάρχει 25 στις κολώνες “TERM\_IND”):

ID	C...	DURATION	END_TIME	RESULT	START_TIME	TERM_IND	USERNAME
► 401	25	5	13/7/2017 7:56:45,580000 MM	196418	13/7/2017 7:56:45,575000 MM	27	oleg@mailinator.com
352	5	1	13/7/2017 12:22:39,620000 ПМ	28657	13/7/2017 12:22:39,619000 ПМ	23	oleg@mailinator.com
351	0	0	13/7/2017 12:10:26,385000 ПМ	17711	13/7/2017 12:10:26,385000 ПМ	22	oleg@mailinator.com
301	0	0	12/7/2017 11:33:53,728000 MM	0	12/7/2017 11:33:53,728000 MM	0	oleg@mailinator.com
251	0	0	12/7/2017 11:21:24,157000 MM	0	12/7/2017 11:21:24,157000 MM	0	oleg@mailinator.com
201	0	0	12/7/2017 11:00:24,376000 MM	5	12/7/2017 11:00:24,376000 MM	5	oleg@mailinator.com
151	0	0	12/7/2017 10:54:06,139000 MM	2	12/7/2017 10:54:06,139000 MM	3	oleg@mailinator.com
102	0	0	12/7/2017 5:25:43,328000 MM	8	12/7/2017 5:25:43,328000 MM	6	oleg@mailinator.com
101	0	0	12/7/2017 5:25:06,028000 MM	5	12/7/2017 5:25:06,028000 MM	5	oleg@mailinator.com
51	0	0	12/7/2017 5:04:51,856000 MM	28657	12/7/2017 5:04:51,856000 MM	23	oleg@mailinator.com
6	30	6	11/7/2017 11:20:46,033000 MM	121393	11/7/2017 11:20:46,027000 MM	26	oleg2@mailinator.com
4	0	0	11/7/2017 11:19:35,779000 MM	34	11/7/2017 11:19:35,779000 MM	9	oleg@mailinator.com
2	0	0	11/7/2017 11:12:22,106000 MM	2	11/7/2017 11:12:22,106000 MM	3	oleg@mailinator.com

ID	BILL_ID	COST	DURATION	END_TIME	FIRST_AND_LAST_NAME	RESULT	START_TIME	TERM_IND	USERNAME
1	2	0	0	2017-07-11 23:12:22	Oleg Sotnikov	2	2017-07-11 23:12:22	3	oleg@mailinator.com
2	4	0	0	2017-07-11 23:19:35	Oleg Sotnikov	34	2017-07-11 23:19:35	9	oleg@mailinator.com
3	6	30	6	2017-07-11 23:20:46	Oleg2 Sotnikov2	121393	2017-07-11 23:20:46	26	oleg2@mailinator.com
51	51	0	0	2017-07-12 17:04:51	Oleg Sotnikov	28657	2017-07-12 17:04:51	23	oleg@mailinator.com
101	101	0	0	2017-07-12 17:25:06	Oleg Sotnikov	5	2017-07-12 17:25:06	5	oleg@mailinator.com
102	102	0	0	2017-07-12 17:25:43	Oleg Sotnikov	8	2017-07-12 17:25:43	6	oleg@mailinator.com
151	151	0	0	2017-07-12 22:54:06	Oleg Sotnikov	2	2017-07-12 22:54:06	3	oleg@mailinator.com
201	201	0	0	2017-07-12 23:00:24	Oleg Sotnikov	5	2017-07-12 23:00:24	5	oleg@mailinator.com
251	251	0	0	2017-07-12 23:21:24	Oleg Sotnikov	0	2017-07-12 23:21:24	0	oleg@mailinator.com
301	301	0	0	2017-07-12 23:33:53	Oleg Sotnikov	0	2017-07-12 23:33:53	0	oleg@mailinator.com
351	351	0	0	2017-07-13 00:10:26	Oleg Sotnikov	17711	2017-07-13 00:10:26	22	oleg@mailinator.com
352	352	5	1	2017-07-13 00:22:39	Oleg Sotnikov	28657	2017-07-13 00:22:39	23	oleg@mailinator.com
401	401	25	5	2017-07-13 19:56:45	Oleg Sotnikov	196418	2017-07-13 19:56:45	27	oleg@mailinator.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Η εμφάνιση του αποτελέσματος του υπολογισμού στον χρήστη γίνεται προφανώς επειδή δεν έχουμε κάνει κάποια υλοποίηση για να ακυρώνεται η εισαγωγή ενός επιτυχούς υπολογισμού στην εσωτερική δομή του FibonacciComputationsStoreSingleton που τους αποθηκεύει. Από την άλλη όμως γίνεται rollback από τους EntityManagers **webDemoEm** και **webDemoCrmEm** που ήταν και ο στόχος μας. Προφανώς σε μία πραγματική εφαρμογή μάλλον θα απαιτούνταν και η ακύρωση της αποθήκευσης του υπολογισμού ή τουλάχιστον της εμφάνισής του στον χρήστη μέχρι να δημιουργηθούν οι αντίστοιχες εγγραφές.

Για να εξετάσουμε και την επαναληπτική υποβολή του JMS Message προς το ComputeFibonacciMDB θα ανατρέξουμε στην κονσόλα της εφαρμογής όπου θα διαπιστώσουμε ότι γίνονται 3 προσπάθειες για επεξεργασία του, όπως και είχαμε ορίσει μέσω της διαχειριστικής κονσόλας του Weblogic Server.

### 1<sup>η</sup> προσπάθεια:

```
... more
ComputeFibonacciServlet.doGet ... started, termInd: 25
ComputeFibonacciServlet.doGet: jsonResponseString:
{"status":"SUCCESS","errorMsg":null,"data":{"compState":"UNDER_PROCESSING","termInd":null,"result":null,"key":null}}
ComputeFibonacciMDB onMessage started at: Thu Jul 13 20:30:53 EEST 2017
JMSMessageId: ID:<523257.1499967053301.0>
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: STARTED
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: COMPUTED
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: CREATED_BILL
<Ιούλ 13, 2017, 8:30:53,330 μμ EEST> <Warning> <EclipseLink> <BEA-2005000> <2017-07-13 20:30:53.33--UnitOfWork(1254762334)--Thread(Thread[[ACTIVE]
ExecuteThread: '11' for queue: 'weblogic.kernel.Default (self-tuning)'.5_Pooled Threads])--Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.6.4.v20160829-44060b6): org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'web_demo_crm.nonexistent_table_name' doesn't exist
Error Code: 1146
Call: INSERT INTO FIB_COMP_CRM_LOG (ID, BILL_ID, COST, DURATION, END_TIME, FIRST_AND_LAST_NAME, RESULT, START_TIME, TERM_IND, USERNAME) VALUES (?, ?, ?, ?, ?, ?, ?)
bind => [451, 451, 5, 1, 2017-07-13 20:30:53.314, Oleg Sotnikov, 75025, 2017-07-13 20:30:53.313, 25, oleg@mailinator.com]
Query: InsertObjectQuery(oosotnikov.demowebapp.services.fibonacci.entity.FibCompCrmLog@53852743)>
java.persistence.PersistenceException: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.6.4.v20160829-44060b6):
org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'web_demo_crm.nonexistent_table_name' doesn't exist
Error Code: 1146
```

### 2<sup>η</sup> προσπάθεια:

```
ComputeFibonacciMDB onMessage started at: Thu Jul 13 20:30:54 EEST 2017
JMSMessageId: ID:<523257.1499967053301.0>
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: STARTED
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: COMPUTED
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: CREATED_BILL
<Τουλ 13. 2017. 8:30:54.390 μμ EEST> <Warning> <EclipseLink> <BEA-2005000> <2017-07-13 20:30:
```

### 3<sup>η</sup> προσπάθεια:

```
... more
ComputeFibonacciMDB onMessage started at: Thu Jul 13 20:30:55 EEST 2017
JMSMessageId: ID:<523257.1499967053301.0>
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: STARTED
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: COMPUTED
ComputeFibonacciMDB: JMSMessageId: ID:<523257.1499967053301.0>, FibCompStep: CREATED_BILL
<Ιούλ 13, 2017, 8:30:55,430 μμ EEST> <Warning> <EclipseLink> <BEA-2005000> <2017-07-13 20:30:55.
```

## Συμπεριφορά CMT κατά την Αποστολή JMS Messages

Για να ολοκληρώσουμε το κεφάλαιο θα αναφέρουμε επίσης ότι η συμπεριφορά του αυτόματου rollback σε ένα CMT κατά την αποστολή ενός JMS Message (όπως στην περίπτωση της μεθόδου `AsncCmpStrSingToMDB.requestAsyncComputation`) είναι η **ακύρωση της αποστολής του μηνύματος**.

Με λίγα λόγια εάν ακόμα και μετά την αποστολή του μηνύματος και το close των resources παρουσιαστεί ένα exception η αποστολή του θα ακυρωθεί. Αυτό μπορεί να διαπιστωθεί εύκολα εάν στο τέλος του finally block της `AsncCmpStrSingToMDB.requestAsyncComputation` προσθέσουμε τον κώδικα `if(true){throw new RuntimeException();};`.

Αυτή η συμπεριφορά μπορεί να μην είναι πάντα επιθυμητή στα πλαίσια μίας μεθόδου η οποία το μόνο που κάνει είναι να στέλνει ένα JMS Message αλλά μπορούμε να φανταστούμε την πρακτική της χρησιμότητα στις περιπτώσεις που θέλουμε να γίνει αποστολή του μηνύματος μόνο εφόσον γίνει καταγραφή αυτής στην ΒΔ.

# Security

## Σύντομη Περιγραφή

Όσον αφορά το Security ενός enterprise application η Java EE μπορεί να καλύψει τις ανάγκες για **Authentication**, **Access Control** και **Confidentiality**.

Αυτό επιτυγχάνεται με τον ορισμό **ενός Security Realm**, **ενός Authentication Scheme (login mechanism)**, των **Πόρων (Web Resources)** που θέλουμε να προστατεύσουμε και στην συνέχεια τον ορισμό **Περιορισμών Ασφάλειας (Security Constraints)** επί αυτών.

Οι περιορισμοί ασφάλειας δηλώνουν **Ρόλους (Roles)** στους οποίους κάποιος χρήστης πρέπει να ανήκει για να προσπελάσει έναν πόρο και επίσης ορίζουν τους πόρους οι οποίοι πρέπει να προσπελάζονται μέσω πρωτοκόλλου το οποίο εξασφαλίζει **Εμπιστευτικότητα (SSL)**.

## Ορισμός των Περιορισμών Ασφάλειας στον Deployment Descriptor

Στην επόμενη εικόνα παρουσιάζεται το κομμάτι του **deployment descriptor (web.xml)** όπου ορίζονται οι ρυθμίσεις ασφάλειας.

```
213<login-config>
214    <auth-method>FORM</auth-method>
215    <form-login-config>
216        <form-login-page>/secure/login.jsp</form-login-page>
217        <form-error-page>/secure/login.jsp?auth-error=1</form-error-page>
218    </form-login-config>
219</login-config>
220
221<security-constraint>
222    <web-resource-collection>
223        <web-resource-name>Https_Resources</web-resource-name>
224        <url-pattern>/*</url-pattern>
225    </web-resource-collection>
226    <user-data-constraint>
227        <description>highest supported transport security level</description>
228        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
229    </user-data-constraint>
230</security-constraint>
231<security-constraint>
232    <web-resource-collection>
233        <web-resource-name>Secured and Access Controlled Content</web-resource-name>
234        <url-pattern>/secure/authenticated/*</url-pattern>
235    </web-resource-collection>
236    <auth-constraint>
237        <role-name>ADMINISTRATOR</role-name>
238        <role-name>USER</role-name>
239    </auth-constraint>
240</security-constraint>
241<security-role>
242    <role-name>ADMINISTRATOR</role-name>
243</security-role>
244<security-role>
245    <role-name>USER</role-name>
246</security-role>
```

```

247<security-role>
248    <role-name>DEFAULT</role-name>
249</security-role>
250<session-config>
251    <session-timeout>5</session-timeout>
252<cookie-config>
253    <name>WLS_WEB_DEMO_SESSIONID</name>
254</cookie-config>
255</session-config>
256</web-app>

```

## Container Managed Authentication στα πλαίσια του Weblogic Server 12C με τα Credentials των Χρηστών να Αποθηκεύονται στην ΒΔ της Εφαρμογής

Το Container Managed Authentication αναθέτει μεγάλο μέρος της ευθύνης για την ταυτοποίηση των χρηστών των εφαρμογών στον Servlet Container.

Γενικά το πώς ακριβώς θα το κάνει αυτό ένας Servlet Container είναι παραμετροποιήσιμο από τους διαχειριστές του και εξαρτάται από το τι δυνατότητες παραμετροποίησης τους προσφέρει. Μπορεί να ορίσουμε όλοι οι χρήστες να αποθηκεύονται σε αρχεία, στην ΒΔ ή και να προέρχονται από ένα τρίτο σύστημα και τα passwords τους να αποθηκεύονται σε κρυπτογραφημένη ή μη κρυπτογραφημένη μορφή.

**Υπάρχουν πολλές δυνατότητες και ο Weblogic Server 12C προσφέρει ένα πλήρως παραμετροποιήσιμο σύστημα για αυτές τις ανάγκες αλλά εμείς θα εξετάσουμε μόνο την περίπτωση όπου τα credentials των χρηστών θέλουμε να αποθηκεύονται σε μία ΒΔ.**

Γενικά για να επιτύχουμε την ταυτοποίηση των χρηστών στα πλαίσια του Container Managed Authentication θα πρέπει να ορίσουμε ρόλους (**roles**) και ομάδες (**groups**) χρηστών.

### Ορισμός των Ρόλων

Οι ρόλοι ορίζονται στον deployment descriptor που παρουσιάσαμε στην αρχή του κεφαλαίου (γραμμές 241-246). Ορίζονται με τα elements “**<security-role>**”. Οι ρόλοι που ορίστηκαν για την DemoWebApp είναι οι **USER**, **ADMIN** και **DEFAULT**. Ορίστηκαν τρεις ρόλοι για να φανεί καλύτερα ο τρόπος αντιστοίχισης μεταξύ ρόλων και χρηστών, πρακτικά όμως οποιοσδήποτε χρήστης κάνει registration στην DemoWebApp εφαρμογή θα αποκτάει και τους τρεις ρόλους.

### Αντιστοίχιση Ρόλων και Ομάδων Χρηστών (ή Μεμονωμένων Χρηστών)

Στην συνέχεια σε γενικές γραμμές για οποιαδήποτε υλοποίηση ενός Servlet Container για τις ανάγκες του Container Managed Authentication ισχύει ότι θα πρέπει να οριστεί μία αντιστοιχία μεταξύ των ρόλων (**roles**) και των ομάδων (**groups**) των χρηστών. Προφανώς θα πρέπει τελικώς να οριστεί και η αντιστοιχία μεταξύ των χρηστών και των ομάδων στις οποίες ανήκουν, αλλά αυτό θα το εξετάσουμε αμέσως μετά.

Παρουσιάζεται το πώς γίνεται η αντιστοιχία ρόλων και ομάδων στον Runtime Descriptor “**weblogic.xml**” του Oracle Weblogic Server 12C.

```

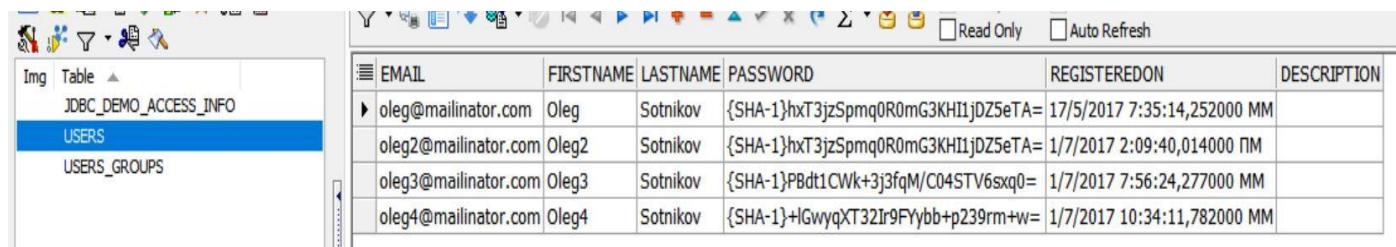
1 <?xml version="1.0" encoding="UTF-8"?>
2<wls:weblogic-web-app xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-app http://xmlns.oracle.com/weblogic/weblogic-web-app.xsd">
3   <wls:weblogic-version>12.1.3</wls:weblogic-version>
4   <wls:context-root>DemoWebApp</wls:context-root>
5
6   <wls:security-role-assignment>
7     <wls:role-name>ADMINISTRATOR</wls:role-name>
8     <wls:principal-name>ADMINISTRATOR</wls:principal-name>
9   </wls:security-role-assignment>
10
11  <wls:security-role-assignment>
12    <wls:role-name>USER</wls:role-name>
13    <wls:principal-name>USER</wls:principal-name>
14  </wls:security-role-assignment>
15
16  <wls:security-role-assignment>
17    <wls:role-name>DEFAULT</wls:role-name>
18    <wls:principal-name>DEFAULT</wls:principal-name>
19  </wls:security-role-assignment>
20
21  <wls:container-descriptor>
22    <wls:servlet-reload-check-secs>1</wls:servlet-reload-check-secs>
23  </wls:container-descriptor>
24
25  <wls:jsp-descriptor>
26    <wls:page-check-seconds>1</wls:page-check-seconds>
27  </wls:jsp-descriptor>
28
29 </wls:weblogic-web-app>

```

Όπως παρατηρούμε δεν χρειάζεται να ορίσουμε τα Groups όπως ορίσαμε προηγουμένως τα Roles. Χρειάζεται μόνο να ορίσουμε την αντιστοιχία μεταξύ τους. Αυτό γίνεται με τα elements “**<wls:security-role-assignment>**”. Με το element “**<wls:role-name>**” δηλώνουμε τον ρόλο ενώ με το element “**<wls:principal-name>**” το αναγνωριστικό μίας ομάδας ή ενός χρήστη στον οποίον θα αντιστοιχεί αυτός ο ρόλος (προφανώς μπορούμε να αντιστοιχίσουμε έναν ρόλο σε περισσότερα από ένα). Επίσης ενώ είναι δυνατόν να ορίσουμε αντιστοίχιση για μεμονωμένους χρήστες προφανώς κάτι τέτοιο δεν είναι πρακτικό να γίνει σε αυτό το σημείο και θα γίνει μέσω της ΒΔ έμμεσα μέσω των ομάδων.

### Αντιστοίχιση των Ομάδων με τους Χρήστες

Εδώ θα παρουσιάσουμε τον πίνακα της ΒΔ ο οποίος αποθηκεύει τα credentials των χρηστών του DemoWebApp (**USERS**) καθώς και τον πίνακα που αποθηκεύει την αντιστοίχιση των χρηστών με τις ομάδες στις οποίες ανήκουν (**USERS\_GROUPS**).



The screenshot shows the Oracle Database SQL Developer interface. On the left, there's a tree view with nodes: JDBC\_DEMO\_ACCESS\_INFO, USERS, and USERS\_GROUPS. The USERS node is selected. On the right, there's a large table grid with the following columns: EMAIL, FIRSTNAME, LASTNAME, PASSWORD, REGISTEREDON, and DESCRIPTION. The data is as follows:

EMAIL	FIRSTNAME	LASTNAME	PASSWORD	REGISTEREDON	DESCRIPTION
oleg@mailinator.com	Oleg	Sotnikov	{SHA-1}hxT3jzSpmq0R0mG3KHI1jDZ5eTA=	17/5/2017 7:35:14,252000 MM	
oleg2@mailinator.com	Oleg2	Sotnikov	{SHA-1}hxT3jzSpmq0R0mG3KHI1jDZ5eTA=	1/7/2017 2:09:40,014000 ΠΜ	
oleg3@mailinator.com	Oleg3	Sotnikov	{SHA-1}PBdt1CWk+3j3fqM/C04STV6sxq0=	1/7/2017 7:56:24,277000 MM	
oleg4@mailinator.com	Oleg4	Sotnikov	{SHA-1}+GwyqXT32Ir9FYybb+p239rm+w=	1/7/2017 10:34:11,782000 MM	

The screenshot shows the Oracle SQL Developer interface. On the left, there's a tree view with nodes: 'Img', 'Table ▾', 'JDBC\_DEMO\_ACCESS\_INFO', 'USERS', and 'USERS\_GROUPS'. The 'USERS\_GROUPS' node is selected. On the right, a table is displayed with the following columns: EMAIL, GROUPNAME, and DESCRIPTION. The data consists of 12 rows, each representing a user and their group assignments:

	EMAIL	GROUPNAME	DESCRIPTION
▶	oleg@mailinator.com	ADMINISTRATOR	
	oleg@mailinator.com	USER	
	oleg@mailinator.com	DEFAULT	
	oleg2@mailinator.com	ADMINISTRATOR	
	oleg2@mailinator.com	USER	
	oleg2@mailinator.com	DEFAULT	
	oleg3@mailinator.com	ADMINISTRATOR	
	oleg3@mailinator.com	USER	
	oleg3@mailinator.com	DEFAULT	
	oleg4@mailinator.com	ADMINISTRATOR	
	oleg4@mailinator.com	USER	
	oleg4@mailinator.com	DEFAULT	

Όπως βλέπουμε ο USERS\_GROUPS αποθηκεύει μία ένα προς πολλά αντιστοιχία μεταξύ των users και των groups όπου ο κάθε user ανήκει και στα τρία groups που ορίσαμε.

Τώρα μας μένει να δούμε τον τρόπο με τον οποίον θα ενημερώσουμε τον Weblogic Server για την ύπαρξη αυτών των πινάκων.

### Pύθιση του Security Realm και του Authentication Provider στον Weblogic

#### Server 12C

**Θα πρέπει να ενημερώσουμε τον Weblogic Server 12C για την αντιστοίχιση των ομάδων με τους χρήστες καθώς και τον πίνακα ο οποίος αποθηκεύει τα credentials τους, για να ολοκληρώσουμε τον μηχανισμό αυθεντικοποίησης.**

Αυτό αρχικά απαιτεί την ύπαρξη ενός **Security Realm**. Επειδή η δημιουργία ενός νέου Security Realm είναι πολύπλοκη και επειδή πολύ σπάνια χρειαζόμαστε πάνω από ένα security realm θα χρησιμοποιήσουμε ένα ήδη υπάρχων (το “**myrealm**”) που παρέχεται έτοιμο με την εγκατάσταση ενός καινούριου domain.

**Change Center**

**View changes and restarts**

Click the Lock & Edit button to modify, add or delete items in this domain.

**Lock & Edit**

**Release Configuration**

**Domain Structure**

- demoWebApp
  - + Environment
  - Deployments
  - + Services
  - Security Realms**
  - + Interoperability
  - + Diagnostics

**Summary of Security Realms**

A security realm is a container for the mechanisms--including users, groups, security roles, and policies--that define security for a WebLogic Server domain. It can contain multiple security realms in a WebLogic Server domain, but only one can be set as the default realm.

This Security Realms page lists each security realm that has been configured in this WebLogic Server domain.

**Customize this table**

**Realms (Filtered - More Columns Exist)**

Click the **Lock & Edit** button in the Change Center to activate all the buttons on this page.

	<b>Name</b>	<b>Default Realm</b>
<input type="checkbox"/>	myrealm	true

**New** **Delete**

Στην συνέχεια θα πρέπει να δημιουργήσουμε έναν **Authentication Provider**. Για να το κάνουμε αυτό, αφού πατήσουμε πάνω στο “myrealm”, μεταφερόμαστε στο tab “**Providers**” και στην συνέχεια “**Authentication**”.

**Change Center**

**View changes and restarts**

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

**Lock & Edit**

**Release Configuration**

**Domain Structure**

- demoWebApp
  - + Domain Partitions
  - + Environment
  - Deployments
  - + Services
  - Security Realms
  - + Interoperability
  - + Diagnostics

**Welcome, weblogic** Connected to: demoWebApp

**Home > Credential Mappings > Providers > WEB\_DEMO\_SQLAuthenticator > Summary of Environment > Summary of Services > Summary of Security Realms > myrealm > Providers**

**Settings for myrealm**

**Providers**

**Authentication** **Password Validation** **Authorization** **Adjudication** **Role Mapping** **Auditing** **Credential Mapping** **Certification Path**

An Authentication provider allows WebLogic Server to establish trust by validating a user. You must have one Authentication provider in a security realm, and you can configure multiple Authentication providers in a security realm. Different types of Authentication providers are designed to access different data stores, such as LDAP servers or DBMS.

**Customize this table**

**Authentication Providers**

<b>New</b>	<b>Delete</b>	<b>Reorder</b>	<b>Name</b>	<b>Description</b>	<b>Version</b>
			DefaultAuthenticator	WebLogic Authentication Provider	1.0
			DefaultIdentityAssertion	WebLogic Identity Assertion provider	1.0

**New** **Delete** **Reorder**

Εδώ θα πρέπει να πατήσουμε το “**Lock & Edit**” και στην συνέχεια το κουμπί “**New**”.

The screenshot shows the Oracle WebLogic Server Administration Console interface. On the left, there's a sidebar titled "Domain Structure" with a tree view of the domain structure, including "demoWebApp", "Environment", "Deployments", "Services", "Security Realms", "Interoperability", and "Diagnostics". On the right, a central panel is titled "Create a New Authentication Provider". It contains fields for "Name" (set to "WEB\_DEMO\_SQLAuthenti") and "Type" (set to "SQLAuthenticator"). At the bottom of this panel are "OK" and "Cancel" buttons. The top navigation bar includes links for Home, Log Out, Preferences, Record, Help, and a search bar.

Ορίζουμε το όνομα που θέλουμε (ορίσαμε το “**WEB\_DEMO\_SQLAuthenticator**”) και ως τύπο, για έναν authenticator που προορίζεται να βασίζεται σε δεδομένα μίας ΒΔ, θα πρέπει να ορίσουμε το “**SQLAuthenticator**”.

Στην συνέχεια αφού πατήσουμε πάνω στο “**WEB\_DEMO\_SQLAuthenticator**” (στο μενού “**Providers**” του “**myrealm**”) πατάμε το “**Configuration**” και από κάτω του το “**Provider Specific**”.

### Settings for WEB\_DEMO\_SQLAuthenticator

**Configuration** Performance

Common Provider Specific

Click the **Lock & Edit** button in the Change Center to modify the settings on this page.

Save

Use this page to define the provider specific configuration of this SQL Authentication provider.

<b>Identity Domain:</b>		The name of the identity domain.
<input type="checkbox"/> <b>Plaintext Passwords Enabled</b>		Specifies whether plaintext passwords are enabled.
<b>Data Source Name:</b>	WEB_DEMO	The name of the JDBC connection.
<b>Group Membership Searching:</b>	unlimited ▾	Specifies whether recursive group membership searching is limited. Valid values are unlimited or limited.
<b>Max Group Membership Search Level:</b>	0	This specifies how many levels of recursive searching are performed. This setting is valid only if Group Membership Searching is enabled. If the value is 0 and positive, no recursive searching will be performed. If the value is negative, searching will go down to the bottom level. <a href="#">More Info..</a>
<input checked="" type="checkbox"/> <b>Password Style Retained</b>		Controls how a password is retained after a user's password is changed. <a href="#">More Info..</a>
<b>Password Algorithm:</b>	SHA-1	The message digest algorithm used. SHA-1 is a standard algorithm defined by the Cryptography Extension API. Cryptography Architecture specifications. <a href="#">More Info..</a>
<b>Password Style:</b>	HASHED ▾	Indicates the password style used. HASHED indicates that passwords are created and stored in hashed form. RETAINED indicates that the password is retained in its original form.
<b>SQL Get User Password:</b>	SELECT PASSWORD FRC	The SQL statement used to get a user's password. This requires a single parameter, the user ID, containing at most a single record.
<b>SQL Set User Password:</b>	UPDATE USERS SET PASS = ?	The SQL statement used to set a user's password. This requires two parameters, the user ID and the new password.
<b>SQL User Exists:</b>	SELECT EMAIL FROM USERS WHERE EMAIL = ?	The SQL statement used to check if a user exists. This requires a single parameter for the email address, returning at most a single record.
<b>SQL List Users:</b>	SELECT EMAIL FROM USERS	The SQL statement used to list all users. This statement searches for all users and returns all records.
<b>SQL Create User:</b>	INSERT INTO USERS(EMAIL, PWD) VALUES (?, ?)	The SQL statement used to create a new user. The requirements depend on the database system. A minimum of two parameters are required. Descriptions Supported. SQL is based on the default syntax. Create User must be up to date with supported changes. <a href="#">More Info..</a>
<b>SQL Remove User:</b>	DELETE FROM USERS WHERE EMAIL = ?	The SQL statement used to remove a user. This requires a single parameter, the user ID.
<b>SQL List Groups:</b>	SELECT GROUPNAME FROM GROUPS	The SQL statement used to list all groups. This statement requires no parameters.

<b>SQL Group Exists:</b>	<code>SELECT GROUPNAME FR</code>	The SQL statement used to look up a group name. The statement requires a single parameter for the group name and returns a resultSet containing the group name if found.
<b>SQL Create Group:</b>	<code>INSERT INTO USERS_GR</code>	The SQL statement used to create a new group. The statement requires one parameter: the group's name. If the group already exists, the statement fails. Requirements depend on the value of Descriptions Supported. If Descriptions Supported is enabled, the group's description is required. If Descriptions Supported is disabled, the default setting of Descriptions Supported is used. If Descriptions Supported is not specified, it is updated manually if the setting of Descriptions Supported is changed.
<b>SQL Remove Group:</b>	<code>DELETE FROM USERS_G</code>	The SQL statement used to remove a group. The statement requires a single parameter: the group name. If the group does not exist, the statement fails. Requirements depend on the value of Descriptions Supported. If Descriptions Supported is enabled, the group's description is required. If Descriptions Supported is disabled, the default setting of Descriptions Supported is used. If Descriptions Supported is not specified, it is updated manually if the setting of Descriptions Supported is changed.
<b>SQL Is Member:</b>	<code>SELECT EMAIL FROM US</code>	The SQL statement used to look up whether a user is a member of a group. The statement requires two parameters: a user name and a group name. It returns a resultSet containing the user's email address if the user is a member of the group.
<b>SQL List Member Groups:</b>	<code>SELECT GROUPNAME FR</code>	The SQL statement used to look up the groups that a user is a member of. The SQL statement requires a single parameter: the user's name. It returns a resultSet containing the group names that the user is a member of.
<b>SQL List Group Members:</b>	<code>SELECT EMAIL FROM US</code>	The SQL statement used to list groups that contain a specific member. The statement requires two parameters: a group name and a wildcarded member name. The SQL statement returns a resultSet containing the user's email address if the user is a member of the group.
<b>SQL Remove Group Memberships:</b>	<code>DELETE FROM USERS_G</code>	The SQL statement used to delete a group membership from all groups to which it belongs. The SQL statement requires two parameters: both refer to the group being removed. Both parameters refer to the group being removed.
<b>SQL Add Member To Group:</b>	<code>INSERT INTO USERS_GR</code>	The SQL statement used to add a specific user to a group. The statement requires two parameters: the group name and the user name being added.
<b>SQL Remove Member From Group:</b>	<code>DELETE FROM USERS_G</code>	The SQL statement used to remove a specific user from a group. The statement requires two parameters: the group name and the user name being deleted from the group.
<b>SQL Remove Group Member:</b>	<code>DELETE FROM USERS_G</code>	The SQL statement used to remove a specific user from a group. The statement requires a single parameter: the user name being removed.
<input checked="" type="checkbox"/> <b>Descriptions Supported</b>		Indicates whether user and group descriptions are supported by the authentication provider.
<b>SQL Get User Description:</b>	<code>SELECT DESCRIPTION FR</code>	The SQL statement used to retrieve a user's description. The statement requires a single parameter for the username. It returns a resultSet containing the user's description if the Descriptions Supported attribute is enabled.
<b>SQLSet User Description:</b>	<code>UPDATE USERS SET DES</code>	The SQL statement used to set a user's description. The statement requires a single parameter for the username and a description. It updates the user's description if the Descriptions Supported attribute is enabled.
<b>SQL Get Group Description:</b>	<code>SELECT DESCRIPTION FR</code>	The SQL statement used to retrieve a group's description. The statement requires a single parameter for the group name. It returns a resultSet containing the group's description if the Descriptions Supported attribute is enabled.
<b>SQL Set Group Description:</b>	<code>UPDATE USERS_GROUPSET DES</code>	The SQL statement used to set a group's description. The statement requires a single parameter for the group name and a description. It updates the group's description if the Descriptions Supported attribute is enabled.

Σε αυτήν την σελίδα θα πρέπει να ορίσουμε το **Data Source** το οποίο θα χρησιμοποιηθεί για την επικοινωνία με την ΒΔ καθώς και τις κατάλληλες τιμές στα πεδία **“Plaintext Passwords Enabled” (false)**, **“Password Algorithm” (SHA-1)**, **“Password Style” (HASHED)** και **“Password Style Retained” (true)**. Το “Password Style Retained” ορίζεται ανάλογα με την προτίμησή μας (δεν έχει αντίκτυπο εάν δεν αποφασίσουμε να αλλάξουμε κάποια στιγμή το Password Style).

Επειδή δεν θέλαμε τα passwords των χρηστών να αποθηκεύονται στην βάση μας αλλά μόνο τα hashed values τους, ορίσαμε ως “Password Algorithm” το “SHA-1” και το “Password Style” σε “HASHED”. Κάνοντας κάτι τέτοιο θα πρέπει παράλληλα να εξασφαλίζουμε ότι η Java εφαρμογή

μας όντως θα αποθηκεύει τα passwords κρυπτογραφημένα, συν τοις άλλοις στο τελικό κρυπτογραφημένο αποτέλεσμα θα πρέπει να εφαρμόζεται BASE64 Encoding. Στην περίπτωσή μας αυτό επιτυγχάνεται στην **UserManagement.validateTweakAndRegisterUser**. Αρχικά το String του password μετατρέπεται σε πίνακα από bytes. Τα bytes αυτού του πίνακα κρυπτογραφούνται από το κατάλληλο instance της κλάσης MessageDigest. Τελικά το αποτέλεσμα της κρυπτογράφησης κωδικοποιείται με BASE64 Encoding από τον **sun.misc.BASE64Encoder**. Το τελικό αποτέλεσμα, μαζί με ένα πρόθεμα που δηλώνει τον αλγόριθμο κρυπτογράφησης σε άγκυστρα (στην περίπτωσή μας το “**{SHA-1}**”) αποθηκεύεται στην βάση.

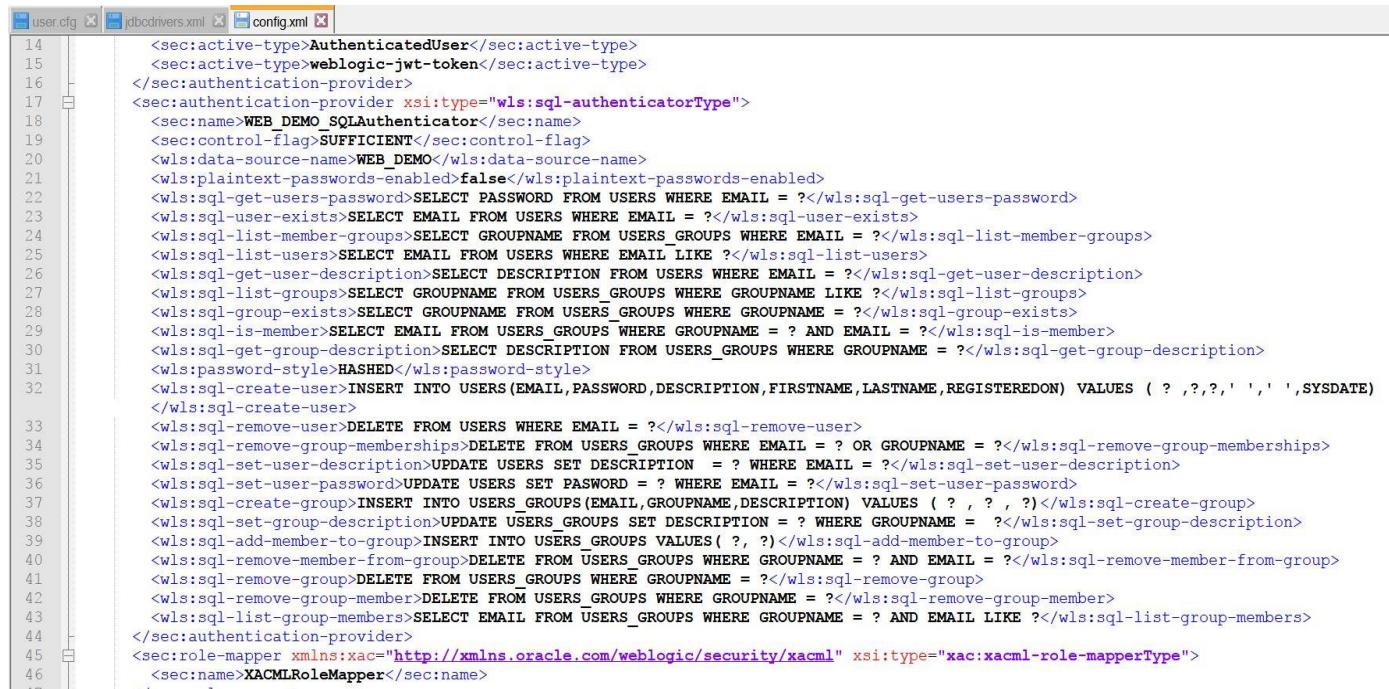
Παρόλα αυτά, όπως διαπιστώνουμε και από τον κώδικα της

**UserManagement.loginAndFetchUser** δεν θα χρειαστεί να κρυπτογραφήσουμε το password που εισάγει ο χρήστης κατά το login. Αυτό, καθώς και η σύγκρισή του με την τιμή που είναι αποθηκευμένη στην ΒΔ θα γίνει αυτόματα από τον Weblogic Server.

Όλα τα υπόλοιπα πεδία ορίζουν τα SQL Statements για να μπορεί ο **WEB\_DEMO\_SQLAuthenticator** να ξέρει με βάση ποιο SQL Query να βρει τα credentials των χρηστών για να τα συγκρίνει με αυτά που δίνονται κατά το login, καθώς επίσης και με βάση ποιο SQL Query να βρει τους ρόλους στους οποίους ανήκει ένας χρήστης.

Οι τιμές όλων των πεδίων μετά την αποθήκευσή τους (**“Save”**) και την ενεργοποίηση των αλλαγών αποθηκεύονται σε xml αρχείο (**config.xml**). Αυτό το αρχείο βρίσκεται στο path **“C:\Oracle\Middleware\Oracle\_Home2\user\_projects\domains\demoWebApp\config”**.

Παρουσιάζεται το τμήμα του που αναφέρει τα SQL Statements που έχουν εισαχθεί στα αντίστοιχα πεδία.



```

14      <sec:active-type>AuthenticatedUser</sec:active-type>
15      <sec:active-type>weblogic-jwt-token</sec:active-type>
16    </sec:authentication-provider>
17    <sec:authentication-provider xsi:type="wls:sql-authenticatorType">
18      <sec:name>WEB_DEMO_SQLAuthenticator</sec:name>
19      <sec:control-flag>SUFFICIENT</sec:control-flag>
20      <wls:data-source-name>WEB_DEMO</wls:data-source-name>
21      <wls:plaintext-passwords-enabled>false</wls:plaintext-passwords-enabled>
22      <wls:sql-get-users-password>SELECT PASSWORD FROM USERS WHERE EMAIL = ?</wls:sql-get-users-password>
23      <wls:sql-user-exists>SELECT EMAIL FROM USERS WHERE EMAIL = ?</wls:sql-user-exists>
24      <wls:sql-list-member-groups>SELECT GROUPNAME FROM USERS_GROUPS WHERE EMAIL = ?</wls:sql-list-member-groups>
25      <wls:sql-list-users>SELECT EMAIL FROM USERS WHERE EMAIL LIKE ?</wls:sql-list-users>
26      <wls:sql-get-user-description>SELECT DESCRIPTION FROM USERS WHERE EMAIL = ?</wls:sql-get-user-description>
27      <wls:sql-list-groups>SELECT GROUPNAME FROM USERS_GROUPS WHERE GROUPNAME LIKE ?</wls:sql-list-groups>
28      <wls:sql-group-exists>SELECT GROUPNAME FROM USERS_GROUPS WHERE GROUPNAME = ?</wls:sql-group-exists>
29      <wls:sql-is-member>SELECT EMAIL FROM USERS_GROUPS WHERE GROUPNAME = ? AND EMAIL = ?</wls:sql-is-member>
30      <wls:sql-get-group-description>SELECT DESCRIPTION FROM USERS_GROUPS WHERE GROUPNAME = ?</wls:sql-get-group-description>
31      <wls:password-style>HASHED</wls:password-style>
32      <wls:sql-create-user>INSERT INTO USERS (EMAIL,PASSWORD,DESCRIPTION,FIRSTNAME,LASTNAME,REGISTEREDON) VALUES ( ?,?,?,' ',' ',SYSDATE)</wls:sql-create-user>
33      <wls:sql-remove-user>DELETE FROM USERS WHERE EMAIL = ?</wls:sql-remove-user>
34      <wls:sql-remove-group-memberships>DELETE FROM USERS_GROUPS WHERE EMAIL = ? OR GROUPNAME = ?</wls:sql-remove-group-memberships>
35      <wls:sql-set-user-description>UPDATE USERS SET DESCRIPTION = ? WHERE EMAIL = ?</wls:sql-set-user-description>
36      <wls:sql-set-user-password>UPDATE USERS SET PASSWORD = ? WHERE EMAIL = ?</wls:sql-set-user-password>
37      <wls:sql-create-group>INSERT INTO USERS_GROUPS (EMAIL,GROUPNAME,DESCRIPTION) VALUES ( ?, ?, ?)</wls:sql-create-group>
38      <wls:sql-set-group-description>UPDATE USERS_GROUPS SET DESCRIPTION = ? WHERE GROUPNAME = ?</wls:sql-set-group-description>
39      <wls:sql-add-member-to-group>INSERT INTO USERS_GROUPS VALUES ( ?, ?)</wls:sql-add-member-to-group>
40      <wls:sql-remove-member-from-group>DELETE FROM USERS_GROUPS WHERE GROUPNAME = ? AND EMAIL = ?</wls:sql-remove-member-from-group>
41      <wls:sql-remove-group>DELETE FROM USERS_GROUPS WHERE GROUPNAME = ?</wls:sql-remove-group>
42      <wls:sql-remove-group-member>DELETE FROM USERS_GROUPS WHERE GROUPNAME = ?</wls:sql-remove-group-member>
43      <wls:sql-list-group-members>SELECT EMAIL FROM USERS_GROUPS WHERE GROUPNAME = ? AND EMAIL LIKE ?</wls:sql-list-group-members>
44    </sec:authentication-provider>
45    <sec:role-mapper xmlns:xac="http://xmlns.oracle.com/weblogic/security/xacml" xsi:type="xac:xacml-role-mapperType">
46      <sec:name>XACMLRoleMapper</sec:name>

```

Αφού τελειώσουμε και με τον ορισμό των SQL Provider Specific ρυθμίσεων θα πρέπει να αλλάξουμε την τιμή στο πεδίο **“Control Flag”** στο tab **“Common”** (κάτω από το tab **“Configuration”**) σε **“SUFFICIENT”**. Αυτό γίνεται διότι υπάρχουν και άλλοι Authenticators και πρέπει να ορίσουμε ότι η ταυτοποίηση των credentials μόνο από τον **WEB\_DEMO\_SQLAuthenticator** είναι αρκετή.

## Java EE Container Managed Authentication API

Αφού ολοκληρώσουμε όλα τα προηγούμενα το μόνο που μένει να κάνουμε είναι να φτιάξουμε τις φόρμες για το login σε όλα τα σημεία της εφαρμογής μας όπου είναι επιθυμητό (στην περίπτωση της webDemoApp μόνο στην welcome.jsp) και να δηλώσουμε να στέλνουν τα requests τους σε έναν Servlet ο οποίος θα καλεί την μέθοδο **HttpServletRequest.login(String username, String password)** για να πραγματοποιηθεί το login του χρήστη και **HttpServletRequest.logout()** για να πραγματοποιηθεί το logout του. Από την στιγμή που θα πραγματοποιηθεί η κλήση της login μεθόδου ο χρήστης θα μπορεί να προσπελάζει τους προστατευμένους πόρους στους οποίους έχει πρόσβαση με βάση τους ρόλους που του αντιστοιχούν.

Στην DemoWebApp εφαρμογή οι συναρτήσεις αυτές καλούνται στις μεθόδους **loginAndFetchUser** και την **logout** του UserManagementServer stateless bean ο κώδικας του οποίου έχει παρουσιαστεί στο κεφάλαιο “Stateless Session Beans”. Οι συναρτήσεις αυτές του UserManagementService με την σειρά τους καλούνται από τους Servlets **LoginServlet** και **LogoutServlet**.

**Τα προηγούμενα αρκούν για να επιτύχουμε την δημιουργία του μηχανισμού ασφάλειας της εφαρμογής μας δηλωτικά, κάτι που καλύπτει συνήθως τις ανάγκες των περισσότερων εφαρμογών.**

Παρακάτω αναφέρουμε επιπλέον συναρτήσεις του Java EE Container Managed Authentication API, οι οποίες ενδεχομένως να φανούν χρήσιμες στις περιπτώσεις όπου χρειαστεί να αποφασίσουμε **προγραμματιστικά κατά το runtime**, για τις παραμέτρους ασφάλειας και την συμπεριφορά της εφαρμογής στις περιπτώσεις όπου γίνεται παραβίαση της ασφάλειάς της. Όλες οι παρακάτω μέθοδοι ανήκουν στην κλάση **HttpServletRequest**.

- **boolean isUserInRole(String role):** Ο μόνος τρόπος για να διαπιστώσουμε σε ποιους ρόλους ανήκει ένας logged in user είναι να χρησιμοποιήσουμε αυτήν την συνάρτηση επανειλημμένα με κάθε δυνατό ρόλο.
- **boolean authenticate(HttpServletRequest response):** Σε περίπτωση που ο χρήστης είναι authenticated επιστρέφει true. Σε αντίθετη περίπτωση επιστρέφει false και κάνει commit το response αντικείμενο που έχει περαστεί ως παράμετρος το οποίο επιστρέφεται στον χρήστη και τον αναγκάζει να ταυτοποιηθεί π.χ. με το να μεταφέρεται στην σελίδα που έχει οριστεί στο element <form-login-page> (του <login-

**config>**) ή με κάποιον άλλον τρόπο ανάλογα με το Authentication Scheme που έχουμε ορίστε (αναφέρεται σε επόμενο κεφάλαιο).

- **String getAuthType:** Επιστρέφει το Authentication Scheme που έχουμε δηλώσει να εφαρμόζει η εφαρμογή (αναλύεται σε επόμενο κεφάλαιο).
- **java.security.Principal getUserPrincipal():** Επιστρέφει το αναγνωριστικό του χρήστη εάν αυτός είναι authenticated και null σε αντίθετη περίπτωση. Το αντικείμενο Principal περιέχει το όνομα (username) του χρήστη που επιστρέφεται με την μέθοδο **Principal.getName()**. **Η μέθοδος αυτή είναι αυτή στην οποία θα πρέπει να βασιζόμαστε εάν θέλουμε να δούμε εάν ο χρήστης είναι authenticated και η Principal.getName() αυτή από την οποία παίρνουμε το username του.**
- **String getRemoteUser():** Είναι παρεμφερής με την προηγούμενη με την διαφορά ότι επιστρέφει κατευθείαν το username του χρήστη εάν είναι authenticated αλλά παρόλο που ένας χρήστης μπορεί να είναι authenticated ενδέχεται να επιστρέψει null εάν δεν κληθεί αμέσως μετά την υποβολή των αναγνωριστικών από αυτόν (εξαρτάται από το Authentication Scheme που έχουμε επιλέξει και την συμπεριφορά του browser του χρήστη) για αυτό και δεν μπορούμε να βασιζόμαστε πάνω της εάν θέλουμε να ελέγχουμε για το αν ένας χρήστης είναι authenticated.
- **boolean isSecure():** Επιστρέφει true στην περίπτωση που το request έγινε μέσω ασφαλούς καναλιού π.χ. SSL και false σε αντίθετη περίπτωση.

## Authentication Scheme

### Γενικά

Το Authentication Scheme αφορά μόνο τον τρόπο με τον οποίον ο Servlet Container θα υποχρεώσει τον χρήστη να προχωρήσει σε ταυτοποίηση (εισαγωγή των credentials του, δηλαδή username και password) στην περίπτωση που επιχειρίσει να προσπελάσει έναν προστατευμένο πόρο χωρίς να έχει γίνει ταυτοποίησή του προηγουμένως.

Ουσιαστικά αυτό σημαίνει ότι όταν ένας χρήστης, που δεν έχει ταυτοποιηθεί, θα εισάγει ένα URL ενός προστατευόμενου πόρου, θα μεταφέρεται αυτόματα σε μία σελίδα όπου θα πρέπει να εισάγει τα credentials του ή αντί να γίνεται αυτό, μπορεί να του εμφανίζεται ένα αντίστοιχο pop up από τον browser (ανάλογα με το Authentication Scheme που έχουμε ορίσει). Εάν εισάγει επιτυχώς τα credentials τότε θα μεταφέρεται στον πόρο τον οποίον είχε ζητήσει εξαρχής. Εάν εισάγει λανθασμένα credentials θα μεταφέρεται στην σελίδα που ορίζεται από το element “**<form-error-page>**” που ορίζεται μέσα στο element “**<login-config>**” του deployment descriptor.

Το Authentication Scheme δεν είναι παρά μία διευκόλυνση για να ορίσουμε την συμπεριφορά της εφαρμογής μας σε περίπτωση που κάποιος χρήστης π.χ. είχε αποθηκεύσει το URL ενός προστατευμένου πόρου (π.χ. με browser bookmark) και επιχείρησε να το προσπελάσει όταν είχε λήξει πια η σύνοδός του, ή στην περίπτωση που ο χρήστης είχε όντως κακόβουλο σκοπό.

Για να ορίσουμε ένα Authentication Scheme δημιουργούμε το element “**<login-config>**” στον deployment descriptor και το ορίζουμε στο element “**<auth-method>**”. Υπάρχουν οι επιλογές **BASIC, DIGEST, FORM και CLIENT-CERT**.

Εάν δεν οριστεί το “**<login-config>**” element το προεπιλεγμένο Authentication Scheme που θα χρησιμοποιηθεί είναι το **BASIC**.

Στην περίπτωση της επιλογής **“FORM”** η οποία είναι η μόνη που μας ενδιαφέρει πρακτικά, θα πρέπει να οριστεί και το “**<form-login-config>**” element που προσδιορίζει το “**<form-login-page>**” και “**<form-error-page>**”. Η σελίδα που δηλώνεται από το element “**<form-login-page>**” είναι αυτή στην οποία θα μεταφέρεται ο χρήστης όταν δεν έχει γίνει ταυτοποίησή του και έκανε προσπάθεια προσπέλασης προστατευμένου πόρου. Η σελίδα αυτή υποτίθεται ότι θα πρέπει να περιέχει μία HTML φόρμα για το login του χρήστη.

## **BASIC**

Η επιλογή **BASIC** βασίζεται στο **Basic Access Protocol Authentication Method του HTTP πρωτοκόλλου**. Δεν προσφέρει καμία ασφάλεια από μόνο του αφού μεταφέρει τα user credentials κωδικοποιημένα σε Base64 κωδικοποίηση (χωρίς κρυπτογράφηση δηλαδή) μέσα σε HTTP Header και για αυτό θα πρέπει να χρησιμοποιείται σε συνδυασμό με SSL. Παρουσιάζει ένα pop up στον χρήστη κάθε φορά που προσπελάζει έναν προστατευμένο πόρο κάτι που μπορεί να μην είναι επιθυμητό στα πλαίσια του συνολικού User Interface του web application.

## **DIGEST**

Η επιλογή **DIGEST** μεταφέρει αντί για το password του χρήστη, το hash του. Αυτό είναι λίγο καλύτερο από την προηγούμενη μέθοδο αλλά και πάλι δεν είναι ασφαλές χωρίς την χρήση επιπρόσθετα και του SSL καθώς κάποιος ενδιάμεσος μπορεί να υποκλέψει το hash και να το στείλει σε δικό του request προς τον server, υποκλέπτοντας πρακτικά την ταυτότητα του αρχικού χρήστη.

## **CLIENT-CERT**

To **CLIENT-CERT** έχει να κάνει με το **“Two Way SSL Authentication”**. Αυτό απαιτεί να υπάρχει SSL Certificate όχι μόνο στον server αλλά και στον client. Χρησιμοποιείται σπάνια και σε ειδικές περιπτώσεις. Προφανώς εάν ο σκοπός μας είναι να φτιάξουμε μία εμπορική εφαρμογή δεν μπορούμε να υποχρεώσουμε τους χρήστες της να αποκτήσουν ο καθένας προσωπικά και από ένα SSL Certificate.

## **FORM**

Ουσιαστικά η επιλογή που μας ενδιαφέρει είναι η **“FORM”**. Αυτή επιτρέπει, σε περίπτωση που ο ένας μη ταυτοποιημένος χρήστης προσπελάσει έναν προστατευμένο πόρο, την αυτόματη ανακατεύθυνση του σε οποιαδήποτε σελίδα της εφαρμογής (δηλώνεται από το “**<form-login-page>**”).

Εάν η δηλωμένη σελίδα περιέχει μία HTML φόρμα που στο attribute **“action”** δηλώνει την τιμή **“j\_security\_check”** και στα input πεδία, στα attributes **“name”** τις τιμές **“j\_username”** και **“j\_password”** τότε μετά από το login μέσω αυτής της φόρμας ο χρήστης θα μεταφερθεί αυτόματα στην σελίδα την οποία είχε ζητήσει εξαρχής.

**Προφανώς και αυτή οι μέθοδος θα πρέπει να γίνεται στα πλαίσια του HTTPS για να υπάρχει ασφάλεια.**

Παρακάτω παρουσιάζεται μία τέτοια φόρμα που υπάρχει στην **DemoWebApp** εφαρμογή στην σελίδα **login.jsp**.

```

1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
2 <%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>
3 <%@ page import="osotnikov.demowebapp.constants.SessionAttributeEnum"%>
4 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
5<html>
6<head>
7
8 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9
10 <%@ include file = "/WEB-INF/includes/head/common-style.html" %>
11
12 <title>Login</title>
13
14 <link rel="stylesheet" type="text/css" href="<%=request.getContextPath()%>/css/auth.css" />
15
16 </head>
17
18<body>
19    <center>
20        <div class="login">
21
22            <!-- did we already try to login and it failed? -->
23            <c:if test="false">
24                <div class="authError">
25                    Invalid User Name or Password. Please try again.
26                </div>
27            </c:if>
28
29            <form action="j_security_check" method="post">
30                <fieldset>
31                    <legend>Login</legend>
32
33                    <div>
34                        <label for="email">Email</label>
35                        <input type="text" id="j_username" name="j_username"/>
36                    </div>
37
38                    <div>
39                        <label for="password">Password</label>
40                        <input type="password" id="j_password" name="j_password"/>
41                    </div>
42
43                    <div class="buttonRow">
44                        <input type="submit" value="Login" />
45                    </div>
46
47                </fieldset>
48            </form>
49        </div>
50    </body>
51 </html>

```

Εάν από την άλλη η δηλωμένη στο element “<form-login-page>” σελίδα περιέχει μία φόρμα η οποία στέλνει τα credentials προς έναν Servlet (όπως γίνεται και στο DemoWebApp από την φόρμα στην welcome.jsp η οποία στέλνει τα credentials προς τον LoginServlet) τότε η συμπεριφορά θα είναι αυτή που ορίζει ο Servlet χωρίς δηλαδή να γίνεται αυτόματη ανακατεύθυνση του χρήστη στον πόρο που είχε ζητήσει αρχικά εάν κάτι τέτοιο δεν υλοποιήθηκε από τον developer.

**Στην περίπτωση που κάνουμε χρήση της φόρμας με το attribute “action” να έχει την τιμή “j\_security\_check” απαιτείται προσοχή διότι μετά το login του χρήστη δεν μπορούμε να ορίσουμε να εκτελεσθεί κάποια άλλη επιπλέον ενέργεια.**

Π.χ. στον LoginServlet εάν ορίζαμε την αποθήκευση των πληροφοριών για τον χρήστη στο session που του αντιστοιχεί, αυτές θα ήταν διαθέσιμες στις σελίδες που τις χρησιμοποιούν μόνο στην περίπτωση που το login είχε γίνει μέσω του LoginServlet. Επειδή μετά το login μέσω της

“j\_security\_check” φόρμας δεν εκτελείται αποθήκευση αυτών των πληροφοριών στο session, οι σελίδες που τις χρησιμοποιούν δεν θα τις εμφάνιζαν. Για αυτόν τον λόγο αποθηκεύουμε τέτοιες πληροφορίες στο Session μέσα στον Filter **AuthenticatedUserStateProviderFilter** που ελέγχει σε κάθε request εάν ο χρήστης που το υπέβαλε είναι authenticated και στην περίπτωση που είναι, συγκεντρώνει και αποθηκεύει αυτές τις πληροφορίες.

## Container Managed Confidentiality

Για την επίτευξη του **confidentiality** (αλλά και του **data integrity**) ως γνωστών χρησιμοποιείται κατά κόρον το **HTTPS** και επακολούθως το **SSL** πρωτόκολλο (το οποίο πια στην πιο πρόσφατη έκδοσή του ονομάζεται **TLS**).

Πρώτα θα δείξουμε πως θα ρυθμίσουμε τον Weblogic Server 12C ώστε να υποστηρίζει επικοινωνία μέσω HTTPS και στην συνέχεια το πώς θα ρυθμίσουμε την εφαρμογή μας ώστε να υποχρεώνει την αυτόματη ανακατεύθυνση του χρήστη στο HTTPS πρωτόκολλο όταν προσπελάζει προστατευμένους πόρους.

### Διαχείριση των ρυθμίσεων SSL, SSL Certificates και Keystores

The screenshot shows the Oracle WebLogic Server Administration Console interface. On the left, there's a navigation pane with sections like 'Domain Structure' (containing 'wl\_server' with various sub-options like Environment, Clusters, Machines, etc.), 'How do I...', and 'System Status'. The main area is titled 'Settings for AdminServer' under the 'General' tab. It includes fields for 'Name' (AdminServer), 'Template' (No value specified), 'Machine' (None), 'Cluster' (Stand-Alone), 'Listen Address' (empty), 'Listen Port' (7001), and 'SSL Listen Port' (7002). A note says 'SSL Listen Port Enabled' is checked. The right side of the screen shows a summary of servers and other administrative links.

Αρχικά μεταβαίνουμε στο **Environment -> Server -> AdminServer** και ελέγχουμε να είναι τικαρισμένη η επιλογή **“SSL Listen Port Enabled”**. Το πεδίο **“SSL Listen Port:”** θα πρέπει να έχει την επιθυμητή τιμή της θύρας στην οποία θέλουμε να γίνεται η επικοινωνία μέσω **HTTPS**.

Κάτω από το tab “**Configuration**” στο tab “**Keystores**” θα πρέπει να δηλώσουμε το **Identity Keystore** το οποίο περιέχει το SSL Certificate που έχουμε (μπορεί **να το έχουμε αγοράσει από κάποιο Certificate Authority** ή μπορεί να έχουμε φτιάξει μόνοι μας ένα **self signed certificate** με την εντολή **keytool**). Επίσης σε αυτήν την σελίδα μπορούμε ορίσουμε και ένα **Trust Keystore**. Τα Trust Keystores είναι δευτερεύουσας σημασίας καθώς περιέχουν τις αναφορές για το ποια certificates εμπιστεύεται ο Server (ένα παράδειγμα τέτοιων certificates είναι τα certificates που έχουν οι users κάποιου web application που απαιτεί **two way SSL authentication**), δηλαδή τα Trust Keystores χρειάζονται σε πολύ ειδικές περιπτώσεις.

Ο Weblogic Server 12C έχει προκαθορισμένο self signed certificate για τις ανάγκες της ανάπτυξης εφαρμογών σε development environment το οποίο βρίσκεται στο **DemoIdentity.jks** keystore στο path το οποίο φαίνεται στην εικόνα. Επίσης έχει και ένα αντίστοιχο τέτοιο Trust Certificate που βρίσκεται στο **DemoTrust.jks**. Αυτά τα keystores χρησιμοποιούνται από τον Weblogic Server by default που σημαίνει ότι τουλάχιστον για τις ανάγκες του development δεν υπάρχει η ανάγκη για δημιουργία και ορισμό identity και trust keystores.

Τα στοιχεία αυτών των δύο keystores συνοψίζονται στον παρακάτω πίνακα:

Property	Value
Trust store location	%ORACLE_HOME%\wlserver\server\lib\DemoTrust.jks
Trust store password	DemoTrustKeyStorePassPhrase
Key store location	%ORACLE_HOME%\user_projects\domains\[your_web_application's_domain]\security\DemoIdentity.jks
Key store password	DemoIdentityKeyStorePassPhrase
Private key password	DemoIdentityPassPhrase

Στην περίπτωση που θέλουμε να φτιάξουμε ένα δικό μας Identity Keystore ή να χρησιμοποιήσουμε κάποιο που το έχουμε αγοράσει πρέπει να πατήσουμε το κουμπί “**Change**” στο πεδίο “**Keystores**” όποτε και μεταβαίνουμε στην παρακάτω οθόνη:

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

**Lock & Edit**

**Release Configuration**

**Domain Structure**

- demoWebApp
  - Domain Partitions
  - Environment
  - Deployments
  - Services
  - Security Realms
  - Interoperability
  - Diagnostics

**Settings for AdminServer**

**Configuration** Protocols Logging Debug Monitoring Control Deployments Services Security Notes

General Cluster Services **Keystores** SSL Federation Services Deployment Migration Tuning Overload Concurrency Health Monitoring Server Start

Web Services Coherence

Save Cancel

Keystores ensure the secure storage and management of private keys and trusted certificate authorities (CAs). This page lets you view and define various keystore configurations. These help you to manage the security of message transmissions.

**Keystores:** Custom Identity and Custom Trust ▾

Which configuration rules should be used for finding the server's identity keystores? [More Info...](#)

Save Cancel

Εδώ μπορούμε να επιλέξουμε τον τύπο των keystores που θα χρησιμοποιήσουμε. Πρακτικά μας ενδιαφέρει μόνο η επιλογή “**Demo Identity and Demo Trust**” που είναι η default και η “**Custom Identity and Custom Trust**” που μας επιτρέπει να ορίζουμε τα keystores της επιλογής μας. Θα εξετάσουμε την περίπτωση “**Custom Identity and Custom Trust**”.

Αφού κάνουμε την επιλογή μας και πατήσουμε “Save” μεταφερόμαστε στην προηγούμενη οθόνη που τώρα πιά καθιστά ενεργά τα πεδία στα οποία θα δηλώσουμε το path προς το Custom Identity Keystore μας και προς το Demo Trust Keystore.

Σημειώνεται ότι για να δημιουργήσουμε ένα self signed SSL certificate πρέπει να κάνουμε χρήστη του **keytool** (βρίσκεται στον φάκελο “bin” του JDK folder και λογικά είναι διαθέσιμο στην κονσόλα του λειτουργικού μας συστήματος).

Παρακάτω παρουσιάζεται ένα παράδειγμα δημιουργίας ενός Identity Keystore που περιέχει ένα self signed certificate που έχει ημ/νία λήξης 360 μέρες μετά την δημιουργία του. Το alias αυτού του keystore είναι το “**demoWebAppIdKeystore**” και το password του “**sotnikov**”.

```
keytool -genkey -keyalg RSA -alias demoWebAppIdKeystore -keystore
demoWebAppIdKeystore.jks -storepass sotnikov -validity 360 -keysize 2048
```

```
Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Bat0u89>cd C:\dev\keystores

C:\dev\keystores>keytool -genkey -keyalg RSA -alias demoWebAppIdKeystore -keystore demoWebAppIdKeystore.jks -storepass sotnikov -validity 360 -keysize 2048
What is your first and last name?
[Unknown]: na
What is the name of your organizational unit?
[Unknown]: na
What is the name of your organization?
[Unknown]: na
What is the name of your City or Locality?
[Unknown]: na
What is the name of your State or Province?
[Unknown]: na
What is the two-letter country code for this unit?
[Unknown]: na
Is CN=na, OU=na, O=na, L=na, ST=na, C=na correct?
[no]: y

Enter key password for <demoWebAppIdKeystore>
(RUNN if same as keystore password):

C:\dev\keystores>
```

Στην τελευταία ερώτηση πατήσαμε enter ώστε το key password και το keystore password να είναι το ίδιο (“sotnikov”), σε όλα τα άλλα πεδία δεν έχει σημασία το τι θα εισάγουμε.

Αφού το έχουμε δημιουργήσει εισάγουμε το path προς αυτό στο πεδίο “**Custom Identity Keystore:**”. Στο πεδίο “**Custom Identity Keystore Type:**” εισάγουμε “jks” και στο “**Custom Identity Keystore Passphrase: & Confirm Custom Identity Keystore Passphrase:**” εισάγουμε “sotnikov”. Συμπληρώνουμε με αντίστοιχο τρόπο τα πεδία του Trust Keystore αλλά στο “**Custom Trust Keystore:**” ορίζουμε το **πλήρες path για το DemoTrust.jks** που αναφέραμε στον προηγούμενο πίνακα. Στο “**Custom Trust Keystore Type:**” συμπληρώνουμε όπως και πριν “jks” και στο “**Custom Trust Keystore Passphrase: & Confirm Custom Trust Keystore Passphrase:**” εισάγουμε την τιμή “**DemoTrustKeyStorePassPhrase**”.

The screenshot shows the Oracle Coherence Management Console's Configuration interface, specifically the Keystores section. The left sidebar shows navigation links like General, Cluster, Services, and Keystores. The main panel has tabs for General, Cluster, Services, Keystores, SSL, Federation Services, Deployment, Migration, Tuning, Overload, Concurrency, Health Monitoring, and Server Start. Under the Keystores tab, there are sections for Identity and Trust. In the Identity section, the Custom Identity Keystore is set to 'C:\dev\keystores\demoWeb' and the type is 'jks'. The Custom Identity Keystore Passphrase and Confirm Custom Identity Keystore Passphrase both contain '.....'. In the Trust section, the Custom Trust Keystore is set to 'C:\Oracle\Middleware\Oracle\jks' and the type is 'jks'. The Custom Trust Keystore Passphrase and Confirm Custom Trust Keystore Passphrase both contain '.....'. A 'Save' button is located at the bottom of the form.

Επιπλέον θα χρειαστεί να παραμετροποιήσουμε τιμές και στο tab “SSL”.

This page lets you view and define various Secure Sockets Layer (SSL) settings for this server instance. These settings help you to manage the security of message transmissions.

**Identity and Trust Locations:**

**Identity**

**Private Key Location:** from Custom Identity Keystore  
The keystore attribute that defines the location of the private key file. [More Info...](#)

**Private Key Alias:** demoWebAppIdKeystore  
The keystore attribute that defines the string alias used to store and retrieve the server's private key. [More Info...](#)

**Private Key Passphrase:** .....  
The keystore attribute that defines the passphrase used to retrieve the server's private key. [More Info...](#)

**Confirm Private Key Passphrase:** .....

**Certificate Location:** from Custom Identity Keystore  
The keystore attribute that defines the location of the trusted certificate. [More Info...](#)

**Trust**

**Trusted Certificate Authorities:** from Custom Trust Keystore  
The keystore attribute that defines the location of the certificate authorities. [More Info...](#)

**Advanced**

Εδώ εισάγουμε το **keystore alias** (“demoWebAppIdKeystore”) καθώς και το **private key passphrase** (“sotnikov”) που δηλώσαμε κατά την δημιουργία του customer identity keystore μας.

**Σημαντικό!** Εάν κάνουμε λάθος στην εισαγωγή των ρυθμίσεων ενδέχεται να μην μπορούμε να έχουμε πρόσβαση στην κονσόλα του Weblogic Server. Σε αυτήν την περίπτωση θα πρέπει να ανοίξουμε το αρχείο

“%ORACLE\_HOME%\user\_projects\domains\[domain\_of\_your\_application]\config\config.xml” και να διαγράψουμε όλα τα elements που έχουν σχέση με τις τιμές που εισάγαμε προηγουμένως και που βρίσκονται μέσα στο element “<server>” όπου το element “<name>” δηλώνει τον server για τον οποίον είχαμε κάνει τις ρυθμίσεις.

Με αυτές τις ενέργειες έχουμε ολοκληρώσει τις ρυθμίσεις του SSL στον Weblogic Server 12C. Τώρα μένει να δούμε μόνο πως πραγματοποιούμε την εφαρμογή του SSL στου προστατευμένους πόρους του web application μας.

## Εφαρμογή του SSL στους Πόρους ενός Web Application

Για να ρυθμίσουμε την εφαρμογή μας ώστε να υποχρεώνει την αυτόματη ανακατεύθυνση του χρήστη στο HTTPS πρωτόκολλο όταν προσπελάζει προστατευμένους πόρους θα πρέπει να ορίσουμε το element “**<user-data-constraint>**” μέσα στα elements “**<security-constraint>**”. Παρουσιάζουμε το αντίστοιχο κομμάτι του web.xml στο οποίο ορίζεται ότι πρέπει να εφαρμόσουμε SSL σε όλους τους πόρους της εφαρμογής.

```
222
223<security-constraint>
224  <web-resource-collection>
225    <web-resource-name>Https_Resources</web-resource-name>
226    <url-pattern>/*</url-pattern>
227  </web-resource-collection>
228<user-data-constraint>
229   <description>highest supported transport security level</description>
230   <transport-guarantee>CONFIDENTIAL</transport-guarantee>
231 </user-data-constraint>
232</security-constraint>
233
```

Αυτό είναι μία πολύ λογική αντιμετώπιση του προβλήματος της ασφάλειας της εφαρμογής καθώς εάν ορίσουμε να εφαρμόζεται SSL μόνο σε μερικούς πόρους, εάν γίνει request προς αυτούς από πόρο που δεν είναι προστατευμένος το request θα γίνει μέσω ενός απλού HTTP καναλιού και ενδέχεται να περιέχει ευαίσθητα στοιχεία. Επίσης εάν ορίσουμε static resources π.χ. εικόνες να μην είναι προστατευμένες από SSL μπορεί να εξετασθούν πολύ εύκολα τα headers από τα αντίστοιχα requests προς αυτά και ως γνωστόν headers π.χ. με δεδομένα όπως το session id στέλνονται με κάθε request. **Με λίγα λόγια εάν αποφασίσουμε να κάνουμε μερική εφαρμογή του SSL στην εφαρμογή μας, την καθιστούμε ευάλωτη σε πάρα πολλές επιθέσεις και ακόμα και αν δεν κινδυνεύει αρχικά μπορεί μια μελλοντική προσθήκη να παρουσιάσει ευκαιρίες για κακόβουλους χρήστες που είναι δύσκολο να προβλέψουμε.**

# Web Service

Όπως αναφέραμε και στο εισαγωγικό κεφάλαιο για τα Webservices, υποστηρίζουν απομακρυσμένη επικοινωνία μέσω του HTTP πρωτοκόλλου με δυνατότητα υποστήριξης επερογενών συστημάτων. Σε αυτό το κεφάλαιο θα παρουσιάσουμε ένα παράδειγμα υλοποίησης **JAX-WS Webservices**.

## JAX-WS Service

### Γενική Περιγραφή

Αρχικά θα δημιουργήσουμε ένα **JAX-WS Service Endpoint**. Αυτό είναι το **server side** κομμάτι της όλης υλοποίησης. Σε γενικές γραμμές αυτό που απαιτείται είναι η δημιουργία μίας Java κλάσης η οποία θα υλοποιεί την επιθυμητή λειτουργικότητα που θέλουμε να καταστήσουμε προσπελάσιμη σε ένα δίκτυο.

Οι μέθοδοι αυτής της κλάσης που θέλουμε να είναι προσπελάσιμες από απομακρυσμένους πελάτες (**Webservice Clients**) θα πρέπει να είναι public και δεν θα πρέπει να είναι final, abstract ή static. Η ίδια η κλάση θα πρέπει να έχει το annotation **javax.jws.WebService**.

**Μόνο αυτά είναι αρκετά για να δημιουργήσουμε ένα Webservice στον Weblogic Server 12C. Από την στιγμή που θα δημοσιεύσουμε το module στο οποίο ανήκει, θα είναι προσπελάσιμο από απομακρυσμένους clients.**

Εάν ένα webservice δεν γίνεται deployed μέσω του Weblogic Server όπως στα παραδείγματα στην συνέχεια του κεφαλαίου, τυπικά, η δημοσίευση ενός Webservice απαιτεί την δημιουργία ενός **WSDL αρχείου** και **artifact class αρχείων** τα οποία προκύπτουν από αυτόματη παραγωγή κώδικα, και αναλαμβάνουν την κατασκευή των ορισμάτων των μεθόδων του Webservice Endpoint από τα HTTP μηνύματα που έρχονται από τους Webservice clients καθώς και την αποστολή των τιμών που επιστρέφουν μέσα σε HTTP μηνύματα. Έτσι ο προγραμματιστής ασχολείται μόνο με το business logic και όχι τις λεπτομέρειες τις επικοινωνίας. Ένα ακόμα πράγμα που αξίζει να αναφέρουμε είναι ότι τα δεδομένα εσωτερικά του HTTP μηνύματος δομούνται με βάση το **SOAP πρωτόκολλο (Simple Object Access Protocol)**.

Το **WSDL (Web Service Description Language)** αρχείο είναι ένα XML αρχείο που περιγράφει το Webservice και δίνει όλες τις απαραίτητες πληροφορίες για να μπορεί κάποιος να κατασκευάσει έναν client που να μπορεί να επικοινωνήσει με το endpoint που δημιουργήσαμε. Και αυτό δημιουργείται και δημοσιεύεται αυτόματα από τον Weblogic Server 12C.

Σε application containers οι οποίοι δεν αναλαμβάνουν την δημιουργία των artifacts και του WSDL αυτά θα πρέπει να δημιουργηθούν από το **wsgen** εργαλείο γραμμής εντολών που βρίσκεται στον bin φάκελο του JDK.

Για την δημιουργία των **artifacts** του DemoWebAppWsEndpoint θα μπορούσαμε να χρησιμοποιήσουμε αυτήν την εντολή:

```
wsgen -verbose -keep -cp . osotnikov.demowebapp.ws.DemoWebAppWsEndpoint
```

Για την δημιουργία του **WSDL** του θα χρησιμοποιούσαμε αυτήν:

```
wsgen -verbose -keep -cp . osotnikov.demowebapp.ws.DemoWebAppWsEndpoint -wsdl
```

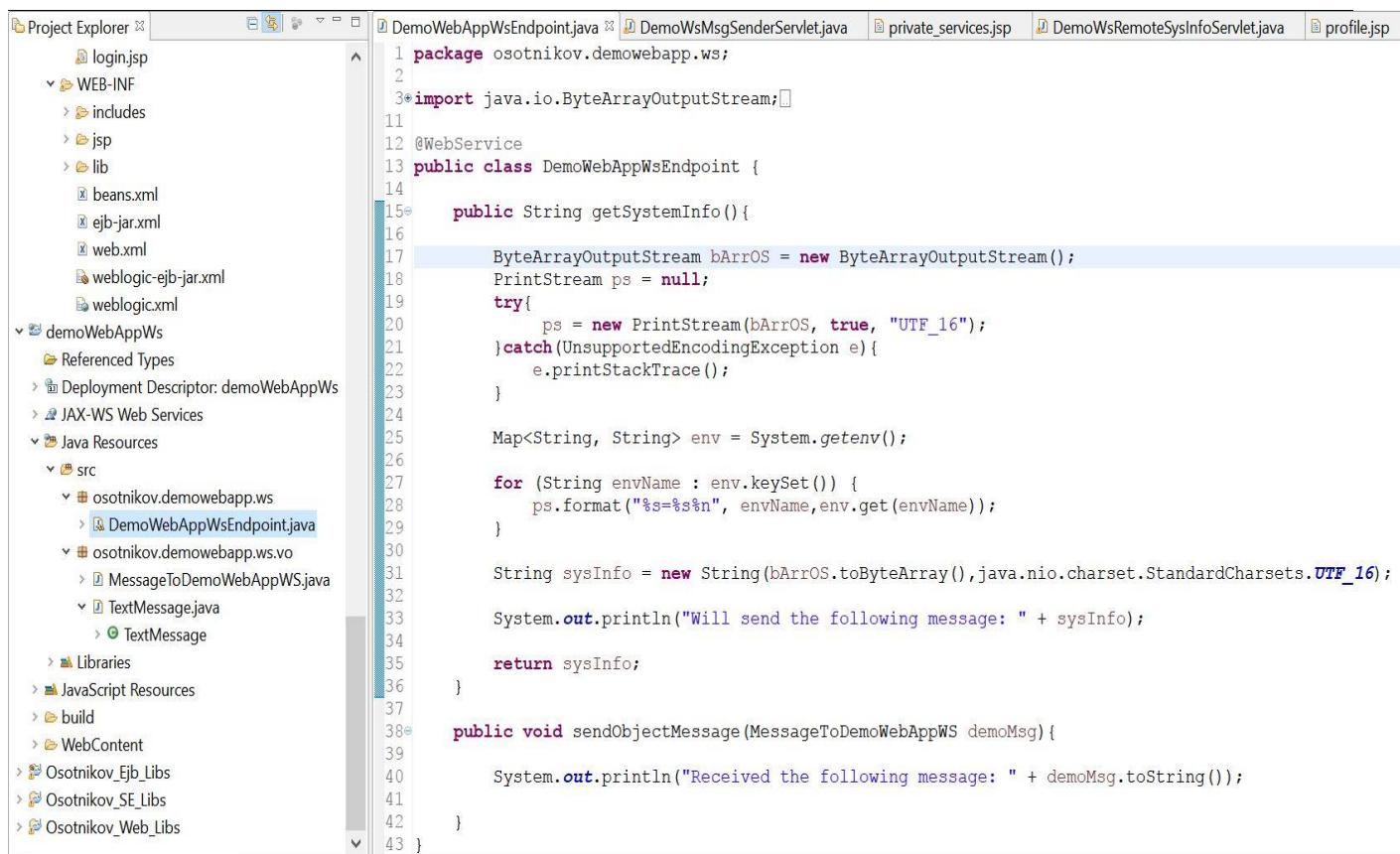
## Παράδειγμα με Κώδικα

### **Δημιουργία Webservice Endpoint**

Για να υλοποιήσουμε ένα παράδειγμα ενός Webservice Endpoint δημιουργήσαμε ένα δεύτερο domain του Weblogic Server (**wsDemoWebAppDomain**), αυτό θα μας επιτρέψει να ξεκινήσουμε ένα δεύτερο instance του Weblogic Server μέσα από το Eclipse, παράλληλα με το instance που θα αντιστοιχεί στο domain demoWebApp στο οποίο αναπτύξαμε όλα τα παραδείγματα των προηγούμενων κεφαλαίων. Στην υπάρχουσα εφαρμογή DemoWebApp αργότερα θα υλοποιήσουμε έναν client για το endpoint που θα παρουσιάσουμε σε αυτό το κεφάλαιο.

Αφού δημιουργήσουμε το domain wsDemoWebAppDomain και το εισάγουμε ως server στον eclipse θα δημιουργήσουμε και ένα καινούριο Dynamic Web Project (**demoWebAppWs**) το οποίο και θα περιέχει το Endpoint.

Παρουσιάζεται το endpoint **DemoWebAppWsEndpoint**:



```

Project Explorer □ DemoWebAppWsEndpoint.java □ DemoWsMsgSenderServlet.java □ private_services.jsp □ DemoWsRemoteSysInfoServlet.java □ profile.jsp
  login.jsp
  WEB-INF
    > includes
    > jsp
    > lib
    beans.xml
    ejb-jar.xml
    web.xml
    weblogic-ejb-jar.xml
    weblogic.xml
  demoWebAppWs
    > Referenced Types
    > Deployment Descriptor: demoWebAppWs
    > JAX-WS Web Services
    > Java Resources
      > src
        > osotnikov.demowebapp.ws
          > DemoWebAppWsEndpoint.java
        > osotnikov.demowebapp.ws.vo
          > MessageToDemoWebAppWS.java
          > TextMessage.java
            > TextMessage
        > Libraries
      > JavaScript Resources
      > build
      > WebContent
    > Osotnikov_Ejb_Libs
    > Osotnikov_SE_Libs
    > Osotnikov_Web_Libs
  □ DemoWebAppWsEndpoint.java
  1 package osotnikov.demowebapp.ws;
  2
  3 import java.io.ByteArrayOutputStream;
  11
  12 @WebService
  13 public class DemoWebAppWsEndpoint {
  14
  15   public String getSystemInfo() {
  16
  17     ByteArrayOutputStream bArrOS = new ByteArrayOutputStream();
  18     PrintStream ps = null;
  19     try{
  20       ps = new PrintStream(bArrOS, true, "UTF_16");
  21     }catch(UnsupportedEncodingException e){
  22       e.printStackTrace();
  23     }
  24
  25     Map<String, String> env = System.getenv();
  26
  27     for (String envName : env.keySet()) {
  28       ps.format("%s=%s%n", envName,env.get(envName));
  29     }
  30
  31     String sysInfo = new String(bArrOS.toByteArray(),java.nio.charset.StandardCharsets.UTF_16);
  32
  33     System.out.println("Will send the following message: " + sysInfo);
  34
  35     return sysInfo;
  36   }
  37
  38   public void sendObjectMessage(MessageToDemoWebAppWS demoMsg) {
  39
  40     System.out.println("Received the following message: " + demoMsg.toString());
  41
  42   }
  43 }

```

Όπως βλέπουμε ο κώδικας είναι πολύ απλός. Υπάρχει δύο μέθοδοι. Η μία έχει σκοπό να παρουσιάσει μία μέθοδο που επιστρέφει ένα αντικείμενο (**getSystemInfo**) και η άλλη μία που να λαμβάνει (**sendObjectMessage**).

Η **getSystemInfo** επιστρέφει ένα String που περιέχει πληροφορίες για τα environment properties του συστήματος. Έτσι εάν ένας client θέλει να μάθει λεπτομέρειες για το απομακρυσμένο σύστημα στο οποίο εκτελείται το Webservice μπορεί να το κάνει μέσω αυτής της μεθόδου.

Η **sendObjectMessage** λαμβάνει ένα σύνθετο αντικείμενο ως όρισμα και το καταγράφει στα logs του συστήματος. Το αντικείμενο αυτό είναι τύπου **MessageToDemoWebAppWS**.

```

1 package osotnikov.demowebapp.ws.vo;
2
3 public class MessageToDemoWebAppWS {
4
5     private TextMessage textMessage;
6     private int messageId;
7     private String callerId;
8
9     public MessageToDemoWebAppWS() {
10         super();
11         // TODO Auto-generated constructor stub
12     }
13
14     public MessageToDemoWebAppWS(TextMessage textMessage, int messageId, String callerId) {
15         super();
16         this.textMessage = textMessage;
17         this.messageId = messageId;
18         this.callerId = callerId;
19     }
20
21     public TextMessage getTextMessage() {
22         return textMessage;
23     }
24
25     public void setTextMessage(TextMessage textMessage) {
26         this.textMessage = textMessage;
27     }
28
29     public int getMessageId() {
30         return messageId;
31     }
32
33     public void setMessageId(int messageId) {
34         this.messageId = messageId;
35     }
36
37     public String getCallerId() {
38         return callerId;
39     }
40
41     public void setCallerId(String callerId) {
42         this.callerId = callerId;
43     }
44
45     @Override
46     public String toString() {
47         return "\ntextMessage:\n" + textMessage + "\n\nmessageId: " +
48                 messageId + "\ncallerId: " + callerId;
49     }
50
51
52 }
53

```

Το πεδίο textMessage είναι τύπου **TextMessage**.

```
1 package osotnikov.demowebapp.ws.vo;
2
3 public class TextMessage {
4
5     private String sender;
6     private String subject;
7     private String body;
8
9     public TextMessage() {
10         super();
11         // TODO Auto-generated constructor stub
12     }
13
14     public TextMessage(String sender, String subject, String body) {
15         super();
16         this.sender = sender;
17         this.subject = subject;
18         this.body = body;
19     }
20
21     public String getSender() {
22         return sender;
23     }
24     public void setSender(String sender) {
25         this.sender = sender;
26     }
27     public String getSubject() {
28         return subject;
29     }
30     public void setSubject(String subject) {
31         this.subject = subject;
32     }
33     public StringgetBody() {
34         return body;
35     }
36     public void setBody(String body) {
37         this.body = body;
38     }
39
40     @Override
41     public String toString() {
42         return "\n\tsender: " + sender + "\n\tsubject: " + subject + "\n\tbody: " + body;
43     }
44
45
46 }
```

## Testing του Webservice Endpoint

Για να ελέγξουμε εάν το Webservice Endpoint μας λειτουργεί κανονικά μεταβαίνουμε αρχικά στην κονσόλα του Weblogic και στο section “**Deployments**”.

Name	State	Health	Type	Targets	Scope	Domain Partitions	Deployment Order
auto_generated_ear_.ear	Active	OK	Enterprise Application	AdminServer	Global		100
Modules							
demoWebAppWs			Web Application				
welcome1			Web Application				
EJBs							
None to display							
Web Services							
DemoWebAppWsEndpointService			Web Service				

Επειδή το module μας παράχθηκε αυτόματα από τον Eclipse έχει το όνομα **\_auto\_generated\_ear\_.ear**. Από κάτω του κάνουμε expand την επιλογή Web Services και κλικ πάνω στο **DemoWebAppWsEndpointService**. Όπως παρατηρούμε το όνομα προέκυψε από το **DemoWebAppWsEndpoint** webservice που φτιάξαμε αλλά προστέθηκε η κατάληξη “**Service**”.

Name	Test Point	Comments
DemoWebAppWsEndpointService		Test points for this WebService module.
/demoWebAppWs/DemoWebAppWsEndpointService	?WSDL	WSDL page on server AdminServer

Στην συνέχεια αφού μεταβούμε στο tab “**Testing**” κάνουμε κλικ στο “?WSDL” το οποίο θα μας μεταφέρει στην σελίδα όπου ο Weblogic Server δημοσίευσε το WSDL αρχείο.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<!--
  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.11-b150616.1732 svn-revision#a247ba216861f2c0baac9a3657c5690bce0c744d.
-->
<!--
  Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.11-b150616.1732 svn-revision#a247ba216861f2c0baac9a3657c5690bce0c744d.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wspl_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://ws.demowebapp.osotnikov/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://ws.demowebapp.osotnikov/" name="DemoWebAppWsEndpointService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ws.demowebapp.osotnikov/" schemaLocation="http://192.168.1.1:8001/demoWebAppWs/DemoWebAppWsEndpointService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="getSystemInfo">
    <part name="parameters" element="tns:getSystemInfo"/>
  </message>
  <message name="getSystemInfoResponse">
    <part name="parameters" element="tns:getSystemInfoResponse"/>
  </message>
  <message name="sendObjectMessage">
    <part name="parameters" element="tns:sendObjectMessage"/>
  </message>
  <message name="sendObjectMessageResponse">
    <part name="parameters" element="tns:sendObjectMessageResponse"/>
  </message>
  <portType name="DemoWebAppWsEndpoint">
    <operation name="getSystemInfo">
      <input wsam:Action="http://ws.demowebapp.osotnikov/DemoWebAppWsEndpoint/getSystemInfoRequest" message="tns:getSystemInfo"/>
      <output wsam:Action="http://ws.demowebapp.osotnikov/DemoWebAppWsEndpoint/getSystemInfoResponse" message="tns:getSystemInfoResponse"/>
    </operation>
    <operation name="sendObjectMessage">
      <input wsam:Action="http://ws.demowebapp.osotnikov/DemoWebAppWsEndpoint/sendObjectMessageRequest" message="tns:sendObjectMessage"/>
      <output wsam:Action="http://ws.demowebapp.osotnikov/DemoWebAppWsEndpoint/sendObjectMessageResponse" message="tns:sendObjectMessageResponse"/>
    </operation>
  </portType>
  <binding name="DemoWebAppWsEndpointPortBinding" type="tns:DemoWebAppWsEndpoint">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getSystemInfo">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="sendObjectMessage">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="DemoWebAppWsEndpointService">
    <port name="DemoWebAppWsEndpointPort" binding="tns:DemoWebAppWsEndpointPortBinding">
      <soap:address location="http://192.168.1.1:8001/demoWebAppWs/DemoWebAppWsEndpointService"/>
    </port>
  </service>
</definitions>
```

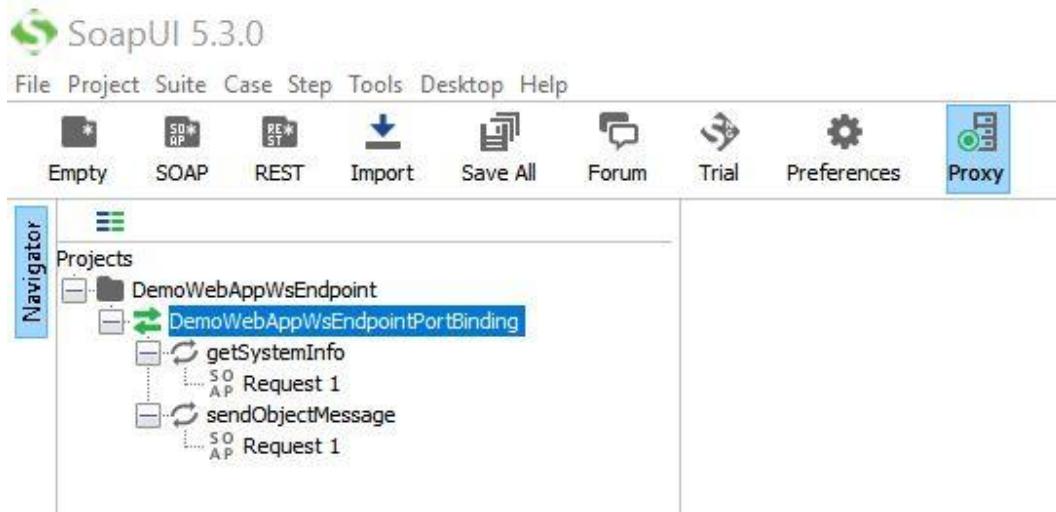
Όπως βλέπουμε το URL του WSDL είναι

<http://localhost:8001/demoWebAppWs/DemoWebAppWsEndpointService?WSDL> και

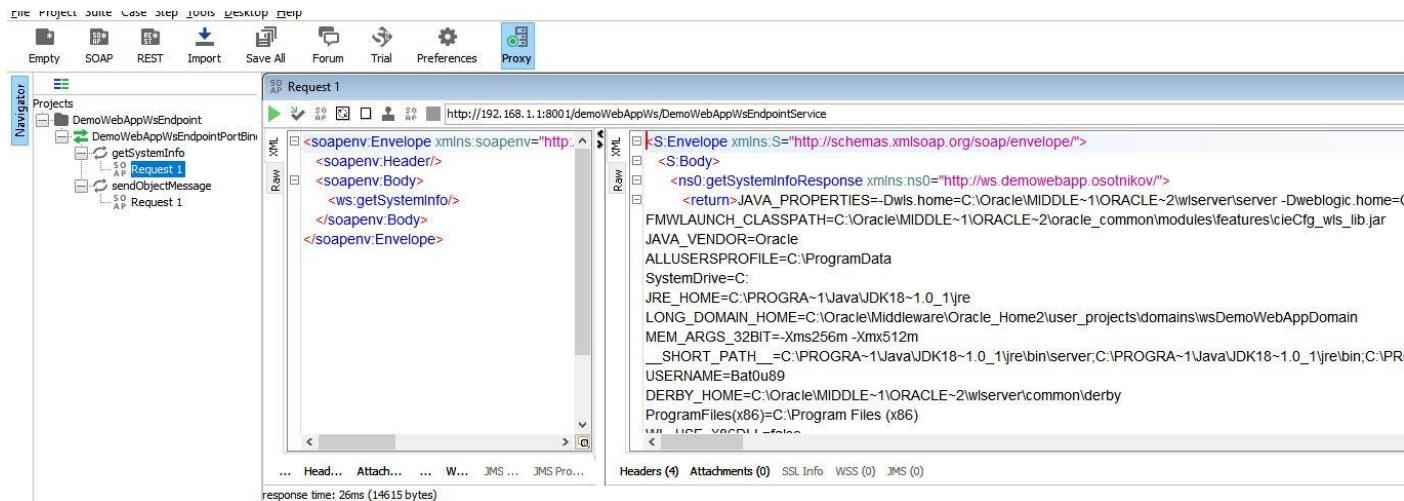
προκύπτει από το όνομα του project στο eclipse ακολουθούμενο από το όνομα της κλάσης του webservice με την κατάληξη “**?WSDL**”.

Για να ελέγξουμε εάν το webservice μας όντως δουλεύει όπως το είχαμε σχεδιάσει μπορούμε να χρησιμοποιήσουμε ένα **SOAP Testing Tool** όπως π.χ. το SoapUI που είναι διαθέσιμο για δωρεάν κατέβασμα από το internet.

Σε αυτό, αρκεί να φτιάξουμε ένα κανούριο **“SOAP Project”** και στο πεδίο **“Initial WSDL”** του αναδυόμενου παραθύρου να εισάγουμε το URL του WSDL. Στο πεδίο **“Project Name”** μπορούμε να εισάγουμε ό,τι θέλουμε και τις υπόλοιπες επιλογές τις αφήνουμε ως έχουν.

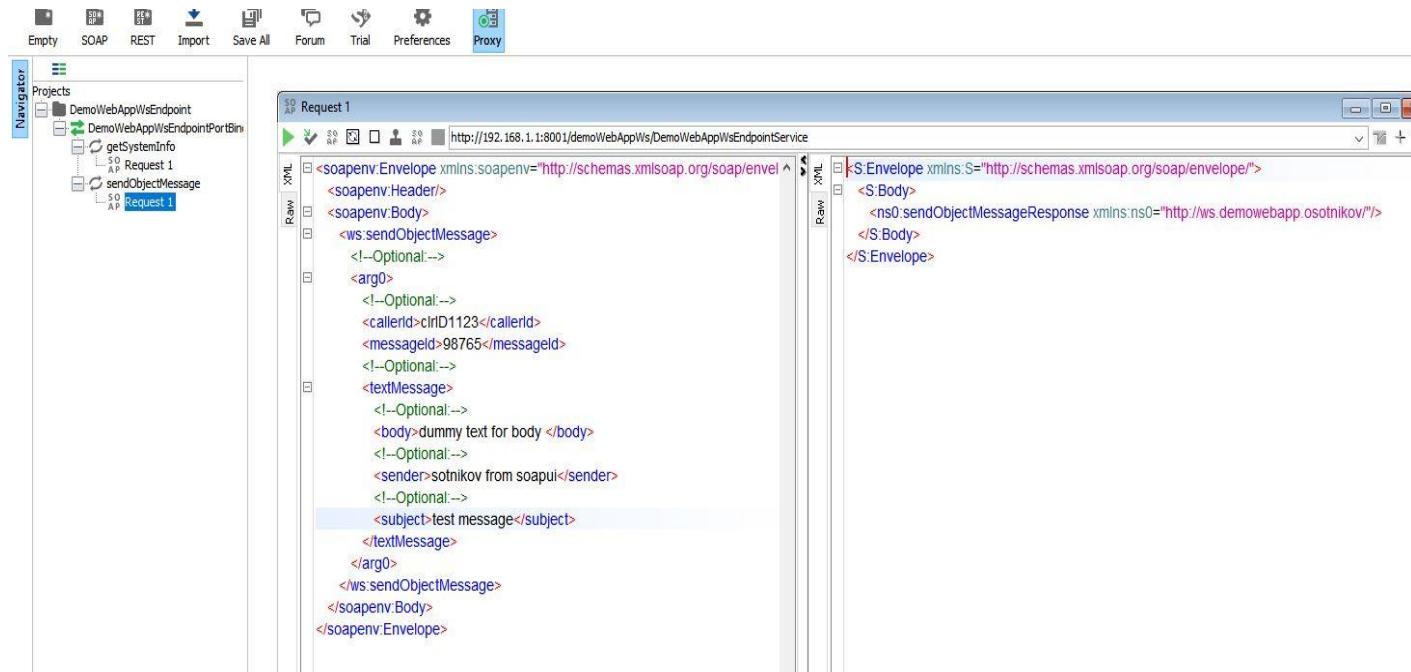


Δημιουργείτε έτσι το project της εικόνας. Εάν κάνουμε διπλό κλικ στο “Request 1” της getSystemInfo παρουσιάζεται το παρακάτω:



Στην αριστερή μεριά του παραθύρου “Request 1” βρίσκεται το SOAP μήνυμα που θα σταλεί προς το Webservice Endpoint. Επειδή ορίσαμε η getSystemInfo να μην δέχεται παραμέτρους δεν χρειάζεται κάποια παραμετροποίηση οπότε μπορούμε να πατήσουμε το κουμπί **“Submit request to specified endpoint URL”** που βρίσκεται τέρμα πάνω αριστερά. Το αποτέλεσμα που επιστρέφεται από το webservice endpoint εμφανίζεται στο δεξιό μέρος του παραθύρου. Επειδή εμφανίζονται οι παράμετροι που αντιστοιχούν στο domain wsDemoWebAppDomain (που διαπιστώνεται από την τιμή της παραμέτρου LONG\_DOMAIN\_HOME) συμπεραίνουμε ότι το webservice μας λειτουργεί χωρίς πρόβλημα.

Από την δοκιμή της μεθόδου **sendObjectMessage** προκύπτει το παρακάτω:



Παρατηρούμε ότι το **MessageToDemoWebAppWS** που είναι ο τύπος του ορίσματος της **DemoWebAppWsEndpoint.sendObjectMessage** μοντελοποιείται κατά το SOAP πρότυπο και τα fields του ως elements με το ίδιο όνομα. Συμπληρώνοντας τις τιμές τους και πατώντας την εκτέλεση της αποστολής λαμβάνουμε ένα κενό μήνυμα διότι η **sendObjectMessage** δεν επιστρέφει τιμή. Επειδή η **sendObjectMessage** απλώς καταγράφει το μήνυμα στο default output stream μεταβαίνουμε στην κονσόλα του eclipse όπου όντως διαπιστώνουμε ότι το μήνυμα έχει έρθει.

```
Markers Properties Servers Data Source Explorer Snippets Console Progress Search REST Annotations WebLogic We
Oracle WebLogic Server 12c R2 (12.2.1) at localhost [wsDemoWebAppDomain] [Oracle WebLogic Server] Weblogic Server
SAVEFILE=C:\PROGRA~1\java\jre1.8.0_131\bin\javaw.exe -Doracle.mIDDLE~1.ORACLE~2=J:\oracle\midDB\ORACLE~2\wlserver\server\lib\jlib\jlib.jar
SERVER_NATIVE_PATH=server\native\win\x64
USERPROFILE=C:\Users\Bat0u89
DEBUG_PORT=8453
TMP=C:\Users\Bat0u89\AppData\Local\Temp
ANT_CONTRIB=C:\Oracle\MIDDLE~1\ORACLE~2\oracle_common\modules\net.sf.antcontrib_1.1.0.0_1-0b3
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
LOCAL_PATH=C:\Oracle\MIDDLE~1\ORACLE~2\wlserver\server\native\win\x64;C:\Oracle\MIDDLE~1\ORACLE~2\wlserver\server\lib\jlib\jlib.jar
PRODUCTION_MODE=true
PROFILE_CLASSPATH=C:\Oracle\MIDDLE~1\ORACLE~2\wlserver\server\lib\weblogic.jar
UTILS_MEM_ARGS=-Xms32m -Xmx1024m
doExitFlag=true

Received the following message:
textMessage:

  sender: sotnikov from soapui
  subject: test message
  body: dummy text for body

messageId: 98765
callerId: clrID1123
```

## Δημιουργία Webservice Client

Οι κλάσεις που χρειάζονται για την απομακρυσμένη κλήση ενός webservice παράγονται με το εργαλείο της γραμμής εντολών **wsimport** που βρίσκεται και αυτό όπως και το **wsgen** στον **bin φάκελο του JDK**. Το wsimport δίνει την δυνατότητα της παραγωγής είτε source είτε class αρχείων με βάση ένα WSDL που μπορεί να διαβαστεί από ένα URL.

Αρχικά θα δημιουργήσουμε τον κατάλογο **C:\dev\temp** και τον κατάλογο **C:\dev\temp\src**. Στην συνέχεια αφού μεταφερθούμε στον κατάλογο **C:\dev\temp** με την εντολή “cd” θα εκτελέσουμε την παρακάτω wsimport εντολή:

```
wsimport -s src -p osotnikov.demowebapp.ws.client  
http://localhost:8001/demoWebAppWs/DemoWebAppWsEndpointService?WSDL
```

Το flag “-s” δηλώνει ότι θα πρέπει να παραχθούν και source αρχεία εκτός από class αρχεία. Αυτά θα πρέπει να αποθηκευτούν στον φάκελο “src”. Το flag “-p” δηλώνει το package στο οποίο θα ανήκουν οι παραχθείσες κλάσεις.

Μετά την εκτέλεση της εντολής αντιγράφουμε τα παραχθέντα source αρχεία και με copy paste τα εισάγουμε στο package **osotnikov.demowebapp.ws.client** στο project **DemoWebApp**.

```
✓ └ osotnikov.demowebapp.ws.client  
    > DemoWebAppWsEndpoint.java  
    > DemoWebAppWsEndpointService.java  
    > GetSystemInfo.java  
    > GetSystemInfoResponse.java  
    > MessageToDemoWebAppWS.java  
    > ObjectFactory.java  
    > package-info.java  
    > SendObjectMessage.java  
    > SendObjectMessageResponse.java  
    > TextMessage.java  
    > META-INF
```

Ένα από αυτά θα είναι αυτό που θα μας δώσει το “**port**” αντικείμενο. Το “**port**” αντικείμενο είναι το αντικείμενο που έχει το ίδιο interface με το endpoint service που φτιάξαμε στο domain **wsDemoWebAppDomain** και είναι αυτό που εκτελεί της απομακρυσμένες σύγχρονες κλήσεις στο webservice.

Για να βρούμε ποια από αυτές τις κλάσεις θα μας δώσει το “**port**” ανατρέχουμε στο WSDL και βρίσκουμε το element “**<service>**” (συνήθως βρίσκεται προς το τέλος του αρχείου). Το attribute **name** αυτού του element δηλώνει την κλάση την οποία πρέπει να κάνουμε instantiate για να πάρουμε το port. Το όνομα του “**port**” δηλώνεται από το attribute “**name**” του element “**port**” και παρέχεται από την αντίστοιχη getter μέθοδο της “**service**” κλάσης. Το interface το οποίο ικανοποιεί το αντικείμενο που επιστρέφεται έχει το ίδιο όνομα με το webservice endpoint.

## DemoWsMsgSenderServlet (DemoWebApp)

Ο servlet αυτός της εφαρμογής DemoWebApp εξυπηρετεί requests για αποστολή μηνυμάτων προς το απομακρυσμένο σύστημα (wsDemoWebAppDomain). Παρουσιάζεται ο κώδικάς του:

```
1 package osotnikov.demowebapp.web;
2
3*import java.io.IOException;□
4
5 public class DemoWsMsgSenderServlet extends HttpServlet{
6
7     protected void doGet(HttpServletRequest request, HttpServletResponse response)
8         throws ServletException, IOException {
9
10        System.out.println("DemoWsMsgSenderServlet.doGet: started for request: "
11                           + request.getRequestURL());
12
13        String jsonData = request.getParameter("jsonData");
14
15        System.out.println("DemoWsMsgSenderServlet.doPost: received jsonData: " + jsonData);
16
17        ObjectMapper mapper = new ObjectMapper();
18
19        MessageToDemoWebAppWS msgToDemoWs = mapper.readValue(jsonData, MessageToDemoWebAppWS.class);
20
21        System.out.println("DemoWsMsgSenderServlet.doPost: parsed jsonData to MessageToDemoWebAppWS: " + msgToDemoWs);
22
23        // Add some more dummy info.
24        msgToDemoWs.setMessageId(new Random().nextInt(1000000000));
25        msgToDemoWs.setCallerId(String.valueOf(this.hashCode()));
26        msgToDemoWs.getTextMessage().setSender(request.getUserPrincipal().getName());
27
28        // GET THE PORT TO THE REMOTE WS
29        DemoWebAppWsEndpointService demoWsService = new DemoWebAppWsEndpointService();
30        DemoWebAppWsEndpoint demoWsServicePort = demoWsService.getDemoWebAppWsEndpointPort();
31        // SEND THE MESSAGE
32        demoWsServicePort.sendObjectMessage(msgToDemoWs);
33
34        JsonResponse jsonRes = new JsonResponse(ResponseStatus.SUCCESS); // The json response object.
35
36        // CREATE THE HTTP RESPONSE BODY
37
38
39        mapper = new ObjectMapper();
40
41        String jsonResponseString = mapper.writeValueAsString(jsonRes);
42
43        System.out.println("DemoWsMsgSenderServlet.doGet: jsonResponseString: " +
44                           jsonResponseString);
45
46        response.setContentType("application/json");
47        response.setCharacterEncoding("UTF-8");
48        PrintWriter resOut = response.getWriter();
49        resOut.write(jsonResponseString);
50
51
52        System.out.println("DemoWsMsgSenderServlet.doGet: ended for request: "
53                           + request.getRequestURL());
54
55    }
56
57
58    protected void doPost(HttpServletRequest request, HttpServletResponse response)
59        throws ServletException, IOException {
60
61        System.out.println("DemoWsMsgSenderServlet.doPost: started ... just calls doGet ...");
62        doGet(request, response);
63
64    }
65
66
67}
68
69
70
71
72}
73
74
75
76
77
78
79}
80
```

Στις γραμμές 35-42 προετοιμάζουμε την παράμετρο της απομακρυσμένης κλήση που είναι ένα αντικείμενο τύπου MessageToDemoWebAppWS. Παρατηρούμε ότι το συγκεκριμένο POJO που ορίστηκε στο project demoWebAppWs παράχθηκε αυτόματα και από το wsimport.

Στις γραμμές 45, 46 παίρνουμε το “service” και το “port” αντικείμενο όπως περιγράψαμε προηγουμένως βρίσκοντας τα ονόματά τους με βάση τα elements “<service>” και “<port>” του WSDL.

Στην γραμμή 48 πραγματοποιούμε την κλήση.

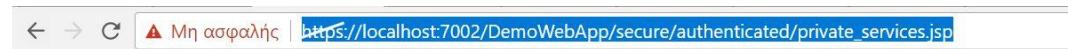
## DemoWsRemoteSysInfoServlet

```
1 package osotnikov.demowebapp.web;
2
3 import java.io.IOException;
4
5 public class DemoWsRemoteSysInfoServlet extends HttpServlet{
6
7     protected void doGet(HttpServletRequest request, HttpServletResponse response)
8         throws ServletException, IOException {
9
10         System.out.println("DemoWsRemoteSysInfoServlet.doGet: started for request: "
11             + request.getRequestURL());
12
13         ObjectMapper mapper = new ObjectMapper();
14
15         // GET THE PORT TO THE REMOTE WS
16         DemoWebAppWsEndpointService demoWsService = new DemoWebAppWsEndpointService();
17         DemoWebAppWsEndpoint demoWsServicePort = demoWsService.getDemoWebAppWsEndpointPort();
18         // RECEIVE THE INFO
19         String remoteSysInfo = demoWsServicePort.getSystemInfo();
20         remoteSysInfo.replace("\n", "<br/>");
21
22         System.out.println("DemoWsRemoteSysInfoServlet.doGet: Received remote system info " +
23             "(remoteSysInfo): " + remoteSysInfo);
24
25         JsonResponse jsonRes = new JsonResponse(ResponseStatus.SUCCESS); // The json response object.
26         jsonRes.setData(remoteSysInfo);
27         // CREATE THE HTTP RESPONSE BODY
28         mapper = new ObjectMapper();
29
30         String jsonResponseString = mapper.writeValueAsString(jsonRes);
31
32         System.out.println("DemoWsRemoteSysInfoServlet.doGet: jsonResponseString: " +
33             jsonResponseString);
34
35         response.setContentType("application/json");
36         response.setCharacterEncoding("UTF-8");
37
38
39         PrintWriter resOut = response.getWriter();
40         resOut.write(jsonResponseString);
41
42         System.out.println("DemoWsRemoteSysInfoServlet.doGet: ended for request: "
43             + request.getRequestURL());
44
45     }
46
47
48     protected void doPost(HttpServletRequest request, HttpServletResponse response)
49         throws ServletException, IOException {
50
51         System.out.println("DemoWsRemoteSysInfoServlet.doPost: started ... just calls doGet ...");
52         doGet(request, response);
53
54     }
55
56 }
57
58 }
```

Παρατηρούμε ότι και το **DemoWsRemoteSysInfoServlet** ακολουθεί την ίδια λογική. Σε αντίθεση με το DemoWsMsgSenderServlet δεν προετοιμάζει κάποια παράμετρο αλλά η απομακρυσμένη κλήση επιστρέφει μία τιμή.

## *private\_services.jsp*

Οι κλήσεις προς αυτά τα Servlets πραγματοποιούνται από την JSP “**private\_services.jsp**” που είναι προσβάσιμη σε authenticated χρήστες στο URL  
**“[https://localhost:7002/DemoWebApp/secure/authenticated/private\\_services.jsp](https://localhost:7002/DemoWebApp/secure/authenticated/private_services.jsp)”.**



# Welcome to your privately accessible services!

## Background Color Selector

Select the background color that you want:  [Submit Color & Reload](#)

Go to the Profile Page (secured, requires authentication)

#### Send Message to Remote System

Subject subject11

dummy text dummy text dummy text dummy text dummy text dummy text dummy  
text dummy text dummy text dummy text dummy text dummy text dummy text dummy  
dummy text  
text dumv text dummy text dumv text dumv text dumv text dumv text dumv text

## Message

**Submit Message**

Your message has been sent.

Get the information about the runtime environment of the remote system (demoWebAppWs).

**Receive Info**

```
JAVA_PROPERTIES=-Dwls.home=C:\Oracle\MIDDLE~1\ORACLE~2\wlserver\server -Dweblogic.home=C:\Oracle\MIF  
FMWLAUNCH_CLASSPATH=C:\Oracle\MIDDLE~1\ORACLE~2\oracle_common\modules\features\cieCfg_wls_lib.jar  
SystemDrive=C: JRE_HOME=C:\PROGRA~1\Java\JDK18~1.0_1\jre LONG_DOMAIN_HOME=C:\Oracle\Middleware\  
MEM_ARGS _32BIT=Xms256m-Xmx512m  
SHORT_PATH =C:\PROGRA~1\Java\JDK18~1.0_1\jre\bin\server;C:\PROGRA~1\Java\JDK18~1.0_1\jre\bin;C:\PRO
```

## Βιβλιογραφία

- Java Platform, Enterprise Edition, The Java EE Tutorial, Release 7
- JSR 318: Enterprise JavaBeans, Version 3.1, EJB Core Contracts and Requirements
- Enterprise JavaBeans 3.1, O'Reilly
- Java EE 7 Essentials, O'Reilly
- Oracle Weblogic 12C Administration Handbook
- Java Message Service Specification Final Release 1.1
- JavaServer Pages Specification, Version 2.0
- Common Annotations for the Java Platform, Version 1.2
- Java Transaction API (JTA) Specification, Version 1.1
- Java Servlet Specification, Version 3.0
- JSR-299: Contexts and Dependency Injection for the Java EE platform
- Oracle Fusion Middleware, Understanding Oracle WebLogic Server 12.1.3